eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

IKASTENBOT II

MEMORIA

Alumno: Pérez Gómez, David

Director: Pereira Varela, Juan Antonio

Curso: 2018 - 2019

Fecha: 20/06/2019

Resumen

Este trabajo consiste en participar en el desarrollo del proyecto Ikastenbot, un bot de Telegram diseñado para ayudar a estudiantes universitarios en el desarrollo y documentación de su Trabajo de Fin de Grado.

La aportaciones al proyecto incluyen mejoras al corrector ortográfico, avisos acerca de las planificaciones temporales recibidas y más opciones para que el usuario pueda ajustar la planificación definida.

El bot está siendo desarrollado en PHP, almacena la información en una base de datos MySQL y hace uso de la API de Telegram para comunicarse con los usuarios.

Palabras clave: bot, chatbot, Telegram, PHP, asistente, Trabajo de Fin de Grado.

Laburpena

Lan hau Ikastenbot proiektuaren garapenean parte hartzea da. Ikastenbot unibertsitateko ikasleei Gradu Bukaerako Lanaren garapenean eta dokumentazioan laguntzen dien Telegram-eko bot-a da.

Proiektuari egindako ekarpenetan sartzen dira zuzentzaile ortografikoarentzako hobekuntzak, jasotako plangintzei buruzko oharrak eta aukera gehiago erabiltzaileak plangintza zehatza finkatu ahal dezala.

Bot-a PHP programazio-lengoaiarekin egin da, informazioa MySQL datubase batean gordetzen du eta erabiltzaileekin komunikatzeko Telegram-eko API-a erabiltzen du.

Gako-hitzak: bot, chatbot, Telegram, PHP, laguntzailea, Gradu Bukaerako Lana.

Abstract

This work consists taking part in the development of the Ikastenbot project, a Telegram bot designed to assist university students in the development and documentation of their Final Degree Project.

The contributions made to the project include improvements to the orthographic corrector, notices about the project management and more options for the user to adjust their planning.

The bot is being developed in PHP, it stores it's information in a MySQL database and makes use of Telegram's API to comunicate with the users.

Palabras clave: bot, chatbot, Telegram, PHP, assistant, Final Degree Project.

Índice

1. Introducción

Telegram es una aplicación de mensajería instantánea. Fue lanzada en 2013 y desde entonces se ha convertido en una de las principales opciones en cuanto a aplicaciones de mensajería. A pesar de que el número de usuarios aún no puede compararse con los de sus mayores competidores como Skype, Facebook Messenger y WhatsApp[1], Telegram ha recibido muchos elogios por su seguridad y resulta muy atractivo para los usuarios gracias al hecho de que es una aplicación de código abierto[2], transparente, y que pretende mantenerse gratis en el futuro[3], así como la variedad de opciones y herramientas que ofrece a usuarios con conocimientos técnicos.

Una de estas herramientas es la posibilidad de crear bots. Estos bots son aplicaciones que se pueden comunicar con sus usuario mediante mensajes y comandos enviados a través de la API de Telegram[4]. Ikastenbot es uno de estos bots, y fue concebido para ser utilizado por estudiantes universitarios como un asistente personal que les ayude a la hora de realizar sus trabajos de fin de grado. Esta ayuda puede venir de muchas formas distintas: tener una lista de preguntas frecuentes (FAQ) y sus respuestas, revisar borradores de la memoria en busca de errores, o enviar recordatorios automatizados para fechas importantes, por nombrar algunos.

Ikastenbot está todavía en fase de desarrollo, ideando y añadiendo nuevas funciones, y no está aún abierto al público. En este trabajo de fin de grado, se va a formar parte del equipo de desarrollo, llevando a cabo los encargos que el líder del equipo o el tutor del trabajo consideren adecuados y necesarios, para añadir nuevas funcionalidades al bot o mejorar las ya existentes.

2. Planteamiento Inicial

2.1. Objetivos

El objetivo principal de este trabajo es llevar a cabo las tareas encargadas y obtener un resultado que cumpla con las necesidades y especificaciones dadas de una manera satisfactoria para todos los implicados (cliente, tutor y alumno).

Por otro lado se pueden definir una serie de objetivos personales que se esperan cumplir con este proyecto. Estos objetivos se centran mayormente en el aprendizaje y en familiarizarse con las tecnologías que se utilizarán: adquirir conocimientos de PHP más allá de lo visto en la carrera; aprender cómo funciona, cómo crear y cómo trabajar con un bot de Telegram, etc. También se busca obtener algo de experiencia al trabajar en un equipo algo más cercano a la realidad que lo experimentado hasta la fecha, en el que todo el trabajo realizado recibe feedback y críticas constructivas y se trabaja para una aplicación real, por lo que el diseño, la documentación y las pruebas son aspectos importantes por razones más allá de solamente cumplir una rúbrica.

2.2. Arquitectura

En este apartado se describe la arquitectura de un bot de Telegram.

Los bots son simplemente cuentas especiales de Telegram que no requieren un número de teléfono. Estas cuentas se pueden comunicar con usuarios reales enviando y recibiendo mensajes y sus acciones se pueden controlar desde una aplicación externa mediante peticiones y respuestas HTTP utilizando la API REST que ofrece Telegram[5].

Cuando la cuenta de un bot recibe un mensaje de un usuario, solamente Telegram es consciente de este hecho. Por ello, para hacer una aplicación que utilice un bot como medio de interacción con el cliente (como es el caso de Ikastenbot), es necesario que la aplicación reciba los mensajes enviados al bot. Telegram ofrece dos estrategias para lograrlo:

- **Polling**

  La estrategia de polling (encuesta) consiste en preguntar directamente a Telegram qué mensajes se han recibido. Se envía una petición HTTP y Telegram responde con una lista de los mensajes (updates) que el bot ha recibido y que aún no se han confirmado como detectadas por la aplicación. Este último hecho significa que, si no se toman las medidas apropiadas, Telegram puede devolver, en ocasiones distintas, listas que contienen mensajes repetidos, por lo que es responsabilidad de la aplicación gestionar qué mensajes han sido leídos para evitar respuestas duplicadas.

  Dado que este método se inicia en la aplicación, para asegurar que los mensajes de los usuarios son respondidos lo antes posible, es necesario ejecutarlo repetidamente en intervalos tan cortos como sea posible.

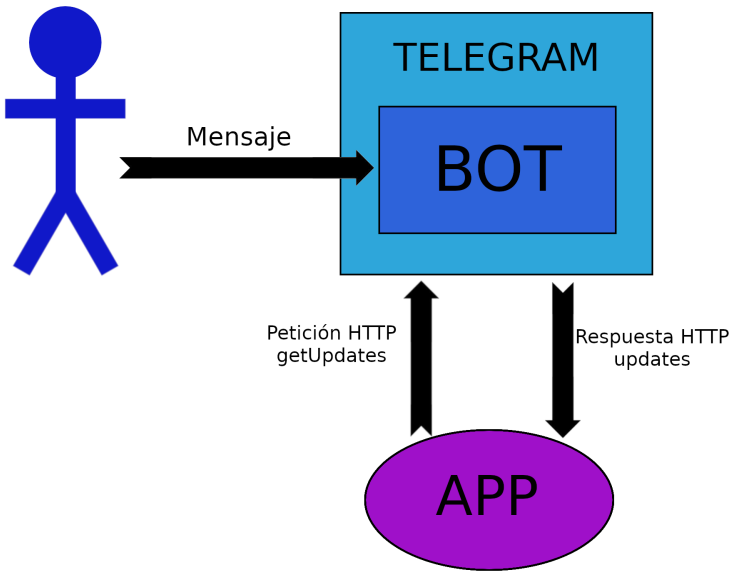

Figura 1: Captura de mensajes mediante polling.

- **Webhooks**

  Un webhook es una callback, una llamada, HTTP definida por un usuario que normalmente se lanza en respuesta a un evento[6]. En este caso,

se puede activar un webhook asociado a un bot y a una dirección url de manera que cuando dicho bot reciba un mensaje, Telegram automáticamente lance una petición HTTP a la url especificada informando del mensaje. De esta forma se pueden recibir los mensajes de los usuario de forma automática, sin necesidad de realizar peticiones constantemente.

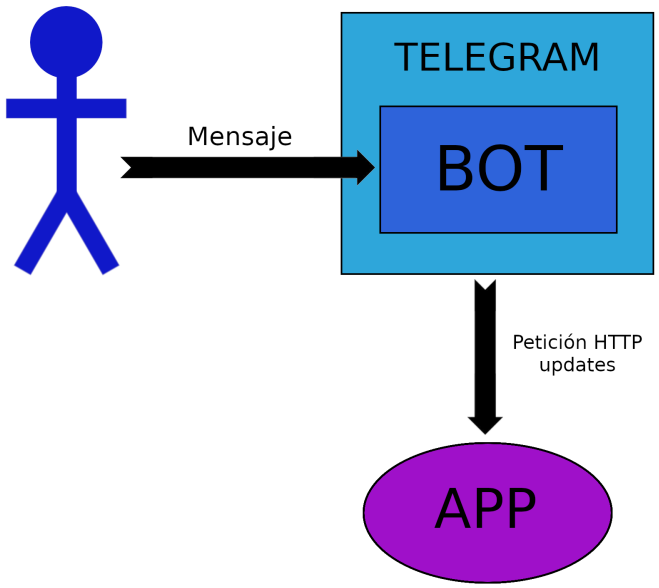

Figura 2: Captura de mensajes mediante webhooks.

Normalmente la estrategia de webhooks se prefiere a la de polling, ya que el recibir mensajes asíncronos de forma automática es considerablemente más eficiente y cómodo que mantener un script corriendo cada pocos segundos y tener que llevar la cuenta de los mensajes ya recibidos para preguntar solo por mensajes posteriores.

Sin embargo, la estrategia de polling sigue teniendo su utilidad, sobre todo durante fases de desarrollo y pruebas, ya que no requiere una url activa con soporte HTTPS y se puede activar o desactivar el bot simplemente ejecutando o deteniendo el script que realiza el polling.

Una vez que la aplicación ha recibido el mensaje del usuario puede proce-

sarlo y llevar a cabo las acciones correspondientes. En cualquier momento del proceso se puede enviar un mensaje al usuario mediante una petición HTTP utilizando la API REST de Telegram.

2.3. Herramientas Utilizadas

En esta sección se describen las herramientas que se utilizarán a lo largo del proyecto.

2.3.1. PhpStorm

Figura 3: Logo PhpStorm.

PhpStorm[1] será el principal entorno de desarrollo integrado (IDE) utilizado.

Se ha escogido este frente a otras alternativas, como Eclipse o NetBeans, debido a la familiaridad. Pese a que esta será la primera vez trabajando extensamente con PHP y nunca se ha utilizado PhpStorm directamente, se tiene considerable experiencia con otros IDEs de la misma empresa, JetBrains, tales como IntelliJ IDEA (para Java) y PyCharm (para Python). Estos entornos funcionan de forma muy similar y han ofrecido una experiencia de usuario y unos resultados más que satisfactorios, por lo que es de esperar que PhpStorm no será diferente.

[1] https://www.jetbrains.com/phpstorm/

2.3.2. Atom

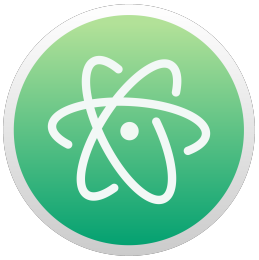

Figura 4: Logo Atom.

Atom[2] es un editor de texto y código de código abierto con soporte para una gran cantidad de lenguajes y alta personalización mediante la instalación de paquetes. Este software se utilizará como editor de código alternativo, siendo una opción más rápida de iniciar y ligera que PhpStorm. A pesar de que no tenga todas las funciones que pueda tener un IDE completo, vendrá bien para modificaciones que no requieran asistencia compleja por parte del editor y para sesiones breves de trabajo.

2.3.3. LibreOffice Calc

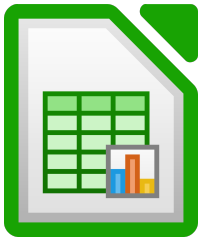

Figura 5: Logo LibreOffice Calc.

LibreOffice Calc[3] es un programa de hojas de cálculo. Se utilizará principalmente para tomar nota del tiempo dedicado a las distintas facetas del trabajo.

[2]https://atom.io/
[3]https://www.libreoffice.org/discover/calc/

2.3.4. PlantUML

Figura 6: Logo PlantUML.

PlantUML[4] es una herramienta de código abierto que permite crear distintos tipos de diagramas UML a partir de archivos de texto plano en un lenguaje similar a Markdown. Se utilizará para crear todos diagramas a lo largo del proyecto, con la excepción de los diagramas Gantt.

La razón de escoger este método en lugar de otros como Visual Paradigm es que PlantUML, al trabajar con archivos de texto, permite ediciones rápidas y fáciles con cualquier editor de textos y gestiona automáticamente el apartado visual (cajas, líneas, tamaños, posiciones, espaciado, etc.), mientras que otros programas requieren trabajar con una interfaz gráfica no siempre cómoda o receptiva.

2.3.5. GanttProject

GanttProject[5] es un programa para de gestión de proyectos especialmente útil para crear diagramas Gantt. Todos los diagramas Gantt de este proyecto de crearán utilizando GanttProject.

Se ha escogido GanttProject principalmente debido a que parte del trabajo a realizar involucra diagramas creados con este programa, por lo que utilizarlo, por un lado ayuda a adquirir conocimientos y a familiarizarse con su funcionamiento, y por otro, aporta más entradas al registro de diagramas Gantt que se lleva en el proyecto.

[4] http://plantuml.com/en/
[5] https://www.ganttproject.biz/

Figura 7: Logo GanttProject.

2.3.6. LaTeX y Overleaf

Figura 8: Logo LaTeX.

LaTeX[6] es una herramienta para producir textos y documentos técnicos. Frente a otros procesadores de texto, LaTeX ofrece ventajas como el tratar con documentos grandes sin ralentizarse o colgarse y el mantener automáticamente un apartado visual coherente y de aspecto profesional.

Figura 9: Logo Overleaf.

LaTeX se puede utilizar localmente mediante varios editores como Texmaker o el propio Atom. Sin embargo, se ha optado por emplear la plataforma Overleaf[7], la cual permite alojar documentos LaTeX en la web, permitiendo acceso desde cualquier dispositivo. Esto, además, hace más fácil compartir el

[6]https://www.latex-project.org/
[7]https://www.overleaf.com/

documento, por ejemplo, con el tutor para revisiones. Overleaf también ofrece un editor con autocompletado, árbol de archivos y una previsualización en tiempo real del resultado.

2.3.7. Git y GitLab

El proyecto de Ikastenbot está alojado en un repositorio de GitLab. Por ello, se utilizará el mismo sistema de control de versiones y repositorio para el trabajo.

Figura 10: Logo Git.

Git[8] es un sistema de control de versiones distribuido (DVCS) que permite gestionar los cambios a lo largo del desarrollo de un proyecto para facilitar su posterior revisión o para evitar que cambios sin relación afecten unos a otros.

GitLab[9] es un sitio web que permite alojar repositorios git en la nube. Todos los cambios y adiciones se enviarán al repositorio ya existente.

2.4. Alcance

El alcance total del proyecto es desconocido al principio, ya que los encargos del cliente no están fijados, sino que se pedirán en función de las necesidades que surjan.

[8]https://git-scm.com/
[9]https://about.gitlab.com/

Figura 11: Logo GitLab.

2.5. Planificación Temporal

Dada la naturaleza del proyecto, no se pueden determinar una serie de tareas para establecer una planificación temporal a priori. En cambio, se realizará una planificación para cada una de las tareas que el cliente asigne tratándolas como trabajos separados e independientes.

2.6. Evaluación Económica

De nuevo, ya que no se puede estimar el tiempo que ocupará el trabajo, no se puede realizar una evaluación económica completa antes de saber los requerimientos del cliente. Sin embargo, sí que es posible establecer las bases necesarias, que serán independientes de la duración de las tareas.

El coste del trabajo se divide en dos categorías: costes directos y costes indirectos. El coste total del proyecto dependerá del tiempo empleado en su desarrollo, pero los costes en función del tiempo pueden definirse a priori, de forma que más adelante será suficiente con multiplicar los costes/hora por las horas estimadas para cada tarea.

- **Costes directos**

  Los costes directos son aquellos asociados directamente con el produc-

to. Estos representan el sueldo del equipo encargado de desarrollar el proyecto. En este caso, dicho equipo es solamente una persona, por lo que es únicamente su sueldo el que hay que gestionar.

De acuerdo a la actualización más reciente del sitio web Indeed[10], el sueldo mensual medio actual de un programador junior ronda los 1,162€. Esto resulta en 4,82€por hora, por lo que ese será el salario que se le aplique al desarrollador para este proyecto. En condiciones ideales, se estiman entre 2 y 3 horas de trabajo diario dedicadas al proyecto. Así, el sueldo diario rondaría los 12,05€. En lugar de este valor, sin embargo, se va a utilizar el sueldo por hora para calcular el coste de las tareas a realizar por ser más preciso.

- **Costes indirectos**

  Los costes indirectos son aquellos que no se pueden acusar directa o exclusivamente al producto. Gastos como la electricidad, la amortización de equipo, etc. Como no se puede determinar el coste exacto para cada uno de estos ámbitos, se realizarán estimaciones en su lugar:

  - **Amortiguación del equipo informático**

    El principal elemento utilizado es el ordenador (550€). Suponiendo una vida útil de 5 años (60 meses) podemos calcular su amortiguación de la siguiente manera[11]:

    $$30 \text{ días/mes} * 24 \text{ horas/día} = 720 \text{ horas/mes}$$

    $$60 \text{ meses} * 720 \text{ horas/mes} = 43200 \text{ horas}$$

    Amortiguación del ordenador por hora = 550 €/43200 horas = 0,0127 €/hora

    El otro elemento informático utilizado es el ratón (60€). Suponiendo vida útil de 4 años (48 meses), su amortización se calcula de la misma manera.

    $$48 \text{ meses} * 720 \text{ horas/mes} = 34560 \text{ horas}$$

    Amortiguación del ratón por hora = 60€/34560 horas = 0,00176 €/hora

[10] https://www.indeed.es/

[11] Por simplicidad, se han redondeado los meses a 30 días.

Estos dos valores resultan en una amortización total del equipo utilizado de 0.0145€/hora.

- **Licencias software**

  De todas las herramientas software empleadas, solo una de ellas requiere una licencia de pago: PhpStorm. Sin embargo, es posible obtener una licencia gratuita temporal gracias al ser estudiante. Por ello no se tendrá en cuenta el coste de ninguna licencia en los gastos indirectos.

Los costes, directos e indirectos, se resumen en la tabla 1. Estos valores se utilizarán al realizar la evaluación económica de cada una de las tareas encargadas por el cliente.

Nombre del coste	Coste por hora
Sueldo del desarrollador	4,82€
Amortización del equipo	0.0145€

Cuadro 1: Resumen de los costes (por hora de trabajo).

2.7. Gestión de Riesgos

Los riesgos se clasificarán en función de su probabilidad y magnitud estimados de acuerdo a las tablas 2 y 3 respectivamente.

Probabilidad del riesgo	**Rango de probabilidad**
Alta	75 %-99 %
Media-alta	50 %-74 %
Media-baja	25 %-49 %
Baja	0 %-24 %

Cuadro 2: Clasificación de riesgos según su probabilidad.

Magnitud del riesgo	Coste en horas
Muy alta	+9 horas
Alta	5-8 horas
Media	3-5 horas
Baja	0-2 horas

Cuadro 3: Clasificación de riesgos según su magnitud.

Problemas técnicos
Descripción: Cualquier tipo de problema relacionado con el equipo técnico: fallo del ordenador, problemas con el software, etc.
Plan de prevención: Tratar de no realizar ningún tipo de acción que pueda causar alguno de estos problemas.
Plan de contingencia: Depende, en función de la gravedad del problema. Desde intentar solucionarlo personalmente hasta buscar asistencia técnica externa.
Probabilidad: Media-baja
Magnitud: Media
Impacto: 4 horas

Problemas de tiempo
Descripción: Dado que el trabajo se está realizando al mismo tiempo que se cursa el último año de la carrera, es muy posible que surjan periodos de tiempo en los que sea necesario ralentizar o detener el desarrollo para priorizar las clases, sobre todo en periodos con alta densidad de exámenes o trabajos.
Plan de prevención: Tratar de evitar retrasos en el curso, ya que se traducirán en retrasos en el trabajo.
Plan de contingencia: Tratar de compensar por el tiempo perdido y recuperar el ritmo de trabajo anterior al retraso.
Probabilidad: Muy alta
Magnitud: Alta
Impacto: 8 horas

Problemas en el diseño
Descripción: El diseño realizado resulta poco apropiado para lo que se necesita. Se requiere un nuevo diseño y tal vez descartar parte del trabajo realizado.
Plan de prevención: Realizar el diseño cuidadosamente y no apresurarse a implementar hasta estar convencido de que el diseño es correcto.
Plan de contingencia: Volver a hacer el diseño aprendiendo de los errores cometidos. Tratar de salvar todo lo posible del trabajo ya realizado pero sin dejar que influencie el nuevo diseño.
Probabilidad: Media-baja
Magnitud: Alta
Impacto: 6 horas

Falta de inspiración
Descripción: Se tienen problemas a la hora de continuar trabajando debido a que no se sabe bien lo que hacer exactamente (en cuando a diseño o implementación).
Plan de prevención: Dedicar el tiempo suficiente al estudio previo a cada encargo para asegurar que se tienen los conocimientos necesarios.
Plan de contingencia: Volver a la fase de estudio, reconsiderar el diseño si es necesario o buscar ayuda por parte del tutor u otra figura con los conocimientos necesarios.
Probabilidad: Media-alta
Magnitud: Media
Impacto: 4 horas

3. Antecedentes

Hay distintas manera de evaluar Ikastenbot a la hora de buscar aplicaciones similares. Por un lado se puede prestar atención a la funcionalidad que ofrece. Existen numerosas aplicaciones que ofrecen servicios similares a los de Ikastenbot: correctores de texto, detectores de plagios, calendarios y recordadores de eventos, etc. Sin embargo, la idea de combinar todos esos servicios y darles un enfoque a los trabajos de fin de grado parece una idea bastante novedosa, ya que no se ha encontrado ninguna aplicación similar.

Por otro lado, Ikastenbot no deja de ser un bot de Telegram, por lo que en ese aspecto hay una infinidad de aplicaciones similares a las que se puede acudir en busca de inspiración. Se puede mirar, sin ir más lejos, a BotFather, el bot oficial de Telegram para automatizar y facilitar la creación y gestión de otros bots.

Y para este trabajo en concreto, el propio Ikastenbot cuenta como aplicación de referencia. Ya que se va a trabajar para añadir nuevas funcionalidades al bot, lo mejor es observar como se han hecho las cosas hasta ahora para ajustarse todo lo posible a ello y mantener la cohesión.

4. Tarea - Alertas Sobre el Gantt

4.1. Planteamiento de la Tarea

4.1.1. Objetivos

A la hora de planificar proyectos, incluir hitos para denotar puntos clave a lo largo del desarrollo y tener en cuenta reuniones de seguimiento se consideran buenas prácticas[7]. Sin embargo, prestar atención a estos aspectos es uno de los errores más comunes que los alumnos tienden a cometer en sus Trabajos de Fin de Grado.

Por ello se desea que Ikastenbot analice los Gantt recibidos y notifique al alumno de estos errores en caso de que no se encuentren hitos o reuniones planificadas.

4.1.2. Alcance

En las figuras 12 y 13 se muestra el esquema de descomposición de tareas ideado para este trabajo. A continuación se enumeran en más detalle cada una de las tareas que lo componen.

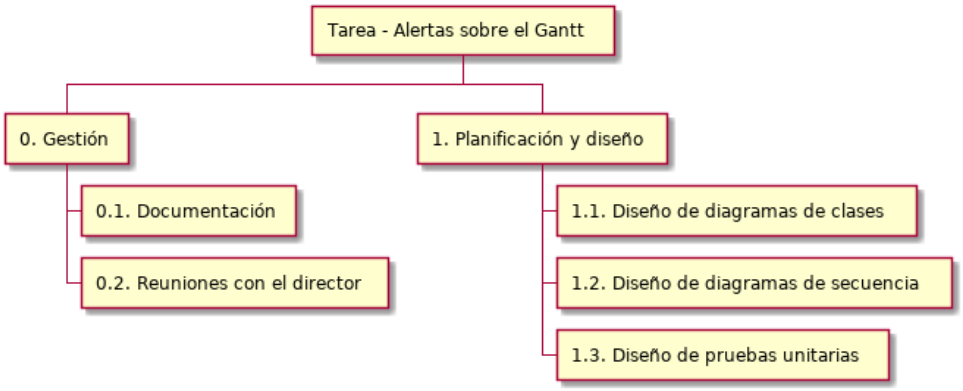

Figura 12: Tarea 1 - EDT (1).

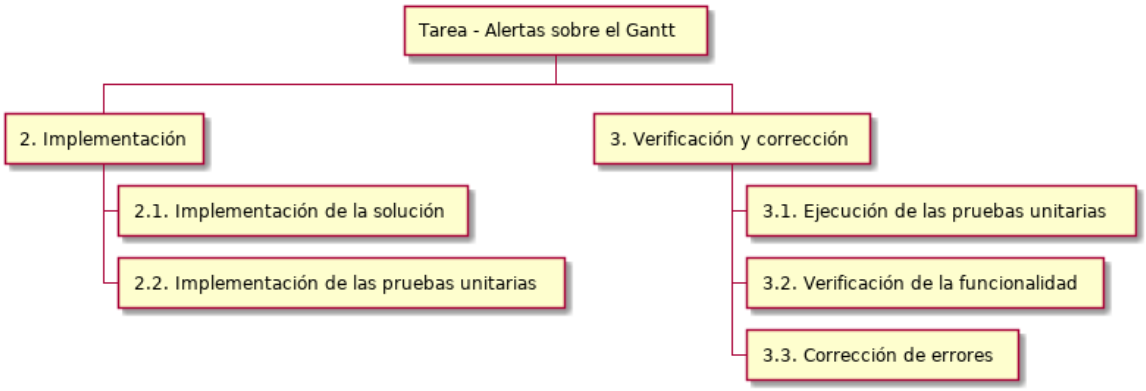

Figura 13: Tarea 1 - EDT (2).

0.1. Documentación
Paquete de trabajo: Gestión
Duración: 7 horas
Descripción: Documentación de toda la planificación y trabajo realizados en relación a esta tarea del proyecto.
Entradas: Ninguna.
Salidas: Documentación del trabajo.
Recursos necesarios: Ordenador
Precedencias: -

0.2. Reuniones con el director
Paquete de trabajo: Gestión
Duración: 3 horas
Descripción: Reuniones regulares con el director del proyecto para evaluar el progreso y/o resolver dudas.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: -
Precedencias: -

1.1. Diseño de diagramas de clases
Paquete de trabajo: Planificación y diseño
Duración: 1 hora
Descripción: Diseñar los diagramas de clases de la funcionalidad a implementar.
Entradas: Ninguna.
Salidas: Diagramas de clases.
Recursos necesarios: Papel y lápiz.
Precedencias: -

1.2. Diseño de diagramas de secuencia
Paquete de trabajo: Planificación y diseño
Duración: 1 hora
Descripción: Diseñar los diagramas de secuencia de las principales funciones de las clases diseñadas.
Entradas: Diagramas de clases.
Salidas: Diagramas de secuencia.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 1.1.

1.3. Diseño de pruebas unitarias
Paquete de trabajo: Planificación y diseño
Duración: 1 horas
Descripción: Realizar el diseño teórico de los test para validar el funcionamiento correcto del código.
Entradas: Diagramas de clases.
Salidas: Diseño de las pruebas unitarias.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 1.1.

2.1. Implementación de la solución
Paquete de trabajo: Implementación
Duración: 5 horas
Descripción: Implementar la solución diseñada.
Entradas: Diagramas de clases y de secuencia.
Salidas: Código fuente de la solución.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 1.2.

3.2. Implementación de las pruebas unitarias
Paquete de trabajo: Implementación
Duración: 2 horas
Descripción: Implementar las pruebas unitarias diseñadas.
Entradas: Diseño de las pruebas unitarias.
Salidas: Código fuente de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 1.3.

3.1. Ejecución de las pruebas unitarias
Paquete de trabajo: Verificación y corrección
Duración: 1 hora
Descripción: Ejecutar las pruebas unitarias sobre el código escrito.
Entradas: Código fuente de la solución y de las pruebas.
Salidas: Informe de errores de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 2.1 y 2.2.

3.2. Verificación de la funcionalidad
Paquete de trabajo: Verificación y corrección
Duración: 2 horas
Descripción: Pruebas manuales para verificar que la funcionalidad funciona correctamente en todos los casos.
Entradas: Código fuente de la solución.
Salidas: Informe de errores de las pruebas de funcionalidad.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 1.1.

3.3. Corrección de errores
Paquete de trabajo: Verificación y corrección
Duración: 3 horas
Descripción: Corrección de todos los errores, fallos y descuidos que se hayan encontrado mediante las pruebas y verificación.
Entradas: Código fuente de la solución e informes de errores obtenidos.
Salidas: Código fuente libre de errores.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 3.1 y 3.2.

4.1.3. Planificación Temporal

En la tabla 4 se resumen las tareas definidas junto con su duración. A partir de estas tareas se planeó el tiempo de trabajo tal y como se muestra en la figura 14.

Tarea	Duración (h)
0. Gestión	10
0.1. Documentación	7
0.2. Reuniones con el director	3
1. Planificación y diseño	3
1.1. Diseño de diagramas de clases	1
1.2. Diseño de diagramas de secuencia	1
1.3. Diseño de pruebas unitarias	1
2. Implementación	7
2.1. Implementación de la solución	5
2.2. Implementación de las pruebas unitarias	2
3. Verificación y corrección	6
3.1. Ejecución de las pruebas unitarias	1
3.2. Verificación de la funcionalidad	2
3.3. Corrección de errores	3
Total	**26**

Cuadro 4: Tarea 1 - Tareas planificadas.

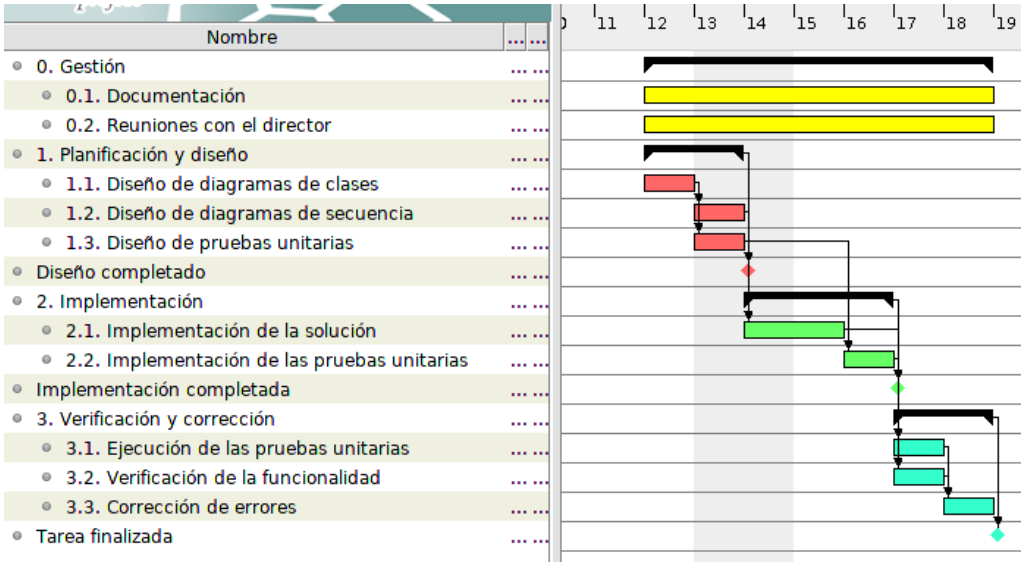

Figura 14: Tarea 1 - Diagrama Gantt.

4.1.4. Evaluación Económica

Sabiendo la duración estimada de la tarea y utilizando los datos calculados en el apartado 2.6 se puede calcular el coste de esta tarea.

$$(4,82\ €/\text{hora} + 0{,}0145\ €/\text{hora}) * 26\ \text{horas} = 125,69\ €$$

4.2. Análisis y Diseño

A simple vista, esta tarea no conlleva una gran dificultad. Aún así, se realizaron los pasos necesarios para el diseño, ideando una clase para contener la funcionalidad necesaria. Esta clase cuenta con dos pares de métodos simétricos. En cada par, una función pública se encarga de notificar al usuario de un posible fallo (falta de hitos, falta reuniones) en el Gantt haciendo uso de la otra función, privada, para determinar si, efectivamente, el fallo existe o no. Se decidió mantener las notificaciones de falta de hitos y de falta

de reuniones en funciones separadas para permitir elegir más fácilmente a cual llamar.

La clase diseñada se muestra en la figura 15 y mientras que las figuras 39 y 40 representan los diagramas de secuencia para sus funciones principales.

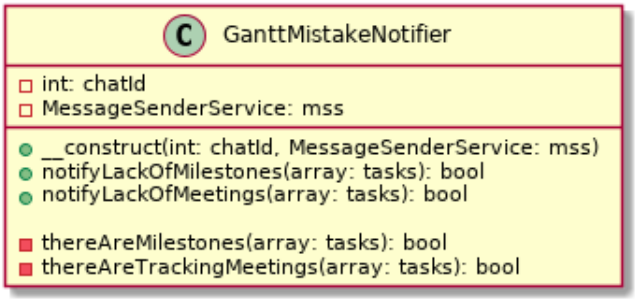

Figura 15: Tarea 1 - Diagrama de clases.

4.3. Desarrollo

4.3.1. Desarrollo Original

Esta tarea fue la primera en ser implementada, y en aquel momento aún no se tenía completamente claras las convenciones y prácticas requeridas para el proyecto de Ikastenbot. La funcionalidad se implementó sin una planificación o diseño previos, de una manera pobre e invasiva al resto del código. Consistía en dos funciones, una para detectar falta de hitos en una lista de tareas, y otra equivalente para falta de reuniones, localizadas en el script del bot que recibe los diagramas Gantt. En el mismo script, se llamaría a esas funciones y, dependiendo del resultado, se respondería al usuario con los mensajes apropiados.

4.3.2. Refactorización

La solución implementada originalmente, aunque funcional, dejaba mucho que desear en cuanto a limpieza del código y buenas prácticas. Y por si fuera

poco, la forma en la que estaba implementada hacía imposible añadir pruebas automáticas.

Por todas estas razones, tras haber terminado varias otras tareas encargadas por el cliente, se decidió reconsiderar ésta. Se eliminó aquello que se había implementado y se empezó de nuevo, esta vez comenzado por la planificación y el diseño explicados anteriormente. La funcionalidad se extrajo a una clase aparte, lo cual ayuda a la modularidad del código y facilita las pruebas.

El segundo desarrollo de esta tarea se lleva a cabo sin ningún suceso especialmente notable. Se implementa la clase requerida siguiendo el diseño previo y se introduce la funcionalidad al comando del bot que recibe los diagramas Gantt sin dificultades ni problemas.

4.4. Verificación y Evaluación

Las pruebas que se diseñaron para esta tarea se dividen en dos frentes: pruebas unitarias para validar el funcionamiento del código, y pruebas manuales para verificar el comportamiento del bot.

4.4.1. Pruebas Unitarias

Tras implementar funcionalidad, se diseñaron una serie de pruebas unitarias para verificar su correcto funcionamiento y para automatizar futuras pruebas en caso de que el código cambiase.

testMilestones
Descripción: Se crea una lista de tareas sin hitos y se comprueba si la función detecta el fallo. Se repite el proceso con una lista de tareas que si contiene hitos.
Resultado esperado: Se detecta la falta de hitos en la primera lista. No se detecta ningún problema con la segunda.

testMeetings
Descripción: Se crea una lista de tareas sin reuniones y se comprueba si la función detecta el fallo. Se repite el proceso con una lista de tareas que si contiene reuniones.
Resultado esperado: Se detecta la falta de reuniones en la primera lista. No se detecta ningún problema con la segunda.

4.4.2. Pruebas de Funcionamiento

Una vez terminada la implementación se llevaron a cabo una sere de pruebas manuales para comprobar que el comportamiento de Ikastenbot es el correcto.

Diagrama Gantt sin hitos ni reuniones
Descripción: Se envía a Ikastenbot un diagrama Gantt que no contiene hitos ni reuniones.
Resultado esperado: Ikastenbot responde con dos mensajes: uno notificando de la falta de hitos y otro notificando de la falta de reuniones.

Diagrama Gantt sin hitos
Descripción: Se envía a Ikastenbot un diagrama Gantt que no contiene hitos.
Resultado esperado: Ikastenbot responde con un mensaje uno notificando de la falta de hitos.

Diagrama Gantt sin reuniones
Descripción: Se envía a Ikastenbot un diagrama Gantt que no contiene reuniones.
Resultado esperado: Ikastenbot responde con un mensaje uno notificando de la falta de reuniones.

Diagrama Gantt con hitos y reuniones
Descripción: Se envía a Ikastenbot un diagrama Gantt que contiene hitos y reuniones.
Resultado esperado: Ikastenbot no devuelve ningún mensaje relacionado con esta funcionalidad.

4.5. Conclusiones

4.5.1. Cumplimiento de Objetivos

La primera implementación, a pesar de que fue aceptada en su momento, puede llegar a considerarse como una solución, pero distaba mucho de ser satisfactoria. Tras el rediseño y la reimplementación se ha llegado a un resultado mucho más aceptable el cuál sí alcanza los objetivos definidos.

4.5.2. Revisión de la Planificación

En la tabla 5 se muestra el tiempo empleado realmente en la tarea en comparación a lo originalmente planeado. Dado que antes de hacer la planificación ya se había hecho una implementación previa que más tarde se descartó, ya se tenía una idea del alcance de la tarea, por lo que la planificación resultó ser bastante acertada.

Tarea	Duración Estimada (h)	Duración Real (h)
0. Gestión	10	9
0.1. Documentación	7	7
0.2. Reuniones con el director	3	2
1. Planificación y diseño	3	3
1.1. Diseño de diagramas de clases	1	1
1.2. Diseño de diagramas de secuencia	1	1
1.3. Diseño de pruebas unitarias	1	1
2. Implementación	7	6
2.1. Implementación de la solución	5	4
2.2. Implementación de las pruebas unitarias	2	2
3. Verificación y corrección	6	4
3.1. Ejecución de las pruebas unitarias	1	1
3.2. Verificación de la funcionalidad	2	2
3.3. Corrección de errores	3	1
Total	26	22

Cuadro 5: Tarea 1 - Duración estimada vs. duración real.

5. Tarea - Arreglo de las Alertas del Bot

5.1. Planteamiento de la Tarea

5.1.1. Objetivos

Cuando Ikastenbot recibe un diagrama Gantt de un usuario, almacena las tareas e hitos para enviar notificaciones cuando se acerquen sus fechas. Cada vez que el usuario envían un nuevo Gantt, se guarda como una nueva versión. Las tareas e hitos de las versiones anteriores no se eliminan de la base de datos, lo que causa que aquellos que estuviesen marcados para ser notificados, sigan causando mensajes de aviso cuando llegue su fecha, a pesar de que deberían haber sido reemplazados por aquellos de la nueva versión.

El objetivo de esta tarea es encontrar e implementar una solución para que estas tareas e hitos que resultan sobrescritos por nuevas versiones de sus diagramas Gantt no causen notificaciones bajo ningún contexto.

5.1.2. Alcance

En las figuras 16 y 17 se muestra el esquema de descomposición de tareas ideado para este trabajo. A continuación se enumeran en más detalle cada una de las tareas que lo componen.

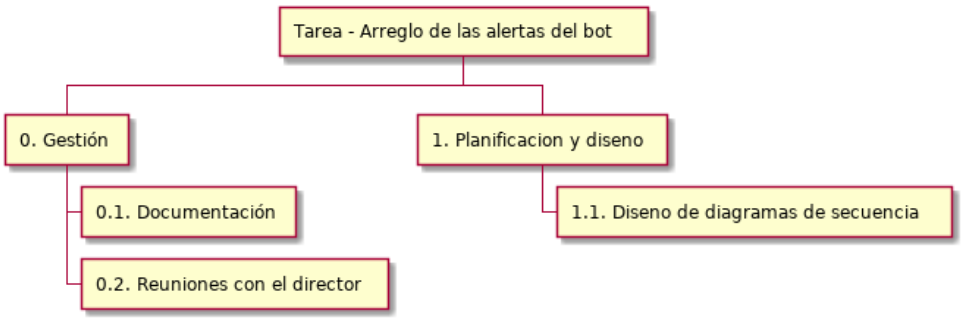

Figura 16: Tarea 2 - EDT (1).

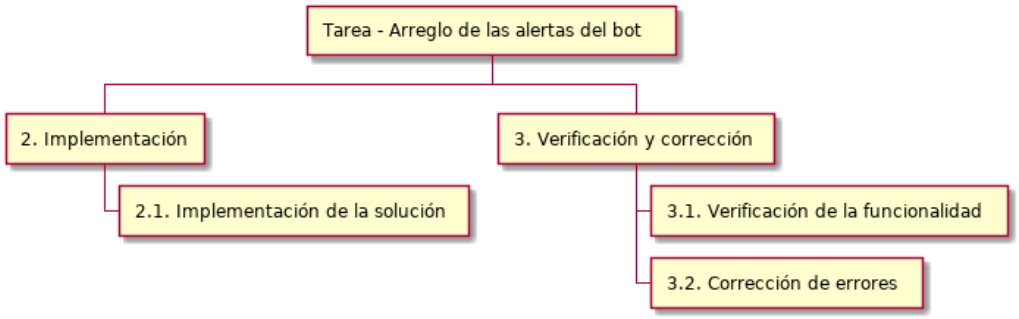

Figura 17: Tarea 2 - EDT (2).

0.1. Documentación
Paquete de trabajo: Gestión
Duración: 5 horas
Descripción: Documentación de toda la planificación y trabajo realizados en relación a esta tarea del proyecto.
Entradas: Ninguna.
Salidas: Documentación del trabajo.
Recursos necesarios: Ordenador
Precedencias: -

0.2. Reuniones con el director
Paquete de trabajo: Gestión
Duración: 2 horas
Descripción: Reuniones regulares con el director del proyecto para evaluar el progreso y/o resolver dudas.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: -
Precedencias: -

1.1. Diseño de diagramas de secuencia
Paquete de trabajo: Planificación y diseño
Duración: 2 horas
Descripción: Diseñar los diagramas de secuencia de las funcionalidad requerida.
Entradas: Ninguna.
Salidas: Diagramas de secuencia.
Recursos necesarios: Papel y lápiz.
Precedencias: -

2.1. Implementación de la solución
Paquete de trabajo: Implementación
Duración: 5 horas
Descripción: Implementar la solución diseñada.
Entradas: Diagramas de secuencia.
Salidas: Código fuente de la solución.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 1.1.

3.1. Verificación de la funcionalidad
Paquete de trabajo: Verificación y corrección
Duración: 2 horas
Descripción: Pruebas manuales para verificar que la funcionalidad funciona correctamente en todos los casos.
Entradas: Código fuente de la solución.
Salidas: Informe de errores de las pruebas de funcionalidad.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.1.

3.2. Corrección de errores
Paquete de trabajo: Verificación y corrección
Duración: 1 hora
Descripción: Corrección de todos los errores, fallos y descuidos que se hayan encontrado mediante las pruebas y verificación.
Entradas: Código fuente de la solución e informes de errores obtenidos.
Salidas: Código fuente libre de errores.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 3.1.

5.1.3. Planificación Temporal

En la tabla 6 se resumen las tareas definidas junto con su duración. A partir de estas tareas se planeó el tiempo de trabajo tal y como se muestra en la figura 18.

Tarea	Duración (h)
0. Gestión	7
0.1. Documentación	5
0.2. Reuniones con el director	2
1. Planificación y diseño	2
1.1. Diseño de diagramas de secuencia	2
2. Implementación	5
2.1. Implementación de la solución	5
3. Verificación y corrección	3
3.1. Verificación de la funcionalidad	2
3.2. Corrección de errores	1
Total	**17**

Cuadro 6: Tarea 2 - Tareas planificadas.

5.1.4. Planificación Económica

Sabiendo la duración estimada de la tarea y utilizando los datos calculados en el apartado 2.6 se puede calcular el coste de esta tarea.

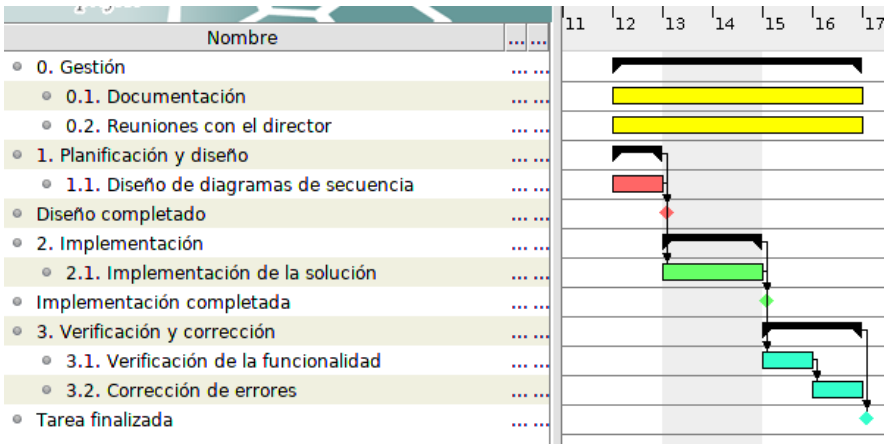

Figura 18: Tarea 2 - Diagrama Gantt.

$$(4,82\ €/\text{hora} + 0{,}0145\ €/\text{hora}) * 17\ \text{horas} = 82,18\ €$$

5.2. Análisis y Diseño

El problema de las notificaciones sobre tareas obsoletas se debe a que en la base de datos, las tareas cuentan con un atributo booleano `notify` que indica si el usuario desea recibir notificaciones cuando la fecha de la tarea se acerque (Fig 19). Puesto que al añadir una nueva versión del diagrama Gantt estas tareas antiguas no se eliminan, el comando que recorre la base de datos en busca de tareas para notificar las detecta igual que si fueran tareas de la última versión.

Una posible solución es eliminar todas las tareas pertenecientes a las versiones anteriores del Gantt cada vez que se recibe una nueva versión del diagrama. Sin embargo, mantener las distintas versiones de los Gantt en la base de datos puede resultar beneficioso, para observar la evolución de los diagramas de los alumnos y las correcciones y cambios que realizan, para acumular más datos para futuras estadísticas, etc.

Por ello se ha optado por la otra solución principal: modificar las entradas

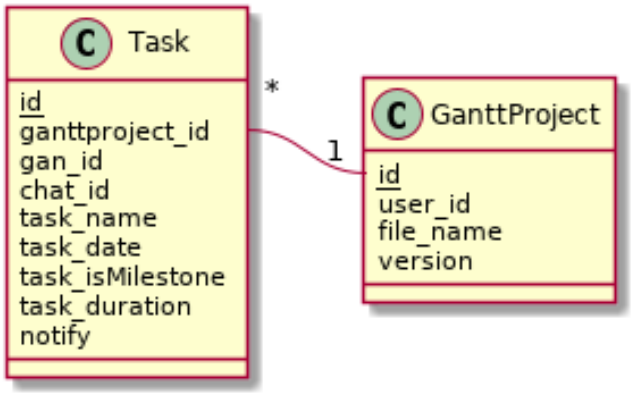

Figura 19: Tarea 2 - Esquema relacional de la BD (parcial).

de las tareas de versiones anteriores para desactivar su atributo `notify`. De esta forma, estas tareas no serán tenidas en cuenta a la hora de mandar notificaciones, y se pueden mantener en la base de datos para futuro uso. Para esto habría que modificar el comando que recibe los Gantt del alumno añadiendo la funcionalidad que busca las tareas de versiones anteriores y desactiva sus notificaciones.

El bot hace uso de la librería Doctrine[12] para facilitar el acceso a la base de datos y el uso de los datos allí almacenados. Doctrine es un Object Relational Mapper (ORM), una herramienta que permite trabajar con los contenidos de una base de datos en un estilo orientado a objetos. Para ello, se definen clases representando las distintas tablas y sus relaciones y Doctrine es capaz de cargar los datos en instancias de la clase correspondiente y transmitir los cambios que se operen sobre las instancias a la base de datos.

Mediante la clase de Doctrine `EntityManager`, se puede utilizar un comando SQL para recoger unos datos concretos de la base de datos y cargarlos en instancias de las clases definidas para las tablas correspondientes. A continuación, habría que modificar los valores deseados de dichas instancias y utilizar `EntityManager` para persistir los cambios a la base de datos. Este diseño se representa en el diagrama de secuencia de la figura 41.

[12]https://www.doctrine-project.org/

5.3. Desarrollo

Gracias a la librería Doctrine, la solución resulta relativamente fácil de implementar. Es suficiente con utilizar Doctrine para cargar las tareas de versiones anteriores del Gantt y modificar el atributo `notify` de todas ellas para evitar que sean notificadas en el futuro.

5.4. Verificación y Evaluación

Como esta tarea no ha requerido implementar ninguna estructura compleja no requiere pruebas unitarias para automatizar la verificación. Sin embargo, se han realizado pruebas manuales para comprobar que el comando funciona correctamente. Estas pruebas han consistido en mandar distintos diagramas Gantt a Ikastenbot, comprobando cada vez en la base de datos que las tareas de versiones anteriores del Gantt se modifican en la base de datos y su campo `notify` se desactiva.

5.5. Conclusiones

5.5.1. Cumplimiento de Objetivos

Esta fue una de las primeras tareas en las que se trabajó. Por ello, de manera similar a lo sucedido con la tarea expuesta en el apartado 4, aún no se estaba familiarizado completamente con las prácticas a seguir en el proyecto. En retrospectiva, el diseño realizado y el producto desarrollado dejan mucho que desear a pesar de resultar en la funcionalidad que se tenía como objetivo. Por ejemplo, y entre otras cosas, el código estaba incrustado en mitad de un script, lo que dificultaba hacer pruebas sobre él.

Poco después de entregar el resultado, el dueño del proyecto reutilizó el código para diseñar una clase separada con las misma funcionalidad, excepto que de una forma mucho más limpia y con tests unitarios asociados. Esta refactorización sirvió como inspiración, no solo para realizar un mejor trabajo en el futuro, sino también para revisar el trabajo realizado hasta ahora. Esta

es la razón de que se planeara, diseñara y desarrollara de nuevo la tarea anterior, tal y como se explica en el apartado 4.3.

Por todo ello, no se considera que en este encargo se hayan alcanzado los objetivos deseados.

5.5.2. Revisión de la Planificación

En la tabla 7 se muestra el uso del tiempo planeado para la tarea frente a lo que realmente se necesitó.

Tarea	Duración Estimada (h)	Duración Real (h)
0. Gestión	7	8
0.1. Documentación	5	6
0.2. Reuniones con el director	2	2
1. Planificación y diseño	2	2
1.1. Diseño de diagramas de secuencia	2	2
2. Implementación	5	2
2.1. Implementación de la solución	5	2
3. Verificación y corrección	3	3
3.1. Verificación de la funcionalidad	2	2
3.2. Corrección de errores	1	1
Total	17	15

Cuadro 7: Tarea 2 - Duración estimada vs. duración real.

6. Tarea - Arreglo del Corrector Ortográfico

6.1. Planteamiento de la Tarea

6.1.1. Objetivos

Una de las funciones de Ikastenbot es, tras haber enviado una versión de la memoria, pedir una corrección ortográfica y gramatical del texto. Esta corrección se lleva a cabo en el idioma que el usuario haya seleccionado previamente para su memoria. Esto causa que, al corregir el texto en un idioma distinto del inglés, todos los términos ingleses se marcan como errores al ser palabras no reconocidas. Dada la naturaleza de los trabajos de estudiantes de carreras técnicas, es muy común el uso de estos términos en inglés, por lo que no deberían considerarse errores incluso en textos escritos en idiomas distintos.

El objetivo de esta tarea es encontrar e implementar una solución de forma que las correcciones de los textos no detecten palabras y términos ingleses como errores independientemente del idioma del texto.

6.1.2. Alcance

En las figuras 20 y 21 se muestra el esquema de descomposición de tareas ideado para este trabajo. A continuación se enumeran en más detalle cada una de las tareas que lo componen.

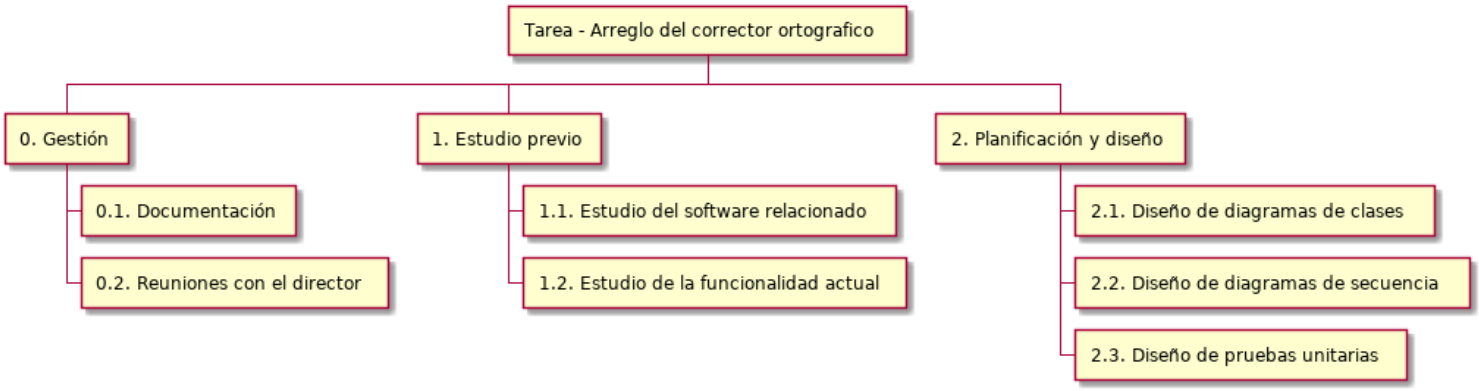

Figura 20: Tarea 3 - EDT (1).

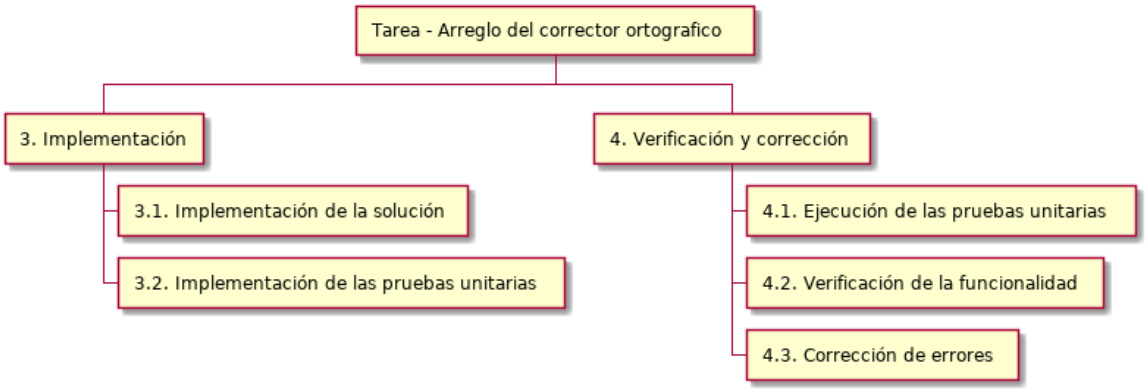

Figura 21: Tarea 3 - EDT (2).

0.1. Documentación
Paquete de trabajo: Gestión
Duración: 7 horas
Descripción: Documentación de toda la planificación y trabajo realizados en relación a esta tarea del proyecto.
Entradas: Ninguna.
Salidas: Documentación del trabajo.
Recursos necesarios: Ordenador
Precedencias: -

0.2. Reuniones con el director
Paquete de trabajo: Gestión
Duración: 3 horas
Descripción: Reuniones regulares con el director del proyecto para evaluar el progreso y/o resolver dudas.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: -
Precedencias: -

1.1. Estudio del software relacionado
Paquete de trabajo: Estudio previo
Duración: 2 horas
Descripción: Estudiar el programa utilizado para generar los diagramas Gantt que Ikastenbot puede leer, GanttProject, para aprender como funciona, y que información contienen los archivos que genera.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el programa GanttProject.
Precedencias: -

1.2. Estudio de la funcionalidad actual
Paquete de trabajo: Estudio previo
Duración: 3 horas
Descripción: Estudiar el código que recibe, analiza y almacena en contenido del Gantt para saber como y donde integrar la nueva funcionalidad.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el código fuente de Ikastenbot.
Precedencias: -

2.1. Diseño de diagramas de clases
Paquete de trabajo: Planificación y diseño
Duración: 2 hora
Descripción: Diseñar los diagramas de clases de la funcionalidad a implementar.
Entradas: Ninguna.
Salidas: Diagramas de clases.
Recursos necesarios: Papel y lápiz.
Precedencias: Tareas 1.1 y 1.2.

2.2. Diseño de diagramas de secuencia
Paquete de trabajo: Planificación y diseño
Duración: 2 hora
Descripción: Diseñar los diagramas de secuencia de las principales funciones de las clases diseñadas.
Entradas: Diagramas de clases.
Salidas: Diagramas de secuencia.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

2.3. Diseño de pruebas unitarias
Paquete de trabajo: Planificación y diseño
Duración: 1 horas
Descripción: Realizar el diseño teórico de los test para validar el funcionamiento correcto del código.
Entradas: Diagramas de clases.
Salidas: Diseño de las pruebas unitarias.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

3.1. Implementación de la solución
Paquete de trabajo: Implementación
Duración: 9 horas
Descripción: Implementar la solución diseñada.
Entradas: Diagramas de clases y de secuencia.
Salidas: Código fuente de la solución.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.2.

3.2. Implementación de las pruebas unitarias
Paquete de trabajo: Implementación
Duración: 3 horas
Descripción: Implementar las pruebas unitarias diseñadas.
Entradas: Diseño de las pruebas unitarias.
Salidas: Código fuente de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.3.

4.1. Ejecución de las pruebas unitarias
Paquete de trabajo: Verificación y corrección
Duración: 1 hora
Descripción: Ejecutar las pruebas unitarias sobre el código escrito.
Entradas: Código fuente de la solución y de las pruebas.
Salidas: Informe de errores de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 3.1 y 3.2.

4.2. Verificación de la funcionalidad
Paquete de trabajo: Verificación y corrección
Duración: 2 horas
Descripción: Pruebas manuales para verificar que la funcionalidad funciona correctamente en todos los casos.
Entradas: Código fuente de la solución.
Salidas: Informe de errores de las pruebas de funcionalidad.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 3.1.

4.3. Corrección de errores
Paquete de trabajo: Verificación y corrección
Duración: 3 horas
Descripción: Corrección de todos los errores, fallos y descuidos que se hayan encontrado mediante las pruebas y verificación.
Entradas: Código fuente de la solución e informes de errores obtenidos.
Salidas: Código fuente libre de errores.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 4.1 y 4.2.

6.1.3. Planificación Temporal

En la tabla 12 se resumen las tareas definidas junto con su duración. A partir de estas tareas se planeó el tiempo de trabajo tal y como se muestra en la figura 36.

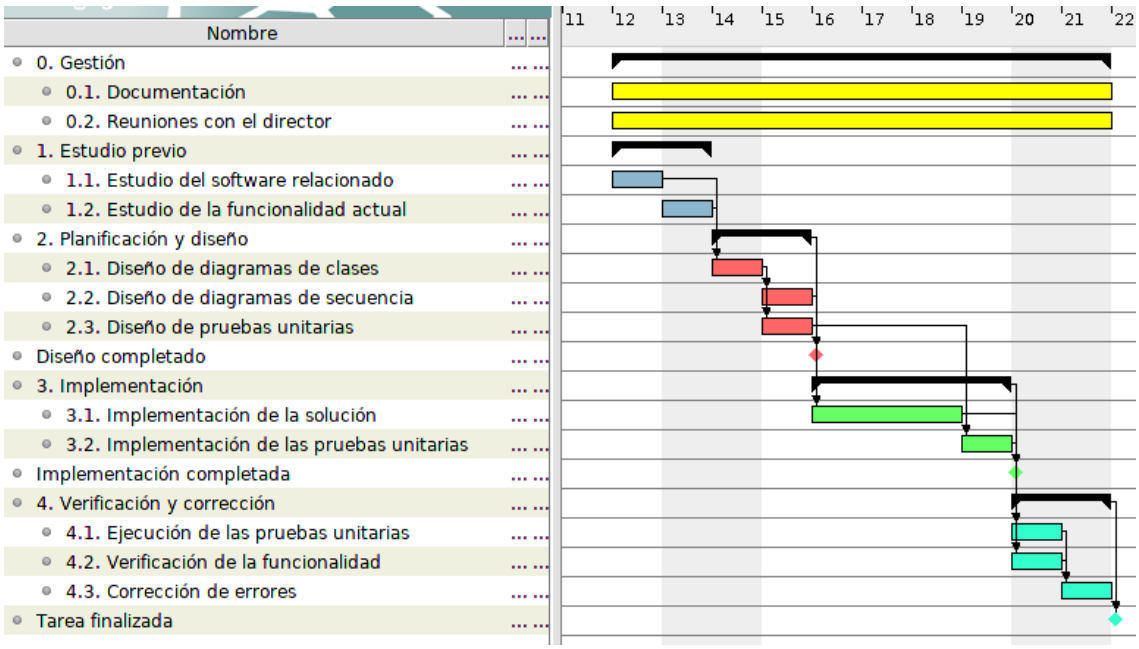

Figura 22: Tarea 3 - Diagrama Gantt.

Tarea	Duración (h)
0. Gestión	10
0.1. Documentación	7
0.2. Reuniones con el director	3
1. Estudio previo	5
1.1. Estudio del software relacionado	2
1.2. Estudio de la funcionalidad actual	3
2. Planificación y diseño	5
2.1. Diseño de diagramas de clases	2
2.2. Diseño de diagramas de secuencia	2
2.3. Diseño de pruebas unitarias	1
3. Implementación	12
3.1. Implementación de la solución	9
3.2. Implementación de las pruebas unitarias	3
4. Verificación y corrección	6
4.1. Ejecución de las pruebas unitarias	1
4.2. Verificación de la funcionalidad	2
4.3. Corrección de errores	3
Total	**38**

Cuadro 8: Tarea 3 - Tareas planificadas.

6.1.4. Planificación Económica

Sabiendo la duración estimada de la tarea y utilizando los datos calculados en el apartado 2.6 se puede calcular el coste de esta tarea.

$$(4,82\ €/\text{hora} + 0{,}0145\ €/\text{hora}) * 38\ \text{horas} = 183,71\ €$$

6.2. Análisis y Diseño

La herramienta utilizada para para la corrección ortográfica y gramatical de los textos es LanguageTool[13]. Su librería Java permite lanzar un servidor local al que enviarle textos para evaluar y cuyas respuestas contienen las correcciones de dichos textos. Esta es la estrategia utilizada por Ikastenbot.

Tras analizar más a fondo el API REST[14] del servidor se encontró el parámetro opcional `altLanguages` (Fig 23). Este parámetro permite declarar un número de idiomas secundarios de modo que, si alguna palabra desconocida en el idioma principal pertenece a alguno de los idiomas alternativos, el error es marcado como 'Hint'. De esta forma, es posible detectar y filtrar los errores causados por palabras en otros idiomas.

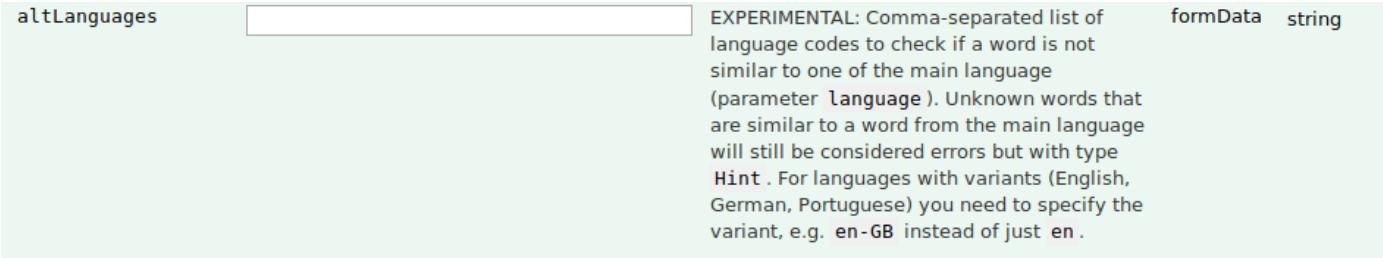

Figura 23: Parámetro altLanguages de la API REST de LanguageTool.

La API marca el parámetro `altLanguages` como experimental, por lo que se realizaron una serie de pruebas para comprobar que su funcionamiento es, por un lado fiable y consistente, y por otro, lo que necesitamos para solucionar el problema.

Tras estas comprobaciones se llegó a la conclusión de que el uso de este parámetro a la hora de realizar las peticiones al servidor es idóneo. Cuando se realiza la petición al servidor local para enviar el texto habrá que añadir el parámetro `altLanguages` con el valor 'en-GB, en-US', para diferenciar los términos ingleses del resto de errores ortográficos. Y a la hora de mostrar los errores recibidos al usuario, será suficiente con filtrar e ignorar aquellos que aparezcan marcados como 'Hint', lo que los identifica como palabras erróneas

[13]https://languagetool.org/
[14]https://languagetool.org/http-api/swagger-ui/#/default

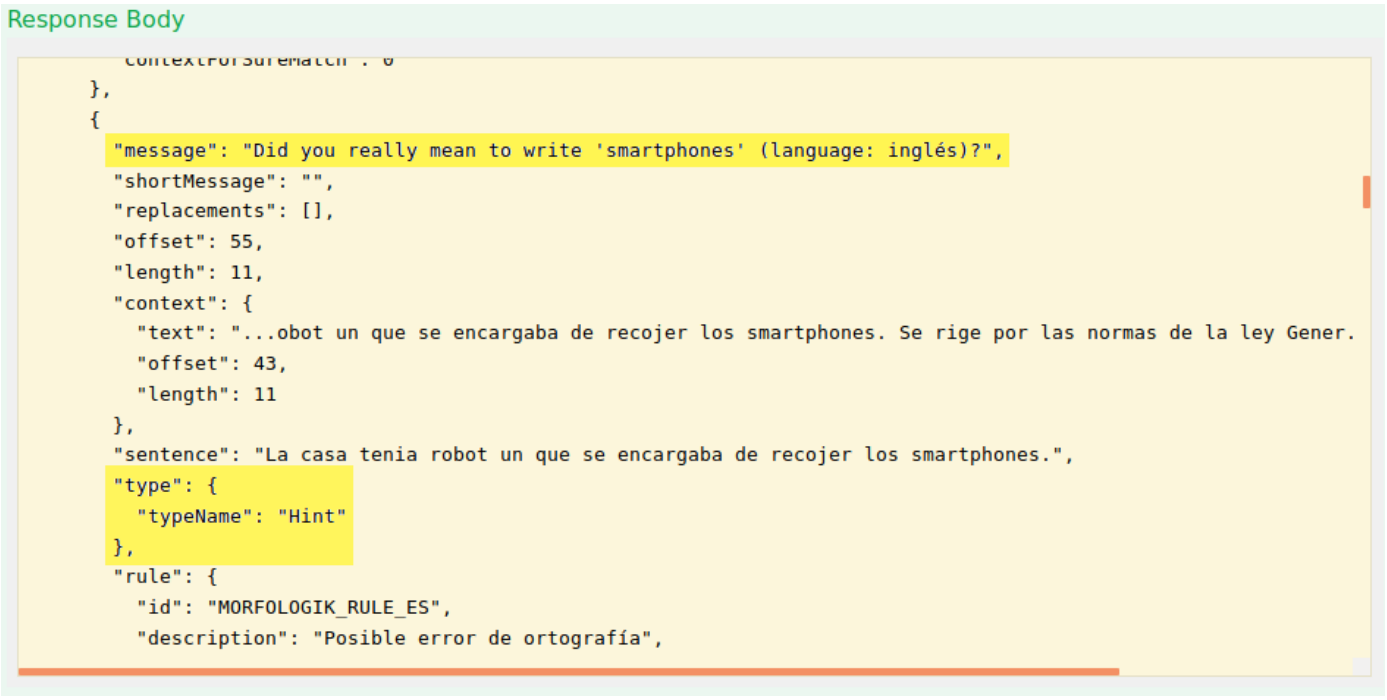

Figura 24: Ejemplo de la respuesta JSON.

en el idioma principal, pero no en los alternativos.

Este parámetro simplifica enormemente la tarea, y hace innecesario un diseño en detalle.

6.3. Desarrollo

Tal como se esperaba tras el análisis, el desarrollo resulta bastante simple. En la funcionalidad ya existente que corrige el texto enviado por el alumno, cuando se hace la llamada al servidor de LanguageTool se añade el nuevo campo `altLanguages` con valor 'en-GB, en-US'. Gracias a esto, la respuesta con los errores detectados marca las palabras en inglés con un valor 'Hint'. Estos campos se detectan y se descartan, de forma que nunca se muestran al usuario como errores ortográficos.

En las figuras 25 y 25 se muestra la diferencia que hace en las correcciones el parámetro `altLanguages` al enviar un documento con el siguiente texto: "La estrategia de polling (encuesta) consiste en preguntar directamente a Telegram qué mensajes se han recibido. Se envía una petición HTTP y Telegram responde con una lista de los mensages (updates) que el bot ha recibido y que aún no se han confirmado como detectadas por la aplicación."

Figura 25: Tarea 3 - Corrección del bot sin `altLanguages`.

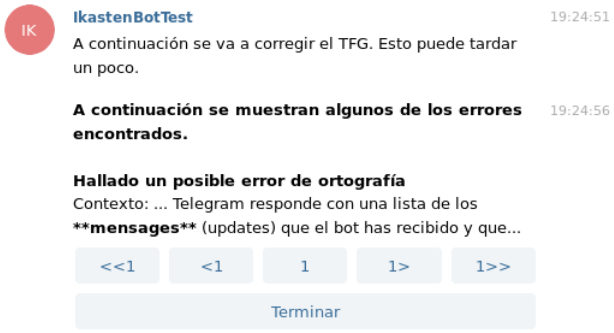

Figura 26: Tarea 3 - Corrección del bot con `altLanguages`.

6.4. Verificación y Evaluación

Para esta funcionalidad no se implementaron pruebas unitarias, ya que el código escrito no conforma un módulo aparte ni es lo bastante complejo como para requerirlas. Sin embargo, sí que se realizaron pruebas manuales, tanto antes como después del desarrollo.

Antes del desarrollo, durante el análisis y el estudio de LanguageTool, se llevaron a cabo varias pruebas con el objetivo de determinar si el parámetro `altLanguages` funcionaba correctamente y si se adaptaba a las necesidades. Estas pruebas consistieron en utilizar la API de prueba para corregir varios textos (desde frases simples hasta textos complejos) con palabras en inglés y verificando que todas estas palabras se detectan como errores y se marcan como 'Hint'. Las mismas pruebas se llevaron a cabo contra u servidor propio de LanguageTool para asegurar que no es solo la API lo que funciona correctamente. Los resultados fueron satisfactorios, por lo que se decidió utilizar esto como solución.

Después del desarrollo se realizaron pruebas similares, esta vez en Ikastenbot, mandando textos preparados y memorias de otros alumnos y verificando que ninguna de las palabras en inglés son detectadas como errores.

6.5. Conclusiones

6.5.1. Cumplimiento de Objetivos

A pesar de la simpleza de la solución, se cumplen los objetivos de ser eficaz y satisfactoria. El hecho de que no se haya necesitado un diseño o un desarrollo avanzados se debe únicamente a que la tarea encargada tenía un posible arreglo mucho más sencillo de lo que se esperaba.

6.5.2. Revisión de la Planificación

Por la misma razón, la planificación temporal no se parece en nada al tiempo necesitado realmente, tal como se muestra en la tabla 9. No han hecho falta diseño, desarrollo o pruebas unitarias. De hecho, los aspectos que más tiempo han necesitado ha sido el estudio necesario para encontrar la solución, y las comprobaciones de que la funcionalidad se comporta correctamente, tanto antes como después de implementarla en Ikastenbot.

Tarea	Duración Estimada (h)	Duración Real (h)
0. Gestión	10	9
0.1. Documentación	7	6
0.2. Reuniones con el director	3	3
1. Estudio previo	5	7
1.1. Estudio del software relacionado	2	5
1.2. Estudio de la funcionalidad actual	3	2
2. Planificación y diseño	5	0
2.1. Diseño de diagramas de clases	2	0
2.2. Diseño de diagramas de secuencia	2	0
2.3. Diseño de pruebas unitarias	1	0
3. Implementación	12	0
3.1. Implementación de la solución	9	1
3.2. Implementación de las pruebas unitarias	3	0
4. Verificación y corrección	6	4
4.1. Ejecución de las pruebas unitarias	1	0
4.2. Verificación de la funcionalidad	2	3
4.3. Corrección de errores	3	1
Total	38	20

Cuadro 9: Tarea 3 - Duración estimada vs. duración real.

7. Tarea - Opciones para Retrasar las Tareas

7.1. Planteamiento de la Tarea

7.1.1. Objetivos

Cuando Ikastenbot envía un recordatorio a un alumno avisándole de que una tarea o hito se acerca, ofrece la opción de retrasar dicho evento en caso de que, por razones de tiempo, el alumno lo considere necesario. El usuario puede especificar una cantidad de tiempo para retrasar la tarea.

La forma en la que el bot retrasa la tarea consiste en mantener su fecha de inicio e incrementar su duración en tantos días como el alumno especifique. En la figura 27 se muestra un ejemplo. Según la percepción del usuario, este comportamiento puede parecer poco intuitivo, ya que no se trata realmente de retrasar la tarea, sino de alargarla para acomodar el tiempo extra requerido.

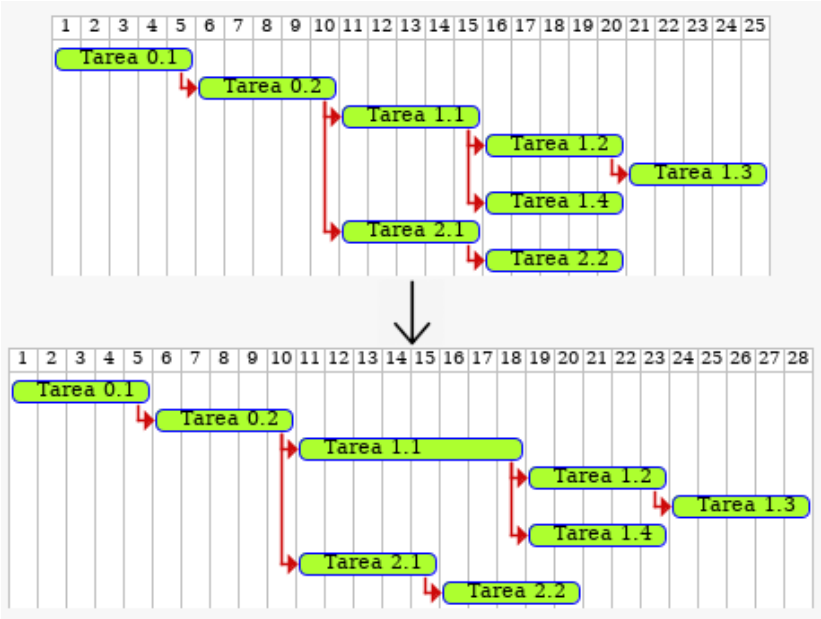

Figura 27: Tarea 4 - Resultado de retrasar la Tarea 1.1. 3 días utilizando el método actual.

Se desea implementar una nueva forma de retrasar las tareas, en la que su duración se mantenga constante mientras que su inicio temprano se pospone tantos días como especificados, tal como se muestra en la figura 28. Del mismo modo, Ikastenbot debe ofrecer una nueva opción de diálogo en la que permite al usuario elegir de qué forma desea retrasar la tarea, junto con una breve explicación de las alternativas.

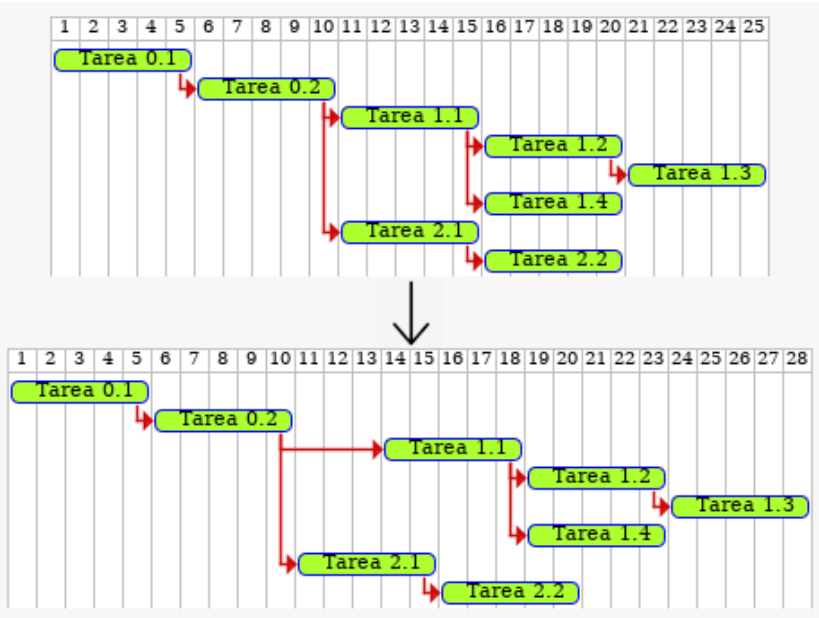

Figura 28: Tarea 4 - Resultado de retrasar la Tarea 1.1. 3 días utilizando el nuevo método.

7.1.2. Alcance

En las figuras 29 y 30 se muestra el esquema de descomposición de tareas ideado para este trabajo. A continuación se enumeran en más detalle cada una de las tareas que lo componen.

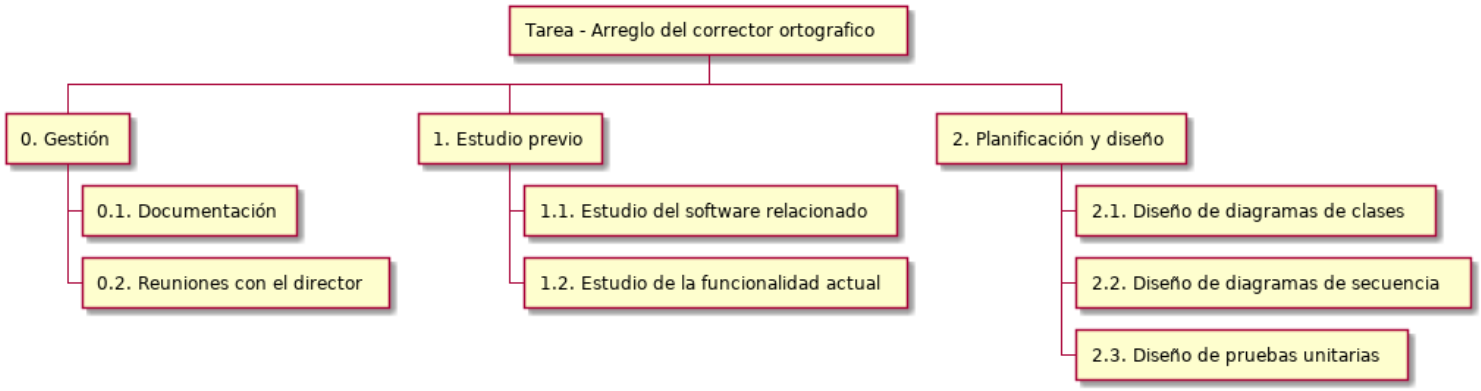

Figura 29: Tarea 4 - EDT (1).

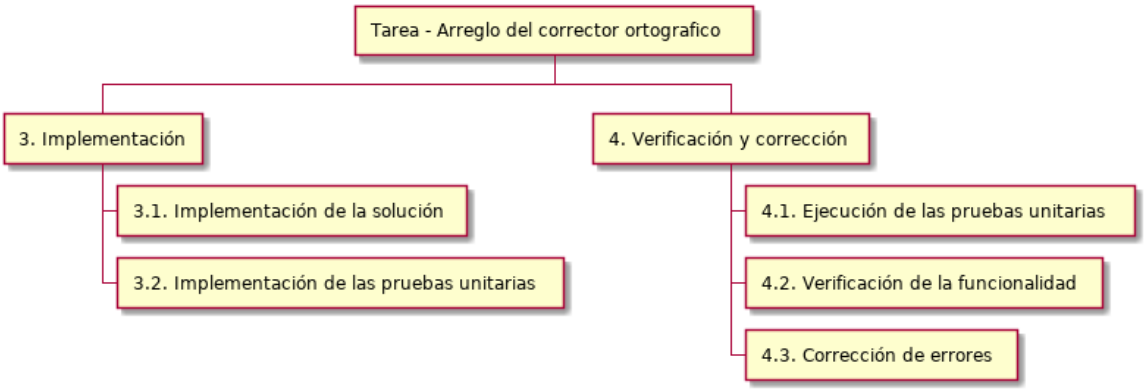

Figura 30: Tarea 4 - EDT (2).

0.1. Documentación
Paquete de trabajo: Gestión
Duración: 7 horas
Descripción: Documentación de toda la planificación y trabajo realizados en relación a esta tarea del proyecto.
Entradas: Ninguna.
Salidas: Documentación del trabajo.
Recursos necesarios: Ordenador
Precedencias: -

0.2. Reuniones con el director
Paquete de trabajo: Gestión
Duración: 3 horas
Descripción: Reuniones regulares con el director del proyecto para evaluar el progreso y/o resolver dudas.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: -
Precedencias: -

1.1. Estudio del software relacionado
Paquete de trabajo: Estudio previo
Duración: 1 horas
Descripción: Repasar la estructura de los archivos generados por GanttProject para recordar el contenido de las tareas y como se relacionan entre sí.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el progama GanttProject.
Precedencias: -

1.2. Estudio de la funcionalidad actual
Paquete de trabajo: Estudio previo
Duración: 2 horas
Descripción: Estudiar el código que gestiona la conversación con el usuario y el código encargado de retrasar las tareas de la forma en la que se hace actualmente.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el código fuente de Ikastenbot.
Precedencias: -

2.1. Diseño de diagramas de clases
Paquete de trabajo: Planificación y diseño
Duración: 2 horas
Descripción: Diseñar los diagramas de clases de la funcionalidad a implementar.
Entradas: Ninguna.
Salidas: Diagramas de clases.
Recursos necesarios: Papel y lápiz.
Precedencias: Tareas 1.1, 1.2.

2.2. Diseño de diagramas de secuencia
Paquete de trabajo: Planificación y diseño
Duración: 2 horas
Descripción: Diseñar los diagramas de secuencia de las principales funciones de las clases diseñadas.
Entradas: Diagramas de clases.
Salidas: Diagramas de secuencia.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

2.3. Diseño de pruebas unitarias
Paquete de trabajo: Planificación y diseño
Duración: 2 horas
Descripción: Realizar el diseño teórico de los test para validar el funcionamiento correcto del código.
Entradas: Diagramas de clases.
Salidas: Diseño de las pruebas unitarias.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

3.1. Implementación de la solución
Paquete de trabajo: Implementación
Duración: 10 horas
Descripción: Implementar la solución diseñada.
Entradas: Diagramas de clases y de secuencia.
Salidas: Código fuente de la solución.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.2.

3.2. Implementación de las pruebas unitarias
Paquete de trabajo: Implementación
Duración: 3 horas
Descripción: Implementar las pruebas unitarias diseñadas.
Entradas: Diseño de las pruebas unitarias.
Salidas: Código fuente de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.3.

4.1. Ejecución de las pruebas unitarias
Paquete de trabajo: Verificación y corrección
Duración: 1 hora
Descripción: Ejecutar las pruebas unitarias sobre el código escrito.
Entradas: Código fuente de la solución y de las pruebas.
Salidas: Informe de errores de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 3.1 y 3.2.

4.2. Verificación de la funcionalidad
Paquete de trabajo: Verificación y corrección
Duración: 2 horas
Descripción: Pruebas manuales para verificar que la funcionalidad funciona correctamente en todos los casos.
Entradas: Código fuente de la solución.
Salidas: Informe de errores de las pruebas de funcionalidad.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 3.1.

4.3. Corrección de errores
Paquete de trabajo: Verificación y corrección
Duración: 3 horas
Descripción: Corrección de todos los errores, fallos y descuidos que se hayan encontrado mediante las pruebas y verificación.
Entradas: Código fuente de la solución e informes de errores obtenidos.
Salidas: Código fuente libre de errores.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 4.1 y 4.2.

7.1.3. Planificación Temporal

En la tabla 10 se resumen las tareas definidas junto con su duración. A partir de estas tareas se planeó el tiempo de trabajo tal y como se muestra en la figura 31.

Figura 31: Tarea 1 - Diagrama Gantt.

Tarea	Duración (h)
0. Gestión	10
0.1. Documentación	7
0.2. Reuniones con el director	3
1. Estudio previo	3
1.1. Estudio del software relacionado	1
1.2. Estudio de la funcionalidad actual	2
2. Planificación y diseño	6
2.1. Diseño de diagramas de clases	2
2.2. Diseño de diagramas de secuencia	2
2.3. Diseño de pruebas unitarias	2
3. Implementación	13
3.1. Implementación de la solución	10
3.2. Implementación de las pruebas unitarias	3
4. Verificación y corrección	6
4.1. Ejecución de las pruebas unitarias	1
4.2. Verificación de la funcionalidad	2
4.3. Corrección de errores	3
Total	**38**

Cuadro 10: Tarea 4 - Tareas planificadas.

7.1.4. Planificación Económica

Sabiendo la duración estimada de la tarea y utilizando los datos calculados en el apartado 2.6 se puede calcular el coste de esta tarea.

$$(4,82\ €/\text{hora} + 0{,}0145\ €/\text{hora}) * 38\ \text{horas} = 183,71\ €$$

7.2. Análisis y Diseño

Cuando Ikastenbot pregunta al alumno si desea retrasar una tarea concreta, si la respuesta es afirmativa, el bot envía un segundo mensaje preguntando

cuánto tiempo, en días, se desea retrasar la tarea. Al recibir una respuesta válida, se ejecutan las acciones necesarias y se envía un último mensaje de confirmación. La mejor forma de permitir al usuario elegir de qué manera se debería retrasar la tarea, sería añadir un nuevo paso a la conversación. Justo antes de preguntar por la duración del retraso, Ikastenbot debería preguntar por la forma en la que se quiere llevar a cabo.

Para esto, sería necesario añadir un nuevo paso a la conversación del bot asociada al comando de retrasar tareas. Este paso enviaría un mensaje preguntando el modo de posponer la tarea y almacenaría la opción elegida por el usuario.

Por otro lado, en la clase `XmlUtilsService` habita la función `delayTaskAndDependants`, encargada de retrasar las tareas, la cual sería necesario modificar para adoptar la nueva alternativa. La primera versión del diseño modificaba dicha función añadiendo un parámetro booleano para determinar que estrategia se debería seguir para retrasar la tarea (extender la duración o posponer el inicio). Sin embargo, este tipo de solución (un parámetro de señal) no se considera un buen diseño, ya que hace que las llamadas al método sean poco intuitivas y más complejas de lo que deberían[8]. Por ello, se decidió separar la funcionalidad en dos método públicos más uno extra privado que contiene el código común. Estos método se describen en la figura 32. En este diagrama solo aparecen los método que se van a crear o modificar. El resto de la clase que ya existía no se ha incluido para evitar confusión.

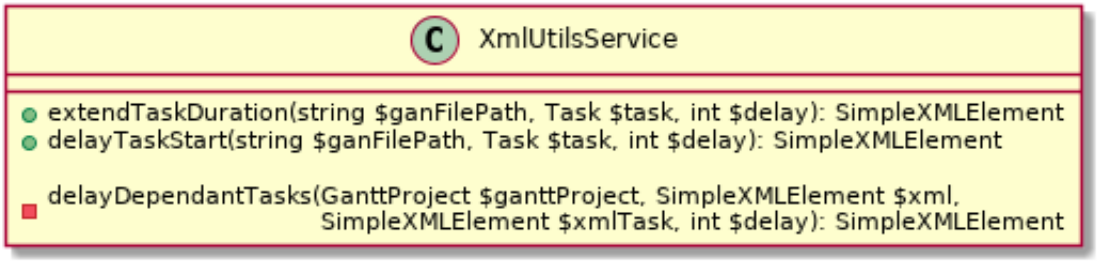

Figura 32: Tarea 4 - Diagrama de clases.

El nuevo de estos métodos se puede ver en las figuras 42, 43 y 44. Estos diagramas de secuencia contienen la funcionalidad que ya existía en el método junto con los cambios planeados.

7.3. Desarrollo

Primero se implementa el nuevo mensaje del bot. Para ello se añade un paso nuevo a la conversación siguiendo la misma estructura que en las conversaciones de otros comandos. Se envía al usuario un mensaje preguntando qué opción quiere utilizar para retrasar la tarea ofreciendo una breve explicación de las alternativas y mostrando un teclado personalizado con el que realizar la selección (Fig 33). Tras recibir la respuesta, antes de continuar con la conversación, se verifica que la respuesta recibida es válida. En caso negativo, se vuelve a enviar un mensaje parecido que indica que la respuesta enviada no es válida.

Figura 33: Tarea 4 - Mensaje de Ikastenbot preguntado de qué forma retrasar la tarea.

La elección del usuario se utiliza para determinar a qué función llamar para retrasar la tarea. Una vez que se tiene la respuesta del usuario, se continúa con la conversación para preguntar cuánto tiempo se desea retrasar la tarea.

En cuanto a las funciones, de la función original (`delayTaskAndDependants`) se extrae a una función privada (`delayDependantTasks`) la funcionalidad común que es necesaria para ambas formas de retrasar tareas. A continuación se crean las dos nuevas funciones (`extendTaskDuration` y `delayTaskStart`), cada una retrasando la tarea de una forma distinta y haciendo uso de `delayDependantTasks` para la parte común.

7.4. Verificación y Evaluación

De nuevo, las pruebas realizadas se dividen en tests automatizados mediante pruebas unitarias y verificación manual de la funcionalidad.

7.4.1. Pruebas Unitarias

A la hora de realizar las pruebas, lo primero fue modificar los test ya existentes para adaptarse al nuevo parámetro de la función y ejecutarlo para comprobar que no se ha estropeado nada durante el desarrollo.

Una vez hecho eso, se diseñaron e implementaron nuevas pruebas unitarias para cubrir los nuevos posibles casos a tratar.

testModifyStartTaskOneDependency
Descripción: Se carga el xml de un diagrama Gantt predefinido y se retrasa una tarea, de la cual depende otra tarea, posponiendo su fecha de inicio y manteniendo su duración.
Resultado esperado: La tarea se retrasa correctamente y la tarea dependiente se desplaza de forma acorde.

testModifyStartTaskAndDependenciesManyDependencies
Descripción: Se carga el xml de un diagrama Gantt predefinido y se retrasa una tarea, de la cual dependen varias tareas, posponiendo su fecha de inicio y manteniendo su duración.
Resultado esperado: La tarea se retrasa correctamente y todas las tareas dependiente se desplazan de forma acorde.

7.4.2. Pruebas de Funcionamiento

Para comprobar si la funcionalidad ha sido implementada correctamente se realizaron una serie de pruebas manuales con el bot.

Retrasar tarea alargando su duración
Descripción: Se envía un diagrama Gantt a Ikastenbot y se pide retrasar una tarea alargando su duración.
Resultado esperado: La duración de la tarea se alarga, se retrasan las tareas dependientes y se modifican los datos en la base de datos.

Retrasar tarea posponiendo su inicio
Descripción: Se envía un diagrama Gantt a Ikastenbot y se pide retrasar una tarea posponiendo su inicio.
Resultado esperado: La fecha de inicio de la tarea se retrasa, se retrasan las tareas dependientes y se modifican los datos en la base de datos.

7.5. Conclusiones

7.5.1. Cumplimiento de Objetivos

Se considera que el resultado obtenido cumple con los objetivos ya que funciona correctamente y está implementado de una manera satisfactoria.

7.5.2. Revisión de la Planificación

En la tabla 11 se compara el tiempo planificado para la tarea con el empleado realmente. En varios puntos se necesitó más tiempo del esperado dado que originalmente se realizó un diseño algo distinto al explicado anteriormente. Después de haber desarrollado dicho diseño, se decidió cambiarlo por el diseño actual. No hizo falta deshacer todo el trabajo realizado, pero sí causó algunas horas más de trabajo en los aspectos de diseño y desarrollo.

Tarea	Duración Estimada (h)	
0. Gestión	10	10
0.1. Documentación	7	8
0.2. Reuniones con el director	3	2
1. Estudio previo	3	4
1.1. Estudio del software relacionado	1	2
1.2. Estudio de la funcionalidad actual	2	2
2. Planificación y diseño	6	9
2.1. Diseño de diagramas de clases	2	3
2.2. Diseño de diagramas de secuencia	2	4
2.3. Diseño de pruebas unitarias	2	2
3. Implementación	13	18
3.1. Implementación de la solución	10	14
3.2. Implementación de las pruebas unitarias	3	4
4. Verificación y corrección	6	10
4.1. Ejecución de las pruebas unitarias	1	2
4.2. Verificación de la funcionalidad	2	5
4.3. Corrección de errores	3	3
Total	38	47

Cuadro 11: Tarea 4 - Duración estimada vs. duración real.

8. Tarea - Comparación de Tareas

8.1. Planteamiento de la Tarea

8.1.1. Objetivos

Ikastenbot tiene la opción de recibir diagramas Gantt representando la planificación temporal que el alumno ha preparado para su proyecto. De esta manera, se pueden almacenar las tareas e hitos programadas para lanzar recordatorios automatizados al alumno cuando estos eventos se aproximen. Gracias a esto, en la base de datos que utiliza el Ikastenbot se recoge un registro de las tareas planeadas por los usuarios.

Se desea utilizar este registro para detectar datos atípicos en las tareas enviadas por los usuarios. Cuando un alumno envíe un diagrama Gantt, se quiere analizar cada una de sus tareas y contrastarlas con tareas similares almacenadas en la base de datos, con el objetivos de notificar acerca de tareas que tengan asignada una duración muy diferente (sea de más o de menos) de la media histórica. El objetivo no es indicar al alumno que su planificación es incorrecta, sino ofrecer un punto de comparación en caso de duda.

8.1.2. Alcance

En las figuras 34 y 35 se muestra el esquema de descomposición de tareas ideado para este trabajo. A continuación se enumeran en más detalle cada una de las tareas que lo componen.

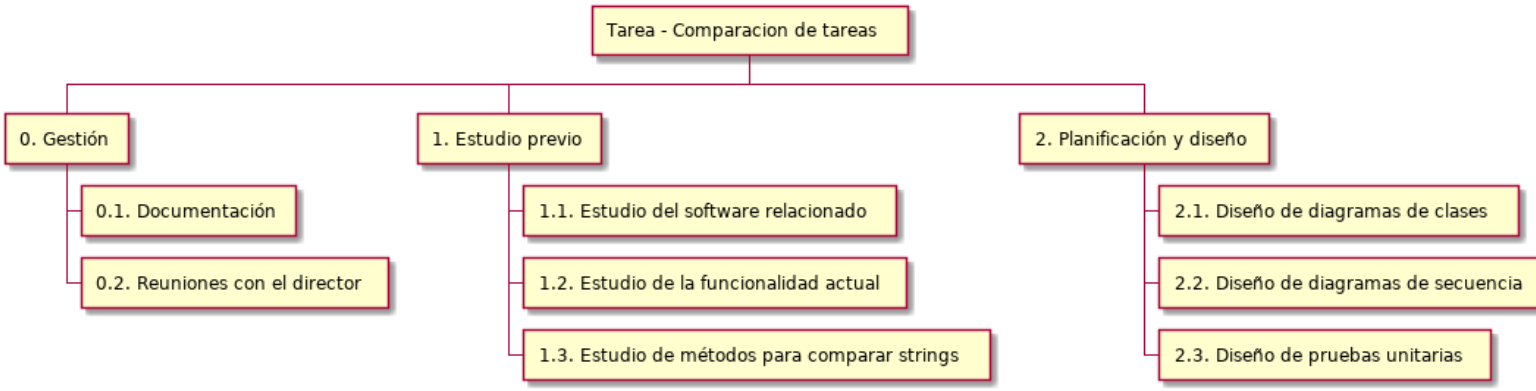

Figura 34: Tarea 5 - EDT (1).

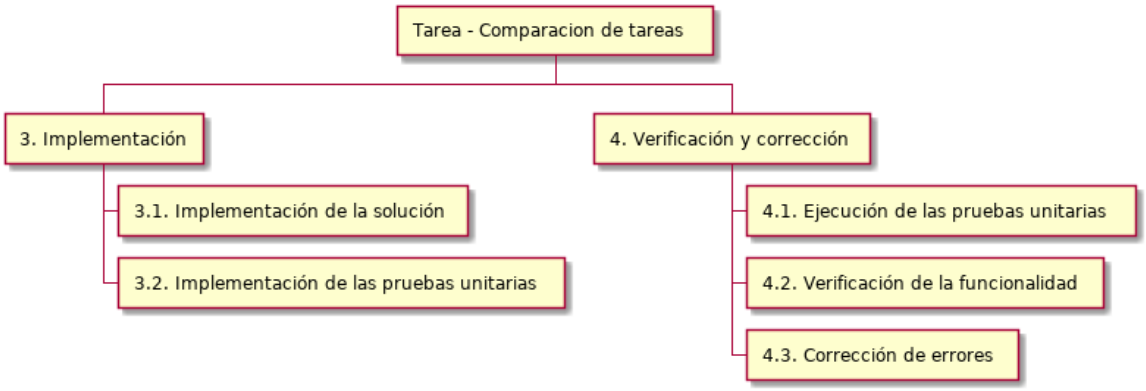

Figura 35: Tarea 5 - EDT (2).

0.1. Documentación
Paquete de trabajo: Gestión
Duración: 8 horas
Descripción: Documentación de toda la planificación y trabajo realizados en relación a esta tarea del proyecto.
Entradas: Ninguna.
Salidas: Documentación del trabajo.
Recursos necesarios: Ordenador
Precedencias: -

0.2. Reuniones con el director
Paquete de trabajo: Gestión
Duración: 3 horas
Descripción: Reuniones regulares con el director del proyecto para evaluar el progreso y/o resolver dudas.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: -
Precedencias: -

1.1. Estudio del software relacionado
Paquete de trabajo: Estudio previo
Duración: 2 horas
Descripción: Estudiar el programa utilizado para generar los diagramas Gantt que Ikastenbot puede leer, GanttProject, para aprender como funciona, y que información contienen los archivos que genera.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el progama GanttProject.
Precedencias: -

1.2. Estudio de la funcionalidad actual
Paquete de trabajo: Estudio previo
Duración: 3 horas
Descripción: Estudiar el código que recibe, analiza y almacena en contenido del Gantt para saber como y donde integrar la nueva funcionalidad.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con el código fuente de Ikastenbot.
Precedencias: -

1.3. Estudio de métodos para comparar strings
Paquete de trabajo: Estudio previo
Duración: 5 horas
Descripción: Buscar distintos métodos, estrategias y métricas para comparar strings. Necesario para determinar si dos tareas se consideran similares o no.
Entradas: Ninguna.
Salidas: Ninguna.
Recursos necesarios: Ordenador con acceso a internet.
Precedencias: -

2.1. Diseño de diagramas de clases
Paquete de trabajo: Planificación y diseño
Duración: 3 horas
Descripción: Diseñar los diagramas de clases de la funcionalidad a implementar.
Entradas: Ninguna.
Salidas: Diagramas de clases.
Recursos necesarios: Papel y lápiz.
Precedencias: Tareas 1.1, 1.2 y 1.3.

2.2. Diseño de diagramas de secuencia
Paquete de trabajo: Planificación y diseño
Duración: 3 horas
Descripción: Diseñar los diagramas de secuencia de las principales funciones de las clases diseñadas.
Entradas: Diagramas de clases.
Salidas: Diagramas de secuencia.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

2.3. Diseño de pruebas unitarias
Paquete de trabajo: Planificación y diseño
Duración: 2 horas
Descripción: Realizar el diseño teórico de los test para validar el funcionamiento correcto del código.
Entradas: Diagramas de clases.
Salidas: Diseño de las pruebas unitarias.
Recursos necesarios: Papel y lápiz.
Precedencias: Tarea 2.1.

3.1. Implementación de la solución
Paquete de trabajo: Implementación
Duración: 15 horas
Descripción: Implementar la solución diseñada.
Entradas: Diagramas de clases y de secuencia.
Salidas: Código fuente de la solución.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.2.

3.2. Implementación de las pruebas unitarias
Paquete de trabajo: Implementación
Duración: 3 horas
Descripción: Implementar las pruebas unitarias diseñadas.
Entradas: Diseño de las pruebas unitarias.
Salidas: Código fuente de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 2.3.

4.1. Ejecución de las pruebas unitarias
Paquete de trabajo: Verificación y corrección
Duración: 1 hora
Descripción: Ejecutar las pruebas unitarias sobre el código escrito.
Entradas: Código fuente de la solución y de las pruebas.
Salidas: Informe de errores de las pruebas unitarias.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 3.1 y 3.2.

4.2. Verificación de la funcionalidad
Paquete de trabajo: Verificación y corrección
Duración: 2 horas
Descripción: Pruebas manuales para verificar que la funcionalidad funciona correctamente en todos los casos.
Entradas: Código fuente de la solución.
Salidas: Informe de errores de las pruebas de funcionalidad.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tarea 3.1.

4.3. Corrección de errores
Paquete de trabajo: Verificación y corrección
Duración: 4 horas
Descripción: Corrección de todos los errores, fallos y descuidos que se hayan encontrado mediante las pruebas y verificación.
Entradas: Código fuente de la solución e informes de errores obtenidos.
Salidas: Código fuente libre de errores.
Recursos necesarios: Ordenador con PhpStorm y acceso a internet.
Precedencias: Tareas 4.1 y 4.2.

8.1.3. Planificación Temporal

En la tabla 12 se resumen las tareas definidas junto con su duración. A partir de estas tareas se planeó el tiempo de trabajo tal y como se muestra en la figura 36 teniendo en cuenta la estimación de 2 ó 3 horas de trabajo al día.

8.1.4. Evaluación Económica

Sabiendo la duración estimada de la tarea y utilizando los datos calculados en el apartado 2.6 se puede calcular el coste de esta tarea.

$$(4,82\ €/\text{hora} + 0{,}0145\ €/\text{hora}) * 54\ \text{horas} = 261,06\ €$$

Tarea	Duración (h)
0. Gestión	11
0.1. Documentación	8
0.2. Reuniones con el director	3
1. Estudio previo	10
1.1. Estudio del software relacionado	2
1.2. Estudio de la funcionalidad actual	3
1.3. Estudio de métodos para comparar strings	5
2. Planificación y diseño	8
2.1. Diseño de diagramas de clases	3
2.2. Diseño de diagramas de secuencia	3
2.3. Diseño de pruebas unitarias	2
3. Implementación	18
3.1. Implementación de la solución	15
3.2. Implementación de las pruebas unitarias	3
4. Verificación y corrección	7
4.1. Ejecución de las pruebas unitarias	1
4.2. Verificación de la funcionalidad	2
4.3. Corrección de errores	4
Total	**54**

Cuadro 12: Tarea 5 - Tareas planificadas.

8.2. Análisis y Diseño

Esta tarea se centra en buscar tareas almacenadas en la base de datos que sean similares en nombre a las definidas por el usuario. Por ello, el aspecto clave a diseñar es un sistema para comparar strings y determinar si son similares o no. Como punto de partida se ha tomado el artículo "SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts"[9], en el cual se describen las estrategias y métricas de comparación de textos utilizadas por los autores. A continuación se enumeran y describen las métricas seleccionadas para utilizar en la aplicación.

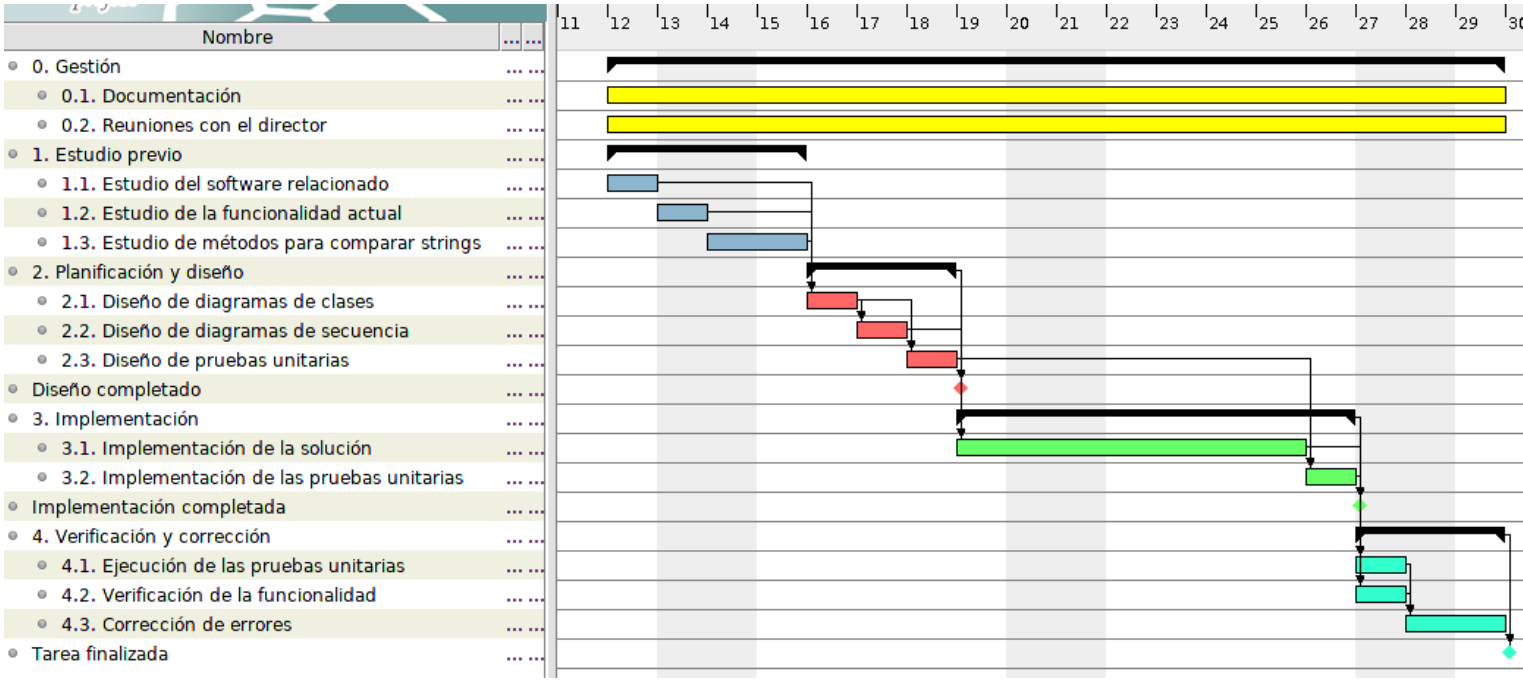

Figura 36: Tarea 5 - Diagrama Gantt.

8.2.1. Métricas de Similitud

Las métricas de similitud asignan un valor numérico en el rango [0, 1] a un par de strings en función de los similares que son entre sí, 1 significando que son idénticas y 0, que son completamente distintas.

8.2.1.1. Edit-based Metrics

Estas métricas se basan en la distancia de edición entre dos strings. Esta distancia es el número de operaciones necesarias para transformar una string en la otra. Se pueden distinguir un número de variantes dependiendo del la cantidad de operaciones que se permiten.

- **Optimal string alignment (OA)**
  Permite las operaciones "insertion of one character" y "deletion of one

character". p.e.:

- **insertion of one character:** "kitten" →"kittens"
- **deletion of one character:** "kitten" →"kittn"

- ***Levenshtein* distance**
  Sobre lo anterior, añade la operación "substitution of one character". p.e.:
  - **substitution of one character:** "kitten" →"kotten"
- ***Damerau-Levenshtein* distance**
  Sobre lo anterior, añade la operación "swap two neighboring characters". p.e.:
  - **swap two neighboring characters:** "kitten" →"kitetn"

Una vez conocida la distancia de edición entre las strings, se puede calcular su similitud.

Siendo S_1 y S_2 dos strings y d la distancia basada en ediciones entre S_1 y S_2, la similitud basada en ediciones se define como:

$$sim_{edit}(S_1, S_2) = \frac{max(|S_1|, |S_2|) - d(S_1, S_2)}{max(|S_1|, |S_2|)}$$

8.2.1.2. Profile-based Metrics

Estas métricas representan las strings como vectores. El espacio de vectorial (V) en el que se representan estos vectores se define a partir del contenido de las strings y se puede obtener mediante distintas estrategias.

- **tokens**
  Las strings se dividen en tokens (palabras separadas por espacios). Cada token distinto representa una dimensión. p.e.:
  S_1 = "casa gato caballo" →["casa", "gato", "caballo"]
  S_2 = "cabaña mega perro" →["cabaña", "mega", "perro"]
  V = ["casa", "gato", "caballo", "cabaña", "mega", "perro"]

- **n-grams**
  Las strings, tras eliminar los espacios, se dividen en subcadenas de n caracteres consecutivos (n-grams). Cada subcadena distinta representa una dimensión. p.e.:

  $n = 5$:

  S_1 = "casa gato caballo" →["casag", "atoca", "ballo"]

  S_2 = "cabaña mega perro" →["cabañ", "amega", "perro"]

  V = ["casag", "atoca", "ballo", "cabañ", "amega", "perro"]

- **n-shingles**
  Las strings se dividen en subcadenas de n tokens consecutivos (n-shingle). Cada subcadena distinta representa una dimensión. p.e.:

  $n = 2$:

  S_1 = "casa gato caballo" →["casa gato", "caballo"]

  S_2 = "cabaña mega perro" →["cabaña mega", "perro"]

  V = ["casa gato", "caballo", "cabaña mega", "perro"]

Dependiendo del método elegido, se considera cada token, n-gram o n-shingle en ambas strings como una dimensión de un vector en el espacio. Cada string se caracteriza como un vector en el espacio que se ha definido, y cada dimensión toma el valor 1 (el token/n-gram/n-shingle correspondiente está presente en string) o 0 (no está presente) (Bag of Words). Alternativamente, se puede tomar el número de apariciones en la string de cada token/n-gram/n-shingle como el valor de la dimensión correspondiente (Term Frequency).

La similitud entre dos string se define como la distancia Mahnattan entre los vectores derivados de cada string.

Siendo V el espacio vectorial definido, para los vectores

$$X = (x_1, x_2, ..., x_n) \ \& \ Y = (y_1, y_2, ..., y_n) \in V^n$$

la distancia Manhattan se define como

$$D(X, Y) = \sum_{i=1}^{n} |x_i - y_i|$$

Como la distancia Manhattan no tiene límite superior, para definir la similitud en el rango [0, 1] hay que obtener la distancia máxima que podría darse entre las strings. Esta distancia máxima es igual a la longitud del vector suma de los vectores que representas las strings. De esta manera, siendo X y Y los vectores que representas las strings S_1 y S_2 respectivamente, se calcula la similitud basada en la distancia Manhattan como

$$sim_{Manhattan}(S_1, S_2) = 1 - \frac{D(X, Y)}{||X + Y||}$$

8.2.1.3. Set-based Metrics

Se consideran todos los tokens/n-grams/n-shingles de las string como conjuntos de elementos y se calculan distintos coeficientes para dichos conjuntos.

Siendo Z_1 y Z_2 los conjuntos de tokens/n-grams/n-shingles de las strings S_1 y S_2 respectivamente:

$$sim_{Jaccard}(S_1, S_2) = Jaccard(Z_1, Z_2) = \frac{|Z_1 \cap Z_2|}{|Z_1 \cup Z_2|}$$

$$sim_{Dice}(S_1, S_2) = Dice(Z_1, Z_2) = \frac{2 \cdot |Z_1 \cap Z_2|}{|Z_1| + |Z_2|}$$

$$sim_{Overlap}(S_1, S_2) = Overlap(Z_1, Z_2) = \frac{|Z_1 \cap Z_2|}{min(|Z_1|, |Z_2|)}$$

8.2.2. Diagrama de Clases

Un vez comprendido el marco teórico se diseñaron las clases que llevarían a cabo la funcionalidad (Fig 37). Este diagrama pasó por varias iteraciones. En la figura se muestra únicamente la última versión.

- `SimilarTasksDurationNotifier:` Clase que inicia todo el proceso. Recibe la lista de tareas del usuario, utiliza las otras clases para obtener la información necesaria y envía el mensaje apropiado.

Figura 37: Tarea 5 - Diagrama de clases.

- `SimilarTaskFinder:` Esta clase también recibe la lista de tareas del usuario y las compara con las tareas de la base de datos, decidiendo si son similares o no. Para cada tarea, obtiene la duración media esperada según el registro de la base de datos.
- `StringComparator:` Esta es la clase que se utiliza para determinar si dos tareas son similares en función de sus nombres. Implementa las métricas de similitud descritas en el apartado anterior.

8.2.3. Diagramas de Secuencia

Con las clases definidas, se pasó a representar la secuencia de llamadas para las principales funcionalidades de las clases (Fig 45, 46 y 47). Estos diagramas no cuentan con ningún actor ya que no son producto directo de las acciones del usuario. En su lugar, la funcionalidad se añade como un paso más a un script ya existente.

8.3. Desarrollo

8.3.1. Comparador de Strings

El desarrollo de esta tarea comienza por la implementación de la clase comparadora de strings, `StringComparator`. Esta clase implementa las métricas de comparación de strings descritas en la sección 8.2.1. También cuenta con una serie de método para seleccionar que estrategia de división de strings se quiere utilizar (tokens, n-grams o n-shingles) y para elegir el valor del parámetro n.

8.3.1.1. Preprocesado

Antes de comenzar la comparación entre strings, es recomendable preprocesarlas para eliminar caracteres y subcadenas que no aportan información relevante pero que afectan a la similitud de las strings. En el preprocesado

genérico que se aplica a todas las string se convierten a minúsculas, se eliminan todos los caracteres no alfanuméricos (guiones, puntuación y otros símbolos), artículos y preposiciones. Por último, se eliminan números al comienzo de la palabra para casos en los que las tareas tengan números indicando su jerarquía (p.e.: "2.3. Diseño de Diagramas").

8.3.1.2. Edit-based Metrics

Las distintas variaciones de la distancia basada en ediciones son realmente el mismo algoritmo con pequeños cambios para tener en cuenta las distintas operaciones permitidas. La implementación de estas métricas se aprovecha de este hecho para unir todas las variaciones en una única función que implementa el algoritmo principal y que presenta dos parámetros booleanos para indicar si se desea permitir movimientos de transposición y substitución (*Damerau-Levenshtein*), substitución pero no transposición (*Levenshtein*) o solamente los movimientos básicos de añadir y eliminar caracteres ('Optimal Alignment').

8.3.1.3. División de Strings

Antes de añadir las métricas de similitud basadas en sets, es necesario implementar las distintas estrategias de división de strings. Cada una de las estrategias cuenta con un preprocesado adicional que aplica a las strings.

La estrategia de tokens, ya que se basa se dividir la string en palabras separadas por espacios, convierte todos los espacios en blanco (tabulaciones, espacios dobles, etc.) es un espacio en blanco simple para asegurar uniformidad. La estrategia de n-grams, la cual selecciona subcadenas de n caracteres seguido sin contar espacios, elimina todos los espacios en blanco. La estrategia n-shingles hace lo mismo que tokens y uniformiza los espacios en blanco.

Por último, para que la estrategia n-grams funcione correctamente, es necesario crear una función para dividir strings en subcadenas, ya que las funciones de PHP están preparadas para trabajar con strings en las que cada caracter ocupa un único byte. Strings con caracteres multi-byte (ñ, á, é, í, ó,

ú, etc.) no se dividirían correctamente: los caracteres multi-byte se contarían como dos caracteres y la división podría hacerse en mitad de uno de estos, resultando en caracteres erroneos. Para solucionar esto, se implementa una función que divide las strings en listas de n caracteres, tendiendo en cuenta caracteres multi-byte.

8.3.1.4. Set-based Metrics

Una vez implementados los métodos para representar las strings como sets de subcadenas, se procede a programar las funciones para calcular la similitud en función de los coeficientes de Jaccard, Dice y Overlap. Estas funciones se limitan a aplicar la fórmula del índice correspondiente a los sets que representan las strings en cuestión, y el resultado es la similitud buscada.

8.3.1.5. Profile-based Metrics

Para la similaridad basada en la distancia Manhattan, se hace uso de las estrategias de división de strings ya mencionadas para representar las strings como vectores. Tras obtener la distancia Manhattan entre los vectores, solo resta convertir a una la relativa entre 0 y 1 para obtener la similitud.

8.3.2. Buscador de Tareas Similares

`SimilarTaskFinder` es la clase encargada de comparar las tareas del usuario con las de la base de datos. Su principal función es la de, dada una serie de tareas, para cada una de ellas encontrar en la base de datos aquellas con nombres similares y determinar su duración media esperada. Para ello es necesario un método de determinar si los nombres de dos tareas se consideran similares o no. Este método hace uso de las métricas de similitud implementadas en `SimilarTaskFinder`.

El artículo del cual se obtuvieron las métricas de similitud tenía como objetivo comparar textos largos, por lo que sus resultados acerca de las métricas más efectivas no son completamente transferibles a este trabajo centrado

en textos mucho más cortos. Es por ello que la combinación de métricas, parámetros y umbrales se obtiene mediante prueba y error, analizando que beneficios aporta cada una de las métricas y estrategias, combinándolas de forma coherente y variando los umbrales hasta obtener resultados satisfactorios consistentemente.

8.3.3. Notificador

La última clase a implementar, `SimilarTasksDurationNotifier`, es la encargada de notificar al usuario. A esta clase se accede desde el script que se ejecuta al recibir un diagrama Gantt y utiliza la clase `SimilarTaskFinder` para obtener las tareas del usuario que divergen mucho en duración con respecto a las de la base de datos. Una vez hecho eso, si es necesario, envía un mensaje al alumno listando cada una de estas tareas, su duración, y su duración esperada. En la figura 38 se muestra un ejemplo del mensaje eviado por Ikastenbot.

8.4. Verificación y Evaluación

La verificación de esta tarea se llevó a cabo en dos partes. Por un lado, las pruebas unitarias para el código, y por otro, pruebas manuales para verificar que el bot se comporta de manera correcta.

8.4.1. Pruebas Unitarias

Tras implementar las clases definidas, se diseñaron una serie de pruebas unitarias para verificar su correcto funcionamiento y para automatizar futuras pruebas en caso de que el código cambiase. Se prepararon tests para las funcionalidades principales de cada una de las clases.

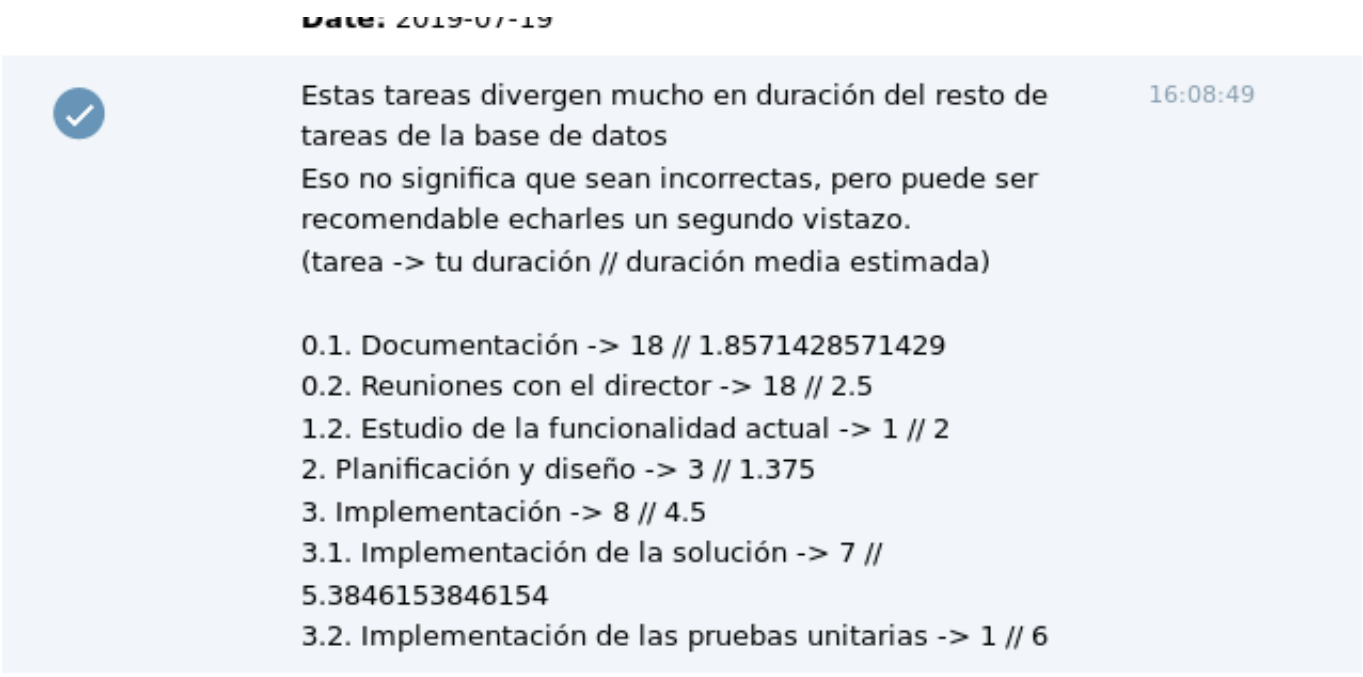

Figura 38: Tarea 5 - Ejemplo de funcionamiento.

8.4.1.1. `SimilarTaskFinder`:

Para esta clase se simuló una base de datos de prueba para tener un entorno controlado del que obtener las tareas que se comparan con las que recibe la clase. Se hicieron tests para sus dos métodos públicos, comprobando que devuelven los resultados esperados.

testGetSimilarTasksDurations
Descripción: Se carga una lista de tareas preparada y se comparan con las tareas de la base de datos simulada en busca de las tareas de la base de datos que se parezcan a las tareas seleccionadas.
Resultado esperado: Se detectan las tareas de la base de datos con nombres similares a alguna de las tareas elegidas y se relacionan correctamente.

testGetTasksWithAtypicalDuration
Descripción: Se carga una lista de tareas preparada y se comparan con las tareas de la base de datos simulada en busca de las tareas que se parezcan a alguna de la base de datos pero cuya duración sea muy distinta de la media de sus semejantes.
Resultado esperado: Se identifican correctamente las tareas que difieren mucho en duración de la media de las tareas de la base de datos con nombre similar.

8.4.1.2. `StringComparator`:

Para esta clase se implementaron métodos para comprobar que todas las métricas, con distintas combinaciones de parámetros, obtienen la similitud correcta entre varias strings. Para ello, previamente se calculó de forma manual la similitud de estas strings.

testSetStrategy
Descripción: Se cambia la estrategia de la instancia de StringComparator a distintos valores.
Resultado esperado: Cada cambio de estrategia se lleva a cabo sin errores y el cambio de las variables se hace correctamente.

testRemoveNumbersAtStart
Descripción: Se aplica la función que elimina números al comienzo de strings a distintas strings.
Resultado esperado: Se eliminan los números al comienzo de las strings correctamente.

testSimilarityOA
Descripción: Se calcula la similitud por 'Optimal Alignment' de dos string y se compara con la similitud obtenida manualmente.
Resultado esperado: La similitud obtenida coincide con la real.

testSimilarityLevenshtein
Descripción: Se calcula la similitud por distancia *Levenshtein* de dos string y se compara con la similitud obtenida manualmente.
Resultado esperado: La similitud obtenida coincide con la real.

testSimilarityDamLev
Descripción: Se calcula la similitud por distancia *Damerau-Levenshtein* de dos string y se compara con la similitud obtenida manualmente.
Resultado esperado: La similitud obtenida coincide con la real.

testSimilarityJaccard
Descripción: Se calculan las similitudes mediante el índice Jaccard de dos string utilizando distintas estrategias de división de strings y se compara con la similitud obtenida manualmente.
Resultado esperado: Las similitudes obtenidas coinciden con las reales.

testSimilarityDice
Descripción: Se calculan las similitudes mediante el índice Dice de dos string utilizando distintas estrategias de división de strings y se compara con la similitud obtenida manualmente.
Resultado esperado: Las similitudes obtenidas coinciden con las reales.

testSimilarityOverlap
Descripción: Se calculan las similitudes mediante el índice Overlap de dos string utilizando distintas estrategias de división de strings y se compara con la similitud obtenida manualmente.
Resultado esperado: Las similitudes obtenidas coinciden con las reales.

testManhattanDistance
Descripción: Se calcula la similitud mediante la distancia Manhattan de dos string y se compara con la similitud obtenida manualmente.
Resultado esperado: La similitud obtenida coincide con la real.

No se implementó test para el método que decide (en términos absolutos: si o no) si dos strings son consideradas similares o no. La razón es que dicha función depende de las métricas implementadas en `StringComparator`, las cuales ya tienen tests propios, y porque la combinación exacta de métricas, parámetros y umbrales está sujeta a cambios. Esta fórmula se alcanzó mediante prueba y error hasta que se obtuvieron resultados satisfactorios, pero nada garantiza que su comportamiento exacto no vaya a modificarse en un futuro.

8.4.2. Pruebas de Funcionamiento

La verificación de la funcionalidad en conjunto de llevó a cabo de forma manual, conociendo el estado de la base de datos, enviando diagramas Gantt que cubriesen los posibles casos de error y comparando los resultados obtenidos con los esperados.

Diagrama Gantt sin tareas
Descripción: Se envía a Ikastenbot un diagrama Gantt vacío, sin ninguna tarea.
Resultado esperado: Ikastenbot no responde con ningún mensaje en relación a esta funcionalidad.

Diagrama Gantt con tarea extraña
Descripción: Se envía a Ikastenbot un diagrama Gantt que incluye una tarea completamente distinta a todas las que hay en la base de datos.
Resultado esperado: El mensaje acerca de la duración de las tareas no menciona la tarea en cuestión.

Diagrama Gantt con tarea aceptable
Descripción: Se envía a Ikastenbot un diagrama Gantt que incluye una tarea similar a otras de la base de datos y que tiene una duración cercana a la media de sus semejantes.
Resultado esperado: El mensaje acerca de la duración de las tareas no menciona la tarea en cuestión.

Diagrama Gantt con tarea atípica
Descripción: Se envía a Ikastenbot un diagrama Gantt que incluye una tarea similar a otras de la base de datos y pero que tiene una duración muy distinta a la media de sus semejantes.
Resultado esperado: El mensaje acerca de la duración de las tareas menciona la tarea en cuestión y especifica su duración media esperada.

Diagrama Gantt sin tarea notables
Descripción: Se envía a Ikastenbot un diagrama Gantt del cual ninguna tarea requiere ser mencionada.
Resultado esperado: Ikastenbot no responde con ningún mensaje en relación a esta funcionalidad.

8.5. Conclusiones

8.5.1. Cumplimiento de Objetivos

Tras haber verificado el correcto funcionamiento, llevadas a cabo las correcciones sugeridas por el lider del proyecto y por el tutor y entregado los últimos resultados, la funcionalidad fue finalmente aceptada e incluida en el repositorio. Con esto, se considera el objetivo principal de entregar un producto funcional y satisfactorio, alcanzado.

8.5.2. Revisión de la Planificación

En la tabla 13 se muestra el tiempo empleado realmente en la tarea en comparación a lo originalmente planeado. Se puede ver como el desarrollo y el diseño, en conjunto, han llevado algo más de tiempo más de lo esperado. La planificación original fue un poco optimista, especialmente en cuanto a la implementación.

Tarea	Duración Estimada (h)	Duración Real (h)
0. Gestión	11	9
0.1. Documentación	8	7
0.2. Reuniones con el director	3	2
1. Estudio previo	10	9
1.1. Estudio del software relacionado	2	2
1.2. Estudio de la funcionalidad actual	3	2
1.3. Estudio de métodos para comparar strings	5	4
2. Planificación y diseño	8	12
2.1. Diseño de diagramas de clases	3	4
2.2. Diseño de diagramas de secuencia	3	6
2.3. Diseño de pruebas unitarias	2	2
3. Implementación	18	26
3.1. Implementación de la solución	15	22
3.2. Implementación de las pruebas unitarias	3	4
4. Verificación y corrección	7	8
4.1. Ejecución de las pruebas unitarias	1	1
4.2. Verificación de la funcionalidad	2	4
4.3. Corrección de errores	4	3
Total	54	64

Cuadro 13: Tarea 5 - Duración estimada vs. duración real.

9. Conclusiones y Trabajo Futuro

Debido a las tareas encargadas, el trabajo no ha tenido una complejidad ni una carga de trabajo excesivas. Sin embargo, complejidad y carga de trabajo no eran objetivos del trabajo, sino que lo era el aportar a un proyecto en desarrollo de la forma en que hiciera falta. Y en ese aspecto se considera que, por lo general, se ha cumplido. En la tabla 14 se resume el contenido de las tareas realizadas y si se considera que han cumplido los objetivos o no.

En cuanto a los objetivos personales, haber trabajado en un entorno algo más cercano a la realidad ha constituido una buena experiencia mostrando las ventajas de la crítica constructiva, la autocrítica, la discusión de alternativas, la mejora continua y la importancia del buen diseño y las buenas prácticas.

Volviendo la vista al trabajo realizado Ikastenbot, siguen existiendo puntos de mejora, como por ejemplo optimizar el algoritmo que determina si los nombres de dos tareas son similares para hacerlo más eficaz, o incluso introducir elementos de *Machine Learning* para obtener un modelo que detecte tareas similares entrenado a partir de el registro de memorias que se tiene.

Tarea	Funcionalidad Original	Funcionalidad Añadida	Objetivos Cumplidos
1	Ikastenbot puede recibir diagramas Gantt creados pos el usuario.	En caso de que sea necesario, Ikastenbot responde con mensajes indicando la falta de reuniones de seguimiento, hitos o ambos en el Gantt recibido.	Sí
2	Ikastenbot envía mensajes automatizados avisando que se acerca el fin de una tarea planificada.	Al recibir nuevas versiones de un Gantt, se desactivan los mensajes que se enviarían avisando de tareas de las versiones anteriores.	No
3	Ikastenbot puede analizar un documento y detectar errores ortográficos y gramaticales.	El corrector ortográfico no considera palabras en inglés en textos no ingleses como errores.	Sí
4	Ikastenbot permite al usuario retrasar las tareas planificadas.	Se ofrecen dos formas distintas de retrasar las tareas para que el usuario pueda elegir la que más le conviene.	Sí
5	Ikastenbot puede recibir diagramas Gantt creados pos el usuario.	Al recibir un Gantt se responde con un mensaje indicando si la duración de algunas de las tareas definidas dista mucho de la duración de tareas similares en la base de datos.	Sí

Cuadro 14: Resumen de las tareas.

Referencias

[1] S. Kemp, "11 new people join social media every second (and other impressive stats)." https://blog.hootsuite.com/11-people-join-social-every-second/, 2018.

[2] "Telegram privacy policy." https://telegram.org/privacy#12-questions-and-concerns.

[3] "Telegram faq." https://telegram.org/faq#q-how-are-you-going-to-make-money-out-of-this.

[4] "Bots: An introduction for developers." https://core.telegram.org/bots.

[5] "Telegram bot api." https://core.telegram.org/bots/api.

[6] M. Guay, "What are webhooks?." https://zapier.com/blog/what-are-webhooks/, 2018.

[7] P. P. Kumar, "Effective use of gantt chart for managing large scale projects: a publication of the american association of cost engineers." https://search.proquest.com/docview/220442584, 2005.

[8] M. Fowler, "Flagargument." https://martinfowler.com/bliki/FlagArgument.html, 2011.

[9] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "Sotorrent: Reconstructing and analyzing the evolution of stack overflow posts." https://empirical-software.engineering/assets/pdf/msr18-sotorrent.pdf, 2018. Mining Software Repositories.

A. Diagramas de Secuencia

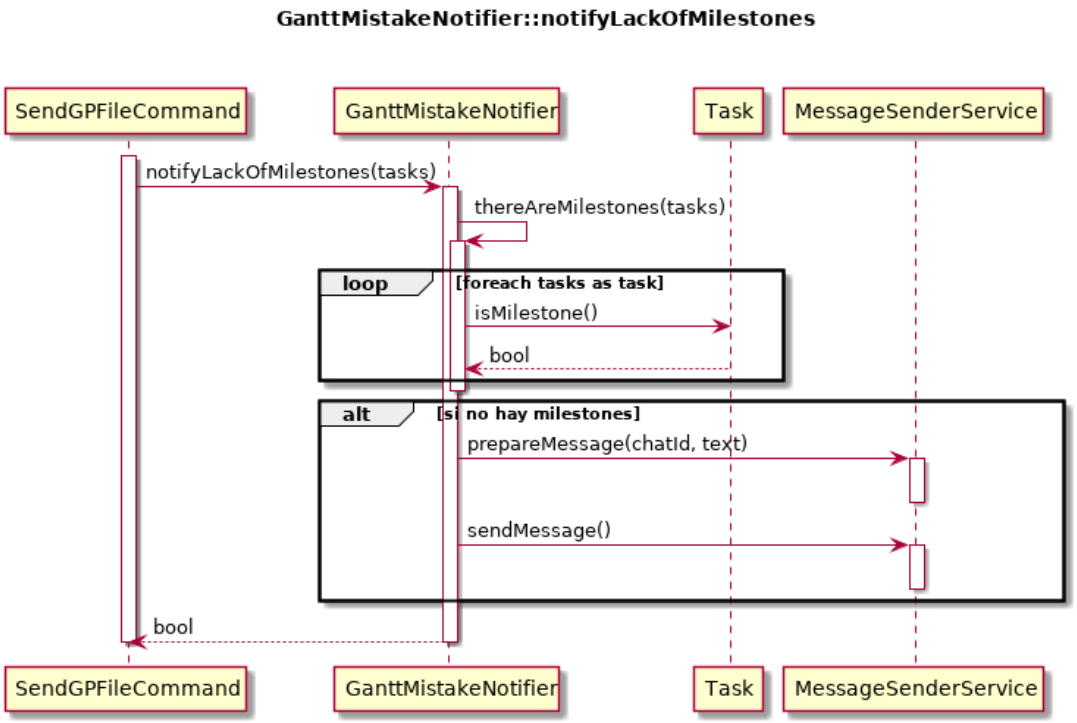

Figura 39: Tarea 1 - Función notifyLackOfMilestones.

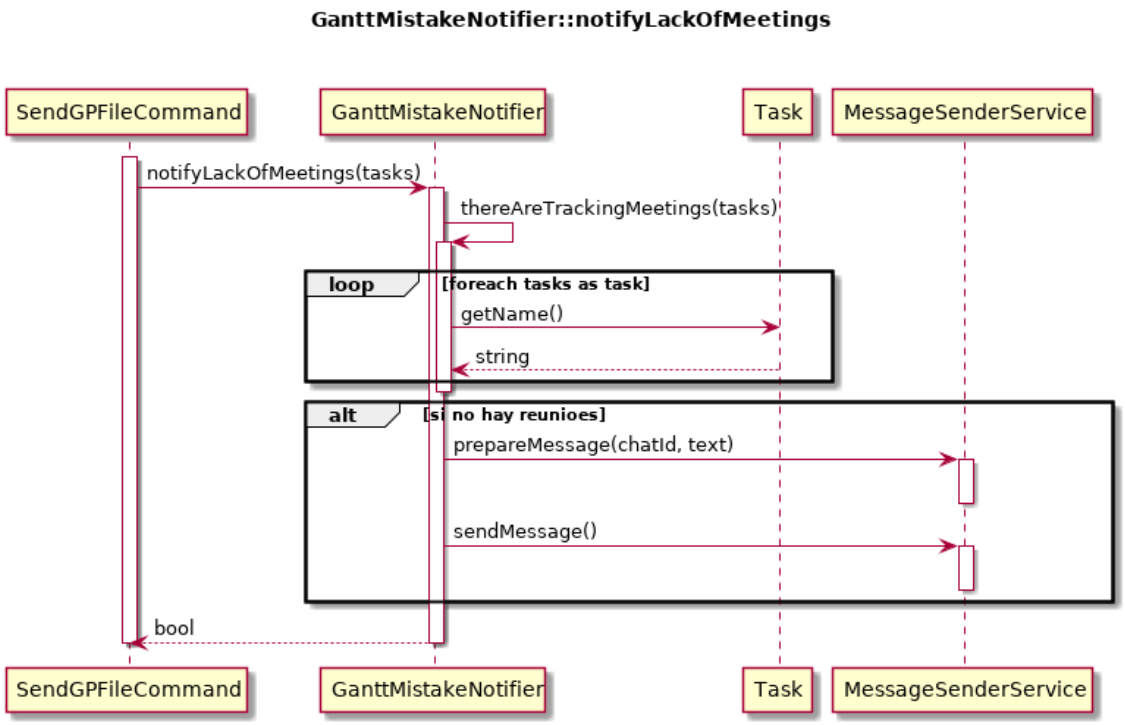

Figura 40: Tarea 1 - Función notifyLackOfMeetings.

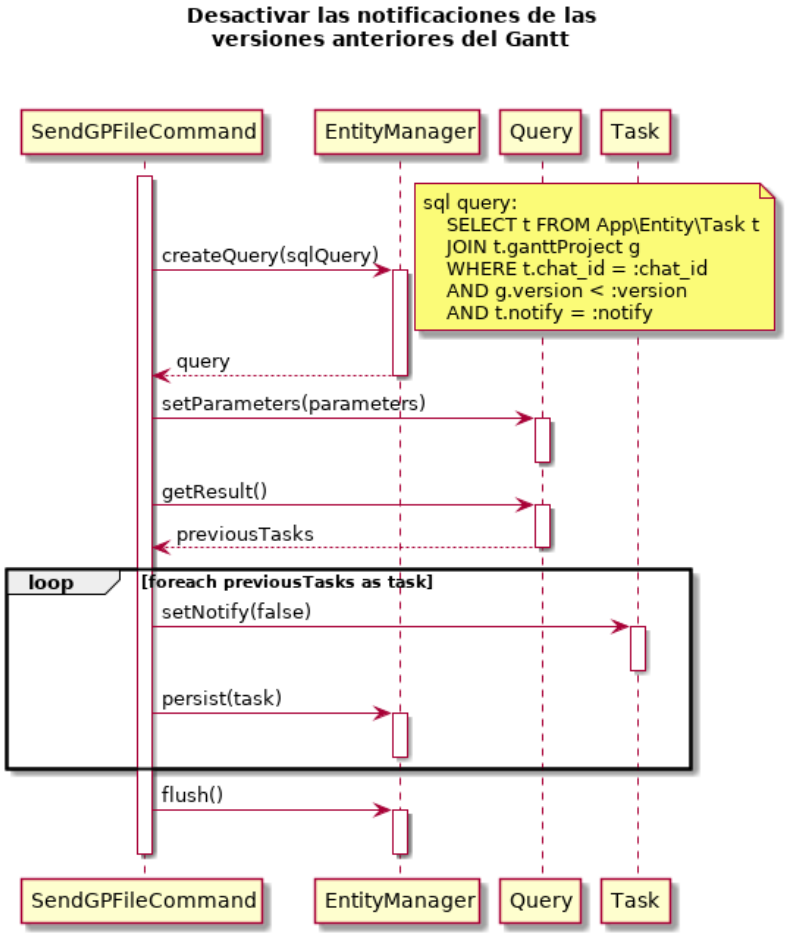

Figura 41: Tarea 2 - Desactivar las notificaciones de las versiones anteriores del Gantt.

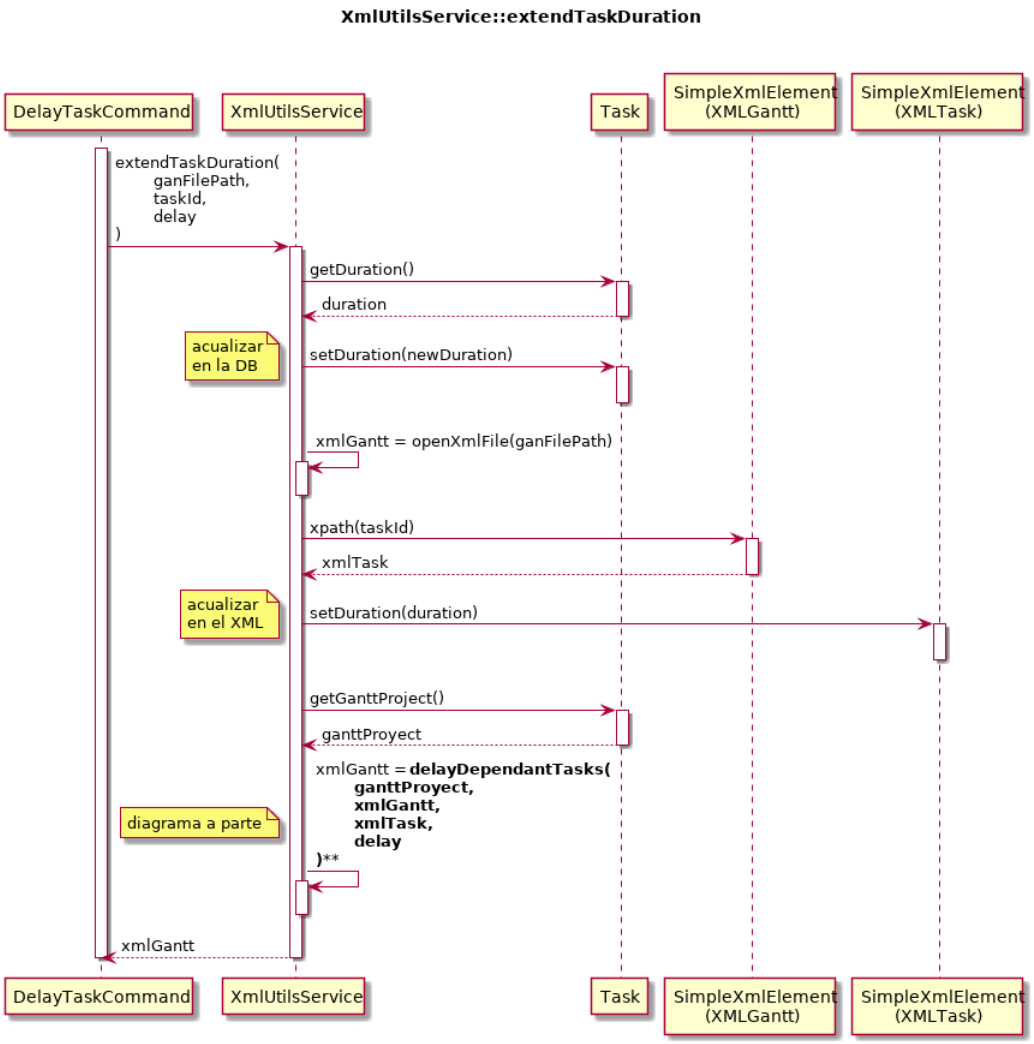

Figura 42: Tarea 4 - Función extendTaskDuration.

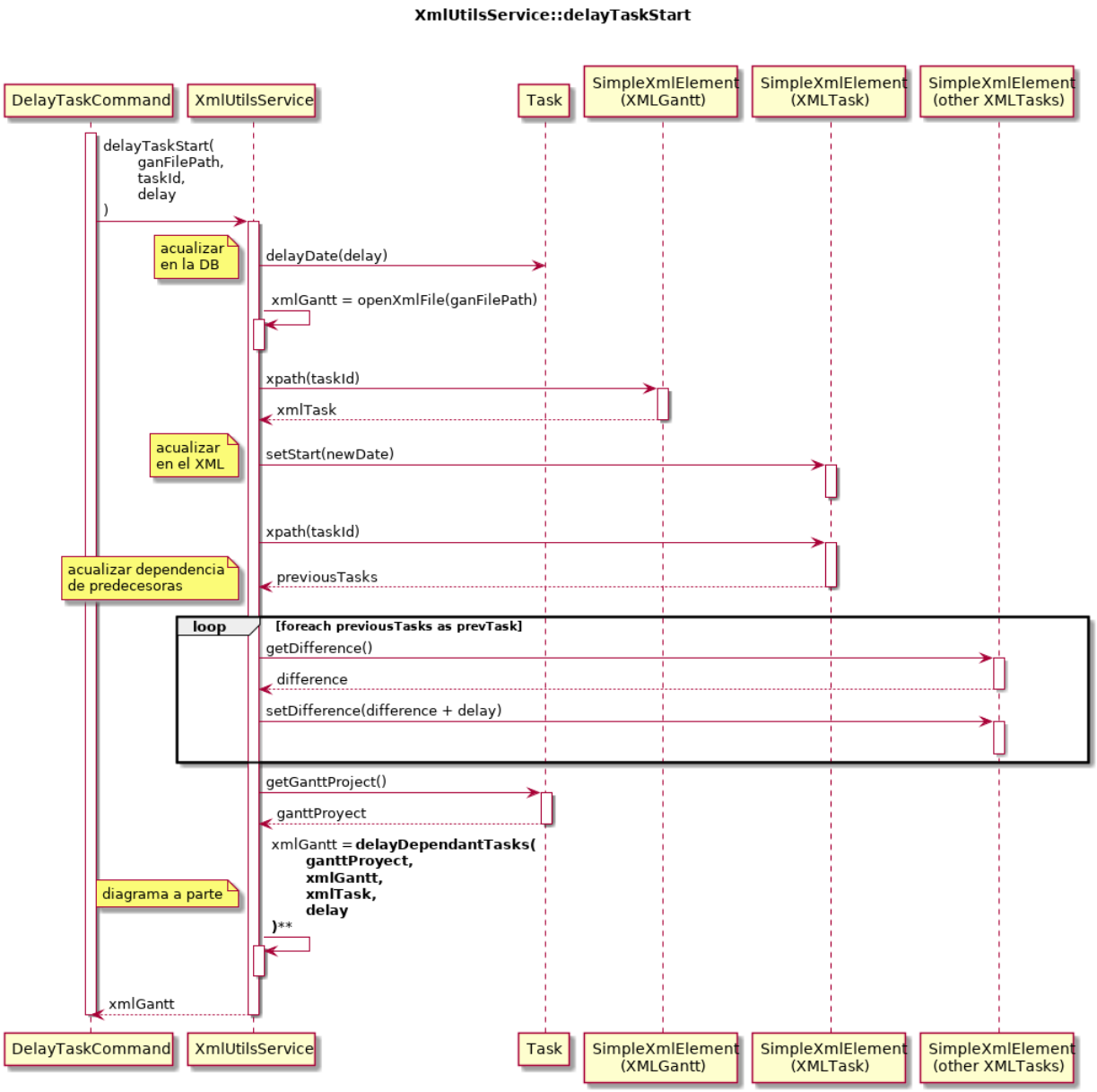

Figura 43: Tarea 4 - Función delayTaskStart.

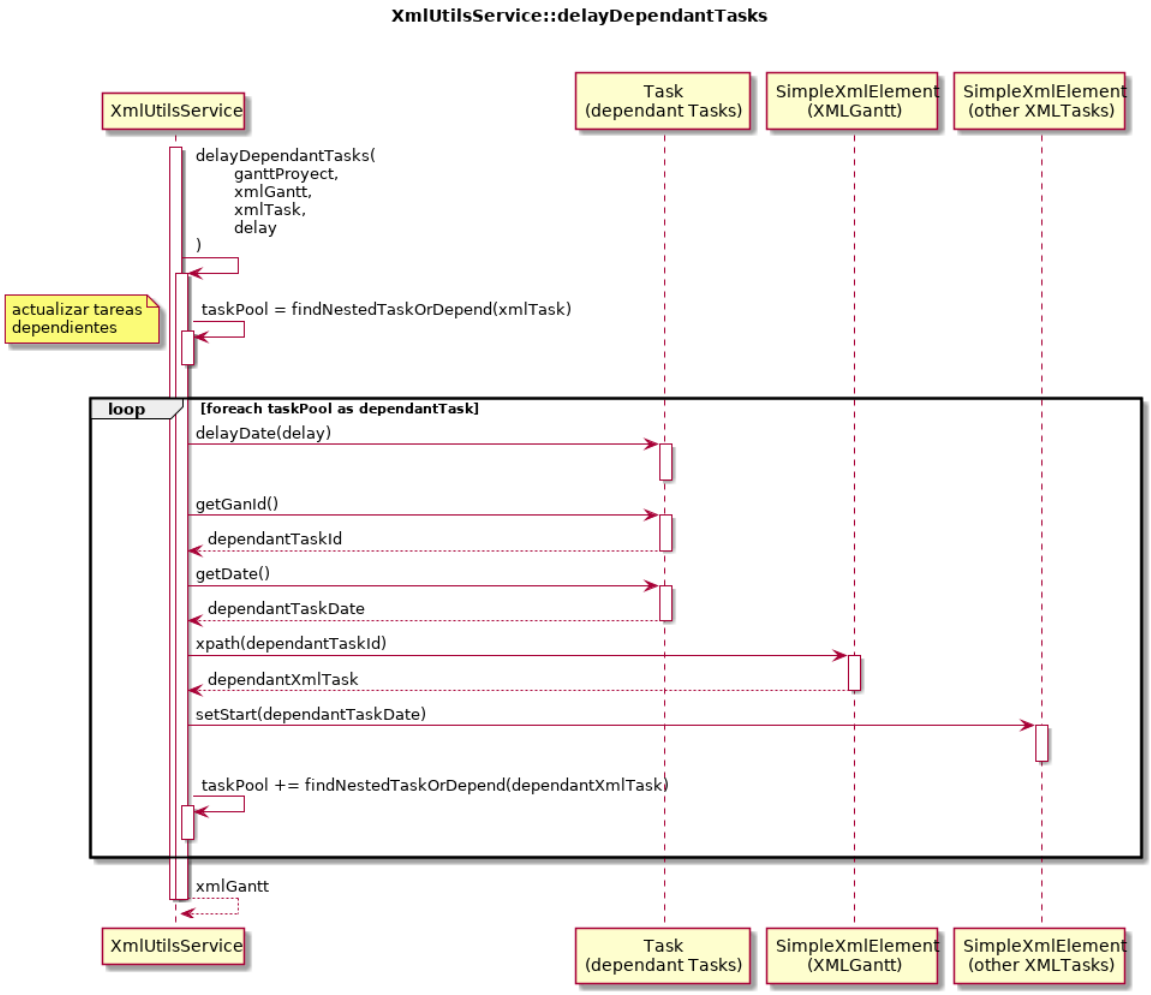

Figura 44: Tarea 4 - Función delayDependantTasks.

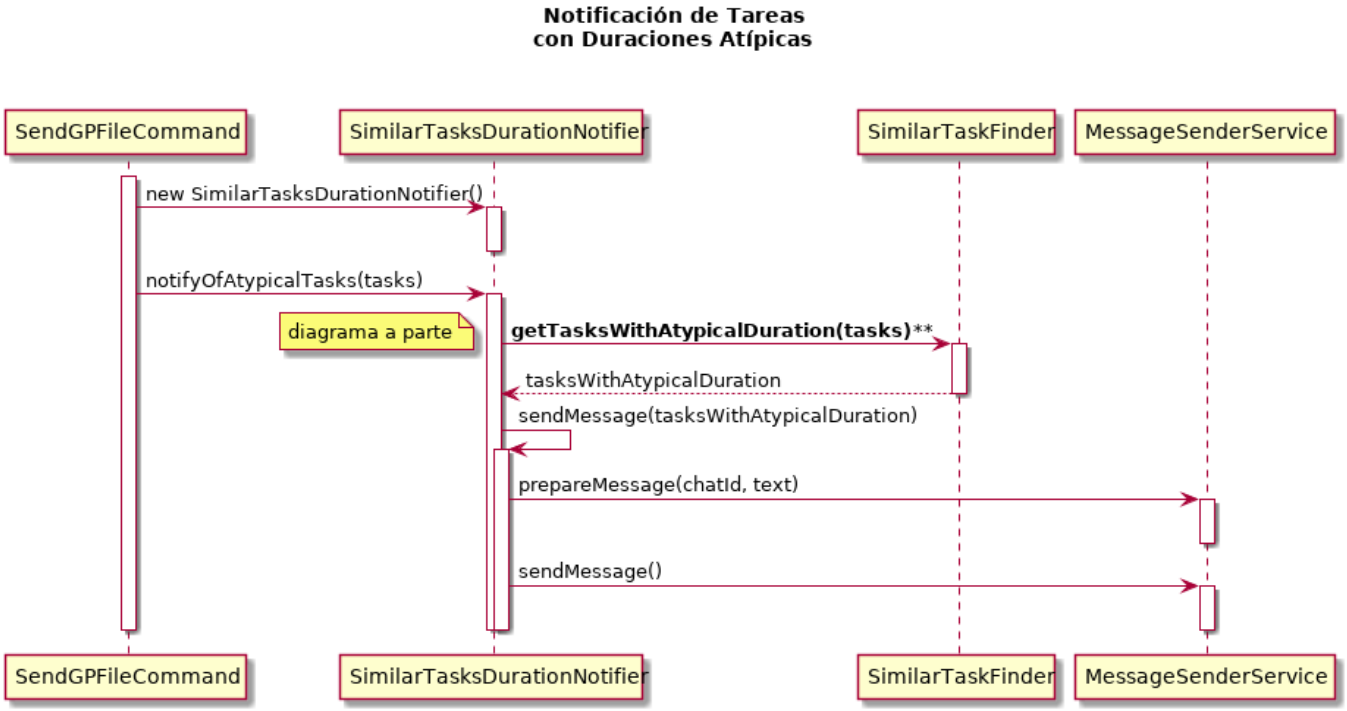

Figura 45: Tarea 5 - Llamada al notificador de las tareas.

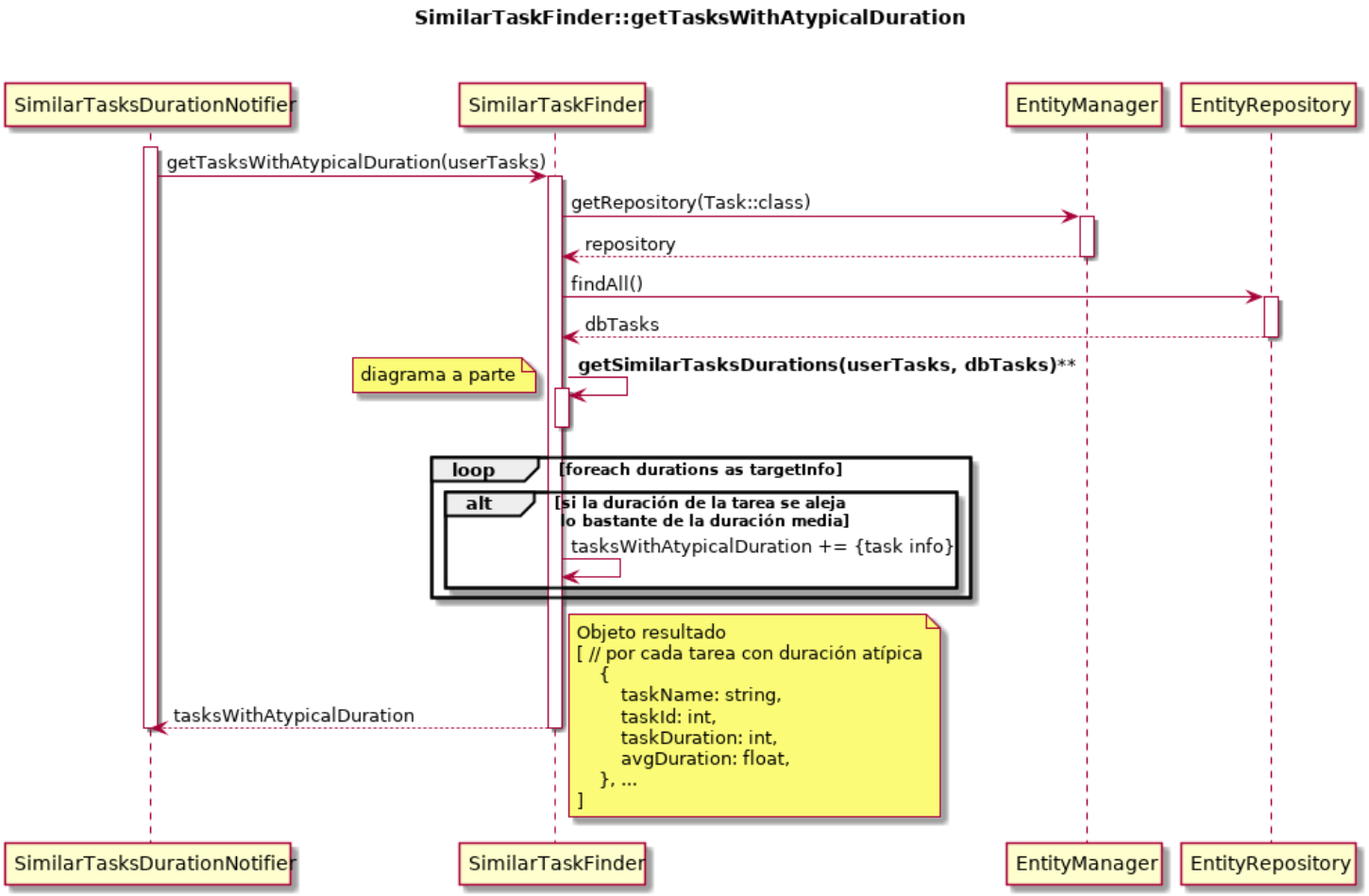

Figura 46: Tarea 5 - Función getTasksWithAtypicalDuration.

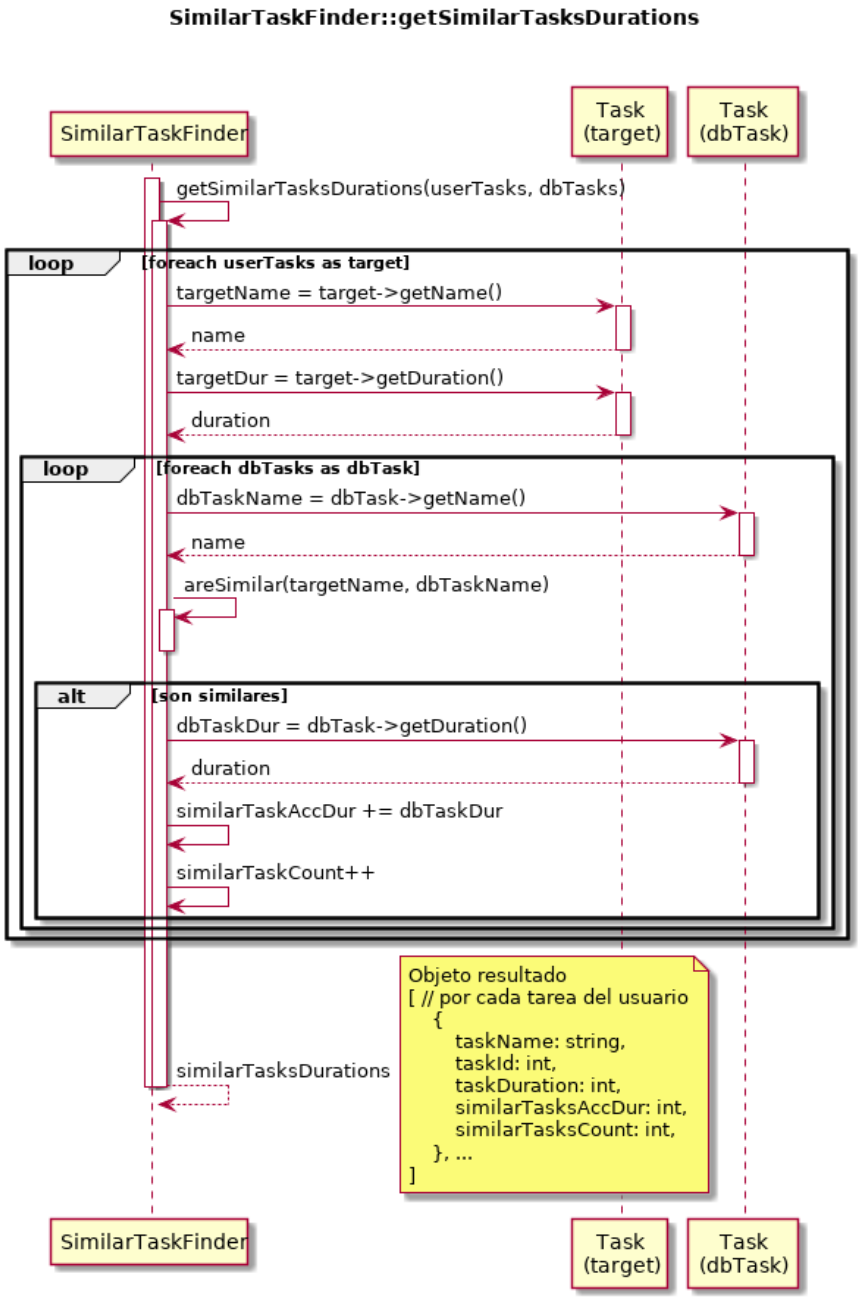

Figura 47: Tarea 5 - Función getSimilarTasksDurations.