Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Proyecto de Fin de Máster

Clepsydra

Mikel Alejo Barcina Ribera
Estudiante del Máster Universitario en Ingeniería Informática
mabarcina001@ikasle.ehu.eus

Dedicado a Manuela y Félix Ángel.

Agradecimientos

Quisiera dar mi más sincero agradecimiento a mi familia, amigos y director de proyecto por ser la gente estupenda y maravillosa que son, por confiar en mí y estar dispuestos a brindarme su apoyo incondicionalmente, y por hacer que los momentos que paso con ellos merezcan la pena siempre.

Quisiera, también, mencionar de forma especial y única a mis queridos padres, los cuales siempre han estado presentes a pesar de todo, brindándome su apoyo, cariño y confianza, e instruyéndome en la vida para llegar a ser quien soy, y estar donde estoy. Me he dado cuenta de que la deuda que he contraído con ellos es tan grande, que no voy a poder pagarla nunca. Sólo espero estar a la altura, como hijo, de lo que ellos se merecen, por el esfuerzo y sacrificio que han hecho conmigo. Por ello y por mucho más, les estaré eternamente agradecido.

Resumen

Clepsydra es el nombre del proyecto de fin de máster realizado por Mikel Alejo Barcina Ribera, alumno del Máster en Ingeniería Informática impartido en la facultad de informática de la universidad del país vasco —UPV/EHU—localizada en Donostia/San Sebastián.

Esta documentación recoge todo lo referente a la concepción, diseño y desarrollo de un producto, y la gestión del proyecto que lo envuelve. La aplicación en sí es un resultado de un desarrollo web que tiene como objetivo ser una *Application Programming Interface* o API, que pueda ser consumida por diferentes capas de presentación que quieran o puedan ser desarrolladas para ella. La finalidad de esta aplicación, es la de poder ofrecer a los gestores y trabajadores de una empresa, las herramientas para poder realizar un seguimiento y control de las actividades que se realizan en los distintos proyectos de la empresa.

Índice general

Índice de figuras

Capítulo 1

Introducción

Se le llama reloj de agua o clepsidra —del griego κλέπτειν *kleptein* "robar", ὕδωρ *hydor*, "agua"—a cualquier reloj que mida el tiempo mediante un flujo regulado de líquido hacia o desde un recipiente. Se considera como el instrumento de medición de tiempo más antiguo de la historia, y aunque no se sabe dónde ni cuando se inventaron, se han encontrado indicios de estos instrumentos en la antigua Babilonia y Egipto —Siglo XVI a.C. —, India y China —incluso algunos autores afirman que apareció en china en el año 4.000 a.C.—. Más adelante los griegos y los romanos mejorarían los relojes de agua introduciendo un ingenioso sistema que aumentaría la precisión de estos instrumentos considerablemente.[41]

1.1. Contexto de la empresa

GuGo Creative S.L. es una empresa emergente localizada en Donostia, la cual centra su actividad en el diseño y desarrollo tanto de soluciones web como de aplicaciones móviles.

1.2. Motivación del proyecto propuesto y objetivos

La empresa utilizaba una aplicación de seguimiento y control de tiempos, que permitía dar de alta clientes, proyectos y tareas, y que recogía las imputaciones de tiempo que los trabajadores dedicaban a cada proyecto. El objetivo era generar un informe para después exportarlo, y mediante un programa externo obtener las rentabilidades de los proyectos y los clientes.

Según se me informó, dicha aplicación la escogieron después de hacer un

estudio de las diferentes alternativas existentes, pero aún y todo, no suplía todas las necesidades específicas de la empresa y además, tenía funcionalidades que no se utilizaban en absoluto.

Por todo ello, se propuso realizar un desarrollo propio, el cual supliera las necesidades que tenía la empresa, y fuera suficientemente extensible o modificable para poder comercializar el producto para otras empresas. También se pretendía que las diferentes capas —capa de presentación, lógica de negocio y acceso a datos —fueran independientes para facilitar el desarrollo de diferentes capas de presentación que pudieran hacer uso de la aplicación: una capa web, móvil o de aplicación de escritorio.

Capítulo 2

Gestión del proyecto

2.1. Gestión del alcance del proyecto

La propuesta del proyecto incluye unas indicaciones generales de cómo quiere la empresa que sea la aplicación, pero son insuficientes para poder realizar un análisis realista del alcance que va a tener el proyecto.

Además, hay que tener en cuenta que el objetivo de este proyecto es realizar una aplicación que funcione de forma cohesionada entre los diferentes departamentos de la empresa, por lo que es muy probable que las requerimientos cambien a lo largo del desarrollo del proyecto, o se tengan que modificar.

2.1.1. Planificación de la gestión del alcance

El alcance del proyecto se generará una vez se conozcan, validen y contrasten todos los requerimientos recogidos. Después, se generará la Estructura de Descomposición de Trabajo (EDT) una vez se disponga de la primera versión del alcance, e incluirá todos los aspectos del Proyecto de Fin de Máster (PFM): planificación, gestión, desarrollo, seguimiento y control y cierre.

El seguimiento y control del alcance se realizará a lo largo del desarrollo del proyecto, revisando también los requerimientos, para evaluar si será necesaria la aplicación de una modificación al alcance y al EDT. Antes de hacer efectiva una modificación se tendrá que evaluar y comprobar que la aplicación de dicha modificación no afecte a la viabilidad del proyecto. Seguidamente se comprobará el EDT y se verificará si es necesario también un cambio en dicha estructura.

2.1.2. Planificación de la gestión de requerimientos

Los trabajadores usan actualmente una aplicación determinada de seguimiento y control de tiempos, clientes y proyectos. Es por ello que es crucial realizar entrevistas con los trabajadores de los diferentes departamentos para extraer de sus respuestas aquellas cosas que les gustan, las que no, y las que mejorarían, además de su experiencia en general usando dicha aplicación. Al procesar estas respuestas se podrán obtener los requerimientos del producto a desarrollar y, finalmente, su validación consistirá en exponer dichos requerimientos ante los interesados del proyecto, dando su visto bueno o no a aquellos puntos con los que estén de acuerdo.

El seguimiento y control de los requerimientos se realizará a lo largo del desarrollo del proyecto, cuando los diferentes interesados de la aplicación contrasten que los requerimientos indicados y las que finalmente han sido implementadas coinciden.

En caso de que los requerimientos tengan que sufrir algún cambio, se evaluará si el nuevo requerimiento, o la modificación de uno ya existente, es viable y se puede llevar a cabo. Se podrá modificar la prioridad e importancia de los diferentes requerimientos, e incluso descartar unos requerimientos por otros. En caso de que los cambios en los requerimientos supongan un cambio drástico en el proyecto, se realizará una reunión con los interesados del proyecto para ratificar dicho cambio.

2.1.3. Recopilación de requerimientos

Tras realizar el proceso descrito en 2.1.2, se validaron los siguientes requerimientos que deberá tener la aplicación, ordenados por importancia:

1. Desarrollar un Back End en forma de Application Programming Interface (API) Representational State Transfer (REST) que permita intercambiar datos con un Front End

2. El Back End ha de proporcionar control de acceso para restringir el contenido visible —ya sea restringiendo el acceso a recursos completos o limitando la información mostrada en recursos específicos —dependiendo de una serie de roles.

3. La aplicación ha de dar la opción de gestionar clientes, y asignarles proyectos y grupos de proyectos a dichos clientes.

4. La aplicación ha de permitir poder gestionar, a parte de los propios proyectos, los presupuestos, bolsas de horas, usuarios, tareas y categorías asociadas.

5. La aplicación ha de desarrollarse de forma que pueda ser fácilmente mantenible y que permita la extensión y mejora en el futuro.

6. Los proyectos de la aplicación deberán tener una estructura común, y los gestores deberán tener la opción de modificar dicha estructura según las necesidades específicas de cada proyecto.

7. La aplicación permitirá crear informes que recopilen datos por proyecto, departamento y por trabajador.

8. La aplicación permitirá crear listas de favoritos, plantillas, o reordenar las tareas por usuario para facilitar la imputación de horas.

9. La aplicación permitirá cerrar y reabrir los proyectos.

10. La aplicación permitirá a los gestores de proyectos imputar horas a terceras personas, por ejemplo en el caso de que haya un trabajador por cuenta propia contratado.

11. La aplicación permitirá enviar recordatorios a aquellos usuarios que se les haya olvidado imputar horas.

12. La aplicación tendrá una integración con Jira

13. La aplicación permitirá crear procesos a los gestores para facilitar la gestión.

14. La aplicación permitirá visualizar de forma sencilla qué personas están de vacaciones.

15. Todos los proyectos tendrán que compartir los mismos colores en las categorías comunes.

16. Una aplicación de escritorio que facilite la imputación de horas y, que entre las funcionalidades estándar de la aplicación, tenga una que ayude a realizar el seguimiento de la tarea que se está realizando en ese momento mediante un botón *play / stop*.

17. La interfaz ha de ser rápida y sencilla de usar para imputar horas de tareas individuales y un grupo de tareas.

18. La interfaz ha de mostrar a cada usuario cuántas horas ha imputado a lo largo de la semana y cuántas le restan por imputar para completar sus horas semanales.

19. La interfaz ha de mostrar sugerencias para las imputaciones: que analice las últimas imputaciones para recomendar y ayudar a imputar cuando se trabaja en las mismas tareas o en los mismos proyectos.

20. La interfaz ha de permitir a cada usuario limitar su vista de horas a la jornada laboral que realiza, ahorrando así el tener que deslizarse hasta la hora de inicio y hora de finalización de la jornada.

21. La interfaz ha de permitir a cada usuario configurar la vista de las imputaciones: diaria, semanal y mensual.

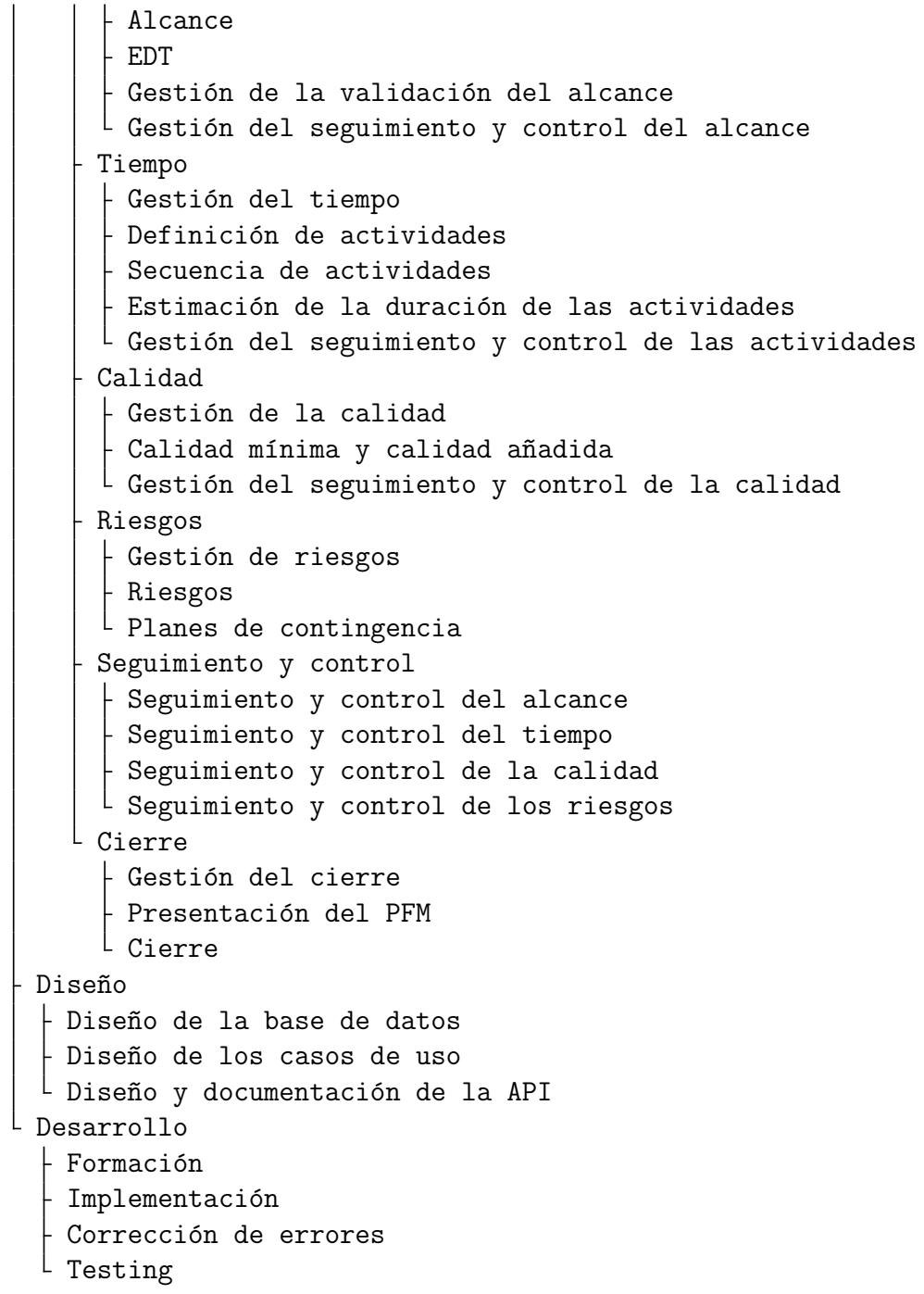
2.1.4. Alcance

El alcance de este proyecto abarca no sólo el desarrollo del producto en sí, sino también la gestión del proyecto que dé como resultado dicho producto. Esto incluye la planificación, diseño y desarrollo del producto, extracción de conclusiones y líneas futuras, realización del seguimiento y control del proyecto y la redacción de la documentación del proyecto.

En cuanto al producto en sí, las secciones de calidad mínima 2.4.2 y calidad añadida 2.4.3, recogen qué requisitos serán tenidos en cuenta a la hora de desarrollar el producto. No obstante, es necesario explicar que los requerimientos recogen especificaciones para el desarrollo tanto de un backend, como de un frontend, y que este proyecto se centra únicamente en el desarrollo del backend. Esto se debe a que en un principio se propuso realizar el proyecto al completo, pero después de recoger los requerimientos y especificar las calidades que debería de tener el producto, consideré que el realizar el desarrollo de las dos partes culminaría en un resultado que haría honor al refrán *"quien mucho abarca, poco aprieta"*. Tras exponer mis dudas a los interesados, se me propuso dividir la propuesta en dos proyectos individuales, de los cuales uno de ellos se me asignó a mí, y el otro proyecto se asignó a otro alumno de la misma facultad donde cursé el máster.

2.1.5. EDT

```
Clepsydra
├ Gestión
│ └ Planificación
│   ├ Alcance
│   │ ├ Gestión del alcance
│   │ ├ Gestión de requerimientos
│   │ ├ Requerimientos
```



2.1.6. Validación del alcance

La validación del alcance se hará junto con los requerimientos del proyecto, en una reunión con los interesados del proyecto. En el caso de que

se propongan alteraciones se tomarán en consideración y se realizarán las modificaciones oportunas acorde con lo expuesto.

En caso de que sea necesario modificar el alcance a lo largo del ciclo de vida del proyecto, se tendrán que validar dichos cambios con los diferentes interesados.

2.1.7. Seguimiento y control del alcance

El seguimiento del alcance se realizará periódicamente, cada Lunes de cada semana, desde la fecha de inicio del desarrollo de la aplicación, comprobando qué requerimientos se han cumplido, cuales faltan por cumplir, y cuales hay que descartar en el caso de que no se dispongan de los suficientes recursos para dedicar a dichos requerimientos.

2.2. Gestión de los interesados

2.2.1. Planificación de la gestión de los interesados

Se realizará una lista con los diferentes interesados en el proyecto, qué rol tienen y cuánto peso tienen en el proyecto. Esta lista se podrá actualizar si se identifica algún interesado más en pla aplicación a lo largo del proyecto.

2.2.2. Interesados en el proyecto

Los interesados que he identificado son los siguientes:

- Los gestores de la empresa.
- Los gestores de proyectos.
- Los usuarios de la aplicación.
- El alumno que se incorporará para desarrollar el frontend.
- Yo mismo.

2.2.2.1. Los gestores de la empresa

Este grupo de interesados son los que han propuesto realizar este proyecto, y el interés que tienen en él es que salga adelante con el objetivo de que les ayude a gestionar mejor la empresa. Debido a que las decisiones que toman afectan al resto de interesados, el peso de su opinión es alto.

2.2.2.2. Los gestores de proyectos

Este grupo de interesados depende de los proyectos que les asignen los gestores de la empresa, y están encargados de gestionar los proyectos en sí: planificarlos, y asignar usuarios y recursos a los mismos. El peso de su opinión se considera medio-alto, ya que la facilidad a la hora de gestionar un proyecto puede afectar significativamente el rendimiento del desarrollo del mismo.

2.2.2.3. Los usuarios de la aplicación

Los usuarios de la aplicación o los trabajadores, es el grupo que va a usar esta aplicación más intensivamente y de forma más restrictiva, ya que serán los que se encarguen únicamente de apuntar todas las tareas realizadas en los proyectos que estén asignados. Por el motivo anteriormente expuesto, el peso de sus opiniones es bajo a la hora de tomar decisiones en la aplicación.

2.2.2.4. El alumno que se incorporará para desarrollar el frontend

Este alumno, debido a que va a realizar el frontend de la aplicación, contará con un peso medio en cuanto a la toma de decisiones, ya que probablemente terminará guiándome en cuanto a los cambios necesarios que requerirá la aplicación para que la capa de presentación sea completa, fácil de usar y utilizable.

2.2.2.5. Yo mismo

Mi interés en este proyecto es muy alto, ya que de él depende que termine mi máster, y también que la empresa termine contando con un producto de calidad que pueda utilizar, modificar y extender en el futuro. Creo que mi opinión en la toma de decisiones tiene un peso alto, ya que al ser el que se va a enfrentar al desarrollo del producto, voy a ser el que actúe como intermediario entre los deseos, ideas y expectativas del resto de los interesados y la realidad del desarrollo, incluyendo sus virtudes y defectos.

2.3. Gestión del tiempo

2.3.1. Planificación de la gestión del tiempo

Una vez se hayan conocido los requerimientos y se hayan realizado las entrevistas oportunas, se procederá a definir la gestión del tiempo del proyecto. No obstante, se evitará invertir demasiado tiempo y esfuerzo en esta parte, debido a que al no haber realizado ningún otro proyecto similar sobre el que

basar las estimaciones, la estimación general del proyecto será imprecisa, incorrecta y estará llena de desviaciones. En cambio, el esfuerzo se focalizará en el apartado de seguimiento y control de esta parte.

2.3.2. Definición de las actividades

1. Crear estructura del documento de LaTeX.
2. Planificación.
    - 2.1. Alcance
        - 2.1.1. Documentar la gestión del alcance.
        - 2.1.2. Documentar la gestión de los requerimientos.
        - 2.1.3. Documentar la gestión de la validación del alcance.
        - 2.1.4. Escribir un borrador con las preguntas a realizar en las entrevistas.
        - 2.1.5. Realizar entrevistas.
            - 2.1.5.1. Realizar entrevista con los trabajadores de desarrollo.
            - 2.1.5.2. Realizar entrevista con los trabajadores de gestión.
            - 2.1.5.3. Realizar entrevista con los trabajadores de contabilidad.
        - 2.1.6. Procesar respuestas y elaborar los requerimientos.
        - 2.1.7. Generar un alcance inicial.
        - 2.1.8. Convocar una reunión con los interesados para validar el alcance.
        - 2.1.9. Corregir los requerimientos y el alcance.
        - 2.1.10. Documentar la gestión del seguimiento y control del alcance.
        - 2.1.11. Generar un EDT.
    - 2.2. Gestión de interesados
        - 2.2.1. Documentar la gestión de interesados.
        - 2.2.2. Identificar y describir a cada interesado del proyecto.
    - 2.3. Tiempo
        - 2.3.1. Documentar la gestión del tiempo del proyecto.
        - 2.3.2. Realizar una definición de actividades.
        - 2.3.3. Analizar y elaborar una secuencia de actividades.
        - 2.3.4. Estimar la duración de las actividades.
        - 2.3.5. Identificar y recoger los hitos.
        - 2.3.6. Realizar un diagrama de Gantt.
        - 2.3.7. Redactar la gestión del seguimiento y control de las actividades.
    - 2.4. Calidad
        - 2.4.1. Documentar la gestión de la calidad del proyecto.
        - 2.4.2. Especificar qué es considerada como calidad mínima y qué como calidad añadida.
        - 2.4.3. Redactar la gestión del seguimiento y control de la calidad.
    - 2.5. Riesgos
        - 2.5.1. Documentar la gestión de los riesgos del PFM.
        - 2.5.2. Analizar y elaborar una lista de posibles riesgos.
        - 2.5.3. Elaborar planes de contingencia.
3. Gestión

3.1. Realizar seguimiento y control del alcance.

3.2. Realizar seguimiento y control del tiempo.

3.3. Realizar seguimiento y control de la calidad.

3.4. Realizar seguimiento y control de los riesgos.

4. Diseño

4.1. Realizar un Modelo entidad-relación del dominio.

4.2. Diseñar y documentar la API.

5. Desarrollo

5.1. Formación

5.1.1. Obtener información de los aspectos básicos de las API.

5.1.2. Aprender los fundamentos del Test Driven Development (TDD).

5.1.3. Aprender a usar el Web Framework *Symfony* [1] con APIs.

5.1.4. Recavar información sobre las diferentes formas de autenticación con una API REST.

5.2. Implementación

5.2.1. Preparar el entorno para la instalación del Web Framework.

5.2.2. Instalar el Web Framework y probar que funciona.

5.2.3. Instalar los módulos necesarios para proveer al Web Framework de las herramientas necesarias para desarrollar la API.

5.2.4. Comenzar con el desarrollo del producto teniendo en cuenta el Modelo entidad-relación, mediante la metodología TDD.

2.3.3. Secuencia de actividades

Las dependencias entre actividades vienen marcadas por el orden en el que se han definido en la sección anterior, con la excepción de la parte de 3. gestión, la cual se puede ir realizando según vaya avanzando el proyecto.

Lo mismo ocurre con el apartado de 4. diseño, el cual a pesar de ser realizado antes de comenzar con la implementación del producto, es probable que se desarrolle junto con el producto, sobre todo la actividad de 4.2. Diseñar y documentar la API.

2.3.4. Estimación de la duración de las actividades

Actividad	Estimación en minutos	Estimación en horas
1.	720	12
2.	1215	20,25
2.1.	615	10,25

[1] https://symfony.com/

2.1.1.	30	0,5
2.1.2.	30	0,5
2.1.3.	30	0,5
2.1.4.	30	0,5
2.1.5.	270	4,5
2.1.5.2.	90	1,5
2.1.5.2.	90	1,5
2.1.5.3.	90	1,5
2.1.6.	60	1
2.1.7.	30	0,5
2.1.8.	30	0,5
2.1.9.	30	0,5
2.1.10	30	0,5
2.1.11.	45	0,75
2.2.	60	1
2.2.1.	30	0,5
2.2.2.	30	0,5
2.3.	330	5,5
2.3.1.	30	0,5
2.3.2.	60	1
2.3.3.	30	0,5
2.3.4.	60	1
2.3.5.	30	0,5
2.3.6.	60	1
2.3.7.	60	1
2.4.	105	1,75
2.4.1.	30	0,5
2.4.2.	45	0,75
2.4.3.	30	0,5
2.5.	105	1,75
2.5.1.	30	0,5
2.5.2.	45	0,75
2.5.3.	30	0,5
3.	720	12
3.1.	180	3
3.2.	180	3
3.3.	180	3
3.4.	180	3

4.	300	5
4.1.	60	1
4.2.	240	4
5.	22020	367
5.1.	720	12
5.1.1.	120	2
5.1.2.	120	2
5.1.3.	300	5
5.1.4.	180	3
5.2.	21300	355
5.2.1.	60	1
5.2.2.	180	3
5.2.3.	60	1
5.2.4.	21000	350
Horas totales		416,25

2.3.5. Hitos

- Inicio del proyecto: 2017-11-13
- Inicio de la fase de desarrollo: 2017-12-01
- El segundo integrante se incorpora: 2018-01-15
- Fin de la fase de desarrollo: 2017-03-28

2.3.6. Diagrama de Gantt

El diagrama de Gantt, debido a su longitud, se ha dividido en tres diagramas más pequeños. Es un diagrama muy sencillo y con una granularidad muy grande con el objetivo de ser una guía genérica, junto con las actividades y los hitos recogidos, para encaminar el proyecto dentro de unas fechas y tiempos específicos. Debido a la longitud del proyecto, el diagrama se ha dividido en tres partes: la que recoge la fase inicial del proyecto, en la figura 2.1, la que muestra otro de los hitos importantes del proyecto hacia la mitad del mismo, en la figura 2.2, y finalmente la parte que muestra el fin del proyecto, en la figura 2.3.

2.3.7. Seguimiento y control

Por los motivos especificados en la sección de planificación 2.3.1, el esfuerzo de la gestión de tiempo del proyecto se realizará en este apartado. La idea es recoger todas, o la gran mayoría de las actividades realizadas a lo largo del proyecto, de forma que después se puedan elaborar conclusiones sobre qué es importante estimar y qué no, en un proyecto de estas características.

2.4. Gestión de calidad

2.4.1. Planificación de la gestión de la calidad

La calidad del producto se definirá mediante dos grupos: la calidad mínima y la calidad extra. Los elementos de calidad mínima serán aquellos que son indispensables para poder realizar un cierre satisfactorio y exitoso del proyecto. Por contra, los elementos que se agrupen dentro de la calidad extra, serán aquellos a los que se les dedicarán recursos siempre y cuando la calidad mínima en el producto haya sido conseguida.

En el caso de que se propongan nuevas ideas, características o requerimientos para el producto final, se tendrá que hacer una evaluación de dichas características para decidir en qué grupo habrá de clasificarla.

Por último, si uno de los elementos considerados de calidad mínima no puede ser completado o desarrollado, se estudiará la posibilidad de moverlo al grupo de elementos de calidad extra, siempre teniendo en cuenta la opinión de los interesados del proyecto.

2.4.2. Calidad mínima

Los elementos que se incluirán en el producto, y que se consideran que son indispensables para el mismo, son los elementos del 1 al 10, especificados en la sección de requerimientos 2.1.3.

Además, se hará hincapié en las pruebas del producto o *t*esting , con el objetivo de que el producto pueda ser considerado estable y robusto, y que sobre todo asista en el desarrollo a la hora de modificar el susodicho producto, indicando qué partes funcionan y cuales no a la hora de introducir los cambios.

2.4.3. Calidad añadida

Los elementos de calidad añadida para el producto son los elementos 11, 12 y 13 especificados en la sección de requerimientos 2.1.3.

2.4.4. Seguimiento y control de la calidad

El seguimiento y control de la calidad se irá haciendo en conjunción con el seguimiento y control de los requerimientos 2.1.7, el alcance 2.1.7 y las actividades 2.3.7, y dependiendo del estado general del conjunto del proyecto se tomarán las decisiones oportunas.

2.5. Gestión de riesgos

2.5.1. Planificación de la gestión de riesgos

Los riesgos se identificarán y recogerán en una lista, y posteriormente se elaborarán planes de contingencia para lidiar con los problemas que los riesgos puedan plantear en un futuro.

2.5.2. Identificación de riesgos

- No cumplir con la calidad mínima especificada.

A pesar de que se tiene noción sobre las tecnologías que se verán involucradas en el producto, nunca se ha realizado ningún desarrollo similar en un entorno real con proyección a comercializar el producto. Es por ello que puede resultar que las estimaciones realizadas sean insuficientes y que no se satisfaga el alcance definido. Este riesgo por lo tanto se considera un riesgo alto.

- No tener un producto mínimo que sea funcional cuando el otro integrante comience a desarrollar su parte.

Relacionado con el riesgo anterior, puede darse la situación en la que el producto no sea mínimamente funcional cuando el otro integrante comience su desarrollo. Este riesgo se clasifica como de nivel medio.

2.5.3. Planes de contingencia

- No cumplir con la calidad mínima especificada.

A la hora de desarrollar, se priorizarán aquellas funcionalidades y especificaciones que tengan que ver con los requerimientos, y si aún y todo el tiempo para cumplir con la calidad mínima empieza a escasear, se centrará el desarrollo en dichas funcionalidades.

Si el problema es debido a la falta de recursos, se valorará la aceptación del producto desarrollado con los interesados del producto. En caso afirmativo, se

dará por concluido el mismo y se aceptará el cierre de la parte de desarrollo del proyecto. En caso negativo, el proyecto se cerrará y se marcará como inacabado.

- No tener un producto mínimo que sea funcional cuando el otro integrante comience a desarrollar su parte.

En este caso se abandonará cualquier otro trabajo para centrarse en proporcionar al otro integrante un producto funcional con el que pueda probar su desarrollo.

2.5.4. Seguimiento y control de la calidad

El seguimiento y control de los riesgos se realizará a lo largo del proyecto, y se recogerán las incidencias, si hubiere, en un apartado que indique la aplicación de los planes de contingencia propuestos y la forma en la que se han llevado a cabo.

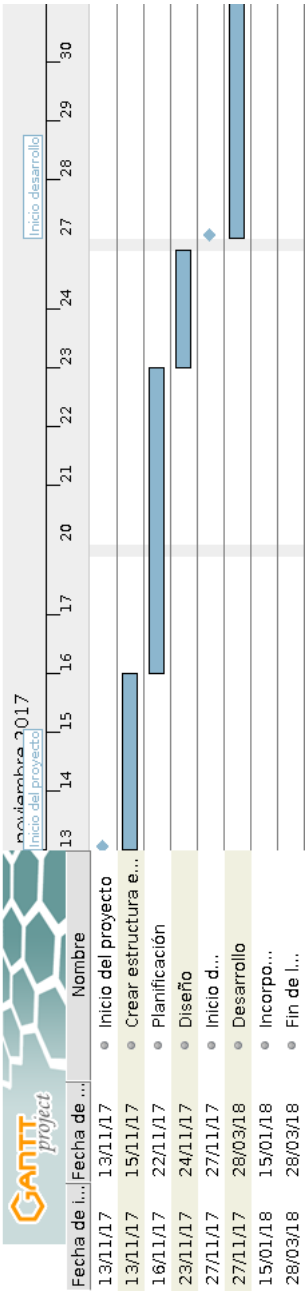


Figura 2.1: Diagrama de Gantt correspondiente a los hitos iniciales del proyecto

Figura 2.2: Diagrama de Gantt correspondiente a los hitos hacia la mitad del proyecto

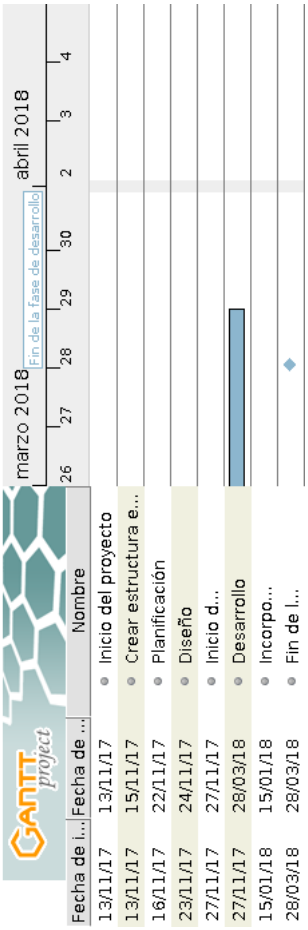


Figura 2.3: Diagrama de Gantt correspondiente de los hitos finales del proyecto

Capítulo 3

Diseño y desarrollo del producto

3.1. Tecnologías involucradas

3.1.1. Asunciones previas

Como se ha indicado en la sección del contexto de la empresa 1.1, su especialidad son los desarrollos web y de aplicaciones móviles. Es por ello que desde el principio se habló de que el desarrollo de la API, sería un desarrollo web.

3.1.1.1. Tecnología propuesta

Debido a que la empresa cuenta con trabajadores que ya realizaron proyectos con una tecnología específica, se me pidió que usara la susodicha tecnología para así facilitar la modificación, adaptación o corrección de errores una vez el proyecto estuviera finalizado. La tecnología propuesta, por lo tanto, fue Symfony.

3.1.2. Frameworks utilizados en la aplicación

3.1.2.1. Symfony

Symfony es un framework escrito en PHP: Hypertext Preprocessor (PHP) y organizado en *bundles* o paquetes. De hecho, una de las características de Symfony es que absolutamente todo se organiza en paquetes. Incluso el propio framework, es un paquete. Otro de los puntos fuertes de Symfony es que gracias a cómo ha sido desarrollado, fuerza a programar siguiendo buenas prácticas y patrones conocidos, que hacen que la calidad del código aumente.

3.1.2.2. API Platform

API Platform es a su vez otro framework que está diseñado especialmente para funcionar sobre Symfony, y que facilita la creación de APIs. Una de sus características más llamativas es que una vez definidas las entidades del dominio en clases PHP, API Platform genera automáticamente operaciones Create, Read, Update, Delete (CRUD). Otras de las funcionalidades destacables que tiene, son entre otras:

- Soporte para diferentes formatos: XML, JSON, CSV y YAML.
- Soporte para la web semántica: JSON-LD y HAL.
- Soporte para APIs Hypermedia As The Engine Of Application State (HATEOAS).
- Generación automática de documentación para las operaciones de la API mediante Swagger UI.
- Paginación de resultados.
- Filtros de resultados.

3.1.3. Librerías destacables utilizadas en la aplicación

Las dependencias de librerías externas en PHP se suelen gestionar con *Composer*: este programa utiliza un fichero JSON que va llevando cuenta de las dependencias requeridas y en qué versión se instalaron, de forma que los proyectos no necesitan incluir dichas librerías cuando se despliegan o cuando los diferentes miembros que contribuyen en él hacen acopio del código fuente. Con un simple comando, *Composer* leerá las dependencias requeridas y las descargará desde el repositorio localizado en `https://packagist.org/`. El formato en el que se recogen las librerías sigue el patrón *vendor/package*, y es el que utilizaré para describir algunas de las librerías más interesantes utilizadas en el proyecto.

3.1.3.1. api-platform/core

El núcleo de API Platform. Esta librería es la que facilita las funcionalidades indicadas en la sección de frameworks 3.1.2.2 .

3.1.3.2. symfony/http-foundation

Define una capa orientada a objetos para la especificación HTTP [24], la cual tiene como finalidad englobar las variables y funciones que PHP utiliza para gestionar y manipular las peticiones recibidas y las respuestas generadas[25]. Por ejemplo, en el siguiente bloque de código, se muestra cómo las variables globales —como *$_GET* o *$_POST*—que en un principio habría que gestionar individualmente, se agrupan en cambio en un único objeto de tipo *Request* mucho más manejable.

```php
<?php

use Symfony\Component\HttpFoundation\Request;

$request = Request::createFromGlobals();

$request = new Request(
    $_GET,
    $_POST,
    array(),
    $_COOKIE,
    $_FILES,
    $_SERVER
);
```

La librería también permite acceder a la sesión, a las cabeceras enviadas, enviar una respuesta, establecer cookies, redireccionar al usuario y múltiples funcionalidades más.

3.1.3.3. symfony/http-kernel

Esta librería se encarga de transformar la petición del usuario recibida en una respuesta. Para ello, se sirve de el sistema de eventos que tienen implementado —*symfony/event-dispatcher*—el cual va disparando varios eventos en cada fase de un proceso que se ilustra en la siguiente ilustración:

Cada número representa un paso del proceso, y algunos de estos pasos dispararán eventos. Al igual que existen eventos, también existen suscriptores a dichos eventos, que básicamente esperan a que un evento de un tipo determinado se dispare para realizar una serie de acciones.

1. El evento *kernel.request* tiene como objetivo añadir más información a la petición —por ejemplo después de determinar en qué idioma debería de darse la respuesta—, inicializar partes del framework o crear

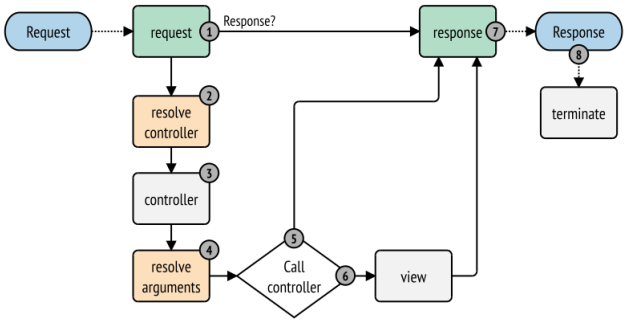


Figura 3.1: Diagrama del proceso de transformación de una petición a una respuesta.

Fuente:
https://symfony.com/doc/current/components/http_kernel.html

una respuesta inmediatamente y finalizar el proceso: como por ejemplo cuando el usuario no tiene la autorización pertinente.

2. El siguiente paso es averiguar qué controlador ha de llamarse para que se ejecute la lógica de negocio acorde a esa respuesta. Esto se determina gracias a la información que se extrae de la petición, y a la que el programador proporciona al framework en forma de ciertas directrices como puede ser emparejar una ruta con un controlador. Ejemplo: *C̈uando una petición se realice a la ruta /hello/world, se ejecutará la función HelloWorld() del controlador HelloWorldController".*

3. El evento *kernel.controller* tiene como objetivo preparar la ejecución del controlador, ya sea inicializando ciertas partes del framework o incluso permitiendo que un suscriptor cambie el controlador que se va a ejecutar.

4. En este punto se determinan los argumentos y los valores que han de pasarse al controlador. Las formas más comunes de determinarlo suelen ser mediante un identificador en la ruta —*/hello/world/argumento1/argumento2*—, la indicación que el programador da para obtener el argumento y finalmente pasando directamente la petición al controlador, donde el programador la procesará y extraerá los argumentos que le interesen.

5. Aquí se llama al controlador y se aplica la lógica de negocio pertinente según lo que el programador haya programado. En el caso de que el

controlador devuelva una respuesta, directamente se salta al último paso. En caso contrario, se realiza un paso más.

6. El evento *kernel.view* transforma un resultado de un controlador en una respuesta. Aquí es donde suele entrar en juego la capa de visualización. Por ejemplo: el controlador devuelve una serie de valores que después se insertan en la página correspondiente que haya que devolver al usuario, como puede ser el nombre de usuario del usuario que realizó la petición.

7. Finalmente se dispara el evento *kernel.response*, que permite a los distintos suscriptores modificar la respuesta antes de que sea enviada al usuario.

3.1.3.4. symfony/framework-bundle

La librería encargada de recoger toda la configuración de los diferentes recursos y elementos que componen el framework: las sesiones, los formularios, validación, enrutamiento[18], gestión de la capa de visualización, traducciones, cache...

3.1.3.5. symfony/security-bundle

Librería encargada de proporcionar control de acceso a los diferentes recursos que componen la aplicación. La configuración se realiza mediante parámetros y patrones de rutas.

3.1.3.6. doctrine/orm

Esta librería provee de Object Relational Mapping (ORM) y Database Abstraction Layer (DBAL) para PHP. Para la escritura de consultas se utiliza la sintaxis Doctrine Query Language (DQL). He aquí un ejemplo de una consulta utilizando DQL y el *Query Builder* de Doctrine:

```
1 <?php
2
3 $queryBuilder->select('h')
4              ->from('house', 'h')
5              ->where('h.stories > 2')
6              ->orderBy('h.stories', 'ASC');
7
```

3.1.3.7. doctrine/annotations

Esta librería permite realizar anotaciones en las entidades del dominio para ayudar después a mapear cada atributo y entidad con su tabla y columnas correspondientes en la base de datos. Un ejemplo de una entidad mapeada podría ser el siguiente:

```php
<?php

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 */
class House
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $name;

    /**
     * @ORM\Column(type="integer")
     */
    private $stories;
}
```

3.1.3.8. symfony/validator

Esta librería facilita la validación de los diferentes atributos de las entidades. La forma de especificación de las restricciones a las cuales están sujetas, es mediante anotaciones. Así que continuando con el ejemplo anterior:

```php
<?php

```

```php
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Validator\Constraints as Assert;

/**
 * @ORM\Entity
 */
class House
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     *
     * @Assert\NotBlank()
     * @Assert\Length(min = 1, max = 255)
     */
    private $name;

    /**
     * @ORM\Column(type="integer")
     *
     * @Assert\GreaterThanOrEqual(0)
     */
    private $stories;
}
```

En este simple ejemplo se ha hecho que el atributo *"name"* tenga que tener una longitud comprendida entre 1 y 255 caracteres, y que el atributo *"stories"* deba ser mayor o igual que 0. Al quedar las restricciones plasmadas en la propia entidad, éstas se comprueban para cualquier acceso a dicha entidad. Por ejemplo, la entidad que se ha usado de ejemplo podría modificarse mediante un formulario web o una petición a una API, pero las restricciones serán las mismas se modifique de una u otra forma. Huelga decir que si se requiriese lo contrario, se podría hacer de forma que en el formulario web se apliquen ciertas restricciones mientras que las llamadas a la API podrían tener restricciones distintas.

3.1.3.9. symfony/serializer

El serializador es un componente que se utiliza para transformar objetos en formatos específicos, como pueden ser: JSON, XML, YAML[40]... En el siguiente diagrama se ilustra el proceso:

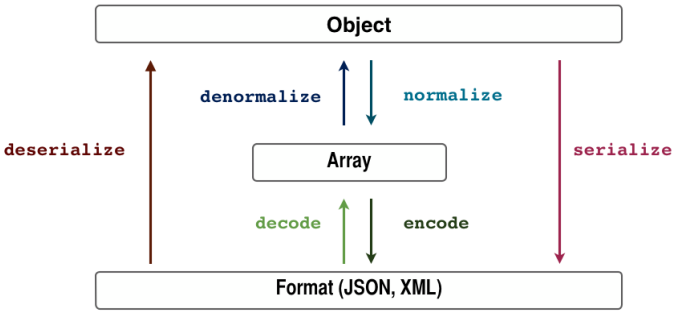


Figura 3.2: Diagrama del proceso de serialización de un objeto
Fuente:
https://symfony.com/doc/current/components/serializer.html

Además, mediante el uso de anotaciones, se pueden crear grupos de serialización, de forma que dependiendo de la petición recibida se pueden serializar distintos atributos. En el siguiente ejemplo se crean tres grupos, *private*, *public* y *edit*:

```
1  <?php
2
3  use Doctrine\ORM\Mapping as ORM;
4  use Symfony\Component\Serializer\Annotation\Groups;
5  use Symfony\Component\Validator\Constraints as Assert;
6
7  /**
8   * @ORM\Entity
9   */
10 class House
11 {
12     /**
13      * @ORM\Id()
14      * @ORM\GeneratedValue()
15      * @ORM\Column(type="integer")
16      *
17      * @Groups({"private"})
18      */
```

```
19      private $id;
20
21      /**
22       * @ORM\Column(type="string", length=255)
23       *
24       * @Assert\NotBlank()
25       * @Assert\Length(min = 1, max = 255)
26       *
27       * @Groups({"public", "edit"})
28       */
29      private $name;
30
31      /**
32       * @ORM\Column(type="integer")
33       *
34       * @Assert\GreaterThanOrEqual(0)
35       *
36       * @Groups({"public"})
37       */
38      private $stories;
39  }
```

Habiendo etiquetado cada atributo en uno o más grupos de serialización, esto después puede utilizarse para gestionar de forma dinámica la entidad dependiendo de las peticiones. Por ejemplo, cuando se hiciera únicamente una operación de lectura por parte de un cliente sin privilegios, la entidad podría ser serializada con el grupo *public*, dejando así el atributo *id* fuera de la respuesta a dar. Lo mismo aplica para los grupos *edit* que podría utilizarse para limitar las modificaciones a un atributo en concreto, o el grupo *private* si la casa está siendo gestionada por un administrador. Los grupos de los nombres en este caso son esos, pero pueden llamarse como se quiera.

3.1.3.10. behat/behat

Un framework Behavior-Driven Development (BDD) que permite realizar pruebas funcionales mediante una sintaxis denominada *Gherkin*, que facilita la redacción y lectura de las pruebas. Dichas pruebas se organizan en funcionalidades, escenarios y pasos que después el framework se encarga de ejecutar. Un ejemplo de una prueba funcional con esta librería podría ser el siguiente:

```
Feature: House management
  In order to manage houses
  As a house owner
  I need to be able to add a house, modify its parameters or remove it

  Scenario: Modify a house
    Given there exists a house named "My fancy house"
    And I am the owner of it
    When I modify its name to "My beautiful house"
    Then its new name should be "My beautiful house"
```

Esta definición de la prueba funcional en *Gherkin* después se mapea a una clase PHP que interpreta y ejecuta lo que en cada paso debe de comprobarse:

```
<?php

class FeatureContext
{
    /**
     * @Given there exists a house named :house
     */
    public function thereExistsHouseNamed($house)
    {
      if(!exists_house($house)) {
        throw new Exception(
          'The house ' . $house . ' doesn't exist'
        );
      }
    }

    /**
     * @Given I am the owner of it
     */
    public function iAmTheOwnerOfTheHouse()
    {
      /** Do stuff */
    }

    /**
     * @When I modify its name to :newName
     */
```

```
28      public function iModifyItsNameTo($newName)
29      {
30        /** Do stuff */
31      }
32
33      /**
34       * @Then its new name should be :expectedName
35       */
36      public function theNewNameShouldBe($expectedName)
37      {
38        /** Do stuff */
39      }
40  }
```

Este ejemplo se ha simplificado para facilitar la asimilación del concepto detrás de *Behat*. Una vez hechos los mapeos correctamente *Behat* se encargará de ejecutar las funciones, y en caso de que no produzcan ningún error los pasos correspondientes se considerarán como correctos.

3.1.3.11. justinrainbow/json-schema

Librería utilizada para validar objetos JSON con *JSON Schema*. Por ejemplo, imaginemos que al realizar una petición al servidor para obtener información sobre una entidad *House*, dicho servidor devuelve lo siguiente:

```
1  {
2    "id": 1,
3    "name": "My beautiful house",
4    "stories": 2
5  }
```

A la hora de realizar las pruebas de la aplicación, la información que devuelva el servidor puede ser variada, a pesar de que el entorno de pruebas cargue una serie de casas predefinidas. En vez de validar que el JSON que devuelve el servidor es exactamente igual a las diferentes casas que existen en la base de datos —porque recordemos que en vez de devolver un único objeto *House*, el servidor podría devolver un *array* de ellos —, se puede comprobar que el objeto JSON devuelto es válido acorde a un esquema. Por ejemplo:

```
1  {
2    "type": "object",
3    "properties": {
```

```
    "id": {"type": "integer"},
    "name": {
      "type": "string",
      "pattern": "^[a-zA-Z0-9]+$"
    },
    "stories": {
      "type": "integer",
      "minimum": 0
    }
  },
  "required": ["id", "name"]
}
```

Este esquema, que ha obviado algunos elementos para simplificar el ejemplo, podría utilizarse para validar las diferentes respuestas que puede dar el servidor con respecto a diferentes casas. Esto puede resultar muy interesante para reducir el tamaño de las pruebas y concentrar en un único esquema las condiciones del objeto JSON que el servidor debería de enviar.

3.1.3.12. lexik/jwt-authentication-bundle

Esta librería provee de una implementación de los JSON Web Token (JWT): Esta tecnología es un estándar abierto[1] que define una forma compacta y autocontenida —debido a que el token tiene toda la información del usuario y no es necesario volver a consultar a la base de datos —de intercambiar información entre dos entidades mediante un objeto JSON. Además, la información puede ser verificada debido a que se firma digitalmente bien utilizando un secreto, bien utilizando claves públicas y privadas. [27]

Como se ha comentado anteriormente, JWT tiene una naturaleza compacta y autocontenida. Esto es realmente útil, por ejemplo, en arquitecturas REST donde no hay estados: debido a que cada llamada a una API REST debe de contener toda la información necesaria para que el servidor pueda procesarla correctamente, los tokens JWT son extremadamente útiles para transmitir, por ejemplo, la información respecto al usuario que está realizando dicha llamada. De esta forma el servidor puede extraer todos los datos del token para comprobarlos y autorizar o no el procesamiento de la petición realizada. He aquí un diagrama explicando el proceso de inicio de sesión:

Tal y como se ve en el diagrama, el token JWT se envía adjuntado en cada petición, en una de las cabeceras de la misma: la cabecera *Authorization*.

[1] https://tools.ietf.org/html/rfc7519

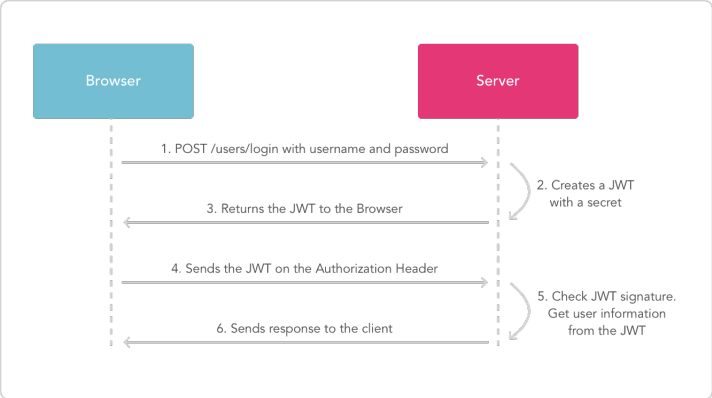


Figura 3.3: Diagrama del proceso de inicio de sesión mediante un token *JWT*. Fuente: https://jwt.io/introduction/

3.1.3.13. api-platform/api-pack

Este es un metapaquete el cual agrupa otras librerías que dan la funcionalidad necesaria para incorporar API Platform en una instalación estándar de Symfony. Entre otras librerías, incluye *api-platform/core*, que es la librería que al fin y al cabo provee de las funcionalidades indicadas en la sección de frameworks 3.1.2.2.

3.2. Clepsydra, la aplicación

3.2.1. Arquitectura de la aplicación

3.2.1.1. Servidor

El servidor donde la aplicación iba a ser probada para su funcionamiento era una máquina virtual GNU
Linux que iba sobre una capa de virtualización Kernel Virtual Machine (KVM). Básicamente la empresa ya tenía KVM montado y simplemente tuve que configurar dicho servidor, adhiriéndome a los parámetros que el informático de sistemas me facilitó.

3.2.1.2. Sistema operativo

El sistema operativo que escogí para montar el servidor fue Debian. El motivo es que he utilizado mucho la susodicha distribución, y por lo tanto pensé que sería mucho más fácil configurar una máquina utilizando una tecnología que ya conocía. Debido a que en la empresa estaban de acuerdo con

ello, fue finalmente la que utilicé.

La distribución se clasifica principalmente en tres ramas: *stable*, *testing* y *unstable*, y debido a que quería ahorrarme quebraderos de cabeza en el futuro escogí la rama estable de la distribución.

No obstante, debido a su naturaleza estable, algunos de los paquetes que requería —principalmente PHP —no estaban en la versión que ciertas librerías de la aplicación requerían, por lo que realicé un pequeño ajuste con lo que se denomina *apt-pinning*.

apt-pinning se realiza cuando se precisa obtener versiones de paquetes, o paquetes enteros, que no están en la rama de la distribución actual. De esta forma se pueden obtener paquetes determinados en una versión más moderna, permitiendo así tener las virtudes de un sistema en un estado estable junto con las modernidades específicas de ciertos paquetes.

Esto se realizó después de comprobar que la primera opción recomendada, los backports para los paquetes necesarios, no estaban disponibles por aquel momento.

3.2.1.3. Base de datos

Para la base de datos de la aplicación se utilizó *MariaDB*, que es un fork del proyecto de base de datos relacional *MySQL*. No se concibió utilizar una tecnología de base de datos diferente debido a que no se esperaba que la aplicación fuera a gestionar un volumen y un tráfico de datos que no fuera fácilmente manejable por una base de datos relacional. No merecía la pena sacrificar ninguno de los principios Atomicity Consistency Isolation Durability (ACID), ni tampoco un Relational Database Management System (RDBMS) que proveyera del cumplimiento de los mismos, por una hipotética necesidad de rendimiento y escalabilidad que fuera a requerir de un paradigma de base de datos diferente.

El diagrama del modelo entidad-relación que usa la aplicación se puede encontrar en la figura 3.4. La idea general es que la aplicación podrá gestionar clientes, que a su vez agruparán proyectos en grupos de proyectos. Cada proyecto puede categorizarse, y además puede tener bolsas de horas —una especie de presupuesto en la que el cliente consume horas de la susodicha bolsa de horas —y presupuestos. Los proyectos predefinidos tienen como fin el ser una plantilla de la que copiar los proyectos que son comunes a distintos grupos de proyectos. Los usuarios pueden asignarse a proyectos, y además son los que crean las diferentes tareas en cada proyecto. Las tareas favoritas tienen el mismo objetivo que los proyectos predefinidos: facilitar la creación de tareas que se repiten obteniéndolas de una lista previamente creada.

Este esquema se creó a raíz de las diferentes reuniones que se mantuvieron

con los interesados, en las cuales se debatió el modelo de gestión que tenían decidido utilizar, y que dieron como fruto el esquema de la figura 3.4 que solventa y refleja las necesidades que se me expusieron.

Además, se puede observar que muchas de las entidades únicamente tienen un par de campos: el identificador y el nombre. Esto es consecuencia de haber consultado con los gestores sobre la adición de nuevos campos en las entidades, que resultaran útiles para definir dichas entidades aún mejor. Pero en el software que utilizaban no les daban un uso significativo, y tampoco preveían que les fueran a dar más uso en esta aplicación. Por lo tanto decidí prescindir de ellos e invertir el tiempo en otras tareas, pero manteniendo la estructura del dominio de forma que si nuevos campos tuvieran que ser añadidos en el futuro, algo que sí preveía probable, se pudiera hacer sin tener que reestructurar el modelo.

3.2.1.4. Sistema de control de versiones y despliegue

Para el control de versiones, se utilizó *git* —ya que era el sistema de control de versiones que mejor sabía utilizar—en una instancia de GitLab, con el esquema de ramas ilustrado en la figura 3.5.

Este esquema de ramas es una versión simplificada basada en *"A successful Git branching model"*, de *Vincent Driessen*: `https://nvie.com/posts/a-successful-git-branching-model/`. La idea era empezar con un commit inicial en la rama *máster*, e ir añadiendo commits en la rama *develop*. Cada nueva funcionalidad se desarrollaría creando una nueva rama a partir de la rama *develop*, y una vez finalizada, se fusionaría la rama *feature* con la rama *develop*. Este proceso se repetiría hasta que llegase el despliegue de la primera versión estable, cosa que se haría fusionando la rama *develop* con la rama *master*.

Los despliegues y la ejecución de las pruebas de la aplicación se habían automatizado utilizando las herramientas de Continuous integration (CI) / Continuous delivery (CD) que proporciona *GitLab*. Tras cada push, se ejecutaban las pruebas de la aplicación. Cuando una funcionalidad se fusionaba en la rama *develop*, además de las pruebas, se desplegaban los cambios automáticamente al servidor de pruebas que el otro alumno utilizaba para desarrollar la aplicación. Faltó por configurar un despliegue a un servidor de producción cuando se fusionaba la rama *develop* en *master*, pero debido a que hasta el final del desarrollo no se hizo ninguna fusión, ni tampoco estaba el servidor de producción listo para alojar la aplicación, no merecía invertir tiempo en ello.

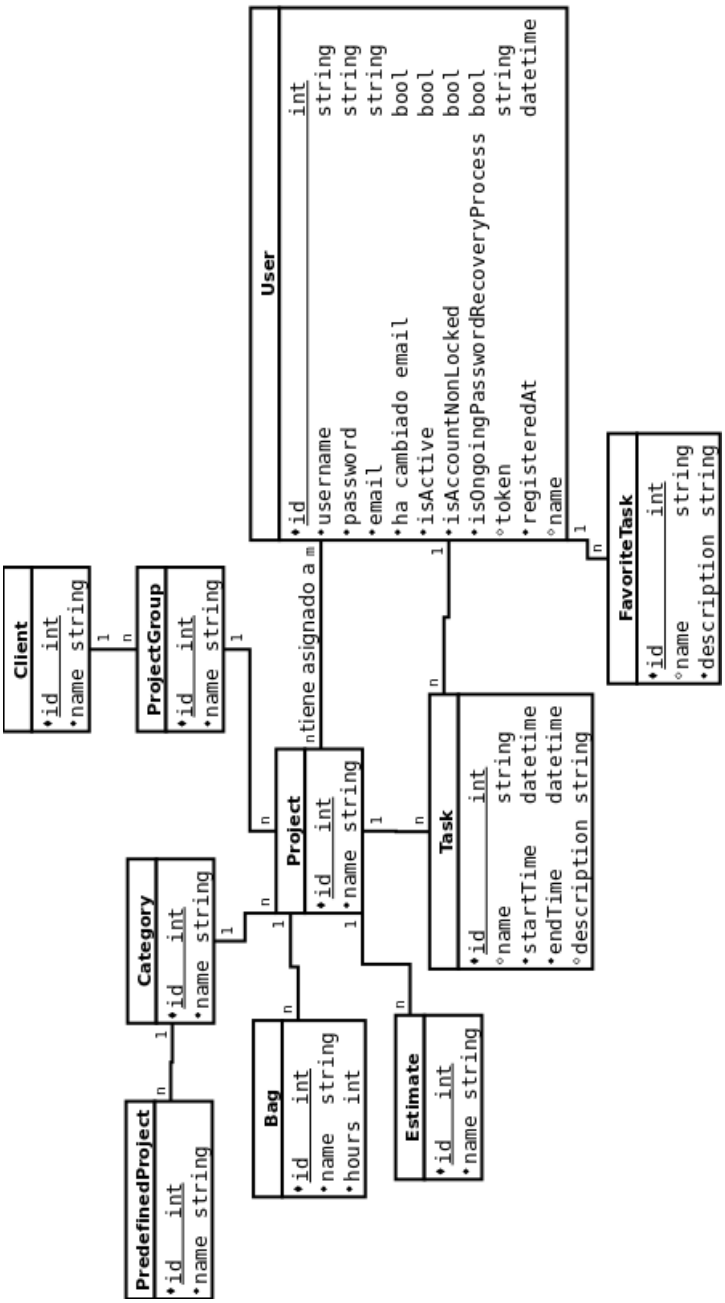


Figura 3.4: Diagrama del modelo entidad-relación de la base de datos

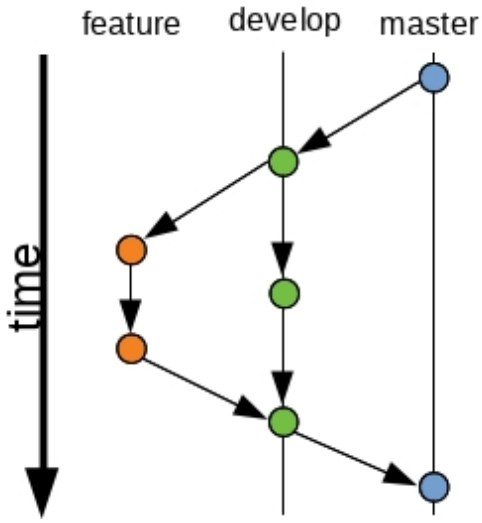


Figura 3.5: Diagrama del modelo de ramas utilizado en el repositorio

3.2.1.5. Roles

Por deseo de los interesados, los usuarios de la aplicación se iban a dividir en tres grupos principales: gestor con privilegios, gestor, y usuario normal. En la figura 3.6 se pueden apreciar los diferentes casos de uso asignados a cada rol. También se puede observar que los roles se heredan, y por lo tanto el gestor con privilegios heredará los casos de uso del gestor regular y el usuario regular.

3.2.1.5.1. Usuario regular El usuario con menos privilegios de la aplicación. La idea de este rol es que los usuarios tengan la sensación de que sólo ellos están utilizando la aplicación, y por lo tanto, se restringe el acceso a toda la información que no tenga que ver directamente con él, y por lo tanto puede —por gestionar se entiende crear, modificar y eliminar—:

- Consultar su perfil de usuario.
- Modificar sus datos de usuario.
- Modificar su email y verificarlo.
- Modificar su contraseña.
- Recuperar la contraseña.
- Consultar los proyectos en los que toma parte —indicándole el nombre, grupo del proyecto y la categoría del mismo —.

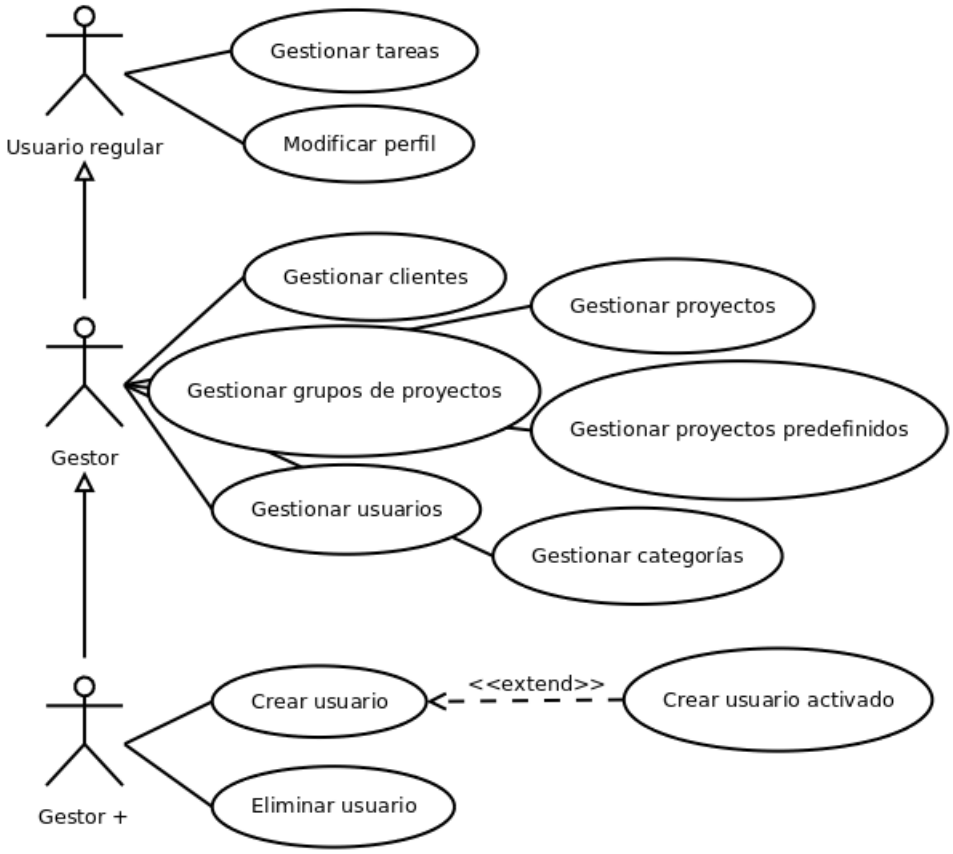


Figura 3.6: Diagrama de casos de uso de la aplicación

- Gestionar tareas en los proyectos a los que está asignado.
- Gestionar sus tareas favoritas.

3.2.1.5.2. Gestor El gestor es el usuario con privilegios al que se le permite acceder y modificar casi cualquier apartado de la aplicación. En la siguiente lista se listan las funcionalidades a las que tiene acceso:

- Gestionar clientes.
- Gestionar grupos de proyectos y asignarlos a clientes.
- Gestionar proyectos y asignarlos a grupos de proyectos, y también categorizarlos.
- Abrir y cerrar proyectos: de esta forma ningún usuario puede imputar más tareas al proyecto.
- Gestionar proyectos predefinidos. Además, los proyectos creados a partir de un proyecto predefinido verán sus campos modificados si un gestor modifica el proyecto predefinido al que está relacionado.
- Gestionar categorías.
- Gestionar bolsas de horas y asignarlas a proyectos.
- Gestionar presupuestos y asignarlos a proyectos.
- Gestionar las tareas de cualquier usuario de la aplicación.
- Gestionar las tareas favoritas de cualquier usuario de la aplicación.
- Modificar el perfil de cualquier usuario de la aplicación.
- Activar y desactivar un usuario, lo que permite restringirle por completo el acceso a la aplicación.
- Asignar y revocar el acceso de los usuarios a los proyectos.

3.2.1.5.3. Gestor con privilegios El gestor con privilegios es el usuario que puede utilizar la aplicación sin ningún tipo de restricción. Los interesados del proyecto insistieron en que únicamente un rol específico debería de poder añadir y eliminar usuarios de la aplicación, ya que opinaban que eso facilitaría la gestión de la misma. Así que las funcionalidades que se le ofrecen a los usuarios con este rol son:

- Crear un usuario en la aplicación.
- Eliminar un usuario de la aplicación.

Capítulo 4

Cierre de proyecto y conclusiones

4.1. Seguimiento y control

4.1.1. Alcance

El alcance especificado ha sido cumplido minuciosamente, y la decisión de haber separado el proyecto en dos más pequeños parece haber sido un acierto, ya que me ha permitido desarrollar la aplicación junto con todas sus pruebas correspondientes, dando como resultado un backend que podría ser considerado como estable y sólido.

4.1.2. Tiempo

El proyecto necesitó un total de 484 horas y 25 minutos para completarse, que se desglosan en 59 horas y 25 minutos de gestión y desarrollo de la documentación, 385 horas de desarrollo, y 40 horas más para la finalización de la documentación, retoques finales y preparación de la defensa. La siguiente lista lo resume:

- Gestión: 59 horas y 25 minutos.
  - Noviembre: 53 horas y 55 minutos.
  - Diciembre: 20 horas y 15 minutos.
  - Marzo: 4 horas y 15 minutos.
- Desarrollo: 385 horas.
  - Diciembre: 20 horas y 45 minutos.

- Enero: 155 horas.
- Febrero: 95 horas y 30 minutos.
- Marzo: 113 horas y 45 minutos.

- Gestión final: 40 horas.

Esto supera por 68 horas y 10 minutos estimación inicial de 416 horas y 15 minutos la planificación de tiempo realizada para el proyecto, y por 34 horas y 25 minutos los recursos totales evaluables del proyecto: 450 horas.

4.1.2.1. Fase inicial

Nota: se puede encontrar el informe completo con las tareas detalladas en el anexo A

Los principales desvíos se han dado en la gestión al inicio del proyecto. Si bien se estimaba que el inicio del desarrollo comenzaría hacia el 28 de Noviembre de 2017, no es hasta el 14 de Diciembre de 2017 que se tiene constancia de la primera hora invertida en el desarrollo del producto. Esto se traduce en 54 horas y 25 minutos, que se invirtieron en las tareas descritas en la lista de abajo. También cabe mencionar que durante ese periodo, la dedicación no fue exclusivamente al proyecto, sino que se compaginó con otras tareas independientes al mismo, que fueron realizadas para la empresa. Volviendo a la lista, el plasmar toda la información en el documento LaTeXllevó más tiempo del deseado, ya que tuve que volver a aprender a utilizar LaTeXcorrectamente, los problemas de compilación, y sobre todo los problemas de corrección y adecuación de formato.

Al comienzo del proyecto me dediqué sobre todo a realizar las entrevistas a los diferentes grupos de interesados de la aplicación, procesar las respuestas que me dieron y plasmar los requerimientos extraídos en la sección de requerimientos —véase sección 2.1.3 —de este documento.

Se realizaron varias reuniones a lo largo de la fase inicial. En ellas se definió la idea que tenían los interesados sobre lo que se quería realizar en este proyecto, y sobre todo se usaron para validar lo que yo había recogido como requerimientos y la definición del alcance.

Por último se dedicó tiempo a diseñar y modelar el modelo entidad-relación.

4.1.2.2. Fase de desarrollo

4.1.2.2.1. Diciembre

Nota: se puede encontrar el informe completo con las tareas detalladas en el anexo A

En diciembre, sobre todo me dediqué a las tareas de instalación y configuración del framework Symfony, aprender sobre algunos componentes clave y poner en práctica lo aprendido en proyectos de prueba.

4.1.2.2.2. Enero *Nota: se puede encontrar el informe completo con las tareas detalladas en el anexo A*

Uno de mis objetivos con la aplicación era que tuviera una documentación de la API clara y concisa. Es por ello que utilicé la librería *nelmio/api-doc-bundle*, la cual facilita la documentación de los diferentes endpoints que conforman una API.

En un principio, comencé a usar *friendsofsymfony/rest-bundle* para generar los endpoints de la API, en vez de utilizar API Platform. Esto se debe a que en un principio consideré más apropiado utilizar una librería que no trajera nada por defecto, de forma que me sirviera de formación para cuando usase el framework de API Platform.

Después de esto, me dediqué a configurar el serializador —véase 3.1.3.9 —*jms/serializer*, básicamente porque lo recomendaba la librería *friendsofsymfony/rest-bundle*.

Una vez configurado, creé el dominio en base al modelo entidad-relación que había diseña, ya que hasta entonces había configurado el serializador con unas entidades simples, que me permitieran centrarme en entender cómo trabajar con él sin tener que lidiar con la incertidumbre de no saber si algo no funciona por causa del serializador o las entidades.

A continuación tocó crear lo que se denominan *fixtures*, o datos de prueba, con las entidades creadas, de forma que pudiera comprobar que no existían fallos de diseño en el modelo entidad-relación. Por desgracia sí que existían problemas no con el diseño, sino con la implementación del mismo, así que tuve que lidiar con ellos y retocar las entidades para que funcionaran bien con el framework Symfony.

Con las bases ya controladas, procedí a crear unos endpoints de prueba con una de las entidades, realicé las pruebas correspondientes y contrasté los resultados obtenidos. Obviamente hubo mucha prueba y error, pero una vez conseguido, continué configurando el serializador para intentar limitar la información que se devolvía dependiendo de los roles establecidos.

Debido a que la fecha de incorporación del alumno que se iba a encargar del frontend de la aplicación estaba muy cerca, me puse manos a la obra con API Platform. La gran ventaja de este framework para APIs es que genera automáticamente todos los endpoints a partir de un modelo, y al tenerlo desarrollado prácticamente me dio la funcionalidad CRUD de toda la aplicación casi al instante.

Además, las pruebas realizadas hasta el momento con la otra librería me permitieron integrar una librería de gestión de autentificación —véase 3.1.3.12 —y otra de gestión de usuarios —*friendsofsymfony/user-bundle* —.

No obstante, en ese momento uno de los interesados de la aplicación me dijo que no podía usar una versión de Symfony que no fuera la estable, la cual por aquel entonces era la versión 3.4. El problema residía en que API Platform usaba una versión de Symfony superior, y que yo no encontraba la forma de hacer que los dos frameworks fueran compatibles en la versión que el interesado pedía. Así que mientras encontraba una solución, continué con el trabajo de formación que había realizado, ya que esto le daba al menos un backend funcional al alumno que se había incorporado.

Finalmente encontré dónde surgía la incompatiblidad: cuando intentaba integrar la versión 3.4 de Symfony con la versión estándar de API Platform, este último framework se quejaba porque la versión de Symfony no estaba entre sus candidatas para poder funcionar correctamente. Por lo tanto, indagando en las dependencias de la versión estándar, descubrí que ésta dependía de otro paquete —véase 3.1.3.13 —el cual proveía de toda la funcionalidad necesaria, y aceptaba la versión 3.4 de Symfony para poder funcionar con ella.

Así que con alivio, pude volver a utilizar API Platform, aunque tuve que trabajar a marchas forzadas para instalar cuidadosamente, una por una, el resto de librerías necesarias y asegurándome de que cada paso quedaba bien registrado en el repositorio donde tenía el código fuente de la aplicación. Al fin y al cabo, al instalar la versión no estándar de API Platform, había ciertas librerías que no estaban incluidas —sobre todo aquellas relativas a las pruebas o a proporcionar integraciones con otras librerías —, tenía que repasar qué es lo que faltaba para incluirlo en la aplicación.

Lo próximo fue meterse de lleno en aprender más sobre API Platform: cómo realizar los denominados como subrecursos —véase el párrafo siguiente —, documentar la API —ya que a pesar de que en esencia valía lo que aprendí con mis primeras pruebas de generación de la documentación, API Platform lo integraba de otra forma —, configurar Behat —véase 3.1.3.10 —y solucionar problemas varios relacionados con la implementación, las pruebas y cumplir con ciertos requerimientos que ya me empezaba a plantear el otro alumno —como que cuando un usuario iniciaba sesión, se mandase también su identificador de usuario para poder guardarlo y usarlo en peticiones futuras —.

Sobre los subrecursos: *subresources* en inglés, son aquellos que permiten obtener unos resultados filtrados con respecto a dos o más entidades. Por ejemplo, si en un dominio existen las entidades *User* y *Book*, un subrecurso útil podría ser el de `https://example.org/users/21/books` para obtener

la lista de libros del usuario con el identificador 21.

4.1.2.2.3. Febrero *Nota: se puede encontrar el informe completo con las tareas detalladas en el anexo A* En el modelo algunas de las relaciones se establecieron del tipo *"uno a uno"*, y por algún motivo daban problemas a la hora de insertar datos en la base de datos, sobre todo con las restricciones de referencia a claves extranjeras en las operaciones de eliminación. Se invirtió mucho tiempo en investigar qué podría estar causando este problema, mirando las sentencias Structured Query Language (SQL) que se generaban e intentando reproducir paso a paso el error que se estaba produciendo, ya que curiosamente cuando aplicaba una relación *"uno a uno"* sin el framework API Platform el error no se daba, pero con esa librería en el proyecto sí. Finalmente tras mucho indagar no pude dar con el problema, y a pesar de haber expuesto el problema en los canales de soporte pertinentes, nadie parecía encontrar explicación a lo que ocurría. No obstante el problema se pudo esquivar debido a que después de analizar el modelo entidad-erlación y debatirlo con los interesados, se debatió que tenía más sentido sustituir dichas relaciones por unas del tipo *"uno a n"*, como era en el caso de los presupuestos de un proyecto, por ejemplo. Aunque la cosa no acabó aquí ya que se dedicó más tiempo a intentar descubrir qué era lo que causaba el problema, y cómo podría solucionarse, a pesar de que ya no hiciera falta saberlo, pero no hubo manera. Por lo que finalmente asumí que probablemente se debería a mi torpeza, y decidí no perder más el tiempo con ello, a pesar de no quedarme satisfecho.

Continué implementando las pruebas restantes para el resto de entidades de la aplicación, ya que al estar el segundo alumno desarrollando su parte, no quería entorpecer su avance y por lo tanto me centraba en implementar las cosas que me iba pidiendo, además de los requerimientos que tenía que cumplir en la aplicación. Pero siempre con la idea en mente de completar todas las pruebas, y finalmente pude hacerlo dejando la aplicación en un estado de *"completamente probada"*. Obviamente estas pruebas evolucionarían y se modificarían en el futuro, y se añadirían nuevas según desarrollase nuevas cosas, pero al menos me había quitado esa *espinita*.

Uno de los elementos que supuso un cambio significativo fue la inclusión del sistema de notificaciones de correo. Se utilizó la librería *swiftmailer/swiftmailer*, debido a que es una librería completísima que facilita el envío de emails utilizando diferentes protocolos de transporte, soporta cifrado e inicio de sesión en servidores de correo, permite enviar mensajes en HyperText Markup Language (HTML)[35] y demás funcionalidades. También porque es la librería recomendada por el propio framework Symfony, y otras

muchas librerías más como por ejemplo la librería de gestión de usuarios *friendsofsymfony/user-bundle*, la cual utiliza el ya mencionado *Swiftmailer* como su sistema de gestión de notificaciones de correo electrónico. El problema principal residía en que a pesar de que API Platform no recomendaba la librería de gestión de usuarios ya mencionada, sí proveía de un *bridge*[17] o puente que facilitaba la integración en el framework de creación de APIs. No especificaba cuáles eran los motivos, pero descubrí uno de ellos al empezar con el desarrollo de las notificaciones de correo: las funcionalidades de recuperación de contraseña, registro, envío de email de confirmación y demás requerían de la visita en un recurso web. Por lo tanto, su forma de implementar las notificaciones, y las generaciones de Uniform Resource Identifier (URL)s con identificadores únicos para verificar la identidad del usuario que realiza las acciones, estaban muy enfocadas a proporcionar dichos servicios accediendo a diferentes recursos web. El problema fundamental residía en que yo estaba desarrollando una API REST, la cual debería ser totalmente independiente del frontend con el que se realizan las peticiones, y por lo tanto el hecho de tener que obligar a que las capas de visualización tuvieran que visitar ciertos recursos web lo consideraba incoherente y limitante. La solución era tener que realizar *"overrides"* de muchísimas partes de la librería de gestión de usuarios, para intentar adaptarla a mis necesidades. Así que siguiendo el consejo de API Platform, decidí crear un sistema de usuarios personalizado —denominado *"custom user provider"* en el mundo de Symfony—, justamente porque el trabajo de tener que crear dicho sistema iba a ser muy similar al de tener que adaptar una librería que no estaba diseñada para trabajar con APIs.

Una vez se implementó el sistema de usuarios y se integró con *Swiftmailer*, se pasó a desarrollar un sistema de invitaciones para restringir el registro en la aplicación: sólo los gestores con privilegios podrían invitar, mediante el envío automatizado de un correo electrónico desde la aplicación, a ciertos usuarios a ser parte de la aplicación. Estos usuarios tendrían que validar su correo haciendo clic en el enlace correspondiente en su correo electrónico, que después el frontend traduciría en una petición a la API.

En ése momento, tras revisar el dominio, me di cuenta de que había cometido algunos errores en el diseño del mismo. No eran errores graves, ya que se podían corregir fácilmente sin tener que desechar nada de lo desarrollado hasta la fecha, pero requerían algunos cambios significativos que sobre todo iban a suponer un trabajo considerable en cuanto a modificar todas las pruebas afectadas. La causa de estos errores se debe a lo siguiente: antes de realizar el primer modelo entidad-relación, se me presentó un modelo de gestión de la empresa que habían desarrollado y que querían que sustituyera el modelo de gestión que venían usando hasta entonces. Desarrollé el mode-

lo entidad-relación equivalente, y lo presenté a los interesados. No obstante, al validarlo con casos reales y actuales de la empresa, mezclamos conceptos de la aplicación de gestión que se estaba usando actualmente con los nuevos conceptos del modelo, y eso unido al modelo entidad-relación diseñado generó muchísima confusión. A pesar de todo ello, el modelo parecía válido porque no encontramos limitaciones a la hora de validar los casos reales. Obviamente, y como menciono al principio del párrafo, al revisar el modelo de nuevo encontré errores que achaqué a la confusión a la hora de debatir este tema, y mi falta de clarividencia a la hora de plasmar lo que resultó de dichos debates, aunque en mi defensa he de decir que todos los involucrados en los susodichos debates salimos algo confundidos de ellos. Aún así, estos errores no suponían cambios muy significativos, así que adapté el modelo entidad-relación al modelo final que se presenta en la figura 3.4, y presenté los cambios a los interesados, esta vez sin tanta confusión y con más facilidad debido a las adaptaciones realizadas. Las consecuencias de estos cambios fueron el tener que adaptar las pruebas de la aplicación y los correspondientes datos de prueba, lo que supuso un trabajo considerable. Por si eso fuera poco, el consumo de memoria de las pruebas se disparó, o más bien me fijé que era muy alto. Tras mucho trastear e investigar por si había realizado algo fuera de lo común que estuviera malgastando recursos, pregunté en ciertos canales de desarrolladores que me comentaron que no tenía nada de que preocuparme ya que los consumos estaban dentro de lo normal. De hecho, de ahí a un tiempo, en una actualización de la librería de pruebas, el consumo descendió muy significativamente, lo que me dio la prueba definitiva de que realmente ése consumo alto de recursos no se debía a algún fallo cometido por mi parte.

Tras tenerlo todo atado, continué desarrollando las validaciones de los diferentes campos de las entidades, ya que hasta este punto únicamente había realizado unas validaciones genéricas.

Finalmente la última funcionalidad que comencé a implementar en el mes de febrero fue la de la confirmación del registro por parte de los usuarios, ya que el sistema de invitaciones de registro funcionaba bien, pero el desarrollo de la confirmación se paró por el asunto de la modificación del dominio. Esto lo realicé con Data Transfer Object (DTO)s. Es decir: la validación en el framework Symfony se realiza sobre las entidades —véase la sección de validaciones 3.1.3.8—, y por lo tanto estos *DTO*s no son más que pequeñas entidades que sirven para un propósito específico, como puede ser el recoger el identificador y el correo electrónico de la confirmación del usuario.

4.1.2.2.4. Marzo

Nota: se puede encontrar el informe completo con las tareas detalladas en el anexo A

En marzo seguí con las modificaciones en la serialización de las entidades, tanto para limitar la información que se mostraba a cada rol del usuario como para satisfacer las necesidades de navegación e información que me pedía el desarrollador del frontend.

Después vino una tremenda refactorización de las pruebas de la aplicación. Hasta entonces, las pruebas consistían en, generalmente, esperar unos valores específicos después de realizar ciertas peticiones a la base de datos que contenía los datos de prueba. No obstante, esto suponía un trabajo considerable y repetitivo a la hora de desarrollar una nueva funcionalidad junto con sus pruebas, o realizar modificaciones a las funcionalidades ya existentes. Por lo que decidí transformar las pruebas a pruebas de validación con JSON Schema —véase la sección 3.1.3.11 —en gran parte, lo que facilitaba mucho la mantenibilidad y reutilización de los esquemas en distintas pruebas.

Finalicé el desarrollo del sistema de verificación de usuarios con el que había comenzado a trastear a finales de febrero, y aproveché para implementar la funcionalidad de cambio de contraseña y de cambio de email, todo ello desarrollado y validado con los DTOs. Junto con esto, ideé unas pequeñas restricciones e indicadores que ayudaban a identificar el estado de la cuenta cuando se estaba llevando a cabo un proceso de cambio de contraseña, recuperación de contraseña o cambio de dirección de correo electrónico: por ejemplo, si se comenzaba con el proceso de recuperar la contraseña pero el usuario iniciaba sesión antes de completarlo, se entendía que o bien no lo había solicitado o que ya se había acordado de ella, y por lo tanto el proceso se daba por finalizado sin que el usuario tuviera que hacer nada más.

Después de limitar la información enviada por cada rol de usuario, terminé de implementar el sistema de control de acceso, el cual hasta el momento estaba mínimamente desarrollado para probar la restricción de acceso a ciertos recursos. Terminó siendo muy sencillo gracias a la herencia de roles de Symfony, que facilita la gestión de los mismos, y que permite hilar muy fino en cuanto a los recursos o serie de recursos que se quieren restringir mediante el uso de expresiones regulares. Por ejemplo, se puede restringir la zona `https://example.org/admin` con una expresión regular $\hat{}/admin$ que haga que todas las subrutas como por ejemplo `https://example.org/admin/check_status` estén restringidas a un rol. Los recursos que debían estar disponibles para los tres roles, pero que debían devolver una información u otra, se restringían programando la lógica de negocio, y aplicando los grupos de serialización —véase sección 3.1.3.9 —que se habían desarrollado para las entidades. Con todo esto, se desarrollaron también las pruebas correspondientes, comprobando que absolutamente todas las restricciones funcionaban como se pretendía que lo hicieran.

Las últimas funcionalidades que se desarrollaron antes de finalizar con la fase de desarrollo fueron las de abrir y cerrar proyectos —restringiendo la posibilidad de añadir o eliminar tareas por parte de ningún usuario, incluyendo los gestores —y permitir a los usuarios que marquen ciertas tareas como favoritas —para ayudarles a guardar tareas que se repiten muchas veces y poder reutilizarlas sin tener que volver a introducir todos los datos—.

El resto del tiempo se dedicó a ajustar la aplicación, arreglar fallos e implementar las últimas peticiones del otro alumno para flexibilizar el backend.

4.1.3. Calidad

Se ha alcanzado la calidad mínima especificada en sección de la gestión de calidad 2.4. En cambio, los elementos de la calidad añadida no han sido desarrollados, ya que no se ha dispuesto de los recursos suficientes como para dedicarles un tiempo de desarrollo que fuera a culminar en una nueva funcionalidad completamente probada. Si no hubiera tenido los desvíos indicados en la sección de seguimiento y control del tiempo —véase sección 4.1.2 —, creo que la única funcionalidad que pudiera haber implementado es la onceava.

Finalmente no he podido utilizar la metodología TDD en el desarrollo, ya que generalmente mi intención era poder *publicar* las funcionalidades cuanto antes para que el otro alumno las tuviera listas para trabajar sobre ellas. Antes de subir los cambios al repositorio que contenía el código de la aplicación, a veces solía dejar el código de los cambios en un servidor de pruebas que usábamos para juntar los dos desarrollos, y de mientras iba realizando las pruebas oportunas para que una vez se hubiera probado que todo funcionaba bien, subir los cambios al repositorio. Esta forma de trabajar me permitía tener un colchón de ventaja sobre el otro alumno y por lo tanto podía trabajar más relajadamente. Por contra, la desventaja de esta forma de trabajo era que si el otro alumno se ponía a desarrollar sobre una funcionalidad que no estaba probada, podía ocurrir que después tuviera que cambiar ligeramente su implementación para adaptarla a las correcciones aplicadas después de realizar las pruebas, algo que sucedió en alguna ocasión.

4.1.4. Riesgos

Debido a que los riesgos identificados 2.5.2 no se han mostrado significativamente a lo largo del desarrollo del proyecto, no se han tenido que poner en marcha los planes de contingenciPor contra, la desventaja de esta forma de trabajo era que si el otro alumno a 2.5.3.

Es cierto que hubo un momento en el que quizás podría considerarse que el riesgo de *"No tener un producto mínimo que sea funcional cuando el otro integrante comience a desarrollar su parte."* estuvo cerca de cumplirse, ya que tal y como relato en la sección de seguimiento y control de tiempo, la que corresponde al mes de enero 4.1.2.2.2, la semana en la que se incorporó el otro alumno estaba todavía trasteando con las tecnologías de formación y las del desarrollo del producto final. Aunque también he de decir que el otro alumno siempre tuvo un backend operativo con el que comenzar su desarrollo.

4.2. Conclusiones

Con respecto al producto, se ha logrado el objetivo de construir una API sobre un backend Symfony que permita ser consumida por diferentes frontends. La aplicación en sí podría resumirse como un conjunto de librerías cohesionadas con las que se ha desarrollado un código que realiza la tarea propuesta. Una de las funcionalidades que quisiera destacar es que la aplicación en sí ha sido construida enteramente con proyectos de código abierto, y es por ello que me apena el no haber conseguido convencer a los interesados para que el código de la aplicación terminara haciéndose público bajo una licencia de software libre o código abierto, aunque entiendo perfectamente los intereses corporativos.

La gestión del proyecto, en mi opinión, ha sido aceptable. Creo que tomé una buena decisión en no invertir demasiado tiempo en intentar crear un plan de ruta, ya que después de todo el proceso de desarrollo, no creo que me hubiera acercado a las estimaciones iniciales. Lo que sí que es cierto es que se retrasó demasiado el comienzo del desarrollo, y también la formación en la tecnología, lo que en caso contrario podría haberme permitido desarrollar alguna funcionalidad más. El objetivo de realizar un seguimiento y control exhaustivo se ha cumplido, ya que he tomado nota de todo lo que he realizado a lo largo del desarrollo, aunque creo que para una próxima vez sería mejor categorizar las tareas realizadas, ya que de esta forma se podría elaborar un informe un poco más abstracto, que diera una mejor idea de todo lo realizado sin tener que leer una por una todas la tareas que se llevaron a cabo.

En cuanto a los objetivos personales, estoy muy satisfecho de haber podido trabajar en un proyecto con el Web Frameworkframework Symfony, ya que uno de mis objetivos era poder profundizar y entenderlo mucho mejor, y lo he conseguido. También quería seguir las mejores prácticas posibles, y es por ello que entre otras cosas he dedicado mucho tiempo a realizar las pruebas de las funcionalidades implementadas, y a intentar seguir minuciosamente los patrones y estilos de código recomendados, para que si en un

futuro se abre el código fuente de la aplicación al público no sólo no se pueda poner en evidencia a la empresa por no seguir las prácticas recomendadas, sino que tampoco se me pueda poner en evidencia a mi por realizar un desarrollo chapucero. Sé que esto último es muy subjetivo, y que a pesar de mis buenas intenciones puede que no haya logrado mi objetivo, pero me quedo muy tranquilo al saber que he hecho todo lo posible por crear un producto de calidad y siguiendo lo que muchos expertos recomiendan y publican en las documentaciones oficiales, blogs y salas de soporte de las diferentes tecnologías.

En general, creo que tanto los interesados como yo hemos terminado muy satisfechos con el trabajo que este proyecto ha supuesto.

4.3. Líneas de trabajo futuras

Las líneas de trabajo futuras creo que van encaminadas a completar el desarrollo de las funcionalidades opcionales que quedaron sin desarrollar, además de las funcionalidades futuras que quieran incluir en la aplicación.

Cuando se me preguntó sobre cómo haría yo para comercializar un producto como este, planteé lo siguiente: al ser una aplicación que no es en sí novedosa, y de la que se pueden encontrar alternativas en el mercado, una idea podría ser la de abrir el código fuente y convertir el proyecto en uno de código abierto, ofreciendo a su vez un servicio en el que la empresa se haría cargo de mantener una instancia del producto por un determinado precio, como por ejemplo hace GitLab[1]. Las ventajas de este modelo de negocio es que cualquiera puede atreverse a auditar, mejorar, sugerir correcciones, aportar críticas e incluso tomar parte en el desarrollo del producto, además de que daría una buena imagen a la empresa por tener un proyecto open source. Las desventajas de este tipo de modelo de negocio son que no se puede dejar un producto como este sin actualizar mucho tiempo, y sin incorporar nuevas características o mejoras constantes, ya que un proyecto que tiene pinta de no haberse actualizado desde hace tiempo resulta poco atractivo. Además, otra de las desventajas es que hay que mimar el producto: para que no se ponga en duda la profesionalidad de la empresa, sería recomendable seguir las mejores prácticas posibles y documentar el código extensamente y de forma inteligible, entre otras cosas. Con todo esto, se podría apostar por pelear con el producto en un mercado que ya está saturado de este tipo de productos.

El servicio prestado tendría que ser fiable y escalable, de forma que los potenciales clientes tuvieran la sensación de que merece la pena pagar por él. De forma que un cliente pudiera gestionar varias empresas, podría hacerse

[1] https://gitlab.com/

para que cada vez que el cliente diera de alta una de ellas, se lanzase una instancia de Docker con el producto, e insertando automáticamente ciertos datos que son necesarios para empezar a organizar la aplicación —básicamente la cuenta de un gestor con privilegios —.

Por lo que, en resumen, estas propuestas esconden en sí más proyectos y líneas de trabajo futuras sobre las que extender el ecosistema del producto desarrollado en este proyecto.

Siglas

ACID Atomicity Consistency Isolation Durability. 33

API Application Programming Interface. 4, 11, 20, 21, 26, 31, 42, 45, 49, *véase:* API

BDD Behavior-Driven Development. 28, *véase:* Behavior-Driven Development

Continuous delivery Continuous delivery. 34, *véase:* Continuous delivery

Continuous integration Continuous integration. 34, *véase:* Continuous Integration

CRUD Create, Read, Update, Delete. 21, 42

DBAL Database Abastraction Layer. 24, *véase:* DBAL

DQL Doctrine Query Language. 24

DTO Data Transfer Object. 46, 47, *véase:* DTO

EDT Estructura de Descomposición de Trabajo. 3, 10

HATEOAS Hypermedia As The Engine Of Application State. 21, *véase:* HATEOAS

HTML HyperText Markup Language. 44

JWT JSON Web Token. 31

KVM Kernel Virtual Machine. 32, *véase:* Kernel Virtual Machine

ORM Object Relational Mapping. 24, *véase:* ORM

PFM Proyecto de Fin de Máster. 3, 10

PHP PHP: Hypertext Preprocessor. 20–22, 24, 29, 33

RDBMS Relational Database Management System. 33

REST Representational State Transfer. 4, 11, 31, 45, *véase:* REST

SQL Structured Query Language. 44

TDD Test Driven Development. 11, 48, *véase:* Test Driven Development

URL Uniform Resource Identifier. 45

Glosario

API Grupo de procedimientos calaremente definidos que facilitan la comunicación entre componentes de software [3] .

Back End Capa de lógica y acceso a datos del software [1] . 4, 6, 40, 43, 48, 49

Backports En Debian, los *backports* se denominan a los paquetes recompilados de las ramas *testing* y *unstable* para poder ser instalados en las distribuciones Debian de la rama estable [4] . 33

Behavior-Driven Development Se considera una extensión de TDD que hace uso de un lenguaje de dominio específico para convertir sentencias escritas en lenguaje natural en pruebas ejecutables [5] .

Commit En sistemas de control de versiones, el *commit* añade los últimos cambios, realizados a parte del código, al repositorio [7] . 34

Continuous delivery La entrega continua consiste en desarrollar el software en ciclos cortos, de forma que se pueda realizar un lanzamiento o un despligue en cualquier comento. El objetivo es construir, probar y lanzar el software con más rapidez y frecuencia [8] .

Continuous Integration La integración continua consiste en fusionar todas las copias funcionales de los desarrolladores en una rama común, varias veces al día, con el objetivo de evitar problemas de integración [9] .

Cookie Pequeño fragmento de datos que las aplicaciones web envían a los usuarios, y que generalmente estos últimos almacenan en su navegador, con la finalidad de guardar cierta información de estado —como por ejemplo sesiones, opciones de personalización de la aplicación o la lista del carro de la compra digital—. Después esta información se manda de vuelta al servidor para que este último adecúe la aplicación al usuario que mandó la petición [23] . 22

DBAL API que tiene como objetivo unificar la comunicación entre la aplicación y las distintas tecnologías de bases de datos, como pueden ser MariaDB, MySQL, SQL Server, etc. La idea es proveer al desarrollador una única interfaz que le evite tener que programar una interfaz por tecnología de base de datos [11] .

Docker Docker es un software que realiza virtualización a nivel de sistema operativo, permitiendo lanzar *"contenedores"* independientes desde una única instancia de Linux, lo que ahorra el coste de gestionar máquinas virtuales [12] . 50

DTO Un objeto que transmite datos entre procesos [10] .

Endpoint La URL de una API o un backend que responden a una petición [2] . 42

Fork Se denomina *fork* a la acción de clonar el código fuente de un proyecto de software y empezar un desarrollo independiente del mismo, creando un software distinto y separado del proyecto original [16] . 33

Front End Capa de visualización del software [1] . 4, 6, 8, 9, 42, 45, 47, 49

GitLab Software web de gestión de repositorios *git* desarrollado con una licencia opensource [21] . 34

HATEOAS Restricción de la arquitectura REST, la cual pretende proveer de indicaciones al consumidor de la API sobre qué pasos se pueden dar en ese punto mediante enlaces a las siguientes operaciones disponibles. En un contexto de una aplicación de un banco, la respuesta que un cliente recibiría al hacer una llamada a una operación de consulta de saldo, podría incluir los enlaces a las operaciones de *transferencia*, *extracción de dinero* o *consultar movimientos*, por ejemplo [31] [22] .

Jira Software de seguimiento de proyectos e incidencias . 5

Kernel Virtual Machine Solución de virtualización para Linux en la que el kernel se convierte en el hipervisor —software, firmware o hardware que se encarga de gestionar y lanzar máquinas virtuales [26] —. Permite virtualizar múltiples sistemas operativos Linux, Windows, BSD, OS X y más, mediante una API con la que los susodichos sistemas operativos pueden interactuar [28] .

Merge En sistemas de control de versiones, el *merge* fusiona los cambios realizados en dos ramas en una sola, resultando en una única colección de ficheros que contienen los cambios de las dos ramas [29] . 34

Method overriding Es una característica que permite, en la programación orientada a objetos, proveer de una implementación específica de un método en una subclase, que sustituya la implementación del método de la súper clase [30] . 45

Modelo entidad-relación Un modelo entidad-relación o diagrama entidad-relación es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información así como sus interrelaciones [15] y propiedades . 11, 33, 41, 42, 44–46

ORM Técnica que permite consultar y manipular datos en una base de datos utilizando un lenguage de programación orientado a objetos [14]. La técnica consiste en transformar la representación lógica de los objetos en una forma atómica que pueda ser guardada en la base de datos, manteniendo las propiedades de dichos objetos y sus relaciones de tal forma que puedan volver a ser cargados como objetos. Un ejemplo son las bases de datos SQL, las cuales sólo guardan valores escalares como enteros o cadenas de caracteres en tablas relacionales[33]. La técnica permitiría transformar los objetos de tal forma que puedan ser guardados en una tabla relacional, junto con sus relaciones, y que se puedan recuperar con forma de objeto con el lenguaje de programación orientado a objetos, y sin tener que usar SQL .

Pruebas funcionales Las pruebas funcionales o *functional testing* se consideran como un proceso para asegurar la calidad del software, y también como un tipo de pruebas de la modalidad caja negra. Generalmente, este tipo de pruebas se realizan introduciendo una entrada y analizando la salida de la funcionalidad en cuestión que se está probando, comprobando que todo funciona según las especificaciones de dicha funcionalidad [19] . 28

Push En el sistema de control de versiones *git*, acción con la que se mandan los cambios de tu repositorio al repositorio remoto [20] . 34

REST Arquitectura que define un grupo de restricciones y propiedades basadas en el protocolo Hypertext Transfer Protocol (HTTP). Los servicios web que cumplen con esta arquitectura, permiten la consulta y modificación de los recursos web mediante una serie de operaciones sin

estado. Las respuestas que remiten estos servicios pueden estar en una amplia variedad de formatos —como HTML, XML, JSON u otros formatos —y suelen incluir información sobre el resultado de la operación realizada [36] .

Software Open-Source Tipo de software del cual su código fuente es públicamente accesible, y que tiene una licencia de copyright asociada que permite estudiar, cambiar el código fuente y distribuirlo a cualquier persona y por cualquier motivo. Generalmente este software se suele desarrollar de una manera colaborativa [34] . 49, 50

Swagger UI Interfaz de usuario que muestra de forma muy amigable las operaciones de una API, además de ofrecer la posibilidad de interactuar con dichas operaciones en la misma página donde se muestran. Esta documentación se genera a partir de una especificación denominada Swagger . 21

Test Driven Development El TDD es un proceso de desarrollo de software que se basa en la repetición de un ciclo de desarrollo muy corto: los requerimientos se convierten en casos de pruebas muy específicos, y únicamente está permitido programar para hacer que dichos casos de prueba se realicen satisfactoriamente [39] .

Web Framework Un *Web Framework* es una abstracción en la que un software que provee de una funcionalidad genérica puede ser selectivamente modificada adiciones, modificaciones o eliminaciones de código. Estos *frameworks* ayudan en el desarrollo de aplicaciones web, incluyendo servicios web, recursos web y APIs web. El objetivo de estas herramientas es el de automatizar ciertas tareas y ahorrar tiempo en las actividades comunes asociadas al desarrollo web . 11, 20–24, 28, 32, 42–46, 49

Web semántica La web semántica es una extensión de la web que mediante estándares propuestos por la W3C, pretenden impulsar formatos de datos comunes, como pueden ser el RDFa, JSON-LD o HAL. De esta forma, se intenta impulsar una web contextualizada en la que los datos tengan relación entre sí, incluso si dichos datos están diseminados por distintos recursos web [37] . 21

Bibliografía

[1] URL: https://en.wikipedia.org/wiki/Front_and_back_ends (visitado 27-11-2017).

[2] amenadiel. *¿Qué es un "entry point" y un "end point"?* URL: https://es.stackoverflow.com/questions/51758/qu%C3%A9-es-un-entry-point-y-un-end-point/51764#51764 (visitado 24-05-2018).

[3] *Application programming interface*. URL: https://en.wikipedia.org/wiki/Application_programming_interface (visitado 27-11-2017).

[4] *Backports*. URL: https://wiki.debian.org/Backports (visitado 10-05-2018).

[5] *Behavior-driven development*. URL: https://en.wikipedia.org/wiki/Behavior-driven_development (visitado 03-05-2018).

[6] *Black-box testing*. URL: https://en.wikipedia.org/wiki/Black-box_testing (visitado 03-05-2018).

[7] *Commit (version control)*. URL: https://en.wikipedia.org/wiki/Commit_(version_control) (visitado 11-05-2018).

[8] *Continuous delivery*. URL: https://en.wikipedia.org/wiki/Continuous_delivery (visitado 11-05-2018).

[9] *Continuous integration*. URL: https://en.wikipedia.org/wiki/Continuous_integration (visitado 11-05-2018).

[10] *Data transfer object*. URL: https://en.wikipedia.org/wiki/Data_transfer_object (visitado 26-05-2018).

[11] *Database abstraction layer*. URL: https://en.wikipedia.org/wiki/Database_abstraction_layer (visitado 25-04-2018).

[12] *Docker (software)*. URL: https://en.wikipedia.org/wiki/Docker_(software) (visitado 30-05-2018).

[13] *Domain-specific language*. URL: https://en.wikipedia.org/wiki/Domain-specific_language (visitado 03-05-2018).

[14] e-satis. *What is an ORM and where can I learn more about it?* URL: https://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-more-about-it/1279678#1279678 (visitado 08-05-2018).

[15] *Entity–relationship model*. URL: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model (visitado 11-12-2017).

[16] *Fork (software development)*. URL: https://en.wikipedia.org/wiki/Fork_(software_development) (visitado 10-05-2018).

[17] *FOSUserBundle Integration*. URL: https://api-platform.com/docs/core/fosuser-bundle (visitado 26-05-2018).

[18] *FrameworkBundle Configuration ("framework")*. URL: https://symfony.com/doc/current/reference/configuration/framework.html (visitado 25-04-2018).

[19] *Functional testing*. URL: https://en.wikipedia.org/wiki/Functional_testing (visitado 03-05-2018).

[20] *Git - git-push Documentation*. URL: https://git-scm.com/docs/git-push (visitado 11-05-2018).

[21] *GitLab*. URL: https://en.wikipedia.org/wiki/GitLab (visitado 11-05-2018).

[22] *HATEOAS*. URL: https://en.wikipedia.org/wiki/HATEOAS (visitado 19-04-2018).

[23] *HTTP cookie*. URL: https://en.wikipedia.org/wiki/HTTP_cookie (visitado 24-04-2018).

[24] *HttpFoundation component*. URL: https://symfony.com/components/HttpFoundation (visitado 24-04-2018).

[25] *HttpFoundation component*. URL: https://symfony.com/components/HttpFoundation (visitado 24-04-2018).

[26] *Hypervisor*. URL: https://en.wikipedia.org/wiki/Hypervisor (visitado 10-05-2018).

[27] *JSON Web Token Introduction*. URL: https://jwt.io/introduction/ (visitado 07-05-2018).

[28] *Kernel-based Virtual Machine*. URL: https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine (visitado 10-05-2018).

[29] *Merge (version control)*. URL: https://en.wikipedia.org/wiki/Merge_(version_control) (visitado 11-05-2018).

[30] *Method overriding*. URL: https://en.wikipedia.org/wiki/Method_overriding (visitado 26-05-2018).

[31] Darrel Miller. *HATEOAS: concise description*. 2012. URL: https://stackoverflow.com/questions/9192648/hateoas-concise-description/9194942#9194942 (visitado 19-04-2018).

[32] *Model–view–controller*. URL: https://en.wikipedia.org/wiki/Model-view-controller (visitado 17-04-2018).

[33] *Object-relational mapping*. URL: https://en.wikipedia.org/wiki/Object-relational_mapping (visitado 08-05-2018).

[34] *Open-source software*. URL: https://en.wikipedia.org/wiki/Open-source_software (visitado 29-05-2018).

[35] *Powerful component based mailing library for PHP – Swift Mailer*. URL: https://swiftmailer.symfony.com/ (visitado 26-05-2018).

[36] *Representational state transfer*. URL: https://en.wikipedia.org/wiki/Representational_state_transfer (visitado 27-11-2017).

[37] *Semantic Web*. URL: https://en.wikipedia.org/wiki/Semantic_Web (visitado 19-04-2018).

[38] *Stateless protocol*. URL: https://en.wikipedia.org/wiki/Stateless_protocol (visitado 27-11-2017).

[39] *Test-driven development*. URL: https://en.wikipedia.org/wiki/Test-driven_development (visitado 11-12-2017).

[40] *The Serializer Component*. URL: http://symfony.com/doc/current/components/serializer.html (visitado 02-05-2018).

[41] *Water clock*. URL: https://en.wikipedia.org/wiki/Water_clock.

Apéndice A

Informes de tareas realizadas

Inicial

Created by Mikel Alejo

Projects DESARROLLO WEB CLEPSYDRA
Users Mikel Alejo
Time interval 01/11/2017 — 14/12/2017

Total	**54 hrs 55 min**
DESARROLLO WEB CLEPSYDRA	**54 hrs 55 min**
DESARROLLO WEB (DWC-1-2)	1 hrs
14/12/2017 10:00 — 11:00	1 hrs
GESTION (DWC-1-1)	53 hrs 55 min
13/11/2017 08:30 — 12:30 Realización del documento de preguntas, realización de las entrevistas y procesado de las respuestas.	4 hrs
14/11/2017 09:15 — 12:30 Realización del documento de preguntas, realización de las entrevistas y procesado de las respuestas.	3 hrs 15 min
16/11/2017 08:30 — 12:30 Procesado de las respuestas de los entrevistados y desarrollo de la documentación.	4 hrs
17/11/2017 08:30 — 12:30	4 hrs
20/11/2017 08:30 — 12:30 Planificación y documentación. Reunión con gestores.	4 hrs
22/11/2017 08:30 — 12:30	4 hrs
23/11/2017 08:30 — 10:15	1 hrs 45 min
23/11/2017 11:00 — 12:15 Reunion	1 hrs 15 min
24/11/2017 08:30 — 10:15	1 hrs 45 min
27/11/2017 09:45 — 10:20	35 min
27/11/2017 11:30 — 12:30	1 hrs
28/11/2017 08:30 — 11:15	2 hrs 45 min
28/11/2017 11:15 — 12:30 Reunión con Joseba	1 hrs 15 min
29/11/2017 11:30 — 12:30	1 hrs
30/11/2017 12:10 — 12:30	20 min
01/12/2017 08:30 — 10:30	2 hrs

04/12/2017 08:30 — 12:00	3 hrs 30 min
11/12/2017 08:30 — 12:30	4 hrs
12/12/2017 08:30 — 10:00	1 hrs 30 min
12/12/2017 10:00 — 12:30	2 hrs 30 min
13/12/2017 08:30 — 12:30	4 hrs
14/12/2017 08:30 — 10:00	1 hrs 30 min
Total	**54 hrs 55 min**

Diciembre

Created by Mikel Alejo

Projects DESARROLLO WEB CLEPSYDRA
Users Mikel Alejo
Time interval 01/12/2017 — 31/12/2017

Total	**41 hrs**
DESARROLLO WEB CLEPSYDRA	**41 hrs**
DESARROLLO WEB (DWC-1-2)	20 hrs 45 min
14/12/2017 10:00 — 11:00	1 hrs
21/12/2017 08:45 — 12:30 Instalación, configuración y pruebas de Symfony REST	3 hrs 45 min
23/12/2017 09:00 — 13:00 Aprender sobre Symfony HTTP-Foundation	4 hrs
23/12/2017 16:00 — 20:00 Poner en práctica lo aprendido en proyecto de prueba	4 hrs
25/12/2017 09:00 — 13:00 Aprender sobre Symfony Kernel	4 hrs
25/12/2017 14:00 — 18:00 Poner en práctica lo aprendido en proyecto de prueba	4 hrs
GESTION (DWC-1-1)	20 hrs 15 min
01/12/2017 08:30 — 10:30	2 hrs
04/12/2017 08:30 — 12:00	3 hrs 30 min
11/12/2017 08:30 — 12:30	4 hrs
12/12/2017 08:30 — 10:00	1 hrs 30 min
12/12/2017 10:00 — 12:30	2 hrs 30 min
13/12/2017 08:30 — 12:30	4 hrs
14/12/2017 08:30 — 10:00	1 hrs 30 min
20/12/2017 08:30 — 09:00 Definición de la API	30 min
20/12/2017 11:00 — 11:30 Documentar y definir la API	30 min
21/12/2017 08:30 — 08:45	15 min

Redactar email para definir el proyecto de Ander

Total	**41 hrs**

Enero

Created by Mikel Alejo

Projects DESARROLLO WEB CLEPSYDRA
Users Mikel Alejo
Time interval 01/01/2018 — 31/01/2018

Total	**155 hrs**
DESARROLLO WEB CLEPSYDRA	**155 hrs**
DESARROLLO WEB (DWC-1-2)	155 hrs
02/01/2018 08:30 — 12:30 Instalar y configurar NelmioApiDocBundle, y configurar gitlabci	4 hrs
03/01/2018 08:30 — 13:50 Instalación y configuración de NelmioApiBundle	5 hrs 20 min
03/01/2018 14:20 — 17:00 Finalizar NelmioApi y comenzar con JWT. Problemas con rutas manuales de FOSRest	2 hrs 40 min
04/01/2018 08:30 — 12:45 Instalar y configurar JWT usando el sandbox que ofrecen y los ejemplos visitados en internet	4 hrs 15 min
04/01/2018 12:45 — 13:30 Configurar JMS con las entidades ORM symfony	45 min
04/01/2018 14:00 — 14:30 Configurar JMS con entidades de Symfony. Al borrar la caché funcionó todo.	30 min
04/01/2018 14:30 — 17:00 Generar entidades, mapearlas correctamente y actualizar gitlabci.	2 hrs 30 min
05/01/2018 08:30 — 09:30 Ordenar repositorio git. Quitar branches sobrantes y mergear gitlab ci a develop.	1 hrs
05/01/2018 09:30 — 12:00 Creación del dominio en Symfony	2 hrs 30 min
05/01/2018 12:00 — 13:20 Testeo del dominio con Fixtures	1 hrs 20 min
05/01/2018 13:50 — 15:00 Añadir data fixtures y probar la base de datos	1 hrs 10 min
07/01/2018 09:00 — 12:30 Optimización de las data fixtures	3 hrs 30 min
07/01/2018 14:30 — 19:30 Refactorización, realización de pruebas y puesta a punto de las fixtures	5 hrs

08/01/2018 08:30 — 12:00 — 3 hrs 30 min

Probar fixtures y arreglar entidades

08/01/2018 12:00 — 13:15 — 1 hrs 15 min

Probar fixtures y arreglar entidades

08/01/2018 13:45 — 17:00 — 3 hrs 15 min

Comenzar con los endpoints API clientes, realizar tests y comprobar resultados

09/01/2018 15:30 — 17:00 — 1 hrs 30 min

Instalar, configurar y probar Xdebug

10/01/2018 08:30 — 09:00 — 30 min

Empezar a organizar todo y poner las entidades con JMS Serializer

10/01/2018 09:45 — 13:30 — 3 hrs 45 min

Activar el serializador en los controladores e integrarlo con el view de FOSRest

10/01/2018 14:00 — 15:45 — 1 hrs 45 min

Serialización, JMS y devolución de valores. Intentar hacer que devuelva los códigos correctos cuando no encuentra el objeto.

11/01/2018 09:00 — 12:15 — 3 hrs 15 min

Leer documentación sobre API Platform, realizar pruebas e iniciar la instalación y configuración de FOSUser y lexikjwt

11/01/2018 16:30 — 17:00 — 30 min

Terminar de configurar FOSUser y LexikJWTAuth

13/01/2018 09:30 — 14:00 — 4 hrs 30 min

Realizar pruebas de los bundles FOSUser y LexikJWTAuth

13/01/2018 17:00 — 20:30 — 3 hrs 30 min

Aprener sobre el event dispatcher de Symfony, y ponerlo en práctica en un proyecto a parte

15/01/2018 08:30 — 11:00 — 2 hrs 30 min

Probar API Platform e intentar implementar la versión con Symfony 3.4. Imposible, por lo que se vuelve a la aplicación anterior.

15/01/2018 11:00 — 13:30 — 2 hrs 30 min

Desarrollo endpoints crud cliente

15/01/2018 14:00 — 17:00 — 3 hrs

Desarrollo endpoints crud cliente y tests funcionales asociados

17/01/2018 08:30 — 11:00 — 2 hrs 30 min

Solventar error con LiipFunctionalTestBundle en el que no se podía realizar dependency injection, debido a que el bundle estaba pensado para obtener los servicios de symfony directamente.

17/01/2018 11:00 — 12:45 — 1 hrs 45 min

Corregir los tests para que funcionen con LiipFunctionalTestBundle e implementar los tests que faltaban

17/01/2018 14:00 — 15:30 — 1 hrs 30 min

Intentar comprender symfony flex

17/01/2018 15:30 — 17:00 — 1 hrs 30 min

Intentar realizar un servicio o una abstracción para realizar llamadas a la API

18/01/2018 08:30 — 10:45 — 2 hrs 15 min

Instalar y configurar Symfony Flex y API Pack

18/01/2018 10:45 — 13:30 — 2 hrs 45 min

Instalar Symfony Flex con Api core y todas sus dependencias

18/01/2018 14:00 — 15:15 — 1 hrs 15 min

Arreglar problemas con incompatibilidades y aprender a usar Flex

18/01/2018 15:15 — 17:00 — 1 hrs 45 min

Lidiar con los problemas a la hora de que LexikJWT devolviera el token. Finalmente el problema residía en que usaba form_login en vez de json_login.

19/01/2018 08:30 — 11:45 — 3 hrs 15 min

Instalar y configurar API Pack, FOS Rest, Lexik y corregir todos los errores.

19/01/2018 11:45 — 13:30 — 1 hrs 45 min

Configurar entidades y probarlas con fixtures

19/01/2018 14:00 — 15:00 — 1 hrs

Intentar configurar subrecursos

20/01/2018 09:00 — 14:00 — 5 hrs

Realizar pruebas de configuración y de integración con API Platform: recursos, subrecrursos, operaciones adicionales...

20/01/2018 17:00 — 20:00 — 3 hrs

Continuar con las pruebas. Búsqueda de información en internet, GitHub, Slack y demás para ayudar en la configuración.

21/01/2018 09:00 — 13:00 — 4 hrs

Buscar e intentar implementar las diferentes formas de documentar la API en API Platform

21/01/2018 16:00 — 19:00 — 3 hrs

Consulta e implementación de dependencias de campos entre entidades

22/01/2018 08:30 — 13:30 — 5 hrs

Ayuda a Ander a debuggear, configuración de subrecursos y mapeo de entidades

22/01/2018 14:00 — 17:00 — 3 hrs

Mapeo de entidades, corrección de error de apache por el cual eliminaba el header de autenticación y no dejaba a Ander hacer llamadas a dev.clepsydra.com

23/01/2018 08:30 — 09:00 — 30 min

Documentar la API correctamente

23/01/2018 09:00 — 13:30 — 4 hrs 30 min

Arreglar servidor: faltaba añadirle CORS

23/01/2018 14:00 — 17:00 — 3 hrs

Configuración de Behat y los tests

24/01/2018 08:30 — 11:15 — 2 hrs 45 min

Desarrollar endpoint personalizado para obtener los proyectos de un usuario

24/01/2018 15:45 — 17:00 Poner los docs de la api forma que se adapten a lo que se especificó en el custom operation de la api	1 hrs 15 min
25/01/2018 08:30 — 11:45 Intentar realizar el subresource para poder obtener los proyectos de un usuario en concreto. Finalmente parece ser que con el annotation bastaba, a pesar de que estaba intentando crear mi propio repositorio de entidades.	3 hrs 15 min
25/01/2018 11:45 — 13:30 Configurar Behat e intentar ponerlo en marcha	1 hrs 45 min
25/01/2018 14:00 — 16:00 Intentar configurar el entorno test para Symfony y poder crear y deshacer las bases de datos. Parece ser que de momento, a parte de soluciones más complicadas que implican crear un fichero bootstrap y demás, lo más fácil por el momento es cambiar entre ficheros .env	2 hrs
25/01/2018 16:00 — 17:00 Configurar Behat con las fixtures, doctrine, y demás	1 hrs
26/01/2018 08:30 — 11:45 Configuración de Behat con Fixtures y Base de datos.	3 hrs 15 min
29/01/2018 10:15 — 13:30 Preparar tests con Behat y añadir fixtures a la base de datos	3 hrs 15 min
29/01/2018 14:00 — 17:00 Corregir CI/CD para la automatización de los tests	3 hrs
30/01/2018 08:30 — 10:15 Solucionar gitlab-ci: no hacía deploy a dev correctamente, ya que al no hacer composer install ni generar las keys con openssl el dev el proyecto se quedaba a medias.	1 hrs 45 min
30/01/2018 10:15 — 13:30 Configurar Behat, editar campos para devolver el ID de usuario	3 hrs 15 min
30/01/2018 16:15 — 17:00 Preparar tests con Behat	45 min
31/01/2018 11:00 — 12:00 Enviar el ID de usuario cuando el usuario hace login	1 hrs
31/01/2018 12:00 — 12:30 Solucionar problema de deploy. Errores en permisos.	30 min
31/01/2018 12:30 — 13:45 Implementar el enviar los roles cuando el usuario se autentifica	1 hrs 15 min
31/01/2018 14:00 — 17:00 Crear las features para el resto de entidades	3 hrs
Total	**155 hrs**

Febrero

Created by Mikel Alejo

Projects DESARROLLO WEB CLEPSYDRA
Users Mikel Alejo
Time interval 01/02/2018 — 28/02/2018

Total	**95 hrs 30 min**
DESARROLLO WEB CLEPSYDRA	**95 hrs 30 min**
DESARROLLO WEB (DWC-1-2)	95 hrs 30 min
02/02/2018 08:30 — 13:00	4 hrs 30 min
Solucionar problema con relación OneToOne entre Proyecto y Presupuesto. Al final se decide usar ManyToOne por compatibilidad en un futuro.	
02/02/2018 13:00 — 13:30	30 min
Devolver ID de usuario cuando se pide la lista de usuarios y camibar entidad de Estimate para que su nombre sea un string y no un entero.	
03/02/2018 09:00 — 13:00	4 hrs
Investigar sobre el error de OneToOne, intentar replicarlo y buscar soluciones	
03/02/2018 15:30 — 19:30	4 hrs
Intentar encontrar el porqué en ciertas ocasiones el OneToOne falla y en otras no. Finalmente se deja de lado.	
05/02/2018 08:30 — 12:30	4 hrs
Añadir tests a las entidades que faltan.	
06/02/2018 08:30 — 10:15	1 hrs 45 min
Terminar de crear los tests para todas las entidades	
06/02/2018 10:15 — 11:15	1 hrs
Cambiar visibilidad de los campos del usuario, para permitir poder modificar y obtener distintos datos	
06/02/2018 11:15 — 12:30	1 hrs 15 min
Intentar limitar la profunidad con la que se muestran algunas relaciones.	
08/02/2018 09:15 — 12:30	3 hrs 15 min
Intentar configurar mailer con fos user	
09/02/2018 09:00 — 12:30	3 hrs 30 min
Modificar FOSUserBundle para que funcione con apis rest	
10/02/2018 11:30 — 14:00	2 hrs 30 min
Quitar FOSUserBundle y aprender sobre un custom user provider	
10/02/2018 16:00 — 20:00	4 hrs
Crear un custom user provider y probar que todo funciona bien	
11/02/2018 13:45 — 19:45	6 hrs

Finalizar el custom user provider	
12/02/2018 08:30 — 10:15 Implementar sistema invitaciones con event subscribers, generación de tokens y demás	1 hrs 45 min
12/02/2018 11:00 — 12:30 Seguir la implementación del sistema de tokens	1 hrs 30 min
13/02/2018 08:30 — 12:30 Implementación de sistema de invitaciones, prototipado del nuevo dominio y reunión para la modificación de dominio.	4 hrs
14/02/2018 08:30 — 12:30 Modificar el dominio y las fixtures para comprobar que ha sido modelado correctamente	4 hrs
15/02/2018 08:30 — 13:30 Modificar dominio y finalizar con los tests	5 hrs
16/02/2018 08:30 — 12:30 Intentar arreglar el que los tests pedían demasiada memoria para realizarse.	4 hrs
19/02/2018 08:30 — 11:15 Añadir rutas de sub recursos a las entidades	2 hrs 45 min
19/02/2018 12:15 — 12:30 Implementar confirmación usuario	15 min
20/02/2018 08:30 — 12:30 Corregir algunos tests que fallaban y aprender sobre JSON Schema.	4 hrs
22/02/2018 08:30 — 12:30 Crear validación: formularios, entidades, subscribers, listeners, etc.	4 hrs
23/02/2018 08:30 — 12:30 Desarrollo de confirmación de usuario con DTOs y suscripciones	4 hrs
24/02/2018 09:00 — 13:00 Información sobre DTOs, documentar los DTOs en la API	4 hrs
25/02/2018 16:00 — 20:00 Intentar implementar alternativas a los DTOs con entidades. Finalmente se ve que es mejor realizarlo con los susodichos DTOs.	4 hrs
26/02/2018 08:30 — 12:30 Terminar la validación del usuario y empezar con los tests	4 hrs
27/02/2018 08:30 — 12:30 Finalizar la implementación probando los tests y la automatización de la ejecución de los mismos en gitlab ci	4 hrs
28/02/2018 08:30 — 12:30 Corregir y refactorizar los tests	4 hrs
Total	**95 hrs 30 min**

Marzo

Created by Mikel Alejo

Projects DESARROLLO WEB CLEPSYDRA
Users Mikel Alejo
Time interval 01/03/2018 — 28/03/2018

Total	**118 hrs**
DESARROLLO WEB CLEPSYDRA	**118 hrs**
DESARROLLO WEB (DWC-1-2)	113 hrs 45 min
01/03/2018 08:30 — 09:15 Implementar filtro de usuarios activos	45 min
01/03/2018 09:15 — 10:30 Serialización de las entidades para devolver ciertos datos	1 hrs 15 min
02/03/2018 08:30 — 11:00 Implementar y añadir los tests a los cambios de serialización pedidos por el desarrollador de la interfaz	2 hrs 30 min
03/03/2018 09:00 — 12:00 Refactorización de los tests con esquemas JSON	3 hrs
03/03/2018 15:30 — 20:30 Refactorización de los tests con esquemas JSON	5 hrs
04/03/2018 10:30 — 14:30 Refactorización de los tests con esquemas JSON	4 hrs
04/03/2018 15:00 — 19:30 Refactorización de los tests con esquemas JSON	4 hrs 30 min
05/03/2018 08:30 — 10:30 Cambiar serialización y añadir nombre del usuario también cuando se piden los tasks. Permitir al manager crear y activar el usuario.	2 hrs
05/03/2018 10:30 — 12:30 Refactorizar mailer para mejorar la mantenibilidad del código	2 hrs
06/03/2018 08:30 — 12:30 Continuar con desarrollo del sistema de verificación de usuarios y cambio de contraseña	4 hrs
07/03/2018 08:30 — 12:30 Continuar con el sistema de verificación de usuarios y con sus tests	4 hrs
08/03/2018 08:30 — 12:30 Finalizar validación de correo e implementar el cambio de contraseña	4 hrs
09/03/2018 08:30 — 12:30 Finalizar la verificación por email	4 hrs

11/03/2018 09:00 — 13:00 4 hrs

Aprender sobre Symfony Security

11/03/2018 16:00 — 19:00 3 hrs

Probar lo aprendido sobre Symfony Security en un proyecto aparte

12/03/2018 08:30 — 09:00 30 min

Instalar y configurar EasyCodingStandardBundle

12/03/2018 09:00 — 10:00 1 hrs

Añadir la funcionalidad de enviar email de verificación cuando el usuario cambia el email

12/03/2018 10:00 — 10:15 15 min

Solucionar problema apache con el front de ander

12/03/2018 10:15 — 10:30 15 min

Solucionar error por el cual los emails no se enviaban en formato HTML

12/03/2018 10:30 — 10:45 15 min

Enviar estado de lock de la cuenta

12/03/2018 10:45 — 12:15 1 hrs 30 min

Añadir funcionalidad de refresco del token

12/03/2018 12:15 — 12:30 15 min

Arreglar el error de que el usuario no recibía una respuesta correcta cuando proporcionaba un token de verificación de la cuenta correcto. El error estaba en que en el UserSubscriber no se devolvía ningún código de error ni datos.

13/03/2018 08:30 — 09:00 30 min

Mirar si es posible ahorrar algunas persistencias con doctirne en el user subscriber. No es posible ya que de lo contrario no funciona.

13/03/2018 09:00 — 10:15 1 hrs 15 min

Implementar la funcionalidad de mandar la URL de la interfaz en los emails y sus tests correspondientes.

13/03/2018 10:15 — 12:30 2 hrs 15 min

Intentar implementar la subida de imágenes de perfil

14/03/2018 08:30 — 12:30 4 hrs

Implementar la restricción de los recursos por usuario

15/03/2018 08:30 — 12:30 4 hrs

Implementar restricciones de acceso. Configurar correctamente el serializador y el normalizador para devolver únicamente los datos pertinentes, sin desvelar ninguna entidad más al usuario como estimates, bags, etc.

16/03/2018 08:30 — 10:00 1 hrs 30 min

Añadir control de acceso

16/03/2018 10:15 — 12:30 2 hrs 15 min

Añadir control de acceso

17/03/2018 09:00 — 13:00 4 hrs

Añadir tests de control de acceso

17/03/2018 15:00 — 20:00 5 hrs

Añadir tests de control de acceso

18/03/2018 15:00 — 19:00 4 hrs

Añadir tests de control de acceso

19/03/2018 08:30 — 12:30 4 hrs

Finalizar los tests de la restricción de acceso y redireccionar los errores a los ficheros log

20/03/2018 08:30 — 10:15 1 hrs 45 min

Solucionar problema de que no se puede quitar un predefined project de un proyecto sin un nombre

20/03/2018 10:15 — 10:30 15 min

Arreglar el fallo de que la fecha de registro no se guarda en la base de datos cuando el gestor crea el usuario

20/03/2018 10:30 — 12:30 2 hrs

Substituir errores manuales por excepciones, incluir createdAt cuando el manager crea una cuenta, y arreglar el que los proyectos se les podía quitar el predefinedProject porque la variable de setPred no era ?PredefProj

21/03/2018 08:30 — 10:15 1 hrs 45 min

Añadir funcionalidad de recuperación de contraseña

21/03/2018 10:15 — 11:00 45 min

Arreglar error: no se puede eliminar el proyecto predefinido de un proyecto porque ponía los name y category a null, y por lo tanto la aserción fallaba

21/03/2018 11:00 — 12:30 1 hrs 30 min

Modificar usuario test de las fixtures para que sea supermanager en vez de test e implementar que el proyecto esté abierto o cerrado.

22/03/2018 08:30 — 09:45 1 hrs 15 min

Terminar la implementación y los tests de cerrar y reabrir proyectos

22/03/2018 10:15 — 10:30 15 min

Enviar el project id cuando el usuario pide sus projects

22/03/2018 10:30 — 10:45 15 min

Quitar date de los task

24/03/2018 09:00 — 13:00 4 hrs

Intentar refactorizar los tests

24/03/2018 15:00 — 20:00 5 hrs

Intentar refactorizar los tests. Dejarlo porque la forma en la que estaban implementados es menos confusa.

26/03/2018 08:30 — 09:00 30 min

Hacer descripciones de las tasks nullables

26/03/2018 09:00 — 09:30 30 min

Actualizar nombre de los proyectos cuando se cambia el nombre de un proyecto predefinido

26/03/2018 09:30 — 11:15 1 hrs 45 min

Añadir funcionalidad de tasks favoritas

26/03/2018 11:15 — 11:30 Normalizar incremento de las variables índice de los loop	15 min
26/03/2018 11:30 — 12:30 Añadir restricción de nombre de usuario	1 hrs
27/03/2018 10:00 — 12:15 Actualizar aplicación, mirar posible problema de fecha de tasks —el problema estaba en no haber actualizado el esquema de la base de datos de develop— y mirar cómo filtrar los tasks de un usuario en un proyecto	2 hrs 15 min
28/03/2018 08:30 — 12:30 Intentar configurar el deployer	4 hrs
GESTION (DWC-1-1)	4 hrs 15 min
01/03/2018 10:30 — 12:30 Reunión con Imanol	2 hrs
02/03/2018 11:00 — 12:30 Reunión con Ioritz para las líneas futuras del proyecto	1 hrs 30 min
23/03/2018 11:45 — 12:30 Explicar a Joseba cómo funciona el proyecto	45 min
Total	**118 hrs**