Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Monitorización de entregas en Deliverit

Autor: Elías Herrero Lázaro
Tutor(a): Lars-Åke Fredlund

Madrid, Junio 2023

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: Monitorización de entregas en Deliverit

Junio 2023

Autor: Elías Herrero Lázaro
Tutor: Lars-Åke Fredlund
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Desde el curso 2020-2021, en la *Escuela Técnica Superior de Ingenieros Informáticos* (ETSIINF) de la *Universidad Politécnica de Madrid* (UPM) se emplea un sistema de corrección, monitorización y entrega, llamado *Deliverit*, y permanece a día de hoy en desarrollo.

Este sistema permite que los alumnos envíen código y este sea corregido y calificado, por lo que la herramienta debe manejar los envíos, estableciendo una cola de entregas y realizando las correcciones en un entorno aislado. Actualmente, existen problemas como la falta de recuperación de dicha cola ante una interrupción no deseada del sistema o la acumulación y retardo de entregas pendientes en horas críticas (normalmente, previas a la fecha de cierre de alguna entrega). Estos problemas afectan tanto a estudiantes como a administradores.

Por lo tanto, el objetivo de este trabajo es, bajo la dirección de Lars-Åke Fredlund y Ángel Herranz, mejorar la forma en la que *Deliverit* procesa las entregas recibidas, aumentar la calidad de vida de la administración mediante nuevas funcionalidades y proporcionar al estudiante mayor información sobre su entrega en curso.

Mi papel en este proyecto ha sido el de documentar la situación actual del sistema con respecto al proceso de cola, analizar los puntos a mejorar e implementar dichas mejoras. En esta memoria constan todos los detalles sobre el análisis y las decisiones tomadas.

Abstract

Since the 2020-2021 academic year, the School of Computer Science and Engineering (ETSIINF) at the Polytechnic University of Madrid (UPM) has been using a correction, monitoring, and delivery system called *Deliverit*, which is still under development.

This system allows students to submit code, which is then corrected and graded. Therefore, the tool must manage the submissions by establishing a delivery queue and performing the corrections in an isolated environment. Currently, there are issues such as the lack of recovery of the queue in the event of an unexpected system interruption or the accumulation and delay of pending deliveries during critical hours (usually preceding the deadline for a submission). These problems affect both students and administrators.

Therefore, the objective of this work is to improve the way *Deliverit* processes received submissions, enhance the administrators' quality of life through new functionalities, and provide students with more information about their ongoing deliveries under the guidance of Lars-Åke Fredlund and Ángel Herranz.

My role in this project has been to document the current situation of the system regarding the queue process, analyze areas for improvement, and implement those enhancements. This report contains all the details about the analysis and the decisions made.

Tabla de contenidos

1. Motivación **1**
1.1. Objetivos del presente trabajo 2
1.2. Estructura de la memoria 2

2. Situación actual **3**
2.1. Descripción general 3
2.2. Almacenamiento de información 4
2.3. Interfaz web 6
2.4. Administración 7

3. Requisitos **9**
3.1. Requisitos 9
3.1.1. Requisitos de los estudiantes 9
3.1.2. Requisitos de los administradores 9
3.1.3. Requisitos del sistema 10
3.2. Metodología 10

4. Análisis y Diseño **13**
4.0.1. REQS-1: Mejora de la información recopilada 13
4.0.2. REQA-3: Límite de entregas concurrentes por usuario 15
4.0.3. REQA-1: Visión general de la cola 16
4.0.4. REQA-2: Recuperación de la cola 17
4.0.5. REQE-1: Estimación de tiempo de entrega 18

5. Implementación **21**
5.0.1. REQS-1: Mejora de la información recopilada 21
5.0.2. REQA-3: Límite de entregas concurrentes por usuario 23
5.0.3. REQA-1: Visión general de la cola 25
5.0.4. REQA-2: Recuperación de la cola 26
5.0.5. REQE-1: Estimación de tiempo de entrega 28

6. Análisis de impacto **31**

7. Conclusiones y trabajo futuro **33**
7.1. Conclusiones 33
7.2. Trabajo futuro 33
7.2.1. Nuevo sistema de cola 34

7.2.2. Sistema de prioridad 34
7.2.3. Detención de entrega única 34
7.2.4. Manejo de errores si hay fallo en el proceso corrector 34

Bibliografía **37**

Capítulo 1

Motivación

A día de hoy, desde el curso 2020-2021, se emplea en la Escuela Técnica Superior de Ingenieros Informáticos (ETSIINF) de la Universidad Politécnica de Madrid (UPM) un sistema de corrección, monitorización y entrega llamado *Deliverit*. Esta herramienta nace del propósito de facilitar la entrega y corrección de prácticas tanto para alumnos como profesores, y permanece a día de hoy en desarrollo. Durante los últimos años, su empleo en diferentes asignaturas ha dado a conocer necesidades sobre el sistema sin satisfacer.

En el sistema, los profesores crean prácticas con pruebas de código que comprueban los envíos de los alumnos. Según las pruebas pasadas satisfactoriamente, se establece la calificación acorde. Si se reciben más envíos de los que el sistema puede corregir en un instante de tiempo, las entregas son almacenadas en una cola, esperando su respectivo turno para ser corregidas. Durante este proceso se recopila información sobre cada entrega (práctica a la que pertenece, identificador del grupo que realiza el envío, instante de llegada al sistema, etc.), pero se han detectado incoherencias en el sistema debido a su falta de completitud. Por ejemplo, una entrega cuya corrección ha fallado sigue manteniendo un estado que indica que permanece en corrección. Esto motiva a analizar y mejorar esta recopilación de información durante el procesado de las entregas.

De esta falta de completitud en la información sobre una entrega surgen otros problemas como la falta de consistencia en caso de interrupción abrupta. A la hora de recuperar el sistema esto produce inconsistencias, lo que afecta negativamente a la robustez del sistema.

Además de los problemas mencionados, surge la necesidad por parte de los administradores de poder visualizar todas las entregas en cola para resolver dudas como, por ejemplo, cual es el motivo por el que una correccción es inusualmente larga, cuantas entregas quedan por corregir, etc. Para resolver este tipo de dudas es necesaria una vista donde se muestre la cola en su totalidad.

Estas necesidades presentadas y otras añadidas como una prioridad a la hora de corregir entregas de diferentes prácticas, una recuperación automática en caso de caída del sistema, la presentación de un tiempo de espera estimado al alumno, etc. constituyen una motivación para mejorar la herramienta *Deliverit*,

continuando con el enfoque original de construir un sistema de entrega unificado para asignaturas, departamentos e incluso escuelas.

1.1. Objetivos del presente trabajo

Por lo tanto, los objetivos de este trabajo son:

- Mejorar la recopilación de información sobre cada entrega:
  Una reforma de la base de datos, que suponga un aumento de análisis del estado de la entrega durante el proceso de corrección y tratamiento de errores.
- Implementar un sistema de recuperación automático:
  En caso de interrupción del sistema, una herramienta capaz de volver a encolar todas las entregas pendientes.
- Desarrollar una herramienta para visualizar todas las entregas en cola:
  Una nueva funcionalidad que aúne todas las entregas en curso para su administración y monitorización.
- Añadir un sistema de prioridad:
  Funcionalidad cuya pretender poder añadir a la cola delante de ciertas entregas otras con mayor prioridad. La intención de este objetivo es el uso de *Deliverit* en pruebas presenciales con tiempo limitado.
- Desarrollar un sistema de predicción de tiempo de entrega:
  Implementar una funcionalidad que permita calcular el tiempo estimado de corrección para cada entrega, visible por alumnos y administradores. Debe tener una precisión aceptable.
- Documentar las mejoras:
  Registro de las mejoras realizadas que pueda servir de guía para posteriores trabajos en la herramienta.

1.2. Estructura de la memoria

Teniendo este enfoque en mente, en los siguiente capítulos se procederá a ahondar en las diferentes cuestiones. Primero se realizará un análisis de la situación actual del sistema, que clarifica y sirve de guía para continuar con el desarrollo del trabajo, donde se especificarán los requisitos, se presentará el entorno de desarrollo y se procederá a definir las mejoras y sus implementaciones. Por último, se presentarán las conclusiones y el trabajo a realizar en el futuro.

Capítulo 2

Situación actual

En este capítulo se detallará el estado actual del sistema, incidiendo en los aspectos más relevantes sobre la gestión de entregas.

2.1. Descripción general

Deliverit es un sistema creado por alumnos (con ayuda de tutores) en Trabajos de Fin de Grado previos (principalmente en *Sistema de Entrega Deliverit* [1] y *Evoluciones y refactorizaciones de un sistema de entrega de prácticas* [2]) escrito en *Elixir*. Este sistema permite la entrega de código por parte de los alumnos para su corrección, y la monitorización y gestión de entregas por parte de los administradores. Para ambas funcionalidades, existen dos webs que permiten la interacción con el sistema, una aplicación que mantiene toda la lógica y una última que atiende las consultas a base de datos.

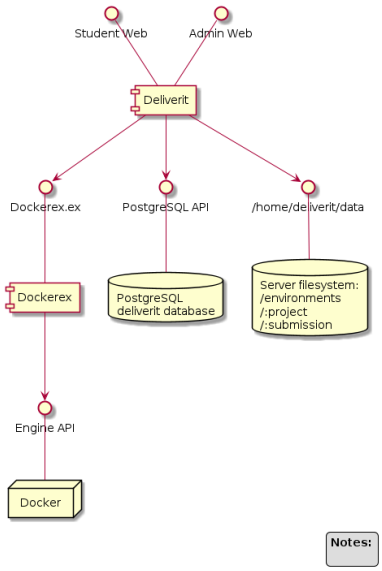

Figura 2.1: Estructura de *Deliverit*

Sobre el proceso de corrección, actualmente el estado de una entrega se actualiza y almacena mientras avanza por las fases de la corrección. La asignación de procesos correctores es delegada a la biblioteca de *Erlang* (al ser *Elixir* heredado de *Erlang* y correr en la *BEAM* [3], sus bibliotecas son compatibles) *Poolboy* [4]. Esta biblioteca ofrece el manejo de un *pool* de procesos, siendo el tamaño de este definido por el programador. Con una configuración correcta, permite emplear eficientemente los recursos de la máquina sin agotarlos.

En el caso de que todos los procesos trabajadores estén ocupados, *Poolboy* posee una cola de tipo *FIFO (First In, First Out)* en la que almacena las peticiones de corrección de entregas sobre el *pool*. Esta es la cola que *Deliverit* emplea para manejar el procesado de las entregas.

La ejecución de las pruebas de código se realiza en contenedores gracias a la tecnología de virtualización de Docker [5], que permite su aislamiento. Para la interacción del sistema con esta biblioteca, Andrés Mareca desarrolló en *Sistema de Entrega Deliverit* la biblioteca *Dockerex* [6]. Esta permite que los procesos trabajadores manejen los contenedores y ejecuten los pasos de las correcciones en ellos.

Presentadas estas características, el proceso de corrección es el siguiente: un grupo (nótese que un grupo puede estar formado únicamente por un alumno) envía una entrega. Una vez recibida la entrega, se le asigna un estado inicial y el controlador de entregas solicita al *pool* de procesos la corrección de la entrega. Este estado asignado se mantiene hasta que el proceso corrector toma el control de la entrega y procede con su corrección. En ese momento, se actualiza el estado para indicar que la entrega está en corrección. Una vez finalizada la corrección, la entrega alcanza un estado final (que detalla si la corrección ha sido satisfactoria o no).

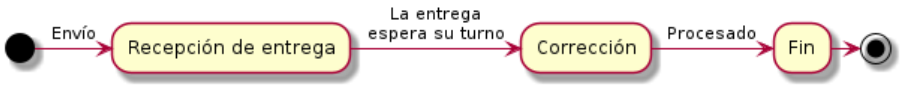

Figura 2.2: Fases del proceso de corrección

2.2. Almacenamiento de información

El almacenamiento de información sobre las entregas se realiza en una base de datos y es actualizada según va avanzando por las fases del procesado.

Para salvar el estado del sistema, es necesario el empleo de una base de datos. Actualmente *Deliverit* emplea la librería *Ecto* [7] para esta tarea junto con la base de datos *PostgreSQL* [8]. *Ecto* es una librería que permite la abstracción del manejo de la base de datos. Esta librería está diseñada para su empleo junto a diferentes bases de datos, como por ejemplo *PostgreSQL*, de ahí la decisión de emplear por parte de trabajos previos.

Una entrega en base de datos está formada por un identificador, varios instantes de tiempo y un estado.

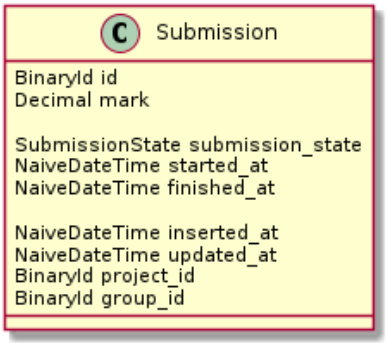

Figura 2.3: Esquema de una entrega

Los instantes de tiempo son varios:

- **`inserted_at`**: Momento en el que la entrega fue registrada en la base de datos.
- **`started_at`**: Instante de tiempo en el que se inició la corrección.
- **`finished_at`**: Instante de tiempo en el que finalizó la corrección.
- **`updated_at`**: Momento en el que se realizó la última modificación sobre la entrega.

El estado consta de 4 opciones:

- **Pendiente** (`:pending`): Aúna cualquier situación previa la corrección de la entrega. Esto aporta información incompleta que afecta negativamente al sistema ya que existen dos estados reales previos a la corrección: el procesado inicial y la propia espera en cola de la entrega.
- **En ejecución** (`:running`): La entrega se está compilando y probando en un entorno controlado.
- **Aprobada** (`:processed`): La entrega ha compilado correctamente y ha pasado todos los test (si los hubiera).
- **Suspendida** (`:failed`): La entrega no ha compilado o no ha pasado los test.

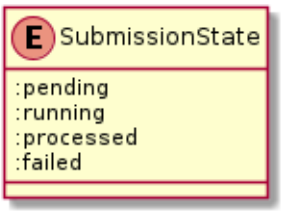

Figura 2.4: Tipos de estados

Teniendo en cuenta estos campos de información y la descripción general del proceso, la recopilación de información se realiza en los siguientes puntos:

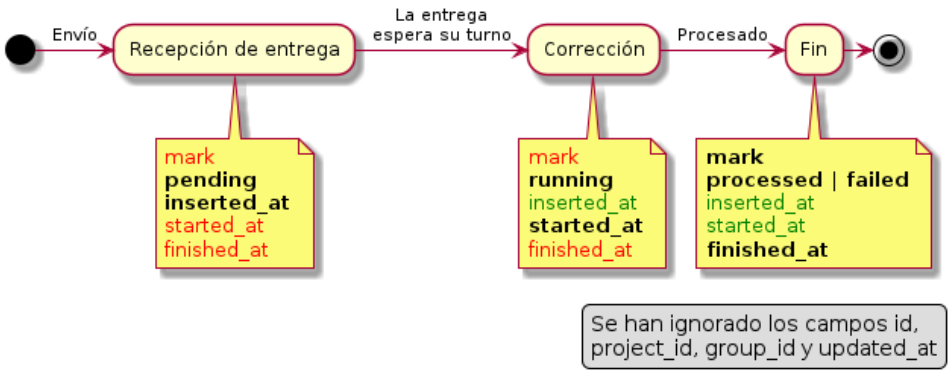

Figura 2.5: Recopilación de información durante el procesado de una entrega

El sistema recibe la entrega, le asigna el estado de `:pending` y solicita un proceso del *pool* para realizar la corrección. Cuando este proceso es asignado, se actualiza a `:running` y una vez finalice, se modifica a `:processed` si ha sido corregida satisfactoriamente, o a `:failed` en caso contrario.

2.3. Interfaz web

Para el manejo del sistema por parte de alumnos y administradores, *Deliverit* posee webs desarrolladas mediante el *web framework Phoenix* [9].

Phoenix es un *web framework* que sigue el patrón MVC (*Model-View-Controller*). Un módulo controlador maneja sus correspondientes módulos *modelo* (que realiza la carga lógica: cálculos, comprobaciones, etc.) y *vista* (que se encarga de mostrar los resultados). Una aplicación con este *framework* suele estar dividida en modelo, vista y controlador por funcionalidad. Para acceder a diferentes funcionalidades, la aplicación consta de un módulo enrutador que, dependiendo del recurso solicitado (crear un nuevo usuario, obtener una entrega, etc.) llama a diferentes controladores.

Actualmente existen dos webs, una para estudiantes y otra para administradores (debido a las diferentes funcionalidades que ambos roles deben tener). Estas permiten la navegación por diferentes asignaturas, prácticas y entregas.

Las plantillas siguen el patrón *Atomic Design* [10], construyéndose modularmente. Generalmente están formadas por el logo del sistema, el icono de usuario, el panel de navegación, diferentes componentes donde se muestra la información relevante (tablas con información sobre entregas, campos con datos sobre prácticas, etc.) y un pie de página.

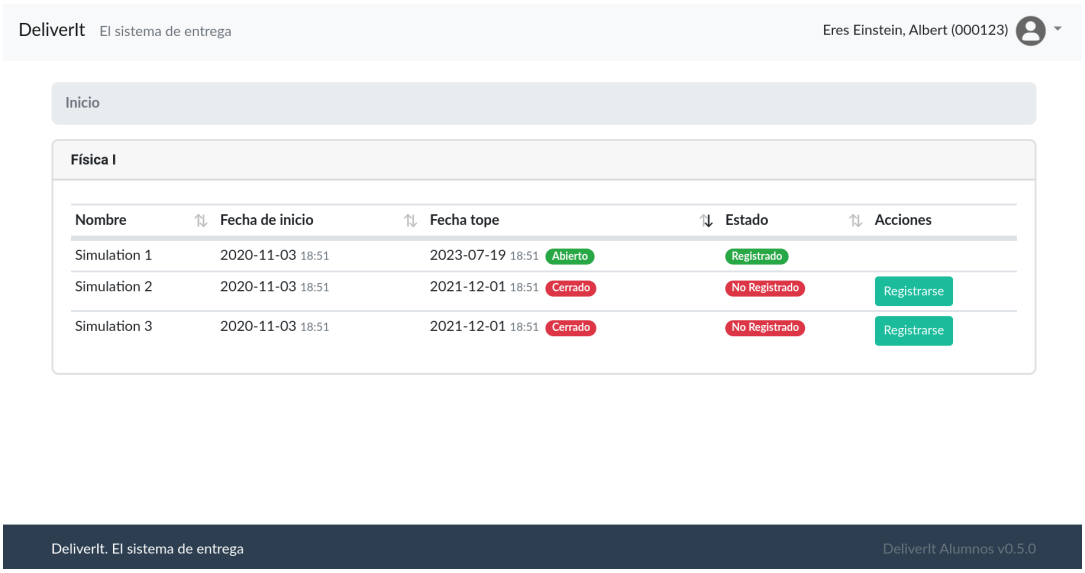

Figura 2.6: Asignaturas y prácticas en las que el usuario está matriculado

En la figura 2.6 se pueden observar los diferentes componentes enunciados anteriormente. En la parte superior se puede ver el logo de la aplicación *Deliverit: El sistema de entrega* y el icono de usuario. Le sigue el panel de navegación (ya que el usuario está en el inicio, no puede realizar ninguna acción con el panel). A continuación hay una tabla con las prácticas sobre una asignatura. En cada entrada de la tabla se puede ver información sobre cada una (nombre, fechas relevantes, estado de la práctica y acciones a realizar por el usuario). Por último, en la parte inferior de la imagen, se puede ver el pie de página. La versión del sistema está indicada en la parte derecha.

2.4. Administración

Ante las numerosas posibilidades del sistema se pueden dar casos no deseados. Por ello, el personal docente (tutores de este trabajo) realiza tareas de monitorización y administración. Ejemplos de estas tareas son volver a corregir una práctica, obtener su código, reestablecer entregas en estados *difusos* (se detallará más adelante). Todo ello se realiza a través de la web de administración.

Capítulo 3

Requisitos

En este capítulo se presentarán los requisitos a desarrollar y la metodología de trabajo.

3.1. Requisitos

Los requisitos del sistema [12] contemplados en este trabajo se dividen en 3 grupos:

1. Del estudiante (***REQE***): requisitos solicitados por los estudiantes.
2. Del administrador (***REQA***): requisitos solicitados por los administradores.
3. Del sistema (***REQS***): requisitos necesarios para el posterior desarrollo de la herramienta.

3.1.1. Requisitos de los estudiantes

REQE-1	Estimación de tiempo de entrega	**Importancia**	Baja
Descripción			
Añadir un campo de información sobre la entrega en corrección que muestre una estimación sobre el tiempo restante para obtener la nota de dicha entrega. Visible tanto para alumnos como para administradores.			

3.1.2. Requisitos de los administradores

REQA-1	Visión general de la cola	**Importancia**	Media
Descripción			
Nueva funcionalidad que muestra una vista a los administradores con todas las entregas en corrección o en cola, dando la capacidad de interactuar con ellas.			

REQA-2	Recuperación de la cola	**Importancia**	Media
Descripción			
Herramienta que habilita al administrador la repetición de una corrección sobre todas las entregas registradas sin nota final. Tiene que estar disponible en una vista desde la que se puedan ver todas las entregas en cola.			

REQA-3	Límite de entregas concurrentes por grupo	**Importancia**	Media
Descripción			
Modificación sobre la política de entrega por parte de un grupo. Sólo se podrá tener una entrega en corrección por grupo y por práctica.			

REQA-4	Sistema de prioridad	**Importancia**	Baja
Descripción			
Funcionalidad que permita a un administrador modificar una práctica para que las entregas realizadas por los alumnos tengan preferencia sobre las demás a la hora de corregirse.			

3.1.3. Requisitos del sistema

REQS-1	Mejora la información recopilada sobre las entregas	**Importancia**	Alta
Descripción			
Ampliación de la información almacenada sobre las entregas (añadir un instante en que la entrega es puesta en la cola de corrección), manejo de casos y actualización de la base de datos en el caso de que el proceso corrector muera abruptamente.			

3.2. Metodología

Este trabajo se realiza en *Elixir* [13], un lenguaje de programación heredado de *Erlang*, que aprovecha características clave como la escalabilidad, mantenibilidad o concurrencia, cualidades muy adecuadas para el sistema en cuestión.

Para el flujo de trabajo se emplea *Git*, el popular sistema de control de versiones de código abierto. Concretamente, para el desarrollo simultáneo ordenado, se emplea *Git Flow* [14], que es una estrategia de desarrollo para *Git* que simplifica y acelera la implementación y publicación de funcionalidades y/o versiones finales. Esta metodología propone realizar el desarrollo en una rama *develop*, desde la cual surgen otras ramas *feature* donde se implementan funcionalidades (por ejemplo, *feature/estimations*). Estas ramas, una vez terminadas, se acoplan a la rama de desarrollo y, al comprobarse su correcta implementación, se unen a *master*. Los cambios en la rama *master* constituyen versiones del sistema. Un ejemplo de esta metodología ha sido la implementación de la mejora de información recopilada (*feature/enqueued_at*) y la vista de entregas en cola (*feature/queue_view*). Una vez se completaron las ramas, los tutores las acoplaron a *develop* y unieron los cambios a la rama *master* después de comprobarlas. Esto ha constituído la versión *0.5* de *Deliverit*.

La organización y coordinación del desarrollo de la herramienta se realiza me-

diante la herramienta *Trello* [15]. Permite gestionar, supervisar y documentar las tareas pasadas, presentes y futuras.

Capítulo 4

Análisis y Diseño

A continuación, se detalla en orden cronológico el análisis realizado y las decisiones de diseño para la implementación de los requisitos.

4.0.1. REQS-1: Mejora de la información recopilada

Según lo descrito en la sección 2.2, la información recopilada sobre las entregas no ofrece el detalle necesario para afrontar todas las situaciones que se pueden dar en el sistema. Por lo tanto, para paliar esta falta de detalle, se propone modificar el esquema que representa una entrega en la base de datos y añadir instantes de actualización del estado de la misma.

Se denota una falta de detalle desde la recepción de una entrega hasta su turno de corrección. Esta situación sólo se representa con el estado `:pending`, y se trabaja con los instantes de tiempo en los que la práctica es recibida por el sistema (`:inserted_at`) y el instante en el que comienza la corrección (`:started_at`).

Por lo tanto, en cuanto al esquema, se ha tomado la decisión de añadir un estado adicional a esta situación `:enqueued`, relegando el estado `:pending` al procesado inicial de la entrega (creación de su registro en base de datos, solicitud al *pool* de procesos para iniciar la corrección, etc.). A su vez, se ha decidido añadir el campo `:enqueued_at`, que indica el momento en el que la entrega ha sido introducida en la cola. Este campo ayudará a realizar el cálculo de cuánto tiempo ha estado una entrega en cola de forma más precisa, pasando a trabajarse con los instantes `:enqueued_at` y `:started_at`.

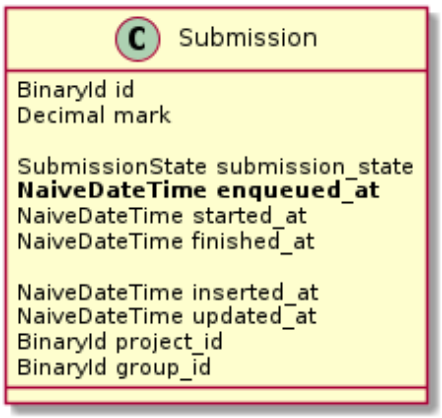

Figura 4.1: Nuevo esquema

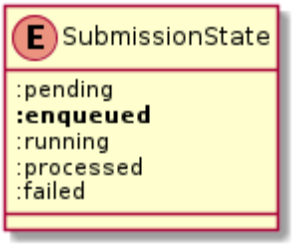

Figura 4.2: Nuevo estado *:enqueued*

En cuanto a las actualizaciones de este esquema, se añade una fase en el momento que se encola la entrega, actualizándose el estado a **enqueued** e inicializándose el instante de tiempo `:enqueued_at`.

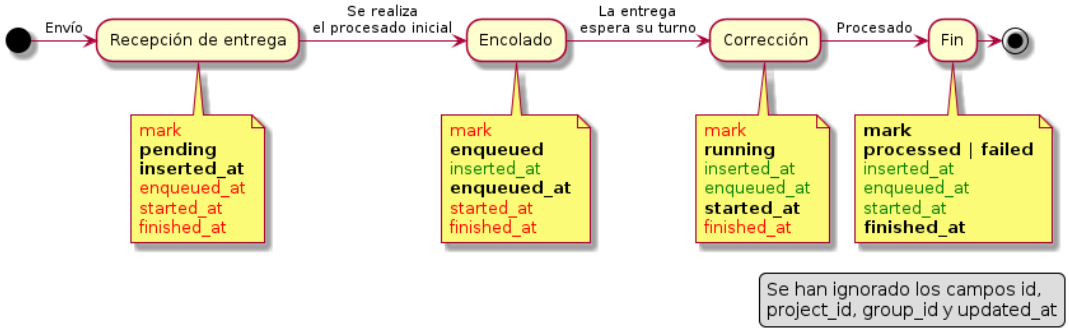

Figura 4.3: Nuevo esquema de recopilación de la información

Para mantener la coherencia con las entregas previas a estos cambios, se ha realizado una migración de la base de datos (delegada en parte a *Ecto*) en la que se define el instante de encolado como el instante de llegada al sistema (`:enqueued_at` = `:inserted_at`).

Para mostrar estos campos a estudiantes y administradores, se han modificado

las plantillas HTML (renderizadas por *Phoenix*) que muestran el estado de una entrega y proyecto, añadiendo el campo `:enqueued_at` y modificando el cálculo del tiempo en cola.

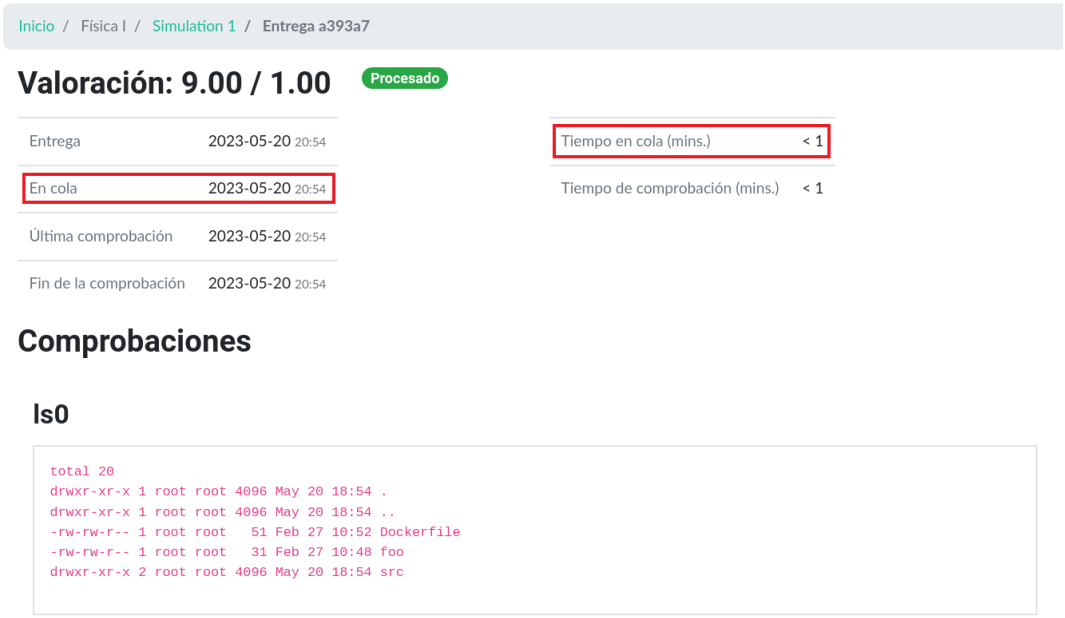

Figura 4.4: Campos modificados en la vista de una entrega

4.0.2. REQA-3: Límite de entregas concurrentes por usuario

Según el proceso de entrega descrito en la sección 2.2, el sistema realiza diferentes comprobaciones previas al procesado de dicha entrega en la fase de recepción. Las comprobaciones son las siguientes:

1. La existencia de la práctica sobre la que se está realizando la entrega.
2. La entrega se realiza dentro del periodo de admisión.
3. La entrega está realizada por un grupo matriculado en la práctica.
4. Los archivos recibidos son adecuados para la corrección.

Actualmente, esta secuencia de comprobaciones se ejecuta en su totalidad cada vez que el usuario realiza un envío, sin limitarse de ninguna forma. Por lo tanto, buscando principalmente aliviar la carga de entregas del sistema, se ha tomado la decisión de expandir esta secuencia de comprobaciones añadiendo un paso en el que se verifica que no hay entregas pendientes, encoladas o corrigiéndose por parte del grupo que envía la entrega. De esta forma, si se da el caso de que exista una entrega en uno de estos estados, el sistema devolverá al usuario una notificación sobre este límite, y la posibilidad de reintentar el envío una vez termine la entrega en curso.

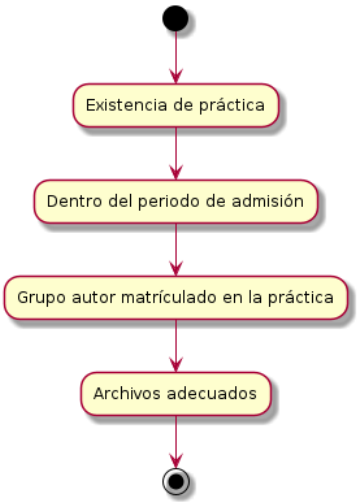

Figura 4.5: Secuencia de comprobaciones en la recepción de una entrega

Figura 4.6: Nueva secuencia de comprobaciones

Para saber si hay entregas en curso del grupo autor, se ha diseñado una consulta (*query*) a la base de datos que devuelve el número de entregas con dichas características.

4.0.3. REQA-1: Visión general de la cola

Sobre la necesidad de los administradores de ver todas las entregas en curso en vez de tener que ir práctica por práctica (que es la visión que tienen actualmente), se ha diseñado una vista general en la que se puedan ver todas las entregas en proceso.

Por la relevancia de esta funcionalidad para los administradores, se decide localizar el acceso a ella en la página de inicio de su web (siendo por lo tanto el recurso localizado en `/queue`).

Figura 4.7: Pantalla de inicio con la nueva funcionalidad

Esta vista general consiste en una plantilla HTML (descrita en la sección 2.3) en la que se muestra en tabla la información relevante sobre todas las entregas pendientes, en cola o en corrección. Para mostrar dicha información, se implementa una consulta a la base de datos sobre las entregas.

Se ha tomado la decisión de mostrar la asignatura y la práctica a la que pertenece cada entrega con el fin de que se puedan visualizar rápidamente errores en prácticas concretas.

Figura 4.8: Panel de la vista

Existe la posibilidad de repetir la corrección de las entregas en la tabla de datos mediante un botón ya implementado en trabajos anteriores.

4.0.4. REQA-2: Recuperación de la cola

Para poder activar o detener la corrección de entregas en masa se han diseñado dos botones, uno para cada respectiva funcionalidad.

Para la activación de corrección en masa, el sistema realiza una consulta a la base de datos sobre todas las entregas pendientes (`:pending`). Esta consulta provee una lista de entregas sobre la que se corrige cada elemento de la lista.

Para la detención de entregas en masa, primero se realiza una consulta a la base de datos obteniéndose todas las entregas en cola o en corrección (`:enqueued` o `:running`). Esta lista de entregas se modifica actualizándose su estado a `:pending`. Una vez salvado el estado de las entregas en la base de datos, se detiene el *pool* de procesos (véase sección 2.1), finalizando todos los procesos de corrección.

Por comodidad, estos botones se han colocado en la vista de la cola definida en la sección anterior.

Figura 4.9: Vista de la cola con los botones de funcionalidad en masa

4.0.5. REQE-1: Estimación de tiempo de entrega

Ante la duda de los estudiantes sobre cuánto va a tardar en corregirse su envío, se ha diseñado un proceso de estimación de tiempo de entrega.

Consideraciones

Se han realizado las siguientes consideraciones:

- El sistema solo posee una cola, por lo que es obligatorio el encolado de entregas correspondientes a distintas prácticas (los tiempos de compilación y ejecución en diferentes lenguajes no son los mismos).
- Los tiempos de procesado de entregas para una misma práctica se pueden considerar similares.
- Las estimaciones dependen de muchos factores (la carga del sistema, la potencia de la máquina en la que el sistema está alojado, etc.), por lo que no deben ser persistentes.
- Ante el continuo acceso a estimaciones y actualizaciones, surge la necesidad de una baja latencia.

Teniendo las consideraciones previas en mente, se propone almacenar la información en dos estructuras de datos de tipo diccionario en memoria volátil. Una

tabla para almacenar los promedios de entrega por práctica y otra tabla con las estimaciones de las entregas en curso.

Proceso

El proceso para obtener las estimaciones para cada entrega es el siguiente:

En el momento de recepción de una entrega, el sistema le asigna un tiempo de corrección igual a la suma de los promedios de las entregas que estén en la cola actualmente más el tiempo medio de la práctica a la que pertenece la entrega. Si no hay datos sobre la práctica, el tiempo medio por defecto son 5 segundos.

Al terminar la corrección, se actualiza la media de entrega de la práctica con el tiempo de corrección de esta entrega terminada recientemente. Se actualizan los tiempos de corrección de las demás entregas en cola, restando el tiempo estimado de la práctica que acaba de terminar y actualizando las estimaciones a la nueva media.

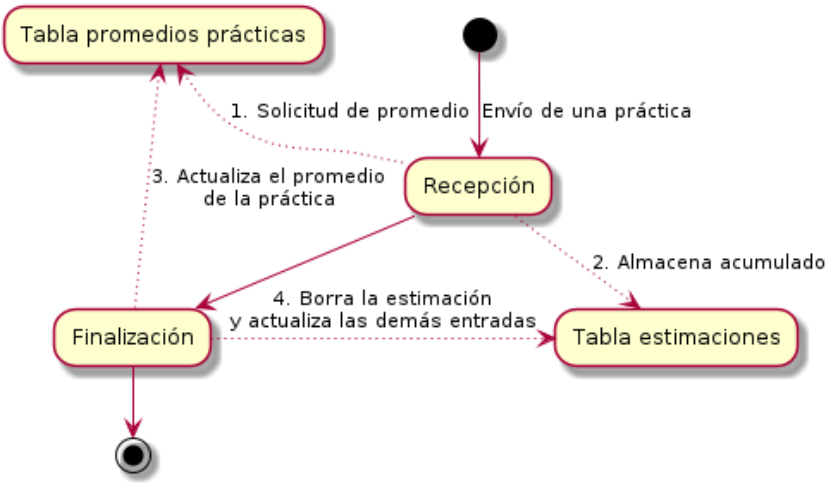

Figura 4.10: Proceso de estimación

En el caso de que un administrador cambie alguna característica relevante para el procesado de las prácticas, toda la información establecida por las estimaciones es borrada. En otro caso (como puede ser modificar la fecha límite de entrega), la información se mantiene.

Si se cancela una corrección, se elimina la estimación correspondiente a la entrega.

Ejemplo:

Teniendo en cola tres entregas (`[E1,E2,E3]`), cada una con un tiempo de procesado estimado distinto (`E1 -> 2s, E2 -> 3s, E3 -> 4s`), el conjunto de estimaciones resultaría en un acumulado progresivo (`[Te1,Te1+Te2,Te1+Te2+Te3]`), lo que para este ejemplo resultaría en `[2,5,8]`.

Capítulo 5

Implementación

En este capítulo se detallan las soluciones realizadas para la implementación de los diseños elegidos.

5.0.1. REQS-1: Mejora de la información recopilada

El esquema empleado para representar una entrega es parte de la biblioteca *Ecto* (*Ecto.Schema*). Este esquema convierte datos externos en estructuras de datos de *Elixir*, que posteriormente se almacenan en la base de datos mediante otra parte de *Ecto*, *Ecto.Repo*. Estos datos son modificados mediante *Ecto.Changeset*, que permite filtrado, validación, conversión de tipos, etc.

La modificación de los esquemas consiste en añadir al esquema de entrega el campo `field(:enqueued_at, :naive_datetime)`, donde se almacena un instante de tiempo.

```
@primary_key false
typed_schema "submissions" do
  field(:id, :binary_id, primary_key: true)
  field(:mark, :decimal)

  # Checking state
  field(:submission_state, Database.Custom.SubmissionState,
        default: :pending)
  field(:enqueued_at, :naive_datetime)
  field(:started_at, :naive_datetime)
  field(:finished_at, :naive_datetime)

  # Extra data
  timestamps()

  # References
  belongs_to(:project, Database.Subjects.Project,
             type: :binary_id)
  belongs_to(:group, Database.Students.Group,
             type: :binary_id)
end
```

Al enumerado de estados se le añade el estado `:enqueued`:

```
defmodule Database.Custom.SubmissionState do
  use EctoEnum,
    pending: "pending",
    running: "running",
    enqueued: "enqueued",
    processed: "processed",
    failed: "failed"
end
```

La nueva actualización del estado de la entrega en el momento de encolado se realiza mediante un *changeset*. En la definición de un *changeset* se indican las acciones a realizar para completar la modificación requerida. Suele ser común la validación de expresiones regulares, la existencia del campo a modificar o la comprobación de permisos del usuario que realiza la acción.

Para poder modificar el instante de encolado `:enqueued_at`, es necesario incluirlo en los atributos manejados por el *changeset*.

```
@doc false
def changeset(submission, attrs) do
  submission
  |> cast(attrs, [
    :id,
    :mark,
    :project_id,
    :group_id,
    :submission_state,
    :enqueued_at,
    :started_at,
    :finished_at
  ])
  |> validate_required([
    :id,
    :project_id,
    :group_id,
    :submission_state
  ])
end
```

Por lo tanto, la nueva actualización que se ejecuta antes del instante encolado de la entrega es la siguiente:

```
  Subjects.update_submission(submission, %{
    mark: nil,
    submission_state: :enqueued,
    enqueued_at: NaiveDateTime.utc_now(),
    started_at: nil,
    finished_at: nil
  })
```

Por último, para implementar la migración se ha empleado otra funcionalidad de *Ecto*, *Ecto.Migration*, que permite definir migraciones y aplicar un conjunto de ellas para alcanzar el estado de la base de datos que interese. Por ejemplo, en cada reinicio de la base de datos se borran todos los datos almacenados, se vuelve al estado inicial y posteriormente se ejecutan todas las migraciones en orden para alcanzar el estado de la base de datos requerida sin datos almacenados.

Para implementar la migración se han definido cambios en el esquema de las entregas, añadiendo la columna *enqueued_at* (que almacena el instante de encolado). Una vez definida la columna para todas las entradas de la tabla, se establece el instante de encolado (`:enqueued_at`) como el instante de llegada al sistema (`:inserted_at`) en las entregas previas a la implementación de esta funcionalidad.

```
defmodule Database.Repo.Migrations.UpdateSubmissionStateEnqueuedAt do
  use Ecto.Migration
  import Ecto.Query

  def change do
    #adds column "enqueued_at"
    alter table(:submissions) do
      add :enqueued_at, :naive_datetime
    end

    #waits until column is created
    flush()

    #if submission is already marked, enqueued_at is inserted_at
    from(s in "submissions",
      update: [set: [enqueued_at: s.inserted_at]],
      where:  s.submission_state == "failed" or
              s.submission_state == "processed"
    )|> Database.Repo.update_all([])
  end
end
```

Una vez realizados todos estos cambios, se modifican las plantillas HTML (2.3) para poder visualizar los nuevos datos.

En la vista de una entrega (tanto para administradores como para alumnos), se incluye un campo para visualizar el instante de encolado y se modifica el cálculo existente del tiempo en cola, que pasa de ser `:started_at - :inserted_at` a `:started_at - :enqueued_at`.

Tanto en la vista de práctica para ambas webs como la vista de cola para la administración se ha incluido el nuevo cálculo de tiempo en cola.

5.0.2. REQA-3: Límite de entregas concurrentes por usuario

La implementación de la nueva cadena de comprobaciones resulta en añadir una fase al proceso. Es importante aclarar que los grupos se forman por práctica, por lo tanto, la limitación sólo se aplica para entregas en una misma práctica.

La comprobación realiza una consulta y en el caso de que la lista de entregas en curso del grupo no sea vacía, se propaga un error al controlador web de entregas en la aplicación de estudiantes, para que este haga saltar una notificación de alerta.

```
check_no_submissions_running: fn
  %{check_group: %{id: group_id}} ->
  n_submissions = Subjects.get_group_running_submissions(group_id)
  if length(n_submissions) > 0,
```

```
    do: {:error, :submission_running},
    else: {:ok, nil}
end,
```

La consulta solicita todas las entregas del grupo emisor que no tengan nota:

```
@doc """
Get pending|enqueued|running submissions from a group in a project
"""
def get_group_running_submissions(group_id) do
  query =
    from(s in Submission,
      where: s.group_id == ^group_id and is_nil(s.mark)
    )
  Repo.all(query)
end
```

El código que controla la creación de nuevas entregas es el siguiente:

```
def create(conn, %{"project_id" => pid} = params, auth) do
  case Create.exec(auth.id, params) do
    {:ok, _submission} ->
      conn
      |> put_flash(:info, gettext("Submission␣successfully␣sent."))
      |> redirect(to: Routes.project_path(conn, :show, pid))

    {:error, %Ecto.Changeset{}} ->
      conn
      |> put_flash(:error, gettext("Invalid␣params"))
      |> redirect(to: Routes.submission_path(conn, :new, pid))

    {:error, error_code} ->
      conn
      |> put_flash(:error, process_error(error_code)) # Manejo de errores
      |> redirect(to: Routes.submission_path(conn, :new, pid))
  end
end
```

En caso de ser errónea la petición, el controlador maneja los códigos de error:

```
  defp process_error(error) do
    case error do
      :invalid_project -> gettext("The␣project␣is␣not␣valid")
      :close -> gettext("The␣submission␣is␣closed")
      :missing_group -> gettext("The␣group␣is␣not␣valid")
      :max_submissions ->
        gettext("You␣have␣already␣made␣all␣the␣available␣submissions")
      :submission_running ->
        gettext(
          "There␣is␣one␣of␣your␣submissions␣running.␣Wait␣until␣is␣done␣to␣send␣another␣one"
        )
      :invalid_files -> gettext("Invalid␣files")
      _ -> gettext("Unexpected␣error")
    end
  end
```

5.0.3. REQA-1: Visión general de la cola

Para crear esta vista, se ha tenido que implementar un controlador que pudiera ser requerido por el enrutador, una plantilla a renderizar en el momento que un administrador quisiera acceder al recurso y varias consultas a la base de datos que proporcionan los datos a visualizar.

Sobre el controlador, se han definido las acciones a realizar cuando un usuario solicite el recurso. Estas acciones son: obtener mediante una consulta a la base de datos todas las entregas pendientes (en cola y en corrección) y posteriormente renderizar una plantilla con los datos obtenidos.

```
def index(conn, _params) do
  running_submissions = Subjects.get_running_submissions(true)
  render(conn, "index.html", submissions: running_submissions)
end
```

La plantilla sigue la estructura general (véase 2.3). Como componente principal tiene una tabla (componente definido en trabajos anteriores) que muestra información sobre las entregas resultantes de la consulta. En ella se muestra la asignatura a la que pertenece la práctica, la práctica a la que pertenece la entrega, el grupo que realiza el envío, el instante de la recepción, el tiempo en cola, el tiempo en ejecución, el estado y las posibles acciones a realizar para la misma (por ejemplo, volver a ejecutar su corrección).

Para obtener las entregas en curso y la asignatura a la que pertenecen hay que realizar sendas consultas a la base de datos (ya que la asignatura no está incluida en el esquema de una entrega). La consulta para obtener las entregas en curso consiste en obtener cualquiera que tenga como nota `nil` (es decir, que no haya finalizado).

```
def get_running_submissions(preload \\ false) do
  query =
    from(
      s in Submission,
      where: is_nil(s.mark)
    )

  query =
    if preload do
      query
      |> join(:left, [p], s in assoc(p, :project))
      |> preload([p, s], group: [:students], project: s)
    else
      query
    end
  Repo.all(query)
end
```

La consulta para obtener el nombre de una asignatura conociendo el identificador de una práctica consiste en realizar una unión entre las tablas *subjects* y *projects*, obteniendo la unión que tenga como identificador de práctica el argumento de la consulta. Devuelve únicamente el nombre de la asignatura.

```
def get_subject_name_from_project(project_id) do
```

```
  query =
    from p in Project,
    join: s in Subject, on: p.subject_id == s.id,
    where: p.id == ^project_id,
    select: s.name

  Repo.one(query)
end
```

5.0.4. REQA-2: Recuperación de la cola

Para implementar las ejecuciones y paradas de correcciones en masa se ha tenido que modificar el controlador de la cola. Se ha añadido lógica al manejo de los estados de entregas y, por último, se han añadido elementos en la plantilla web para poder acceder a la funcionalidad.

En el enrutador, se han definido los siguientes recursos para las acciones a implementar:

```
scope "/queue", AdminWeb do
  pipe_through(:browser)

  get("/", QueueController, :index)
  get("/stop_all", QueueController, :stop_all)
  get("/run_pending", QueueController, :run_pending)
end
```

En el controlador, para que estos recursos sean funcionales, se han añadido dos funciones que manejan las peticiones de los administradores. En ambas, el controlador realiza la petición a la lógica del sistema, y una vez terminado, se renderiza en la web una notificación sobre la terminación exitosa de la operación.

```
def stop_all(conn, _params) do
  ProcessorPool.stop_running_submissions()
  conn
  |> put_flash(:info, gettext("Queue␣successfully␣stopped."))
  |> redirect(to: Routes.queue_path(conn, :index))
end

def run_pending(conn, _params) do
  ProcessorPool.run_pending_submissions()
  conn
  |> put_flash(:info, gettext("Pending␣submissions␣successfully␣enqueued."))
  |> redirect(to: Routes.queue_path(conn, :index))
end
```

En cuanto a la lógica sobre la detención de las entregas en curso, se obtienen todas las entregas en corrección o en cola (`:running` y `:enqueued`), se actualiza su estado a `:pending` y se llama al *pool* de procesos para la detención de todos los trabajadores.

```
@doc """
Stops all running processes and updates state to pending
"""
```

```
def stop_running_submissions() do
  # db
  running_submissions =
    Subjects.list_submissions_by_state(:running) ++
      Subjects.list_submissions_by_state(:enqueued)

  Enum.each(
    running_submissions,
    fn x ->
      Subjects.update_submission(x, %{
        submission_state: :pending,
        enqueued_at: nil,
        started_at: nil,
        finished_at: nil
      })
    end
  )

  Task.async(fn ->
    :poolboy.stop(Logic.Submissions.ProcessorPool)
  end)
end
```

Respecto a la ejecución de entregas en masa, se ha decidido que sea únicamente para procesos en estado `:pending`. El proceso es similar a la detención en masa, escogiéndose las entregas en dicho estado y realizando peticiones consecutivas al *pool* de procesos trabajadores para poner en cola las entregas seleccionadas.

```
@doc """
Runs all pending submissions
"""
def run_pending_submissions() do
  pending_submissions = Subjects.list_submissions_by_state(:pending)

  Enum.map(
    pending_submissions,
    fn s ->
      run(s.id)
    end
  )
end
```

Se ha diseñado la consulta `Subjects.list_submissions_by_state(state)` para ambas funcionalidades. Devuelve todas las entregas con el estado argumentado, ordenadas de la más antigua a la más moderna.

```
@doc """
Gets submissions by state: pending | running | enqueued | finished
"""
def list_submissions_by_state(state) do
  state_type =
    case state do
      :pending -> "pending"
      :running -> "running"
      :enqueued -> "enqueued"
:processed -> "processed"
:failed -> "failed"
    end
```

```
  query =
    from(
      s in Submission,
      where: s.submission_state == ^state_type,
      order_by: [asc: s.enqueued_at]
    )

  Repo.all(query)
end
```

Una vez implementada la lógica, se han creado los botones en la vista de la cola. Se han añadido a la cabeza de la tabla previamente implementada por comodidad y visibilidad durante labores de mantenimiento por parte de los administradores. Cada botón llama al correspondiente recurso definido previamente en el enrutador.

5.0.5. REQE-1: Estimación de tiempo de entrega

Para el almacenamiento de tiempos medios por práctica y la estimación actual de cada entrega se ha decidido emplear la biblioteca *:ets (Erlang Table Storage System)* [16]. Esta permite almacenar grandes cantidades de datos en memoria volátil y acceder a ellos desde cualquier proceso mediante estructuras llamadas *tablas*.

Para implementar el diseño descrito en 4.0.5, se definen dos tablas, una que almacena los **promedios de tiempo por entrega** y otra que almacena la **fecha estimada de resolución**.

La tabla de promedios tiene como clave el identificador de una práctica y como valor una tupla con el promedio y el número de entregas registradas desde que se inició la instancia actual del sistema. La tabla de fechas tiene como clave el identificador de una entrega y como valor el instante de tiempo estimado de resolución.

Se ha decidido implementar una solución con instantes de tiempo en vez de valores numéricos, ya que al realizar la diferencia entre el instante almacenado y el actual se consigue una mayor precisión a lo largo del paso del tiempo. Con valores numéricos, las estimaciones se mantienen inmutables entre el final de una corrección y la siguiente, lo que ofrecería una menor precisión.

Ejemplo:

Cola: `[2,5,6,10]` (en segundos, nótese que las estimaciones son acumuladas)

Implementación con valores numéricos:

-1s: `[2,5,6,10]`

-2s: `[2,5,6,10]`

-3s: `[3,4,8]`

-4s: `[3,4,8]`

-5s: `[3,4,8]`

Implementación con instantes de tiempo (en 4.0.5 se describe la diferencia realizada para poder obtener esta estimación):

-1s: `[2,5,6,10]`

-2s: `[1,4,5,9]`

-3s: `[3,4,8]`

-4s: `[2,3,7]`

-5s: `[1,2,6]`

Como se puede ver, la implementación numérica representa una estimación aceptable únicamente en los momentos en los que las entregas finalizan su corrección. Por lo tanto, se emplea la solución con los instantes de tiempo para la implementación de la funcionalidad.

Capítulo 6

Análisis de impacto

Este proyecto se concibió con el objetivo de aumentar la calidad del sistema mediante una remodelación y mejora del sistema de entrega. Para los administradores, la implementación de las nuevas funcionalidades tales como la ejecución en masa de tareas pendientes o la visión global de la cola de entregas en curso aportará calidad de vida a las tareas de administración, mantenimiento y resolución de errores futuros.

En cuanto al alumnado, este proyecto pretende proporcionar mayor certidumbre en las entregas mostrando un tiempo de procesado estimado y mejorando la información sobre la entrega durante el proceso.

En cuanto a recursos, la limitación de entregas por parte de un grupo y la capacidad de detener entregas en masa puede aligerar la carga del sistema, lo que permite un menor consumo de la máquina y un sistema más robusto.

La implementación de estimaciones según instantes de tiempo está diseñada con la idea de facilitar la implementación de la tecnología *LiveView* de *Phoenix* (véase la sección de situación actual 2.3) con el actual sistema de estimación. El objetivo final de este sistema es la visión en tiempo real del tiempo restante estimado para la corrección de una entrega, no siendo necesario el actual renderizado sucesivo de la página entera para poder obtener los datos actualizados.

En cuanto al impacto personal, la realización de este trabajo me ha permitido poner en práctica mis habilidades adquiridas durante estos años de carrera. La necesidad de definir requisitos y, posteriormente, realizar un diseño e implementarlo me ha permitido discernir cómo es el proceso de desarrollo en un trabajo de gran tamaño. En concreto, el proceso de conceptualizar diseños y presentarlo al dueño del proyecto, que lo puede denegar o aprobar.

Además, este proyecto escrito en *Elixir* me ha descubierto la programación funcional, paradigma con el que me he sentido cómodo rápidamente y en el que tengo intención de continuar realizando proyectos diversos.

Capítulo 7

Conclusiones y trabajo futuro

Para concluir esta memoria, se presentarán las conclusiones acerca del trabajo realizado y se indicarán puntos para futuros desarrollos.

7.1. Conclusiones

Este trabajo implementa mejoras en torno al trato de las entregas por parte del sistema. En el momento en el que se redacta esta memoria, la mejora de recopilación de información sobre las entregas [4.0.1] (que añade el nuevo estado `:enqueued`, el campo con el instante de encolado `:enqueued_at` y la actualización en el momento de encolado) y la vista de cola para la administración [4.0.3] (que añade una nueva vista sobre todas las entregas en curso en el sistema) están en producción. Las restantes se encuentran en desarrollo, pendientes de acoplarse al desarrollo general del sistema.

En cuestiones de desarrollo, se prevé que este trabajo realice las siguientes aportaciones a *Deliverit*:

- Moderada liberación de la carga de entregas al sistema.
- Aumento de la confianza del estudiante con la herramienta.
- Mejora sustancial en la calidad de vida del administrador en sus labores.
- Reducción de las discordancias en el estado de una entrega con la realidad.

7.2. Trabajo futuro

De cara a futuros trabajos se proponen varios puntos:

- Un nuevo sistema de cola y *pool* de procesos trabajadores.
- Sistema de prioridad.
- Funcionalidad en la vista de cola que permita detener la corrección de una única entrega.

- Manejo de errores si hay fallo en el proceso corrector.

7.2.1. Nuevo sistema de cola

Debido al análisis sobre la realización de un sistema de prioridad y la inclusión de la detención de corrección sobre una única entrega, se ha detectado una carencia en el sistema de encolado.

Actualmente, se delega toda la gestión de procesos trabajadores (los cuales realizan las correcciones propiamente dichas) en la biblioteca *Poolboy*. Esto produce una cierta incertidumbre sobre el proceso que corrige una entrega en concreto, lo que imposibilita la implementación de ciertas funcionalidades. Por ello, se considera actualmente *Poolboy* insuficiente para las necesidades de *Deliverit*.

Para este cambio, se sugiere el uso de *HoneyDew* [17], un *pool* de trabajadores, y una cola de tareas a su vez, escrita en *Elixir*. Esta biblioteca permite configurar una base de datos como la cola de tareas, lo que puede solucionar este problema.

7.2.2. Sistema de prioridad

Para poder establecer una prioridad en el orden de corrección, es necesario poder manejar los procesos que realizan dichas correcciones.

En el caso de que todos los procesos trabajadores estén ocupados, la tarea es encolada **siempre** en la última posición de la cola, lo que imposibilita la capacidad de priorizar entregas, ya que no es posible anteponer ciertas entregas con mayor prioridad a otras pese a ser incluidas más tarde en la cola de tareas del *pool*.

Por ello, se propone, una vez implementado el nuevo sistema de cola, la modificación del esquema de entrega incluyendo cierto valor de prioridad (por ejemplo, numérico) que habilita la implementación de este requisito.

7.2.3. Detención de entrega única

Durante el diseño de las funcionalidades de manejo masivo de entregas se analizó la posibilidad de implementar la detención de una única entrega, para facilitar la labor de la administración.

Durante este análisis se detectó la falta de conocimiento sobre qué proceso es el que ejecuta la corrección, y directamente la inaccesibilidad a la cola de *Poolboy*.

Por lo tanto, previamente implementado el nuevo sistema de cola, se propone implementar una estructura de datos que almacene los identificadores de los procesos y la correspondiente entrega que procesan. A partir de esta estructura, adaptar el proceso general de corrección del sistema y añadir esta funcionalidad.

7.2.4. Manejo de errores si hay fallo en el proceso corrector

Por último, se propone actualizar el estado de las entregas en las que suceda un error durante la corrección (en el entorno aislado). Esta actualización marcaría

una entrega como `:broken` y sería visible desde la vista de cola.

Bibliografía

[1] A. Mareca. (2020) Sistema de entrega deliverit: Prototipo. [Online]. Available: https://oa.upm.es/63096/1/TFG_ANDRES_MARECA_MINGUEZ.pdf

[2] A. Contreras. (2020) Evoluciones y refactorizaciones de un sistema de entrega de prácticas. [Online]. Available: https://oa.upm.es/66206/1/TFG_AARON_CONTRERAS_FONTANILLO.pdf

[3] E. Stenman. (2020) The erlang runtime system. [Online]. Available: https://blog.stenmans.org/theBeamBook/

[4] K. W. Devin Torres, Andrew Thompson. (2014) Poolboy - a hunky erlang worker pool factory. [Online]. Available: https://github.com/devinus/poolboy

[5] D. Inc. (2023) What is a container. [Online]. Available: https://www.docker.com/resources/what-container/

[6] A. Mareca. (2020) Dockerex. [Online]. Available: https://github.com/aherranz/dockerex

[7] Dashbit. (2020) Ecto. [Online]. Available: https://github.com/elixir-ecto/ecto

[8] T. P. G. D. Group. (1996-2023) Postgresql database management system. [Online]. Available: https://www.postgresql.org/

[9] B. T. Chris McCord and J. Valim, *Programming Phoenix 1.4*. The Pragmatic Programmers, 2019.

[10] B. Frost, *Atomic Design*. Brad Frost, 2016.

[11] B. A. Tate and S. DeBenedetto, *Programming Phoenix LiveView*. The Pragmatic Programmers, 2023.

[12] IEEE. (1998) Especificaciones de los requisitos del software. [Online]. Available: https://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf

[13] T. E. Team. (2023) Elixir. [Online]. Available: https://elixir-lang.org/

[14] V. Driessen. (2010) A successful git branching model. [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/

[15] Atlassian. (2023) What is trello? [Online]. Available: https://trello.com/tour

[16] S. Ericsson. (1997-2023) Erlang table storage system. [Online]. Available: https://www.erlang.org/doc/man/ets.html

[17] M. Shapiro. (2014) Honeydew. [Online]. Available: https://github.com/koudelka/honeydew

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed May 31 19:33:14 CEST 2023
	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)