



Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo Fin de Grado

Desarrollo en la nube: ChefManagement

Development in cloud: ChefManagement

ULL - ESIT
Aarón Socas Gaspar

Tutor
Vicente Jose Blanco Perez

La Laguna, 5 de septiembre de 2015

D. **Vicente Jose Blanco Perez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada: *Desarrollo en la nube: ChefManagement*.

ha sido realizada bajo su dirección por D. **Aarón Socas Gaspar**, con N.I.F. 78.702.938-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos portunos firman la presente en La Laguna a 5 de septiembre de 2015.

Agradecimientos

Si he llegado hasta aquí ha sido gracias al apoyo de mi familia y mis amigos. Agradezco sus ánimos y ayuda por facilitar mi labor.

Una mención especial para mi madre Pilar, por enseñarme como persona lo que los libros no pueden,

a mi novia Pilar, por quererme, aguantarme y tener paciencia conmigo,

a mi hermano Josué, porque siempre está ahí y se preocupa por mí,

a mis tíos y abuelos, por ayudarme en todo y más,

a mi amigo Pablo, porque sé que puedo contar contigo para lo que sea.

También a mis profesores, que han sido muchas las horas en tutorías aguantando mis preguntas, en especial a Vicente Jose Blanco Perez, Casiano Rodríguez León, Gara Miranda Valladares y Jose Luis Roda García.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

En este proyecto se abordarán aspectos del desarrollo de aplicaciones web y entornos de trabajo Cloud (PaaS). También el uso de la metodología ágil y el desarrollo dirigido por pruebas (TDD). La aplicación a desarrollar tiene como objetivo gestionar el escandallo de recetas haciendo uso de aplicaciones de terceros (APIs). Para hacer esto me ayudaré de herramientas orientadas al desarrollo de software, uso de repositorios (Github) y servidores de pruebas (Travis).

Palabras clave: Cloud, PaaS, TDD, escandallo, API.

Abstract

This project will approach issues web application development and Cloud working environment (PaaS). In addition to use of agile methodology and Test-driven development (TDD). The purpose of the application manage recipe cost production using third application (APIs). To do this, I will use tools oriented to software development, repository (Github) and test cloud (Travis).

Keywords: *Cloud, PaaS, TDD, recipe production cost, API.*

Índice general

1. Introducción	7
1.1. Objetivos	7
1.2. Antecedentes y estado actual del tema	7
1.3. Metodología de trabajo	11
2. Descripción de la aplicación	13
2.1. Acceso a la aplicación	14
2.2. Crear recetas	16
2.3. Editar y eliminar recetas	17
2.4. Calculadora	17
2.5. Importar y exportar	18
2.6. Opciones	18
2.7. Diseño de la base de datos	18
3. Gestión de la configuración	21
3.1. PaaS	21
3.2. Heroku	22
3.3. OpenShift	24
4. Metodología de desarrollo	25
4.1. Metodología ágil	25
4.2. Estructura de la aplicación	26
5. Conclusiones y trabajos futuros	29
6. Summary and Conclusions	31
7. Presupuesto	33
8. Glosario	35
Bibliografía	35

Índice de figuras

1.1. Lenguajes soportados en GAE	9
1.2. Esquema de metodología ágil	11
2.1. Acceso a la aplicación	14
2.2. Recuperar contraseña	15
2.3. Registro	15
2.4. Modelo entidad-relación	19
3.1. Modelos de nubes	22
3.2. Configuración de la BBDD con DataMapper	23
3.3. Configuración de la BBDD para Chefmanagement	23
4.1. Modelo-Vista-Controlador	26
4.2. ActiveRecord 4.2 no está soportado completamente por JRuby	27

Listings

Capítulo 1

Introducción

1.1. Objetivos

El objetivo de este trabajo de fin de grado (en adelante, TFG) es el desarrollo de una aplicación web para la gestión del escandallo, es decir, controlar el precio de producción de una receta. Ésta se llamará **Chefmanagement** y para llevarlo a cabo usaremos Cloud Servers, concretamente Google Cloud Platform, sin embargo, por algunas de las restricciones que mostraba, las cuales se describirán más adelante, se reorientó el TFG hacia otras plataformas como servicio (SaaS). Para su desarrollo usé la metodología de software ágil basada en el desarrollo dirigido por pruebas (TDD). Como framework, Sinatra (Ruby) para desarrollar la aplicación, integrar APIs de terceros, y hacer uso de varias nubes de producción a la vez y también herramientas de soporte para el desarrollo de software como son Github y Travis, entre otras.

1.2. Antecedentes y estado actual del tema

Los antecedentes y estudio de campo no sólo se centran en el tipo de aplicación que vamos a desarrollar, también tendremos en cuenta el entorno de producción, es decir, la nube donde estará disponible la aplicación web. Por lo tanto, tenemos dos puntos iniciales de estudio:

- Sobre la aplicación:
 - ¿Qué ofrece y por qué?
 - ¿Qué otras aplicaciones similares existen?
 - Estudio del sector del mercado.
 - Soporte y aspectos de la usabilidad.
- Y respecto a la nube:
 - ¿Cuál es la mejor nube que se adapte a nuestras necesidades?
 - ¿Qué lenguajes y frameworks soporta?
 - Período, costos, capacidad, etc.

Aplicación

La finalidad de esta aplicación nace de la necesidad de optimizar los gastos, tanto en negocios de restauración como en los propios hogares. Hacer la comida controlando gastos es posible. En la actualidad existen en el mercado aplicaciones con esta idea, algunas de las mas destacadas son:

- Recipe Cost Calculator ([23]): Quizás sea la aplicación base como referencia, su fácil gestión y múltiples funciones la hace una herramienta útil y potente. Esta principalmente orientada a negocios.
- Recipe Costing ([24]): Esta aplicación va más allá de las funcionalidades básicas, presenta extras como cálculos de menús, gestión de inventario, órdenes de compra a proveedores, etc.
- Pearson Kitchen Manager ([20]): Se trata de una API, un banco de información que contiene más de 3000 mil recetas etiquetadas y con otra información como sus valores nutricionales.
- Recipe Costing Calculator ([25]): Es más sencilla que las anteriores pero se trata de App disponible en iTunes. Las cosas más sencillas pueden ser las más útiles, hay que tener en cuenta que un cocinero/a prefiere una tablet a un ordenador en la cocina.

La segmentación del mercado está orientado especialmente a negocios de restauración: comedores, restaurantes, bares, pastelerías, panaderías, etc. Por otro lado, existe otro sector que no se tiene en cuenta debido a que no genera tantos beneficios, se trata de los hogares. Las personas también pueden hacer uso de esta herramienta pues sus necesidades son las mismas pero a menor escala. En el capítulo 2, se describe toda la funcionalidad de la aplicación, al igual que nos enseña como usarla y veremos las diferencias en la aplicación en función de a qué mercado está dirigido.

Por último, hay que tener en cuenta el soporte para la aplicación. Aprovechando la infraestructura y poder de Internet, la mejor opción es crear una aplicación web, que aunque en esta primera versión se diseñará de forma adaptativa, la idea es poder utilizarla en el futuro en dispositivos móviles y tablets. Además, se ha tenido en cuenta los aspectos de usabilidad durante su diseño.

Nube (cloud)

Inicialmente la idea es trabajar en la nube de Google ([8]). Se trata de una plataforma como servicio (PaaS), la cual permite crear y mantener de forma sencilla una aplicación en la infraestructura de Google. Además permite una fácil escalabilidad de transeferencia de datos y almacenamiento gracias a sus módulos.

Empleando los conocimientos adquiridos durante los últimos cursos en Ruby y sus variedad de frameworks (Ruby on Rails, Padrino, Sinatra) lo usaré para

crear la aplicación. Tras investigar y ver los servicios que ofrece Google App Engine (GAE) parece viable la puesta en marcha de la aplicación. GAE soporta cuatro lenguajes y sus correspondientes frameworks:

- Python ([22]) con webapp2 y Jinja2.
- Java ([14]) con maven.
- PHP ([21]) con Cloud SQL.
- Go ([12]) con el paquete html/plantilla.

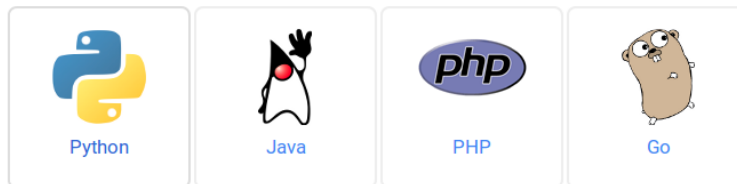


Figura 1.1: Lenguajes soportados en GAE

Sin embargo, también es posible hacerlo funcionar en Ruby con la ayuda de java, juntos forman JRuby ([15]), el cual es una implementación 100 % del lenguaje Ruby. Además funciona como lenguaje embebido dentro de la máquina virtual de Java. Gracias a esta capa, podré programar en Ruby y Jruby establecerá una capa intermedia entre el código fuente en Ruby y el servidor en java. Para interactuar con la aplicación en la nube provee de una herramienta o kit, en este caso se trata de una gema llamada google-appengine ([10]), sin embargo, la gema dedicada a tal fin esta desfasada y el proyecto ha sido archivado ([16]).

Tras la incompatibilidad del proyecto de JRuby en GAE busco la opción más parecida posible que me ofrezca soporte, en este caso Google Cloud Platform. Tras investigar encuentro multitud de lenguajes compatibles ([17]), entre ellos Ruby, sin embargo, los servicios de esta nube son de pago y aunque existe una versión demo durante 60 días. Debido al tiempo que emplearía en el estudio, desarrollo y pruebas, este período puede resultar corto. Me pongo en contacto con el servicio de clientes y soporte de Google para bajar otras opciones, pero no ofrecen nada que me sea viable para desarrollar mi proyecto.

De esta forma se modifica el requisito inicial del TFG de usar los servicios de la nube de Google para encontrar y usar otros entornos de producción. La idea es intentar usar más de uno, demostrando la modularidad de la aplicación, esto significa que el código fuente de la aplicación debe ser único para las distintas nubes de producción. El siguiente paso es investigar las nubes que se ofertan. La siguiente tabla muestra un resumen:

Nube	Comentario
Google Cloud ([?])	De pago. Prueba gratuita de 60 días. Multitud de servicios, distintos tipos de bbdd y soporta muchos lenguajes (Ruby, NodeJs o JRuby), frameworks (Django o ASP.NET) y CMS (WordPress o Drupal).
Azure ([4])	De pago. Prueba de 30 días. Soporta varios lenguajes: C#, Go, Java, NodeJs, PHP y Python.
Cloud9 ([6])	Se trata de un entorno de trabajo, una nube orientada al desarrollo, no es lo que buscamos.
Nitrous ([18])	Es similar a la anterior. Orientada al trabajo colaborativo en entornos de desarrollo.
DotCloud ([9])	Esta es una propuesta válida. Se trata de una nube de producción con servicios de pago y versión básica gratuita. Soporta varios lenguajes (Java, NodeJs, PHP, Ruby y Python). El uso de módulos como bbdd o rendimiento del servidor son de pago.
Heroku ([13])	Ideal para experimentar con aplicaciones en la nube en un módulo limitada. Su uso es ilimitado, pero tiene algunas condiciones. Soporta varios lenguajes (Ruby, Java, NodeJs, Python, Clojure, Scala y Go) y frameworks (Ruby on Rails, Django).
OpenShift ([19])	Es una alternativa muy interesante. Existe la opción de uso gratuito de hasta 3 aplicaciones. Soporta varios lenguajes (Java, PHP, NodeJs, Python, Ruby, Perl, Ceylon) y distintas versiones del mismo, así como 3 tipos de bbdd (PostgreSQL, MySQL y MongoDB) y los cartuchos (módulos) que añadamos son de pago. Podemos crear y subir nuestros propios cartuchos.
Appfog ([3])	De pago con versión de prueba de 30 días. Soporta varios lenguajes: Go, Java, NodeJs, PHP, Python y Ruby. El enfoque de esta PaaS es facilitar el trabajo al equipo de desarrollo y que solo se centre en la aplicación y en los datos, mientras la nube gestionaría el resto.
Amazon ([2])	Presenta una gran infraestructura de servicios y recursos. Es de pago pero tiene una versión de prueba de 12 meses. Soporta varios lenguajes y dispone de multitud de productos orientados a distintos tipos de mercado.
DigitalOcean ([7])	Es de pago. Provee de infraestructura remota para que el desarrollador pueda levantar el tipo de servidor que desee. Soporta varios lenguajes (Java, Perl, PHP, Python y Ruby) y frameworks.
Cloud66 ([5])	Es de pago pero tiene un período de 14 días de prueba. Esta orientada a mejorar las aplicaciones que ya tenemos en otras nubes, en cuanto aspectos de crecimiento y seguridad.
1 and 1 ([1])	Es de pago. Soporta diferentes lenguajes, frameworks y CMS.

Cuadro 1.1: Production Clouds

Una vez analizadas se propone como entornos de producción las nubes Heroku y Openshift, de las cuales se entrará en detalle en el capítulo 3.

1.3. Metodología de trabajo

Durante la investigación y desarrollo de este proyecto se ha apostado por las metodologías ágiles, si bien no ha sido en un grupo de trabajo, se puede aplicar en lo que se refiere al análisis, al desarrollo y diseño de la aplicación y a las reuniones con el *cliente*, cuyo rol lo asume el tutor junto con el de *jefe de proyecto*. Además me entrevisté con dos negocios de restauración y un proveedor con el fin de establecer requisitos, los cuales pueden ser considerados también *clientes*.

Estos métodos están basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos. La idea es minimizar riesgos desarrollando software en períodos cortos. Este período se llama **iteración** cuya duración se define en función de los requisitos del proyecto, especialmente recursos de tiempo y humano.

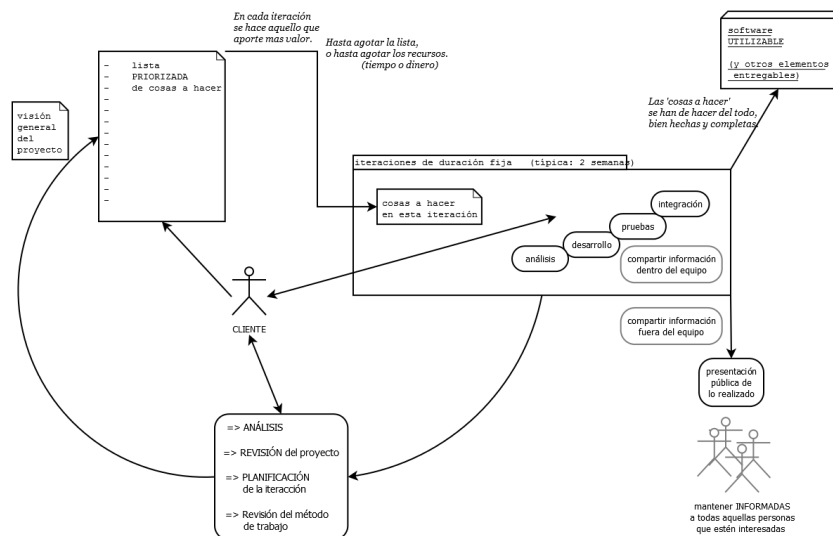


Figura 1.2: Esquema de metodología ágil

Cada iteración del ciclo de vida incluye:

- Planificación: organización del trabajo durante una iteración. En este caso, la planificación corresponde con las tutorías y seguimiento semanal con el tutor.
- Análisis de requisitos: cada nueva funcionalidad o idea para la aplicación debe ser estudiada. Así, por ejemplo, si quiero integrar una API debo

considerar si es útil o no, si es compatible con los entornos de producción y con el lenguaje, documentarme para su integración en la aplicación, etc.

- Diseño: interfaz de la aplicación, características y aspectos de usabilidad.
- Codificación: corresponde por lo general a escribir código, aunque esto no se limita solo a realizar la propia aplicación, además se incluyen los tests y todos aquellos prototipos y pruebas durante la etapa de estudio de campo.
- Revisión: incluye superar los tests con éxito y la aceptación por parte del cliente.
- Documentación: para este trabajo en concreto, se realizó un diario semanal, cuya síntesis se engloba en este documento.

Durante este proyecto se ha seguido dicho ciclo de vida con especial importancia. He intentado adaptar la metodología Scrum a este trabajo, aunque no se ha tenido en cuenta el aspecto de los roles. En Scrum se realizan entregas parciales y regulares del producto final. Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Cabe destacar dada la utilización del framework de Sinatra, el uso inherente del modelo-vista-controlador (MVC) durante la etapa de diseño, desarrollo y pruebas. Se describirá detalladamente en el capítulo 4 así como su uso para crear Chefmanagement.

Capítulo 2

Descripción de la aplicación

En el capítulo anterior 1 se ha introducido tanto los antecedentes como se describió brevemente la aplicación. En este capítulo explicaremos toda las funcionalidades y sus características con detalle.

Partimos de qué tipo de aplicación queremos crear. Se trata de desarrollar un software que ayude a las personas a gestionar el costo de producción de las recetas, dependiendo del precio de sus ingredientes y el número de raciones. Ésta se llamará **ChefManagement** y sus características iniciales son:

- Crear, ver, editar y eliminar recetas.
- Listar todas las recetas.
- Crear backup de las recetas y cargarlo si es necesario.
- Calcular el precio de una determinada receta para un determinado número de comensales.

Para esta primera versión, además de ser capaz de realizar esta serie de tareas, debe cumplir con las necesidades y expectativas del usuario y facilitar su interacción con la aplicación. Éstos son sus requisitos iniciales:

- Dos entornos de producción: la aplicación puede ser usada desde dos nubes diferentes.
- Tener en cuenta aspectos de usabilidad: adaptación, entendimiento y facilidad de uso.
- Importar y exportar archivos en formato json: almacenar copias de las recetas en el equipo cliente y restaurarlas si es necesario.
- Acceso mediante registro o haciendo uso de APIs de distintas redes sociales como Google+ o Facebook. Hay que facilitar el acceso a los usuarios.

Tanto las características como los requisitos de la aplicación serán aplicadas a la versión 1.0, pudiendo ser mejorados o añadidos más en futuras versiones 5.

2.1. Acceso a la aplicación

Para usar esta aplicación podemos acceder directamente a sus entornos de producción:

- **Heroku:** <http://chefmanagement.herokuapp.com/>
- **OpenShift:** <http://chefmanagement-esit.rhcloud.com/>

O bien acceder al repositorio de la aplicación (se trata de licencia libre y código abierto) y seguir las instrucciones de instalación.

Una vez en la aplicación encontraremos una pantalla inicial de acceso, que nos permite acceder usando redes sociales, en este caso Google+ y Facebook. Esto es posible gracias al uso de su API (o interfaz de programación de aplicaciones) correspondiente, la cual es un conjunto de funciones o métodos que ofrece una librería para ser utilizado por otro software. En otras palabras, el equipo de desarrollo de Facebook crea esta herramienta para que otras aplicaciones, como Chefmanagement puedan hacer uso de ella. Esto tiene ventajas importantes como la seguridad, desde el punto de vista de desarrollo de software y la comodidad para los usuarios finales.

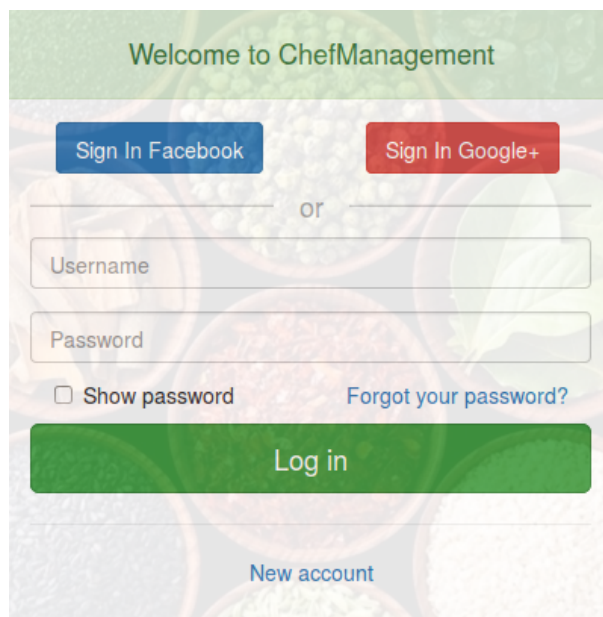


Figura 2.1: Acceso a la aplicación

Además existe la opción de validarnos en la aplicación, una vez que se haya completado el registro. Si el usuario de entrada o contraseña fueran incorrectos la aplicación lo notificaría. En caso de pérdida de contraseña podemos solicitar una nueva accediendo al enlace *Forgot your password?* y proporcionar nuestro nombre de usuario. Posteriormente puede modificarse la nueva contraseña en opciones 2.5.

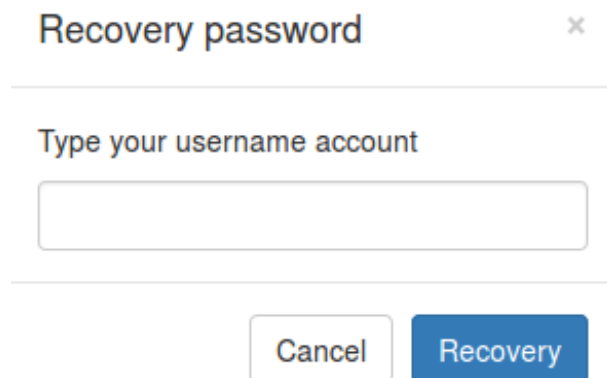
A web form titled "Recovery password" with a close button (X) in the top right corner. Below the title is a label "Type your username account" followed by a text input field. At the bottom, there are two buttons: "Cancel" and "Recovery".

Figura 2.2: Recuperar contraseña

Para registrarse accedemos a través del enlace *New account*. En caso de cualquier problema durante el proceso de registro, como que el correo o nombre de usuario estén en uso, la aplicación lo notificará al usuario.

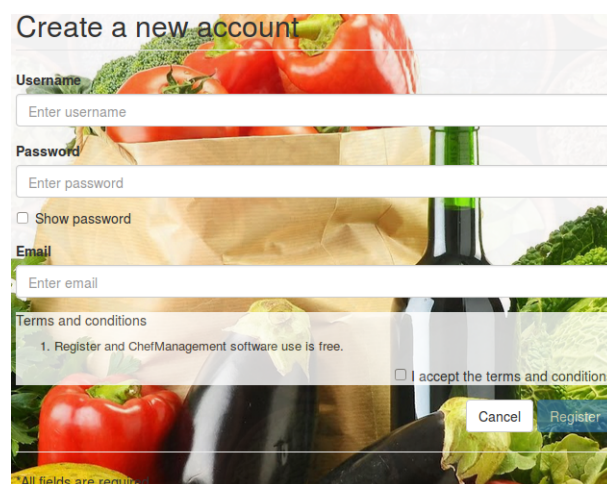
A web form titled "Create a new account" with a background image of vegetables. It contains fields for "Username" (with placeholder "Enter username"), "Password" (with placeholder "Enter password" and a "Show password" checkbox), and "Email" (with placeholder "Enter email"). Below these is a "Terms and conditions" section with a checkbox for "I accept the terms and conditions". At the bottom are "Cancel" and "Register" buttons. A note at the bottom left says "All fields are required".

Figura 2.3: Registro

Una vez dentro de la aplicación accederemos al *home* del usuario. Existe una barra superior donde podemos acceder a la configuración y salir de la aplicación (Log out). También encontramos un menú lateral el cual facilita la interacción del usuario con la aplicación y ayudar a moverse y gestionar las distintas tareas. Por defecto, cuando accedemos a la aplicación, en *home* se nos despliega la lista de recetas creadas por ese usuario. Las opciones del menú son las siguientes:

- Recipe list: muestra el listado de recetas del usuario. Podemos acceder, editar y borrar recetas a través de esta lista.
- Recipe calculator: calcula el costo de producción de una receta y sus ingredientes para una determinada receta y raciones.

- New recipe: crea una nueva receta.
- Import: carga una copia de seguridad de las recetas del usuario.
- Export: almacena en el equipo local una copia de seguridad de las recetas del usuario.

2.2. Crear recetas

Para acceder a crear receta, nos situamos en el menú lateral y pinchamos en **New recipe**. Esto nos llevará a una nueva ventana, la cual presentará un formulario con una serie de campos que se detallan a continuación:

- Name: es un campo obligatorio y clave primaria en la bbdd. Nombre de la receta.
- Portions: Numero de raciones para crear esa receta. Este campo es muy importante ya que se tomará como referencia para el cálculo de recetas dado un determinado número de platos.
- Cost: es un campo de lectura, se calcula automáticamente con cada ingrediente.
- Ration cost: relación entre el coste de la receta y el número de raciones.
- Order: campo que sirve como etiqueta. Indica si se trata de un primer o segundo plato, un postre, etc.
- Type: segundo campo de etiqueta. Nos da información sobre el tipo de comida: snack, comida casera y otros.
- Nivel: nivel de dificultad de producción de la receta.
- Time: horas y minutos que se estima que lleva preparar esa receta.
- Vegan: indica si la receta es apta para vegetarianos.
- Allergens: campo de texto para introducir alimentos que puedan producir alergias.
- Origin: información sobre el país originario de la receta.

Una vez completado podemos guardar la receta con el botón **Save recipe** o cancelar el proceso y volver a *home*. Si guardamos la receta el sistema notificará de su registro en la bbdd y abrirá nuevos campos:

- Un área de texto desplegable que se utiliza para escribir las instrucciones de la receta. He utilizado la herramienta CKeditor ([?]) para integrarla en la aplicación.
- Ingredientes, que puede ser:
 - Un **nuevo ingrediente** creador por el usuario. Campos:

- Nombre del ingrediente
- Cantidad: puede ser: peso, volumen o una determinada unidad.
- Precio en relación con su cantidad
- El porcentaje de merma o pérdida del ingrediente durante su preparación.
- **Una receta** ya creada por el usuario. De forma que las recetas puedan servir de base para crear otras, por ejemplo, la salsa de tomate es una receta que se puede usar en otras. Si una receta usa otra, la primera no puede ser usada para crear otra y no aparecerá como opción en este apartado.

A medida que vayamos incluyendo ingredientes a la receta, en el margen lateral derecho se desplegará una lista informativa de la receta con los ingredientes agregados hasta el momento.

2.3. Editar y eliminar recetas

Desde el listado de recetas podemos editar o eliminar según nuestras necesidades. Para ello pinchamos en una determinada receta y accedemos a toda la información de la receta. Esta opción es de sólo lectura. En la parte superior derecha encontramos los iconos para eliminar y editar la receta.

Para eliminar la receta pulsamos el botón de eliminar. El programa pedirá la confirmación para eliminarla. Una vez eliminada no se puede recuperar. Las recetas que sirven de base a otras no pueden ser eliminadas hasta que no se rompa este vínculo, para ello debemos entrar en el modo edición.

Si pulsamos en edición, nos llevará a una nueva pantalla donde podemos editar todos los campos de la receta (excepto el nombre y los costos que se calculan automáticamente). Podemos añadir nuevos ingredientes, editar o borrar los actuales o bien borrar *recetas base* (en futuras aplicaciones se recomienda poder añadir *recetas bases*). Así mismo podemos editar las instrucciones de la receta.

2.4. Calculadora

Esta opción nos permite elegir entre el listado de recetas del usuario. Una vez hecho esto seleccionamos el número de raciones que queremos producir y pulsamos en calcular. Se mostrará una tabla resultante con el listado de ingredientes, sus cantidades para las raciones seleccionadas y su precio. De la misma forma se mostrarán las *recetas bases* si contiene alguna y por último el precio de producción final.

A modo de información adicional se muestra las instrucciones de la **receta original**.

2.5. Importar y exportar

Estas opciones nos permiten crear y cargar una copia de seguridad de las recetas y sus ingredientes del usuario. Para crear una copia, seleccionamos la opción **Export** del menú lateral. Se mostrará un campo de texto para elegir el nombre del archivo, (por defecto *recipe*) el cual podemos editar y pulsamos en crear. Se comprueba si la cadena introducida contiene caracteres inválidos, en caso contrario se notifica al usuario la operación exitosa y se descarga el archivo al equipo local en formato **json**.

Por otro lado, está la opción de cargar la copia de seguridad. Para ello pulsamos en la opción **Import**, nos permitirá seleccionar un archivo solo válido en formato json. Una vez seleccionado pulsamos en cargar, en caso de error, la aplicación lo notificará con un mensaje, y si el archivo se ha cargado correctamente nos redirige a *home* con el listado de recetas. Hay que tener en cuenta que si cargamos una copia de seguridad borrará el contenido actual de recetas para cargar ésta.

2.6. Opciones

Las opciones se encuentran en la parte derecha de la barra superior de la aplicación. Si pulsamos nos lleva a una nueva pantalla que nos muestra el perfil del usuario. Éste se podrá editar en caso de que el usuario haya accedido mediante registro en la propia aplicación, si accedió vía redes sociales debe acceder a las mismas para editar su perfil.

También permite la opción de darse de baja de la aplicación. Ello conlleva el borrado de todas sus recetas de la bbdd.

2.7. Diseño de la base de datos

El modelo propuesto para esta primera versión atendiendo a los requisitos de la aplicación, es el siguiente:

- **Usuarios (User):** Se trata del modelo base y del que depende la gestión del resto. Sus campos son: *username*, *email*, *password* y *network*. Siendo *username* y *email* sus claves primarias. La contraseña y correo del usuario pueden ser editados siempre y cuando no se haya accedido a la aplicación vía Google+ o Facebook.
- **Recetas (Recipe):** Este modelo contiene información sobre la receta creada por un determinado usuario. Establece la relación con la clase Usuario, del que depende de uno a muchos (1:M) y con las clases ingredientes y recetas base, cuya relación es en ambos casos de 1:M. Tiene muchos campos, de los que cabe destacar: *name*, nombre de la receta y clave primaria; *nracion*, campo requerido y que indica el número de platos, y *username* que es el autor de la receta. Sus relaciones son: una receta

puede tener varios ingredientes (clase Ingredients) o varias recetas base (clase Recipe2).

- **Ingredientes (Ingredient):** Se trata de los ingredientes para cada receta. Por lo tanto depende de la clase receta con una relación de 1:M. Sus campos más importantes son su *id* (clave primaria), *name* que es el nombre del ingrediente y *cost* que corresponde a su coste, siendo estos dos último campos requeridos.
- **Recetas base (Recipe2):** Esta clase es una extensión de recipe, que sirve para identificar las recetas usadas como ingredientes en otras, esto es una receta base. Su relación con la clase receta se presenta de 1:M.

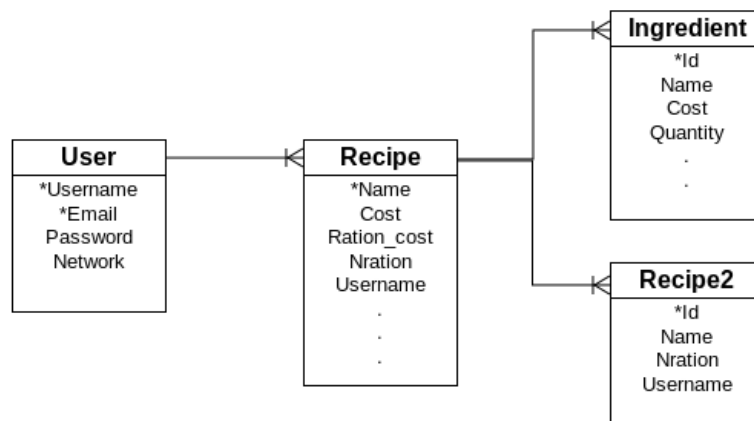


Figura 2.4: Modelo entidad-relación

Los cambios producidos en una clase puede implicar la modificación de las clases con las que está relacionada. Por ejemplo, si se elimina un usuario, se debe eliminar sus recetas y en consecuencia los ingredientes de esa receta. El modelo propuesto se ha diseñado de tal forma que sea capaz de realizar todas las acciones del CRUD, acrónimo de **C**rear, **O**btener, **A**ctualizar y **B**orrar (Create, Read, Update and Delete) para cada clase.

Puede ver con detalle el código fuente del modelo de la aplicación en `model.rb`.

Capítulo 3

Gestión de la configuración

3.1. PaaS

En el capítulo de introducción 1 se explicó los cambios en la elección del entorno de producción, así como su estudio. En este capítulo se explicará los dos entornos de producción escogidos, cómo funcionan y se configuran.

Antes de empezar debe quedar claro los modelos de servicios que ofrecen las nubes. Las empresas ofertan distintos servicios y prestaciones, de los cuales destacan:

- **SaaS:** Software como servicio. Se trata de cualquier servicio basado en la web. En este tipo de servicios accedemos normalmente a través del navegador sin atender al software. Todo el desarrollo, mantenimiento, actualizaciones, copias de seguridad es responsabilidad del proveedor. Son ejemplos conocidos *Google docs*, *Hotmail* o *Dropbox*.
- **PaaS:** Plataforma como servicio, es una encapsulación del entorno de desarrollo y un conjunto de módulos con el fin de proporcionar una funcionalidad que se traduce como servicio. En este modelo de servicio al usuario se le ofrece la plataforma de desarrollo y las herramientas de programación por lo que puede desarrollar aplicaciones propias y controlar la aplicación, pero no controla la infraestructura. Por ejemplo, *Heroku*, *Google App Engine* o *Windows Azure*.
- **IaaS:** Infraestructura como servicio. Tendremos más control que con PaaS, lo que implica la gestión de la infraestructura. Es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas se concentran para manejar tipos específicos de cargas de trabajo. Algunos ejemplos son: *Amazon Web Services*, *Google Cloud Storage* y *VMware*.

Para esta aplicación, trabajaremos con el modelo PaaS. Normalmente, para interactuar con la PaaS elegida, el proveedor proporciona una herramienta. Esta cambia en función de la nube elegida, pero su funcionamiento y opciones son similares: descargar la aplicación de un determinado repositorio y hacerla correr

en la nube a modo de producción. Además ofrecen distintos servicios o módulos, normalmente de pago. En Heroku se conoce como *dynos* y en OpenShift se llama *cartridge*. Éstos pueden ser la base de datos con la que interactuará la aplicación, mejoras en el equipo servidor (nube de producción) como mayor capacidad de disco, memoria RAM, CPU, sistema dedicado, incluso aumentar el ancho de banda para atender más peticiones.

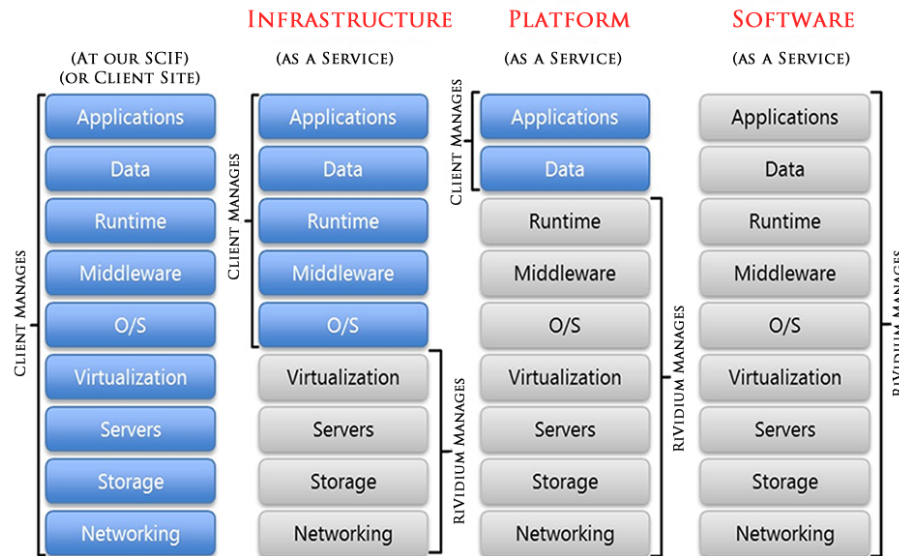


Figura 3.1: Modelos de nubes

3.2. Heroku

Se trata de una plataforma de aplicaciones cloud que permite construir y desplegar aplicaciones web. Soporta distintos lenguajes y frameworks. Para poder usar esta plataforma lo primero es registrarse y acceder a la misma. Desde la pantallas de *dashboard* podremos gestionar las aplicaciones o bien mediante los comandos de su herramienta *Heroku Toolbelt* ([?]). Existen tres formas de configurar nuestra aplicación en Heroku.

- El **dashboard**: Desde esta pantalla seleccionamos la app deseada y encontramos un menú con una serie de opciones que nos permiten gestionar la aplicación en la nube: añadir add-ons, activar *dynos*, seleccionar repositorio, información como métricas y actividad del servidor, permiso de acceso a la app y la configuración propia de la app (variables, dominio, etc.).
- Mediante **Heroku Toolbelt**: Este comando (`$ heroku COMMAND`) tiene múltiples opciones que permiten interactuar con la consola desde consola. Subir app, configurar la bbdd y variables, comprobar logs del servidor,

etc. En resumen, las opciones que nos proporciona el *dashboard* en el navegador, pero desde consola. Ejecute `$ heroku -h` para acceder a la ayuda y opciones.

- A través de **archivos de configuración**: Esta opción afecta más a la configuración de la propia aplicación. Aquí podemos crear variables, base de datos y otros, sin embargo, no se gestionan los dynos o add-ons pero sí usarlos o interactuar con ellos si están creados.

Para integrar la base de datos en producción podemos hacerlo de dos formas:

- Utilizar la herramienta con el comando: `$ heroku addons:create heroku-postgresql`.
- O bien, desde el *Dashboard* → Settings → Config Variables.

Para visualizar la configuración general usaremos el comando `$ heroku config`. Con esta variable ya podemos crear el adaptador para la aplicación. En este proyecto se trabaja con DataMapper ([?]) para el manejo y creación de la bbdd. De forma general, su configuración es:

```
configure :emph{ENTORNO} do
  | DatMapper.setup(:default, ENV['VARIABLE'])
end
```

Figura 3.2: Configuración de la BBDD con DataMapper

donde:

- *ENTORNO*: es el entorno para el que se está configurando: desarrollo, test o producción.
- `default`: la configuración por defecto. Podemos establecer un archivo de configuración por defecto. Las nubes de producción tienen el suyo, aunque a veces hay que adaptarlo a nuestras necesidades o crear nuestra propia configuración.
- `ENV['VARIABLE']`: corresponde a una variable que guarda la ruta a la base de datos. También puede ponerse la ruta directamente pero no es recomendable.

En nuestro caso sería:

```
configure :production do
  | DatMapper.setup(:default, ENV['OPENSIFT_POSTGRESQL_DB_URL'])
end
```

Figura 3.3: Configuración de la BBDD para Chefmanagement

Debido a que en OpenShift no puede modificarse el nombre de las variables, la renombramos en Heroku de igual forma para utilizar una única variable de configuración que es válida para ambos entornos de producción a la vez. Cada servidor contendrá este nombre de variable con un adaptador distinto.

3.3. OpenShift

Es otra PaaS similar a la anterior montada en Linux Red Hat. Permite desarrollar rápidamente, hosting y la escalabilidad de aplicaciones en entornos de la nube. Al contrario que Heroku, en la versión *free* permite crear hasta tres aplicaciones (y la primera cinco), sin embargo, aunque Heroku solo permite postgresql en esta versión, OpenShift permite escoger entre MongoDB, MySQL y PostgreSQL. Para **ChefManagement**, hemos tenido que adaptarnos al cuello de botella de Heroku en cuanto bbdd y trabajaremos con postgres en ambas plataformas.

Aunque OpenShift no dispone de tantas opciones de configuración en su *dashboard* si que permite añadir *cartridges* incluso creados por nosotros mismos. Por ejemplo, un cartucho ([?]) para hacer funcionar Jruby en OpenShift.

En cuanto a la herramienta del cliente ([?]) permite la gestión con la nube de igual forma que con la herramienta de Heroku. Sin embargo, en esta nube, no se puede añadir ni modificar variables. La nube almacena la configuración por defecto en caso de que la proporcionada no sea válida. En todo caso habría que usar su herramienta o archivos de configuración.

Los archivos de configuración son muy útiles si no queremos usar las variables por defecto que proporciona la nube. Hay dos formas de editar estos archivos, mediante la herramienta cliente o bien editando manualmente estos archivos de configuración ([?]).

Para configurar el adaptador de la base de datos en OpenShift podemos crear la variable ([?]) de la siguiente forma:

```
$ rhc env set VARIABLE=VALUE -a App_Name
```

Por defecto en esta nube esta información esta almacenada en la variable `OPENSIFT_POSTGRESQL_DB_URL`. Podemos acceder a la nube y consultar todas las variables mediante ssh:

```
$ rhc ssh APLICACION
```

Para ver la lista de variables creadas escribimos en consola `$ rhc env-list -a chefmanagement`, y para ver todas las variables de entorno nos conectamos por ssh y usamos el comando *env*.

Capítulo 4

Metodología de desarrollo

En el capítulo 1 se explicó la metodología ágil. En este capítulo repasaremos esta metodología usando como ejemplo el desarrollo de **Chefmanagement** y también el modelo-vista-controlador y su estructura.

4.1. Metodología ágil

Se había comentado anteriormente el uso de la metodología ágil, esto es debido a la naturaleza del proyecto: se trata de crear una aplicación en un período determinado y cuyas características no están claras, lo cual se debe a que los requisitos no están definidos, no existe un cliente con una necesidad, en este caso se trata de experimentar. Dentro de este tipo de metodologías, se ha usado *Scrum* para obtener resultados rápidamente y donde los requisitos pueden cambiar en cada iteración. La necesidad de innovar y la flexibilidad responden bien con esta metodología.

Hay que tener en cuenta, que debido a las circunstancias de este proyecto (no hay un equipo de desarrollo ni un cliente propiamente dicho) se ha adaptado *Scrum* a las necesidades, sin embargo, esta metodología se ha realizado de forma estricta en reuniones con el tutor y en la elaboración de la aplicación. A continuación se explica de forma general una iteración durante el desarrollo de esta aplicación:

- **Planificación:** Cómo me iba a organizar el trabajo durante esa semana. Planificar tiempo a cada una de las siguientes etapas. Corresponde a las reuniones semanales con el tutor y programar los nuevos requisitos en función de los resultados obtenidos.
- **Análisis de requisitos:** Cada vez que se proponía una idea ésta era analizada con el fin de comprobar su viabilidad en la aplicación. Se trata del estudio de campo e investigaciones específicas según la tarea planificada.
- **Diseño:** Cada vez que se desee agregar una funcionalidad al programa se debe estudiar el cambio en el diseño de la misma, de forma que optimice su usabilidad. El diseño también hace referencia a la forma en la que se va

a crear los distintos modelos de la base de datos y sus relaciones. Crear la estructura del proyecto de forma que sea escalable.

- **Codificación:** Parte del tiempo en el que se desarrollaba código. Durante el estudio de campo esta etapa se destino a crear programas sencillos de prueba.
- **Revisión:** Comprobar la aplicación hasta el punto actual. Tanto las características nuevas como las anteriores. También se incluyen las pruebas de código.
- **Documentación:** Durante el tiempo de vida del proyecto se ha ido elaborando un diario, en él se guarda la información de los resultados obtenidos semanalmente y expuestos durante las reuniones: lo qué se ha averiguado, lo qué se ha conseguido, los problemas encontrados y las soluciones propuestas.

Una de las ventajas más importantes del uso de esta metodología es que volver atrás o realizar cambios en la aplicación, conlleva consecuencias gracias a las iteraciones cortas.

4.2. Estructura de la aplicación

La aplicación ha sido diseñada y desarrollada de acuerdo con el framework de Sinatra ([?]) cuya arquitectura software se base en el MVC. Por un lado define componentes para la representación de la información, y por otro la interacción del usuario. La filosofía de este modelo es la reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

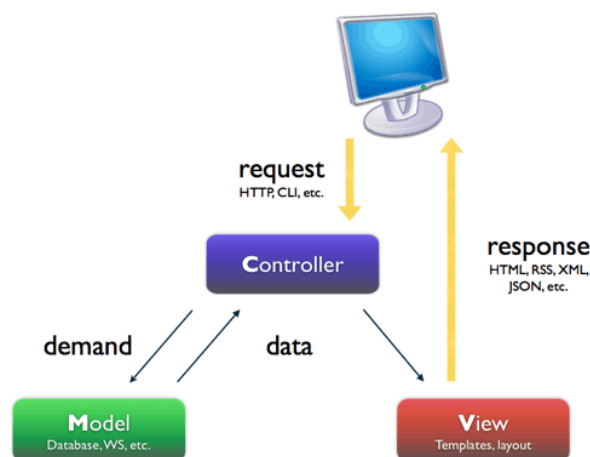
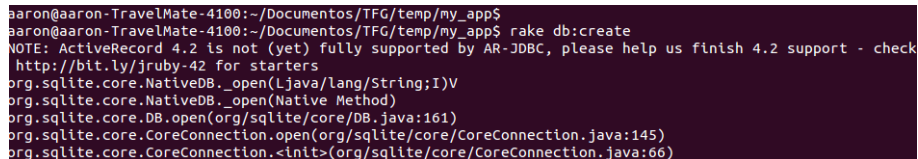


Figura 4.1: Modelo-Vista-Controlador

Durante la etapa de análisis y estudio de campo se empezó diseñando en Ruby on Rails [26], otro framework en lenguaje Ruby, sin embargo, uno de los requisitos era que la aplicación sea soportada bajo JRuby. Durante las pruebas muchas de gemas utilizadas en Rails presentaban problemas de incompatibilidad por lo que había que bajarlas de versión, y esto a su vez con otras y la aplicación. Es el caso de la gema *ActiveRecord*.



```
aaron@aaron-TravelMate-4100:~/Documentos/TFG/temp/my_app$
aaron@aaron-TravelMate-4100:~/Documentos/TFG/temp/my_app$ rake db:create
NOTE: ActiveRecord 4.2 is not (yet) fully supported by AR-JDBC, please help us finish 4.2 support - check
http://bit.ly/jruby-42 for starters
org.sqlite.core.NativeDB._open(Ljava/lang/String;I)V
org.sqlite.core.NativeDB._open(Native Method)
org.sqlite.core.DB.open(org/sqlite/core/DB.java:161)
org.sqlite.core.CoreConnection.open(org/sqlite/core/CoreConnection.java:145)
org.sqlite.core.CoreConnection.<init>(org/sqlite/core/CoreConnection.java:66)
```

Figura 4.2: ActiveRecord 4.2 no está soportado completamente por JRuby

La idea de la compatibilidad con JRuby se considera importante para este proyecto no solo para ampliar el soporte de la aplicación, también en futuros módulos como el desarrollo en plataformas móviles o tablets con Ruboto ([?]), cuyas apps pueden ser desarrolladas usando JRuby, MRI o Rubinius.

Por esta razón, y la libertad en diseño de la aplicación que proporciona Sinatra frente a Rails, así como su sencilla puesta en marcha, se optó finalmente por este framework para el desarrollo de Chefmanagment. La estructura que se diseñó es la siguiente (también puede verse en [11]):

- **app:** la codificación modelo-vista-controlador se encuentra aquí. Contiene el modelo, el controlador y las vistas de la aplicación, así como funciones auxiliares (helpers) y los tests.
 - config.yml: contiene los tokens e identificadores para las distintas APIs y correo usado por la aplicación.
 - controllers: gestiona la aplicación en función de las peticiones del usuario e interactúa con el modelo y las vistas (salida al usuario).
 - helpers: son funciones auxiliares definidas específicamente para esta aplicación. Cabe destacar el uso de la gema Mail ([?]) para la gestión de acceso, registro y notificaciones de la aplicación.
 - models: diseño de la base de datos. Más información en la sección 2.6
 - tests: la definición y declaración de las pruebas. Además de comprobar las funciones y métodos creados, se ha utilizado la herramienta Selenium ([?]) para interactuar con la propia aplicación y ponerla a prueba. Puede ver los resultados en Travis.
 - views: su función es interactuar con el usuario, es el resultado de las acciones del usuario con la aplicación, en un formato que éste pueda entender, como páginas webs.
- **public:**
 - css: el estilo de las vistas.
 - js: los archivos javascripts que controlan las entrada de datos e informan al usuario.

- uploads: está destinada a almacenar los archivos importados y exportados de las listas de recetas de cada usuario.
 - resources: carpeta para almacenar recursos como imágenes, etc.
 - otros: otras carpetas de apis usadas en la aplicación.
- .travis.yml: archivo de configuración para realizar las pruebas de la aplicación en un servidor remoto. Se trata de un modelo de integración continua. Se configuró de forma que se realizaran las pruebas en distintos navegadores: Firefox (por defecto) y Chrome.
 - Gemfile: contiene las gemas configuradas en los distintos entornos (desarrollo, producción y tests) para que la aplicación pueda funcionar.
 - Procfile: archivo de configuración de inicio del servidor en las nubes de producción.
 - Rakefile: contiene las distintas tareas y opciones que se pueden realizar.
 - config.ru: inicia la aplicación bajo *rackup*.

Capítulo 5

Conclusiones y trabajos futuros

A lo largo de este documento hemos visto los distintos aspectos relacionados con el desarrollo de **Chefmanagement**, utilizando la metodología ágil. Se han usado los conocimientos adquiridos durante la formación académica, esto incluye la investigación y análisis y la práctica real en proyectos y empresas.

Gracias a la experiencia adquirida en las prácticas de empresa he dado importancia al estudio de mercado, lo que me ha llevado a entrevistas con distintos restaurantes y proveedores y saber sus necesidades sobre la aplicación. Así mismo, la idea de innovar en este tipo de aplicaciones incluye un estudio de campo de las mismas.

También hay que tener en cuenta la investigación de las nubes de producción, las pruebas y analizar cual se adapta mejor a las necesidades de este proyecto.

Resumiendo este proyecto recoge protocolos y metodologías como: Scrum (metodología ágil), uso de frameworks basado en MVC, TDD (locales y en la servidores remotos), uso de repositorios y programación en la nube.

Se ha introducido una nueva PaaS (OpenShift) y se ha desarrollado una nueva aplicación con perspectivas de seguir evolucionando, procurando un diseño escalable e incluso exportable a otras arquitecturas.

A continuación se propone mejoras para incluir en módulos de futuras versiones de la aplicación:

- Integrar APIs de precios de productos de proveedores (Mercadona, Alcampo, etc). Esto permite un precio real y actualizado de la receta. Aunque no está integrada en la aplicación se propone usar el listado de productos (en formato json) de **Mercatenerife**.
- Actualizar y mejorar el diseño con temas personalizables.
- Incluir en el modelo información sobre el plato (kcal, proteínas, hidratos, grasas, etc.).
- Añadir a la funcionalidad *Calculadora* evaluar los **costos de menús**.
- Desarrollo en dispositivos smartphones y tablets, aprovechando la compatibilidad con JRuby se propone desarrollar en **Ruboto**.
- La aplicación puede diversificarse atendiendo a su mercado: **profesional** (negocios de restauración) o enfoque de **red social**.
- Para el mercado de red social se propone:
 - **Sistema de puntuación:** comprar recetas o conseguir a través de las puntuaciones que consigas compartiendo tus recetas.
 - **Otras características de red social:** compartir recetas, marcar como favoritos, comentarios, etc.
 - **Estadísticas:** platos-puntos, usuarios-recetas, platos-precio, etc.
- Para el mercado de negocio:
 - **Buscar** un determinado ingrediente en un determinado momento. Ver el precio actual.
 - **Comprar** al proveedor a través de la aplicación.

Capítulo 6

Summary and Conclusions

Capítulo 7

Presupuesto

En este capítulo abordaremos el caso comercial de puesta en marcha de la aplicación propuesta creando un presupuesto para su desarrollo. Vamos a partir del supuesto siguiendo los datos de la asignatura de TFG de un trabajo realizado de 300 h. y un precio de 12 €/h.

Como gastos en producción se propone dos entornos a elegir uno:

■ Opción 1: Heroku

Servicio	Base	Subtotal	Comentario
Horas trabajadas	300	3600	300 h x 12 €/h
Heroku	67.34 (€/mes)	808.08	Se calcula el precio para un año. Se ha elegido la opción estándar.
Confinación	60	60	Gastos para preparar y configurar el entorno de producción.
Otros gastos	60	180	Gastos destinados a entrevistas. En este caso 3.
Mantenimiento	300 (€/año)	0	Este servicio es gratis el primer año.
TOTAL		4648.08 €	

Estos precios son variables en función del tiempo y las prestaciones contratadas, puede calcularse de forma online en Heroku, aunque el precio se calcula en dólares americanos.

■ Opción 2: OpenShift

Servicio	Base	Subtotal	Comentario
Horas trabajadas	300	3600	300 h x 12 €/h
OpenShift	15 (€/mes)	180	Se calcula el precio para un año. Se ha elegido la opción estándar base.
Confiración	60	60	Gastos para preparar y configurar el entorno de producción.
Otros gastos	60	180	Gastos destinados a entrevistas. En este caso 3.
Mantenimiento	300 (€/año)	0	Este servicio es gratis el primer año.
TOTAL		4020 €	

En el caso de esta PaaS podemos consultar sus precios según las necesidades. El precio está calculado para el plan *Silver*.

Capítulo 8

Glosario

CMS: [1.1] Es un sistema de gestión de contenidos. Se utiliza para la creación y administración de contenidos, principalmente en páginas web, por parte de los administradores, editores, participantes y demás usuarios. Algunos ejemplos conocidos son *Joomla* y *Wordpress*.

CRUD: [2.7] Acrónimo de Crear, Obtener, Actualizar y Borrar.

Escandallo: [1.1] Cálculo del precio de un producto en relación con sus componentes. En el caso de la restauración, cálculo de producción de una receta en función de sus ingredientes.

GAE: [1.2] Google App Engine.

MVC: [4.2] Modelo-vista-controlador.

Nube (cloud): [1.2] Es el entorno de producción. Se trata de un servidor que ofrece unos determinados servicios o software.

PaaS: [3.1] Plataforma como servicio.

SaaS: [3.1] Software como servicio.

Scrum: [4.1] Es un tipo de metodología ágil.

TDD: [1.1] Desarrollo dirigido por pruebas.

Bibliografía

- [1] 1 and 1 (cloud). <http://www.1and1.es/>.
- [2] Amazon (cloud). https://aws.amazon.com/es/?nc2=h_lg.
- [3] Appfog (cloud). <https://www.appfog.com/>.
- [4] Azure (cloud). <http://azure.microsoft.com/en-us/>.
- [5] Cloud66 (cloud). <http://www.cloud66.com/>.
- [6] Cloud9 (cloud). <https://c9.io/>.
- [7] DigitalOcean. <https://www.digitalocean.com/community/>.
- [8] Documentación GAE. <https://cloud.google.com/appengine/docs>.
- [9] dotCloud (cloud). <https://www.dotcloud.com/>.
- [10] Gema GAE. <https://rubygems.org/gems/google-appengine>.
- [11] GitHub (Repositorio). <https://github.com/alu0100207385/ChefManagement>.
- [12] Go en GAE. <https://cloud.google.com/appengine/docs/go/gettingstarted/introduction>.
- [13] Heroku (cloud). <https://www.heroku.com/>.
- [14] Java en GAE. <https://cloud.google.com/appengine/docs/java/gettingstarted/introduction>.
- [15] JRuby. <http://jruby.org/>.
- [16] JRuby GAE. <https://code.google.com/p/appengine-jruby/>.
- [17] Lenguajes Google Cloud. <https://cloud.google.com/launcher/explore>.
- [18] Nitrous (cloud). <https://www.nitrous.io/>.
- [19] OpenShift (cloud). <https://www.openshift.com/\protect\begin\group\immediate\write\@unused\def\MessageBreak\let\protect\edefYourcommandwasignored.\MessageBreakTypeI<command><return>to replace it with another command,\MessageBreak or<return>to continue without it.\errhelp\let\def\>

```
MessageBreakΩ(inputenc)\def\errmessagePackageinputencError:
Unicodechar\u8:notsetupforusewithLaTeX.
ΩΩSeetheinputencpackagedocumentationforexplanation.
ΩTypeH<return>forimmediatehelp\endgroup.
```

- [20] Pearson Kitchen Manager. <http://developer.pearson.com/apis/pearson-kitchen-manager>.
- [21] PHP en GAE. <https://cloud.google.com/appengine/docs/php/gettingstarted/introduction>.
- [22] Python en GAE. <https://cloud.google.com/appengine/docs/python/gettingstartedpython27/introduction>.
- [23] Recipe Cost Calculator. <https://recipecostcalculator.net/>.
- [24] Recipe Costing. <http://www.recipe-costing.com/>.
- [25] Recipe Costing Calculator. <https://itunes.apple.com/es/app/recipe-costing-calculator/id646877156?mt=8>.
- [26] Ruby on Rails framework (Ruby). <http://rubyonrails.org/>.