Facultad de Ciencias

Adaptación de portal web para ejecución de trabajos de machine learning en clouds híbridos

(Web portal adaptation for the execution of machine learning jobs in hybrid clouds)

Trabajo de fin de grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autora: Clara Torre García-Barredo

Director: Álvaro López García

Julio - 2020

Agradecimientos

Quiero dedicarle este trabajo a mis padres y a mi hermana. Ha sido un año agotador, lleno de exámenes, trabajos y prácticas, y ellos me han apoyado todo el camino, y sé que seguirán haciéndolo siempre. Gracias por hacerme fotocopias y leeros páginas y páginas de temas que no domináis. Os quiero. También quiero agradecerles el apoyo a mis abuelos, que siempre me dicen que saben que puedo llegar hasta donde yo me proponga.
A mis amigos, dentro y fuera de la Universidad, que me aguantan cuando no me salen las cosas y que me esperan aunque salga de la biblioteca corriendo para poder llegar a verles.
A mi tutor, Álvaro López García, quiero agradecerle que haya dedicado su tiempo a ayudarme con esta tarea, y que haya entendido mis dudas aun cuando yo no estaba segura de cuáles eran.
Ha sido un año raro, histórico, y vivir la experiencia de escribir este trabajo en medio de una vorágine mundial ha sido cuanto menos un apasionante reto; estoy muy orgullosa de haber sido capaz de llegar hasta donde he llegado en estas circunstancias, y ahora solo queda seguir adelante.
Gracias. Disfruten la lectura.

Adaptación de portal web para ejecución de trabajos de machine learning en clouds híbridos

Resumen

palabras clave: machine learning, SQLAlchemy, DEEPaaS, Flask

En la presente memoria se expone el trabajo que se ha realizado para adaptar el portal web DEEP Training Dashboard para la ejecución de trabajos de machine learning en clouds híbridos.

Este portal se compone de un marketplace que permite a los usuarios acceder a diversos módulos de inteligencia artificial y aprendizaje, con el propósito de ejecutarlos con diversos experimentos o entrenarlos con sus propios datos. El portal también tiene un panel en el que el usuario puede subir sus propios módulos para entrenar, y controlar las ejecuciones que ya ha realizado o que están en progreso.

Este proyecto tiene como objetivo mejorar la funcionalidad de este portal ya existente, añadiendo una base de datos y páginas que puedan ayudar al usuario a analizar las ejecuciones de trabajos de machine learning de una manera más exhaustiva.

Para ello, una de las mejoras que se han implementado ha sido añadir la posibilidad de distribuir las ejecuciones de los distintos modelos en experimentos con características similares, a gusto del usuario, para, por ejemplo, realizar pruebas con los mismos datos en modelos distintos.

Al comienzo del proyecto, la aplicación web hacía uso de un servicio existente, llamado DEEP PaaS Orchestrator, para mostrar a los usuarios los datos de sus ejecuciones, pero mostraba una lista demasiado amplia y a bajo nivel, que no es lo ideal para cierto tipo de usuarios. Este trabajo se enfoca en mejorar esa interacción con el usuario para poder mostrarle solamente los datos interesantes en relación con los trabajos de machine learning que se estén realizando.

(Web portal adaptation for the execution of machine learning jobs in hybrid clouds)

Abstract

keywords: machine learning, SQLAlchemy, DEEPaaS, Flask

This paper focuses on the work that has been performed to adapt the web portal DEEP Training Dashboard to the correct execution of machine learning jobs in hybrid clouds.

This portal holds a marketplace that allows the users to access several different Artificial Intelligence and Machine Learning modules, with the purpose of executing them with different experiments or train them with their own data. The portal also has a dashboard in which the user can upload their own modules to train, and control the executions that they have already performed or that are currently in progress.

This project has the goal to enhance the functionality of this already existing portal, adding a database and pages that could help the user analyze the machine learning jobs' executions in a more exhaustive manner.

To this extent, one of the improvements that have been added is to develop the possibility to distribute the executions of the different models in experiments with alike features, to the user's taste, allowing them to try the same datasets in different models, for example.

At the start of the project, the web application adopted an already existing service, called DEEP PaaS Orchestrator, to show the users the data obtained from their executions, but it delivered a list that was too large and too low-level, which is not ideal for a certain kind of user. This paper sums up the work that has been done to improve that interaction with the user, aiming to be able to show them only the interesting data in relation to the machine learning jobs they were running at the moment.

Índice

1 Introducción

DEEP Hybrid Datacloud es un proyecto que pretende ayudar a los desarrolladores de módulos de inteligencia artificial y *machine learning* a experimentar con sus elementos gracias a una estructura *serverless*. Añadir nuevas funcionalidades a las existentes en este proyecto podría aumentar la calidad de la experiencia de usuario.

El presente trabajo, realizado como soporte del Trabajo de Fin de Grado (TFG) para acceder al Grado en Ingeniería Informática, se ha centrado en crear e implementar una serie de funciones que permitan al usuario de la plataforma DEEP Hybrid Datacloud organizar las ejecuciones de sus diversos módulos de inteligencia artificial, favoreciendo un análisis más exhaustivo de los parámetros empleados, así como de los *datasets* probados y de los resultados obtenidos.

Se ha pretendido, además, crear una base de datos que soporte una nueva función: la creación de un historial de ejecuciones que permita al usuario repasar combinaciones anteriores de los diversos parámetros y *datasets* que haya podido utilizar, aun cuando esas ejecuciones se hayan borrado de la plataforma por ser antiguas o de menor interés para el usuario en la actualidad.

Por último, esta nueva base de datos originará también un cambio sustancial en la arquitectura actual de la aplicación; en particular, en el DEEP Training Dashboard, o Panel de Entrenamiento, utilizado por la mayoría de usuarios de DEEP, que se ha pretendido mejorar, objetivo hacia el que se ha enfocado una parte sustancial del esfuerzo de este trabajo.

1.1. El proyecto DEEP

DEEP Hybrid Datacloud es un proyecto que se compone de varias plataformas. Su principal objetivo es preparar una nueva generación de e-infraestructuras para soportar Aprendizaje Profundo (Deep Learning) y otras técnicas de computación intensiva para explotar grandes fuentes de datos [5].

Los principales servicios de los que se compone son:

- DEEP Marketplace.
  - Trainable modules.
  - Pre-trained modules.

- DEEPaaS.
- DEEPaaS API.
- Data storage resources.
- DEEP Training Dashboard.
- General Purpose Dashboard.

1.1.1. DEEP Marketplace

El marketplace es una plataforma a la que todos los usuarios tienen acceso, y donde se listan todos los módulos que hay disponibles para ellos. Los módulos se dividen en "trainable" o entrenables, y "pre-trained" o entrenados, aunque algunos de ellos pueden ubicarse en ambas categorías.

Los **módulos "trainable", o entrenables**, son módulos que se han añadido a la plataforma para que los usuarios puedan entrenarlos con sus propios datos, habitualmente con la intención de crear un nuevo servicio, pero también como experimento o para comprobar las características de sus datos.

Un ejemplo del uso asociado a este módulo podría ser entrenar un módulo clasificador de imágenes con números escritos a mano para crear un identificador de recibos, un traductor de manuscrito a texto procesado, u otro servicio semejante.

Por otro lado, los **módulos "pre-trained", o entrenados**, son módulos que se han creado y entrenado para una tarea específica. Estos módulos ahorran al usuario la tarea de entrenarlos si lo que se busca es resolver una tarea específica, pero están más enfocados hacia un objetivo concreto, y por tanto no son útiles para resolver varios objetivos distintos, circunstancia que sí pueden lograr los modelos entrenables.

Podemos pensar en un clasificador de imágenes al que se ha entrenado con imágenes de perros de distintas razas para crear un clasificador de razas de perro. Puede utilizarse para ese objetivo sin tener que entrenar previamente un clasificador de imágenes general, pero no podría identificar distintos tipos de plantas, por ejemplo.

Los módulos que pueden utilizarse para ambos propósitos son módulos que han sido previamente entrenados, pero con objetivos genéricos. En el caso del ejemplo expuesto anteriormente, sería un analizador de imágenes que ha sido entrenado para clasificar imágenes, pero de ningún tipo en concreto. Aunque pudiera entrenarse más específicamente para crear un clasificador de plantas, o perros, también puede utilizarse para clasificar imágenes generales, sin necesidad de ser entrenado.

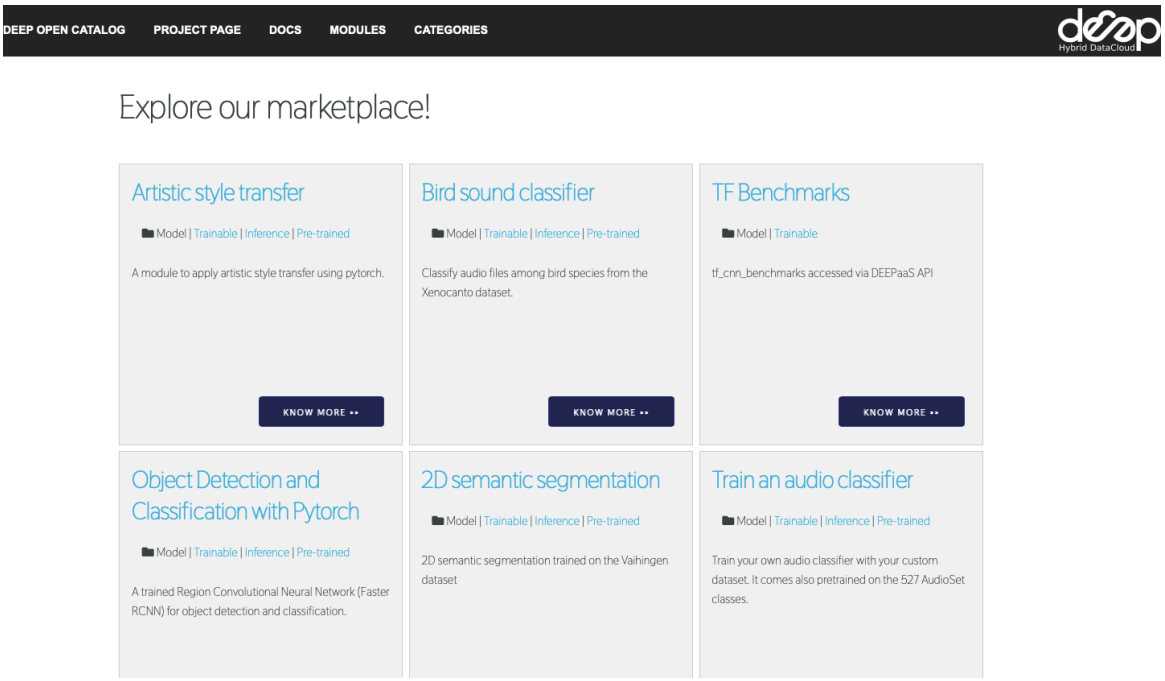


Ilustración 1: DEEP Marketplace.

1.1.2. DEEPaaS

DEEP as a Service (DEEPaaS) es un servicio que permite utilizar aplicaciones ya desarrolladas. Tiene escalabilidad horizontal gracias a que se ha creado desde una perspectiva *serverless*. La perspectiva *serverless* resume las aplicaciones que contratan los servicios de lógica y estado de servidor a empresas externas [9]. El concepto de *serverless* engloba tanto BaaS como FaaS, e incluso es habitual encontrarlos juntos.

Backend as a Service (BaaS) consiste en ofrecer como servicio todo aquello relacionado con la parte del servidor de una aplicación, para que el usuario solamente se ocupe del lado del cliente. De acuerdo con Igor, Costa y col [3], los proveedores de BaaS son un puente entre las aplicaciones y los backends basados en arquitecturas *cloud*, unidos por una API unificada.

Functions as a Service (FaaS) consiste, por otro lado, en ofrecer funciones ya desarrolladas, para que los clientes puedan añadirlas a sus aplicaciones. Estas funciones están pensadas para conectar el lado del cliente con el lado del servidor, que habitualmente está externalizado. Es por ello que habitualmente BaaS (el lado del servidor) y FaaS (la conexión entre el servidor externalizado y el cliente) se contratan conjuntamente.

Según Mike Roberts [9], FaaS trata de ofrecer a los clientes la posibilidad de ejecutar código de servidor sin tener que administrar sus servidores ni sistemas. FaaS también favorece la escalabilidad horizontal, ya que en este caso es completamente automática y es competencia del proveedor de servicios FaaS.

Con DEEPaaS, los dueños de los módulos solamente tienen que preocuparse del proceso del desarrollo de la aplicación. Pueden añadir más contenido una vez que el módulo está en la plataforma, ya que el sistema de automatización puede recibirlo y administrarlo. Solamente se permite probar los módulos en tiempo real; para entrenarlos se debe usar el DEEP Training Dashboard, ya que entrenar los módulos exige un consumo más abundante de recursos.

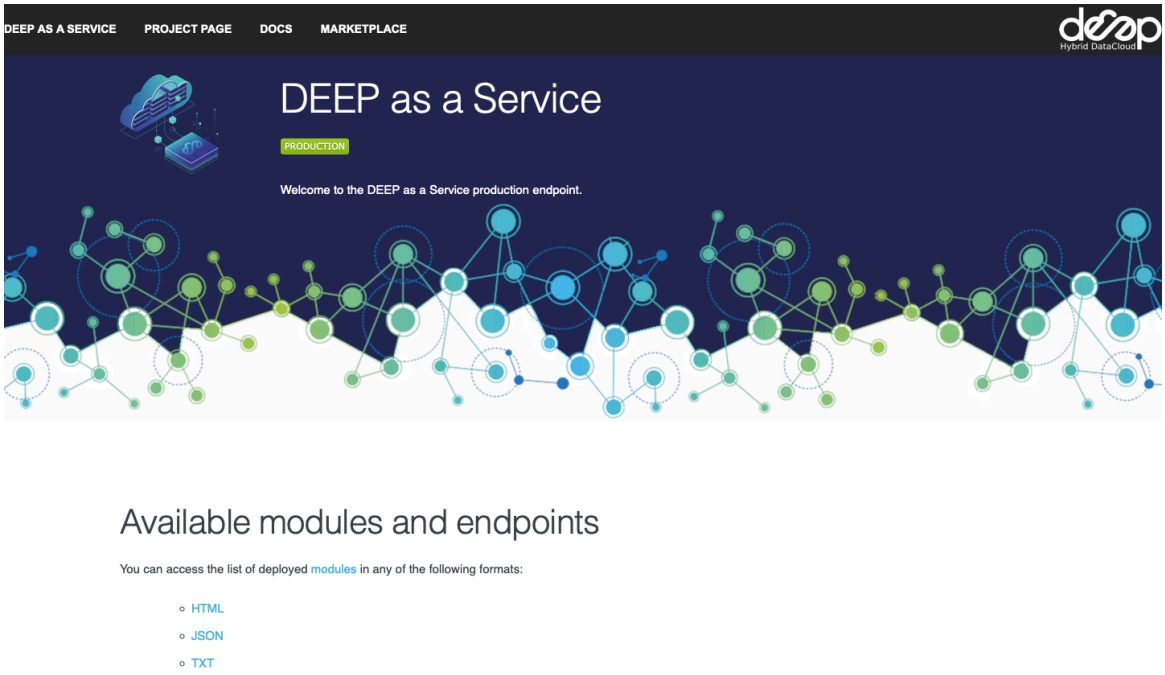


Ilustración 2: DEEPaaS.

Desde la plataforma de DEEPaaS se puede acceder a la lista de los módulos actualmente disponibles en el marketplace en cualquiera de los siguientes formatos:

- HTML.
- JSON.
- TXT.

También se puede acceder a las interfaces Swagger de los módulos. Las interfaces Swagger de los módulos son una ayuda para que el cliente pueda visualizar e interactuar con los recursos recogidos en las APIs sin la necesidad de tener la lógica implementada [10].

1.1.3. DEEPaaS API

La DEEPaaS API está creada para favorecer la interacción del usuario con los módulos disponibles en el marketplace. Está diseñada para facilitar tanto el entrenamiento como la experimentación con módulos ya entrenados, y permitir, para usuarios de perfil más avanzado, orientar la creación de los nuevos módulos para funcionar con la API.

La DEEPaaS API expone una API con la funcionalidad más común entre los modelos de aprendizaje máquina y aprendizaje profundo. Además, la DEEPaaS API ofrece al usuario la posibilidad de desarrollar el modelo siguiendo la arquitectura *serverless* mencionada anteriormente [7].

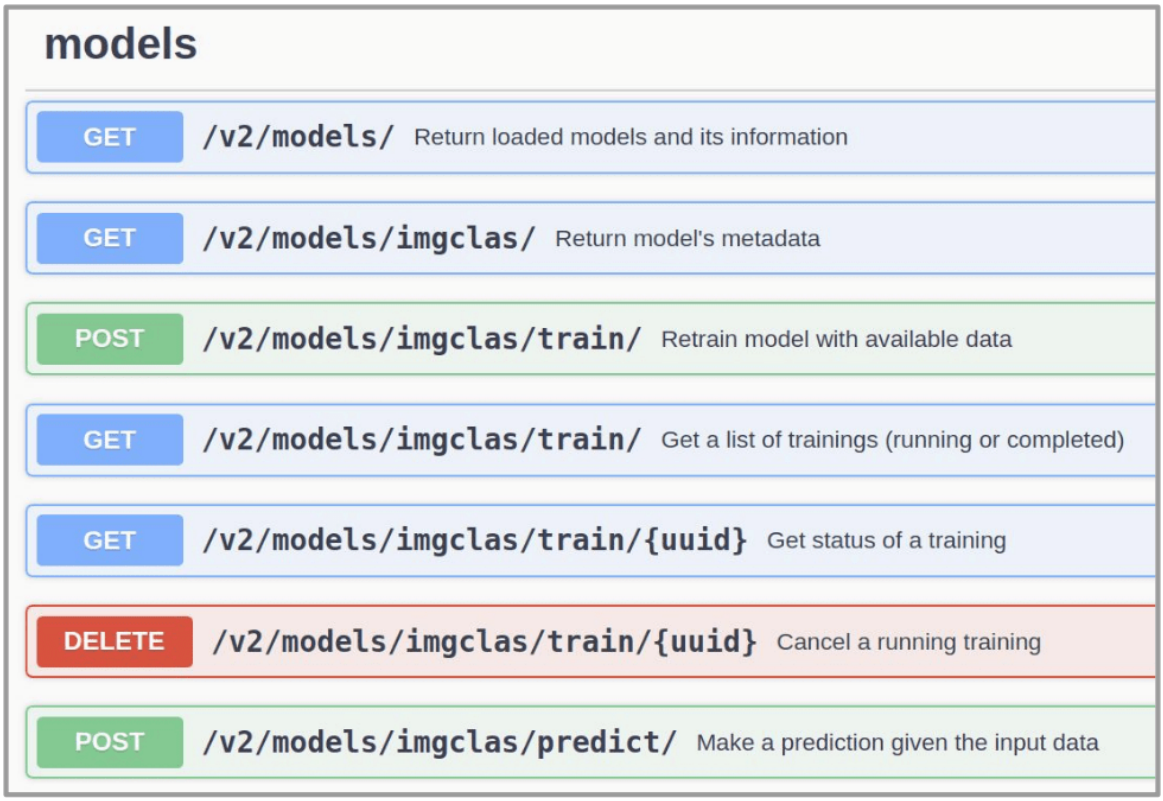


Ilustración 3: DEEPaaS API. Imagen de DEEPaaS Docs [4].

De este modo, los usuarios pueden interactuar con los distintos métodos de la API mediante una interfaz fácil de usar y accesible para todo tipo de usuarios, con independencia de su nivel de conocimiento sobre la materia. Estos métodos están definidos uno a uno en la documentación para que los usuarios más avanzados que quieran crear sus propios módulos sepan cómo adaptarlos para su correcta integración con la plataforma.

1.1.4. Data storage resources

El almacenaje de datos es vital en un proyecto de este tipo. Si los usuarios desean experimentar con un cierto modelo, lo más probable es que prueben a ejecutarlo con diversas variaciones de datos para ver cómo dicho modelo interactúa con ellas, y almacenar los resultados para su posterior análisis.

Por este motivo, al inicio de este proyecto la información se guarda en DEEP-Nextcloud. Uno de los objetivos de este trabajo es añadir una base de datos que guarde los parámetros definidos para cada ejecución, para que los usuarios puedan analizarlos y definir sus experimentos en función de los parámetros de configuración.

1.1.5. DEEP Training Dashboard

El DEEP Training Dashboard, o Panel de Entrenamiento, es el Panel que permite a los usuarios interactuar con los módulos que ya se han subido al marketplace, tanto utilizar los ya entrenados como entrenar módulos para su uso posterior. Es el utilizado por la mayoría de los usuarios de DEEP, y es el servicio en el que se ha enfocado una gran parte del esfuerzo de este trabajo.

Como recoge la documentación [4], este Panel de Entrenamiento también puede utilizarse para ejecutar imágenes externas de DockerHub, aunque este es un uso menos habitual.

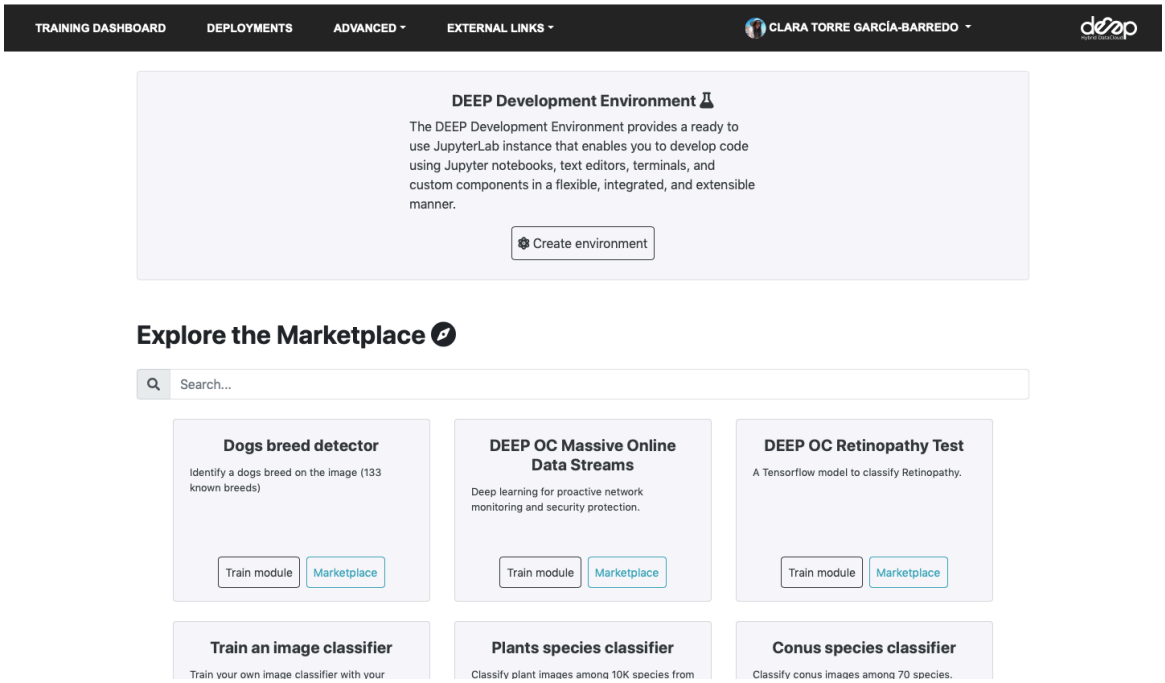


Ilustración 4: DEEP Training Dashboard.

El Panel de Entrenamiento, como muestra la ilustración 4, permite crear entornos de JupyterLab para probar los distintos módulos. Un entorno de JupyterLab es una interfaz de usuario basada en web para Jupyter [8]. Esta interfaz permite al usuario interactuar con sus módulos de una manera más cercana y visual, ofreciendo terminales de Python u otros lenguajes, permitiendo controlar la ejecución y el entrenamiento de los módulos del usuario.

Gracias a ello, se pueden decidir las épocas que se van a usar en el entrenamiento de los módulos, así como ver en tiempo real los resultados que estos entrenamientos parciales van devolviendo. También se puede acceder a los archivos de entrenamiento, ya que JupyterLab es capaz de abrir achivos con multiples formatos, como .JSON, .tex o .HTML, por ejemplo.

1.1.6. General Purpose Dashboard

En lugar de interactuar con los módulos disponibles en el marketplace desde el DEEP Training Dashboard, el General Purpose Dashboard permite al usuario interactuar con los TOSCA Templates que se encuentran por debajo arquitectónicamente de los módulos, y ser más precisos en sus intenciones con los distintos experimentos.

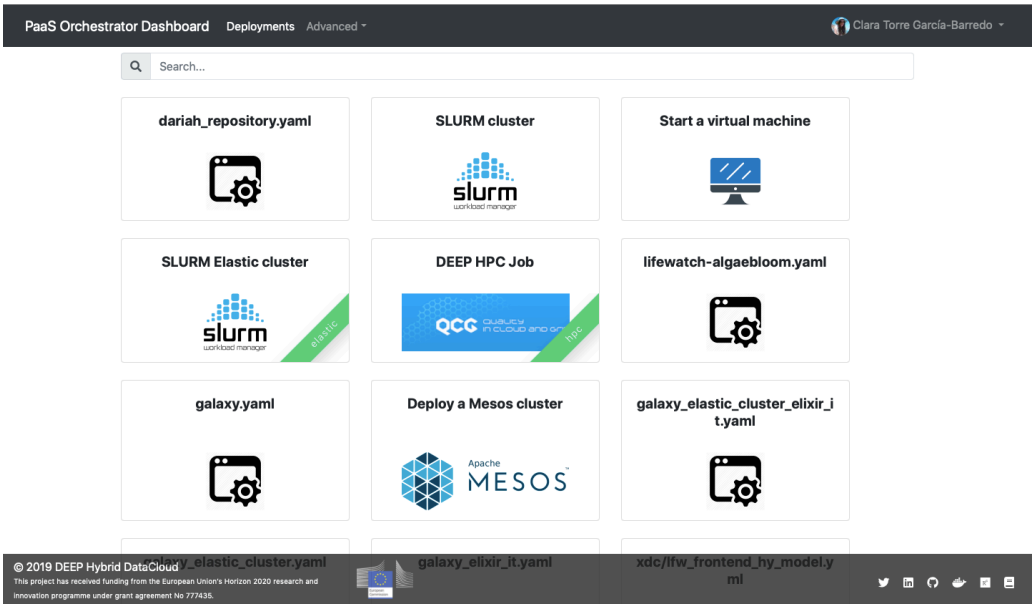


Ilustración 5: General Purpose Dashboard.

«To enable the creation of portable cloud applications and the automation of their deployment and management, the application's components, their relations, and management must be modeled in a portable, standardized, machine-readable format.»[2]

En su capítulo sobre el estándar TOSCA, Binz y col. exponen con claridad que el objetivo de este estándar es permitir crear aplicaciones *cloud* que sean fácilmente exportables y automatizables. Esta circunstancia provoca que las aplicaciones puedan ser utilizadas desde cualquier dispositivo, requiriendo únicamente un acceso *online* a su ubicación en la nube, y para ello, tanto los componentes de la aplicación como sus relaciones y el manejo de cada uno de ellos debe estar sujeto al estándar creado con este propósito, esto es, al estándar TOSCA.

«This is where TOSCA — the Topology and Orchestration Specification for Cloud Applications — proposes an XML-based modeling language tackling these issues by formalizing the application's structure as typed topology graph and capturing the management tasks in plans.»[2]

Como recoge la cita anterior, el estándar TOSCA, que recibe sus siglas de Especificación de la Topología y Orquestación para Aplicaciones en la Nube en sus términos en inglés, propone un lenguaje basado en XML que permite formatear la estructura de las aplicaciones como un gráfico topológico, ayudando a su homogeneización.

Esta propuesta permite la funcionalidad principal del General Purpose Dashboard, que se basa en la posibilidad de utilizar *templates* o plantillas para crear las aplicaciones *cloud* de una manera mucho más específica y detallada que lo que la creación de módulos en el DEEP Training Dashboard permite.

Los usuarios que se decidan por este Panel en lugar del Panel de Entrenamiento deberán decidir si desean utilizar una plantilla general para desarrollar su aplicación, o si por el contrario desean adaptar una de las plantillas específicas que se encuentran entre las proporcionadas por la plataforma para crear una plantilla personalizada para su proyecto.

A partir del análisis y estudio de los elementos de partida expuestos, de su funcionamiento y de la integración de todos ellos dentro del portal web DEEP Training Dashboard, el objetivo del trabajo a desarrollar ha consistido en realizar una ampliación de la funcionalidad de dicho portal, que permita a los usuarios la ejecución de módulos de *machine learning* e inteligencia artificial en *clouds* híbridos. Dicho objetivo, no exento de dificultad, se ha orientado fundamentalmente hacia la mejora de la funcionalidad del portal ya existente, para ayudar a los diferentes usuarios a analizar las ejecuciones de *machine learning* de una manera más exhaustiva. Indudablemente, dado el objetivo del *machine learning*, resulta esencial un seguimiento y control de los aprendizajes, por un lado, dados los volúmenes de datos a manejar; por otro, dado que del aprendizaje, a partir de la revisión de los datos, se desprende la posibilidad de predecir comportamientos futuros; y finalmente, porque en este contexto los sistemas se mejoran de forma autónoma, a medida que desarrollan iterativamente su capacidad de autoaprendizaje. En este entorno, con la necesidad de tratamiento de datos masivos, resulta necesaria, por la capacidad de espacio requerida, la utilización de estas aplicaciones en entornos *clouds* híbridos. Pues bien, ese es el objetivo final del trabajo. Permitir a los usuarios un acceso amigable a la realización de ejecuciones de *machine learning*, ofreciéndoles diferentes alternativas de ejecución e incorporando opciones que mejoren la interacción con el usuario.

1.2. Estructura de la memoria

Este trabajo ha sido estructurado en seis capítulos. Se introduce brevemente, a continuación, cada uno de ellos:

1. Introducción. En esta sección se explica la aplicación de partida, cuya funcionalidad se pretende mejorar, sus distintos servicios y los elementos que la componen.
2. Recursos y metodología. En este capítulo se expone la información sobre los distintos recursos empleados en la realización de este trabajo, así como el diseño metodológico utilizado.
3. Requisitos. Este capítulo recoge las *user stories* y detalla los requerimientos, tanto funcionales como no funcionales.
4. Diseño de la aplicación. Se describen las soluciones aportadas, y se presentan los nuevos diseños de la aplicación. Así mismo, se detalla el cambio de arquitectura de la misma.
5. Implementación y pruebas. En este capítulo se describe el proceso de creación de la aplicación y las pruebas realizadas, desarrollando las diversas mejoras que se deseaba conseguir con este proyecto y comprobando el correcto funcionamiento de todas ellas en todos los escenarios.
6. Conclusiones y trabajo futuro. Se proporcionan las conclusiones extraídas del trabajo y se proponen posibles acciones futuras.

2 Recursos y metodología

En este capítulo se describen los recursos tecnológicos que se han empleado para llevar a cabo este proyecto.

También se describe la metodología empleada en el desarrollo del proyecto, comenzando por la revisión del material proporcionado con la aplicación inicial y continuando con el análisis del proyecto que se quiere presentar a través de esta memoria.

2.1. Recursos empleados

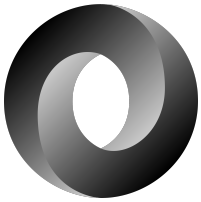
En este proyecto se han utilizado diversos recursos que han sido escogidos para contribuir al desarrollo de las mejoras que se deseaba introducir en la aplicación incial. La elección de algunos de ellos estaba condicionada por el proyecto de partida. Otros se han elegido específicamente para poder implementar las mejoras requeridas.

2.1.1. Lenguajes

Python es el lenguaje de programación predominante en este proyecto, que se caracteriza por su facilidad de manejo y su versatilidad a la hora de coordinarse con otros lenguajes o entornos. Python era también el lenguaje predominante en la aplicación con la que se comenzó el proyecto, y se decidió mantenerlo. Otra de las razones de elección de este lenguaje está asociada a un motivo personal, dado que, habiendo cursado la mención de Computación, este ha sido uno de los lenguajes más utilizados durante el tercer y cuarto curso.

HTML es el lenguaje de marcado que se ha utilizado para estructurar todas las páginas web empleadas para dar forma a la aplicación. Está integrado con las instrucciones de Python para poder conseguir que las páginas interactúen entre sí, y dotar de funcionalidad a la plataforma. Se ha escogido por ser el lenguaje de la mayoría de páginas web, por su facilidad de integración con los frameworks usados en la aplicación, y porque es el lenguaje de creación de las páginas web ya existentes en la aplicación.

2.1.2. Formatos



JSON es el formato de archivo, aparte de los correspondientes a los diversos lenguajes de programación, que se ha escogido para contener información importante a la que se accede desde diversos puntos de la plataforma. Se han utilizado varios archivos de formato .JSON para crear diversos ambientes para las pruebas de funcionamiento, uno local con el cliente y las direcciones locales, y otro de tipo Docker con las direcciones dentro del contenedor y otros recursos necesarios para esta aplicación.

2.1.3. Frameworks

Flask es el framework utilizado a lo largo de toda la plataforma para crear la aplicación, integrando tanto las plantillas de páginas en HTML como la funcionalidad desarrollada en Python. Este framework, empleado para crear la aplicación inicial, se ha escogido porque cuenta con una de las mayores comunidades de soporte, y por su facilidad de integración con otros sistemas y porque favorece la rapidez de respuesta de la plataforma. Se integra con Python a la perfección, y tiene muchas extensiones que ayudan a crear la funcionalidad deseada para la aplicación.

Bootstrap es el framework que se ha utilizado para aportar el diseño y la apariencia de las distintas páginas web que componen la plataforma. Es un framework con el que resulta cómodo trabajar, ya que contiene muchos elementos homogéneos en diseño entre sí que ayudan a elevar la simplicidad de la aplicación, y aumentar el nivel de usabilidad a nivel de cliente, permitiendo a usuarios de distintos niveles de conocimiento interactuar con la plataforma sin dificultad. Era el escogido por el equipo anterior.

2.1.4. APIs

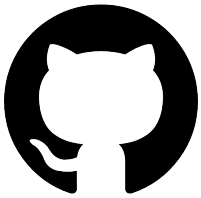
DEEPaaS Orchestrator es una RestAPI diseñada por DEEP Hybrid Datacloud para ayudar con el desarrollo de esta plataforma. Colabora en el acceso a ejecuciones de aplicaciones de inteligencia artificial, permitiendo que la plataforma realice llamadas HTTPS para acceder a la API y mostrar al usuario los resultados solicitados. Permite hacer operaciones de GET, POST y PUT, interaccionando con los módulos disponibles en la plataforma y requeridos por el usuario, para ofrecer los parámetros pasados, los resultados parciales de los entrenamientos o el resultado final.

2.1.5. Toolkits

SQLAlchemy es un "toolkit" o grupo de herramientas diseñadas para integrarse con Python y una base de datos SQL. Permite que la base de datos y la aplicación se desarrollen de forma paralela, pero separada, y se unan en aquellas funciones en las que el desarrollador lo desea únicamente. La construcción de las "queries" está basada en las funciones creadas en Python, permitiendo que la base de datos se pueda ir creando y ampliando desde el propio código de la aplicación si fuese necesario.

2.1.6. Aplicaciones

Docker es una aplicación que permite crear contenedores en los que almacenar entornos de programación completos, para poder exportarlos y ejecutarlos desde cualquier dispositivo, sin necesidad de ser reconfigurados. A lo largo de este proyecto, esta aplicación ha sido especialmente útil a la hora de realizar diversas pruebas, tanto para comprender la plataforma antes de comenzar las mejoras, como para probar lo implementado a medida que avanzaba el desarrollo del proyecto.



GitHub es una aplicación de control de versiones que permite mantener un registro de los distintos cambios que se van realizando en la aplicación para poder volver atrás en caso de errores, o para poder colaborar con otras personas en el mismo proyecto. Esta aplicación se ha utilizado exhaustivamente a lo largo del trabajo para poder llevar un control de los cambios que se iban realizando, y además, porque la aplicación original estaba almacenada en un repositorio de GitHub.

2.1.7. Entornos

Visual Studio Code es el entorno de programación que se ha utilizado para desarrollar toda esta aplicación. Es un editor de código que tiene integradas varias extensiones para ayudar al desarrollador en la programación de su aplicación, enlazándose con facilidad con todas las tecnologías mencionadas anteriormente, además de permitir llevar el control de versiones de GitHub dentro del entorno, así como la ejecución de los distintos contenedores de Docker empleados.

2.2. Metodología

El método utilizado a la hora de afrontar este trabajo ha sido uno enfocado en la sencillez, para poder seguir todos los pasos de manera adecuada. En concreto, se compone de las cuatro siguientes fases:

1. Analizar y entender la plataforma inicial.
2. Estudiar y aprender las tecnologías nuevas.
3. Desarrollar las mejoras e incorporarlas a la plataforma.
4. Testear los cambios añadidos.

Cada uno de estos cuatro pasos generales contiene a su vez especificaciones necesarias para el desarrollo de la aplicación, que se explican en detalle en los siguientes subapartados.

2.2.1. Analizar y entender la plataforma inicial

Como el proyecto no consistía en empezar una aplicación de cero, sino en añadir mejoras a una ya existente, el primer paso era analizar con detalle cómo se había desarrollado esa plataforma, y cómo funcionaba, para poder integrar el código que se generara de la manera más natural posible, minimizando el número de errores o *bugs* que se pudieran generar a raíz de esa unión de funcionalidades.

El acceso a la aplicación fue proporcionado mediante un repositorio ubicado en la aplicación GitHub. Este repositorio se conectó al espacio de trabajo de Visual Studio Code, clonándose para poder acceder a sus contenidos de forma local, y probar las diferentes páginas que ofrecía.

Igualmente, se empleó la extensión Docker en el entorno de VSCode, para poder acceder a los contenedores ya creados para la aplicación y ejecutarla en un *environment* configurado previamente por los desarrolladores de la plataforma original. Se editaron los archivos de configuración .JSON para poder ejecutar el contenedor correspondiente en local y con el cliente configurado con los permisos necesarios, y se ejecutó la aplicación en el contenedor para probarla y ver cómo funcionaba exactamente.

2.2.2. Estudiar y aprender las tecnologías nuevas

Después de comprender el objetivo de la plataforma y su funcionalidad, el siguiente paso fue dominar las tecnologías involucradas. Hay dos en concreto que requerían de un estudio en profundidad, por no estar entre las utilizadas en el grado: Flask y SQLAlchemy. El proyecto también sienta sus bases sobre una RestAPI diseñada específicamente para él, que también requería un análisis dedicado.

Flask y SQLAlchemy juegan un papel muy importante en este proyecto, creando la arquitectura de la plataforma tanto desde el lado del cliente como desde la base de datos que se desea añadir.

La aproximación a Flask se basó en el estudio detallado del libro "Flask Web Development", de Grinberg [6]. A lo largo del manual, el autor va introduciendo nuevas características de Flask mientras crea una aplicación desde cero (disponible para entender los ejemplos en un repositorio público de GitHub), explicando al lector cómo integrar los nuevos elementos para añadir funcionalidad paso a paso. Este libro ayudó a comprender cómo funcionaba la aplicación de la que partía el proyecto, además de proporcionar los conocimientos necesarios para ampliarla a lo largo del trabajo.

Para aprender SQLAlchemy, por otro lado, se ha empleado la documentación ofrecida por la página oficial del recurso [1], así como diversos tutoriales de Internet que explicaban cómo utilizar elementos concretos del paquete. Acceder directamente a la documentación oficial es una buena forma de afrontar el uso de cualquier paquete nuevo, ya que permite comprender el uso de funciones específicas para el proyecto en curso sin la opinión de otra persona que haya analizado el paquete anteriormente y pueda haber escogido otra función que no sea la adecuada en otro caso.

En cuanto a la RestAPI Orchestrator, se decidió comprobar cómo funcionaba después de leer el artículo asociado a ella [7], que explicaba la intención tras su creación y su funcionamiento básico. Se utilizó el entorno ya creado para estudiar la aplicación inicial y se probaron las distintas llamadas que ofrece la API para comprobar su alcance y su interacción con el resto de elementos de la plataforma.

2.2.3. Desarrollar e incorporar las mejoras definidas

Una vez que se había comprendido el funcionamiento tanto de la aplicación inicial como de las tecnologías que se iban a emplear, era momento de empezar a desarrollar las mejoras. Para ello, se siguió el orden lógico y habitual de los proyectos de software:

1. Recoger los requisitos del cliente.
2. Realizar los diseños pertinentes.
3. Programar las mejoras.

Esta es la parte más importante de la metodología, y sus diversos pasos están descritos con mayor detalle en los siguientes tres capítulos: Capítulo 3. Requisitos, Capítulo 4. Diseño de la aplicación, y Capítulo 5. Implementación y pruebas. Por este motivo, este apartado solamente recoge un breve resumen del método empleado para ello.

Los requisitos se obtuvieron a lo largo de varias conversaciones con Álvaro López, el director de este proyecto, y de las *user stories* desarrolladas para el mismo, adjuntas en el capítulo 3. Una vez obtenidos los requisitos de sistema, el siguiente paso era afrontar el diseño de las mejoras.

Por un lado, se deseaba añadir una base de datos, y por otro lado, reorganizar los *deployments* en distintos experimentos. Para ello era necesario crear varias páginas nuevas, que pudieran dar a los usuarios acceso a las nuevas funciones que se pensaba implementar. Por ello, se diseñaron varios *mock-ups* de las páginas para comprobar el orden óptimo de los elementos en las diversas páginas, y la interacción de esos elementos entre sí. Estos diseños pueden verse en el Capítulo 4.

Una vez obtenidos los diseños definitivos, el paso restante era desarrollar esos diseños para añadirlos a la plataforma. El código se escribió en Python y HTML, empleando elementos de Flask y de Bootstrap para dar forma a los diversos elementos. Una explicación más detallada de esta parte del proyecto puede verse en el Capítulo 5.

2.2.4. Testear los cambios añadidos

El testeo de estas mejoras se ha realizado no solo comprobando que funcionasen de forma correcta, sino comprobando, además, que los errores previstos se presentaran en la forma correcta, y no hubiese ningún caso sin contemplar que pudiese generar errores imprevistos.

Al ser una aplicación web, estas comprobaciones se han llevado a cabo de manera manual, recreando la experiencia del usuario a través de las distintas páginas disponibles para comprobar que se pudiera acceder sin problemas a todas las funciones que se habían añadido, y que las acciones sin permisos o erróneas muestren su correcta página de error.

Esto se trata de manera más exhaustiva en el Capítulo 5. Implementación y pruebas.

3 Requisitos

Los requisitos son textos o frases que definen las necesidades que tiene el cliente del producto a desarrollar, y que desea que ese producto pueda satisfacer. También pueden incluir información sobre cómo quiere el cliente que sea el programa o producto final, sobre su aspecto o su funcionamiento.

Los requisitos son importantes porque ayudan a visualizar la idea que se tiene del producto final, y también definen qué se considera como tal, y qué es insuficiente. Las preguntas importantes a la hora de afrontar la recolección de requerimientos son dos: "¿Qué?" y "¿Cómo?". ¿Qué quiere el cliente que tenga la aplicación a desarrollar? ¿Cómo quiere que sea su apariencia? ¿Cómo quiere que funcione? Cuanto más específicos sean estos requerimientos, mejor trabajo podrá realizar el ingeniero que diseñe la aplicación.

En este caso, los requerimientos se han obtenido, en primer lugar, de las *user stories* (historias de usuario) proporcionadas por el director del proyecto: peticiones cortas y concisas, contadas desde el punto de vista del usuario; en segundo lugar, de las sesiones de toma de requisitos, tratando de averiguar qué quería exactamente de este trabajo y cómo podía conseguirse. Estos requerimientos están recogidos más adelante, en el segundo apartado de este capítulo.

3.1. User stories

A continuación se expone la lista de *user stories* proporcionadas por el director del proyecto.

«Los usuarios del portal requieren del mismo la siguiente funcionalidad:

- »**US1** El usuario quiere poder enviar las aplicaciones existentes en el catálogo de aplicaciones, o ser capaces de entrenar una aplicación propia. En cualquier caso, solo quiere preocuparse de los parámetros de su aplicación, y no de otros parámetros dependientes de la infraestructura.

- »**US2** El usuario quiere poder interactuar con las aplicaciones desplegadas desde el propio portal, sin necesidad de tener que ir a una nueva URL o página web, de forma que no haya que salir de él. Quiere saber desde el propio portal con que parámetros se ha realizado el entrenamiento.

- »**US3** El usuario quiere poder agrupar los entrenamientos en experimentos. Un experimento puede consistir en un mismo modulo entrenado diferentes veces con diferentes parámetros, o bien diferentes módulos entrenados con el mismo dataset, o bien diferentes módulos entrenados con diferentes datasets. Asimismo, dentro de un experimento, se quiere poder visualizar los resultados de diferentes entrenamientos, y poder compararlos. Además se quiere poder reenviar un entrenamiento ya existente, de forma que se pueda comprobar que los resultados son los correctos.

»»

3.2. Requerimientos

A continuación se expone la lista de requerimientos obtenidos, divididos en dos grupos.

3.2.1. Requerimientos funcionales

Los requerimientos funcionales son aquellos que definen la funcionalidad del software; es decir, qué tiene que hacer la aplicación para satisfacer las necesidades del cliente.

Para este proyecto se han recogido los siguientes requisitos funcionales:

- **RS1** Se cambiará la arquitectura para incluir una base de datos que guardará la información sobre los parámetros que cada usuario pasa a cada *deployment* que crea en su perfil, con independencia de si ese *deployment* se borra más adelante.
- **RS2** Se podrán agrupar los *deployments* en experimentos definidos por el usuario.
- **RS3** Los experimentos podrán crearse de antemano, o al mismo tiempo que se crea un nuevo *deployment*.
- **RS4** Los *deployments* podrán seguirse creando sin ser asignados a ningún experimento concreto.
- **RS5** Podrá accederse a la información de la base de datos desde una página nueva que recoja una historia o un *log* de los *deployments* hechos por el usuario.
- **RS6** Para acceder a las nuevas pantallas, el usuario debe estar correctamente registrado en la aplicación, como sucede con el resto de pantallas del programa original.
- **RS7** Siempre que se pueda, las distintas funcionalidades de la aplicación deben mantener al usuario en la misma, no llevarle a otras URLs que ofrezcan ese servicio.
- **RS8** No debe ser obligatorio para el usuario rellenar los parámetros de configuración dependientes de la infraestructura, pero sí debe ser una posibilidad para aquellos usuarios avanzados que quieran hacer uso de ellos.

3.2.2. Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que definen cómo deben comportarse las características de la aplicación en los entornos en los que se usa, y ante qué circunstancias o bajo qué límites el programa debe comportarse de una forma u otra. Estos requerimientos responden a la pregunta de cómo tiene que funcionar la aplicación para satisfacer al cliente.

Los requisitos no funcionales de esta aplicación son los siguientes:

- **RNS1** La aplicación mantendrá la cohesión estilística, en sus nuevas funcionalidades, con la que tenía en las características originales.
- **RNS2** La aplicación será fácil de usar, independientemente del nivel de conocimiento del usuario.
- **RNS3** La aplicación será accesible a todo aquel usuario que lo requiera, independientemente del medio.
- **RNS4** Para mantener el estilo del resto del programa, las mejoras añadidas deben tener un diseño simple y hacer uso de las herramientas ya desarrolladas.

Cuadro 1: Requisitos de la aplicación

R-ID	**Descripción**	**US-ID**
RS1	Añadir una base de datos	US2.
RS2	Agrupar *deployments* en experimentos	US3.
RS3	Crear experimentos antes o durante un *deployment*	US3.
RS4	Crear *deployments* sin experimento	US3.
RS5	Historial de *deployments*	US2.
RS6	El usuario debe estar registrado	US1.
RS7	Mantener al usuario en la aplicación	US2.
RS8	Ocultar parámetros de infraestructura	US1.
RNS1	Cohesión estilística con la aplicación original	US2.
RNS2	La aplicación será fácil de usar	US1.
RNS3	La aplicación será accesible	US1.
RNS4	Las mejoras deben tener un diseño simple	US3.

4 Diseño de la aplicación

Una vez que se han obtenido los requerimientos, es hora de pasar al diseño de la aplicación, o, en este caso, al diseño de las mejoras que se van a añadir a la aplicación ya existente. Para este proyecto es muy importante tener en cuenta el diseño que ya existe dentro de la plataforma, para que las nuevas funciones añadidas mantengan la coherencia estilística necesaria con el resto de las páginas de la aplicación.

Para ello, la decisión lógica es implementar las nuevas funciones haciendo uso de las tecnologías que se han empleado para crear el resto de páginas de la plataforma. En este caso, esa tecnología es principalmente Bootstrap, un framework que aporta un diseño minimalista a cualquier página que lo emplee, adaptándose a usuarios de todos los niveles de conocimiento.

Uno de los requerimientos implica modificar la arquitectura de la aplicación para añadir una base de datos, y páginas que permitan consultar la información en dicha base de datos desde la aplicación. Esta es la razón que ha motivado añadir una sección en la que se explica la diferencia entre la arquitectura inicial y la nueva arquitectura en este capítulo, pues la composición de la arquitectura es una parte importante del proceso de diseño de una aplicación.

4.1. Bocetos de las nuevas implementaciones

El primer paso para diseñar cualquier página es crear un boceto de cómo se espera que sea su apariencia. En este caso, se han realizado bocetos de todas las páginas que se esperaba añadir a la plataforma, y a continuación se expone cada uno de ellos.

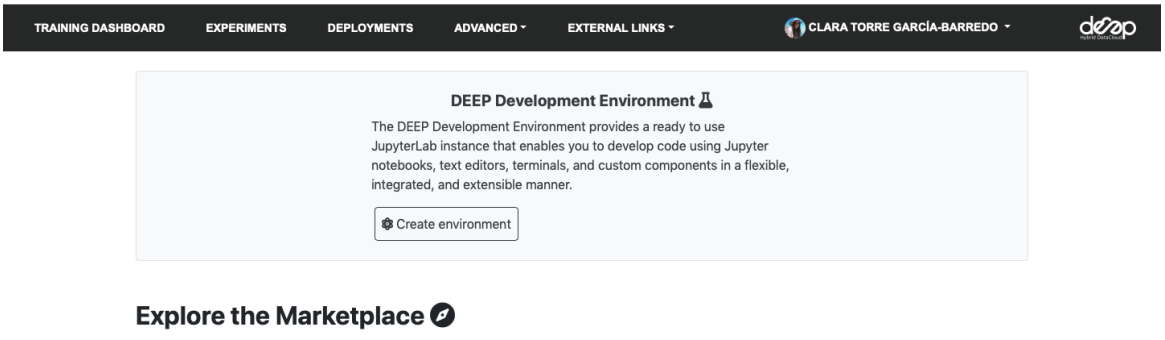


Ilustración 6: Boceto de la navbar.

El boceto que se muestra en la ilustración 6 incluye únicamente un cambio sobre el diseño actual, y es la incorporación de una nueva pestaña en el menú para los Experimentos, una de las mejoras deseadas para la aplicación. Esa nueva pestaña redirige a la ruta /experiments, donde el usuario puede ver sus *deployments* divididos en experimentos.

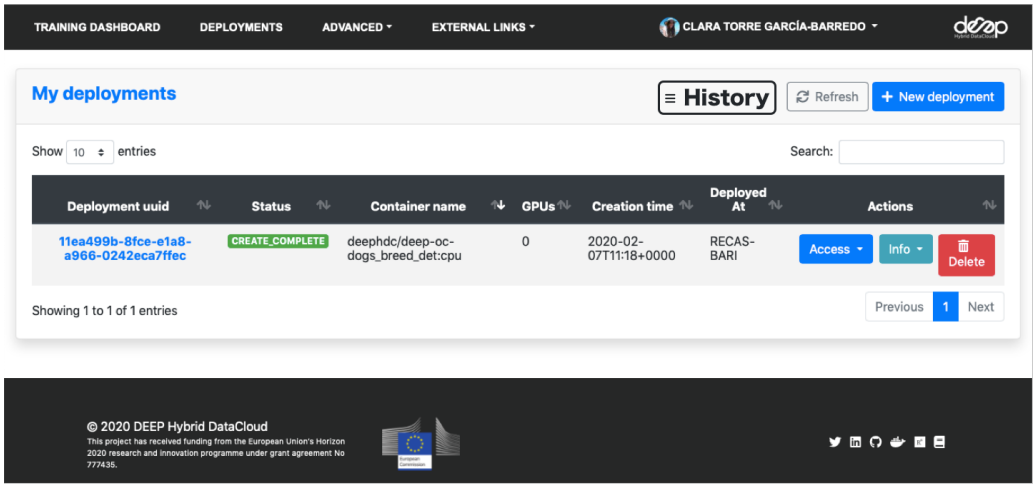


Ilustración 7: Boceto del log history.

El boceto que se muestra en la ilustración 7 también incluye únicamente un cambio en el diseño actual, la incorporación de un nuevo botón en la barra superior de los *deployments* para poder acceder a la historia o *log* de todos los *deployments* que haya hecho el usuario, con independencia de si estos han sido borrados y ya no aparecen en la página principal. Este botón abrirá una página nueva.

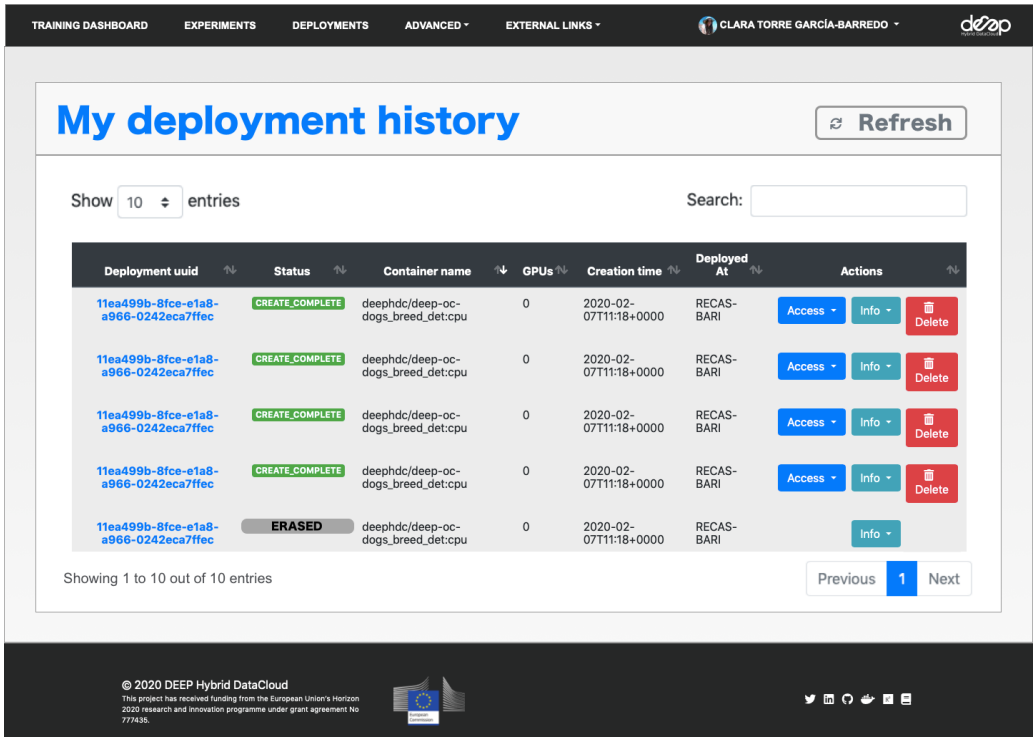


Ilustración 8: Boceto del deployment history.

El boceto que se muestra en la ilustración 8 incluye una página completamente nueva, que permite a los usuarios acceder a la información guardada en la base de datos sobre los *deployments* que han realizado anteriormente, con independencia de si han sido borrados más tarde o no. A aquellos que no han sido borrados podría accederse desde aquí, del resto solamente se tienen los registros.

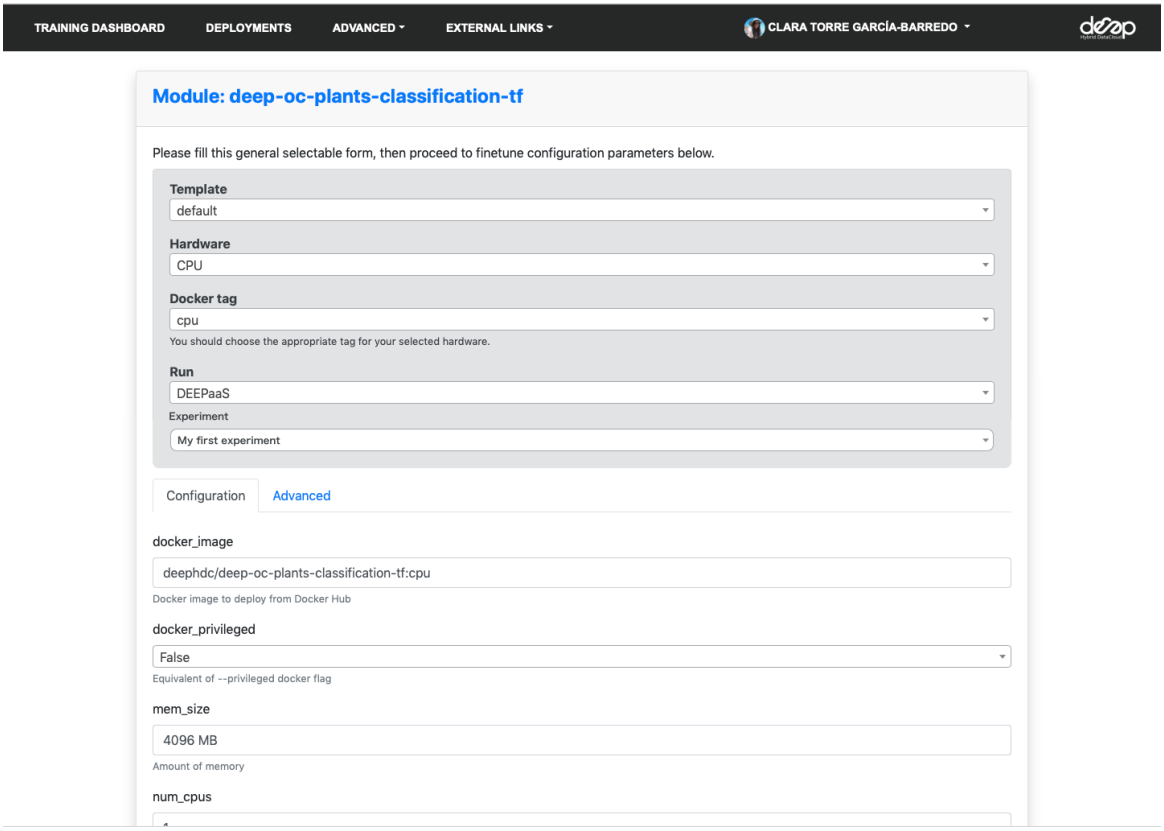


Ilustración 9: Boceto de crear un nuevo deployment.

El boceto que se muestra en la ilustración 9 incluye una modificación del formulario para crear un nuevo *deployment*; se añade un nuevo campo en formato *dropdown menu* que permite escoger el experimento al que se quiere asociar ese nuevo *deployment* que se está creando.

Este nuevo campo se añade al formulario para que, al crear el nuevo *deployment*, este se añada directamente al experimento que se ha escogido, y se pueda acceder a él desde la página de experimentos que se quería crear con estas mejoras. Si el *deployment* se crea sin escoger un experimento en ese momento, se podrá seguir accediendo a él desde la página de Deployments, que era una funcionalidad incluida en la página inicial y que no se ha modificado.

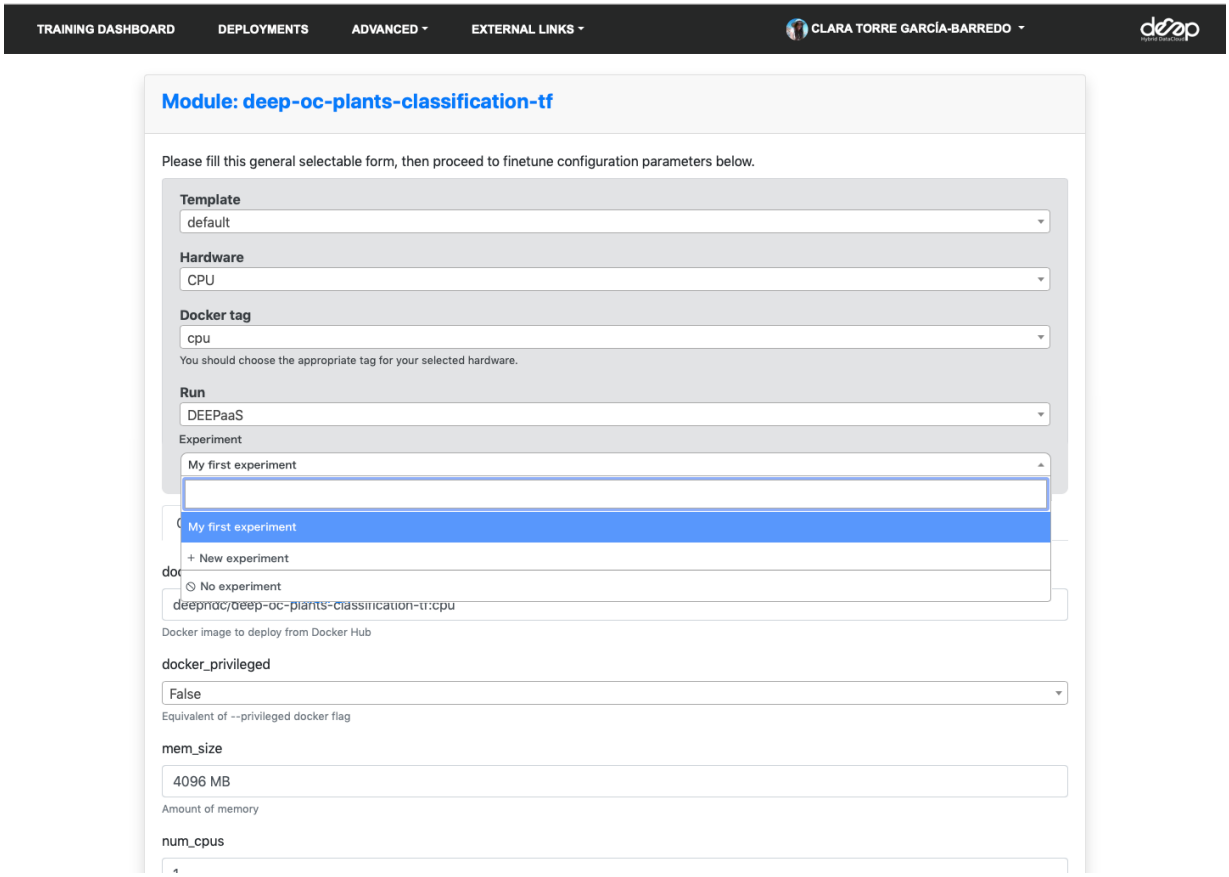


Ilustración 10: Boceto de crear un nuevo deployment.

La ilustración 10 muestra la misma pantalla anterior con el nuevo campo desarrollado en formato *dropdown menu* desplegado. De esta forma, se puede ver el menú desarrollado con las diversas opciones para escoger entre experimentos ya creados, un nuevo experimento, o que el *deployment* se cree sin estar asociado a ningún experimento, con la posibilidad de cambiar esa circunstancia más adelante.

En el ejemplo del boceto solamente puede verse un experimento creado, pero en la implementación aparecerían todos los distintos experimentos que haya creado el usuario para poder escoger entre ellos directamente. Si se escoge la opción de "New Experiment", se creará automáticamente un nuevo experimento con ese nombre y el usuario podrá editarlo más adelante en la sección de Experiments del DEEP Training Dashboard, accesible desde la navbar, como se ha visto en anteriores bocetos.

Los parámetros que se ven en este formulario son los que posteriormente se podrán consultar en el log history que se ha enseñado anteriormente, incluyendo el experimento en el que este *deployment* en concreto se organizó, además del resto de parámetros que ya se podían rellenar en el formulario de la aplicación original.

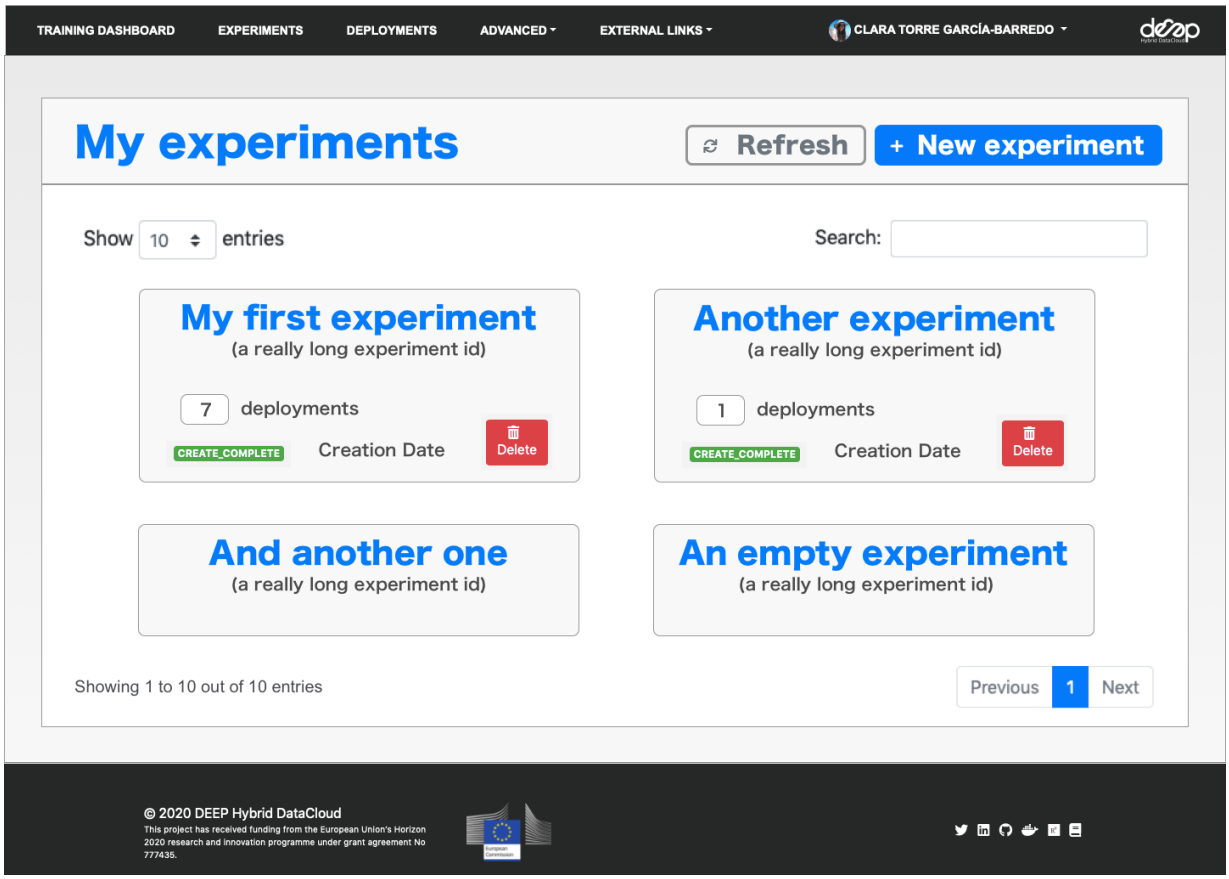


Ilustración 11: Boceto de Experiments.

El boceto que se muestra en la ilustración 11 incluye una página completamente nueva, representando una de las mejoras más importantes de este proyecto: la posibilidad de organizar los distintos *deployments* creados por el usuario en varios experimentos. Esta distinción permite al usuario controlar la creciente lista de *deployments* que pueda tener y comparar el comportamiento de distintos parámetros o módulos entre sí.

Con el diseño que se presenta se ha deseado reflejar la distinción entre los distintos experimentos de manera gráfica y visual, dividiéndolos en distintas tarjetas sobre las que se puede pinchar para acceder a la vista individualizada de cada experimento por separado.

En todo caso, sin necesidad de acceder a la vista individualizada del experimento se puede saber cuántos *deployments* contiene actualmente, además de saber cuándo fue creado y el estado de ese experimento en concreto. También puede ser eliminado desde esta vista.

Para crear un nuevo experimento sin tener que hacerlo a la vez que un nuevo *deployment* se puede utilizar el botón “New Experiment” que se encuentra en la parte superior derecha, junto a un botón para refrescar la vista y poder consultar los cambios que se hayan podido producir.

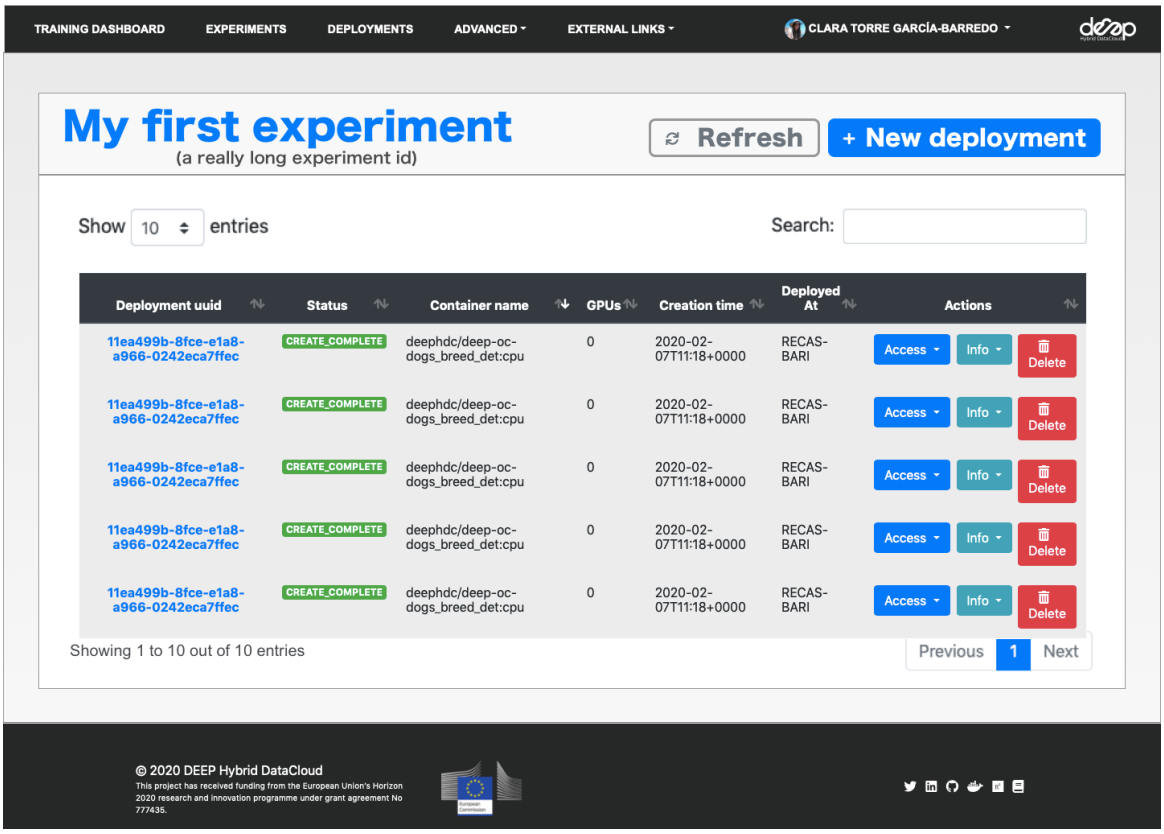


Ilustración 12: Boceto de My first Experiment.

El boceto mostrado en la ilustración 12 incluye otra página completamente nueva diseñada para contener información individualizada de cada experimento creado por el usuario. Contiene el id del experimento, así como todos los *deployments* que han sido asociados a él y sus características.

La página está diseñada en forma de lista para imitar lo más posible a la estructura de la vista de Deployments que ya existía en la aplicación original, con la intención de que el usuario encuentre la nueva página lo más cómoda posible, y pueda acceder de forma intuitiva a la nueva información proporcionada por la página.

4.2. Nueva arquitectura

Añadir una base de datos cambia por completo la arquitectura de cualquier aplicación, y en este proyecto también. Inicialmente, esta plataforma tenía una arquitectura del tipo Modelo-Vista-Controlador (arquitectura MVC), y al añadir la base de datos la arquitectura ha variado, pasando a ser de tipo Arquitectura por Capas. A continuación se explicará cómo son ambas arquitecturas y cómo afecta el cambio.

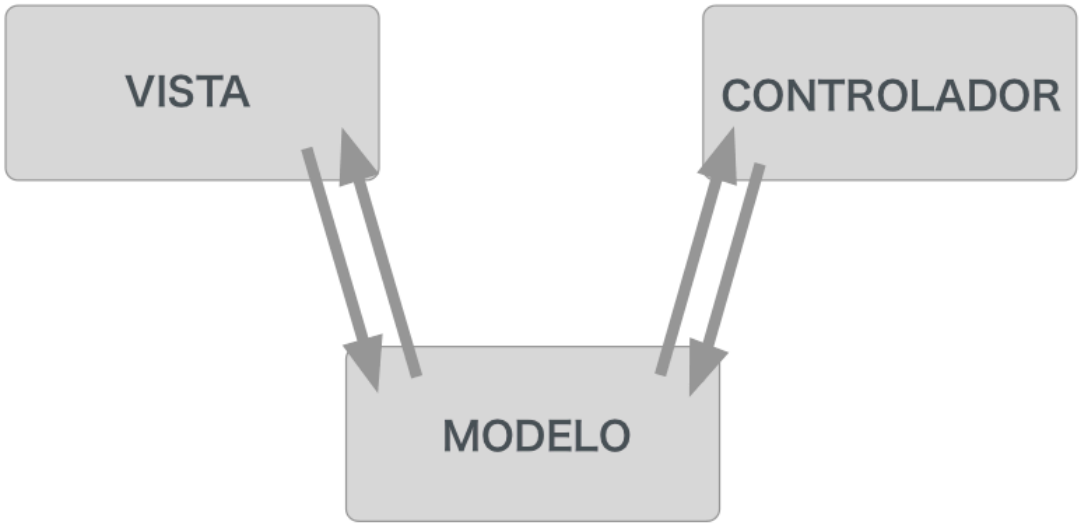


Ilustración 13: Arquitectura Modelo-Vista-Controlador.

La **arquitectura Modelo-Vista-Controlador (MVC)** está compuesta por tres elementos: el modelo, la vista y el controlador. Pasemos a analizar cada elemento por separado:

- **Modelo**: se encarga de gestionar la lógica y la información de toda la plataforma, y se sitúa entre la vista y el controlador, relacionándose con ambos. Recibe del controlador las acciones que el usuario requiere, y envía a la vista información para contestar a las peticiones del usuario de manera visual.
- **Vista**: es la parte visible de la aplicación; la parte con la que el usuario interactúa para ejecutar las acciones que desea llevar a cabo. Recibe la información que debe presentar del modelo, y también puede enviar al modelo información para que este se actualice tras alguna acción del usuario.
- **Controlador**: se encarga de aceptar la información entrante y convertirla en comandos para el modelo o la vista. La información entrante suele ser la interacción del usuario con el sistema, y el controlador la acepta y la procesa, para transformarla en mensajes apropiados que puedan servir de parámetros para las funciones del modelo.

Esta arquitectura está muy generalizada en páginas web, ya que es fácil de implementar y muy sencilla de adaptar, puesto que cambiando uno de los módulos de la aplicación puede adaptarse a otros formatos, sin necesidad de modificar el resto de los módulos. También permite mayor claridad a la hora de programar, pues separa claramente las funciones de las distintas partes de la aplicación, facilitando tanto la programación como el posterior testeo de las funciones.

Esta arquitectura Modelo-Vista-Controlador era la arquitectura que había en la aplicación inicial, sin una base de datos. Tras los cambios realizados en este trabajo, como se ha indicado, la nueva arquitectura puede definirse como una Arquitectura por Capas.

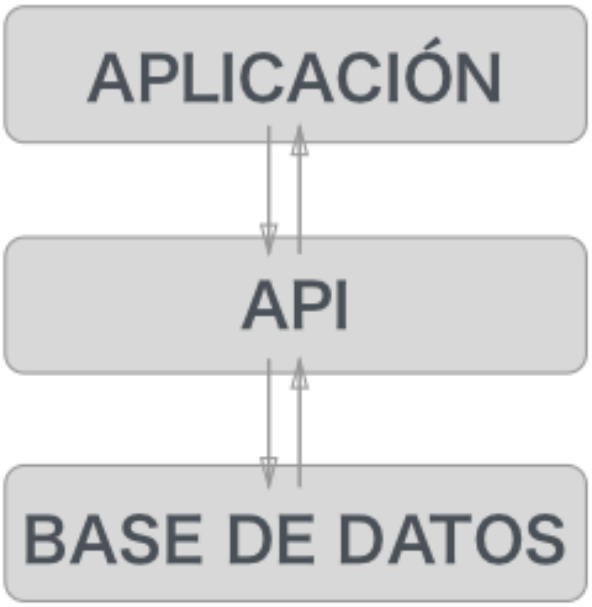


Ilustración 14: Arquitectura por Capas.

La **arquitectura por capas** está basada en un núcleo principal, sobre el cual se desarrollan el resto de capas. En este caso, tenemos tres capas, que se comentan en detalle a continuación:

- **Base de datos**: es la base de la arquitectura, almacena la información y la comparte con el resto de las capas cuando es necesario. Interactúa con la API recibiendo nueva información que almacenar en las tablas, y devolviendo las respuestas a las diversas consultas de los usuarios.
- **API**: La API interactúa con la aplicación y con la base de datos para transmitir consultas e información de una a otra. Esta API, la DEEPaaS RestAPI, ya estaba programada, y ha sido explicada en el apartado de recursos del Capítulo 2. Recursos y metodología.
- **Aplicación**: comprende el resto de los elementos de la plataforma, tanto las páginas visibles como las funciones que interactúan con la API para recibir información y enviar consultas a la base de datos.

Esta arquitectura suele emplearse porque permite añadir y eliminar capas con bastante libertad, permitiendo el crecimiento y la optimización de la plataforma sin tener que editar todo el código para conseguirlo. Se ha escogido esta arquitectura para las mejoras del proyecto porque integra la base de datos de una forma coherente y segura en la arquitectura que ya existía anteriormente.

5 Implementación y pruebas

Una vez que se ha definido el diseño de la aplicación, y los materiales con los que se han desarrollado las diversas mejoras planteadas en este proyecto, en esta sección se describe el proceso que se ha seguido para llevar a cabo la implementación de dichas mejoras en la plataforma, así como las pruebas que se han llevado a cabo sobre la aplicación para verificar su correcto funcionamiento.

La implementación es un proceso largo, que en este caso ha estado fuertemente entrelazado con el proceso de testeo con un objetivo: comprobar que los cambios que se iban realizando funcionaban correctamente. Este método de actuación permitía ir avanzando, de forma lenta pero segura, con el fin de evitar descubrir un error estructural en una prueba final que obligara a reiniciar el proceso desde el principio.

Se ha tomado esta decisión porque el trabajo de testeo lo realizaba la misma persona que el trabajo de implementación; en el caso de haber contado con dos equipos, la decisión habría sido implementar mejoras por bloques y testear dichas mejoras mientras se desarrollaban otras nuevas.

5.1. Implementación

Para llevar a cabo la implementación de la solución tecnológica se han distinguido tres categorías de desarrollos:

- Páginas nuevas.
- Páginas ya existentes, a modificar.
- La base de datos.

En cuanto a las **páginas nuevas**, la implementación ha comenzado por estudiar la estructura de las páginas ya creadas, para comprender cómo había que desarrollar las nuevas. Primero se incorpora una nueva etiqueta a la *navbar*, para poder incluir la opción de acceder a la URL de Experiments, que es una de las páginas completamente nuevas que se han añadido.

Otra de las páginas completamente nuevas que se ha incorporado es la de My Deployment History, a la que se puede acceder desde un botón añadido a otra página ya existente.

También se ha tenido en cuenta que todo elemento que se añadiese como funcionalidad a la nueva página debía ser fácil de utilizar, y bastante intuitivo. Tenía que tener sentido dentro del diseño, y ninguno de los enlaces a añadir podían fallar o dirigir fuera de la página a no ser que esta funcionalidad fuese la deseada.

En resumen, lo importante de la creación de páginas nuevas es añadir un elemento a la estructura ya existente que permita acceder a esa nueva página desde la plataforma, y no solamente conociendo la dirección URL a la que está asociada. Esto puede realizarse incorporando la página a la *navbar* o creando un nuevo botón o acceso en otro lugar de la plataforma.

Sobre los **añadidos a páginas ya existentes**, como el botón que da acceso a la página de My Deployment History, el procedimiento ha sido semejante. En este caso, también era importante estudiar la estructura de la página original para poder añadir los nuevos elementos sin alterar la estructura existente. Después, se escogía un elemento que encajase con el resto del diseño, y se añadía, comprobando que cumpliese con su funcionalidad.

Por ejemplo, el mencionado botón de acceso a My Deployment History está en la página ya existente de Deployments, que consta de un marco en el que en la parte superior derecha ya había dos botones: “Refresh” y “New Deployment”. El botón añadido, “History”, tiene la misma estructura estilística que los otros dos, para poder conservar la apariencia que ya tenía la página.

A continuación se puede observar como la incorporación del botón no ha estropeado la estructura de la página en dos imágenes comparativas. La primera imagen, mostrada en la ilustración 15, es la página inicial, que no contiene el botón, y la segunda imagen, mostrada en la ilustración 16, es la página editada, que sí contiene el botón diseñado para acceder al historial de los *deployments*.

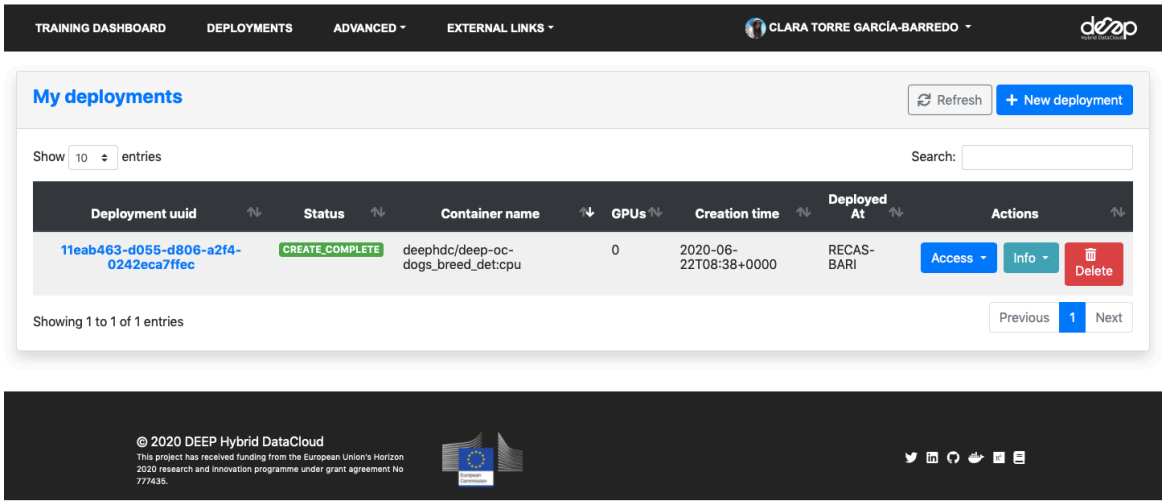


Ilustración 15: Página de Deployments sin botón de log.

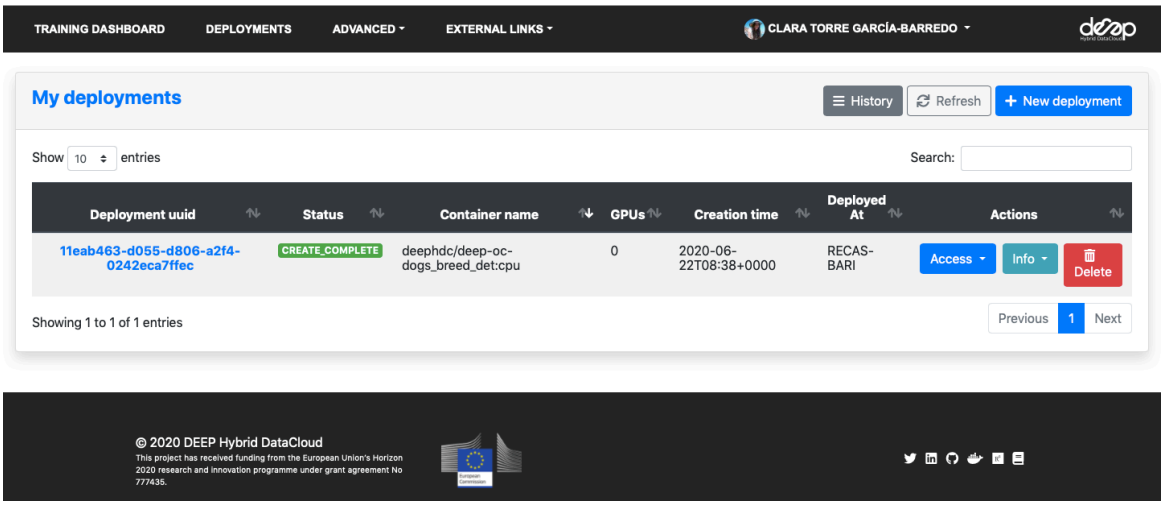


Ilustración 16: Página de Deployments con botón de log.

Otro elemento que tienen en común tanto la implementación de páginas nuevas como la de mejoras en páginas ya existentes es que, debido a la arquitectura original de la plataforma, no es posible realizar cambios únicamente en la parte HTML del código. También es importante crear funciones en Python para añadirlas al modelo, que sean capaces de manejar la información y las consultas que se realizan en las páginas que se han creado, y que sean capaces de hacer funcionar las redirecciones de los elementos nuevos.

La **base de datos** requiere de una implementación por partes, en la que primero se desarrollan las tablas de la base de datos, y posteriormente se enlaza dicha base de datos con el resto de funciones de la plataforma. En la ilustración 17 se puede ver una representación de la base de datos.

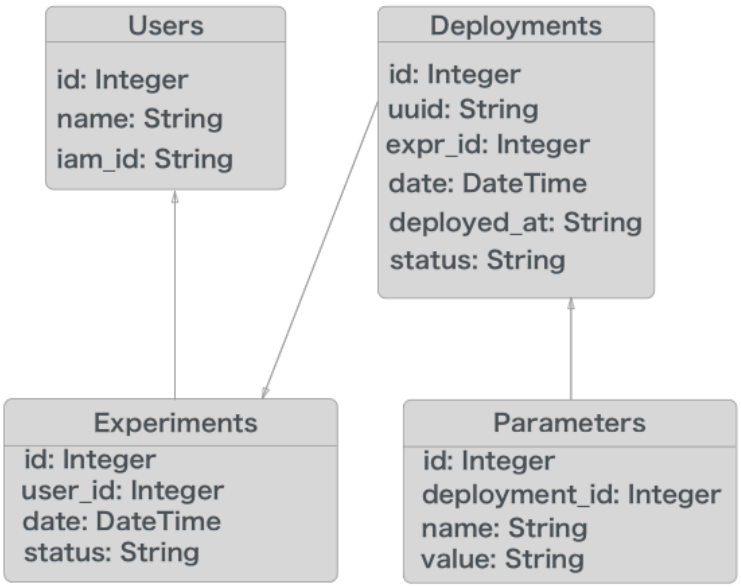


Ilustración 17: Representación gráfica de la base de datos.

Como se puede observar en la ilustración 17, la base de datos está compuesta por cuatro tablas:

- Users.
- Experiments.
- Deployments.
- Parameters.

La primera de ellas contiene la información de los usuarios, en concreto el id de usuario y su nombre y id de IAM, pues son los únicos datos que se necesita conocer para representar el historial de *deployments*, que es la principal función de la base de datos.

La segunda contiene la información de los experimentos, en concreto la fecha de creación, el estado, el id y a qué usuario pertenece.

La tercera de las cuatro contiene la información de los *deployments* de cada usuario, con una referencia al id del experimento, asi como a la fecha, estado y espacio virtual donde se hizo el *deployment* en concreto. En caso de no estar enlazadas a ningún experimento, este campo estará vacío. Con el uuid (id de IAM) se puede saber a qué usuario pertenece ese *deployment* en particular, así que este dato no es necesario almacenarlo en la base de datos. Y, por supuesto, el id específico del *deployment*. Esta es la información que se expone en el historial de *deployments* cuando se consulta a la base de datos desde un usuario concreto.

La última de las tablas de la base de datos contiene la información de los parámetros que se pasaron al crear un cierto *deployment*. Contiene, por tanto, su propio id y el id del *deployment*, además del nombre y valor del parámetro concreto, para poder representarlos todos cuando el usuario requiera la información.

Otro ejemplo de página en la que se ha integrado el cambio tratando de respetar al máximo la estructura que había anteriormente es el formulario en el que se recogen estos parámetros mencionados, al crear un nuevo *deployment*. Los cambios se pueden ver en la ilustración 18 y la ilustración 19 respectivamente:

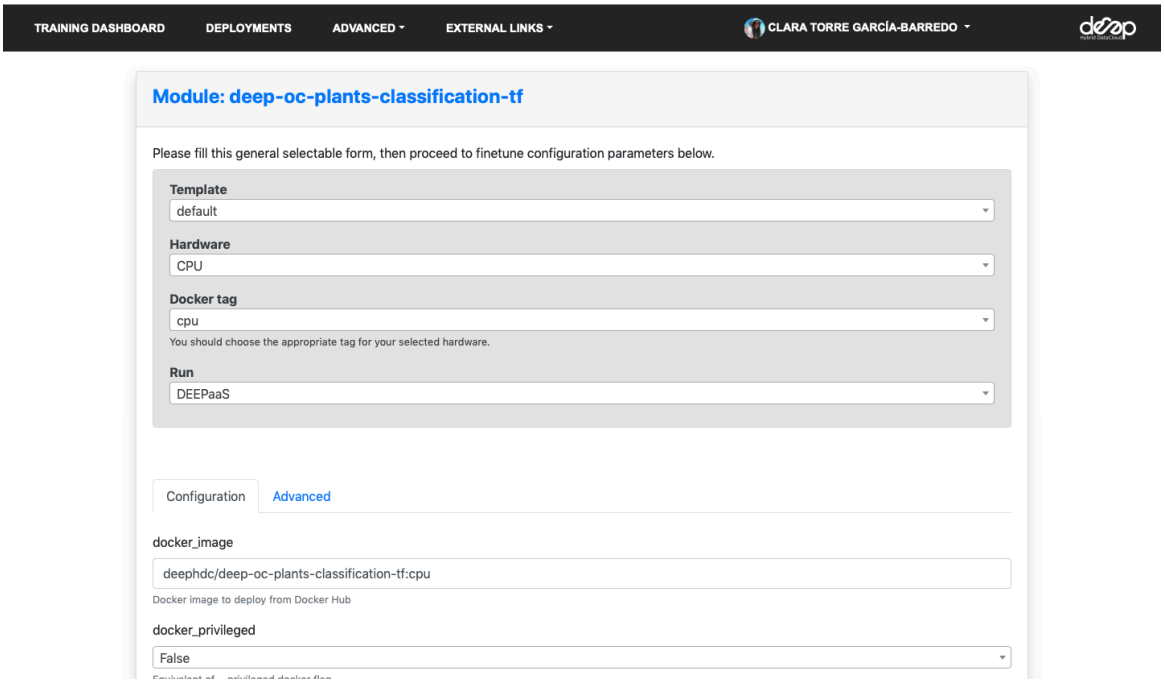


Ilustración 18: Formulario de creación de deployment sin experimentos.

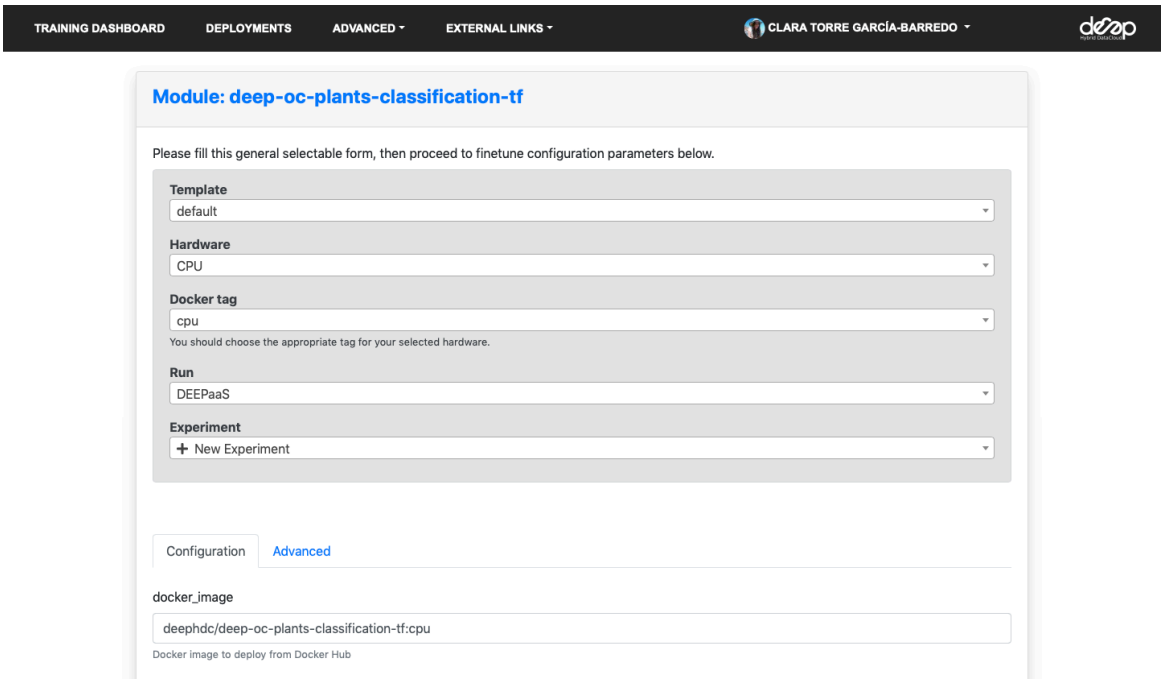


Ilustración 19: Formulario de creación de deployment con experimentos.

5.2. Pruebas

En cuanto a las pruebas realizadas, su objetivo principal era comprobar, desde la perspectiva del usuario, que todas las mejoras funcionasen correctamente a medida que se iban desarrollando, para evitar errores de carácter estructural.

Esta forma de actuar obedece a que las funcionalidades implementadas, la mayoría de carácter intuitivo, se comprobaban mejor interaccionando con la plataforma como lo haría el propio usuario, buscando los posibles errores a base de ejecutar las distintas acciones.

Así pues, se han creado nuevos *deployments* desde la página de creación de deployments, comprobando que el formulario de creación funcionase correctamente a pesar de los cambios implementados para añadir la funcionalidad de asociar un *deployment* a un experimento en el momento de su creación, y se han añadido a experimentos ya creados, para comprobar que la función de asociar un *deployment* nuevo a un experimento existente estuviera correcta.

Por supuesto, también se ha comprobado que la creación de un experimento vacío desde la página de experimentos estuviese disponible y funcionando correctamente. Se ha comprobado que se pudiera acceder a este experimento, y que la lista mostrase que no había ningún *deployment* asociado a él todavía.

También se ha comprobado que todas las interacciones posibles con la página de los experimentos estuviesen disponibles. Es decir, que además de poder crear un nuevo experimento vacío, se pudiera acceder a cada experimento para comprobar los *deployments* que estaban asociados a él. También se ha comprobado que se pudiera borrar un experimento, y que si se creaba un experimento desde la creación de un nuevo *deployment*, este apareciese correctamente en la lista de experimentos al finalizar su creación.

Desde la página de Deployments, también se ha comprobado que se podía acceder al historial de *deployments*, comprobando que todos los que se habían creado hasta el momento estaban ahí representados, con su información correcta. Además, se ha probado a borrar un deployment y luego acceder al historial de *deployments*, para comprobar que este *deployment* seguía ahí, aunque ya no fuese accesible, pues estaba eliminado.

En conclusión, se ha comprobado que todas las funciones que se habían añadido funcionasen correctamente, y que los errores que están programados aparezcan solamente cuando se les espera.

6 Conclusión

El proyecto DEEP Hybrid Datacloud tiene como principal objetivo preparar una nueva generación de infraestructuras para soportar Deep Learning y otras técnicas de computación intensiva para explotar grandes fuentes de datos. A lo largo de este trabajo se han implementado mejoras que buscan aumentar la calidad de la experiencia de usuario, añadiendo funcionalidad.

6.1. Conclusiones

A diferencia de otros trabajos, este no partía de cero, sino que tenía como base una aplicación completamente funcional, el DEEP Training Dashboard, y su objetivo principal era aumentar la calidad de la experiencia de usuario con algunas funciones de gran utilidad.

El resultado de este trabajo se va a utilizar dentro del proyecto, ya que el código se va a añadir al repositorio del proyecto, y va a ser empleado en producción por los diferentes usuarios de la plataforma.

Con la realización de este proyecto se han satisfecho los siguientes requisitos funcionales:

- Se ha incluido una base de datos que recoge la información de los *deployments* creados por el usuario.
- Se ha creado la página Experiments, que permite agrupar los *deployments* del usuario.
- Se ha añadido al formulario de creación de los *deployments* una opción que permite incluir al *deployment* en un experimento existente o uno nuevo. También se ha creado la posibilidad de crear nuevos experimentos en la página de Experiments.
- También se ha añadido la opción de no incluir el *deployment* a ningún experimento en el formulario de creación.
- Se ha creado el historial de *deployments*, donde se puede consultar la información de la base de datos.
- Utilizando la funcionalidad creada para la aplicación inicial, se ha comprobado que los usuarios solo pueden acceder a las nuevas páginas estando registrados.

También se han satisfecho los siguientes requisitos no funcionales:

- Se ha mantenido la cohesión estilística con la aplicación inicial utilizando las mismas tecnologías para implementar las mejoras.
- Se ha cuidado la usabilidad de la aplicación, añadiendo iconos explicativos en las mejoras.

En el ámbito personal, la realización de este proyecto ha sido un gran reto. Me ha permitido conocer herramientas que no había tenido la oportunidad de estudiar a lo largo de la carrera y que considero muy interesantes, como han sido Flask o SQLAlchemy, profundizar en una plataforma web de este calibre, pudiendo analizar con detalle todas sus partes, y fundamentalmente, aprender a organizar mi tiempo para poder superar cada uno de los objetivos en las diferentes fases de un proyecto.

También me he enfrentado durante la escritura de esta memoria, extensa y dedicada, al reto de describir con detalle cada uno de los pasos realizados, argumentando mis decisiones y documentando mis procesos. Me ha enseñado a ser constante y a mantener un registro de las tareas realizadas y de las pendientes por realizar, como la estructura Agile que predomina en muchos proyectos *software*.

6.2. Trabajo futuro

Durante el desarrollo de este proyecto han surgido nuevas ideas de mejoras que se podrían añadir en un futuro, y que aumentarían el valor de la plataforma. Algunas de esas sugerencias son las siguientes:

- Extender la funcionalidad de los experimentos para poder linkar datasets. Esta idea trata de añadir posiblidades a los experimentos, para poder detallar, además del módulo y los parámetros empleados en los *deployments*, qué datasets se han empleado, e incluso unos enlaces a su localización pública, a través del DOI u otros métodos de identificación.
- Extender la funcionalidad de los experimentos para poder obtener estadísticas y gráficas sobre los parámetros usados. Esta nueva mejora ayudaría a los usuarios a comprobar qué parámetros funcionan mejor con determinado módulo o dataset, para conseguir optimizar sus entrenamientos.
- Extender la funcionalidad de los *deployments* para poder ejecutar varios módulos de forma paralela, empleando el mismo formulario de parámetros y el mismo dataset. Con ello, el usuario podría, entre otras opciones, comparar la eficacia de un módulo entrenado con uno entrenable.
- Extender la funcionalidad de los *deployments* para poder ejecutar un módulo con los resultados de otro de forma programada. Esta función permitiría al usuario programar el *deployment* de varios módulos de forma secuencial, de forma que al terminar la ejecución del primero, los resultados sirvieran como dataset para el segundo.

Bibliografía

[1] Michael Bayer. *SQLAlchemy Documentation*. URL: https://www.sqlalchemy.org. (last accessed 23.06.2020).

[2] Tobias Binz y col. «TOSCA: Portable Automated Deployment and Management of Cloud Applications». En: *Advanced Web Services*. Ed. por Athman Bouguettaya, Quan Z. Sheng y Florian Daniel. New York, NY: Springer New York, 2014, págs. 527-549. ISBN: 978-1-4614-7535-4. DOI: 10.1007/978-1-4614-7535-4_22. URL: https://doi.org/10.1007/978-1-4614-7535-4_22.

[3] Igor Costa y col. «Availability Evaluation and Sensitivity Analysis of a Mobile Backend-as-a-service Platform». En: *Quality and Reliability Engineering International* 32.7 (2016), págs. 2191-2205. DOI: 10.1002/qre.1927. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qre.1927. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/qre.1927.

[4] DEEP-Hybrid-Datacloud. *DEEP Hybrid Datacloud Documentation*. URL: https://docs.deep-hybrid-datacloud.eu/en/latest/user/overview/api.html. (last accessed 17.06.2020).

[5] DEEP-Hybrid-Datacloud. *The Project: DEEP Hybrid Datacloud*. URL: https://deep-hybrid-datacloud.eu. (accessed: 15.06.2020).

[6] Miguel Grinberg. *Flask Web Delevopment*. Sebastopol, CA, USA: O'Reilly Media, 2014.

[7] Álvaro López García. «DEEPaaS API: a REST API for Machine Learning and Deep Learning models». En: *Journal of Open Source Software* 4.42 (25 de oct. de 2019), pág. 1517. ISSN: 2475-9066. DOI: 10.21105/joss.01517. URL: http://dx.doi.org/10.21105/joss.01517.

[8] Jupyter Project. *JupyterLab Documentation*. URL: https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html. (last accessed 22.06.2020).

[9] Mike Roberts. *Serverless Architectures*. URL: https://martinfowler.com/articles/serverless.html#WhatIsServerless. (last accessed: 15.06.2020).

[10] Swagger UI. *Swagger UI Documentation*. URL: https://swagger.io/tools/swagger-ui/. (last accessed 16.06.2020).