UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Portal web para el despliegue y ejecución de modelos de inferencia de IA sobre plataformas serverless

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Reinoso Hernandez, Santiago Andres

Tutor/a: Moltó Martínez, Germán

Director/a Experimental: CALATRAVA ARROYO, AMANDA

CURSO ACADÉMICO: 2023/2024

Departamento de sistemas informáticos y computación
Universitat Politècnica de València

Portal web para el despliegue y ejecución de modelos de inferencia de IA sobre plataformas serverless

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y Altas Prestaciones

Autor: Santiago Andrés Reinoso Hernández

Tutor/a: Germán Moltó

Directora experimental: Amanda Calatrava

Curso 2023-2024

Resumen

Este proyecto se centra en integrar nuevas capacidades en un portal web *serverless* basado en la arquitectura JamStack, desde el portal se permitirá desplegar y ejecutar modelos de IA encapsulados contenedores de Docker. El enfoque JamStack permite la creación y distribución eficiente de sitios web estáticos, lo que resulta en una experiencia de usuario fluida y veloz.

La aplicación se compone de un frontend desarrollado con el *framework* de JavaScript Angular, en colaboración con el equipo del proyecto europeo AI4EOSC. Se busca extender el portal para simplificar el uso de los modelos de inferencia de IA, haciendo uso de OSCAR, una plataforma de código abierto para la ejecución dirigida por eventos de aplicaciones computacionalmente intensivas.

Para la integración con OSCAR, se desarrolla un cliente utilizando Node.js y Typescript. Este cliente permitirá realizar solicitudes a los servicios de OSCAR a través de su API, así como validar y transformar los datos antes de enviarlos al servicio correspondiente de OSCAR. La creación de un servicio de OSCAR desde el portal web es simple, rápida e intuitiva para el usuario. El cliente de TypeScript actúa como una dependencia dentro del portal web, realizando validaciones de datos y generando un nuevo servicio de OSCAR en un clúster especificado. Como resultado, el usuario obtiene un modelo de IA desplegado como un servicio de OSCAR.

Una vez desplegado, el servicio de OSCAR puede probar fácilmente desde el portal web seleccionando la opción "Try me". Esto desencadena una invocación que ejecuta el modelo de IA dentro de un contenedor Docker en la plataforma OSCAR. Después de recibir la respuesta, el portal web ha sido mejorado para reaccionar dinámicamente a esta, mostrándola de forma amigable para el usuario.

En cuanto a los modelos de inferencia de IA, el proyecto AI4EOSC ofrece una innovadora funcionalidad al permitir a los usuarios acceder a un marketplace existente de imágenes Docker, con una amplia variedad de modelos disponibles para probar. El acceso a este marketplace es libre y brinda a los usuarios la posibilidad de seleccionar y utilizar los modelos que mejor se adapten a sus necesidades, lo que otorga una flexibilidad y personalización en el proceso de inferencia.

Este trabajo académico proporcionará una valiosa ventaja en el trabajo de científicos, investigadores y académicos al ofrecer acceso a una amplia gama de modelos de inferencia, eficiencia en el despliegue, adaptabilidad, escalabilidad y la oportunidad de colaborar con otros expertos en el campo. Estos beneficios combinados contribuirán a mejorar la calidad de sus investigaciones y proyectos en inteligencia artificial.

Palabras clave: Computación en la nube, Javascript, Inteligencia artificial, Contenedores, Kubernetes, Serverless.

Abstract

This project is focused around integrating new capabilities into a serverless web portal based on the JamStack architecture, allowing to deploy and run AI models encapsulated in Docker containers. The JamStack approach enables an efficient creation and distribution of static websites, resulting in a fluent and fast user experience.

The application is composed by a frontend developed with the Angular JavaScript framework, together with the European AI4EOSC project team. It seeks extend the portal in order to simplify the use of AI inference models, using OSCAR, an open-source platform for the event-driven execution of applications that are highly computationally intensive.

For the integration with OSCAR, a Node.js and Typescript client is developed. This client will allow requests to OSCAR services through its API, as well as validate and transform the data before sending it to the corresponding OSCAR service. The creation of an OSCAR's service using the web portal is simple, fast and intuitive for the user. The TypeScript client acts as a dependency inside the web portal, performing data validations and generating a new OSCAR service in a specified cluster. As a result, the user gets an AI model deployed as an OSCAR service.

Once deployed, the OSCAR service can be easily tested from the web portal using the "Try me" option. This triggers an invocation that runs the AI model inside a Docker container on the OSCAR platform. After receiving the response, the web portal has been enhanced to dynamically react to it, displaying it in a user-friendly way.

Regarding AI inference models, the AI4EOSC project offers innovative functionality allowing users to access an existing marketplace of Docker images, with a large variety of models available for test. Access to this marketplace is free and gives users the ability to select and use the models that best suit to their needs, providing flexibility and customization in the inference process.

This academic work will provide a valuable advantage in the work of scientists, researchers, and academics by offering access to a wide range of inference models, efficiency in deployment, adaptability, scalability and the opportunity to collaborate with experts in the field. These combined benefits will contribute to improving the quality of your research and projects in artificial intelligence.

Keywords: Cloud Computing, Javascript, Artificial Intelligence, Containers, Kubernetes, OSCAR, Serverless.

Tabla de contenidos

Tabla de figuras

Glosario

EOSC: *European Open Science Cloud*, es una iniciativa de la comisión europea que pretende desarrollar una red de datos y servicios para la ciencia europea.

OSCAR: *Open Source Serverless Computing for Data-Processing Applications,* plataforma que permite el despliegue de aplicaciones diseñadas dirigidas por eventos.

Kubernetes: Es una plataforma portable que permite administrar contenedores de aplicaciones. Facilita la automatización de cargas de trabajo bajo la configuración declarativa.

AI: *Artificial Intelligence*, es un campo de las ciencias de la tecnología asociado a la resolución de problemas cognitivos enfocado en la imitación de la inteligencia humana, como el aprendizaje, la creación y el reconocimiento de imágenes.

ML: *Machine Learning* en español Aprendizaje basado en máquina, es una rama de la inteligencia artificial que pretende desarrollar técnicas que permitan que las computadoras aprendan.

DL: *Deep Learning* en español Aprendizaje profundo, una rama que intenta enseñar a las computadoras a procesar datos de una manera que se inspira en el cerebro humano.

Angular: Es una plataforma de desarrollo web basada en TypeScript que se enfoca en aplicaciones de una sola página.

TypeScript: Lenguaje de programación basado en JavaScript, que añade tipos y objetos basados en clases.

JamStack: JamStack es una arquitectura diseñada para hacer que la web sea más rápida, segura y fácil de escalar.

OIDC: *Open ID Connect* es un protocolo de autenticación basado en el marco de especificaciones de OAuth 2.0.

1. Introducción

La computación ha sido fundamental en el avance de la tecnología moderna, impulsando la innovación y transformando industrias enteras. Desde el comienzo de la computación, cuando solo existían máquinas de cálculo, hasta las poderosas computadoras de hoy, se ha presenciado una evolución sin precedentes en la forma en que se procesa y utiliza la información. En las últimas décadas, esta trayectoria ha dado un giro significativo con la introducción de la computación en la nube, un paradigma que ha facilitado el acceso a los recursos computacionales y ha abierto a empresas de todos los tamaños la posibilidad escalar sus operaciones de manera eficiente y rentable. Por ejemplo, servicios como Google Drive o Dropbox permiten almacenar y acceder a archivos desde cualquier lugar, mientras que plataformas como las de los proveedores de servicio Microsoft Azure o Amazon Web Services (AWS) ofrecen recursos informáticos que las empresas pueden utilizar para impulsar sus negocios de manera más eficiente y asequible.

Con la nube, ahora es más fácil innovar, ya que la infraestructura de TI ya no es un obstáculo, sino una ayuda para ser más ágil y transformarse digitalmente. El próximo paso en esta evolución es el aumento de las plataformas sin servidor (conocidas en inglés como plataformas *serverless*), donde los desarrolladores pueden crear y lanzar aplicaciones sin tener que preocuparse por la administración de los servidores. Esto significa que pueden centrarse más en escribir el código y en las ideas de negocio, lo que hace que el proceso de desarrollo de software sea más rápido y permite probar nuevas ideas y proyectos con más libertad.

La Inteligencia Artificial (IA) constituye un campo de estudio y aplicación de tecnología informática que busca dotar a los sistemas computacionales de capacidades cognitivas similares a las humanas. En la actualidad, los paradigmas de computación en la nube y *serverless* han desempeñado un papel fundamental al proporcionar acceso libre y flexible a una amplia gama de recursos computacionales. Estas tecnologías no solo permiten a los desarrolladores crear aplicaciones más inteligentes y eficientes, sino que también facilitan el proceso de entrenamiento e implementación de modelos de IA. Esto permite su integración con gran variedad de aplicaciones y contextos, lo que contribuye al avance y la adopción global de la inteligencia artificial.

En el contexto de la computación moderna, son numerosas las iniciativas y proyectos de investigación que se están llevando a cabo, tanto en ámbito privado como público. Es aquí donde encontramos el proyecto AI4EOSC [1], que busca impulsar la ciencia abierta en Europa al integrar la IA para acelerar la investigación y desarrollo de nuevos algoritmos en múltiples áreas, como la agricultura y la biodiversidad. Basado en el proyecto previo DEEP-Hybrid-DataCloud [2], proporciona servicios avanzados que fomentan el análisis de datos y la colaboración científica. Con un énfasis en el aprendizaje automático y los principios FAIR (Findable, Accessible, Interoperable, Reusable), AI4EOSC está marcando un punto de referencia en la computación científica contemporánea.

1.1 Motivación

La Universitat Politècnica de València (UPV) participa activamente en el desarrollo del proyecto "Artificial Intelligence for the European Open Science Cloud" (AI4EOSC) [1]. Este proyecto, liderado por el Consejo Superior de Investigaciones Científicas (CSIC) y compuesto por un consorcio de 10 miembros de 5 países europeos, entre los que se encuentran centros de investigación, universidades, empresas y organizaciones europeas, tiene como objetivo principal proporcionar una plataforma integral para el desarrollo, entrenamiento y despliegue de modelos de inteligencia artificial, aprendizaje automático y aprendizaje profundo (AI/ML/DL), que abarca funciones avanzadas como el aprendizaje distribuido, federado y dividido, así como la implementación de nuevas herramientas para el análisis de los metadatos de procedencia de los modelos. Además, ofrece servicios de procesamiento de datos basados en eventos y facilita el aprovisionamiento de modelos de AI/ML/DL basados en computación sin servidor [3].

Para lograr este gran objetivo, es crucial integrar nuevas funcionalidades que permitan el despliegue y ejecución de servicios de IA para los usuarios. La plataforma "AI4EOSC Dashboard" [4] está diseñada para responder a estas necesidades mediante la integración con el proyecto OSCAR (Open Source Serverless Computing for Data-Processing Applications), una herramienta de código abierto desarrollada por el grupo de Grid y Computación de Altas Prestaciones (GRyCAP) de la UPV que permite la creación y ejecución de aplicaciones altamente paralelas orientadas a eventos en un entorno sin servidor o *serverless*. [5]. Esta integración entre el AI4EOSC Dashboard y OSCAR, que es el principal objetivo de este Trabajo Fin de Máster, permitirá ampliar la oferta de capacidades disponibles ofrecidas en el portal, beneficiando principalmente a investigadores en el ámbito de la Inteligencia Artificial aplicada a cualquier disciplina científica que lo requiera.

1.2 Objetivos

El propósito fundamental de este este proyecto es ampliar las capacidades del portal web para habilitar la ejecución y despliegue de modelos de inteligencia artificial en su fase de inferencia en plataformas de computación *serverless*. De este objetivo principal se derivan los siguientes objetivos específicos:

- Crear nuevos servicios de inteligencia artificial desde el portal web AI4EOSC *Dashboard*, a partir de los modelos disponibles en *Deep Open Catalog* [6].
- Invocar los modelos de inteligencia artificial a través del portal web AI4EOSC *Dashboard*, de manera que se ejecuten de forma síncrona sobre clústeres OSCAR.
- Desarrollar un cliente en lenguaje TypeScript que posibilite la interacción directa con la plataforma OSCAR desde el portal web, para así facilitar la integración entre el *Dashboard* y los clústeres OSCAR.
- Integrar los desarrollos realizados en la base de código del proyecto AI4EOSC para posibilitar la puesta en producción de la integración realizada.

1.3. Estructura de la memoria

Después de haber expuesto la introducción y los objetivos establecidos en este proyecto, el resto del documento se estructura de la siguiente manera. La sección 2, Estado del arte, se centra en proporcionar un panorama completo de las tecnologías relevantes que intervienen en el desarrollo de portales web modernos. Se comparan aspectos como los *frameworks* de JavaScript más utilizados, tales como React, Vue.js y Angular, así como la arquitectura JAMStack, que combina JavaScript, APIs y servicios de Markup. Además, se presentan tecnologías clave en infraestructuras de IT, como Kubernetes, utilizado para desplegar los componentes necesarios para soportar la plataforma OSCAR, y se exploran plataformas *serverless* que permiten ejecutar código sin necesidad de gestionar servidores. Esta sección ofrece una comprensión del ámbito tecnológico en el que se enmarca el proyecto.

En la sección 3, Desarrollo e implementación, se detalla el proceso de desarrollo tanto en el *frontend* como en el *backend* del portal web. Se describe cómo se inicia el desarrollo en ambas partes, utilizando *frameworks* y tecnologías como Angular para la parte de desarrollo web y Node.js para el desarrollo del paquete de cliente. Se explica el despliegue de un clúster local de OSCAR, así como el uso de tokens de OpenID Connect (OIDC) [7], un protocolo de autenticación y autorización basado en OAuth 2.0 utilizado para garantizar la seguridad de la aplicación. Además, en esta sección se realiza un recorrido por las clases más relevantes del cliente de TypeScript, detallando cómo este decodifica y comprende la respuesta recibida. Esto facilita la identificación de la naturaleza de la respuesta, ya sea una imagen en formatos como “png” o “jpg”, o un archivo comprimido en formato “zip”. Este proceso resulta fundamental para la implementación exitosa del proyecto.

La sección 4, Resultados y análisis, se enfoca en presentar los procesos de prueba utilizados para validar el *frontend* y *backend* del portal web. Inicialmente, se explicar el despliegue de un clúster propio de OSCAR, detallando cada paso y su respectiva verificación. Posteriormente, se profundiza en la funcionalidad de despliegue de un servicio de OSCAR, revisando los datos requeridos para crear un servicio en un clúster de OSCAR específico. Asimismo, se ofrece una explicación detallada del proceso de invocación de un servicio desde el AI4EOSC *Dashboard*, incluyendo los datos de entrada necesarios. Una vez comprendido el funcionamiento, se llevan a cabo pruebas que permiten observar la respuesta del servicio en la interfaz web, adaptándose al tipo de contenido de dicha respuesta.

Finalmente, la sección 5, Conclusiones, resume los logros alcanzados en el proyecto y evalúa si han sido cumplidos los objetivos establecidos inicialmente. Se incluyen reflexiones sobre los desafíos hallados durante el desarrollo y se brindan consejos para futuros trabajos en áreas de mejora o expansión. Esta sección cierra el documento con una mirada retrospectiva sobre el proyecto y orientaciones para investigaciones y desarrollos posteriores.

2. Estado del arte

En este capítulo, se abordarán las tendencias actuales en tecnología que están influyendo en el desarrollo de aplicaciones. Se explorará el surgimiento de la computación en la nube, los enfoques *serverless*, los avances en inteligencia artificial, así como el uso de *frameworks* de JavaScript y componentes de la plataforma OSCAR.

2.1 Cloud computing

La computación en la nube es un modelo que te permite acceder fácilmente a un conjunto de recursos informáticos, como redes, servidores, almacenamiento, aplicaciones y servicios. Estos recursos se pueden configurar y utilizar según tus necesidades específicas, y puedes acceder a ellos desde cualquier lugar y en cualquier momento. Además, puedes añadir o quitar recursos rápidamente sin tener que preocuparte por la gestión o la interacción con el proveedor de servicios [8]. Este modelo se despliega a través de diferentes modelos de servicios, como Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS). Esto ofrece ventajas como la flexibilidad, la eficiencia, la seguridad y la reducción de costos [9].

La Infraestructura como Servicio (IaaS) es la base de los servicios en la nube y ofrece recursos de computación virtualizados accesibles a través de Internet. IaaS permite utilizar servidores, almacenamiento y redes, abonando únicamente por el uso que hacen de estos recursos, lo que reduce y elimina la necesidad de realizar inversiones importantes en hardware (máquinas físicas). Este enfoque es idóneo para compañías que buscan flexibilidad y escalabilidad, ya que les permite ajustar los recursos según sus necesidades en cualquier momento [10].

La Plataforma como Servicio (PaaS) proporciona un entorno completo de desarrollo e implementación en la nube, que abarca no solo la infraestructura, sino también middleware, herramientas de desarrollo y otros servicios requeridos para la creación de aplicaciones. Mediante el modelo de servicio PaaS, los desarrolladores pueden crear, probar y desplegar aplicaciones de manera más eficaz, sin tener que ocuparse de la gestión de la infraestructura subyacente. Esto agiliza el ciclo de vida del desarrollo de aplicaciones y disminuye los costos generales [11].

El Software como Servicio (SaaS) representa el modelo más común de entrega de software en la nube, en el cual las aplicaciones residen en servidores remotos y los usuarios acceden a ellas a través de Internet, comúnmente utilizando un navegador web. SaaS elimina la necesidad de instalar y ejecutar aplicaciones en dispositivos individuales, simplificando así el acceso a software actualizado y reduciendo los costes asociados con la infraestructura de TI. Las aplicaciones SaaS están disponibles desde cualquier ubicación y son especialmente útiles para la colaboración en equipo y los servicios que requieren alta disponibilidad [12].

2.2 Serverless computing

La computación *serverless*, también conocida como "sin servidor", es una evolución significativa en el paradigma de la computación en la nube. En este modelo, los desarrolladores pueden diseñar aplicaciones y servicios sin necesidad de ocuparse de la infraestructura subyacente, lo que implica que no tienen que preocuparse por la configuración o el mantenimiento de servidores. En lugar de ello, pueden centrarse únicamente en el código y en la lógica de su aplicación, permitiendo ser más eficientes y ágiles. Además de simplificar el proceso de desarrollo, la computación *serverless* optimiza el uso de recursos al ejecutar código solo cuando es necesario, lo que contribuye a reducir costos de operación y aumentar la escalabilidad de las aplicaciones. Este enfoque permite a las empresas adaptarse rápidamente a las demandas del mercado, sin comprometer la calidad o la fiabilidad de sus servicios [13]

Como se menciona anteriormente, *serverless* ha transformado la manera en que se desarrollan y despliegan las aplicaciones, brindando a los desarrolladores la capacidad de concentrarse en el código delegando al proveedor de servicio en la nube la administración de la infraestructura. Un ejemplo destacado es AWS Lambda [14] donde las funciones se ejecutan en respuesta a eventos, y los usuarios solo pagan por el tiempo de ejecución. Otras tecnologías relevantes incluyen Google Cloud Functions [15], que ofrece un entorno similar para ejecutar código en la infraestructura de Google Cloud, y Azure Functions de Microsoft [16] que proporciona capacidades *serverless* integradas en la plataforma Azure. En la Figura 1 se muestran las plataformas de infraestructura sin servidor más conocidas. La mayoría de los proveedores de este tipo de computación son de tipo público, pero hay algunos como Knative que son *open source* que permiten personalizar el servicio de funciones y desplegarlo en servidores *on-premise*.

Figura 1. Logos de diferentes plataformas serverless [1]

Además de las funciones *serverless* que se ejecutan en plataformas de proveedores de la nube, existen otras tecnologías y enfoques que siguen el paradigma *serverless*. Por ejemplo, *Backend as a Service* es un modelo que proporciona a los desarrolladores un

[1] Imagen tomada de la web: https://blog.back4app.com/es/los-10-principales-proveedores-de-alojamiento-sin-servidor/

backend completo en la nube, incluyendo bases de datos, autenticación de usuarios, notificaciones, y almacenamiento de archivos, sin la necesidad de gestionar la infraestructura subyacente.

Otro enfoque es el uso de *Frontend as a Service*, que ofrece servicios de frontend listos para usar, como hosting de sitios web, entrega de contenido y funciones de frontend que se pueden integrar fácilmente con el backend. Además, plataformas como Netlify[2] y Vercel[3] permiten a los desarrolladores desplegar sitios web estáticos y aplicaciones web con funciones *serverless* integradas, facilitando el proceso de desarrollo y despliegue [17].

2.3 Inteligencia artificial

La inteligencia artificial (IA) representa un campo de estudio dentro de la informática que tiene como objetivo desarrollar sistemas con capacidad para llevar a cabo tareas que comúnmente requieren la intervención humana, como el aprendizaje, la detección de patrones, la toma de decisiones y la resolución de problemas. Se apoya en algoritmos y modelos de aprendizaje automático para analizar y extraer conocimiento de grandes conjuntos de datos, mejorando su desempeño con el tiempo. [18]. La IA tiene aplicaciones diversas que abarcan desde el reconocimiento de voz e imágenes hasta la robótica y el análisis predictivo, lo que implica una transformación significativa en la forma en que interactuamos con la tecnología y procesamos la información [19].

Probablemente, la inteligencia artificial y la computación *serverless* se relacionan en que ambas buscan optimizar y simplificar procesos. La IA se centra en simular capacidades cognitivas humanas para resolver problemas y aprender de los datos, mientras que la computación *serverless* permite ejecutar aplicaciones y servicios sin la gestión directa de servidores. Juntas, estas tecnologías pueden potenciar la eficiencia y la escalabilidad de soluciones inteligentes, permitiendo que sistemas de IA se desplieguen de manera más ágil y económica, aprovechando la flexibilidad y el dinamismo del *serverless* para responder a eventos en tiempo real y escalar según la demanda [18].

2.4 Kubernetes

Kubernetes[4] es una plataforma de código abierto creada para hacer más fácil el proceso de poner en marcha, hacer crecer y administrar aplicaciones que están en contenedores. En otras palabras, ayuda a manejar conjuntos de contenedores que trabajan juntos como una sola aplicación, lo que simplifica su manejo y hace que sea más sencillo descubrir y utilizar sus servicios. Además, Kubernetes es crucial en el entorno de la computación en la nube, ya que se encarga de organizar y controlar la infraestructura de cómputo, redes y almacenamiento en un grupo de servidores conocido como clúster. Esto significa que Kubernetes es una pieza fundamental para aprovechar al máximo los beneficios de la nube, como la flexibilidad y la escalabilidad.

[2] Netlify: https://www.netlify.com/
[3] Vercel: https://vercel.com/
[4] Kubernetes: https://kubernetes.io/es/docs/concepts/

En Kubernetes, el nodo master constituye el plano de control del clúster. Se encarga de gestionar el estado del clúster y coordina todas las actividades, como el lanzamiento de aplicaciones, el mantenimiento de la salud del clúster y la escalabilidad. Este nodo ejecuta varios componentes críticos, incluyendo el *API Server*, el *Scheduler* y el *Controller Manager* y se puede ejecutar en diferentes nodos para disponer de alta disponibilidad [20].

Los nodos *workers*, por otro lado, son las máquinas donde se ejecutan las aplicaciones empaquetadas en contenedores. Cada nodo *worker* ejecuta servicios esenciales como Kubelet, que se comunica con el nodo master y gestiona los contenedores en el nodo, y Kube-proxy, que maneja el enrutamiento de red para los contenedores. En la Figura 2 se pueden identificar los componentes vitales de Kubernetes.

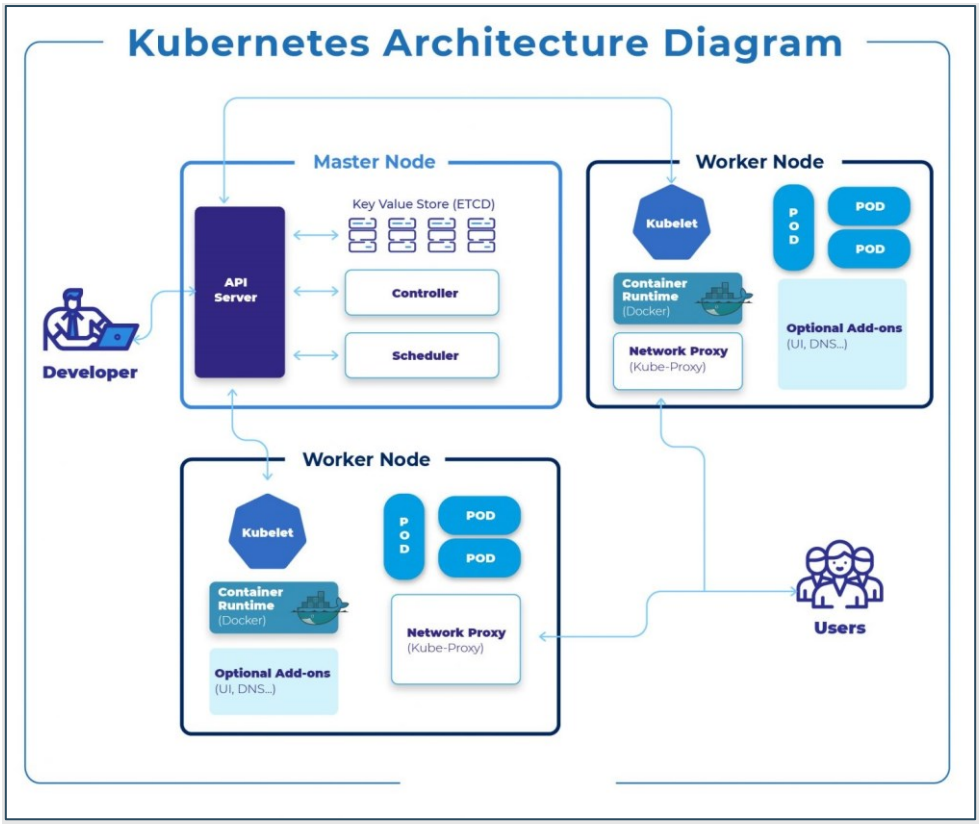

Figura 2. Diagrama de arquitectura de Kubernetes [5]

2.5 Knative

Knative es una tecnología *serverless* que se integra con Kubernetes para facilitar el despliegue y la gestión de aplicaciones sin que los desarrolladores se preocupen por la infraestructura subyacente. Ofrece una serie de componentes que automatizan diversas tareas, como el escalado de aplicaciones, la gestión de eventos y la transformación de código fuente en contenedores. Esto significa que Knative simplifica el proceso de desarrollo, permitiendo a los desarrolladores concentrarse en la escritura de código

[5] Tomado de la web: https://www.fisclouds.com/kubernetes-enabling-flexible-and-scalable-cloud-deployments-9311/

mientras la plataforma se encarga automáticamente de implementar y ajustar los recursos necesarios [21].

Una de las ventajas más destacadas de Knative es su capacidad de escalar a cero, lo que significa que los recursos no se consumen a menos que el código necesite ejecutarse. Esto permite una utilización óptima de los recursos y una reducción significativa de los costos operativos, ya que los desarrolladores solo pagan por el tiempo de procesamiento que realmente utilizan. Además, Knative maneja automáticamente el inicio y la detención de instancias, lo que facilita la gestión de aplicaciones con demandas de tráfico impredecibles o que cambian rápidamente [22].

2.6 Amazon S3

Amazon S3, conocido como *Simple Storage Service*, es un servicio de almacenamiento en la nube ofrecido por AWS. Es uno de los primeros servicios que ofreció esta plataforma en la nube y se destaca por su escalabilidad y facilidad de almacenamiento, desde pequeñas cantidades hasta varios terabytes o incluso petabytes de datos sin preocupaciones sobre la gestión de la infraestructura. Ofrece una durabilidad y disponibilidad excepcionales mediante la replicación automática de datos en múltiples ubicaciones dentro de las regiones de AWS. Esto garantiza la resiliencia ante posibles fallos y facilita el acceso a los datos desde cualquier lugar mediante APIs o herramientas de gestión de archivos [23].

Además, Amazon S3 brinda avanzadas opciones de seguridad y control de acceso, permitiendo configurar políticas para regular quién accede y las que acciones pueden hacer sobre los datos almacenados. Integrado con muchos otros servicios de AWS, es compatible con diversas aplicaciones y *frameworks* [23].

El servicio de Amazon S3 se puede utilizar eficazmente como un servicio de hosting para páginas web estáticas desarrolladas con *frameworks* populares como Vue.js, React.js y Angular. Con esta configuración, los archivos estáticos de la aplicación web, como HTML, CSS, JavaScript y archivos multimedia, se cargan en un *bucket* (un contenedor para objetos almacenados en Amazon S3). Luego, se puede habilitar la funcionalidad de sitio web estático en el *bucket*, lo que permite que Amazon S3 sirva los archivos directamente a los usuarios a través de una URL pública [24]. Además, al combinar Amazon S3 con servicios como Amazon CloudFront, un servicio de distribución de contenido (CDN – *Content Delivery Network*), se puede mejorar aún más el rendimiento y la velocidad de entrega de la aplicación web estática a nivel global. CloudFront también posibilita el uso de certificados TLS (*Transport Security Layer*) para permitir la conexión cifrada entre el navegador del cliente y el servidor.

2.7 Minio

Minio es un servidor de almacenamiento de objetos distribuidos de alto rendimiento, que está diseñado para la infraestructura de nube privada a gran escala. Minio agrega los volúmenes persistentes (PVs) en el almacenamiento de objetos distribuidos escalable. Además, utiliza el API REST de Amazon S3 [25].

El almacenamiento de objetos es una tecnología que gestiona datos en un formato llamado objetos binarios conocidos como Binary Large Object (BLOB). Generalmente son archivos como imágenes, videos, correos electrónicos, páginas web, datos de sensores, etc. Diferenciándose del almacenamiento tradicional basado en archivos o bloques, el almacenamiento de objetos maneja los datos como unidades individuales, cada una con metadatos y un identificador único, ayudando a la administración de grandes cantidades de datos no estructurados de manera eficiente y escalable.

MinIO organiza los objetos utilizando *buckets*, que son similares a los directorios en un sistema de archivos, pero con la capacidad de contener un número ilimitado de objetos. Los *buckets* de MinIO ofrecen la misma funcionalidad que los *buckets* de Amazon S3, lo que significa que las aplicaciones que ya están configuradas para interactuar con Amazon S3 pueden hacerlo también con MinIO. Esto proporciona una mayor flexibilidad y compatibilidad para las aplicaciones que utilizan este tipo de almacenamiento en la nube [26].

2.8 OpenID Connect

OpenID Connect es un sistema de autenticación basado en el estándar OAuth 2.0, diseñado para permitir a los usuarios iniciar sesión de manera segura en aplicaciones y servicios en línea. Este proceso funciona redirigiendo al usuario a la página de inicio de sesión de un proveedor de identidad, como Google o Microsoft, donde se autentica y le otorga permisos a la aplicación. Después de eso, el proveedor de identidad envía un token de identificación a la aplicación o servicio, que puede utilizarse como verificador del usuario y conocer la información básica de su perfil [27].

Esta técnica agiliza el proceso de inicio de sesión único (SSO), lo que posibilita a los usuarios acceder a diversos servicios utilizando una única cuenta. Además, OpenID Connect gestiona el consentimiento del usuario de forma integrada, lo cual es esencial cuando se comparten datos personales. Esta solución resulta especialmente ventajosa para aplicaciones móviles y servicios orientados a consumidores, donde la seguridad y la facilidad de uso son aspectos clave.

2.9 OSCAR

OSCAR [28] es una plataforma de computación sin servidor de código abierto diseñada para aplicaciones de procesamiento de datos basadas en Docker. Gracias al servicio Infrastructure Manager (IM) [29], los clústeres OSCAR se despliegan en clústeres de Kubernetes elásticos que pueden estar en múltiples nubes, incluyendo nubes públicas, privadas y federadas.

También, permite la creación de flujos de trabajo *serverless* orientados a datos mediante un lenguaje de definición de funciones (FDL – Functions Definition Language) [30]. Estos flujos de trabajo pueden activarse en respuesta a la carga de archivos en un almacenamiento de objetos, como MinIO, y ejecutar un script de shell definido por el usuario dentro de un contenedor provisto de una imagen Docker. Los resultados se orquestan como trabajos por lotes de Kubernetes y los datos de salida se cargan en algún

sistema de almacenamiento de objetos compatible, como MinIO, Amazon S3 o EGI DataHub, entre otros [5].

OSCAR se beneficia de la elasticidad proporcionada por CLUES [31], que monitorea la carga de trabajo y ajusta automáticamente el número de nodos según sea necesario. Además, OSCAR se integra con herramientas como SCAR para ejecutar aplicaciones genéricas en AWS Lambda, facilitando la creación de flujos de trabajo serverless que abarcan desde la nube hasta el borde de la red. Esto permite aprovechar la computación en la nube para realizar procesamiento intensivo de datos en cualquier punto de la red, garantizando una mayor eficiencia y flexibilidad en el despliegue de servicios [5]. La Figura 3 representa a la arquitectura de OSCAR y sus componentes.

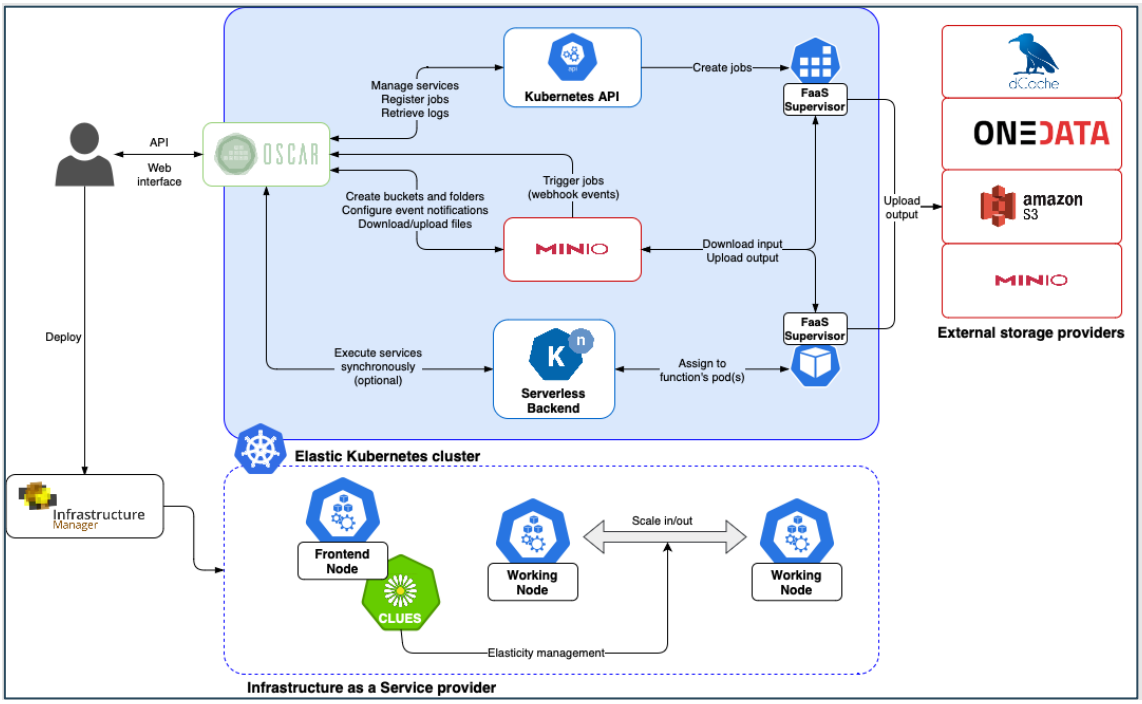

Figura 3. OSCAR diagrama de componentes [6]

2.10 Javascript frameworks

Existen varios *frameworks* que guían al desarrollador en el proceso de creación una aplicación web basada en Javascript. A continuación, se nombran los tres *frameworks* con más adopción por la comunidad.

2.10.1 React

React [7], desarrollado por Facebook, es una biblioteca de JavaScript que se utiliza para construir interfaces de usuario interactivas. Se centra en la creación de componentes reutilizables, lo que facilita de aplicaciones web y móviles rápidas y dinámicas. Además, React ofrece una eficiente actualización del DOM, lo que significa que solo se actualizan las partes de la interfaz que han ido cambiando, mejorando así el rendimiento de la

[6] Imagen tomada de la web: https://docs.oscar.grycap.net/
[7] React: https://react.dev/reference/react

aplicación. Sin embargo, una desventaja de React es que su curva de aprendizaje puede ser empinada para algunos desarrolladores debido a su uso de JSX (JavaScript Syntax eXtension) y su enfoque en un paradigma de programación funcional.

2.10.2 Angular

Angular [8], creado por Google, es un *frameworks* muy completo que proporciona una amplia gama de funcionalidades para el desarrollo de aplicaciones web. Su arquitectura basada en componentes y su conjunto de herramientas integradas hacen que sea fácil construir grandes aplicaciones escalables y mantenibles. Una ventaja de Angular es su potente sistema de enlace de datos bidireccional, que simplifica la manipulación de datos y la actualización de la interfaz de usuario. Sin embargo, su complejidad puede resultar abrumadora para algunos desarrolladores, especialmente para aquellos que recién comienzan en el desarrollo web.

2.10.3 Vue

Vue.js [9] es un *framework* que se centra en la capa de la vista y se puede integrar fácilmente con otras bibliotecas o proyectos existentes. Vue es conocido por su simplicidad y su enfoque en la experiencia del desarrollador, lo que lo hace accesible para principiantes y una excelente herramienta para los profesionales. Una ventaja clave de Vue.js es su bajo tiempo de aprendizaje, lo que permite a los desarrolladores comenzar a construir aplicaciones rápidamente. Sin embargo, debido a su menor adopción en comparación con React y Angular, es posible que Vue.js no tenga tantos recursos y bibliotecas disponibles.

2.11 JamStack

La arquitectura JamStack es una forma moderna de crear páginas web. Se basa en la combinación de JavaScript, APIs y Markup (HTML/CSS). Con este enfoque, los sitios web se crean de manera que se pre-renderizan y luego se entregan a través de un CDN. Esto hace que los sitios sean más rápidos, seguros y escalables. Con JamStack, se pueden construir tanto sitios web estáticos como aplicaciones web dinámicas. Al servir el contenido estático desde una CDN y utilizar JavaScript para funciones dinámicas, junto con APIs para obtener datos, los desarrolladores pueden crear sitios web más rápidos y seguros [32]. La Figura 4 representa algunas tecnologías asociadas con JamStack.

[8] Angular: https://angular.io/guide/what-is-angular
[9] Vue.js: https://es.vuejs.org/v2/guide/

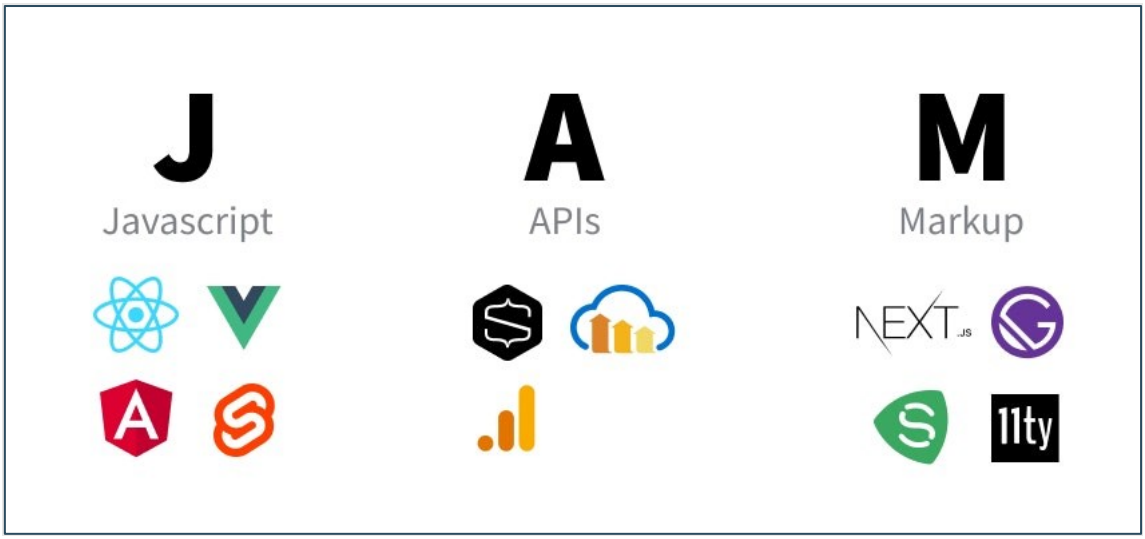

Figura 4. Tecnologías de JamStack [10]

2.12 Node.js

Node.js ha revolucionado el mundo del desarrollo al ofrecer un entorno de ejecución de JavaScript del lado del servidor. Su modelo de operaciones no bloqueantes y orientadas a eventos lo hacen perfecto para aplicaciones que necesitan manejar múltiples conexiones simultáneas y alto rendimiento. Además, gracias a npm (Node Package Manager), los desarrolladores pueden compartir y reutilizar fácilmente código a través de una amplia variedad de paquetes y módulos [33].

[10] Imagen tomada de la web: https://snipcart.com/blog/jamstack

3. Desarrollo e implementación

Este capítulo engloba varios aspectos fundamentales para comprender el desarrollo y la implementación de las tecnologías clave en este proyecto. Se inicia con una exploración exhaustiva de OSCAR, detallando su despliegue como plataforma serverless y examinando cada uno de sus componentes y funcionalidades. Posteriormente, se hace un análisis detallado del proceso de desarrollo del cliente oscar-js, proporcionando una visión completa del código y destacando los fragmentos más relevantes para su comprensión y utilización. Además, se explora la creación de nuevos componentes para el portal AI4EOSC Dashboard, desde su diseño y desarrollo hasta su implementación utilizando el *framework* Angular. Finalmente, se dedica una sección al proceso de pruebas unitarias, diseñadas para garantizar que todos los componentes del desarrollo cumplan con los estándares de funcionalidad y calidad establecidos.

Con el fin de cumplir los objetivos establecidos, se ha diseñado una integración con OSCAR a través de un cliente conocido como oscar-js. El cliente de oscar-js es un nuevo desarrollo que se ha creado con el fin de conseguir una forma sencilla para que desarrolladores de Javascript puedan interactuar con clústeres de OSCAR. La función principal de este cliente es conectarse a un clúster de OSCAR y gestionar las peticiones directas al clúster, permitiendo el despliegue e inferencia del modelo. Además, oscar-js actúa como enlace entre el *dashboard* de AI4EOSC y la plataforma *serverless* OSCAR. En cuanto a la parte visual del proyecto, se ha utilizado el *framework* de JavaScript, Angular. Esta elección se considera adecuada para una aplicación web que requiere ser robusta y escalable.

La Figura 5 presenta de forma gráfica la solución propuesta, mostrando la integración de todas las partes involucradas en el proyecto. Además, ilustra cómo se implementan las funcionalidades de creación e invocación servicios de OSCAR directamente desde el AI4EOSC Dashboard.

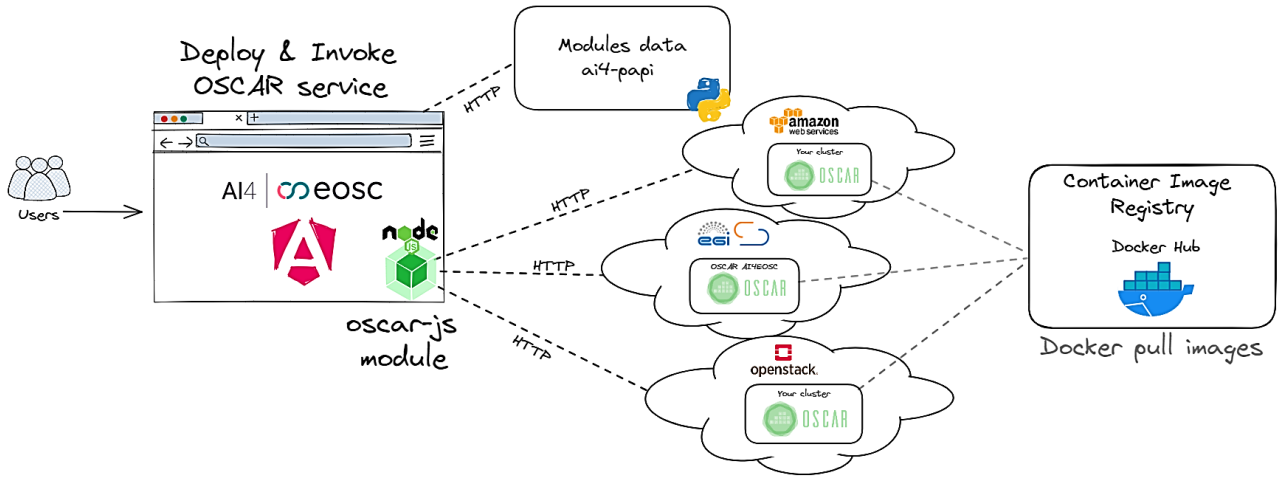

Figura 5. Diagrama integración de componentes oscar-js module. Portal web y diferentes clústeres de OSCAR

Los usuarios pueden utilizar el AI4EOSC *Dashboard* como punto de entrada, donde pueden visualizar los diversos modelos disponibles en un *markeplace* de imágenes Docker ya existente. Este *markeplace* ofrece una amplia gama de modelos para su evaluación y prueba. Por ejemplo, la Figura 6 corresponde a la página inicial del *dashboard* que muestra el componente que contiene una lista con los modelos disponibles.

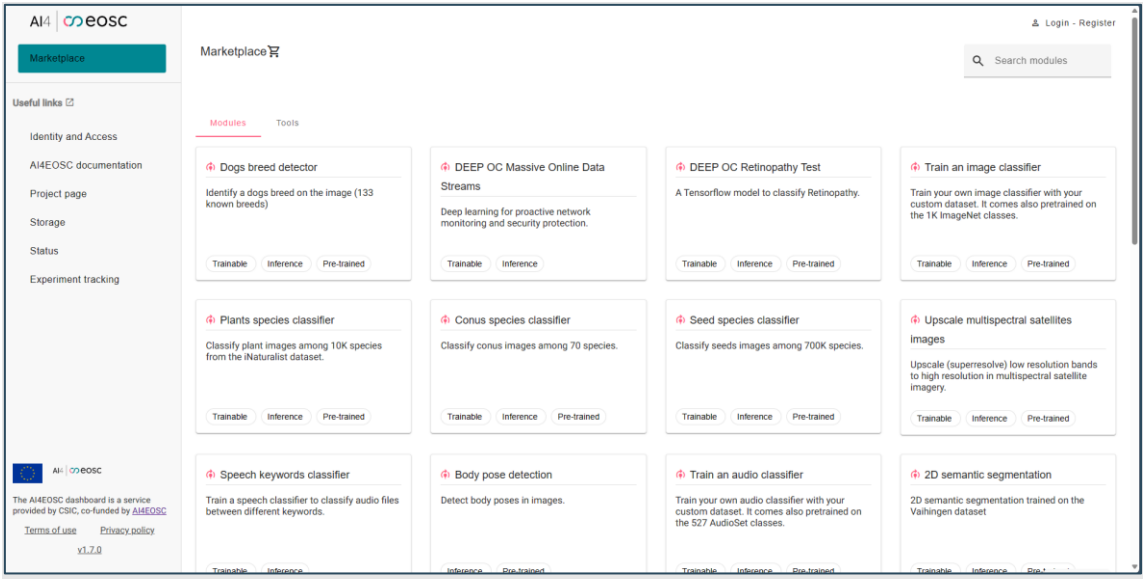

Figura 6. Markteplace de AI4EOSC Dashboard

La información sobre cada uno de estos modelos es proporcionada por ai4-papi[11], un proyecto que expone datos sobre los servicios de AI4EOSC a través de una API. Entre los datos de interés se encuentran el nombre del modelo, la descripción, el repositorio de GitHub y la fecha de creación, entre otros detalles relevantes. En el siguiente fragmento de código se muestra un ejemplo de la respuesta a la solicitud GET en el *endpoint* "v1/catalog/modules/detail" a la API ai4-papi. Esta solicitud devuelve información general de cada uno de los modelos disponibles.

```
[
  {
    "title": "Dogs breed detector",
    "summary": "Identify a dogs breed on the image (133 known breeds)",
    "keywords": [
      "tensorflow-v1",
      "image classification",
      "cnn",
      "trainable",
      "inference",
      "pre-trained",
      "api-v2"
    ],
    "name": "deep-oc-dogs_breed_det"
  },
  {
    "title": "DEEP OC Massive Online Data Streams",
    "summary": "Deep learning for proactive network monitoring and security
protection. ",
    "keywords": [
```

[11] ai4-papi: https://github.com/ai4os/ai4-papi

```
      "services",
      "docker",
      "api-v2",
      "trainable",
      "inference",
      "deep learning",
      "keras",
      "tensorflow"
    ],
    "name": "deep-oc-mods"
  },
  {
    "title": "DEEP OC Retinopathy Test",
    "summary": "A Tensorflow model to classify Retinopathy.",
    "keywords": [
      "tensorflow",
      "docker",
      "deep learning",
      "trainable",
      "inference",
      "pre-trained",
      "image classification",
      "api-v2"
    ],
    "name": "deep-oc-retinopathy_test"
  },
  {
    "title": "Train an image classifier",
    "summary": "Train your own image classifier with your custom dataset. It comes
also pretrained on the 1K ImageNet classes.",
    "keywords": [
      "tensorflow",
      "docker",
      "deep learning",
      "trainable",
      "inference",
      "pre-trained",
      "image classification",
      "api-v2",
      "general purpose"
    ],
    "name": "deep-oc-image-classification-tf"
  },
  {
    "title": "Plants species classifier",
    "summary": "Classify plant images among 10K species from the iNaturalist
dataset.",
    "keywords": [
      "tensorflow",
      "docker",
      "deep learning",
      "trainable",
      "inference",
      "pre-trained",
      "image classification",
      "api-v2"
    ],
    "name": "deep-oc-plants-classification-tf"
  },
  …
]
```

El portal web AI4EOSC *Dasboard* utiliza oscar-js como una dependencia adicional del proyecto. Una vez instalado, se obtiene acceso a funciones, clases y tipos que se utilizan para interactuar con un clúster de OSCAR. Se crean nuevos componentes en el *dashboard* que pueden hacer uso de las funciones proporcionadas por oscar-js. Al utilizar alguna de estas funciones, es necesario especificar el *endpoint* del clúster de OSCAR sobre el cual

se quiere operar. Esto permite que el portal pueda crear servicios en cualquier clúster de OSCAR que proporcione un *endpoint* conocido. Se proporcionan más detalles sobre oscar-js en la sección 3.3.

3.1 Despliegue local de OSCAR

Para comenzar el desarrollo de este proyecto, primero se realiza la instalación de un clúster de Kubernetes en un entorno local. En este clúster, se implementa la plataforma OSCAR. Además, se examinan todos los componentes clave de Kubernetes y se verifica su correcta instalación mediante una interfaz de usuario. Para iniciara se opta por Kind [12], una herramienta que facilita la creación de un clúster de Kubernetes utilizando contenedores de Docker como nodos. Se sigue detenidamente la guía de instalación proporcionada por kind para este propósito [34]. Una vez que el clúster de kind está instalado y configurado, se procede con la instalación de los componentes de OSCAR. Para ello, se siguen los pasos detallados en la guía de instalación local, disponible en la documentación oficial [35].

Se puede comprobar la instalación correcta de todos los componentes de OSCAR, buscando la configuración "kubeconfig" para el clúster desplegado anteriormente, usando el comando en la consola:

```
kind get kubeconfig --name <cluster-name>
```

Este comando muestra en consola, la configuración de *kind* que permite conectar y acceder el clúster utilizando Lens[13]. Esta herramienta brinda una interfaz fácil de usar para monitorear y administrar el clúster de Kubernetes. Todos los objetos se aprecian en la Figura 7, objetos creados y disponibles en clúster local de Kubernetes

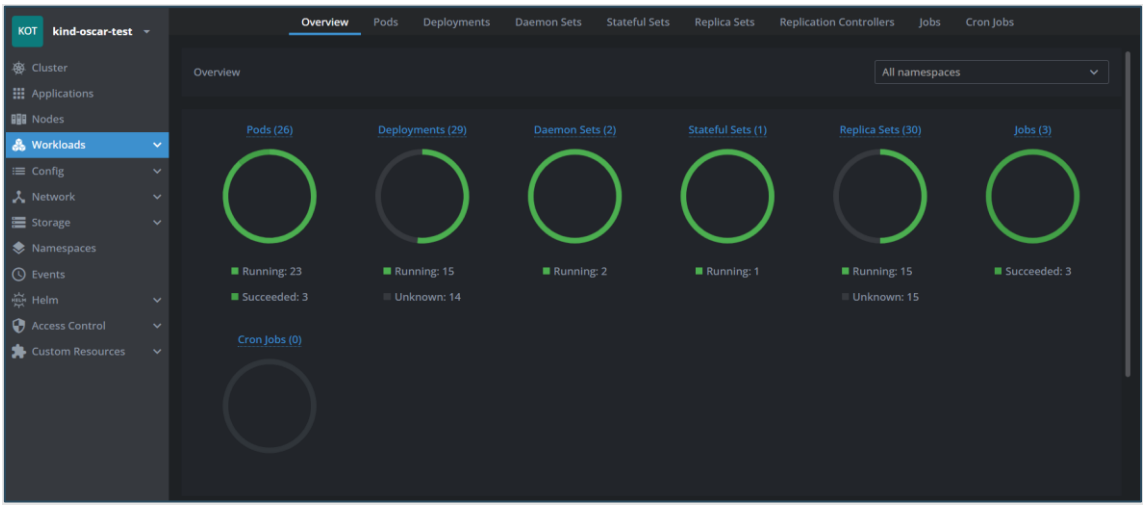

Figura 7. Componentes desplegados en Lens

[12] Kind: https://kind.sigs.k8s.io/

[13] Open Lens: https://k8slens.dev/

Una funcionalidad destacada de Lens es su capacidad para poder mostrar todos los *namespaces* creados. Estos *namespaces* organizan los *pods* que trabajan en conjunto con la plataforma OSCAR. La lista de *namespaces* se puede visualizar en Lens como se muestra en la Figura 8.

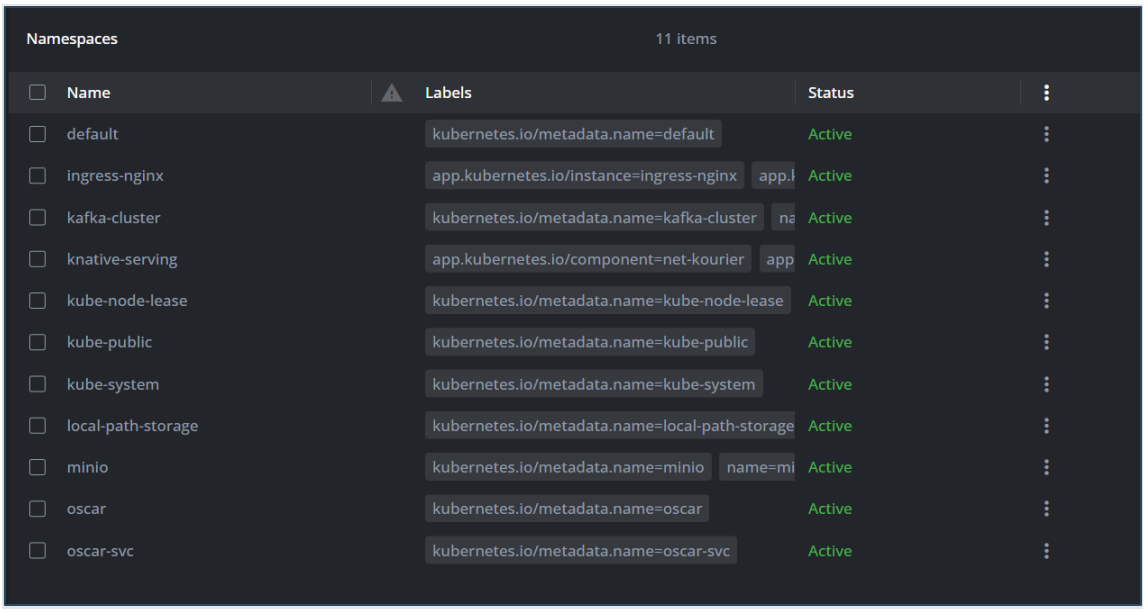

Figura 8. Lista de namespaces en Lens

Es importante prestar atención al *namespace* denominado 'oscar-svc', pues es donde se crean los *pods* que ejecutan las cargas de trabajo asignadas, ya sea una ejecución síncrona o asíncrona. Esta ejecución se lleva a cabo mediante un *"shell script"* que se ejecuta desde dentro del contenedor y provoca la ejecución de la aplicación.

Después de tener todo desplegado, se procede a explorar los objetos de Kubernetes desplegados en el clúster. Es crucial verificar su accesibilidad mediante las interfaces de usuario disponibles para las aplicaciones. Se puede observar una sencilla interfaz de usuario para gestionar el almacenamiento de objetos de MinIO, así como la administración de los servicios de OSCAR a través de su propia interfaz de usuario.

Por ejemplo, accediendo en el navegador a la ruta http://localhost:30301/tools/metrics (como se muestra en la Figura 9), se puede apreciar un panel con métricas de los objetos existentes de MinIO.

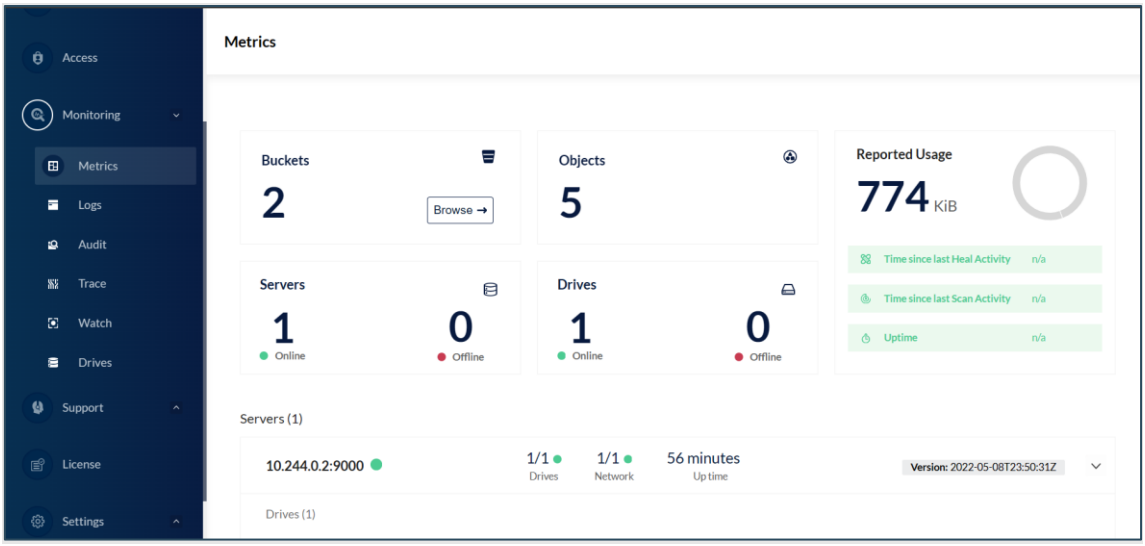

Figura 9. Control de métricas MinIO

También se puede verificar el ingreso a la interfaz de usuario de OSCAR a través de la ruta https://localhost:80/ui/#/services. La Figura 10 representa la página principal en la que se pueden administrar los servicios creados.

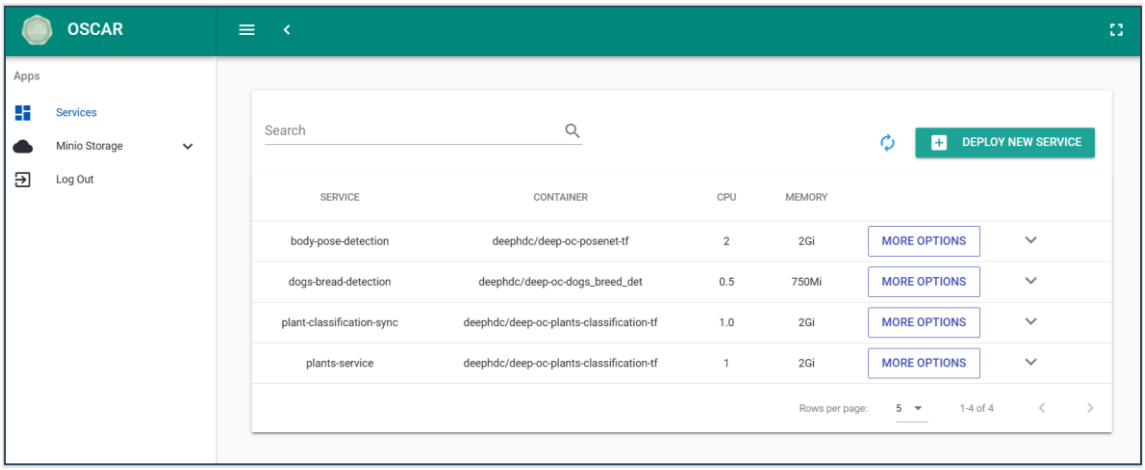

Figura 10. Administración de servicios de OSCAR

3.2 Creación de servicios en OSCAR

Para comenzar, es fundamental comprender cómo se emplea OSCAR para la creación y ejecución de servicios, así como para obtener una lista de los servicios creados. OSCAR proporciona una API segura que se implementa en el nodo maestro del clúster de Kubernetes. El acceso al API es a través de un "Ingress", un componente de Kubernetes diseñado para permitir el acceso a los servicios del clúster y distribuir la carga entre ellos, típicamente es utilizado para el tráfico HTTP (Protocolo de Transferencia de Hipertexto) [36].

3.2.1 Servicio

Se trata de una entidad de OSCAR que representa la creación de una aplicación dentro del clúster. Pueden existir varias instancias de un servicio, y la diferencia principal radica en que cada servicio utiliza una imagen Docker asociada a un modelo de IA específico. Cada servicio está definido por varios atributos, cuya estructura se puede visualizar en el Anexo 1 de este documento. Las peticiones para interactuar con los servicios son las siguientes:

- **Petición HTTP GET /system/services**: Obtiene la lista de los servicios existentes dentro del clúster.
- **Petición HTTP GET /system/services/{serviceName}:** Obtiene información de un servicio filtrando por su nombre, enviando el nombre de servicio como parámetro.
- **Petición HTTP POST /system/services:** Crea un nuevo servicio en el clúster. Para crear un servicio como mínimo se deben enviar en el cuerpo de la petición las propiedades nombre, imagen, memoria, CPU y script.
- **Petición HTTP PUT /system/services:** Actualiza un servicio existente. Para ello, se deben enviar como mínimo las propiedades nombre, imagen, memoria, CPU y script en el cuerpo de la petición.
- **Petición HTTP DELETE /system/services/{serviceName}:** Elimina un servicio existente indicándole el nombre de servicio como parámetro.

3.2.2 Ejecución

Las ejecuciones en OSCAR pueden ser síncronas o asíncronas [37].

- **Ejecución asíncrona**: Este tipo de ejecución representa la ejecución basada por eventos, y, por tanto, se inicia cuando se ha subido un fichero en el *bucket* de entrada del proveedor de almacenamiento. MinIO como proveedor de almacenamiento se encarga de administra los volúmenes y ficheros asociados a los directorios de entrada y salida del servicio. Una vez depositado el fihero en el *bucket,* se genera un evento que provoca la creación de un trabajo de Kubernetes que termina en una invocación del servicio. Para realizar este tipo de ejecución, se utiliza el método **HTTP POST** en la ruta "/job/{serviceName}". Los datos resultantes se almacenarán en el directorio de salida configurado en el servicio y que se guardan en MinIO.

- **Ejecución síncrona**: Esta ejecución consiste en realizar una llamada directamente al servicio, proporcionando los datos de entrada que sean necesarios. Se realiza mediante la petición **HTTP POST** en la ruta "/run/{serviceName}", y permite obtener el resultado de la ejecución como respuesta. Las solicitudes a esta ruta invocan los servicios OSCAR y desencadenan la creación de un nuevo pod para el servicio correspondiente como se puede ver en la Figura 11. En este caso Knative Serving [38] actúa respondiendo a la carga entrante en el servicio y crea automáticamente los pods necesarios para responder a las solicitudes. Una vez se deja de recibir tráfico, Knative elimina los pods que no son necesarios, optimizando recursos. Cuando la ejecución ha finalizado, **la respuesta se entrega codificada en base64** y puede variar su contendido dependiendo del servicio y del script utilizado. Los tipos de salida más comunes incluyen imágenes, archivos comprimidos en formato zip o archivos JSON.

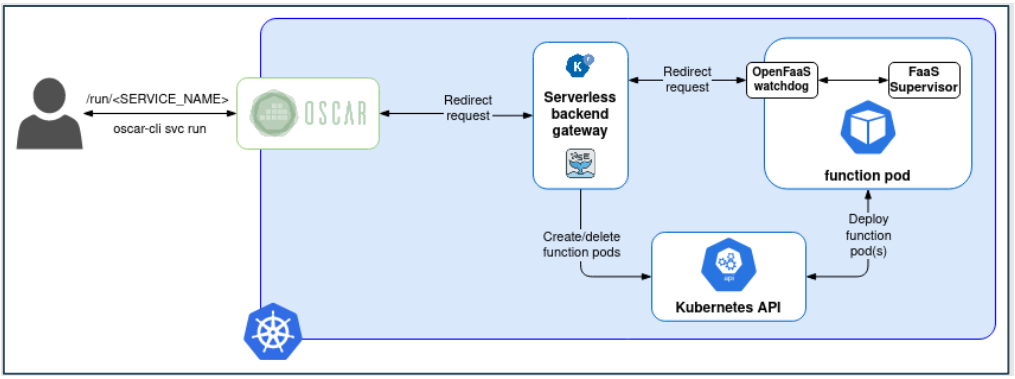

Figura 11. OSCAR invocación síncrona [14]

Para probar la ejecución síncrona, se ha creado un nuevo servicio en OSCAR llamado "plant-classification-sync". Para hacerlo, se emplea una terminal de Linux junto con la herramienta *curl*, que permite realizar solicitudes a servidores usando métodos HTTP. El comando utilizado para invocar el servicio es el siguiente:

```
base64 images/arbol-roble.jpg | curl --request POST
'http://localhost/run/plant-classification-sync' --header
'Authorization: Bearer
0078734d58ac07da88b7c616a3262c867de61afe454e1ad44a86048ae
2d6aa6e' --data @-
```

En la Figura 12 se muestra la respuesta de la ejecución, la cual es exitosa y devuelve el contenido codificado en base64.

[14] Tomado de la web: https://docs.oscar.grycap.net/invoking/

Figura 12. Respuesta invocación síncrona codificada en base64

3.3 Modulo cliente oscar-js

El módulo de oscar-js se destaca como una de las aportaciones más significativas en este proyecto, dado que su creación es esencial para alcanzar las metas establecidas. Por lo tanto, es crucial en el despliegue y ejecución de servicios sobre distintos clústeres de OSCAR. Es importante mencionar que OSCAR ya disponía de un cliente diseñado para el lenguaje de programación Python (oscar_python)[15]. Sin embargo, este cliente no era compatible con la integración de las funciones de OSCAR en el AI4EOSC *Dashboard*, que está desarrollado con tecnología Angular. Por lo tanto, fue necesario desarrollar un cliente de OSCAR específicamente para JavaScript/TypeScript. El código de oscar-js se puede encontrar en GitHub como un repositorio público [39].

Iniciando con el desarrollo, se crea un proyecto básico de Node.js que usa Typescript como lenguaje tipado. Una vez creado el proyecto, se accede al archivo "package.json". Este archivo contiene información relevante del proyecto de Node.js como el nombre, descripción, licencia, autor y versión. Además, en el archivo se especifican las dependencias utilizadas en el proyecto, que son módulos necesarios para su correcto funcionamiento. La sección de scripts define tareas comunes como compilar o desplegar el módulo. La estructura de ficheros del proyecto tiene un aspecto como el que se ve en la Figura 13.

[15] Repositorio oscar_python: https://github.com/grycap/oscar_python

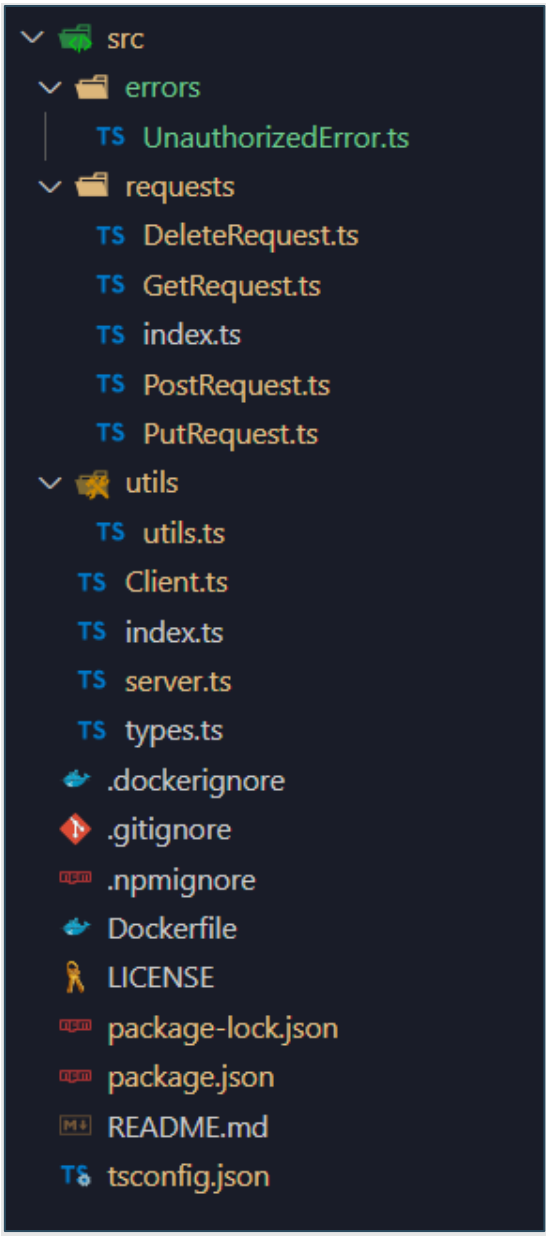

Figura 13 . Estructura de ficheros de oscar-js

En el proyecto, un archivo de configuración crucial es el "tsconfig.json". Este archivo almacena la configuración necesaria para convertir el código de TypeScript a JavaScript durante el proceso de compilación. En el fichero se especifica cómo se debe comportar el módulo y opciones del compilador. La sección “compilerOptions” define opciones de compilación del código importantes para que el módulo sea compatible con las versiones más recientes de Node.js. Por ejemplo, una propiedad importante es “esModuleInterop”, ya que permite la interoperabilidad entre módulos antiguos y los más nuevos, permitiendo que el módulo funcione con cualquiera de éstos. También está la propiedad “target”, esta indica en qué versión de ECMAScript [40] se debe compilar el código, en nuestro caso la más reciente “ES2022”. En la se Figura 14 puede observar la configuración del fichero “tsconfig.json” para oscar-js.

Figura 14. Fichero tsconfig.json para oscar-js

Explicando un poco el contenido de la *Figura 13*, se ubica el directorio "requests" en este se agrupan todas las clases relacionadas con los métodos HTTP. Por lo tanto, se encuentran las clases “GetRequest.ts”, “PostRequest.ts”, “PutRequest.ts” y “DeleteRequest.ts”. Continuando con el proyecto, se establece el directorio "utils", donde se crea la clase “utils.ts”. Esta clase exporta funciones de utilidad globales que pueden ser utilizadas desde otros archivos, promoviendo así la centralización y reutilización del código. Avanzando en el desarrollo, se crea el directorio de errores, donde se definen errores personalizados. En este caso, se ha creado la clase “UnauthorizedError.ts”.

Posteriormente, se crea la clase más importante, denominada “Client.ts”, la cual se exporta y contiene todas las funciones que pueden ser importadas después de instalar oscar-js como dependencia. Además de “Client.ts”, hay archivos que no están ubicados en ningún directorio, como “types.ts”. Este archivo se encarga de exportar todos los tipos de datos utilizados en el proyecto, los cuales corresponden a los esquemas empleados en OSCAR. Por ejemplo, existe el esquema de *Service* encargado de establecer las posibles propiedades y tipos que tiene un servicio, otros ejemplos de esquemas son *Info, StorageProviders, JobInfo,* entre otros. En la Figura 15 se puede ver un ejemplo de los esquemas de *Service* y *StorageIOConfig*.

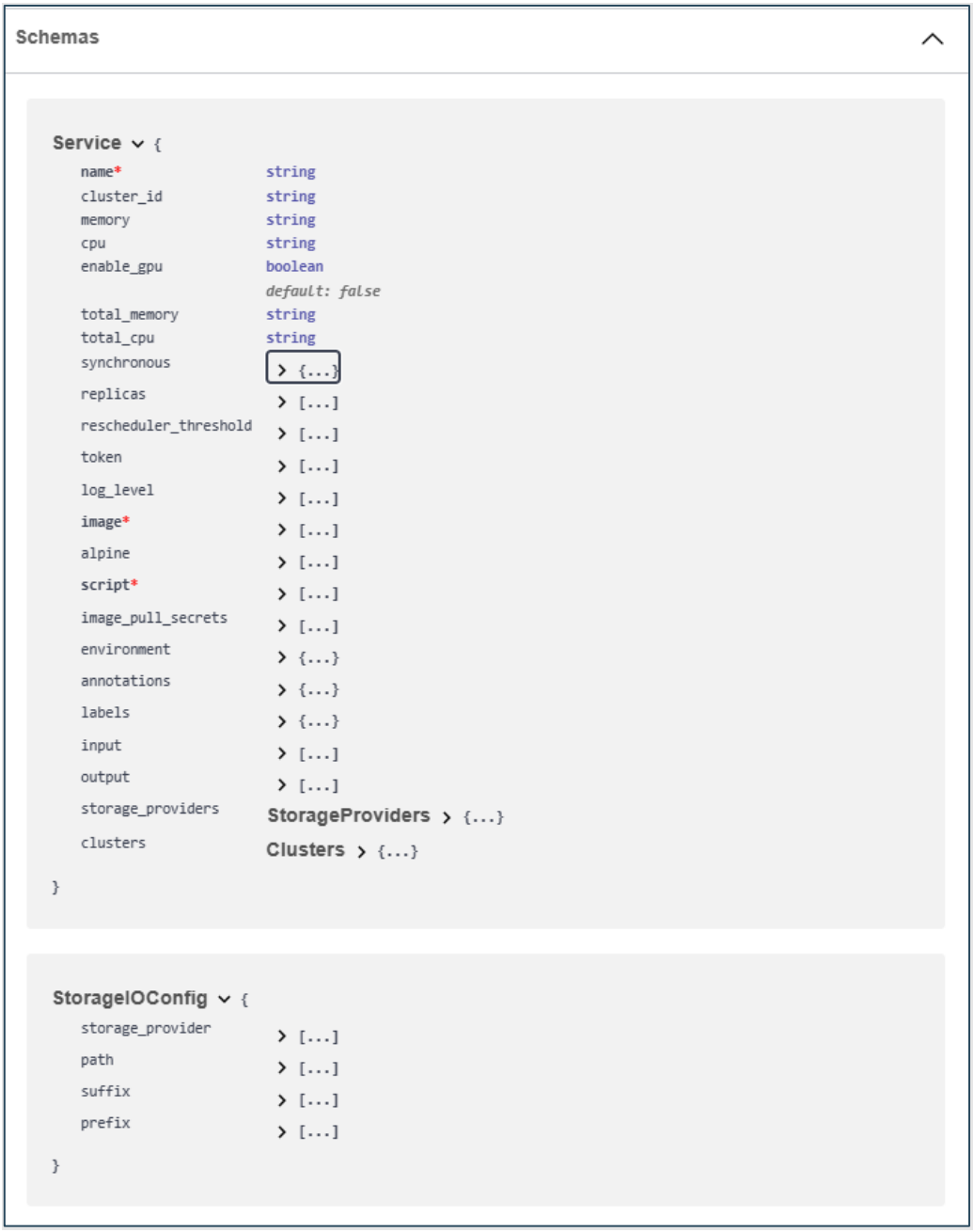

Figura 15. Esquemas de OSCAR presentados en OpenAPI

3.3.1 Client.ts

Esta clase representa el cliente para comunicarse con un clúster de OSCAR. Para crear una instancia cliente se envía como parámetro del constructor de clase un objeto de tipo "ClusterAuth" se puede ver el tipo en la Figura 16.

```typescript
export type ClusterAuth = {
    clusterId: string;
    oscar_endpoint: string;
    username?: string;
    password?: string;
    oidc_token?: string;
};
```

Figura 16. Definición de tipos para el objeto ClusterAuth

En el constructor se evalúa que existan y tengan valores las propiedades *username* y *password,* porque de esta forma se construirá el cliente usando la autenticación básica, que es una forma de autenticación en la que se envían a la red los datos de usuario contraseña codificados en base64 [41]. En el caso de que exista una propiedad *oidc_token,* se usará este como parámetro como "autenticación de portador", también conocida como "Bearer Authentication" [42] . Este es otro tipo de autorización en el que se pasa una cada codificada llamada token, generalmente generada por el servidor en respuesta a una solicitud de inicio de sesión, dicho token se debe enviar en cada solicitud como una propiedad del encabezado de la solicitud [41].

Otra propiedad muy importante en este objeto es "oscar_endpoint", ya que aquí se guarda la ruta al clúster de OSCAR al que se harán las peticiones. Es esencial asegurarse de que las credenciales, como *username* y *password*, o el *oidc_token*, correspondan a la dirección URL del clúster de referencia. Si las credenciales no son correctas, recibiremos un mensaje de error indicando falta de autorización. En la Figura 17 se puede ver el ejemplo de creación de una instancia del cliente usando la autenticación básica.

```typescript
const basic_auth = {
    clusterId: "cluster-id",
    oscar_endpoint: "https://my-oscar-cluster",
    username: "oscar",
    password: "secret",
};

const oscar_client: Client = new Client(basic_auth);
```

Figura 17. Instancia de oscar_client con Basic Auth

Una vez se crea una instancia de la clase, es posible acceder a los métodos de ésta. A continuación, se explican los más relevantes y se describe el funcionamiento de los más básicos. La Tabla 1 recoge toda esta información, donde aparecen organizados y documentados los nombre de los métodos, descripción y tipo de dato que devuelven. La mayoría de ellos retornan una promesa de JavaScript, un objeto que representa un valor en el futuro que puede ser ahora, en el futuro o nunca. Normalmente se utiliza en

funciones asíncronas que hacen peticiones a servidores, pues permiten controlar el flujo de una forma ordenada y sencilla [43].

Nombre	Descripción	Tipo de dato retornado
getEndpoint	Retorna el valor de la URL del clúster de OSCAR que se utilizó para crear la instancia	string
getUsername	Retorna el valor de la propiedad *username*	string | undefined
getPassword	Retorna el valor de la propiedad *password*	string | undefined
getOidcToken	Retorna el valor de la propiedad *oidc_token*	string | undefined
getAuthType	Retorna el tipo de autenticación usada, con base en los parámetros existentes en la instancia	Oidc | BasicAuth
getClusterInfo	Obtiene información del clúster enviando una solicitud GET */system/info*	Promise<Info>
getClusterConfig	Obtiene información sobre la configuración del clúster haciendo una solicitud GET */system/config*	Promise<Config>
getHealthStatus	Obtiene información de la salud del clúster, se envía la solicitud GET */health*	Promise<string>
getServices	Retorna una lista de los servicios creados actualmente en el clúster, se hace una solicitud GET */system/services*	Promise<Service[ ]>
getServiceByName	Retorna un servicio buscando por su nombre de servicio, se envía como parámetro el nombre del servicio, se hace una solicitud GET */system/services/{serviceName}*	Promise<Service>
createService	Crea un nuevo servicio en el clúster de OSCAR, se recibe como parámetro un objeto de tipo *Service* que se enviará como el *body* de la solicitud POST */system/service*	Promise<Service>
updateService	Actualiza un servicio existente en el clúster de OSCAR, se recibe como parámetro un objeto de tipo *Service* que se enviará como el *body* de la solicitud PUT */system/service*	Promise<Service>
runService	Ejecuta el servicio de forma síncrona, se recibe por parámetro el servicio que quiere ser ejecutado y los datos para la inferencia codificados en base64, se hace una solicitud POST */run/{serviceName}*	Promise<any>
deleteService	Elimina un servicio existente en el clúster, se recibe por parámetro el nombre del servicio que quiere ser eliminado, se hace la solicitud a DELETE */system/services/{serviceName}*	Promise<string>

Tabla 1. Lista de métodos disponibles en la clase Client.ts

Se puede ver con más detalle todos los métodos de la clase *Client.ts* en el Anexo 2 de este documento.

Para realizar la ejecución síncrona del servicio OSCAR, se necesita enviar un *Bearer Token* en el encabezado de autorización de la solicitud. Este token debe corresponder al del servicio en cuestión. Es decir, la solicitud al clúster se hará utilizando este token en

lugar del *oidc_token*, y no es posible hacerlo utilizando autorización básica. El token del servicio se puede encontrar en la información del servicio, que se puede obtener usando el método *getServiceByName* o buscándolo de forma manual en la interfaz de usuario de OSCAR. En la Figura 18 se muestra dónde encontrar dicho token de servicio.

Figura 18. Verificación de token de servicio

La ventaja de utilizar tokens de servicios OSCAR de larga duración es que posibilita el uso de inferencia de modelos para usuarios externos, que no estén autenticados en el clúster OSCAR, siempre que el propietario del servicio así lo estime oportuno. Por ejemplo, esto permite que usuarios que naveguen por el AI4EOSC *Dashboard* puedan ejecutar rápidamente la inferencia de un modelo de IA sin necesidad de ser los propietarios del servicio OSCAR.

3.3.2 GetRequest.ts

Esta clase, como su nombre indica, se encarga de realizar una solicitud de tipo GET al *endpoint* del clúster de OSCAR previamente especificado. Para llevar a cabo este tipo de peticiones, se utiliza la dependencia "node-fetch", una biblioteca que proporciona varias herramientas para realizar solicitudes HTTP. Para este caso haremos peticiones al API expuesta por OSCAR a través de su *ingress*. Con la ayuda de "node-fetch", se pueden incluir cabeceras, como la autorización, tipo de contenido, entre otras. La Figura 19 muestra un fragmento de código que ilustra cómo se crea una solicitud GET y se guarda la respuesta en una constante llamada 'response'.

```
const headers = {
  'Content-Type': 'application/json',
  'Authorization': auth
}

const response = await fetch(url, {
  method: 'GET',
  headers: headers
});
```

Figura 19. Código fuente creación solicitud GET

En la respuesta de la solicitud, hay propiedades que indican si la petición se completó correctamente o si ocurrió algún error. Esto se puede determinar al acceder a la propiedad *status* de la constante "response". Por un lado, si hay algún tipo de error, se lanzará una

excepción específica según el error detectado. Por otro lado, si la respuesta es exitosa, se devuelve la respuesta de la solicitud utilizando el método *response.json()* o *response.text()*, según sea necesario. Para ver en detalle del código sobre cómo se genera la url y datos de autorización puede encontrarse en el Anexo 3 de este documento.

3.3.3 PostRequest.ts

Al igual que en la solicitud anterior, se emplea la dependencia "node-fetch" para realizar las solicitudes, pero en esta ocasión, se utiliza el método POST en lugar del GET. Además, se incluye un parámetro del cuerpo (*body*) en la solicitud, el cual contiene el contenido necesario para interactuar con el servidor. Es importante destacar que el *body* de la solicitud lleva la información que el servidor necesita para procesar la petición de manera adecuada. La imagen a continuación Figura 20 muestra cómo se crea la petición POST haciendo uso del método *fetch()* proporcionado por la biblioteca "node-fetch".

```
const response = await fetch(url, {
    method: "POST",
    headers,
    body: JSON.stringify(body),
});
```

Figura 20. Código fuente creación solicitud POST

Para ver en detalle del código sobre cómo se genera la URL y datos de autorización para la clase *PostRequest* puede encontrarla en el Anexo 4 de este documento.

3.3.4 PutRequest.ts

En este caso, se hace una solicitud de tipo PUT que es muy parecida al método anterior, para crear la petición usando la función *fetch()* se hace especificando el tipo de método usado para la petición, En la imagen Figura 21 se deja el fragmento de código.

```
const response = await fetch(url.toString(), {
    method: "PUT",
    headers,
    body: JSON.stringify(body),
});
```

Figura 21. Código fuente creación solicitud PUT

Para ver en detalle del código sobre cómo se genera la URL y datos de autorización para la clase *PutRequest* puede encontrarla en el Anexo 5 de este documento.

3.3.5 DeleteRequest.ts

Esta clase, como su nombre indica, se encarga de realizar una solicitud de tipo DELETE al *endpoint* del clúster de OSCAR previamente especificado. Para poder eliminar datos la mediante la API se hace utiliza la función *fetch()* especificando en este caso el tipo de

método DELETE. En la Figura 22 se deja un fragmento del código usado para crear la solicitud de tipo delete.

```
const response = await fetch(url.toString(), {
    method: "DELETE",
    headers,
});
```

Figura 22, Código fuente creación solicitud DELETE

Para ver en detalle del código sobre cómo se genera la URL y datos de autorización para la clase *DeleteRequest* puede encontrarla en el Anexo 6 de este documento.

3.3.6 Utils.ts

En este fichero de Typescript se declaran funciones de utilidad que van a ser utilizadas por las diferentes clases. Este fichero comparte varias funciones importantes, por ejemplo, existe una función llamada *setAuthParams()* que crea la cabecera de autorización según corresponda, también están las funciones para codificar o decodificar el contenido en base64 y una función muy importante *getMimeTypes()* utilizada para identificar el *MimeType* desde una cadena codificada en base64. El tipo de *MIME* (también conocido como media *type* o tipo de medio) es una cadena de texto que identifica un formato de archivo especifico.

Para determinar la extensión o tipo de archivo de la respuesta, se utilizan **los bytes mágicos,** también conocidos como **firma del archivo**. Estos bytes, que son los primeros del archivo, proporcionan información sobre el tipo de archivo en la mayoría de los casos. Para obtener una referencia de los bytes mágicos correspondientes a los diferentes tipos de archivo, se puede consultar una lista en Wikipedia [44]. Este enfoque permite identificar de manera precisa el tipo de archivo sin depender únicamente de la extensión del nombre del archivo.

Después de realizar una invocación síncrona en OSCAR, la respuesta se recibe codificada en base64, tal cómo se mencionó en la sección 3.2.2 de este documento. Esta es la razón por la que se utilizan los bytes mágicos, que intentan detectar el contenido *MIME* del contenido codificado. Tras una ejecución exitosa, OSCAR devuelve el contenido codificado en base64. Este contenido se pasa como argumento a la función *getMimeTypes()*, la cual verifica los primeros bytes y devuelve el tipo de *MIME* detectado. En la Figura 23 se muestra la implementación de este método, el cual tiene un mapa de posibles bytes y sus correspondientes tipos de *MIME*. En este casos si se encuentra una coincidencia, devuelve una cadena con el tipo de *MIME*; de lo contrario, devuelve un tipo por defecto.

```typescript
export function getMimeType(base64String: string): string {
    const decodedString = base64ToHex(base64String);
    const signature = decodedString.substring(0, 16).toUpperCase();
    console.info("Signature: ", signature);

    const signatures: { [signature: string]: string } = {
        "EFBBBF": "text/plain", // TXT
        "89504E470D0A1A0A": "image/png", // PNG
        "47494638": "image/gif", // GIF
        "FFD8FF": "image/jpeg", // JPEG
        "504B0304": "application/zip", // ZIP
        "667479704D534E56": "video/mp4", // MP4
        "52494646": "audio/wav", // WAV
        "494433": "audio/mpeg", // MP3
        // Añadir más firmas según sea necesario
    };

    for (const [signatureBytes, mimeType] of Object.entries(signatures)) {
        if (signature.startsWith(signatureBytes)) {
            return mimeType;
        }
    }

    // Tipo MIME por defecto para datos binarios
    return "application/octet-stream";
}
```

Figura 23. Función para determinar la firma y asignar tipo de MIME

3.4 Módulo de despliegue del servicio - "Deploy via OSCAR"

En esta sección, se detalla el proceso de desarrollo e implementación de la nueva funcionalidad en el portal web del *dashboard* AI4EOSC, centrada en la capacidad de desplegar modelos sobre OSCAR mediante la creación de servicios. Se aborda el análisis completo desde el inicio del proyecto, incluyendo la etapa de diseño con la elaboración de bocetos detallados. Se profundiza en la creación de nuevos componentes y formularios, elementos clave para lograr la funcionalidad propuesta. Además, se comentan las consideraciones técnicas que influyeron en el desarrollo de la solución, así como las pruebas y ajustes realizados para garantizar su efectividad y usabilidad.

Antes de continuar, es importante mencionar que el proyecto del portal web (AI4EOSC *Dashboard*) ya está en curso y es gestionado por el Instituto de Física de Cantabria (IFCA) del Consejo Superior de Investigaciones Científicas (CSIC). Por lo tanto, el desarrollo de estas funcionalidades ha debido adaptarse al diseño y la experiencia de usuario que el portal busca mantener.

En el proceso de desarrollo del portal web, se opta por utilizar Angular como *framework* principal. Esta elección se basa en la capacidad de Angular para crear componentes de manera eficiente y organizar la lógica de la aplicación en módulos independientes. Para comenzar, se inicia con la familiarización e investigación de funcionalidades del portal. Durante esta exploración se encuentran los distintos módulos de la aplicación. En este caso se pone especial foco en los componentes dentro del módulo de *marketplace*. Cada componente en Angular se compone de cuatro ficheros, el código HTML, un fichero de TypeScript, hoja de estilos y fichero para pruebas.

En primer lugar, se encuentra el archivo HTML, el cual contiene el *template* o estructura visual del componente. En este archivo se define la disposición de los elementos que se mostrarán en la interfaz de usuario. Luego, el archivo TypeScript (".ts"), donde se define la lógica y la funcionalidad del componente. En este se especifican los métodos, las propiedades y eventos que controlan el comportamiento del componente. Además, está el archivo de prueba (“spec.ts"), que se usa para escribir pruebas unitarias para el componente, asegurando su correcto funcionamiento. Por último, está el archivo de estilos (".css" o “.scss"), donde se define la apariencia del componente mediante reglas de estilo y diseño. Estos cuatro archivos trabajan en conjunto para definir y dar vida al componente dentro de la aplicación Angular.

Dentro de los componentes de *marketplace* se encuentran los siguientes:

- **module-list:** Es un componente que se encarga de mostrar en forma de tarjetas los modelos de IA disponibles. Cada tarjeta muestra el nombre, descripción y los tags del modelo. La Figura 24 a continuación, muestra la vista relacionada con este componente.

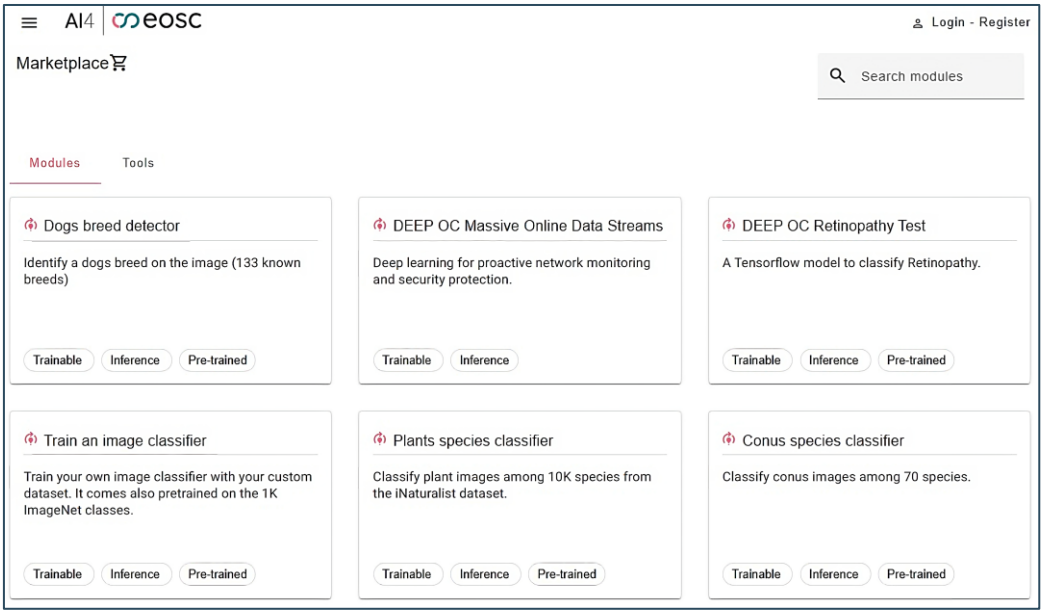

Figura 24. Lista de módulos en el martkeplace

- **module-detail:** Este componente tiene la información más detallada sobre un modelo en específico. Como, su descripción, datos de entrenamiento, repositorio de Docker, etc. La Figura 25 representa este componente.

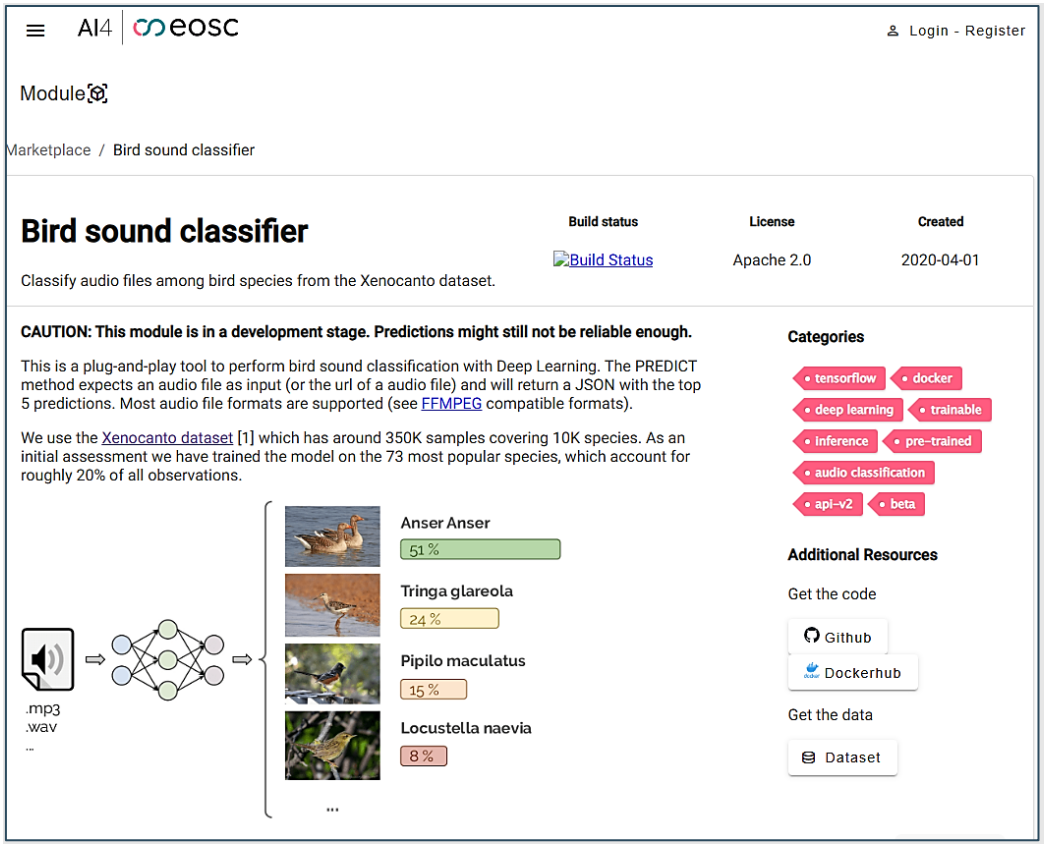

Figura 25. Detalle del modelo "Bird sound classifier"

- **module-train:** Este componente consiste en un formulario para rellenar datos y hacer un entrenamiento personalizado del modelo. Se habilita una vez el usuario ha iniciado sesión. En la Figura 26 se ve que el componente es un formulario guiado por pasos.

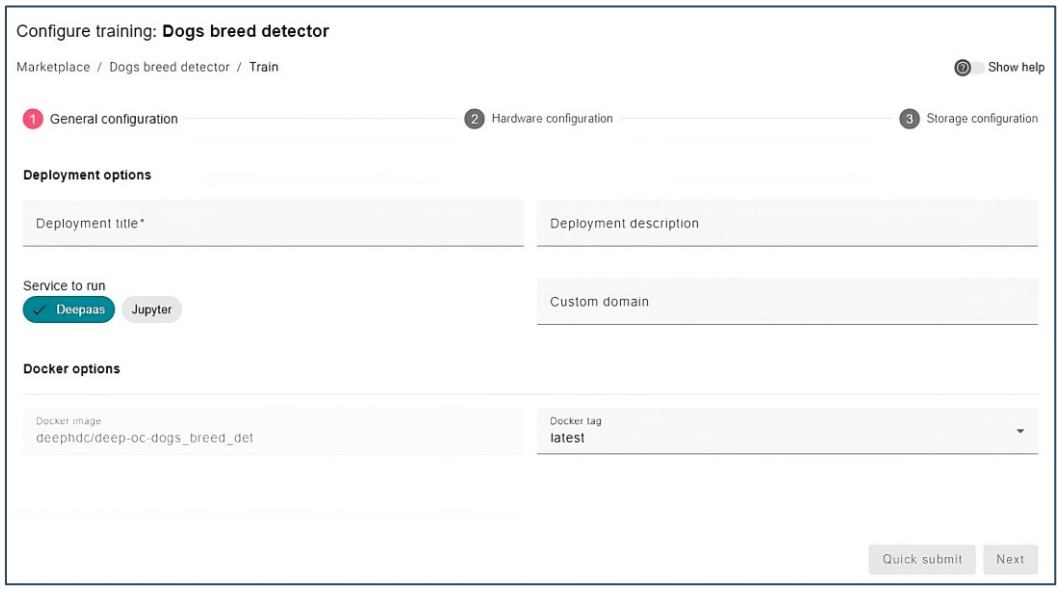

Figura 26. Formulario relacionado con el entrenamiento de un modelo

3.4.1 Bocetos de diseño

Basándose en los datos necesarios para crear el servicio en OSCAR, se ha calculado que el formulario constará de tres puntos. En el primer paso se solicitará la información general, en el segundo paso se pedirá información sobre cómputo y el último paso ofrecerá un resumen de los datos introducidos en los pasos anteriores.

Una vez que se ha definido como se verá el componente, el desarrollo comienza a tomar forma. Para tener una visión clara de cómo debería lucir este componente, se crean bocetos iniciales. Estos bocetos actúan como una guía visual para lo que se implementará más tarde en HTML y CSS. Para mejorar la experiencia del usuario, se ha optado por crear un formulario guiado por pasos. Esta estrategia se utiliza para evitar abrumar al usuario con demasiada información de una sola vez y minimizar los errores [45].

La implementación de un formulario paso a paso, conocido como "Stepper Form", se realiza en Angular utilizando la biblioteca de componentes Angular Material [46]. Esta biblioteca ofrece una amplia gama de componentes fácilmente personalizables y de fácil implementación en el proyecto. Angular Material es la biblioteca de componentes que ya se utilizaba en el portal web de AI4EOSC, por lo que se seguirá utilizando para mantener la armonía del diseño, buscando siempre proporcionar una experiencia coherente al usuario. Por ejemplo, el diseño del componente "module-oscar-deploy" está inspirado en el formulario por pasos utilizado en el componente "module-train", manteniendo así la consistencia en toda la plataforma. A continuación, se detallan los pasos de los que consta el formulario, junto con los bocetos diseñados para los mismos:

1. Configuración general - Formulario despliegue. La Figura 27 representa la vista inicial, el primer paso del formulario. Aquí se solicitan los datos del nombre del servicio, la imagen de Docker correspondiente, opciones para definir variables de entorno y etiquetas, además permite subir un fichero que contiene el script que se ejecuta dentro del contenedor.

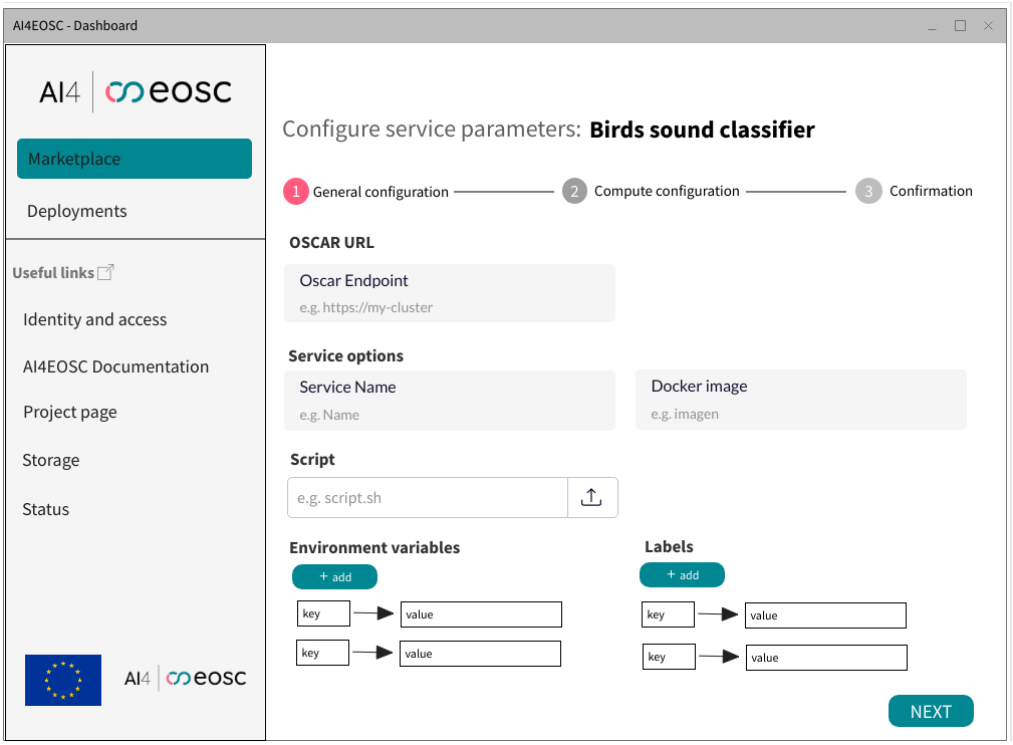

Figura 27. Boceto paso 1 "formulario de despliegue"

2. Configuración computo - Formulario despliegue. La Figura 28 representa la vista y los campos en el paso dos. En este paso se solicitan los datos correspondientes a las capacidades de cómputo para este servicio, como la cantidad de CPU y memoria RAM; también se puede indicar si el modelo necesita de uso de GPU.

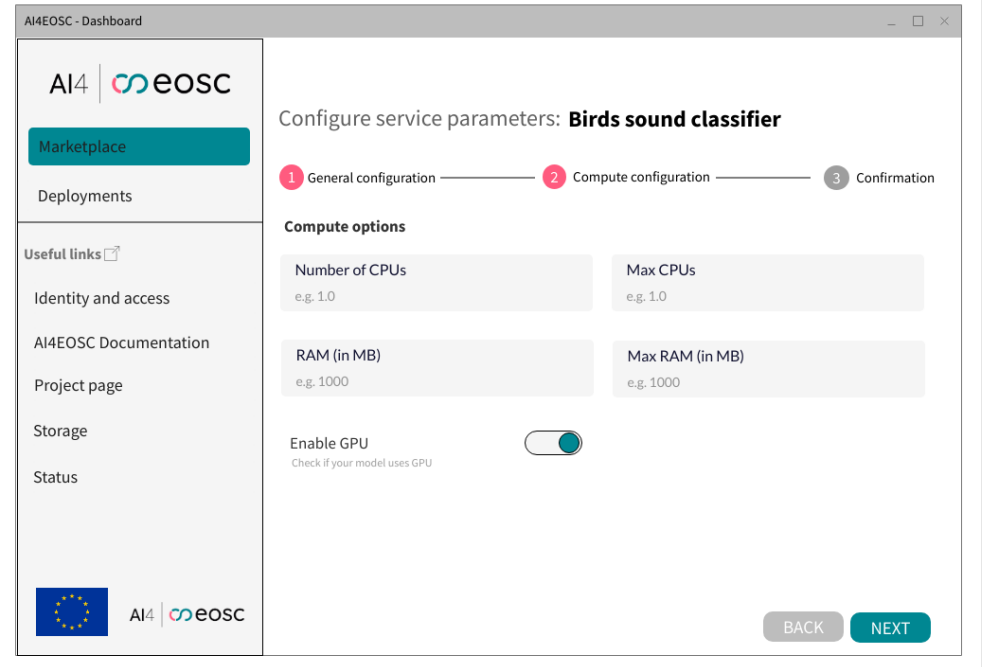

Figura 28. Boceto paso 2 "Formulario despliegue"

3. Confirmación - Formulario despliegue. El paso tres tiene un resumen de los datos introducidos, la imagen de la Figura 29 representa como debería verse este componente.

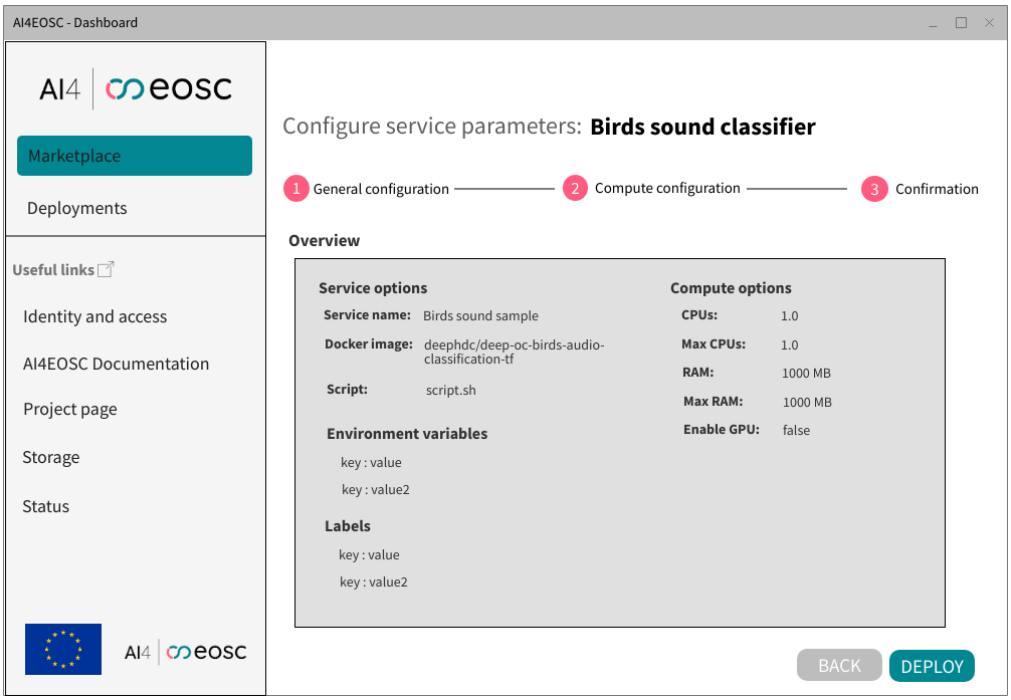

Figura 29. Boceto paso 3 "Formulario despliegue"

3.4.2 Creación del componente

El nuevo componente pretende desplegar los modelos de IA sobre un clúster de OSCAR. Usando las ventajas de Angular para modularizar secciones de la aplicación, se crea el módulo del *Marketplace* en el que se agrupan varios componentes, servicios y configuraciones propias del módulo. El nuevo componente recibe el nombre de **module-oscar-deploy,** se ha creado usando el CLI[16] que proporciona Angular usando el comando:

```
ng generate component modules/marketplace/components module-oscar-deploy
```

Para cada uno de los pasos en el formulario se crean dos nuevos componentes, cada uno representa información para la creación de un servicio en OSCAR. El primer paso tiene configuración general del servicio y se crea usando el siguiente comando:

```
ng generate component modules/marketplace/components/module-oscar-deploy general-config-form
```

Siguiendo con el formulario, el segundo paso se crea para la configuración de cómputo y otros atributos, el componente se crea usando el siguiente comando:

```
ng generates component modules/marketplace/components/module-oscar-deploy compute-config-form
```

[16] Angular CLI: https://angular.io/cli

Los componentes mencionados serán subcomponentes del componente “module-oscar-deploy”. Se emplean para dividir el formulario en varios componentes y tener un código más limpio, modular y adaptable. Para integrar el componente “general-config-form” se debe utiliza el selector del componente para usarlo en el componente padre, tal como lo representa la Figura 30.

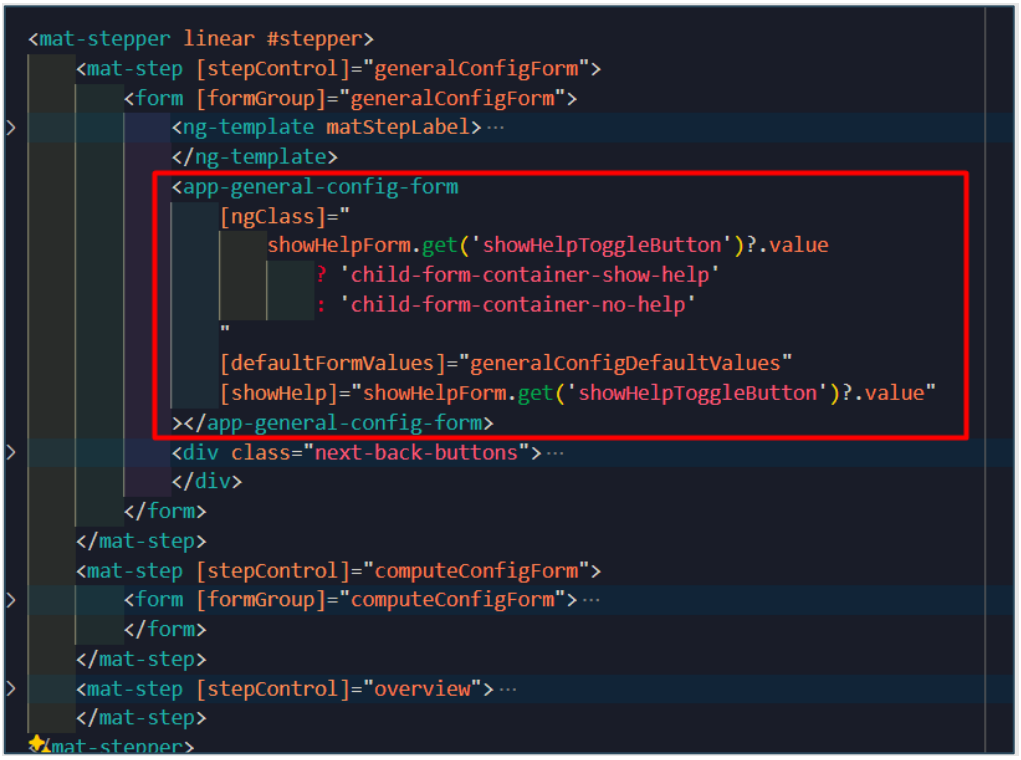

Figura 30. Código fuente uso de selector de componente hijo

En la imagen anterior, se observa el uso de las etiquetas "mat-stepper", "mat-step" y "form", las cuales se combinan para construir un formulario de tipo "stepper-form". Cada "step" o paso del formulario corresponde a una sección específica del proceso. En este caso, se crea un formulario destinado a la configuración general del servicio.

Con lo anterior, la estructura del proyecto cambia y tras la creación de los nuevos componentes, el módulo de *Marketplace* luce como en la Figura 31.

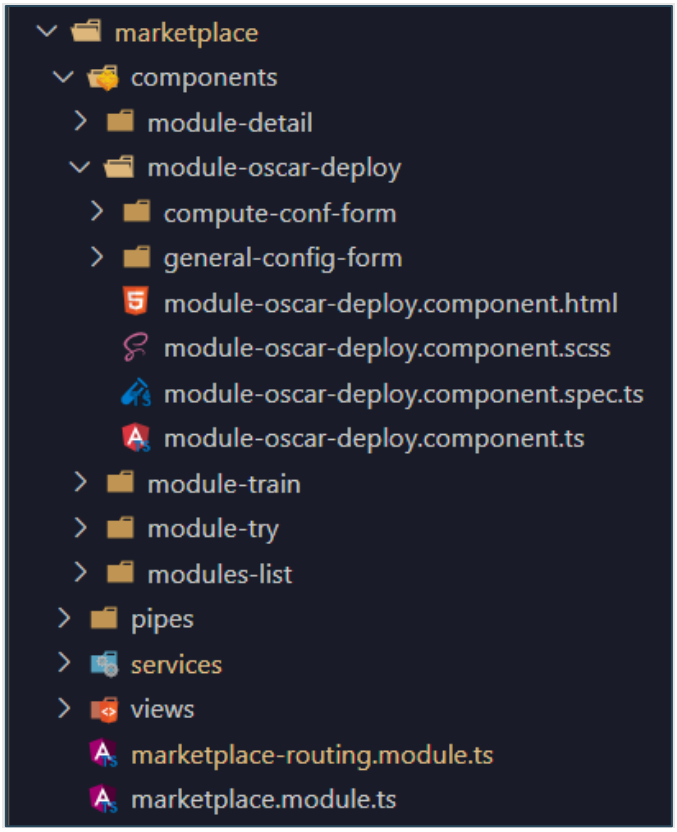

Figura 31. Estructura de ficheros módulo de Marketplace

3.4.3 Integración con oscar-js

En esta sección se detalla la integración entre el desarrollo del paquete "oscar-js" y el nuevo módulo integrado en el AI4EOSC Dashboard. Como se mencionó previamente, dado que el proyecto está modularizado, todas las funcionalidades requeridas se encuentran en el módulo de Marketplace. En Angular, existe un tipo de componente llamado "Service", que actúa como un proveedor que encapsula la lógica y las operaciones relacionadas con el negocio. Estos servicios facilitan la comunicación entre componentes y la interacción con servicios web, como las llamadas a APIs externas [47].

Teniendo esto en cuenta, se emplea el servicio "ModuleService", diseñado específicamente para manejar las operaciones de los componentes del Marketplace. Este servicio realiza solicitudes al API de ai4-papi[17], responsable de suministrar servicios de AI4EOSC, Se han implementado diversas funciones en este servicio para interactuar con ai4-papi. Por ejemplo:

- La función *getModulesSummary()*, que devuelve una lista de todos los modelos disponibles para AI4EOSC.
- La función *getModule()*, que acepta el nombre de un modelo como parámetro y devuelve información detallada sobre el mismo.

[17] API ai4-papi: https://github.com/ai4os/ai4-papi

- Por último, la función *getModuleConfiguration()*, que proporciona información de configuración para un modelo específico cuando se le pasa el nombre del modelo como parámetro.

Una vez completada esta implementación, se realiza la inclusión de "oscar-js" en el proyecto Angular. Para ello, es necesario ejecutar el siguiente comando:

```
npm install @grycap/oscar-js
```

Este comando instalará "oscar-js" como dependencia del proyecto. En el fichero "package.json", se puede verificar que la dependencia ha sido añadida correctamente en la sección de dependencias, como se muestra en la Figura 32.

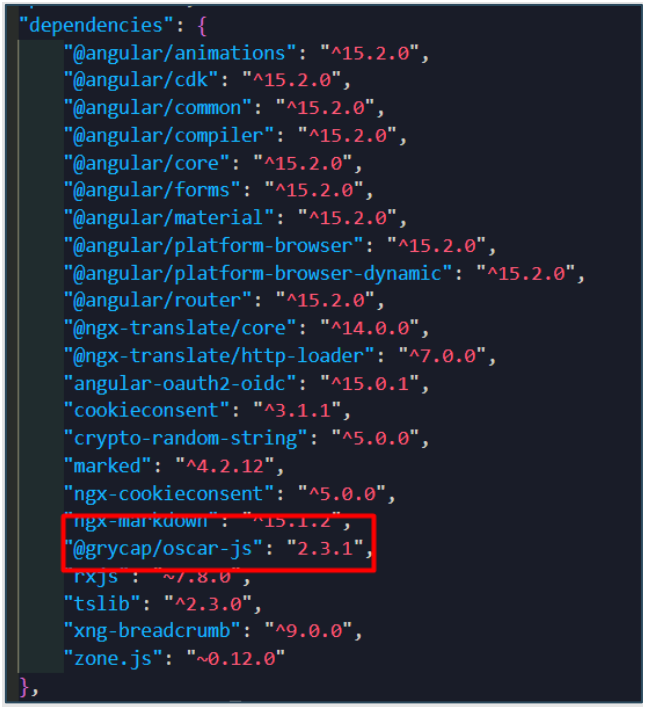

Figura 32. Fichero package.json con oscar-js integrado como dependencia

A partir de este momento se contará con acceso al cliente de "oscar-js". Para ello, será necesario importarlo desde el paquete previamente instalado. El proceso de importación se puede observar en la Figura 33,

```
import { HttpClient, HttpParams } from '@angular/common/http';
import { AppConfigService } from '@app/core/services/app-config/app-config.service';
import { Client } from '@grycap/oscar-js';
import { OAuthStorage } from 'angular-oauth2-oidc';
```

Figura 33. Importación de la clase Client en el AI4EOSC Dashboard

Con esto, se podrá acceder a las funciones creadas en la sección 3.3 de este documento. A partir de este punto, se procederá a desarrollar funciones que permitan ejecutar y crear los servicios desde el portal web. En la Tabla 2 se hace un resumen de las funciones nuevas implementadas en el servicio "ModuleService".

Nombre	Descripción	Tipo de dato retornado
getServices	Utiliza el cliente de oscar-js y llama al método *getServices()* que retornará los servicios creados en un clúster de OSCAR, se pasa cómo parámetro la url del clúster de OSCAR	Promise<Service []>
createService	Esta función crea una instancia del cliente de oscar-js y llama al método *createService(),* envía la petición para la creación del servicio enviando cómo parámetros la url del cluster de OSCAR y las propiedades necesarias para la creación del servicio.	Promise<Service>
runService	Esta función crea una instancia del cliente de oscar-js y llama al método *runService()* que invocará de forma síncrona el servicio, se envían cómo parámetros la url del clúster de OSCAR, el nombre del servicio y el archivo codificado para hacer la inferencia	Promise<any>
getAccessToken	Retorna el token de acceso que se ha guardado para el usuario actual en la aplicación, se hace gracias a la dependencia de "angular-oauth2-oidc"	string

Tabla 2. Lista de nuevos métodos en moduleService.ts

A continuación, se puede observar en la Figura 34 un ejemplo de la implementación y uso del cliente de oscar-js, para caso es la función "createService()" la cual crea un servicio en el clúster de OSCAR.

```
/**
 * Create new service in OSCAR
 * @param oscar_endpoint oscare endpoint
 * @param service service object to create
 * @returns service created
 */
createService(oscar_endpoint: string, service: Service) {
    const oidc_token = this.getAccessToken();
    const client: Client = new Client({
        clusterId: '1',
        oscar_endpoint,
        oidc_token,
    });

    const request: any = {
        ...service,
    };
    return client.createService(request);
}
```

Figura 34. Función para crear un servicio desde el proyecto de AI4EOSC Dashboard

3.5 Módulo de ejecución del servicio – "Try Me"

Para este módulo se ha realizado un desarrollo de componentes de Angular que permite invocar un servicio de IA previamente desplegado en OSCAR. Este módulo va a permitir probar a los usuarios del AI4EOSC Dashboard cualquier modelo pre-entrenado disponible, de forma sencilla e intuitiva. La invocación de este servicio se hará de forma síncrona [37]. En este caso el formulario es más sencillo y no hay necesidad de que el usuario rellene tantos campos, ya que solo debe seleccionar un fichero (imagen, vídeo, etc) determinado por el modelo de IA que se esté utilizando, que se envía para ejecutar la inferencia de este.

Una vez ejecutada la opción de lanzar, se recibe una respuesta del servicio con la inferencia de la IA que viene encriptada en base64 con el contenido de la respuesta. Un parte importante es que la respuesta pasa por oscar-js e identifica el *Mime* type *(Multipurpose Internet Mail Extensions or MIME type),* identificándolo puedo conocer la naturaleza y el formato del documento para mostrarlo en la web de acuerdo con su formato (png, jpg, mpeg, wav, zip, video, text).

A continuación, se describe el desarrollo e implementación de esta nueva funcionalidad en el portal web del AI4EOSC *Dashboard*. Esta funcionalidad se centra en la capacidad de realizar una invocación síncrona a un servicio de OSCAR y mostrar su resultado en la web. Los resultados de los modelos pueden variar dependiendo de qué modelo se esté empleando, y se ha logrado adaptar la respuesta para mostrarla de manera adecuada, ya sea como imágenes, textos JSON, videos, ficheros u otros formatos.

El componente consiste en un formulario con múltiples campos de entrada de texto, así como un campo especial para subir archivos. Con este archivo, el usuario puede realizar experimentos de inferencia para el modelo correspondiente, como, por ejemplo, la clasificación de plantas para intentar determinar la especie a la que pertenece la imagen. Tal y como se ha realizado en la sección anterior, en las siguientes subsecciones se presentan los bocetos del diseño realizado, la creación del componente y los detalles de integración con oscar-js.

3.4.1 Bocetos de diseño

Para invocar un servicio de manera síncrona con OSCAR, se ha diseñado un formulario simple que recopila los datos necesarios para la invocación. En primer lugar, se encuentra un campo de texto obligatorio donde se debe agregar la URL del clúster de OSCAR al que se desea acceder para probar el servicio.

Siguiendo con el formulario, se encuentra un menú desplegable que se completa automáticamente con los nombres de los servicios existentes en el clúster que coinciden con la imagen Docker del modelo. En caso de no haber servicios creados, se puede crear uno utilizando el formulario "Deploy via OSCAR". Además, hay un campo que muestra el nombre de la imagen Docker asociada a ese modelo de IA.

Por último, se incluyen campos para ingresar los datos sobre los cuales se realizará la inferencia. La inferencia se puede realizar a través de un campo de texto, donde se puede

ingresar una entrada en texto plano, aunque no es lo más común. Otra opción interesante es la carga de archivos multimedia, como archivos .mp3 para sonidos y .mp4 para videos.

La Figura 35 muestra el boceto creado para el componente de "Try me", manteniendo la sintonía con la paleta de colores y estilos del AI4EOSC *Dashboard*.

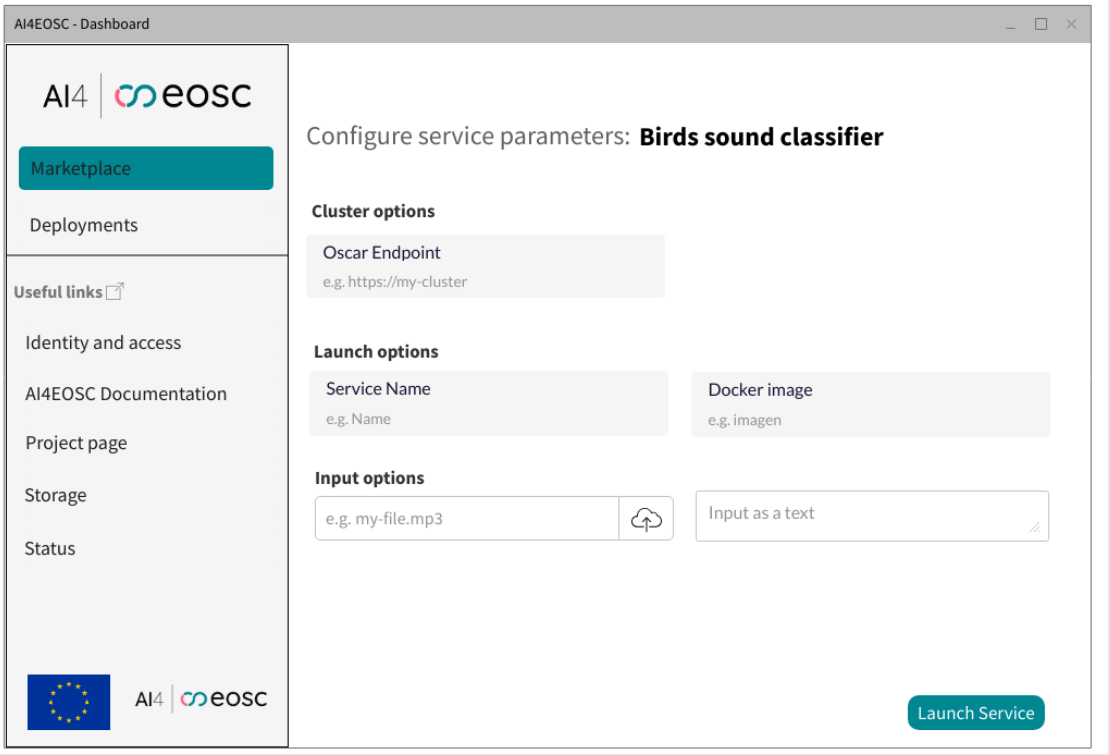

Figura 35. Boceto invocación de servicio de OSCAR

Dando clic en el botón para lanzar (invocar) un modelo, comienza la fase de inferencia. Este proceso suele tardar entre 30 y 60 segundos. Mientras este tiempo pasa el usuario estará viendo un elemento de *spinner* en el centro de la pantalla, el cual le hará entender al usuario que la ejecución está en progreso, además de que se bloquea el acceso a los campos del formulario. En la Figura 36 se refleja el momento en el que se ha invocado un servicio y se está a la espera de respuesta, por lo tanto, se muestra el *spinner*.

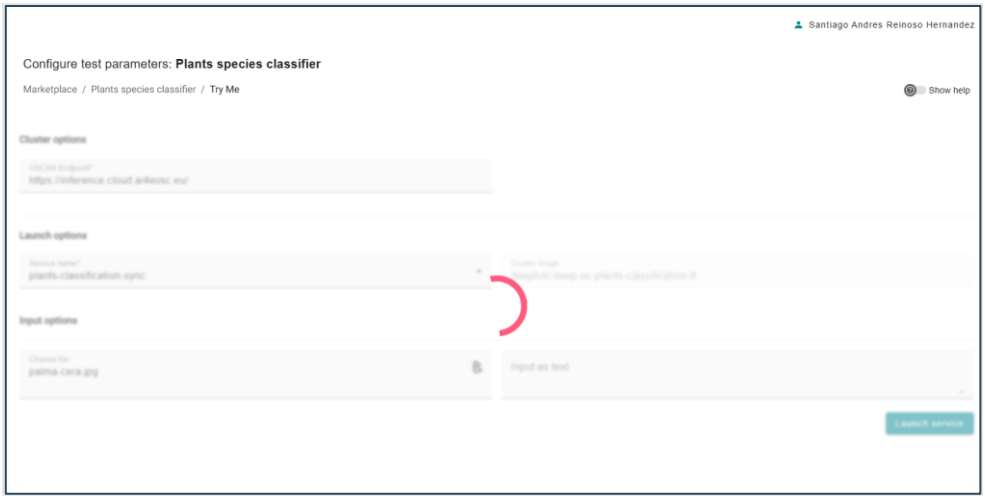

Figura 36. Spinner mientras se espera la respuesta

Después de que la ejecución concluya con éxito, se producirá una modificación en los elementos que se visualizan en el componente, revelando los resultados obtenidos y ocultando el formulario. La naturaleza de estos resultados variará según el tipo de *MIME*, que ha sido previamente calculado por 'oscar-js', tal como se detalla en la sección 3.3.6 de este documento. En la Figura 37 se ejemplifica la representación gráfica de un resultado del tipo JSON para que los resultados se muestren en una tabla.

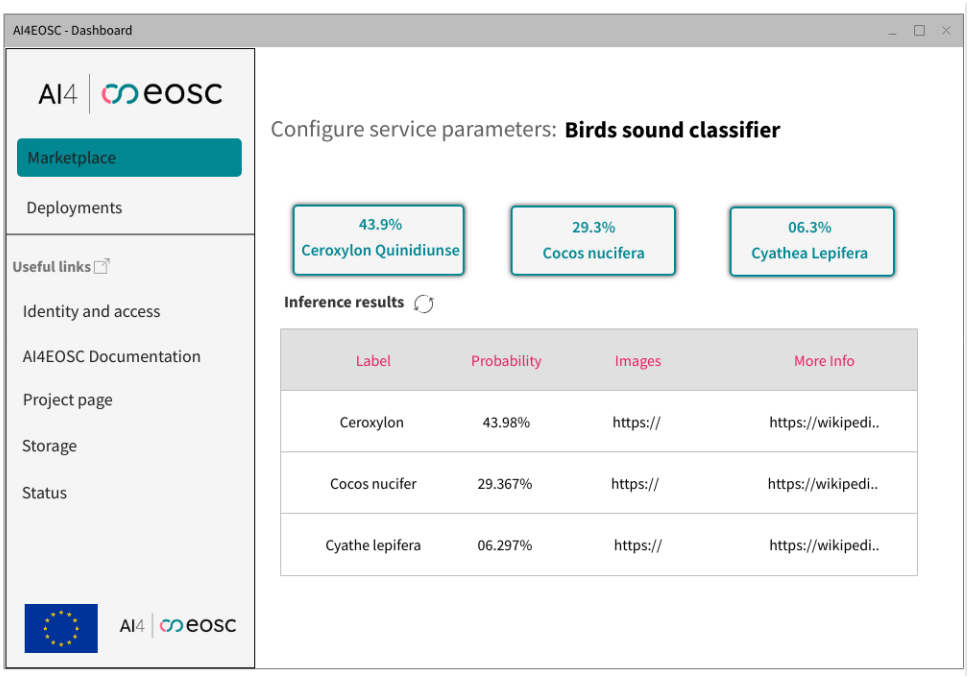

Figura 37. Boceto de resultados en formato de tabla

Por su parte en la Figura 38, se representa un ejemplo de salida si el contendido que devuelve el modelo es un fichero comprimido. En este caso el usuario podrá descargarse el fichero resultante.

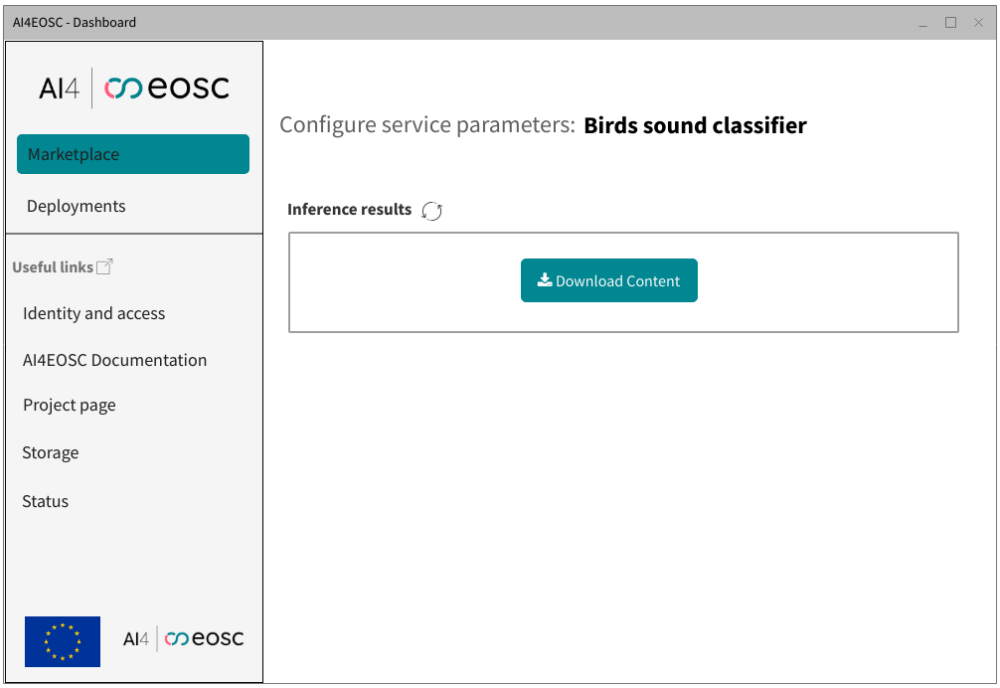

Figura 38. Boceto de respuesta en formato de fichero

La Figura 39 a continuación, muestra el boceto en el caso que el contendido sea de tipo multimedia como una imagen, un sonido o un video. El contenido ocuparía la mayor parte del componente.

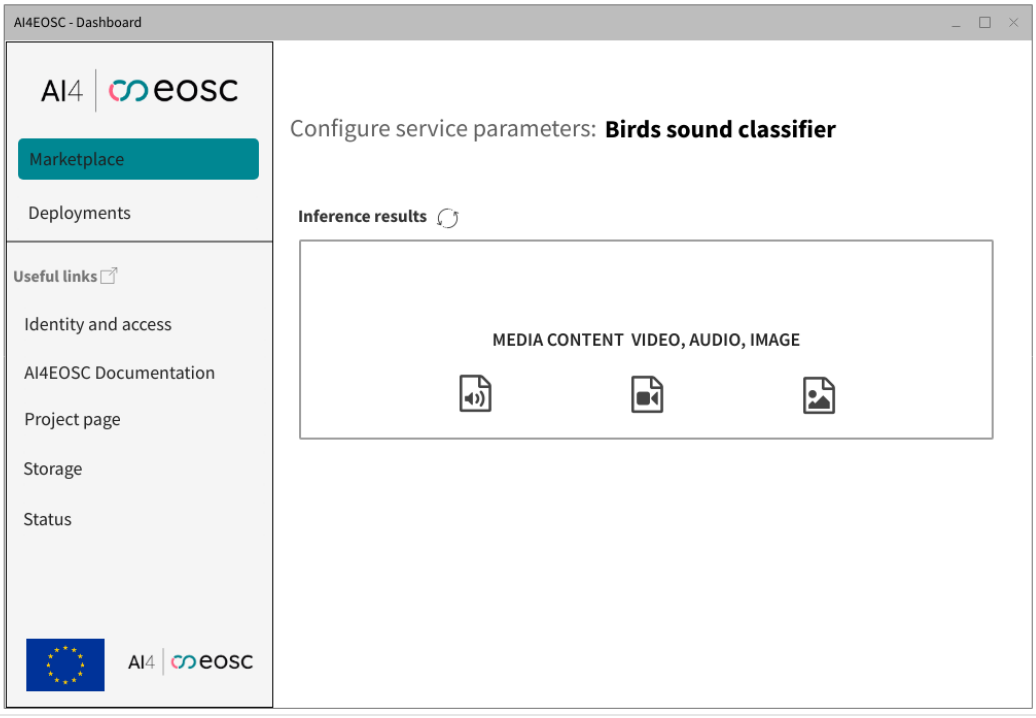

Figura 39. Boceto de resultado en formato multimedia

3.4.1 Creación del componente

Este es un componente nuevo de Angular que tiene el objetivo de recoger los datos para que un servicio pueda ser invocado en un clúster de OSCAR. Por medio del paquete de oscar-js se enviará la solicitud OSCAR, se espera la respuesta y se muestra en al usuario en el mismo portal web.

De la misma forma que se ha creado el anterior componente, pero con el nombre adecuado, creamos este nuevo componente. Para ello se ejecuta el comando a continuación:

```
ng generate component modules/marketplace/components module-try
```

Una vez que se haya creado el componente, se encontrarán cuatro archivos (HTML, TS, SCSS y SPEC.TS) dentro del directorio "module-try"', ubicado en el módulo del Marketplace. Tanto este componente como el anterior utilizan la interface "OnInit", que en Angular se emplea para realizar una acción al iniciar el componente. En este caso, se utiliza para realizar una petición a la API de ai4-papi con el fin de obtener información sobre la imagen Docker asociada al modelo de IA. Esta solicitud se lleva a cabo mediante los métodos *getModule()* y *getModuleConfiguration()*, haciendo uso del servicio "ModuleService". La Figura 40 muestra la implementación de "OnInit" al inicio del componente.

```
ngOnInit(): void {
    this.route.parent?.params.subscribe((params) => {
        this.modulesService.getModule(params['id']).subscribe((module) => {
            this.deploymentTitle = module.title;
        });

        this.modulesService
            .getModuleConfiguration(params['id'])
            .subscribe((moduleConf: ModuleConfiguration) => {
                const moduleGeneral = moduleConf.general;
                this.dockerImageName = moduleGeneral.docker_image.value;
                this.trymeFormGroup
                    .get('dockerImageInput')
                    ?.setValue(this.dockerImageName);
            });
    });
}
```

Figura 40. Implementación de interface ngOnInit

Es importante resaltar la necesidad de completar el selector desplegable que aparece en el formulario. Este selector se llena mediante un filtro aplicado a la lista de servicios disponibles en el clúster. El objetivo es mostrar únicamente los nombres de los servicios cuya imagen Docker coincide con la del modelo de IA sobre el cual se desea realizar la inferencia. En caso de no encontrar ningún servicio que corresponda a esa imagen, se

mostrará una opción por defecto con el valor 'NO OPTIONS'. A continuación, la Figura 41 representa un fragmento de código que muestra cómo se rellena a lista desplegable con los valores del nombre de los servicios.

```
/**
 * Populates the array which has the services that load in the template dropdown.
 */
async loadServices() {
    this.modulesService
        .getServices(this.oscar_endpoint)
        .then((services) => {
            this.enableInputFields();
            this.serviceList = services;
            this.servicesNames = this.serviceList
                .filter((service) => service.image == this.dockerImageName)
                .map((service) => service.name);

            if (this.servicesNames.length <= 0) {
                this.servicesNames.push('NO OPTIONS');
            }
            this.setIcon();
        })
        .catch((error) => {
            this._snackBar.open('Invalid url:  ' + error, 'close', {
                duration: 5000,
                panelClass: ['red-snackbar'],
            });
        });
}
```

Figura 41. Llenar opciones del campo desplegable del formulario "ServiceNames"

Un aspecto crucial a destacar en este componente es la variación de los elementos antes y después de la inferencia. Antes del proceso, el componente exhibe un formulario que requiere ser completado para iniciar la inferencia. Después, su contenido cambia dinámicamente en función del tipo de respuesta obtenida. Se implementa un método específico para manejar cada tipo de contenido, y posteriormente se muestra de manera dinámica en el *template*. La Figura 42 tiene un fragmento de código que muestra como desde el *template* con la directiva de Angular "ngIf" se puede mostrar el contendido si este cumple cierta condición. Así, cuando la condición es verdadera se muestran unos elementos y cuando es falsa, no se muestran.

```
<div class="prediction-container" *ngIf="showPrediction">
    <div *ngIf="showTable; then resultTable; else otherResult"></div>
    <ng-template #resultTable>
        <div class="result-container">···
        </div>

        <mat-card class="prediction-card">
            <div···
            </div>
            <mat-card-content>
                <div class="table-container">···
                </div>
            </mat-card-content>
        </mat-card>
    </ng-template>

    <ng-template #otherResult>
        <div *ngIf="imageUrl !== ''" style="display: flex; align-self: center">
            <mat-card class="prediction-card">
                <div···
                </div>
                <img mat-card-image [src]="imageUrl" alt="Result Image" />
                <mat-card-content> </mat-card-content>
                <!--mat-card-actions> </mat-card-actions-->
            </mat-card>
        </div>
        <div *ngIf="videoUrl !== ''" style="display: flex; align-self: center">
            <mat-card class="prediction-card">
                <div···
                </div>
                <video···
                </video>
                <mat-card-content> </mat-card-content>
                <!--mat-card-actions> </mat-card-actions-->
            </mat-card>
        </div>
        <div *ngIf="audioUrl !== ''" style="display: flex; align-self: center">
            <mat-card class="prediction-card">
                <div···
                </div>
                <audio [src]="audioUrl" controls>
                    Tu navegador no soporta el elemento de vídeo.
                </audio>
                <mat-card-content> </mat-card-content>
```

Figura 42. Fichero HTML que muestra el contenido dinámicamente

3.4.3 Integración con oscar-js

Esta sección pretende exponer la integración de oscar-js con este componente, con el fin de lograr la ejecución de un servicio y la recepción de su respuesta.

Como se mencionó previamente, al emplear **los bytes mágicos,** también conocidos como la **firma del archivo**, se puede inferir la naturaleza del archivo y su tipo MIME. Esta tarea es llevada a cabo por el paquete oscar-js, el cual, como respuesta a la ejecución, devuelve un objeto con dos propiedades: “data”, que en este caso está codificada en base64, y el tipo MIME asociado a esta data.

A continuación, la Figura 43 ilustra cómo se recibe la respuesta del método *runService()*, creado para recibir tres parámetros: la URL del clúster de OSCAR donde se debe realizar la solicitud, el nombre del servicio y la información codificada en base64 del contenido sobre el cual el modelo de IA realiza la inferencia.

```
async runService(oscar_endpoint: string, serviceName: string, file: File) {
    try {
        const stringBase64 = await this.encodeFileToBase64(file);
        const response = await this.modulesService.runService(
            oscar_endpoint,
            serviceName,
            stringBase64
        );
        const { mime, data } = response;
        console.log('MIME TYPE', mime);

        switch (mime) {
        case 'application/zip':
            this.handleZip(data);
            break;
        case 'image/png':
        case 'image/gif':
        case 'image/jpeg':
            this.handleImage(mime, data);
            break;
        case 'audio/wav':
        case 'audio/mpeg':
            this.handleAudio(mime, data);
            break;
        case 'video/mp4':
            this.handleVideo(mime, data);
            break;
        case 'application/octet-stream':
            this.handleJson(data);
            break;
        default:
            console.error('MIME TYPE NOT FOUND', mime);
            break;
        }
    } catch (error) {
        console.error('Error ocurred executing service!!', error);
    }
}
```

Figura 43. Estrategia para manejar respuesta según tipo de contenido

Una vez identificado el tipo de archivo, y para mostrar el contenido de forma dinámica de manera comprensible y sostenible a largo plazo, se utiliza una estructura *switch-case* que maneja cada posible tipo de contenido, como, por ejemplo: “application-zip”, “image/png”, “application/octet-stream”. La Figura 43 representa como se ha implementado.

3.6 Pruebas unitarias de los componentes

En un proyecto software es buena práctica y fundamental la implementación de pruebas que garanticen la validez del comportamiento. El AI4EOSC *Dashboard* se beneficia de

las pruebas de integración y unitarias, las cuales ofrecen varias ventajas, como la detección temprana de errores y la mejora de la calidad del código. Estas pruebas se realizan utilizando el *framework* de *testing* de JavaScript conocido como Jest [18], reconocido por su facilidad de uso y robustez en la detección de fallos.

Para comenzar, se realizan pruebas en el componente "ModuleService", que también está integrado con oscar-js, por lo que se ha creado un conjunto específico de pruebas para este componente. Se inicia el proceso creando un conjunto de pruebas utilizando el método *describe()* de "Jest", el cual se utiliza para agrupar pruebas relacionadas y facilitar su organización y ejecución, lo que mejora la legibilidad y mantenibilidad del código de prueba. Posteriormente, se desarrollan varias pruebas donde se evalúan diferentes funciones y se verifica que los datos devueltos coincidan con los datos esperados. A continuación, se muestra un fragmento de código en la Figura 44 que ilustra el inicio de estas pruebas.

```
describe('Module oscar-js', () => {
    let service: ModulesService;
    let httpMock: HttpTestingController;

    beforeEach(() => {
        TestBed.configureTestingModule({
            imports: [HttpClientTestingModule],
            providers: [
                OAuthStorage,
                { provide: AppConfigService, useValue: mockedConfigService },
            ],
        });
        service = TestBed.inject(ModulesService);
        httpMock = TestBed.inject(HttpTestingController);
    });

    afterEach(() => {
        httpMock.verify();
    });


    it('should oscar client initialization', () => {
        const client = new ClientMock();
        expect(client).toBeDefined();
    });

    it('should list oscar services', () => {
        const client = new ClientMock();
        const services = client.getServices();
        expect(services).toBeInstanceOf(Array);
        expect(services).toHaveLength(2);
    });
```

Figura 44. Definición de pruebas unitarias del cliente de oscar-js

[18] Jest: https://jestjs.io/es-ES/

En el anterior fragmento de código se define el bloque de pruebas llamado 'Module oscar-js' dentro de este bloque se definen las pruebas individuales. En la función *beforeEach*() se coloca el código que se ejecutará antes de cada prueba, seguido de esto se configuran los módulos de Angular necesarios para la prueba, en este caso, se importa "HttpClientTestingModule" y se proporcionan algunos servicios como "OAuthStorage" y "AppConfigService".

Para llevar a cabo esta prueba, primero se procede a inyectar los servicios necesarios en el componente. En este caso, se requieren los servicios "ModulesService" y "HttpTestingController". Una vez que se ha definido todo lo necesario para la ejecución de cada prueba, se procede a crearla utilizando la función *it(),* que representa una prueba individual para una función específica del componente, como se muestra en la imagen anterior (Figura 44). Se crea una prueba individual con un nombre que sugiere que se está probando la inicialización del cliente oscar-js. En esta prueba, se instancia un objeto ClientMock y se verifica que esté definido utilizando la siguiente línea de código:

```
expect(client).toBeDefined().
```

Una vez creada la prueba, se ejecuta para comprobar que se están cumpliendo los criterios y que los resultados son los esperados. El comando para ejecutar esta prueba del componente es el siguiente:

```
npx jest src/app/modules/marketplace/services/modules-
service/modules.service.spec.ts
```

El resultado de las pruebas se puede ver en la terminal, tal y como se muestra en la Figura 45. La imagen muestra que se han hecho ocho casos de pruebas y que todos han pasado correctamente las pruebas.

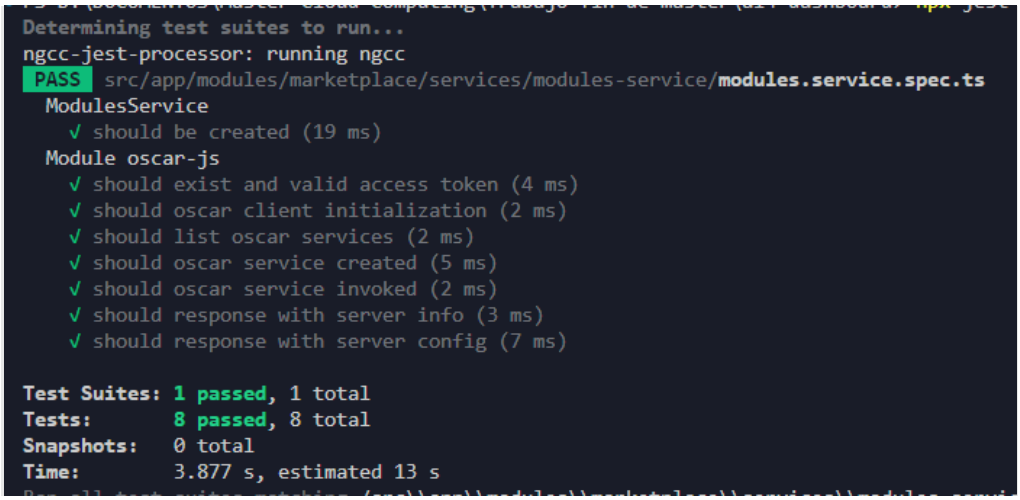

Figura 45. Resultado de pruebas unitarias de oscar-js

Después de explicar cómo se crean las pruebas, es hora de mostrar los resultados de los componentes creados en este proyecto. Se realizan dos pruebas simples para verificar que

los componentes se inicializan correctamente y que los servicios y módulos se integran adecuadamente. Esto es importante para asegurarse que todo funciona como debería.

La Figura 46 corresponde a los *tests* del componente creado en la sección 3.4 de este documento y representa el resultado de las pruebas después de ejecutar el comando:

```
npx jest src/app/modules/marketplace/components/module-oscar-deploy/module-oscar-deploy.component.spec.ts
```

Figura 46. Resultado de pruebas unitarias de componente "module-oscar-deploy"

La Figura 47 corresponde a los test del componente creado en la sección 3.5 de este documento y representa el resultado de las pruebas después de ejecutar el comando:

```
npx jest src/app/modules/marketplace/components/module-try/module-try.component.spec.ts
```

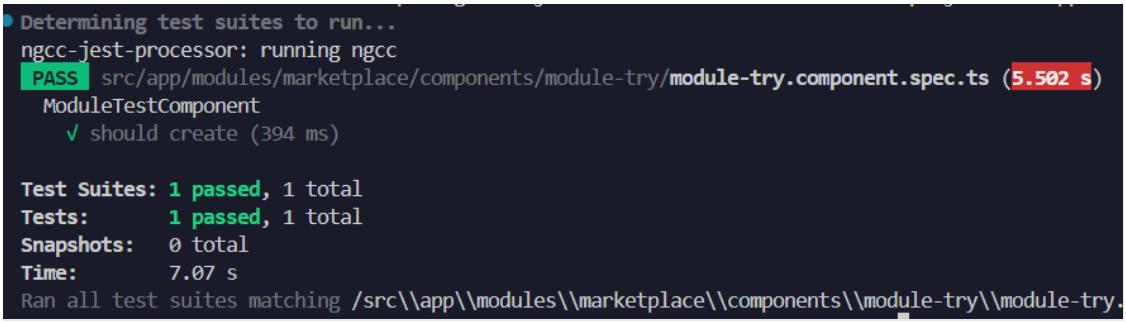

Figura 47. Resultados de pruebas unitarias de componente "module-try"

Es importante destacar que estas pruebas son específicas para los nuevos componentes desarrollados. Además, hay que mencionar que los componentes ya existentes también cuentan con sus propios casos de pruebas correspondientes. De esta forma se asegura que cada componente de la aplicación se somete a un proceso de evaluación antes de cualquier nueva publicación o *release*. Todas las pruebas deben ser validadas y pasar satisfactoriamente para cumplir con el ciclo de entrega y la integración continua que está configurada en este repositorio a través de GitHub Action [48].

4. Resultados y pruebas

4.1 Despliegue de OSCAR usando *Infraestructure Manager* (IM)

En este experimento, se busca demostrar el proceso de despliegue de un clúster utilizando una infraestructura gestionada. El objetivo principal es crear un entorno nuevo en el que se puedan implementar y probar diversos servicios. Para ello, se utilizará la plataforma de gestión de infraestructura (IM), proyecto que desarrollado por el grupo GRyCAP de la UPV. Esta herramienta permite automatizar y simplificar la administración de recursos informáticos, como máquinas virtuales y servicios en la nube, facilitando la implementación de proyectos y experimentos [29].

El portal solicita el inicio de sesión con EGI Check-in, un sistema de autenticación implementado por *European Grid Infrastructure* (EGI) [49], la federación europea de proveedores de recursos informáticos y de almacenamiento unidos por la misión de ofrecer servicios avanzados de computación y análisis de datos para la investigación y la innovación.

El acceso está disponible para todos los colaboradores de diversas instituciones en Europa, incluidos los miembros de la UPV. Una vez iniciada la sesión, se muestra una vista con varias opciones para desplegar la infraestructura. La opción que seleccionada será *Deploy an-OSCAR virtual cluster*. La Figura 48 muestra la pantalla inicial y la opción que se debe elegir.

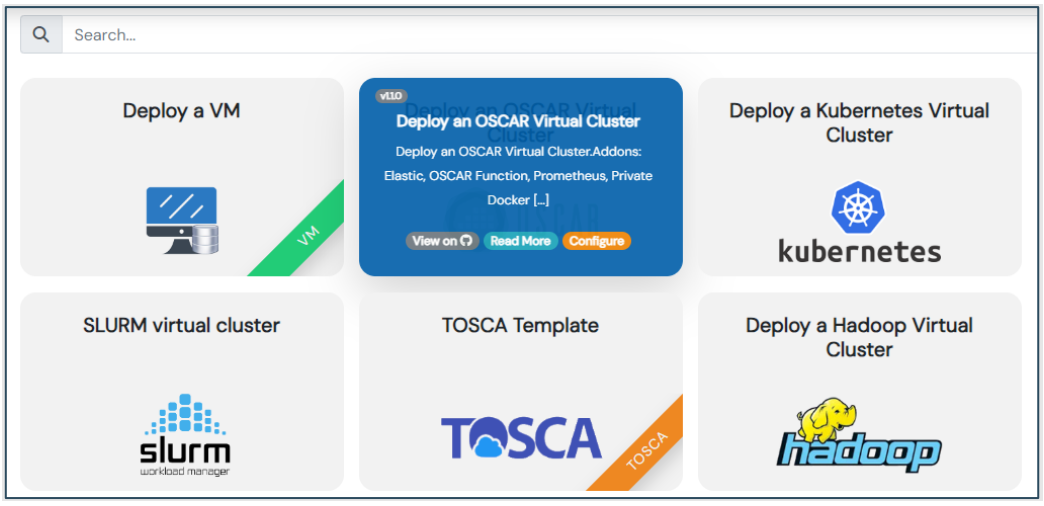

Figura 48. Desplegando un clúster virtual de OSCAR en IM

IM ofrece opciones flexibles para el despliegue de clústeres. A continuación, se detallan las posibilidades disponibles.

- **Despliegue estático:** Es un tipo de despliegue de infraestructura donde el clúster tiene un número de nodos que se mantiene constante y predefinido. Esto significa que el clúster tiene un tamaño fijo, independientemente de la carga de trabajo o la demanda.

- **Despliegue elástico:** Es un tipo de despliegue donde el número de nodos varía automáticamente según la carga de trabajo. Por ejemplo, en un clúster de Kubernetes, los nodos se escalan hacia arriba o hacia abajo según la carga de trabajo de este.

Para este experimento se usa la opción del despliegue elástico, el cual es perfecto para las cargas de trabajo variables de este proyecto. Ahora se debe dar clic en el botón de color naranja que está en la parte inferior para continuar con la configuración del clúster. En la Figura 49 se puede ver la opción escogida.

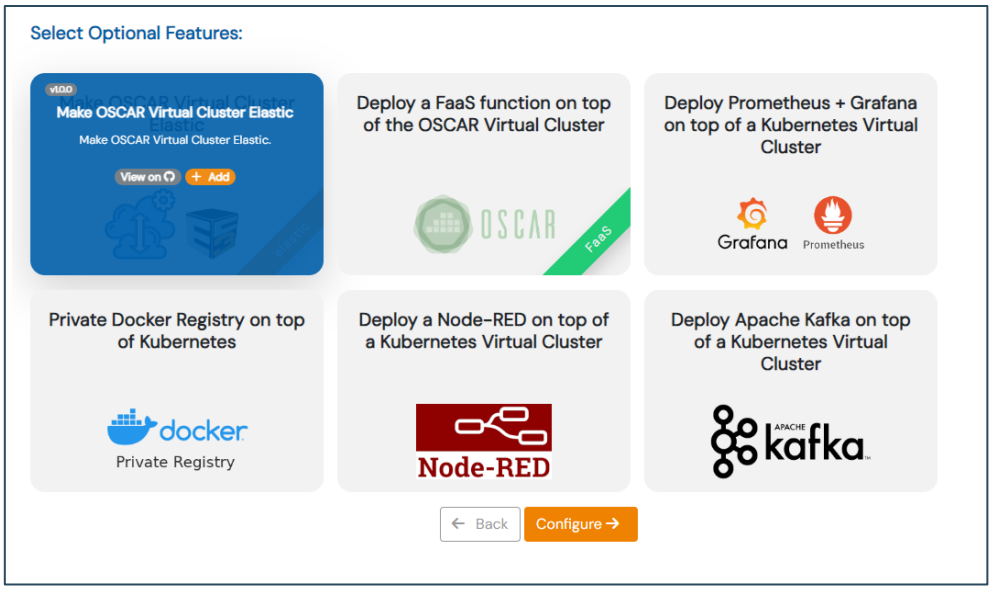

Figura 49. Configuración de clúster virtual elástico

Siguiendo con la configuración del clúster, se presenta un nuevo formulario que consta de cuatro secciones o pestañas. Avanzaremos a través de cada una de ellas para describir detalladamente el proceso de despliegue.

1. **HW Data:** En esta sección se encuentran las opciones relacionadas con el hardware que se desea configurar en el clúster. Para este caso se detallan en la Tabla 3 las opciones de cómputo escogidas para crear el clúster.

Elemento	Cantidad
Virtual Working Nodes (WNs)	2 nodes
Number of CPUs WNs	2 cpu
Amount of Memory WNs	4GB
Number of CPUs front-end node	4 cpu
Amount of Memory front-end node	4GB
HD size	50 GB

Tabla 3. Datos de Hardware para el clúster.

2. **OSCAR Data:** En esta sección se debe especificar propiedades de OSCAR, se eligen los *passwords* de acceso y configuración de propiedades extras en el clúster

como por ejemplo soporte para GPUs NVIDIA. En la Tabla 4 se detallan la especificación para este clúster.

Elemento	Valor
Access Token Kubernetes admin	<secret-token>
OSCAR *password*	<secret-password>
MinIO *password*	<secret-password>
Email used for encrypt issuer	sareiher@posgrado.upv.es
VO to support	vo.ai4eosc.eu
Add NVIDIA support	False
Install Apache Yunikorn	False

Tabla 4. Datos de plataforma OSCAR.

3. **Elastic Data:** En esta pestaña se definen las cantidades máximas y mínimas de nodos trabajadores que tendrá el clúster. En la Tabla 5 se detallan los valores configurados

Elemento	Cantidad
Max number of WNs	2
Min number of WNs	0

Tabla 5. Datos de crecimiento elástico del clúster.

4. **Cloud provider selection:** Se puede elegir el proveedor cloud para el despliegue de la infraestructura. La plataforma IM admite muchos tipos de proveedores entre los que destacan AWS, Azure, Google Cloud, EGI Cloud, entre otros. En la siguiente Figura 50, se puede ver la creación de las nuevas credenciales para el despliegue usando EGI Cloud. Se debe especificar una organización virtual (conocida en inglés como *Virtual Organization* (VO)) a la que pertenece el usuario y de la que va a requerir el uso de los recursos, y un proveedor de recursos (*site provider)*.

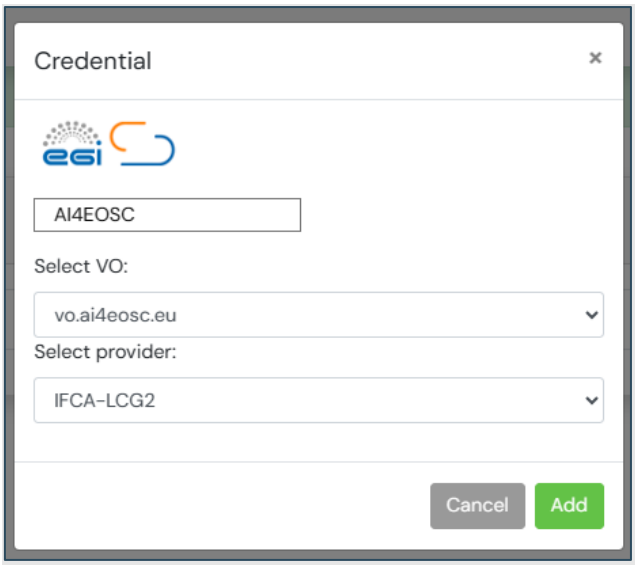

Figura 50. Credenciales cloud federado

Una vez se han colocado las credenciales del proveedor en la nube, se debe elegir la imagen que se usarán las máquinas virtuales que actuarán como los nodos del clúster. La Figura 51 representa la vista del formulario para terminar la selección.

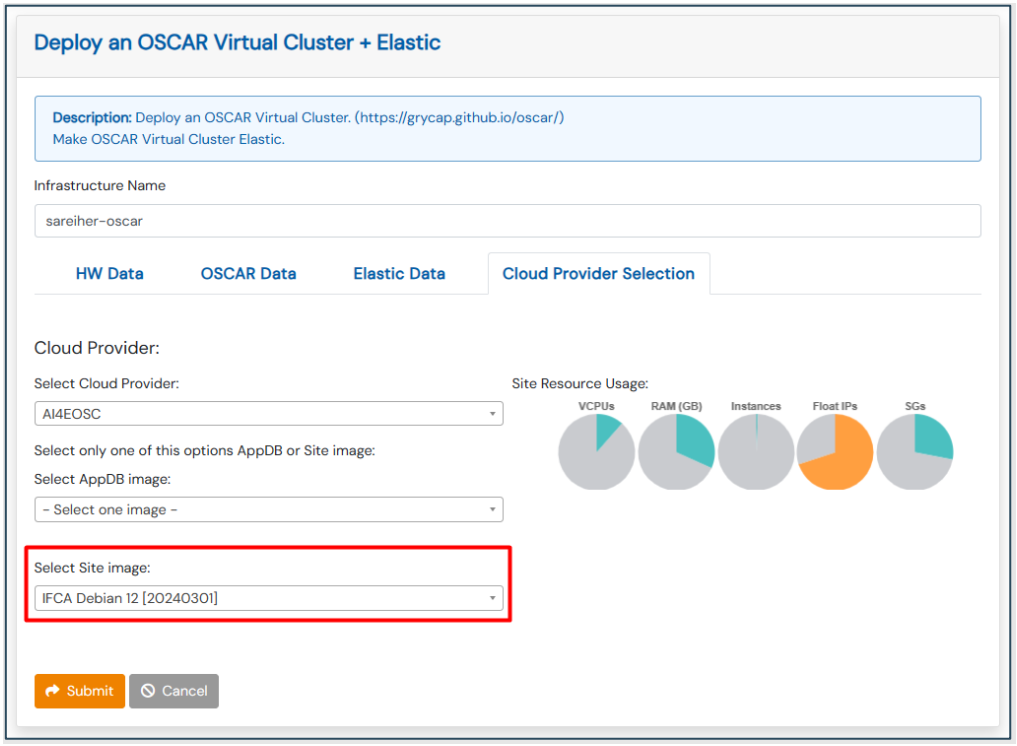

Figura 51. Selección de proveedor cloud e imagen del sistema

Una vez completados todos los pasos para el despliegue, se genera una nueva infraestructura, que se puede consultar en la tabla de infraestructuras creadas por el usuario. Inicialmente, las instancias tienen un estado de "Pending" mientras se inician, luego pasan a "Running", como se ve en la Figura 52, mientras se ejecuta el proceso de configuración. Durante esta etapa, se instalan y configuran los componentes necesarios para crear un clúster de Kubernetes e instalar OSCAR como plataforma *serverless*.

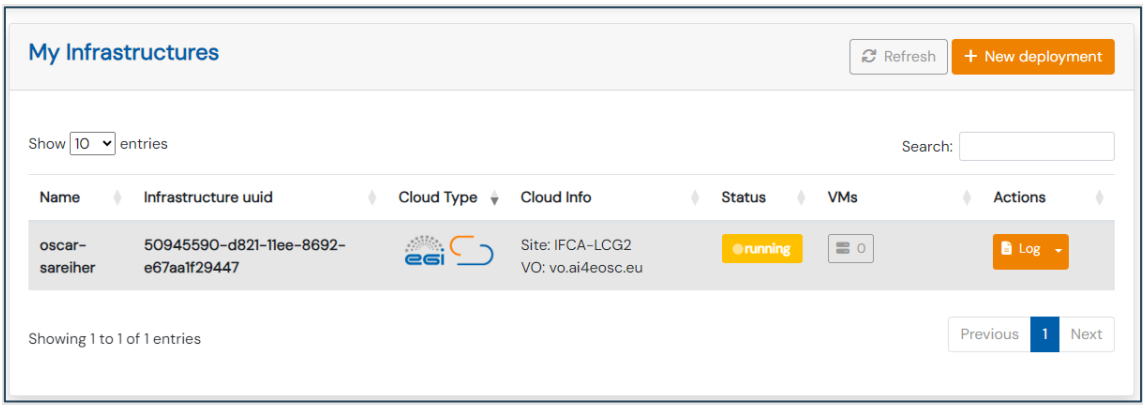

Figura 52. Creación inicial de infraestructura

Después de unos minutos, la infraestructura queda desplegada correctamente con todos los componentes necesarios para un clúster de OSCAR instalados, como se ilustra en la Figura 53, y el estado del clúster cambia a "Configured", mostrándose en color verde. Además, en la sección de acciones, aparece un nuevo botón etiquetado como "Outputs".

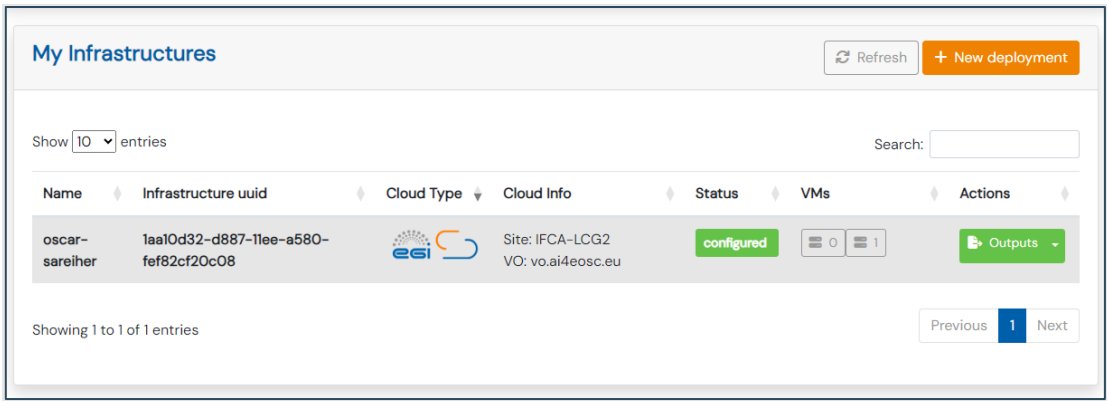

Figura 53. Infraestructura configurada correctamente en IM

Al hacer clic en el botón "Outputs", se abrirá un modal que mostrará información importante, como las URL para acceder a la interfaz de usuario de OSCAR y MinIO. La Figura 54 a continuación, muestra los datos que aparecen en el modal.

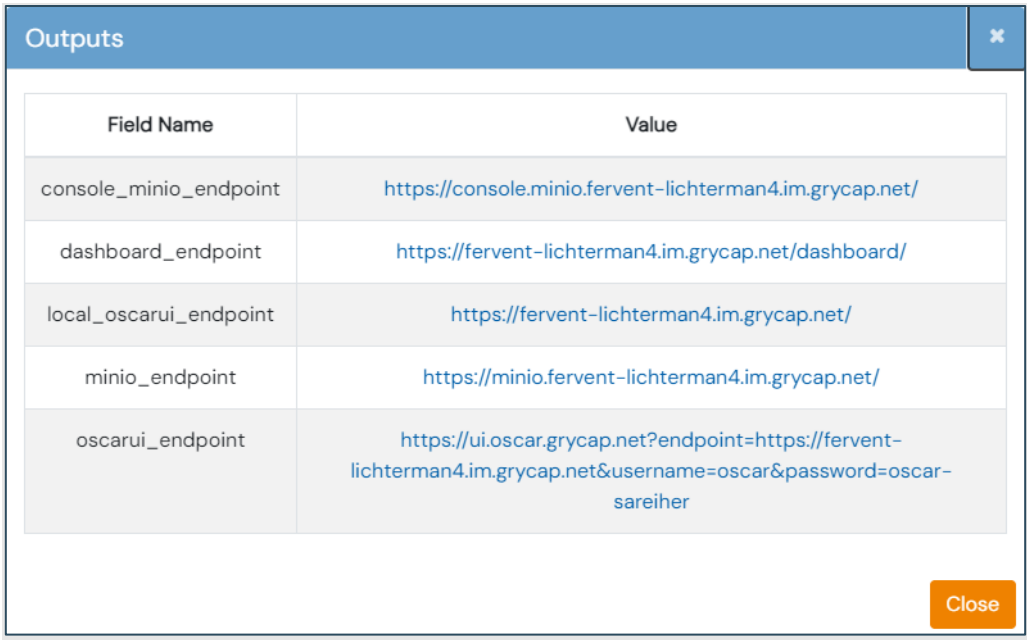

Figura 54. Rutas de acceso a los recursos del clúster.

4.2 Creación de un servicio de OSCAR desde el portal

En la siguiente sección del capítulo, se explorará cómo crear un nuevo servicio en OSCAR utilizando las funciones más recientes del portal AI4EOSC *Dashboard*. Es esencial tener acceso a EGI Check-in para iniciar sesión en el portal, y también se debe verificar que el usuario esté asociado con la VO correspondiente a este proyecto. Las funcionalidades del portal web están disponibles para usuarios en las organizaciones

vo.ai4eosc.eu o vo.imagine-ai.eu. Esto debe tenerse en cuenta para realizar despliegues y procesos de inferencia.

El AI4EOSC Dashboard está disponible para uso experimental en la dirección web https://dashboard-stage.cloud.ai4eosc.eu/marketplace y para uso en producción se encuentra en la dirección https://dashboard.cloud.ai4eosc.eu/marketplace. Al acceder, se encontrará con una vista que muestra todos los modelos disponibles en el *dashboard*. En la esquina superior derecha se encuentra el botón de inicio de sesión. La Figura 55 señala su ubicación específica.

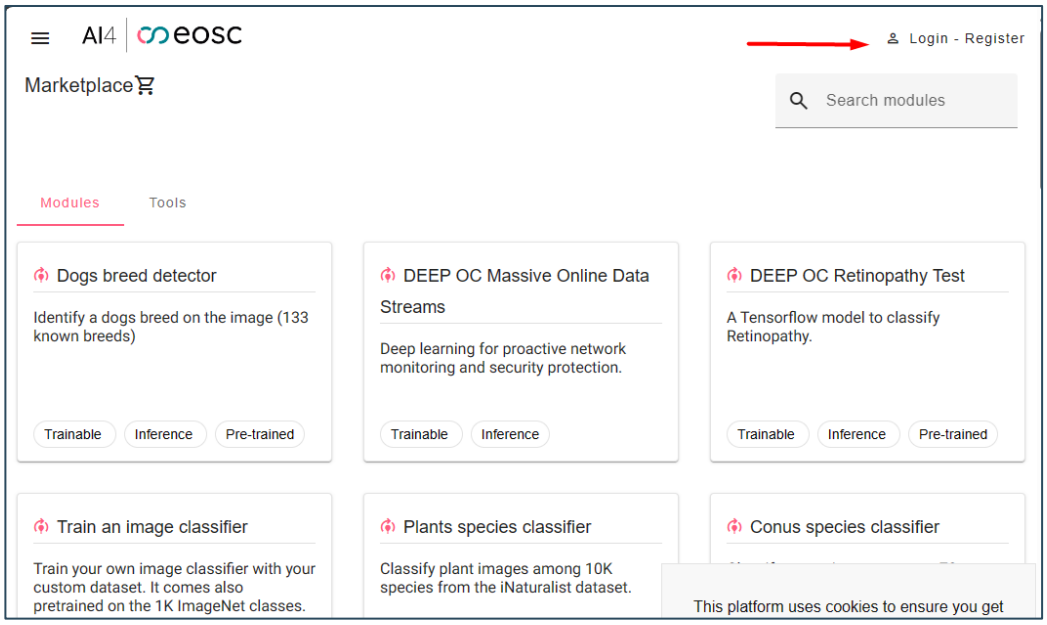

Figura 55. Vista de martketplace indicando inicio de sesión

Al hacer clic en el enlace "Login - Register", el usuario es redirigido a la página web de EGI check-in. En este sitio, el usuario puede ingresar sus credenciales. EGI check-in es compatible con el inicio de sesión de varias instituciones y aplicaciones. La Figura 56 proporciona un ejemplo visual de lo que se puede ver en esa página.

Figura 56. Portal de inicio con EGI Check-In

Para este proyecto se hace inicio de sesión con las credenciales de la UPV asociadas a las VOs mencionadas anteriormente, apareciendo el nombre de la persona que ha iniciado sesión. La Figura 57 muestra un ejemplo de esto.

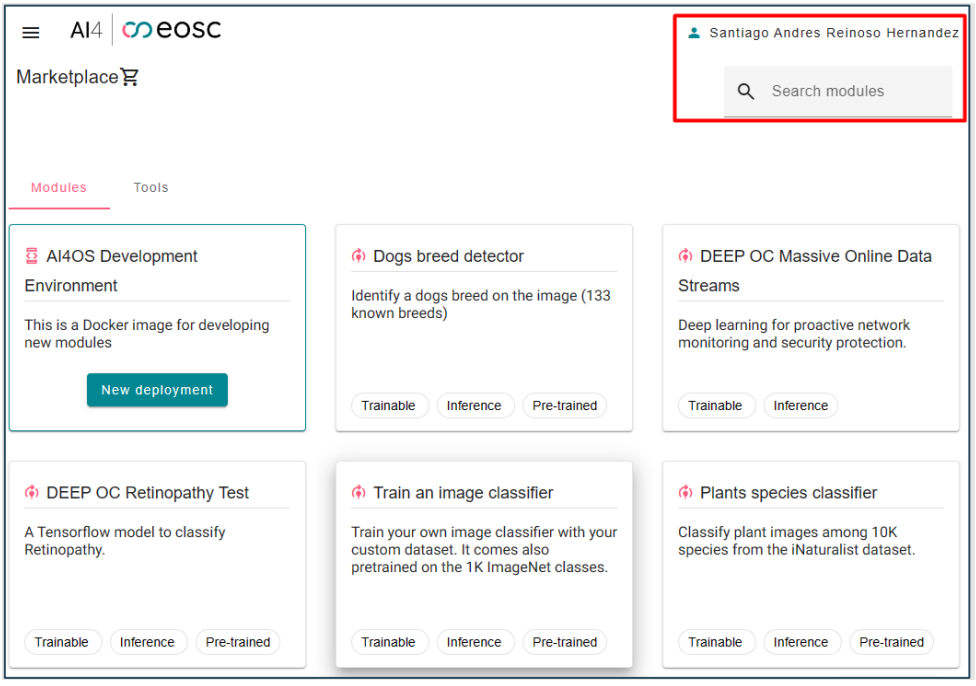

Figura 57. Marketplace con inicio de sesión correcto

Ahora, el usuario puede explorar los modelos de IA disponibles en el *marketplace* del AI4EOSC Dashboard. Los modelos que están listos para realizar inferencias son aquellos etiquetados como "Inference". Al hacer clic en la tarjeta de un modelo, se accede a

información más detallada sobre el mismo, lo que corresponde al componente *module-detail*. Después de iniciar sesión, como se mencionó anteriormente, se habilitan unos botones de interacción con el modelo como ilustra la Figura 58 con los botones habilitados.

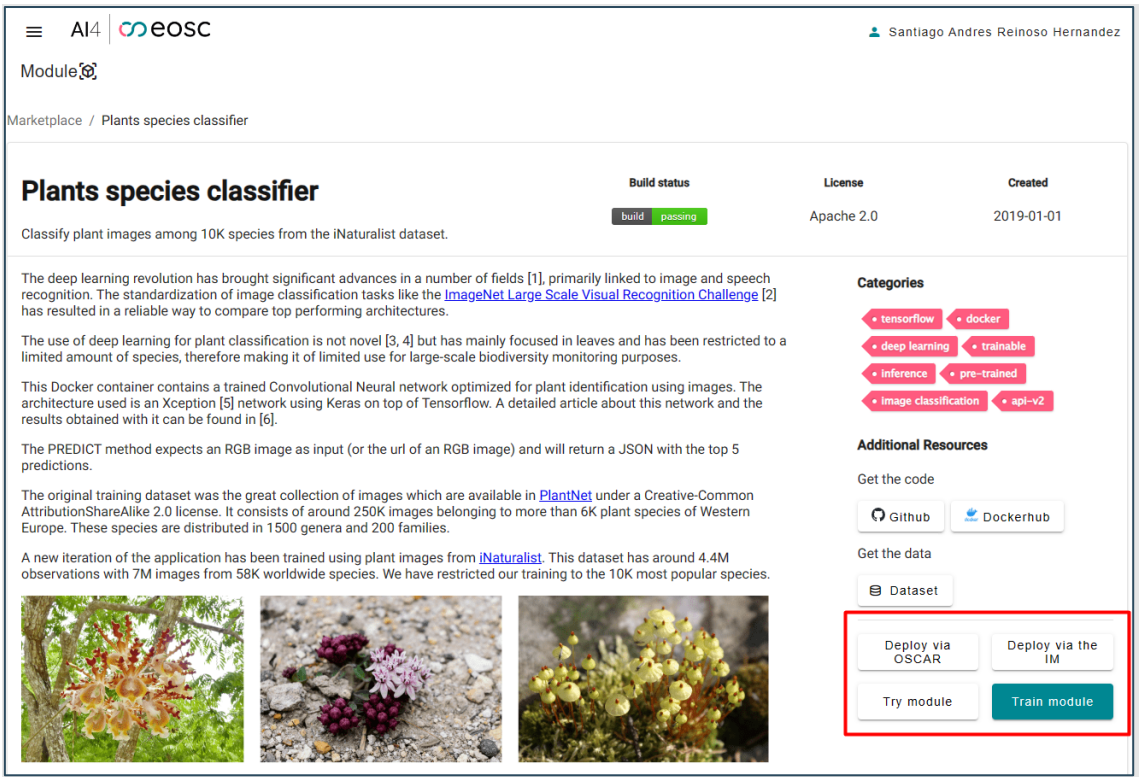

Figura 58. Detalle del modelo con opciones de interacción

El modelo elegido para ejemplificar el proceso de despliegue es el "Plants Species Classifier", un modelo pre-entrenado con miles de imágenes de diferentes especies de plantas. La función de este modelo consiste recibir una imagen de un tipo de planta y su tarea principal es identificar el tipo de especie al que pertenece la planta. Como resultado, proporciona un documento en formato JSON que contiene las probabilidades y los nombres científicos de las plantas inferidas.

Esta sección se enfoca en el botón "Deploy via OSCAR", que proporciona acceso al componente diseñado con este propósito, detallado en la sección 3.4 del documento. Al hacer clic en este botón, el usuario es redirigido al formulario que facilita la creación de un nuevo servicio de OSCAR. En la Figura 59 se observa en la parte superior derecha un elemento tipo *Switch*. Este elemento activa un texto breve debajo de cada campo del formulario, proporcionando una ayuda visual explicando cuales datos se deben ingresarse en cada campo.

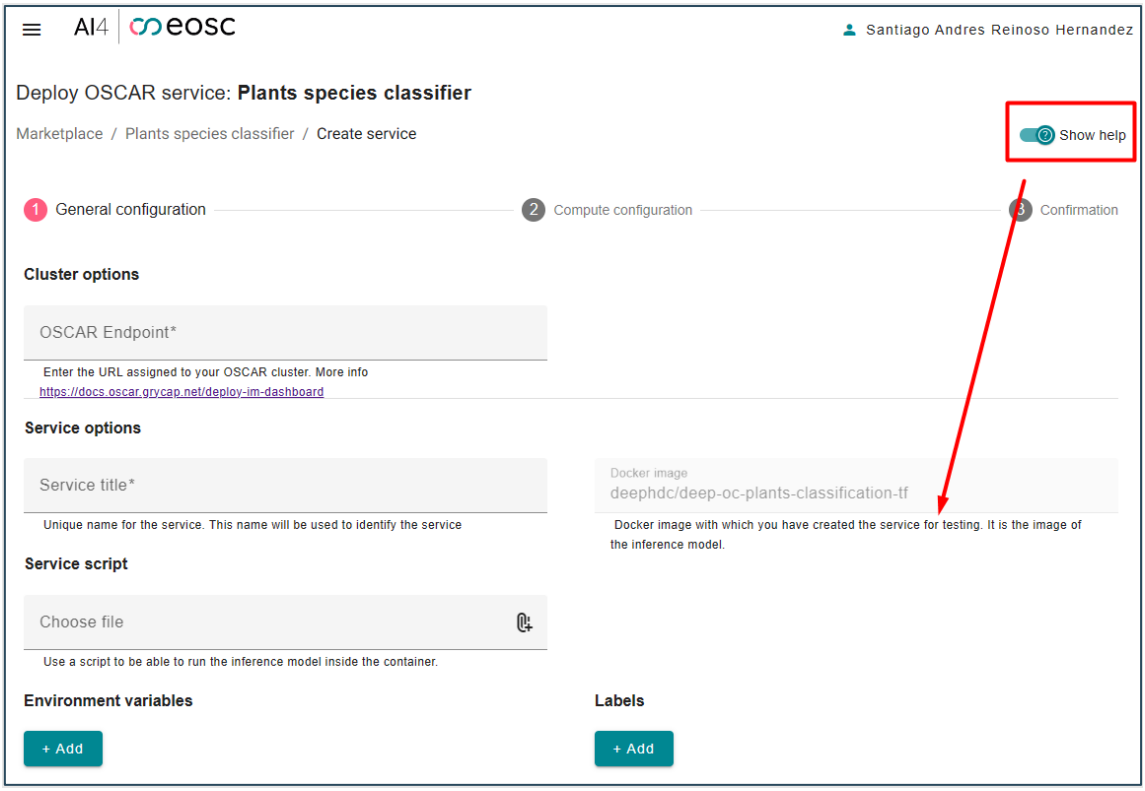

Figura 59. Formulario de despliegue con ayuda visual

4.3.1 Formulario: paso 1

Los datos en esta sección del formulario corresponden a información general del servicio de OSCAR. El primer elemento del formulario es un campo de entrada de texto donde se debe ingresar la URL del clúster de OSCAR sobre el cual se desea realizar la operación. En la Figura 60, se muestra que este campo se completa con el valor de la URL del clúster de OSCAR generado previamente durante el despliegue de la infraestructura, con más detalles en la sección 4.2 de este documento.

Figura 60. Campo de OSCAR Endpoint URL

Siguiendo con el formulario se encuentran los campos de "Service title" y "Docker image". En estos campos de debe introducir un nombre identificativo (elección del usuario) para este servicio y la imagen de Docker que rellana forma automática. La Figura 61 detalla un ejemplo de cómo se debe rellenar.

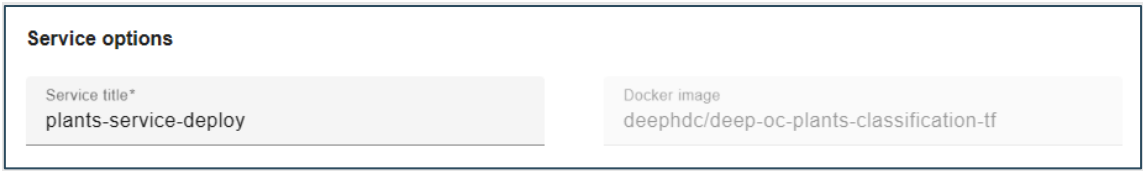

Figura 61. Campos de formulario nombre servicio e imagen Docker

El siguiente campo del formulario es muy importante para ejecutar el modelo. Se refiere al *script* dentro de la imagen Docker del modelo. En la Figura 62, se muestra el *script* usado para invocar los modelos. Este *script*, escrito en "bash", utiliza las variables de entorno INPUT_FILE_PATH para almacenar el contenido del archivo usado en la inferencia y TMP_OUTPUT_DIR para indicar la ubicación del resultado final. Finalmente, se invoca el comando "deepass-predict" para procesar la inferencia, pasándole como argumentos el archivo de entrada y la ruta de salida.

```
#!/bin/bash

IMAGE_NAME=`basename $INPUT_FILE_PATH`
OUTPUT_FILE="$TMP_OUTPUT_DIR/$IMAGE_NAME"

# Adding the extension to the input file
mv $INPUT_FILE_PATH "$INPUT_FILE_PATH.jpg"

echo "SCRIPT: Invoked deepaas-predict command. File available in $INPUT_FILE_PATH."
deepaas-predict -i "$INPUT_FILE_PATH.jpg" -o $OUTPUT_FILE
```

Figura 62. Ejemplo de script

Al hacer clic en el campo del script (identificado con un ícono), se abrirá una ventana que permitirá al usuario cargar un archivo con extensión ".sh", donde estará el contenido del script previamente mencionado como se puede ver en la Figura 63. Una vez que se carga el archivo, su contenido será visible en la web, lo que le permitirá al usuario verificar que el archivo subido es el correcto y realizar cualquier modificación necesaria.

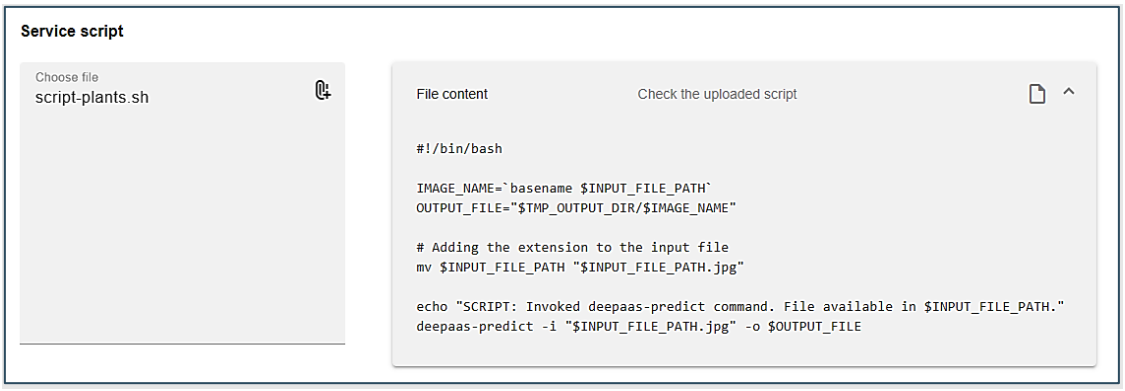

Figura 63. Campo de formulario de script

Al final del formulario, se encuentran unos campos dinámicos que pueden ser agregados o eliminados según sea necesario. Estos campos, denominados "Environment variables" y "Labels", pueden ser útiles sí la imagen de Docker del modelo requiere esta información

(aunque no es algo común). La Figura 64 muestra valores de ejemplo que ilustran cómo funcionan estos campos.

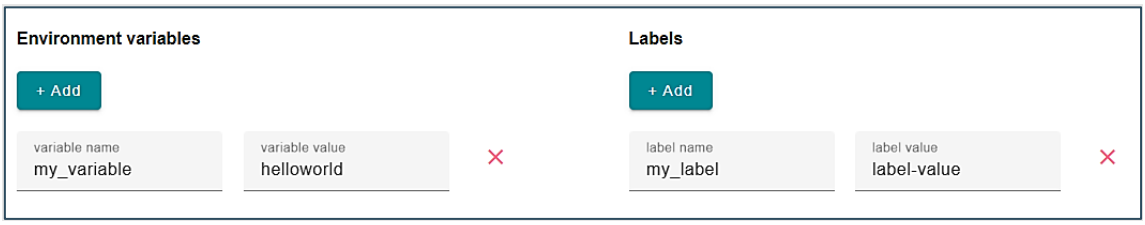

Figura 64. Campos formulario dinámicos

4.3.2 Formulario: paso 2

La segunda parte de este formulario se centra en las configuraciones de cómputo y hardware para el servicio de OSCAR. Al igual que en la sección anterior, se revisarán los campos de este formulario para asegurarse de que estén correctamente completados. El componente de ayuda también está disponible en esta parte como lo muestra la Figura 65 y proporciona detalles sobre la información esperada en cada campo.

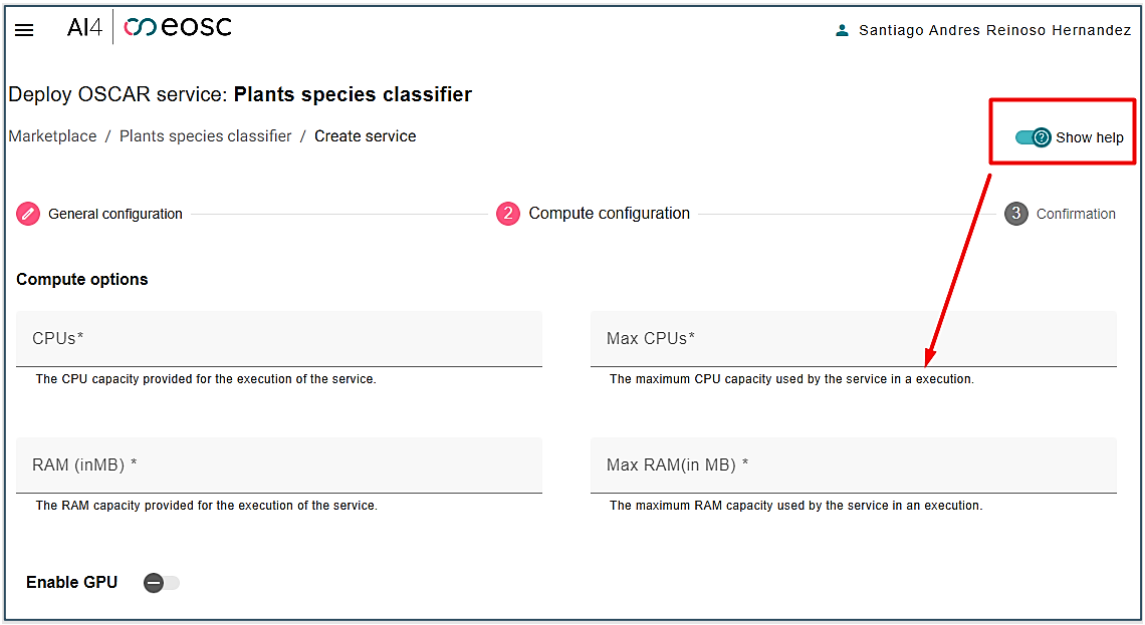

Figura 65. Ejemplo de formulario paso dos

Los siguientes campos para completar son los relacionados con la asignación de CPU para el servicio y el máximo de CPU que puede consumir. En este caso, se trata de dos campos de entrada numérica que pueden contener un valor decimal (por ejemplo, 1.5 CPUs). La Figura 66 muestra los valores utilizados para la creación de este servicio de OSCAR.

Figura 66. Campos de formulario CPU y Max CPU

Los campos siguientes también son de entrada numérica y se refieren a los valores de RAM (Memoria de Acceso Aleatorio) que tendrá este servicio de OSCAR. La Figura 67 muestra como ejemplo los valores que se deben ingresar. Es importante destacar que estos valores se ingresan en megabytes.

Figura 67. Campos de formulario RAM y Max RAM

Finalmente, en esta sección, se encuentra un campo de tipo booleano que determina si el servicio que se está creando necesitará recursos para su ejecución en GPUs. La Figura 68 muestra que para este modelo no será necesario. En caso de que el modelo lo requiera, solamente se debe marcar la opción para activarlo.

Figura 68. Campos de formulario habilitar GPU

4.3.3 Formulario: paso 3

En el tercer paso de este formulario guiado por pasos, no es necesario ingresar más datos, pues en este paso se muestra un resumen de la información introducida en los pasos anteriores. Para crear un servicio en OSCAR, solo se requiere agregar unos pocos datos, lo que lo hace sencillo para cualquier usuario. Como se observa en la Figura 69, el primer dato que aparece es la URL del clúster de OSCAR donde se desea crear el servicio. Es crucial que esta URL sea la correcta, de lo contrario, podrían producirse errores y el servicio no se crearía.

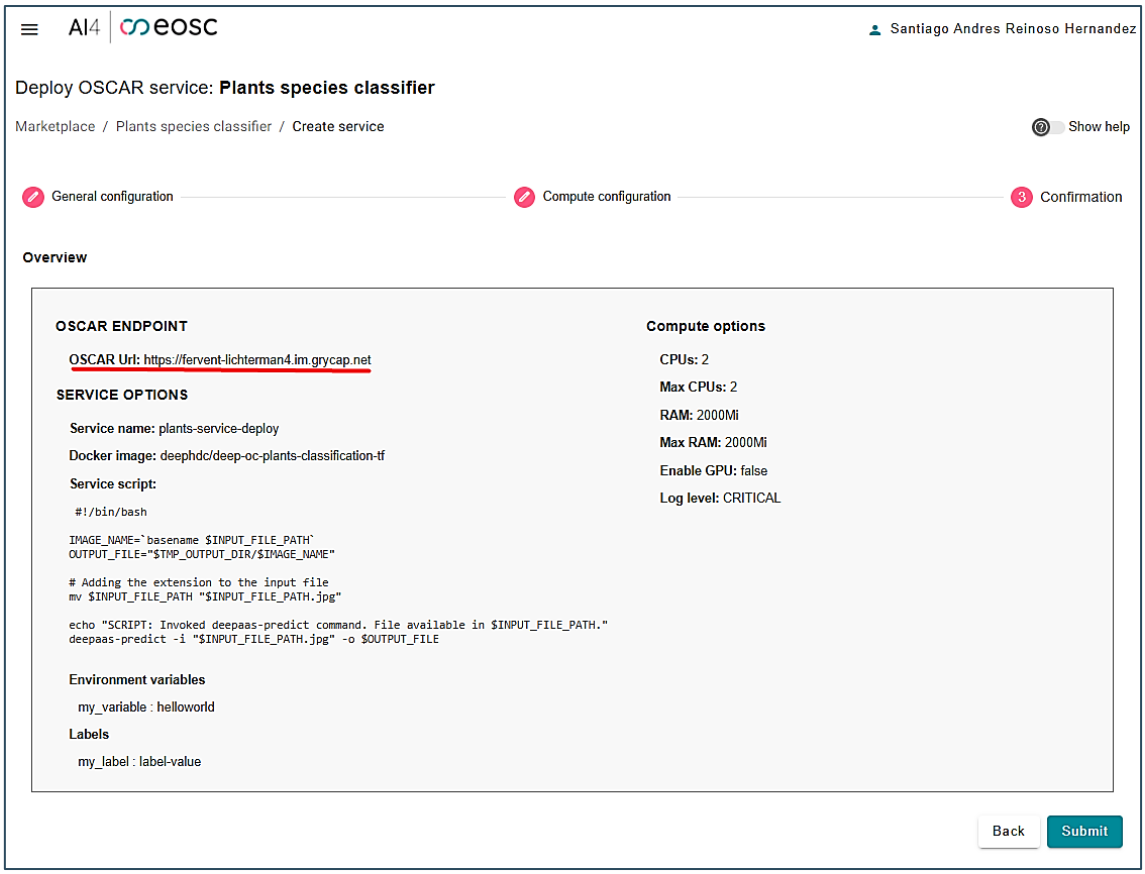

Figura 69. Formulario de despliegue, confirmación de datos

Una vez que todo está listo y revisado, se debe hacer clic en el botón "Submit" para iniciar la creación del servicio. Los datos recogidos por el formulario son enviados al cliente de oscar-js, que se encargará de llevar a cabo la creación en el clúster de OSCAR. Una vez que la solicitud se ha completado con éxito, el usuario recibe una notificación en la web y es redirigido al detalle del modelo como se puede ver en la Figura 70.

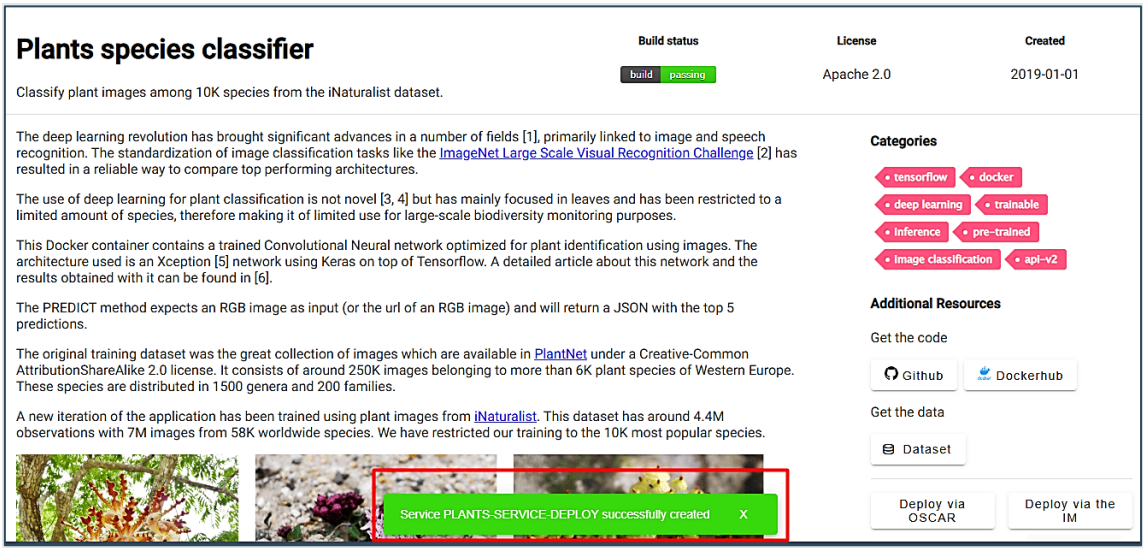

Figura 70. Notificación de confirmación, despliegue de servicio

4.3 Invocación de un servicio de OSCAR desde el portal

En esta sección se muestra la forma en que se invoca un servicio de OSCAR por medio del AI4EOSC *Dashboard*. Tras haber desplegado el modelo como un servicio en OSCAR, es hora de probar la invocación del modelo y mostrar su resultado de forma amable con el usuario.

En el componente de detalle del modelo (*module-detail*), se activan unos botones de interacción con el modelo. Esto se puede observar en la Figura 70. En este contexto, se utiliza el botón "Try module", que redirige al usuario al componente destinado a realizar la invocación del servicio en el clúster de OSCAR. La ubicación del botón "Try module" se muestra en la siguiente Figura 71.

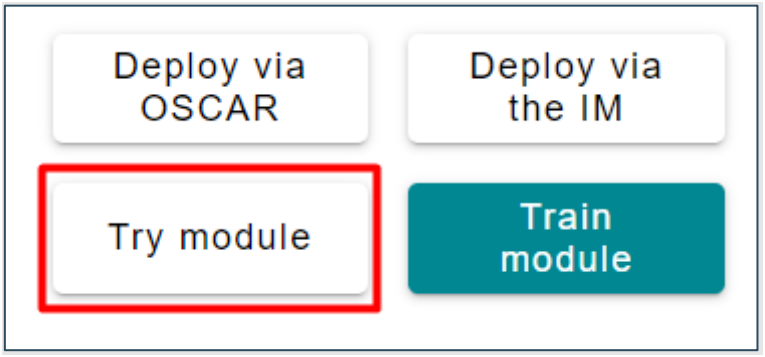

Figura 71. Opciones de interacción con el modelo

Este componente presenta un formulario que se encarga de solicitar los datos mínimos necesarios para iniciar la ejecución. La Figura 72 muestra un ejemplo de del formulario.

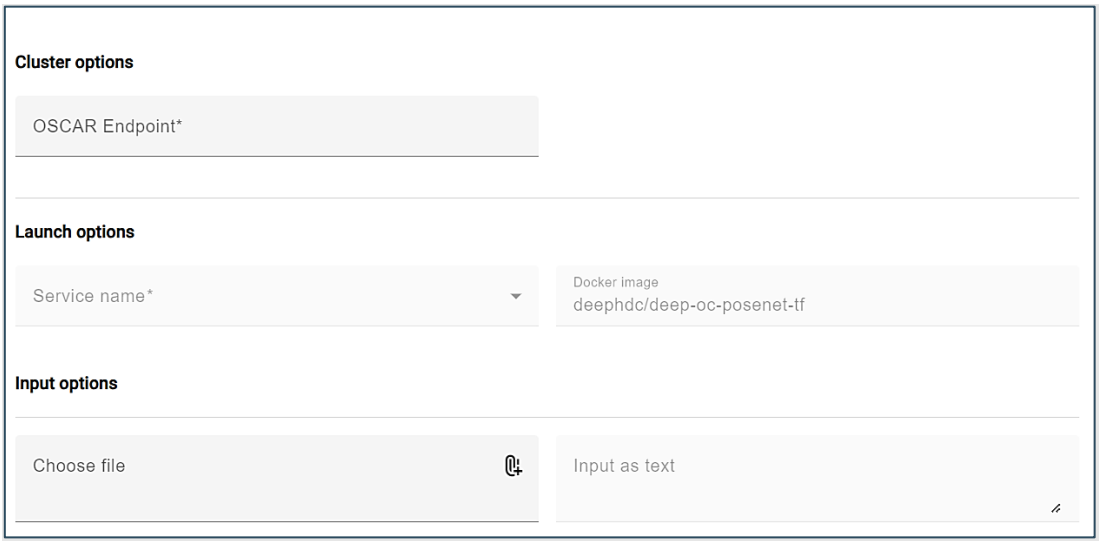

Figura 72. Formulario de invocación del servicio de OSCAR

El primer campo que debe completarse en este formulario es el campo de la URL del clúster de OSCAR donde se desea ejecutar el servicio. La Figura 73 muestra un ejemplo del *endpoint* de OSCAR clúster que ha sido generada previamente.

Figura 73. Formulario de despliegue, especificación del endpoint de OSCAR

Continuando con el formulario, como se muestra en la Figura 74, se encuentra un campo de selección del servicio. En este campo se cargarán todos los servicios existentes en el clúster de OSCAR indicado. Para este caso, se debe seleccionar el servicio creado anteriormente como se puede ver en la Figura 75. Anteriormente, como se explicó en la sección 4.3 de este documento, este servicio corresponde el modelo "Plants Species Classifier" un modelo que tiene como objetivo principal identificar el tipo de especie de la planta.

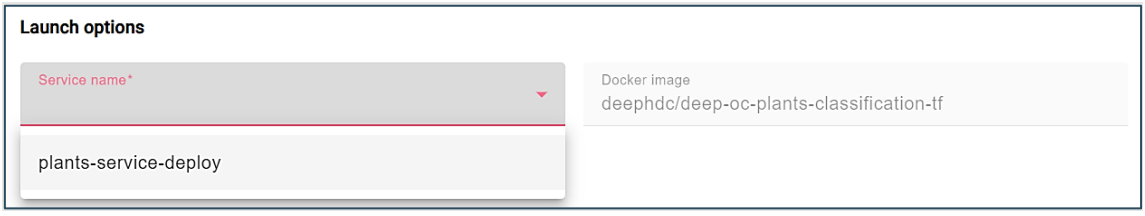

Figura 74. Opciones disponibles para el modelo de ejemplo

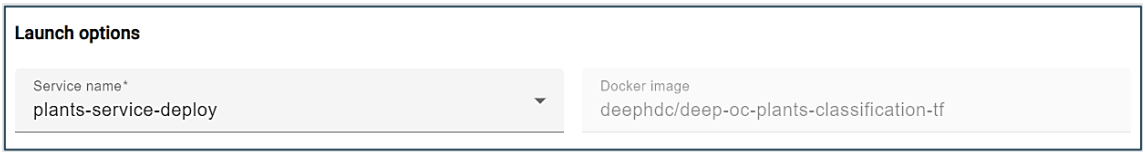

Figura 75. Nombre del servicio para hacer la prueba

Finalmente, se encuentran los campos de entrada de datos. Aquí es crucial mencionar que se debe cargar el archivo o el texto sobre el cual se quiere realizar el proceso de inferencia. Por lo tanto, este campo es obligatorio en el formulario. En la Figura 76 se puede observar un ejemplo de estos campos. En este caso, se seleccionará una imagen que será utilizada para invocar el servicio de OSCAR.

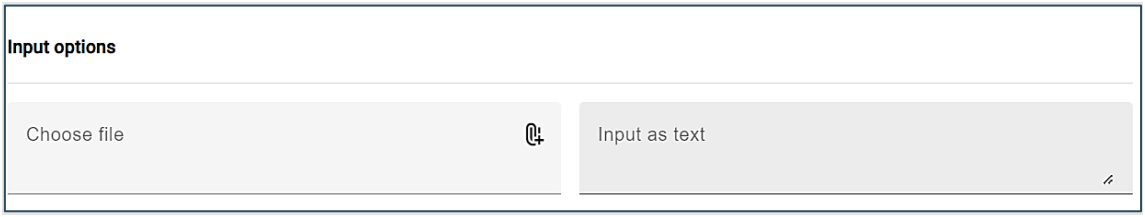

Figura 76. Formulario de despliegue, campos de entrada de datos

Para realizar esta prueba, se carga una imagen que representa un tipo específico de planta. En este caso, se utiliza la Figura 77 que retrata la palma de cera del Quindío, una especie nativa de los bosques montañosos húmedos andinos del Parque Nacional Natural Los

Nevados, en Colombia [50]. Esta planta se conoce científicamente como ***Ceroxylon quindiuense***.

Figura 77. Imagen de ejemplo, prueba modelo de clasificación de plantas

Una vez que se han proporcionado todos los datos requeridos en el formulario, el botón "Launch Service", ubicado en la esquina inferior derecha, se habilita. Al hacer clic en este botón, se ejecuta la función *runService()* en el cliente de "oscar-js", esta función envía una solicitud al clúster de OSCAR para invocar este servicio de manera síncrona, transmitiendo los datos de la imagen que se cargó previamente. La Figura 78 muestra el formulario completo y listo para invocar el servicio.

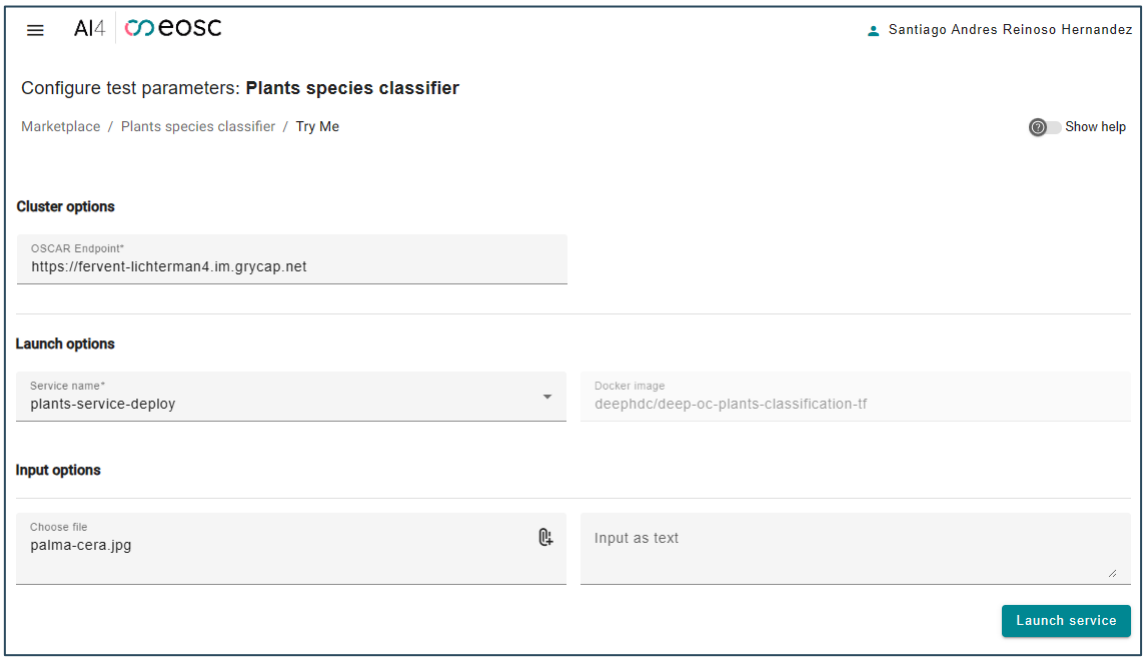

Figura 78. Formulario para invocar servicio de OSCAR completado

La Figura 79 ilustra un *spinner* el cual indica al usuario que se está llevando a cabo una ejecución y que debe esperar a que llegue la respuesta.

Figura 79. Invocación del servicio en progreso.

Después de finalizar la ejecución, el servicio recibe la respuesta de la inferencia. Esta respuesta es procesada por oscar-js, el cual examina el contenido y asigna el tipo de *MIME* correspondiente a la respuesta obtenida. Se obtiene cómo respuesta un objeto que contiene la propiedad "data", correspondiente al contenido codificado en base64 y la propiedad "mime", correspondiendo al tipo de *MIME* de la respuesta. En el siguiente fragmento de código se puede ver un ejemplo de la respuesta.

```
{
   "data": "base64 string",
   "mime": "application/octet-stream"
}
```

Para obtener más detalles sobre este proceso, se sugiere consultar la sección 3.3.6 de este documento. Una vez identificada la firma del archivo, el *dashboard* puede reaccionar al contenido de la respuesta y mostrarlo de manera amigable al usuario. En este caso, la Figura 80 ilustra el resultado de la ejecución en una tabla de resultados. Además, en la parte superior de la tabla se destacan los tres resultados con mayor porcentaje de precisión.

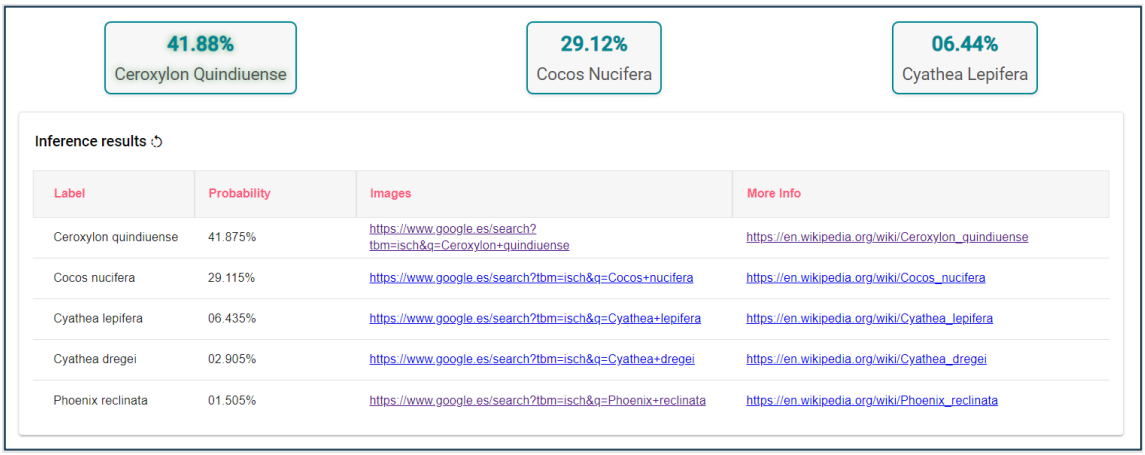

Figura 80. Resultado de la invocación en formato de tabla

La tabla de resultados en este ejemplo muestra la información pertinente en columnas. La primera columna presenta la etiqueta del resultado, mientras que la segunda muestra el porcentaje de probabilidad. En la tercera y cuarta columna se proporcionan los enlaces para obtener la imagen y la información correspondiente, respectivamente.

4.4 Prueba con diferentes módulos del marketplace

En esta ocasión, se llevará a cabo una prueba con varios modelos de IA previamente entrenados y disponibles en el *marketplace* de AI4EOSC *Dashboard* y se analizarán sus distintos tipos de respuestas. Se podrá observar cómo cada modelo reacciona ante diferentes entradas y escenarios. El AI4EOSC *Dashboard* responderá de manera dinámica a estas variadas respuestas, presentando el contenido de manera óptima para el usuario.

A través del AI4EOSC *Dashboard*, se procederá a crear varios servicios de OSCAR, cada uno diseñado para recibir un archivo de entrada específico. Como resultado, se obtendrá una respuesta que se mostrará en pantalla o un contenido multimedia, el cual será representado de manera interactiva en la interfaz web.

4.4.1 Birds sound classifier

Este modelo se enfoca en la clasificación de sonidos de aves utilizando Aprendizaje Profundo. El método PREDICT espera un archivo de audio como entrada y devuelve un JSON con las 5 principales predicciones. En la Figura 81 se muestra en la parte superior el nombre del módulo y su descripción. En esta ocasión, se aborda directamente la invocación del servicio de OSCAR. El proceso de despliegue sigue el mismo procedimiento descrito en la sección 4.2 de este documento, lo que implica realizar cambios tanto en el script como en la imagen Docker del modelo.

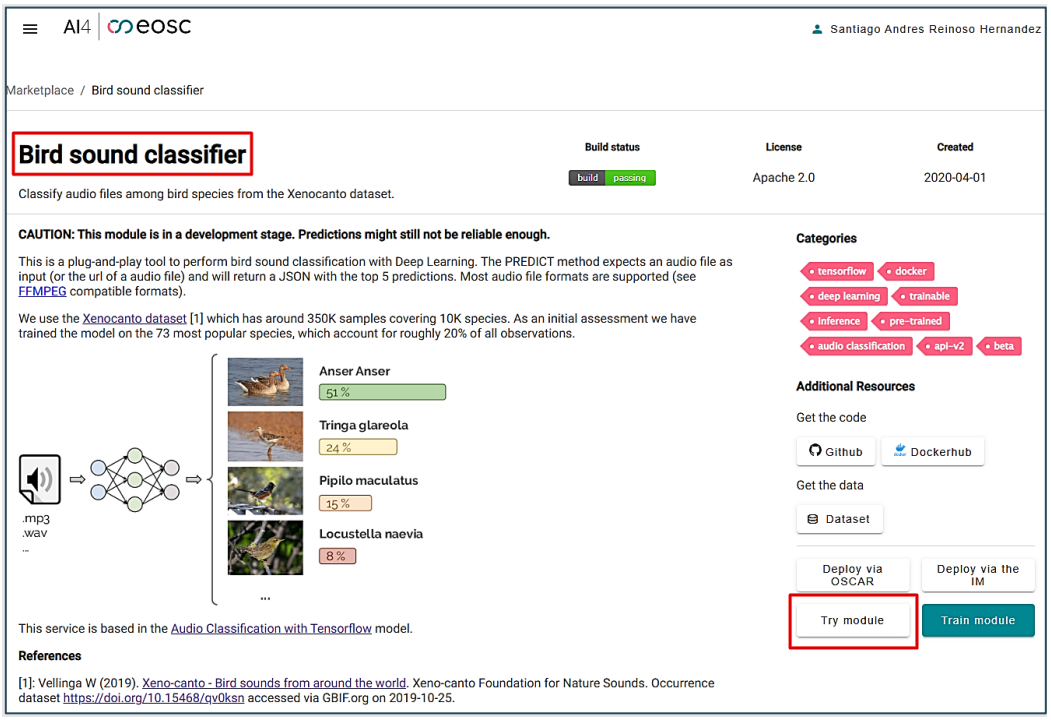

Figura 81. Detalle del modulo

Esta prueba requiere de un fichero de audio como entrada para hacer la inferencia. Para este caso, se utiliza el sonido de un cuervo (Corvus corax), una especie de ave común en diversos lugares del mundo. La Figura 82 fue obtenida del sitio web de GBIF (Global Biodiversity Information Facility), una red internacional respaldada por varios países. GBIF proporciona acceso abierto a datos sobre la biodiversidad mundial [51].

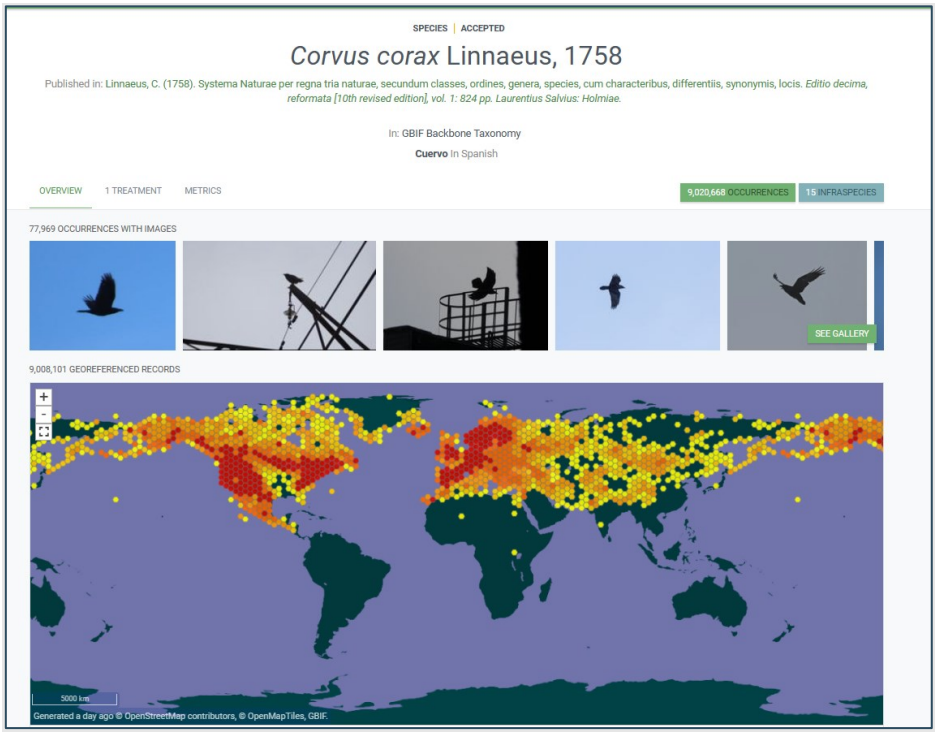

Figura 82. Información del ave utilizada para la prueba

Después de obtener el archivo de audio en formato MP3 para utilizar como entrada en este modelo, se activa el servicio de OSCAR diseñado específicamente para esta tarea. Este servicio se encarga de analizar el audio y determinar la especie de ave asociada con él. La activación del proceso se lleva a cabo desde el portal web, como se muestra en detalle en la Figura 83. Es importante destacar que este paso marca el inicio del proceso de inferencia, donde el sistema procesará la información y generará los resultados correspondientes. Una vez completada la ejecución, los resultados estarán disponibles para su visualización y análisis.

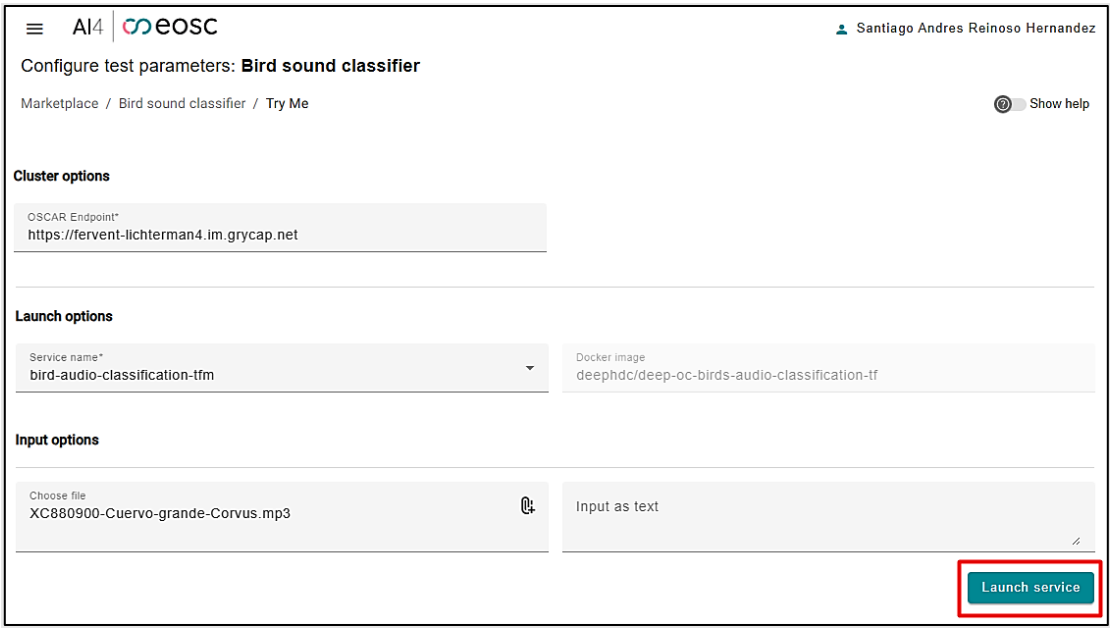

Figura 83. Formulario de invocación del servicio con los campos llenos

Después de que el servicio se ha ejecutado, se recibe la respuesta generada por oscar-js. Esta respuesta incluye las propiedades "data" y "mime", las cuales determinan el contenido que se muestra dependiendo del tipo *MIME*. Como se muestra en la Figura 84, la respuesta se presenta al usuario en formato de tabla.

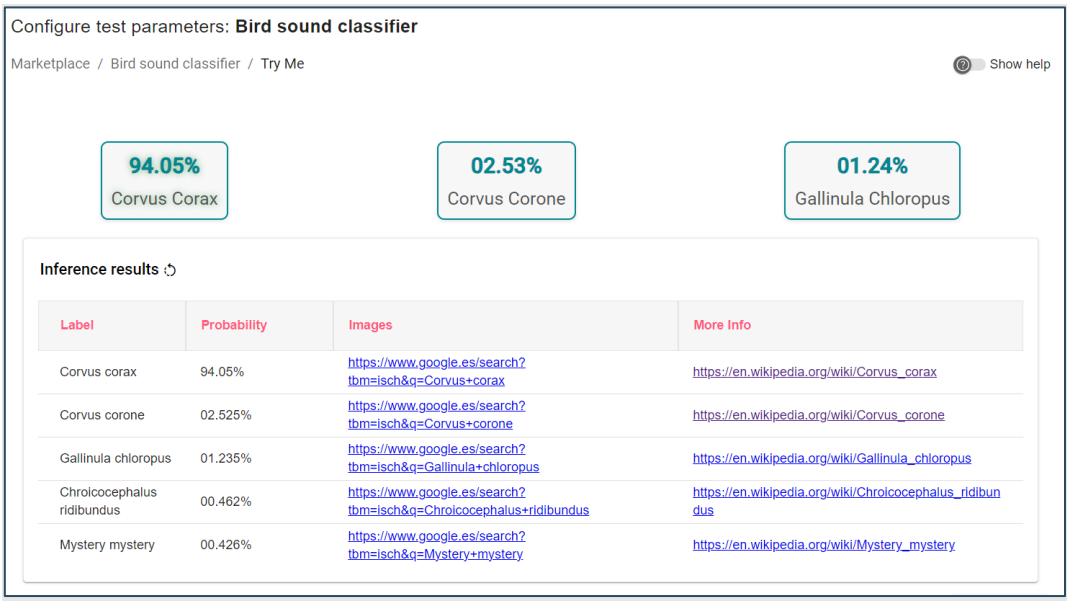

Figura 84. Respuesta del modelo probado en formato de tabla

Para entender el proceso de forma más visual, se ha dejado el enlace a un breve vídeo demostrativo que muestra el proceso de ejecución de un modelo como el anterior [19].

[19] Enlace demo: https://www.dropbox.com/scl/fi/ljrkc0mk496otd6dg7kex/demo-execution-1.mp4?rlkey=wqlqjbki1wslnuip50guv8bys&dl=0

4.4.2 Body pose detection

En este caso, se trata de un modelo que utiliza redes neuronales para estimar la postura de las personas y puede detectar tanto una como varias personas en una imagen, en la Figura 85 se puede ver en detalle la descripción de este modelo. El modelo recibe como entrada una imagen en formato JPG o PNG. Como resultado, se puede obtener una carpeta comprimida en formato ZIP o una imagen que marca con puntos las partes importantes del cuerpo, como los ojos, la nariz, los brazos, entre otros. El proceso de despliegue sigue el mismo procedimiento descrito en la sección 4.2 de este documento, lo que implica realizar cambios tanto en el script como en la imagen Docker del modelo.

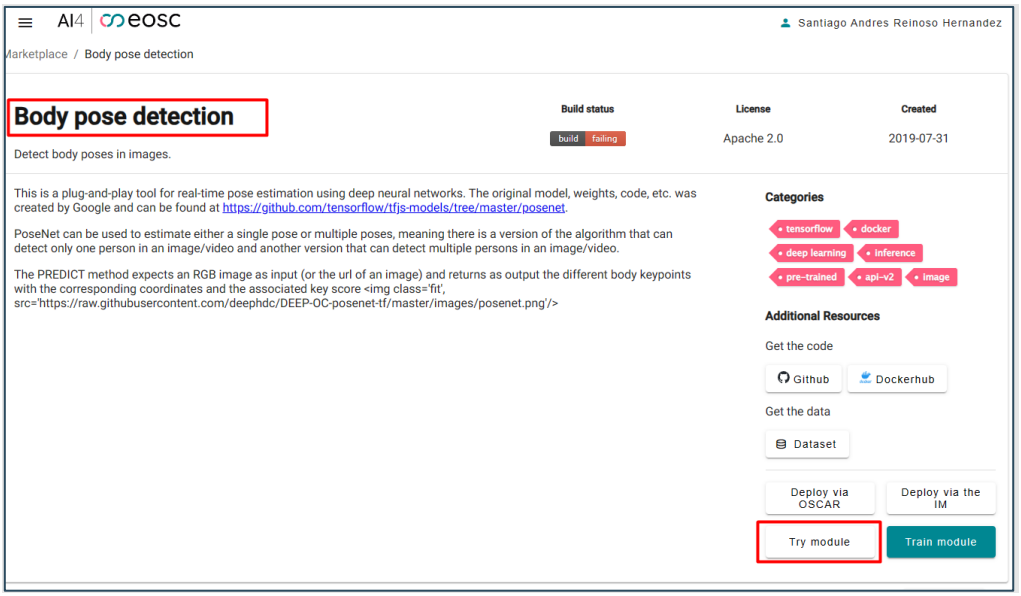

Figura 85. Descripción de modelo "Body pose detection"

Para hacer la prueba se ha escogido una imagen en la web de una persona con una posición corporal con la que se puede hacer el experimento. La Figura 86 muestra la imagen seleccionada para la prueba, donde se ve la pose de un conocido futbolista celebrando un gol.

Figura 86. Imagen de ejemplo para el modelo "Body pose"

Una vez se ha ejecutado el servicio se obtiene la respuesta proveniente de oscar-js, esta respuesta tendrá como propiedades “data” y “mime” con esta respuesta se puede mostrar el contenido dependiendo del tipo *MIME*. La Figura 87 revela la respuesta al usuario esta vez en un formato de imagen.

Figura 87. Respuesta del modelo en formato de imagen

Después de verificar la salida como imagen para este modelo, también se busca realizar el experimento, pero en esta ocasión se establece un nuevo servicio que devuelve un directorio comprimido como archivo ZIP. Se repite la ejecución del nuevo servicio utilizando la misma imagen, pero ahora se espera una salida en un formato diferente. El objetivo de esta prueba es validar que el AI4EOSC *Dashboard* puede reaccionar y mostrar datos de cualquier tipo conocido, como imágenes, JSON, videos o archivos ZIP. La Figura 88 muestra el resultado de la ejecución del nuevo servicio de OSCAR.

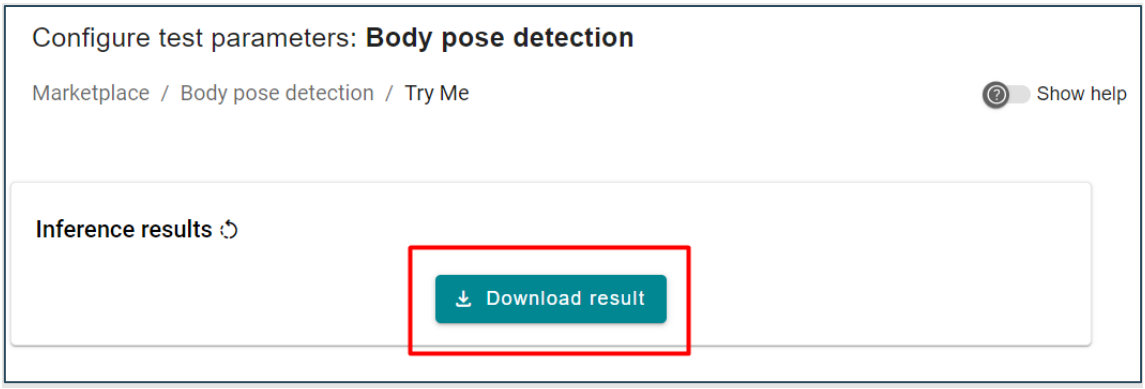

Figura 88. Respuesta de la invocación de modelo como descargable

Este botón descargará un fichero comprimido en formato ZIP como se ha mencionado anteriormente. La Figura 89 representa el contenido comprimido en el fichero descargado.

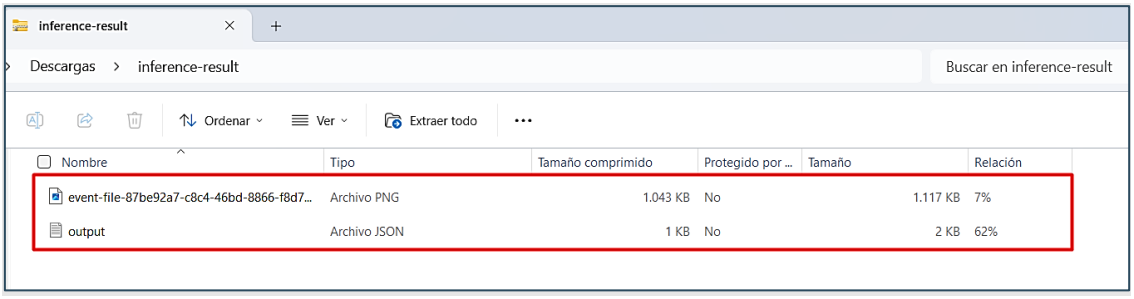

Figura 89. Contenido del fichero descargado como respuesta

La respuesta consta de dos ficheros, el primero es una imagen que contiene los puntos marcados en la imagen como se hizo en la ejecución anterior, el segundo es un fichero en formato JSON que contiene la información sobre los puntos marcados en la imagen.

Para entender el proceso de forma más visual, se ha dejado el enlace a un breve vídeo demostrativo que muestra el proceso de ejecución de un modelo como el anterior [20].

[20] Enlace demo: https://www.dropbox.com/scl/fi/ljrkc0mk496otd6dg7kex/demo-execution-1.mp4?rlkey=wqlqjbki1wslnuip50guv8bys&dl=0

En el siguiente fragmento de código se coloca el contenido del fichero JSON entregado como resultado de la ejecución previa.


```
[
  {
    "img_path": "/tmp/2024-04-12 00:23:14-173682/event-file-31ac4bff-7b88-4421-
80ea-c940de2a64c4.png",
    "nose": {
      "coordinate_x": 106.72471000771046,
      "coordinate_y": 554.845741417953,
      "score": 0.99779212474823
    },
    "leftEye": {
      "coordinate_x": 93.30078666199795,
      "coordinate_y": 565.0920224223372,
      "score": 0.9976756572723389
    },
    "rightEye": {
      "coordinate_x": 94.32337038402807,
      "coordinate_y": 541.0681302857183,
      "score": 0.9986294507980347
    },
    "leftEar": {
      "coordinate_x": 102.122546488038,
      "coordinate_y": 584.905848169855,
      "score": 0.9502601623535156
    },
    "rightEar": {
      "coordinate_x": 105.74319020065064,
      "coordinate_y": 525.3401003527377,
      "score": 0.9304054975509644
    },
    "leftShoulder": {
      "coordinate_x": 149.18506647229023,
      "coordinate_y": 611.2969945502305,
      "score": 0.24620959162712097
    },
    "rightShoulder": {
      "coordinate_x": 163.97690350846733,
      "coordinate_y": 500.46214043962027,
      "score": 0.2648695111274719
    },
    "leftElbow": {
      "coordinate_x": 116.24797466704393,
      "coordinate_y": 715.5358972309339,
      "score": 0.6413379311561584
    },
    "rightElbow": {
      "coordinate_x": 157.89717589822666,
      "coordinate_y": 389.2903419025955,
      "score": 0.23144972324371338
    },
    "leftWrist": {
      "coordinate_x": 80.9730371041291,
      "coordinate_y": 801.8698898883985,
      "score": 0.573189377784729
    },
    "rightWrist": {
      "coordinate_x": 138.3455639134644,
      "coordinate_y": 287.177440816182,
      "score": 0.7218508124351501
    },
    "leftHip": {
      "coordinate_x": 345.2894170675292,
      "coordinate_y": 625.7041526095264,
      "score": 0.5182304382324219
    },
    "rightHip": {
      "coordinate_x": 368.38779414859323,
      "coordinate_y": 542.4883311599162,
      "score": 0.3256857097148895
```

```
    },
    "leftKnee": {
      "coordinate_x": 490.20584006787044,
      "coordinate_y": 657.9066154810357,
      "score": 0.7466671466827393
    },
    "rightKnee": {
      "coordinate_x": 530.2580278393839,
      "coordinate_y": 570.3921947959446,
      "score": 0.18472418189048767
    },
    "leftAnkle": {
      "coordinate_x": 638.8946796190237,
      "coordinate_y": 692.0494655520774,
      "score": 0.8536093235015869
    },
    "rightAnkle": {
      "coordinate_x": 536.0483343957644,
      "coordinate_y": 558.4584059239875,
      "score": 0.09147587418556213
    }
  }
]
```

5. Conclusiones

En el presente capítulo, se presentarán las conclusiones obtenidas a partir de este trabajo académico. Se destacará la relevancia de los nuevos artefactos desarrollados durante el proyecto y cómo estas funcionalidades contribuyen al campo de estudio de la inteligencia artificial. Además, se explorarán las posibles áreas de desarrollo futuro que podrían surgir a partir de los resultados obtenidos en este trabajo.

En este proyecto, se ha logrado una integración exitosa entre un clúster de OSCAR y el AI4EOSC *Dashboard*. Este logro se debe en gran parte al reciente desarrollo del módulo cliente oscar-js, que ha desempeñado un papel fundamental en esta tarea. El objetivo principal del módulo oscar-js es proporcionar una interfaz fácil de usar para interactuar con un clúster de OSCAR utilizando JavaScript, una capacidad que previamente no estaba disponible. Gracias a este cliente, ahora es posible realizar diversas operaciones en el clúster de OSCAR, como la creación, edición y eliminación de servicios, de una manera cómoda para el desarrollador. Además, la versatilidad y facilidad de implementación de oscar-js lo hacen adecuado para su uso en una variedad de proyectos que requieran la utilización de JavaScript, no limitándose únicamente al proyecto AI4EOSC *Dashboard.*

Con el desarrollo del cliente oscar-js, se dio el inicio de la implementación de nuevos componentes en el AI4EOSC *Dashboard*. Estos componentes, llamados **"module-oscar-deploy"** y **"module-try",** han dotado al portal web con funcionalidades adicionales. Específicamente, estos nuevos componentes tienen la tarea de recopilar los datos esenciales para la creación e invocación de los servicios de OSCAR.

Con la información previa, podemos repasar los objetivos de este trabajo y confirmar si se han alcanzado todos. A continuación, destacamos los siguientes puntos:

- Usando OSCAR como plataforma *serverless* ahora es posible crear y ejecutar modelos desde el AI4EOSC *Dashboard*.
- Es posible crear servicios de inteligencia artificial en clústeres distribuidos de OSCAR a través del AI4EOSC *Dashboard* utilizando el componente "module-oscar-deploy".
- Las invocaciones síncronas de servicios de OSCAR son posibles desde el AI4EOSC *Dashboard,* haciendo uso del componente “module-try”.
- Se ha desarrollado un cliente utilizando TypeScript que facilita la integración entre un clúster de OSCAR y el AI4EOSC *Dashboard*. Este cliente es empleado por el *dashboard* como una dependencia adicional.
- Las nuevas funcionalidades del *dashboard* están listas para ponerse en un ambiente productivo.

Más allá de los objetivos de este trabajo, se busca contribuir a la comunidad investigadora y, principalmente, facilitar a los investigadores en el ámbito de la Inteligencia Artificial

el acceso y la utilización de modelos existentes o propios. Se proporciona un entorno fácil y de rápida respuesta para la realización de pruebas. A través del uso del AI4EOSC *Dashboard*, los investigadores pueden aprovechar OSCAR como plataforma *serverless*, lo que les permite beneficiarse del ahorro en costos de infraestructura gracias a la escalabilidad a cero que ofrece OSCAR.

En todos los proyectos de desarrollo, es común enfrentarse a dificultades a lo largo del camino, y este caso no fue una excepción. Se ha observado que el **API ai4-papi** no proporciona información sobre el tipo de respuesta que se espera del modelo. Por ejemplo, no se puede determinar si el modelo puede devolver imágenes, archivos comprimidos en formato zip o simplemente datos en formato JSON. Esta falta de información presenta un desafío para el AI4EOSC *Dashboard*, ya que no puede renderizar adecuadamente el contenido sin conocer su tipo. Actualmente, la solución consiste en devolver una respuesta codificada en base64 y probar diferentes métodos hasta que se logre mostrar el contenido, aunque este enfoque no es muy eficaz.

Este problema se identificó como una oportunidad clave para mejorar, ya que llevó a una investigación sobre cómo determinar el tipo de contenido de la respuesta codificada en base64 generada por la invocación del servicio en OSCAR. La solución encontrada consistió en utilizar los "bytes mágicos", que son los primeros bytes del archivo y generalmente contienen la firma del archivo. Esta verificación de los bytes mágicos se realiza en el cliente de oscar-js antes de que la respuesta se envíe al AI4ESOC *Dashboard*. Una vez que el cliente de oscar-js encuentra una coincidencia entre los bytes mágicos y el contenido del archivo, devuelve al *dashboard* un objeto con la respuesta codificada en base64 y el tipo de contenido de la respuesta. Con esta validación, el AI4ESOC *Dashboard* puede prepararse para mostrar el contenido de manera adecuada, lo que mejora la experiencia del usuario al eliminar la preocupación sobre el tipo de contenido que el modelo responde.

Un aspecto destacable de este trabajo fue la colaboración y coordinación con el equipo de CSIC, encargado de administrar y mantener el proyecto del AI4EOSC *Dashboard*. Mantener una comunicación fluida con este equipo fue crucial para la integración de nuevos cambios en el repositorio y los despliegues en diferentes entornos. Dado que el proyecto ya contaba con un diseño estandarizado, fue esencial mantener la coherencia visual de la plataforma, utilizando la misma biblioteca de componentes y respetando la paleta de colores establecida.

En los nuevos componentes del AI4EOSC *Dashboard*, se ha buscado proporcionar *feedback* constante al usuario sobre los datos ingresados. Por ejemplo, al desplegar un nuevo servicio de OSCAR, se brinda al usuario la oportunidad de revisar la información proporcionada. Además, es posible visualizar una vista previa del contenido del script cargado directamente en la web.

5.1 Recomendaciones y trabajos futuros

Una mejora potencial para el proyecto oscar-js sería la implementación de un sistema de integración y entrega continúa utilizando GitHub Actions. De esta manera, cada vez que

un desarrollador crea una *Pull Request* o se añade nuevo contenido a la rama principal del repositorio del proyecto, se lanzará automáticamente una nueva versión del cliente. Esto facilitará su implementación en otros proyectos, ya que solo será necesario instalar la versión más reciente disponible.

Una mejora adicional en este trabajo implica la utilización directa del API del servicio que ofrece OSCAR a través de un "ingress". Esta opción se logra aprovechando los servicios expuestos proporcionados por OSCAR. La ventaja de esta estrategia es que el modelo solo necesita inicializarse una vez y podrá responder de manera rápida a las próximas solicitudes.

Para usuarios externos, sería muy útil si no fuera necesario estar afiliado a las organizaciones virtuales (VOs) para realizar operaciones dentro del AI4EOSC *Dashboard*. Solicitar la afiliación implica invertir tiempo en completar formularios, encontrar la VO adecuada, solicitar autorización y esperar la aprobación, lo que puede demorarse varios días. Sin embargo, este requerimiento viene impuesto por la naturaleza de colaboración con un proyecto de investigación y por las características de acceso a los recursos de EGI.

Bibliografía

[1] Europe Union, «AI4EOSC,» [En línea]. Available: https://ai4eosc.eu.

[2] European Open Science Cloud, «Deep Hybrid Datacloud,» [En línea]. Available: https://deep-hybrid-datacloud.eu.

[3] Europe Union, «CORDIS | European Commision,» 01 09 2022. [En línea]. Available: https://cordis.europa.eu/project/id/101058593.

[4] European Open Science Cloud, «AI4EOSC Dashboard,» [En línea]. Available: https://dashboard.cloud.ai4eosc.eu.

[5] Grupo de Grid y Computación de Altas Prestaciones, «OSCAR Documentation,» [En línea]. Available: https://docs.oscar.grycap.net.

[6] European Open Science Cloud, «DEEP Open Catalog,» [En línea]. Available: https://marketplace.deep-hybrid-datacloud.eu.

[7] OpenID Foundation, «How OpenID Connect Works,» [En línea]. Available: https://openid.net/developers/how-connect-works.

[8] National Institute of Standards and Technology, «The NIST Definition of Cloud,» Septiembre 2011. [En línea]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

[9] Z. Mahmood, de *Cloud Computing for Enterprise Architectures: Concepts, Principles and Approaches*, Springer, London, 2011.

[10] S. K. S. Manvi y G. K. Shyam, «Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey,» *Journal of Network and Computer Applications,* vol. Volume 41, pp. 424-440, 2014.

[11] A. Giessmann y K. Stanoevska, «Platform as a Service – A Conjoint Study on Consumers Preferences,» *International Conference on Information Systems, ICIS,* vol. 4, pp. 1-16, 2012.

[12] A. Benlian y T. Hess, «Opportunities and risks of software-as-a-service: Findings from a survey of IT executives,» *Decision Support Systems,* vol. 52, pp. 232-246, 2011.

[13] Amazon Web Services, «Serverless Computing - Amazon Web Services,» 2024. [En línea]. Available: https://aws.amazon.com/es/serverless.

[14] Amazon Web Services, «AWS Lambda,» 2024. [En línea]. Available: https://aws.amazon.com/es/lambda.

[15] Google Cloud Platform, «Cloud Functions,» 2024. [En línea]. Available: https://cloud.google.com/functions.

[16] Microsoft Azure, «Azure Functions,» 2024. [En línea]. Available: https://azure.microsoft.com/en-us/products/functions.

[17] Netlify, «Serverless Web Apps,» 2024. [En línea]. Available: https://www.netlify.com/for/web-applications/.

[18] M. Helm, A. Swiergosz, H. Haeberle, J. Kornuta, J. Schaffer, V. Krebs, A. Spitzer y P. N. Ramkumar, «Machine Learning and Artificial Intelligence: Definitions, Applications, and Future Directions,» *Current Reviews in Musculoskeletal Medicine,* vol. 13, p. 69–76, 2020.

[19] Amazon Web Services, «¿Que es la inteligencia artificial?,» 2024. [En línea]. Available: https://aws.amazon.com/es/what-is/artificial-intelligence.

[20] D. D'Silva y D. Ambawade, «Building A Zero Trust Architecture Using Kubernetes,» de *6th International Conference for Convergence in Technology (I2CT)*, Maharashtra, India, 2021.

[21] Baeldung, «Serverless Architecture with Knative,» 2024. [En línea]. Available: https://www.baeldung.com/ops/knative-serverless.

[22] The Knative Authors, «Overview - Knative,» 2024. [En línea]. Available: https://knative.dev/docs/concepts/.

[23] Amazon Web Services, «¿Qué es Amazon S3?,» 2024. [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html#CoreConcepts.

[24] Amazon Web Services, «Alojamiento de un sitio web estático mediante Amazon S3,» 2024. [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/WebsiteHosting.html.

[25] IBM Corporation, «MinIO - Documentation IBM,» 2017. [En línea]. Available: https://www.ibm.com/docs/es/cloud-private/3.2.x?topic=private-minio.

[26] Minio Inc, «MinIO Object Storage Container,» 2024. [En línea]. Available: https://min.io/docs/minio/container/administration/concepts.html.

[27] Okta Inc, «¿Qué es OpenID Connect y para qué se usa?,» 2024. [En línea]. Available: https://auth0.com/es/intro-to-iam/what-is-openid-connect-oidc.

[28] A. Pérez, S. Riasco, D. M. Naranjo, M. Caballer y G. Moltó, «On-Premises Serverless Computing for Event-Driven Data Processing Applications,» de *2019 IEEE 12th International Conference on Cloud Computing*, Milan, Italy, 2019.

[29] M. Caballer, G. Moltó, A. Calatrava y I. Blanquer, «Infrastructure Manager: A TOSCA-Based Orchestrator for the Computing Continuum,» *Journal of Grid Computing,* vol. 21, p. 51, 203.

[30] Grupo de Grid y Computación de Altas Prestaciones, «Functions Definition Language,» 2022. [En línea]. Available: https://docs.oscar.grycap.net/fdl.

[31] Grupo de Grid y Computación de Altas Prestaciones, «CLUES - Cluster energy saving,» 2010. [En línea]. Available: https://www.grycap.upv.es/clues/es/index.php.

[32] Netlify, «Welcome to Jamstack,» 2024. [En línea]. Available: https://www.netlify.com/jamstack.

[33] Wikipedia, «Node.js - Wikipedia,» 2009. [En línea]. Available: https://en.wikipedia.org/wiki/Node.js.

[34] Kind, «kind- Quick Start,» [En línea]. Available: https://kind.sigs.k8s.io/docs/user/quick-start/#creating-a-cluster.

[35] Grupo de Grid y Computación de Altas Prestaciones, «Local testing - OSCAR Documentation,» 2022. [En línea]. Available: https://docs.oscar.grycap.net/local-testing.

[36] Grupo de Grid y Computación de Altas Prestaciones, «OpenAPI Specification - OSCAR Documentation,» 2022. [En línea]. Available: https://docs.oscar.grycap.net/api/.

[37] Grupo de Grid y Computación de Altas Prestaciones, «Invoking services - OSCAR Documentation,» [En línea]. Available: https://docs.oscar.grycap.net/invoking.

[38] Kubernetes, «Overview - Knative,» [En línea]. Available: https://knative.dev/docs/serving/.

[39] Grid y Computación de Altas Prestaciones, «Javascript client for OSCAR,» 2024. [En línea]. Available: https://github.com/grycap/oscar-js.

[40] Wikipedia, «ECMAScript,» 1997. [En línea]. Available: https://es.wikipedia.org/wiki/ECMAScript.

[41] Mozilla Corporation, «Autenticación HTTP,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Authentication.

[42] Internet Engineering Task Force, «JSON Web Token,» 2015. [En línea]. Available: https://jwt.io/.

[43] N. C. Zakas, Understanding Javascript Promises, Lulu.com, 2020.

[44] Wkipedia, «List of file signatures,» [En línea]. Available: https://en.wikipedia.org/wiki/List_of_file_signatures.

[45] J. Enders, Design UX: Forms, SitePoint, 2016.

[46] Google, «Stepper Angular Material,» 2014. [En línea]. Available: https://material.angular.io/components/stepper/overview.

[47] Google, «Introduction to services and dependency injection,» 2014. [En línea]. Available: https://angular.io/guide/architecture-services.

[48] GitHub Inc, «Documentación de GitHub Actions,» 2024. [En línea]. Available: https://docs.github.com/es/actions.

[49] European Grid Infrastructure, «EGI - Advanced computing service for research,» 2024. [En línea]. Available: https://www.egi.eu.

[50] Wikipedia, «Ceroxylon Quindiuense,» [En línea]. Available: https://es.wikipedia.org/wiki/Ceroxylon_quindiuense.

[51] Global Biodiversity Information Facility, «What is GBIF?,» 2024. [En línea]. Available: https://www.gbif.org/what-is-gbif.

Anexo 1. Esquema Service

Propiedades del servicio de OSCAR

Anexo 2. Clase *Client*

```
<import { DeleteRequest, PostRequest, PutRequest } from "./requests";
import { GetRequest } from "./requests/GetRequest";
import { UnauthorizedError } from "./errors/UnauthorizedError";
import { AuthType, ClusterAuth, Config, Info, JobInfo, RequestProps, Service } from
"./types";
import { decodeFromBase64, getMimeType } from "./utils/utils";

const pathConfig = "/system/config/";
const pathInfo = "/system/info/";
const pathHealth = "/health";
const pathServices = "/system/services/";
const pathLogs = "/system/logs/";
const runSync = "/run/";

export class Client {
    private _oscar_endpoint: string;
    private _username: string | undefined;
    private _password: string | undefined;
    private _oidc_token: string | undefined;
    private _auth_type: AuthType;

    public constructor({ clusterId, oscar_endpoint, username, password, oidc_token
}: ClusterAuth) {
        this._oscar_endpoint = oscar_endpoint;
        this._username = username;
        this._password = password;
        this._oidc_token = oidc_token;

        // Set auth type looking for input params
        if (username !== undefined && password !== undefined) {
            this._auth_type = AuthType.BasicAuth;
        } else {
            this._auth_type = AuthType.Oidc;
        }
    }

    getOscarEndpoint(): string {
        return this._oscar_endpoint;
    }

    getUsername(): string | undefined {
        return this._username;
    }

    getPassword(): string | undefined {
        return this._password;
    }

    getAuthType(): AuthType {
        return this._auth_type;
    }

    getOidcToken(): string | undefined {
        return this._oidc_token;
    }

    /**
     * Get OSCAR cluster information
     * @returns cluster info
     */
    async getClusterInfo(): Promise<Info> {
        const props: RequestProps = {
            client: this,
            path: pathInfo,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };
```

```
        const request = new GetRequest<Info>();
        const rta = await request.getRequest(props);
        return rta;
    }

    /**
     * Get OSCAR cluster configuration
     * @returns configuration params
     */
    async getClusterConfig(): Promise<Config> {
        const props: RequestProps = {
            client: this,
            path: pathConfig,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new GetRequest<Config>();
        const rta = await request.getRequest(props);
        return rta;
    }

    /**
     * Get OSCAR cluster health status
     * @returns configuration params
     */
    async getHealthStatus(): Promise<string> {
        const props: RequestProps = {
            client: this,
            path: pathHealth,
            authorization: this._oidc_token,
            textResponse: true,
            authType: this._auth_type,
        };

        const request = new GetRequest<string>();
        const rta = await request.getRequest(props);
        return rta;
    }

    /**
     * Get list of services in OSCAR cluster
     * @returns services
     */
    async getServices(): Promise<Service[]> {
        const props: RequestProps = {
            client: this,
            path: pathServices,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new GetRequest<Service[]>();
        const rta: Service[] = await request.getRequest(props);
        return rta;
    }

    /**
     * Get service by name
     * @param serviceName servce namer identifier
     * @returns service
     */
    async getServiceByName(serviceName: string): Promise<Service> {
        const props: RequestProps = {
            client: this,
            path: pathServices + serviceName,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
```

```
        };

        const request = new GetRequest<Service>();
        const rta: Service = await request.getRequest(props);
        return rta;
    }

    /**
     * Get logs by service name.
     * @param serviceName service name
     * @returns logs of executed jobs
     */
    async getLogsByService(serviceName: string): Promise<JobInfo[]> {
        const props: RequestProps = {
            client: this,
            path: pathLogs + serviceName,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new GetRequest<JobInfo[]>();
        const rta: JobInfo[] = await request.getRequest(props);
        return rta;
    }

    /**
     * Get logs by service an job id
     * @param serviceName service name
     * @param jobName job id
     * @returns logs of executed job
     */
    async getLogsByJob(serviceName: string, jobName: string): Promise<JobInfo> {
        const props: RequestProps = {
            client: this,
            path: pathLogs + serviceName + "/" + jobName,
            authorization: this._oidc_token,
            textResponse: true,
            authType: this._auth_type,
        };

        const request = new GetRequest<JobInfo>();
        const rta: JobInfo = await request.getRequest(props);
        return rta;
    }

    /**
     * Create a new service into OSCAR cluster.
     * @param body service params
     * @returns created service
     */
    async createService(body: Service): Promise<Service> {
        const props: RequestProps = {
            client: this,
            path: pathServices,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new PostRequest<Service>();
        const response: Service = await request.postRequest(props, body);
        return response;
    }

    /**
     * Update existing service into OSCAR cluster.
     * @param body service params
     * @returns update service
     */
    async updateService(body: Service) {
        const props: RequestProps = {
```

```
            client: this,
            path: pathServices,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new PutRequest<Service>();
        const response: Service = await request.putRequest(props, body);
        return response;
    }

    /**
     * Run a service synchronously
     * @param serviceName service you wish to run
     * @param dataEncoded input data encode in base64
     * @returns response inference of the invoked service
     */
    async runService(serviceName: string, dataEncoded: string): Promise<any> {
        const props: RequestProps = {
            client: this,
            path: runSync + serviceName,
            authorization: this._oidc_token,
            textResponse: true,
            authType: this._auth_type,
        };

        const service = await this.getServiceByName(serviceName);
        const serviceToken = service.token;
        props.authorization = "ServiceToken " + serviceToken;

        const request = new PostRequest<JSON>();
        const response: any = await request.postRequest(props, dataEncoded);

        const mimeType = getMimeType(response);
        return { mime: mimeType, data: response };
    }

    async deleteService(serviceName: string): Promise<string> {
        const props: RequestProps = {
            client: this,
            path: pathServices,
            authorization: this._oidc_token,
            textResponse: false,
            authType: this._auth_type,
        };

        const request = new DeleteRequest<Service>();
        const response: Service = await request.deleteRequest(props, serviceName);
        return response.name;
    }
}
```

Anexo 3. Clase *GetRequest*

Clase que hace una solicitud HTTP GET a la URL de OSCAR.

```
import { Client } from "../Client";
import { RequestProps } from "../types";
import { UnauthorizedError } from "../errors/UnauthorizedError";
import { setAuthParams } from "../utils/utils";
import fetch from "node-fetch";

export class GetRequest<T> {
    constructor() {}

    async getRequest(props: RequestProps, baseUrl?: string): Promise<any> {
        const { textResponse, path, client, authorization, authType } = props;
        let url;
        let auth;

        // Set authorization parameters according to configuration
        if (client) {
            url = new URL(path, client.getOscarEndpoint()).toString();
            auth = setAuthParams(props);
        } else {
            url = new URL(path, baseUrl).toString();
            auth = setAuthParams(props);
        }

        const headers = {
            "Content-Type": "application/json",
            Authorization: auth,
        };

        console.log("GET request in url: ", url);
        const response = await fetch(url, {
            method: "GET",
            headers: headers,
        });

        // Verify if response have an error (4xx or 5xx)
        if (!response.ok) {
            if (response.status == 401) {
                throw new UnauthorizedError("Unauthorized!!");
            }
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        if (textResponse) {
            return await response.text();
        }

        const data = await response.json();
        return data;
    }
}
```

Anexo 4. Clase *PostRequest*

Clase que hace una solicitud HTTP POST a la URL de OSCAR.

```
import { RequestProps } from "../types";
import { setAuthParams } from "../utils/utils";
import { UnauthorizedError } from "../errors/UnauthorizedError";
import fetch from "node-fetch";

export class PostRequest<T> {
    constructor() {}

    async postRequest(props: RequestProps, body: any, baseUrl?: string): Promise<T>
{
        const { textResponse, path, client, authorization, authType } = props;
        let url;
        let auth;

        // Set authorization parameters according to configuration
        if (client) {
            url = new URL(path, client.getOscarEndpoint()).toString();
            auth = setAuthParams(props);
        } else {
            url = new URL(path, baseUrl).toString();
            auth = setAuthParams(props);
        }

        const headers = {
            "Content-Type": "application/octet-stream",
            Authorization: auth,
        };
        console.log("POST request in url: ", url);

        const response = await fetch(url, {
            method: "POST",
            headers,
            body: JSON.stringify(body),
        });

        // Verify if response have an error (4xx or 5xx)
        if (!response.ok) {
            if (response.status == 401) {
                throw new UnauthorizedError("Unauthorized!!");
            }
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        //console.log("Response text --->", await response.text())
        if (response.status === 200) {
            return new Promise((resolve, reject) => {
                body = response.text();
                resolve(body);
            });
        }
        if (response.status === 201) {
            return new Promise((resolve, reject) => {
                resolve(body);
            });
        } else {
            throw new Error(`Unexpected status code: ${response.status}`);
        }
    }
}
```

Anexo 5. Clase *PutRequest*

Clase que hace una solicitud HTTP PUT a la URL de OSCAR.

```
import { Client } from "../Client";
import { RequestProps } from "../types";
import { UnauthorizedError } from "../errors/UnauthorizedError";
import { setAuthParams } from "../utils/utils";
import fetch from "node-fetch";

export class PutRequest<T> {
    constructor() {}

    async putRequest(props: RequestProps, body: any, baseUrl?: string): Promise<T>
{
        const { textResponse, path, client, authorization, authType } = props;
        let url;
        let auth;

        // Set authorization parameters according to configuration
        if (client) {
            url = new URL(path, client.getOscarEndpoint()).toString();
            auth = setAuthParams(props);
        } else {
            url = new URL(path, baseUrl).toString();
            auth = setAuthParams(props);
        }

        const headers = {
            "Content-Type": "application/octet-stream",
            Authorization: auth,
        };
        console.log("PUT Request in url: ", url);

        const response = await fetch(url.toString(), {
            method: "PUT",
            headers,
            body: JSON.stringify(body),
        });

        // Verify if response have an error (4xx or 5xx)
        if (!response.ok) {
            if (response.status == 401) {
                throw new UnauthorizedError("Unauthorized!!");
            }
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        if (response.status === 200) {
            return new Promise((resolve, reject) => {
                body = response.text();
                resolve(body);
            });
        }
        if (response.status === 201) {
            return new Promise((resolve, reject) => {
                resolve(body);
            });
        } else {
            throw new Error(`Unexpected status code: ${response.status}`);
        }
    }
}
```

Anexo 6. Clase *DeleteRequest*

Clase que hace una solicitud HTTP DELETE a la URL de OSCAR.

```
import { RequestProps } from "../types";
import { setAuthParams } from "../utils/utils";
import { UnauthorizedError } from "../errors/UnauthorizedError";
import fetch from "node-fetch";

export class DeleteRequest<T> {
    constructor() {}

    async deleteRequest(props: RequestProps, baseUrl?: string): Promise<any> {
        const { textResponse, path, client, authorization, authType } = props;
        let url;
        let auth;

        // Set authorization parameters according to configuration
        if (client) {
            url = new URL(path, client.getOscarEndpoint()).toString();
            auth = setAuthParams(props);
        } else {
            url = new URL(path, baseUrl).toString();
            auth = setAuthParams(props);
        }

        const headers = {
            Authorization: auth,
        };

        console.log("DELETE request in url: ", url);

        const response = await fetch(url.toString(), {
            method: "DELETE",
            headers,
        });

        // Verify if response have an error (4xx or 5xx)
        if (!response.ok) {
            if (response.status == 401) {
                throw new UnauthorizedError("Unauthorized");
            }
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        const status = response.status;
        return status;
    }
}
```

Anexo 7. Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.	X			
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante				X
ODS 8. Trabajo decente y crecimiento económico		X		
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.			X	
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Este proyecto está estrechamente relacionado con el Objetivo de Desarrollo Sostenible 4 (ODS 4). Su objetivo principal es reducir las barreras para la experimentación en el campo de la inteligencia artificial, promoviendo el uso de herramientas de código abierto accesibles para todos. Las funcionalidades de este proyecto permitirán a investigadores, estudiantes y expertos en inteligencia artificial crear, entrenar y desplegar nuevos modelos en una infraestructura europea abierta. De esta manera, se fomenta una educación inclusiva, equitativa y de calidad, en consonancia con el ODS 4.

El proyecto AI4EOSC, financiado por fondos europeos, se alinea con el Objetivo de Desarrollo Sostenible 8 (ODS 8), centrado en el trabajo decente y el crecimiento económico. Su objetivo es ampliar el conjunto de herramientas disponibles mediante el desarrollo de modelos de inteligencia artificial (IA). Este proyecto ha generado nuevos empleos y ha impulsado el crecimiento económico en instituciones de estudio como la

UPV, contribuyendo así al empleo pleno y productivo y al crecimiento económico sostenible promovidos por el ODS 8.

En sintonía con el Objetivo de Desarrollo Sostenible 9 (ODS 9), que impulsa la industria, la innovación y la infraestructura, este proyecto busca estimular la innovación en diversos ámbitos. Los modelos de inteligencia artificial (IA) desarrollados en el contexto de este proyecto tienen como meta agilizar la investigación y el diseño de nuevos algoritmos en diferentes áreas, como la agricultura y la biodiversidad. De esta forma, el proyecto no solo respalda la innovación tecnológica, sino que también contribuye a fortalecer infraestructuras resilientes y a promover industrias sostenibles, aspectos esenciales del ODS 9.

Este proyecto se alinea con el Objetivo de Desarrollo Sostenible 12 (ODS 12), que promueve la producción y el consumo responsables. Gracias a la implementación de tecnologías en la nube, el proyecto asegura un uso y consumo eficiente de los recursos necesarios, como la energía de máquinas y servidores. Esto no solo reduce el consumo de energía, sino que también minimiza la generación de desechos electrónicos, contribuyendo así a la gestión eficiente de los recursos naturales y a la reducción de la generación de desechos, aspectos clave del ODS 12.