Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de plataforma web de realización de SCAMPI con CMMI DEV

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Jiménez Fabra

Tutor: María Llanos Cuenca González

2014/2015

Agradecimientos

Quisiera agradecer en primer lugar a María Llanos Cuenca González, mi tutora y profesora durante la carrera al igual que otros tantos profesores que han estado impartiendo clase y enseñándonos para que nos convirtamos en los profesionales en que ahora nos convertimos, y sin los cuales este proyecto nunca se habría llevado a cabo.

También a mis amigos, en especial a Javi Jiménez y Albert Duato que no dejaron de animarme a terminar. A mi familia, que ha soportado la distancia y el tiempo para que consiguiera finalizar y a mis compañeros de trabajo, que me han apoyado en todo.

A todos les agradezco el incansable ánimo que me han dado para poder terminar el proyecto. Gracias.

Resumen

El proyecto se enmarca dentro de gestión de Tecnologías de la Información. Más concretamente en la evaluación de procesos para el desarrollo de sistemas software mediante CMMI. CMMI puede llevar a cabo una evaluación de estos procesos para detectar la realización o no de buenas prácticas descritas en los modelos de CMMI. CMMI tiene tres modelos diferentes aunque el proyecto se centrará en un principio en CMMI-DEV (CMMI para el desarrollo de Software). Dentro de CMMI hay una forma de evaluar los procesos de desarrollo de una empresa mediante la realización de un modelo de encuestas SCAMPI. Estas encuestas intentan responder a las preguntas de qué se está haciendo, cómo se está haciendo (verificar que es así) y verificar si cumple con las mejores prácticas descritas en los modelos CMMI. El objetivo del trabajo de fin de grado es una herramienta web que permitiera llevar a cabo un SCAMPI en una empresa desde un portátil o tablet. La herramienta será capaz de gestionar las preguntas realizadas por SCAMPI (así como una guía para poder realizarla a un equipo de desarrollo), almacenar resultados, analizar y verificar qué prácticas se están cumpliendo correctamente de acuerdo al modelo CMMI-DEV y finalmente mostrar estos resultados. Es un trabajo que trata desarrollo web, diseño de bases de datos, consultoría TI y gestión de TI.

Palabras clave: Plataforma web, SCAMPI, CMMI-DEV, AngularJs, NodeJs, JavaScript, Procesos Software, CMMI-Dev, MongoDB.

Abstract

The Project is framed inside IT Managing. More specifically in software systems developing processes evaluation with CMMI. CMMI can evaluate these processes in order to detect the realization of CMMI good practices. CMMI has three different models but we will focus on CMMI-Dev (CMMI for Development). Inside CMMI there is a way of evaluate development processes with SCAMPI surveys model. This model tries to answer questions like “what’s doing here?”, “how’s doing it?” (Verifying it does) and verify if it accomplishes the best practices described in CMMI models. The objective of the final degree Project is a web tool that allow us to make a SCAMPI in a Company from a Tablet or a computer. The tool will be able to manage SCAMPI questions (as well as SCAMPI guide to make it to a development team), save results, comparisons, analyze and verify which practices are doing well according to CMMI-DEV and finally show this results. This is a work that uses web development, data-base management, data-base design, IT consulting and IT managing.

Keywords : Web Platform, SCAMPI, CMMI-DEV, AngularJS, NodeJS, JavaScript, Software Processes, CMMI-Dev, MongoDB.

Tabla de contenidos

Tabla de figuras

1. Objetivo y motivación

Durante la segunda parte de la carrera, en la especialidad de “Sistemas de la información”, se pudo ver en diferentes asignaturas la importancia del uso de procesos de software en las empresas de desarrollo hoy en día, así como el uso de procesos de negocio tanto en estas como en cualquier empresa.

El uso de procesos de software es vital para muchas empresas, pero muchas veces estos procesos, como también se vio en diferentes asignaturas durante el cuatrimestre B de tercero y el cuatrimestre A de cuarto, no están del todo optimizados y se pierden o malgastan recursos que bien utilizados o aprovechados podrían aportar el cien por cien de la ventaja que el uso de procesos de software podría aportar a la empresa. Así, en estos dos cuatrimestres se investigó y se supo más sobre el uso de procesos, su optimización y, sobretodo, dos marcos para el desarrollo o mejora de éstos: ITIL (*Information Technology Infrastructure Library*) y CMMI (*Capability Maturity Model Integration*).

Se aprendió que ambos marcos proporcionaban un conjunto de buenas prácticas a cumplir en la empresa para un uso correcto de los procesos de negocio, pero no fue hasta investigar más a fondo CMMI cuando se vio que era posible llevar a cabo una evaluación de los procesos de software de una empresa para determinar el estado de éstos y las áreas de mejora que cabían en dicha evaluación sin la necesidad de ser un profesional certificado por el SEI (*Software Engineering Institute*).

Después de todo lo que se había aprendido sobre el uso de procesos, el que se pudiera llevar a cabo una evaluación, previa a la posible certificación, de “cómo de bien se está desarrollando software” en una empresa, era algo con un valor espectacular. La posibilidad de mejorar y aumentar la eficiencia y eficacia de los procesos, aumentando así la competitividad, no era algo que se pudiera conseguir todos los días.

La evaluación o modelo de evaluación que con CMMI para el desarrollo se lleva a cabo, conocido como SCAMPI, es complicado de realizar. No sólo el pasar un cuestionario difícilmente entendible por el encuestado, sino el cálculo de resultados es algo tedioso. Así, surgió la idea de la realización de una herramienta que sirviera para realizar las evaluaciones SCAMPI de CMMI y que proporcionara a la persona que realiza la evaluación una facilidad para llevarlas a cabo.

Ya con la decisión tomada, se añade la idea de convertir el proyecto no sólo en un medio para realizar la herramienta, sino una forma de investigar tecnologías que han aparecido recientemente y que durante la carrera no se pudo profundizar en su conocimiento y uso como es, por ejemplo, NodeJS.

En esencia, el objetivo de este proyecto es conseguir una herramienta web para que las empresas puedan obtener información sobre sus procesos de desarrollo de acuerdo a un marco de buenas prácticas de talla internacional como es CMMI para el desarrollo y servir así de fuente de información para mejorar sus procesos y así su competitividad dentro del sector usando nuevas tecnologías en el desarrollo técnico.

Por otra parte, la motivación aumentó con el proyecto ya comenzado cuando se trabajó en una empresa de producción de software que a pesar de desarrollar un buen producto, éste no le proporciona los beneficios esperados por, en parte, tener unos procesos de desarrollo muy poco trabajados y madurados para conseguir un desarrollo eficiente. La falta de control de calidad, requerimientos, documentación, registros y demás buenas prácticas hizo disminuir su competitividad, mermando sus beneficios a medio plazo y dificultando su crecimiento.

2. Introducción

Vivimos en un mundo globalizado. Las empresas de desarrollo de software tienen que enfrentarse entre sí en un contexto de competitividad total en el cual el éxito de una empresa se rige por la eficacia y eficiencia de sus productos o servicios. Estas dos cualidades son la piedra angular que toda empresa debe intentar manejar para alcanzar sus objetivos.

Es por ello que muchas veces el éxito de una empresa de desarrollo de software no es el producto o servicio que llevan a cabo, sino la competitividad que se pueda derivar de este proceso. Si se desarrolla un buen servicio o producto de software pero la eficiencia o eficacia no ha sido la esperada, la competitividad se verá reducida y así el éxito final de la empresa. El factor determinante de mejora en el desarrollo de software es el uso de procesos de software.

CMMI (*Capability Maturity Model Integration*) es un modelo para la mejora de procesos de desarrollo que permite a las empresas conseguir una alta eficacia en sus procesos software. Así, dentro de CMMI se establece CMMI-DEV *(CMMI for development*), uno de los modelos dentro del propio modelo global de CMMI, que se encuentra especializado en mejorar los procesos de desarrollo y producción de software, desde su concepción hasta su entrega o mantenimiento.

CMMI nos da una visión global del estado de nuestros procesos de desarrollo software y una serie de buenas prácticas a seguir para la consecución de un proceso de desarrollo eficaz y beneficioso. CMMI permite no sólo guiar “en el buen camino” sino que dota a los procesos de una evaluación que permitirá ver el margen de mejora que tienen por delante. En otras palabras, CMMI para el desarrollo aporta a las empresas información sobre el estado actual de sus procesos de desarrollo de software y una vista de lo que podrían conseguir siguiendo una serie de buenas prácticas.

La competitividad es posiblemente el objetivo más importante que una empresa ha de conseguir hoy en día para hacerse un hueco en un sector tan dinámico, cambiante y duro como es el sector de tecnologías de la información, por lo que tener unos procesos eficientes y eficaces es clave para ser competitivos. Basta decir que CMMI y su certificación es una forma de seguir la senda correcta y por ello el desarrollo de una herramienta web para realizar evaluaciones SCAMPI en empresas y medir el estado de los procesos de desarrollo era un proyecto interesante y con posibilidades más allá de las puramente académicas.

Dentro de las evaluaciones SCAMPI se diferencian tres tipos: A, B y C. El tipo de evaluación A es el método oficial de evaluación por parte de un profesional certificado por el SEI y es consecuente de una nota final y oficial sobre los procesos de software. El tipo B, que se escogió para este proyecto, no proporciona una nota pero da una idea aproximada de cómo se están llevando a cabo los procesos software de acuerdo a CMMI-DEV en la empresa. El tipo C es similar al tipo B pero es mucho menos estricto y requiere de menos medios para realizarlo. Más adelante (en el punto 3.4.2) explicamos porqué se eligió el tipo B.

2.1. Distribución del proyecto

El proyecto en sí ha cubierto el desarrollo de un marco teórico para la correcta realización de evaluaciones SCAMPI mediante personal que no necesariamente debe estar certificado para realizar evaluaciones SCAMPI de tipo A (consecuentes de una nota oficial) y el desarrollo técnico de la herramienta y el objetivo del mismo era la consecución de una herramienta que pudiera satisfacer las necesidades de aquellas empresas que quisieran obtener una certificación CMMI oficial pero no pudieran determinar el estado, y por ende el cumplimiento de buenas prácticas, de sus procesos de desarrollo.

2.1.1. Preparación

En primer lugar, se realizaron un par de reuniones con la tutora del proyecto, Llanos Cuenca, y posteriormente se realizó una planificación inicial del proyecto. Se determinaron las fases del desarrollo del proyecto. La programación se estableció de acuerdo a los conocimientos que ya se poseían y los conocimientos que faltaban por adquirir. El proyecto sufrió retrasos que se detallan en este documento (apartados 2.1.4, 4.4.2 y 7)

A continuación una captura de pantalla de la distribución de las diferentes fases del proyecto una vez terminado:

	Nombre de tarea	Comienzo real	Fin real
1	Preparar informacion	lun 27/04/15	vie 22/05/15
2	Diseño	lun 25/05/15	ie 29/05/15
3	Desarrollo	lun 01/06/15	m 02/08/15
4	Testing	lun 29/06/15	un 03/08/15
5	Deployment	lun 27/07/15	ie 07/08/15
6	Memoria	lun 01/06/15	ie 28/08/15

Figura 1: Tiempos reales al finalizar el proyecto.

Como se puede observar, se establecieron tiempos para cada tarea, siendo el grueso del proyecto la búsqueda de información y documentación y el *testing* de la aplicación. Finalmente el desarrollo y el *testing* llevaron mucho más tiempo del esperado y provocaron un retraso en el proyecto de cuatro semanas. Más adelante se detallan las dificultades que surgieron a nivel técnico.

2.1.2. Documentación

Se llevó a cabo diferentes tareas de investigación sobre CMMI, consulta de fuentes oficiales, consulta de experiencias previas en la materia y una planificación sobre cómo llevar a cabo las encuestas. Se llevó a cabo un resumen de todos los elementos que iban a intervenir en las encuestas, así como qué información se iba a necesitar al realizarlas (descripción, ayuda…).

Se establecieron preguntas, descripciones y aclaraciones que se deberían incluir en el proyecto final para facilitar la realización de las encuestas. Fue la única fase del proyecto junto a diseño que no conllevó retraso alguno para el proyecto final, aunque no era una tarea problemática debido a la cantidad de información de la que se disponía de asignaturas en las que se había visto CMMI. En este documento existe un ejemplo de encuesta, en el apartado 3.4.4.

2.1.3. Diseño

En el diseño de la herramienta se realizaron diferentes diagramas de tipo árbol para determinar la estructura de la misma además de diagramas de flujo para poder ver la distribución de las pantallas de la aplicación y cómo debería estar organizada además de un diseño de estructura de datos de la herramienta, con toda la información que pudiera necesitarse de la extraída en la fase de Documentación.

Es también en esta fase cuando se determinan las tecnologías a utilizar durante el desarrollo técnico del proyecto y se decide optar por MongoDB en lugar una base de datos relacional:

-**Front-end (parte cliente)**: HTML5, CSS3, Bootstrap, Javascript y AngularJS.

-**Back-end (parte servidor)**: NodeJS y MongoDB.

La elección de las herramientas de Front-end fue un acierto en todos los sentidos. Al conocer previamente HTML5, CSS3 y Bootstrap, muchos de los componentes del diseño gráfico ya estaban realizados y al conocer JavaScript el comportamiento del front-end tampoco resultó una tarea complicada mediante el uso de AngularJS.

AngularJS proporcionó una cantidad de recursos y facilidades que aceleraron el proceso de desarrollo y en gran medida ayudó a que el proyecto estuviera terminado a tiempo pero el proceso de aprendizaje a utilizar el framework llevó a mucha refactorización del código escrito. Por otro lado, la elección de MongoDB como base de datos en el *back-end* fue, con el tiempo, una decisión que conllevó diferentes dificultades imprevistas puesto que después de comenzar el desarrollo, hubo que cambiar ciertos aspectos de la estructura de datos para poder utilizar MongoDB de acuerdo a una serie de buenas prácticas, lo que dificultó el uso de los datos almacenados en el front-end. Más adelante se detallan estas cuestiones.

En cuanto a NodeJS, su uso fue para tan importante para el término del proyecto como AngularJS, ya que permitió la creación de una API REST que facilitó en gran medida el acceso a los datos almacenados.

2.1.4. Desarrollo

En un principio, el desarrollo estaba pensado para ocupar un mes o 5 semanas de trabajo, puesto que se conocía JavaScript y el diseño estaba claro, sin embargo, surgieron diferentes cuestiones con el tratamiento de datos en el cliente que alargaron el desarrollo.

Hubo que hacer un tratamiento de datos extra en el *front-end*, pero a cambio se aligeró el servidor todavía más, lo que proporciona beneficios como resistencia a fallos por denegación de servicio o una posible explotación como servicio.

Al final, el isomorfismo del que dotó MongoDB a la aplicación por el uso de AngularJS en *front-end* y la API de NodeJS en *back-end* hizo que todo el desarrollo se acelerase un poco, pero éste se alargó durante toda la fase de *testing* debido a un continuo camino de detección de fallos y refactorización del código fruto del continuo aprendizaje que llevaba a cabo de AngularJS.

Hubo un punto en el que todo tuvo que dejarse como estaba para poder continuar con el proyecto. Finalmente, el desarrollo concluyó con varias semanas de retraso aunque a tiempo. Más adelante se explica en detalle el desarrollo técnico del proyecto y las dificultades que se han encontrado.

2.1.5. Testing

La fase de *testing* se alargó por los diferentes problemas encontrados en el diseño de bases de datos y la refactorización continua que sufrió el código. Al final terminó al mismo tiempo que la fase de desarrollo, mucho más retrasada, pero ayudó a la detección de problemas en el formato de envío de peticiones al servidor, problemas de tratamiento de datos en el servidor…etc.

Durante esta fase se realizaron diferentes ensayos con las tipologías, modos de entrada, creación masiva de datos y consultas desde diferentes clientes. Se trataba, en definitiva, de probar la resistencia del servidor, el funcionamiento de las funcionalidades implementadas y descubrir errores que no hubieran sido detectados.

2.1.6. Deployment

El *deployment* se llevó a cabo sin problemas, probándose en dos servidores y llevando a cabo una prueba en una empresa de desarrollo de software. Dicha prueba fue satisfactoria y se llevó a cabo sin problemas.

2.1.7. Distribución previa

Tras llevar a cabo la planificación de cada una de las partes del proyecto, la planificación temporal previa al inicio queda así:

Tarea	Comienzo	Final
Preparación	27 Abril 2015	22 Mayo 2015
Diseño	25 Mayo 2015	29 Mayo 2015
Desarrollo	1 Junio 2015	26 Junio 2015
Testing	29 Junio 2015	24 Julio 2015
Despliegue	27 Julio 2015	31 Julio 2015
Memoria	1 Junio 2015	7 de Agosto 2015

Tabla 1: Tiempos del proyecto programados.

2.2. Planificación real

Al finalizar el proyecto (incluyendo la redacción de este documento) la distribución del tiempo real dedicado al proyecto (según un diagrama de Gantt) queda así:

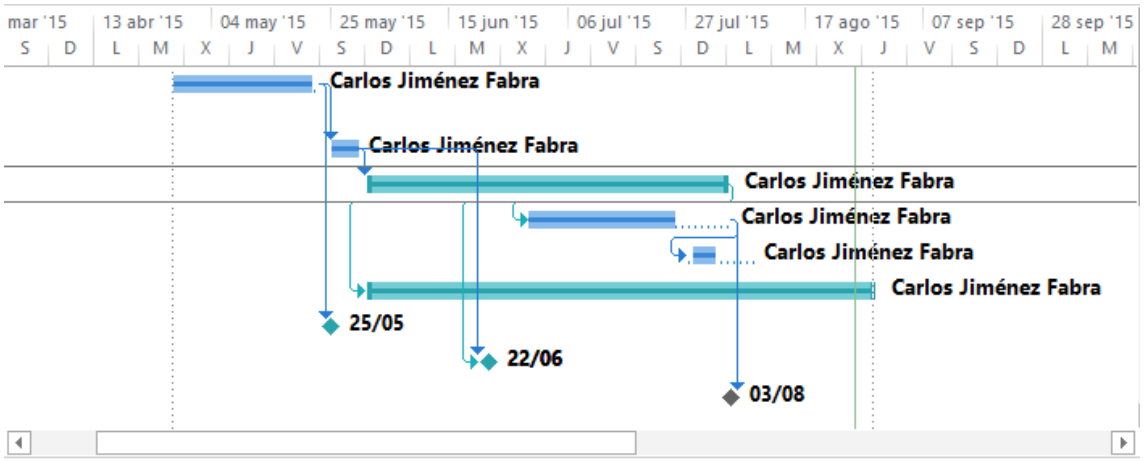

Figura 2: Diagrama de Gantt de distribución del proyecto.

Como se puede observar, el retraso en el desarrollo obligó a retrasar el *testing* y el *deployment* y finalmente acabaron casi a la vez. Finalmente, destacar la memoria, tarea que se comenzó junto con el desarrollo pero que se previó acabara el 1 de agosto. Al final, debido a los retrasos en otras tareas, y puesto que la realización de la memoria se hacía de manera parcial, su realización se alargó hasta el final de agosto.

3. Marco teórico

3.1. ¿Qué es CMMI-DEV?

CMMI es un marco de mejora de procesos software compuesto por tres sub-marcos de trabajo o "constelaciones": desarrollo, adquisición y servicios. Cada uno de estas constelaciones contiene sus elementos propios y todos comparten una serie de elementos comunes.

La constelación de CMMI para el desarrollo (CMMI-DEV a partir de ahora) es un marco de trabajo dirigido a mejorar las actividades de desarrollo de los productos o servicios desde la concepción del producto hasta su mantenimiento mediante el cumplimiento de diferentes prácticas.

Los elementos de CMMI-DEV son las 22 áreas de proceso con sus correspondientes metas específicas y a su vez cada meta específica consta de sus correspondientes prácticas específicas. La metodología del cumplimiento de CMMI-DEV funciona así: cumplir un conjunto de prácticas llevadas a cabo que suponen cumplir unas metas y cuando las metas de un área de proceso están cumplidas, se cumple un área de proceso.

3.2. Historia

Para encontrar las fases iniciales de vida de un modelo o marco de mejora de procesos tan complejo como CMMI tenemos que remontarnos hasta los años 30 del siglo pasado, cuando Walter Stewhart comienza a desarrollar los principios de control por estadística de la calidad. Estos principios son usados por Crosby, Deming, Juran y Humphrey y los refinan durante los años 70 y 80 hasta que Watts Humphrey y Ron Radice entre otros los amplían en IBM y en el SEI (*Software Engineering Institute*) a finales de los años ochenta.

El SEI es el instituto que, financiado en parte por dinero público, fue establecido en la universidad Carnegie Mellon por el departamento de defensa de los Estados Unidos para adaptar los viejos sistemas mainframe a un mundo cada vez más cambiante como eran las tecnologías de la información en los años 80. Humphrey es el que en su libro "*Managing the Software Process*" describe los principios de los CMMs (*Capability and Maturiry Models*) y de su posterior versión y actual modelo CMMI.

Los CMMs que dieron lugar a CMMI fueron desarrollados durante los años noventa y fueron evolucionando desde el primer CMM que el SEI publicó en "The Capability Maturity Model: Guidelines for Improving the Software Process".

Este primer CMM dio lugar a tantos otros CMMs y el SEI se vio con el problema de que los CMMs no buscaban la mejora de toda la empresa sino solo de ámbitos aislados, por ello desde 1987 hasta 1997 tuvo lugar lo que se conoce como "CMM Integration Project" que es el proceso de unificación de diferentes CMMs en un solo marco de trabajo. Este proyecto dio como lugar lo que hoy conocemos como CMMI.

3.3. Componentes, representación y niveles

3.3.1. Componentes

Como ya se ha mencionado antes, cada constelación de CMMI está compuesta por diferentes elementos propios y únicos de la constelación y además modelos generales compartidos con el resto de constelaciones. Así, CMMI para el desarrollo está compuesto por 22 áreas de proceso con metas y prácticas específicas además de otros elementos. De estas 22 áreas de proceso, 16 son áreas compartidas con el resto de constelaciones y 6 son propias de CMMI-DEV. Antes de profundizar en las diferentes áreas de proceso que se encuentran en CMMI-DEV, se hará una breve definición de cada uno de los elementos.

Áreas de proceso: Las áreas de proceso son áreas (también se puede llamar aspectos o características) que son cubiertas por los procesos de software de una empresa, es decir, un conjunto de elementos o características que deben estar cubiertas por los procesos a través del cumplimiento de sus metas específicas.

Meta específica: Una meta específica describe las características fundamentales para satisfacer el área de un proceso. Es un componente requerido y se utiliza para evaluar la satisfacción de un área de proceso. Hay que tener en cuenta que solo la declaración de la meta es un componente específico, ya que título, descripción y demás se considera un componente informativo. Para cumplir una meta específica o genérica, se deben llevar a cabo todas sus prácticas genéricas y específicas.

Meta genérica: Se denominan genéricas porque es una meta que puede ser aplicada de manera global a más de un área de proceso de la organización. Describe características que deben estar presentes para institucionalizar la implementación de un área de proceso. Es decir, como debe realizarse la implementación de un área de proceso. Las metas genéricas sirven para evaluar la satisfacción de un área de proceso. Es un componente requerido en su declaración. Título y notas asociadas son componentes informativos.

Prácticas genéricas: Se denominan genéricas porque, al igual que las metas genéricas, las prácticas también se pueden aplicar a múltiples áreas de proceso, lo que la convierte en una práctica genérica. Suelen estar asociadas con una meta genérica, en ese caso describen actividades que pueden ser importantes para lograr la meta genérica y además ayudar a institucionalizar los procesos asociados a un área de proceso. Un ejemplo que aparece en la documentación oficial de CMMI-DEV v1.3 es: "Proporcionar recursos adecuados para realizar el proceso, desarrollar los productos de trabajo y proporcionar los servicios del proceso". Como ya hemos dicho previamente, únicamente la declaración de la práctica es un componente esperado, título, notas y demás es componente informativo del modelo.

Elaboración de las prácticas genéricas: Aparecen después de las prácticas genéricas a las áreas de proceso. Es un componente informativo del modelo. Para comprenderlo mejor, el manual oficial de CMMI nos ofrece el siguiente ejemplo: "*Esta política establece las expectativas de la organización para estimar los parámetros de la planificación, para definir compromisos internos y externos, y para desarrollar un plan para gestionar el proyecto*" para la práctica "*Establecer y mantener una política de la organización para planificar y realizar el proceso*" en el área *Planificación del Proyecto*.

Prácticas específicas: Una práctica específica se utiliza para describir una actividad que se considera importante o fundamental para lograr una meta específica asociada a una meta. La declaración de la práctica específica se ve reflejada como un componente en el modelo. Además el título y las anotaciones vinculadas a la práctica específica se ven como componentes informativos del modelo.

Componentes informativos: Debido a que en muchas empresas y organizaciones aparece la necesidad de proporcionar más información para describir un concepto, CMMI-DEV contiene el apartado de Componentes informativos, que consta de los siguientes elementos:

- **Notas**: es un texto que puede acompañar o estar vinculado a cualquier componente del modelo, proporciona detalles, antecedentes o análisis razonado sobre estos.

- **Ejemplos**: son un componente que consta de texto enmarcado en un recuadro, contiene una lista de elementos, y puede acompañar a cualquier otro componente del modelo, su función es aportar uno o varios ejemplos para aclarar el concepto o la actividad en cuestión.

- **Referencias**: una referencia es un elemento informativo del modelo y consta de un enlace que prácticamente puede acompañar a cualquier tipo de componente del modelo.

Ahora que se han explicado los componentes de CMMI-DEV, se puede llegar a la conclusión de todo en una frase:

"Las áreas de proceso agrupan una serie de metas en un ámbito relacionado, las cuales se alcanzan mediante el cumplimiento de un conjunto de prácticas".

3.3.2. Representación

CMMI define dos tipos de representación, la representación por etapas y la continua. La representación continua es una forma de producir mejoras en la empresa centrándose en las áreas de proceso que considera más importantes para su empresa, en cambio la representación por etapas les da una especie de "camino a seguir" para mejorar su empresa y un nivel de madurez para comparar su empresa y su madurez a lo largo del tiempo.

Por consiguiente, habrá que definir niveles para poder seguir las dos representaciones. Esto implica que cada área de proceso tendrá un nivel para cumplirla de forma continua y por etapas, es decir, un nivel de capacidad y un nivel de madurez.

Al seguir la representación por etapas, habrá que alcanzar un nivel de madurez en la empresa alcanzando o cumpliendo con las áreas de proceso de ese nivel madurez. Por otro lado, tendrá unos niveles de capacidad que se aplicarán a las metas y no a las áreas de proceso directamente.

Antes de continuar explicando los niveles y entrando un poco más en CMMI-DEV, hay que anotar que solo un profesional cualificado puede calificar con un nivel de madurez a una empresa, y puesto que la meta de este trabajo es dar la posibilidad de ver el estado de los procesos software de la empresa sin ser profesionales cualificados por el CMMI institute, el proyecto no se centrará en los niveles de madurez o capacidad, sino que valorará el cumplimiento de las diferentes prácticas, metas y áreas de proceso. La metodología del SCAMPI de tipo B diseñado para el proyecto se explica en el apartado 3.4 de este documento.

3.3.3. Niveles

Ahora que se han visto cuales son los componentes, hay que centrarse en una de las partes más importantes de un modelo de mejora, los niveles de clasificación.

Los niveles en CMMI-DEV se utilizan para describir un camino evolutivo para mejorar los diferentes procesos que se utilizan en la elaboración de los productos o servicios de la empresa, además también se puede evaluar a las empresas y clasificarlas según dichos niveles.

CMMI-DEV tiene dos grandes tipos o áreas de componentes que son el nivel de capacidad y de madurez:

• **Nivel de capacidad**: tiene 4 clasificaciones, y se pueden alcanzar uno de esos niveles de capacidad cuando se satisfacen todas las metas genéricas y específicas de un área del proceso hasta ese nivel. Propio de la representación continua.

• **Nivel de madurez**: tiene 5 clasificaciones y para alcanzar cada uno de estos niveles existen diferentes prácticas genéricas y específicas relacionadas entre sí que se deben cumplir, el nivel de madurez se utiliza para caracterizar y mejorar el rendimiento de una organización. Propio de la representación por etapas.

En esta sección veremos además cada uno de los niveles con un poco más de detalle:

Según el nivel de capacidad:

• **Nivel 0** (Incompleto): se dice que un proceso es incompleto cuando este no se realiza o bien, se realiza pero parcialmente, es decir, existen al menos una o más metas específicas del área del proceso que no se satisfacen y no existen metas genéricas para este nivel.

• **Nivel 1** (Realizado): se dice que un proceso está realizado cuando este lleva a cabo el trabajo necesario para producir los productos o servicios que le han sido asignados, es decir, todas sus metas específicas han sido alcanzadas.

• **Nivel 2** (Gestionado): se dice que un proceso está gestionado cuando este pertenece al nivel 1 y además se planifica y ejecuta siguiendo la política de la empresa, además de contar con el personal adecuado para realizar las tareas, las diferentes fases se monitorizan, controlan y revisan, con esta disciplina que caracteriza al nivel 2 se asegura que se mantienen las practicas existentes para el correcto funcionamiento del proceso.

• **Nivel 3** (Definido): se dice que un proceso está definido cuando este pertenece al nivel 2 y que se adapta a los procesos estándar de la organización siguiendo las guías de adaptación de dicha organización, además sus procesos se mantienen y contribuyen a los activos de proceso de la organización.

Según el nivel de madurez:

• **Nivel 1** (Inicial): este nivel se caracteriza porque sus procesos son generalmente ad hoc y caóticos, ya que la organización no proporciona un entorno estable para dar soporte a los procesos. El éxito en estas organizaciones depende de la competencia y de la heroicidad del personal de la organización y no del uso de procesos probados.

• **Nivel 2** (Gestionado): este nivel se caracteriza porque los procesos se planifican y ejecutan siguiendo las políticas de la organización, también emplean personal cualificado y además de monitorizar, controlar y revisar los procesos, los servicios y productos de trabajo satisfacen sus descripciones, estándares y procedimientos específicos.

• **Nivel 3** (Definido): este nivel se caracteriza porque los procesos se describen en estándares, procedimientos, herramientas y métodos. El conjunto de procesos estándar que son la base de la organización se mejoran a lo largo del tiempo. Una diferencia crítica entre el nivel 2 y el 3 es el alcance de los estándares y las descripciones de los procesos. Otra diferencia crítica es que los procesos del nivel 3 se gestionan de una manera proactiva en comparación a los procesos del nivel 2.

• **Nivel 4** (Gestionado cuantitativamente): este nivel se caracteriza porque se establecen objetivos cuantitativos para la calidad y el rendimiento del proceso, que se basan en las necesidades del cliente y también posteriormente son utilizados como criterios para la gestión de los proyectos. Una diferencia crítica entre los niveles 3 y 4, es que en el nivel 4 el rendimiento de los proyectos y de los subprocesos se controla mediante técnicas estadísticas y las predicciones se basan en el análisis estadístico de los datos.

• **Nivel 5** (En optimización): este nivel se caracteriza por la utilización de un enfoque cuantitativo para comprender la variación inherente en el proceso y los resultados de este. Este nivel se centra continuamente en mejorar el rendimiento de los procesos y de la tecnología. Una diferencia crítica entre el nivel de madurez 4 y 5, es que en el 5º, hay una preocupación por el rendimiento global de la organización y para lograrlo analizan y se basan en los datos recogidos de múltiples proyectos, ya que estos les muestran la existencia de posibles lagunas en los procesos.

3.4. SCAMPI

3.4.1. ¿Qué es?

SCAMPI son las siglas de "*Standard CMMI Appraisal Method for Process Improvement*", el método oficial de calificación del SEI (*Software Engineering Institute*) de acuerdo a los modelos CMMI, entre los que se encuentran CMMI-DEV.

Un SCAMPI es la forma mediante la que se pueden detectar las áreas de mejora que existen en una empresa según el modelo CMMI escogido, detectando por lo tanto fortalezas y debilidades en una empresa de acuerdo a las áreas de proceso sobre las que se realiza la SCAMPI.

Este análisis de los procesos software, como ya se ha comentado en este documento, permite mejorar el estado de los procesos de la empresa, así como el uso eficiente que se hace de los recursos que dispone aumentando la eficacia de su actividad. Todos estos beneficios llevan a mejorar la competitividad, pero para conseguirla necesitamos saber cuál es el estado de los procesos respecto CMMI y qué prácticas no se están cumpliendo.

En general, cuando se lleva a cabo un SCAMPI en una empresa, el resultado es un análisis del estado del proceso de la empresa, puesto que la calificación que se obtiene es sobre el cumplimiento del modelo CMMI y por lo tanto de sus áreas de proceso, metas y prácticas.

El objetivo último depende del tipo de SCAMPI, puesto que hay diferentes tipos de SCAMPI a realizar, pero todos tienen en común el análisis y mejora de los procesos que existen en una empresa.

Existen tres tipos de SCAMPI: tipo A, que sirve para dotar de una calificación oficial a la empresa en la que se realiza y que solo puede ser realizada por un profesional cualificado por el SEI para ello, tipo B y tipo C, que pueden ser realizadas por alguien que no esté cualificado por el SEI para llevar a cabo SCAMPIs de tipo A, pero no dota de una nota final, sino que dota a la empresa de información acerca de sus procesos y qué áreas de proceso están dominadas por las debilidades o las fortalezas, es decir, es una especie de "vista previa" de nuestros procesos de software sin que vayamos a recibir una nota. Muchas empresas utilizan estos dos últimos tipos de SCAMPI para ver el estado de sus procesos.

3.4.2. Porqué tipo B

Este proyecto utiliza SCAMPIs de tipo B por diferentes razones:

- Es más flexible que las de tipo A.
- Proporciona un resultado parecido al tipo A, viendo cual podría ser el resultado oficial.
- No se necesita demasiado personal ni recursos.
- Es más riguroso que el tipo C.

Las diferencias entre el tipo B y C son pocas, pero importantes. Principalmente, lo que diferencia al tipo B del tipo C es la granularidad del proceso. El tipo C puede aplicarse a cualquier alcance del modelo, pudiendo situarse en prácticas o metas, mientras que el tipo B podría aplicarse a cualquier alcance (esto está algo discutido por expertos de CMMI) el proyecto se centra en el cumplimiento de las áreas de proceso por completo por lo que este nivel de granularidad es el correcto dentro de los SCAMPI de tipo B.

3.4.3. Metodología

La metodología que se va a utilizar para realizar la evaluación es sencilla:

- Se determina un nuevo SCAMPI sobre una nueva empresa.
- Se determinan las diferentes áreas de proceso sobre las que se va a realizar la evaluación.
- Se lleva a cabo una entrevista individual a cada uno de los miembros del equipo de desarrollo y se le realiza la encuesta.
- Se cierra la SCAMPI y se determinan los resultados obtenidos.

Por lo tanto, lo que se lleva a cabo es una encuesta individual a cada uno de los trabajadores o miembros del equipo de desarrollo y se determina mediante una media cual es el estado de las prácticas de cada meta y de cada área de proceso, determinando el estado de éstas.

Cada práctica conlleva una pregunta, la cual se responde en uno de los siguientes grados de afirmación:

- Siempre (*Always*)
- Casi siempre (*Usually*)
- A veces (*Sometimes*)
- Casi nunca (*Almost never*)
- Nunca (*Never*)

Las puntuaciones que se obtienen son 10, 6.6, 5, 3.3 y 0 respectivamente. Esta evaluación se realiza con cada una de las prácticas que componen cada meta que componen cada área de proceso seleccionada. Así, se realiza una media de las notas obtenidas en cada una de las prácticas específicas y se obtiene una media de meta, que tras una media con el resto de metas se obtiene una nota de área de proceso.

Así, tenemos una vista previa del cumplimiento de las áreas de proceso en nuestra empresa. Es una forma de ver cuál es el estado de los procesos y qué deberíamos mejorar en nuestra empresa pues nunca garantizará que el cumplimiento se dé a ciencia cierta pues el encuestador no será un profesional cualificado por el SEI. Es, ante todo, una herramienta de análisis y dotación de información sobre qué se está haciendo bien o mal en el proceso de desarrollo y, quizás, justificar una baja efectividad o eficacia del proceso de desarrollo de software.

3.4.4. Ejemplo de encuesta

En este apartado, se muestran los componentes de una encuesta sobre un área de proceso de una SCAMPI. En este caso se ha elegido el área de proceso *Process and Product Quality Assurance* (PPQA), que trata prácticas sobre el control de calidad del proceso y el producto. Es, además, el área de proceso que se utilizó para validar el proyecto en la empresa.

La encuesta que se diseñó se componía de: Nombre, Empresa y Áreas de proceso sobre los que responder. Cada área de proceso tenía código, nombre y una serie de metas que a su vez se componían de múltiples prácticas. Cada meta, a parte de sus prácticas, tenía nombre y código y cada práctica tenía nombre, código, descripción, pregunta y, de forma opcional, aclaración o ejemplos.

Una encuesta sobre el área de proceso PPQA sería:

Nombre del SCAMPI.

Company: Compañía.

Process Areas to accomplish:

PPQA - Process and Product Quality Assurance

Goal SG1

Objectively Evaluate Processes and Work Products

Practice SP11 - Objectively Evaluate Processes

Objectivity in quality assurance evaluations is critical to the success of the project. A description of the quality assurance reporting chain and how it ensures objectivity should be defined.

Do you objectively evaluate selected performed processes against applicable process descriptions, standards, and procedures?

Never - Almost never - Sometimes - Usually - Always

Practice SP12 - Objectively Evaluate Work Products

Example Work Products 1. Evaluation reports 2. Noncompliance reports 3. Corrective actions Sub practices 1. Select work products to be evaluated based on documented sampling criteria if sampling is used. Work products can include services produced by a process whether the recipient of the service is internal or external to the project or organization. 2. Establish and maintain clearly stated criteria for the evaluation of selected work products. The intent of this sub practice is to provide criteria, based on business needs, such as the following: What will be evaluated during the evaluation of a work product when or how often a work product will be evaluated, how the evaluation will be conducted, who must be involved in the evaluation 3. Use the stated criteria during evaluations of selected work products.

Do you objectively evaluate selected work products against applicable process descriptions, standards, and procedures?

Never - Almost never – Sometimes – Usually - Always

Goal SG2

Provide Objective Insight

Practice SP21 - Communicate and Resolve Noncompliance Issues

Noncompliance issues are problems identified in evaluations that reflect a lack of adherence to applicable standards, process descriptions, or procedures. The status of noncompliance issues provides an indication of quality trends. Quality issues include noncompliance issues and trend analysis results. When noncompliance issues cannot be resolved in the project, use established escalation mechanisms to ensure that the appropriate level of management can resolve the issue. Track noncompliance issues to resolution

Do you communicate quality issues and ensure the resolution of noncompliance issues with the staff and managers?

Never - Almost never – Sometimes – Usually - Always

Practice SP22 - Establish Records

Example Work Products 1. Evaluation logs 2. Quality assurance reports 3. Status reports of corrective actions 4. Reports of quality trends Sub practices 1. Record process and product quality assurance activities in sufficient detail so that status and results are known. 2. Revise the status and history of quality assurance activities as necessary.

Do you establish and maintain records of quality assurance activities?

Never - Almost never – Sometimes – Usually – Always

3.5. Porqué utilizar CMMI-DEV

CMMI-DEV proporciona una serie de buenas prácticas a llevar a cabo en la empresa con la que se puede mejorar su competitividad mejorando el uso que ésta hace de sus recursos y convirtiendo los procesos software en procesos eficientes y eficaces. CMMI-DEV ayuda a que las empresas utilicen sus recursos de manera adecuada y es por esto que muchas empresas desean adquirir una certificación CMMI [3].

El conjunto de beneficios que aporta CMMI-DEV a la empresa son [2]:

- Mejoras en el profesionalismo y creatividad de los trabajadores.
- Aumento de la satisfacción con el trabajo.
- Mejora el uso eficaz y eficiente de los recursos.
- Aumento de la ventaja competitiva.
- Detección de fortalezas y debilidades en una empresa.

Cuando una empresa quiere obtener una certificación de CMMI, un profesional certificado por el SEI debe llevar a cabo una evaluación de sus procesos de software para ver si cumple con las buenas prácticas descritas en el modelo. Sin embargo, una empresa pocas veces puede permitirse lanzarse a llevar a cabo una certificación si no sabe que su cumplimiento está asegurado, es por esto que en CMMI se puede llevar a cabo una "vista previa" de la empresa y el estado de sus procesos antes de intentar obtener una certificación.

Por ello, CMMI no sólo sirve como método de certificación, sino que proporciona un medio de análisis y observación del estado de los procesos de desarrollo de la empresa, que prácticas se están cumpliendo, cuáles no, y, ya en el caso de que la empresa quisiera alcanzar una certificación, servir de "Vista previa" sobre el resultado que obtendrá.

4. Aplicación

4.1. Planteamiento

En las primeras fases del trabajo se establecen las etapas de desarrollo teniendo en cuenta las tecnologías que se van a necesitar y los tiempos disponibles. Por ello, se deciden dos grandes bloques de desarrollo, el desarrollo front-end y el desarrollo *back-end*.

Sin embargo, estos dos bloques no pueden ser tratados como bloques igualitarios pues el *front-end* comprende las vistas y el diseño gráfico mientras que el *back-end* comprende tratamiento de datos en cliente, acceso a los recursos en bases de datos… por ello se crean dos bloques alternativos: cliente y servidor.

En cuanto al cliente podemos encontrar las tareas de:

- Diseño de la aplicación.
- Desarrollo de las vistas
- Desarrollo de los controladores.
- Desarrollo de servicios.

Por la parte del servidor encontramos las tareas de:

- Desarrollo de modelos.
- Desarrollo de servicios.
- Desarrollo de API.
- Gestión de base de datos.

Hay que destacar que el lenguaje utilizado en todo el proyecto ha sido el inglés, puesto que la documentación de CMMI-DEV aunque lleve unos meses traducida al castellano por el INTECO, se encuentra en todo el resto de constelaciones en este idioma y la procedencia de los modelos es Estados Unidos. Además, las buenas prácticas en cuanto al desarrollo de software indican que el desarrollo y los prototipos deben hacerse siguiendo la lengua anglosajona. Por esto, la versión de la herramienta que aquí se presenta es en inglés, constando todos los elementos de la interfaz y toda la declaración de variables y diferentes elementos de desarrollo en este idioma a excepción de comentarios puntuales.

4.2. Herramientas utilizadas, introducción a cada una.

4.2.1. Lenguajes de programación utilizados.

En la fase de diseño se determinó cuáles serían los lenguajes y *frameworks* utilizados en el proyecto. Se puso énfasis en el uso de las últimas especificaciones de CSS, CSS3, y

HTML, HTML5, en el maquetado y diseño web, así como en el uso de JavaScript, conocido por el autor, y NodeJS en el servidor y MongoDB en bases de datos.

También se puso énfasis en la creación de una API REST con Express en NodeJS y el uso de AngularJS en el cliente por el hecho de ser una tecnología de reciente creación y Bootstrap por la facilidad y diseño responsive que aportaba al proyecto.

4.2.2. AngularJS

AngularJs es un *framework* de creación de aplicaciones web que proporciona un ecosistema MVC ("*Model, View, Controller*") y un gran número de servicios para facilitar su desarrollo como los servicios de localStorage (para almacenar datos en el navegador del cliente) o resource (para establecer conexiones con APIs REST).

Es, en esencia, un *framework* que facilita la creación de aplicaciones web siguiendo una arquitectura MVC. Es precisamente esta arquitectura la seguida en el cliente (y en parte también en el servidor) y el uso de AngularJS ha sido determinante en la finalización del proyecto pues ha proporcionado herramientas que habría costado al proyecto una inversión en tiempo que no se tenía. AngularJS se utiliza en todo el desarrollo del cliente y está desarrollado en JavaScript [7].

4.2.3. NodeJS/Express

NodeJs es una plataforma o *framework* orientado a eventos que permite construir aplicaciones o servidores escalables. Node tiene una gran cantidad de "módulos" para construir servidores, pero se ha decantado por uno externo, Express, que es un *framework* de creación de servidores web adicional que permite definir rutas específicas y mapping de funciones adicionales en estas rutas para crear una API. Así, la API funciona sobre Express que a la vez funciona sobre Node. NodeJS y Express utiliza también JavaScript [6].

4.2.4. Bootstrap

Bootstrap es un *framework* de diseño web que proporciona multitud de herramientas y recursos web para el diseño de la aplicación web así como un "grid" o cuadrícula para posicionar los elementos en pantalla.

Bootstrap se ha utilizado en la creación de todas las vistas de la parte cliente y el diseño apenas se ha cambiado de la versión original. Ha permitido ahorrar mucho tiempo en diseño web y además da a la aplicación un diseño elegante y moderno que habría costado mucho conseguir. Bootstrap está escrito en CSS3 y JavaScript [8].

4.2.5. Git

Debido a la multitud de cambios y tecnologías que se iban a realizar, además del uso de un servidor remoto para probar el correcto funcionamiento de la API, se recurrió a GIT como medio para mantener un control de versiones en la nube y además poder desplegar el servicio de forma remota y sencilla sin tener que recurrir a otras opciones más difíciles o pesadas de realizar como sería el uso de SSH.

Git era, sencillamente, lo más sencillo. Por otro lado, su control de versiones ayudaría en caso de tener que revertir cambios o volver a una versión anterior, aspecto que no hubo que utilizar [10].

4.2.6. MongoDB

MongoDB es una plataforma para la creación y gestión de bases de datos documentales no relacionales. Proporciona una forma fácil y sencilla de almacenar documentos de tipo JSON en colecciones y éstas en bases de datos Mongo. Funciona con NodeJS y utiliza el lenguaje JavaScript como lenguaje de consultas y explotación de su propia API [9].

La decisión de usar MongoDB fue una decisión tomada cuando el proyecto ya estaba iniciado, pero se optó por su uso debido a:

a) En un afán de experimentar y aprender una nueva tecnología de bases de datos.
b) Dotar al proyecto de un isomorfismo total, usando NodeJS, AngularJS y MongoDB se tenía JavaScript como lenguaje en todas las capas de la aplicación, incluyendo la capa de datos.
c) Falta de esquema. Propiedad de los documentos de MongoDB que permitía crear documentos totalmente diferentes en una misma colección, algo que las bases de datos relacionales (con esquema) no daba.

4.3. Arquitectura

En la práctica, el desarrollo de los bloques cliente y servidor comprende el desarrollo de cada bloque en un servidor diferente, es decir, un servidor para el cliente y otro para el servidor. Esto permite un mejor control de los recursos utilizados y permite la creación de múltiples instancias de cada uno en caso de necesitarlo.

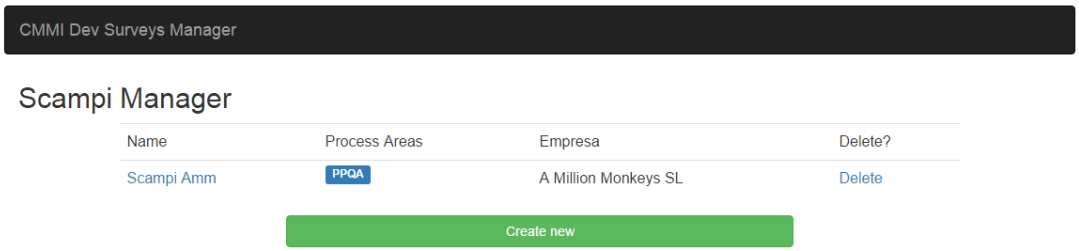

Figura 3: Imagen de la página principal de la aplicación.

Durante el desarrollo del proyecto, se contrató un servidor Unix en remoto para poder probar la API y su correcto funcionamiento además de ver cómo se comportaba el servidor sin tener dos servidores NodeJS iniciados en la misma máquina, una especie de simulacro por si en algún momento el desarrollo del servidor se convierte en un servicio mantenido en lugar de una parte de un producto.

4.3.1. Cliente

El cliente es la parte del proyecto que comprende la web de encuestas, con la creación de éstas, muestra de datos en *front-end* y en definitiva el servidor que se encarga de las gestiones previas al almacenamiento de datos en las bases de datos. Es el servidor de salida de datos almacenados y entrada de datos nuevos provenientes de las acciones del usuario con las vistas. Contiene vistas, controladores y servicios de acceso a la API del servidor.

- **Vistas**: componente visual de la aplicación. Es la interfaz visual que interactúa con el usuario. Muestra los datos que el usuario introduce y los datos que el controlador le pasa. Tendremos múltiples vistas que comprenden la maquetación de cada vista.
- **Controladores**: llevan a cabo funciones de control, modificación y comunicación con los servicios. Es un middleware entre las vistas y los servicios del cliente. Controla los *scope* de la aplicación así como el *scope* global de ésta. Obtiene información desde las vistas y ejecuta funciones de servicios acorde a esta. También solicita y obtiene información de los servicios y los comunica a las vistas.
- **Servicios**: gestiona la comunicación con la API de la parte servidor y los modelos de datos de la aplicación. El cliente tiene dos modelos de datos básicos como veremos más adelante, el modelo Scampi y el modelo ProcessArea. Estos dos modelos son gestionados por los servidores ScampiSvc y ProcessAreaSvc.

Además habrá que crear un servicio de acceso a los recursos del servidor mediante comunicación con la API, resources.

A través de las vistas un usuario ve e introduce datos que los controladores manejan tanto para mostrar en las vistas como para pasar a los servicios pertinentes y éstos llevan a cabo las funciones de comunicación con la API en la parte servidor.

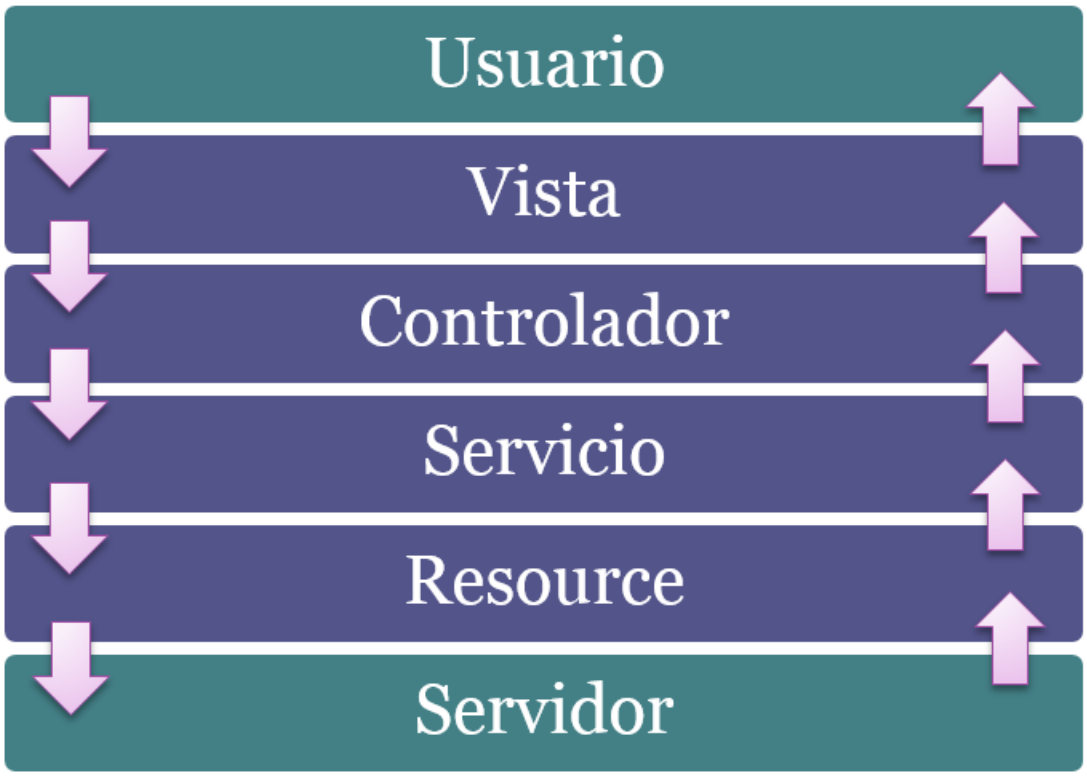

Figura 4: Arquitectura del cliente.

4.3.2. Servidor

El servidor se encarga de manejar una API REST y el acceso a los datos almacenados en base de datos así como las bases de datos. También define los modos de acceso a las entidades contenidas en las bases de datos y qué datos se obtiene en cada *endpoint* definido en la API.

- **API server**: la API define los *endpoints* a través de los cuales se accede a los datos contenidos en base de datos. Si se accede a uno u otro *endpoint*, entonces se accede a uno u otro servicio y por lo tanto a uno u otro modelo y uno u otro dato. En los *endpoints* se definen la entidad a la que se accede y en algunos casos algún modo especial de acceso. Es en el body de la petición HTTP donde se define si es entrada o salida de datos. Establecemos un servidor Express y diferentes rutas a través de las cuales se ejecuta un servicio u otro y el modelo correspondiente.
- **Servicios**: los servicios son una capa middleware entre los *endpoints* contenidos en la API y las funciones de los modelos. Es decir, es un mapeado desde los

endpoints hasta las funciones de los objetos de los modelos. Se referencia un modelo y se crea un objeto de este cuando se accede a un *endpoint*. Esta función de *mapping* la realizan los servicios del servidor.

- **Modelos**: los modelos que aquí se detallan son en realidad la capa de servicios de acceso a datos más baja que existe antes de las propias bases de datos. Es decir, son un conjunto de servicios que se comunican directamente con la base datos y crean un objeto con las funciones de acceso a base de datos y entidad pertinentes (P.Ej: el modelo Area contiene las funciones de acceso a las instancias Area de la base de datos a las que se puede acceder al crear un objeto de tipo Área).
- **Entidades en base de datos**: son las entidades definidas en la base de datos. Estas entidades se definen a modo de contenedor de información y en nuestro caso hemos utilizado documentos dentro de colecciones en bases de datos MongoDB.

En resumen, lo que el servidor hace es definir unos puntos de acceso, a través de los cuales se accede a un objeto que contiene funciones de acceso a base de datos que se ejecutan y se lleva a cabo el traspaso de información entre cliente y servidor.

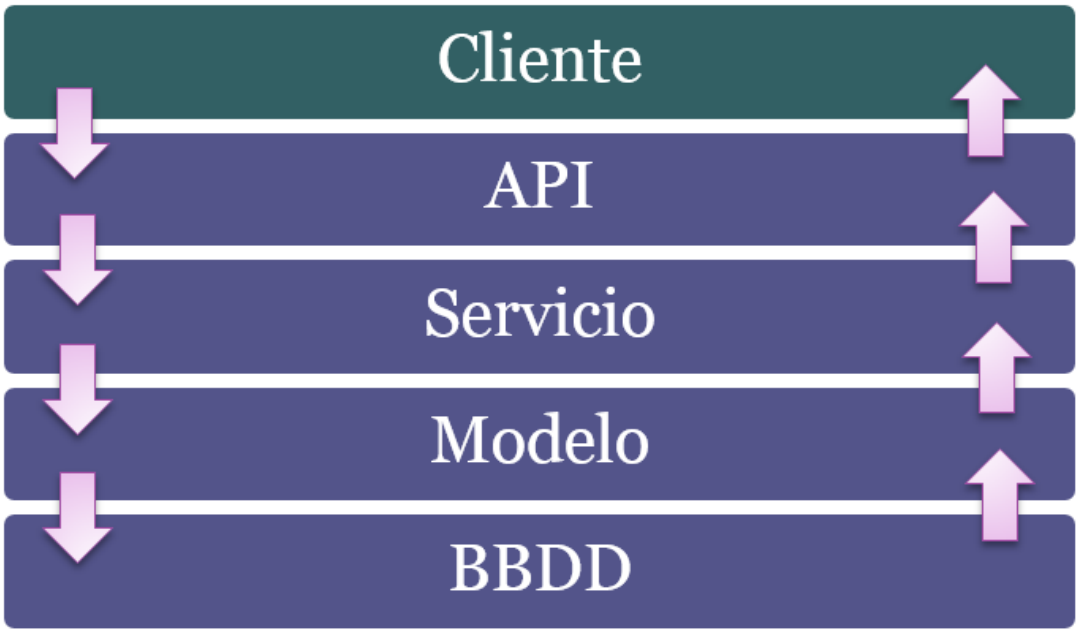

Figura 5: Arquitectura del servidor.

4.4. Front-end

El desarrollo del *front-end* comprendió casi un 70% del tiempo total del desarrollo, principalmente por lo costoso de crear tantas vistas y controladores, pero sobre todo por la complejidad de programación que había en el tratamiento de SCAMPIs ya realizados y en el tratamiento de resultados, los cuales se convirtieron en un hito final ya que la elección de la arquitectura de MongoDB y sus modelos no relacionales complicaron bastante las tareas de recuperación de datos en cliente.

Sin embargo, la elección de un cliente con tanta complejidad restó esta complejidad en el servidor lo que convierte al servidor en un ente algo más escalable y resistente a fallos por denegación de servicio lo que al final fue algo positivo. Además, un cliente no puede ser ejecutado por más de una instancia por usuario, es decir, un usuario solo puede ejecutar una instancia de cliente a la vez, así que esa complejidad se ve disminuida.

Como veremos en este apartado, la elección de AngularJS y Bootstrap fueron quizás los grandes aciertos del proyecto junto con NodeJS/Express en la parte servidor. Facilitaron el desarrollo e hicieron todo mucho más profesional y estable ahorrando tiempo en el desarrollo.

4.4.1. Carpetas y archivos en la aplicación cliente

El diseño de la aplicación cliente (también llamada en este documento "parte cliente del proyecto" o simplemente "cliente") se llevó a cabo de manera más o menos improvisada aunque las necesidades básicas estaban claras desde un principio. Tenemos una carpeta ("/app") donde está contenida la aplicación en AngularJS.

Debido a las necesidades de AngularJS y NodeJS, el proyecto cuenta en la parte cliente con un archivo de configuración e iniciación de la aplicación de AngularJS, "/app/app.js", donde se definen las rutas, la configuración inicial y algunas tareas iniciales, y un archivo de configuración del servidor web en NodeJS que inicia el servidor, "/webserver.js".

Además de esto, se definieron carpetas para los servicios ("/app/services"), archivos css ("/app/css") que contenía todo lo necesario para que Bootstrap funcionara, scripts externos ("/app/js"), la página índice inicial ("/app/index.html") donde se referencian todos los archivos necesarios por la aplicación y una última carpeta para los servicios de acceso a la API ("/app/resources").

A todo esto hay que añadir la carpeta que contiene las vistas y controladores de la aplicación. Hay muchas discusiones acerca de cómo se puede definir la arquitectura en AngularJS, aunque la más aceptada es aquella en la que la distribución de carpetas y archivos contiene en una sola las vistas y sus controladores que confieren funciones individuales a la aplicación. Se siguió esta buena práctica y se definió "/app/features" donde se contienen cada una de las vistas divididas por características.

En un principio, la aplicación cliente se disponía con las siguientes vistas:

- "/app/features/home": Vista general de la aplicación.
- "/app/features/createScampi": Vista de creación de nueva SCAMPI.
- "/app/features/scampi": Vista de detalle de SCAMPI.
- "/app/features/scampi/new": Vista de adición de encuesta a la SCAMPI seleccionada.
- "/app/features/commons/structure": Vista de la estructura y header de la aplicación.

Sin embargo, a estas cinco vistas hubo que añadir una página adicional de gestión del contenido en bases de datos inicial que complicó las cosas. Antes de continuar con los detalles de este apartado de la aplicación, al haber sido improvisado durante el desarrollo del proyecto, lo comentaremos en un apartado de este documento más adelante donde explicaremos los diferentes detalles y razones de esta página de gestión de las bases de datos del servidor.

Cada una de las vistas creadas en cliente necesitó de un controlador que se introdujo en la carpeta de cada una de las "features" o vistas correspondiente.

Así, la distribución por carpetas y archivos era la siguiente:

Figura 6: Distribución de carpetas en cliente.

4.4.2. Desarrollo

El desarrollo del cliente comprendía la creación de una multitud de archivos diferentes entre los que se encuentran la creación de vistas, controladores, servicios y resources. Como ya se ha explicado, los resources son en realidad una compilación de servicios de acceso a API REST.

El desarrollo seguía el orden Servicio-Vista-Controlador. Es decir, en primer lugar se determinaban las necesidades de datos de la vista o "feature" y a partir de ahí se creaba el servicio si los existentes no satisfacían esas necesidades de datos y entonces se desarrollaba la vista y el controlador a la vez aunque el controlador se creaba a partir de las necesidades de la vista ya que el controlador es un middleware entre servicio y vista. En este documento separamos el desarrollo no por funciones sino por tipo para hacerlo más sencillo.

Features: Vista + Controlador

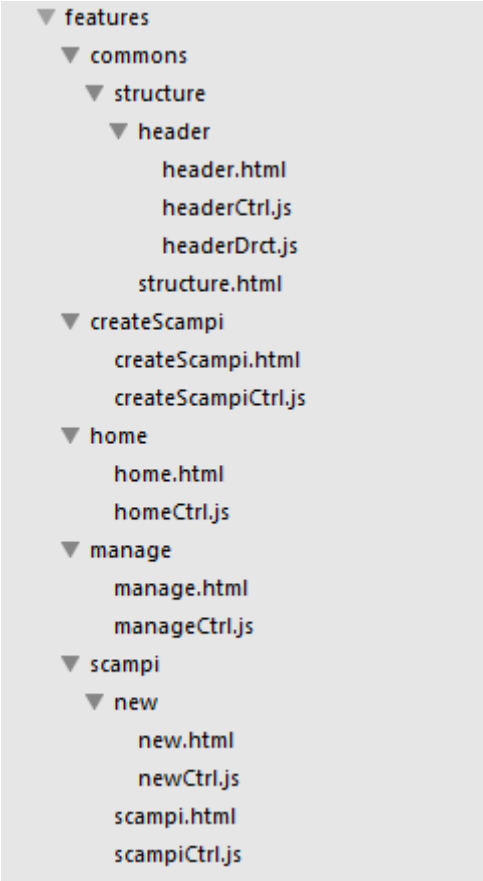

Figura 7: Distribución de carpetas de "features".

- **"/app/features/home": Vista general de la aplicación.**

La vista home contiene simplemente una lista de los SCAMPIs que se ha llevado a cabo o, en el caso de que no se tuviera ninguna SCAMPI iniciada o terminada, muestra simplemente un botón que lleva a la vista de creación de nuevos SCAMPI. Si aparece la lista, desde esta se puede acceder al detalle del SCAMPI o eliminarlo directamente (Ver Figura 1.)

El controlador ("homeCtrl") contiene dos funciones definidas en el *scope* de la vista, "initialize" y "deleteScampi" las cuales dan inicio al estado de la vista o eliminan un SCAMPI.

La función "initialize" se comunica con el servicio "scampisSvc" y obtiene la información de todos los SCAMPIs, los almacena en un array para mostrarlos en la vista a través de una tabla HTML mientras que la función "deleteScampi" hace lo propio eliminando un SCAMPI.

```
$scope.initialize=function(){
	scampisSvc.getScampis().then(function(response){
		$scope.scampis=response;
	});
}

$scope.deleteScampi=function(scampi){
	scampisSvc.deleteScampi(scampi).then(function(response){
		var idx=$scope.scampis.indexOf(scampi);
		$scope.scampis.splice(idx, 1);
	});
}
```

Figura 8: Funciones "initialize" y "deleteScampi".

La tabla que muestra la información de los SCAMPIs por pantalla contiene una iteración del tipo “ng-repeat” que lleva a cabo iteraciones de contenido de acorde a los datos contenidos en un array. La función “deleteScampi” elimina un SCAMPI pasando como parámetro el objeto SCAMPI completo.

- **“/app/features/createScampi”: Vista de creación de nuevo SCAMPI.**

La vista “createScampi” está formada principalmente por el formulario de creación de un nuevo SCAMPI, por lo que la vista en HTML está formada por un formulario con varios campos que el usuario debe completar para poder crear un SCAMPI nuevo y además debe seleccionar las áreas de proceso sobre las que se realizará el SCAMPI.

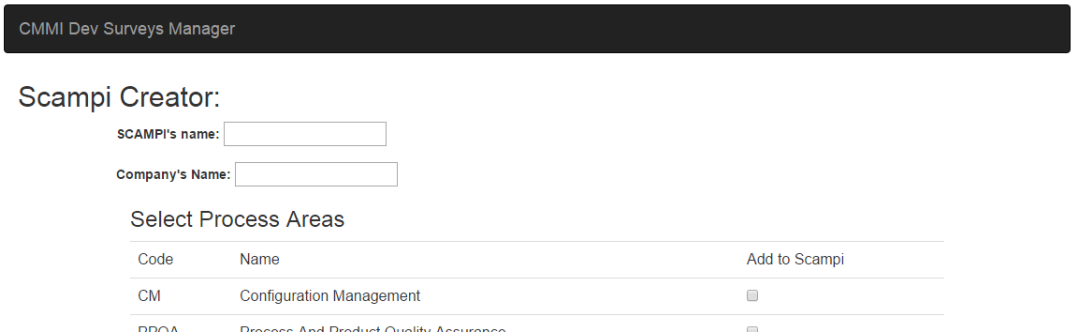

Figura 9: Imagen de la vista de creación de un SCAMPI.

El controlador “createScampiCtrl” contiene tres funciones: “initialize”, que inicializa el estado de la aplicación obteniendo las áreas de procesos y los SCAMPIs almacenados en base de datos mediante los servicios “processAreaSvc” y “scampiSvc”, “toggleSelection”, que controla las áreas de proceso seleccionadas para formar parte del SCAMPI nuevo y el array que contiene esta información, y “createScampi”, que mediante “scampiSvc” crea el nuevo SCAMPI en base de datos.

```
$scope.initialize=function(){
      //Estas funciones nos devuelven una promesa.
      //Las esperamos con el servicio $q de AngularJS.
      var scampisPromise=scampisSvc.getScampis();
      var areasPromise=processAreasSvc.getProcessAreas();
      $q.all([scampisPromise,areasPromise]).then(
            function(responses){
                  $scope.scampis=responses[0];
                  $scope.areas=responses[1];
            });
}
```

Figura 10: Función "initialize" de createScampi.js.

La función "createScampi" renueva el formulario cuando se confirma la creación del SCAMPI y las áreas de proceso seleccionadas se almacenan en un array controlado por la función "toggleSelection". Por otra parte, la creación de un nuevo SCAMPI es un POST a través del servicio "scampiSvc".

La vista también muestra los errores que puedan producirse en la creación de SCAMPIs u obtención de datos desde *back-end*. Por otra parte, las áreas de proceso seleccionadas se colorearán de verde en la vista para mostrar fácilmente las áreas seleccionadas.

- **"/app/features/scampi": Vista de detalle de SCAMPI.**

La vista "scampi" está formada por diferentes elementos y es probablemente la vista más compleja y costosa de hacer puesto que lleva a cabo varios procesados de datos en la parte cliente mientras el SCAMPI se mantiene abierto.

La vista en sí contiene apartados como la información general del SCAMPI (nombre, empresa, áreas de proceso sobre las que se realiza...), contiene el número de encuestas llevadas a cabo y una vista de los resultados parcial o final. Si es parcial, el cliente tiene que llevar a cabo una serie de cálculos porque estos no están guardados en base de datos y si está cerrada muestra los resultados obtenidos, sin embargo esta última funcionalidad de cierre de SCAMPIs no se ha implementado. Las razones se detallan más adelante en este documento.

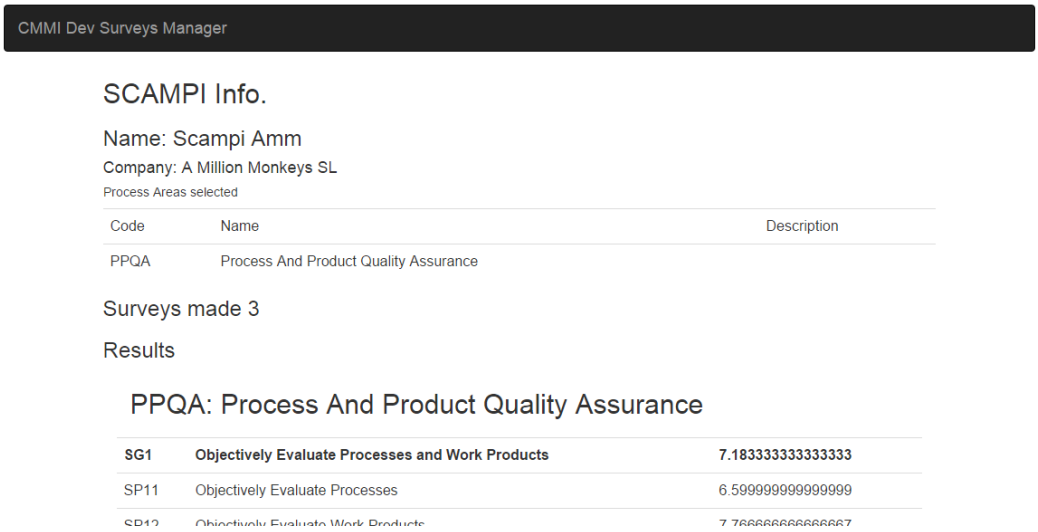

Figura 11 : Imagen de detalle del SCAMPI con resultados obtenidos.

Es este cálculo de resultados el proceso más complicado de llevar a cabo en todo el desarrollo de la parte cliente de la aplicación, puesto que como ya se ha dicho, la elección de un sistema de bases de datos no relacional complicó mucho el acceso a datos parciales como por ejemplo resultados de un SCAMPI, teniendo que crear objetos de gran tamaño. Es por eso que al tener listas de datos dentro de otras listas se complicó el desarrollo principalmente por la navegación entre elementos, sin embargo, terminado este proceso el desarrollo fue casi dado por terminado.

El controlador (que no mostraremos aquí sino en el anexo) contiene diferentes funciones entre las que destacamos "initialize" que establece el estado inicial de la vista obteniendo la información del SCAMPI al que se está accediendo. La función quizá más importante es la de "processResults" y es también la más compleja y complicada de programar(a pesar de que utiliza varias funciones auxiliares).

Su función es la de determinar los resultados del SCAMPI a partir de las diferentes encuestas que se han cumplimentado. Es una tarea compleja debido a lo mencionado antes, la navegación en listas iterativas, y ha supuesto varias tardes de trabajo. Finalmente podemos decir que funciona correctamente y que su cálculo de resultados es correcto. Además, esta función necesita de otras de forma auxiliar como "getResultsFromProcessArea" o "paringResultsProcessAreas" y algunas de búsqueda para simplificar la función inicial.

La función también modifica el documento SCAMPI original con los resultados y una vez cerrado este se modifica a nivel de base de datos aunque debido al correcto funcionamiento que tiene el cliente al finalizar el proyecto y a una duda sobre la modificación de los SCAMPIs en el tiempo, esta función no se ha llevado a cabo en el resultado final. En el caso de que un SCAMPI no pudiera modificarse en el tiempo (posibilidad que se elimina al dar por finalizada un SCAMPI) se añadiría la posibilidad de cerrar un SCAMPI. Por el momento, este caso queda así y su funcionamiento sigue siendo correcto con y sin esa funcionalidad que en algún momento podría añadirse a modo de archivo del caso.

- **"/app/features/scampi/new": Vista de adición de encuesta al SCAMPI seleccionado.**

La vista "new" es la vista que posibilita la adición de una nueva encuesta realizada en el entorno de un SCAMPI seleccionado. Contiene un cuestionario por cada práctica a cumplir en cada meta de cada área de proceso que se está evaluado en un SCAMPI.

Cada pregunta sobre una práctica contiene el nombre de la práctica, su descripción, la pregunta, y los cuadros de respuesta. La práctica se enmarca dentro de una meta y esta meta dentro de un área de proceso, al igual que en la especificación teórica por lo que se responde a cada área de proceso en orden y a sus metas y prácticas de la misma manera.

El controlador "newCtrl" no conlleva tanta complejidad como el anterior. En primer lugar se ejecuta la función "initialize" para iniciar el estado de la vista donde se obtienen los datos referentes al SCAMPI y se obtiene la información sobre las áreas de proceso sobre las que se realiza para poder mostrarlos por pantalla.

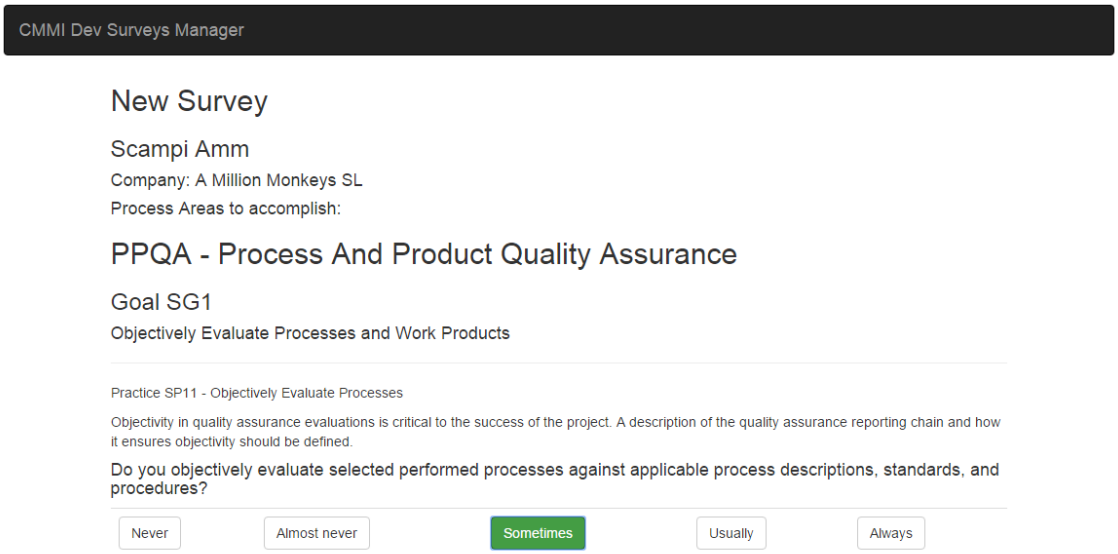

Figura 12: Página de adición de encuesta a una SCAMPI.

También destacar un grupo de funciones ("toggleSelection", "checkResult", "clearResult", "searchByCode") que realizan un check de cada meta y práctica para ver si se han contestado y añadirlas a un objeto "respuesta". De este modo se puede cambiar de respuesta y responderlas de forma aleatoria sin tener que preocuparse del JSON que tendremos que enviar a *back-end*.

- **"/app/features/commons/structure": Vista de la estructura y header de la aplicación.**

Esta vista contiene simplemente la estructura externa de la aplicación a nivel de diseño así como un controlador para el header. Es importante pues así se puede mantener un marco común a toda la aplicación que es la barra que comprende el header. También se definió una directiva para mostrar el header directamente sin tener que copiarlo en el index y dejándolo más limpio todavía.

Servicios

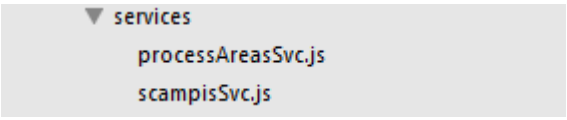

Figura 13: Distribución de la carpeta "services".

- **"app/services/processAreaSvc.js":Servicio de áreas de proceso.**

El servicio "processAreaSvc" es el servicio que controla el estado y las comunicaciones con servidor referentes a las áreas de proceso. El servicio utiliza el resource "processAreaRsc" para las llamadas a la API. Las funciones que componen el servicio son:

- "setProcessAreas": asigna un nuevo valor a las áreas de proceso locales. Cuando se obtienen datos desde *back-end* se llama a esta función para mantener un estado local de la aplicación y no tener que llamar a *back-end* cada vez que un controlador las solicita.
- "getLocalProcessAreas": esta función es un GET de la información de áreas de proceso que hay en local. Es la primera función a la que hay que recurrir para obtener información sobre áreas de proceso.
- "getProcessAreas": esta función es la que proporciona la información de las áreas de proceso desde *back-end*. Lleva a cabo una llamada a *back-end* y devuelve las áreas de proceso además de establecer las áreas de proceso locales.
- "postProcessArea": si se crea una nueva área de proceso, esta es la función que permite añadir el área de proceso en *back-end*.
- "editProcessArea": permite editar un área de proceso y devuelve el área de proceso actualizada.
- "deleteProcessArea": es la función que permite eliminar un área de proceso.

- **"app/services/scampiSvc.js": Servicio de SCAMPIs.**

El servicio "scampiSvc", al igual que el servicio "processAreaSvc", permite mantener un estado local de los SCAMPIs en este caso y además a través de "scampisRsc" se pueden realizar acciones en *back-end* mediante uso de la API. Las funciones que contiene el servicio son:

- "setScampis": permite establecer un nuevo valor para el array de SCAMPIs locales. Al igual que con el servicio de áreas de proceso, permite guardar un estado local y temporal de la aplicación para no tener que llevar a cabo una llamada a *back-end* cada vez que se quiera la información de los SCAMPIs.
- "getLocalScampis": devuelve la información referente a los SCAMPIs guardados en local.
- "getScampis": lleva a cabo una llamada a *back-end* para obtener y devolver la información almacenada en *back-end* sobre los SCAMPIs.
- "postScampi": permite añadir un nuevo SCAMPI en *back-end*.
- "editScampi": permite editar un SCAMPI y devuelve el objeto actualizado.
- "deleteScampi": permite eliminar un SCAMPI en *back-end*.
- "getScampi": permite obtener un SCAMPI en concreto en lugar de obtener todos. Es útil cuando se quiere acceder al detalle de un SCAMPI en concreto.

Resources

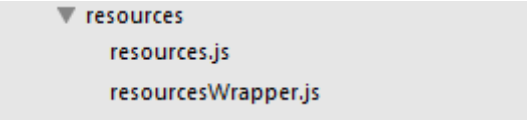

Figura 14: Distribución de la carpeta "resources".

- **"app/resources/resources.js": Servicio de conexión a APIs.**

Un "resource" es en realidad un servicio que tiene AngularJS para conectar con una API REST. Proporciona un *framework* dentro del propio *framework* de AngularJS para que se pueda acceder a la API fácilmente y no se tenga que programar un CRUD para APIs de nuevo cada vez que se programe una nueva aplicación.

Así, se definen dos resources, uno para la base de datos de áreas de proceso y otro para la base de datos de SCAMPIs. El funcionamiento de estos resources es fácil: el servicio define una serie de acciones de manera general (CRUD básico para acceder a la API REST) y en cada resource se puede llevar a cabo una re-escritura o adición de acciones a llevar a cabo, como puede ser añadir elementos a la URI o al querystring de la llamada a API o cambiar un modo de recibir y mandar datos a la API.

En otras palabras, AngularJS proporciona una caja de herramientas con una colección de llaves inglesas que se pueden aumentar o modificar según los tornillos que tengamos que apretar. Un resource funciona definiendo simplemente una URL y una entidad de identificación y puede personalizarse para recibir objetos, arrays, determinar múltiples entidades en la URI... etc.

Además, el servicio es bastante inteligente, por decirlo de alguna manera, y sabe que si hacemos un GET tendrá que introducir datos en la querystring si no se dice lo contrario y si es un POST los parámetros de llamada se añadirán al body de la petición. Se tuvo que definir explícitamente que la id de los objetos se pasaban en forma de entidad URI y nunca de forma querystring.

En el recurso "processAreaRsc" se definieron acciones para obtener áreas y un solo área puesto que en la definición por defecto no dejaba establecer que la respuesta fuera un objeto o un array y se tuvo que definir estas dos acciones de forma personalizada.

En el caso de "scampisRsc" se tuvo el mismo problema y se tuvo que definir acciones personalizadas para acceder a los datos por array o por objeto además de definir otras como la edición de un SCAMPI o su eliminación, acciones que por defecto no terminaban de funcionar.

Señalar que la forma de funcionar de los resources no terminó de cumplir las necesidades que se tenían así que se utilizó un wrapper (agrupador) de estos resources para que pudieran utilizarse con promesas (*Q promises*), muy bien especificadas en AngularJS.

```
angular.module('app').factory("processAreasRsc", ["$resource",
"config", "ResourceWrapper",
    function ($resource, config, ResourceWrapper) {
        var url = config.remoteServiceBaseUrl + '/areas/:_id';
        var customActions = {
            getAreas: {
                  method: 'GET',
                  isArray: true,
                  params: {}
            },
            getSingleAreaByCode: {
                  method: 'GET',
                  isArray: true,
                  params: {
                        listController:'code',
                        docController: '@code'
                  }
            },
            update: {
                  method: 'PUT',
                  isArray:false,
                  params: {
                        _id: '@_id'
                  }
            }
        };
        var resource = $resource(url, {_id: '@_id'}, customActions);
        return new ResourceWrapper(resource);
    }
]);
```

Figura 15: Código de uno de los resources utilizados, "processAreasRsc".

General

- **"app/app.js": Configuración, estado inicial de la aplicación y definición y control de rutas y estados.**

El archivo "app.js" es el archivo inicial de una aplicación AngularJS. Es donde se define la aplicación, las rutas, la configuración y la función de estado inicial de la aplicación. Es, por decirlo así, el punto de partida de cualquier aplicación hecha en AngularJS.

Se definieron las diferentes rutas que se iban a necesitar en la aplicación puesto que se utilizó un servicio propio de AngularJS para utilizar estados, lo que permitió crear vistas con estados heredados o estados múltiples y definir las rutas para cada uno de éstos.

```
$urlRouterProvider.otherwise("/");
$stateProvider.state('app.newSurvey',{
      url:"^/scampi/:id/new",
      templateUrl:"/app/features/scampi/new/new.html",
      controller:"newCtrl"
});
```

Figura 16: Código de definición de una ruta en "app.js".

También se usó para definir la configuración de la aplicación que era un objeto global que contiene la URL remota del servidor, para poder acceder desde los resources que hemos mencionado en este mismo capítulo.

4.5. Back-end

El desarrollo de *back-end* comprendió un 30% más o menos del total de tiempo de desarrollo. Fue el primer elemento que se desarrolló puesto que sin un *back-end* trabajado y estable habría sido imposible desarrollar un *back-end* en condiciones.

El *back-end* se apoya tecnológicamente en NodeJS y su módulo Express y en MongoDB. En esencia, el *back-end* es un servidor con una API definida a la que se puede acceder de manera pública y que permite acceder a los recursos almacenados en los documentos de las bases de datos en MongoDB.

El desarrollo fue complicado por la inexperiencia desarrollando una API y por el uso de diferentes capas para aumentar el nivel de reutilización de código y aislamiento de puntos de error, esto quiere decir que en el caso de que tuviéramos un problema grave, el uso de diferentes capas de aplicación permite pivotar y cambiar cosas de una manera más sencilla que usando una sola capa única. Así, tenemos diferentes capas:

1. **API**: La primera capa es una capa de acceso a los recursos, donde se establecen las rutas a usar para acceder a la API y algunos detalles de configuración. Es la capa más externa y más sencilla de programar pero más complicada de plantear.
2. **Servicios**: Los servicios componen un middleware entre el routing de API (definición de rutas) y lo modelos de bases de datos. Esto permite cambiar el comportamiento o tratamiento de datos antes de devolverlos por la API. Es una capa de tratamiento de datos y ejecución de funciones dependiendo de los datos que llegan desde *back-end*.
3. **Modelos**: Es la capa más profunda del *back-end*. Está por encima de la capa de datos y realiza las tareas de acceso a datos explícitamente, es decir, es el que realiza las consultas y devuelve a los servicios los datos requeridos.

4.5.1. Diseño de bases de datos

La base de datos es de tipo documental no-relacional, usando MongoDB como plataforma. La base de datos en cuestión se vio en la carrera y puesto que es sencillo

encontrar bases de datos relacionales y fueron tratadas de manera muy extensa en la carrera, su uso era más orientado a la investigación y la prueba.

El uso de MongoDB aportó multitud de beneficios a la aplicación como el isomorfismo global que tiene en todas las capas, el uso único de JavaScript como lenguaje de programación o la falta de esquema en base de datos, pero el tratamiento de datos en la parte cliente supuso un reto que se superó a lo largo del desarrollo.

En primer lugar, se tuvo que diseñar objetos que funcionaran en MongoDB como documentos y no como tablas, puesto que de haber sido así el tratamiento de datos en cliente habría sido más sencillo. MongoDB carece de esquema, lo que fue una ventaja para llevar a cabo documentos provenientes de formularios de talla indefinida pero a la hora de acceder a los datos fue algo más complicado complicado. Seguramente, este trabajo de tratamiento de datos no fue tan complicado como utilizar formularios de talla indefinida con bases de datos relacionales, lo que es un beneficio significativo.

Se planificaron documentos con una gran pero necesaria longitud, pues MongoDB no admite relaciones y el acceso a datos individuales se hizo en cliente mediante la navegación en múltiples niveles de iteración.

Hay dos colecciones principales, “scampi” y “processArea” que contienen los SCAMPIs realizados a diferentes empresas y las áreas de proceso contenidas en *back-end*, con la información sobre metas, prácticas y demás. El problema con los segundos es que hubo que determinar que un área de proceso contiene metas y las metas contienen prácticas, lo cual hizo que un documento de SCAMPI contuviera un objeto del mismo tipo. Al final se tenían listas de listas en ambos documentos.

Al final el diseño de los documentos quedaron de esta manera:

Documento área de proceso:

```
{
	"name":"Process And Product Quality Assurance",
	"code":"PPQA","purpose":"The purpose of the project.",
	"ml":"2",
	"specificGoals:[
		{
			"code":"SG1",
			"name":"Objectively Evaluate Processes",
			"specificPractices":[
				{
					"name":" Evaluate Processes",
					"code":"SP11",
					"question":"Question to answer…",
					"shortDescription":"Short",
					"description":"Large description"
				}
			],
			"shortDescription":"Short description"
		}
	],
	"_id":"55c3821b3488fd9104ffee15"
}
```

Figura 17: Ejemplo de documento de área de proceso almacenado en base de datos.

Documento SCAMPI:


```
{
	"name":"Scampi Amm",
	"firmName":"A Million Monkeys SL",
	"description":"",
	"processAreasSelected":["PPQA"],
	"surveys":[
		{
			"id":1438941098711,
			"res":[
				{
					"code":"PPQA",
					"_id":"55c3821b3488fd9104ffee15",
					"results":[
						{
							"code":"SG1",
							"practices":[
								{
									"code":"SP11",
									"result":6.6
								},
								{
									"code":"SP12",
									"result":3.3
								}
							]
						}
					]
				}
			]
		}
	],
	"finished":false,
	"_id":"55c47f988b9735d2131b2ad9"
}
```


Figura 18: Ejemplo de documento de SCAMPI almacenado en base de datos.

4.5.2. Desarrollo

El desarrollo del *back-end* fue la parte de desarrollo que menos tiempo llevó, cerca de un tercio del desarrollo total, y se siguió un desarrollo de las capas desde la capa más interna a la más externa, comenzando primeramente con los modelos, pasando por servicios y finalmente con la página de configuración de la API y el servidor.

Los modelos y servicios no difieren demasiado entre ellos mismos, simplemente el tipo de objeto que se crea en los modelos y los diferentes tratamientos de datos o funciones tras analizar las promesas y cómo se devuelve la información a la capa de la API.

Modelos

Los modelos son la parte más baja o interna del servidor por encima de la capa de base de datos. Es aquella capa que lleva a cabo las llamadas a base de datos explícitamente. En este caso tenemos que llevar a cabo consultas a MongoDB, las cuales se llevan a cabo en lenguaje JavaScript y son sencillas de realizar. Por la similitud entre los diferentes modelos, se utilizará como ejemplo una función de uno de ellos aunque en este documento puede consultar el código completo del servicio. A continuación se puede observar una de las funciones que contienen los modelos así como la creación del objeto.

```
function Scampi() {
    if (!(this instanceof Scampi)) {
            return new Scampi();
      }
    // requiere mongodb
    var mongo = require('mongodb');
    // conectamos a nuestra base de datos local con monk
    //necesitamos monk para realizar tareas CRUD de forma sencilla
    var db = require('monk')('mongodb://localhost:27017/cmmi');
    // obtenemos una referencia a la colección de documentos SCAMPI
    this.Scampi = db.get('Scampis');
};

 // Obtenemos una lista con todos los elementos de la colección
Scampi.prototype.findAll = function(success, error) {
    this.Scampi.find({},{},response(success, error));
};
```

Figura 19: Ejemplo de modelo.

Podemos ver que creamos una clase de tipo “Scampi” (siguiendo un paradigma orientado a objetos) y que la clase contiene las funciones para modificar o consultar la base de datos explícitamente. Tenemos funciones de consulta como “findById” pasando la Id del objeto como parámetro, o “findAll” (presente en el código mostrado anteriormente) que devuelve todos los documentos de la colección. Hay que decir que en la creación de una instancia de la clase se define la conexión a base de datos y a la colección contra la que se va a llevar a cabo las consultas o modificaciones.

También se tienen funciones de creación o modificación, “create” y “update”, y también función de eliminación de documentos, “remove”. Además de funciones típicas de un sistema CRUD, se tienen una función especial, “response”, que trata de forma básica las respuestas que llegan de base de datos y lleva a cabo el callback de las funciones que se pasan como parámetros en el nivel superior, la capa de servicios, que se verá a continuación.

Además, como se puede ver en el código, cuando se lleva a cabo una llamada de una función, se lleva a cabo una llamada explícita a base de datos como podría ser una query en SQL. Por ejemplo, en “findAll” se lleva a cabo una llamada a la función “find” de MongoDB, y esta se realiza de forma correcta porque en la creación de la instancia se establece la conexión a base de datos, de ahí que se llame de la forma “this.Scampi.find(...)”. De esta forma se puede aumentar el número de funciones a este

nivel y que en niveles superiores no haya que implementar apenas nada y se pueda utilizar sin mayor problema.

Servicios

La siguiente capa que se desarrolló fue la capa de servicios, que es una capa de middleware, por decirlo de alguna manera, que lleva a cabo un tratamiento de las peticiones que llegan desde la capa más externa, la de API, y comunica las peticiones con sus correspondientes parámetros a la capa de modelos. Cuando la petición se ha completado, la capa de modelos devuelve a servicios la respuesta y se ejecuta el callback que se le ha pasado por servicios. En servicios se trata la respuesta, y según el callback la respuesta a la capa de la API es una u otra…

Cuando llega una petición a servicios, se puede definir el tratamiento que se hará de la petición, modificaciones en los parámetros que transporta la API y decidir a qué función de la capa de modelos se llama. Para simplificar la explicación de esta capa y como ya se ha hecho previamente en este capítulo, este documento se centrará en uno de los servicios debido a la similitud de los diferentes servicios. A continuación, una parte del código del servicio de SCAMPIs (el servicio completo lo puede encontrar en el anexo de este documento).

```
var exports=module.exports;
var db = require('monk')('mongodb://localhost:27017/cmmi');
var scampi = require('../modelsSvcs/scampi.js')();

exports.findAll = function(req, res) {
    // lógica para manejar GET /scampi
    var ok = function(doc) {
          res.json(doc);
    };
    var err = function(err) {
          res.send(404);
    };
    scampi.findAll(ok, err);
};
```

Figura 20: Trozo de código de scampiSvc.js en back-end.

Como se puede ver, se tienen diferentes elementos, entre ellos una conexión a base de datos a través de monk, un *framework* de conexiones a bases de datos que facilita el CRUD. De nuevo, se definen una serie de funciones, pero esta vez no una clase, sino que simplemente se ofrecen todas las funciones y el modelo referenciado para poder utilizarlo libremente en el nivel inmediatamente superior.

Cuando en API se llama a una de las funciones de servicios, el servicio crea una instancia del modelo que se está necesitando con todas las funciones definidas en él, y según la función que se requiere se llama a una u otra función del modelo y se lleva a cabo un

tratamiento de la petición y la respuesta mediante parámetros de entrada y callbacks respectivamente.

En la petición o "request" se puede llevar a cabo una transformación que se necesite, como obtener el parámetro de una petición para pasarlo a una función del modelo en el caso de la petición o llamar a una función de éxito o error en el caso de que la respuesta a la petición desde modelo sea satisfactoria o no. Así, se puede simplificar, modificar o tratar la comunicación entre la API y la base de datos de forma sencilla.

Por ejemplo, en modelo "finById" necesita de una id como parámetro, y en el servicio lo que se hace es obtener este parámetro a través de la petición misma a la API. Se pasa en la petición la ID y en *back-end* se recoge y se utiliza, pasando y devolviendo un JSON a la API. Todo este proceso se realiza mediante promesas.

API Endpoints y configuración

```
var scampiSvc = require('./services/scampiSvc.js');
router.get('/scampis', scampiSvc.findAll);
router.get('/scampis/:id', scampiSvc.findById);
router.post('/scampis', scampiSvc.create);
router.delete('/scampis/:id', scampiSvc.deleteById);
router.put('/scampis/:id', scampiSvc.update);
```

Figura 21: Configuración de rutas de los SCAMPIs.

Por último pero no menos importante, se desarrolló la configuración del servidor y la API. Es en esta parte cuando se definen cómo se inicia el servidor, que parámetros se definen para utilizar la API y qué rutas se asignan a los recursos contenidos en servicios. Puede ver la configuración de las rutas de la API en la Figura 22.

En primer lugar, se definen los recursos de NodeJS que se iban a necesitar, como es Express y su router, también se configuró el HEADER de las peticiones y respuestas de la API y una ruta básica para ésta así como el tipo de archivo que se iban a utilizar.

Más tarde se definen una serie de rutas para cada uno de los recursos que se iban a querer utilizar desde el cliente con sus correspondientes parámetros opcionales (que contienen un ":" delante) y siempre asignando una ruta a un recurso. Para ello, se inicializa un objeto por cada servicio que se fuera a utilizar y se personalizó.

Por ejemplo, la ruta "/api/areas/323787827382" dará el objeto de la colección "areas" con id "323787827382" puesto que la API define que si llega algo con "/api" de inicio puede usar las rutas de la API, si además contiene "/areas/:id" (siendo ":id" cualquier cosa) entonces se crea una instancia del servicio que utiliza áreas y se accede a la función de esta área (y consecuente del modelo áreas) con ese parámetro. Es, en esencia, una cadena en la que las peticiones API viajan capa por capa al igual que las respuestas.

4.6. Página de gestión de las bases de datos

Por último, destacar una tarea de desarrollo que tuvo que realizarse de manera improvisada por las dificultades a la hora de crear datos en las bases de datos. La dificultad de usar la terminal para introducir objetos complejos en MongoDB planteó la cuestión de si una página de gestión de las bases de datos era importante y, puesto que el coste de crear un área de proceso era complicado, se decidió desarrollar una página de gestión de las bases de datos.

En esta página, el usuario puede consultar y modificar los documentos en la base de datos que contiene la información acerca de las áreas de proceso como es el nombre, descripción, metas, prácticas y hasta la pregunta que aparece en pantalla. Es, en esencia, una página para facilitar la introducción de datos complejos como es la información que requiere CMMI en bases de datos. Además se puede añadir y eliminar áreas de proceso.

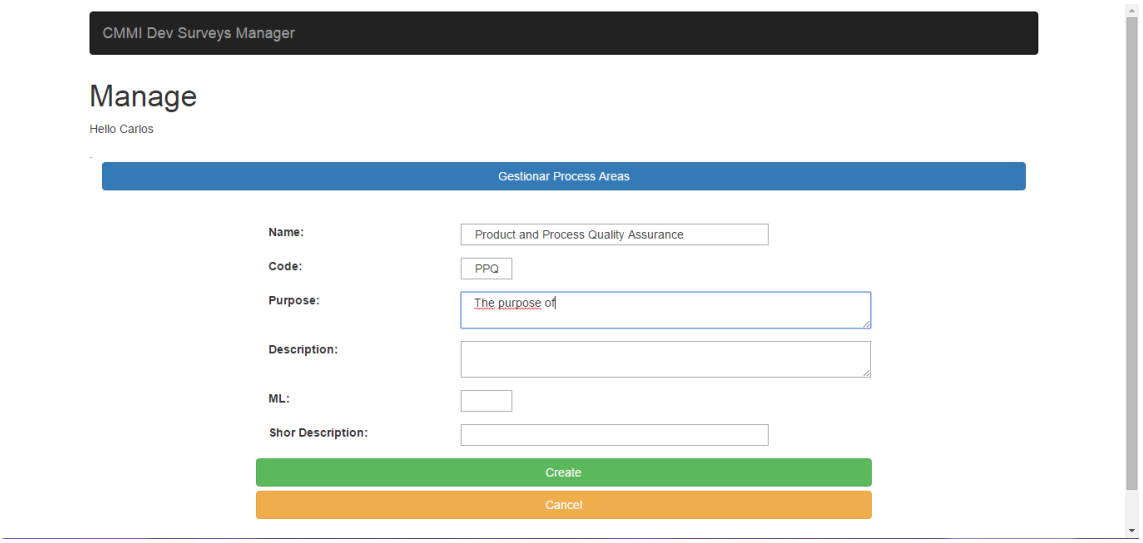

Figura 22: Imagen de la página de gestión de bases de datos.

Para desarrollarla, se creó un nuevo apartado en la aplicación de AngularJS normal, en el cliente, y se utilizaron los resources y los servicios ya definidos con anterioridad. Fue un acierto y una prueba de que con una buena arquitectura por capas la reutilización de código es efectiva y facilita el proceso de añadir funcionalidades a una aplicación. Los recursos que daba AngularJS y Bootstrap facilitaron el desarrollo pero, al no estar previsto en el desarrollo normal, alargó todo el tiempo de desarrollo.

5. Estudio económico

Al finalizar el proyecto, es necesario hacer una retrospectiva del proyecto y ver cuál ha sido el coste del proyecto, qué se ha obtenido, qué se puede obtener y ante todo, ver si ha sido una inversión beneficiosa (aunque esas conclusiones finales se tratan en el apartado "Conclusiones" de este documento). Para poder determinar el coste del proyecto, se tiene que hacer un análisis de qué estaba previsto en coste temporal y monetario y cual ha sido el coste final.

5.1. Coste temporal

El coste temporal del proyecto comprende el número de jornadas que se han tenido desarrollando el proyecto desde el momento que se comienza a leer documentación y diseñar el proyecto hasta el momento en el que se da por finalizada la memoria. Se puede observar que hay tareas que gastan mucho más tiempo que otras, pero puesto que no se dedicaba días exclusivamente a un único propósito, es complicado dedicar un análisis exhaustivo por horas.

En una planificación inicial, las tareas estaban así definidas:

- Preparar información: 4 semanas, 20 días.
- Diseño: 1 semana, 5 días.
- Desarrollo: 4 semanas, 20 días.
- *Testing*: 4 semanas, 20 días.
- *Deployment*: 3 días.
- Memoria: 8 semanas, 40 días (0 semanas de dedicación exclusiva).

Y la dedicación final fue:

- Preparar información: 4 semanas, 20 días.
- Diseño: 1 semana, 5 días.
- Desarrollo: 8 semanas, 40 días.
- *Testing*: 4'2 semanas, 21 días.
- *Deployment*: 4 días.
- Memoria: 4 semanas, 20 días (4 semanas de dedicación exclusiva).

Coste temporal inicial.

Inicio: 27 de Abril de 2015.

Final: 1 de Agosto de 2015.

Tiempo total transcurrido: 13’6 semanas, 68 días.

Coste temporal final.

Inicio: 27 de Abril de 2015.

Final: 28 de Agosto de 2015.

Tiempo total transcurrido: 18 semanas, 90 días.

5.2. Coste monetario

El coste monetario comprende todos los gastos realizados para obtener o contratar recursos y servicios para poder llevar a cabo el proyecto. Comprenden gastos como el personal utilizado y los medios de producción. En el apartado de recursos, en el caso de que sea un recurso humano o de uso temporal, este se calcula multiplicando el número de horas por el coste por hora del recurso. Puesto que solo ha habido un desarrollador, el cálculo del precio por hora trabajada se obtiene de las ganancias por hora que tiene a fecha de comienzo del proyecto.

Coste monetario inicial.

Recurso: Carlos Jiménez (desarrollador de software).

Horas: 104 días, dedicando 3h. al día, 312h.

Precio por hora: 5€/h.

Total: 1560€.

Recurso: Portátil Asus F550D

Precio: 399€.

Total: 399€.

Recurso: Aplicación Sublime Text.

Precio: 49€.

Total: 49€.

Recurso: Aplicación Microsoft Office 365 personal.

Precio: 37’57€.

Total: 37,57€.

Recurso: Servidor FTPit en Boston.

Horas: 17 semanas, 2856h.

Precio por hora: 0,00137€.

Total: 3,91€.

Total coste monetario previsto: 2049,48€.

Coste monetario final.

Recurso: Carlos Jiménez (desarrollador de software).

Horas: 130 días, 390h.

Precio por hora: 5€/h.

Total: 1950€.

Recurso: Portátil Asus F550D

Precio: 399€.

Total: 399€.

Recurso: Aplicación Sublime Text.

Precio: 49€.

Total: 49€.

Recurso: Aplicación Microsoft Office 365 personal.

Precio: 37’57€.

Total: 37,57€.

Recurso: Servidor FTPit en Boston.

Horas: 17 semanas, 2856h.

Precio por hora: 0,00137€.

Total: 3,91€.

Total coste monetario real: 2439,48€.

5.3. Valoración de costes

Como era de esperar, el mayor coste monetario ha sido el del propio recurso humano pues la cantidad de horas invertidas ha sido inmenso, y, suponiendo que el precio por hora es el mismo que el del trabajo que realizaba en el momento de comenzar el proyecto, se puede afirmar que un recurso humano es de largo el recurso más costoso en un proyecto como este.

Además, teniendo la comparación de los proyectos antes y después de la realización del mismo, se puede observar como 4 semanas más de desarrollo se convierten en una diferencia del 25% respecto al coste del proyecto inicial cuando 4 semanas implica un 22'22% de diferencia en el proyecto final.

De haber demorado más el plazo de entrega, los costes habrían seguido aumentando, a razón de 75€/semana. Como ya se ha dicho dicho, el retraso se produce en la fase de Desarrollo, que multiplica por dos el tiempo estimado y es la tarea más cara de todo el proyecto con un coste de 600€.

6. Validación

Tras realizar el desarrollo técnico completo del proyecto, una de las tareas era llevar a cabo un despliegue y prueba del proyecto en una empresa de software para verificar que el software funcionaba y además contemplar un escenario real de acción y consideración de CMMI-DEV. La empresa elegida fue “A Million Monkeys SL”.

6.1. Metodología

Como la empresa se trataba de una PYME y CMMI-DEV está orientado a empresas grandes, se decidió que ya que el tipo de SCAMPI es B y permite una granularidad a nivel de área de proceso, decidimos realizarla sobre una única área de proceso, PPQA “Process and Product Quality Assurance” que trata el control de calidad sobre el proceso y el producto dentro de la empresa.

Esta área de proceso se compone de prácticas como el registros sobre errores, cuestiones de calidad, comunicación con los superiores…etc. Trata algunas prácticas que se realizan de forma más o menos general en la mayoría de empresas de tamaño mediano que han nacido en los últimos años.

La encuesta que se les pasó es la encuesta mostrada antes en este documento, en la sección 3.4.4.

6.2. Proceso

Se pactó un día con la empresa, y ese día durante el horario laboral se realizó entrevistas individuales a cada uno de los trabajadores pertenecientes al área de desarrollo, terminando el proceso en 1 hora aproximadamente. Tras cada una de las entrevistas, se comentó con el entrevistado aspectos acerca del uso de buenas prácticas y el proceso en sí. Al terminar, se comunicó los resultados al CTO de la empresa y se dio por terminada el SCAMPI.

Durante el proceso, hubo confusiones sobre algunas de las preguntas, y el software sirvió de mucha ayuda para poder realizar el SCAMPI de manera correcta. Las descripciones en la pantalla de realización del SCAMPI, sugerencias y la propia pregunta predefinida hizo que aunque al preparar el SCAMPI ya se hubieran visto los puntos complicados de entender, las dificultades de comprensión por parte del personal de algunos elementos se viera minimizada.

6.3. Resultados

6.3.1. Del SCAMPI realizado en la empresa.

Los resultados de la evaluación fueron positivos y se notificó las prácticas que había que mejorar en la empresa y cuales se estaban llevando a cabo de manera correcta. Al comunicar al director de tecnología de la empresa el resultado del SCAMPI, este se mostró dispuesto a mejorar las prácticas evaluadas y valorar la idea de llevar a cabo SCAMPI de tipo B sobre otras áreas de proceso.

PPQA				
Meta	SG1		SG2	
E1	6.6	3.3	10	5
E2	6.6	3.3	10	6.6
E3	6.6	10	10	6.6
E4	6.6	10	10	6.6
Media Prácticas	6.6	6.65	10	7.85
Media Metas	6.625		8.925	
Media global	7.775			

6.3.2. Del uso de la herramienta

En el caso del uso de la herramienta, esta prueba fue una demostración de las cosas que se pueden mejorar, y de posibles formas de hacer más sencillo para el encuestador explicar las preguntas y realizar un SCAMPI de manera correcta.

CMMI-DEV utiliza elementos del ámbito empresarial y de gestión que pueden resultar de difícil comprensión por parte de un colectivo de trabajadores técnicos y a veces resulta difícil explicar estos conceptos. Cuesta llevar a cabo una explicación más o menos correcta pero no complicada para que se comprendiera por lo que el uso de elementos visuales en las explicaciones, ejemplos y demás puede ser una forma de mejorar la herramienta y facilitar la realización de evaluaciones SCAMPI.

La validación ha permitido:

- Descubrir puntos de mejora en la herramienta.
- Verificar el funcionamiento de la herramienta.
- Explorar y analizar posibles usos futuros de la herramienta.
- Recibir un feedback por parte de los encuestados sobre el diseño de las preguntas y ayudas mostrado en la herramienta.

En definitiva, la verificación en la empresa no solo ha servido para comprobar que la herramienta funciona y que es posible llevar a cabo una evaluación SCAMPI con ella,

sino que ha proporcionado un conjunto de puntos en los que mejorarla y seguir actualizándola para crear un producto o servicio más atractivo y competitivo.

7. Conclusiones

7.1. CMMI-DEV

CMMI-DEV es un marco de mejora de procesos software realmente interesante y exigente pero que aporta grandes beneficios. Cuando se realiza la validación en la empresa, aparece la dificultad de llevar a cabo las prácticas contenidas en el modelo con una empresa de menos de 50 trabajadores, pues para cumplirlas de forma exacta al modelo debería haber trabajadores de administración y calidad controlando todo lo referente a registros, control y, en general, cumplimiento de la especificación de CMMI.

Es una certificación exigente, pero es una de las certificaciones más buscadas por las empresas hoy en día. Quizás en un afán más de cumplimiento que de realmente aprovechar las oportunidades que CMMI otorga, las empresas afrontan una certificación de estas características sin darse cuenta a lo que se enfrentan, más las empresas sin un departamento de sistemas de la información trabajado, es por eso que esta herramienta gana en utilidad y necesidad por parte de estas empresas.

CMMI aporta valores tan necesarios en una empresa como son la eficacia y la eficiencia de sus procesos o la satisfacción de los trabajadores en su puesto, lo que a la larga significa más competitividad y menos riesgo de fracaso en nuestra empresa. Teniendo un buen producto, si no se es competitivo, se puede perder todo. CMMI es el medio que puede aportarnos ese valor que tantas empresas necesitan.

A lo largo del proyecto y de la validación, se ha visto un conjunto de problemas de comprensión que ha costado cierto esfuerzo resolver antes y durante el proceso y tendrá una dificultad mayor si el SCAMPI es conducido por alguien que no sepa demasiado de términos como procesos software, procesos de negocio y buenas prácticas en el desarrollo. El entrenamiento del personal que conduzca el SCAMPI es casi necesario para que se pueda llevar de forma correcta.

Dentro de estos problemas de comprensión hay que realizar un esfuerzo por parte de los desarrolladores para facilitar la comprensión de las diferentes prácticas a llevar a cabo con ejemplos, mejores explicaciones y, en definitiva, facilitar todo mucho más para que una certificación así no se convierta en algo imposible de alcanzar por su exigencia o complicación, ya que la no comprensión del marco de trabajo implica el abandono de este. Es el riesgo que hay que minimizar.

Por otro lado, la exigencia de CMMI en su certificación hace hincapié en la necesidad de herramientas para llevar a cabo SCAMPIs de tipo B y C como esta. Que una empresa se lance a obtener una certificación como esta sin saber si la va a obtener o no es un riesgo que muchas veces no puede correr y quizás es incluso pronto para llevar a cabo una auditoría. Esta herramienta tiene la utilidad de decirnos cómo están nuestros procesos y qué hay que hacer para mejorarla. Quizás tiene una gran utilidad aun cuando no se esté preparando la empresa para una certificación de CMMI-DEV.

7.2. Tecnología utilizada

Respecto a las tecnologías y elecciones sobre éstas hechas a lo largo del proyecto, hay que destacar las elecciones de MongoDB como motor de base de datos, AngularJS como *framework* central para la aplicación del cliente y NodeJS como *framework* para construir la API REST.

En cuanto a MongoDB, supuso un reto diseñar y gestionar una base de datos no-SQL. MongoDB aportó multitud de ventajas como la falta de esquema, el isomorfismo completo en la aplicación o el uso de un lenguaje conocido (JavaScript).

Por otro lado, hubo que llevar a cabo trabajo extra en las tareas de desarrollo de los servicios de tratamiento de datos, pero aun así habría que reflexionar si este tiempo no se habría invertido, en el caso de utilizar una base de datos relacional, en solventar el problema de los formularios con campos indefinidos en SQL que en MongoDB no existe.

Siguiendo con las tecnologías empleadas, AngularJS ha sido la herramienta que mejor se ha adaptado a las necesidades del proyecto. Ha proporcionado unos recursos que de haberse tenido que programar desde cero, habrían supuesto un retraso de varias semanas. Aprender a utilizar todo el ecosistema que nos aporta AngularJS ha sido otro reto que se ha conseguido superar a lo largo del proyecto mediante lecturas de documentación, pruebas, ensayos…etc.

Ha sido un reto también difícil, pero de nuevo, los beneficios que ha aportado al proyecto han compensado con creces el tiempo empleado. Se trata además de una tecnología puntera apoyada en parte por Google, por lo que tiene mucho camino por delante.

Volviendo a las tecnologías empleadas en el *back-end*, realizar casi desde cero una API REST en NodeJS siguiendo una arquitectura por capas también ha supuesto un proceso desafiante. Ha sido complejo pero gratificante construirla con todos sus elementos, así como volver a utilizar tecnologías y conceptos vistos a lo largo de la carrera (promesas).

En cuanto al diseño de la aplicación, se diseñó siguiendo una arquitectura por capas para facilitar la detección de fallos y modificación de funcionalidades sin tener que modificar código compartido, además de facilitar la explotación de recursos remotos como es el uso de la API. De no haber seguido esta arquitectura, no se habría podido llevar a cabo la página de gestión de bases de datos de manera tan rápida.

El uso además de dos servidores ha permitido separar cliente y servidor de manera total, permitiendo más flexibilidad y seguridad en caso de fallos por denegación de servicio. Así, el servidor puede soportar muchos más usuarios, teniendo que encargarse únicamente del *back-end*. De tener que encargarse de la aplicación entera, el número de usuarios conectados simultáneos sería mucho menor. Es un diseño serio y profesional.

En definitiva, se han explotado tecnologías nuevas y tecnologías que se habían visto brevemente en la carrera, por lo que el proyecto ha supuesto un medio de descubrimiento y exploración de algunas de las tecnologías más demandadas hoy en día por el mercado de las tecnologías de la información. Por otra parte, el diseño ha permitido una mayor fluidez en el desarrollo y ha sentado las bases de un servicio serio y profesional que puede seguir creciendo.

7.3. Futuro

Hoy en día, el camino que las tecnologías de la información están siguiendo para continuar creciendo y evolucionar es el uso de la computación en la nube (*cloud computing*) y explotación de servicios e infraestructuras remotos (*PaaS, SaaS*), por lo que se puede decir que el proyecto que se ha desarrollado ha seguido este camino.

Uno de los ejemplos que se tienen en el proyecto, es el uso de dos servidores diferentes. Mediante el uso de dos servidores, se pueden ofrecer dos tipos de productos o servicios al cliente. Por un lado, se puede personalizar el producto e implantar la herramienta en la empresa del cliente y, por otro lado, se puede ofrecer el uso del *back-end* y de un cliente centralizado como un servicio más económico y, evidentemente, menos personalizado y limitado. Son dos fuentes de ingreso con una misma arquitectura y un producto que se puede instanciar y modificar para crear nuevos productos ya que la API puede alojar nuevas certificaciones y toda la infraestructura ya estaría preparada.

Estos beneficios vienen dados por el diseño que se realizó de la herramienta, poniendo énfasis en una arquitectura por capas y uso de una API, que permiten modificar el contenido del producto o servicio sin alterar el software base que compone la herramienta.

Habría que discutir el hecho de usar un *PaaS* como medio para instaurar el *back-end* y así poder tener un servicio más sencillo de mantener y manejar, pudiendo centrar todos los esfuerzos en los servicios que se prestarían y no en el mantenimiento de los servidores.

De continuar con el proyecto, habría que plantear el desarrollo de un sistema de seguridad con *tokens* o contraseñas temporales asociadas a niveles de acceso a datos, puesto que una API pública sin seguridad es peligroso.

Algunos de los caminos a seguir con este proyecto son:

1. Seguridad: sería interesante ofrecer una capa de seguridad mediante usuario y contraseña para acceder a la herramienta y *tokens* entre servidor y cliente usando por ejemplo “OAuth”.
2. Otras certificaciones: con la arquitectura definida, el grueso de los cambios podría centrarse en añadir nuevo contenido al producto o servicio. Quizás añadir otras certificaciones o tipos de encuestas es un camino a seguir ya que con la arquitectura hecha, solo se tendrían que definir nuevos servicios y tratamientos de datos además de entidades en base de datos.
3. Migración a SQL u otro paradigma relacional: sería interesante también poder ver qué tanta dificultad habría tenido el tratamiento de datos en cliente siguiendo un paradigma de datos relacional.
4. Uso de un *PaaS*: los *PaaS* están creciendo en uso y demanda y, en un futuro, si este proyecto se quisiera convertir en un servicio, sería interesante alojar todo el *back-end* en un *PaaS* para no tener que seguir haciendo tareas de

mantenimiento, escalabilidad, balance de carga…etc. podría ser una forma de establecer las bases de un servicio funcional y comercial.

7.4. Tras la finalización

La palabra que mejor define el proyecto es satisfacción. Ha sido una experiencia totalmente enriquecedora y a veces desafiante que en ocasiones ha asombrado al observar la gran cantidad de elementos teóricos y técnicos que se estaban tratando, pero al final del proyecto la sensación que queda es gratificante.

Durante el proyecto se han solventado varios retos y, sobretodo, aprendido de ellos. Hemos tenido que leer una documentación de gran extensión sobre un conjunto de buenas prácticas complejo y exigente que ya habíamos visto durante la carrera pero que al profundizar sobre él se ha confirmado como algo de gran utilidad para las empresas. Se han planteado cuestiones sobre arquitectura de sistemas, manejo de servidores, creación y gestión de APIs, diseño web, desarrollo web…etc. Ha sido un proyecto muy complejo y muy completo.

Ya para finalizar este documento, podemos decir que la herramienta para realizar SCAMPIs siguiendo el modelo de CMMI-DEV ha sido un proyecto complejo, difícil, a veces desafiante, pero un trabajo de fin de grado completo técnica y teóricamente que nos deja muy satisfechos por el tiempo empleado.

8. Anexo

8.1. Scripts Cliente

newCtrl.js

```
(function () {
	'use strict';
	var controllerId = 'newCtrl';
	angular.module('app').controller(controllerId, ['$scope',
"scampisSvc", "$state", "processAreasSvc", newCtrl]);
	function newCtrl($scope, scampisSvc, $state, processAreasSvc) {
		$scope.scampi=null;
		$scope.scampiProcessAreas=[];
		$scope.processAreas=[];
		$scope.selectedPa=[];
		$scope.newSurvey={};
		$scope.newSurvey.id=null;
		$scope.newSurvey.res=[];
		var practicesAux=[];
		var goalsAux=[];
		$scope.initialize=function(){
			var id=$state.params.id;
			scampisSvc.getScampi(id).then(function(response){
				$scope.scampi=response;

	$scope.selectedPa=response.processAreasSelected;

	processAreasSvc.getProcessAreas({}).then(function(response){
				$scope.processAreas=response;
				for(var i=0; i<$scope.selectedPa.length;
i++){
					for(var j=0;
j<$scope.processAreas.length; j++){

	if($scope.selectedPa[i]==$scope.processAreas[j].code)

	$scope.scampiProcessAreas.push($scope.processAreas[j]);
					}
				}
				var results=[];
				for(var i=0;
i<$scope.scampiProcessAreas.length; i++){
					var
resPa=extractGoals($scope.scampiProcessAreas[i].specificGoals);
					var paSurvey={
						code:
$scope.scampiProcessAreas[i].code,
						_id:
$scope.scampiProcessAreas[i]._id,
						results: resPa
					};
					results.push(paSurvey);
```

```
                        }
                        $scope.newSurvey.res=results;
                        var date=new Date();
                        var time=date.getTime();
                        $scope.newSurvey.id=time;
                  });
            });
      }
      function extractPractices(prArray){
            var practicesSurvey=[];
            for(var i=0; i<prArray.length; i++){
                  var      practice={code:      prArray[i].code,
result:null};
                  if(practicesSurvey.indexOf(practice)<0)
                        practicesSurvey.push(practice);
            }
            return practicesSurvey;
      }
      function extractGoals(glArray){
            var goalsSurvey=[];
            for(var i=0; i<glArray.length; i++){
                  var
specificPractices=extractPractices(glArray[i].specificPractices);
                  var   goal={code:   glArray[i].code,   practices:
specificPractices};
                  if(goalsSurvey.indexOf(goal)<0)
                        goalsSurvey.push(goal);
            }
            return goalsSurvey;
      }
      $scope.toggleSelection=function(pa, sg, sp, newResult){
            var    paId=$scope.searchByCode($scope.newSurvey.res,
pa);
            var goals=$scope.newSurvey.res[paId].results;
            var sgId=$scope.searchByCode(goals, sg);
            var
practs=$scope.newSurvey.res[paId].results[sgId].practices;
            var spId=$scope.searchByCode(practs, sp);

      if($scope.newSurvey.res[paId].results[sgId].practices[spId].resu
lt==newResult)

      $scope.newSurvey.res[paId].results[sgId].practices[spId].result=
null;
            else

      $scope.newSurvey.res[paId].results[sgId].practices[spId].result=
newResult;
      }
      $scope.checkResult=function(pa, sg, sp, result){
            var    paId=$scope.searchByCode($scope.newSurvey.res,
pa);
            var goals=$scope.newSurvey.res[paId].results;
            var sgId=$scope.searchByCode(goals, sg);
            var
practs=$scope.newSurvey.res[paId].results[sgId].practices;
            var spId=$scope.searchByCode(practs, sp);
            return
$scope.newSurvey.res[paId].results[sgId].practices[spId].result==resul
t;
      }
```

```
            $scope.clearResult=function(pa, sg, sp, result){
                  var    paId=$scope.searchByCode($scope.newSurvey.res,
pa);
                  var goals=$scope.newSurvey.res[paId].results;
                  var sgId=$scope.searchByCode(goals, sg);
                  var
practs=$scope.newSurvey.res[paId].results[sgId].practices;
                  var spId=$scope.searchByCode(practs, sp);

      $scope.newSurvey.res[paId].results[sgId].practices[spId].result=
null;
            }
            $scope.searchByCode=function(array, code){
                  for(var i=0; i<array.length; i++){
                        if(array[i].code==code)
                              return i;
                  }
                  return -1;
            }
            $scope.addSurvey=function(survey){
                  //Scampi Validation...
                  $scope.scampi.surveys.push(survey);

      scampisSvc.editScampi2($scope.scampi).then(function(response){
                        console.log(response);
                        history.go(-1);
                  }).catch(function(error){
                        console.log(error);
                  });
            }
            $scope.initialize();
      }
})();
```

scampiSvc.js

```
(function () {
      'use strict';
      var serviceId = "scampisSvc";
      angular.module('app').factory(serviceId,      ["$http",      "$q",
"scampisRsc", scampisSvc]);
      function scampisSvc($http, $q, scampisRsc) {

            return {
                  setScampis:setScampis,
                  getLocalScampis:getLocalScampis,
                  getScampis:getScampis,
                  postScampi:postScampi,
                  editScampi: editScampi,
                  editScampi2:editScampi2,
                  deleteScampi:deleteScampi,
                  getScampi:getScampi
            };
            var scampis=[];
            function setScampis(scampis){
                  scampis=scampis;
```

```
		}
		function getLocalScampis(){return scampis;}
		function getScampi(id){
			var deferred=$q.defer();
			var scampi=null;

	scampisRsc.queryScampi({_id:id}).then(function(response){
				scampi=response[0];
				deferred.resolve(response);
				return deferred.promise;
			}).catch(function(error){
				deferred.reject(error);
				return deferred.promise;
			});
			return deferred.promise;
		}
		function getScampis(){
			var deferred=$q.defer();
			scampisRsc.getScampis({}).then(function(response){
				setScampis(response);
				deferred.resolve(response);
				return deferred.promise;
			}).catch(function(error){
				var res={};
				res.isError=true;
				res.errorMessage="Error getting all the process
areas";
				deferred.reject(res);
				return deferred.promise;
			});
			return deferred.promise;
		}
		function postScampi(scampi){
			var deferred=$q.defer();
			scampisRsc.save(scampi).then(function(response){
				console.log(response)
				deferred.resolve(response);
				return deferred.promise;
			}).catch(function(error){
				console.log(error);
				deferred.reject(error);
				return deferred.promise;
			});
			return deferred.promise;
		}
		function editScampi(scampi){
			var deferred=$q.defer();

	scampisRsc.remove({id:scampi._id}).then(function(response){

	scampisRsc.save(scampi).then(function(response){
					deferred.resolve(Scampi);
					return deferred.promise;
				}).catch(function(error){
					console.log(error);
					deferred.reject(error);
					return deferred.promise;
				});
			}).catch(function(error){
				console.log(error);
				deferred.reject(error);
```

```
                        return deferred.promise;
                  });
                  return deferred.promise;
            }
            function editScampi2(Scampi){
                  var deferred=$q.defer();
                  scampisRsc.update(Scampi).then(function(response){
                        deferred.resolve(Scampi);
                        return deferred.promise;
                  }).catch(function(error){
                        console.log(error);
                        deferred.reject(error);
                        return deferred.promise;
                  });
                  return deferred.promise;
            }
            function deleteScampi(scampi){
                  var deferred=$q.defer();
                  scampisRsc.remove({_id:
scampi._id}).then(function(response){
                        console.log(response)
                        deferred.resolve(response);
                        return deferred.promise;
                  }).catch(function(error){
                        console.log(error);
                        deferred.reject(error);
                        return deferred.promise;
                  });
                  return deferred.promise;
            }
      }
})();
```

app.js

```
(function () {
    "use strict";
    var app = angular.module('app', [
            //Angular modules
            'ngRoute', //Routing
            'ui.router',
            'ngResource'
    ]);
    var config = {
        remoteServiceBaseUrl: "http://172.245.137.174:8080/api"
      };

    app.constant('config', config);

    app.run(["$rootScope",
            function ($rootScope) {
            var language = "es";
            }
    ]);
    app.config(['$stateProvider', '$urlRouterProvider',
      function ($stateProvider, $urlRouterProvider) {
```

```
            $urlRouterProvider.otherwise("/");
            $stateProvider
            .state('app', {
                  url: "/app",
                  abstract: true,
                  templateUrl:
"/app/features/commons/structure/structure.html"
              })
              .state('app.home', {
                  url: "^/",
                  templateUrl: "/app/features/home/home.html"
              })
              .state('app.manage',{
                  url:"^/manage",
                  templateUrl:"/app/features/manage/manage.html",
                  controller:"manageCtrl"
              })
              .state('app.createScampi',{
                  url:"^/createScampi",

      templateUrl:"/app/features/createScampi/createScampi.html",
                  controller:"createScampiCtrl"
              })
              .state('app.scampi',{
                  url:"^/scampi/:id",
                  templateUrl:"/app/features/scampi/scampi.html",
                  controller:"scampiCtrl"
              })
              .state('app.newSurvey',{
                  url:"^/scampi/:id/new",
                  templateUrl:"/app/features/scampi/new/new.html",
                  controller:"newCtrl"
              });
      }
    ]);
})();
```

8.2. Scripts servidor

Scampi.js

```
module.exports = Scampi;
function Scampi() {
        if (!(this instanceof Scampi)) {
                return new Scampi();
        }
        var mongo = require('mongodb');
        var db = require('monk')('mongodb://localhost:27017/cmmi');
        this.Scampi = db.get('Scampis');
};
Scampi.prototype.findAll = function(success, error) {
```

```
    this.Scampi.find({},{},response(success, error));
};

Scampi.prototype.findById = function(id, success, error) {
    this.Scampi.findById(id, response(success,error));
};
Scampi.prototype.create = function(scampi, success, error) {
    this.Scampi.insert(scampi, response(success,error));
};

Scampi.prototype.update = function(scampi, success, error) {
        this.Scampi.findAndModify(scampi._id,
                scampi, response(success, error));
};

Scampi.prototype.removeById = function(id, success, error) {
        this.Scampi.remove({_id : id}, response(success, error));
};
var response = function(success, error) {
        return function(err, doc) {
                if (err) {
                        error(err);
                } else {
                        success(doc);
                }
        };
}
```

scampiSvc.js

```
var exports=module.exports;
var db = require('monk')('mongodb://localhost:27017/cmmi');
var scampi = require('../modelsSvcs/scampi.js')();
exports.findAll = function(req, res) {
    // logic to handle GET /scampi
    var ok = function(doc) {
          res.json(doc);
    };
      var err = function(err) {
                res.send(404);
        };
        scampi.findAll(ok, err);
};

exports.findById = function(req, res) {
    // logic to handle GET /scampi/id
    var ok = function(doc) {
                res.json(doc);
        };
        var err = function(err) {
                res.send(404);
        };
        scampi.findById(req.params.id, ok, err);
};
exports.findByCode=function(req, res){
    var ok=function(doc){
        res.json(doc);
    };
```

```
    var err=function(err){
        res.send(404);
    }
    scampi.findByCode(req.params.code, ok, err);
};
exports.create = function(req, res) {
    // logic to handle POST /scampi
    var ok = function(doc) {
                res.location('/scampi/doc._id');
                res.send(201);
        };
        var err = function(err) {
                res.send(409, "Failed to create scampi");
        };
        scampi.create(req.body, ok, err);
}
exports.update = function(req, res) {
      if (!req.body._id) {
                res.send(404, "id required");
        } else {
                var ok = function(doc) {
                        res.send(200);
                };
                var err = function(err) {
                        res.send(409, "update failed");
                };
                scampi.update(req.body, ok, err);
        }
    // logic to handle PUT /scampi/
}
exports.deleteById = function(req, res) {
    // logic to handle DELETE /scampi/
     var ok = function(doc) {
                res.send(200);
        };
        var err = function(err) {
                res.send(409, "Failed to remove scampi");
        };
        scampi.removeById(req.params.id, ok, err);
}
```

Server.js

```
var express = require("express");
var app = express();
var router = express.Router();
var bodyParser = require('body-parser');

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE');
  res.header("Access-Control-Allow-Headers",         "X-Requested-With,
Content-Type");
  next();
});
app.use(bodyParser.json())
app.use('/api', router);
```

```
var processAreaSvc = require('./services/processAreaSvc.js');
router.get('/areas', processAreaSvc.findAll);
router.get('/areas/:id', processAreaSvc.findById);
router.post('/areas', processAreaSvc.create);
router.delete('/areas/:id', processAreaSvc.deleteById);
router.put('/areas/:id', processAreaSvc.update);

var scampiSvc = require('./services/scampiSvc.js');
router.get('/scampis', scampiSvc.findAll);
router.get('/scampis/:id', scampiSvc.findById);
router.post('/scampis', scampiSvc.create);
router.delete('/scampis/:id', scampiSvc.deleteById);
router.put('/scampis/:id', scampiSvc.update);

app.listen(8080, function(){
      console.log('Magic happens on port ' + 8080);
});
```

9. Glosario de términos

- ***Front-end***: también conocida como "interfaz" o "cliente". Es uno de los dos extremos de la aplicación y en este caso es el final que se ocupa de la interacción con los usuarios.
- ***Back-end***: es el otro extremo de la aplicación. Este caso su traducción es algo complicado de traducir pero "servidor" o "parte servidor" sería lo más cercano. Es el componente de la aplicación que procesa y envía la información que llega y va desde o hacia el *front-end*.
- ***API REST***: una API es un conjunto de recursos de servicio agrupados y ordenados de forma que se puede acceder a ellos de manera remota. Es una forma de tener acceso a los recursos de uno o varios servicios y poder hacer uso de ellos de manera más o menos estandarizada. Una API REST normalmente se apoya en un sistema de almacenamiento de datos y en cuatro funciones básicas: creación de datos, actualización de datos, obtención de datos y eliminación de datos.
- ***Refactorización***: proceso de revisión y re-escritura del código de programación una vez solucionada una funcionalidad o problema. Es mejorar lo escrito y ya funcional.
- ***Testing***: parte del proceso de producción de software que contempla pruebas y escenarios simulados del software desarrollado para destapar errores o fallos que aparezcan durante el uso normal del software y no durante el desarrollo.
- ***Deployment***: también conocido como "despliegue". Parte del proceso de producción de software que contempla la implantación del producto en el ordenador del/los usuario/s para poder utilizarlo de manera normal.
- ***Bug***: error de software que nos lleva a un comportamiento anómalo del sistema así como a un resultado inesperado.
- **Framework**: también conocido como marco de trabajo. Nos proporciona una serie de herramientas para llevar a cabo acciones dentro de un entorno específico.
- **Diseño responsive**: serie de cláusulas o buenas prácticas que intentan orientar el diseño a su adaptación al medio en el que se encuentra, teniendo que llevar a cabo un diseño simultáneo para diferentes plataformas.
- **MVC**: siglas de "Model View Controller". Paradigma de programación que establece las bases de la arquitectura de una aplicación siguiendo una orientación a la separación por capas en vistas, controladores de las vistas y modelos de datos.
- **Scope**: también conocido como "alcance". Es el entorno en el que se encuentra una serie de funcionalidades dentro del código. Ejemplo: Cada función crea un nuevo scope dentro de la misma función y todo lo que ocurre dentro, a no ser que se referencie fuera, sean variables globales, se retorne como resultado o se pase como parámetro, tiene el alcance de la función misma.
- **Endpoint**: uno de los extremos de un *back-end* con API. Un *endpoint* es un punto de conexión al que se puede acceder y conectar de forma pública y que nos proporciona una o varias funcionalidades según la información que le introduzcamos. Es un "puerto" al que ir a parar cuando se necesita cierto recurso.
- **Middleware**: es el nombre de una capa de aplicación que hace de intermediario entre dos capas, una superior y una inferior. Puede ser una interfaz, un

controlador, un servicio… tiene poca o ninguna funcionalidad propia y hace las veces de puente entre dos capas.

- **Callback**: función que se ejecuta cuando una promesa nos proporciona un resultado. Según la promesa y su resultado, la callback puede ser de éxito "success" o error.

10. Referencias

[1]CMMI Product Development Team. *CMMI for Development, Version 1.3* [En línea]. Software Engineering Institute, Carnegie Mellon University, Pittsburg, 2010. [Consulta: 22 de Abril de 2015]. Formato PDF, 6,33MB. Disponible en: http://www.sei.cmu.edu/library/assets/whitepapers/Spanish%20Technical%20Report%20CMMI%20V%201%203.pdf

[2]GOLDBLAT, Reuven. *Capability Maturity Practices: Contributions to the Competitive Advantage of an Organization*. Dirigido por Shannon Anderson. Arizona, Estados Unidos: Northcentral University, 2013. Tesis doctoral.

[3]GIBSON, Dennie y GOLDENSON, Dennis. *Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results* [en línea]. Software Engineering Institute, Carnegie Mellon University, Pittsburg, 2003. [Consulta el 2 de Agosto de 2015]. Formato PDF, 393 KB. Disponible en: http://resources.sei.cmu.edu/asset_files/SpecialReport/2003_003_001_14117.pdf

[4]PREUNINGER, Ricky Don. *The Advantages of Implementing Software Engineering Process Models* [en línea]. Arlington, Texas: The University Of Texas At Arlington, 2006. Trabajo final de master. [Consulta el 4 de Agosto de 2015]. Formato PDF, 354 KB. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.8872&rep=rep1&type=pdf

[5]Software Engineering Institute. *Brief History of CMMI* [En línea]. Carnegie Mellon University, Pittsburg. [Consulta: 5 Agosto de 2015]. Formato PDF, 73,3 KB. Disponible en: http://www.sei.cmu.edu/library/assets/cmmihistory.pdf

[6]NodeJS [en línea]. [Consulta: 8 de Agosto de 2015]. Disponible en: https://nodejs.org/

[7]AngularJS [en línea]. [Consulta: 8 de Agosto de 2015]. Disponible en: https://angularjs.org

[8]Bootstrap [en línea]. [Consulta: 8 de Agosto de 2015]. Disponible en: http://getbootstrap.com/

[9]MongoDB [en línea]. [Consulta: 8 de Agosto de 2015]. Disponible en: https://www.mongodb.org/

[10]Git [en línea]. [Consulta: 23 de Agosto de 2015]. Disponible en: https://git-scm.com/