**ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.**

**GRADO EN INGENIERÍA INFORMATICA EN**

**TECNOLOGIAS DE LA INFORMACIÓN**

**ÁREA DE INGENIERIA TELEMATICA**

**TRABAJO FIN DE GRADO Nº 11111111111**

***“****Aplicación web para la gestión de un repositorio****”***

|  |  |
| --- | --- |
| **Autor:** | **Tutor:** |
| **Raúl García Fernández** | **Raquel Blanco Aguirre** |

**Junio de 2017**

# Índice

[Índice 1](#_Toc483309042)

[Ilustraciones 3](#_Toc483309043)

[Tablas 4](#_Toc483309044)

[1. Objetivos y alcance: 5](#_Toc483309045)

[2. Estudio del mercado: 6](#_Toc483309046)

[2.1. Amazon web services (AWS): 6](#_Toc483309047)

[2.2. Cloud9: 7](#_Toc483309048)

[2.3. Docker: 8](#_Toc483309049)

[2.4. TensorFlow: 9](#_Toc483309050)

[3. Enfoque de la aplicación: 10](#_Toc483309051)

[3.1. Concepto de usuario y administrador: 10](#_Toc483309052)

[3.2. Concepto de proyecto y ejecución: 10](#_Toc483309053)

[3.3. Concepto de grupo y permiso: 11](#_Toc483309054)

[3.4. Definición formal del *Proyecto UniApi*: 11](#_Toc483309055)

[4. Análisis de alternativas: 12](#_Toc483309056)

[4.1. Elección de arquitectura del sistema: 12](#_Toc483309057)

[4.2. Repositorio de información: 13](#_Toc483309058)

[4.3. Negocio de la aplicación: 14](#_Toc483309059)

[4.4. Aplicación web: 15](#_Toc483309060)

[4.5. Transferencia de archivos: 19](#_Toc483309061)

[4.6. Ejecución de proyectos: 20](#_Toc483309062)

[5. Planificación temporal: 21](#_Toc483309063)

[5.1. Planificación de la duración entorno de ejecución: 21](#_Toc483309064)

[5.2. Planificación de duración de grupos: 22](#_Toc483309065)

[5.3. Planificación duración proyectos: 22](#_Toc483309066)

[5.4. Planificación duración ejecuciones: 22](#_Toc483309067)

[5.5. Planificación duración UserLogin&Persona: 23](#_Toc483309068)

[6. Ampliaciones: 25](#_Toc483309069)

[7. Conclusiones: 26](#_Toc483309070)

[8. Bibliografía: 27](#_Toc483309071)

# 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)

# Objetivos y alcance:

El objetivo del proyecto es la creación de una aplicación web que permita la gestión de un repositorio, para ello, esta es 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 departamentos.

La necesidad de realización de este proyecto viene imperada por el uso recurrente de programas de carácter científico en ámbitos informáticos que no facilitan su integración. Los diversos científicos o estudiantes apoyándose en las tecnologías de desarrollo de alta abstracción (Como R, Python o Octave) desarrollan diversos programas no enfocados al ámbito de las tecnologías web.

La diferencia de abstracción en los lenguajes usados por la comunidad científica, provoca un proceso de traducción y desarrollo para la integración de programas científicos en aplicaciones web implementadas por los desarrolladores. La traducción o implementación de programas de un lenguaje a otro suelen generar diversos problemas de ejecución en el software que, normalmente, antes no estaban. Éstos problemas suelen encontrarse debido al poco conocimiento del desarrollador TIC en los proyectos de carácter científico que está traduciendo o implementando.

La solución planteada a esta necesidad por parte de esta memoria es la creación de un repositorio que almacene diversos proyectos de carácter educativo o científico y que puedan ser ejecutados por los usuarios del repositorio o cualquier aplicación web. Los proyectos son ejecutados de forma opaca, no importando su implementación para el usuario final, lo cual es objetivo del repositorio.

Los proyectos almacenados en el repositorio serán organizados en un sistema de jerarquía de contenedores de proyectos denominados “grupos”. A estos, pertenecerán un conjunto de usuarios los cuales podrán incluir un conjunto de proyectos, que serán gestionados por el grupo que los alberga. Los grupos, a su vez, podrán contener un conjunto de grupos gestionados por el grupo contenedor denominados “subgrupos”.

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, facilitando el uso de esta aplicación a usuarios con conocimientos básicos en el uso de aplicaciones web.

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.

# 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 generales 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, 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.

## Amazon web services (AWS):



Ilustración ‑ 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.

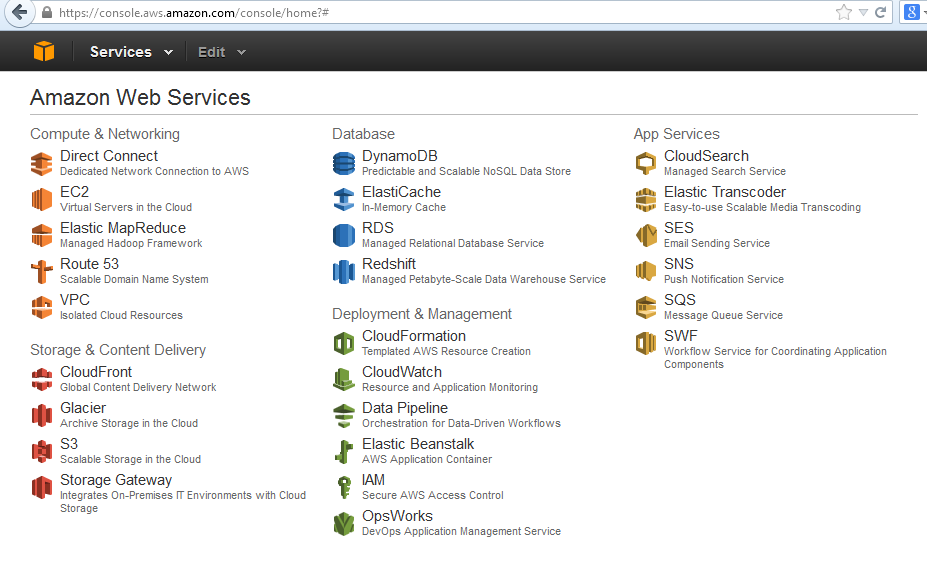


Ilustración ‑ AWS WebPage

Todas estas tecnologías están en la nube, externa al usuario y sus condiciones. Los usuarios solo 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.

## Cloud9:



Ilustración ‑ Cloud9 logo

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

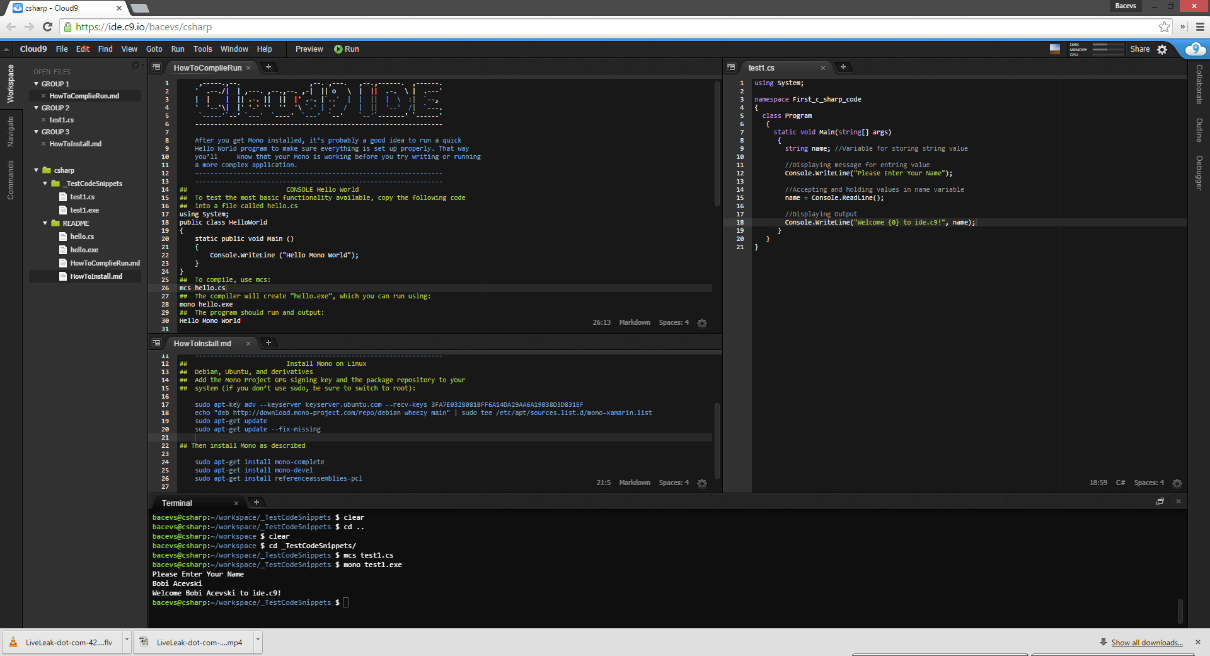


Ilustración ‑ Cloud9 IDE

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.

## Docker:

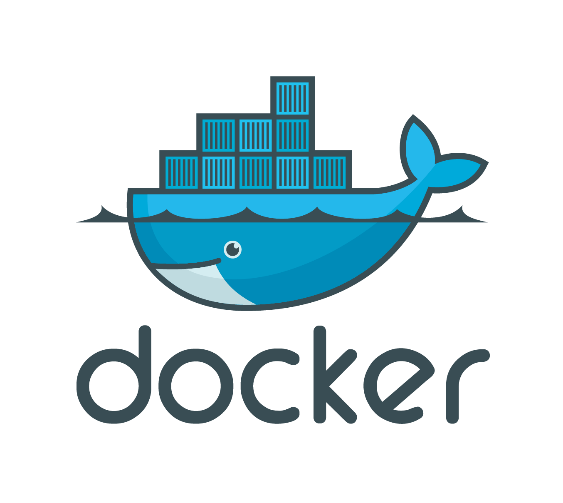


Figure ‑ Docker logo

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 e 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.

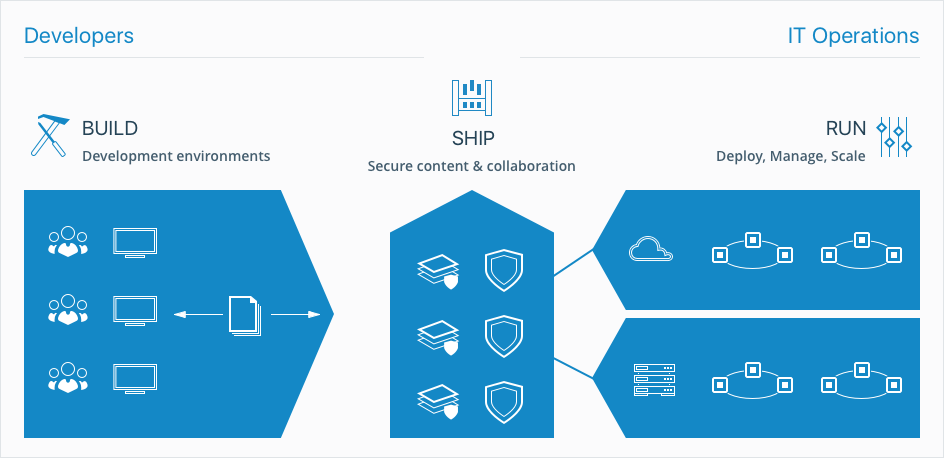


Figure ‑ Aquitectura docker

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.

## 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. En TensorFlow, el investigador solo debe aplicar los conjuntos de datos y destacar las variables que tienen importancia para las predicciones.

Los investigadores que usan Python y que a su vez tienen conocimientos en regresiones lineales y matemáticas, son capaces gracias a este framework de realizar aprendizajes automáticos a maquinas con las cuales, si se programara de manera natural, conllevaría realizar unos estudios mínimos de Ingeniería Informática además de estudios de Postgrado.

# Enfoque de la aplicación:

En este punto describiremos **UniApi** de manera concisa y nos centraremos en la utilidad del proyecto, es decir, en que va a hacer **UniApi** en vez de en cómo se va a implementar.

Como se explica en el punto objetivo y alcance, este proyecto tiene como objetivo la creación de una aplicación web que gestione un repositorio. Este mismo no va a ser un repositorio normal, sino que gestionará diversos proyectos que podrán contener 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 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 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 varias cosas. Un “proyecto” contendrá un código 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 este grupo imperará unas restricciones dadas por el propietario, estas restricciones 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.

## Definición formal del *Proyecto UniApi*:

Al final **UniApi** se basa en una aplicación web que facilita la comunicación con un repositorio, el cual, contiene diversos usuarios de distinta índole organizados en grupos creados por ellos. Estos usuarios **crean y gestionan proyectos de diversos tipos**, los cuales 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.

# Análisis de alternativas:

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

## 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 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 así que desarrollaremos los más importantes desde el punto de vista de la aplicación:

* 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 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.

## 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…). Nuestro repositorio utilizara datos persistencia a largo plazo para almacenar la información y así no perder la información ante un corte inesperado del servicio.

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 y no es importante que los datos estén disponibles siempre.
* **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 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, son grandes cantidades de relaciones entre otras entidades y datos importantes que es necesario tener disponibles para aplicaciones web. Por otro lado, están los códigos y los ficheros del proyecto. Estos datos no suelen ser consultados por la aplicación web, entonces 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: Hay ciertos tipos de datos que 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.

Se han analizado dos opciones:

* La primera, es donde ubicar los proyectos de los usuarios. Se podría utilizar una base de datos relacional, estas Base de datos 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 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. Entonces 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.
* Para los demás datos del sistema sus cualidades principales son su poca necesidad de consistencia y el gran número de relaciones entre ellos. Toda información de una aplicación suele almacenarse en una base de datos. La elección de base de datos en la actualidad está dividida en si queremos una base datos Relacional o una No relacional. Ambas son buenas para esta solución, aunque una de las bases de datos no relacional se ajusta a la 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 datos orientada a grafos (Neo4j) donde el motor de esta base de datos son las relaciones entre los nodos. Las Base de datos orientadas a grafos son BD no relacionales teniendo las cualidades típicas de una NO-SQL como una velocidad de acceso elevado y una disponibilidad elevada.



## 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 ha utilizado el lenguaje de programación Java y un conjunto de tecnologías para facilitar la generación de esta capa.

* **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.



Figure ‑ Spring Logo

Figure ‑ Maven logo



Figure ‑ Java Logo

## 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 debe usar la tecnología **MVC (Modelo-Vista-Controlador)**. Existen muchas tecnologías para realizar esta solución mostraremos dos de las más conocidas.



Figure ‑ 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. Al realizar todas las gestiones en el servidor, el servidor provee un control de seguridad elevado y una gestión de las vistas más cercanas al negocio del repositorio.

* **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 para el usuario. 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, generando que nuestros servicios sean más rápidos y puedan atender a más gente. En contra posición, 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 servidores de apoyo**, 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.

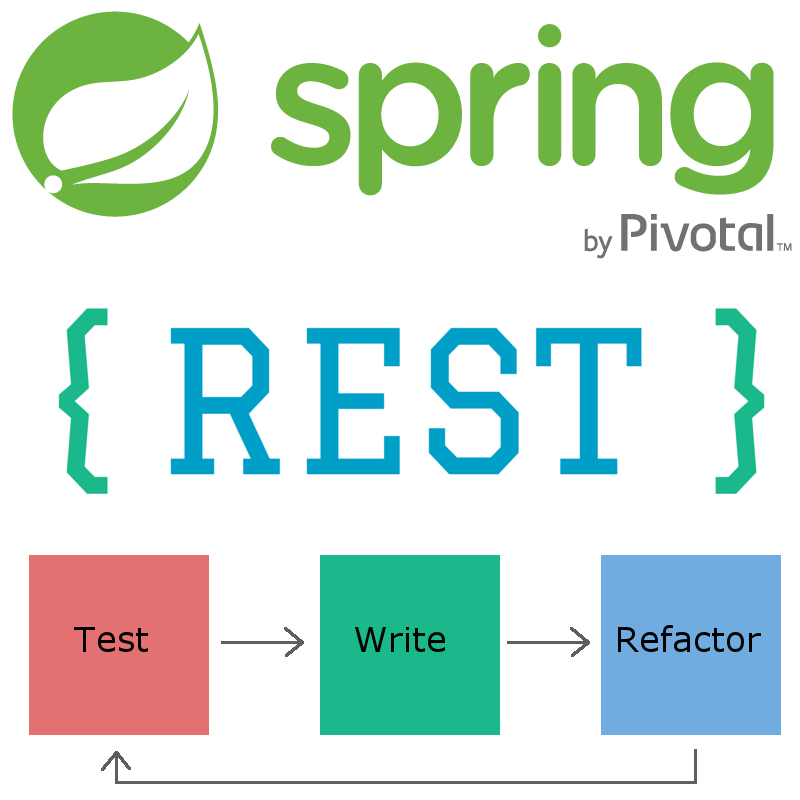


Figure ‑ Spring Rest Logo



Figure ‑ django logo

Figure ‑ Express.js logo

Los servidores REST existen para cualquier tipo de lenguaje en especial vamos a destacar tres 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 el cual es un servidor REST muy potente implementado en Java y Django mellizo de expressJS pero implementado en python.

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:

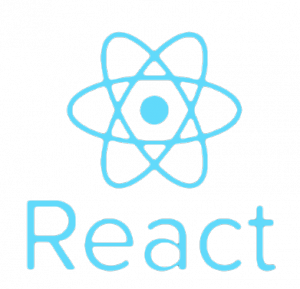


Figure ‑ React.js Logo

Figure ‑ Angular.js Logo



Figure ‑ 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 SPA. Angular es un framework desarrollado por google que se basa en la actualización del modelo habitual de MVC, pero orientado a paginas SPA. 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.

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 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 el repositorio empleará 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. Esta elección acarreará una serie de pros y contras:

* **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:

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

* **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 gestionar el envío y recepción de archivos.
* **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.

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

# Planificación temporal:

A continuación, vamos a explicar la planificación que ha seguido para realizar este proyecto. La siguiente tabla representa las fases que sean realizado para cada fase.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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:

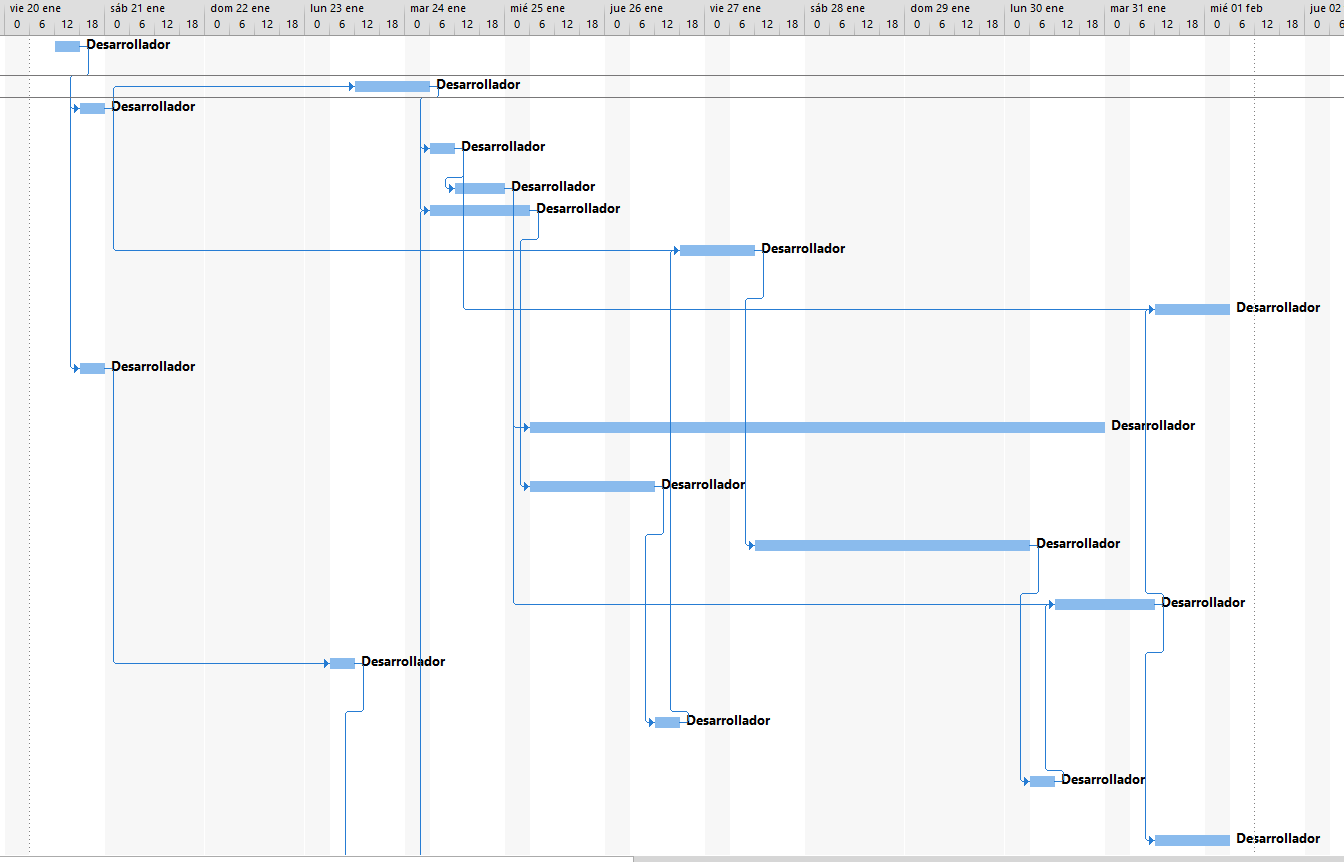


Ilustración ‑ Diagrama Grant

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 serían posibles para mejorar el proyecto actualmente creado. Estas ampliaciones abarcan soluciones de todo tipo desde mejoras de seguridad, como ampliaciones de software o inclusión de software de terceros.

* **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 (Docker). 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 y los demás proyectos.
* **Tienda online de ejecuciones**: Una ampliación plausible para el proyecto sería el de ampliar la aplicación web con un sistema de pagos y gestión de tiempos de ejecución. La aplicación web gestionaría quien podría y quien no podría ejecutar proyectos terminados. 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 que necesiten estos servicios.
* **Generar permisos individualizados para cada usuario**: Actualmente el proyecto contiene grupos que contienen un conjunto de permisos que interaccionan con TODOS los miembros de ese grupo indistintamente. La ampliación de esta funcionalidad seria 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 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**. Se ha querido plasmar ciertas ideas y mejoras que se ha ido recibiendo de forma retroactiva durante el desarrollo del proyecto. El proyecto se empezó con una simple idea, donde su simplicidad parecía asociada con el tamaño del proyecto a realizar y es debido eso que se decidió llevarlo a cabo. Algo que aprendió bastante rápido es que la simplicidad de una idea no está relacionada directamente con una creación de esa idea.

El proyecto acabo siendo una aplicación de pequeño tamaño, tirando a mediano tamaño. Esto hizo que el desarrollador del proyecto pudiera ver la importancia de generar grupos de desarrollo y la importancia de la imagen de un jefe de proyecto o del arquitecto del sistema. Un proyecto grande como este obliga a ver teorías e investigar en 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…

Cabe destacar, que el punto que más ha difícil de este proyecto ha sido sin duda la documentación. Esta última esta aplicada a proyectos software con una visión monolítica y es 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 se insta en este proyecto a que se actualice la realización de documentación a una visión más ágil y orientada al desarrollo del software.

Y para terminar: Personalmente 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 de este proyecto orientada a la computación en el cliente. Gracias a ese interés acabe realizando la aplicación en una tecnología puntera y 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.

# Bibliografía:

* **Docker -** [**https://www.docker.com/**](https://www.docker.com/)
* **TensorFlow -** [**https://www.tensorflow.org/**](https://www.tensorflow.org/)
* **AWS -** [**https://aws.amazon.com/es/**](https://aws.amazon.com/es/)
* **ABC -** [**http://www.abc.es/sociedad/20141108/abci-universidad-empresa-201411072128.html**](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**](https://es.wikipedia.org/wiki/Git)
* **FTP -** [**https://es.wikipedia.org/wiki/File\_Transfer\_Protocol**](https://es.wikipedia.org/wiki/File_Transfer_Protocol)
* **HTTP -** [**https://es.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol**](https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol)