Universidad de Oviedo


# ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

## GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

### ÁREA DE INGENIERÍA TELEMÁTICA

### TRABAJO FIN DE GRADO Nº 17010523

*"Aplicación web para la gestión de un repositorio"*

**Autor:**
**Raúl García Fernández**

**Tutor:**
**Raquel Blanco Aguirre**

**Junio de 2017**

# Memoria

# Índice

# Ilustraciones

# Tablas

# 1. Objetivos y alcance

El objetivo de este trabajo fin de grado es la creación de una aplicación web que permita la gestión de un repositorio. Para ello, ésta será capaz de gestionar todas las funciones del repositorio: la comunicación del repositorio, la gestión de usuarios, la gestión de elementos y la navegación por sus elementos.

El trabajo fin de grado intenta ser un nexo de unión entre proyectos de distinta índole sin importar el lenguaje en el que están implementados. Para ello, este trabajo proporcionará una aplicación web que implementará un repositorio, donde los usuarios podrán compartir sus proyectos, ejecutarlos y comparar sus resultados, con el fin último de poder integrarlos con otros proyectos.

El objetivo secundario del proyecto es mejorar la integración entre proyectos de diferentes entornos, sin la necesidad de realizar estructuras intermedias como, por ejemplo, comunicación, gestión de usuarios o de seguridad. La realización de estructuras intermedias genera un sobrecoste de tiempo que podría ser empleado en otras facetas del desarrollo como la mejora o la optimización.

La solución planteada a esta necesidad por parte de esta memoria es la creación de un repositorio que almacene diversos proyectos de desarrollo, que puedan ser ejecutados por los usuarios del repositorio o cualquier aplicación web, y la creación de una aplicación web que facilite la gestión de los usuarios y el repositorio.

La aplicación, además, facilitará la compartición de proyectos entre usuarios mediante la utilización de un sistema de grupos. Los proyectos almacenados en el repositorio serán organizados en un sistema de jerarquía de contenedores de proyectos denominados “grupos”, los cuales pueden contener “subgrupos”. Cada uno de los grupos del repositorio tendrá asociado un conjunto de usuarios, los cuales podrán incluir un conjunto de proyectos dentro del mismo

Debido al complejo esfuerzo que conlleva comunicarse con un repositorio y entender su funcionamiento, 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.

Por último, se le ha proporcionado al proyecto un nombre para el desarrollo tanto de la documentación como para el software. A partir de ahora el proyecto se denominará “*UniApi*” o “*Proyecto*” indistintamente.

# 2. Estudio del mercado

A continuación, se van a describir distintas soluciones presentes en el mercado que pueden hacer frente de forma parcial a las necesidades expresadas en el punto anterior de objetivo y alcance, con el fin de generar soluciones e ideas generales para el desarrollo de la aplicación **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, se tienen un conjunto de modelos de negocio que han tenido éxito en sus campos y pueden ser de ayuda para respaldar la idea del proyecto.

## 2.1. Amazon web services (AWS)


*Ilustración 2-1 AWS logo*

Amazon Web Services es un servicio proveedor de tecnologías web bajo demanda. Máquinas virtuales, DNS, Active Directories, Almacenamiento en la nube… Todas estas tecnologías son instaladas, configuradas y proveídas en un solo click de ratón.

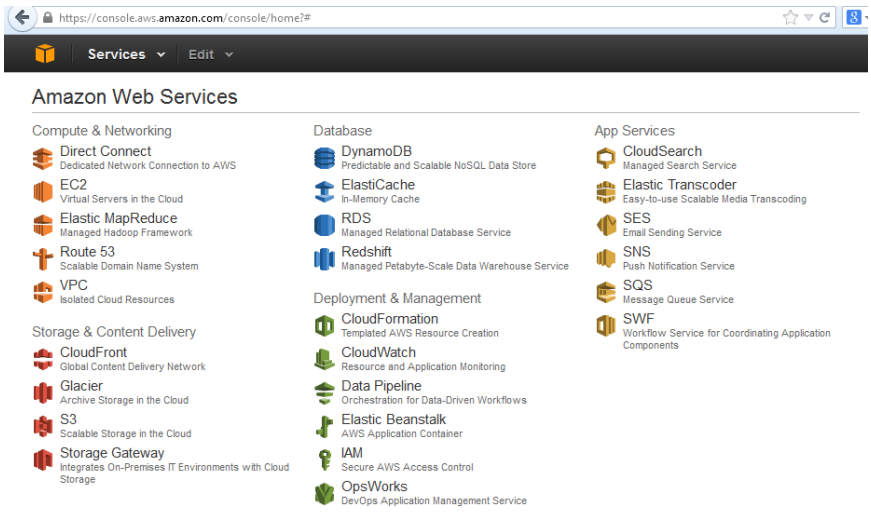


*Ilustración 2-2 AWS WebPage*

Todas estas tecnologías están en la nube, externa al usuario y sus condiciones. Los usuarios sólo pagan por lo que quieren en el tiempo que deseen. Tras eso el usuario no se tiene que preocupar de configuraciones o de dependencias de tecnologías. Tiene lo que quiere cuando lo necesita.

### 2.2. Cloud9


*Ilustración 2-3 Cloud9 logo*

Cloud9 es un IDE de desarrollo en la nube con una plataforma de despliegue en MV. 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.

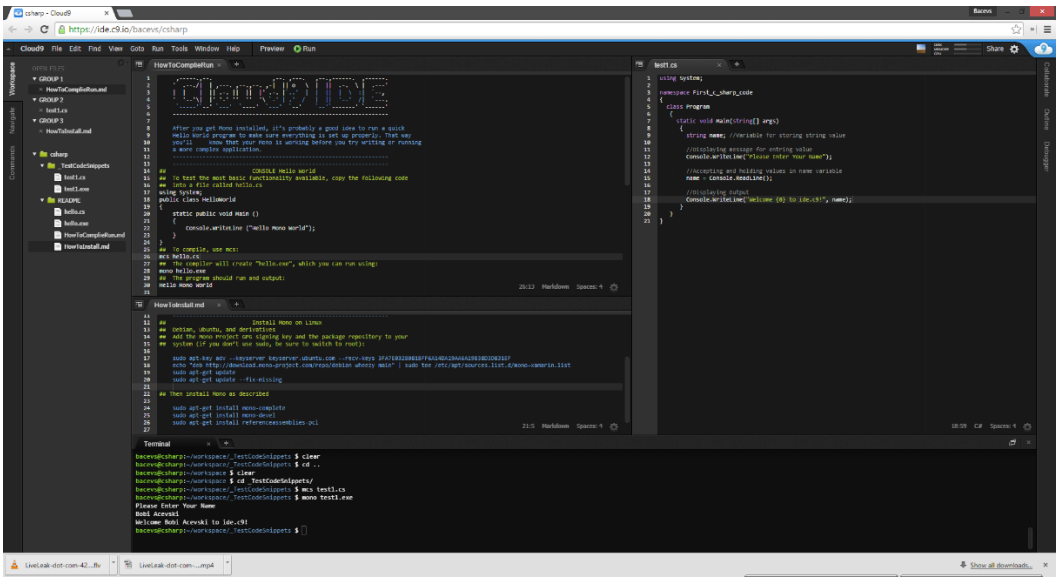

*Ilustración 2-4 Cloud9 IDE*

Cloud9 se excede del ámbito del proyecto que se va a realizar, el cual resuelve el objetivo de ejecución de diferentes lenguajes de programación de manera genérica, abstrayendo al usuario de las configuraciones previas y centrándolo en la programación.

### 2.3. Docker

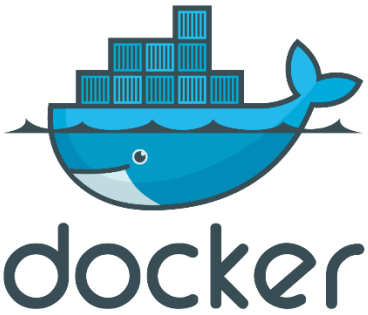


*Ilustración 2-5 Docker logo*

Docker es una tecnología de producción genérica. Los proyectos complejos de programación conllevan consigo un sinfín de configuraciones, dependencias y programas de interacción con nuestro proyecto. Estas últimas 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 preconfigurados y estables para la producción de software. Según los propios estudios de Docker, un desarrollo en tecnologías de pre-configuración genera una mejora en el desarrollo, modificación y producción del software.

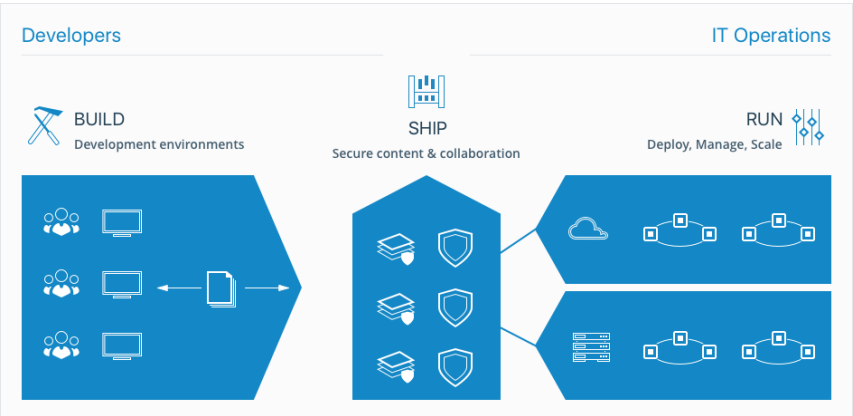


*Ilustración 2-6 Arquitectura Docker*

Docker ejecuta de manera segura, sin interferir en las demás ejecuciones, los diversos programas colocados en producción por los desarrolladores. Estos programas están monitorizados y gestionados en tiempo real.

### 2.4. TensorFlow


*Ilustración 2-7 Logo TensorFlow*

TensorFlow es una tecnología realizada por Google que facilita el desarrollo de modelos de predicciones, usando redes neuronales o Machine Learning. En TensorFlow, el usuario solo debe aplicar los conjuntos de datos y destacar las variables que tienen importancia para las predicciones.

Los usuarios que usan Python, y que a su vez tienen conocimientos en regresiones lineales y matemáticas, son capaces gracias a este framework de realizar ejecuciones de aprendizaje automático y redes neuronales en ordenadores ajenos, utilizando en su ejecución tecnologías concurrentes y paralelas sin ni si quiera darse cuenta. Esta tecnología reduce el tiempo de desarrollo de los proyectos que la utilicen, pudiendo usar ese tiempo para implementar el código mejor o buscar un modelo que se ajuste mejor a la predicción.

# 3. Enfoque de la aplicación

Como se explica en el punto objetivo y alcance, este proyecto tiene como objetivo la creación de una aplicación web que gestione un repositorio, el cual gestionará diversos proyectos que podrán estar desarrollados en distintos lenguajes de programación. El repositorio estará gestionado por una aplicación web que mostrará 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 de la aplicación **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** es un usuario con permisos especiales, pudiendo gestionar todas las entidades generadas por el repositorio y realizar modificaciones especiales que un usuario no puede realizar.

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 varios elementos. Un "proyecto" contendrá un **código fuente** que se ejecutará por orden del usuario, utilizando unas **entradas** definidas por el mismo 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 generará **ejecuciones** de "proyectos" en función de las órdenes de los usuarios o las aplicaciones, donde cada ejecución será diferente en función del tipo del "proyecto". Las **ejecuciones** serán unos espacios lógicos (almacenamiento, cpu, memoria…) ocupados por el repositorio para la ejecución de los proyectos en el sistema operativo donde se encuentra albergado el repositorio.

- **Concepto de grupo y permiso:**

Un **grupo** es una entidad lógica de almacenamiento de proyectos y otros grupos, que pueden ser accedidos por parte de diversos usuarios miembros del mismo. En un grupo imperarán unas restricciones dadas por el propietario, estas mismas son heredadas a los grupos hijos, conocidos como **subgrupos**. Las restricciones o **permisos**, como los conoceremos a partir de ahora, son inalterables salvo por el propietario y afectan a las acciones realizadas tanto en el grupo como en sus subgrupos y en los proyectos que contenga el grupo.

En resumen, **UniApi** se basa en una aplicación web que facilita la comunicación con un repositorio, al cual acceden diversos usuarios de distinta índole organizados en grupos creados por ellos. Estos usuarios **crean y gestionan proyectos de diversos tipos**, que pueden ser ejecutados por otros 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, siendo no volátil y pudiendo ser consultada por los usuarios y aplicaciones cuando ellos deseen.

# 4. Análisis de alternativas

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

## 4.1. Elección de arquitectura del sistema

Para hablar del sistema del proyecto **UniApi** hay que centrarse 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 se pueden 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 ellas 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, por lo que aquí se comentarán los más importantes desde el punto de vista del proyecto:

- El sistema estructurado en capas favorece la replicación y escalabilidad de servicios críticos o con grandes niveles de usabilidad, sin tener que replicar el servicio completo con todos sus subsistemas, como el subsistema monolítico.
- El uso de sistemas monolíticos favorece el desarrollo ágil en aplicaciones pequeñas y medianas, mediante el desarrollo en funcionalidades, mientras que en el sistema estructurado, el desarrollo es más arduo debido a la generación de comunicación entre capas.
- La independencia de capas en el sistema estructurado genera una tolerancia a fallos más elevada que en un sistema monolítico, donde un fallo en las capas inferiores puede afectar a todo el sistema.

Es por eso que, 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 sí. 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.

### 4.2. Repositorio de información

El repositorio de información tiene como objetivo gestionar datos y almacenarlos de forma permanente a largo plazo. Dado que existen diversos gestores de almacenamiento, se analizará la información que guardará en el repositorio sus particularidades para se elegir una forma debemos centrarnos en qué va a guardar nuestro repositorio y qué particularidades tiene cada tipo de dato:

- **Datos personales**: Son de poco peso, no son muy importantes para el funcionamiento de la aplicación y no es importante que los datos estén siempre disponibles.

- **Grupos**: Son de poco peso, pero 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 o en un tiempo razonable para la aplicación.

- **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, presentan una gran cantidad de relaciones con otras entidades y son datos importantes que es necesario tener disponibles para la aplicación. Por otro lado, están los códigos fuente, y los ficheros del proyecto; estos datos no suelen ser consultados por la aplicación web, 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, se observa que hay dos grandes tipos: existen ciertos tipos de datos que necesitan una disponibilidad elevada y una velocidad de consulta cuanto más rápida mejor, mientras que otros que no necesitan dicha velocidad pero sí una consistencia de los datos y una alta disponibilidad.

En base a lo anterior, se ha estudiado las alternativas para su almacenamiento, las cuales se describen a continuación.

- Para los códigos de los proyectos, la primera opción para su almacenamiento es utilizar una base de datos relacional. Estas bases de datos se rigen por ser muy consistentes y de gran seguridad, algo que se necesita para no corromper los datos. La segunda opción es usar la jerarquía del SO que se esté usando; esta opción contiene todos los factores que se busca y facilita una simplicidad que no puede conseguir la Base de datos relacional.

  Por tanto, la alternativa a seleccionar para el guardado de proyectos de usuarios en **UniApi**, es el uso del sistema de archivos de los SO. Este sistema nos garantiza una consistencia de datos y lo más favorecedor, un entorno amigable con el entorno de la ejecución.

- En cuanto al resto de tipos de datos del sistema, sus cualidades principales son su poca necesidad de consistencia y el gran número de relaciones entre ellos. Para almacenar estos tipos de datos suele utilizarse una base de datos, la cual puede ser relacional o NoSQL (clave-valor, basada en documentos, familia de columnas u orientada a grafos).

  La característica más importante de los datos a guardar son las relaciones entre ellos. Es por eso que se va a utilizar la base de datos orientada a grafos Neo4j, en la cual las relaciones entre las entidades son manejadas de forma más eficiente que en las bases de datos relaciones. Neo4j almacena los datos en forma de nodos y relaciones entre ellos, y tanto los nodos como las relaciones pueden contener propiedades.


*Ilustración 4-1 Logo Neo4j*

### 4.3. Negocio de la aplicación

El negocio de la aplicación es la parte de lógica del proyecto, es decir, las funcionalidades del repositorio. Para desarrollar el negocio de la aplicación se puede utilizar cualquier lenguaje de programación, siendo la opción más conveniente el uso de un lenguaje de programación orientados a objetos, como podría ser Python, C#, C++ o Java. Debido a la poca experiencia del programador en el uso de diferentes lenguajes para la construcción de modelos de negocio, se ha utilizado el lenguaje de programación Java y un conjunto de tecnologías para facilitar la generación de esta capa. Las tecnologías empleadas son las siguientes:

- **Spring**: Es un "framework" que se basa en la inyección de dependencia y JEE2 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 para utilizarlo en aplicaciones java desarrolladas por nosotros mismos.


*Ilustración 4-2 Logo Maven*


*Ilustración 4-3 Logo Spring*


*Ilustración 4-4 Logo Java*

### 4.4. Aplicación web

La aplicación web tiene como objetivo el comunicarse con el usuario a través del protocolo HTTP y los navegadores web. Estas aplicaciones están divididas en dos lados: un lado cliente y un lado servidor. Cada lado puede contener unas funcionalidades para la ejecución de páginas web.

Actualmente, el desarrollo de aplicaciones web es muy flexible en la distribución de funcionalidades entre los dos lados y es necesario establecer sus responsabilidades antes de desarrollar el proyecto. Hay dos tipos de metodologías para desarrollar una aplicación web:

- **Multiple Page Aplication (MPA)**: Esta visión es la clásica. Todo el peso está dentro del servidor, el cual, gestiona todos los eventos y genera las vistas para el usuario. Para realizar esta visión se utiliza el patrón **MVC (Modelo-Vista-Controlador)**. Existen múltiples frameworks para llevar a cabo esta solución, siendo JSF y Spring dos de las más conocidas.


*Ilustración 4-5 Logo JSF*


*Ilustración 4-6 Logo Spring MVC*

JSF y Spring son dos grandes framework para la construcción de MPA. 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. Al realizar todas las gestiones en el servidor, éste provee un control de seguridad elevado y una gestión de las vistas más cercanas al negocio del repositorio.

- **Single Page Aplication (SPA)**: Una visión nueva creada en torno al 2010. Esta tecnología promueve que todo el peso de la aplicación se encuentre en el cliente, donde se gestionan todos los eventos y se realizan las vistas para el usuario. Esta visión tiende a transformar a los servidores en meros repositorios de información. Dicha solución alivia el poder computacional de los servidores, obligando a que el poder computacional lo acarree el navegador del cliente, generando que nuestros servicios sean más rápidos y puedan atender a más gente. En contraposición, esta visión genera un coste elevado de transferencia de archivos ya que todo el código es comunicado al cliente, el cual, podrá observarlo, identificado y posiblemente copiado o modificado.

Para la realización de la parte servidor, está en auge el uso de sistemas REST. Estos servidores son servidores web pero especializados en la arquitectura REST. Existen múltiples tecnologías de servidores REST, cada una focalizada en un lenguaje de programación. Entre los múltiples servidores REST se encuentran ExpressJS, Spring Rest y Django. ExpressJS es un servidor WEB MVC implentado en JavaScript muy flexible, ya que se puede adaptar como servidor REST para lanzar aplicaciones web SAP (Single Page Aplication), Spring REST es un servidor REST muy potente implementado en Java y Django es mellizo de ExpressJS, pero implementado en python.

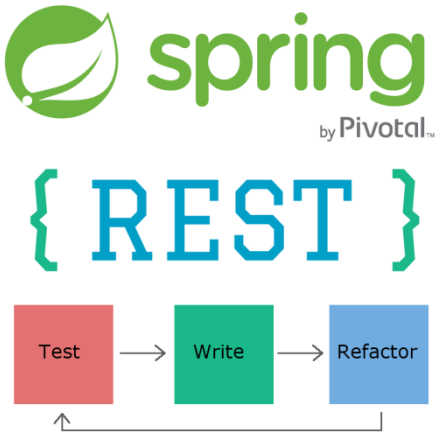


*Ilustración 4-7 Logo Spring REST*


*Ilustración 4-9 Logo Django*


*Ilustración 4-8 Logo ExpressJS*

Para la realización de la parte cliente se puede escoger entre un sinfín de tecnologias (debido al auge de estos tiempos), entre las que cabe destacar AngularJS, React y Vue. Todas estas tecnologías están implementadas en JavaScript y son ejecutadas por el navegador en el cliente. Cada una tiene una visión diferente de la implementación de una SPA. Angular es un framework desarrollado por Google que se basa en la actualización del modelo habitual de MVC (Mode-View-Controller), pero orientado a páginas SPA**.** React es un framework desarrollado por Facebook siguiendo un modelo de componentes. Por último, Vue es un MVVM (Model-View-View-Model), un framework que evoluciona el modelo MVC (Mode-View-Controller) e integra las cualidades de los dos anteriores.

*Ilustración 4-11 Logo Angular.js*

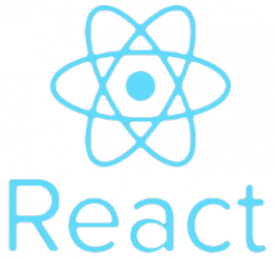


*Ilustración 4-10 Logo React.js*


*Ilustración 4-12 Logo Vue.js*

Para la realización 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 subdivisión en la aplicación web entre la APP-Web y el negocio. Por ello, se utilizará un servidor web básico basado en Node.js y Express.js, debido a su simplicidad, que inyectará el framework **Angular.js**, realizando la separación APP-Repositorio mediante SAP. **El repositorio** solo

tendrá la tarea única y exclusiva de realizar su propio negocio y comunicarse con las demás capas. Es por eso que éste empleará un **servidor REST Spring** que está implementado en el mismo lenguaje que el propio repositorio y que será comunicado por la aplicación Angular.js para realizar las gestiones del negocio. Esta elección acarreará una serie de pros y contras.

Como ventajas cabe destacar las siguientes:

- **Dos núcleos de ejecución**: App Web (lado servidor) y repositorio. Esto genera que una independencia, en caso de fallo, pueda ser asumida, así como la replicación exitosa de estos elementos.
- **Peso computacional**: La parte servidor ejecutará 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.

Como desventaja, indicar que parte del código de negocio es ubicado en el lado del cliente de manera transparente, lo que supone un fallo de seguridad.

### 4.5. Transferencia de archivos

Existen múltiples tecnologías para la transferencia de archivos. Éstas promueven la transferencia de datos genérica para todos los dispositivos, independientemente tanto del código como del sistema operativo.

**FTP (File transfer Protocol)**: Éste protocolo es un sistema de transferencia de archivos entre sistemas conectados entre sí. Funciona utilizando un conjunto de órdenes 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 están 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 es que hay que usar programas de compresión para gestionar el envío y recepción de archivos.
- **GIT o SVN**: Es un control de versiones pensado para el desarrollo y mantenimiento de desarrollo software. GIT o SVN no son programas de transferencia de archivos como tal, sino que almacenan las modificaciones que se realizan sobre una jerarquía de archivos y luego une las partes. 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 lo es descentralizado.

Se usará GIT para la trasferencia de archivos ya que 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 identificará y lo modificará en el repositorio.

# 5. Planificación temporal:

A continuación, se indica la planificación a seguir para realizar este proyecto. La siguiente tabla representa las fases que se han realizado para cada fase:

*Tabla 5-1 Tabla planificación*

| Nombre de la tarea | Duración |
|---|---|
| **Estudios y análisis previos** | 11 Días |
| **Requisitos de usuario** | 5 Días |
| **Análisis del sistema** | 8 Días |
| **Diseño del sistema** | 6 Días |
| **Construcción del sistema** | 61 Días |
| **Pruebas** | 6 Días |
| **Manual del usuario** | 4 Días |
| **Conclusiones** | 2 Días |

A partir del listado de las tareas y su relación, se genera el diagrama de Gantt que describe las tareas en función de una línea temporal.

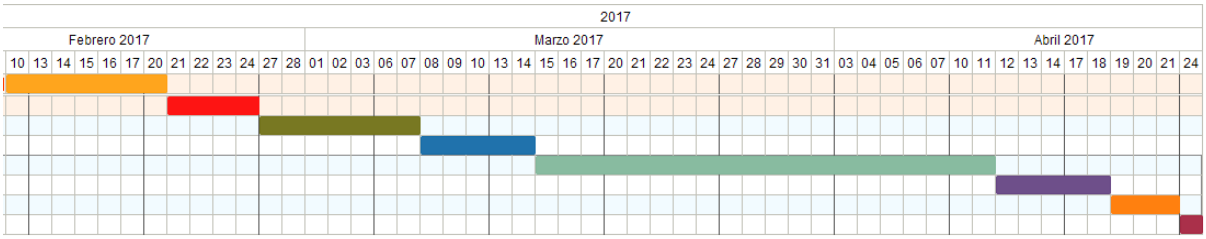


*Ilustración 5-1 Diagrama de Grant*

# 6. Ampliaciones:

A continuación, se muestran posibles ampliaciones para mejorar el proyecto actualmente creado:

- **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 utilizará el repositorio (Docker). Gracias a esta solución, se mejoraría la tolerancia a fallos del repositorio. Si algún usuario no programara bien el código, usará instrucciones con privilegios o simplemente intentará romper el sistema. Las máquinas virtuales harían de dique para que el problema no llegara a influenciar al repositorio y los demás proyectos.

- **Tienda online de ejecuciones**: Un incremento plausible para el proyecto sería la ampliación de la aplicación web con un sistema de pagos y gestión de tiempos de ejecución. La aplicación web gestionaría quién podría y quién no ejecutar proyectos terminados. Esta ampliación tendría como objetivo el crear una tienda web, donde los usuarios podrían vender los resultados de sus códigos para empresas u otros usuarios que necesiten estos servicios.

- **Generar permisos individualizados para cada usuario**: Actualmente el proyecto contiene grupos que poseen un conjunto de permisos que interaccionan con todos los miembros de ese grupo indistintamente. La ampliación de esta funcionalidad estaría orientada a que los permisos fueran individualizados para cada usuario.

- **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, utilizando el programa Cloud9 o usando un prototipo de él. El prototipo podría evolucionar en un entorno de desarrollo genérico para toda la comunidad.

- **Introducir parámetros de entrada individualizados**: Actualmente, los parámetros de entrada son por defecto para todos los usuarios. Una posible ampliación consistiría en generar entradas y salidas personalizadas por defecto para cada usuario.

# 7. Conclusiones:

Tras la finalización del proyecto **UniApi**, se han querido plasmar ciertas ideas y mejoras que se han ido recibiendo de forma retroactiva durante el desarrollo del proyecto.

El proyecto acabó siendo una aplicación de pequeño o mediano tamaño. Esto hizo que el desarrollador del proyecto pudiera ver la importancia de generar grupos de desarrollo y la imagen de un jefe de proyecto o del arquitecto del sistema. Un proyecto como éste obliga a ver teorías e investigar formas de realizar proyectos con una escala significativa de una manera eficiente. Esto lleva a conocer teorías como la inyección de dependencia y el uso de ciertos patrones de diseño como, por ejemplo: Singleton, Abstract Factory, Adapter…

Para terminar, desde el inicio en prácticas de empresa y el conocimiento de las primeras tecnologías JavaScript, en concreto Ajax, me interesé en poder realizar una aplicación con un alto poder computacional en el cliente. Esto es algo que me influenció a la hora de elegir la forma de realizar la aplicación web de este proyecto. Este interés me llevó a realizar la aplicación sobre una tecnología muy demandada por las empresas, como es Angular.js. 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.

# 8. Bibliografía


- **Docker (Mayo 2017) - https://www.docker.com/**
- **TensorFlow (Mayo 2017) - https://www.tensorflow.org/**
- **AWS (Mayo 2017) - https://aws.amazon.com/es/**
- **ABC (Mayo 2017) - http://www.abc.es/sociedad/20141108/abci-universidad-empresa-201411072128.html**
- **¿Qué venden las universidades? (Mayo 2017) - https://es.linkedin.com/pulse/qu%C3%A9-venden-las-universidades-fernando-basto-correa**
- **MPA vs SPA (Mayo 2017) - http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications**
- **GIT (Mayo 2017) - https://es.wikipedia.org/wiki/Git**
- **FTP (Mayo 2017) - https://es.wikipedia.org/wiki/File_Transfer_Protocol**
- **HTTP (Mayo 2017) - https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol**