**Informe del primer sprint**

GPS-Fooding



Cristina Lahoz Egea (544393)

Patricia Lázaro Tello (554309)

Jorge Martínez Lascorz (571735)

Alejandro Royo Amondarain (560285)

Jaime Ruiz-Borau Vizárraga (546751)

**Índice**

**1. Introducción**

El producto a desarrollar consiste en una aplicación de recetas de comida para smartphones, concretamente dispositivos con sistema operativo Android, que ayude a personas sin grandes conocimientos de cocina a hacer la comida.

El equipo que desarrolla este proyecto se compone de cinco integrantes. Los cinco miembros del equipo componen el *equipo de desarrollo*, Cristina Lahoz es la *dueña del producto* y Alejandro Royo es el *Scrum Master*.

En las siguientes secciones del documento se explica el estado del producto al finalizar el sprint y las diversas estrategias seguidas en el proceso durante el sprint.

En la sección del *Producto* se plantea el estado actual del producto, incluyendo la pila de producto, plan de producto y planificación de lanzamientos, así como su diseño arquitectural.

En la sección del *Proceso* se explican las diversas estrategias de control de versiones, test y construcción automática de software seguidas, así como diagramas de burnup, velocidad del equipo y esfuerzos, y los resultados de la retrospectiva del sprint.

En la última sección, *Conclusiones*, se enumera el grado de cumplimiento de los objetivos de la evaluación del proyecto, así como un pequeño sumario del documento.

**2. Producto**

**2.1. Plan de producto**

**Nuestra visión**

**Fooding** es una aplicación para Android que conciencia a las personas para que coman sano mediante un sistema de puntuación y rankings globales. Para comer sano no hay nada mejor que el hábito, y Fooding trata de crear precisamente esto, hábitos.

Sabiendo que no todos los usuarios tienen grandes conocimientos de comida, se pone a su disposición una gran colección de recetas con instrucciones detalladas y paso a paso y diversas ayudas, como generación de listas de la compra o búsqueda de supermercados cercanos.

La aplicación está dirigida a estudiantes en pisos de estudiantes y personas jóvenes que viven solas o que no tienen tiempo para comer, pues son las que peor comen, ya sea por dejadez, falta de tiempo o de ganas.

Con Fooding, será muy fácil mantener buenos hábitos de comida, respetando el tiempo mínimo para comer, sin saltarse comidas y sin recurrir a la siempre presente comida basura.

**Requisitos de la aplicación**

La aplicación permitirá al usuario encontrar las recetas en función de los ingredientes que tenga disponibles en un momento determinado (tres ingredientes como máximo). La búsqueda de dichas recetas se podrá filtrar además mediante nombres, ingredientes a utilizar y por tipo de receta (carne, pescado, verdura, postre y pasta). Si el usuario no tiene algún ingrediente para completar una receta la aplicación buscará el supermercado más cercano y permitirá realizar la compra digitalmente. Para que el usuario pueda mantener un registro de las últimas comidas y repetir fácilmente la aplicación permitirá guardar las recetas favoritas.

Por otro lado, la aplicación fomenta la interacción con otros usuarios utilizando valoraciones sobre recetas y clasificaciones en función de dichas valoraciones. Además el usuario podrá competir con la comunidad compartiendo sus recetas y ganando puntos por ello. Estos puntos permitirán al usuario subir de nivel y ser más visible dentro de la comunidad. Así, cuando publique alguna receta esta tendrá prioridad respecto al resto de publicaciones. Los puntos se podrán obtener contribuyendo a la comunidad de manera activa, esto es, compartiendo nuevas recetas de nuevos tipos de comida, valorar recetas de otros usuarios o recibir buenas valoraciones de las recetas ya publicadas.

**Hoja de ruta**

A continuación se describe el hoja de ruta a seguir durante el primer año de desarrollo:

El *primer lanzamiento* corresponde a enero de 2016: incluirá las funcionalidades correspondientes al buscador de recetas, sus diferentes filtros, localización del usuario y supermercados cercanos mediante GPS.

El *segundo lanzamiento* se realizará en abril de 2016: incluirá la primera parte de la comunidad de usuarios, con sistema de puntuación, valoración de recetas, ranking por usuarios y aportaciones por parte de los usuarios.

El *tercer lanzamiento* se producirá en julio de 2016: incluirá la segunda parte de la comunidad de usuarios, con sistema de alianzas y ranking por alianzas.

El *cuarto lanzamiento* corresponde a septiembre de 2016: incluirá la compra automática por internet en supermercados.

**2.2. Análisis de riesgos**

**2.3. Pila del producto**

**2.4. Estado de la aplicación**

**2.4.1. Arquitectura**

**3. Proceso**

**3.1. Definición de hecho**

Se considera que una tarea está hecha cuando, después de las fases de análisis, diseño, implementación y pruebas, sea entregada a la dueña del producto y ésta dé su aprobación.

Concretamente, se sigue el siguiente esquema de checklist:

* El código correspondiente a la tarea se encuentra alojado en los repositorios de GitHub.
* La aplicación de Android y la aplicación servidor han pasado todos los tests automáticos después de incorporar el nuevo código.
* En Openshift (el entorno de desarrollo del servidor) debe estar alojado un WAR con la nueva funcionalidad implementada.
* Se ha notificado a la dueña del producto de la finalización de la tarea.
* El código correspondiente a la tarea se encuentra correctamente documentado; si es necesario, se añadirán nuevas entradas a la Wiki del repositorio, incluyendo los diagramas, imagénes, bocetos... que se consideren necesarios.

**3.2. Test y construcción del software**

En la aplicación para Android se ha utilizado *Gradle* como herramienta de gestión de dependencias. Para los tests unitarios se ha utilizado la herramienta *Robotium*, pues se ha decidido hacer pruebas de interfaz, probando los métodos y funciones asociadas a las funcionalidades sin llamar directamente a tales métodos.

Para la compilación se ha utilizado también *Gradle*, habiéndose creado un script de compilación que resuelve las dependencias y realiza la compilación de la aplicación, generando un archivo con extensión APK.

No se han utilizado herramientas de cobertura de tests automáticos, pues se ha comprobado al final del sprint el nivel de cobertura de tests manualmente.

En la aplicación servidor se ha utilizado *Maven* como herramienta de gestión de dependencias y compilación. Para los tests unitarios se ha utilizado *Junit* y un mock de *SpringFramework* (para probar los métodos de acceso desde Internet en un servlet, como GET y POST).

La política seguida en el servidor para los tests fue, implementada la funcionalidad, realizar tests en todos los niveles, desde llamadas a los métodos de acceso a la Base de Datos hasta mocks para comprobar el parseo de datos.

Además, se han realizado pruebas adicionales desde la aplicación y desde distintos navegadores para comprobar su corrección.

Para la compilación también se ha usado *Maven*. Los tests son pasados automáticamente con cada push al repositorio del servidor mediante el script de Maven, que también se encarga de crear un archivo WAR y desplegarlo en Openshift.

Se ha utilizado *JaCoCo* como herramienta para la cobertura de tests automáticos, habiéndose implementado un script para Windows para descargar al archivo HTML que genera JaCoCo con cada compilación en el servidor. Adicionalmente, esta herramienta desglosa en paquetes la cobertura de código, además de mostrar el porcentaje de código cubierto globalmente.

**3.3. Control de versiones**

Se han creado en GitHub un total de tres repositorios: un repositorio para la aplicación, otro repositorio para el servidor y un último repositorio para la documentación.

El repositorio del servidor es una copia exacta del repositorio privado de Openshift en el que se aloja la aplicación servidor: el workflow en este repositorio es *centralizado*, utilizando la rama master como rama principal a la que todos los miembros del equipo suben sus cambios.

Para mantener el repositorio de GitHub sincronizado con el repositorio privado de Openshift se tuvo que crear un enlace a remoto en cada ordenador, así como subir de forma automática al repositorio de GitHub lo mismo que se sube al repositorio de Openshift.

Se decidió crear un conjunto de scripts para automatizar este proceso, siendo estos usados por todos los miembros del equipo. La documentación asociada a estos scripts se encuentra en la Wiki del repositorio, junto a instrucciones de instalación y setup de Openshift.

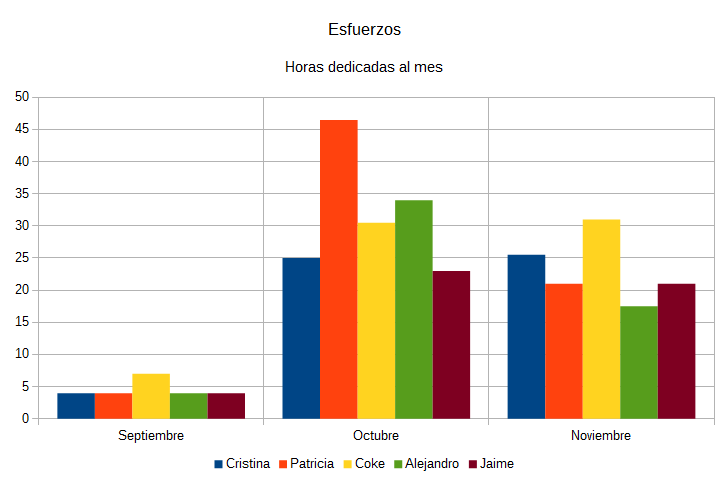
El repositorio de la aplicación también tiene un *workflow centralizado* en torno a la rama master; aunque para implementar la comunicación entre cliente y servidor se creó una rama (connection\_server) separada que iba uniéndose con la rama principal cuando una nueva característica era añadida y testeada.

Por último, en el repositorio para la documentación también se ha seguido un esquema de *workflow centralizado*, pues los cambios en la documentación no son tan frecuentes como para que supusieran conflictos y no se dio por tanto la necesidad de crear ramas adicionales.

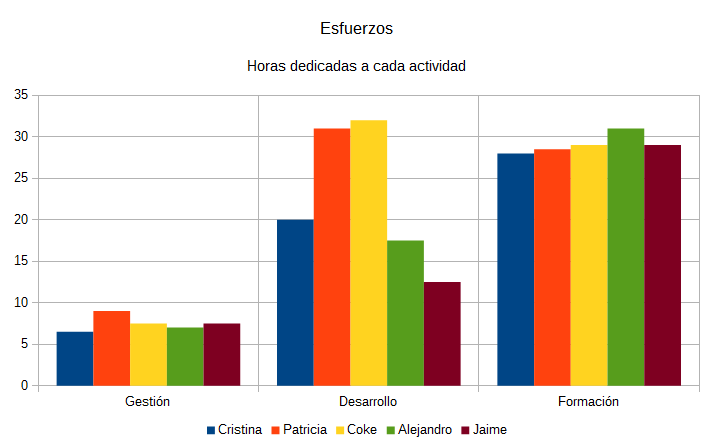
**3.4. Estadísticas del equipo**

**3.4.1. Esfuerzos**

Esfuerzos (medidos en horas) de cada miembro del equipo por mes:



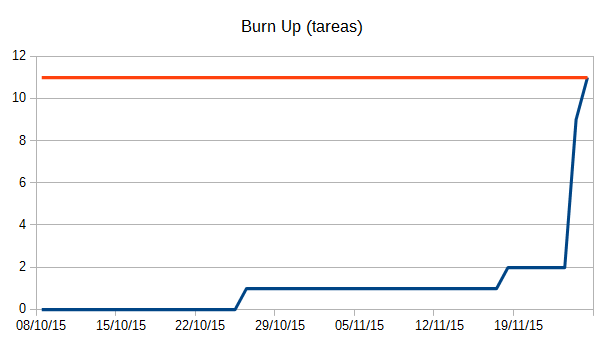
Esfuerzos (medidos en horas) de cada miembro del equipo por actividad:

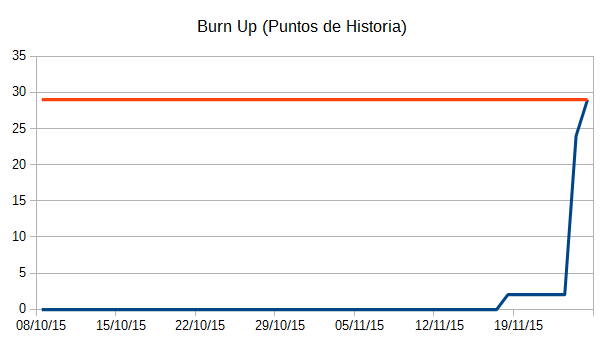


**3.4.2. Velocidad**

En este sprint, el equipo ha completado 29 puntos de historia; como no se tienen datos históricos del equipo, no se puede dar un rango de velocidades. La velocidad del equipo será, por tanto, *29 puntos de historia*.

A continuación, se muestran dos *diagramas de burn up* o trabajo completado, uno haciendo referencia al número de tareas completadas y el otro con puntos de historia:





La mayoría de las tareas fueron completadas en el último tramo del sprint debido a problemas que impedían pasar los tests automáticos (problemas de configuración).

**3.5. Retrospectiva del sprint**

**4. Conclusiones**

**4.1. Sumario**

**4.2. Cumplimiento de objetivos**

* El código del proyecto se aloja en GitHub y se trabaja de forma habitual contra git.
* Cobertura de tests automáticos:
* Historias de usuario/requisitos implementados en el sprint evaluados con buenos criterios de aceptación:
* Definición de hecho clara, usada e incluye que se pasen todos los tests automáticos.
* Documentación arquitectural (vista de módulos/componente y de despliegue, discusión sobre razones arquitecturales):
* Cumplimiento suficiente de requisitos/características:
* Control de esfuerzos:
* Compilación y gestión de dependencias basada en scripts (despliegue automático, generación de binarios, triggers automáticos...):
* Toda la documentación del proyecto está bajo control de versiones.
* Tests de aceptación automáticos para criterios de aceptación implementados en el sprint: