Reunión TFG – 13/06/2018

# Horario Patricio

# Documento de propuesta de trabajo

## Desarrollo de software basado en microservicios: un caso de estudio para evaluar sus ventajas e inconvenientes.

Las arquitecturas basadas en microservicios son una tendencia actual que emerge asociada a conceptos clave como: integración continua, desarrollo centrado en el dominio del problema, o despliegue en contenedores. En estas arquitecturas diferentes funcionalidades se encapsulan en servicios pequeños y autónomos que cooperan entre ellos. En términos de diseño, principios como el de Responsabilidad Única son más fáciles de conseguir, y los desafíos de organización del código pueden abordarse de formas más diversas por la baja granularidad de la arquitectura.

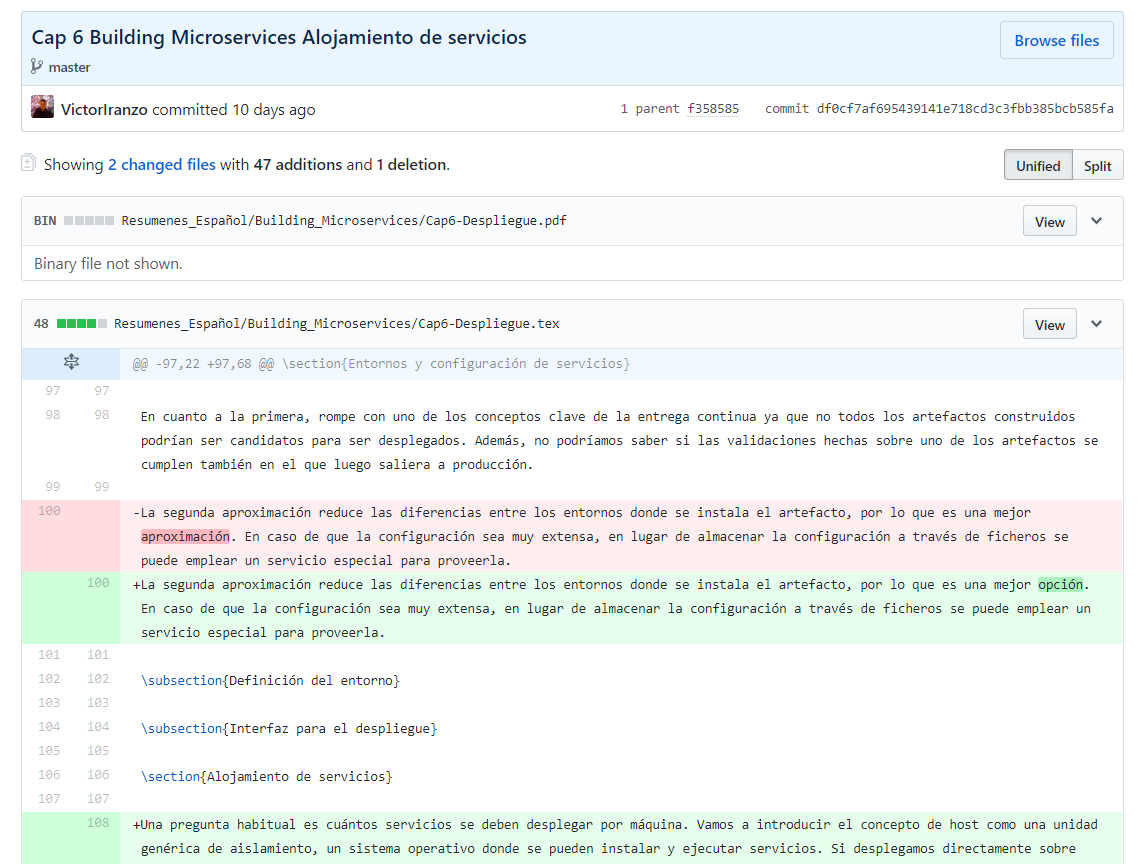
El objetivo de este TFG es validar con un caso de estudio las ventajas de una arquitectura basada en microservicios frente a una arquitectura tradicional. En este TFG se hará una revisión bibliográfica de la tecnología de microservicios para ilustrar su estado actual. Posteriormente, se implementará una misma aplicación siguiendo dos arquitecturas diferentes: una basada en microservicios y otra por capas. De la comparación del desarrollo y posible mantenimiento de ambas arquitecturas se extraerán conclusiones asociadas a sus ventajas y desventajas.

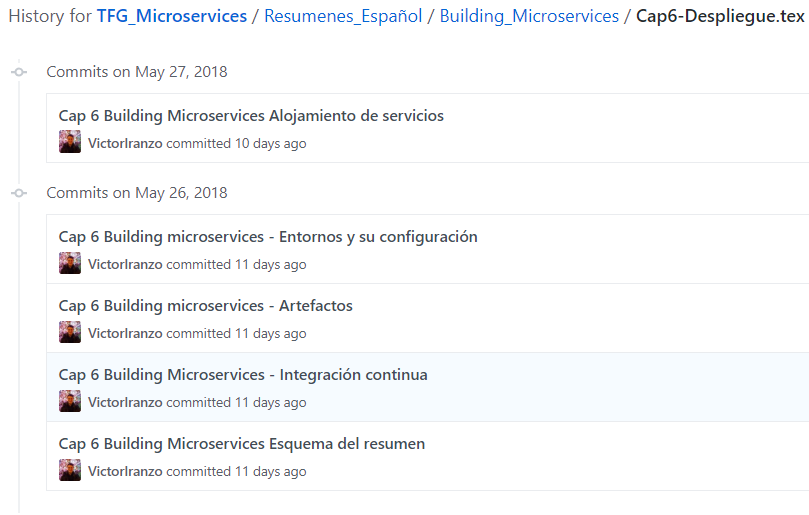
* ¿Cómo se va a comparar el desarrollo y mantenimiento de ambas soluciones?
* ¿Cómo se va a gestionar la lógica común que comparten ambas soluciones respecto al proceso de desarrollo y las medidas que se quieran obtener de este?
* ¿Cuál es la bibliografía más relevante? ¿Qué autores hay?

# Repositorio de código

<https://github.com/VictorIranzo/TFG_Microservices>

* Control de versiones: en cualquier momento se puede volver a una versión anterior.
* Control de cambios: cada commit viene identificado por un nombre que explica el cambio realizado. En el commit de los ficheros .tex se puede ver los párrafos añadidos y modificados. Además, se puede consultar el histórico de cambios y el “blame” de un fichero.
* Comentarios, ramas, issues, projects, pull requests, tags…





Contenido

[Documento de propuesta de trabajo 1](#_Toc516675832)

[Desarrollo de software basado en microservicios: un caso de estudio para evaluar sus ventajas e inconvenientes. 1](#_Toc516675833)

[Repositorio de código 1](#_Toc516675834)

[Índice del trabajo 5](#_Toc516675835)

[1. Introducción 5](#_Toc516675836)

[1.1. ¿Qué son los microservicios? 5](#_Toc516675837)

[1.2. Motivación 5](#_Toc516675838)

[1.3. Objetivos 5](#_Toc516675839)

[1.4. Metodología 5](#_Toc516675840)

[1.5. Estructura 6](#_Toc516675841)

[1.6. Convenciones 6](#_Toc516675842)

[2. Estado del arte 7](#_Toc516675843)

[2.1. Beneficios de los microservicios 7](#_Toc516675844)

[2.2. Comparación con otras arquitecturas 7](#_Toc516675845)

[2.3. Técnicas para el diseño de microservicios 7](#_Toc516675846)

[2.4. Integración de microservicios 7](#_Toc516675847)

[2.5. Despliegue de microservicios 7](#_Toc516675848)

[2.6. Pruebas de microservicios 7](#_Toc516675849)

[2.7. Escalabilidad de servicios 8](#_Toc516675850)

[2.8. Crítica al estado del arte 8](#_Toc516675851)

[2.9. Propuesta 8](#_Toc516675852)

[3. Análisis del problema 8](#_Toc516675853)

[3.1. Casos de uso del sistema a desarrollar 8](#_Toc516675854)

[3.2. Especificación de requisitos 9](#_Toc516675855)

[3.3. Modelo de Dominio 9](#_Toc516675856)

[3.4. Análisis de seguridad 9](#_Toc516675857)

[3.5. Análisis de riesgos 9](#_Toc516675858)

[3.6. Identificación y análisis de soluciones posibles 9](#_Toc516675859)

[3.7. Plan de Trabajo 10](#_Toc516675860)

[3.8. Presupuesto 11](#_Toc516675861)

[4. Diseño de la solución 11](#_Toc516675862)

[4.1. Arquitectura del sistema 11](#_Toc516675863)

[4.2. Diseño Detallado 11](#_Toc516675864)

[4.3. Tecnología Utilizada 11](#_Toc516675865)

[5. Desarrollo de la solución propuesta 12](#_Toc516675866)

[6. Implantación 12](#_Toc516675867)

[7. Pruebas 12](#_Toc516675868)

[8. Conclusiones 12](#_Toc516675869)

[9. Relación del trabajo desarrollado con los estudios cursados 13](#_Toc516675870)

[10. Trabajos futuros 13](#_Toc516675871)

[11. Referencias 13](#_Toc516675872)

[12. Anexos 14](#_Toc516675873)

[13. Glosario 14](#_Toc516675874)

[Bibliografía 15](#_Toc516675875)

# Índice del trabajo

## Introducción

### ¿Qué son los microservicios?

Revisión bibliográfica de qué son los microservicios.

### Motivación

#### Motivación profesional (breve – media página)

* Explicar que es una tendencia arquitectónica.
* Poner ejemplos de empresas que empleen microservicios (Netflix) o que hayan migrado hacia esta arquitectura por sus ventajas (en Building Microservices comentan alguna como Gilt).
  + TODO: Buscar un artículo o noticia.
* Explicar ideas de otros TFGs posibles: un TFG que solo fuera desarrollar una solución basada en microservicios, sin compararla con una arquitectura monolítica. **TFGs hechos en la escuela.**
* Comentar estudios que se hayan realizado para evaluar sus ventajas (si hay alguno) y las diferencias con el nuestro.

#### Motivación personal

* Curiosidad por investigar en nuevas tecnologías: Docker, Kubernetes, .NET Core, Javascript, ¿Xamarin?.
* Curiosidad por investigar en patrones arquitectónicos.
* ~~Experiencia profesional en ADD Informática. Alineación entre lo que he aprendido y mi experiencia. No mencionarla.~~

### Objetivos

¿Alguna hipótesis inicial que se pueda deducir del estado del arte?

#### Objetivos principales

1. Validar con un caso de estudio las ventajas de una arquitectura basada en microservicios frente a una arquitectura monolítica.
   * 1. Desarrollo: proceso de desarrollo.
     2. Mantenimiento: plantear modificaciones que se podrían llevar a cabo.
     3. Requisitos no funcionales: no se pueden hacer comparaciones con la monolítica. Plantear situaciones en las que la monolítica falla: escalabilidad, tolerancia a falllos.

#### Objetivos secundarios

1. Evaluar el estado del arte de las arquitecturas basadas en microservicios.
2. Comparar las ventajas e inconvenientes que ofrecen diferentes tecnologías para el desarrollo de microservicios.
3. Estudiar las principales herramientas para el desarrollo de microservicios.

### Metodología

1. Crear un modelo de calidad que nos permita comparar una solución basada en microservicios y una arquitectura monolítica.
2. Analizar el sistema a desarrollar, extrayendo los requisitos en forma de Pruebas de Aceptación.
3. Diseñar e implementar ambas soluciones:
   1. Si el sistema tiene interfaz de usuario, será la misma para ambas soluciones. Así se separará el trabajo de frontend y backend. En consecuencia, la interfaz que ofrecen ambas soluciones deberá ser la misma.
   2. Si se van a tomar métricas sobre el proceso de desarrollo (tiempo de desarrollo, errores detectados…), la lógica que comparten ambas soluciones se debe desarrollar como un proyecto independiente, que luego se reestructura en una arquitectura u otra. En otras palabras, si se trabaja con TUNE-UP, se deberán crear UTs específicas para implementar la lógica de, por ejemplo, la generación de informes y luego mover esa lógica a cada solución. También se debería hacer otro repositorio de código.
4. Aplicar el modelo de calidad a ambas soluciones.
5. Obtener conclusiones de los resultados y la experiencia del desarrollo. También se pueden aplicar herramientas como SonarQube para medir la calidad de las soluciones, la deuda técnica…

### Estructura

Comentar la estructura del trabajo de manera redactada y sin entrar en profundidad de todos los epígrafes.

### Convenciones

• Las palabras extranjeras se remarcarán en cursiva.

• Se entrecomillarán las citas textuales externas a la obra.

## Estado del arte

¿Sigue demasiado la estructura de “Building Microservices”?

### 2.1. Beneficios de los microservicios

Como punto de partida se pueden comentar los beneficios que comenta Newman en Building Microservices: despliegue independiente, heterogeneidad tecnológica, independencia de equipos dentro de una organización…

Comparar con otros autores.

* TODO: Buscar más fuentes para comparar.

### 2.2. Comparación con otras arquitecturas

Los beneficios que se han citado en la sección anterior son respecto a una arquitectura monolítica. En esta sección se compararán con respecto a otras arquitecturas.

* TODO: Buscar bibliografía.

### 2.3. Técnicas para el diseño de microservicios

* Comentar las técnicas que explica Newman para la extracción de microservicios basada en Domain Driven Development. Explicar relación con DDD.
* TODO: Revisar técnicas que presentan otros autores.

### 2.4. Integración de microservicios

* Comentar diferentes tecnologías para la integración: HTTP, SOAP,…
* Comentar comunicaciones síncronas y asíncronas, patrones petición-respuesta y basado en eventos, y patrón coreógrafo y orquestador.
* Restricciones de integridad.

NOTA: Todas estas ideas provienen del libro de Newman de Building Microservices.

### 2.5. Despliegue de microservicios

* Comparación entre tecnologías de virtualización y de contenedores.
* Relación con integración continua y entrega continua.
* Comentar tecnologías y herramientas: Docker y Kubernetes. (¿AWS?)

### 2.6. Pruebas de microservicios

* Comentar clasificación de Newman de las pruebas: unitarias, de servicio y de extremo a extremo. También comentar el balance en el número de pruebas que se debe hacer de cada uno.
* TODO: Hay mucha discusión sobre cómo clasificar las pruebas. Comentar la clasificación de otros autores.
* TODO: Buscar información sobre cómo se deben implementar las pruebas de extremo a extremo para que no dependan de una infraestructura (levantar servicios en el mismo test, uso de fakes…). Hacerlo independiente de la plataforma.

### 2.7. Escalabilidad de servicios

* Gestión de fallos e incremento de la demanda.
* Aproximación reactiva y predictiva.
* Monitorización de servicios.
* Teorema de CAP.
* Herramientas para la gestión de la escalabilidad.

### 2.8. Crítica al estado del arte

* Trabajos relacionados de la ETSINF: en una primera búsqueda no se han encontrado muchos resultados.
  + TODO: Buscar otros trabajos y artículos de profesores. Quizás no tan relacionado con microservicios sino con la evaluación de arquitecturas. ¿Algún profesor con el qué acotar la búsqueda?

### 2.9. Propuesta

* Explicar que no se ha encontrado ningún trabajo donde se sigue una metodología similar a esta.

## Análisis del problema

### Casos de uso del sistema a desarrollar

Administrador

* CRUD sobre empleado.
* Generar informe de “Empleados y roles”.
* Cualquier acción del resto de actores.

Empleado Dpto. Económico

* CRUD sobre clientes.
* CRUD sobre facturas.
* CRUD sobre productos: añadir productos al catálogo, modificar el stock.
* Aprobar un pedido.
* Generar informe de “Factura de un cliente”.
* Enviar notificación de pago.

Empleado Dpto. Atención al cliente

* Añadir notificación en una incidencia para un cliente o empleado.
* Cambiar el estado de un ticket.
* Generar informe de “Gestión de una incidencia”.

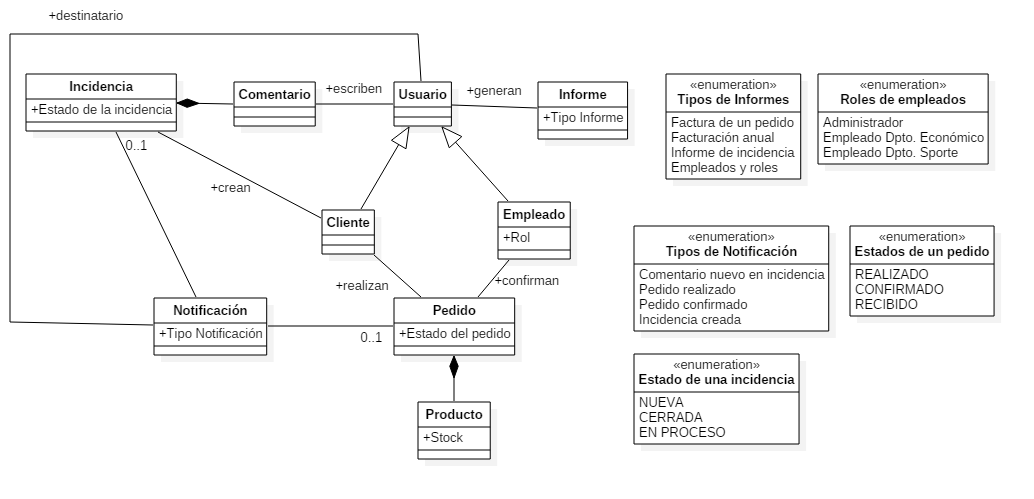
Cliente

* CRUD sobre pedidos: realizar un pedido, consultar su estado, cancelarlo en caso de que no se haya aprobado todavía y actualizar su estado confirmando la entrega.
* CRUD sobre incidencias: crear ticket, poner comentarios en el ticket, cambiar su estado…

### Especificación de requisitos

* Emplear pruebas de aceptación de TUNE-UP. Usar TUNE-UP nos puede servir para medir el proceso de Desarrollo de ambas soluciones.
* Usar plantillas para definir los requisitos.

### Modelo de Dominio



### Análisis de seguridad

* Aplicar seguridad a nivel de datos.
* Aplicar seguridad a nivel de funcionalidad.

### Análisis de riesgos

* TODO: Revisar.
* En la solución orientada a microservicios, cualquier integración entre microservicios puede suponer un riesgo.

### Identificación y análisis de soluciones posibles

* TODO: Revisar <https://github.com/dotnet-architecture/eShopOnContainers> .
* **Solución basada en microservicios**: una primera aproximación de los componentes a desarrollar es la siguiente:
* **Informes**: un motor que almacena las plantillas de los informes que se pueden generar y simplemente combina esta con los datos que recibe en un formato independiente. En .NET Framework.
* **Notificaciones**: un servicio asíncrono que permite generar diferentes notificaciones (enviar correo electrónico, bandeja de notificaciones). La notificación consta de un mensaje (se puede extraer de un recurso) y un destinatario. Empleado por varios servicios.
* **Incidencias**: permite transitar una incidencia por diferentes estados, añadir comentarios…
* **Seguridad**: contiene las acciones que puede realizar cada usuario y a los datos a los que puede acceder.
* **Catálogo**: control del stock de los productos.
* **Usuarios**: operaciones CRUD del administrador para crear clientes y empleados (y gestionar sus roles).
* **Pedidos**: permite transitar un pedido por diferentes estados.
* **Facturación**: un servicio que realiza una gran cantidad de computo para calcular el balance de la empresa. Se puede llamar de forma asíncrona. Podría emplearse otra tecnología para mejorar su rendimiento (es una ventaja de los microservicios el poder emplear diferentes tecnologías).
* **¿Configuración?**: contiene donde se encuentra desplegado cada microservicio.
* **¿Servicio que consuma una API de terceros?**

Cada microservicio se haría en una solución, ¿repositorio?, contenedor diferente.

* Los artefactos para referenciarse unos a otros serán paquetes NuGet: tiene que existir un Proxy que se ejecuta en el servicio invocador.
  + 5 capas: Contratos, Proxy, Servicios, Aplicación, Persistencia.
* Las invocaciones se hacen a través de HTTP.
  + 4 capas: ¿Contratos? (nadie los emplea para invocar a un servicio porque se hace a través de HTTP directamente), Servicios, Aplicación, Persistencia.
* **Solución UI compartida**: si no se emplea un Proxy, la UI tiene que invocar a través de HTTP. Si se emplea un proxy, los contracts de ambas aproximaciones deben de ser iguales. ¿Podría ofrecer una configuración para modificar en runtime la solución que emplea?
* **¿Solución monolítica?**: se emplea un diseño 5 capas donde cada una de las capas de todos los servicios en un nivel se juntan en el mismo proyecto.

Una solución y un repositorio. ¿Se despliega con un servicio enorme o junto con la UI? En teoría, la segunda es más monolítica.

### Plan de Trabajo

* Si se emplea TUNE-UP, podrían ser las diferentes UTs que se crean, con su estimación en esfuerzo (puntos, horas…), su prioridad, su relación entre ellas…
* A partir de esto se podría generar un Diagrama de Gantt.
* Se deben anotar también desviaciones.

### Presupuesto

* Material HW y SW necesario para poder desarrollar el TFG.
* Las horas-hombre empleadas.

## Diseño de la solución

* TODO: Buscar bibliografía sobre la diferencia entre diseño y arquitectura. (abstracción)

### Arquitectura del sistema

* Diagrama de componentes de ambas soluciones.
* Contenido de sección 3.6.
* Esquema de BD de ambas soluciones.

### Diseño Detallado

* Clases e interacción entre clases.
  + TODO: Buscar herramienta para generar las interacciones entre clases automáticamente.
* Directorios de las soluciones.

### Tecnología Utilizada

* .NET Core y Framework.
* Docker.
* ¿Kubernetes?
* Xamarin.
* Entity Framework.
* ¿SQL? ¿SQLLite? ¿NoSQL? (¿soporte en EF?)
* ¿JavaScript?
* ¿Firebase?
* Se podría hacer una discusión sobre las ventajas y desventajas de usar en todos los microservicios el mismo stack tecnológico.

## Desarrollo de la solución propuesta

* Problemas y dificultades.
* Decisiones tomadas.
* Poco código (partes complejas, novedosas, críticas). Se puede emplear apéndices.
* Librerías de terceros utilizadas.

## Implantación

En este apartado se presenta la etapa de implantación de la solución. En esta fase lleva el desarrollo realizado a explotación. Para ello, puede ser necesaria la instalación y/o puesta en marcha (o en producción) del sistema desarrollado para probarlo y obtener resultados que nos ofrece la solución final.

* Puesta en producción de ambas soluciones.
* Aplicación del modelo de calidad.

## Pruebas

* Si se emplea TUNE-UP, testeo manual a partir de las PAs. Hacer balance del tiempo invertido y los defectos detectados. Problema: ¿quién pasa las pruebas?
  + ¿Programador = Tester?
* Pruebas unitarias, de servicio y de extremo a extremo implementadas. Si se ha hecho una Build para ejecutar las pruebas del microservicio, explicarla.
* ¿Pruebas de carga y de recursos?

## Conclusiones

Esta sección es obligatoria y debe redactarse en forma sencilla, exhibiendo concordancia con las hipótesis aceptadas u objetivos formulados inicialmente. Todo lo que aparezca en esta sección debe de haber aparecido también al principio de la memoria en el apartado Objetivos de la Introducción. En esta sección se debe poner de manifiesto claramente si se han alcanzado todos los objetivos planteados y si se han desarrollado éstos satisfactoriamente, proponiendo ideas, soluciones o incluso nuevos objetivos surgidos a raíz de los anteriores.

En la memoria debe quedar claro el dominio que tiene el alumno de diferentes tecnologías y de cómo las ha empleado para poder dar solución al problema planteado en la introducción de la memoria. Debe poner de manifiesto que ha sido capaz de integrar los conocimientos de diferentes disciplinas o áreas para resolver el problema planteado haciendo un uso adecuado de la tecnología de vanguardia. Posiblemente esas tecnologías no se hayan visto durante la carrera o incluso se hayan visto de manera insuficiente para el contenido del TFG. Es necesario explicitar qué se ha aprendido que no se sabía o si ha sido muy difícil el desarrollo.

Hay que evitar cometer el error muy frecuente de repetir en la conclusión los resultados. Otro error es ir más allá de los objetivos planteados. El alumno tiene aquí la posibilidad de reflexionar sobre el trabajo realizado:

• Qué problemas se han encontrado y cómo los ha solucionado.

• Qué errores se han cometido y cómo se podrían haber evitado.

• Qué ha aprendido el alumno con este proyecto tanto profesional como personalmente. En este punto hay que indicar también qué nuevos conocimientos y tecnologías ha hecho falta que aprenda el alumno para poder realizarlo y el dominio alcanzado con ello.

## Relación del trabajo desarrollado con los estudios cursados

* Competencias transversales.
* Relación con asignaturas.

**Obligatorio**. Es conveniente e interesante pararse a pensar y realizar un ejercicio de introspección que incluya un análisis de la relación de los estudios realizados con el trabajo desarrollado por el alumno.

Este punto es una justificación de que el contenido del TFG es conforme a los estudios cursados. El objetivo del trabajo es poner en marcha y coordinar conocimientos recibidos a lo largo de los estudios con el fin de demostrar que se saben dar soluciones a problemas reales en el mundo laboral. Tener en cuenta que:

1. Un trabajo que demuestre el uso de muy pocas tecnologías, o muy específicas, demuestra que:

1. Es un trabajo pobre desde el punto de vista de la demostración del dominio tecnológico amplio del alumno.

2. Tal vez el alumno haya realizado el trabajo de aquello que se ha enterado y no sabe mucho más.

2. Un trabajo muy amplio que emplee muchas tecnologías puede ser agotador y acabar siendo muy largo o, caso de que haya tiempos máximos de entrega, acabe poco lucido por el coste de aprendizaje de las tecnologías no enseñadas en clase.

Por último, destacar que sería adecuado comentar qué competencias transversales y en qué grado se han requerido y puesto en práctica para la elaboración del TFG.

## Trabajos futuros

* Explorar otras tecnologías para evaluar su integración: microservicios en Java, Python…
* Comparar la arquitectura basada en microservicios con otras empleando el mismo modelo de calidad.
* Caminos a NO seguir: incrementar la complejidad del sistema desarrollado porque para hacer mediciones no parece relevante.

## Referencias

Las referencias permiten al lector comprobar varias cosas acerca del trabajo:

1. La verificación de las fuentes originales en las que se basa el trabajo.

2. Que el alumno no ha reinventado la rueda y que ha construido sobre trabajo previamente desarrollado.

3. Es un indicador directo del grado de profundidad del trabajo realizado.

En este apartado se valora la abundancia de las fuentes de información, la relevancia y la adecuación al tema trabajado en el TFG.

El nivel de detalle al que se puede llegar referenciando desde el texto principal puede ser extenuante. Como regla de sentido común, hay que pensar en el lector o miembro del tribunal como un profesional de la informática con amplios conocimientos comunes pero que no es especialista en la materia de la que versa el TFG. Por lo tanto, no es necesario documentar ni referenciar permanentemente a la Wikipedia2 cada vez que se emplea una palabra por primera

2http://www.wikipedia.org/vez ni para aclarar conceptos básicos de primero de carrera de informática. Sí que hará falta referenciar a artículos serios o científicos cuando haya que introducir determinados algoritmos, mejoras o determinadas tecnologías específicas propias de la temática del TFG.

Actualmente todo está en internet por lo que la tentación de convertir una bibliografía en una colección de enlaces a páginas web es muy alta. A continuación, se presentan algunas recomendaciones prácticas a tener en cuenta:

● No abusar de referencias *bibliográficas* a páginas web. Dejar para esta sección, al final del todo, todas las referencias a libros, artículos y revistas, libros o artículos serios de revistas científicas, sean electrónicos o no.

● Cualquier referencia a una página web del fabricante o del producto fabricado (herramienta empleada por el alumno o programas que sirven de referencia al TFG a desarrollar), se resuelve como una nota al pie de página. Normalmente se referencia la primera vez que aparecen.

● No puede aparecer ninguna referencia bibliográfica en la bibliografía que no esté referenciada al menos una vez en alguna parte del texto.

● No puede aparecer ninguna referencia en el texto a una referencia bibliográfica inexistente en la bibliografía.

● Puede haber varias referencias en el texto a la misma fuente bibliográfica.

Se deben documentar todas las fuentes bibliográficas utilizadas en un formato de citación estándar. Existen muchísimas normativas para establecer referencias bibliográficas. Se recomienda la norma internacional ISO 690-2010 para recoger las referencias bibliográficas [Bezos14]. Algunas bibliotecas han realizado un resumen organizado de la norma como la de la Universidad de Alicante3 que, a modo de prontuario, resuelven las principales dudas que puedan aparecer durante la redacción de referencias bibliográficas.

3http://rua.ua.es/dspace/bitstream/10045/33984/1/Norma\_ISO\_Doctorado.pdf

**Referencias a pie de página**

**Recomendable**. Cualquier referencia a una página web del fabricante o del producto fabricado (herramienta empleada por el alumno o programas que sirven de referencia al TFG a desarrollar), se resuelve como una nota al pie de página. Normalmente se referencia la primera vez que aparecen.

Si al buscar información por internet aparecen varias referencias que finalmente dirigen a una web del fabricante o del producto, hay que referenciar a la web final, a la original y a la cual se refiere en el artículo. Visitarla y verificar que lo que dice el primer autor es cierto y añadir lo que se considere por el autor.

## Anexos

**Opcional**. Normalmente se reserva para esta parte todos aquellos aspectos del TFG que no sea necesario leer en la memoria para entender su contenido de forma completa. Suele recoger detalles técnicos, manual de usuario, trozos de código, ampliaciones de una determinada parte, etc. Esta parte va dedicada a aquellos lectores que buscan algo muy específico o que tienen especial predilección por un tema concreto y quieren profundizar un poco más en ese aspecto por cuestiones personales.

También se puede colocar material suplementario que se ha desarrollado pero que se descartó pero que presenta interés. Es una forma de poner en valor la cantidad de trabajo realizado, aunque no todo haya ido en la dirección final del TFG.

## Glosario

* Término, definición, página donde aparece, siglas…
* ¿Al principio o al final?

# Bibliografía

* **Building Microservices, Sam Newman.**
* **Microservices AntiPatterns and Pitfalls, Mark Richards.**
* **Microservicios .NET: Arquitectura para Aplicaciones .NET Contenerizadas, Cesar de la Torre et al.**
* Advanced Microservices, Thomas Hunter II.
* Artículos de Martin Fowler.
  + <https://www.martinfowler.com/articles/microservices.html>
* <http://microservices.io/>
* Documentación oficial de herramientas.
* <http://www.javiergarzas.com/2015/06/microservicios.html>
* ISO SQUARE (si se sigue un Modelo de Calidad).