CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Matemáticas e Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Entrega Deliverit: Evolución envío de artefactos

Autor: Patricia López García

Director: Ángel Herranz

MADRID, JUNIO DE 2020

A mis familiares, amigos y seres queridos
que me han soportado durante todos estos años.

A mi primer tutor profesional Ignacio del Pozo, compañero y amigo,
que me contagió su pasión por la tecnología y consiguió sacar lo mejor de mí.

A mi tutor académico Ángel Herranz y mi compañero Andrés Mareca,
por hacer posible este TFG y ayudarme a desarrollarlo a pesar de las circustancias.

A todos vosotros, os dedico mi mas sincero y enorme Gracias.

Índice general

Índice de figuras

Resumen

Desde la experiencia de alumna en la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF), durante toda la carrera me he enfrentado a distintos tipos de asignaturas, lenguajes y plataformas en las que entregar los respectivos ejercicios prácticos de dichas asignaturas. Esto último puede resultar bastante laborioso, además de poco óptimo en cuanto a inversión de tiempo se refiere. Desde el punto de vista del profesorado el problema es similar: moverse de una asignatura a otra implica aprender un nuevo sistema de entrega.

Para intentar optimizar y simplificar el trabajo, nació la idea de desarrollar un nuevo sistema. Como bien podemos observar en el nombre del presente trabajo fin de grado, ***Entrega Deliverit: Evolución envío de artefactos***, se trata de un sistema de entregas de prácticas para facilitar la entrega a los alumnos y la corrección de los profesores de éstas mismas, siendo un único sistema y no varios.

Dado que en el comienzo de este trabajo ya existía una version beta del Sistema, me he encargado de continuar mejorando el funcionamiento de la plataforma, con el desarrollo de nuevas funcionalidades y resolución de problemas, las más importantes las podemos ver a continuación:

Mejora información Docker: Algunos de los errores que podían provocar los contenedores no estaban bien expresados en la salida de errores de la interfaz gráfica cuando se realizaba una nueva práctica.

Conectividad SSE: Se requería mostrar los *logs* de un contenedor, es decir salida estándar, en el front a los alumnos. Para ello se ha implementado una conexión mediante SSE (*Server-sent events*).

Durante la memoria de este trabajo se encontrarán con los puntos claves de arquitectura para el entendimiento del sistema, decisiones tomadas, herramientas empleadas y procesos implementados explicados. Además de ésto, servirá como continua documentación del sistema para futuros estudiantes o profesores que quisieran colaborar en el desarrollo de éste mismo.

Abstract

From the experience of a student in the *Escuela Técnica Superior de Ingenieros Informáticos* (ETSIINF), throughout the career I have faced different types of subjects, languages and platforms in which to deliver the respective practical exercises of these subjects. The latter can be quite laborious, as well as not optimal in terms of time investment. From the point of view of instructors, the problem is similar: moving from one subject to another implies learning a new delivery system.

To try to optimize and simplify the work, the idea of developing a new system was born. As well we can observe in the name of the present final Project Degree, ***Entrega Deliverit: Evolución envío de artefactos*** in English *Deliverit:Evolution sending artifacts*, it is a system to facilitate the delivery of the students's practical exercises and the correction by teachers of the same, being a single system and not several.

Since at the beginning of this work there was already a beta version of the system, I have been able to continue improving the operation of the platform, with the development of some functionalities or problems that were present, some of the most relevant ones are presented below:

Improve Docker Information: Some of the errors that containers could cause were not well expressed in the graphical interface errors output when a new practice was performed.

Server-Sent Events: It was required to show the logs of a container, standard output, in the student's front. For this, an SSE connection has been implemented, which I explain later.

During the memory of this work will be found with the key architectural points for understanding the system, decisions made, tools used and implemented processes explained. In addition to this, it will serve as a continuous documentation of the system for future students or teachers who would like to collaborate in its development.

1

Introducción

Actualmente en la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) de la Universidad Politécnica de Madrid (UPM), durante toda la carrera tanto mis compañeros como yo, nos hemos enfrentado a distintos tipos de asignaturas. Muchas de ellas, contaban con diferentes lenguajes de programación y plataformas en las que entregar las respectivas prácticas de dichas asignaturas. Esto último puede resultar bastante laborioso en algunos casos, además de una inversión de tiempo extra.

Si pensamos en los profesores, también se encontrarán con el mismo problema: tener distintas plataformas para la corrección de prácticas de cada una de las asignaturas que imparten. Imagínense impartir distintas asignaturas de distintos niveles, con distintas plataformas, métodos de entregas y correcciones, requisitos elementales.

¿Podríamos ayudar a ambos miembros de la comunidad educativa a optimizar este proceso de entrega de prácticas? La respuesta es sí. Como bien podemos observar en el presente trabajo fin de grado, ***Entrega Deliverit: Evolución envío de artefactos***, se trata de un proyecto liderado por Ángel Herranz y con la colaboración de otros miembros del equipo docente de la ESTIINF, en el que se quiere desarrollar un sistema de entrega de prácticas para facilitar tanto la entrega a los alumnos como la corrección de los profesores de éstas mismas, unificándolo así en una sola plataforma y no varias.

Ciertamente, los alumnos de esta facultad, se convertirán en futuros ingenieros informáticos, programadores, informáticos y matemáticos, etc. Durante su estancia en los años de

estudio en la Universidad, deben someterse al mismo tiempo a una preparación para el mundo profesional, dejando atrás el estudiantil. En relación al mundo laboral, cada uno de los profesionales se ven en la necesidad de aprender a emplear nuevas interfaces gráficas, así como herramientas y plataformas continuamente. Pero en el periodo de formación, el tiempo y esfuerzo que un estudiante invierte en actualizar su conocimiento en cada uno de los sistemas, en gran parte de las situaciones, se desvía de los objetivos principales de las sucesivas asignaturas, perdiendo la oportunidad de profundizar todavía más en los conceptos que se pretenden asentar a través de los ejercicios prácticos.

Como ya se ha comentado, esta idea fue propuesta por el profesor Ángel Herranz, perteneciente al conjunto del profesorado de la Escuela. Un sistema de entregas único, con la capacidad de soporte suficiente para la entrega de la amplia variedad de prácticas y métodos de evaluación llevados a cabo en la facultad. Ofreciendo una interfaz unificada y de fácil uso, que tanto alumnos como profesores emplearían durante los distintos cursos académicos. Esta herramienta, además de su uso en los distintos grados que se imparten en nuestra facultad, podría facilitarse a otros programas como Máster, Doctorado, e incluso en un futuro, a Escuelas y Universidades alternativas.

Además de esto, dicha plataforma cuenta con tecnologías muy innovadoras y potentes en estos tiempos, como pueden ser Phoenix-Elixir, Elm y Docker. Esto último es muy importante porque hoy en día muchas empresas continúan con tecnologías que ya están obsoletas y la mayor parte de las veces sin soporte, lo que les expone a un claro peligro y fácil ataque de seguridad informático. Dichas empresas continúan sus negocios y desarrollos sin dar una oportunidad a la innovación, sin darse cuenta de lo potente que pueden llegar a ser las nuevas tecnologías, que por miedo al desconocimiento de sus funcionalidades, ignoran.

Es por ello por lo que elegí este proyecto, por la doble innovación que conlleva, siendo la primera la creación de un nuevo sistema conjunto para las distintas prácticas y la apuesta por las nuevas tecnologías. Si no somos las nuevas generaciones, con la ayuda de los profesionales con más experiencia, las que aportamos nuestro granito de arena al desarrollo de nuevas tecnologías y herramientas ¿Quién lo hará?

Cabe a destacar que cuando me uní a este proyecto, el sistema estaba avanzado, contando ya con una versión beta desarrollada e implementada por Ángel Herranz y Andrés Mareca (alumno de la Escuela) principalmente, así como con colaboración de otros miembros del equipo docente y alumnado de la ETSIINF. Para lograr la versión beta del sistema, tuvieron que diseñar e implementar la arquitectura de éste mismo, teniendo en cuenta las necesidades tanto de profesores como alumnos, para intentar cubrir el máximo posible de los mismos.

¿Y qué papel tengo yo en este proyecto? Bien, aunque se contase con una versión beta del sistema, éste se encontraba con las funcionalidades básicas, por lo que mi trabajo ha constado en mejorar y desarrollar algunas de las funcionalidades con las que el sistema todavía no contaba, ampliando así el campo de necesidades cubiertas. Este trabajo se realizó a la par que

mi compañero Andrés Mareca y dos alumnos más de la Escuela.

Durante la continua lectura del presente trabajo fin de grado, se encontrarán con los sucesivos capítulos donde explicaré y desarrollaré desde qué es, para qué sirve y como funciona el Sistema Deliverit *capítulo 3*, así como una visión general de la arquitectura y tecnologías empleadas; la mejora de información en la tecnología Docker y lo que supuso, *capítulo 4*; el desarrollo de la conectividad mediante SSE para la muestra de *logs* en la entrega de una práctica *capítulo 5*; hasta finalmente las conclusiones y futuro trabajo que pueda realizarse en el sistema, *capítulo 6*.

Considero que este proyecto es realmente importante y necesario en nuestra facultad, dado que muchas de las tecnologías empleadas en las plataformas de entregas ya existentes, estan obsoletas, así como las interfaces gráficas, uso y configuración de muchas de ellas, desde el punto de vista de usuario.

1.1 Metodología

Este proyecto, lo he realizado al mismo tiempo que mi compañero Andrés Mareca y otros dos alumnos más de la Facultad y es por esto por lo que nace la presente sección, donde presentaré las tecnologías y herramientas que hemos empleado para el continuo desarrollo del trabajo y comunicación entre los miembros del equipo, junto con el director del proyecto, Ángel Herranz.

Para el desarrollo de este proyecto, se instauraron prácticas y procedimientos inspirados en marcos de implementación de metodologías ágiles, como *Scrum* [18] o *Kanban* [2]. Estas surgen para dar respuesta al cambio de manera rápida y efectiva durante el desarrollo de proyectos *software* [19]. Motivadas por la naturaleza evolutiva de las necesidades de los proyectos, establecen estrategias para el desarrollo iterativo e incremental. La decisión para adoptar estas técnicas venía motivada por diferentes factores.

***Product Backlog* y ciclo de vida de la unidad de trabajo.** El *Product Backlog* es un listado de todas las necesidades conocidas del sistema que deben ser implementadas. Las tareas se ordenan según su prioridad, de modo que el equipo de desarrollo pueda anticiparse y prever las necesidades de los siguientes ciclos. El dueño del producto, es decir Ángel Herranz, es el responsable último de mantener este listado actualizado.

Por su parte, el equipo de desarrollo debe asegurarse de que las tareas incluidas en el *Product Backlog* contienen toda la información necesaria para ser desarrolladas, de modo que puedan abordarse en los ciclos de desarrollo.

Se estableció el siguiente ciclo de vida para las unidades de trabajo, el cual ya estaba

definido cuando comencé el proyecto:

- **Por hacer**: Son las tareas que se deben abordar en el ciclo de desarrollo actual.
- **En progreso**: Son aquellas tareas en las que el equipo de desarrollo ya ha empezado a trabajar.
- **Bloqueada**: Son tareas para las que se han encontrado dificultades durante su desarrollo, sean técnicas o por falta de definición, que impiden completarlas.
- **Hecha**: Son aquellas tareas que ya han sido abordadas por el equipo de desarrollo y se considera que están finalizadas.
- **Validada**: Una vez las tareas hechas se validan con el dueño del producto, pasan a este último estado.

Este ciclo de vida se implementó utilizando un tablero *Kanban*. Estos tableros consisten en una serie de columnas y tarjetas. Las columnas se corresponden con los distintos estados (empezando con el *Product Backlog*), mientras que las tarjetas se corresponden con unidades de trabajo o tareas. Para ello se empleó la herramienta software *Trello*, que presento más detalladamente a continuación.

Trello. Trello es un software de administración de proyectos con interfaz web y con cliente para iOS y android. Sirve para gestionar tareas, permitiendo organizar el trabajo en grupo de forma colaborativa mediante tableros virtuales compuestos de listas de tareas en forma de columnas. Es versátil y fácil de usar pudiendo usarse para cualquier tipo de tarea que requiera organizar información.

Estas listas se subdividen en entradas denominadas 'tarjetas' que recogen el contenido o labores pendientes en tu proyecto dentro de cada una de las listas. En dichas tarjetas podrás reflejar todo el contenido relevante a tu trabajo: crear comentarios, generar checklists. Además, puedes incluir archivos adjuntos, establecer calendarios y alarmas para cada una de las entradas o tarjetas del tablero, fechas de vencimiento, etiquetas para diferenciar que tienes que hacer.

Otro de los aspectos interesantes que pone a tu disposición Trello, es que permite el trabajo en equipo de forma instantánea y telemática, a tiempo real. Esto es, permite invitar a otras personas a tus tableros, de forma que se pueda iniciar conversaciones mediante comentarios en relación a entradas concretas, compartir archivos en streaming y desarrollar de forma conjunta la gestión de proyectos online al mismo tiempo.

1.1 Metodología

Ciclos de desarrollo cortos. Se establecieron reuniones periódicas cada semana y ciclos de desarrollo de la misma duración. Estos ciclos de desarrollo reciben comúnmente el nombre de *sprints*. Dado las condiciones que hemos tenido que experimentar toda la ciudadanía, dichas reuniones al principio del proyecto eran presenciales, pero terminaron realizándose de forma online mediante la plataforma de Microsoft Teams [15].

Durante cada reunión, el dueño del producto y el equipo de desarrollo validan conjuntamente los avances del último ciclo de desarrollo. El dueño del producto puede aceptar los desarrollos llevados a cabo en esta iteración, o rechazarlos en caso de que estos no cumplan con sus expectativas. Si una tarea es aceptada se marca como *Validada*, mientras que si se rechaza volverá al estado *por hacer* y se refinará durante el siguiente ciclo. Para estas últimas, se establecen los puntos pendientes o los cambios necesarios para darlas como válidas.

Gitlab En este apartado, presentaré la metodología que se ha llevado a cabo durante el desarrollo de este proyecto a nivel de código, la cual ha servido de ayuda para la continua coordinación y colaboración entre los distintos miembros del equipo de desarrollo y el dueño del proyecto.

Para ello, como el propio nombre de este apartado indica, se empleó la herramienta *Gitlab*. Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto. Cabe resaltar que además de ser un GIT, GitLab es una suite completa que permite gestionar, administrar, crear y conectar los repositorios con diferentes aplicaciones y hacer todo tipo de integraciones con ellas.

El código se almacena en un repositorio el cual tenia que mantener un cierto orden, para ello el dueño del proyecto y Andrés Mareca, decidieron utilizar *Git-Flow* [6]. Trabajar con este flujo de trabajo implica que cada vez que queramos hacer algo en el código, tendremos que crear la rama que corresponda, trabajar en el código, incorporar el código donde corresponda y cerrar la rama. A lo largo de nuestra jornada de trabajo necesitaremos ejecutar varias veces al día los comandos git, merge, push y pull así como hacer checkouts de diferentes ramas, borrarlas, etc. Git-flow son un conjunto de extensiones que nos ahorran bastante trabajo a la hora de ejecutar todos estos comandos, simplificando la gestión de las ramas de nuestro repositorio.

El trabajo se organiza en dos ramas principales:

- **Rama master:** cualquier commit que pongamos en esta rama debe estar preparado para subir a producción.
- **Rama develop:** rama en la que está el código que conformará la siguiente versión planificada del proyecto.

Además de estas ramas, se trabaja con las siguientes ramas auxiliares:

- **Feature:** Estas ramas se utilizan para desarrollar nuevas características de la aplicación que, una vez terminadas, se incorporan a la rama develop (también se originan a partir de esta misma).
- **Release:** Estas ramas se utilizan para preparar el siguiente código en producción. En estas ramas se hacen los últimos ajustes y se corrigen los últimos bugs antes de pasar el código a producción incorporándolo a la rama master.
- **Hotfix:** se utiliza para corregir errores y bugs en el código en producción. Funcionan de forma parecida a las Releases Branches, siendo la principal diferencia que los hotfixes no se planifican.

1.2 Estructuración Memoria

Una pequeña guía de la estructuración que se lleva acabo, a partir de esta sección, en el presente trabajo fin de grado. A continuación se exponen los distintos capítulos y secciones de este mismo, así como una breve introducción sobre lo que trata cada uno de ellos.

- **Capítulo 3.** En este capítulo explico qué es, para qué sirve y cómo funciona el Sistema Deliverit en general. Constará de una sección del estudio previo 3.1 *Estudio Previo* que tuve que realizar para aprender algunas de las tecnologías empleadas 3.1.1 *Tecnologías*, así como entender la arquitectura 3.1.2 *Arquitectura* y funcionalidades del sistema 3.2 *Modelo de interacción*.
- **Capítulo 4.** Se presenta la mejora de información en las interfaces gráficas 4.1, así como la salida de errores de los contenedores Docker 4.2. Además de esto, se presentan las traducciones realizadas en el sistema 4.3 *Traducciones*.
- **Capítulo 5.** Una de las funcionalidades más importantes desarrolladas en este proyecto, donde se explicarán la motivación 5.1, herramientas 5.2 *Herramientas*, decisiones previas 5.3 *Decisiones previas* e infraestructura SSE 5.4 *Infraestructura* Server to Client, así como el desarrollo 5.5 *Desarrollo*, donde se podrá leer sobre la extracción de la información en Docker 4 *Extracción información Docker* y un breve testeo en el sistema 5.5.2 *Pruebas*.
- **Capítulo 6.** Por último, las conclusiones del presente trabajo fin de grado, así como el trabajo futuro 6.1 *Trabajo Futuro* que queda por realizar y motivación 6.2 *Motivación* para que futuros estudiantes colaboren en el proyecto.

2

Glosario

Este capítulo ofrece un glosario de términos inspirado en el concepto *ubiquous language* de Eric Evans [9]. El glosario debe servir al equipo de desarrollo a hablar un lenguaje común. En palabras del propio autor:

> The UBIQUITOUS LANGUAGE can help to tie the two components together. Although it is a lot of work, and mapping may seem to make it unnecessary, corresponding elements in the objects and the relational tables should be named meticulously the same and have the same associations. Subtle differences in relationships will cause a lot of confusion. The tradition of refactoring that has increasingly taken hold in the object world has not really affected relational database design much. What's more, serious data migration issues discourage frequent change. This may create a drag on the refactoring of the object model, but if the object model and the database model start to diverge, transparency can be lost quickly.
>
> Domain-Driven Design Tackling Complexity in the Heart of Software.
>
> Eric Evans

2 Glosario

Assistant: hace referencia a cualquier persona encargada de las prácticas de una o varias asignaturas. No es necesario que sea un profesor, puesto que en muchas ocasiones con los estudiantes de Doctorado los que se dedican a esta labor dentro de sus prácticas.

Base (Código de apoyo): este término hace referencia al código que es necesario añadir en el entorno seleccionado, para poder lanzar el código de un alumno a ejecutar asociado a una práctica.

Environment (Entorno): conjunto de características que definen el contexto en el que se va a ejecutar el código base junto con el código de un alumno.

From (Desde): indica la fecha a partir de la cual un alumno puede comenzar a realizar entregas.

Mark (Calificación): nota que obtiene un alumno después de la ejecución de su entrega.

Max Mark (Calificación Máxima): se establece a la hora de crear una práctica y se corresponde a la máxima calificación que puede obtener un alumno al entregar dicha práctica.

Group (Grupo): correspondiente al conjunto de alumnos que realizan una práctica y que pueden efectuar entregas sobre ella.

Member (Miembro): participación de un alumno dentro de un grupo.

Project (Práctica): proyecto software de código a realizar dentro de una asignatura.

Registration number (Número de matrícula): identificador propio de un alumno dentro del Sistema.

Step (Paso): intrucción a ejecutar dentro de un entorno correspondiente a una entrega.

Subject (Asignatura)

Submission (Entrega): acción del sistema donde el estudiante entrega su código de la práctica en un momento espefíco.

To (Hasta): indica la fecha a partir de la cual un alumno no puede realizar más entregas.

Note: comentarios que un profesor realiza dentro de un grupo.

Mandatory (Requerido): término dentro de una práctica que establece como necesario el éxito de un paso para ejecutar el siguiente.

Regex (Project): expresión regular encargada de buscar en la salida de un paso dos enteros, que deberán devolver los tests exitosos de los alumnos y el numero total de tests.

3

Deliverit

En este capítulo podrán encontrar la presentación general del Sistema Deliverit. En ella se desarrollará, por un lado; el estudio previo que tuve que realizar para obtener los conocimientos necesarios y la suficiente confianza como para poder desarrollar y mejorar algunas de las funcionalidades que se me plantearon; así como por otro lado, el funcionamiento del Sistema, qué es, cómo se comporta, usabilidad, tecnologías empleadas y arquitectura desarrollada, focalizándolo desde el punto de vista del entendimiento del sistema y no de la construcción del mismo.

Como he comentado anteriormente, Deliverit nació de la mano del profesor Ángel Herranz y del estudiante Andrés Mareca. Ellos diseñaron la arquitectura del sistema desde cero, escogiendo las tecnologías adecuadas y estrategias que llevarían a cabo para el desarrollo del mismo. Imagino que esto tuvo que ser un trabajo bastante laborioso y complejo, dado que implementar un sistema tan potente desde cero siendo tan pocos desarrolladores, tuvo que ser, si me permiten ser coloquial, un auténtico dolor de cabeza.

Sin más dilaciones, a los presentes lectores de este trabajo fin de grado, les introduzco a la explicación del Sistema Deliverit, cuya vida es corta, pero con firme intención de crecimiento.

3.1 Estudio previo

Para comenzar con esta sección, volveré a remarcar que cuando comencé en este proyecto, el Sistema Deliverit ya contaba con una versión beta desarrollada, por lo que era de crucial importancia, ponerme al día con las tecnologías que no conocía, así como con la arquitectura y funcionamiento de este.

Si bien es cierto que yo ya conocía una de las tecnologías que se empleaban en el sistema y que para la parte en la que iba a trabajar yo, era bastante importante. Dicha tecnología se trataba de los Dockers, que explicaré más adelante, en el capítulo *3.1.1 Tecnologías*

En cuanto a la documentación del sistema, se disponía de la memoria del TFG de un alumno que colaboró meses antes en el proyecto, junto con Andrés Mareca, por lo que teníamos una pequeña guía que seguir para poder entender ciertos aspectos del proyecto como son la arquitectura, distribución de las aplicaciones y modelo de datos del sistema, aspectos fundamentales para el entendimiento del sistema, más adelante en la sección ***3.2 Sistema*** se exponen los principales diagramas de interacción establecidos.

Comenzaré por el final. Una vez desplegado el sistema, las funcionalidades implementadas con Deliverit, eran el alta de alumnos, creación de prácticas, asignaturas y entregas de las mismas. Todo era bastante intuitivo y de fácil uso y manejo dentro del panel de administración, dado que la interfaz gráfica escogida era muy simple y la organización de la misma muy adecuada. Así que si, podría decirse que una vez tienes desplegado el sistema en tu maquina local, es bastante fácil de usar. Pero ¿que tuve que aprender hasta llegar ahí?

Para el correcto despliegue del sistema, tuve que instalar las tecnologías Erlang, Elixir, node y postgress, además de crearme una máquina virtual en local con el sistema operativo Linux. Las que más trabajo me costaron entender fueron Erlang y Elixir, al contrario que su instalación, la cual fue bastante sencilla. Antes de desplegar el sistema real, probé a crearme mi propia aplicación Phoenix, con la que pude *trastear* para descubrir las funcionalidades que se podrían llevar a cabo con esta misma.

Después de todo esto, comencé a adentrarme más en el sistema y organizarme un poco por el mismo, dado que una de las cosas, digamos no más complicadas sino liosas, fue las rutas a las distintas aplicaciones, donde se encontraba cada cosa, donde tenía que ir si quería modificar o añadir una funcionalidad del sistema, la lógica establecida, archivos para las interfaces graficas de cada operativa, etc. Mas adelante explicare esto con más detalle. A continuación, doy paso a la explicación de las tecnologías empleadas.

3.1.1 Tecnologías

En este apartado del capítulo 2, presentaré y explicaré las tecnologías empleadas en Deliverit. Si bien es cierto que, yo no he participado en la elección de las tecnologías del sistema, pero después de trabajar con ellas y profundizar más, confirmo que no podría haber sido mejor elección, para cubrir las necesidades que se exponían mediante el desarrollo software.

- **HTML/CSS (Bootstrap) + Javascript** [1]: es utilizado para el desarrollo de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS [7], así como extensiones de JavaScript adicionales [8]. A diferencia de muchos frameworks web, solo se ocupa del desarrollo *front-ends*, empleado en este proyecto. Es bastante fácil de usar y entender sin haberlo conocido antes, lo cual te facilita el tiempo invertido en estudio previo antes de comenzar el desarrollo.

- **Erlang**: Erlang es un lenguaje de programación concurrente y un sistema de ejecución que incluye una máquina virtual y bibliotecas. El subconjunto de programación secuencial de Erlang es un lenguaje funcional, con evaluación estricta, asignación única, y tipado dinámico [14].

  Para el desarrollo de una de mis tareas, he tenido que invertir tiempo en comprender las aplicaciones Erlang/OTP, las cuales, aunque explique más adelante, doy una breve introducción. Una aplicación Erlang es un grupo de código y procesos relacionados. Una aplicación OTP usa específicamente comportamientos OTP para sus procesos, y luego los envuelve en una estructura muy específica que le dice a la VM cómo configurar todo y luego derribarlo.

- **Phoenix/Elixir**: Phoenix es un marco de desarrollo web escrito en el lenguaje de programación funcional Elixir [13]. Phoenix utiliza un patrón de modelo-vista-controlador del lado del servidor.

  Para ejecutar Elixir, necesitamos la máquina virtual Erlang porque el código Elixir se compila a código byte Erlang. Elixir comparte las mismas abstracciones para desarrollar aplicaciones distribuidas y tolerantes a fallos. Todo esto ayuda a que el sistema sea flexible.

- **Npm**: npm es el registro de software más grande del mundo. Los desarrolladores de código abierto de todos los continentes usan npm para compartir y tomar prestados paquetes. Npm consta de tres componentes distintos: el sitio web, la interfaz de línea de comandos *(CLI)* y el registro. [17].

- **Postgres**: es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL, similar a la BSD o la MIT [16].

- **Docker**: es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. En este sistema, los empleamos para los ejercicios prácticos entregados por los alumnos, convirtiendo el proceso más seguro debido a la ejecución aislada de cada uno de ellos [20].

3.1.2 Arquitectura

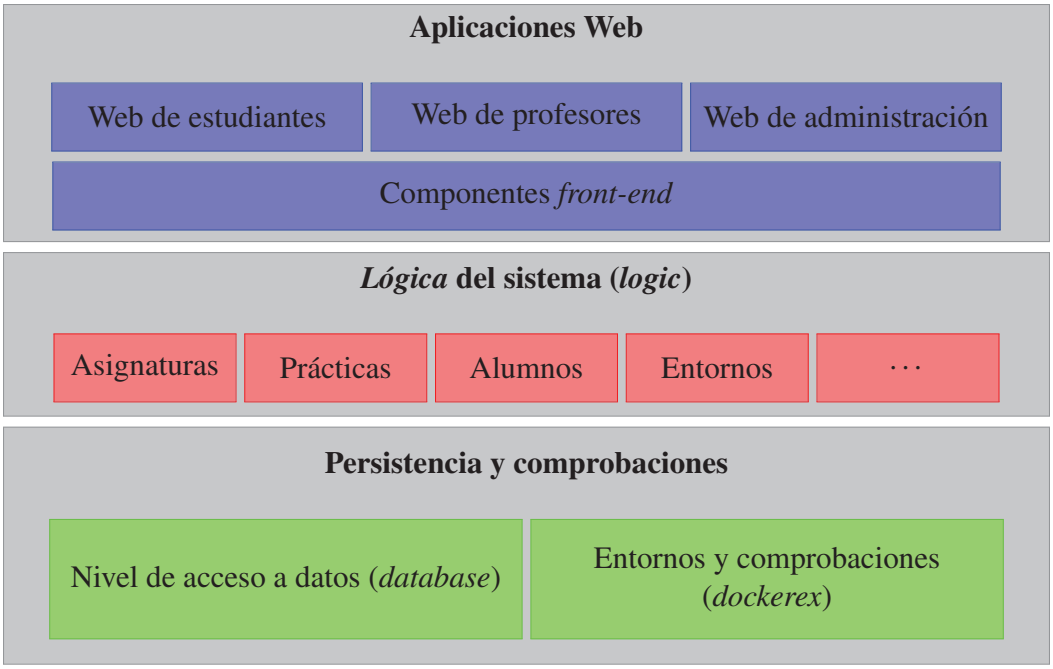

Figura 3.1: Arquitectura de Deliverit

La figura 3.1 muestra la arquitectura del sistema en diferentes niveles. Las conexiones de los diferentes componentes arquitectónicos se realiza principalmente en forma de llamadas a bibliotecas Erlang/Elixir y la figura 3.2 muestra el diagrama de componentes de Deliverit. Explicaré a lo largo de esta sección cada uno de los puntos claves.

Cabe destacar que el sistema se divide en dos partes/entidades, alumnos y profesores, estando mi trabajo más focalizado en los primeros. El sistema debía ser capaz de persistir los distintos ficheros de los ejercicios prácticos entregados por cada uno de los alumnos, correspondientes a distintos grupos de asignaturas, de tal manera que dichos ejercicios prácticos se puedan emplear en las distintas fases establecidas para cubrir las necesidades, como son comprobación, validación y descarga realizada por los alumnos.

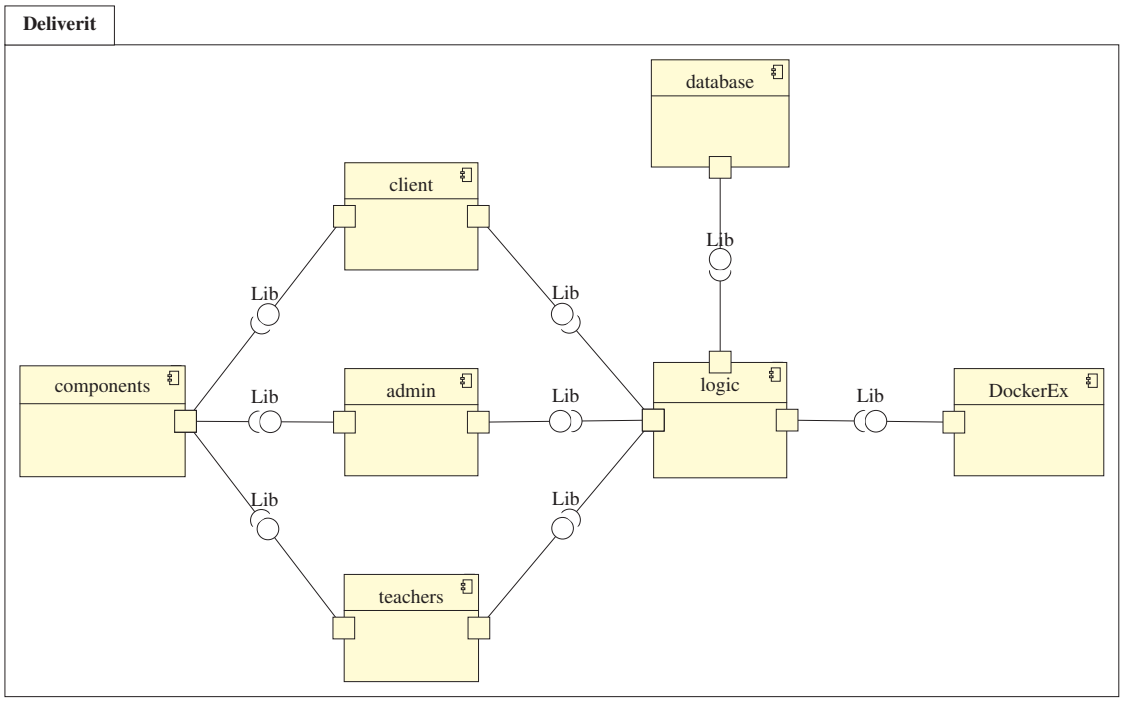

Figura 3.2: Diagrama de componentes de Deliverit

Por otro lado, la ejecución de dichos ejercicios prácticos se realiza en un entorno independiente, inaccesible y transparente para los estudiantes, con la cualidad de ser configurable para cada tipo de práctica, dado que no todas exigen los mismos requisitos ni lenguajes. Como todos bien sabemos, siempre hay un periodo en la entrega de prácticas próximo a la finalización del tiempo de entrega de esta misma, en la que los alumnos realizan más entregas para la comprobación de estas mismas, lo cual supone una carga mayor para el sistema, pudiendo saturarse. Para ello se estableció la utilización de una cola, de tal forma que las entregas realizadas por los distintos grupos se van añadiendo a la cola para ser evaluadas.

Si nos fijamos en la figura 3.1 *Diagrama de componentes de Deliverit* nos encontramos con varias clases/aplicaciones como son *Components, client, admin, teachers, database, logic y Dockerex*. En las siguientes líneas de lectura del presente trabajo explico cada una de ellas y la conexión entre las mismas.

Como he comentado anteriormente Phoenix te permite crear aplicaciones. Para el desarrollo de este sistema se establecieron bajo el dominio de Umbrella cinco aplicaciones, correspondientes a *Components, client, admin, database y logic* respectivamente, que se pueden encontrar en el directorio `/deliverit/apps` del sistema.

Comenzando de izquierda a derecha presento lo siguiente:

- **Components**: sin exponerse como frontal web independiente, proporciona assets estáticos y componentes web reutilizables para las aplicaciones Admin y Client. Como he comentado anteriormente, el proyecto divide dos entidades, la de los profesores y la de los alumnos. Los alumnos podrán asociarse a la aplicación Client, los administradores del sistema en la aplicación Admin y los profesores que empleen este sistema en teachers.

- **Admin, Client y Teachers**: Como tal, en los directorios desarrollados en el sistema no os encontrareis con una aplicación "Assistants", se expone en la arquitectura para el entendimiento de la separación de entidades. Por el contrario, los directorios que si podréis encontrar son `/deliverit/apps/admin` y `/deliverit/app/client` respectivamente, donde se desarrollan las interfaces del sistema, el panel de administración y el portal de entregas, con cada uno de los archivos configurables respectivamente. Dichos archivos, se respaldan en la aplicación Logic, que explico a continuación.

- **Logic**: en esta aplicación, se desarrolla toda la lógica empleada en el funcionamiento del sistema. Podríamos decir que se trata de la más importante, dado que por ejemplo es la única aplicación Phoenix que se comunica con la base de datos del sistema (aplicación Database). Una de las funcionalidades más relevantes de esta aplicación es la gestión de las entregas.

  Como hemos comentado anteriormente, el sistema debe poner en cola las entregas de los ejercicios prácticos realizados por los distintos estudiantes, de tal forma que dichos ejercicios no sean evaluados directamente. Las evaluaciones se realizan de forma independiente, para ello definieron el lanzamiento de un proceso independiente que se encarga de obtener de esa cola las entregas pendientes y posteriormente, evaluarlas.

  Dentro de esta misma aplicación, se encuentra la comunicación con Docker, para la evaluación de estas entregas. Y es aquí donde mi tarea empieza, creando un archivo de configuración en la Logica, para la conectividad mediante SSE que explico en el capitulo 5 *Conectividad SSE*

- **Database**: su funcionalidad principal es aislar los accesos a la base de Datos del sistema mediante Ecto, tanto de escritura como de lectura. Ecto es un lenguaje específico de dominio para escribir consultas e interactuar con bases de datos en el lenguaje Elixir.

  Además de esto, contiene los modelos de datos, el esquema de la base de datos y sus migraciones.

- **Dockerex**: por último, aquí encontramos todo lo relacionado con los contenedores para el despliegue y funcionamiento del sistema, donde además, encontramos una librería custom desarrollada por Andrés Mareca para cubrir las distintas necesidades de las operativas del sistema.

Además de todo esto, se cuenta con una capa de interfaces donde usuarios como los estudiantes y profesores puedan acceder al uso del sistema. Tienen presentes dos interfaces, una para la consola/panel de administración y otra para la entrega de los ejercicios prácticos de los alumnos.

3.2 Modelo de interacción

En este capítulo se expondrán los distintos diagramas de interacción de los que dispone el sistema Deliverit y de los cuales es importante tener en cuenta y entender para saber el correcto funcionamiento de este mismo, así como para entender los requisitos que se han logrado cubrir y los que faltan para un futuro.

Antes de introducir cada uno de los diagramas, es importante resaltar varias características desde el punto de vista funcional. Como he comentado, este sistema se divide en dos entidades. Por un lado, tenemos los "administradores", que tienen los permisos y operaciones de dar de alta a los alumnos, asignaturas y prácticas. Dado que pueden realizar esas operaciones, también pueden inscribir a los alumnos en las distintas asignaturas, descargar los documentos, ficheros y ejercicios prácticos entregados. Con esto último, podrán evaluar cada una de las practicas entregadas.

Por otro lado, tenemos a los "alumnos", los cuales pueden consultar las asignaturas en las que están inscritos, darse de alta en una práctica del sistema asociada a una asignatura y por consiguiente permisos para entregar los ejercicios prácticos de cada una de las respectivas asignaturas.

En cada una de las asignaturas, existirá un grupo asociado a una práctica que haya que entregar. Este concepto es importante dado que no siempre las entregas son individuales. En dichos grupos todos los miembros, deberán estar dados de alta en el sistema y por correspondiente en la asignatura asociada.

Modelo de interacción del cliente para Alumnos

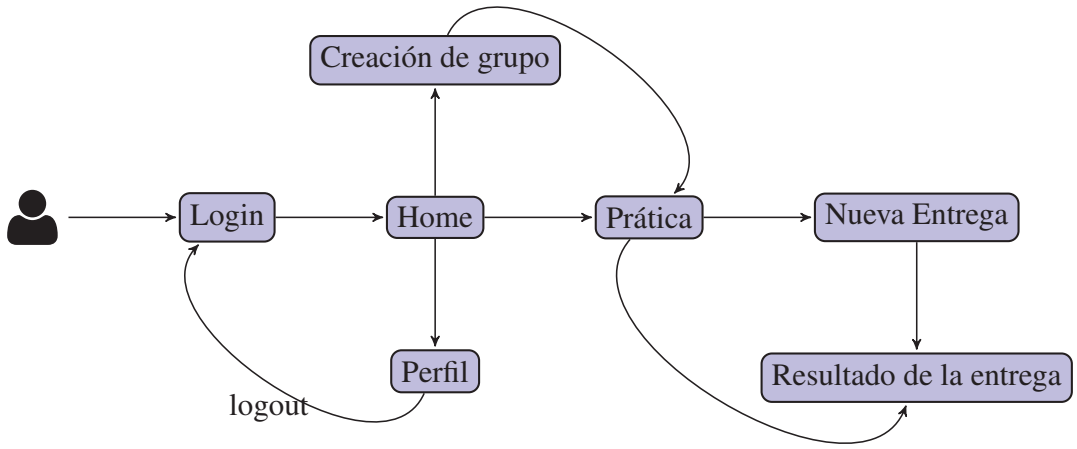

Figura 3.3: Modelo de interacción del cliente para Alumnos

El presente diagrama representa el flujo del portal de entregas, esto es desde que un usuario (más concretamente, un alumno) se *logea* es decir, inicia sesión en el sistema, accede a la plataforma, hasta las distintas opciones y operativas que puede realizar, ya sea acceder a su perfil y salir del mismo, así como crear un grupo de alumnos asociados a la entrega de una práctica que a su vez está asociada a una asignatura.

Un usuario como es el alumno no tiene los permisos necesarios para darse de alta en el sistema, esto es, el profesor es el que se encarga de registrar a los alumnos y darles de alta en las asignaturas asociadas, junto con los grupos de prácticas, si están predefinidos por el profesor con anterioridad.

Las opciones permitidas a una práctica desde el punto de vista de Client, son la realización de una nueva entrega y la obtención del resultado de esta misma. Comentando lo anterior sobre los grupos, cabe resaltar que un grupo de alumnos puede estar formado por un solo estudiante o varios, así como dicho grupo puede estar preseleccionado ya por un profesor. En este último caso el grupo tendría que generarse por un administrador a través del panel de administradores.

Modelo de interacción del cliente para profesores

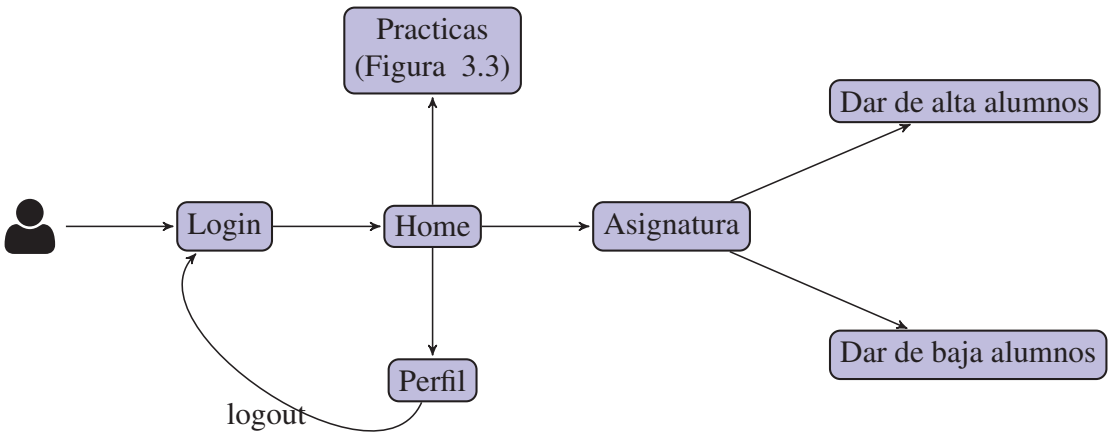

Figura 3.4: Modelo de interacción del cliente para profesores

En el caso de los profesores, se presenta el diagrama que pueden encontrar superiormente sobre el modelo de interacción del cliente para los profesores. En este se expone el flujo que debe seguir un profesor, esto es, al igual que los alumnos tienen la acción de iniciar sesión en el sistema, junto con alguna más explicada anteriormente, los profesores poseen esa misma acción, es decir, pueden iniciar sesión en el sistema con su usuario y contraseña, así como cambiar esta última.

Por el contrario, además de esto tienen ciertos permisos y acciones de las que los alumnos carecen, como puede ser administrar a los alumnos que estén matriculados en las asignaturas que estos mismos impartan. Dicha administración de los alumnos podrá realizarse mediante la carga "masiva" de alumnos. Esta misma se realizaría con un archivo csv, el cual sería capaz de dar de alta en el sistema a todos aquellos alumnos que no estén dados de alta con anterioridad, con la siguiente acción del profesor de añadirlos a las asignaturas que correspondan.

Diagrama de interacción administradores. Cliente - Alumnos

Continuamos con el diagrama de interacción para administradores, Cliente - Alumnos. En el presentamos el flujo de las operativas que se pueden realizar en cuanto a la gestión de estos mismos, viéndose claramente la creación o alta de un alumno, la edición de este mismo (ya sean sus datos, permisos o asignaturas a las que está asociado) y el Bulk de alumnos.

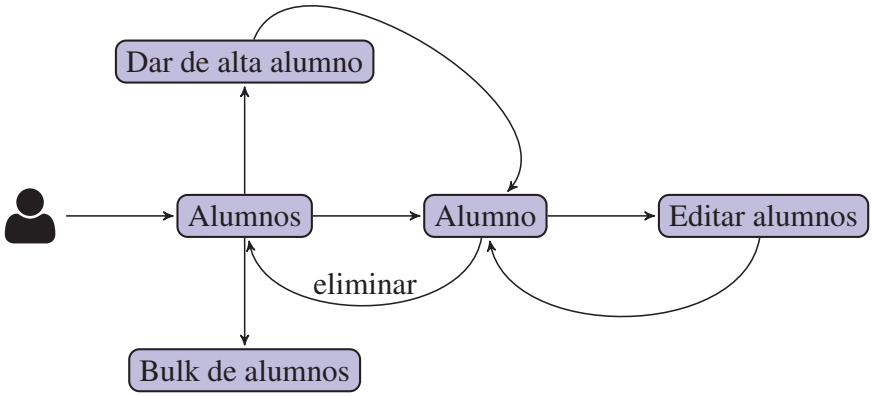

Figura 3.5: Diagrama de interacción administradores. Cliente - Alumnos

Mas detalladamente, un administrador podrá dar de alta a un estudiante, tanto individualmente o con la carga masiva de estos mismos mediante la subida de un documento csv, como ya se ha comentado anteriormente; podrá editar a un estudiante; eliminar a un estudiante; consultar los datos de un estudiante; consultar las asignaturas en las que se encuentra dado de alta un estudiante; y por último consultar los grupos a los que pertenece un estudiante.

En necesario tener en cuenta que los alumnos son usuarios del sistema, que deben estar matriculados en asignaturas. Los profesores solo podrán controlar a los alumnos que estén matriculados en las asignaturas que estos impartan.

Diagrama de interacción administradores. Entornos

En el presente diagrama de interacción para administradores, en función del Entorno, sigue el mismo patrón, esto es, la misma distribución y ciclo de vida que el de asignaturas (que veremos más adelante) y alumnos.

Durante mi estancia en el proyecto, cuando comencé, para dar de alta una práctica, simplemente la creabas y con eso ya era suficiente, pero poco después Andrés Mareca y Ángel Herranz implementaron un paso más que eran los llamados entornos, siendo obligatorio la creación de un entorno para la posterior creación de una práctica. Un entorno, básicamente es una imagen de Docker que está preparada para la ejecución de prácticas sobre estos mismos.

Como bien podemos observar las operativas disponibles en este flujo son dar de alta un entorno, es decir crearlo, así como editarlo, eliminarlo o reconstruirlo.

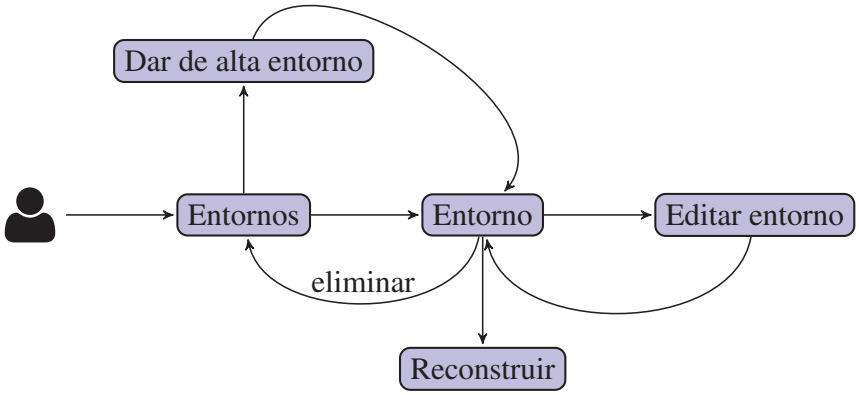

Figura 3.6: Diagrama de interacción administradores. Entornos

Diagrama de interacción administradores. Asignaturas

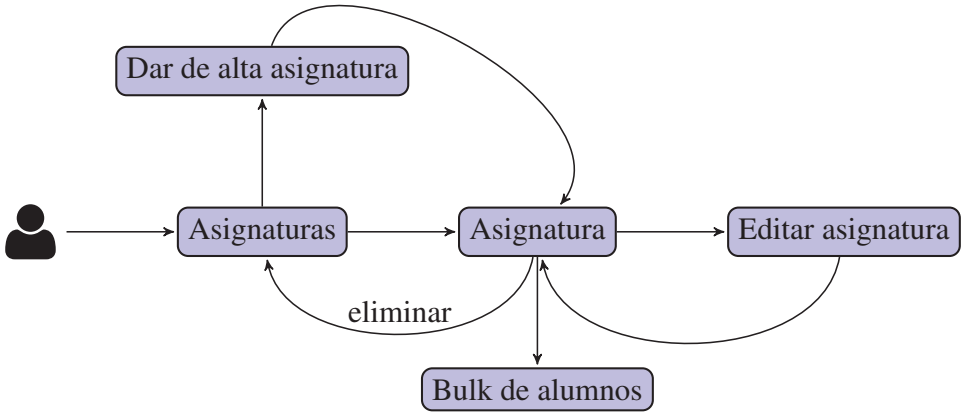

Figura 3.7: Diagrama de interacción administradores. Asignaturas

Como bien podemos observar se trata de un diagrama correspondiente al flujo de interacción para administradores en función de las Asignaturas, el cual sigue la misma estructura que el de alumnos pero correspondiendo a las distintas asignaturas de la Facultad, que empleen este sistema, y no a los alumnos.

Las operativas que se pueden realizar son dar de alta una asignatura; editar una asignatura; eliminar una asignatura; consultar los datos de una asignatura; consultar las prácticas de una

asignatura; dar de alta a un estudiante en una asignatura; dar de baja a un estudiante de una asignatura; dar de alta a múltiples estudiantes en una asignatura; dar de alta estudiantes en masa en una asignatura; dar de baja a múltiples estudiantes de una asignatura; dar de baja a todos los estudiantes de una asignatura.

Diagrama de interacción administradores. Practicas

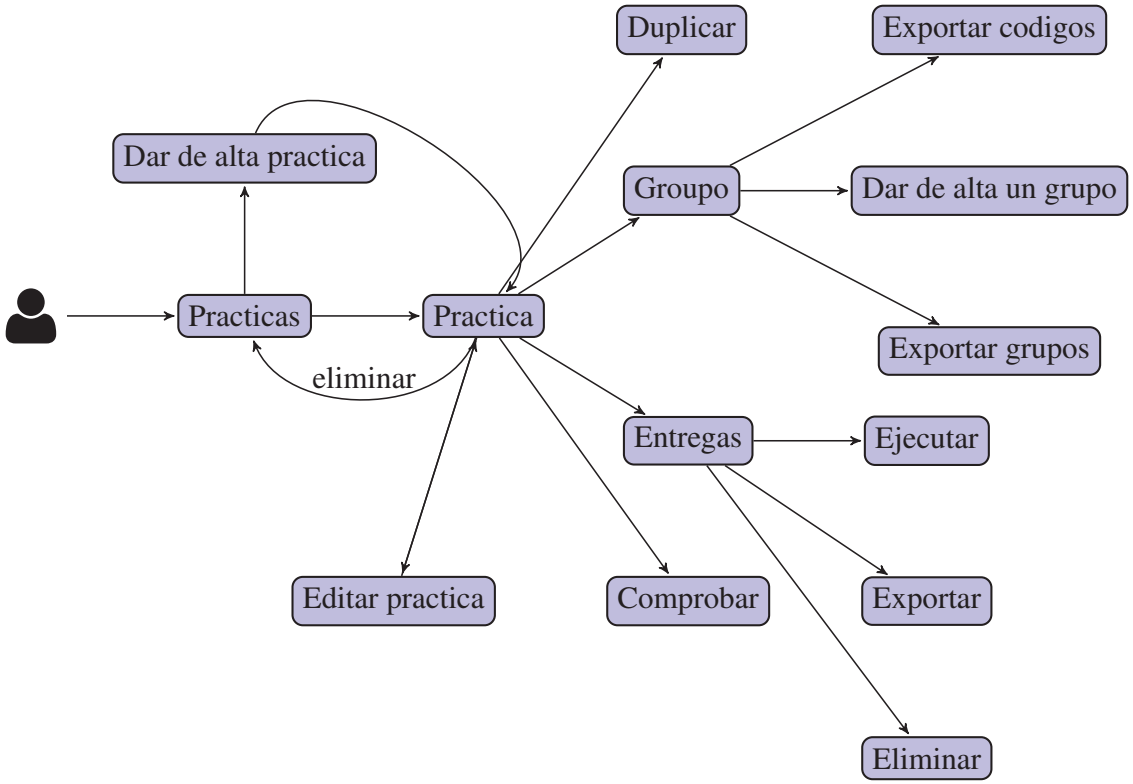

Figura 3.8: Diagrama de interacción administradores. Prácticas

Por último, tenemos el diagrama de interacción para administradores, Practicas. Se trata del diagrama más complejo de todos, dado que requiere muchas características, permisos, operativas y necesidades.

Para comenzar, a raíz del usuario como administrador, en el primer "mini" flujo, observamos que podremos realizar las siguientes operativas, dar de alta una práctica para una asignatura; editar una práctica; eliminar una práctica; duplicar una práctica; así como comprobar una práctica.

3.2 Modelo de interacción

Posteriormente tenemos dos operativas cuyas funcionalidades son más amplias y estas son cuando hablamos de un grupo o una entrega. Comenzare por el grupo. Las operativas permitidas en la sección grupo son exportar los códigos de las respectivas prácticas, dar de alta a un grupo, como bien comentamos anteriormente, el grupo de alumnos podía ser creado por el profesor y no por los alumnos, es en este momento donde se realizaría, y por último la exportación de los grupos existentes. Además de todo esto, en la gestión de un grupo podremos, siempre como administrador, consultar un determinado grupo, añadir información a un grupo, consultar las últimas entregas de cada grupo realizadas para una práctica, así como todas las que se hayan hecho con anterioridad.

Por otro lado, nos encontramos con las entregas, donde se permite realizar la ejecución, exportación y eliminación de una práctica. En la exportación de una entrega se refiere a la descarga de esta misma (todos sus ficheros) para una práctica, así como la descarga de todas las entregas para una práctica.

3 Deliverit

4

Extracción de información de los contenedores

En este capítulo vamos a ver la implementación que se llevó a cabo para la mejora de información en Docker, así como la implementación de una mejora en la explicación de los atributos a declarar y rellenar en la creación de una práctica, explicándolo mediante un cuadro de texto en la interfaz gráfica. También veremos la implementación que se llevó a cabo en la salida de errores de los contenedores, añadiendo algunos de ellos y cambiando expresiones.

4.1 Ayuda en pantalla “nueva práctica”

Para comenzar, expondré los pasos que se realizaron para incluir en la interfaz gráfica un cuadro de texto con la explicación de los campos a rellenar en la creación de una práctica, posterior a la creación de un entorno. Para ello, tuvimos que modificar el fichero `/apps/admin/lib/admin_web/templates/project/form.htm.eex` añadiendo el siguiente código expuesto a continuación:

```
<div class="alert alert-info container">
  <div class="row">
    <div class="col-12">
    <p>
      <span><%= gettext "Creating or updating a project" %></span>
    </p>
    <p><%= (gettext "For <strong>each project</strong> the following
        information must be provided:") %>
      <ul>
      <li><%= gettext "Is mandatory to check the 'Active' bottom in order
         to show the students the project" %></li>
      <li><%= gettext "If you don't specify a calification it becames a
         binary tester" %></li>
      <li><%= gettext "If you don't select any environment the tester only
         store the students file" %></li>
      <li><%= gettext "At the time of deffining steps in an environment you
          must be aware of: " %></li>
      <ul>
        <li><%= gettext "If you provide a regex, it would return the
           succesfully passed test of the students and the total number of
            tests" %></li>
        <li><%= gettext "If you select the 'mandatory' bottom and the steps
            execution fail, the next steps never will run" %></li>
      </ul>
      <li><%= gettext "If an environment is changed it will re-execute the
         students deliveries" %></li>
      </ul>
    </p>
    </div>
  </div>
  </div>
```

Dicho texto introducido en Inglés, lo incluiríamos posteriormente en Español en el fichero `/apps/components/priv/gettext/es_ES/LC_MESSAGES/default.po` insertando las traducciones necesarias en las líneas correspondientes. Más adelante, en la siguiente sección *4.3 Traducciones* se explican las traducciones con mas precisión. La figura 4.1 muestra la pantalla con este cuadro de texto.

Se expone todo lo comentado anteriormente, para que si en un futuro se quisiese modificar, el ususario sepa qué documento debe configurar y cuál es el procedimiento para ello.

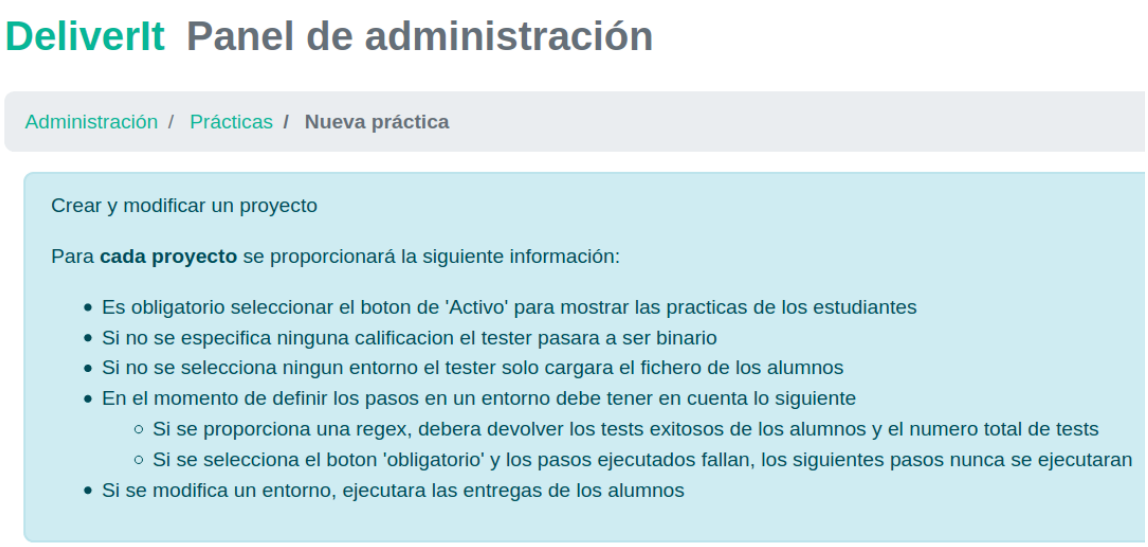

Figura 4.1: Reproducción del cambio en la pantalla de “nueva práctica”

4.2 Mejora de la información del contenedor

Además de esto, como se ha comentado al comienzo del capítulo, se reconfiguraron algunos archivos para que la salida de errores de los contenedores fuese más concreta y las expresiones no fuesen Elixir/Erlang.

Comenzaremos por la especificación de errores modificada en el archivo `project_controller.ex` situado en la ruta `/apps/admin/lib/admin_web/controllers` donde encontramos el siguiente código en el que incluímos los posibles errores que podrían surgir en la realización de una nueva práctica.

```
defp process_error(error) do
  case error do
    :invalid_template -> gettext "The template entered is invalid"
    :docker_error -> gettext "An error occurred during the generation of
       the docker image"
    :invalid_to -> gettext "To value must be higher than from value"
    :missing_base -> gettext "A base file is required for this template"
    :invalid_environment -> gettext "The environment is not valid"
    :missing_project -> gettext "A project is required for this template"
    _ -> gettext "Unexpected error"
  end
end
```

Para terminar, modificamos el archivo `build.ex` en la ruta `/apps/logic/lib/logic/environments` con el código que presento a continuación para la impresión por pantalla del error sin expresiones Elixir/Erlang.

```
error_string =
  case e do
    %MatchError{term: {:error, :request_error}} ->
      gettext "Unable to connect to Docker."
    %MatchError{term: {:error, _, error}} ->
       Poison.encode!(error)
    _ ->
      gettext "Unexpected error"
  end
```

4.3 Traducciones

Por lo general, los programas están escritos y documentados en inglés, y usan el inglés en el momento de la ejecución para interactuar con los usuarios. Esto es cierto no solo desde *Deliverit*, sino también en una gran cantidad de software propietario y libre. Usar un lenguaje común es bastante útil para la comunicación entre desarrolladores, mantenedores y usuarios de todos los países. Por otro lado, la mayoría de las personas se sienten menos cómodas con el inglés que con su propio idioma nativo, y prefieren usar su lengua materna para el trabajo diario, en la medida de lo posible. A muchos simplemente les encantaría ver que la pantalla de su computadora muestra mucho menos inglés y mucho más de su propio idioma.

Gettext es un paso importante para el Proyecto de traducción de Deliverit, ya que es un activo sobre el cual podemos construir muchos otros pasos. Este paquete ofrece a los programadores, traductores e incluso usuarios, un conjunto bien integrado de herramientas y documentación. Específicamente, las utilidades gettext de GNU [21] son un conjunto de herramientas que proporciona un marco para ayudar a otros paquetes de GNU a producir mensajes en varios idiomas.

Estas herramientas incluyen un conjunto de convenciones sobre cómo deben escribirse los programas para admitir catálogos de mensajes, un directorio y una organización de nombres de archivos para los propios catálogos de mensajes, una biblioteca en tiempo de ejecución que admite la recuperación de mensajes traducidos y algunos programas independientes para dar mensajes de varias maneras los conjuntos de cadenas traducibles, o cadenas ya traducidas.

En el caso del Sistema Deliverit, por toma de decisiones se implementó todo en Inglés,

para intentar hacerlo mas abierto y posteriormente se añadió el idioma Español, de esta forma si en un futuro el Sistema llegase a otras Escuelas, Universidades o centros de Postgrado, se podrá dar la opción de escoger el idioma Español o Inglés. Éste último sobre todo para estudiantes que no tengan nociones de Español.

Anteriormente, se ha comentado el archivo y la ruta asociada al mismo, que hay que modificar e introducir para tener ambos idiomas. A continuación se expone un ejemplo de dicho archivo de traducción, perteneciente al sistema Deliverit:

```
#: lib/admin_web/controllers/group_controller.ex:154
#: lib/client_web/controllers/project_controller.ex:140
msgid "Selection of group mates must follow some rules. Check for duplication, for example"
msgstr "La selección de companeros de grupo debe seguir algunas reglas. Comprueba por ejemplo que no hay duplicados"

#: lib/admin_web/templates/project/form.html.eex:22
msgid "At the time of deffining steps in an environment you must be aware of: "
msgstr "En el momento de definir los pasos en un entorno debe tener en cuenta lo siguiente"

#: lib/admin_web/templates/project/form.html.eex:15
msgid "Creating or updating a project"
msgstr "Crear y modificar un proyecto"

#: lib/admin_web/templates/project/form.html.eex:17
msgid "For <strong>each project</strong> the following information must be provided:"
msgstr "Para <strong>cada proyecto</strong> se proporcionará la siguiente información:"

#: lib/admin_web/templates/project/form.html.eex:27
msgid "If an environment is changed it will re-execute the students deliveries"
msgstr "Si se modifica un entorno, ejecutara las entregas de los alumnos"

#: lib/admin_web/templates/project/form.html.eex:21
msgid "If you don’t select any environment the tester only store the students file"
msgstr "Si no se selecciona ningun entorno el tester solo cargara el fichero de los alumnos"

#: lib/admin_web/templates/project/form.html.eex:20
msgid "If you don’t specify a calification it becames a binary tester"
msgstr "Si no se especifica ninguna calificacion el tester pasara a ser binario"

#: lib/admin_web/templates/project/form.html.eex:24
msgid "If you provide a regex, it would return the succesfully passed test of the students and the total number of tests"
msgstr "Si se proporciona una regex, debera devolver los tests exitosos de los alumnos y el numero total de tests"

#: lib/admin_web/templates/project/form.html.eex:25
msgid "If you select the ’mandatory’ bottom and the steps execution fail, the next steps never will run"
msgstr "Si se selecciona el boton ’obligatorio’ y los pasos ejecutados fallan, los siguientes pasos nunca se ejecutaran "

#: lib/admin_web/templates/project/form.html.eex:19
msgid "Is mandatory to check th ’Active’ bottom in order to show the students the project"
msgstr "Es obligatorio seleccionar el boton de ’Activo’ para mostrar las practicas de los estudiantes"
```

Siguiendo este proceso, si en algún momento se desease incluir más idiomas, tan solo con añadirlo a este fichero sería suficiente, consiguiendo así en un futuro y con múltiples idiomas, que el sistema Deliverit fuese comprensible por cualquier usuario.

4 Extracción de información de los contenedores

5

Conectividad SSE

En este capítulo hablaremos de mi principal aportación al proyecto: la infraestructura para poder enviar mensajes desde el *back-end* al *front-end*.

5.1 Motivación

Como se ha comentado anteriormente en la sección ***3.2 Sistema***, Deliverit cuenta con una entidad que son los alumnos. Sabemos de antemano que los alumnos tienen el permiso de subir los determinados ejercicios prácticos asociados a las asignaturas en las que están dados de alta en el sistema y, que cada vez que se carga una práctica, se lanza un contenedor Docker.

Cuando yo me incorporo al proyecto, el alumno está obligado a realizar recargas contínuas de la página hasta ver el resultado de las comprobaciones. Obviamente la experiencia de usuario es *nefasta* y además genera una carga sobre el sistema que no está justificada: cuando se acerca la fecha límite de entrega de una práctica puede haber cientos de alumnos haciendo recargas de páginas de forma compulsiva.

Por suerte, la biblioteca de Docker para Elixir (*dockerex*) desarrollada por Andrés Mareca tiene la capacidad de enviar los *logs* de forma incremental a un proceso.

La funcionalidad a implementar se basaba en mostrar *logs* del contenedor en *tiempo real* y de forma *incremental*a los alumnos. Para implementar esta funcionalidad es necesario usar

tecnologías que permitan al servidor enviar información a los clientes. Estamos hablando de cosas como *WebSockets* [22] o *Server-sent Events* [5] (SSE). A lo largo de las siguientes secciones, explicaré las decisiones que fueron tomadas, las herramientas asi como la explicación de las mismas y el proceso de desarrollo, sirviendo así este capítulo, como documentación complementaria del sistema.

5.2 Herramientas

Una vez decidido el modelo y forma en la que se llevaría a cabo esta nueva funcionalidad, era hora de ponerse al día, así como investigar las tecnologías que emplearíamos para el desarrollo de este capítulo. A continuación, se presenta una breve explicación de cada una de ellas, para posteriormente, continuar con la lectura del caso en concreto en el sistema.

- **Procesos - Elixir**: esta sección es una de las más complejas y que más me costó asentar debido al desconocimiento del lenguaje de programación, principalmente.

  En Elixir, todo el código se ejecuta dentro de los procesos. Los procesos están aislados unos de otros, se ejecutan concurrentemente entre sí y se comunican mediante el paso de mensajes. Los procesos no sólo son la base de la concurrencia en Elixir, sino que también proporcionan los medios para construir programas distribuidos y tolerantes a fallos.

  Los procesos de Elixir no deben confundirse con los procesos del sistema operativo. Éstos son extremadamente livianos en términos de memoria y CPU (incluso en comparación con los subprocesos que se usan en muchos otros lenguajes de programación). Debido a esto, no es raro tener decenas o incluso cientos de miles de procesos ejecutándose simultáneamente.

  En el capítulo *5.5 Desarrollo*, aprenderemos sobre las construcciones básicas para generar nuevos procesos, así como también para enviar y recibir mensajes entre procesos, más bien, sobre las construcciones de los procesos que tuve que desarrollar para esta funcionalidad.

- **Aplicaciones OTP**: como ya he comentado anteriormente y que ahora mismo vuelvo a remarcar, una aplicación Erlang es un grupo de código y procesos relacionados. Una aplicación OTP usa específicamente comportamientos OTP para sus procesos, y luego los envuelve en una estructura muy específica que le dice a la *VM* cómo configurar todo y luego derribarlo.

  Entonces, en el siguiente capítulo, vamos a explicar la construcción de una aplicación con componentes OTP. Ésta será sobre la implementación de un grupo de procesos (explicados en el punto anterior). La idea detrás de este grupo de procesos es administrar y limitar los recursos que se ejecutan en el sistema.

5.2 Herramientas

En Deliverit, como ya se ha comentado, se crearon varias aplicaciones OTP, las cuales cuentan con un modelo en el que participan un *Supervisor* y un *Worker*, es por ello que antes de continuar con el resto de herramientas partícipes en el desarrollo de este capítulo, se va a resaltar algunas de las características más comunes de este modelo.

Modelo Supervisor - Trabajador Para ayudarnos a diseñar una aplicación con supervisores, es útil tener una idea de lo que necesita supervisión y cómo debe ser supervisada. Una característica con la que los nuevos e incluso experimentados programadores de Erlang tienen problemas para lidiar es generalmente cómo tratar la pérdida de estado. Los supervisores matan los procesos, el estado se pierde. Para solventar esto se identifican diferentes tipos de estado:

- **Estado estático.** Este tipo de estado se puede recuperar fácilmente desde un archivo de configuración, otro proceso o el supervisor que reinicia la aplicación.
- **Estado dinámico.** Compuesto de datos que se pueden volver a calcular. Esto incluye el estado que se tuvo que transformar desde su forma inicial para llegar a donde está ahora.
- **Datos dinámicos.** Datos que no se pueden volver a calcular. Esto puede incluir la entrada del usuario, datos en vivo, secuencias de eventos externos, etc.

Los supervisores pueden manejar el estado estático, así cuando el sistema se está iniciando, etc. Cada vez que un hijo muere, el supervisor los reinicia y puede inyectarles algún tipo de estado estático, siempre disponible. Cada capa de supervisión que se agrega actúa como un escudo que protege su aplicación contra el fallo y pérdida de estado.

El estado dinámico compuesto de datos que se pueden volver a calcular tiene muchas soluciones disponibles: construirlo a partir de los datos estáticos enviados por los supervisores, ir a buscarlo a otro proceso, base de datos, archivo de texto, el entorno actual, son algunos ejemplos. Debería ser fácil recuperarlo en cada reinicio. El hecho de que tenga supervisores que realicen un trabajo de reinicio puede ser suficiente para ayudarlo a mantener vivo ese estado.

Todo tipo de operaciones relacionadas entre sí deberían formar parte de los mismos árboles de supervisión, y las no relacionadas deberían mantenerse en diferentes árboles. Dentro del mismo árbol, las operaciones que son propensas a fallo, pero no vitales pueden estar en un subárbol separado. Cuando sea posible, solo reiniciar la parte del árbol que lo necesita.

Los requisitos incluyen: poder iniciar la aplicación de grupo como un todo, tener muchos grupos y cada grupo tener muchos trabajadores que se puedan poner en cola.

Es necesario un *gen_server* por grupo. El trabajo del servidor será mantener el contador de cuántos trabajadores hay en el grupo. Por conveniencia, el mismo servidor también debe contener la cola de tareas. El servidor podría estar a cargo de pasar por alto a cada uno de los trabajadores. Necesita rastrear los procesos para contarlos y supervisarlos. Además, ni el servidor ni los procesos pueden bloquearse sin perder el estado de todos los demás, de lo contrario, el servidor no es capaz de rastrear las tareas después de reiniciarse. Una buena manera de asegurarse de que todos los trabajadores estén debidamente contabilizados sería utilizar un supervisor solo para ellos. La ventaja de hacer las cosas de esa manera es que debido a que el supervisor deberá seguir solo a los trabajadores de OTP de un solo tipo, se garantiza que cada grupo se trata de un tipo de trabajador bien definido.

Debido a que todos los grupos están bajo el mismo supervisor, un grupo o servidor determinado que se reinicia demasiadas veces en un corto período de tiempo puede eliminar todos los demás grupos. Esto significa que lo que queremos hacer es agregar un nivel de supervisión, teniendo más sentido desde la perspectiva de que todos los grupos son independientes, los trabajadores son independientes entre sí y el servidor estará aislado de todos los trabajadores.

- **Event Bus - SSE**: Desarrollar una aplicación web que use SSE (eventos enviados por el servidor) es más fácil que con los WebSockets. Se necesitará un poco de código en el servidor para transmitir eventos al *front-end*, pero el código del lado del cliente funciona casi de manera idéntica para manejar cualquier otro evento.

  Server-Sent Events (SSE) es un protocolo liviano y estandarizado para enviar notificaciones de un servidor HTTP a un cliente. A diferencia de WebSocket, que ofrece comunicación bidireccional, SSE solo permite la comunicación unidireccional del servidor al cliente.

- **Docker**: Para los presentes lectores de este trabajo fin de grado se expone en el siguiente punto, los conceptos, imágenes y contenedores de los Dockers.

Conceptos Docker Docker es una plataforma para desarrolladores y administradores de sistemas para construir, ejecutar y compartir aplicaciones con contenedores. El uso de contenedores para implementar aplicaciones se denomina contenedorización. Los contenedores no son nuevos, pero su uso para implementar aplicaciones fácilmente lo es.

La *dockerización* es cada vez más popular porque los contenedores son:

- **Flexible:** incluso las aplicaciones más complejas se pueden contener en contenedores.

- **Ligero:** los contenedores aprovechan y comparten el núcleo del host, lo que los hace mucho más eficientes en términos de recursos del sistema que las máquinas virtuales.
- **Portátil:** puede construir localmente, implementar en la nube y ejecutar en cualquier lugar, lo cual es un gran avance y optimización de tiempo.
- **Acoplado libremente:** los contenedores son muy autosuficientes y encapsulados, lo que le permite reemplazar o actualizar uno sin interrumpir otros.
- **Escalable:** puede aumentar y distribuir de forma automática réplicas de contenedor en un centro de datos.
- **Seguro:** los contenedores aplican restricciones agresivas y aislamientos a los procesos sin ninguna configuración requerida por parte del usuario.

Imágenes y contenedores Un contenedor no es más que un proceso en ejecución, con algunas características de encapsulación adicionales aplicadas para mantenerlo aislado del host y de otros contenedores.

Uno de los aspectos más importantes del aislamiento del contenedor es que cada contenedor interactúa con su propio sistema de archivos privado. Este sistema de archivos es proporcionado por una imagen Docker. Dicha imagen incluye todo lo necesario para ejecutar una aplicación: el código o binario, tiempos de ejecución, dependencias y cualquier otro objeto del sistema de archivos requerido.

Es totalmente customizable, hay imágenes creadas por defecto, las cuales simplemente llamas desde la terminal y se descargan automáticamente desde el DockerHub, como podrían ser Ubuntu, MySQL, Java, etc. Una infinidad de ellas, y otras que mediante un fichero llamado *Dockerfile* [4] puedes desarrollar tú mismo.

Contenedores vs Máquinas virtuales Ciertamente, mucha gente define un contenedor como una "Máquina Virtual". En esta sección dejaremos claras las características más relevantes de ambas.

Un contenedor se ejecuta de forma nativa en Linux y comparte el núcleo de la máquina host con otros contenedores. Es decir, ejecuta un proceso discreto, no ocupa más memoria que cualquier otro ejecutable, lo que lo hace liviano.

Por el contrario, una máquina virtual (VM) ejecuta un sistema operativo *guest* completo con acceso virtual a los recursos del host a través de un hipervisor. En general, las máquinas virtuales incurren en una sobrecarga más allá de lo que consume la lógica de su aplicación.

Además de todo esto, cabe resaltar que la librería de Docker, es una interfaz con la api de Docker, por lo que para conocer mejor que hace cada método, es necesario leerse la documentación del api de Docker [3].

5.3 Decisiones previas

Como he comentado la idea principal era utilizar WebSockets o SSE para enviar los eventos. Los eventos son un tipo de estructura que se emplea dentro de la librería *EventBus*. Esta librería implementa bus de eventos rastreables, extensibles y minimalista para Elixir. En la siguiente sección se profundizará mas en este tema. Para realizar la conexión SSE era necesario cubrir los siguientes pasos

- Guardar relación contenedor - entrega.
- Montar los *EventBus* - SSE con Phoenix.
- Conectar el Proceso, dado que cada conexión que se establezca será un proceso, con los *logs* de contenedor.

Era preferible no introducir la lógica de vincular los *logs* a un *evento_bus* dentro de la lógica de una entrega. En el capítulo ***5.5 Desarrollo*** se profundizara más en todo este proceso.

La cuestión es ¿Por qué se decidió implementar SSE y no Websockets? Bien, esta decisión fue compleja de tomar, yo misma tuve que realizar una labor de búsqueda de información, para saber cuáles nos convendrían más en la funcionalidad a desarrollar. Para que los presentes lectores de este trabajo puedan entender las diferencias más características de estas dos tecnologías, a continuación les redacto unas de las principales ventajas e inconvenientes, así como un *versus* de ambas tecnologías, por si en un futuro algún miembro se vuelve a encontrar en tal situación.

WebSockets vs SSE

Esta es en gran medida una cuestión de deuda técnica, que, en lugar de ser categóricamente un objeto *malo*, a veces puede ser apalancado y / o ahorrar tiempo a corto plazo. Los WebSockets son, sin duda, más complejos y exigentes que las SSE, y requieren un poco de información del desarrollador por adelantado. Para esta inversión, obtiene una conexión TCP *full-duplex* que es útil para una gama más amplia de escenarios de aplicación. Por ejemplo, los WebSockets tienden a ser preferibles para casos de uso como juegos multijugador o aplicaciones basadas en la ubicación. Sin embargo, SSE + AJAX [8] se puede utilizar técnicamente para lograr esto, pero podría hacer que el dominio se multiplexe ya que las solicitudes de AJAX no están realmente sincronizadas.

SSE es una solución más simple y rápida, pero no es extensible: si los requisitos de nuestra aplicación cambiaran, lo más probable es que eventualmente se necesite ser refactorizada

usando WebSockets. Aunque la tecnología WebSocket presenta un trabajo más avanzado, es un marco más versátil y extensible, por lo que es una mejor opción para aplicaciones complejas que probablemente agregarán nuevas características con el tiempo.

En nuestro caso, dado que solo lo necesitábamos para la impresión de los *logs* a los alumnos cuando entregaban una práctica, resultó más óptimo decantarnos por SSE.

A continuación, expongo una serie de ventajas e inconvenientes de cada una de las opciones:

WebSockets

Ventajas

- WebSockets ofrece comunicación bidireccional en tiempo real.
- Los WebSockets generalmente no usan 'XMLHttpRequest', y como tal, los encabezados no se envían cada vez que necesitamos obtener más información del servidor. Esto, a su vez, reduce las costosas cargas de datos que se envían al servidor.
- Las conexiones WebSocket pueden enviar y recibir datos desde el navegador. Una aplicación de chat es un buen ejemplo de una aplicación básica que podría usar WebSockets.
- WebSockets puede transmitir datos binarios y UTF-8 posibles bloques de tropiezo.

Inconvenientes

- Cuando se terminan las conexiones, los WebSockets no se recuperan automáticamente; esto es algo que necesitas implementar tú mismo y es parte de la razón por la cual existen muchas bibliotecas del lado del cliente.
- Hay que tener en cuenta que los navegadores anteriores a 2011 no admiten conexiones WebSocket.
- Algunos firewalls empresariales con inspección de paquetes tienen problemas para manejar WebSockets (especialmente SophosXG Firewall, WatchGuard, McAfee Web Gateway).

5 Conectividad SSE

SSE

Ventajas

- Transportado a través de HTTP simple en lugar de un protocolo personalizado.
- Se puede rellenar con Javascript para reforzar SSE a los navegadores que aún no lo admiten.
- Soporte incorporado para reconexión e id de evento.
- No hay problemas con los firewalls corporativos que realizan la inspección de paquetes.
- Útil para aplicaciones que permiten la comunicación unidireccional de datos, por ejemplo, precios de acciones en vivo.

Inconvenientes

- SSE está limitado a UTF-8 y no admite datos binarios.
- SSE está sujeto a limitaciones con respecto al número máximo de conexiones abiertas. Esto puede ser especialmente doloroso al abrir varias pestañas, ya que el límite es por navegador y se establece en un número muy bajo.
- SSE es mono-direccional.

5.4 Infraestructura *Server to Client*

Server-Sent Events (SSE) es un protocolo liviano y estandarizado para enviar notificaciones de un servidor HTTP a un cliente. SSE solo permite la comunicación unidireccional del servidor al cliente. SSE tiene las ventajas de ser mucho más simple, confiar solo en HTTP 1.1 y ofrecer una semántica de reintento en conexiones interrumpidas por el navegador.

A lo largo de esta sección, el presente lector encontrará aspectos básicos para la instalación y funcionamiento del servicio. Para comenzar, hablaremos sobre la instalación. El paquete se puede instalar agregando SSE a la lista de dependencias en `mix.exs`

```
def deps do
  [
    {:sse, "~> 0.4"},
    {:event_bus, ">= 1.6.0"}
  ]
end
```

Se recomienda utilizar la última versión de la librería *event_bus* [10]. Además, es necesario asegurarse de que la aplicación *event_bus* se inicie antes que la librería SSE.

La librería *EventBus* es una implementación de bus de eventos rastreable, extensible y minimalista para Elixir con una tienda de eventos integrada y un observador de eventos basado en ETS. Permite pub / sub básico para la comunicación interna del proceso. Está diseñado para poca huella de memoria y velocidad.

EventBus viene con una solución eficiente para la comunicación interna de procesos. A diferencia de otras implementaciones de *EventBus*, *EventBus* no entrega los datos del evento directamente a sus suscriptores. Ofrece el identificador del evento y el nombre del tema que se puede poner en cola con una huella de memoria mínima. Con este comportamiento, los suscriptores consultan los datos del evento cuando están listos para procesar el evento.

Para desarrollar una aplicación web que use eventos enviados por el servidor *(SSE)* se necesitará un poco de código en el servidor para transmitir eventos al *front-end*. Esta es una conexión unidireccional, por lo que no puede enviar eventos desde un cliente a un servidor.

La API de *event* enviada por el servidor está contenida en la interfaz *EventSource*; para abrir una conexión con el servidor para comenzar a recibir eventos del mismo, es necesario crear un nuevo objeto *EventSource* con la URL de un script que genera los eventos:

```
const evtSource = new EventSource("sse.html");
```

Si el script del generador de eventos está alojado en un origen diferente, se debe crear un nuevo objeto *EventSource* con la URL y un diccionario de opciones. Por ejemplo, suponiendo que el script del cliente está en `example.com`:

```
const evtSource = new EventSource("//api.example.com/sse.html", {
   withCredentials: true } );
```

Una vez que se haya creado una instancia de su fuente de eventos, puede comenzar a escuchar los mensajes del servidor adjuntando un controlador `handler` para el evento del mensaje. También puede escuchar eventos con `addEventListener ()`.

SSE.Chunk

Para enviar fragmentos de eventos al cliente, se debe crear una estructura de datos `SSE .Chunk` y una estructura de datos de eventos para entregar los eventos. A continuación se exponen los distintos atributos configurables para ambas estructuras, esta vez, nosotros no hemos implementado todas ellas dado que no era necesario, pero se exponen por si en un futuro se necesitasen.

`Chunk`, tiene los siguientes atributos y solo se requiere el atributo de datos, el resto de los atributos son opcionales:

- ***Comment***: la línea de comentarios se puede usar para evitar que las conexiones se agoten; un servidor puede enviar un comentario periódicamente para mantener viva la conexión. Hay que tener en cuenta que el paquete SSE mantiene la conexión activa, no tiene que enviar comentarios.
- ***Data***: refiere al campo de datos para el mensaje. Cuando EventSource recibe múltiples líneas consecutivas que comienzan con datos:, las concatenará, insertando un carácter de nueva línea entre cada una. Se eliminan las nuevas líneas finales.
- ***Event***: se trata de una cadena que identifica el tipo de evento descrito. Si se especifica esto, se enviará un evento en el navegador al oyente para el nombre del evento especificado. Pr otro lado el código fuente del sitio web debe usar `addEventListener ()` para escuchar eventos con nombre. Por último, se llama al controlador `onmessage` si no se especifica ningún nombre de evento para un mensaje.
- ***Id***:consta del id del evento para establecer el último valor del id de evento del objeto*EventSource*.
- ***Retry***: particularmente se trata del atributo que guarda el tiempo de reconexión que se debe usar al intentar enviar el evento. Debe ser un número entero, especificando el tiempo de reconexión en milisegundos. Si se especifica un valor no entero, el campo se ignora.

  Ejemplo de preparación de datos:

```
Chunk =%SSE.Chunk{
  comment: String.t() | nil,
  data: [String.t()],
  event: String.t() | nil,
  id: String.t() | nil,
  retry: integer() | nil
}
```

Event

Para entregar *chunks*, debe notificar una estructura `%EventBus.Model.Event{}` al *topic* deseado.

La librería *event_bus* emite todos los eventos con nombres de temas, a los cuales nos referiremos como *topics*. El nombre del tema es simplemente una variable de tipo átomo. Por ejemplo; si desea emitir un evento después de crear usuarios, entonces `:user_created` podría ser un buen nombre para el *topic* del evento. Antes de entregar estos eventos para el *topic* `:user_created`, se deberá registrar el nombre del *topic* (`:user_created`) una vez en el ciclo de vida de la aplicación.

La librería *event_bus* permite crear / registrar temas de dos maneras:

- Usando el archivo de configuración de su proyecto `config.exs` (recomendado)
- Uso de la función `register_topic/1` (bajo demanda)

Una estructura de evento puede tener al menos tres valores:

- ***Id***: identificador de evento único *(entero | String.t)*
- ***data***: datos de fragmentos *(SSE.Chunk.t)*
- ***topic***: Nombre del topico para entregar el evento *(átomo)*

  Preparación de datos de muestra

```
Chunk =%SSE.Chunk{data: "algunos datos"}
event =%EventBus.Model.Event{id: UUID.uuid4 (), data: chunk, topic::
   a_topic_name}
```

Finalmente para terminar la configuración, es necesario añadir en el archivo `config.exs`, lo siguiente:

```
config :mime, :types, %{
  "text/event-stream" => ["text"]
}
```

5.5 Desarrollo

Una vez aclaradas las decisiones tomadas, herramientas y tecnologías empleadas, podemos pasar a la fase de desarrollo de la esperada funcionalidad.

5 Conectividad SSE

Teníamos claro que lo que teníamos que implementar y los pasos a seguir. Dado que *EventBus* me devuelve el tipo de mensaje cuando me suscribo a un log, lo primero que debíamos hacer era conectarnos mediante SSE y comunicar que me suscribo a todos los logs. Posteriormente, en función del parámetro *Event*, decidir el cuadro de texto del *front* en el que lo escribo. Cabe resaltar que todo se trata de paso de mensajes.

Para todo esto, se tuvo que modificar la aplicación Logic, para la implementación de la conexión mediante SSE, la aplicación Client, para imprimir el cuadro de texto en el *front-end* y por último la aplicación Components, que siguiendo la ruta del archivo `/apps/components/lib/components_web/views/molecules.ex` se modifica la función `submission_logs` donde se introduce los logs y url donde ejecutar el script.

Posteriormente, siguiendo la ruta `/apps/client/lib/client_web/templates/submission` donde encontramos el archivo `wait.html.eex`, tuvimos que modificar varias cosas, dado que era necesario para la correcta implementación. Es en este archivo donde configuramos la salida en el *front* a los alumnos, es decir donde "pintamos" los contenedores que son donde están conectados los distintos logs.

Por último, se tuvo que generar un archivo llamado `scripts_wait.html.eex` en la ruta `/apps/client/lib/client_web/templates/submission` en el cual introducimos la función que realiza la conexión SSE, configurándola de tal forma que según nos vaya llegando el texto se lo vamos añadiendo al cuadro de texto que saldrá por pantalla. El siguiente código se corresponde con dicho fichero:

```
<script type="text/javascript">
  const eventSource = new EventSource("<%= Routes.submission_url(@conn, :
     logs, @project.id, @submission.id) %>");

  function event_manager(event) {
    console.log(event);
    var container = $( "#log_output");
    container.html(container.html() + event.data + "<br/>");

}
  <%= for %{id: name} <- @project.steps do %>
     eventSource.addEventListener("<%= name %>", event_manager);
    <% end %>
</script>
```

Cabe resaltar que uno de los archivos más importantes a desarrollar fue el `sse.ex` localizado en la aplicación de la lógica del sistema, es decir en el directorio `/apps/logic/lib/logic/submissions` del sistema. En este fichero creamos 7 métodos distintos, cuyo código asociado se expone a continuación.

```elixir
defmodule Logic.Submissions.Sse do
  use GenServer
  alias SSE.Chunk
  alias EventBus.Model.Event

  def start_link(id_practica_entrega) do
    GenServer.start_link(__MODULE__, [id_practica_entrega])
  end

  def init([id_practica_entrega]) do
    EventBus.register_topic(id_practica_entrega)
    {:ok, %{id: id_practica_entrega, name: nil}}
  end

  def handle_call({container, name}, _, ctx) do
    Dockerex.Containers.logs(container, self(), %{stdout: true})
    {:reply, :ok, %{ctx | name: name}}
  end

  def handle_info({:error, _}, ctx) do
    {:noreply, ctx}
  end

  def handle_info({:end, _}, ctx) do
    {:noreply, ctx}
  end

  def handle_info({:ok, _, data}, ctx = %{name: name, id: id}) do
    chunk = %Chunk{data: data, event: name}
    event = %Event{id: Ecto.UUID.generate(), data: chunk, topic: id}
    EventBus.notify(event)
    {:noreply, ctx}
  end

  def terminate(_, %{id: id}) do
    EventBus.unregister_topic(id)
  end
end
```

A continuacion se presenta una pequeña descripcion de cada uno de los metodos desarro-

llados en el codigo:

- **def** `start_link`: donde iniciamos toda la aplicacion, con el id de la practica a entregar. Inicia un proceso GenServer vinculado al proceso actual, inicializamos un link correspondiente al id de la practica.

  Esto se usa a menudo para iniciar *GenServer* como parte de un árbol de supervisión.

  Una vez que se inicia el servidor, se llama a la función `init/1` del módulo dado con `init_arg` como argumento para inicializar el servidor. Para garantizar un procedimiento de inicio sincronizado, esta función no regresa hasta que `init/1` haya regresado.

  Un *GenServer* [11], es un proceso que invoca un conjunto limitado de funciones en condiciones específicas. Es la manera que tenemos de comunicarle a la máquina virtual que tiene que arrancar un *thread* con el código del fichero

- **def** `init`: donde, mediante la tecnología de los `event_bus`, registramos el *topic* del id de una práctica. Este método, es el que se ejecuta para iniciar el proceso, es decir se invoca cuando se inicia el servidor. `start_link` se bloqueará hasta que regrese.

- **def** `handle_call`: método en el que hacemos referencia a la librería de Andres Mareca para suscribirnos a los distintos logs. Se trata de un método síncrono, utilizado para hacer llamadas que esperan respuestas desde un código. Requiere que un usuario realice una llamada y espere respuesta.

  `call/3` se bloqueará hasta que se reciba una respuesta (a menos que se agote el tiempo de la llamada o se desconecten los nodos).

  Existen tres casos de uso principales para no responder utilizando el valor de retorno:

  - Para responder antes de regresar de la devolución de llamada porque la respuesta es conocida antes de llamar a una función lenta.
  - Para responder después de regresar de la devolución de llamada porque la respuesta aún no está disponible.
  - Para responder desde otro proceso, como una tarea. Al responder desde otro proceso, el *GenServer* debería salir si el otro proceso sale sin responder ya que la persona que llama estará bloqueando en espera de una respuesta.

- **def** `handle_info`: el código continúa con tres `handle_info` correspondientes al caso `error` y **end**, en cuyo caso deben volver a la suscribcion de logs del sistema, y uno ultimo correspondiente a la configuración `chunk e event` asociada al SSE, donde se notifica el evento. Lo empleamos para la comunicación entre los distintos procesos, para el envío de mensajes entre ellos.

Este método consta de dos parámetros de entrada, el primero se corresponde con `msg`, esto es el mensaje y el segundo parámetro `state` es el estado actual del *GenServer*. Cuando se produce un tiempo de espera, el mensaje es: tiempo de espera.

- **`def terminate`**: donde se termina la conexión mediante SSE y se desregistra del tópico asociado al id de la práctica. Terminación exitosa del proceso.

  Se invoca cuando el servidor está a punto de salir. Debería hacer cualquier limpieza requerida. El motivo es el motivo de salida y el estado es el estado actual del *GenServer*. El valor de retorno es ignorado.

Dado que el presente trabajo fin de grado tiene como finalidad también realizar la función de documentación del proyecto, se dispone una cita de *GenServer* [12] además de la expuesta anteriormente, para futuros estudiantes que tengan más curiosidad o lo necesiten como apoyo de formación, en caso de tener que realizar algo similar.

Continuando con el desarrollo, además de esto, dado que el sistema contaba con un `Worker` fue necesario realizar determinadas configuraciones en dicho fichero, así como en el `runner` añadiendo los parámetros extras necesarios y el link al SSE:

```
...
{:ok, pid} = Sse.start_link(submission.id)
...
```

Se añadio tambien una ruta en el fichero router del sistema:

```
get(
      "/:project_id/submission/:id/logs",
      SubmissionController,
      :logs
    )
```

Así como las determinadas dependencias y líneas de configuración en los ficheros `config` y `mix.exs` que se han comentado anteriormente en la sección ***5.4 Infraestructura 'Server to Client'***.

5.5.1 Extracción de información de Docker

En esta tarea hay una función clave, que gracias a mi compañero Andrés Mareca, desarrollador con más experiencia en este lenguaje y gran mundo de Erlang y Elixir, pudimos resolver mediante azúcar sintáctico y sin dar más vueltas de las necesarias. Para ser más precisos, más que una función de trata de una línea que marca la diferencia con el resto del código, *linea 3*.

```
1 def handle_call({container, name}, _, ctx) do
2     Dockerex.Containers.logs(container, self(), %{stdout: true})
3     {:reply, :ok, %{ctx | name: name}}
4   end
```

Con la ya comentada *linea 3* del código superior, lo que hacemos es un "proxy de un proxy" donde este primero proviene de un proxy de los logs y envía un proxy de eventos, tomando así, los logs del contenedor.

5.5.2 Pruebas

Para continuar con esta sección, daré una serie de ejemplos y pasos a seguir para realizar las pruebas de conexión mediante SSE, en local y posteriormente un ejemplo en el sistema.

Prueba Local

El primer paso, como es natural, es que el servicio de contenedores este levantado y funcionando. Por si el presente lector no tiene los conocimientos necesarios para ello a continuación se expone el comando a ejecutar en la consola para su funcionamiento:

```
$ sudo service docker start
```

Posteriormente, será necesario abrir tres terminales. Como recomendación, el programa *Terminator* en Ubuntu es bastante eficaz y útil para esto, te permite en una misma terminal, dividir la pantalla tantas veces como se desee y en vertical u horizontal. En la primera de ellas levantaremos el sistema Deliverit. Si se es nuevo en la instalación del Sistema, será preciso seguir los pasos documentados en el *README* del proyecto. En la segunda, levantaremos un contenedor cualquiera, para poder comprobar que lo que realicemos dentro de él, salga por pantalla. Por ejemplo, un contenedor Ubuntu:

```
$ docker run -it ubuntu:latest bash
```

Una vez tenemos levantado el contenedor, realizamos los siguientes pasos en la consola *iex* del sistema Deliverit que tenemos funcionando.

```
iex(2)> Dockerex.Containers.list
iex(3)> Dockerex.Containers.logs("
   a4c453ee83hrbtd8be69d9603f28102779acc8b02f3e8c38bc8ea7e", self(), %{
   stdout: true})
iex(4)> {:ok, pid}=Logic.Submissions.Sse.start_link("1234")
iex(5)> GenServe.call(pid,{"",""})
```

Donde con el primer comando, comprobamos la lista de los contenedores que estan activos en ese instante; con el segundo imprimimos los logs del contenedor al que nos estemos refiriendo, esto es, el primer *String* es el ID del contenedor que hayamos escogido en la lista de contenedores del comando anterior; posteriormente inicializamos el link *Sse* y finalmente lanzamos la llamada *GenServe*.

Seguido de esto, en la tercera terminal que nos queda por utilizar, realizamos la siguiente llamada `curl`:

```
$ curl -v http://localhost:4000/123/submission/1234/logs
```

Donde nos imprimirá, todo lo que realicemos en el contenedor Ubuntu que levantamos posteriormente. Es divertido a la par que curioso realizar estas pruebas, además de totalmente recomendable para entenderlo mejor.

Prueba con el Sistema Deliverit

Dicho todo esto, paso a mostrar como quedaría esta impresión por pantalla de los logs del contenedor en una entrega de una práctica en el sistema Deliverit real.

5 Conectividad SSE

Figura 5.1: Impresion logs contenedor. Etapa *Compile*

test

```
Welcome to Gradle 5.4.1!

Here are the highlights of this release:
 - Run builds with JDK12
 - New API for Incremental Tasks
 - Updates to native projects, including Swift 5 support

For more details see https://docs.gradle.org/5.4.1/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

> Task :test

LibraryTest > testSomeLibraryMethod1 PASSED

LibraryTest > testSomeLibraryMethod2 PASSED

LibraryTest > testSomeLibraryMethod3 PASSED

LibraryTest > testSomeLibraryMethod PASSED

------------------------------------------------------------------------
|  Results: SUCCESS (4 tests, 4 successes, 0 failures, 0 skipped)  |
------------------------------------------------------------------------

BUILD SUCCESSFUL in 8s
3 actionable tasks: 3 executed
[0m[?12l[?25h
```

Figura 5.2: Impresion logs contenedor. Etapa *Test*

5 Conectividad SSE

También se muestra la siguiente imagen que presenta una captura del sistema en tiempo real:

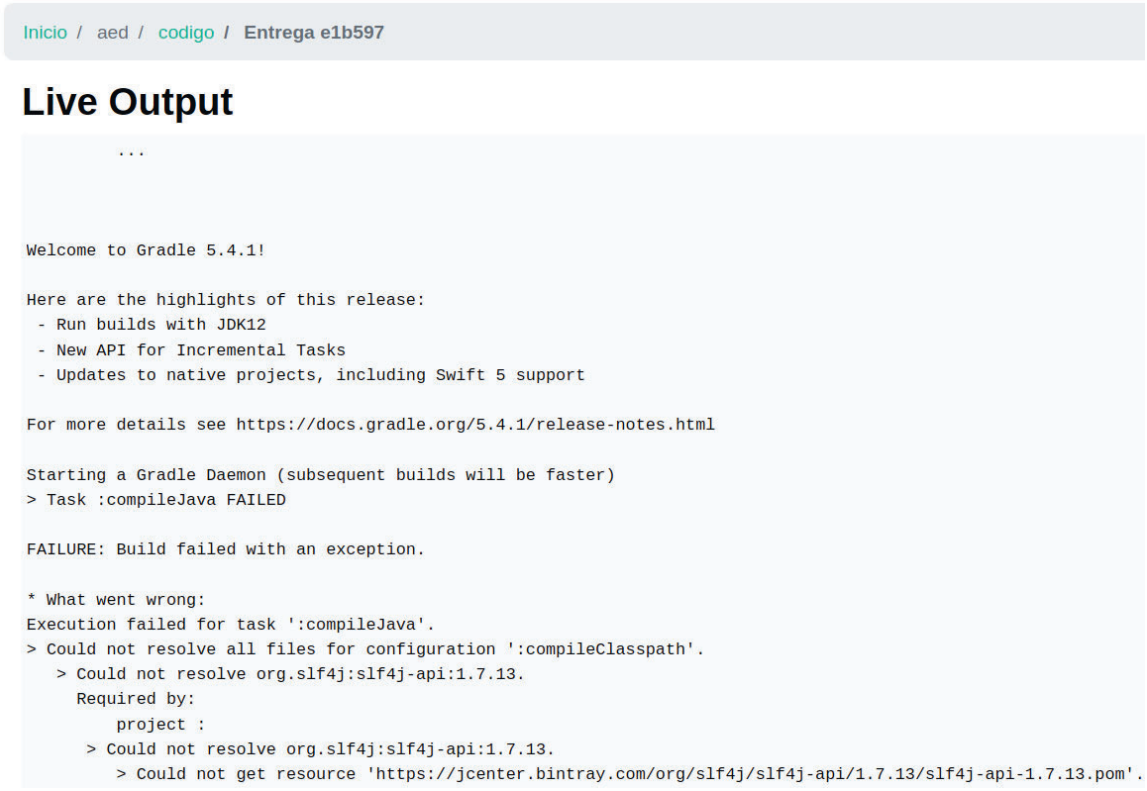

Figura 5.3: Impresion logs contenedor. Tiempo real.

6

Conclusiones

Llegamos al último apartado de la memoria, donde los presentes lectores podrán observar los puntos claves de este proyecto así como el futuro trabajo que se deberá seguir realizando en el sistema para que sea cada vez más potente y completo, pudiendo así llegar a otras escuelas y Universidades.

El presente trabajo de fin de grado implementa el soporte y continuo desarrollo de nuevas funcionalidades a un sistema de entrega de prácticas que, existente en una versión beta, ha sido ya utilizado por el profesorado de la ETSIINF.

Actualmente dicho sistema lo componen por el momento dos aplicaciones Web, correspondientes al portal de alumnos, donde se realizan las entregas y el panel de administración, donde se presentan a los administradores. El trabajo ha abarcado por un lado una parte de formación, siendo esta el entendimiento del sistema, arquitectura, funcionalidades, flujos de los modelos de interacción, así como el aprendizaje de un nuevo lenguaje de programación y herramientas como son Elixir, Phoenix, Node.js, Postgres; y por otro la implementación de nuevas funcionalidades, así como la mejora de algunas antiguas.

En el ámbito del desarrollo, mis principales aportaciones al proyecto a través del presente trabajo fin de grado han sido:

- Investigación servidores OAuth que nos pueda proporcionar la Universidad.
- Muestra de los logs de los contenedores en tiempo real. Conectividad SSE.

- Impresión por pantalla de dichos logs en la interfaz gráfica para los alumnos.
- Mejora en la información de los campos a rellenar en la creación de una práctica. Interfaz Gráfica.
- Ampliación de errores Docker en la salida de una nueva práctica. Interfaz Gráfica.

Además de las actividades de desarrollo que he podido aportar para las nuevas funcionalidades, otra parte del trabajo ha sido documentar desde el punto de vista del entendimiento la arquitectura y distribución del sistema, para que cuando futuros alumnos colaboren en el proyecto, tengan documentación sobre que hace el sistema, como está distribuido; y en cuanto a lo que he desarrollado yo personalmente, en que directorios pueden localizarlo y manejarlo. Con esto, espero poder ayudar a dichos futuros alumnos a comprender mejor el sistema desde cero y optimizar ese tiempo para que puedan comenzar antes con sus tareas de desarrollo.

6.1 Trabajo futuro

Pruebas. Queda como propuesta elaborar tests unitarios y tests de integración utilizando las herramientas que ofrece Phoenix/Elixir. Además, resulta imprescindible que se hagan pruebas con prácticas reales para comenzar a someter al sistema a cierto estrés.

Integración. Es importante estudiar los puntos de integración del sistema con otros sistemas. Detallamos una lista de los que consideramos fundamentales para mejorar la experiencia de usuario de alumnos, profesores y administradores:

- Integración con servicios de directorio de la universidad, como por ejemplo LDAP: `https://www.etsiinf.upm.es/?pagina=1521`.
- Integración con servicios web de la universidad, como por ejemplo apiUPM: `https://www.upm.es/apiupm/webServices.html`.
- Integración con servidores de fichero remotos para almacenar los artefactos fruto de las entregas, como por ejemplo UPMdrive del CESVIMA o S3 en AWS.
- Integración con sistemas de *hosting* de control de versiones como fuentes de códigos de comprobación y de artefactos de las entregas. Por nombrar algunos de estos sistemas: GitLab (`https://about.gitlab.com`), Gitea (`https://gitea.io`) o Gitolite (`https://gitolite.com`).

- Integración con sistemas de autenticación OAuth de la Universidad Politécnica de Madrid. podría existir la posibilidad de integrarlo, pero para ello el sistema tendría que estar operativo y no en fase de desarrollo.

6.2 Motivación

Finalmente animo a futuros estudiantes a que colaboren en el desarrollo de este proyecto, dado que trata con tecnologías muy innovadoras y fuertes.

Es una buena forma de colaborar con la universidad y desarrollar un proyecto que sabes que va a servir de ayuda, tanto a la comunidad de estudiantes, como profesores. Es una oportunidad de terminar la carrera aportando 'tu granito de arena' en una plataforma más próxima a la realidad, sirviendo en la propia experiencia de crecimiento para el estudiante.

No hay que tener miedo si no se dispone del conocimiento de todas las tecnologías, el director del proyecto Ángel Herranz, te dará pautas para ello. Recuerda la pregunta *Si no somos las nuevas generaciones, con la ayuda de los profesionales con más experiencia, las que aportamos nuestro granito de arena al desarrollo de nuevas tecnologías y herramientas ¿Quién lo hará?.*

6 Conclusiones

Bibliografía

[1] *Bootstrap*. Bootstrap. URL: https://getbootstrap.com/docs/4.5/getting-started/introduction/.

[2] M. Burrows y L. Hohmann. *Kanban from the Inside*. Blue Hole Press, 2014. ISBN: 9780985305192.

[3] *Docker Engine API (v1.40)*. Docker. URL: https://docs.docker.com/engine/api/v1.40/.

[4] *Dockerfile reference*. Docker. URL: https://docs.docker.com/engine/reference/builder/.

[5] MDN web docs. *Using server-sent events*. URL: https://github.com/mustafaturan/sse.

[6] Vincent Driessen. *Gitflow Workflow*. URL: https://nvie.com/posts/a-successful-git-branching-model/.

[7] Jon Duckett. *HTML and CSS: Design and Build Websites*. 1.ª ed. Wiley Publishing, 2014. ISBN: 1118871642.

[8] Jon Duckett. *JavaScript and JQuery: Interactive Front-End Web Development*. 1st. Wiley Publishing, 2014. ISBN: 1118531647.

[9] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003, pág. 529. ISBN: 0321125215.

[10] *EventBus*. Mustafaturan. URL: https://github.com/otobus/event_bus.

[11] *Genserver - Elixir*. Elixir. URL: https://elixir-lang.org/getting-started/mix-otp/genserver.html.

[12] *Genserver - Elixir*. Elixir. URL: https://hexdocs.pm/elixir/GenServer.html#module-example.

[13] Benjamin Tan Wei Hao. *The Little Elixir and Guidebook*. 1st. USA: Manning Publications Co., 2016. ISBN: 1633430111.

[14] Fred Hebert. *Learn You Some Erlang for Great Good! A Beginner's Guide*. USA: No Starch Press, 2013. ISBN: 1593274351.

[15] *Microsoft Teams*. URL: https://www.microsoft.com/es-es/microsoft-365/microsoft-teams/group-chat-software.

[16] Bruce Momjian. *PostgreSQL: Introduction and Concepts*. USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 0201703319.

[17] *Npm*. URL: https://www.npmjs.com/.

[18] Ken Schwaber y Mike Beedle. *Agile Software Development with Scrum*. 1st. USA: Prentice Hall PTR, 2001. ISBN: 0130676349.

[19] James Shore y Shane Warden. *The Art of Agile Development*. First. O'Reilly, 2007. ISBN: 9780596527679.

[20] James Turnbull. *The Docker Book: Containerization Is the New Virtualization*. James Turnbull, 2014. ISBN: 9780988820203.

[21] François Pinard Ulrich Drepper Jim Meyering. *GNU gettext tools: Native Language Support Library and Tools*. Ed. por Samurai Media Limited. 2015, pág. 272. ISBN: 9888381563.

[22] Dimitris Zorbas. *Phoenix WebSockets Under a Microscope*. URL: https://zorbash.com/post/phoenix-websockets-under-a-microscope/.

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Sun Jun 07 20:28:42 CEST 2020
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)