Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Grado en Ingeniería Informática

Trabajo Fin de Grado

Deliverit 1.0. Mejoras en los Entornos de Ejecución de las Prácticas

Autor: Ionel Constantin Trifan
Tutor(a): Ángel Herranz Nieva

Madrid, Junio de 2021

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Grado en Ingeniería Informática

Título: Deliverit 1.0. Mejoras en los Entornos de Ejecución de las Prácticas

Junio de 2021

Autor: Ionel Constantin Trifan
Tutor: Ángel Herranz Nieva
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Quiero agradecer la realización de este TFG a las personas que han confiado en mí, me han apoyado e impulsado para que consiguiera mis objetivos.

En primer lugar, a mis padres, por enseñarme que todo se consigue con esfuerzo y que esta carrera y este TFG no iba a ser menos.

En segundo lugar, a mi mujer, por haber estado a mi lado durante todo este proceso, por todo su apoyo y su comprensión durante las duras jornadas de estudio en temporadas de exámenes, por su paciencia y su cariño.

Y por último, y no por ello menos importante, a mi tutor del TFG Angel Herranz por haber creado este proyecto y darme la oportunidad de participar en él, por su profesionalidad en su área y por haberme guiado siempre que lo he necesitado.

Gracias por todo.

Resumen

Durante los años cursados en la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) he realizado un gran número de prácticas para diferentes asignaturas, las cuales la gran mayoría son dependientes de un sistema de evaluación propio desarrolladas en distintas tecnologías, lo que implica a su vez que por cada uno de los diferentes métodos de entrega y corrección se ha tenido que emplear una gran cantidad de esfuerzo por parte de los profesores en su desarrollo, puesta en marcha y mantenimiento. Por otro lado, los alumnos tienen que familiarizarse con cada uno de ellos, por lo que para ambas partes este método se hace algo laborioso.

Para suplir esta necesidad de optimizar las entregas de las practicas llega el Delivery System llamado Deliverit, la cual proporciona un sistema de entregas centralizado, seguro y escalable, contando con una interfaz moderna y usable, librando a los profesores de la tediosa tarea de crear y mantener un sistema propio, y permitir a los alumnos la entrega y corrección de todas las practicas requeridas para su formación en un solo lugar. El sistema se encuentra en funcionamiento, aunque por ahora en una fase beta, por lo que tanto yo como otros compañeros de la carrera, aportamos nuestro granito de arena, identificando puntos de mejora he intentado solucionar los posibles fallos encontrados.

Mi trabajo en este gran proyecto es la de mejorar la presentación de los resultados durante la ejecución de las correcciones de las practicas, intentando optimizar el impacto en el sistema que tiene la solución actual además de mejorar la experiencia de usuario.

A lo largo de este documento se detallará todo lo relacionado con este sistema, desde las tecnologías utilizadas en su desarrollo hasta su arquitectura, además de detallar las posibles alternativas disponibles y la solución finalmente implementada para el problema descrito.

Abstract

During the years studied at the School of Computer Engineering (ETSIINF) I have done a large number of practices for different subjects, which the vast majority are dependent on a proprietary evaluation system developed in different technologies, which in turn implies that for each of the different methods of delivery and correction has had to use a lot of effort by teachers in their development, implementation and maintenance. On the other hand, the students have to familiarize themselves with each of them, making this method somewhat laborious for both parties.

To meet this need to optimize the delivery of practices comes the Delivery System called Deliverit, which provides a centralized, secure and scalable delivery system, with a modern and usable interface, freeing teachers from the tedious task of creating and maintaining their own system, and allowing students to deliver and correct all the practices required for their training in one place. The system is up and running, although for now in a beta phase, so both I and other colleagues in the career, we contribute our bit, identifying areas for improvement and trying to solve the possible failures found.

My work in this great project is to improve the presentation of the results during the execution of the corrections of the practices, trying to optimize the impact on the system that has the current solution in addition to improving the user experience.

Throughout this document I will detail everything related to this system, from the technologies used in its development to its architecture, as well as detailing the possible alternatives available and the solution finally implemented for the described problem.

Tabla de contenidos

Bibliografía **35**

Capítulo 1

Introducción

En la actualidad la mayoría de las asignaturas de la Escuela Superior de Ingenieros Informáticos (ETSIINF), para evaluar los conocimientos impartidos cuentan con una o varias prácticas, las cuales son dependientes de un sistema propio de entrega y evaluación, esta situación hace que tanto para los alumnos, que tienen que familiarizarse con cada uno de los sistemas de entrega, como para los profesores sea una tarea laboriosa, y un tanto tediosa.

Vista la creciente necesidad de optimizar las entregas de los trabajos y prácticas realizadas por los alumnos de la escuela y facilitar a los profesores entornos seguros, eficientes y estables para la evaluación de dichas prácticas, desde la unidad de programación de la ETSIINF, se ha elaborado un sistema de entregas único, contando a su vez con una interfaz moderna y usable, tanto para los alumnos, así como para los profesores. Por lo que, mediante la centralización de los sistemas de entrega en uno solo, todas las practicas de las distintas asignaturas pueden ser entregadas y evaluadas, reduciendo el tiempo y el trabajo requerido para esta tarea, para todas las partes involucradas.

Este sistema utiliza una tecnología muy innovadora, que brinda una serie de ventajas tanto en su ejecución como en la escalabilidad. Para su desarrollo se ha optado por el framework Phoenix utilizando el lenguaje de programación funcional Elixir el cual cuenta con una sintaxis fácil de entender, lo cual ayuda a la mantenibilidad del proyecto, es distribuido por los que esta pensado para distribuir la carga del trabajo en los distintos nodos que lo forman, es concurrente, tiene una gran resistencia a fallos y errores, garantizando un alto nivel de disponibilidad, además de ser rápido ya que utiliza todos los núcleos de procesamiento disponibles.

Para complementar esto, se ha optado por la plataforma Docker, mediante contenedores, para la ejecución de los distintos entornos. La utilización de esta tecnología ofrece otra serie de ventajas como pueden ser, el beneficio económico por el hecho de que los contenedores son gratuitos y de código abierto, además de tener unos requisitos de infraestructura reducidos, por lo que se reduce el coste de mantenimiento y de infraestructura. Además, permite reducir el tiempo de los despliegues, hace al sistema mas escalable, simplifica las configuraciones,

ofrece aislamiento y una buena seguridad garantizando que las aplicaciones estén aisladas y segregadas.

Este trabajo se centrará en las mejoras de los entornos de ejecución de las practicas, identificando e implementando las características que quedan por añadir, solventando los posibles problemas y minimizando la deuda técnica. Como parte final se documentará todo el trabajo realizado, para que pueda servir de ayuda a futuros desarrolladores, que sigan trabajando en el sistema.

1.1. Estructura del trabajo

- **Capitulo 1: Introducción** Breve descripción del sistema de entregas Deliverit asi como las necesidades desde las cuales a surgido la idea y posterior desarrollo.
- **Capitulo 2: Deliverit** En esta sección profundizaremos el sistema de entregas Deliverit analizando las tecnologías usadas para su implementación así como la arquitectura y modelo de datos.
- **Capitulo 3: Punto de Partida** presentamos la situación actual del sistema analizando las posibles mejoras de los fallos encontrados.
- **Capitulo 4: Diseño** donde vamos a presentar las diferentes alternativas posibles que servirían para resolver el problema descrito en el capitulo anterior.
- **Capitulo 5: Implementación** en esta sección viene detallada la solución que se ha implementado dentro de las alternativas propuestas con el fin de solventar los fallos encontrados
- **Capitulo 6: Conclusiones y Trabajo Futuro** Hablaremos de las principales conclusiones a las que se ha llegado así una pequeña visión sobre el futuro del sistema y de hasta donde puede llegar.
- **Capitulo 7: Análisis de Impacto** En este ultimo capitulo analizaremos el impacto del trabajo realizado viculándolo en el mayor grado posible con los objetivos de Desarrollo Sostenible.

1.2. Metodología

En esta sección describiremos las tecnologías y herramientas que hemos utilizado para el desarrollo continuo bajo la tutela del profesor Ángel Herranz Nieva, tutor de este proyecto.

Para facilitar la gestión de los desarrollos se ha empleado el sistema de control de versiones Git, el cual tiene numerosas ventajas, como pueden ser que permite mantener un histórico de las distintas versiones del desarrollo, nos permite visualizar los principales autores, el numero de versiones, así como los cambios hechos en el código. Ademas facilita mucho el trabajo en equipo y la velocidad

del desarrollo permitiendo paralelizar el trabajo realizado por diferentes desarrolladores.

En concreto para este proyecto estamos usando el servicio web GitLab, el cual es una suite completa la cual nos permite la creación, gestión, y el conexionado distintos repositorios con diferentes aplicaciones.

Para la correcta gestión del desarrollo, utilizamos diferentes ramas , entre las cuales están, la rama **master** donde se aloja el código listo para su puesta en producción

la rama **development** es la rama en la que se integran las nuevas funcionalidades o correcciones de la aplicación, en esta rama se hacen las pertinentes pruebas para su posterior mergeo con la rama master. y por ultimo se encuentrasn las distintas ramas las cuales corresponden a las nuevas funcionalidades en desarrollo, todas estas ramas empiezan con el prefijo /features. Para facilitar la gestion del equipo de trabajo utilizamos la ....(mirar lo que es) gitflow. de hay las ramas features.

En cuanto a la gestión y seguimiento del proyecto utilizamos la herramienta Trello [1], que personalmente no había usado, en la cual mediante tarjetas se asignan tareas concretas a desarrollar. Para el buen seguimiento hay representadas 5 columnas, cada una representa diferentes estados de las tareas

- **Product Backlog** En la que se describen una o más características deseadas en el producto, estas se ordenan por prioridad, se estiman si es necesario entre los integrantes del equipo y se mueven a la lista de TODO.
- **Todo** En esta columna se describen las tarjetas estimadas y listas para empezar por un miembro del equipo
- Ready En esta columna están representadas tareas consideradas listas para ser integradas.
- **Doing**: En esta columna se representan las tareas que se encuentran en desarrollo
- **Done**: En esta última están las tareas acabadas.

Capítulo 2

Deliverit

En este capitulo se va a presentar en profundidad el sistema de entregas unificado Deliverit. Donde veremos en profundidad todas las partes implicadas para suj desarrollo las empezando por tecnologías utilizadas hasta la arquitectura de sus sistema.

2.1. Introducción del sistema

Para presentar Deliverit, en primer lugar hay que entender la necesidad que hay detrás de desarrollo, como hemos comentado anteriormente, en la Universidad hay varios sistema de entrega y corrección de practicas, casi tantos como asignaturas que tienen practicas para corregir. Mediante este nuevo sistema se consigue la unificación de todos los sistemas de entrega, independientemente del lenguaje de programación requerido.

2.2. Tecnologías

Las principales tecnologías usadas para el desarrollo del sistema son:

Elixir

Es el lenguaje de programación elegido para el desarrollo, el cual tiene una serie de ventajas que han motivado su elección para este proyecto. Es un lenguaje funcional, concurrente que se ejecuta sobre la maquina virtual de Erlang llamada BEAM. Dado que Elixir está escrito sobre Erlang este comparte las mismas abstracciones para desarrollar se pueden crear aplicaciones distribuidas y tolerantes a fallos

Phoenix

Es el framework web utilizado para trabajar en el sistema escrito en el lenguaje elixir, utiliza el modelo de arquitectura Modelo-Vista-Controlador(MVC). Entre

sus ventajas se encuentra la alta productividad en la parte de desarrollo y un rendimiento excelente en el rendimiento de la aplicación (mirar esto)

Ecto

Es el wrapper de base de datos y generador de consultas SQL para el lenguaje Elixir, Las ventajas de usar ecto principalmente son su API estandarizada y el conjunto de abstracciones que permiten interactuar con diferentes tipos de base de datos, por lo que se puede consultar cualquier base de datos usando construcciones similares haciendo la interaccion con la base de datos mas sencilla para los desarrolladores

PostresSQL

Es un sistema orientado a objetos y de código abierto de administración de base de datos relacionales, presenta una serie de ventajas como son la escalabilidad, integridad y seguridad. Ademas es uno de los mas usados en todo el mundo.

Docker

Es un servicio que tiene la ventaja de automatizar el desplique de las aplicaciones dentro de contenedores de software, la gran ventaja de utilizarlo es la capa adicional de abstracción y automatización de aplicaciones en diferentes sistemas operativos por lo que hace esta tecnolgía 100*100 portable e independiente del entorno al que se ejecuta. Otra ventaja que presenta es su gran ligereza que hace que no se requieran de maquinas muy costosas para su ejecucion, permitiendo ahorrar costes.

HTML

Es el lenguaje de marcado utilizado para hacer la interfaz de usuario del sistema

CSS

Las siglas CSS (Cascading Style Sheets) que en español significa Hojas de Estilo en cascada, permiten aplicar estilos a los documentos HTML.

JavaScript

Para poder hacer las paginas web, osea la interfaz, dinámica, con html y css no es suficiente, para ello se utiliza javascript que es un lenguaje de programación que permite implementar funciones complejas en paginas web, como actualizaciones de contenido, mapas interactivos, animación de gráficos en 2d y 3d, etc

Bootstrap

Es un framework de estilos para interfaces (front-end) utilizado en el desarrollo de aplicaciones web que se adaptan a las diferentes pantalla de cualquier dispositivo en otras palabras facilita al desarrollador hacer su aplicación responsive..

2.3. Arquitectura del Sistema

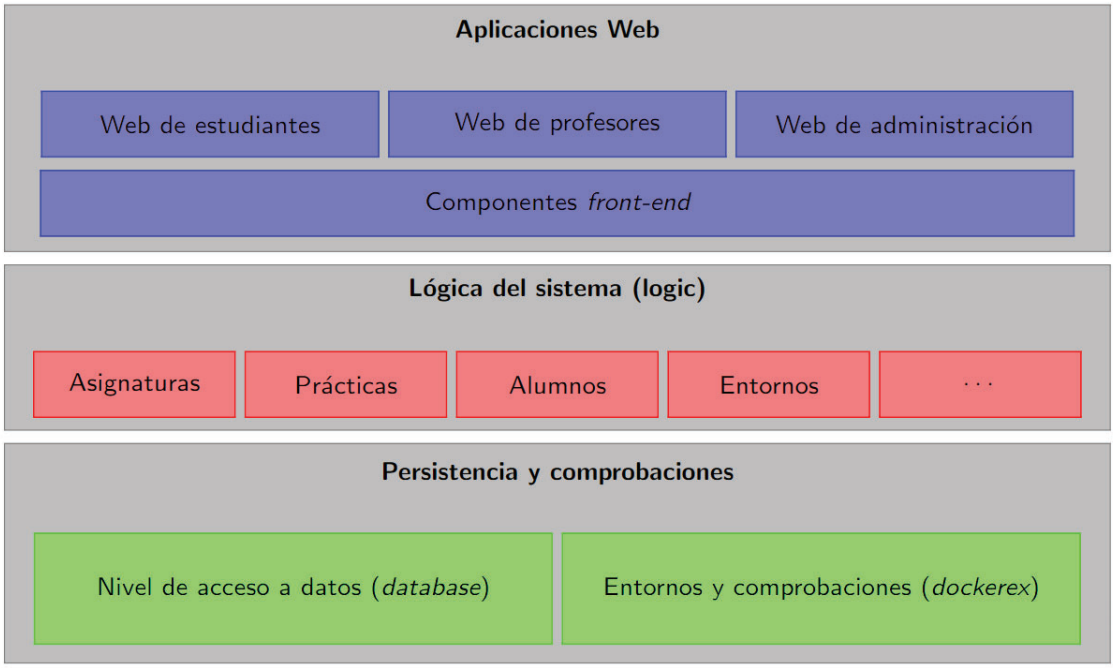

Figura 2.1: Arquitectura de Deliverit. (TFG de Andrés Mareca).

En la figura anterior podemos observar de manera general la arquitectura de Deliverit. En esta se describen 3 niveles o capas principales: Aplicaciones Web, Lógica del sistema y por ultimo Persistencia y Comprobaciones. Detallaremos cada uno de ellos en secciones.

Capa de Aplicación web

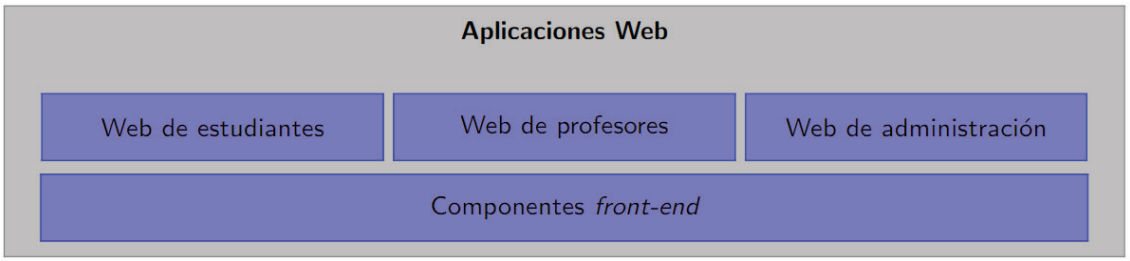

Figura 2.2: Capa aplicaciones web.

En este nivel como podemos comprobar en la imagen, hay tres aplicaciones web mediante la cual los distintos usuarios (Alumnos, Profesores y Administradores) pueden acceder y utilizar el sistema. Los Alumnos, que es la parte más restrictiva de la aplicación, pueden registrarse en la aplicación, registrarse a asignaturas en las cuales tiene practicas pendientes, entregarlas y verificar los resultados, en esta última parte es la funcionalidad en la que hemos centrado el trabajo en este TFG. En cuanto a la parte de los Profesores/Administradores, pueden crear y registrar prácticas, comprobar el estado de las prácticas de los alumnos, así como modificarlas y por ultimo la parte de administración que es donde se pueden crear, borrar y modificar cuentas de los nuevos usuarios, prácticas y administrar todas las partes del sistema.

Capa de Lógica del Sistema

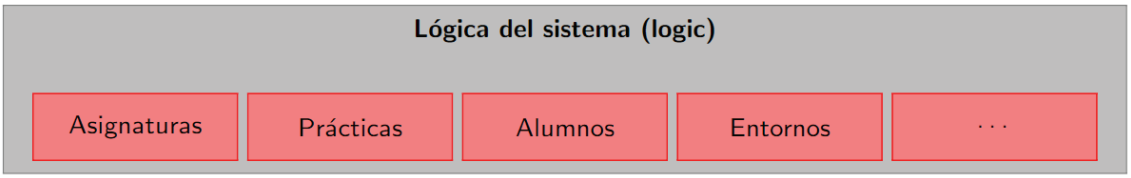

Figura 2.3: Capa lógica del sistema.

En este nivel se centra toda la parte de la lógica de negocio del sistema, está dividida en diferentes módulos, que permiten hacer más mantenible las distintas partes que componen la aplicación. Entre otras funcionalidades, esta capa es la encargada de manejar la corrección de las practicas, la comunicación con la base de datos, el manejo de los entornos de ejecución, así como del almacenamiento en el sistema de ficheros tanto de la practica como de los resultados de las correcciones, donde en esta ultima parte se centra el trabajo de este TFG.

Capa de Persistencia y comprobaciones

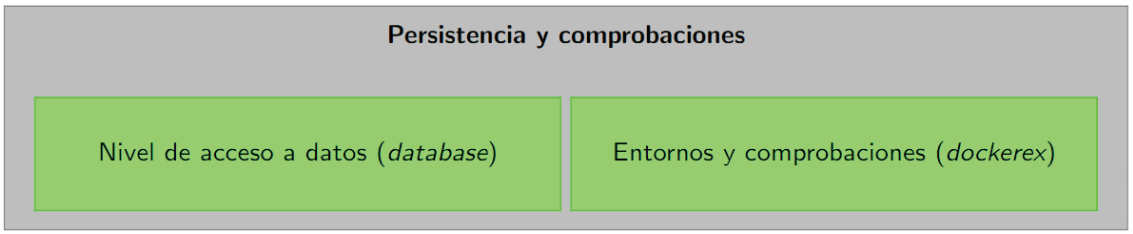

Figura 2.4: Diseño del sistema. (TFG de Aaron Contreras).

La capa de persistencia utiliza ecto, que como hemos comentado con anterioridad a la hora de explicar esta tecnología, permite aislar los accesos a la base de datos del resto del sistema, además de contener tanto el modelo de datos, el esquema de la base de datos y las migraciones.

Por último, encontramos el componente de comprobación el cual permite ejecutar las correcciones sobre el código entregado por los alumnos de forma segura. Tanto el código de los alumnos como el de los profesores, se ejecutarán mediante etapas en Docker, mediante una biblioteca escrita en Elixir y desarrollada por un antiguo compañero Andrés Mareca y mantenida y ampliada por el profesor Ángel Herranz, que es capaz de manejar las distintas construcciones de Docker como por ejemplo la creación de imágenes, contenedores, volúmenes, etc. llamada Dockerex.

Ademas de todo esto, la librería dispone de una funcionalidad mediante la cual podemos mejorar la experiencia de los alumnos a la hora de comprobar los re-

sultados de sus entregas. Esto es la recogida y envío de los logs (registros) generados en los diferentes contenedores Docker que se ejecutan en las distintas etapas de las correcciones.

En cuanto al modelo de datos, en la figura 2.5 podemos observar el modelo entidad-relación, en el cual se han extraído los atributos para una mayor simplicidad a la hora de representarlo.

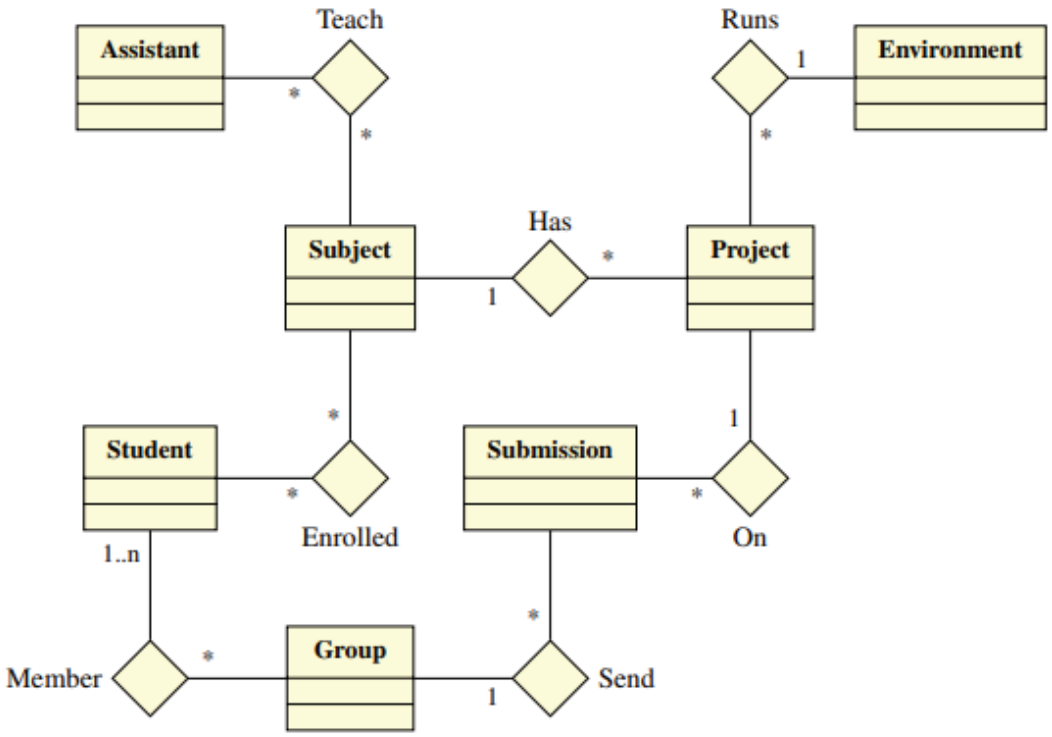

Figura 2.5: Modelo Entidad-Relación. (TFG de Andrés Mareca).

Por ultimo en figura 2.6 podemos apreciar las relaciones existentes entre los componentes del sistema el cual se divide en cinco aplicaciones , todas contenidas en un unico proyecto bajo el dominio de Umbrella

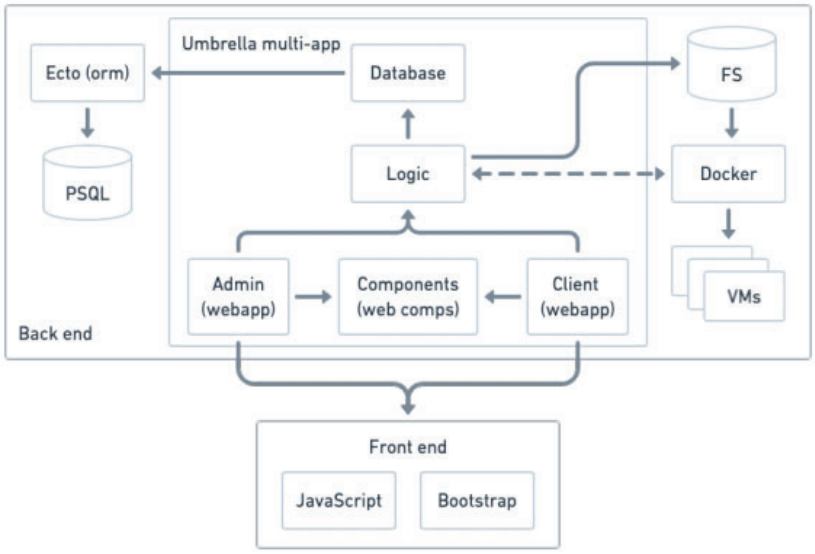

Figura 2.6: Diagrama Relaciones del Sistema. (TFG de Aaron Contreras).

Capítulo 3

Punto De Partida

Como hemos comentado anteriormente el sistema Deliverit ya está implementado y desplegando en producción, aunque actualmente se encuentra en una versión beta, por lo que aun requiere algunas mejoras. Ya existen varias asignaturas que lo está usando, lo que es una gran ventaja, ya que tanto los profesores como los alumnos de estas asignaturas pueden proporcionar un buen feedback de los errores y mejoras que se puedan detectar.

Para entender un poco más el contexto del problema a solucionar en este trabajo, vamos a explicar el proceso de envío y corrección de las practicas.

Actualmente todos los alumnos registrados tienen permiso para enviar las prácticas de las diferentes asignaturas en las que está dado de alta para su posterior corrección a la plataforma Deliverit.

Al loguearse en la plataforma, al alumno entra en la pantalla principal se presentan las asignaturas en las que está dado de alta, y las practicas disponibles, pero para poder hacer un envió, tiene que registrarse a esa práctica previamente, para ello, debe pulsar el botón de **Registrarse** que podemos ver en la figura 3.1.

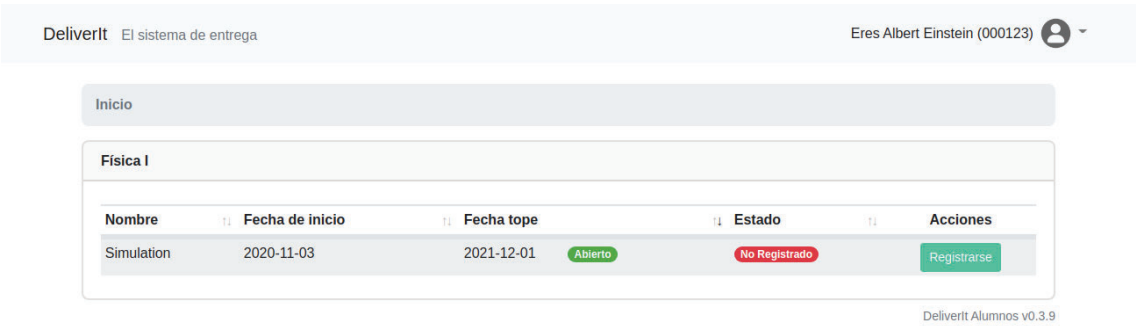

Figura 3.1: Pantalla principal.

Posteriormente, el alumno será dirigido al formulario de registro de prácticas

donde puede seleccionar la practica en la que desea hacer las entregas así como, los alumnos involucrados, en caso de que sea una práctica en grupo. Para terminar con el registro una vez seleccionadas las opciones deseadas, el usuario debe pulsar el botón de **Finalizar** como podemos observar en la figura 3.2.

Figura 3.2: Pantalla registro de practicas.

Realizado el paso anterior, se nos presenta la pantalla de entregas realizadas, donde podemos visualizar todas las practicas corregidas hasta el momento, con sus respectivas calificaciones, además desde aquí también podemos realizar nuevas entregas. Para realizar el envío de una práctica el alumno debe pulsar el botón **Nueva Entrega** el cual le redirigirá al siguiente paso, como podemos observar en la figura 3.3.

Figura 3.3: Pantalla registro de practicas.

En esta pantalla, el alumno selecciona el archivo con el código fuente de la práctica, en este caso “Submitted.java”, y pulsa el botón de **Enviar** para empezar subir el fichero al servidor, como podemos observar en la figura 3.4.

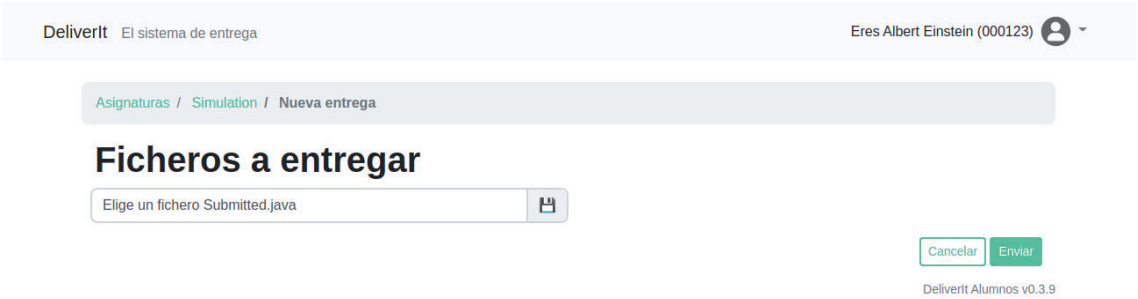

Figura 3.4: Pantalla registro de practicas.

Enviado el fichero al servidor, llegamos a la parte donde se nos presenta el problema a resolver en este trabajo, en la figura 3.5, podemos observar la interfaz de resultados de comprobación. Una vez subida una práctica para su comprobación este se encola, y el alumno solamente puede esperar y pulsar repetidamente el botón de **Actualizar la página** para poder ver los resultados de la corrección de su práctica, esta implementación presenta varios problemas tanto desde el punto de vista de la experiencia de usuario, como en el rendimiento general del sistema, ya que al forzar al alumno a tener que recargar la página constantemente, se introduce una sobrecarga al sistema, que dependiendo del número de usuarios concurrentes puede tener unos resultados desastrosos, que pueden ser desde hacer que el sistema funcione mucho más lento de lo normal hasta bloquear por completo la aplicación.

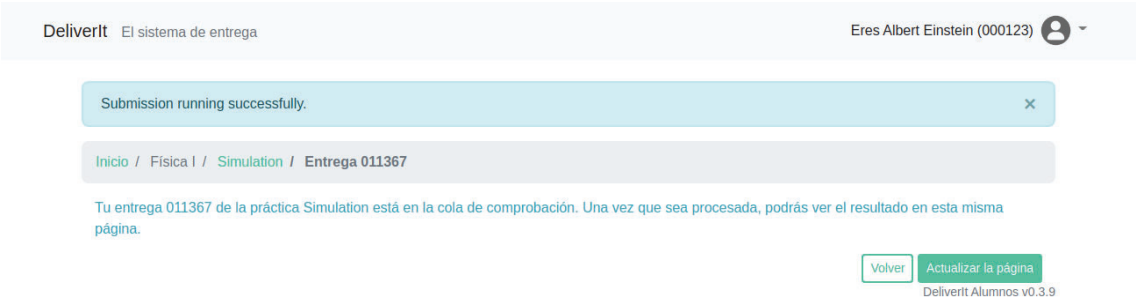

Figura 3.5: Pantalla registro de practicas.

En la figura 3.6, mostramos un ejemplo de la interfaz donde se muestran la información de todos los pasos de la corrección de la práctica. El objetivo es mostrar esta información casi al mismo tiempo que esta se va generando. Para ello en el siguiente capítulo detallaremos las distintas alternativas con las que podemos solventar este problema.

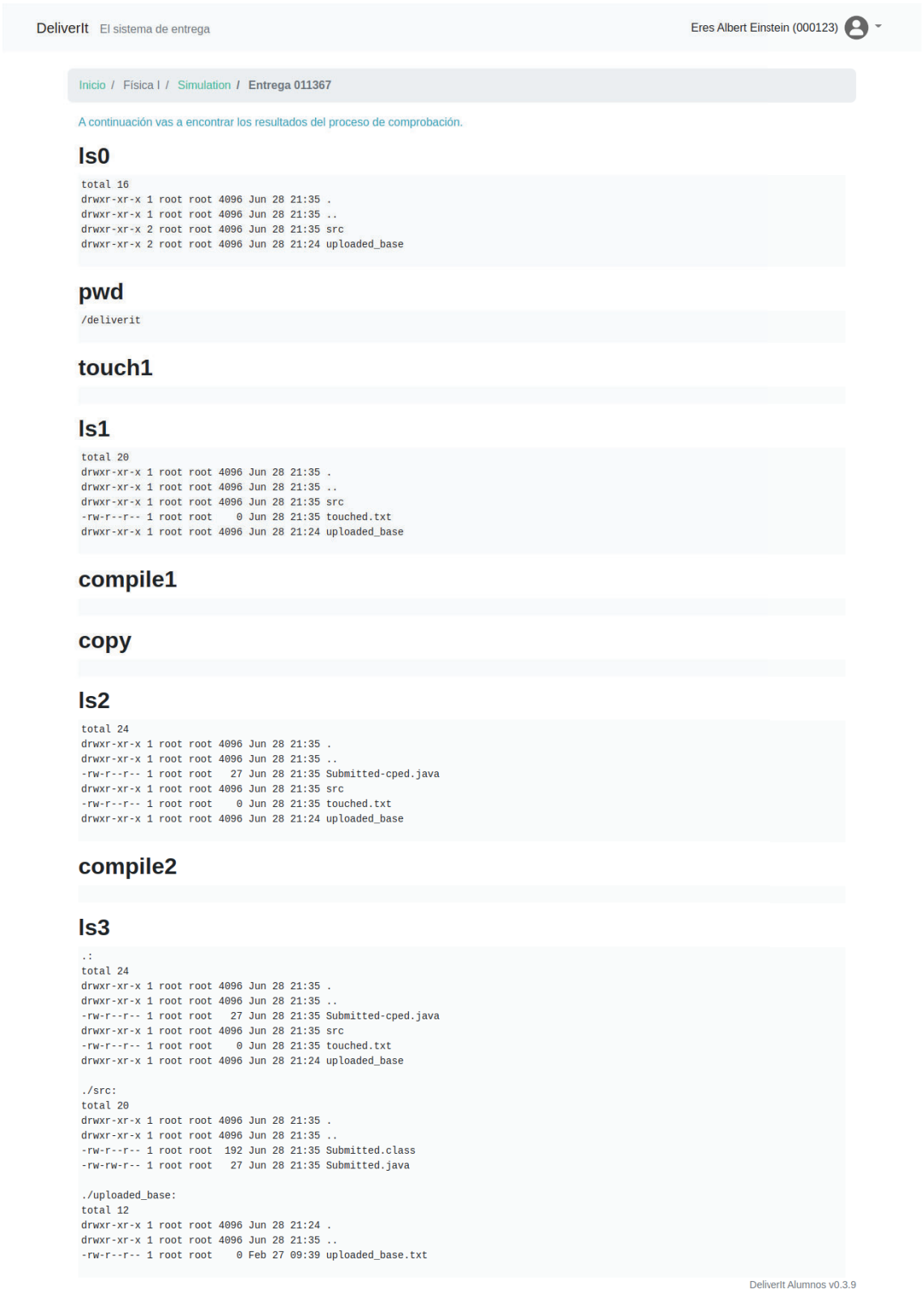

Figura 3.6: Interfaz resultados del proceso comprobación de la practica.

En la siguiente figura podemos observar mediante un diagrama de secuencia el flujo actual del desarrollo, del proceso de ejecución de una practica.

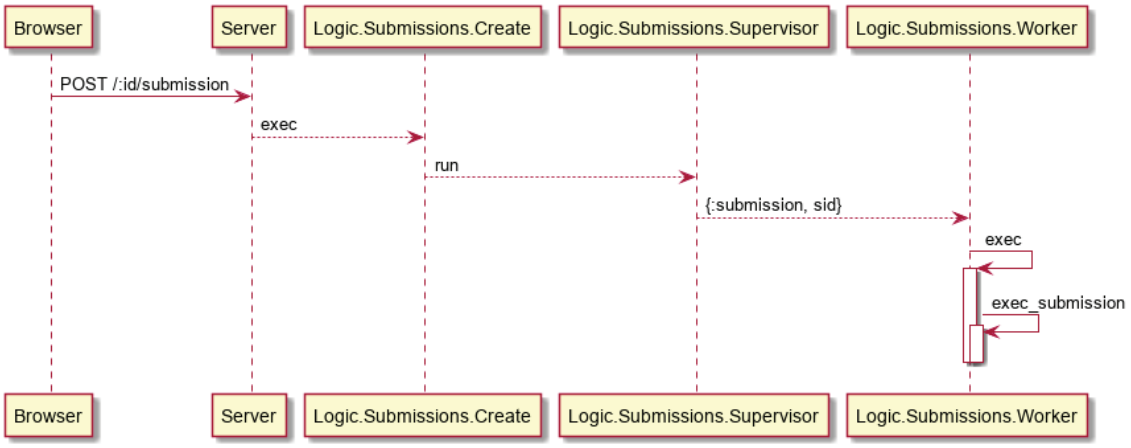

Figura 3.7: Diagrama de secuencia de ejecucion de una practica.

Observamos, que desde el navegador, se manda una petición POST al servidor, este ejecuta la lógica de creación de una entrega con exec, posteriormente se llama al Supervidor, el que a su vez pone en marcha un Worker enviándole un mensaje con el id de la entrega. El Worker en este caso es el que pone en marcha todos los procesos de ejecución de las etapas de verificación, poniendo en marcha un contenedor Docker por cada etapa, donde una vez finalizado, notifica mediante email al alumno de su resultado.

Capítulo 4

Diseño

En este capitulo vamos a presentar tres diferentes alternativas mediante las cuales podemos solventar el problema con la información de la corrección de las practicas que visualizan los alumnos disminuyendo a su vez el impacto al sistema producido por la actual implementación.

Las diferentes alternativas basan su implementación en la utilización de eventos enviados por el servidor (SSE) y la librería Dockerex, la cual recoge y envía de manera asíncrona los logs producidos en los diferentes contenedores de las distintas etapas de la ejecución de las correcciones de las practicas. (revisar)

Para entender mejor el proceso de corrección, mencionar que esta tiene varias etapas, (un ejemplo en la figura 4.1), el sistema crea y ejecuta un contenedor Docker por cada una de estas etapas, y la librería Dockerex envia los logs al back de la aplicación y este los guarda en un archivo llamado "logs. tal", y se muestran en el front (la interfaz) renderizando ese archivo. Un ejemplo de su como se muestra esta en la ultima figura del anterior capitulo

4.1. Primera Alternativa - Soft Real Time

Este seria el escenario ideal, donde los logs de la corrección de la practica aparecerían en pantalla casi en tiempo real , usando el sistema asíncrono de logs de la biblioteca Dockerex,

En primer lugar presentamos la solución optima al problema, en la cual se mostrarían los logs en “Soft Real-Time”, para lograr esta funcionalidad, necesitamos una manera de poder enviar de forma asíncrona estos datos (los logs generados por los diferentes steps) desde el contenedor Docker al back-end de la aplicación y desde este ultimo enviarlo al front (interfaz de la aplicación) para que el usuario pueda visualizar la información, una breve ilustración la podemos observar en la figura 4.1.

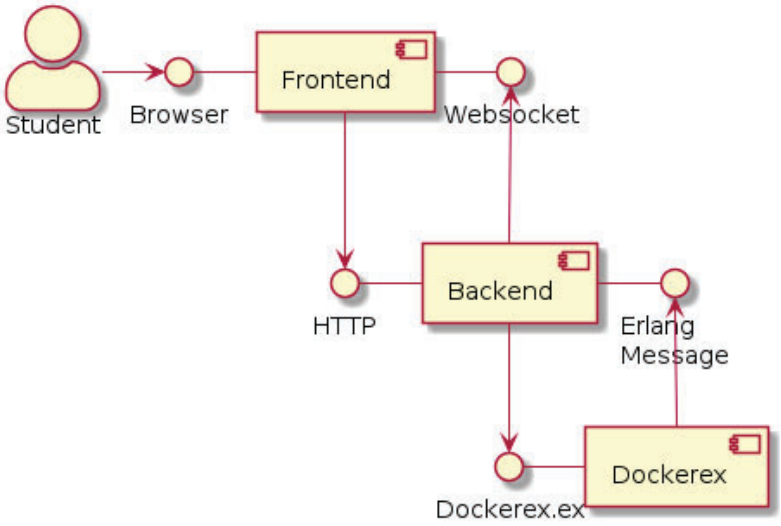

Figura 4.1: Diagrama paso de logs.

Aquí es donde entra en juego la librería Dockerex mencionada anteriormente, la cual dispone de una funcionalidad en la que con un id de contenedor, manda la salida estándar del entorno a un proceso elixir, de manera asíncrona.

El desarrollo en la parte servidor (back-end) seria la de crear un servicio, conectándolo a un endpoint "/sse", que al ser invocado, ejecutaría la función "Containers.logs/3" de Dockerex, mediante la cual recibiríamos los logs, posteriormente, procesaríamos los logs y los enviaríamos a la interfaz utilizando Eventos enviados por el servidor(SSE).

El front-end por su parte al entrar en la interfaz de sumbmissions, hace una llamada al endpoint /sse para ejecutar todo el proceso, y se mantiene escuchando y mostrando en por pantalla cada una de las lineas de logs recibidas. Con lo que el usuario puede visualizar todo el proceso mientras este se está ejecutando, mejorando así la experiencia de usuario y disminuyendo de manera drástica el impacto producido por la anterior implementación, ya que no haria falta el continuo refresco de la pagina.

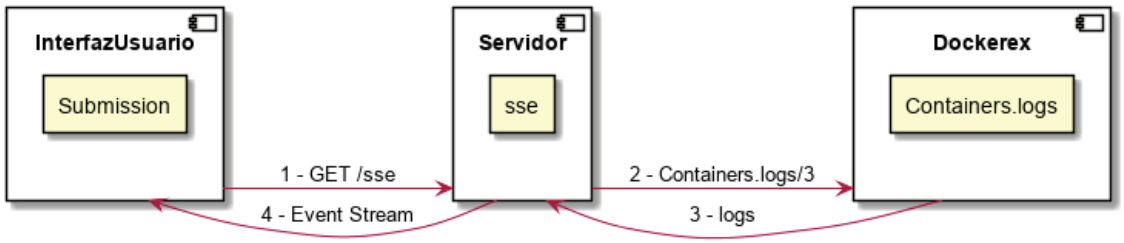

Figura 4.2: Diagrama paso de logs.

Esta alternativa no se ha abordado ya que exige una reestructuración importante de la arquitectura de la aplicación ya que en ahora no tenemos una relación

entre el id de una entrega con el id del contenedor.

tino : por favor Angel completa esto

4.2. Segunda Alternativa- Actualizar al Terminar

En segundo lugar presentamos otra alternativa (mas rápida de implementar) en la cual reutilizamos parte del desarrollo actual, concretamente la obtención de los logs. La idea de esta alternativa es que el alumno no tenga que estar recargando constantemente la interfaz de corrección hasta tener los resultados, en este desarrollo, sería la propia aplicación la encargada de enviar la información y mostrarla en pantalla.

Actualmente, cuando acaba el proceso de corrección de la practica, se envía una notificación de ello vía email al alumno, por lo que justamente en el envío de dicha notificación, y mediante la tecnología de Eventos enviados por el servidor (SSE), la parte servidor de la aplicación (back-end) accedería a los ficheros de logs generados , para posteriormente recogerlos, procesarlos y enviarlos a la interfaz (front-end). Este flujo seria invocado mediante una petición GET al endpont /sse.

La parte frontend de la aplicación, seria la encargada de hacer la petición al endpoint, en el momento que termina la corrección de la practica, para ejecutar todo el proceso del servidor, momento en el cual, escucharía los eventos enviados, los procesaría obteniendo los logs que posteriormente mostraría en la interfaz de usuario.

Por lo que aunque no se pueda ver toda la información al mientras se están ejecutando los procesos, podríamos ver la información sin tener que estar recargando constantemente la pagina, lo que reduciría la sobrecarga al sistema, causado por las continuas peticiones de recarga realizadas por el alumno.

Aunque para aplicar llevar a cabo la implementación de esta alternativa, seguimos teniendo el mismo problema que en la anterior, habría que modificar la arquitectura ya que no tenemos el conn y el submission id, con el desarrollo actual no hay forma de hacerselo pasar al back. (revisar esto ultimo)

(Introducir un diagrama)

4.3. Tercera Alternativa - Actualizar en cada Paso

Esta seria la tercera alternativa, que seria un punto intermedio entre la primera y segunda alternativa, donde imprimiríamos los logs de los contenedores, al acabar cada etapa de la ejecución de la practica.

En primer lugar mencionar que hay actualmente, se esta creando un fichero de log por cada etapa ejecutada durante el proceso de corrección de las practicas. Por lo que se pueden acceder a estos ficheros e ir imprimiendo por pantalla las distintas lineas de logs que hay en cada uno de ellos según se van generando.

Para entender un poco mejor la lógica del proceso de corrección de las practicas, en la figura 4.3 podemos ver el panel de administración, con la configuración de

una practica, donde entre otros datos, vienen los pasos a ejecutar (Steps), que hay configurados. Como hemos comentado anteriormente, cada uno de estos "Steps" se ejecuta en un contenedor Docker distinto, por lo que al acabar su ejecución la biblioteca Dockerex genera un fichero de log. por cada ejecución y los guarda en una carpeta llamada /logs dentro del sistema de archivos de las entregas (Submissions) como podemos observar en la figura 4.4.

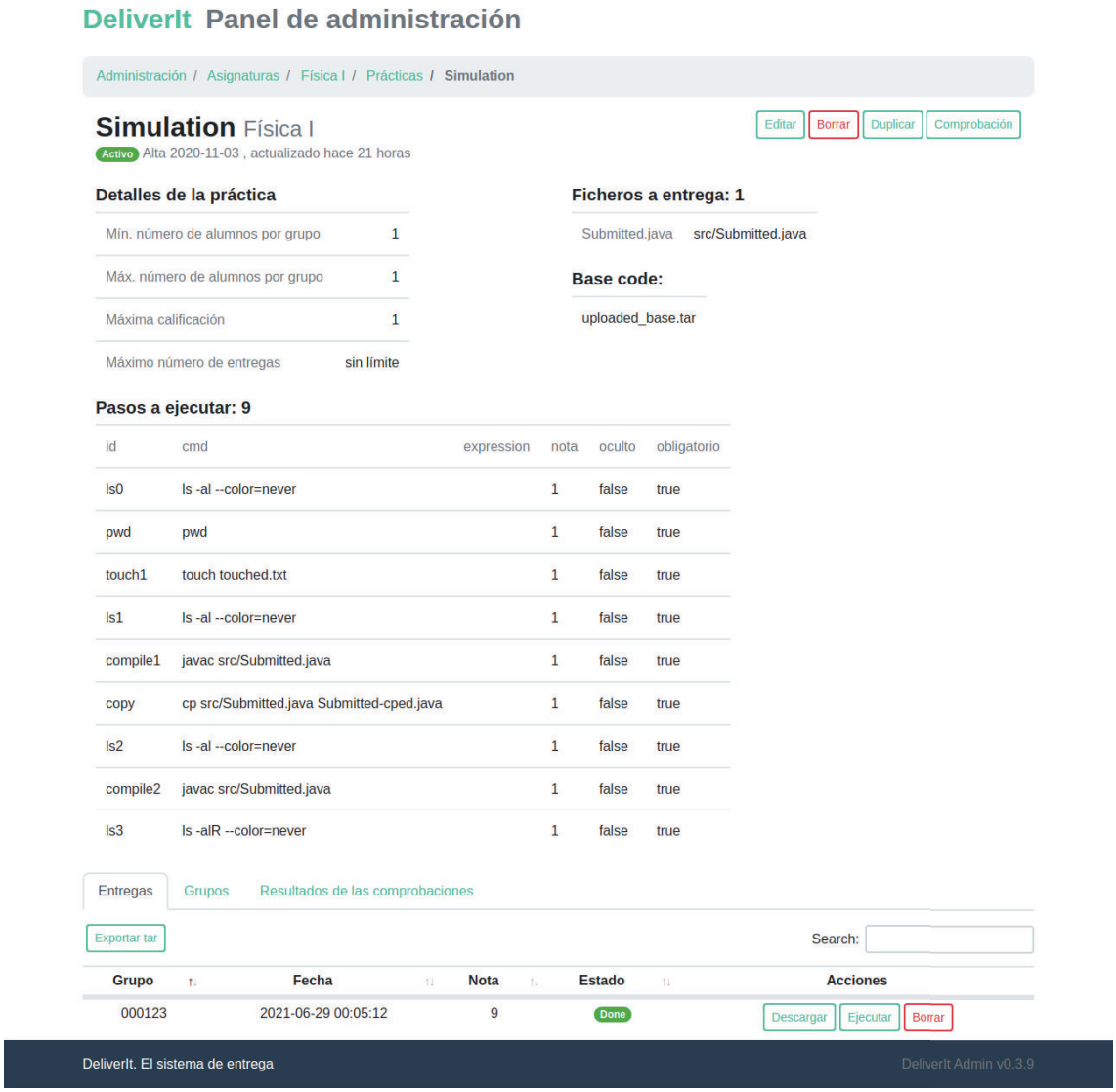

Figura 4.3: Panel de Administración - Práctica.

En la figura 4.4 viene ilustrada la estructura de archivos utilizada por la aplicación, dentro hay un directorio por proyecto que tiene por nombre el ID del proyecto en cuentión. Este contiene en su interior las carpetas de las entregas realizadas por los alumnos que al igual que la carpeta de proyecto tiene por nombre un ID en este caso es el ID de la entrega (Submission). Dentro de cada una de las carpetas de entrega, viene el directorio logs, que en su interior alberga cada unos de los ficheros de log los cuales son generados al finalizar la ejecución cada uno de los Steps (etapas de ejecución).

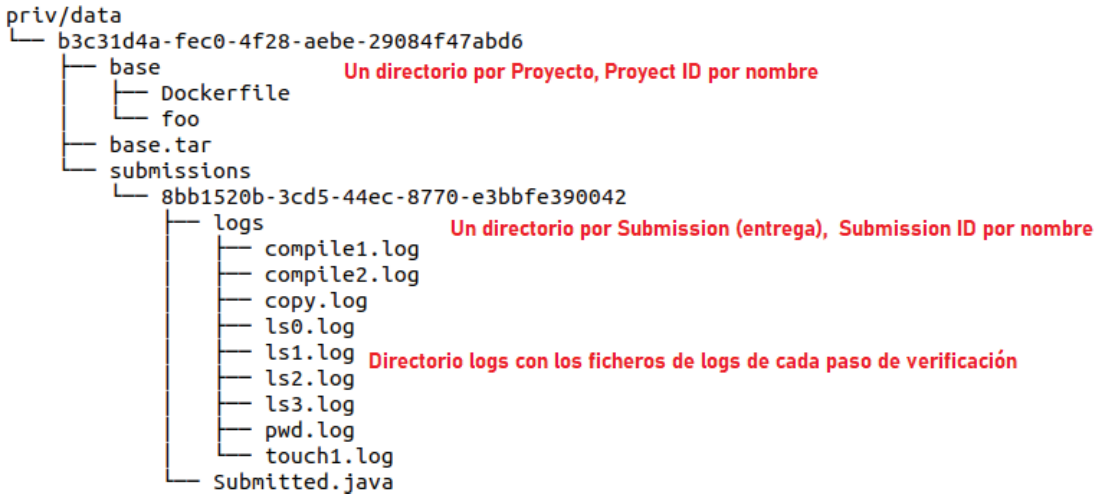

Figura 4.4: Estructura de directorios.

Para desarrollar esta funcionalidad, reutilizaremos parte de la desarrollo existente, concretamente la parte donde gracias a la librería Dockerex , se van creando los distintos ficheros de logs correspondientes a cada una de las etapas de verificación de las entregas.

Mediante eventos enviados por el servidor (SSE), enviaremos la información contenida en cada uno de estos ficheros a la interfaz de usuario.

4.4. Cuarta Alternativa - *Polling*

Esta ultima alternativa, seria una solución transitoria, ya que por el problema de la arquitectura actual no se pueden implementar las anteriores soluciones.

Como hemos comentado anteriormente, el mayor problema de la solución actual es que el alumno tiene que refrescar repetidamente y de forma manual la pagina para poder visualizar los logs, ya que hasta que no se haya terminado el proceso de corrección, la pagina sigue sin mostrar el resultado.

Una solución viable para abordar este problema es con la arquitectura actual, es recargar la pagina automáticamente, mostrando paso a paso los resultados de las etapas disponibles en ese momento. Terminando este proceso cuando todas las etapas hayan finalizado.

Explicaremos en detalle la implementación de este desarrollo en el siguiente capitulo.

Capítulo 5

Implementación

En este capítulo detallaremos el desarrollo de la cuarta alternativa descrita en el anterior capitulo, la cual como se ha comentado es una solución transitoria, ya que sin hacer una reestructuración sustancial de la arquitectura, es el camino más optimo de abordar el problema.

La solución implementada,se basa principalmente en hacer polling a los distintos ficheros que contienen los logs de las diferentes etapas de ejecucion de las practicas, y mediante refresco automático de la interfaz de usuario lograr la funcionalidad deseada.

5.1. Código

En primer lugar empezamos describiendo el desarrollo implementado en la parte servidor de la aplicación, en concreto en el módulo submission_controller.ex el cual se encarga de manejar las distintas peticiones, a continuación detallamos los cambios que han permitido lograr la funcionalidad deseada.

El el siguiente fragmento de código, es donde formamos el path, donde están alojados los logs de las diferentes etapas de ejecución, podemos ver la estructura de los datos en la figura 4.4. Para ello, necesitamos el id tanto de la practica (submission.project_id), como el de la entrega (submission.id), una vez formado este path lo utilizaremos mas adelante para obtener los datos de los ficheros de logs.

```
project = Subjects.get_project(submission.project_id, true)

# Get Files
path =
Path.join([
Logic.get_path(),
submission.project_id,
"submissions",
submission.id,
"logs"
```

```
])
```

En el siguiente fragmento filtramos las etapas visibles, (las que no estan ocultas), posteriormente formamos un mapa, donde la clave es la etapa ejecutada y el valor es la información que corresponde a los logs de dicha etapa. Para obtener la información de los ficheros hacemos uso de la función **get_file/1**, que detallaremos mas adelante.

```
steps = project.steps |> Enum.filter(fn %{hide: h} -> not h end)

logs =
Enum.map(steps, fn %{id: name} ->
{name, get_file(Path.join([path, "#{name}.log"]))}
end)
```

Para poder obtener la información de los ficheros de logs, se ha creado la función privada **get_file/1**, la cual recibe por parámetro el path anteriormente creado y lee los ficheros de logs haciendo uso del Módulo **File** y su función **read/1**.

```
defp get_file(path) do
        case File.read(path) do
        {:ok, binary} -> binary
        _ -> nil
        end
end
```

El mapa creado anteriormente lo enviamos mediante la función render, a la interfaz de corrección de la practica que sera la encargada de mostrar por pantalla su contenido, junto con el enviamos la nota (submission.mark) la cual nos sera de utilidad para esta funcionalidad. El template de la interfaz de corrección de la practica se llama show.html

```
render(
conn,
"show.html",
submission: submission,
project: project,
mark: submission.mark,
logs: logs
```

Una vez preparado el envío de los logs de las distintas etapas, en el template show.html.eex (el cual es la interfaz donde se muestran las correcciones), creamos un pequeño script el cual recarga automáticamente la pagina, cada 3 segundos hasta que la practica tenga una nota, lo que quiere decir que el proceso de ejecución de todas las etapas de verificación que conforman la corrección de la practica han acabado.

Los diferentes logs se muestran en la molécula submission.logs

```
<%= if is_nil(@mark) do %>
```

```
<script>setTimeout(function(){window.location.reload(1);}, 3000);</script>
<div class="row">
<p class="col-12␣text-info">
<%=  gettext("Your␣summision␣%{id}
␣of␣the␣project␣%{project_name}␣is␣being␣processed.",
id: short_id,
project_name: @project.name)
%>
</p>
<div class="col-12␣text-right">
<span><%= link gettext("Back"),
 to: Routes.project_path(@conn, :show, @project),
  class: "btn␣btn-sm␣btn-outline-primary" %></span>
<span><%= link gettext("Refresh"),
 to: Routes.submission_path(@conn, :show, @project, @submission),
  class: "btn␣btn-sm␣btn-primary" %></span>
</div>
</div>
<% end %>
<div class="row">
<%= Molecules.submission_logs(@logs) %>
</div>
```

5.2. Resultado

En esta sección, se mostrara el resultado del desarrollo anterior donde podremos observar los cambios en la interfaz de usuario. Para hacer esta prueba hemos generado una nueva entrega con un proyecto de ejemplo de la asignatura registrada Física I, la practica utilizada en este ejemplo se llama Simulation, la cual esta escrita en el lenguaje de programación Java.

Para apreciar la diferencia que hay entre antes y después mostramos un ejemplo de como es el proceso de visualización de los logs con el nuevo desarrollo y otro ejemplo de como era anteriormente la interfaz de espera, donde el usuario no podía visualizar ninguna información..

En la figura 5.2, podemos observar como sin actualizar manualmente la practica, se van mostrando los logs de las diferentes etapas de verificación, según termina su ejecución.

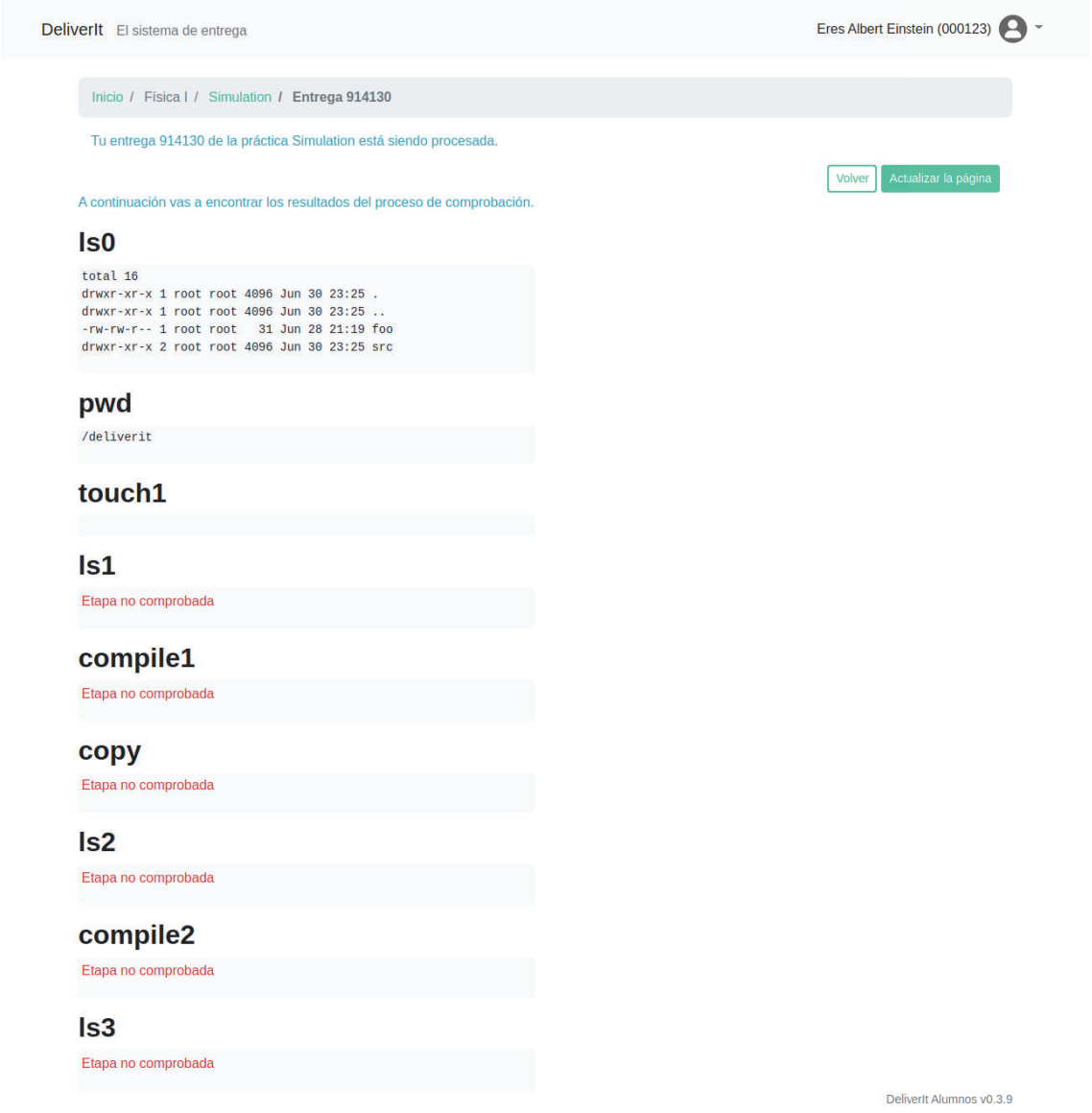

Figura 5.1: Interfaz resultados del proceso comprobación de la practica.

Una vez hayan terminado de ejecutarse todas las etapas de verificación podemos apreciar en la figura 5.2 los logs de todas las etapas , y como este ya ha acabado, no se vuelve a recargar la pagina automáticamente. La interfaz con la etapa de ejecucion acabada podemos verla en la siguiente figura.

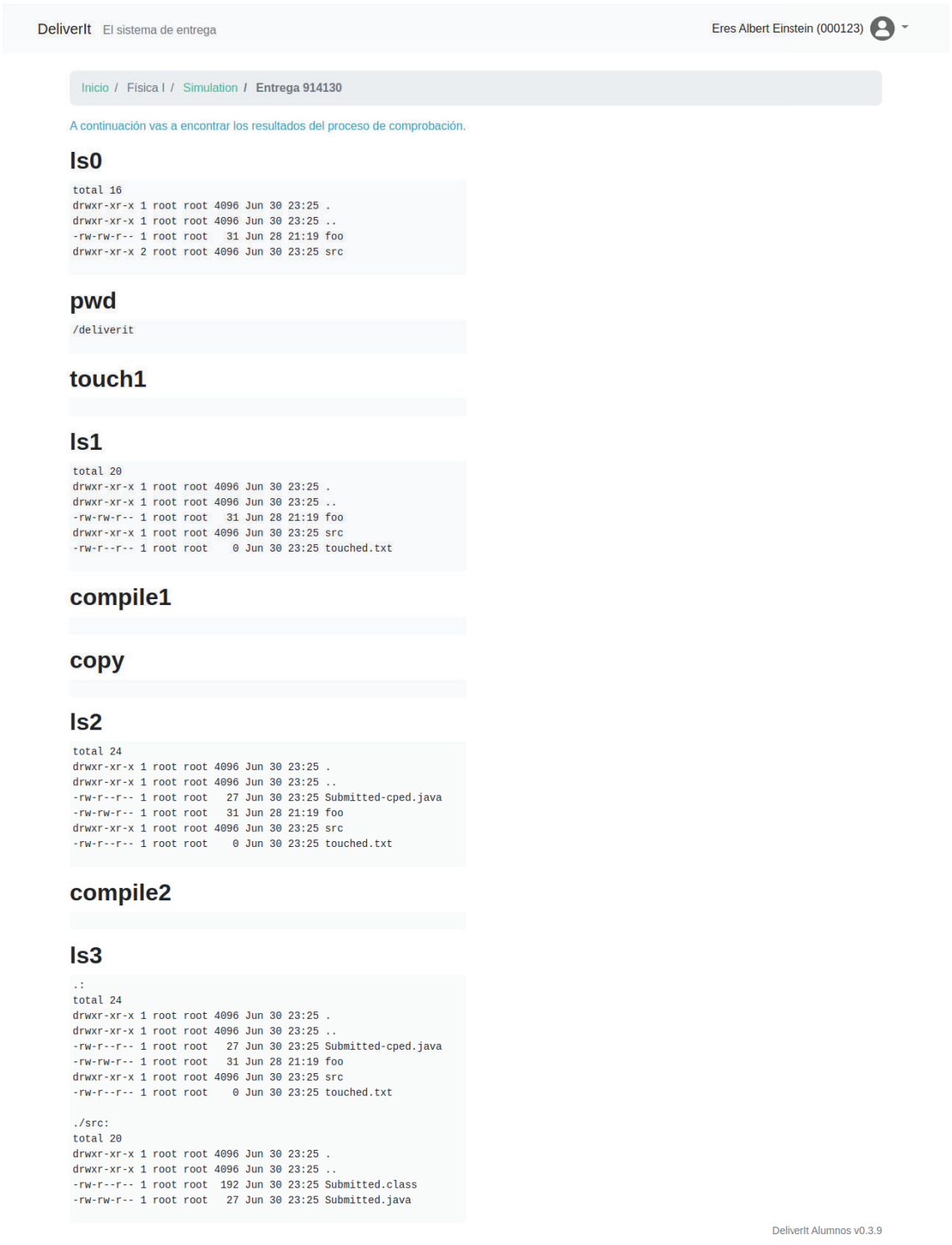

Figura 5.2: Interfaz resultados del proceso comprobación de la practica.

En la figura 5.3 mostramos la interfaz de espera del usuario tal cual estaba implementada. Donde podemos observar, que al alumno se le informa de ningún resultado, y tiene que permanecer sin saber si su entrega tiene algún error es correcta.

Figura 5.3: Interfaz resultados del proceso comprobación de la practica.

Con este desarrollo, aunque no hemos podido abordar una solución optima, hemos mejorado la experiencia de usuario (UX). Ahora el alumno puede saber el resultado de las diferentes etapas, antes de que acabe todo el proceso.

Capítulo 6

Conclusiones y Trabajo Futuro

Terminamos el ultimo capítulo de la memoria presentando las conclusiones que se han sacado a partir del trabajo realizado. Incluiremos también una opinión del trabajo futuro que se puede llevar a cabo para seguir mejorando este útil sistema. Para finalizar realizaremos un análisis del impacto de los resultados obtenidos durante la realización del presente trabajo, vinculándolo dentro de lo posible a uno de los objetivos de desarrollo sostenible previstos de alcanzar en los próximos 15 años.

6.1. Conclusiones

Como alumno de la universidad Politécnica de Madrid, a lo largo de los años de la carrera, he tenido que realizar multitud de practicas, por lo que he podido apreciar, los numerosos sistemas de entrega que existen actualmente. Aprender el funcionamiento de cada uno de ellos , se traduce en un gasto de tiempo importante por parte del alumnado. Desde la perspectiva del profesorado, el desarrollo y mantenimiento de todos estos sistemas, conlleva una gran inversión tiempo y dinero.

Con el sistema de entregas Deliverit, se esta intentando encontrar una solución a este problema utilizando tecnologías modernas, escalables y fáciles de mantener. Tengo la esperanza de que este sistema pueda ser usado en todas las asignaturas de la carrera, y porque no en toda la universidad

Gracias a este trabajo, he podido dar mis primeros pasos en un lenguaje funcional como es Elixir y aprender sus bondades, lo que me ha permitido cambiar mi manera de pensar, a esto hay que sumarle las diferentes tecnológicas así como las metodologías ágiles con las que no he tenido oportunidad de trabajar hasta ahora. Toda la experiencia adquirida me será de gran utilidad en mi futura carrera profesional.

En cuanto al tiempo dedicado al trabajo que en total ha sido de unas 324 horas, se ha repartido en las distintas tareas de la siguiente manera:

- Familiarización con las herramientas de desarrollo

Un 8% se ha dedicado a al aprendizaje y familiarización de las distintas herramientas y tecnologías, que se han usado para el desarrollo del trabajo.

- Familiarización con el estado actual del sistema y su arquitectura

  Alrededor de un 14% del total de las horas, han sido dedicadas a la familiarización de la arquitectura, y el estado actual del sistema.

- Identificar características, puntos de mejora en la creación y uso de los entornos de ejecución, así como deuda técnica

  Un 14% se han invertido en investigación y análisis de los puntos a mejorar dentro de la aplicación.

- Encontrar propuestas y desarrollar las mejoras en los entornos de ejecución, mejorar la UX al ejecutar las prácticas y minimizar deuda técnica Aproximadamente un 38% del tiempo se a dedicado a identificar las diferentes alternativas a desarrollar, y en la implementacón y pruebas.

- Documentación Finalmente un 26% lo hemos dedicado a documentar los cambios en el sistema así como los pasos para los futuros desarrolladores que trabajen en este proyecto.

6.2. Trabajo Futuro

En esta sección aportare algunas mejoras que se pueden abordar, para mejorar el sistema de entregas Deliverit.

Optimización. Como posible mejora respecto a la manera en la que se puede obtener la información de corrección de las practicas habría que revisar la arquitectura e implementar los cambios oportunos para mejorar el desacoplamiento. De esta forma se podría abordar el problema descrito en anteriores capítulos aportando una solución mas optima y definitiva.

Bajo mi punto de vista esta solución podría ser la primera alternativa presentada en el capitulo 4 donde describimos diferentes soluciones. Ya que presenta muchas ventajas, al ejecutarse de manera concurrente reduciría el impacto en el sistema ocasionado por las continuas peticiones realizadas por los alumnos, ademas mejoraría enormemente la experiencia de usuario (UX).

Pruebas. Una buena practica, para mejorar la calidad del código y asegurarse de que las nuevas funcionalidades que se van implementando y no rompen la funcionalidad actual, sería la realización de test unitarios y de integración. Esto se puede conseguir mediante las herramientas que ofrece Elixir/Phoenix.

Integración Una mejora fundamental es integrar Deliverit con los diferentes sistemas que hay actualmente en la Universidad.

6.3. Análisis de Impacto

En esta sección analizaremos si nuestro trabajo cumple con alguna de las 17 propuestas de desarrollo sostenible establecidos por las Naciones Unidas.

El primero de los objetivos que cumple nuestro sistema es el numero 4 **Educación de Calidad**, las mejoras en la UX implementadas mejoran la experiencia de los alumnos a la hora de entregar las distintas practicas que tienen que realizar a lo largo de su vida Universitaria.

Cabe mencionar también que la plataforma de entregas Deliverit, ayuda al profesorado, minimizando el trabajo invertido tanto en la gestión como en el proceso de corrección de las practicas.

Otro punto importante que aporta el sistema, es el numero 7 **Energía asequible y no contaminante**, recordamos que el principal objetivo de Deliverit es el la unificación de los distintos métodos de entrega, por lo que , esta idea combinada con utilización de **Docker**, minimizara el numero de servidores que necesiten para mantener activo el sistema, por consiguiente la demanda energética también disminuirá.

El ultimo punto al cual aporta Deliverit es el numero 8 **Trabajo Decente y Crecimiento Económico**, y las principales razones para ello son, que el sistema proporciona una educación de calidad, permitiendo al alumno la obtención de un buen empleo aportando a las distintas empresas mejores soluciones.

Bibliografía

[1] *Trello*. dirección: https://trello.com/es/about.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Thu Jul 01 23:57:02 CEST 2021
	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)