Aplicación gestión de tareas para un grupo de personas

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Joan Juan Belda

**Tutor**: José Angel Carsí Cubel

2020-2021

Resumen

El proyecto aborda el desarrollo de una aplicación para la creación, la gestión y el reparto de tareas entre los miembros de un grupo, así como el control de la realización de cada tarea por parte de los usuarios. Para el desarrollo de la aplicación se usará el framework o el subconjunto de sistemas llamado MEAN (acrónimo para: MongoDB, Express, Angular, Node).

**Palabras clave:** angular, gestión, tareas, mean, mongo.

Tabla de contenido

[1. Introducción 7](#_Toc74676309)

[1.1. Motivación 7](#_Toc74676310)

[1.2. Objetivos 8](#_Toc74676311)

[1.3. Impacto esperado 8](#_Toc74676312)

[1.4. Estructura 9](#_Toc74676313)

[2. Estado del arte 9](#_Toc74676314)

[2.1. Situación actual de la tecnología 10](#_Toc74676315)

[2.1.1. Remember the milk 10](#_Toc74676316)

[2.1.2. Todoist 10](#_Toc74676317)

[2.1.3. Trello 10](#_Toc74676318)

[2.1.4. Jira 11](#_Toc74676319)

[2.2. Crítica al estado del arte 11](#_Toc74676320)

[2.3. Propuesta 12](#_Toc74676321)

[2.4. Conclusiones 13](#_Toc74676322)

[3. Metodología 13](#_Toc74676323)

[3.1. Metodología 13](#_Toc74676324)

[3.2. Enfoque 16](#_Toc74676325)

[3.3. Framework 16](#_Toc74676326)

[3.4. Control de versiones 18](#_Toc74676327)

[3.5. Integración/distribución continua 18](#_Toc74676328)

[4. Análisis y especificaciones de requisitos 19](#_Toc74676329)

[4.1. Actores 20](#_Toc74676330)

[4.2. Casos de uso 20](#_Toc74676331)

[4.3. Requisitos funcionales 29](#_Toc74676332)

[4.4. Requisitos no funcionales 31](#_Toc74676333)

[5. Diseño 31](#_Toc74676334)

[5.1. Modelado de la base de datos 32](#_Toc74676335)

[5.2. Arquitectura 34](#_Toc74676336)

[5.3. Interfaz (prototipos) 36](#_Toc74676337)

[6. Implementación 41](#_Toc74676338)

[6.1. Contexto tecnológico 41](#_Toc74676339)

[6.2. Tecnologías 41](#_Toc74676340)

[6.3. Entornos 41](#_Toc74676341)

[6.4. Integración continua 41](#_Toc74676342)

[6.5. Frontend 41](#_Toc74676343)

[6.6. Backend 41](#_Toc74676344)

[6.7. Base de datos 41](#_Toc74676345)

[7. Pruebas 42](#_Toc74676346)

[7.1. Pruebas unitarias 42](#_Toc74676347)

[7.2. Pruebas de integración 42](#_Toc74676348)

[7.3. Pruebas E2E 42](#_Toc74676349)

[8. Conclusión 42](#_Toc74676350)

[8.1. Relación proyecto-estudios cursados 42](#_Toc74676351)

[8.2. Trabajos futuros y líneas de mejora 42](#_Toc74676352)

# Introducción

El proyecto actual tiene como objetivo el desarrollo de una aplicación para la gestión de las tareas en un determinado grupo de usuarios para permitir a los usuarios una gestión de las tareas rápida y eficaz, así como poder ver en cualquier momento una imagen del estado actual de las tareas que tienen asignadas y permitir al gestor del grupo ver el estado actual de lo que se está haciendo por parte de los demás usuarios. También se ofrece la posibilidad al gestor de obtener informes del tiempo que se ha dedicado a cada tarea durante un período determinado de tiempo.

El proceso de desarrollo de la aplicación está enfocado en la integración continua, por lo que se detallará a lo largo del proyecto como se ha implementado cada fase de la integración como la gestión de versiones y entornos, los tipos de test implementados para cada parte de la aplicación y los métodos de subida a los diferentes entornos. Todo esto para intentar simular y entender cómo debería estar implementado todo el proceso de desarrollo y mantenimiento de un software en un entorno real.

## Motivación

La realización de este proyecto surge a partir de la necesidad que tenía para organizar la cantidad de tiempo de las tareas que estaba gestionando en mi puesto de trabajo.

Actualmente las aplicaciones de gestión de tareas que hay en el mercado, al menos las gratuitas, solamente te permiten ver las tareas que tienes pendientes, las que estás haciendo y las que has hecho ya. Con puedes llegar a hacerte una imagen de la carga de trabajo que tienes. El problema surge cuando las tareas tienen una estimación de horas y una fecha de entrega determinada, además de tener que mandar informes diarios del tiempo que has dedicado a cada tarea. Esta última parte es en la que no se enfocan la mayoría de las aplicaciones y que necesitaba automatizar como fuese, ya que consumía una cantidad ingente de tiempo gestionándolo “a mano”.

Por todo esto decidí desarrollar una aplicación con la idea de facilitar la gestión del tiempo de cada una de las tareas y poder ver cuánto tiempo lleva un usuario en una tarea determinada y si queda mucho para que se pase del tiempo que se estimó en el momento de su creación.

A parte de lo anterior, conforme iba todo cogiendo forma me di cuenta de que el proceso de desarrollo no era práctico, además de que, por experiencias laborales previas, me di cuenta de la gran importancia del apartado de test en un software y de la facilidad y ahorro de tiempo que supone implementar todo el proceso de desarrollo en la integración continua. Así que decidí darle otro punto de enfoque y aprender sobre el test de software y a la creación de un entorno de desarrollo real, con una buena gestión de versiones y entornos reales.

En resumen mi motivación surge al ver la cantidad de tiempo que tenía que invertir en dos de los apartados que creo que tiene cualquier empresa de desarrollo de software, que son: la gestión de las tareas y el tiempo que se le dedica a cada una, así como el reporte de horas al mánager del proyecto y el reporte de horas para facturar por parte del mánager del proyecto al cliente, el tiempo y la poca seguridad que implica probar el software manualmente y el tiempo y la poca seguridad de subir los cambios manualmente a los respectivos entornos.

## Objetivos

El objetivo principal es la realización de una aplicación que permita gestionar el tiempo dedicado a cada tarea. Con una interfaz sencilla y usable.

Otro de los objetivos fundamentales es el buen uso y la buena conminación de todos los elementos que componen el framework MEAN, que es el entorno en el que desarrollaremos la aplicación. Este framework se compone de la base de datos no relacional MongoDB, la parte de servidor Express y la parte visual Angular.

Otro de los objetivos importantes en los que se enfoca el proyecto es en la parte de pruebas del software. En esta parte nos centraremos en la implementación de las pruebas de cada una de las partes que componen la aplicación. Se crearán pruebas unitarias para cada una de las partes, pruebas de integración para cada par de componentes que se usen conjuntamente y pruebas e2e para simular y probar cómo reacciona el sistema delante de un usuario normal en un entorno real.

Por último, también se quiere desarrollar todo el proyecto en un proceso de integración continua, para esto tendremos que definir los entornos en los que implantaremos la aplicación (desarrollo, preproducción y producción), así como estructurar los entornos en la gestión de versiones y ejecutar una serie de procesos (pruebas y subida) cuando se suba una versión en un entorno determinado.

## Impacto esperado

Mediante esta aplicación se espera reducir el tiempo que se invierte por parte de los usuarios de un grupo o proyecto en la gestión de sus tareas y el tiempo que invierten en cada una para que cada uno tenga una imagen clara de la carga de trabajo y de lo que se tiene que entregar en cada momento.

Además, se espera que el manager de un grupo pueda tener una foto del estado actual de las tareas del grupo, así como poder obtener reportes del tiempo que se ha invertido en un intervalo de tiempo en el grupo o en una determinada tarea. Así como ver la carga de trabajo que lleva cada uno de los usuarios.

## Estructura

El presente proyecto se ha estructurado en bloques que representan el proceso habitual en el desarrollo de software:

* En este primer punto se ha explicado las motivaciones que han provocado la realización de la aplicación, los objetivos que nos hemos puesto y el impacto que queremos que tenga en los usuarios que la usen.
* El segundo punto nombraremos las diferentes aplicaciones que ya hay disponibles en el ámbito de la que hemos desarrollado y veremos que aportan y que no. Y finalmente nombraremos que propone nuestra aplicación en frente de las anteriores.
* El tercer punto se explica la metodología de desarrollo de software que se ha seguido, así como el framework o conjunto de subsistemas en los que hemos desarrollado la aplicación.
* El cuarto punto se enfoca en el análisis del problema previo que tenemos y recopilaremos los requisitos que resuelven la problemática.
* El quinto punto explica la arquitectura que hemos seguido, así como el modelado de la base de datos (el conjunto de objetos que participan en la resolución del problema) y los prototipos de la interfaz.
* El sexto punto ya pasa a los detalles técnicos del desarrollo de la aplicación, como las tecnologías que hemos usado para el desarrollo, en que se compone cada capa de la aplicación y como hemos implementado la integración continua.
* El séptimo punto detalla todas las pruebas que hemos implementado para asegurar la robustez y la calidad del software que estamos desarrollando.

El último punto reflexionaremos sobre la relación de este proyecto con los estudios cursados y propondremos algunas mejoras para ampliar la funcionalidad de la aplicación.

# Estado del arte

Existen múltiples aplicaciones en el mercado para realizar la gestión de tareas, tanto para un solo individuo como para un grupo de usuarios. El problema o la carencia de funcionalidades que vemos en la mayoría de las aplicaciones es la falta de una gestión de tiempo orientada a lo que sería un proyecto real, es decir, en la necesidad de tener un seguimiento de los recursos (en este caso temporales, medidos en horas) que se espera que se le vaya a dedicar a una tarea, en la necesidad de tener una imagen de los recursos que se han aportado o se están aportando en cada una de las tareas y el reporte de las horas que le ha dedicado un usuario o un grupo de usuarios a una tarea concreta o las horas que ha invertido un usuario en sus distintas tareas.

Cabe señalar que actualmente las herramientas de gestión de tareas se pueden dividir en dos tipos según el sistema de trabajo:

-Método GTD "Get the things done": la organización de las tareas sigue un orden de cinco pasos, recopilar y organizar las tareas en contenedores, procesar las tareas según su prioridad, organizarlas según sus necesidades, revisarlas y ejecutarlas.

-Sistema Podomoro: establece tiempos para la ejecución de las tareas (estimaciones) y fechas de entrega según los objetivos que el gestor crea convenientes.

## Situación actual de la tecnología

En este apartado se describen varias aplicaciones dedicadas a la gestión de tareas que están en el mercado.

### Remember The Milk

La primera aplicación que hemos probado es Remember The Milk. Se trata de un gestor de tareas enfocado al método GTD para organizar nuestra propia tarea.

Su principal característica es la integración con diversos entornos como: Gmail, Twitter, Hangouts, Skype. Todos estos se podrían vincular con Remember The Milk y poder enviarnos recordatorios de las tareas que tenemos en marcha. También podemos disfrutar de esta aplicación en varios dispositivos, ya que cuenta con aplicación web, y aplicación para dispositivos móviles, tanto Smartphone como relojes.

### Todoist

Probablemente sea la primera que nos encontremos al buscar un gestor de tareas. Según los requerimientos del usuario se puede usar según el método GTD o el sistema Podomoro.

Las características más importantes son la capacidad de programar tareas repetitivas, por ejemplo, una tarea rápida que tengamos que hacer cada día. Y el tablero Kanban que te permite tener una imagen clara de tu carga de trabajo.

Una funcionalidad que hemos echado de menos es la capacidad de estimar cuanto tiempo podría tardar un usuario a realizar una determinada tarea y la capacidad de poder ver el historial de tareas. Esta última funcionalidad solamente se encuentra en el plan de pago. Podríamos decir que esta herramienta está más enfocada a la gestión personal de las tareas que a la gestión de un proyecto real.

### Trello

Trello es un gestor de tareas que usa el sistema Podomoro para organizar las tareas de un grupo. Su funcionamiento está basado en los paneles.

La principal ventaja es la capacidad de adjuntar o crear tareas en base a cualquier cosa, como una imagen, un correo electrónico que no haya llegado pidiendo una determinada cosa etc. También cuenta con la herramienta de la creación de informes en los que se resume, usando gráficos de tarta, barras y lineales, las tareas que se han hecho en un determinado periodo de tiempo.

Aunque, a priori, parezca muy completa, seguimos echando en falta la capacidad de estimar el tiempo que vamos a dedicar a la tarea, no solo indicar la fecha de entrega y la elaboración de informes sobre las horas que han invertido los usuarios en una o varias tareas.

### Jira

Jira es un gestor de tareas que sigue el sistema Podomoro y es el más completo que hemos encontrado.

Jira es una herramienta en línea para la gestión de tareas de un proyecto, el seguimiento de sus errores e incidencias y para la gestión operativa de proyectos.

Su principal característica es su organización de flujos de trabajo. Esta característica permite que una organización pueda definir el flujo que tiene que seguir una tarea, y actuar en esa tarea según su criterio, definiendo unas actuaciones en cada una de sus fases.

Tiene las principales características que encontramos en las otras aplicaciones: creación de paneles, avisos a nuestro correo, gestión de las fechas de entrega, gestión de prioridades y gestión de usuarios del proyecto.

Además de poder usar la herramienta como un gestor de tareas en un proyecto, podemos enfocar el uso de la herramienta a un ámbito de soporte. Ya que permite la creación de incidencias por parte de los usuarios. Estas incidencias serán tratadas como tareas realmente, pero tendrán un solicitante y en todo momento podremos contactar directamente con este desde la propia herramienta.

A diferencia de los anteriores gestores, este sí que cuenta con la capacidad de poder estimar los recursos que se le va a dar a una tarea, asignar los recursos que se les va dando y hacer un informe al finalizar la tarea o en cualquier momento del proyecto.

La única pega que podemos ver es la complejidad, pero queda ensombrecida por la gran cantidad de funcionalidades y soluciones que nos aporta la herramienta.

## Crítica al estado del arte

Las aplicaciones que hemos descrito anteriormente, y en general, la mayoría que existen en el mercado tienden a estar o bien muy enfocadas a la gestión de las tareas que lleva un único usuario o bien muy enfocadas a la gestión de las tareas de un proyecto o grupo de usuarios.

En el caso de las que están más enfocadas a la gestión de nuestras propias tareas, suelen seguir el método GDT, es decir, solamente hacen hincapié en el listado de tareas que nos queda por hacer (ordenándolas por prioridad o por fechas de entrega). Con este enfoque se pierde por ejemplo las horas que llevamos dedicadas a cada tarea o un indicador de cuánto tiempo nos queda hasta que lleguemos a las horas que hemos acordado que usaríamos para dar por finalizada la tarea. Por otra parte, también echaríamos en falta la funcionalidad para obtener un informe de las horas que hemos dedicado a cada una de las tareas en una semana o en un intervalo de tiempo determinado.

En el caso de las que están dirigidas a la gestión de las tareas de un proyecto echamos en falta la simplicidad para un usuario de ver únicamente sus tareas y tener una imagen clara de la situación de sus tareas en el proyecto. Las aplicaciones que hemos visto que mejoran esta parte con tableros Kanban o la utilidad de sacar informes de horas no las ofrecen en su plan gratuito.

En conclusión, se echa en falta la gestión de tareas que pueda necesitar una empresa, ya que en estas se tiene mucho en cuenta el poder dar una estimación al cliente del tiempo que va a necesitar una tarea. A todo esto, hay que sumarle la necesidad de poder obtener informes de horas gastadas por parte de los trabajadores en cada tarea o en cada proyecto. Ya que esas horas son al final de lo que cobraría la empresa.

## Propuesta

Como se ha especificado en la introducción, en este trabajo se pretende desarrollar una herramienta para la gestión de tareas tanto para las tareas individuales como para las tareas que llevan a cabo varios trabajadores en un proyecto. A diferencia de las aplicaciones que hay en el mercado, intentaremos enfocar nuestra aplicación para una gestión de las tareas de un proyecto, pero sin perder las características que necesitaría un usuario individual a la hora de gestionarse el mismo sus tareas. Todo esto sin llevar al usuario a perder interés en la aplicación por la alta complejidad que pueda suponer, es decir, simplificarle la gestión. Para conseguir esto, primero vamos a listar lo que necesitaría un usuario gestor de proyecto y lo que necesitaría un usuario que trabajase en un proyecto.

Si nos ponemos en la piel del gestor, a parte de la capacidad para crear, editar, estimar en tiempo, eliminar y repartir las tareas a los usuarios, necesitaría tener la capacidad de ver en todo momento una imagen del proyecto, es decir, las tareas que hay en curso en cada estado, las tareas que están cerca de sobrepasar el tiempo que se ha estimado en ellas y las que ya se ha pasado. También podría filtrar todas las tareas por un usuario, para comprobar la carga de trabajo que está soportando actualmente. Por último, también necesitaría tener la capacidad de elaborar informes con las horas que se les ha dedicado al proyecto o a una tarea específica en un intervalo de tiempo determinado.

Si nos ponemos en la piel de un usuario normal, lo más importante sería poder ver el estado o la carga de trabajo que está soportando, así como saber la relación entre la cantidad de tiempo invertido actualmente en las tareas y sus estimaciones (al igual que lo necesitaba saber el gestor de proyecto). También habría que mejorar la forma en la que los usuarios añaden horas a las tareas, ya que añadirlas a mano podría ser un poco tedioso o una pérdida de tiempo, habíamos pensado en que el tiempo se añadiese automáticamente a las tareas. Es decir, cuando el usuario mueva una tarea al estado de “en desarrollo”, se iniciará un temporizador con el tiempo que le va dedicando. Para flexibilizar este tema, el usuario podría editar siempre que quisiese el tiempo dedicado a las tareas. Además, podría obtener informes de las horas que ha dedicado a las tareas en un intervalo de tiempo determinado.

También vamos a hacer que toda la aplicación se mueva en un entorno lo más ágil posible. Para ello hemos elegido crear una SPA ya que, aunque la carga inicial de la aplicación es más costosa, el cambio entre interfaces es mucho más rápido que las páginas webs antiguas, dando una experiencia más agradable al usuario.

## Conclusiones

El mercado ya cuenta con una gran variedad de aplicaciones dedicadas a la gestión de tareas. Hemos descrito las dos tipologías en las que se podrían clasificar y hemos enumerado unos ejemplos de las aplicaciones más populares y con las que ya tenía experiencia. Hemos descrito sus ventajas y sus inconvenientes para ver que podemos aportar a este nicho.

Lo que aportaremos es una aplicación más centrada a la gestión de los tiempos, tanto del tiempo dedicado a las diferentes tareas como el tiempo que se cree que va a costar desarrollar una tarea. Todo esto sin perdernos en complejidades y dándole al usuario una experiencia ágil y simple.

# Metodología

Este apartado se centrará en describir el proceso de desarrollo de software que se ha seguido en este proyecto. Se explicará que metodología y enfoque se han usado, en qué consisten y el porqué de su elección. Al mismo tiempo indagaremos en apartados más técnicos como son la elección del framework de desarrollo, el sistema de control de versiones y l parte de la integración continua y su utilidad respecto a la metodología y el enfoque que hemos seguido.

## Metodología

Nos referimos a metodología de desarrollo de software como al entorno de trabajo que es usado para estructurar, planear y controlar el desarrollo del producto de software y que consisten en un enfoque sobre el proceso en sí del desarrollo y las herramientas, modelos y métodos que asisten al proceso de desarrollo de software.

De entre todos los modelos de desarrollo, nos centraremos en el desarrollo ágil del software, más concretamente en la metodología Scrum.

El desarrollo ágil del software se basa en el desarrollo iterativo e incremental, que en gran medida consiste en repartir el proceso de desarrollo en varias etapas repetitivas (iteraciones) en la que en cada etapa se añaden una porción de las funcionalidades respecto de la totalidad de los requerimientos del proyecto. Explicaremos en más profundidad el desarrollo iterativo e incremental en el siguiente apartado.

Una de las principales características del desarrollo ágil del software es el ciclo de vida de cada iteración, que incluye la planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Cabe mencionar la importancia de finalizar las tareas en cada fase iterativa, ya que el objetivo de esta metodología no es lanzar el producto con toda su funcionalidad, sino ir lanzado versiones del producto finalizadas en cada iteración. Entendiendo finalizadas como versiones funcionales y sin errores, de ahí la importancia de la fase de pruebas en cada iteración.

Con todo esto en mente, la metodología ágil que hemos aplicado en este proyecto es la metodología Scrum.

Se trata de un marco de trabajo sobre la metodología de desarrollo ágil en la que se aplican un conjunto de buenas prácticas para trabajar colaborativamente y obtener el mejor resultado posible. Se basa y se caracteriza por:

* Estrategia de desarrollo incremental. Muy diferente a la planificación y ejecución completa del producto. En cada fase se muestra el resultado al cliente, para que este tome las decisiones necesarias.
* Se solapan las distintas fases del desarrollo, en lugar de realizarlas secuencialmente o en cascada. No se espera que finalice una fase para empezar otra.
* Desarrollo incremental de los requisitos totales del proyecto en bloques temporales cortos y fijos. Dando prioridad al que tiene más valor para el cliente.
* El equipo de trabajo (normalmente pequeños, de 3 a 9 personas cada uno) se sincroniza diariamente y se realizan las adaptaciones o modificaciones pertinentes (Scrum diario).

El proceso de un proyecto con una metodología Scrum constaría de carias fases que vamos a definir y explicar.

En la primera fase de obtendrían los requisitos de toda la aplicación. Es decir, se definirían las funcionalidades que tendría, a priori, sin realizar aún ninguna iteración del proceso, la aplicación final. De todos estos requisitos crearíamos tareas a realizar que dividiríamos en diferentes Sprint según las funcionalidades que queremos ir entregando en las diferentes fases o Sprint. Cabe destacar que las longitudes temporales de los Sprint no deberían superar el mes.

Una vez ya tengamos definidos y planificados los diferentes Sprint con sus diferentes tareas, obtenidas de los requisitos iniciales, el equipo se reunirá para empezar el primer sprint. Cada día se acuerda una reunión llamada Scrum diario, que tiene como objetivo que los diferentes miembros del equipo se mantengan actualizados unos a otros sobre el trabajo de cada uno desde la última reunión, qué problemas han encontrado o prevén encontrar, que hacer y hablar sobre las distintas soluciones que pueden ir aportando los demás miembros del equipo, o discutir sobre las integraciones que pueden haber entre las diferentes partes del proyecto con las otras partes de los otros miembros.

Al finalizar un sprint, el equipo realiza dos eventos, la revisión del sprint, donde se presenta el trabajo realizado, y la retrospectiva del sprint, donde los miembros dejas sus impresiones del sprint superado y se proponen mejoras sobre el proceso.

Los beneficios que hemos visto a la hora de trabajar con esta metodología son los siguientes:

* Capacidad de reacción a los cambios. Al ser un proceso iterativo de un mes, en el que cada mes se entrega una versión al cliente, tenemos el beneficio de obtener rápidamente su opinión sobre el producto presentado y si el producto se ajusta a los requerimientos obtenidos inicialmente. Si hay algunas variaciones con lo que realmente necesita el cliente final no supone un gran problema, ya que aún no hemos entregado la versión definitiva del producto y adaptarlo sería más fácil. Además, al tener una reunión diaria con el equipo de trabajo, se pueden obtener diferentes puntos de vista que pueden ayudar a ver si nos estamos desviando del requerimiento inicial o poder detectar si un requerimiento está mal tomado y pedir una reunión con el cliente final. Pudiendo así detectar de una manera rápida cualquier duda entre el requerimiento que se ha obtenido inicialmente y lo que realmente quería el cliente.
* El cliente puede obtener una versión inicial de la aplicación antes de que esté totalmente terminado. Todo esto en el entorno final en el que se quedará la aplicación final, por lo que evitamos tanto el costoso como la arriesgada subida a producción de un producto final con todas sus funcionalidades. Así mismo, en cada iteración se realizan pruebas de cada nueva funcionalidad.
* Se logra una mayor productividad del equipo, ya que, entre otras razones, se elimina la burocracia y el equipo puede estructurarse de manera autónoma.
* Se puede llegar a conocer con cierto margen la velocidad del equipo en cada sprint, con lo que se consigue poder decir al cliente una predicción de los tiempos en el que estará cada iteración y cada nueva funcionalidad.
* Divide y vencerás. Con esta metodología se puede estructurar y dividir el proyecto final en diferentes fases claramente separadas por la funcionalidad que representan. Esto consigue no saturar a los miembros del equipo con toda la información de un proyecto completo e ir centrándose en la porción de funcionalidades que representa cada sprint.

En este proyecto, aun no siendo un equipo completo de personas, pues se ha hecho de forma individual. Sí que hemos podido experimentar los beneficios de esta metodología referentes a la capacidad de ir lanzando versiones del proyecto de una forma incremental y obtener una mejora en la productividad al dividir todo el total de tareas en diferentes iteraciones. En el caso de las reuniones como los Scrum diarios, se ha sustituido por una revisión de las tareas que había pendientes en ese sprint y ver si las soluciones que si iban implantando satisfacían lo esperado en un primer momento.

## Enfoque

En este apartado vamos a explicar en qué consiste el enfoque que hemos dado a la metodología de desarrollo de software que hemos seguido, porque lo hemos seguido y los beneficios que hemos visto.

El enfoque que hemos dado es el desarrollo iterativo e incremental. Se basa en dividir las tareas agrupándolas en pequeñas etapas repetitivas (iteraciones). El objetivo de cada etapa es sumar una serie de funcionalidades ya planificadas anteriormente al producto final (incremental), así, el punto de vista del cliente es de una primera entre con lo que más necesita este, y una sucesión de entregas que van ampliando la funcionalidad de la aplicación a lo que se acordó inicialmente. Todo esto con las diferentes modificaciones que proponga el cliente en cada etapa.

Este proceso de divide en dos etapas principales:

* Etapa de inicialización: Se presenta una versión inicial de la aplicación. La meta de esta etapa es crear un producto con el que el cliente pueda interactuar y retroalimentar todo el proceso que seguirá.
* Etapa de iteración: El análisis de una iteración se basa en la retroalimentación por parte del cliente y en el análisis de las funcionalidades de las que dispone la aplicación.

## Framework

Este apartado tiene como objetivo explicar el entorno de trabajo que hemos elegido para el desarrollo de la aplicación. En desarrollo de software, nos referimos a entorno de trabajo a la estructura conceptual y tecnológica que nos sirven como base para la organización y desarrollo de software. Suele incluir una serie de programas y librerías, que, actuando conjuntamente nos facilitan llegar al objetivo. Más tarde, explicaremos también el control de versiones y la integración continua.

En este proyecto se ha elegido como entorno de trabajo el MEAN Stack (acrónimo para: MongoDB, ExpressJS, Angular 2, NodeJS). Se trata de un framework que tiene como objetivo la creación de páginas webs dinámicas basándose en el lenguaje de programación JavaScript. Cada subsistema del MEAN Stack es de código abierto y gratuito. Está compuesto por los siguientes elementos:

* MongoDB: Se trata de un sistema de base de datos NoSQL. La principal característica es que este sistema no usa SQL como leguaje principal de consultas ni requiere una estructura fija de tablas. Tiene una estructura en documentos JSON y una baja cantidad de restricciones y relaciones en comparación con un sistema SQL, todo esto le aporta una mayor velocidad y escalabilidad en las operaciones de lectura, pero con el coste de aumentar la gestión y coste de las operaciones de escritura, ya que, al poder tener datos duplicados en otros documentos, al modificarlos habría que modificarlos en todos.
* NodeJS: Es un entorno en tiempo de ejecución asíncrona multiplataforma basado en el lenguaje de programación JavaScript. Fue creado con el objetivo de ser útil en la creación de servidores altamente escalables.
* ExpressJS: Se trata de un módulo básico que se puede incorporar a un servidor NodeJS que nos proporciona la capacidad de facilitar y manejar las diferentes peticiones HTTP y establecer los puertos por los que vamos a leer.
* Angular 2: Angular es un framework para el desarrollo de aplicaciones web desarrollada en TypeScript (hereda de JavaScript añadiendo la gestión de objetos). Su objetivo es facilitar el desarrollo de aplicaciones dinámicas de una página con capacidad de Modelo Vista Controlador, así como facilitar las pruebas. Angular está basado en componentes, este elemento es el principal con el que trabajaremos, estos componentes incluyen una parte de la vista total de la aplicación y está ligada a una parte de lógica, en la que trataremos todos los datos. Están pensadas para ser reutilizadas a lo largo de toda la aplicación. También cuenta con unos elementos llamados servicios, que contienen solamente partes lógicas que se encargan de la obtención, administración y gestión de datos. En base, son los que proveen a los componentes de datos, mientras que los componentes son los encargados de cómo mostrarlos y cómo manejarlos interactuando con otros componentes.

Se ha elegido este entorno de trabajo porque es uno de los más populares actualmente, es propiedad de Google, por lo que tiene un gran soporte. A parte de esto, estamos hablando de que cubre todo el ciclo completo de desarrollo utilizando únicamente JavaScript, admite la arquitectura Modelo Vista Controlador para que el proceso fluya sin problemas, la estructura del proyecto en componentes y servicios mantiene el desarrollo y el mantenimiento muy organizado, tiene una extensa cantidad de herramientas, vistas y librerías ya hechas y está integrado con varias herramientas de pruebas, tanto para pruebas unitarias como para pruebas E2E.

## Control de versiones

Otra de las partes imprescindibles en todo proyecto software, más si se trabaja en equipo, es el sistema de control de versiones que se emplee. El sistema de versiones nos da la capacidad de gestionar los diversos cambios que se realizan, quien las ha realizado y que se ha modificado exactamente. Así como la capacidad de poder volver hacia atrás en cualquier momento (por algún error en el producto o por la eliminación de algún requisito por parte del cliente). También crear, gestionar y mezclar ramas del desarrollo, con lo que se facilita bastante el proceso de añadir una mejora o solucionar un error a una versión concreta ya desplegada y funcional. En este caso el desarrollador solamente tendría que crear una rama en base a otra, trabajar en esa rama que ha creado, testearla para asegurarse de que no exista ningún error y mezclar su rama con la de destino. Al hecho de mezclar se le conoce comúnmente como *merge* si mandamos nuestros cambios a la rama principal, o *pull request* si obtenemos los cambios de la rama original a la que hemos desarrollado. Más adelante explicaremos la parte de la integración continua y su implicación en esta parte concreta del proceso de desarrollo.

En este proyecto hemos elegido Git como gestor de control de versiones por las siguientes razones:

* Fuerte apoyo al desarrollo no lineal, es decir, una sencilla y rápida gestión de las ramas y el mezclado de versiones que hemos explicado anteriormente.
* Gestión distribuida, cada desarrollador del equipo tiene una copia local del proyecto, o repositorio, así como de todas las ramas y el historial de cambios de cada archivo. Permite tener un repositorio remoto en un servidor al que poder subir los cambios que se van haciendo en local.
* Compatibilidad con la plataforma online y gratuita GitHub. GitHub es una plataforma que permite el alojamiento de proyectos con sistema de gestión de versiones Git. Una de las principales características es su integración con varios sistemas que se encargan de la integración continua que explicaremos más adelante.

Cabe mencionar que hemos usado una herramienta para la gestión de las tareas y los Sprint llamada ZenHub. Se trata de una aplicación web que nos permite gestionar las tareas o agrupaciones de tareas (Sprint) de un repositorio que tengamos en GitHub. Las tareas que creemos son interpretadas como incidencias de GitHub, y se pueden añadir comentarios a cada tarea y comentar un enlace del *commit* que hagamos para que al pulsar nos mueva directamente a los cambios que se han hecho.

## Integración/distribución continua

La integración/distribución continua es un método para distribuir aplicaciones a los clientes con una frecuencia más alta gracias al uso de la automatización de tareas en las etapas del desarrollo de aplicaciones. Se trata de una solución para los problemas que pueda generar la integración y distribución de las nuevas funcionalidades que se van desarrollando y su puesta en marcha en los equipos de operaciones. Concretamente, estas herramientas incorporan la automatización y el control permanente en todo el ciclo de vida de las aplicaciones, desde las etapas de integración y pruebas hasta las de distribución e implementación.

Nos ha parecido interesante añadir estas herramientas o esta metodología al proceso de desarrollo porque en proyectos grandes donde participen un número elevado de personas (incluso de empresas diferentes), el proceso de hacer unos cambios al código, o añadir nuevas funcionalidades y después hacer las pruebas unitarias, de integración y de E2E a mano, junto con la fusión con la rama principal del producto y luego su distribución a los equipos reales puede llegar a ser muy tediosa. Con estas herramientas se permite que una vez se realiza una fusión con la rama principal del código, automáticamente se ejecuten todas las pruebas del código pertinentes (que deberían estar bien desarrolladas) y si han sido satisfactorias, se distribuye el código automáticamente.

Para implementar la parte de la integración/distribución continua hemos hecho uso de la herramienta GitHub Actions. Esta herramienta nos permite automatizar, personalizar y ejecutar los flujos de trabajo que creamos convenientes. Todo esto directamente en nuestro repositorio de GitHub. Lo analizaremos más detenidamente en el apartado de implementación, pero por ahora nos basta con saber que cada trabajo que definamos en el archivo de la acción se ejecuta cuando le indiquemos (un *push*, un *pull request*, un *merge* a la rama que indiquemos) en una maquina remota que inicia GitHub en el momento de ejecución y ejecuta los comandos que queramos en el orden que queramos. Comandos que podrían ser por ejemplo hacer un *build* a la aplicación y ejecutar una serie de pruebas y luego subir al entorno que queramos.

Como hemos mencionado, las GitHub Actions nos permiten subir la aplicación donde quisiéramos, pero en este proyecto hemos optado por la plataforma Heroku. Esta nos permite alojar nuestra aplicación en la nube sin ningún coste, y tiene la característica de que puede estar ligada a un repositorio de GitHub, por lo que cuando las GitHub Actions ejecuten las pruebas que le hemos indicado satisfactoriamente, se haría la fusión de código, por lo que Heroku lo detectaría y lo subiría a la plataforma.

# Análisis y especificaciones de requisitos

Después de hacer una propuesta de una aplicación que mejores un aspecto concreto de lo que flaquean las demás en el mercado, se procede a realizar el análisis del problema. En este capítulo se van a recopilar las necesidades de la aplicación, así como definir las acciones que pueden hacer los diferentes tipos de usuarios que tendrá.

Se trata de una de las fases más importantes en el desarrollo de la aplicación, ya que si fallamos en la interpretación de lo que querrá el usuario final o en la relevancia de una funcionalidad específica después será mucho más costoso de arreglar cuando tengamos ya desarrollado el código. También es de gran ayuda dejar todos los requisitos y todos los posibles flecos claro en esta etapa, ya que, con todo esto claro, la complejidad del desarrollo de aligera bastante.

Primero definiremos los diferentes actores o tipos de usuarios que van a interactuar con nuestra aplicación. Explicaremos que roles tienen y luego pasaremos a definir los casos de uso. Que es más que definir qué puede hacer cada usuario con la aplicación.

Cuando tengamos los casos de uso pasaremos a definir los requisitos funcionales y no funcionales, en los que detallaremos que tiene que ofrecer el sistema al usuario para que pueda conseguir lo que hemos definidos para él en los casos de uso.

## Actores

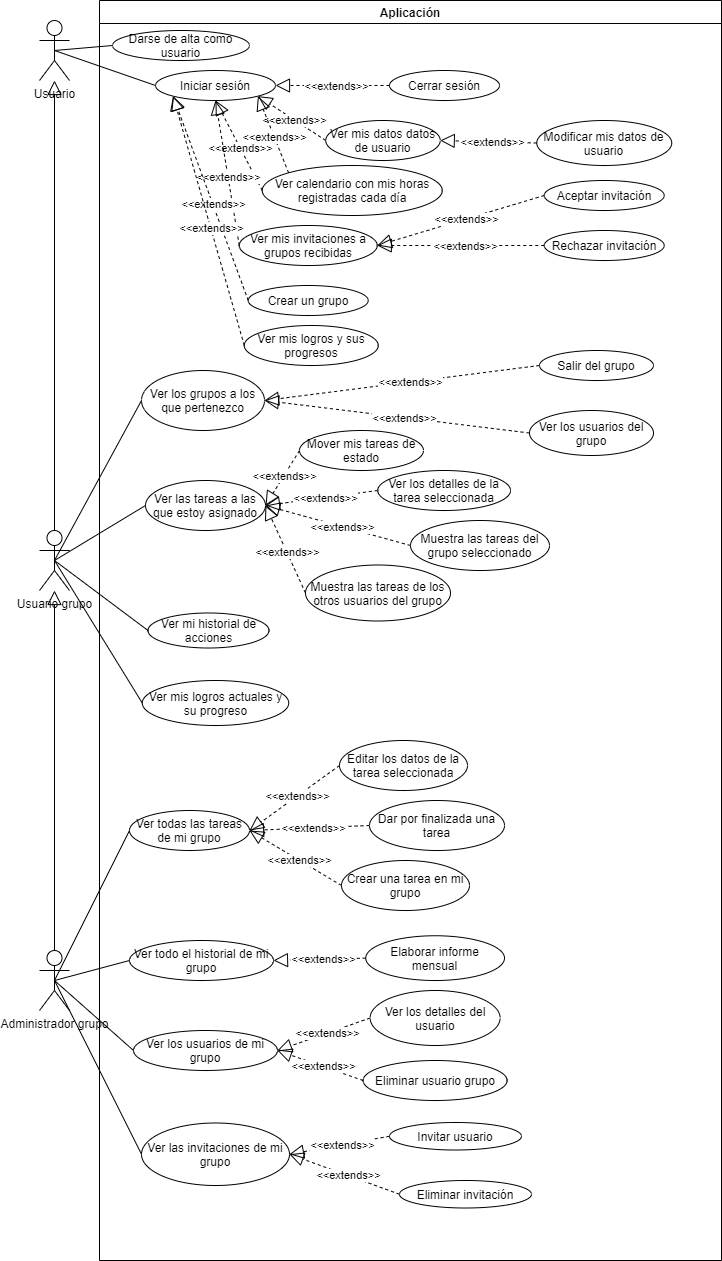
Los actores son los diferentes tipos de usuario que van a interactuar con nuestra aplicación, y según sean de un tipo u otro, podrán hacer unas determinadas acciones o no. Lo actores que hemos definido son los siguientes:

* Usuario: Se trata de un usuario que entra a la aplicación por primera vez. En este caso únicamente podría registrarse, crear un grupo o aceptar una invitación de un grupo. Entendemos como grupo un equipo de trabajo en un proyecto determinado.
* Usuario grupo: Es el usuario que ya ha aceptado una invitación a un grupo y por lo tanto pertenece a un grupo. La idea es que pueda gestionar sus propias tareas, mover de estado sus tareas y ver las tareas de los demás usuarios del grupo. Este usuario hereda todas las acciones que puede realizar un usuario normal.
* Administrador grupo: Es un mánager de proyecto. Se trata de un usuario registrado que ha creado un grupo. Puede crear, editar, eliminar y asignar tareas a usuarios. Puede ver el estado de todas las tareas y puede generar informes de horas en cualquier momento. Este usuario hereda todas las acciones que puede hacer un usuario de grupo.

## Casos de uso

Un caso de uso es una descripción de una acción que puede realizar un usuario sobre el sistema. El diagrama de casos de uso representa un sistema con un conjunto de interacciones que se desarrollarán entre el usuario y el sistema.

El diagrama que hemos obtenido es el siguiente, donde se reflejan los actores a la izquierda, fuera del rectángulo que representa el sistema. Las acciones están dentro de los óvalos y las líneas y flechas representan la interacción entre el usuario y el caso de uso. Las líneas continuas representan una asociación o comunicación y las discontinuas representan la opcionalidad, es decir, el usuario podría hacer o no la acción.



A continuación, después de ver una imagen general de cómo se relacionarían los usuarios con el sistema, vamos a definir cada caso de uso:

|  |  |
| --- | --- |
| Caso de uso | Darse de alta como usuario |
| Actores | Usuario |
| Propósito | Registrarse como usuario de la aplicación |
| Resumen | El usuario rellena un formulario con los siguientes datos: Nombre, apellido, correo y contraseña. Si todo es válido, al pulsar en aceptar se registra como usuario |
| Precondiciones | Pulsar en registrar en la pestaña inicial de la aplicación |
| Postcondiciones | El usuario queda registrado en la aplicación |

|  |  |
| --- | --- |
| Caso de uso | Iniciar sesión |
| Actores | Usuario |
| Propósito | Autenticarse para poder acceder a la aplicación |
| Resumen | El usuario puede rellenar su correo y su contraseña para acceder a la aplicación |
| Precondiciones | - |
| Postcondiciones | Si el usuario está registrado con esas credenciales, accede a la pantalla principal, si no, se le muestra un mensaje con que no existe |

|  |  |
| --- | --- |
| Caso de uso | Cerrar sesión |
| Actores | Usuario |
| Propósito | Cerrar la sesión que había iniciado |
| Resumen | Un usuario que ya se haya autenticado, puede cerrar sesión en todo momento |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | Se finaliza la sesión del usuario |

|  |  |
| --- | --- |
| Caso de uso | Ver mis datos datos de usuario |
| Actores | Usuario |
| Propósito | Ver los datos con los que el usuario se ha registrado |
| Resumen | Un usuario puede ver los datos con los que se registró (nombre, apellido, correo) |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Modificar mis datos de usuario |
| Actores | Usuario |
| Propósito | Editar los datos con los que el usuario se ha registrado |
| Resumen | Un usuario puede editar los datos con los que se registró |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | Los datos del usuario quedan modificados |

|  |  |
| --- | --- |
| Caso de uso | Ver calendario con mis horas registradas cada día |
| Actores | Usuario |
| Propósito | Ver las horas propias dedicadas a las tareas en una vista de calendario |
| Resumen | Un usuario puede ver las horas que ha ido dedicando a cada tarea en una vista de calendario. En un principio se muestra el mes actual, pero puede navegar por todos los años y meses |
| Precondiciones | El usuario debe haber hecho horas en alguna tarea |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Ver mis invitaciones a grupos recibidas |
| Actores | Usuario |
| Propósito | Ver las invitaciones recibidas de otros grupos |
| Resumen | Un usuario puede ver las invitaciones que otros grupos le han hecho para acceder a ellos. |
| Precondiciones | Un grupo debe invitar al usuario |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Aceptar invitación |
| Actores | Usuario |
| Propósito | Aceptar una invitación de un grupo |
| Resumen | Un usuario puede aceptar la invitación de un grupo para inscribirse en él |
| Precondiciones | Un grupo debe invitar al usuario |
| Postcondiciones | El usuario pertenece al grupo aceptado y la invitación se elimina |

|  |  |
| --- | --- |
| Caso de uso | Rechazar invitación |
| Actores | Usuario |
| Propósito | Rechazar una invitación de un grupo |
| Resumen | Un usuario puede rechazar la invitación de un grupo |
| Precondiciones | Un grupo debe invitar al usuario |
| Postcondiciones | La invitación se elimina |

|  |  |
| --- | --- |
| Caso de uso | Crear un grupo |
| Actores | Usuario |
| Propósito | Crear un grupo |
| Resumen | Un usuario puede crear un grupo y automáticamente se convierte en el administrador de ese grupo |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | El usuario pertenece a su grupo como administrador de grupo |

|  |  |
| --- | --- |
| Caso de uso | Ver mis logros y sus progresos |
| Actores | Usuario |
| Propósito | Ver los logros obtenidos y los progresos acumulados |
| Resumen | Un usuario puede ver los logros y los progresos de cada logro. Los progresos se aumentan al hacer acciones y los logros se obtienen al llegar al 100% de progreso |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Ver los grupos a los que pertenezco |
| Actores | Usuario grupo |
| Propósito | Ver los grupos a los que pertenece el usuario |
| Resumen | Un usuario puede ver los grupos a los que pertenece |
| Precondiciones | El usuario debe haber iniciado sesión |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Salir del grupo |
| Actores | Usuario grupo |
| Propósito | Salir del grupo seleccionado |
| Resumen | Un usuario de un grupo puede salir del grupo en todo momento |
| Precondiciones | El usuario debe estar en el grupo |
| Postcondiciones | El usuario ya no se encuentra en el grupo |

|  |  |
| --- | --- |
| Caso de uso | Ver los usuarios del grupo |
| Actores | Usuario grupo |
| Propósito | Ver los usuarios de un grupo |
| Resumen | Un usuario puede ver los usuarios que hay en el grupo al que pertenece |
| Precondiciones | El usuario debe estar en el grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Ver las tareas a las que estoy asignado |
| Actores | Usuario grupo |
| Propósito | Ver las tareas que el usuario tiene asignadas |
| Resumen | Un usuario puede ver las tareas a las que está asignado y el estado en el que están en un tablero Kanban |
| Precondiciones | El usuario debe estar en el grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Mover mis tareas de estado |
| Actores | Usuario grupo |
| Propósito | Mover la tarea de estado |
| Resumen | El usuario puede mover la tarea de estado |
| Precondiciones | El usuario debe estar en el grupo y tener tareas asignadas |
| Postcondiciones | La tarea cambia de estado y de columna en el tablero Kanban |

|  |  |
| --- | --- |
| Caso de uso | Ver los detalles de la tarea seleccionada |
| Actores | Usuario grupo |
| Propósito | Ver los detalles de la tarea (descripción, horas estimadas, fecha de entrega, horas hechas) |
| Resumen | El usuario puede pulsar en una tarea que tenga asignada y ver los datos de esa tarea |
| Precondiciones | El usuario debe estar en el grupo y tener tareas asignadas |
| Postcondiciones | Se muestran los detalles de la tarea |

|  |  |
| --- | --- |
| Caso de uso | Muestra las tareas del grupo seleccionado |
| Actores | Usuario grupo |
| Propósito | Ver las tareas totales que tiene el grupo |
| Resumen | El usuario puede ver todas las tareas que hay en el grupo |
| Precondiciones | El usuario debe estar en el grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Ver mi historial de acciones |
| Actores | Usuario grupo |
| Propósito | Ver el historial de horas en tareas |
| Resumen | El usuario puede ver el historial de sus horas dedicadas a cada tarea y filtrarlo por intervalo de tiempo, grupo y tarea |
| Precondiciones | El usuario debe estar en el grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Editar los datos de la tarea seleccionada |
| Actores | Administrador grupo |
| Propósito | Editar los datos de una tarea seleccionada |
| Resumen | El administrador del grupo puede seleccionar una tarea y editar sus datos |
| Precondiciones | Ser administrador de grupo y seleccionar una tarea |
| Postcondiciones | Se registran los cambios en la tarea |

|  |  |
| --- | --- |
| Caso de uso | Dar por finalizada una tarea |
| Actores | Administrador grupo |
| Propósito | Finalizar el ciclo de una tarea |
| Resumen | El administrador puede dar por finalizada la tarea que desee |
| Precondiciones | Ser administrador de grupo y seleccionar una tarea |
| Postcondiciones | La tarea se finaliza |

|  |  |
| --- | --- |
| Caso de uso | Crear una tarea en mi grupo |
| Actores | Administrador grupo |
| Propósito | Crear una tarea en el grupo |
| Resumen | El administrador puede rellenar el formulario de creación de la tarea y crear la tarea |
| Precondiciones | Ser administrador de grupo |
| Postcondiciones | Se crea la tarea |

|  |  |
| --- | --- |
| Caso de uso | Ver todo el historial de mi grupo |
| Actores | Administrador grupo |
| Propósito | Ver el historial de todas las horas en tareas |
| Resumen | El administrador de grupo puede ver el historial de todas las horas dedicadas a cada tarea y filtrarlo por intervalo de tiempo, grupo y tarea |
| Precondiciones | Ser administrador de grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Elaborar informe mensual |
| Actores | Administrador grupo |
| Propósito | Elaborar informe en Excel de las horas mensuales |
| Resumen | El administrador de grupo puede generar un informe de las horas del grupo, o de una tarea determinada en un intervalo de tiempo determinado en formato Excel |
| Precondiciones | Ser administrador de grupo |
| Postcondiciones | Genera un informe que se descarga |

|  |  |
| --- | --- |
| Caso de uso | Ver los usuarios de mi grupo |
| Actores | Administrador grupo |
| Propósito | Ver los usuarios que hay en el grupo |
| Resumen | El administrador del grupo puede ver los usuarios que hay en el grupo |
| Precondiciones | Ser administrador del grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Ver los detalles del usuario |
| Actores | Administrador grupo |
| Propósito | Ver los detalles del usuario seleccionado del grupo |
| Resumen | El administrador del grupo puede ver los detalles de los usuarios del grupo |
| Precondiciones | Ser administrador de grupo y seleccionar un usuario |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Eliminar usuario grupo |
| Actores | Administrador grupo |
| Propósito | Eliminar a un usuario de un grupo |
| Resumen | El administrador del grupo puede eliminar a cualquier usuario del grupo |
| Precondiciones | Ser administrador de grupo y seleccionar un usuario |
| Postcondiciones | El usuario se elimina del grupo |

|  |  |
| --- | --- |
| Caso de uso | Ver las invitaciones de mi grupo |
| Actores | Administrador grupo |
| Propósito | Ver las invitaciones que hay pendientes del grupo |
| Resumen | El administrador puede ver las invitaciones que se han lanzado y están pendientes de respuesta por parte del usuario invitado |
| Precondiciones | Ser administrador de grupo |
| Postcondiciones | - |

|  |  |
| --- | --- |
| Caso de uso | Invitar usuario |
| Actores | Administrador grupo |
| Propósito | Invitar a un usuario al grupo |
| Resumen | El administrador puede invitar a un usuario al grupo introduciendo un correo |
| Precondiciones | Ser administrador de grupo |
| Postcondiciones | Se crea una invitación al usuario con ese correo |

|  |  |
| --- | --- |
| Caso de uso | Eliminar invitación |
| Actores | Administrador grupo |
| Propósito | Eliminar una invitación |
| Resumen | El administrador puede eliminar una invitación hecha |
| Precondiciones | Ser administrador de grupo y tener invitaciones hechas |
| Postcondiciones | La invitación se elimina |

## Requisitos funcionales

En el apartado anterior hemos definido como tiene que comportarse el sistema en frente a un usuario real. Ahora en este apartado vamos a definir qué funciones debe cubrir el sistema para cubrir los casos de uso que hemos comentado anteriormente. Una función puede ser descrita como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser cálculos, detalles técnicos, manipulación de datos, etc. A continuación, se van a detallar los requisitos funcionales:

* Registro: Todo usuario de la aplicación puede y debe registrarse antes de iniciar sesión. El formulario de registro obtiene el nombre, apellidos, correo y contraseña del usuario y lo registra como usuario en la base de datos.
  + Entrada: Datos del usuario.
  + Salida: Usuario registrado en la base de datos.
* Iniciar sesión: Todo usuario registrado puede acceder a la aplicación usando sus credenciales.
  + Entrada: Correo y contraseña del usuario.
  + Salida: Crea sesión en el navegador si credenciales correctas, muestra mensaje si credenciales incorrectas.
* Cerrar sesión: Un usuario que haya iniciado sesión puede cerrarla, finalizando la sesión.
  + Entrada: Usuario pulsa botón de cerrar sesión.
  + Salida: Se elimina la sesión del usuario del navegador y la aplicación vuelve a la pantalla de inicio de sesión.
* Leer invitaciones: Todo usuario puede leer las invitaciones a grupos que le hayan mandado otros usuarios.
  + Entrada: El usuario accede a la interfaz de invitaciones.
  + Salida: Se muestran las invitaciones hechas al usuario actual.
* Aceptar invitaciones: Al aceptar una invitación, el usuario se une al grupo que le ha invitado.
  + Entrada: El usuario acepta una invitación.
  + Salida: El usuario pasa a ser miembro del grupo aceptado.
* Rechazar invitaciones: Al rechazar una invitación, se elimina la invitación sin ninguna repercusión.
  + Entrada: El usuario rechaza una invitación.
  + Salida: La invitación se elimina.
* Invitar a un usuario: Un usuario administrador de grupo puede invitar a otros usuarios a su grupo usando el correo del invitado.
  + Entrada: El usuario introduce el correo del usuario invitado y pulsa en el botón de crear.
  + Salida: Se crea la invitación y se le muestra al usuario invitado.
* CRUD grupos: Un usuario puede crear un grupo, leer los grupos en los que está inscrito, actualizar los datos del grupo en el que sea administrador y eliminar un grupo en el que sea administrador.
* CRUD tareas: Un usuario administrador de grupo puede crear, leer, actualizar los datos y eliminar las tareas de su grupo. Un usuario que pertenece a un grupo puede únicamente leer las tareas, actualizar las horas imputadas y actualizar el estado de una tarea.
* CRUD historial: Un usuario administrador puede leer el historial de las acciones que han realizado los usuarios de su grupo en las tareas, así como filtrar según la tarea, el usuario y por fecha. Un usuario de un grupo únicamente puede hacerlo en sus propias acciones. Todo usuario puede actualizar las horas imputadas en una tarea en el historial y eliminar una entrada.
* Imprimir historial: Un usuario administrador puede imprimir en una hoja Excel el historial de acciones de su grupo.
  + Entrada: Usuario filtra el historial y pulsa en el botón de imprimir.
  + Salida: Se descarga un archivo Excel con el historial filtrado.
* Mover tarea a estado “desarrollo”: Cuando un usuario mueva una tarea a estado “desarrollo”, automáticamente se inicia el cronómetro sumándole la cantidad de tiempo que el usuario va dedicándole a la tarea.
  + Entrada: Usuario mueve la tarea al estado “desarrollo”.
  + Salida: Se actualiza el estado de la tarea y se inicia el cronómetro del tiempo que va dedicando el usuario a la tarea.
* Quitar la tarea del estado “desarrollo”: Cuando un usuario mueva una tarea fuera del estado “desarrollo”, se pausa el cronómetro y se guardan las horas dedicadas a esa tarea en el historial de acciones del usuario.
  + Entrada: Usuario mueve la tarea del estado “desarrollo” a otra.
  + Salida: Se actualiza el estado de la tarea y se finaliza el cronómetro, registrándose el tiempo dedicado en el historial de acciones del usuario.
* Ver logros: Un usuario puede ver los logros que tiene completados y el progreso de los que aún no tiene completados.
  + Entrada: Usuario pulsa en el botón de logros.
  + Salida: Se muestran los logros completados y el progreso de los no completados.
* Integración con google tasks: Todas las tareas estarán integradas con la cuenta de google task del usuario que tenga asignada la tarea.

## Requisitos no funcionales

Los requisitos no funcionales se refieren todos los requisitos que no describen las funciones que hemos descrito en los requisitos funcionales, entendiendo como función un conjunto de entradas, comportamientos y salidas. Se centran más en las características que complementan el funcionamiento que en el funcionamiento específico de la aplicación. A continuación, se detallan los requisitos funcionales:

* Interfaz de usuario: La aplicación debe presentar una interfaz intuitiva, dinámica y sencilla. Debe ofrecer un tablero Kanban de las tareas a los usuarios. Y la aplicación debe estar implementada como una SPA para que sea lo más dinámica posible.
* Adaptabilidad: La interfaz de la aplicación debe adaptarse a los posibles formatos de pantalla de los diferentes dispositivos. Como móviles, tabletas y ordenadores.
* *Feedback*: La aplicación debe retornar una señal, en forma de mensaje o en forma de actualización en la interfaz, cada vez que el usuario realice una acción.
* Disponibilidad: El sistema debe estar disponible las 24 horas de todos los días del año.
* Compatibilidad de navegadores: La aplicación debe funcionar exactamente en todos los navegadores.
* Tiempo de peticiones HTTP: El tiempo de cada petición que se haga a la API REST no debe superar 1 segundo.

# Diseño

Una de las fases del ciclo por el que tiene que pasar el desarrollo de software es la etapa de diseño. En esta etapa se tiene que llegar a una planificación o diseño de las soluciones que tienen que resolver los requisitos que hemos definido en el apartado anterior. Cabe destacar el concepto de diseño, ya que no se trata de soluciones ya codificadas sino de modelar una solución sin llegar a implantarla, esta última fase la veremos en el apartado de implementación.

Normalmente, la fase de diseño suele contar con estas etapas:

* Modelado de datos: El modelado de datos es el proceso de documentar la forma en que los datos necesitan fluir en un sistema de software como un diagrama de fácil comprensión.
* Diseño de la arquitectura: La arquitectura de software es el diseño de más alto nivel de la estructura en un proyecto software. Consiste en un conjunto de elementos y abstracciones que proporcionan un marco claro para interactuar con el código fuente del software.
* Diseño de la interfaz: El diseño de la interfaz de usuario tiene como objetivo definirla forma, función, utilidad y ergonomía para solucionar los requisitos que hemos definido anteriormente ofreciéndole la máxima usabilidad al usuario.

## Modelado de la base de datos

El sistema de información, como ya hemos mencionado en el proyecto, representa un gestor de tareas. También hay que tener en cuenta que los usuarios deben estar en un grupo, aunque el grupo sea solamente de una persona, para poder gestionar las tareas. Por lo que los objetos principales serán los usuarios, los grupos y las tareas. Por otro lado, tenemos que gestionar las invitaciones que se crean así que también tenemos que definir este objeto. Las tareas también pueden tener el estado como un objeto independiente, ya que en un futuro es posible que los usuarios creen diferentes estados y estos, en el momento de creación de la tarea puedan asignarse, por lo que también tendremos el objeto estado de la tarea. También habría que definir el historial de acciones que un usuario lleva a cabo en una tarea determinada y los objetos de los logros que están disponibles para que adquieran los usuarios. Por último, habría que crear un registro con el identificador del dispositivo o dispositivos que usa el usuario, esto será de gran utilidad a la hora de enviar notificaciones a su dispositivo.

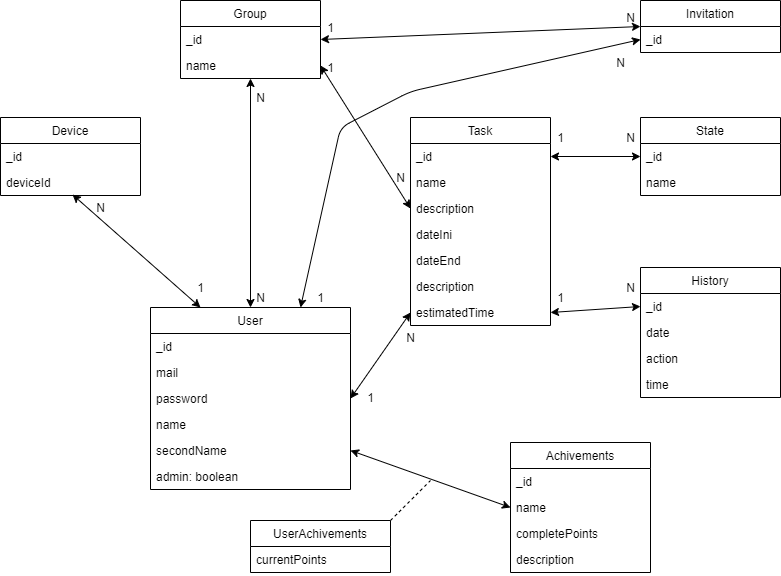
Los objetos que hemos definido anteriormente deberían tener los siguientes atributos para satisfacer las necesidades de los usuarios:

* Grupo
  + Identificador único
  + Nombre
  + Usuarios
  + Tareas
* Usuario
  + Identificador único
  + Correo
  + Nombre
  + Apellidos
  + Contraseña
  + Administrador
  + Grupos
  + Tareas
  + Invitaciones
* Tarea
  + Identificador único
  + Nombre
  + Descripción
  + Grupo
  + Usuario
  + Fecha de inicio
  + Fecha fin
  + Estado
  + Tiempo estimado
  + Fecha fin estimada
  + Tiempo imputado
  + Historiales
* Invitación
  + Identificador único
  + Grupo invitado
  + Usuario invitado
* Historial acciones
  + Identificador único
  + Tarea
  + Usuario
  + Grupo
  + Fecha
  + Acción
  + Tiempo de la acción
* Dispositivo
  + Identificador único
  + Usuario
* Logros
  + Identificador único
  + Nombre
  + Descripción
  + Puntos para completar

Los objetos se relacionan con los otros de la siguiente forma:

* Grupo
  + Un grupo contiene diferentes usuarios y al menos debe contener uno, que será el usuario administrador de grupo.
  + Contendrá también las tareas que tienen los usuarios en el grupo
  + Un grupo tiene un número ilimitado de invitaciones que ha hecho a otros usuarios.
* Usuario
  + Un usuario puede tener diferentes tareas asignadas.
  + Un usuario puede tener uno o varios grupos a los que está inscrito.
  + Un usuario puede ver las invitaciones que le han hecho.
  + Además, tendrá uno o varios logros que ha completado o que aún están en progreso.
  + El usuario podrá tener relacionados varios dispositivos.
* Tarea
  + Cada tarea, necesariamente, debe estar relacionada con un usuario y con un grupo.
* Invitación
  + Una invitación debe involucrar necesariamente a un grupo y a un usuario.
* Historial acciones
  + Una entrada en el historial debe estar relacionada con una tarea y un usuario.
* Dispositivos
  + Un dispositivo debe estar relacionado con un usuario.
* Logros
  + Un logro debe tener varios usuarios, y cada relación con uno deberá incluir si el logro está completo o el progreso actual.

El diagrama del modelado de los objetos que usaremos en el proyecto quedaría de la siguiente forma:



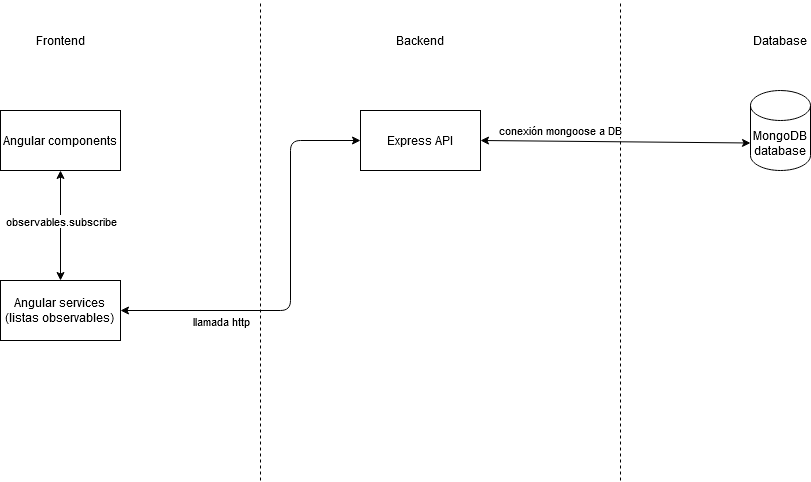
## Arquitectura

Definimos la arquitectura de software como la estructura que debe tener un proyecto software en tanto a las piezas individuales que lo forman y el modo en el que estas piezas se deben juntar y como trabajan entre ellas.

La arquitectura del proyecto la podemos dividir en tres partes claramente diferenciadas, a las que llamaremos frontend, backend y base de datos. Pasemos a definirlas:

* Frontend: Se trata de la parte que se encarga de mostrar las vistas a los usuarios, así de cómo mostrar los datos que recuperamos desde la base de datos y como introduce datos el usuario para que sean almacenados. Esta parte estará desarrollada en Angular. Cabe mencionar que Angular no maneja un patrón clásico de MVC (modelo, vista, controlador), sino que el modelo tiene mucha relación con la vista. Esto es así por el concepto base de Angular de two-way data binding, ya que la forma de sincronizar los datos entre la vista y el modelo-vista es totalmente dependiente, es decir, en la vista podemos modificar el modelo y en el modelo podemos modificar la vista, haremos más hincapié en esto en el apartado de implementación. Ahora mismo habría que dejar claro que Angular es el encargado de proporcionar las vistas a los usuarios.
* Backend: Es la parte que se encarga de todos los procesos necesarios para que la aplicación funcione de forma correcta. En nuestro caso se trata de un servidor en el que esperaremos peticiones que nos hará el frontend. Es decir, esperaremos por ejemplo una petición de obtener las tareas de un usuario y nuestro servidor se conectará con la base de datos y efectuará la consulta necesaria, modelará los datos obtenidos al formato correcto y los enviará al frontend, que se encargará de mostrarlos. También se pueden programar acciones cada cierto tiempo, como repartir tareas programadas o enviar notificaciones a los dispositivos de los usuarios. En este proyecto hemos optado por crear un servidor Express.js por simplicidad a la hora de gestionar peticiones HTTP y por poder instalar un modelador de objetos frente a la base de datos llamado Mongoose, que nos permitirá obtener objetos JSON de los esquemas que le definamos. Es decir, al crear un esquema, podemos efectuar peticiones a la base de datos usando ese esquema y la petición nos devolverá ya un objeto JSON con el mismo formato que hemos definido en el esquema. Esto nos ayudará a mapear los datos que tenemos en base de datos con los objetos que definamos en el frontend.
* Base de datos: Se trata de la parte donde se almacenan todos los datos que va a gestionar la aplicación. Hemos optado por la base de datos NoSQL MongoDB por su gran adaptación en todo el framework de Angular + Express.js.

El diagrama de la arquitectura software quedaría de la siguiente forma:

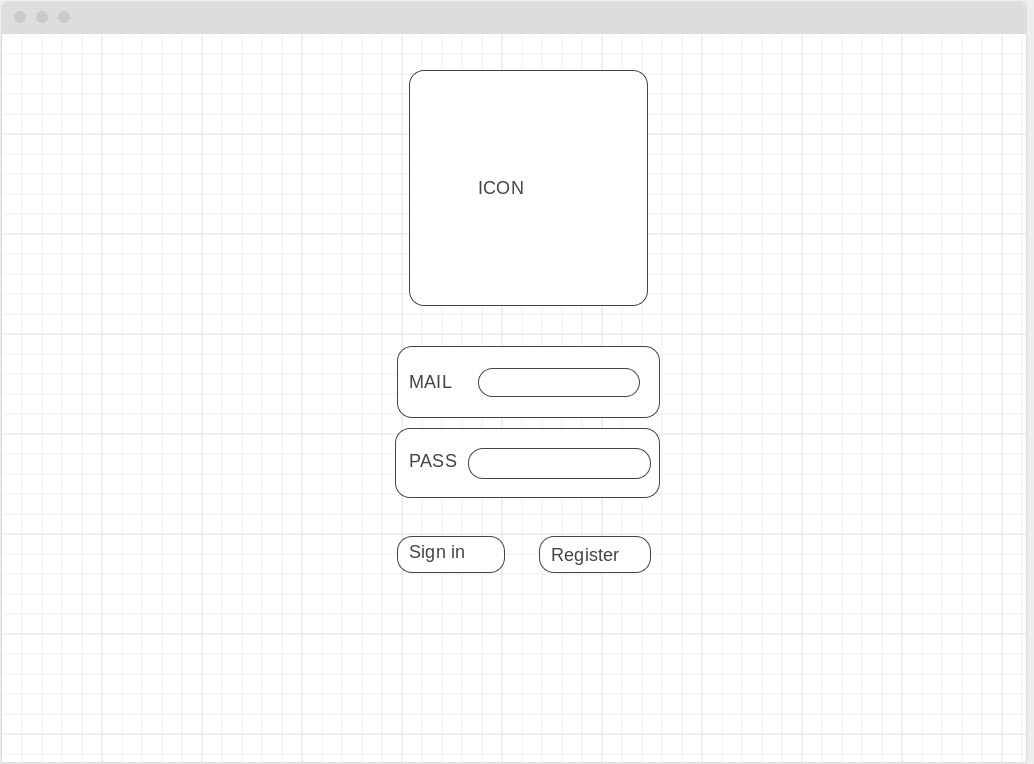


En el diagrama podemos ver las tres partes principales que hemos definido anteriormente y como se comunican entre ellas. Mencionar que en la parte de frontend, el componente de Angular es el modelo-vista, este modelo contiene se suscribe a un tipo de dato llamado Observable, estos observables se encuentran en los archivos de servicios de Angular, que son las partes de Angular encargadas de comunicarse con el backend por peticiones http y pasan los datos obtenidos a las listas observables, que al final son las listas donde lee y muestra los datos el componente. Lo explicaremos más profundamente en el apartado de implementación.

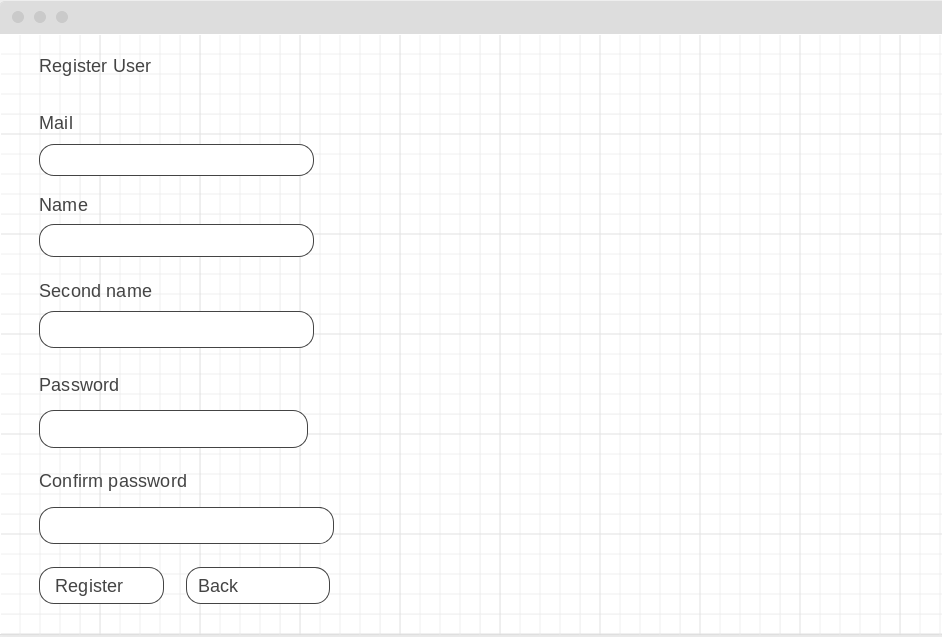
## Interfaz (prototipos)

Como hemos mencionado anteriormente. La interfaz es la encargada de permitir a los usuarios interactuar con la aplicación y permitirles cumplir los requisitos y casos de uso que hemos definido. Para tener una idea clara de cómo vamos a hacerlo, hemos creado una serie de prototipos y definido como el usuario va a interactuar entre las diferentes pantallas.

* Pantalla inicial: Esta es la primera pantalla que ve el usuario al entrar a la aplicación, le permite acceder usando sus credenciales o acceder a la pantalla de registro si aún no está registrado, en grande aparecerá el icono de la aplicación:



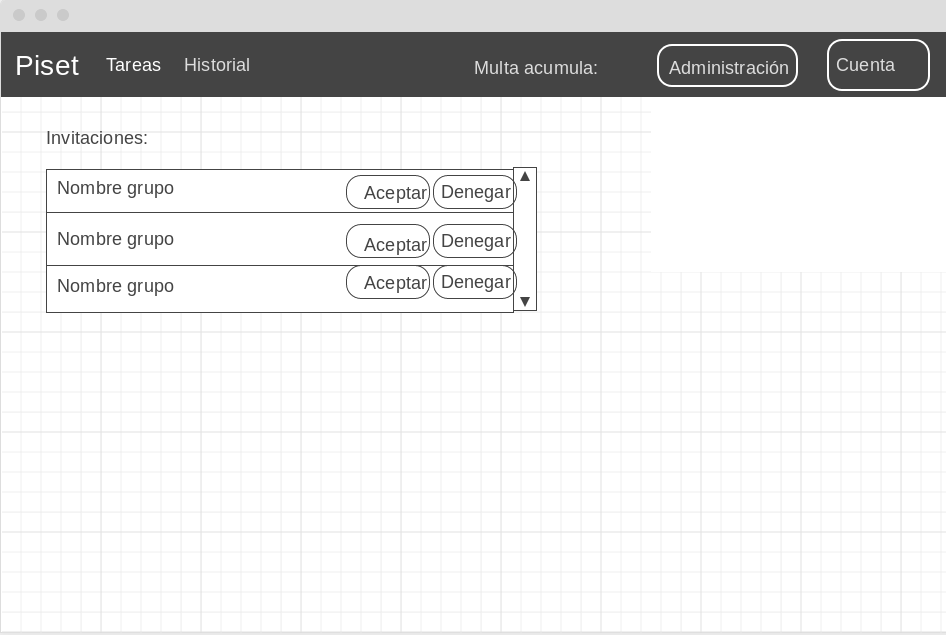
* Pantalla de registro: Esta pantalla se accede desde la pantalla de inicio, en esta interfaz el usuario puede introducir sus datos personales y registrarse como usuario de la aplicación, donde después podrá acceder desde la pantalla de inicio.



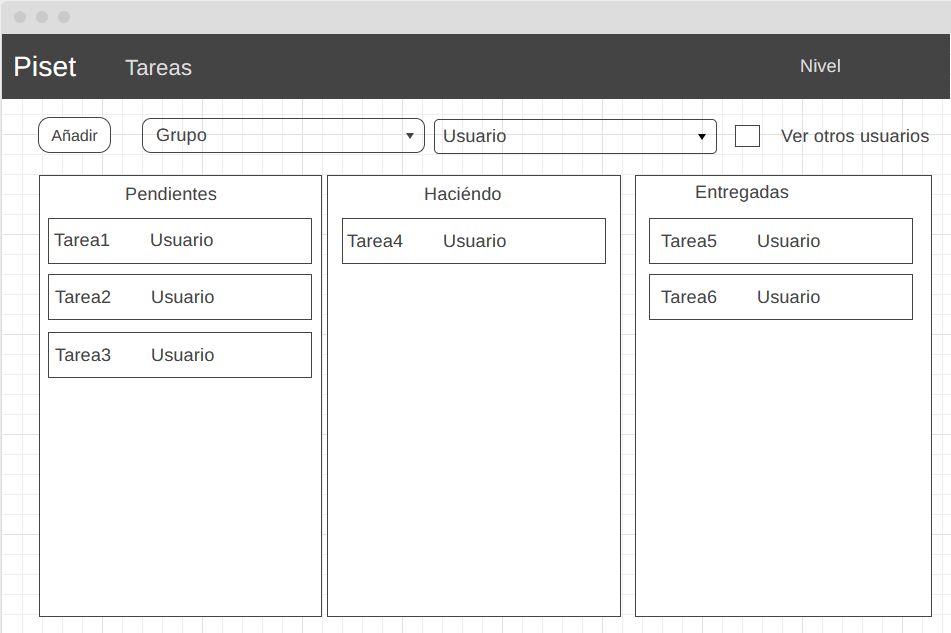
* Pantalla de ajustes de usuario: En esta vista el usuario ya registrado puede actualizar sus datos de usuario. Puede actualizar su nombre, apellidos y contraseña.



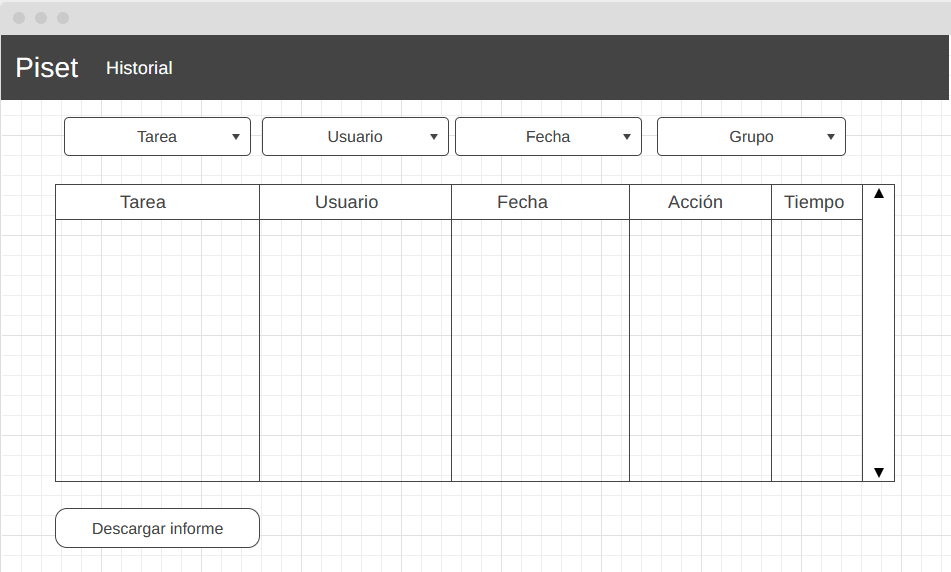
* Pantalla de invitaciones: En la siguiente pantalla un usuario puede ver las invitaciones que le han llegado, y puede aceptar y denegar las invitaciones que quiera.



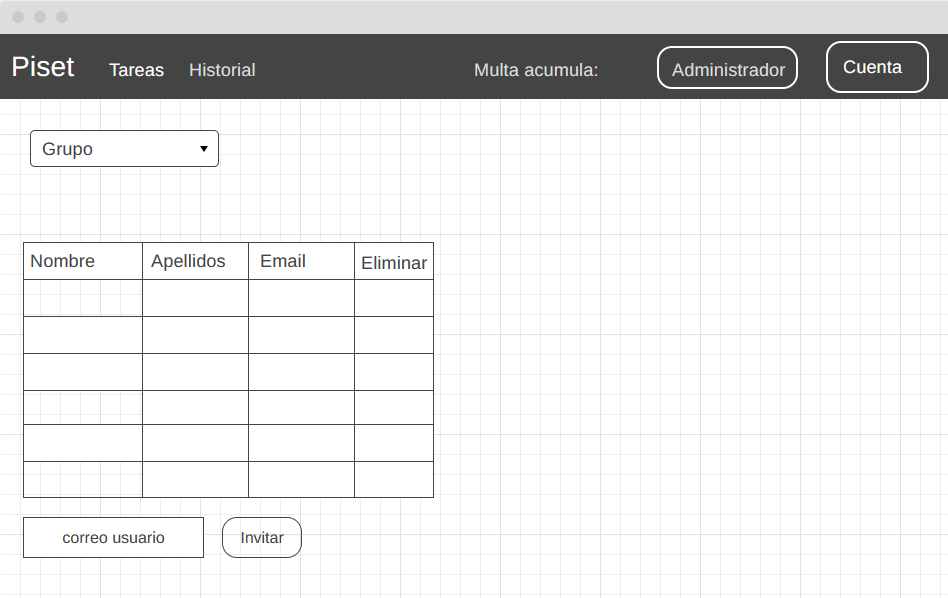
* Pantalla principal de las tareas: En esta vista los usuarios pueden ver en un tablero Kanban el estado de todas las tareas en las que están asignados. Al pulsar en una tarea pueden ver sus detalles, y en el caso de los administradores del grupo pueden editar los detalles de las tareas. En la parte de arriba del panel pueden aplicar los filtros que quieran, para mostrar las tareas según el grupo y los usuarios. Los usuarios pueden mover las tareas de estados.



* Pantalla del historial de acciones: En esta pantalla los usuarios pueden ver todo su historial de acciones sobre las tareas, pudiendo filtrar por grupo, tarea, intervalo de fechas, y en el caso de los administradores del grupo, por usuarios. También pueden pulsar en el botón de descargar informe para descargar un informe de las tareas filtradas en una hoja Excel.



* Pantalla de gestión de usuarios: En esta vista los gestores del grupo pueden eliminar o añadir los usuarios que quieran al grupo que tienen seleccionado, así como ver una lista de los usuarios de cada grupo.



# Implementación

En este apartado vamos a explicar todo el proceso de desarrollo del proyecto. Una vez ya tenemos todas las partes diseñadas y tenemos un camino que seguir, vamos a proceder a implementar todas esas soluciones que hemos diseñado en el apartado anterior.

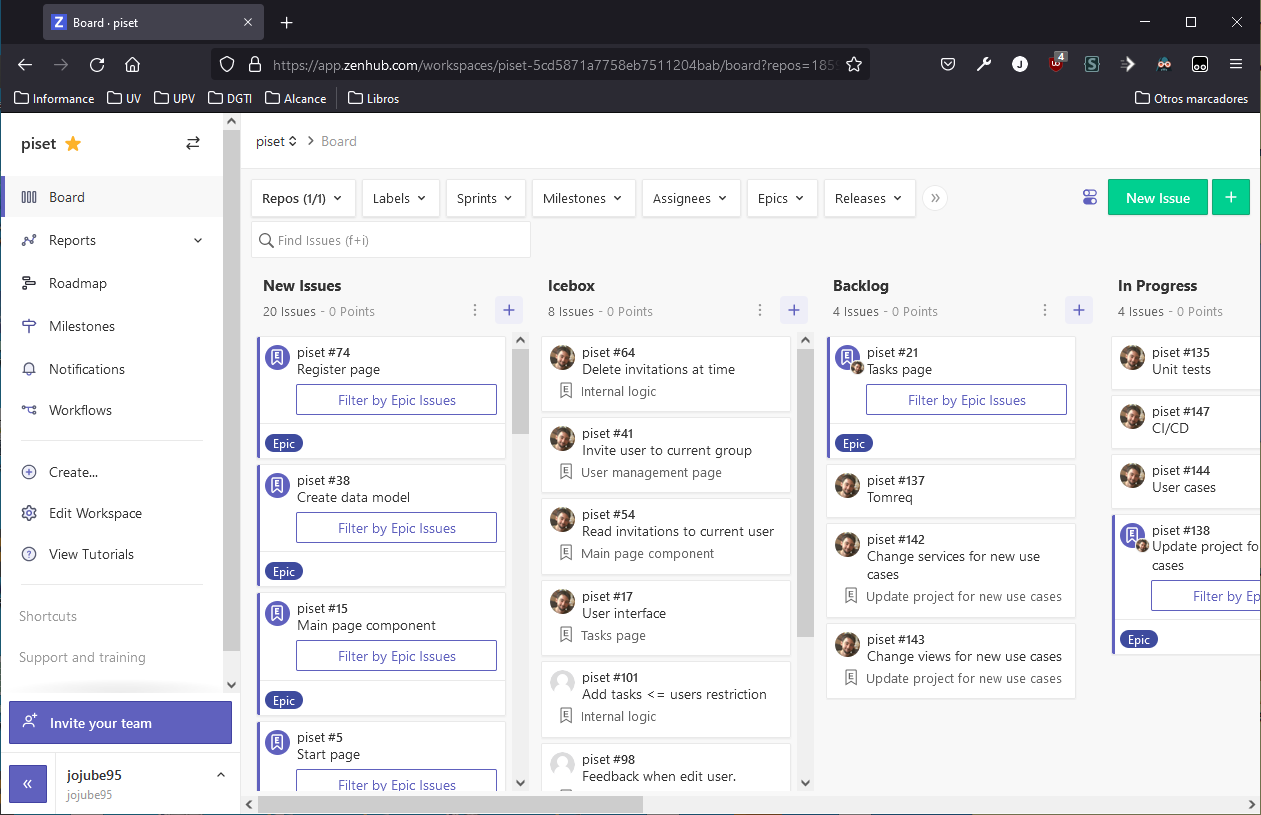
Para explicarlo de la mejor forma posible primero comentaremos todas las herramientas de desarrollo que hemos utilizado, el porqué de su uso y el propósito. Seguidamente explicaremos el modelo de gestión de versiones que vamos a usar, los repositorios sobre los que vamos a trabajar y las ramas que tendrá cada uno de ellos. Después de esto vamos a definir los entornos donde se ejecutará cada parte del proyecto y como se relacionarán con los diferentes repositorios y ramas que hemos definido anteriormente. Una vez tengamos todo esto definido pasaremos a explicar cómo hemos implementado la integración continúa, es decir, como se ha automatizado la relación entre las ramas de los repositorios y los diferentes entornos para que cuando se produzca un cambio en una rama, se ejecuten las pruebas pertinentes y se suba al entorno que toque.

Con toda la metodología ya definida con la que vamos a trabajar, ya podemos pasar a explicar el desarrollo del proyecto como tal, para hacerlo de la forma más simple posible, hemos optado por ir explicando el desarrollo empezando por la implementación de la base de datos, pasando por el backend y finalizando por el frontend.

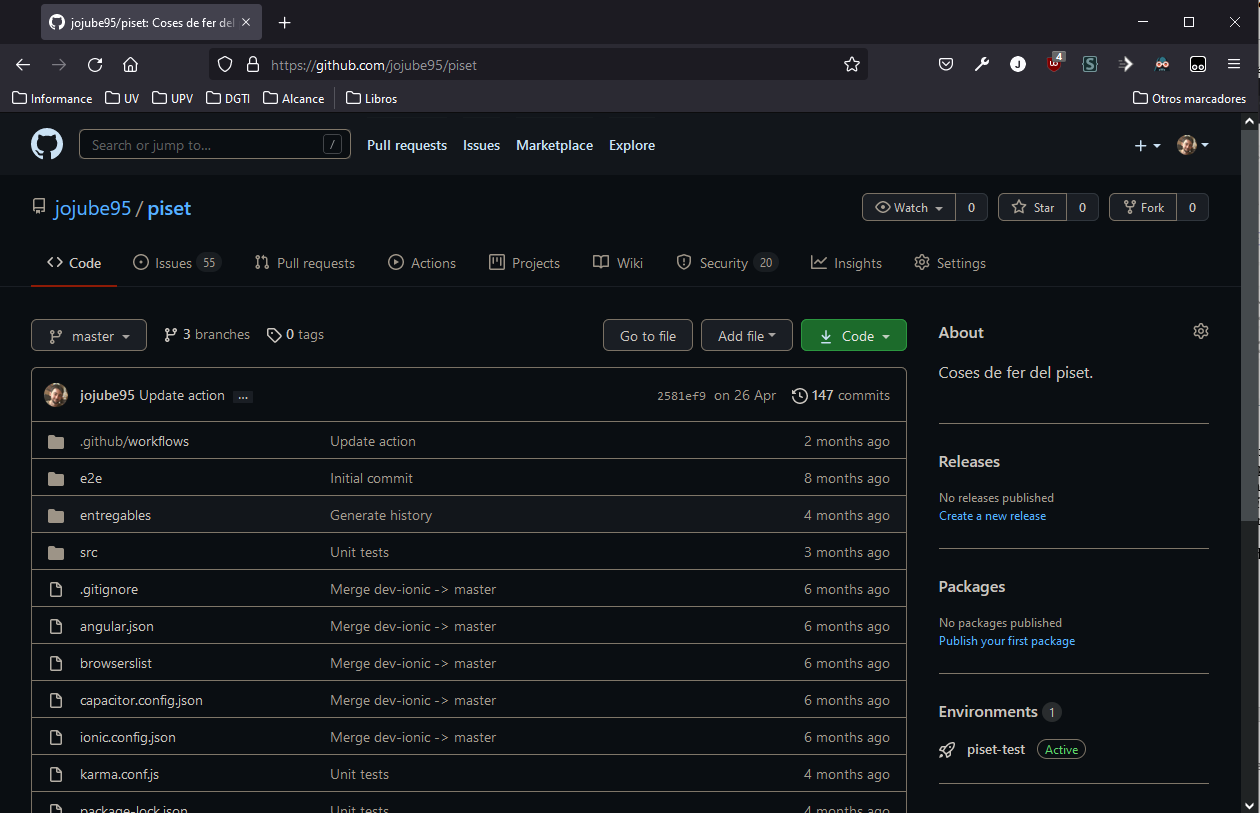
## Contexto tecnológico

Vamos a mencionar y explicar por qué hemos elegido las siguientes aplicaciones para que nos faciliten el trabajo a la hora de desarrollar el proyecto. Dividiremos las aplicaciones según la parte de la arquitectura del proyecto en la que las hemos usado.

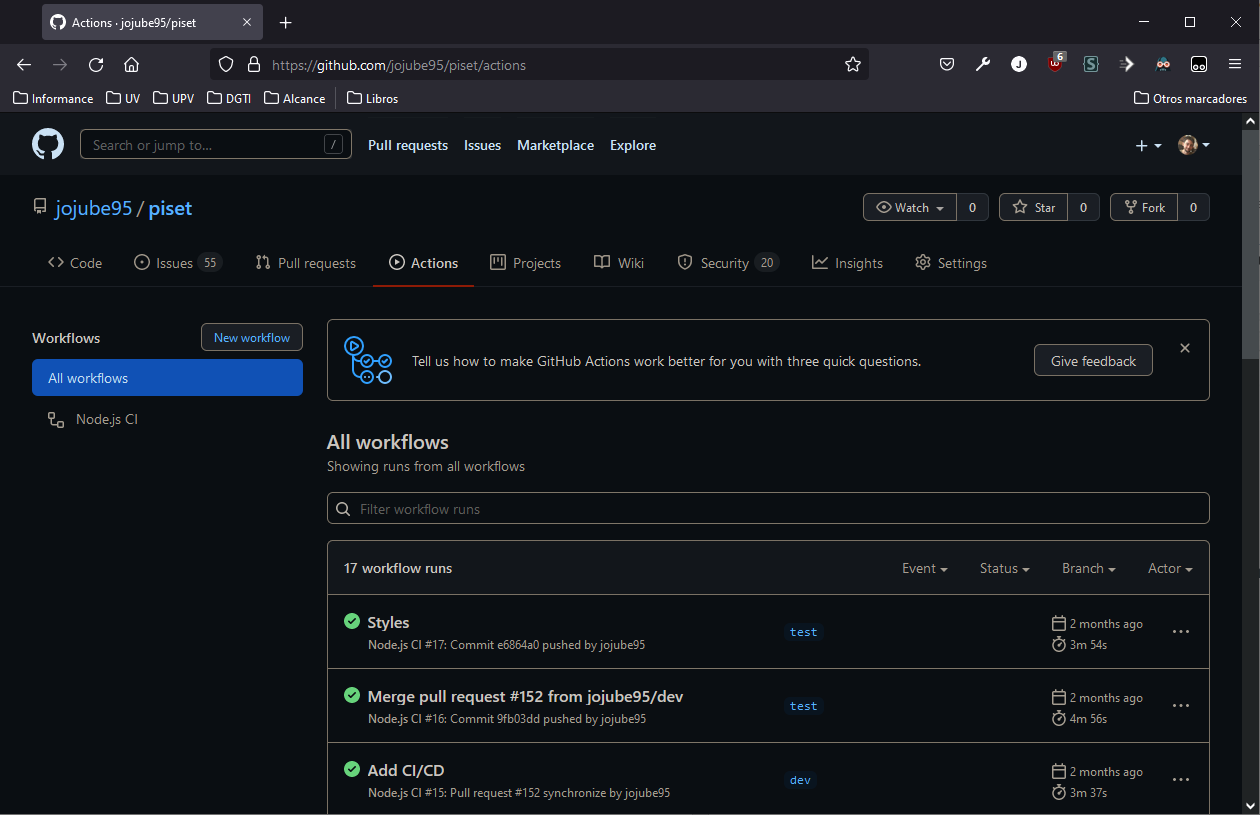
* Gestión de carga de trabajo:
  + ZenHub: Se trata de una aplicación que nos permite crearnos tareas sobre repositorios Git. Esto nos permite tener bien gestionadas cada tarea que vamos a hacer para el desarrollo del proyecto con cada *commit* del repositorio donde se aloja el código del proyecto. Permite crear entradas épicas que las forman un conjunto de tareas y poder moverlas de estado:



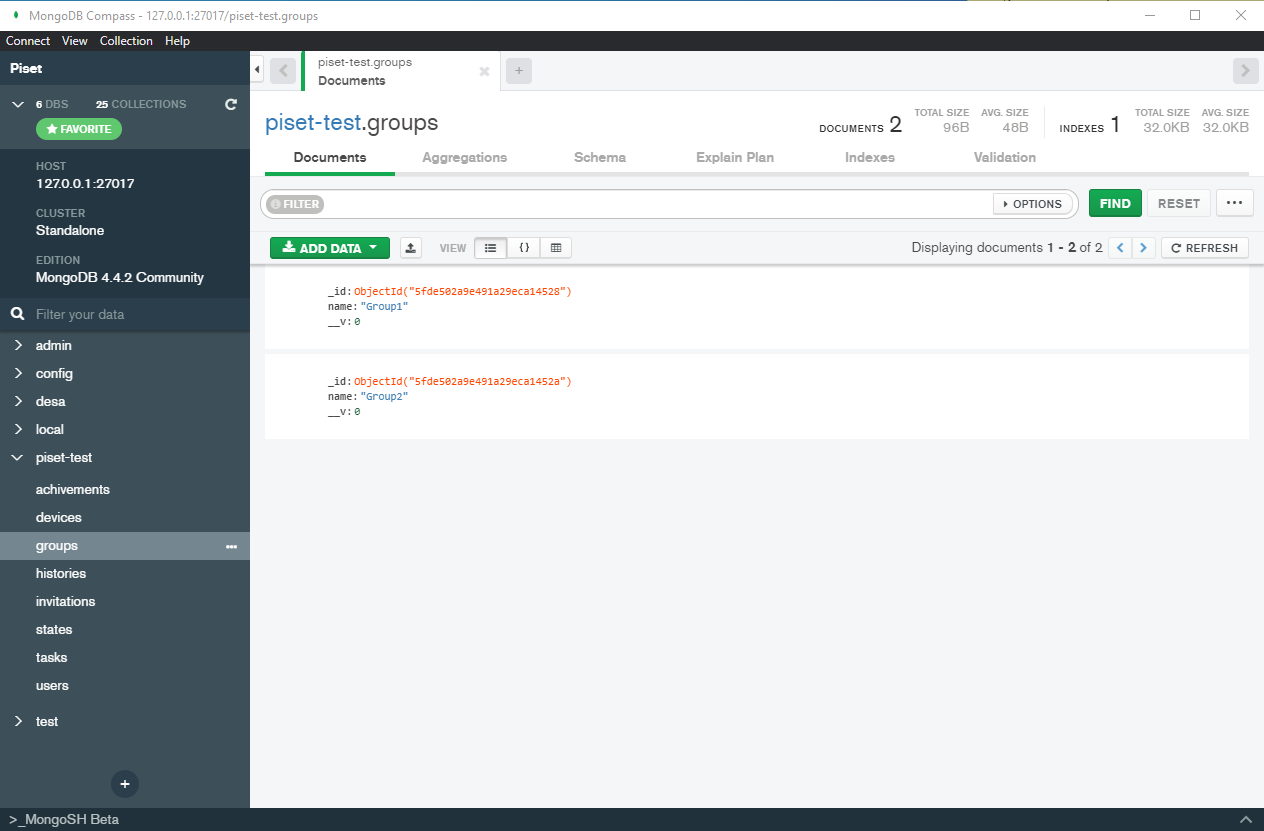
* Gestión de versiones:
  + Git: Hemos instalado y usado Git para la gestión de versiones del código que vamos desarrollando. Se trata de una aplicación por línea de comandos, aunque hay una versión con interfaz de usuario. Esta herramienta nos permite realizar *commits* de cambios en el código y subirlos a GitHub usando el comando *push*.
  + GitHub: Se trata de una plataforma donde alojar y poder gestionar un repositorio Git desde una página web:



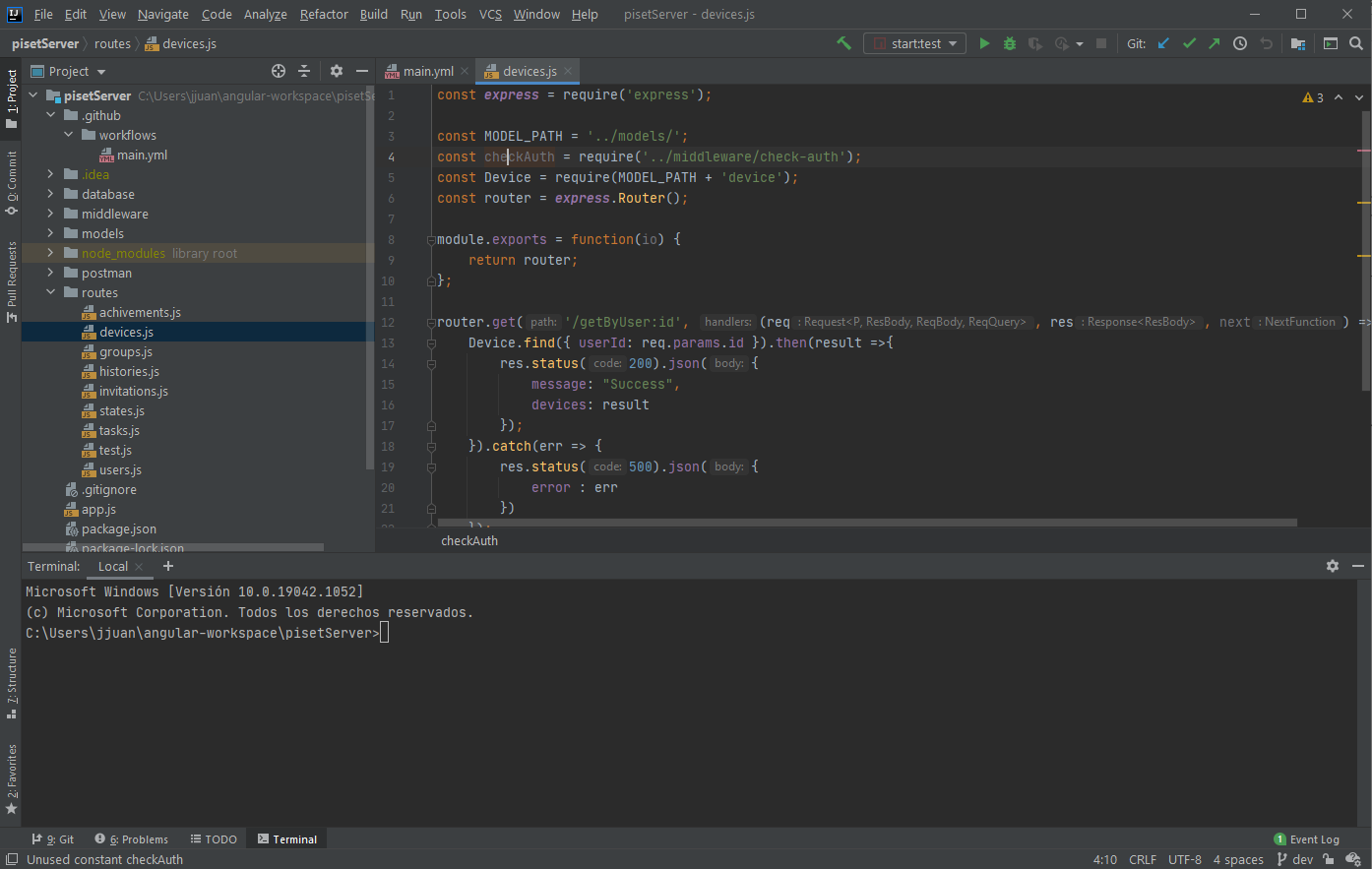
* Integración continua:
  + GitHub Actions: Para implementar la integración continua hemos optado por usar la herramienta GitHub Actions que nos proporciona la propia plataforma de GitHub. La hemos elegido por su integración directa con el repositorio de GitHub. En esta herramienta puedes definir un archivo con una serie de acciones que se ejecutarán en una máquina virtual que crea GitHub al vuelo cuando se detecte una determinada acción en el repositorio, como un *push*, *merge* o *pull* *request*, así como ver el resultado de cada ejecución que se haya lanzado de esas acciones:



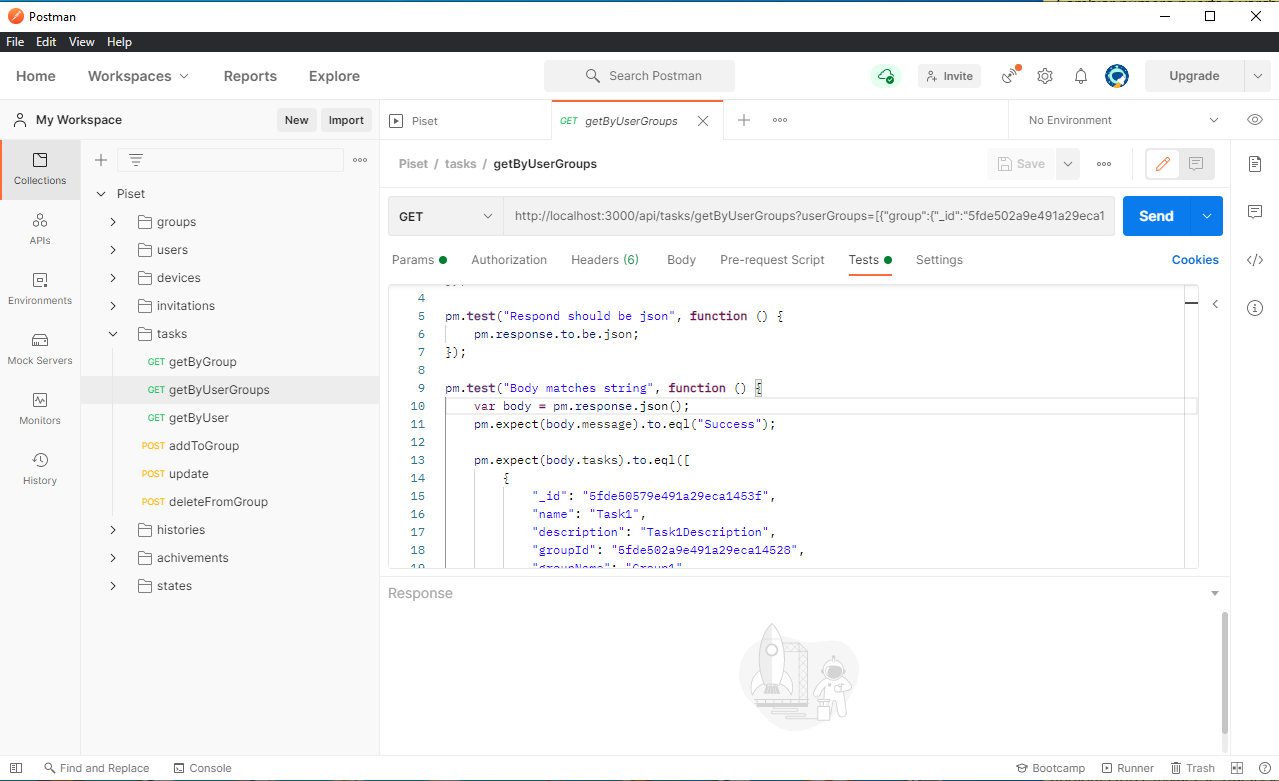
* Base de datos:
  + MongoDB Compass: Esta aplicación nos permite gestionar las diferentes bases de datos añadiéndolas al programa simplemente con un enlace de MongoDB, que incluye la dirección donde se aloja la base de datos y el usuario y contraseña del administrador. Con este programa podemos ver en una interfaz intuitiva el contenido de cada una de las bases de datos que estemos gestionando, así como editar sus datos y cargar documentos desde un archivo JSON:



* Back end y front end:
  + Intellij IDEA: Se trata de un entorno de desarrollo integrado (IDE) para el desarrollo de programas informáticos. Nos proporciona todas las herramientas y facilidades a la hora de desarrollas código, sobre todo código en JavaScript, ya que viene con un revisor de calidad de código y la función de autocompletado, así como una extensión para generar componentes y servicios de Angular. También nos permite ver en todo momento en que rama de la versión estamos trabajando y que archivos han sido editados respecto al último cambio de la rama. Lo hemos usado tanto para el desarrollo de la parte front como del servidor Express:



* + Postman: Postman es una herramienta de gestión de consultas HTTP. Con esta herramienta podemos crear consultas a nuestro servidor y ver como este responde frente a una petición real. Básicamente nos permite testear nuestro servidor creando las consultas que creamos pertinentes. Al ejecutar las consultas que hemos definido, la herramienta nos muestra los resultados de cada una. Podemos crear pruebas para cada consulta, como por ejemplo el tipo de resultado HTTP, el contenido del resultado, o el tiempo que tarda en hacer la consulta. Luego este listado de consultas se puede exportar a un solo archivo para ser ejecutado desde la línea de comandos. Esto últimos es muy importante, ya que es lo que usaremos para probar nuestro servidor en la integración continua:



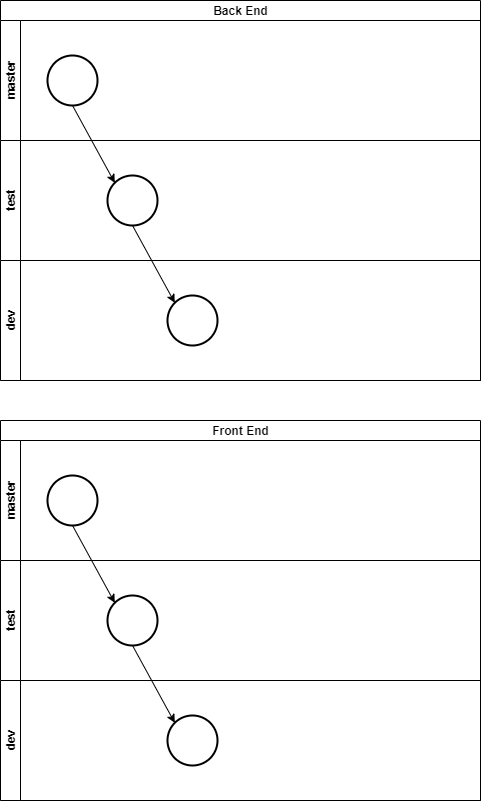
## Gestión de versiones

Como hemos comentado en apartados anteriores, la gestión de versiones es esencial a la hora de mantener nuestro código fuente, así como para almacenarlo y tener un histórico de los cambios que vamos aplicando con la posibilidad de volver a una versión anterior en todo momento. También sería muy útil a la hora de trabajar en equipo, ya que otro trabajador solamente tendría que crear una rama en base a otra principal y trabajar en su rama, hasta que ya lo tuviera todo listo y fusionaría sus cambios con la rama principal.

En este proyecto usaremos el controlador de versiones Git. Vamos a definir qué repositorios vamos a crear y que ramas van a tener.

La idea principal es tener en repositorios diferentes la parte de front end y la de back end. Esto se debe a que las dos partes estarán alojadas en entornos diferentes, por lo que a la hora de desplegarlas es conveniente tenerlas también en repositorios separados ya que del despliegue se encargará el GitHub Action que definamos en cada repositorio. Se podrían haber alojado ambas partes en un mismo repositorio, pero entonces se debería definir en el GitHub Action que archivos van a ser desplegados y en que entorno irían a parar, con la separación en dos repositorios simplificamos este trabajo.

En el siguiente diagrama hemos representado el esquema Git del control de versiones de este proyecto:



Cada piscina representa un repositorio, tenemos el repositorio del back end y del front end. Cada línea de la piscina representa una rama. En ambos repositorios hemos creado las tres ramas principales, la rama master, que representa el código que estará alojado en el entorno de producción, la rama test, que debe ser lo más parecida a la rama master y es donde se alojarán los cambios más recientes para que lo prueben los usuarios finales, y la rama de dev que es donde trabajaremos nosotros a la hora de desarrollar los cambios pertinentes. Como vemos gracias a las flechas del diagrama, cada rama está creada en base a la anterior. Por lo tanto, la metodología de trabajo sería la siguiente:

1. Creamos una rama en base a la rama principal dev y hacemos los cambios que tengamos que hacer.
2. Una vez los tengamos ya hechos con sus pruebas, fusionamos la rama que hemos creado anteriormente con la rama principal dev. Gracias a la integración continua, al fusionar nuestra rama con la rama dev, se ejecutarán todas las pruebas del proyecto, más las que hemos hecho nosotros para la parte que estamos implementando. Si han ido correctamente, tendremos la rama principal dev con los nuevos cambios.
3. Una vez la rama principal dev esté estable con todos los cambios que hayamos hecho, fusionaremos la rama dev con la rama test. Una vez más, la integración continua realizará todas las pruebas y si son satisfactorias dejará el código fusionado en la rama test y desplegará el proyecto en el entorno de test.
4. Una vez lo hayan validado los usuarios y nos den su confirmación para subir los cambios a la rama de master o producción, fusionaremos la rama de test con la rama mastes, la integración continua hará su trabajo y ya tendremos los nuevos cambios implementados y desplegados en producción.

## Entornos

En este apartado vamos a definir en qué entornos estarán alojadas las diferentes partes de nuestro proyecto y que relación guardan con los repositorios y ramas que hemos definido en el apartado anterior. El proyecto consta de tres partes principales que tendremos que alojar en diferentes servidores: front end, back end y base de datos. En el apartado anterior hemos definido tres ramas o entornos en los que trabajaremos: entorno de dev o desarrollo, entorno de test o preproducción y entorno master o producción. Por lo tanto, cada parte del proyecto tendrá que alojarse en tres entornos diferentes, vamos a definir en qué servidores vamos a alojar cada parte:

* Desarrollo
  + Front end: Crearemos una instancia local de desarrollo de Angular usando el comando de Angular “ng serve”. Este comando crea una instancia de desarrollo de la aplicación Angular que se reinicia rápidamente cuando apliquemos un cambio en el código fuente, lo que nos permite mayor agilidad a la hora del desarrollo. Aplicando este comando, Angular crea la instancia usando el archivo de entorno de desarrollo de Angular. Los archivos de entorno son archivos donde guardaremos parámetros que hay que tener en cuenta al ejecutar en diferentes entornos, como puede ser la dirección del servidor back end.
  + Back end: Crearemos una instancia local de desarrollo usando el comando “nodemon server.js desa”. Este comando ejecuta el servidor usando el paquete nodemon, que nos permite aplicar cambios en el código del servidor sin tener que reiniciarlo manualmente, ya se encarga de reiniciarlo la propia máquina. Le pasamos el parámetro “desa” para indicarle que la base de datos se encuentra en nuestra máquina local.
  + Base de datos: Crearemos una instancia de base de datos MongoDB en nuestra máquina local. El servidor sabe que tiene que apuntar a esta base de datos por el parámetro “desa” que le pasamos al iniciarlo anteriormente.
* Test
  + Front end: La parte de front estará alojada en la plataforma llamada Heroku. Se trata de una plataforma como servicio que nos permite alojar aplicaciones en la nube de forma gratuita. Además, tiene integración con nuestro código alojado en GitHub, lo que nos resultará de mucha ayuda para la integración continua. También cuenta con variables de entorno, que usaremos también para indicarle a las aplicaciones en que entorno se van a desplegar en el momento de despliegue. Es decir, podemos almacenar aquí los enlaces y credenciales para que la parte de front end se conecte con el servidor, y la parte de servidor se conecte con la base de datos.
  + Back end: La parte de back estará también alojada en la plataforma Heroku.
  + Base de datos: Alojaremos la base de datos en la plataforma MongoDB Atlas. Esta plataforma nos permite mantener una base de datos de MongoDB en un servidor externo de forma gratuita, y nos proporciona un enlace de conexión para que nos conectemos desde nuestro servidor.
* Producción
  + Front end: Alojada en otra máquina Heroku.
  + Back end: También se encuentra en Heroku.
  + Base de datos: Alojada en otra máquina de MongoBD Atlas

## Integración continua

En este apartado vamos a explicar cómo hemos implementado la integración continua en este proyecto. Lo que queremos conseguir es la automatización de los test y el despliegue en el entorno. Esto se ejecutará cada vez que se suban cambios a las ramas principales del repositorio que definimos anteriormente (master, test, dev). Más tarde explicaremos como hemos desarrollado las pruebas de software que se ejecutará en la integración continua.

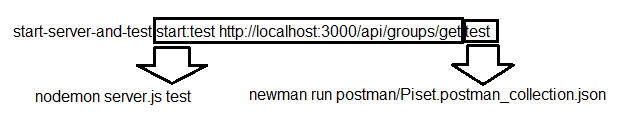
Como hemos mencionado anteriormente, vamos a usar la herramienta de GitHub Actions ya que vamos a alojar nuestro código en la misma plataforma de GitHub, por lo que la integración con esta herramienta es instantánea.

Para implementar la integración continua hay que generar un archivo yml y añadirlo en la ruta .github/workflows. Ahora vamos a explicar el archivo de configuración de la integración continua que hemos definido para cada uno de los repositorios.

* Back end:
* *# This is a basic workflow to help you get started with Actions*name: CI  
    
  *# Controls when the action will run.*on:  
   *# Triggers the workflow on push or pull request events but only for the master branch* push:  
   branches: [ test ]  
   pull\_request:  
   branches: [ test ]  
    
   *# Allows you to run this workflow manually from the Actions tab* workflow\_dispatch:  
    
  *# A workflow run is made up of one or more jobs that can run sequentially or in parallel*jobs:  
   build:  
   runs-on: ubuntu-latest  
   strategy:  
   matrix:  
   node-version: [14.x]  
   mongodb-version: [4.4]  
    
   steps:  
   - name: Git checkout  
   uses: actions/checkout@v2  
    
   - name: Use Node.js ${{ matrix.node-version }}  
   uses: actions/setup-node@v1  
   with:  
   node-version: ${{ matrix.node-version }}  
    
   - name: Start MongoDB  
   uses: supercharge/mongodb-github-action@1.3.0  
   with:  
   mongodb-version: ${{ matrix.mongodb-version }}  
   - run: mongorestore "mongodb+srv://cluster0.53xnf.mongodb.net/piset-test" -u root -p root --archive=database/test --drop  
   - run: npm install  
   - run: npm run ci  
   env:  
   CI: true

El archivo de ejecución de la integración continua se divide en varias partes:

* name: Indica el nombre de la acción de GitHub Actions.
* on: Esta parte indica en que eventos del control de versiones git se ejecutará la acción, en este caso hemos definido que se ejecute cuando se haga un push o un pull request a la rama de test. En la rama master lo tenemos definido para la rama master.
* jobs: Esta parte agrupa los trabajos que se van a ejecutar en distintas maquinas, para nuestro caso solo hemos definido una secuencia de actividades en una maquina Ubuntu, indicando la versión de node y la versión de mongodb.
* steps: Agrupa las acciones que se van a ejecutar. En estas partes podemos usar código que ya está hecho y es reutilizable, es decir, podemos incluir Actions. En nuestro caso tenemos un Action que instala node usando la versión que le hemos indicado anteriormente y otro que instala mongodb.
* run: Se usa para ejecutar los comandos que creamos pertinentes en la maquina virtual, estos comandos se ejecutarán en la ruta absoluta del repositorio que tengamos alojado en GitHub. Vamos a explicar cada comando:
  + mongorestore "mongodb+srv://cluster0.53xnf.mongodb.net/piset-test" -u root -p root --archive=database/test –drop: Este comando ejecuta un restore de la base de datos de MongoDB (alojada en el entorno de test de MongoDB Atlas, comentado en apartados anteriores) usando unos datos de test almacenados en el archivo del repositorio Git database/test. Con este comando dejamos la base de datos lista para ejecutar los tests pertinentes sobre el servidor. Se explicará en el apartado de pruebas porque se ha usado esta metodología.
  + npm install: Instala las dependencias de node definidas en el archivo package.json que necesita el servidor para ejecutarse correctamente.
  + npm run ci: Este comando sirve para ejecutar los scripts que definamos en el archivo package.json del proyecto. En este caso ejecuta el siguiente script, explicado en esta imagen para que quede más claro:



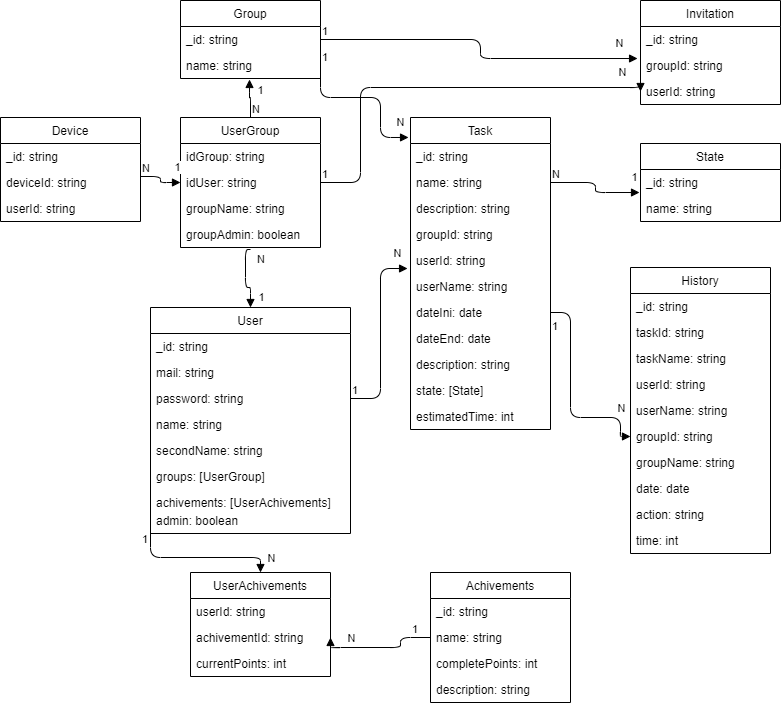
* start-server-and-test: Este comando o herramienta admite dos entradas, un comando que inicie un servidor y un comando que ejecute pruebas. El primero se ejecuta con nodemon server.js test y el segundo ejecuta unas pruebas de Postman sobre el servidor usando su ejecutable por línea de comandos llamado Newman, si todas las pruebas son satisfactorias el resultado es positivo y si falla una es negativo, indicándole a la integración continua que no debe desplegar nada en el entorno de Heroku.
* Front end:
* name: Node.js CI  
    
  on:  
   push:  
   branches: [ test ]  
   pull\_request:  
   branches: [ test ]  
    
  jobs:  
   build:  
    
   runs-on: ubuntu-latest  
    
   strategy:  
   matrix:  
   node-version: [10]  
    
   steps:  
   - uses: actions/checkout@v2  
   - name: Use Node.js ${{ matrix.node-version }}  
   uses: actions/setup-node@v2  
   with:  
   node-version: ${{ matrix.node-version }}  
   - run: npm ci  
   - run: npm run lint  
   - run: npm run test:gitHubActions  
   - run: npm run e2e

En este archivo hemos declarado los pasos que sigue la integración continua para el repositorio de front end. Los únicos cambios están en los steps que se usan. Vamos a explicar cada comando:

* npm ci: Se trata de un comando similar al npm install, pero se usa en un entorno automático como es el caso de plataformas de pruebas o entornos de integración continua, como este caso.
* npm run lint: Se trata de un comando que comprueba la buena escritura de código, es decir, si está todo bien formateado para favorecer la futura lectura.
* npm run test:gitHubActions: Ejecuta el script definido en el archivo package.json: ng test --browsers=ChromeHeadless --watch=false --code-coverage. Este comando ejecuta las pruebas unitarias de la aplicación Angular, definidas en su lenguaje integrado llamado Jasmin, se explicarán en más detalle en el apartado de pruebas.
* npm run e2e: Ejecuta las pruebas end to end, es decir las pruebas que consisten en simular lo que haría un usuario final de la aplicación. Se ejecutan usando la herramienta de pruebas Cypress.io. Estas pruebas se explicarán también en el apartado de pruebas.

Con esto ya tendríamos los procesos de integración continua para ambos repositorios implementada. Cabe mencionar que si algún comando de pruebas que hemos definido termina su ejecución con algún test fallido, automáticamente se pausa el proceso de integración, por lo que esa versión no llegaría a desplegarse en el entorno indicado del servidor Heroku.

## Base de datos



## Backend

Servidor, uso de postman…

## Frontend

Angular, componentes, pipes, servicios…

# Pruebas

## Pruebas unitarias

## Pruebas de integración

## Pruebas E2E

## Integración coninua

# Conclusión

## Relación proyecto-estudios cursados

## Trabajos futuros y líneas de mejora

-Añadir comentarios a las tareas

-Poder dividir las tareas en proyectos, aparte de por grupos.

-Añadir más estados de tareas al grupo.