Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Deliverit 1.1. Experiencia de Usuario en un Sistema de Entrega de Prácticas

Autor: Darío Hernández García
Tutor(a): Ángel Herranz Nieva

Madrid, Febrero 2022

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 1.1. Experiencia de Usuario en un Sistema de Entrega de Prácticas

Febrero 2022

Autor: Darío Hernández García
Tutor: Ángel Herranz Nieva
Lenguajes y Sistemas Informáticos e Ingeniería de Sofware
ETSI Informáticos
Universidad Politécnica de Madrid

A mis padres y mi hermana, por haberse preocupado más por mi futuro que yo mismo la mayoría de las veces, insistiéndome en no abandonar día si y día también.

A mis amigos, por ese día en el que se rieron cuando no aprobé ninguna el primer cuatrimestre, haciéndome ver que no podía ponerles tan fácil reírse de mi y por hacerme recordar los días de estudio como algo bueno.

A Patri, por apoyarme siempre y alegrarse por mis aprobados incluso más que yo.

Y por último, a todos los que estuvieron y ya no están.

Resumen

El presente trabajo fin de grado (TFG) se enmarca dentro del desarrollo de DeliverIt. DeliverIt es un sistema de entrega de prácticas de programación. Entre las funcionalidades están la definición de prácticas y criterios de aceptación de las mismas por parte de los profesores, la entrega de prácticas y la comprobación de su aceptación.

La primera versión de DeliverIt, un prototipo, se puso en marcha en febrero de 2020 y se usó en la asignatura de Concurrencia. Desde entonces varios TFGs han evolucionado el sistema, algunas de estas evoluciones no se han llegado a probar suficientemente y por lo tanto no se han integrado el sistema en producción.

Este trabajo consiste en estudiar todas esas funcionalidades no integradas, corregir los errores que en ellas se puedan encontrar e introducir mejoras en la experiencia de usuario del alumno y del profesor. Las actuaciones más importantes han sido las siguientes:

- Análisis y cierre de ramas abiertas
- Implementación de nuevas funcionalidades
- Solución de bugs

Abstract

This Final Project (TFG) is part of the development of DeliverIt. DeliverIt is a delivery system for programming practices. Among the functionalities are the definition of internships and criteria for their acceptance by teachers and the and the delivery of internships and the verification of their acceptance.

The first version of DeliverIt, a prototype, went live in February 2020 and was used in the subject of Concurrency. Since then several TFGs have evolved the system, some of these evolutions have not been sufficiently tested and therefore the system has not been integrated into the system in production.

This work consists in studying all these non-integrated functionalities, correcting the errors that can be found in them and introducing improvements in the and introduce improvements in the user experience of the student and the teacher. The most important actions have been the following:

- Analysis and closure of open branches
- New funcionalities implementation
- Bugs solution

Tabla de contenidos

Capítulo 1

Introducción

En este primer capítulo se va a hablar sobre el estado actual del proyecto, la finalidad de mi trabajo en él y hacia donde se dirigirá en un futuro.

1.1. Justificación

Este TFG tiene el fin de continuar con el desarrollo y mejora de la herramienta DeliverIt, una aplicación de entrega de prácticas universitarias del ámbito informático (prácticas de programación), que tiene el fin de crear una plataforma que unifique el lugar de entrega de estas, ahorrando todo el tiempo que rodea a las mismas, tanto para profesores como para alumnos. El inicio de este proyecto se remonta 2 años atrás. A día de hoy la herramienta sigue necesitando correcciones a nivel de código, por posibles errores internos existentes en las últimas versiones, implementación de nuevas utilidades, mejoras a nivel de experiencia de usuario (solventando errores) y mejoras de UX.

1.2. Trabajo previo

Desde el momento en el que este trabajo es iniciado antiguos compañeros de la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) han empezado y desarrollado el proyecto, desde levantarlo con todas las tareas de programación de *Back-End*, para que todo funcione correctamente hasta las tareas más visuales de *Front-End*. Antes de comenzar a desarrollar este TFG se ha tenido que llevar acabo una labor de investigación para poder familiarizarse con Linux (sistema operativo con el que no había trabajado nunca), con Elixir y Erlang que son los lenguajes en los que se basa la aplicación DeliverIt y por último en Git que es la herramienta de control de versiones que se utiliza en el desarrollo de código.

1.3. Estado actual

Actualmente la plataforma se encuentra en un estado completamente funcional y estable.

De cara al usuario tiene 2 aplicaciones, la accesible por los alumnnos y la de administradores. En el inicio del proyecto se pretendía tener una tercera aplicación específica de profesores, pero por los contratiempos se dejó a un lado, hasta que esta se desarrolle los profesores están utilizando la aplicación de administradores.

1.4. Objetivos

Los principales objetivos de este TFG son los siguientes:

1. Identificación de características que aún quedan por añadir al sistema.
2. Identificación de experiencia de usuario insatisfactoria.
3. Identificación de deuda técnica.
4. Implementación de características nuevas.
5. Eliminación de experiencia de usuario insatisfactoria.
6. Eliminación de deuda técnica.
7. Elaboración de manuales de usuario y desarrollo.

Cada uno de estos objetivos conlleva varias tareas para poder alcanzarlos.

1.5. Estructura de la memoria

En este apartado se va a explicar cual será la estructura y división del TFG mediante capítulos.

- En el **capítulo 2** se definirán unos conceptos que se mantendrán a lo largo de la vida de este proyecto y que permitirá al equipo de desarrollo hablar un lenguaje común, evitando así confusiones.
- En el **capítulo 3** se hablará sobre la herramienta DeliverIt, profundizando en las tecnologías que se han utilizado para su desarrollo y qué mejorías aporta cada una de ellas en este.
- En el **capítulo 4** se hablará sobre el punto de partida en el que se encuentra el proyecto en el momento de inicio de este TFG, ramas abiertas, funcionalidades por integrar y funcionalidades nuevas que se vayan a implementar.
- En el **capítulo 5** se hablará sobre las aportaciones que he realizado a este proyecto en las labores de desarrollo.

- En el **capítulo 6** se hará un análisis del impacto de este TFG una vez finalizado en el proyecto DeliverIt. También, se dará un punto de vista del impacto que podría tener este proyecto respecto a la *Agenda 2030* según la *ODS*.
- En el **capítulo 7** se hablará sobre las conclusiones que se han sacado de este trabajo a lo largo de mi colaboración en el proyecto con el TFG y de los pasos futuros para los siguientes compañeros.

Capítulo 2

Vocabulario

Este capítulo es heredado del TFG del compañero Andrés Mareca [1], autor original del mismo. El fin de este capítulo es ofrecer un vocabulario de términos inspirado en el concepto *ubiquous language* de Eric Evans [2]. El vocabulario debe servir al equipo de desarrollo para 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.
>
> *Eric Evans*

Assistant (Profesor): 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 son los estudiantes de Doctorado los que se dedican a esta labor.

Base (Código de apoyo): Este término hace referencia al código que es necesario añadir dentro de un entorno, para poder ejecutar el código de un alumno.

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 de una organización.

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

Subject (Asignatura): Permite agrupar tanto a los alumnos como a los proyectos para una mejor gestión de los mismos.

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

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

Note (Comentario): 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 número total de tests respectivamente.

Capítulo 3

DeliverIt

En este capítulo se ofrece un resumen de lo que es el sistema Deliverit desde diferentes perspectivas: motivación, funcionalidad, arquitectura, diseño, tecnología y metodología.

3.1. Motivación

En la ETSIINF existen varias plataformas para gestionar las prácticas de las diferentes asignaturas, esto conlleva un tiempo de aprendizaje por parte del alumno para cada una de las plataformas y un tiempo de desarrollo de los profesores para poner en marcha la plataforma y su mantenimiento. Algunas de estas plataformas de gestión de prácticas llevan en funcionamiento varios años, alguna incluso más de 20 años. En esos 20 años ha surgido tecnología y experiencias de usuario que no son sencillas de integrar e implementar en dichos sistemas de entrega.

Para aliviar los problemas mencionados en el párrafo anterior y otros como la poca seguridad que ofrecen los sistemas de entrega con tanta antigüedad, hace casi 2 años, de la mano de Ángel Herranz y un antiguo compañero de la escuela, Andrés Mareca, se inició el proyecto DeliverIt. Su objetivo era crear una plataforma lo suficientemente atractiva como para que se convirtiera en un sistema de entrega eficaz tanto para profesores como para alumnos en la ETSIINF y, porqué no, que pudiera ser compartido con otras universidades e instituciones de enseñanza de la informática.

A día de hoy DeliverIt está funcionando y es estable gracias al trabajo de varios profesores y TFGs de alumnos, aún así sigue teniendo un gran margen de mejora que tenemos que explotar.

El presente TFG se centrará en los puntos que se comentan en el apartado 4.

Para contextualizar este TFG y que sea más facil entenderlo, primero se hablará sobre las funcionalidades que ofrece el sistema, en segundo lugar sobre la arquitectura de este, después sobre las tecnologías y diseño que se utilizan para que

este proyecto sea posible y finalmente se comentará la metodología de trabajo seguida para el desarrollo del sistema.

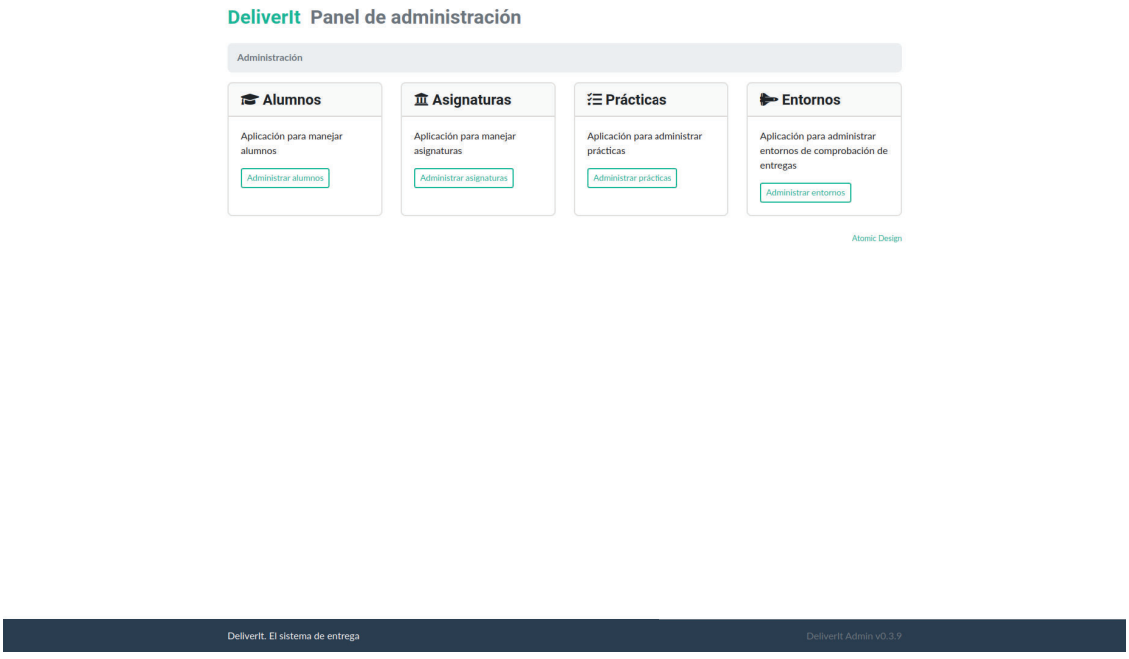
3.2. Funcionalidad

DeliverIt tiene las funcionalidades necesarias para una asignatura a la hora de ser administrada por sus profesores y las necesarias para los alumnos de dichas asignaturas. A continuación, se describen brevemente todas las funcionalidades disponibles hasta el momento.

Los administradores/profesores:

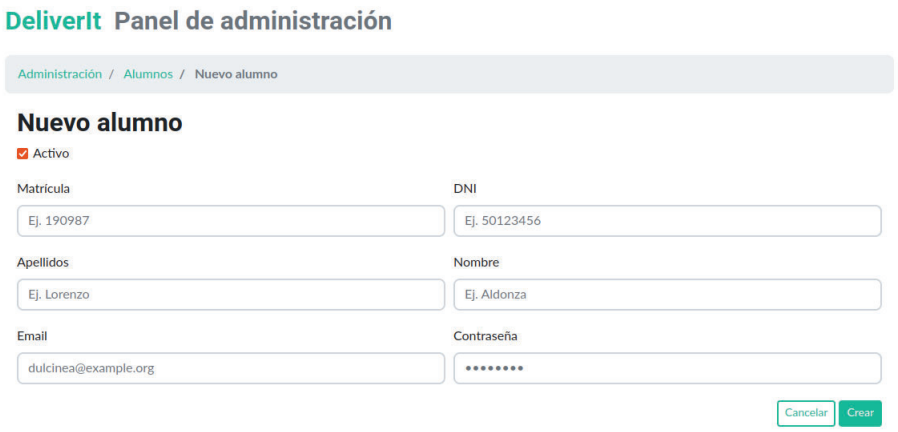
Desde su página principal o *Home* pueden ver todos los elementos que pueden administrar.

Como se puede observar en la captura de la herramienta los elementos posibles son «Alumnos», «Asignaturas», «Prácticas» y «Entornos».



Por no hacer demasiado extenso este punto del TFG se mostrará solo como es la administración de alumnos ya que en los demás elementos es lo mismo.


DeliverIt Panel de administración
Administración / Alumnos / 000123
Einstein, Albert 000123
Activo Alta hace 20 segundos , actualizado hace 20 segundos
Editar
Borrar
Información del alumno
Matrícula 000123
DNI 5013865
Nombre Albert
Apellidos Einstein
Email albert.einstein@student.ethz.ch
Último acceso --
IP de último acceso --
Asignaturas del alumno
Código
Nombre
2315772067 Física I
Dar de baja
Dar de alta en una nueva asignatura
DeliverIt. El sistema de entrega
DeliverIt Admin v0.3.9


DeliverIt Panel de administración
Administración / Alumnos / Nuevo alumno
Nuevo alumno
Activo
Matrícula
Ej. 190987
DNI
Ej. 50123456
Apellidos
Ej. Lorenzo
Nombre
Ej. Aldonza
Email
dulcinea@example.org
Contraseña
Cancelar
Crear

Las tareas que se podrán llevar acabo serán:

- Dar de alta un alumno
- Dar de alta un grupo masivo de alumnos desde un archivo `.csv`
- Borrar alumnos individualmente o masivamente

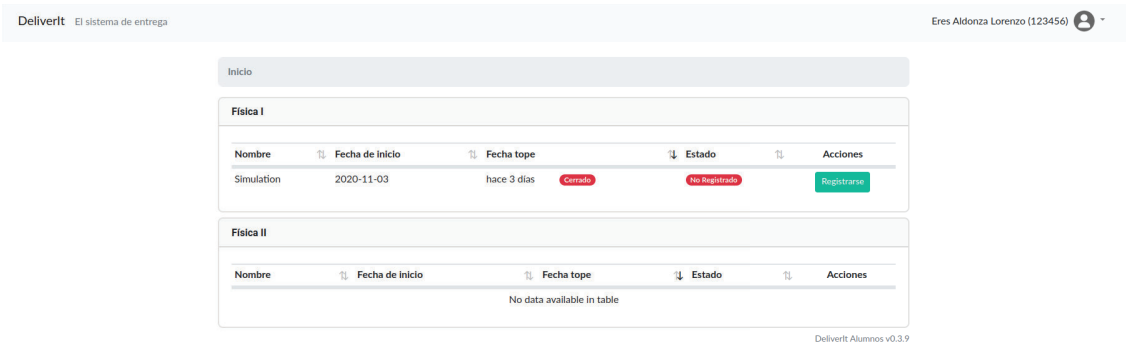
Como observación especial sobre las acciones a realizar sobre los alumnos, en el momento de dar de alta a estos, se les enviará un correo de aviso.

Estas acciones las podemos extrapolar a los tres elementos restantes que se muestran en la pantalla principal.

Los alumnos:

Al intentar ingresar en el sistema necesitarán haber sido dados de alta previamente por algún administrador/profesor.

En el momento de acceder a DeliverIt el alumno podrá observar las asignaturas en las que ha sido dado de alta (si es que lo está en alguna).



Cuando este tenga disponible una práctica en la que registrarse, le aparecerá dentro del bloque de la asignatura a la que pertenezca, dando paso a la pantalla de registro de grupo.

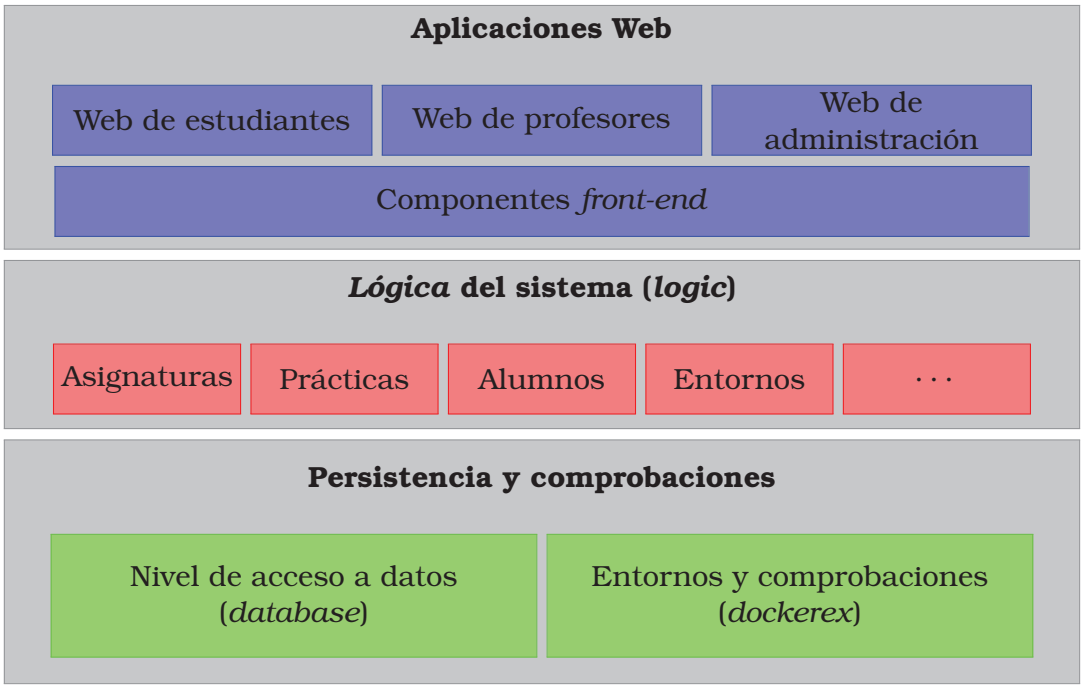
Si la práctica está configurada para hacerse en grupos, le permitirá añadir a sus compañeros, en caso contrario, el selector estará deshabilitado.

3.3. Arquitectura

Como se puede leer en el TFG de Andrés Mareca, la arquitectura de Deliverit ofrece a través de una aplicación Web todos sus servicios y funcionalidades, por lo que el sistema se divide en una parte de *back-end* (código que se ejecuta en el servidor) y una parte *front-end* (código que se ejecuta en los navegadores) [3].

En el inicio de la aplicación se pensó para que estuviese compuesta por tres aplicaciones distintas: una para alumnos, otra para profesores y una última aplicación Web que hará de panel de administración global del sistema, pero finalmente por problemas con el tiempo de entrega la aplicación de los profesores se omitió y utilizan la de administradores. Estas tres aplicaciones comparten tres librerías que se usan tanto para interaccionar con la base de datos, como para interaccionar con Docker y una última librería que contiene toda la lógica de negocio.

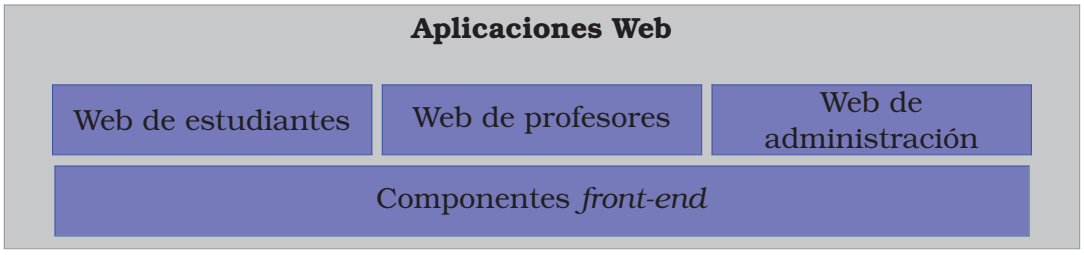
En este diagrama se puede observar como es la estructura del proyecto.



El lenguaje de implementación elegido para el desarrollo ha sido Elixir y las tres principales herramientas que han ayudado en la arquitectura han sido *Phoenix* (*framework Web*), *Umbrella* (estructura de aplicaciones) y *Ecto* (adaptador a bases de datos SQL). Se describen sus detalles a lo largo de los capítulos *Arquitectura* e *Implementación* del TFG de Andrés Mareca y en este trabajo en este mismo capítulo en las secciones 3.5 y 3.4.

Es necesario destacar que la biblioteca que conecta Elixir con Docker se ha decidido extraerla fuera del sistema ya que puede ser utilizada por cualquier otro proyecto fuera de Deliverit. Y además se entiende que con esa librería se puede ayudar a otros desarrolladores a usar Docker desde Elixir sin necesidad de programarlo ellos mismos.

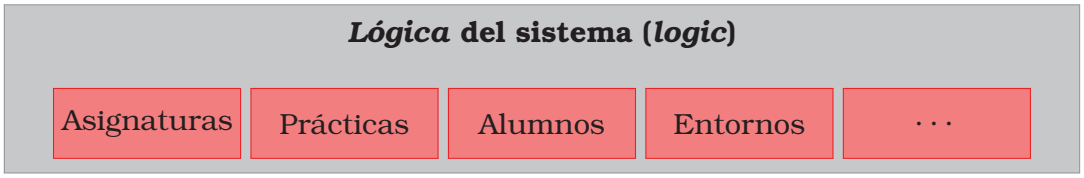
3.3.1. Aplicaciones Web



Para aislar las funcionalidades de cada tipo de usuario (alumnos, profesores y administradores) se decidió crear tres aplicaciones Web. Para ofrecer una uniformidad en la interfaz de usuario se ha diseñado un subsistema que ofrece componentes de la interfaz para que sean usados en todos los *front-ends* de las aplicaciones.

Las aplicaciones Web se han implementado utilizando el *framework* Phoenix en el lenguaje de programación Elixir. Ya se ha hablado en el capítulo 1 tanto de Phoenix como de Elixir. Aunque se entrará en detalles en la sección 3.5.

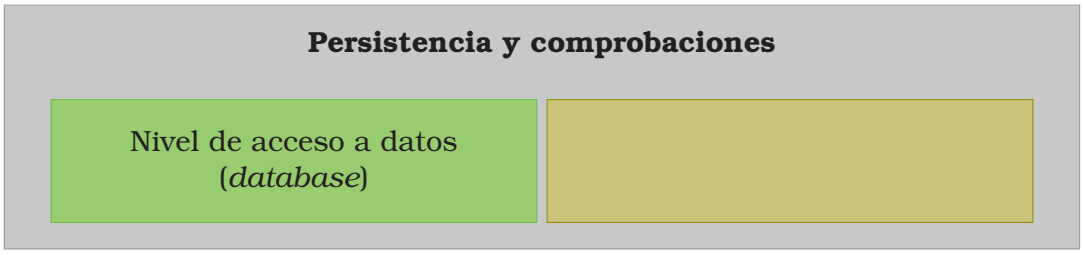
3.3.2. Lógica de la aplicación



Toda la lógica del sistema está encapsulada en este componente. Cada módulo en dicho componente se encarga de un contexto de la aplicación, desde los alumnos hasta las entregas pasando por asignaturas o prácticas.

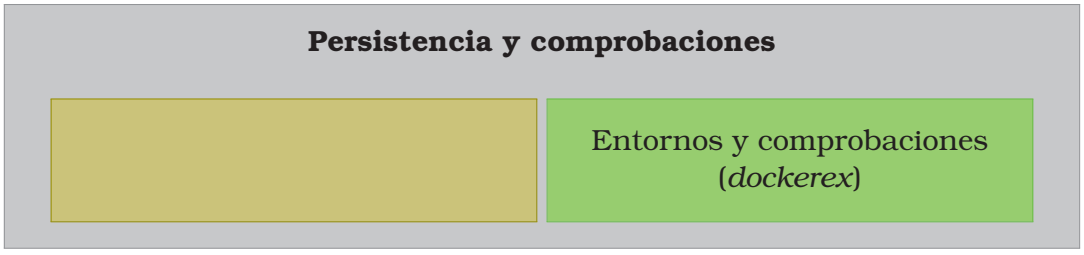
El componente es una aplicación Elixir con supervisores y construcciones OTP que permiten controlar la escalabilidad del sistema de forma sencilla.

3.3.3. Persistencia



Como se ha visto en la sección 3.4.2, el modelo lógico de datos está directamente basado en Ecto. Este componente es un nivel que aísla la lógica del sistema de Ecto, implementando funciones agrupadas por contextos, que hablan el idioma de esa lógica. Es un *data access layer* clásico. La convención del equipo es no realizar llamadas directas a Ecto desde los niveles superiores.

3.3.4. Comprobación de entregas



Finalmente, el último componente es el que permite ejecutar comprobaciones de forma segura sobre el código entregado por los alumnos. Todo el código de los estudiantes, y también el código de los profesores, se ejecutan en contenedores Docker (se explica a continuación en la sección 3.5). Para ello, se desarrolló una biblioteca en Elixir que es capaz de manejar las construcciones de Docker (imágenes, contenedores, volúmenes, etc.).

3.3.5. Diagrama de componentes

Este el capítulo del TFG de Andrés termina con el diagrama UML de componentes en el que el lector puede ver que las interacciones entre los componentes son llamadas de bibliotecas y aplicaciones de Elixir. Se muestra dicho diagrama a continuación:

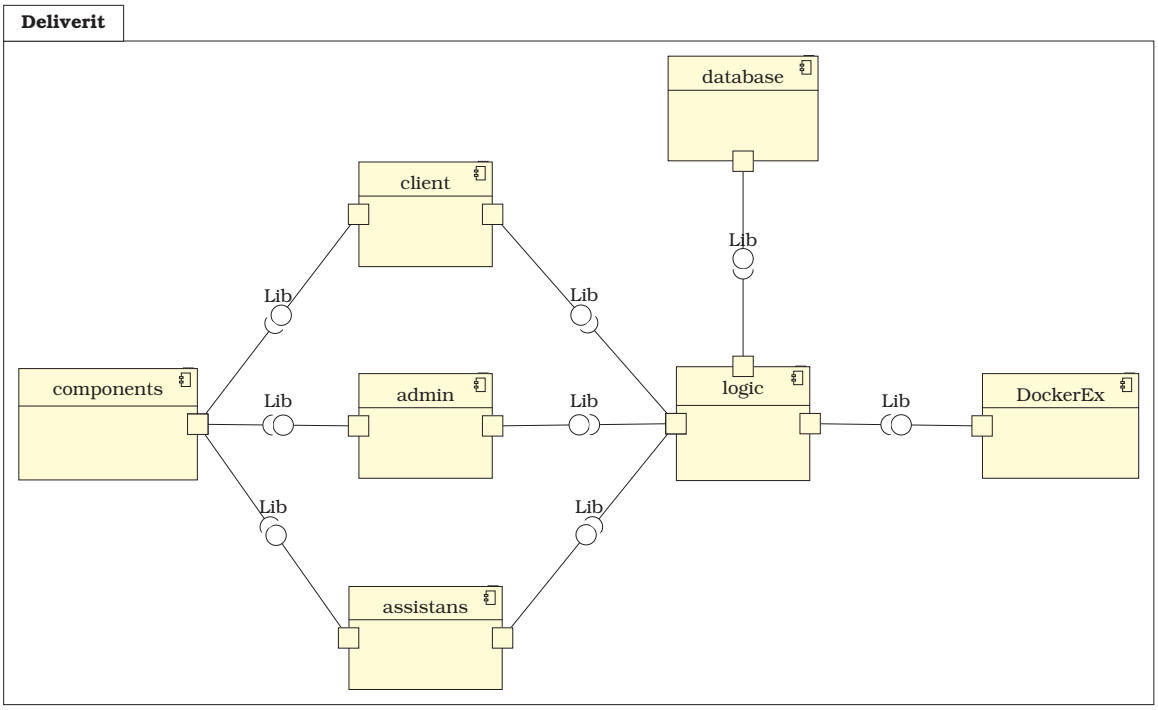


Figura 3.1: Diagrama de componentes

3.4. Diseño

En este apartado se hablará del diseño del sistema para que los lectores del trabajo lo puedan entender mejor. Este apartado está desarrollado más extensa y específicamente en el TFG de Andrés Mareca, el compañero que inició DeliverIt junto a Ángel Herranz.

En cualquier aplicación o sistema una de las partes más importantes son los datos que utiliza para mantenerla. En DeliverIt, el modelo de datos pasó por tres fases, primero el modelo conceptual, posteriormente el modelo lógico y finalmente el modelo físico.

3.4.1. Modelo Conceptual

Desde un primer momento en este proyecto se identificaron dos entidades claras, *alumnos* y *profesores* que serían parte de la composición del sistema.

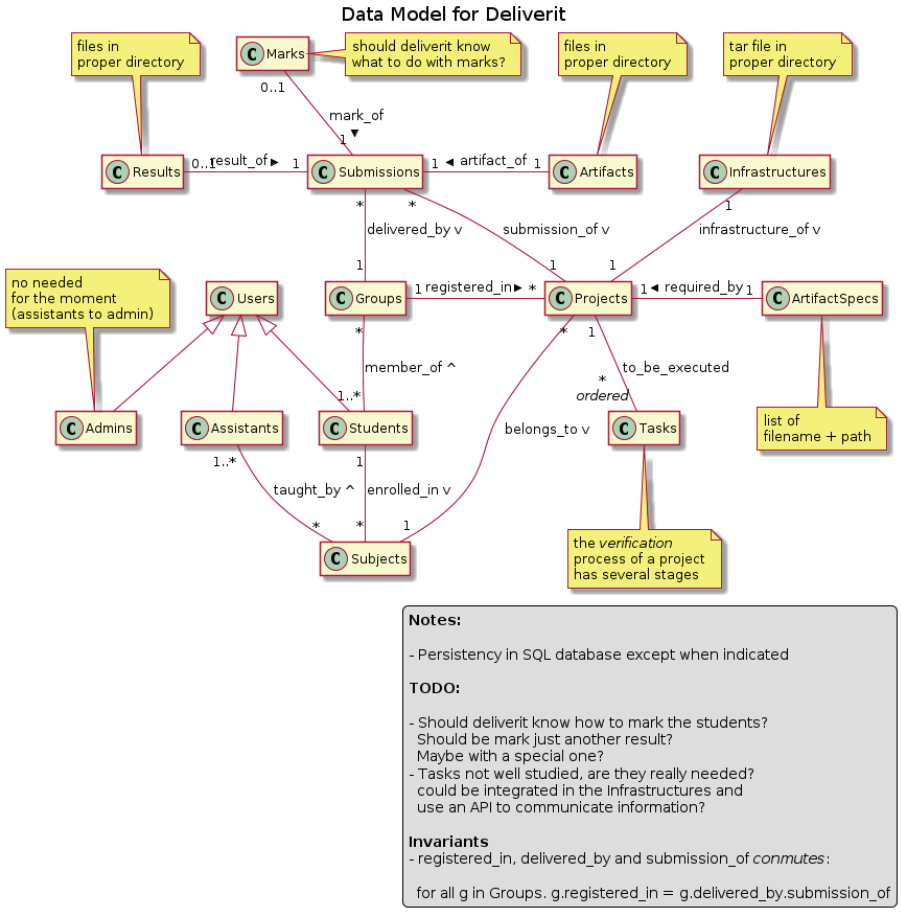
Como el sistema es una plataforma de entrega de prácticas, obviamente se tuvo que añadir otra entidad *práctica* que se encarga de almacenar todos los datos

correspondientes a la ejecución de estas, tanto por parte de los alumnos como por parte del sistema para realizar la entrega. Como había que almacenar todas las entregas que realiza un alumno hubo que declarar otra entidad *entrega*, que almacena lo entregado por el alumno y los datos relacionados.

Al no ser obligatorio que las prácticas se realicen de forma individual nació otra entidad *grupo*, que almacena toda la información necesaria para el control de este.

Uno de los principales requisitos del sistema es que pueda ser utilizado por múltiples asignaturas, por esto se tuvo que añadir otra entidad *asignatura*, que sirve para agrupar prácticas y facilita a profesores y alumnos la creación y entrega de dichas prácticas.

Desde el punto de seguridad, se decidió utilizar entornos estancos para evitar problemas por el código de alumnos y se añadió la entidad *entornos*.



Modelo Entidad Relación

El modelo entidad relación permite representar de una forma simplificada los componentes que forman parte de un proceso de negocio y el modo en el que se relacionan entre sí. Cualquier modelo entidad relación tiene tres elementos principales:

- **Entidades:** El modelo contará con una entidad por cada uno de los componentes.
- **Atributos:** Los atributos, componente fundamental de cada modelo entidad-relación, nos permiten describir las propiedades que tiene cada entidad.
- **Relaciones:** Con las relaciones se establecen vínculos entre parejas de entidades.

3.4.2. Modelo lógico y físico en Ecto

Elixir cuenta con una librería para la interacción con la base de datos, esta librería es Ecto. Ecto permite trabajar en todo momento con el modelo lógico abstrayéndose del sistema gestor de bases de datos que se encuentre por debajo.

Gracias a que Ecto trabaja directamente con el modelo lógico, se va a utilizar para ilustrar el modelo lógico de nuestra aplicación. A pesar de ser código, la metaprogramación usada en Ecto hace que las descripciones de los esquemas sean sencillas de entender por cualquier profesional.

Schemas y *Changesets*

A la hora de describir el modelo lógico, Ecto permite introducir construcciones llamadas esquemas (*schemas*). Cada esquema acabará siendo una tabla del modelo físico pero el nivel de descripción es más elevado. En particular, cada esquema, además del nombre, describe atributos (`field`) y relaciones del modelo conceptual (`has_many`, `many_many` and `belongs_to`). Dichas construcciones y su parámetros permiten a Ecto, además, regalar código de comprobaciones de consistencia y precarga de las entidades relacionadas.

La construcción *changeset* en Ecto permite construir o modificar entidades del nivel lógico con comprobaciones que anticipan posibles errores en la base de datos. Se usan además, para hacer conversiones de datos del exterior, por ejemplo, permiten introducir un entero en la base de datos a pesar de que en origen fuera una cadena de carácteres.

Subject

Como se puede observar en el diagrama entidad relación de la figura entidad-relación, la asignatura es el nexo de unión entre los profesores y los estudiantes. Gracias a la entidad asignatura se pueden agrupar las prácticas por asignatura y facilitar la navegabilidad de los usuarios dentro del sistema. La entidad asignatura permite que el sistema pueda ser utilizado a nivel de universidad. Los campos que se almacenarán serán: el nombre de la asignatura, un *flag* que permita activar y desactivar dicha asignatura y por último un código que la identificará internamente.

Assistant

Los profesores son uno de los grandes grupos que van a interactuar con el sistema, son una entidad tan grande que tienen su propio portal para realizar todas sus actividades. Los campos que componen a un profesor son:

- Nombre completo
- Un identificador único para el profesor (Se utilizará para iniciar sesión)
- Una contraseña

A parte de estos campos, se utilizarán otros para controlar tanto lo cambios de esa entidad como los accesos al sistema de esa entidad.

Teach

Se encarga de almacenar la relación que existe entre los profesores y las asignaturas, ya que un profesor puede enseñar varias asignaturas y una asignatura puede ser enseñada por varios profesores.

Student

Otro de los principales grupos que van a hacer uso de este sistema son los estudiantes, son una entidad tan grande que también tienen su propio portal para realizar todas sus actividades. Los campos que compone a un estudiante son:

- Nombre completo
- Un identificador único para el alumno (Se utilizara para iniciar sesión)
- Una contraseña
- Un correo (se utilizará para comunicarle el resultado de las acciones importantes)

A parte de estos campos, se utilizarán otros para controlar tanto los cambios de esa entidad como los accesos al sistema de la misma.

Enrolled

Se encarga de almacenar la relación que existe entre los alumnos y las asignaturas, ya que un alumno puede participar en varias asignaturas y una asignatura puede incluir varios alumnos.

Enviroment

Como uno de los puntos fuertes de este sistema es la ejecución en entornos independientes, se necesita guardar referencias a los entornos Docker que se han construido. Los entornos Docker pueden venir de tres fuentes distintas:

- De una imagen Docker desde un registry [4] de Docker

- Desde un respositorio git que contenga un Dockerfile [5] en el primer nivel
- Desde un fichero comprimido en formato *tar* que cumpla la misma condición que el repositorio git.

A parte del tipo, mencionado anteriormente, también se guarda un nombre, la salida de la creación del entorno, el estado en el que se encuentra y el identificador de la imagen Docker resultado.

Project

Las prácticas son el núcleo principal de la aplicación por lo que tienen su propia entidad. Las prácticas pertenecen a una asignatura y guardan información sobre:

- El entorno sobre el que van a ejecutar
- El límite tanto superior como inferior de estudiantes por grupos
- Nota máxima
- El nombre y la ruta de todos los ficheros que tiene que entregar un alumno
- Los comandos a ejecutar dentro del entorno
- El rango de fechas en los cuales estará disponible para entregar
- El número máximo de entregas
- El identificador a la imagen de Docker que contiene el entorno junto al código del profesor

Group

Como se ha mencionado anteriormente, las prácticas las realizan grupos comprendidos entre uno y N usuarios, por lo tanto necesitamos almacenar esa relación entre todos los usuarios que forman un grupo y una práctica. También se almacenará la última nota que ha obtenido el grupo junto un campo para almacenar anotaciones realizadas por el profesor.

Member

Se encarga de almacenar la relación que existe entre los alumnos y los grupos, ya que un alumno puede ser miembro de varios grupos y un grupo puede incluir varios alumnos.

Submission

Las entrega la realizan los grupos, de esta manera cualquier miembro del grupo puede realizar una entrega y ver el resultado de la misma sin necesidad de compartir las credenciales con el resto de miembros del grupo que ha realizado dicha entrega. Una entrega por lo tanto contiene el resultado en forma de nota de dicha entrega, una referencia a la práctica y al grupo que la ha realizado y

fuera del modelo de datos almacena la salida estándar de la ejecución de los pasos que componen esa práctica.

En cuanto a la estructura de este proyecto, Phoenix permite crear diferentes aplicaciones dentro de un mismo proyecto. Para el desarrollo de este sistema se establecieron bajo el dominio de Umbrella cinco aplicaciones, correspondientes a *Components*, *client*, *admin*, *database* y *logic*. A continuación, se va a explicar para que sirve cada uno de ellas.

- **Components:** esta aplicación no es una aplicación web independiente visible al usuario final, sino que proporciona *assets* estáticos y componentes web reutilizables para las aplicaciones Admin y Client. El sistema se divide en dos entidades, la de los profesores y la de los alumnos. Los alumnos podrán asociarse a la aplicación Client, los administradores del sistema y profesores en la aplicación Admin, en un principio iba a existir una tercera aplicación para los profesores denominada Teachers pero por la falta de tiempo se pospuso su desarrollo.

- **Admin, Client:** Como ya se ha comentado, el sistema dispone de dos aplicaciones accesibles al usuario final: la aplicación de Admin y la de Client. Aquí se desarrollan ambas interfaces de usuario. Las dos están respaldadas por la aplicación Logic, que como su nombre indica contiene la lógica del sistema. De ella hablaremos a continuación.

- **Logic:** como se acaba de comentar, en esta aplicación, se desarrolla toda la lógica empleada en el funcionamiento del sistema. Podría decirse que se trata de la más importante, dado que es la única aplicación Phoenix que se comunica con la base de datos del sistema, entre otras. De esta aplicación se hablará en el siguiente apartado. Una de las funcionalidades más relevantes de esta aplicación es la gestión de las entregas. 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 se definió el lanzamiento de un proceso independiente que se encarga de obtener de esa cola las entregas pendientes y posteriormente, evaluarlas.

- **Database:** su función principal es aislar los accesos a la base de datos del sistema mediante Ecto, tanto de escritura como de lectura. Además, contiene los modelos de datos, el esquema de la base de datos y sus migraciones, de las cuales se hablará más adelante.

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

3.5. Tecnología

Ahora se enumerarán y explicarán las tecnologías que se utilizan en el proyecto, además de las ventajas que nos aportan.

- **Elixir:**[6] El eje sobre el que gira este sistema, no puede ser otro que el lenguaje en el que se desarrolla. Se ha elegido elixir por ser un lenguaje funcional, moderno y dinámico. ¿Por qué este y no cualquier otro lenguaje con esas mismas características? Tenemos varias razones para justificarlo. Elixir está apoyado en la máquina virtual *Erlang* [7]. Erlang es un lenguaje de programación y un entorno de ejecución tolerante a fallos y concurrente, de lo que se aprovecha Elixir. Por esto, podemos asegurar que elegir este lenguaje es una de las mejores opciones para desarrollar un sistema como este ya que:
  - Es escalable.
  - Es concurrente.
  - Es tolerante a fallos.
  - Cuenta con numerosas librerías propias además de poder utilizar las librerías de Erlang.
- **HTML/CSS + Bootstrap:** [8] Puesto que DeliverIt es una aplicación web, se utilizó HTML y CSS para crear las páginas web y sus correspondientes estilos. Además, se ha decidido apostar por el framework de Bootstrap como apoyo, lo que nos proporciona numerosas funcionalidades y estilos que hacen que el diseño sea homogéneo y consistente. Otra ventaja de Bootstrap es su sistema de filas y columnas, que bien empleado, hace que DeliverIt sea *responsive*, es decir, que la aplicación se adapte de forma automática y sin mucho esfuerzo a distintos tamaños de pantalla. Teniendo en cuenta que muchos de los alumnos consultan los resultados de sus entregas desde un dispositivo móvil, esto hace que la interfaz sea mucho más amigable.
- **Phoenix:**[9] Phoenix es un framework de desarrollo para aplicaciones web escrito en Elixir. Está basado en el patrón modelo, vista, controlador (MVC). Este framework nos proporciona tanto un alto rendimiento en nuestra aplicación, como una gran productividad para los desarrolladores. Además, Phoenix también hace uso de Ecto [10], un lenguaje para poder interactuar con la base de datos de una forma bastante sencilla.
- **PosgreSQL:**[11] Es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.
- **Docker:**[12] Al igual que el lenguaje de programación, Docker es otra de las piedras angulares del sistema. Dado que este proyecto es un sistema de entrega de prácticas, debe estar preparado para los posibles imprevistos de cara a la subida de entregables por parte de los alumnos. Por esta razón se optó por el uso de contenedores, es decir, espacios aislados del sistema en los cuales ejecutar el código en cuestión. Docker es una de las opciones más populares para desempeñar esta tarea y cuenta con una biblioteca de

imágenes muy amplia, lo cual amplía las posibilidades de nuestro sistema. En DeliverIt esto se conoce como *entornos*.

3.6. Metodologías

En cuanto a la metodología de trabajo en el TFG ha sido común a la de todos los compañeros que han formado parte de este proyecto, tutelados todos por el jefe de proyecto Ángel Herranz.

Se han utilizado metodologías ágiles [13], como *Kanban*. [14] Estas metodologías surgieron en su día para poder hacer cambios de manera rápida y controlada en el desarrollo de proyectos de este tipo.

Product Backlog y ciclo de vida. El Product Backlog es un listado de todas las acciones necesarias en el sistema y que tienen que implementarse. Estas acciones necesarias tienen diferentes niveles de prioridad para que así los desarrolladores puedan adelantar trabajo de cara a los siguientes ciclos. El dueño del producto es el responsable último de mantener este listado actualizado.

El equipo de desarollo tiene la obligación de asegurar que las tareas que aparecen en el Product Backlog poseen la información necesaria para su desarrollo en los ciclos correspondientes.

Se estableció el siguiente ciclo de vida para las unidades de trabajo:

- **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 poder llevar a cabo esto, se ha usado la herramienta **Trello**.

Trello [15] es una herramienta online que nos permite la organización y gestión de tareas simultáneamente a varios usuarios, mediante tableros. Dentro de cada tablero se pueden crear diferentes tareas para cada usuario y añadir información específica a cada una de estas.

Ciclos de desarrollo cortos. Se establecieron reuniones periódicas cada semana y ciclos de desarrollo de la misma duración. Estos ciclos de desarrollo normalmente reciben el nombre de *Sprints*.

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.

En último lugar, hablaremos de la herramienta que se ha utilizado para el control de versiones de nuestro código, algo imprescindible cuando se trabaja con distintos desarrolladores en un mismo proyecto.

Gitlab [16] es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git, uno de los sistemas de control de versiones más extendido y utilizado.

Para llevar acabo el desarrollo de la aplicación, se decidió adoptar el flujo de trabajo *Git-Flow* [17]. Git flow es un flujo de trabajo basado en Git que brinda un mayor control y organización en el proceso de integración continua. El modo de trabajar usando git flow es el siguiente.

Deben existir dos ramas principales:

- **Master.** Esta rama debe contener todos los cambios que estén preparados para ser llevados a producción.
- **Develop.** Esta rama debe contener el código que conformará la siguiente versión del proyecto.

Por otro lado, se trabaja con otras ramas auxiliares:

- **Feature:** Estas ramas se originan a partir de la rama develop y se utilizan para desarrollar nuevas características del sistema. Una vez se ha terminado el desarrollo, se incorporan los cambios a la rama develop.
- **Release:** En estas ramas se llevan a cabo los últimos ajustes y se corrigen los bugs menores que se encuentren antes de hacer el pase a producción.
- **Hotfix:** Estas ramas se utilizan para corregir errores críticos o inesperados dentro del sistema.

Adicionalmente se ha utilizado **Gitkraken** [18], un cliente de git que nos permite gestionar nuestros repositorios con una interfaz gráfica bastante fácil de entender, de esta forma ver los cambios que se introducen en cada commit es mucho más rápido.

Capítulo 4

Punto de Partida y Objetivos

En este capítulo se hablará del punto en el que se encuentra el proyecto al inicio del TFG y los objetivos en los que se va a trabajar dividiéndolo en ramas.

Además del trabajo que tendrán estas ramas ha habido un trabajo de *checking* en funcionalidades que se implementaron con anterioridad a este TFG y se ha tenido que comprobar el correcto funcionamiento de todas y su corrección en el caso de que haya sido necesario. En el capítulo 5 se entrará en detalles sobre cuales se han corregido y cuales se han implementado.

De esta forma se podría dividir este trabajo en dos partes:

1. Análisis y cierre de ramas abiertas.
2. Nuevas funcionalidades y solución de *bugs*.

4.1. Análisis y cierre de ramas abiertas

En el momento que se inicia este TFG las ramas abiertas en el Git del proyecto son: ***master*** y todas las ramas que cuelgan de ***develop***.

Master es la rama principal, donde se encuentra el código de producción, es decir, el código que está funcionando ahora mismo.

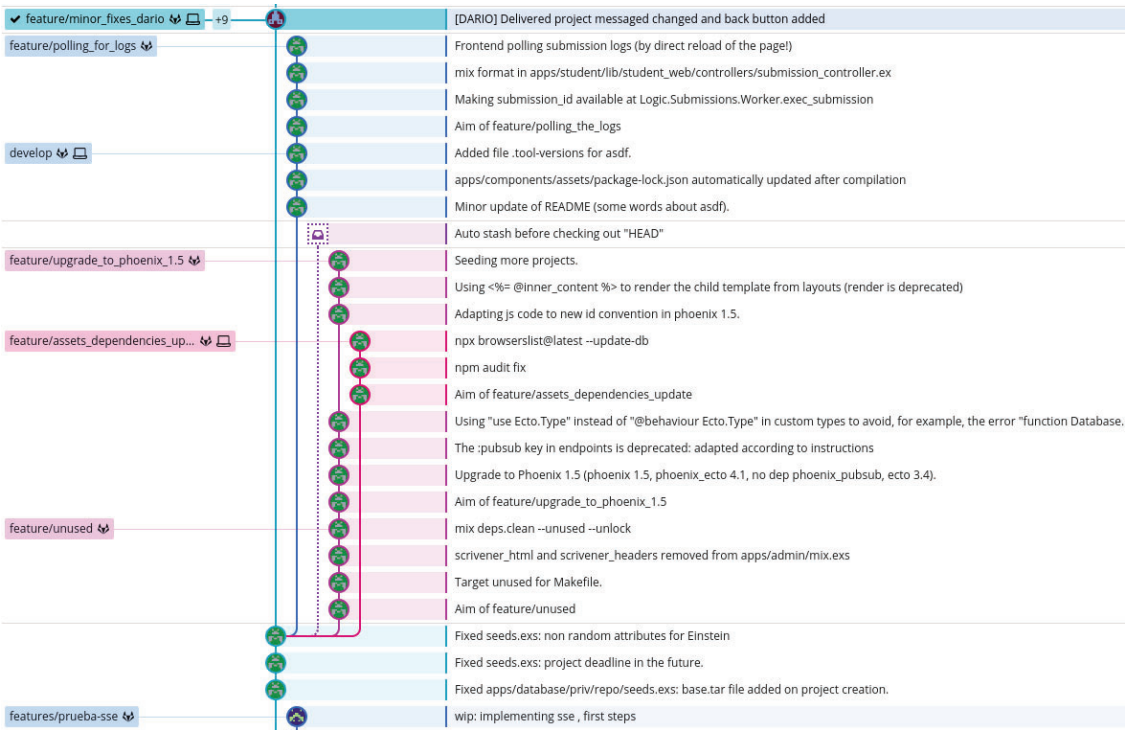
La rama ***Develop***, es una rama a partir de la cual nacen otras subramas, una para cada nueva implementación que se este desarrollando, de esta forma no interferirá con la versión actual y estable que está funcionando en el servidor.

Actualmente nos encontramos colgando de develop las siguientes ramas:

- `feature/upgrade_to_phoenix1.5` rama de actualización del *framework* Phoenix a su versión 1.5.
- `feature/assets_dependencies_update` rama de soporte de las dependencias *Front-End*.
- `feature/unused` rama de identificación de código inutil.
- `feature/polling_the_logs` rama para dar información post-entrega.

- `feature/liveview_sandbox` rama para pruebas.
- `feature/upgrade_to_phoenix1.6` rama de actualización del *framwork* Phoenix a su versión 1.6.
- `feature/minor_fixes_dario` rama de corrección de errores menores.

A continuación se puede ver el árbol de cambios de Git para las ramas que afectan a este TFG, creándose un nivel por cada rama dependiendo de la dependencia y una línea por cada *commit* dentro de la rama.



Ahora se explicará que finalidad tienen las ramas mencionadas anteriormente:

`feature/upgrade_to_phoenix1.5` y `feature/upgrade_to_phoenix1.6`

Estas dos ramas tienen el mismo origen y finalidad, surgen con el lanzamiento de una nueva versión de Phoenix y tienen la finalidad de actualizar de la herramienta Phoenix a una versión más actual para así poder utilizar e implementar funcionalidades más punteras en el proyecto. La rama que hace referencia a la versión 1.5 está obsoleta por lo que se cerrará para implementar la versión 1.6 directamente y así ahorrar trabajo.

`feature/assets_dependencies_update`

Esta rama se crea para mantener actualizadas todas las dependencias utilizadas en la parte *Front-End* del proyecto implementadas con *Javascript* y *CSS*.

feature/unused

Esta rama se crea para identificar todo el código inservible o que no se utiliza en el proyecto y poder eliminarlo sin causar ningún problema en la rama que está en funcionamiento.

feature/polling_the_logs

Esta rama se crea con la finalidad de mostrar los logs obtenidos tras la corrección de una práctica en la página de entrega de los alumnos.

feature/liveview_sandbox

Esta rama se crea con la finalidad de permitir tener un lugar seguro en el que probar funcionalidades visuales sin comprometer la rama que está en funcionamiento.

feature/minor_fixes_dario

Esta es una de las últimas ramas creadas en el proyecto, la finalidad de esta es agrupar soluciones de problemas menores que no tengan el peso suficiente para tener una rama propia.

Dentro de cada rama encontramos todas las carpetas y archivos necesarios para que el sistema funcione, además el archivo CHANGELOG.md describe todos los cambios que hay en cada versión nueva.

4.2. Nuevas funcionalidades y bugs

Tras el análisis del sistema se han detectado una serie de bugs y funcionalidades que se necesitan solucionar/implementar respectivamente.

feature/register_student_subject

Esta rama se crea tras identificar un comportamiento mejorable de la aplicación de los administradores, a la hora de dar de alta alumnos en asignaturas solo permite añadirlo de una en una, cambiando a la página principal de ese alumno automáticamente sin opción a añadir al alumno a varias asignaturas. El fin de esta rama será investigar por qué está funcionando así y solucionarlo.

feature/delivery_page_improvements

Esta rama se crea tras la identificación de un error en la página de entrega de prácticas de los alumnos, en la cual se mostraba un mensaje genérico hubiera logs o no que mostrar y no tenía ningún botón para volver a la página anterior. El fin de esta rama será implementar las mejoras mencionadas.

feature/partial_translation_warning

Esta rama se crea tras la identificación de un fallo que presentaba la aplicación de los administradores en ciertas operaciones. Este fallo muestra un mensaje parcialmente traducido al idioma en el que se está utilizando DeliverIt y el resto en inglés. El fin de esta rama será corregir dicho error.

feature/sticky_footer

Esta rama se crea por la necesidad de implementar un footer fijo en la aplicación de los alumnos, al igual que se muestra en la de administradores.

A nivel de código esto se hará modificando los estilos de *Front-End* añadiendo el código requerido por un *footer*.

feature/updating_csv_data

Esta rama se crea para mejorar la información que se aporta en el archivo `.csv` a la hora de hacer una carga masiva de alumnos, puesto que a fecha de inicio de este TFG es algo escueta.

A nivel de código habrá que trabajar con el lenguaje de **CSV**, similar a un Excel pero en plano.

feature/trimar_inputs

Esta rama se crea por la identificación de un problema con el inicio de sesión de los alumnos, puesto que si se introduce un espacio en blanco por error antes o después del nombre no permite iniciar sesión. El fin de la rama es solucionar esto para que se ignoren esos espacios en blanco.

Esto a nivel de código se hará mediante un tratamiento de *Strings*.

feature/logs_steps

Esta rama se crea para mejorar la pantalla de logs que se muestra a los alumnos cuando hacen una entrega. Se intenta que sea más ilustrativa e intuitiva, añadiendo símbolos para identificar que pruebas han pasado satisfactoriamente y cuales no.

Esto a nivel de código se hará asociando el mensaje de éxito con un símbolo tic y el mensaje de fallo con un símbolo cruz.

feature/admin_project_export

Esta rama se crea por la necesidad que pueden tener los profesores de recuperar el código base que han utilizado para una práctica para un uso futuro. Esta funcionalidad aunque podría ser básica, no está implementada.

Capítulo 5

Desarrollo

En este capítulo se ofrecerá un resumen de las tareas de desarrollo llevadas a cabo en este TFG, implementación de nuevas funcionalidades y corrección de errores.

Para llevar un trabajo organizado y saber hacia donde se dirije el proyecto en todo momento, se ha utilizado un tablón de **Trello** (explicación de qué es Trello en la sección 3.6) en el que se representa cada una de las necesidades mediante tarjetas, de esta forma, es más fácil llevar un control del trabajo que aún está por realizar y saber qué tiene mayor prioridad.

A continuación, se entrará en detalle de cada una de las tareas realizadas, haciendo referencia al nombre de la rama que se ha creado para tratarlas:

`feature/sticky_footer`

Debido a la inexperiencia con el lenguaje Elixir esta fue la primera tarea de desarrollo que llevé acabo, puesto que la dificultad y el nivel de desarrollo necesario era el más bajo dentro de las tareas que tenía que realizar.

Esta tarea pertenece al desarrollo *Front-End* puesto que lo único necesario para llevar acabo esta tarea ha sido modificar los estilos de la aplicación.

Para mantener la consistencia y apariencia entre la aplicación de estudiantes y administradores se han utilizado los mismos estilos para esta implementación.

El archivo modificado para realizar los cambios pertinentes es `app.html.eex` y se encuentra en la ruta `deliverit/apps/student/lib/student_web/templates/layout`

A continuación se puede observar el código añadido:

```
<footer class="footer␣mt-auto␣py-3">
  <div class="container␣d-flex␣justify-content-between">
    <span><%= gettext("DeliverIt.␣The␣turn␣in␣system") %></span>
    <span class="text-muted">
      <%= gettext("DeliverIt␣Student␣v%{version}",
```

```
                    version: Application.spec(:student, :vsn)) %>
    </span>
  </div>
</footer>
```

feature/register_student_subject

Esta tarea tiene un nivel superior a la anterior puesto que hay que entrar en la modificación del comportamiento del sistema, para ello primero fue necesario localizar el archivo que definía el comportamiento de la pantalla en la que se añaden las asignatura a un alumno en concreto.

Una vez localizado el archivo habrá que modificar la lógica del evento que tiene lugar al pulsar el botón *Submit* de modo que no nos envíe a la página principal.

feature/delivery_page_improvements

Esta tarea consiste en mejorar la página post-entrega de una práctica de los alumnos, al momento de iniciar este TFG, se entregase lo que se entregase el sistema devolvía un mensaje genérico que muchas veces no hacía referencia a la práctica entregada. Además, no existía ningún botón «Volver» para ir a la página anterior.

Para implementar esto se tuvo que incluir en el código de la pantalla un operador condicional, que en caso de no devolver logs la entrega, mostrase un mensaje y si devuelve logs mostrase otro diferente.

Finalmente, el botón para volver a la página anterior se implementa para evitar la necesidad de ir a la página *Home* del sistema si quieres salir de la página de correción.

Los archivos modificados para llevar a cabo esta tarea son los siguientes:

- `submission_logs_html.eex`
- `molecules.ex`
- `show.html.eex`

A continuación se puede observar el código modificado en el orden mencionado:

submission_logs_html.eex

```
<% else %>
<p class="text-info"><%= gettext("No more logs, go back.") %></p>
<% end %>
<%= if ! @back do %>
<span><%= link gettext("Back"),
    to: @back, class: "btn btn-sm btn-outline-primary" %></span>
<% end %>
```

molecules.ex

```
@spec submission_logs(list(), String.t()|nil) :: HTML.safe()
def submission_logs(logs, url \\ nil) do
  render("submission_logs.html", %{logs: logs, back: url})
```

show.html.eex

```
<%= Molecules.submission_logs(@logs,
    Routes.project_path(@conn, :show, @project)) %>
```

feature/partial_translation_warning

Esta tarea consiste en corregir los mensajes emergentes que aparecen en la aplicación de los administradores en dos ocasiones sin traducir al idioma del sistema:

1. Se va a eliminar una práctica.
2. Se va a dar de baja un alumno de una asignatura.

Desde el momento del descubrimiento de este *bug* se identificó como un error del módulo de Elixir *Gettext* [19]. Por lo que la investigación empezó directamente por ahí, sin embargo, al llegar al archivo en el que estaba el error, se vió que no era fallo del módulo, sino simplemente de código, ya que el mensaje estaba *Hardcoded*[1]

El archivo modificado es `index.html.eex` cuya ruta es `deliverit/apps/admin/lib/admin_web/templates/project`.

A continuación se puede observar el código modificado:

```
<tr href="<%= Routes.project_path(@conn, :show, project) %>">
        <td><%= nilable project.name %></td>
        <td><%= nilable project.subject.name %></td>
        <td><%= nilable project.mark %></td>
        <td><%= Atoms.pill_bool(project.active) %></td>
        <td><%= nilable project.from %></td>
        <td><%= nilable project.to %></td>
        <td>
          <%= link "Edit", to: Routes.project_path(@conn, :edit, project),
                class: "btn btn-sm btn-outline-primary" %>
          <%= link "Duplicate", to: Routes.project_path(@conn, :clone, project),
                class: "btn btn-sm btn-outline-secondary" %>
          <%= link "Delete", to: Routes.project_path(@conn, :delete, project),
            method: :delete, data: [confirm: gettext("Are you sure?")],
                class: "btn btn-sm btn-outline-danger" %>
        </td>
      </tr>
```

[1] En el ámbito informático este término se utiliza para código implementado sin ningún tipo de declaración de variable.

feature/updating_csv_data

Esta tarea consiste en mejorar la información ofrecida por el archivo `.csv`. A partir de las cabeceras del `.csv` se procede a analizarlas de manera funcional para así devolver a la vista como cabeceras de nuestra tabla los datos de las cabeceras que componen el `.csv`.

Gracias al análisis de los estados de la entrega de un proyecto inyectamos en la información mostrada el parámetro que corresponde, sea una entrega no realizada o una entrega fallida.

feature/trimar_inputs

Esta tarea consiste en conseguir que el sistema ignore los posibles espacios en blanco en la casilla *input* del nombre de usuario en la aplicación de los alumnos.

Esto se conseguirá mediante el tratamiento de *Strings* haciendo «escapar» el caracter para así evitar el fallo.

El archivo modificado para solucionar este comportamiento es `index.html.eex` cuya ruta es `deliverit/apps/student/lib/student_web/templates/auth`.

Al ser demasiado código se ha decidido no incluirlo en la memoria, se mostrará en la demostración del sistema.

feature/logs_steps

Esta tarea consiste en aumentar la información ofrecida en la pantalla de los logs tras la entrega de una práctica por parte de un alumno.

Se quería diferenciar cada prueba a la que el código es ejecutada marcando el texto en negrita, además se quería representar de una forma más visual qué pruebas había superado y cuales no, por lo que se optó por asignar el símbolo del *tic* a las pruebas exitosas y el símbolo de la *X* a las pruebas erróneas. Esto se ha conseguido mediante una comparación muy sencilla, en la que se comprueba si el estado es *Success* se imprimirá el símbolo correspondiente y en el caso del estado *Failed* igual.

feature/admin_project_export

Esta tarea consiste en que los profesores puedan recuperar el archivo que suben como código base para las comprobaciones de las prácticas de los alumnos. Al inicio de este TFG el profesor solo podía subir el código, pero no recuperarlo.

Para solucionar este problema se debía implementar un botón de descarga en la página de administración de las prácticas como el que tienen los alumnos en su aplicación.

Una vez implementado ese botón en la pantalla correspondiente se crea el evento que permite descargar el código al pulsar dicho botón.

Capítulo 6

Análisis de impacto

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del TFG de cara a los objetivos marcados por la **Agenda de 2030** según la **ODS**.

Para hablar de la **Agenda de 2030**, primero debemos saber qué es, por lo que haré un breve resumen.

La ONU (Organización de las Naciones Unidas) adoptó en Septiembre de 2015 un conjunto de objetivos para erradicar la pobreza, luchar contra el cambio climático y así proteger el planeta entre otras cosas.

Para que esto se pueda conseguir será necesario que ciudadanos, gobiernos y empresas cumplan con su parte de aquí a 15 años, por eso el nombre Agenda 2030. Los objetivos se han organizado en una lista de 17, que a su vez se dividen en diferentes tareas.

De cara a la Agenda 2030, el presente TFG haría especial hincapié en el punto **nº4** y el **nº8**, que hacen referencia a la educación de calidad, al trabajo decente y al crecimiento económico. Concretamente se desarrolla una herramienta para que todos aquellos que lo necesiten puedan tener acceso a una educación libre y gratuita, desde cualquier parte del mundo; esto deriva en un desarrollo del conocimiento que hará que, más adelante, se integren en el mercado laboral.

Se apuesta por una educación más completa, que es la base para el desarrollo de cualquiera de los puntos que se establecen en el tratado. Por ejemplo, integrar el cuidado del clima, la vida submarina o los ecosistemas terrestres es fácil si previamente se ha apostado por unas comunidades sostenibles y una industria innovadora, lo cual solo se consigue manejando mucha información, teniendo acceso a la documentación y favoreciendo la transmisión de valores y conocimientos gracias al sistema desarrollado en este TFG.

Capítulo 7

Conclusiones y trabajo futuro

En este último capítulo de la memoria se podrán observar unas conclusiones personales acerca del trabajo realizado y los resultados obtenidos.

7.1. Conclusiones

Para iniciar estas conclusiones, pondré un ejemplo personal de por qué pienso que es necesario tener un sistema de entregas como es DeliverIt y por qué habría que seguir mejorándolo.

He pasado 6 años en esta escuela y son innumerables las prácticas que hay que entregar. Cada una de una asignatura y por supuesto, cada una con su método de entrega. Esto muchas veces genera una sensación de rechazo hacía la práctica puesto que tienes que dedicarle un tiempo y esfuerzo extra únicamente para poder entregarla.

Siendo conocedores de este problema, me parece imprescindible el trabajo que se está llevando a cabo con este proyecto, utilizando tecnologías modernas e innovadoras que aportan un muy buen rendimiento y dejan espacio a la escalabilidad. Este último aspecto es un pilar fundamental puesto que, el objetivo principal de DeliverIt es no quedarse atrapado solo en la asignatura de Concurrencia, si no expandirse a más asignaturas e incluso otras universidades. Gracias a la metodología de trabajo que Ángel ha elegido (metodología *Agile*) la adaptación al ritmo de trabajo cuando te incorporas al proyecto es intuitivo y además muy útil de cara al mundo laboral puesto que esto se utiliza cada vez más.

Desde el primer momento, mi trabajo en el proyecto consistía en la identificación de errores o malos funcionamientos en el sistema y la implementación de nuevas funcionalidades que habían sido identificadas en trabajos de compañeros anteriores, todas estas tareas se corresponden con el *Front-End* del sistema.

Para la realización de estas tareas ha sido necesario el estudio y aprendizaje de tecnologías como HTML, Elixir y Phoenix, las cuales hasta el momento de iniciar

el trabajo desconocía, sin embargo ha sido un aprendizaje bastante satisfactorio por la utilidad que tienen de cara a mi futuro en el mundo laboral.

Para finalizar las conclusiones me parece oportuno comentar que actualmente DeliverIt es un prototipo totalmente usable. Está siendo probado en varias asignaturas con varias prácticas cada una y con una cantidad importante de alumnos, por lo que la tarea principal para la que se desarrolló ya está implementada y por esto tanto mi trabajo, como los trabajos futuros irán siendo cada vez más específicos hasta el momento que solo quede el mantenimiento del sistema.

7.2. Resultados

Antes de entrar en los resultados de este TFG a nivel técnico, me gustaría hablar de los resultados desde los siguientes puntos de vista:

7.2.1. Personal

A nivel personal este trabajo ha tenido un impacto bastante positivo, puesto que el tiempo dedicado al aprendizaje de las tecnologías necesarias para poder contribuir en el trabajo del sistema es algo que muy posiblemente nunca hubiera llevado acabo en otra situación.

Son tecnologías innovadoras que no se enseñan en la universidad por su especificidad y que raramente a día de hoy vas a encontrar en el mundo laboral. Haber trabajado con ellas y haberlas aprendido me proporciona un punto diferenciador que en el contexto de una entrevista podrían decantar la balanza a mi favor.

7.2.2. Empresarial

A nivel empresarial, si tomamos la Universidad Politécnica de Madrid (UPM) como una empresa el impacto también es positivo, ya que con este sistema de entregas mejoraremos el funcionamiento general de las asignaturas, tanto para profesores como para alumnos, ahorrando tiempo que profesores pueden dedicar a investigación u otras labores que lleven acabo en su jornada y los alumnos podrán dedicar el tiempo ahorrado a la realización de las mismas prácticas o al estudio.

7.2.3. Social

A nivel social el proyecto tendrá un impacto positivo de cara al prestigio de la universidad, puesto que implementar un sistema de entrega como este, basado en unas tecnologías tan innovadoras puede ser un ejemplo a seguir para otras universidades.

7.2.4. Económico

A nivel económico este trabajo no tendría impacto por si solo pero, tomando el proyecto DeliverIt como referencia sí que tendría impacto económico puesto

que la implantación de un único sistema de gestión de entregas en la universidad ahorraría servidores dedicados para cada uno de los sistemas que utilizan las asignaturas y este ahorro de servidores se traduce directamente en ahorro monetario que se puede dedicar a mejora de infraestructuras o cualquier otra necesidad.

7.2.5. Medioambiental

A nivel medioambiental y haciendo referencia al apartado anterior, con el ahorro de servidores contratados por parte de la UPM se disminuiría el gasto energético y de materiales consumibles.

7.2.6. Cultural

A nivel cultural tendrá impacto a la hora de posibles favoritismos o discriminaciones en cuanto a corregir las prácticas de los alumnos, por temas raciales, de género o por problemas pasados entre alumno y profesor, puesto que al ser un corrector automático entre el código del alumno y el código base subido por el profesor no habrá necesidad de interacción entre ambos autores.

Por otro lado, los resultados técnicos del trabajo podrían resumirse en las tareas más destacables que he llevado a cabo en estos meses:

- Implementar la funcionalidad para que como profesor pueda **descargar con facilidad el código base** que he subido a la plataforma para la corrección de prácticas de alumnos.
- **Mejorar la información ofrecida** en la pantalla de logs tras entregar una práctica, para que de este modo sea fácilmente identificable que pruebas se han superado y cuales no.
- **Trimar** [1] los espacios en blanco en el input de nombre de usuario en la pantalla *login*, puesto que a fecha de inicio de este trabajo si encontraba algún blanco al inicio o final del nombre no permitía entrar al sistema.
- Permitir que como profesor se pueda **añadir a un alumno a varias asignaturas** sin que retroceda automáticamente a la pantalla anterior tras añadir una.
- **Mejorar la información ofrecida por el archivo .csv** que se utiliza para la carga masiva de alumnos.
- Implementación de un ***Sticky footer*** en la aplicación de los alumnos, de modo que este siempre sea visible y se mantenga en la misma posición.
- **Correción de traducciones parciales** del sistema en mensajes de aviso de la aplicación de administradores.

Además de estas implementaciones mencionadas, se han añadido y corregido otras funcionalidades menores que no tienen el suficiente peso para ser mencionadas.

[1] Trimar: Hace referencia a la acción de ignorar los espacios en un parámetro de entrada.

7.3. Trabajo futuro

Pese al esfuerzo y tiempo dedicado a este trabajo, aún quedan tareas que requieren atención y esfuerzo, por lo que los siguientes alumnos que contribuyan con este proyecto deberán abordarlas. Algunas de las más importantes son:

- **Mejorar la información** que ofrece *Docker* cuando algo no funciona correctamente.
- Permitir que como profesor, pueda **copiar facilmente** nombres, números de matrícula o correos desde las tablas.
- **Mejorar** la *UI* para informar sobre la entrega de una práctica.
- **Mejorar** la tolerancia a fallos y el número de entregas simultáneas que puede soportar el sistema.

Bibliografía

[1] A. Mareca, "Sistema de entrega deliverit: Prototipo," Trabajo Fin de Grado, Universidad Politécnica de Madrid, 2021.

[2] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison Wesley, 2003.

[3] C. Arturo, "Explicando que es front-end, que es back-end y sus caracteristicas." [Online]. Available: http://www.falconmasters.com/web-design/que-es-front-end-y-que-es-back-end/

[4] "Docker registry." [Online]. Available: https://docs.docker.com/registry/

[5] "Dockerfile reference." [Online]. Available: https://docs.docker.com/engine/reference/builder/

[6] B. T. W. Hao, *The Little Elixir and Guidebook*, 1st ed. USA: Manning Publications Co., 2016.

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

[8] J. Duckett, *HTML and CSS: Design and Build Websites*, 1st ed. Wiley Publishing, 2014.

[9] B. T. Chris McCord and J. Valim, *Programming Phoenix 1.4*, 1st ed. O'Reilly UK Ltd., 2019.

[10] "Ecto." [Online]. Available: https://github.com/elixir-ecto/ecto

[11] B. Momjian, *PostgreSQL: Introduction and Concepts*. USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[12] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization*. James Turnbull, 2014.

[13] J. Shore and S. Warden, *The Art of Agile Development*, 1st ed. O'Reilly, 2007.

[14] M. Burrows and L. Hohmann, *Kanban from the Inside*. Blue Hole Press, 2014.

[15] Trello, "¿qué es trello?" [Online]. Available: https://trello.com/es/about

[16] GitLab, “What is gitlab?” [Online]. Available: https://about.gitlab.com/what-is-gitlab/

[17] Altassian, “Gitflow workflow.” [Online]. Available: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[18] GitKraken, “About gitkraken.” [Online]. Available: https://www.gitkraken.com/about

[19] U. Drepper, *GNU gettext tools: Native Language Support Library and Tools*. Samurai Media Limited, 2015.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Thu Jan 20 20:55:00 CET 2022
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)