Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

DELIVERIT 1.0. BACK-END

Autor: Alberto Martín de Pablo
Tutor(a): Ángel Herranz

Madrid, Junio 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 Ingeniería Informática

Título: DELIVERIT 1.0. BACK-END

Junio 2021

Autor: Alberto Martín de Pablo
Tutor: Ángel Herranz
LENGUAJES Y SISTEMAS INFORMÁTICOS E INGENIERÍA DE SOFTWARE
ETSI Informáticos
Universidad Politécnica de Madrid

Agradezco a . . .

Mis profesores de la universidad, por el esfuerzo dedicado para que nosotros podamos aprender, en especial a Ángel Herranz por su apoyo para poder desarrollar este tfg y por lo bien que explica en sus clases.

A mis compañeros de estudios por que ha su lado la carrera se ha hecho mucho más amena.

Y por último de una forma especial a mi madre por haber luchado por mi futuro cuando ni yo mismo lo hacia.

Resumen

Una de las principales partes del aprendizaje de los alumnos de la ETSIINF es la realización y entrega de prácticas de tipos muy diferentes. Estas prácticas necesitan de una parte importante del tiempo del alumnado y del profesorado para sacarse adelante, con ellas los alumnos demuestran los conocimientos adquiridos a los profesores para que puedan ser evaluados.

Durante años esta entrega de prácticas ha sido independiente de cada departamento e incluso de cada profesor, forzando a alumnos y profesores a dedicar un tiempo extra en aprender cada sistema de prácticas.

DeliverIt (Entrégalo) es una aplicación web que surge con la misión de ayudar a resolver estos problemas, unificando en uno todos los sistemas de entregas de la facultad, proporcionando un tiempo extra a alumnos y profesores que se puede invertir en tiempo dedicado a la consolidación de los conocimientos de cada asignatura.

Para conseguir estos resultados DeliverIt debe ser una aplicación simple y fluida que se mejore año tras año, permitiendo la simplificación de su uso para alumnos y profesores.

Además de lo comentado anteriormente, gracias al uso de la tecnología Erlang/OTP obtenemos una aplicación escalable y tolerable a fallos. Junto con el framework Phoenix obtenemos una aplicación web fluida y fácil de usar. También empleamos el uso de contenedores de Docker que da mayor flexibilidad y seguridad a las entregas de los alumnos, al estar las mismas, separadas del sistema.

A lo largo de esta memoria se explicará las mejoras que se han desarrollado para la mejora de este sistema, además de detalles sobre la arquitectura y composición de la aplicación.

Abstract

One of the main parts of ETSIINF students' learning is the completion and delivery of practices of very different types. These practices require an important part of the time of the student and the teachers to get ahead, with them the students show the knowledge acquired to the teachers so that they can be evaluated.

For years the delivery of these practices have been independent of each department and even of each teacher, forcing students and teachers to spend extra time learning each system of those practices.

DeliverIt is a web application that arises to solve these problems, unifying in one all the delivery systems of the faculty, in addition to extra time for students and teachers that can be invested in time dedicated to the consolidation of the knowledge of the course.

To achieve these results DeliverIt must be a simple and fluid application that is improved year after year, allowing the simplification of its use for students and teachers.

In addition to the aforementioned, thanks to the use of Erlang / OTP technology and together with the Phoenix framework we obtain a fluid and easy-to-use web application. We are also able to use the Docker containers, which gives greater flexibility and security to student submissions, since they are separated from the system.

Throughout this report, the improvements that have been developed to improve this system will be explained, as well as details about the architecture and composition of the application.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Justificación

Actualmente, en la ETSIINF (Escuela Técnica Superior de Ingenieros Informáticos) de la UPM (Universidad Politécnica de Madrid) nos encontramos con multitud de sistemas de entregas, esto supone un problema tanto para profesores como para alumnos, ya que ambos deben perder un tiempo en aprender o construir un nuevo sistema de entrega, en vez de dedicar este tiempo al traspaso de conocimientos del profesor al alumno.

De esta problemática nace DeliverIt, como necesidad de un sistema de entregas común para toda la escuela y las diferentes asignaturas.

Puesto que se trata de unificar los sistemas de la escuela en uno solo, debemos tratar de adaptarlo para que sea lo mas amigable posible, tanto para profesores como alumnos, con el fin de que DeliverIt se convierta en el sistema de entregas preferido de una forma natural y no forzada, este trabajo se centra en reforzar esta facilidad de uso y aplicar mejoras y corrección de errores que pudieran aparecer en versiones previas.

1.2. Trabajo previo

Antes de comenzar en la mejora de DeliverIt, hemos realizado una investigación sobre diversas tecnologías asociadas y metodologías para optimizar el trabajo, y facilitar la cooperación entre los alumnos que hemos estado implementando las mejoras sobre DeliverIt.

Para ello hemos seguido una metodología Agile [1] adaptada a las necesidades de este proyecto, teniendo como punto de partida una pizarra virtual para simular la metodología Kanban[2] en Trello [3], que es una aplicación web en la que se han ido documentando las diferentes historias de usuario que hay que desarrollar. Esto nos ha permitido tener una visión global del proyecto en cada momento y ver el avance de cada desarrollador.

Para favorecer el desarrollo continuo de software en paralelo por los diferentes

estudiantes que hemos estado colaborando, nos adaptamos al flujo de trabajo de Gitflow[4].

Queda por hablar de las tecnologías que hemos estudiado para desarrollar el proyecto. DeliverIt es una aplicación escrita en Elixir [5], un lenguaje de programación funcional y concurrente que se ejecuta sobre la máquina virtual de Erlang [6].

Para la parte más visual hemos usado Phoenix [7], un framework sobre Elixir que dota a la web de mucha fluidez y simplifica bastante la tarea del desarrollador.

Con el fin de abstraer a DeliverIt de las diferentes necesidades de las asignaturas se han empleado los contenedores de docker para correr las prácticas con seguridad y de forma independiente al resto del sistema.

Todas estas tecnologías y metodologías han tenido que ser aprendidas y probadas por el alumno para poder llevar a cabo el proyecto constituyendo así una importante parte del tiempo empleado previo a realizar modificaciones sobre el proyecto.

1.3. Estado actual

Actualmente DeliverIt se encuentra en un estado funcional tras varios tfgs de alumnos que han ido mejorando muchas de las primeras funcionalidades, hasta llegar a un estado bastante óptimo de la aplicación pero con muchas mejoras y actualizaciones por delante.

Además, aunque DeliverIt es bastante moderno, al igual que cualquier otro software debe ser actualizado según van saliendo nuevas mejoras y correcciones en las tecnologías usadas o directamente nuevas tecnologías que van dejado atrás a las usadas en un primer instante para evitar que se convierta en un software inmantenible.

1.4. Objetivos

El principal objetivo de este proyecto es la mejora de la aplicación DeliverIt, tanto visualmente como en su núcleo para darle mayor consistencia, este objetivo se conseguirá resolviendo historias de usuario que son principalmente arreglo de errores y mejoras de la aplicación.

Por otro lado este proyecto también comprueba como funcionan las diferentes tecnologías aplicadas, especialmente el lenguaje Elixir y su framework phoenix, para crear una aplicación web escalable, tolerante a fallos, fluida y sencilla de usar.

También se aprovecha esta práctica para experimentar con las metodologías Ágiles y el trabajo en paralelo de varios desarrolladores, gracias a Trello, Git[8] y las reuniones periódicas con el tutor.

1.5. Metodología

Para el desarrollo del tfg hemos adaptado la metodología Agile a nuestras propias necesidades, nos hemos apoyado en la pizarra virtual de Trello, donde están las historias de usuario, con los "bugs" que hay que arreglar y las posibles mejoras que puede tener la aplicación.

Hemos mantenido una reunión semanal para poder ver el estado de las historias de usuario que hemos ido desarrollando y las posibles dudas que pudieran surgir con respecto a las mismas.

Para sincronizar los diferentes desarrollos existe un repositorio en Gitlab[9] sobre el cual los desarrolladores hemos podido ir subiendo nuestras partes del código.

1.6. Estructura de la memoria

A continuación paso a detallar los diferentes apartados que nos encontramos en esta memoria.

En primer lugar esta breve introducción para ir poniendo al lector al corriente de lo que se va a detallar más adelante.

En el capítulo 2 es un capítulo dedicado a la metodología, en él explicamos como hemos ido llevando a cabo el desarrollo de este proyecto y como ha sido la comunicación entre desarrolladores y con nuestro tutor.

A continuación en el capítulo 3pasamos a detallar todo el sistema de DeliverIt, las tecnologías con que está construido y la arquitectura general del proyecto.

Previo a la resolución de las historias de usuario tenemos que hablar en el apartado de 4 sobre todo el estudio y montaje del entorno de desarrollo local que replica al que actualmente esta en producción.

Una vez entendido como está construido el sistema, explicamos las historias de usuario, que son de una forma resumida, los enunciados de las tareas que hemos tenido que desarrollar para elaborar este proyecto.

Estas historias de usuario dan como solución un resultado que es lo hemos detallado en el siguiente capitulo, también explicamos como se llega a esa solución al problema, en los casos que ha sido posible encontrarla.

En el siguiente capítulo comentamos unas conclusiones personales de lo aprendido con este proyecto y de como ha sido la experiencia del mismo.

A continuación se hace un análisis de que se ha conseguido tras este proyecto y por último, nos encontramos la bibliografía en la que poder ampliar la información de muchas de las cosas de las que hablamos en esta memoria.

Capítulo 2

Metodología

Introducción a la metodología

Para facilitar el trabajo en equipo y el desarrollo continuo de código, nuestro tutor, Ángel Herranz Nieva, ha optado por emplear una adaptación de las metodologías ágiles para facilitar el trabajo en paralelo de los distintos alumnos, que hemos estado trabajando en este proyecto. A continuación detallo como lo hemos llevado a cabo.

2.1. Metodologías ágiles y tecnologías asociadas

En primer lugar hicimos una reunión con la explicación general del proyecto y de como íbamos a ir trabajando en la mejora del mismo, en esta reunión se nos asignaron las primeras tareas, que consistieron principalmente en el estudio de todas las tecnologías asociadas al proyecto y concluimos hacer una reunión semanal, que equivaldría a una "daily" para ver el estado de las tareas y poder solucionar posibles problemas que nos fuéramos encontrado y realizar las explicaciones necesarias.

Una vez terminado el estudio previo pusimos la aplicaciones en marcha en local y nuestro tutor nos enseñó la pizarra de Trello en la que teníamos colgadas las posible tareas que tendríamos que ir desarrollando.

Trello es una aplicación web en la que puedes definir varias columnas y colgar tareas para ir realizando e ir cambiando las mismas de columna según el estado de avance. En nuestro caso teníamos definidas las siguientes columnas de izquierda a derecha por donde íbamos moviendo las tareas:

- Product Backlog: Tareas que están pendientes de ser analizadas para pasarlas a Todo y que puedan ser implementadas.
- Todo: Son las tareas que han sido analizadas y están listas para ser implementadas por un desarrollador
- Doing: Son las tareas que ya se están realizando por un desarrollador,

Trello permite asignar tareas a un usuario o a varios, en esta columna las tareas ya han sido asignadas a un desarrollador.

- Ready to test: Mejoras o "bugs" que ya han sido desarrollados y que se están probando antes de hacer la subida del código a producción.
- Done: Sirve para mantener un histórico de las tareas que ya han sido realizadas.

Esta pizarra virtual sumada a las reuniones semanales, nos han permitido hacer un desarrollo de código fluido corrigiendo rápidamente los problemas encontrados.

Todo lo anterior no serviría de nada sin tener un repositorio bien gestionado por nuestro tutor, en el que cada alumno pueda ir subiendo sus partes de código y que puedan ser rápidamente unidas con el código del resto de compañeros y con la rama principal.

Para esto hemos empleado el sistema de control de versiones git y git flow para el flujo de trabajo, con un sistema de ramas y un repositorio en gitlab donde dejar el código, a continuación paso a explicar cada parte.

2.2. Control de versiones y flujo de trabajo

Cuando alguien desarrolla código puede cometer errores y a veces al intentar arreglar una funcionalidad estropear otras más críticas para la aplicación, por ello es importante usar un sistema de control de versiones, en nuestro caso se apostó por Git

Git es un software de control de versiones lanzado en 2007 y que a día de hoy es uno de los más usados en proyectos de todo el mundo, uno de los más importantes es el Kernel de Linux.

Entre las ventajas de usar Git se encuentran que facilita el trabajo cooperativo, reduce la cantidad de información que se envía, ya que solo envía los cambios y no todo el archivo y permite la creación y uso de ramas, esta última parte la explicaré con detalle más adelante.

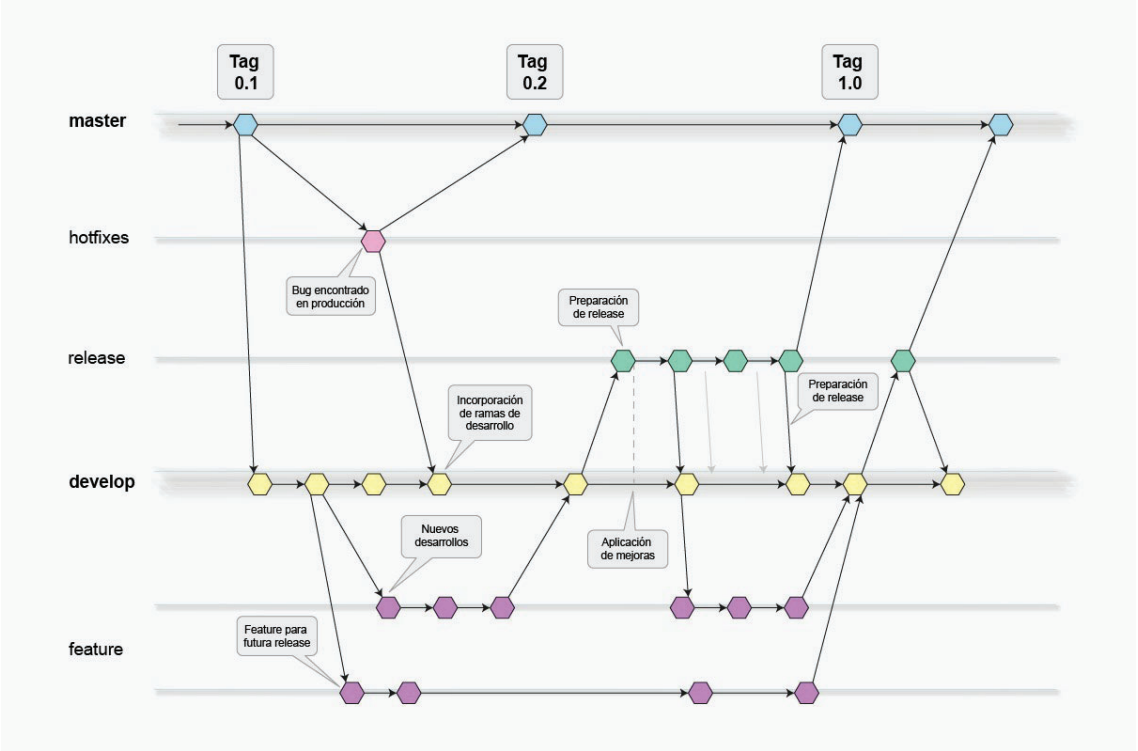
Para almacenar y mantener el código se ha empleado Gitlab que consiste en un repositorio de código en la nube con una interfaz web muy amigable para el desarrollador, también es uno de los repositorios más utilizados en el mundo junto con Github.

A menudo en los proyectos que tienen más de un desarrollador y empiezan a tener un tamaño considerable se hace uso de las ramas, estas son copias del proyecto que permiten mantener versiones estables mientras se van desarrollando mejoras y arreglos.

En nuestro caso contamos con una rama Master protegida, que es la rama en la que se sube el código ya probado y listo para poner en producción, esta rama debe ser estable y vigilada por el administrador del proyecto.

Para los nuevos desarrollos y las correcciones de posibles errores tenemos la rama de Develop y las ramas de Feature que vamos creando para implementar las correcciones en nuestro código y que una vez probadas subiremos a la rama Master.

A continuación pongo un esquema de las ramas que tenemos activas actualmente y en las cuales se esta trabajando en alguna corrección o mejora.



La creación de ramas, su destrucción y el "mergeo" con otras puede ser una tarea bastante compleja, para ello usamos Gitflow que nos simplifica bastante este proceso.

Capítulo 3

DeliverIt

Introducción a DeliverIt

Como ya hemos comentado anteriormente DeliverIt (Entrégalo) es un sistema creado para unificar y facilitar la entrega de prácticas de nuestra escuela.

Este sistema es muy útil para los alumnos, ya que permitirá que solo tengan que dominar un sistemas de entregas, habituándose a usarlo durante todo su aprendizaje independientemente de la asignatura que estén cursando.

Aún será más útil para el profesorado, ya que evitará que cada departamento o incluso cada profesor tenga que desarrollar su propio sistema de entregas con el tiempo que ello conlleva.

Por otro lado el unificar el desarrollo de DeliverIt está pensado para aumentar la seguridad en la entrega de prácticas, evitando que código potencialmente peligroso como pueden ser las prácticas de los alumnos puedan provocar problemas en los servidores de la facultad.

A continuación trataré de explicar con el mayor detalle posible como esta desarrollado DeliverIt y definiré de que partes se compone.

3.1. Tecnologías

3.1.1. Elixir

La elección de Elixir como lenguaje para este proyecto fue determinada por el grupo Babel ya que buscaban un lenguaje funcional con el que poder crear una aplicación web fluida y fácil de mantener.

Elixir permite incluirle el framework Phoenix que facilita el desarrollo web del cual hablaremos a continuación, además también permite la integración con la librería Ecto[10] que nos simplifica considerablemente todo el manejo de la base de datos, abstrayendo al desarrollador de consultas innecesarias.

Por otro lado la integración con Docker nos permite envolver a las prácticas en un entorno de ejecución seguro.

3.1.2. Phoenix

Phoenix es un framework para Elixir que nos permite crear aplicaciones web de una forma sencilla y fluida independizándolo de la parte "back", esto permite a desarrolladores "front" y "back" trabajar en paralelo de una forma más fluida.

3.1.3. Ecto

Es una librería que permite la comunicación de nuestra aplicación web creada en Elixir con la base de datos, independizando la lógica del acceso a datos y facilitando el trabajo del desarrollador.

3.1.4. PostgreSQL

La elección de la base de datos con PostgreSQL[11] nos permite una gran escalabilidad, esto es especialmente útil en sistemas de muchos usuarios como puede ser una web de la facultad, con alumnos y profesores insertando y extrayendo información continuamente de la base de datos.

3.1.5. Docker

La elección de Docker es especialmente crítica en este proyecto. Docker nos permite crear contenedores donde poder ejecutar prácticamente cualquier cosa, incluidos sistemas operativos, ejecutables o aplicaciones de cualquier tipo.

Esto es una pieza clave, ya que permitirá a los profesores ejecutar las prácticas de los alumnos en estos contenedores evitando la interacción con el resto del sistema, dotándolo de independencia de recursos y de mayor seguridad al ejecutarse todo en entornos virtuales y evitando la ejecución directa contra las máquinas de la facultad, que podrían ocasionar graves fallos de seguridad y escasez de recurso.

3.1.6. Entorno de desarrollo

Para poder desarrollar nuestros cambios y hacer pruebas sin tener que parar la aplicación en producción, usamos un entorno de desarrollo en nuestro computador, totalmente independiente de la aplicación con los datos reales, en donde agrupamos todas las tecnologías anteriores para facilitar el desarrollo.

Este entorno está montado bajo el sistema operativo Ubuntu, en él tenemos nuestro proyecto bajo el control de versiones de git conectado con el repositorio de Gitlab, para bajar o subir los cambios realizados en el proyecto por nosotros mismos o por otros desarrolladores.

Para el proceso de compilación y despliegue nos apoyamos en un Makefiel, en el cual indicamos los pasos que queremos que se realicen para poder poner en funcionamiento toda la aplicación en nuestro entorno.

Como IDE de desarrollo en este caso hemos elegido *Visual Studio Code* por su facilidad para ser usado sin necesitar de un aprendizaje previo excesivamente complejo. Otra buena opción podría ser *Spacemacs* aunque la curva de aprendizaje es extensa.

Para el desarrollo de documentación de este proyecto, incluida esta memoria hemos empleado *LaTeX* con el editor *TeXstudio*. Esto nos ha permitido hacer una memoria y una documentación perfectamente organizada bajo un mismo formato.

3.2. Arquitectura

3.2.1. Proyectos Umbrella

Cuando los proyectos empiezan a crecer demasiado, pueden volverse inmanejables, por la cantidad de archivos y ficheros de configuración que generan. Por ello resulta muy útil dividirlos en varias aplicaciones, esto nos lo permite mix[12] con los proyectos Umbrella[13].

Un proyecto Umbrella define una misma configuración para las diferentes aplicaciones que se creen bajo él. En nuestro caso estas aplicaciones o módulos del sistemas se diferencian en tres capas diseñadas para seguir el modelo vista-controlador, independizando de esta manera la parte visual de la aplicación del resto de lógica y acceso a base de datos.

3.2.2. Modelo vista controlador

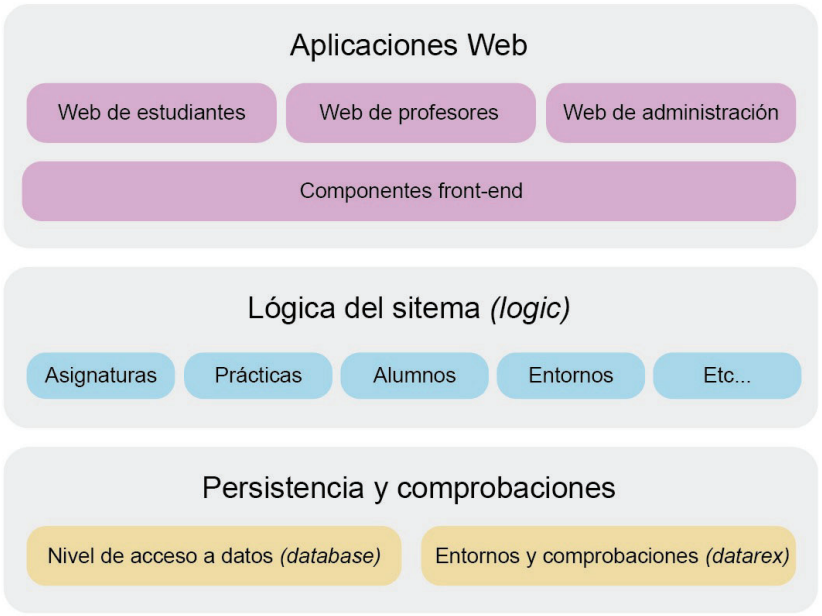
Gracias a Phoenix conseguimos poder seguir el modelo *modelo-vista-controlador*. Este modelo nos permite separar el moldeado de la vista, de los datos a mostrar y de la lógica que requiera el propio sistema.

Los diferentes controladores de la aplicación sirven a la vista todo aquello que necesite para pintar las páginas.

Por detrás de todo esto y transparente al usuario se encuentra el acceso a base de datos y el modelado de los datos, adecuándolos para ser mostrados al usuario final.

3.2.3. Esquema

En este esquema mostramos de una manera más visual la arquitectura de la aplicación web:



3.2.4. Vista

En la parte de *front-end* nos encontramos las aplicaciones divididas en función de los roles que van a seguir los usuarios de la aplicación:

- *admin*: la web de administración
- *client*: la web de estudiantes
- *components*; esta última es un sistema que proporciona funcionalidades para las otras dos.

Estas aplicaciones están creadas con el framework *Phoenix* que ayuda al desarrollador de Elixir a crear componentes web.

En el esquema podéis ver que se incluye una web de profesores, ya que la idea es que en un futuro se divida la web de administración en dos, una con las funciones propias de administración de la aplicación y la otra con las funcionalidades que permitirán a los profesores dar de alta: prácticas, alumnos y otros temas relacionados con la gestión de las asignaturas y sus entregas.

Por otro lado el esquema nos muestra, por debajo de las webs de estudiantes, profesores y administración, una cuarta de componentes front-end que sirve como apoyo para las otras tres y para unificar ciertas configuraciones de las aplicaciones.

3.2.5. Lógica

En la siguiente capa se detalla la lógica del sistema, agrupada en diferentes módulos para hacer la aplicación más sostenible en el tiempo. En esta capa se hacen los ajustes necesarios para el manejo de la información entre la base de

datos y la vista, permitiendo que cada capa sea completamente independiente de las otras.

Esta lógica del sistema esta dividida en funcionalidades, en el esquema las hemos agrupado en *Asignaturas*, *Prácticas*, *Alumnos*, *Entornos*, y *Etc...*.

3.2.6. Base de datos

Como ya hemos indicado la base de datos elegida ha sido PostgreSQL y nos hemos apoyado en la librería Ecto para el acceso a la misma, en ella nos encontramos las entidades y las relaciones que entre ellas forman la parte de persistencia de la aplicación.

Capítulo 4

Desarrrolo del trabajo

Introducción al desarrollo

Eb este capítulo se explica como ha sido todo el proceso de preparación para ser capaces de abordar los diferentes cambios en la aplicación de DeliverIt.

Explicaremos como se ha llevado a cabo el estudio de las tecnologías, la comunicación entre el tutor y alumnos y el aprendizaje de los conceptos que son necesarios para abordar este proyecto.

4.1. estudios previos

Antes de hablar de como se ha encontrado la solución para las historias de usuario, debemos hablar sobre todo el trabajo de aprendizaje e investigación que se ha llevado a cabo y que ha significado la mayor parte de este proyecto.

Comenzamos con un estudio de las metodologías ágiles para la organización general, buena parte de este conocimiento nos fue transmitido directamente por nuestro profesor, así acordamos como organizarnos desde un primer momento haciendo el equivalente a las *dailys* pero de forma semanal, estas reuniones semanales nos permitieron que la comunicación fuera mucho más fluida.

El estudio de las tecnologías empezó con el estudio de la programación en Elixir, en su mayor parte siguiendo la página de Elixirschool en la que viene buena parte de las particularidades de este lenguaje.

Para el estudio de Phoenix nos hicimos varias aplicaciones “dummies”, así descubrimos lo fácil que es poner en marcha una aplicación web con este framework operativa desde un primer momento.

También hemos tenido que aprender sobre Docker y el funcionamiento de sus contenedores, esto a nivel profesional es especialmente útil para pasar aplicaciones entre diferentes entornos, facilitando la integración continua.

4.2. Puesta en marcha del entorno de desarrollo

Para poner a funcionar la aplicación en local, nos hicimos una máquina virtual con Ubuntu en la que instalamos todas las dependencias que DeliverIt requiere, en el repositorio de Gitlab hay subido un fichero README que indica todo lo que es necesario instalar, en concreto las dependecias de Erlang, mix, Phoenix y Docker, además de algunas otras complementarias que vas necesitando según vas instalando dependencias.

Una vez que tienes todas las dependencias instaladas y el servidor de Docker funcionando la aplicación requiere de un reseteo de la base de datos, esto sirve entre otras cosas para meter un primer usuario en la base de datos para hacer pruebas.

Por el momento la aplicación dedicada a las funciones de administración de la propia aplicación y de las asignaturas y entregas es levantada en el puerto 4002, mientras que la de los alumnos es levantada en el puerto 4000.

Después de todo este estudio de las tecnologías necesarias y de ser capaces de tener la aplicación corriendo en nuestra computadora es necesario aprender a sentirse cómodo por la aplicación, para esto lo mejor es empezar a resolver las historias de usuario e ir aprendiendo con ellas, a medida que las desarrollamos, los entresijos de la aplicación.

Para ello se han ido eligiendo las historias en función de su dificultad y de los conocimientos necesarios para hacer su desarrollo.

En el caso de nuestro proyecto estas historias se han intentado centrar en la parte de la lógica y del acceso a datos de la aplicación, aunque en muchos casos han sido resueltas con un cambio en la vista de la aplicación.

Capítulo 5

Historias de usuario

Introducción a las historias de usuario

Las metodologías agiles pretenden evitar el exceso de papeleo y burocracia de las metodologías más clásicas y centrar el desarrollo del software en los deseos de las personas, consiguiendo un acabado final, más rápido y mucho más práctico que en las versiones más clásicas.

En la toma de requisitos clásica, el cliente y el desarrollador o desarrolladores acordaban una serie de requisitos que tenía que cumplir el código y eso era lo que se implementaba. Esto tenia múltiples desventajas, ya que muchas veces la idea que tenía el cliente y la que consideraba el desarrollador eran completamente dispares, dando un resultado final que no solo no era agradable para el cliente, si no que en muchas ocasiones tampoco era útil.

Para ello en las metodologías ágiles se apuesta por hacer un ciclo de desarrollo circular, en el que el cliente comenta algo que desaría que fuera capaz de hacer el código. Esto es una pequeña funcionalidad que puede ser desarrollada de una forma rápida y esta es entregada al cliente para su revisión y así, funcionalidad tras funcionalidad, se va construyendo el proyecto al completo.

Para conseguir esto, es necesario eliminar el exceso de documentación a realizar y establecer reuniones periódicas en las que se entregan las partes desarrolladas hasta ese momento, estos periodos de tiempo son conocidos como “sprints”. De esta forma el cliente puede ir viendo resultados tangibles y pidiendo mejoras y nuevas funcionalidades según el avance del proyecto.

Estas peticiones del cliente, tanto de correcciones como de nuevos desarrollos se establecen en forma de historias de usuario.

En nuestro caso, puesto que la aplicación ya está en producción, las historias de usuario nos vienen de los profesores y alumnos que la están utilizando.

A continuación detallo las historias de usuario en las que hemos estado trabajando:

5.1. Hora de la comprobación

> *Como alumno o profesor quiero saber a qué hora exacta se ha comprobado una entrega.*

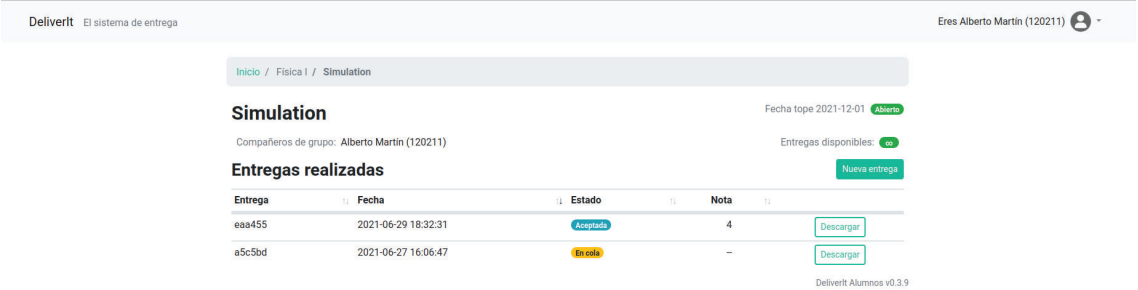
Al usar la aplicación nos encontramos con la fecha de comprobación de la entrega esta mal especificada, creemos que es porque esta en otro uso horario que no es el adecuado.

Hay que tener en cuenta que la hora que pone al comprobar una práctica debe ser la que hay en madrid en ese momento y no la que tenga el servidor en el que esta alojado DeliverIt o la de otra franja horaria distinta a la de madrid.

Esto tendría que ser revisado si exportamos la aplicación a otros países en el futuro.

5.2. Entrega con corrección pendiente

> *Como admin no quiero que un grupo pueda entregar si tiene una corrección pendiente.*



Esto es para evitar que un grupo pueda hacer una entrega que sobrescriba a la actual, antes de que esta sea corregida.

5.3. Estado de la entrega

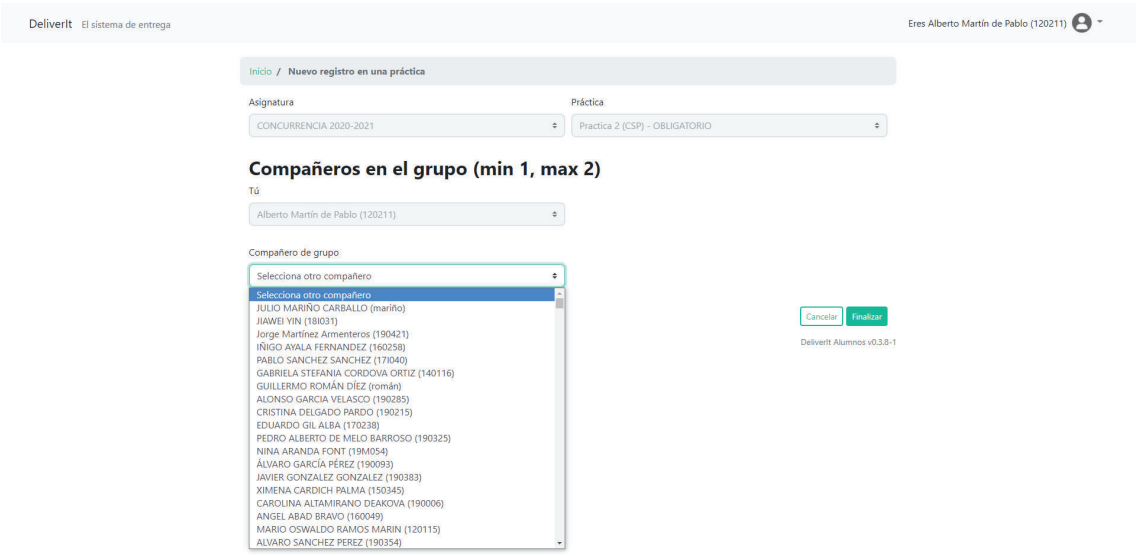
> *Como profesor/administrador quiero saber si una entrega ha fallado en su ejecución (contenedor) o si está pending, si está pending debería ejecutarse automáticamente de nuevo.*

Es importante conocer perfectamente el estado en el que se encuentran las prácticas cuando son ejecutadas.

Además hay que comprobar que si la entrega se ha quedado en el estado pendiente debe ser ejecutada automáticamente de nuevo.

5.4. Nombres de alumnos ordenados

Como alumno quiero que los nombres de mis compañeros al formar grupo aparezcan en un orden razonable"(alfabético).

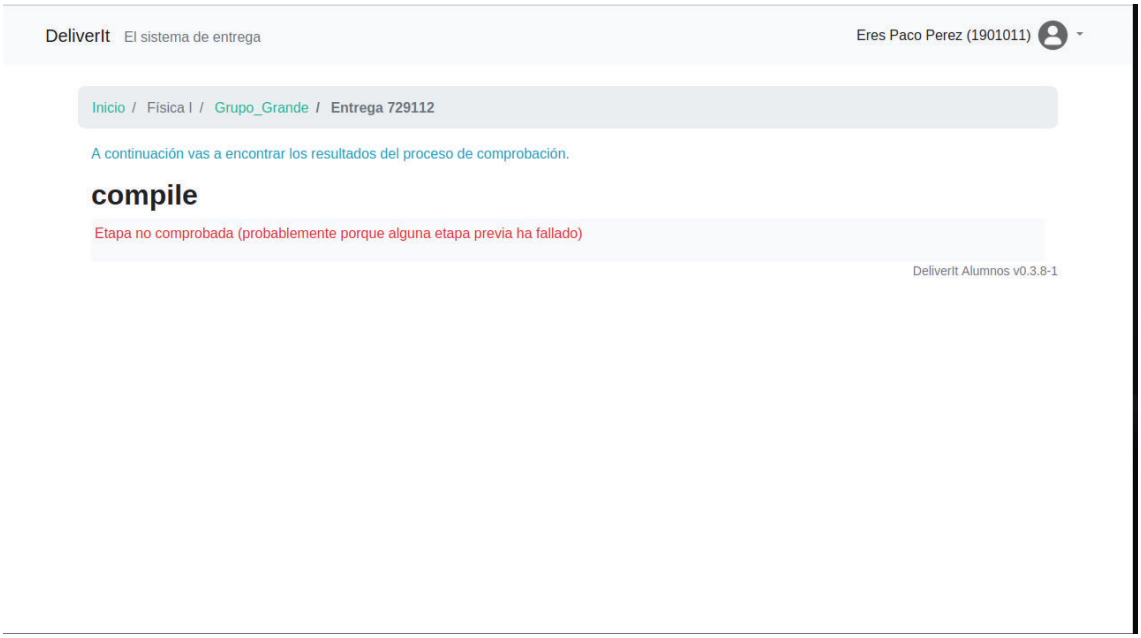


Cuando los alumnos tratan de elegir a sus compañeros de grupo el desplegable aparece sin ningún tipo de orden, esto supone un problema especial cuando hay muchos alumnos matriculados en una asignatura ya que se pierde mucho tiempo buscando a los compañeros de grupo.

Se propone como medida de corrección ordenarlos alfabéticamente.

5.5. Información más clarificadora.

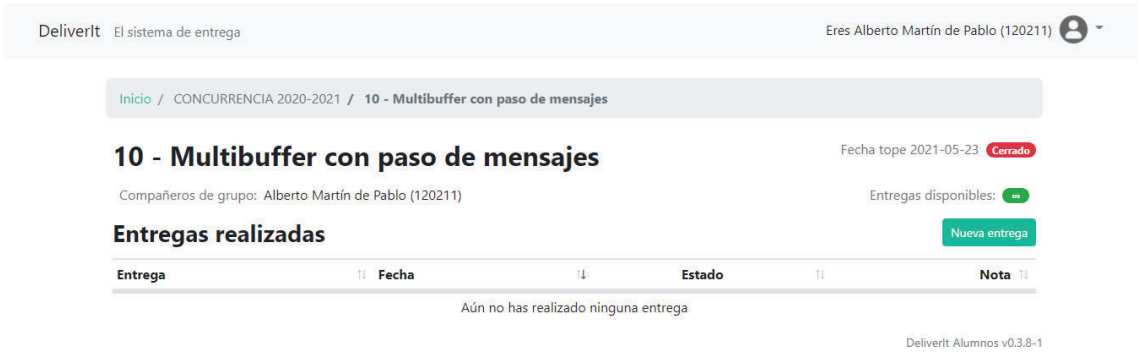
Modificación de lo que se muestre al realizar una entrega, para que la información que se imprime sea más clarificadora.



Cuando en la aplicación se realiza una entrega, se espera que esta pase por una serie de pasos, lo que se pretende con esta historia de usuario es que se muestren esos pasos.

5.6. Recuperar código de entrega.

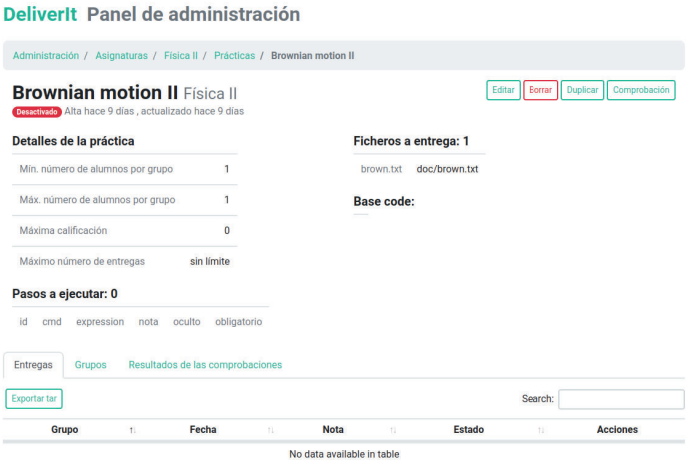
Como estudiante quiero poder recuperar el código de una entrega concreta.



Resulta útil poder tener el código de una entrega de práctica para poder buscarla luego por la aplicación.

5.7. Error en la navegación.

Existe un error al navegar en una determinada práctica.



La mejor forma de ir mejorando y actualizando la aplicación para hacerla más amigable y tolerante a fallos es recoger los comentarios de los que la van usando, cuanto más gente use DeliverIt, más probado estará todo el sistema.

Uno de los profesores que actualmente esta usando la aplicación en producción nos comenta que navegando por la aplicación, si se encuentra en una practica de algoritmos y estructura de datos en la parte superior le aparecen algunos enlaces separados por /:

También nos comenta que si hace click en practicas, no le lleva a las prácticas de su asignatura, sino a todas las prácticas, para ello se ha valorado eliminar esta navegación.

Capítulo 6

Implementación

6.1. Introducción a la implementación

En esta sección procedemos a explicar como hemos llegado a la solución de las historias de usuario que hemos ido desarrollando.

6.2. Hora de la comprobación

> *Como alumno o profesor quiero saber a qué hora exacta se ha comprobado una entrega.*

Para la corrección de la fecha hemos buscado donde se introduce la hora de la entrega y le hemos cambiado la franja horaria

6.3. Entrega con corrección pendiente

> *Como admin no quiero que un grupo pueda entregar si tiene una corrección pendiente.*

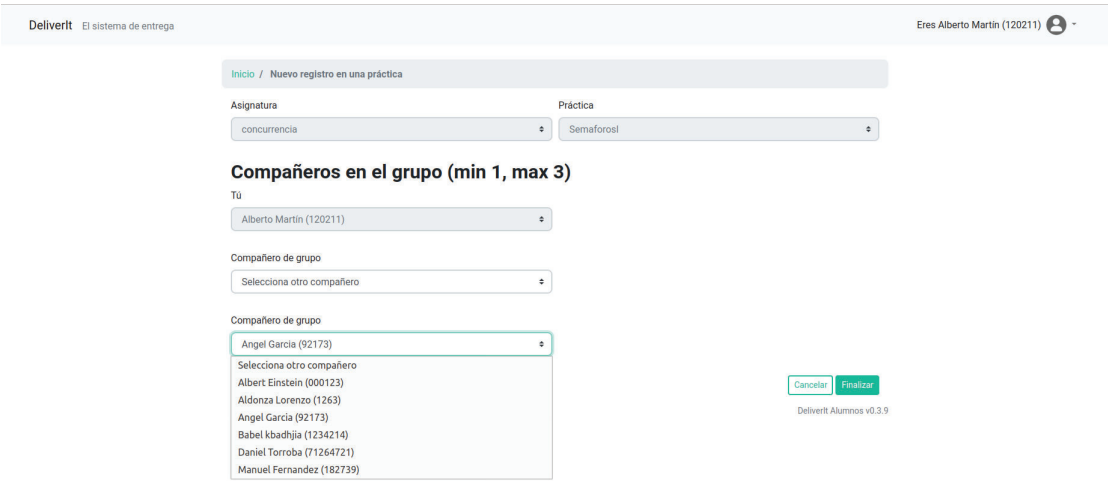
Para llegar a la solución consultamos en el momento de la entrega de una práctica si ya hay alguna pendiente, impidiendo en este caso la entrega.

6.4. Estado de la entrega

> *Como profesor/administrador quiero saber si una entrega ha fallado en su ejecución (contenedor) o si está pending, si está pending debería ejecutarse automáticamente de nuevo.*

6.5. Nombres de alumnos ordenados

> *Como alumno quiero que los nombres de mis compañeros al formar grupo aparezcan en un orden razonable"(alfabético).*



Para encontrar la solución hicimos una búsqueda de la frase “selecciona otro compañero” con el comando grep, para encontrar donde estaba la vista que

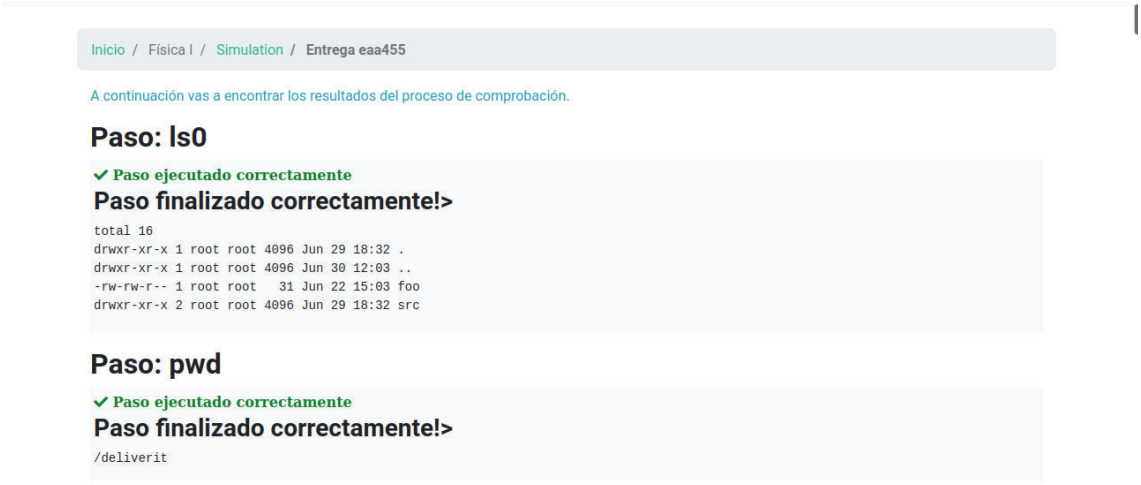
hacia referencia a esta página de la vista.

En un primer momento consideramos hacer una consulta ordenada de la base de datos, pero después de investigar un poco, encontramos una función que te ordena la lista *Enum*.sort(*tu_lista*), la implementamos en la vista.

```
<%= for _ <- gen_seq(1, @project.min_students) do %>
  <div class="form-row">
    <div class="form-group␣col-md-6">
      <%= label f, :student, gettext("Group␣mate") %>
      <%= select f,
                 :student,
                 [{gettext("Select␣other␣mate"), nil}
                  | Enum.sort(students_in_select)],
                 class: "custom-select",
                 required: true,
                 name: "request_params[students][]" %>
      <p class="text-danger"><%= error_tag f, :student %></p>
    </div>
  </div>
<% end %>
<%= for _ <- gen_seq(@project.min_students,
                     @project.max_students) do %>
        <div class="form-row">
         <div class="form-group␣col-md-6">
          <%= label f, :student, gettext("Group␣mate") %>
          <%= select f,
                     :student,
                     [{gettext("Select␣other␣mate"), nil}
                      | Enum.sort(students_in_select)],
                     class: "custom-select",
                     required: false,
                     name: "request_params[students][]" %>
          <p class="text-danger"><%= error_tag f, :student %></p>
         </div>
        </div>
<% end %>
```

6.6. Información más clarificadora.

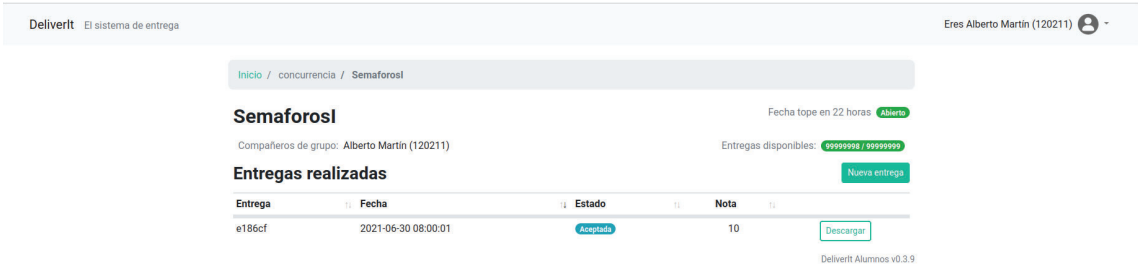
Modificación de lo que se muestre al realizar una entrega, para que la información que se imprime sea más clarificadora.



En este caso hemos mejorado el log que se muestra en la vista, a este log le hemos añadido unos mensajes en verde cuando se ha pasado correctamente y en rojo en el caso contrario, con unos stickers añadidos, además de modificar un poco el formato.

6.7. Recuperar código de entrega.

> *Como estudiante quiero poder recuperar el código de una entrega concreta.*



para este caso hemos agregado en la sección de las entregas realizadas el código de la práctica para que sea visible por el usuario.

6.8. Error en la navegación.

> *Existe un error al navegar en una determinada práctica*

La solución de esta historia de usuario ha consistido en impedir al usuario de la aplicación navegar por ese menú, quitandole esa posibilidad.

Capítulo 7

Conclusiones

7.1. Un proyecto para la universidad

Creo que una de las cosas que más me ha gustado de este proyecto es saber que estoy contribuyendo a modernizar la universidad, cada mejora en la aplicación de DeliverIt es tiempo de ahorro para futuros alumnos que les permitirá dedicarle más tiempo a las asignaturas a sus proyectos personales. También para los profesores que les permitirá dedicar más tiempo a preparar los conocimientos para los futuros alumnos, a investigar o igual que en el caso de los alumnos dedicarse a actividades extraescolares como estar con sus familias.

7.2. Tecnologías aprendidas

El manejo de un lenguaje funcional y concurrente como elixir y todas las tecnologías asociadas me ha aportado una nueva visión en la programación no solo en el ámbito escolar, si no también laboral, en el que estoy convencido que me servirá para introducir soluciones que sean útiles para las empresas.

7.3. Metodologías agiles y trabajo en equipo

Me ha encantado trabajar con mis compañeros y tutor con metodologías ágiles descubriendo como funcionan, es algo que echaba de menos en mi carrera profesional y que me va a ayudar como desarrollador y posiblemente como gestor de equipos en un futuro no muy lejano.

7.4. Análisis del impacto

Se espera que el desarrollo de este trabajo de fin de grado suponga una mejora sustancial en la mejora de la aplicación de DeliverIt.

A nivel personal, me ha supuesto el aprendizaje de muchas tecnologías que no dominaba y de una forma de trabajar muy especial, además del placer que ha

supuesto el trabajo con mi tutor y compañeros.

Espero que el DeliverIt acabe convirtiéndose en una aplicación de entregas de prácticas de informática de gran recorrido. Ahorrando presupuesto a la universidad pública española que tanto lo necesita.

7.5. Trabajo a futuro

Aún queda mucho camino por recorrer, me gustaría animar a futuros a alumnos a contribuir en este proyecto, por el futuro de la universidad y por ellos mismos, que considero aprenderán cosas muy útiles para su futuro profesional.

Bibliografía

[1] J. Shore y S. Warden, *The Art of Agile Development*. 2007.
[2] E. Brechner, *Agile Project Management with Kanban*. 2015.
[3] *Trello*. dirección: https://trello.com/es/about.
[4] V. Driessen, *Gitflow Workflow*. dirección: https://nvie.com/posts/a-successful-git-branching-model/.
[5] *Elixir*. Elixir. dirección: https://elixir-lang.org/getting-started/introduction.html.
[6] F. Hebert, *Learn You Some Erlang for Great Good! A Beginner's Guide*. 2013.
[7] *Phoenix framework*. Phoenix framework. dirección: https://hexdocs.pm/phoenix/Phoenix.html.
[8] Git, *Git*. dirección: https://git-scm.com/.
[9] Gitlab, *Gitlab*. dirección: https://about.gitlab.com/.
[10] *Ecto*. Ecto. dirección: https://hexdocs.pm/ecto/Ecto.html.
[11] *PostgreSQL*. PostgreSQL. dirección: https://www.postgresql.org/docs/12/index.html.
[12] *Mix*. dirección: https://hexdocs.pm/mix/Mix.html.
[13] *Umbrella*. dirección: https://elixirschool.com/en/lessons/advanced/umbrella-projects/.

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 23:39:03 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)