# Índice

[Índice 1](#_Toc481751985)

[Ilustraciones 3](#_Toc481751986)

[Tablas 4](#_Toc481751987)

[1. Introducción: 5](#_Toc481751988)

[2. Objetivos y alcance: 6](#_Toc481751989)

[3. Estudio del mercado: 8](#_Toc481751990)

[3.1. Amazon web services (AWS): 8](#_Toc481751991)

[3.2. Cloud9: 9](#_Toc481751992)

[3.3. Docker: 10](#_Toc481751993)

[3.4. TensorFlow: 11](#_Toc481751994)

[4. Enfoque de la aplicación: 12](#_Toc481751995)

[4.1. Concepto de usuario y administrador: 12](#_Toc481751996)

[4.2. Concepto de proyecto y ejecución: 12](#_Toc481751997)

[4.3. Concepto de grupo y permiso: 13](#_Toc481751998)

[4.4. Definición formal del *Proyecto UniApi*: 13](#_Toc481751999)

[5. Análisis de alternativas: 14](#_Toc481752000)

[5.1. Elección de arquitectura del sistema: 14](#_Toc481752001)

[5.2. Repositorio de información: 15](#_Toc481752002)

[5.3. Negocio de la aplicación: 16](#_Toc481752003)

[5.4. Aplicación web: 17](#_Toc481752004)

[5.5. Transferencia de archivos: 21](#_Toc481752005)

[5.6. Ejecución de proyectos: 22](#_Toc481752006)

[6. Planificación temporal: 23](#_Toc481752007)

[6.1. Planificación de la duración entorno de ejecución: 23](#_Toc481752008)

[6.2. Planificación de duración de grupos: 24](#_Toc481752009)

[6.3. Planificación duración proyectos: 24](#_Toc481752010)

[6.4. Planificación duración ejecuciones: 24](#_Toc481752011)

[6.5. Planificación duración UserLogin&Persona: 25](#_Toc481752012)

[7. Ampliaciones: 27](#_Toc481752013)

[8. Conclusiones: 28](#_Toc481752014)

[9. Bibliografía: 29](#_Toc481752015)

# Ilustraciones

[Ilustración 3‑1 AWS logo 6](#_Toc481751957)

[Ilustración 3‑2 AWS WebPage 6](#_Toc481751958)

[Ilustración 3‑3 Cloud9 logo 7](#_Toc481751959)

[Ilustración 3‑4 Cloud9 IDE 7](#_Toc481751960)

[Ilustración 6‑1 Diagrama Grant 23](#_Toc481751961)

[Ilustración 6‑2 Tabla de tareas 24](#_Toc481751962)

# Tablas

[Tabla 6‑1 Planificacion de grupos 23](#_Toc481751977)

[Tabla 6‑2 Planificación de proyectos 23](#_Toc481751978)

[Tabla 6‑3Planifucacion de ejecuciones 23](#_Toc481751979)

[Tabla 6‑4 Planificación UserLogin&Persona 24](#_Toc481751980)

# Introducción:

El siguiente documento es de carácter divulgativo desarrollado para la asignatura de “*Trabajos final de grado*” de la escuela politécnica de ingeniería de Gijón. El “Trabajo final de grado” o “TFG” como va a ser nombrado desde ahora, es la declaración de habilidades y conocimientos de un estudiante en un grado de ingeniera de la escuela.

El TFG se basa en una propuesta realizada por una entidad externa. Pudiendo ser un profesor, una empresa o el propio estudiante. La propuesta a realizar esta propuesta por Raquel Blanco Aguirre: Se basa en un desarrollo software, con el objetivo de “*crear una aplicación web que gestione un repositorio*”.

Desarrollaremos el significado de la propuesta con el fin de tener un objetivo principal para el TFG. Una aplicación web se basa en una página web cuyas interacciones del usuario, son gestionadas, medidas y reaccionadas por un servidor web. Podría decirse que una aplicación web es una página web con una inteligencia dada por un servidor. Un repositorio es un sistema donde se almacenan datos para posteriormente consumirlos. Se podrá decir que es como un almacén. Como todos almacenes, un repositorio contiene medidas de seguridad, medios de gestión de elementos y gestión de logística. En resumen, tenemos que crear una página web con aditivos, que se conecta a un almacén digital.

Los TFG’s de desarrollo conllevan una carga de documentación por sí misma. Se necesitan documentar los análisis y diseños que se van implementar antes de realizar un desarrollo. Para que sea más fácil la comprensión de los diversos documentos y su estructura. Se va a realizar una descripción breve en este documento.

* Memoria: Documento principal del TFG, explicación del proyecto.
* Presupuesto: Documento que describe el coste del proyecto.
* Documentos técnicos: Documentos de desarrollo software.
  + Requisitos del sistema: Serie de requisitos (mandatos) impuestos para el desarrollo del proyecto.
  + Análisis del sistema: Descripción de alta abstracción del funcionamiento del proyecto a realizar.
  + Diseño del sistema: Descripción de media y baja abstracción del funcionamiento del proyecto a realizar.
  + Pruebas del sistema: Diseño y realización de las pruebas que se han utilizado en el desarrollo del proyecto.
* Manual del usuario: Documentación del uso de la versión del proyecto realizado.

# Objetivos y alcance:

El objetivo del proyecto es la creación de una aplicación web que permita realizar la gestión de un repositorio. La aplicación deberá ser capaz de gestionar todas las funciones del repositorio, comunicación del repositorio, gestión de usuarios, gestión de elementos, navegación por sus departamentos.

De por si el objetivo del TFG es de un carácter muy abierto, dejando la búsqueda del objetivo secundario del TFG para el alumno. La búsqueda del objetivo se realiza en las prácticas en empresa del alumno, en el servicio de *I+D+I de Treelogic*, donde al alumno se le muestra una necesidad que puede ser cubierta por el objetivo principal del TFG.

La necesidad viene imperada por el uso recurrente de programas y proyectos de carácter científico, en un ámbito informático que facilita su desarrollo. En concreto, los diversos científicos y graduados, apoyándose en las tecnologías de desarrollo de alta abstracción, como R, Python o Octave, desarrollan diversos programas no enfocados a las tecnologías web. Esto provoca un proceso de traducción para enfocarlos a las tecnologías web. Donde se suelen encontrar diversos problemas de traducción ocasionando por ello, fallos de ejecución en el software que normalmente antes no estaban. Estos problemas suelen encontrarse debido al poco conocimiento del desarrollador de tecnologías web en el proyecto del científico o el graduado.

La solución a esta necesidad por parte del creador de este proyecto. Es la creación de un repositorio que almacene diversos proyectos de carácter educativo o científico, donde puedan ser consultados (ejecutados) por las diversas aplicaciones TIC que deseen estos datos para su propio funcionamiento. Debido al complejo esfuerzo que conlleva comunicarse con un repositorio, se creará una aplicación web que ayude a los usuarios y administradores a comunicarse con el repositorio de una forma más natural y más abstracta.

Esta solución viene dada por el uso cada vez más extendido y globalizado de las aplicaciones web en nuestra vida diaria. Existiendo un vacío de comunicación entre las comunidades científicas universitarias y los diversos modelos de negocio en el mundo web.

Con este proyecto se podrá incluso generar un nuevo modelo de negocio para las universidades. Este modelo se está empezando a aplicar en diversas universidades de alto calibre como Stanford con sus redes neuronales y conjuntos de datos para el uso de Machine learning o las grandes bases de datos de Cambridge para el uso de computación sintáctica. En estos ejemplos se puede ver que las universidades se van a convertir en grandes fuentes, literalmente, de conocimiento sin poder de modelo de negocio alguno. Donde las empresas con un objetivo de negocio pagarán por el uso y consumo de las API’s de las universidades. Donde todos los beneficios de ese uso se acapararán entre los investigadores y las universidades y no en las empresas. Haciendo que los ingresos de las universidades sean orientados en el I+D+I y en la automejora de la universidad.

La aplicación a realizar es demasiado poco extensa como para poder realizar una implementación de esta gran necesidad, pero es un llamado o aviso para la posible realización de un trabajo fin de master o proyecto de un equipo de investigación de la universidad orientado a la realización de una API de la universidad de Oviedo.

Aun así, el proyecto a realizar es el núcleo de la idea desarrollada anteriormente. Es decir, una aplicación web que podrá ser usada por la universidad para comunicar diversos proyectos web con proyectos científicos para demostrar el poder de esta idea.

Es por eso que el proyecto se ha denominado **UniApi** ya que lo que intenta esta aplicación es ser la versión primigenia de una API de la universidad de Oviedo.

# Estudio del mercado:

A continuación, se va a describir la posición actual en el mercado frente a la necesidad expresada en el punto anterior de objetivo y alcance. Con el fin de generar soluciones e ideas para el desarrollo del proyecto **UniApi**.

Aunque ya existe la tecnología para realizar una solución a esta necesidad, no existe una solución de carácter equivalente a la que se expone en este proyecto. Cabe destacar que, para apoyarse en ejemplos para el desarrollo de una solución, tenemos un conjunto de modelos de negocio que han tenido éxito en sus campos y pueden ser un de ayuda para respaldar la idea del proyecto.

## Amazon web services (AWS):



Ilustración ‑ AWS logo

Amazon web services es un claro ejemplo de que los productos de carácter TIC bajo demanda está a la orden del día. Este servicio se podría considerar como un supermercado de las tecnologías web. Máquinas virtuales, DNS, Active directories, Almacenamiento en la nube…

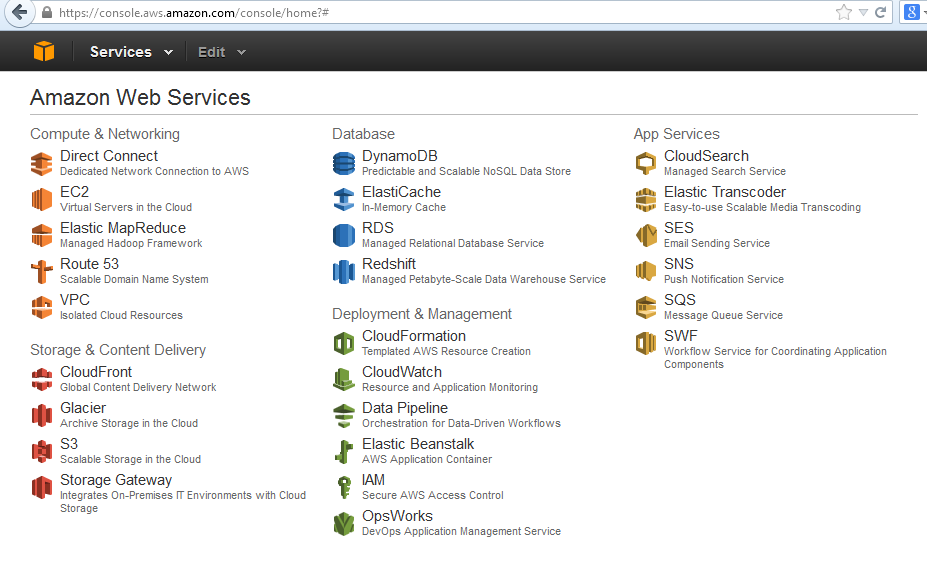


Ilustración ‑ AWS WebPage

Todas estas tecnologías están en la nube, externa al usuario y sus condiciones. Los usuarios solo pagan por lo que desean y el tiempo que lo desean. De este modo, el usuario no se tiene que preocupar de configuraciones o de dependencias de tecnologías y tiene lo que quiere cuando lo necesita. Quizá, esta sea la meta principal de **UniApi**: **Debe de dar la solución al usuario bajo demanda y reduciendo a la mínima la configuración a aportar por parte tanto del consumidor (App web, usuarios) como del productor (científicos, graduados, investigadores)**. Todo acaba siendo una aplicación web transparente como un mundo por debajo como Amazon web services.

## Cloud9:



Ilustración ‑ Cloud9 logo

Cloud9 es un IDE de desarrollo en la nube que sigue la dinámica de transparencia y mínima configuración para el usuario que buscamos de UniApi. En Cloud9 solo debemos elegir el tipo de proyecto que deseamos desarrollar y el IDE se encarga de realizarnos un entorno de desarrollo configurado para la tecnología que vamos a desarrollar.

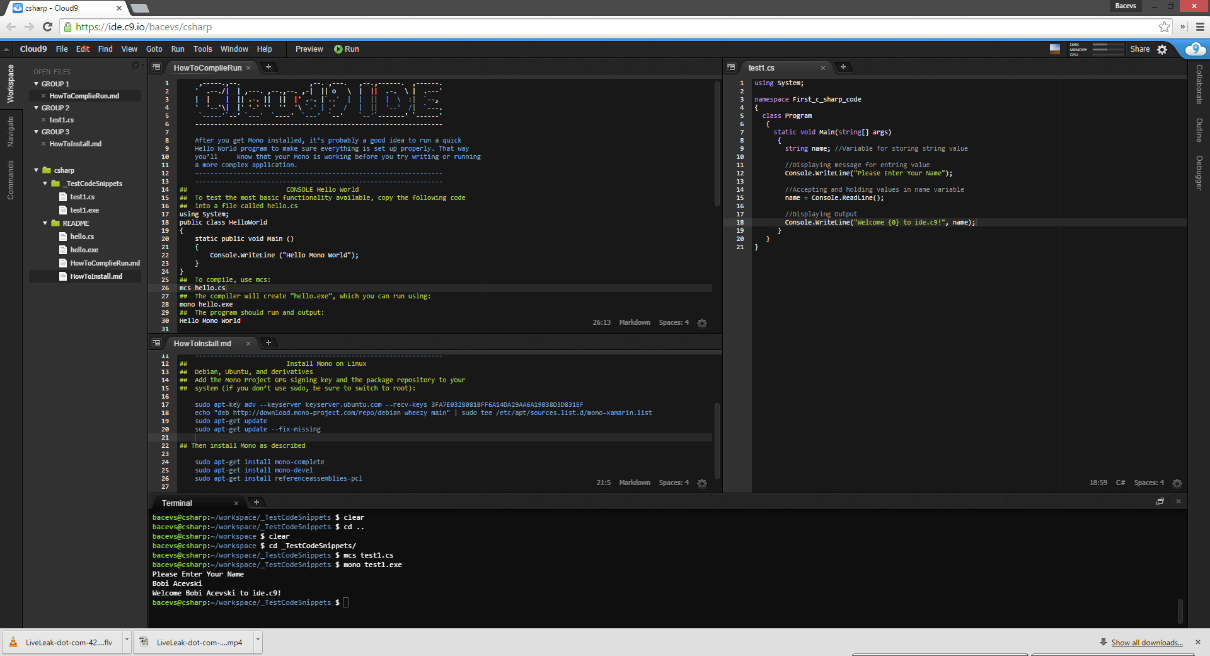


Ilustración ‑ Cloud9 IDE

Cloud9 no es un ejemplo para la cobertura de desarrollo que vamos a realizar. Pero si es un claro ejemplo de la proyección que se debería dar del proyecto, si este se ampliara. **UniApi** debe aprender de Cloud9 **la ausencia de configuración para el desarrollo y prueba de los códigos de manera genérica teniendo un abanico de tecnologías de desarrollo diferentes entre sí.**

## Docker:

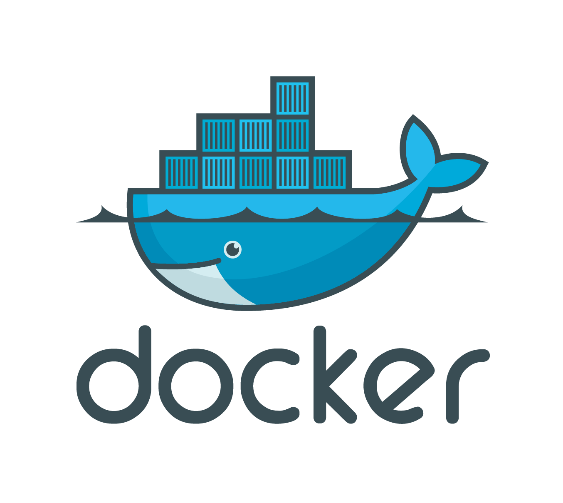


Figure ‑ Docker logo

Docker es una tecnología de producción que se está poniendo de moda estos años. Los proyectos complejos de programación conllevan consigo un sinfín de configuraciones, dependencias, programas de interacción entre nuestro proyecto... Que generan que las personas que quiera realizar un proyecto complejo, deban tener conocimientos avanzados de sistemas operativos para la instalación y configuración de proyectos para su puesta en marcha o su mera prueba ante errores.

Esto genera un periodo de inactividad y baja eficiencia en el desarrollador de lenguajes de cualquier tipo. Docker gracias a su sistema de Docker-Scripts genera entornos pre configurados e estables para la producción de software. Según los propios estudios de docker un desarrollo en tecnologías de pre configuración generan una mejora en el desarrollo, modificación y producción del programa.

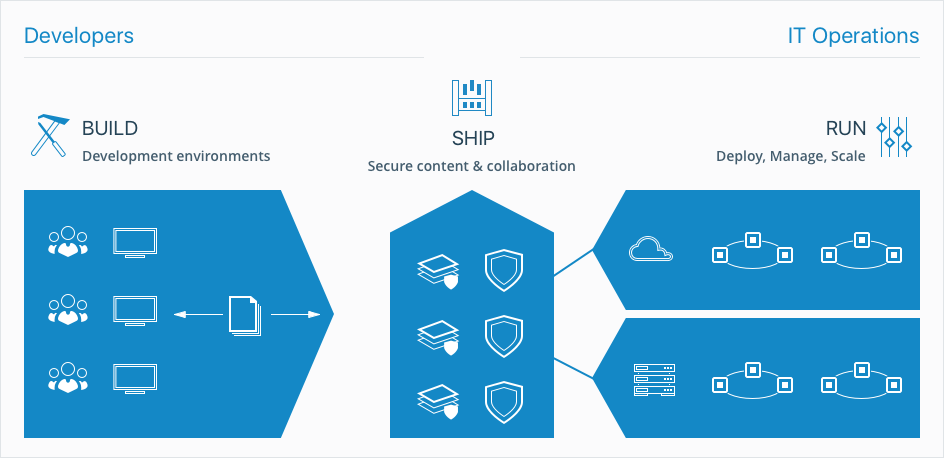


Figure ‑ Aquitectura docker

UniApi debe tomar como lema para el desarrollo, la facilitación de configuraciones preinstaladas para facilitar el desarrollo a estudiantes, investigadores o científicos que no tienen los recursos ni el conocimiento para aprender teoría de sistemas operativos. UniApi facilitara entonces el desarrollo e incluso producción de los proyectos de sus usuarios para que después puedan venderlos por su utilización a las diversas empresas.

## TensorFlow:



Figure ‑ Tensor flow logo

TensorFlow es una tecnología realizada por google que facilita el desarrollo de modelos de predicciones usando redes neuronales o Machine Learning. TensorFlow desarrolla la misma necesidad de que UniApi, pero solo para el sector de la inteligencia Artificial. UniApi debe aprender de este rival que el mercado existe y su necesidad también. Solo que los elementos claves para realizar una API de conocimiento por demanda no se han realizado todavía.

En tensorFlow el investigador solo debe aplicar los conjuntos de datos y destacar las variables que tienen importancia para las predicciones. Utilizando un lenguaje tan extendida en la comunidad científica como es el Python. Cualquier científico que tenga conocimientos en Regresiones lineales y matemáticas, es capaz gracias a este framework de realizar aprendizajes automáticos a maquinas con las cuales, si se programara de manera natural, conllevaría realizar una carrera mínima de ingeniería informática y estudios postgrado.

# Enfoque de la aplicación:

Aunque el proyecto **UniApi** ya se ha ido desarrollando en puntos previos de esta memoria. Sera en este punto donde describiremos **UniApi** de manera concisa. Nos centraremos en la parte abstracta del proyecto, es decir, en qué va a hacer UniApi en vez de en cómo se va a hacer. Si se desea saber cómo se va a hacer, se invita a los lectores a que lean los artículos **Análisis del sistema** y **Diseño del sistema**, integrantes de los documentos técnicos de esta memoria.

Como se explica en el punto objetivo y alcance. El objetivo de este proyecto es el de crear una aplicación web que gestione un repositorio. Este repositorio no va a ser un repositorio normal, sino que gestionará diversos proyectos que podrán contener distintos lenguajes de programación. Este repositorio estará gestionado por una aplicación web que mostrara los datos de gestión de manera fácil para los usuarios y a la vez hará de intérprete entre el usuario y el repositorio.

A continuación, se explican unos conceptos básicos para la descripción del enfoque del proyecto **UniApi**:

## Concepto de usuario y administrador:

Un **usuario** es una entidad física que se comunica con el repositorio para crear entidades como grupos, proyectos o ejecuciones. Un usuario no tendrá permisos para realizar acciones administrativas esa es la función del administrador.

Un **administrador** no es más que un usuario con permisos especiales, pudiendo gestionar todas las entidades generadas por el repositorio y realizar modificaciones especiales que un usuario no puede realizas.

Existe una entidad especial llamada aplicación. Una aplicación es un permiso realizado por un usuario para realizar acciones con su nombre por parte de aplicaciones web.

## Concepto de proyecto y ejecución:

**UniApi** entenderá como “**proyecto**” una entidad lógica que almacenará varias cosas. Un “proyecto” contendrá un código que se ejecutará por orden de un usuario, utilizando unas entradas definidas por el mismo usuario o prefijadas por el creador del “proyecto”. Tras su ejecución el “proyecto” contendrá una salida formateada para la comprensión tanto del usuario como de la máquina.

El repositorio generara ejecuciones de “proyectos” en función de las órdenes de los usuarios o aplicaciones. Cada ejecución será diferente en función del tipo “proyecto”. Las **ejecuciones** serán unos espacios lógicos (almacenamiento, cpu, memoria…) ocupado por el repositorio para en la ejecución de un proyecto.

## Concepto de grupo y permiso:

Un **grupo** es una entidad lógica de almacenamiento de proyectos y otros grupos por parte de diversos usuarios que forman parte de este grupo. En este grupo imperara unas órdenes dadas por el propietario, esas órdenes son heredadas a los grupos hijos, conocidos como **subgrupos**, Esas órdenes las conocemos como **permisos**. Los permisos son inalterables salvo por el propietario y afectan a las acciones realizadas tanto en el grupo como en los subgrupos y en los proyectos que contenga este grupo.

## Definición formal del *Proyecto UniApi*:

Al final **UniApi** se basa en una aplicación web que se comunica con un repositorio, por la cual, diversos usuarios de distinta índole organizados en grupos creados por ellos. **Crean y gestionan proyectos de diversos tipos**, los cuales, pueden ser ejecutados por los usuarios que ellos permitan. Estos proyectos podrán ser **ejecutados con unas entradas variables** y **recibir a cambio unas salidas determinadas por el proyecto** ejecutado en el repositorio. Toda esta información es guardada en el repositorio y no volátil pudiendo ser consultada por los usuarios y aplicaciones cuando ellos deseen.

# Análisis de alternativas

En este punto se describen las distintas alternativas analizadas para el desarrollo del proyecto, así como las justificaciones de las elecciones realizadas.

## Elección de arquitectura del sistema:

Cuando hablamos del sistema nos referimos a un conjunto de subsistemas que unidos entre si proporcionan unas funcionalidades, que, unidas sinérgicamente, resuelven los problemas planteados por el proyecto. Para hablar del sistema del proyecto **UniApi**, debemos centrarnos en dos grandes subsistemas: El repositorio (encargado del almacenamiento y la gestión de la información) y la aplicación web. Al realizar el diseño de estos dos grandes subsistemas podremos usar dos estilos tradicionales:

* **Sistema monolítico**: Donde toda la aplicación se desarrolla de abajo a arriba. Todas sus funcionalidades están en el mismo programa, y tanto el repositorio como la aplicación web son el mismo programa ejecutándose en el mismo lugar.
* **Sistema estructurado en capas**: La aplicación se separa en pequeños subsistemas o capas independientes entre sí, que por sí solas funcionan (se ejecutan) independientemente y se comunican entre ellos para realizar tareas. Existirá como tal un director u orquestador que dirigirá la gestión total del proyecto.

Los pros y los contras de estos sistemas son muy extensos, existe por ello en internet un sinfín de documentación que abarca estos sistemas, en esta documentación se narra desde las mejoras en seguridad que aporta el sistema estructurado en capas usando la tecnología de MV (Virtual Machine). Como el uso de sistemas monolíticos para un desarrollo más ágil en aplicaciones pequeñas y medianas. El autor ha querido reseñar varios de estos puntos para la comprensión de la duda del autor sobre qué sistema utilizar, pero, basándose en el problema principal del proyecto. Cabe destacar que el autor ha llegado a esta conclusión:

Ante un proyecto de carácter pequeño y mediano donde los servicios se ejecutarán en pequeños servidores u ordenadores de poco calibre. Es mejor para el programa el dividirse en pequeños subprogramas independientes. Que trabajan al unísono entre si. También cabe destacar el carácter crítico del entorno de ejecución, ya que un error en este entorno podría acabar con la ejecución de TODA la aplicación. Es por eso que unas subdivisiones de los subsistemas darían una buena tolerancia a fallos.

Es por eso que el autor ha decidido romper lo máximo posible la aplicación generando subprogramas que trabajen independientemente. Los siguientes análisis de alternativas van a tener como punto de partida el buscar una solución que nos permita generar la mayor independencia posible de los subsistemas del programa. Con el fin de mejorar la disponibilidad de un servicio tan poco tolerante a fallos como es la ejecución de un programa.

## Repositorio de información:

El repositorio de información tiene como objetivo el gestionar datos y almacenarlos de forma permanente. Cuando hablamos de almacenar información debemos hablar de persistencia. La persistencia es un tipo de dato en informática que puede ser de dos tipos: a corto plazo (RAM, Archivo temporal…) o a largo plazo (Bases de datos, Ficheros…). Es evidente que nuestro repositorio utilizara datos persistencia a largo plazo para almacenar la información.

Hay muchas formas de almacenar datos en persistencia a largo plazo. Para elegir una forma debemos centrarnos en que va a guardar nuestro repositorio y que particularidades tiene cada tipo de dato:

* **Datos personales**: Son de poco peso, no son muy importantes para el funcionamiento de la aplicación. No es importante que los datos estén consistentes siempre.
* **Grupos**: Son de poco peso tienen un poder relacional muy elevado, es decir existe muchas entidades relacionadas con un simple grupo. Es necesario que estos datos y relaciones estén siempre disponibles y a un tiempo razonable para una aplicación web.
* **Proyectos**: Se pueden dividir en dos partes. Por un lado, los datos del proyecto, los cuales tienen un carácter muy parecido a los grupos. Es decir, grandes cantidades de relaciones entre otras entidades y datos importantes que es necesario tener disponibles para aplicaciones web. Por otro lado, existen códigos y ficheros del proyecto. Estos datos no son consultados por la aplicación web se necesitan menos que los datos básicos y suelen ser usados por las ejecuciones.
* **Ejecuciones**: Entidades lógicas con datos sobre las ejecuciones que se hacen sobre los proyectos. Deben tener una disponibilidad elevada y un tiempo de consulta medianamente elevado.

Tras analizar los diversos tipos de datos, nos damos cuenta que hay dos grandes tipos de datos: Unos, los cuales, necesitan una disponibilidad elevada y una velocidad de consulta cuanto más rápida mejor. Y otros que no son necesario tanto la velocidad como la consistencia de los datos y su acceso a ellos en cualquier momento.

El autor aquí ha desarrollado dos varias elecciones:

* La primera la más sencilla es donde ubicar los proyectos de los usuarios. Podría utilizar una base datos relacional. Estas BD se rigen por ser muy consistentes y de gran seguridad, algo que necesitamos para no corromper los datos. La segunda opción es usar la jerarquía del SO que esté usándose. Esta opción contiene todos los factores que buscamos y facilita una simplicidad que no puede conseguir la BD relacional. El autor cita la solución que va a aportar:

Para el guardado de proyectos de usuarios en UniApi, se va a defender el uso del sistema de archivos de los SO. Este sistema nos garantiza una consistencia y lo más favorecedor un entorno amigable con la ejecución.

* Para los demás datos sus cualidades principales es que no es necesario que sean muy consistentes, y que contienen muchas relaciones entre ellos. Toda la información de una aplicación por defecto se almacena en una base de datos. Esto hace que nos preguntemos si queremos una base datos Relacional o una No relacional. Ambas son buenas para esta solución, aunque el autor cita esta alternativa como solución:

La característica más importante de los datos a guardar son las relaciones entre ellos. Es por eso que vamos a utilizar una base de dato orientada a grafos (Neo4j). Donde el motor de esta base de datos son las relaciones entre los nodos. Una cualidad que tienen todos nuestros datos. Las BD orientadas a grafos son BD no relacionales teniendo las cualidades típicas de una NO-SQ. Una velocidad de acceso elevado y una disponibilidad elevada.

Como explica el autor se va a utilizar como BD para el almacenamiento de datos una BD orientada a grafos. Existen varias de este tipo en funcionamiento, no obstante. La proveedora de la propuesta de proyecto, ha solicitado el uso de la **Base de Datos Neo4j.**

## Negocio de la aplicación:

El negocio de la aplicación estará albergado en el repositorio tendrá el objetivo de gestionar todos los servicios mandados por los diversos usuarios de UniApi. Esta parte de la aplicación puede ser realizada en cualquier lenguaje de programación que se quiera.



Figure ‑ neo4j logo

Aquí hay poco tema de discusión en elección de lenguaje de programación, salvo cual es el mejor. Todos valen para esta tarea. En concreto el usuario va a utilizar el lenguaje de programación **Java**. Ya que este lenguaje le lleva acompañando desde el inicio de la carrera.

Para facilitar el desarrollo de la aplicación de negocio y en general la aplicación de repositorio el autor ha pensado en introducir ciertas tecnologías que ha ido aprendiendo en sus prácticas en empresas. Estas tecnologías favorecen al uso de subsistemas independientes que se comunican entre ellos:

* **Spring**: Es un framework que se basa en la inyección de dependencia para realizar aplicaciones e introducir funcionalidades que no interfieren en las ejecuciones independientes de otras funcionalidades.
* **Maven**: Maven es un framework que facilita la búsqueda y mantenimiento de software de terceros que se utiliza en aplicaciones java desarrolladas por nosotros mismos.



Figure 5‑2 Spring Logo

Figure ‑ Maven logo



Figure ‑ Java Logo

## Aplicación web:

La aplicación web tiene como objetivo el comunicarse con el usuario. Su función es generar vistas interactivas, las cuales, son monitorizadas buscando eventos generados por el usuario. La aplicación web lo interpreta y realiza nuevas vistas en función de estas órdenes.

Los sistemas de aplicaciones web están divididas por dentro en dos subsistemas. Estos subsistemas se ejecutan en lugares diferentes. A estos subsistemas se le conocen como **Lado cliente** y **Lado servidor**. En el pasado estos lados estaban muy separados y cada lado tenía una funcionalidad diferente y única. El lado cliente se enfocaba en representar las vistas y gestionar los eventos de los usuarios, que eran enviados al lado servidor. El lado servidor gestiona los eventos, interpreta las respuestas y genera vistas para ser ejecutadas en el lado cliente.

Con paso del tiempo, el lado cliente ha ido cogiendo funcionalidades del lado del servidor, pudiendo realizar todas las funcionalidades del servidor. Es aquí donde se descubren las diversas alternativas de las aplicaciones web, pudiendo elegir hasta que cierto punto que cantidad de peso de la aplicación queremos proporcionar al cliente. A continuación, el autor expresara ciertas alternativas que pueden ser usadas en la aplicación web y por último la seleccionada por el autor.

* **Multiple Page Aplication (MPA)**: Esta visión es la clásica. En el todo el peso está dentro del servidor, el cual, gestiona todos los eventos y genera las vistas. Para realizar esta visión se debe usar la tecnología **MVC (Modelo-Vista-Controlador)**. Existen muchas tecnologías para realizar esta solución mostraremos dos de las más conocidas.



Figure 5‑5 Spring MVC logo

Figure ‑ JSF logo

JSF y Spring son dos grandes framework para la construcción de multiple page aplication. JSF está realizado por el core de desarrollo de java para la construcción de páginas web dinámicas. Spring MVC está realizado por el CORE de Spring, optimizado para la inyección de dependencia que sigue este CORE.

Este tipo de visión provee un gasto computacional en el repositorio elevado, ya que todas las gestiones web, deben ser gestionadas por el servidor web (subsistema posible en el repositorio). El ejecutar todas las gestiones en el servidor, provee un control de seguridad elevado y una gestión de las vistas más cercanas al negocio.

* **Single Page Aplication (SPA)**: Es una visión nueva creada en torno al 2010. Esta tecnología promueve que todo el peso de la aplicación se encuentre en cliente donde se gestionan todos los eventos y se realizan las vistas de la aplicación. Esta visión tiende a transformar a los servidores en meros repositorios de información.

Esta solución alivia el poder computacional de los servidores, obligando a que el poder computacional lo acarre el navegador del cliente. Esto genera que nuestros servicios sean más rápidos y puedan atender a más gente. En contra posición, esta visión contiene un fallo y es que todo el código es comunicado al cliente. El cual, podrá observarlo, identificarlo y posiblemente copiarlo o modificarlo.   
  
Para realizar sistema es necesario el uso de servidores de apoyo que proporcionen los códigos y recursos necesarios para la aplicación, así como los datos para ser representados. Para el lado del cliente es necesario un potente programa o framework que realice el trabajo MVC en el lado cliente.

**Para la realización de servidores**, se usará sistemas REST. Estos servidores son servidores web pero especializados en la arquitectura REST. Existen múltiples tecnologías de servidores REST, pero proporcionaremos varias de distintos lenguajes.

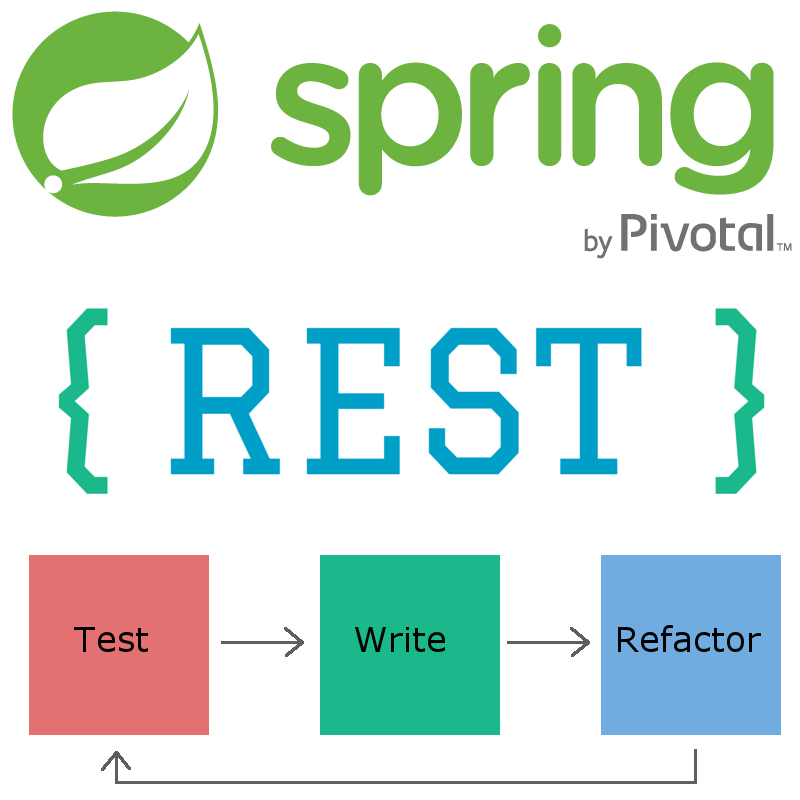


Figure 5‑7 Spring Rest Logo



Figure 5‑8 django logo

Figure 5‑9 Express.js logo

Los servidores REST existen para cualquier tipo de tecnologia, cada uno defiende su modelo. En especial vamos a destacar dos tecnologias: **ExpressJS** es un servidor WEB MVC implentado en JavaScript. Este servidor es muy flexible ya que se puede adaptar como Servidor REST para lanzar aplicaciones web SAP. Spring REST es un servidor REST muy potente implementado en Java, es muy estricto con su implementación. Pero tiene unas funcionalidades y extensiones muy potentes. Django es el mellizo de expressJS implementado en python. Es agil y habil sirve para realizar aplicaciones MVC como para realizar servicios REST.

Cuando se tiene uno o varios servidores que proporcionan la informacion y los codigos. Se debe de utilizar un framework MVC que gestione todo el computo en el cliente. Para la **realización de la parte cliente** podemos escoger entre un sinfín de tecnologias (debido al auge de estos tiempos). El autor citara los tres grandes pilares de esta tecnologia:

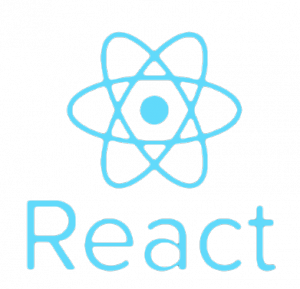


Figure 5‑10 React.js Logo

Figure 5‑11 Angular.js Logo



Figure 5‑12 Vue.js Logo

Todas estas tecnologías son tecnologías implementadas en JavaScript y son ejecutas por el navegador cliente. Cada una tiene una visión diferente de la implementación de una MPA. Angular es un framework desarrollado por google que se basa en la actualización del modelo habitual de MVC, pero orientado a paginas MPA. React es un framework desarrollado por Facebook siguiendo un modelo de componentes. Por ultimo Vue es un MVVM, es un framework relativamente nuevo e integra las cualidades de los dos anteriores.

A continuación, se va a citar la solución del autor sobre la selección de alternativas para la elección de la aplicación web:

Para la selección de la **aplicación web**, se va a utilizar un modelo **Single Page aplication (SAP).** Este modelo es interesante para el proyecto UniApi ya que realiza una sub división en la aplicación web entre la APP-Web y el negocio. Por ello se utilizará un servidor web básico basado en Express.JSdebido a su simplicidad. Que inyectara el framework **Angular.js**.   
  
Realizando la separación APP-Repositorio. **El repositorio** solo tendrá la función única y exclusiva de realizar su propio negocio y comunicarse con las demás aplicaciones. Es por eso que el repositorio empleara un **servidor REST Spring**, que esta implementado en el mismo lenguaje que el propio repositorio. Y que será comunicado por la aplicación Angular.js para realizar las gestiones del negocio.

Como ha descripto el autor el proyecto UniApi tendrá una implementación web basado en SAP. Esta visión genera unas ciertas mejoras y desventajas:

* **Dos núcleos de ejecución**: App Web (lado servidor) y repositorio. Esto genera que una independencia que en caso de fallo pueda ser asumida. Así como la replicación exitosa de estos elementos.
* **Peso computacional**: La parte servidor ejecutara justo lo necesario para realizar el servicio. Esto favorece mucho en un sistema como **UniApi**, que busca ejecutarse en servidores de poco poder computacional.
* **Posibles mejoras**: Todas las posibles mejoras y ampliaciones se realizarán de forma independiente. Pudiendo mejorar un núcleo sin tener que modificar el otro.
* **Fallo de seguridad (Desventaja)**: El código de negocio es ubicado en el cliente de manera transparente.

## Transferencia de archivos:

Cuando un usuario quiera agregar, modificar o simplemente borrar un elemento de un proyecto en el repositorio. El repositorio tendrá que enviar múltiples archivos de alta capacidad. Es a esto lo que llamamos trasferencia de archivos. Existe múltiples tecnologías para la transferencia de archivos desde que se inventó internet.

Estas tecnologías promueven la transferencia de datos genérica para todos los dispositivos, independientemente tanto del código como del sistema operativo. Es por eso que deberemos utilizar una tecnología de sincronización de archivos. Ya que, si quisiéramos implementar un medio de comunicación, deberíamos ocuparnos de un montón de problemas de bajo nivel de abstracción. A continuación, describiremos ciertas tecnologías. Mostraremos sus pros y sus contras, además el autor explicara la solución utilizada para el proyecto:

* **FTP (File transfer Protocol)**: Este protocolo es un sistema de transferencia de archivos entre sistemas conectados entre sí. Ese protocolo funciona utilizando un conjunto de ordenes emitidas de cliente a servidor utilizando la tecnología SSH (Comunicación cifrada). Este protocolo facilita mucho la transferencia entre los usuarios y el repositorio, pero no está pensado para generar modificaciones automáticas de las fuentes de datos del cliente.
* **HTTP**: Todo el protocolo sobre el que se está montando los sistemas del proyecto, puede ser a su vez un buen método de transferencia de archivos. HTTP proporciona un método genérico de comunicación de datos que permite la transferencia de archivos de gran capacidad. El mayor problema de este protocolo es que hay que usar programas de compresión para mandar jerarquías completas de los programas y después construirlos en él emisor.
* **GIT o SVN**: Es un control de versiones pensado para el desarrollo y mantenimiento de desarrollo software. Como tal, GIT o SVN no son programas de transferencia de archivos, sino que almacenan las modificaciones que se realizan sobre una jerarquía de archivos y luego pega los cachos. Esta tecnología se sincroniza de forma “automática” viendo si el código está en la última modificación y gestiona los cambios y la trasferencia de archivos nuevos de manera genérica y segura. La gran diferencia entre estas dos tecnologías, a parte del software es que SVN es un control de versiones centralizado, mientras que GIT es un control de versiones descentralizado.

Tras miras las siguientes soluciones el autor ha descrito la siguiente solución para el proyecto:

Para la sincronización de datos con los usuarios usaremos el control de versiones GIT. GIT es ahora un protocolo muy extendido en el mundo del desarrollo. Este nos facilita la comunicación entre el servidor y los diferentes proyectos de los usuarios. Cuando el usuario quiera realizar una modificación, el software de GIT lo identificara y lo modificara en el repositorio. Es por eso que creemos que usar GIT es la mejor solución.

Como explica el autor, el uso de GIT facilitara la comunicación y en especial las modificaciones del código entre el usuario y el repositorio. Su comunicación también es genérica y probada para la comunicación entre sistemas operativos.

## Ejecución de proyectos:

La ejecución de proyectos es la ejecución de los diversos proyectos que se encuentran en el proyecto **UniApi.** Para esta parte de la aplicación no existe muchas tecnologías que solucionen el problema de realizar ejecuciones de carácter genéricas. Es por eso que el autor ha decidido esto:

Como no existen tecnologías de ejecución de programas de carácter genérico o al menos no son conocidas por mí. Utilizaremos el poder de la librería de SO de java para mandar comandos de ejecución personalizados al sistema operativo que albergue el repositorio.

Esta solución descrita por el autor conlleva unos riesgos muy grandes en la consistencia del software y dando en confianza al usuario que desarrolle el programa toda la vitalidad del programa. Existen ciertas mejoras de esta solución para que no tengan tantos fallos de seguridad, pero estas soluciones podrían ser en si un proyecto y serán explicados como ampliación de este proyecto.

# Planificación temporal:

A continuación, vamos a explicar la planificación que hemos seguido para realizar este proyecto. Este proyecto se ha realizado siguiendo unas tareas como se van a describir a continuación. La siguiente tabla representa las fases que sean realizado para cada subsistema.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fase / Subsistema | Diseño | Implantación DAO | Implementación negocio | Implementación App Web |
| Entorno Ejecución | X | - | X | - |
| Ejecución | X | X | X | X |
| Grupos | X | X | X | X |
| Proyectos | X | X | X | X |
| UserLogin & Persona | X | X | X | X |

Tras saber las tareas que han sido realizadas en la aplicación vamos a describir su duración:

## Planificación de la duración entorno de ejecución:

|  |  |
| --- | --- |
| Nombre de la tarea | Duración |
| Diseño del entorno | 0S 0D 4H |
| Implementación negocio del Entorno de ejecución | 0S 4D 0H |

## Planificación de duración de grupos:

Tabla ‑ Planificacion de grupos

|  |  |
| --- | --- |
| Nombre de la tarea | Duración |
| Diseño de los grupos | 0S 0D 6H |
| Implementación del DAO de los grupos | 0S 0D 8H |
| Implementación del negocio de los grupos | 0S 0D 10H |
| Implementación de la App Web de los grupos | 0S 0D 4H |

## Planificación duración proyectos:

Tabla ‑ Planificación de proyectos

|  |  |
| --- | --- |
| Nombre de la tarea | Duración |
| Diseño de los proyectos | 0S 0D 2H |
| Implementación del DAO de los proyectos | 0S 0D 4H |
| Implementación del negocio de los proyectos | 0S 0D 6H |
| Implementación de la App Web de los proyectos | 0S 0D 4H |

## Planificación duración ejecuciones:

Tabla ‑Planifucacion de ejecuciones

|  |  |
| --- | --- |
| Nombre de la tarea | Duración |
| Diseño de los ejecuciones | 0S 0D 3H |
| Implementación del DAO de los ejecuciones | 0S 0D 4H |
| Implementación del negocio de los ejecuciones | 0S 0D 8H |
| Implementación de la App Web de los ejecuciones | 0S 0D 5H |

## Planificación duración UserLogin&Persona:

Tabla ‑ Planificación UserLogin&Persona

|  |  |
| --- | --- |
| Nombre de la tarea | Duración |
| Diseño de los UserLogin&Persona | 0S 0D 1H |
| Implementación del DAO de los UserLogin&Persona | 0S 0D 2H |
| Implementación del negocio de los UserLogin&Persona | 0S 0D 3H |
| Implementación de la App Web de los UserLogin&Persona | 0S 0D 6H |

Para la planificación del proyecto se ha tenido en cuenta que el usuario es un estudiante y que solo pude trabajar una media de 3 horas al día. Es decir que la jornada laboral será de unas tres horas. A continuación, se muestra el diagrama de grant:

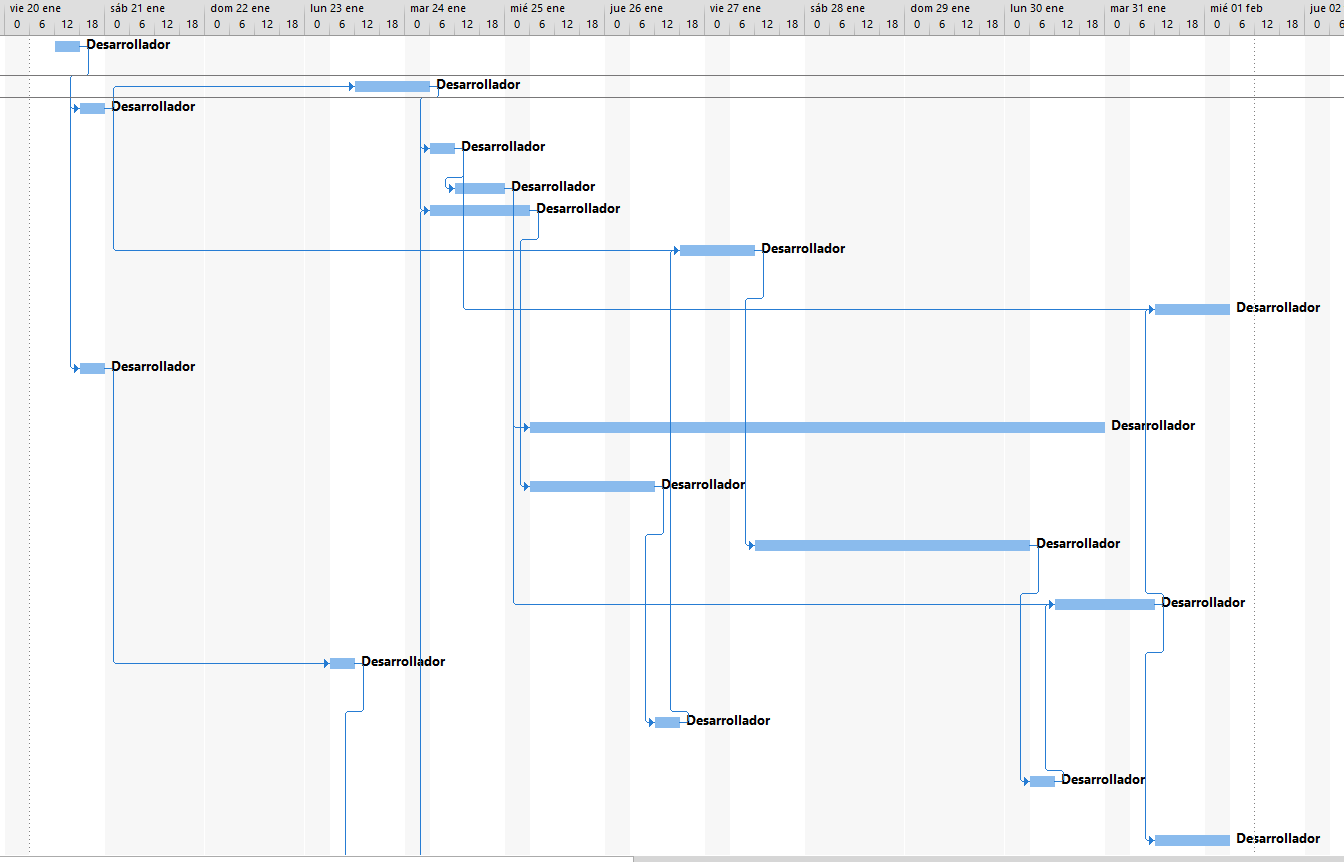


Ilustración ‑ Diagrama Grant

Y para mejor vista de las taras se exportará la tabla de las tareas con sus sucesores:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
| Diseño del UserLogin&Persona | 1 hora | vie 20/01/17 | vie 20/01/17 |  |
| Diseño de los grupos | 6 horas | lun 23/01/17 | mar 24/01/17 | 3 |
| Diseño de los proyectos | 2 horas | vie 20/01/17 | vie 20/01/17 | 1 |
| Diseño de los ejecuciones | 3 horas | mar 24/01/17 | mar 24/01/17 | 2 |
| Diseño del entorno | 4 horas | mar 24/01/17 | mar 24/01/17 | 4 |
| Implementación del DAO de los grupos | 8 horas | mar 24/01/17 | mié 25/01/17 | 2;18 |
| Implementación del DAO de los proyectos | 4 horas | jue 26/01/17 | vie 27/01/17 | 3;15 |
| Implementación del DAO de los ejecuciones | 4 horas | mar 31/01/17 | mié 01/02/17 | 4;13 |
| Implementación del DAO de los UserLogin&Persona | 2 horas | vie 20/01/17 | vie 20/01/17 | 1 |
| Implementación negocio del Entorno de ejecución | 10,67 días | mié 25/01/17 | lun 30/01/17 | 5 |
| Implementación del negocio de los grupos | 10 horas | mié 25/01/17 | jue 26/01/17 | 6 |
| Implementación del negocio de los proyectos | 6 horas | vie 27/01/17 | lun 30/01/17 | 7 |
| Implementación del negocio de los ejecuciones | 8 horas | lun 30/01/17 | mar 31/01/17 | 5;16 |
| Implementación del negocio de los UserLogin&Persona | 3 horas | lun 23/01/17 | lun 23/01/17 | 9 |
| Implementación de la App Web de los grupos | 4 horas | jue 26/01/17 | jue 26/01/17 | 11 |
| Implementación de la App Web de los proyectos | 4 horas | lun 30/01/17 | lun 30/01/17 | 12 |
| Implementación de la App Web de los ejecuciones | 5 horas | mar 31/01/17 | mié 01/02/17 | 13 |
| Implementación de la App Web de los UserLogin&Persona | 6 horas | lun 23/01/17 | mar 24/01/17 | 14 |

Ilustración ‑ Tabla de tareas

# Ampliaciones:

A continuación, se muestran todas las posibles ampliaciones que el autor cree que serían posibles para mejorar el proyecto actualmente creado. Estas ampliaciones abarcan soluciones todo tipo desde mejoras de seguridad, como ampliaciones de software.

* **Entornos de ejecución virtualizados**: El proyecto puede ser ampliado generando un sistema de máquinas virtuales que ejecuten los proyectos de cada usuario. Estas máquinas virtuales serán gestionadas por un software que utilizara el repositorio. Gracias a esta solución se mejoraría la tolerancia a fallos del repositorio. Si algún usuario, no programara bien el código, usara instrucciones con privilegios o simplemente intentara romper el sistema. Las máquinas virtuales harían de dique para que el problema no llegara a influenciar al repositorio.
* **Tienda online de ejecuciones**: Una ampliación plausible para el proyecto sería el de ampliar la aplicación web con un sistema de pagos y tiempos de ejecución. La aplicación web gestionaría quien podría y quien no podría ejecutar proyectos terminados o para negocio. Esta ampliación seria con el objetivo de crear una tienda web donde los usuarios podrían vender los resultados de sus códigos para empresas o usuarios.
* **Generar permisos individualizados para cada usuario**: Actualmente el proyecto contiene grupos, los cuales, contienen un conjunto de permisos que interaccionan con TODOS los miembros de ese grupo. La ampliación seria que los permisos fueran individualizados.
* **IDE virtual para la gestión**: El proyecto sincroniza los datos mediante GIT. Esto puede ampliarse dando la posibilidad de modificar los datos utilizando un IDE (Interfaz de desarrollo) online que centraría el desarrollo en la aplicación en vez de externalizarla. Siguiendo el ejemplo de Cloud9 , el proyecto podría evolucionar en un entorno de desarrollo genérico para toda la comunidad de la universidad.
* **Introducir parámetros de entrada individualizados**: Actualmente los parámetros de entrada son por defecto para todos los usuarios. Una ampliación para el proyecto sería el de generar entradas por defecto para cada usuario, pudiendo dar salidas personalizadas por defecto para cada usuario.

# Conclusiones:

Tras la finalización del proyecto **UniApi**. He querido plasmar ciertas ideas y mejoras que he ido recibiendo de forma retroactiva durante el desarrollo del proyecto. Empecé este proyecto con una simple idea, pero que tenía mucho potencial. La simplicidad de la idea, parecía asociada con el tamaño del proyecto a realizar. Es por eso que decidí llevarla a cabo. Algo que aprendí bastante rápido, es que la simplicidad de una idea no está relacionada directamente con una creación simple de esa idea.

El proyecto acabo siendo una aplicación de pequeño tamaño, tirando a mediano tamaño. Esto hizo que mi papel como desarrollador viera la importancia de generar grupos de desarrollo y la importancia de la imagen del jefe de proyecto o del arquitecto del sistema. Gracias a lo dicho. Un proyecto grande me hizo ver teorías e investigar en formas de realizar un proyecto con una escala significativa para un solo desarrollador de una manera eficiente. Esto, me llevo a conocer teorías como la inyección de dependencia y el uso de ciertos patrones de diseño que había estudiado en la carrera y que no había utilizado. Como, por ejemplo: singleton, abstract Factory, adapter…

Desde que empecé a trabajar en prácticas de empresa y conocí las primeras tecnologías JavaScript, en concreto, Ajax. Me obsesione con el poder realizar una aplicación con un alto poder computacional en el cliente. Esto es algo que me presiono a elegir la forma de la realización de la aplicación web orientada a esa idea. Gracias a esa presión acabe realizando la aplicación en una tecnología puntera y muy demandada por las empresas como es Angular. También el uso de tecnologías SAP, me empujó a aprender y entender un modelo web muy importante en estos tiempos como es la arquitectura REST y el uso en la práctica.

Cabe destacar que el punto que más me ha constado realizar de este proyecto, ha sido sin duda la documentación. Esta documentación esta aplicada a proyectos software con una visión monolítica. Algo que se ha estudiado que genera que se gaste más tiempo en el desarrollo de una documentación, que en el desarrollo del propio programa. Es por eso que en esta conclusión insto a que se actualice la realización de documentación a una visión más ágil y orientada al desarrollo del software.

# Bibliografía:

* **Docker - <https://www.docker.com/>**
* **TensorFlow -** [**https://www.tensorflow.org/**](https://www.tensorflow.org/)
* **AWS - <https://aws.amazon.com/es/>**
* **ABC - <http://www.abc.es/sociedad/20141108/abci-universidad-empresa-201411072128.html>**
* **¿Qué venden las universidades? -** [**https://es.linkedin.com/pulse/qu%C3%A9-venden-las-universidades-fernando-basto-correa**](https://es.linkedin.com/pulse/qu%C3%A9-venden-las-universidades-fernando-basto-correa)
* **MPA vs SPA -** [**http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications**](http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications)
* **GIT - <https://es.wikipedia.org/wiki/Git>**
* **FTP - <https://es.wikipedia.org/wiki/File_Transfer_Protocol>**
* **HTTP - <https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol>**