UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA DE INGENIERÍA INFORMÁTICA

PROTOTIPO DE APLICACIÓN DE RECOMENDACIONES SOBRE LA DIETA MEDITERRÁNEA

Autor: Héctor Acosta González

Tutor: Agustín Trujillo Pino

Cotutor: Marcos León Martín

Las Palmas de Gran Canaria, Enero 2015

ÍNDICE

1 ESTADO ACTUAL Y OBJETIVOS

Dentro del marco de la revolución tecnológica que ha sucedido en las últimas dos décadas, uno de los hechos más importantes es el de la aparición de los denominados *smartphones*, cuyo éxito es tal que se ha convertido en el componente electrónico que nos acompaña la mayor parte de nuestro tiempo.

Dentro de la gran variedad de campos en los que la incursión de los teléfonos inteligentes ha supuesto una revolución, una de las más destacables es la de la salud. Hay cada vez una mayor comunidad de usuarios que busca aplicaciones que le permiten conocer a ciencia cierta datos sobre su actividad física, su estado de salud etcétera.

Esta aplicación intenta cubrir una de las necesidades principales en aplicaciones relacionadas con la salud: la dieta. Se ha demostrado científicamente que la dieta mediterránea es una de las dietas más saludables y completas, por lo que el contenido de la aplicación se ha centrado sobre la misma.

Como objetivos principales de este desarrollo, establecemos:

- Disponer de una aplicación ejecutable en dispositivos iOS.
- Permitir a un usuario, mediante la realización de un sencillo test, conocer el grado de adecuación de su alimentación a la dieta mediterránea.
- Aconsejar al usuario sobre cambios en su dieta mediante un menú personalizado, que tenga en cuenta las alergias que padece el usuario.
- Enseñar al interesado qué medidas puede llevar a cabo para cocinar mejor los alimentos, y por tanto, que sean más saludables.
- Indicar qué cambios se pueden realizar en el día a día para complementar a la dieta con el objetivo de tener una vida más saludable.
- Tener una base de datos de alimentos, clasificable sergún una serie de criterios y accesible desde la aplicación.

Para poder lograr estos objetivos, hemos contado con la colaboración de estudiantes del último año de Medicina en la ULPGC, tutorizados a su vez por D/Luis Serra Majem.

2 JUSTIFICACIÓN DE LAS COMPETENCIAS ESPECÍFICAS CUBIERTAS

De todas las competencias que cubre la realización de este proyecto, cabe destacar:

G4. Transmitir información, ideas, problemas y soluciones a un público tanto especializado como no especializado.

Este trabajo se ha elaborado en colaboración con alumnos de sexto curso de Medicina de la Facultad de Medicina de la Universidad de Las Palmas de Gran Canaria. Además, está cotutorizado por la empresa Inventiaplus S.L, que ha proporcionado ayuda a nivel de herramientas y asesoría.

Por tanto, ha sido necesario que en todo momento se hayan realizado labores de comunicación tanto a un público no especializado (alumnos de medicina), especialmente a la hora de contextualizar y adaptar sus requisitos al desarrollo de la aplicación; como a un público especializado, particularmente a la hora de explicar problemas en el ámbito de desarrollo para buscar su solución.

T2. Capacidad para dirigir las actividades objeto de los proyectos del ámbito de la informática, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada.

A la hora de realizar proyectos informáticos de esta índole, es un aspecto crítico el llevar una planificación a todos los niveles: tanto a la hora de realizar procesos de análisis como en la propia construcción del software. Durante el desarrollo se han realizado tareas propias de la dirección de proyectos (reuniones con los interesados, selección de tecnologías…) como de dirección del propio desarrollo (establecer hitos, estimaciones temporales…).

T5. Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad, de acuerdo con los conocimientos adquiridos según lo establecido en el apartado 5 de la resolución indicada.

Para la elaboración de este trabajo se ha fijado un enfoque completo hacia las metodologías propias de la ingeniería de software, integrándolas lo máximo posible durante todo el desarrollo desde la captura de requisitos inicial hasta la integración final.

Se ha realizado un énfasis especial en el análisis y diseño previo a la propia implementación, con el objetivo de mantener unos criterios de calidad tanto en el prototipo final como en el propio proceso de desarrollo.

T6. Capacidad para concebir y desarrollar sistemas o arquitecturas informáticas centralizadas o distribuidas integrando hardware, software y redes, de acuerdo con los conocimientos adquiridos según lo establecido en apartado 5 de la resolución indicada.

El proyecto objeto de esta memoria ha constado en la elaboración de un sistema cliente-servidor, por lo que se ha aunado la integración del software desarrollado con el hardware tanto del servidor en el que está alojado el *back-end* como la aplicación que se ejecuta en el dispositivo.

Dada esta configuración, es necesaria la integración de las redes de comunicación para el intercambio de información entre las dos aplicaciones, además de la utilización de los protocolos de red necesarios para la implantación completa del sistema (tales como comunicaciones por FTP, SSH etcétera).

T8. Conocimiento de las materias básicas y tecnologías, que capaciten para el aprendizaje y desarrollo de nuevos métodos y tecnologías, así como las que les doten de una gran versatilidad para adaptarse a nuevas situaciones.

Las tecnologías utilizadas para este proyecto destacan en la versatilidad y facilidad a la hora de implementar los cambios, gracias en parte a la facilidad de adaptación de los patrones de diseño de la ingeniería del software.

Además, se ha optado por estas tecnologías en favor de otras teniendo especialmente en cuenta que la versión presentada para el proyecto es un prototipo con grandes signos de evolución y desarrollo no solo para alcanzar su fase de finalización del producto sino para características futuras.

CII02. Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

Este proyecto se ha basado en obtener un prototipo partiendo desde la fase de captura de requisitos, lo que ha permitido obtener experiencia en todos los ámbitos que se tratan en esta capacidad.

En todo momento se ha tenido en cuenta maximizar el impacto social de este desarrollo, realizando cambios que aportasen un mayor valor al producto sobre todo en el ámbito de la educación para la salud. También se han tomado en consideración las limitaciones económicas existentes a la hora de realizar el desarrollo.

TFG01. Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas

Cada una de las etapas de este trabajo ha tenido como objetivo la elaboración de esta memoria y posterior lectura ante el tribunal, intentando aunar la mayor cantidad de competencias adquiridas en los estudios de esta titulación.

3 APORTACIONES

Como se indicaba en la introducción de esta memoria, una de las principales corrientes de desarrollo de aplicaciones móviles en la actualidad tiene como protagonista el campo de la salud, destacando especialmente aplicaciones que se centran en la medida de coeficientes biométricos o recopiladores de estadísticas deportivas, todos buscan que sus usuarios mejoren su salud; todo ello sin contar con los dispositivos *wearables* que permiten obtener información minuto a minuto de nuestra actividad.

Sin embargo, esta prototipo busca informar al usuario de las alternativas y consejos que debe tener en cuenta a la hora de alimentarse, otro de los pilares básicos de la buena salud. A pesar de que existe una gran variedad de aplicaciones de recetas, ninguna está lo suficientemente enfocada en educar sobre qué alimentos son los más recomendables y sobre todo, cómo hay que cocinarlos.

Por otro lado, otra de las opciones que no se encuentran en gran parte de las aplicaciones ya existentes es la posibilidad de obtener recetas teniendo en cuenta las alergias que padece el usuario, lo que permite a cualquier persona que presente algún tipo de alergia o intolerancia, obtener un menú que se ajuste a su alimentación.

Centrándonos en un punto de vista más técnico, se ha establecido una estructura básica que permite categorizar con exactitud los alimentos que cocinamos. A pesar de que existen alternativas que siguen una estructura similar, no permiten hacer filtrados como alergias (muchas alergias no son aplicables únicamente a un ingrediente, sino a grupos de los mismos). Por tanto, sería posible tomar la estructura de base de datos existente actualmente y tomarla como base para un proyecto que le aporte una mayor potencialidad.

En el punto de vista social, el contar con expertos en este campo, tanto estudiantes como personas responsables de la Fundación Dieta Mediterránea posibilita una mayor repercusión de este proyecto en la sociedad, tanto para los especialistas en este campo como para el resto de la población.

4 DESARROLLO

En este apartado vamos a detallar cada una de las etapas de este proyecto, haciendo hincapié en qué puntos han resultado más conflictivos y cómo se han solucionado:

4.1 HERRAMIENTAS DE DESARROLLO

Para el desarrollo de este trabajo, se ha hecho uso de las siguientes herramientas y entornos de desarrollo:

XCODE

Para el desarrollo del prototipo para el dispositivo, se ha utilizado XCode teniendo especialmente en cuenta las restricciones de Apple a la hora de generar compilaciones, ya que requiere esta aplicación para poder generar el fichero final que se envía a la App Store o a los dispositivos de desarrollo.

XCode es el principal IDE para aplicaciones dirigidas a sistemas operativos de Apple. Desde él podemos emular los diferentes dispositivos de la compañía, facilitando el trabajo de detección de errores y pruebas. Además, facilita en gran medida el proceso de creación de la interfaz gráfica de la aplicación, realizándose de una manera rápida e intuitiva.

XCode está disponible de forma gratuita para sistemas operativos OS X a través de Mac App Store, y no tiene ninguna restricción a la hora de desarrollar y ejecutar (no así a la hora de poder probar las aplicaciones en dispositivos reales, ya que requiere de unos certificados de desarrollador sólo obtenibles disponiendo de una cuenta de Apple para desarrollo).

OBJECTIVE-C

Objective-C es el lenguaje de programación utilizado para construir la aplicación de iPhone. Es un lenguaje orientado a objetos utilizando la base de C y tomando el modelo de objetos similar a Smalltalk.

Al ser un superconjunto de C, permite la compilación de C, por lo que es posible incluir código escrito en este lenguaje en una clase de Objective-C. Por otro lado, su modelo de objetos nos aporta ventajas tales como el tipado dinámico, a costa de un mayor tiempo de ejecución con respecto a otros lenguajes derivados de C como C++.

Cabe destacar que la utilización de Objective-C para el desarrollo de aplicaciones en sistemas Apple ha sufrido importantes cambios en los últimos años, especialmente con la incorporación de ARC (*Automatic Reference Counting*), que permitió simplificar a los desarrolladores el manejo de la memoria, que antes se hacía manualmente utilizando las sentencias *retain* y *release*.

En la actualidad, Apple recomienda utilizar Swift para el desarrollo de sus aplicaciones; lenguaje de programación creado por ellos mismos. Cabe destacar que no se ha empleado este lenguaje en el trabajo debido a que su presentación se produjo cuando el desarrollo del mismo ya estaba bastante avanzado. Aun así, Swift es compatible en gran parte con Objective-C, lo que permitiría realizar ampliaciones de este prototipo utilizando Swift.

RUBY ON RAILS

Ruby on Rails es un *framework* para aplicaciones web de código abierto empleando Ruby. Este framework se caracteriza por la versatilidad y rapidez a la hora de desarrollar.

Para lograr esto ultimo, Rails se basa en:

- Paradigma MVC (Modelo Vista Controlador), facilitando el mantenimiento del código y su correcta estructuración.
- Principio DRY (*Don't repeat yourself*). Rails ofrece una integración entre componentes de tal manera que las definiciones se realicen una única vez en todo el código.
- Principio CoC (*Convention over configuration*), por lo que el programador solo va a tener que dedicar tiempo a la configuración de aquello que no sea convención, permitiendo reducir líneas de código y evitar dedicar más tiempo de desarrollo del necesario para realizer configuraciones.

La selección de este entorno de desarrollo se debe principalmente a la potencia de librerías como el ActiveRecord, que facilita en gran parte el mantenimiento y manejo de las bases de datos, otorgándole una gran versatilidad y facilidad de adaptación a cambios al software del lado del servidor.

TESTFLIGHT

Testflight es una plataforma online que busca facilitar el proceso de testeo de aplicaciones. Permite, mediante un sistema de invitación, acceder a las compilaciones que se han subido a la plataforma, evitando así tener que instalar el software dispositivo por dispositivo.

Testflight es especialmente útil cuando se están prototipando aplicaciones, ya que permite notificar a los interesados cuándo una nueva compilación está disponible. Además, una vez se han instalado la aplicación, permite que el desarrollador pueda obtener información total sobre qué versión de dispositivo y sistema operativo tiene instalada el usuario, lo que facilita el proceso de detección de errores.

Actualmente Testflight es propiedad de Apple y está en fase de integración con la tienda de aplicaciones de iPhone, permitiendo aún mayor facilidad de uso a los interesados.

HEROKU

Heroku es una PaaS (*Platform as a Service*) propiedad de Salesforce, utilizada en este proyecto como alojamiento para la aplicación de administración. Heroku es una de las plataformas principales de alojamiento de código Ruby en la nube, facilitando las tareas de mantenimiento y subida a producción del código que se ha desarrollado en un entorno local.

Heroku se caracteriza principalmente por la sencillez y la escalabilidad, adaptándose en todo momento a los requisitos de nuestro sistema. Así, sus servicios se dividen en *dynos*, que ajustan el plan de precios según el número de ellos que se utilice.

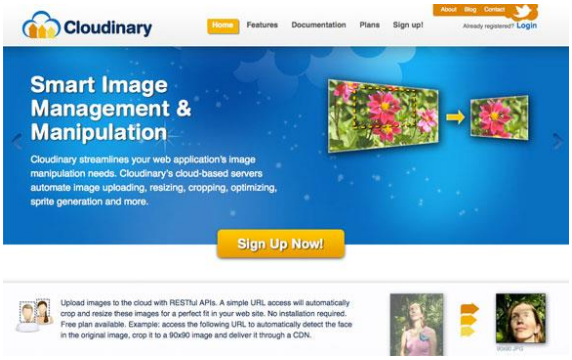
Cuenta con una opción de alojamiento gratuita con limitaciones especialmente en la base de datos, además de que los ficheros ajenos al propio código de la aplicación son eliminados cada día, por lo que su uso en este proyecto se ha limitado a servir como servidor de pruebas para este prototipo, teniendo que buscar un servidor aparte o bien pasar a una suscripción de pago para la aplicación final.

CLOUDINARY

Debido en parte a las limitaciones especificadas en el apartado de Heroku, era necesario utilizar un servicio que nos permitiese almacenar las imágenes sin que estas fueran eliminadas cada día.

Cloudinary es una PaaS orientada a la manipulación de imágenes. Ofrece una API para Ruby on Rails que permite controlar aspectos tales como la resolución o procesamiento de dichas imágenes, utilizando algoritmos que aseguran los mejores resultados.

Cloudinary cuenta con una versión gratuita que limita el número de imágenes y el ancho de banda mensual, por lo que de nuevo, hemos utilizado esta plataforma para la etapa de prototipado.



4.2 ANÁLISIS

Tras una etapa en la que se realizó una discusión sobre los requisitos que ambas aplicaciones deberían tener, se especificaron como requisitos funcionales:

- La aplicación de administración debe permitir crear recetas.
- La aplicación de administración debe permitir asignar ingredientes a las recetas.
- La aplicación debe permitir asignar alergias a esas recetas, para poder filtrar dependiendo de las del usuario.
- La aplicación del dispositivo permitirá realizar un test de adecuación a la dieta mediterránea.
- La app deberá pedir los datos relativos a las alergias del usuario.
- La aplicación deberá generar un menú que se ajuste a la dieta mediterránea, teniendo en cuenta las alergias del usuario.
- El software permitirá acceder a datos de recetas, con posibilidad de filtrar por categoría.
- La aplicación facilitará información de interés sobre la dieta mediterránea, con datos tales como las cantidades recomendadas o los alimentos adecuados de cada temporada.

Por otro lado, dada las características del proyecto que se iba a presentar se definieron como requisitos no funcionales:

- El prototipo estará disponible para plataforma iOS.
- La comunicación entre la aplicación y el servidor se realizará sobre una API REST.

Comenzamos la documentación del análisis definiendo cuáles son los casos de uso de este proyecto:

Nombre	**Registrar usuario**
Actor	Usuario
Precondiciones	
Poscondiciones	• Usuario registrado en el sistema.
Flujo	1. El usuario inicia la aplicación. 2. Se presenta la pantalla de inicio de sesión, pulsa sobre la opción “Registrarse”. 3. El usuario accede a la pantalla de registro en donde introduce sus datos: dirección de correo electrónico y contraseña. 4. Se presenta la pantalla de selección de alergias, el usuario selecciona las que padece y pulsa en “finalizar”

	5. Si se produce un registro correcto, se muestra la pantalla de introducción de la aplicación.
Flujos alternativos	3.1 Si se produce un error al introducir la información en la base de datos, se muestra un mensaje de aviso y se vuelve al paso 3. 3.1 En caso de que se produzca algún error al almacenar la infomación de las alergias, retorna al paso 4.

Nombre	**Iniciar sesión**
Actor	Usuario
Precondiciones	• El usuario está registrado en el sistema.
Poscondiciones	• Usuario con sesión iniciada.
Flujo	1. El usuario inicia la aplicación. 2. Accede a la pantalla principal, en la que se le pide datos de usuario y contraseña. 3. Usuario redirigido al menú principal de la aplicación.
Flujos alternativos	2.1 Si se produce un error al iniciar sesión, se muestra un mensaje de aviso y vuelve al paso 2.

Nombre	**Realizar test**
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión.
Poscondiciones	• El usuario obtiene una puntuación final.
Flujo	1. El usuario llega a la pantalla de presentación del test. 2. El usuario contesta a cada una de las preguntas seleccionando una opción y pulsando “Siguiente pregunta”. 3. Al llegar a la última pregunta, el usuario pulsa “Finalizar el test, donde se le presentará una pantalla con los resultados.

Nombre	Acceder a corrección del último test
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión. • El usuario ha completado un test.
Poscondiciones	• El usuario ha visualizado las preguntas con su solución.
Flujo	1. El usuario accede a las correcciones desde la pantalla de finalización del test o desde su perfil de usuario. 2. El usuario accede a cada una de las preguntas con la corrección correspondiente.

Nombre	Visualizar perfil
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión.
Poscondiciones	• Perfil de usuario visualizado con nota del último test.
Flujo	1. El usuario accede desde el menú principal a la opción de ver su perfil. 2. Se le presenta una pantalla con su nombre y la puntuación del último test.

Nombre	Obtener menú personalizado
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión. • El usuario ha completado el test inicial.
Poscondiciones	• Menú separado por semanas obtenido.
Flujo	1. El usuario accede a la opción de menú desde el menú principal o desde la pantalla de resultados del test. 2. Se presenta una pantalla en donde puede seleccionar la semana del mes que desee. 3. Cuando selecciona la semana, se le presentan las recetas separadas por día y plato.

Nombre	Ver receta
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión. • El usuario ha obtenido un menú o ha accedido a la lista de categorías.
Poscondiciones	• Visualización de la receta.
Flujo	1. El usuario pulsa la celda de la tabla correspondiente a la receta. 2. Se le muestra una pantalla desde la que puede ver todos los campos de la receta.

Nombre	Ver consejos
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión.
Poscondiciones	• Visualización de consejo.
Flujo	1. El usuario accede a la pantalla de consejos desde el menú. 2. Se le muestra una tabla con los diferentes consejos. 3. El usuario pulsa sobre un consejo y se le muestra la información.

Nombre	Ver alimentos de temporada
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión.
Poscondiciones	• Visualización de pantalla con Imágenes de los alimentos de temporada.
Flujo	1. El usuario accede a la opción de menú para ver los alimentos de temporada. 2. El usuario entra en una pantalla con pestañas separadas por estaciones. 3. El usuario puede cambiar de estación pulsando en otra pestaña.

Nombre	Ver cantidades recomendadas
Actor	Usuario
Precondiciones	• El usuario ha iniciado sesión.
Poscondiciones	• Visualización de pantalla con cantidades recomendadas.
Flujo	1. El usuario accede a la opción de menú de “cantidades recomendadas”. 2. Se muestra una pantalla con ilustraciones de las cantidades recomendadas para cada alimento.

Podemos observar como en todos los casos de uso de la aplicación, se requiere un registro previo del usuario, incluso en aquellas secciones que no dependan de la información del usuario para mostrar el contenido. Esto se debe a que los colaboradores de la aplicación desean obtener el máximo número de tests posibles para obtener datos estadísticos más completos y significativos, por lo que forzar el registro era una de las opciones principales.

En cuanto al *back-end*, se ofrece al administrador un CRUD que le permite agregar contenido de diferentes categorías (Recetas, Ingredientes, Alergias, Tipos de platos y Tipos de ración), como veremos posteriormente en el apartado de diseño de este documento. Por tanto, los casos de uso de este entorno son los siguientes cuatro, variando en cada caso dependiendo del contenido:

Nombre	Añadir contenido
Actor	Administrador
Precondiciones	
Poscondiciones	• Contenido añadido a la base de datos
Flujo	1. El administrador de la aplicación accede al portal y selecciona la categoría correspondiente en el menú. 2. Pulsa el botón “Añadir” de la categoría correspondiente. 3. Se presenta un formulario en el que rellena y asigna los campos correspondientes. 4. Pulsa sobre “Guardar” para almacenar el contenido. 5. La aplicación redirige al índice del contenido correspondiente.
Flujos alternativos	4.1 Si se produce algún error al guardar, vuelve al paso 3, mostrando un mensaje de error.

Nombre	Eliminar contenido
Actor	Administrador
Precondiciones	• Contenido a eliminar existente en la base de datos
Poscondiciones	• Contenido eliminado de la base de datos
Flujo	1. El administrador pulsa el botón “Eliminar” correspondiente al contenido que desea eliminar. 2. Se abre una ventana de diálogo pidiendo confirmación. 3. Tras aceptar el mensaje el contenido se elimina. 4. Se muestra el índice de la categoría correspondiente actualizado.
Flujos alternativos	4.2 El usuario cancela el mensaje. 4.3 Se cierra la ventana de diálogo, mostrándose el índice del contenido sin hacer ninguna modificación sobre el mismo.

Nombre	Editar contenido
Actor	Administrador
Precondiciones	• Contenido existente en la base de datos.
Poscondiciones	• Contenido guardado con campos actualizados.
Flujo	1. El administrador pulsa el botón “Editar” correspondiente al contenido que desea editar. 2. Se presenta el formulario de creación de contenido, con la información precargada de los campos que corresponden. 3. El administrador modifica la información que considere oportuna y hace click en el botón guardar.
Flujos alternativos	3.1 En caso de que se produzca un error al guardar la información, el sistema vuelve a presentar el formulario de edición y muestra el mensaje de error.

Nombre	Ver contenido
Actor	Administrador
Precondiciones	• Contenido existente en la base de datos.
Poscondiciones	• Se visualiza el contenido de los campos del contenido.
Flujo	1. El administrador pulsa el botón “ver” del contenido escogido. 2. Se presenta una vista donde puede ver los campos del contenido.

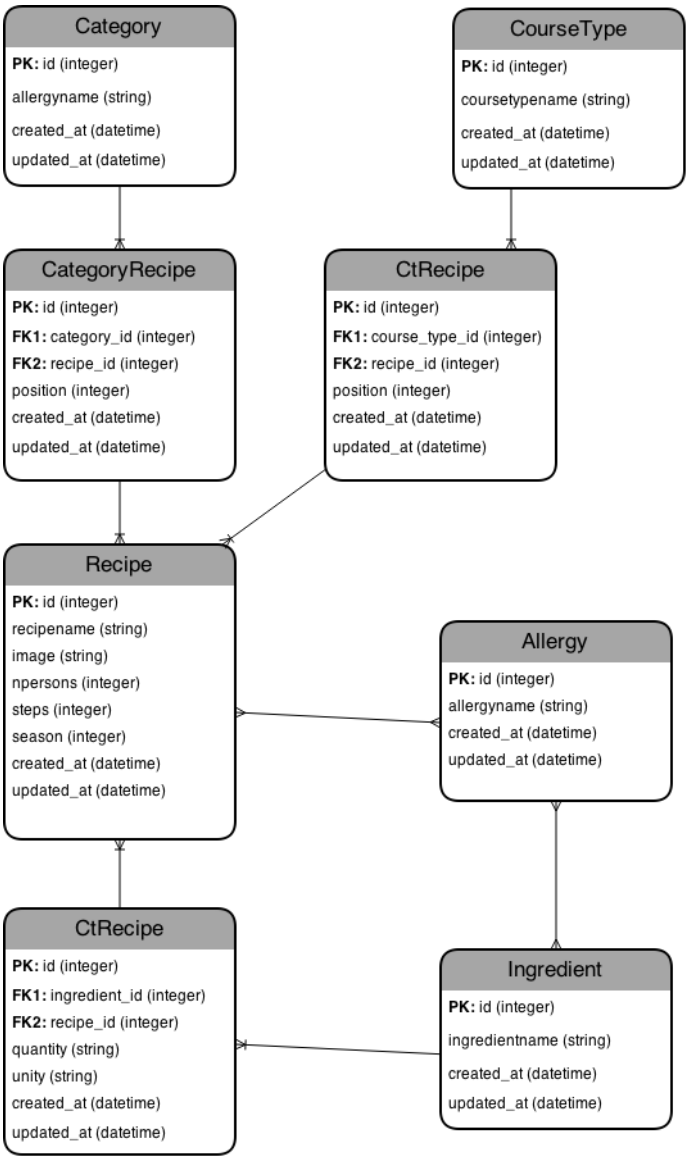
4.3 DISEÑO

Una vez detallada la fase de análisis, pasamos a detallar la fase de diseño del proyecto. En este caso, se comenzó el diseño desde la aplicación de administración, dada la importancia del mismo a la hora de construir la aplicación para el dispositivo móvil.

BACK-END

En la aplicación del *backend* se incluye toda la lógica de las recetas, que nos permitirá categorizarlas para establecer todos los filtros necesarios a la hora de realizar las peticiones a este servidor desde la aplicación del dispositivo.

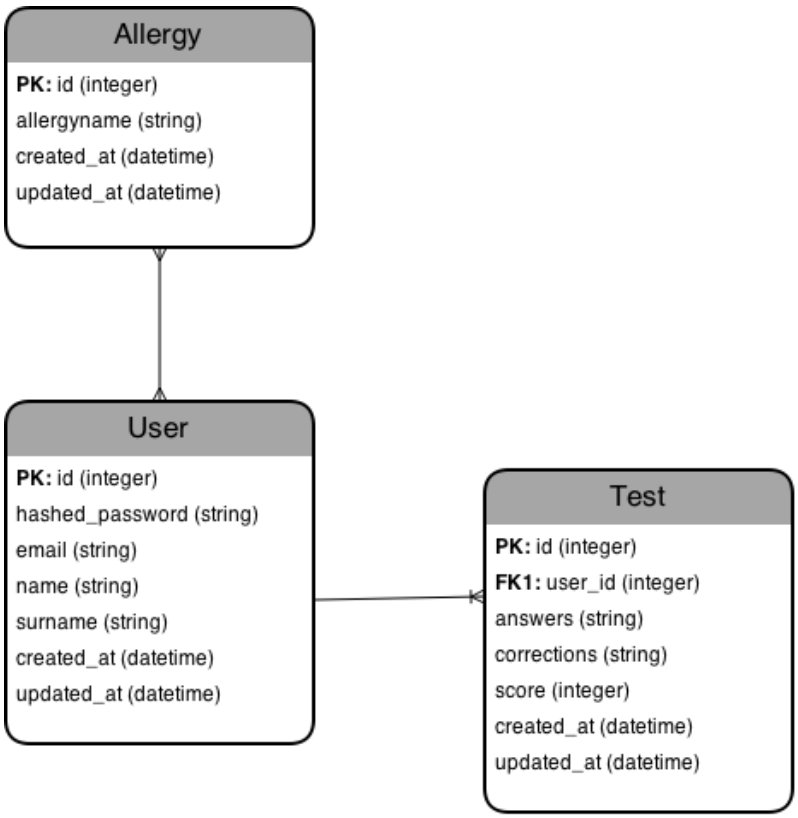
En esta vista podemos ver qué entidades y relaciones existen en el sistema relativas a las recetas. Justificamos a continuación la existencia de cada una de estas entidades:



- **Recetas(Recipes):** en esta tabla se encuentran todos los datos de la receta relativos a sus características y su elaboración.
- **Ingredientes(Ingredients):** en esta tabla se almacenarán todos los ingredientes que se den de alta en el sistema. De esa manera, el administrador puede añadir ingredientes a las recetas a través de relaciones.
- **Ingrediente-Receta(Ingredient-Recipe):** esta tabla intermedia une a las recetas con los ingredientes, permitiendo añadir más datos a la relación tales como la cantidad y unidad.
- **Categorías(Categories):** tabla utilizada para especificar las diferentes categorías de platos que están dados de alta en el sistema, y que se utilizarán para clasificar las recetas.
- **Categoría-Receta(CategoryRecipe):** tabla intermedia utilizada esencialmente para guardar la relación de orden de las categorías con respecto a las recetas. Este orden es necesario para los diferentes algoritmos que afectan a las recetas, ya que en algunos casos nos interesará únicamente la categoría principal.
- **Alergias(Allergies):** Tabla en la que se registran las diferentes alergias que se quieren tener en cuenta en la aplicación para generar los menús. Hay una relación existente entre las recetas y las alergias para un caso futuro en el que sea necesario asignar directamente la alergia a la receta. No obstante, para la generación del menú las alergias que se tienen en cuenta son aquellas que están asociadas a través de los ingredientes, y que serán visualizables desde la índice de ingredientes de la aplicación del lado del servidor.
- **Tipo de plato(CourseType):** En esta tabla se almacenan las diferentes tomas de alimentos que se producen durante el día, con el objetivo de especificar con mayor exactitud en qué comida es más adecuada tomar una receta.
- **Tipo de plato-Receta(Ct-Recipe):** Al igual que ocurría con las categorías, se requiere de una tabla intermedia para poder almacenar la relación de orden existente entre las categorías con respecto a las recetas.

Esta aplicación también es la encargada de almacenar los usuarios que forman parte de la aplicación. Estos usuarios son dados de alta mediante comunicaciones con la aplicación móvil, proceso que se explicará más adelante en este documento. En las relaciones con el usuario, tenemos:

- **Usuario(User):** Contiene todos los datos relativos al usuario que son requeridos desde la aplicación móvil.
- **Alergias(Allergies):** Esta tabla es la que contiene todas las alergias que se quieren hacer constar en el sistema, por lo que también existe una relación entre estas alergias y los usuarios registrados.
- **Tests:** En esta tabla se almacena en formato texto las respuestas y la corrección de cada uno de los tests, que se relacionan con el usuario.



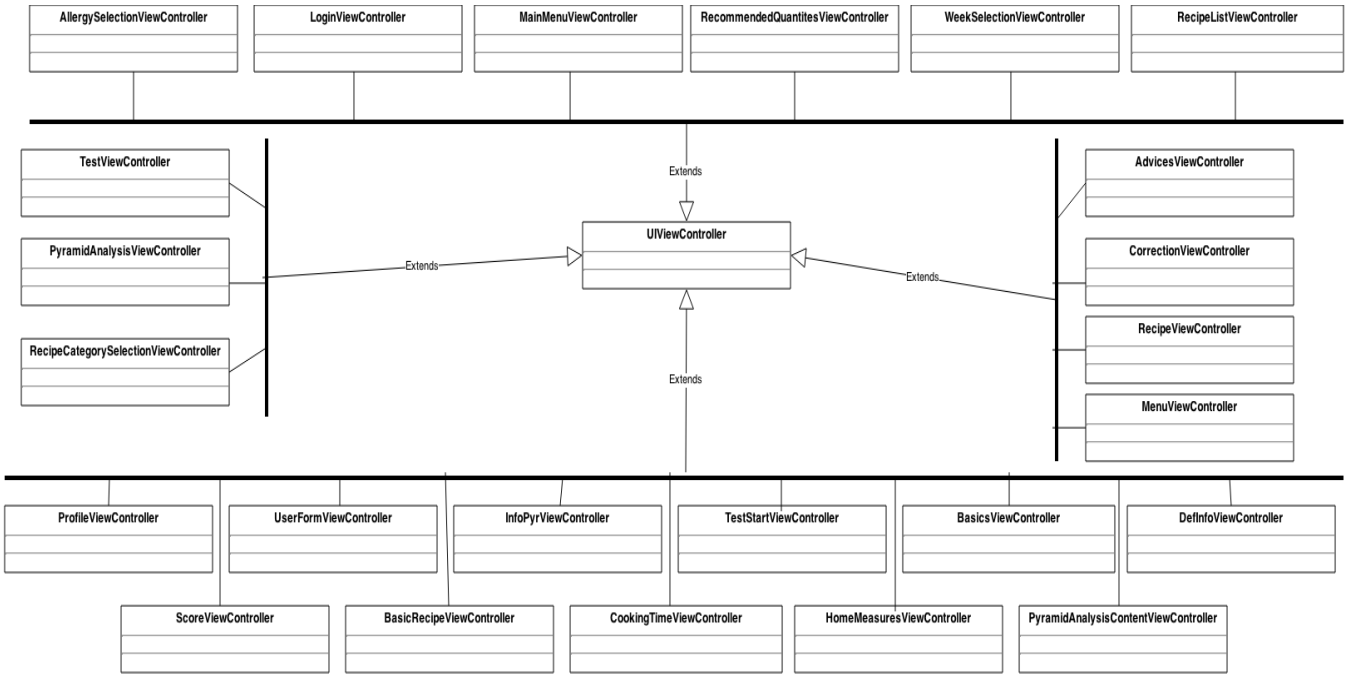
FRONT-END

Para la aplicación de dispositivos móviles, se ha buscado adaptarse lo máximo posible a los principios de la guía de desarrollo de aplicaciones para iPhone de Apple.

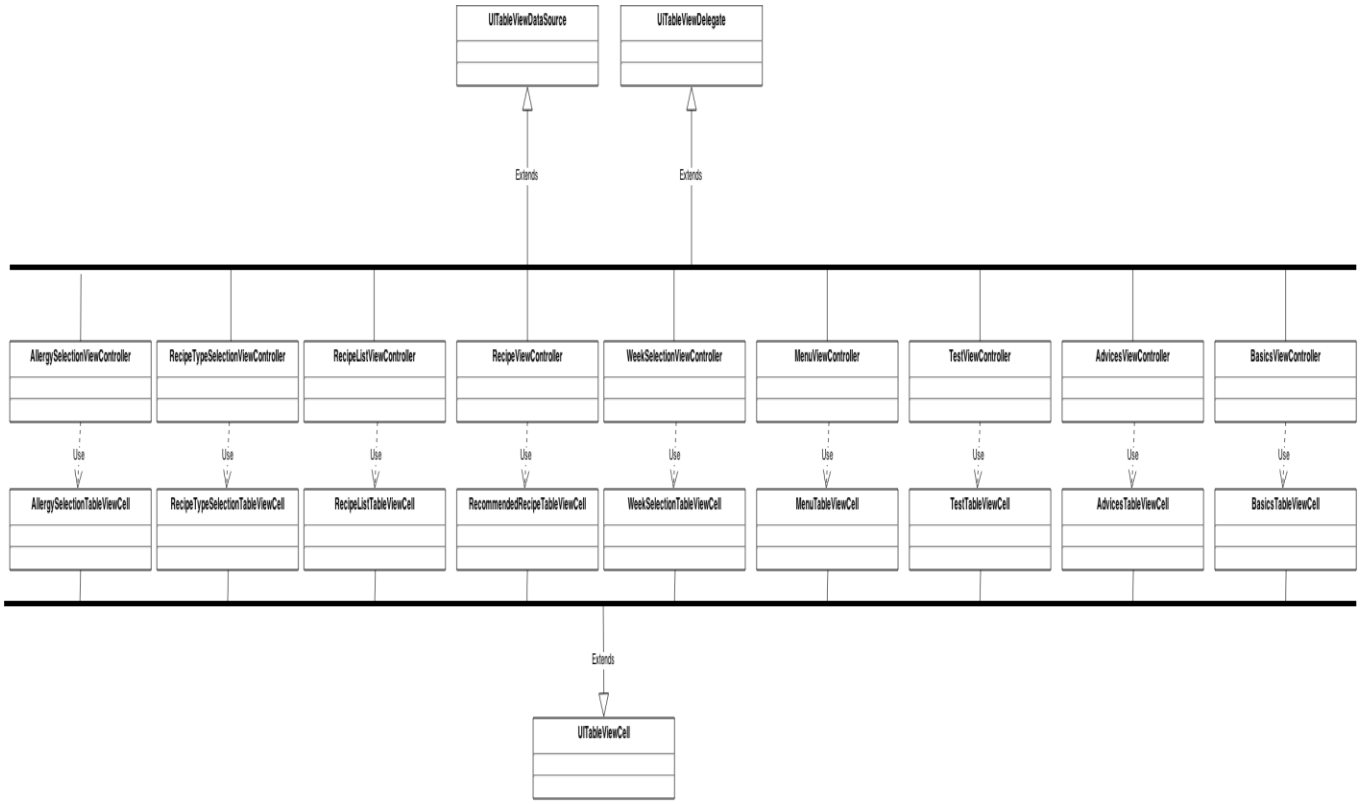
Concretamente, y con el objetivo de facilitar la construcción de las interfaces de usuario con el *Interface Builder* de Xcode, cada pantalla de la aplicación tendrá un controlador asociado que hereda de UIViewController. De esta manera, comentamos de manera breve la funcionalidad de cada una de las clases que forman parte del prototipo. Para ver las relaciones entre ellas, consulte el anexo 1:

- **LoginViewController:** Controlador para la pantalla de inicio de sesión en la aplicación.
- **ProfileViewController:** Controlador que se encarga del manejo de la vista de perfil del usuario.
- **UserFormViewController:** Es el controlador para el primer paso del registro del usuario, en el que se le pregunta sus datos personales.
- **AllergySelectionViewController:** Controlador para la pantalla de selección de alergias en el proceso de registro.
- **MainMenuViewController:** Contiene toda la lógica del menú principal, principalmente, llamar al controlador correspondiente a cada sección.
- **TestStartViewController:** Se encarga de ejecutar la lógica asociada a la pantalla de inicio del test.

- **TestViewController:** Controlador específico para el test, tanto a nivel de la interfaz del test como para la lógica de corrección del test.
- **ScoreViewController:** Lógica para la pantalla que muestra la puntuación final del test y que permite acceder a las otras opciones.
- **CorrectionViewController:** Controlador para visualizar la corrección del test.
- **RecommendedQuantitesViewController:** El controlador para la pantala de visualización de las cantidades recomendadas.
- **RecipeCategorySelectionViewController:** Este controlador será llamado cuando el usuario quiera acceder a las recetas sin necesidad de que éstas estén en el menú. Implementa la lógica necesaria para la pantalla de selección de categoría de alimento.
- **RecipeViewController:** Controlador para mostrar la información de las recetas almacenadas en el servidor.
- **WeekSelectionViewController:** Pantalla de selección de la semana cuando se solicita un menú.
- **MenuViewController:** Controlador para el menú semanal, en el que se mostrará cada uno de los platos con el día y toma correspondiente.
- **RecipeListViewController:** Controla la lógica de la pantalla de selección de recetas cuando se accede desde el listado de categorías.
- **PyramidAnalysisViewController:** Controlador para la visualización del apartado de desglose de la pirámide.
- **PyramidAnalysisContentViewController:** Controlador que se llama después de haber seleccionado un contenido en la anterior pantalla, por lo que muestra la información de esa categoría en concreto.
- **AdvicesViewController:** Controlador para el apartado de consejos, en donde el usuario seleccionará de cuál de ellos desea tener más información.
- **DefInfoViewController:** Primera pantalla de información cuando el usuario se registra.
- **InfoPyrViewController:** Pantalla de información de la dieta mediterránea que se presenta al continuar desde la pantalla DefInfo.
- **BasicsViewController:** Controlador que muestra las recetas consideradas básicas.
- **BasicRecipeViewController:** Controlador invocado al seleccionar una receta básica, encargado de mostrar los atributos de dicha receta.
- **HomeMeasuresViewController:** Controlador encargado de mostrar las imágenes para las medidas caseras.
- **SeasonalFoodViewController:** Controlador propio de la pantalla de alimentos de temporada.



Por otro lado, es recomendable que cada vez que se usen tablas, se cree una clase correspondiente para las celdas correspondiente. En el siguiente diagrama se muestran los controladores que utilizan esas clases para las celdas:



Por último, contamos con unas clases de modelo para aquellos objetos que se manipulen durante la ejecución del programa. En este caso:

- **User:** Singleton que contiene los datos del usuario. Cuando se realiza el registro o se inicia sesión, los atributos de este objeto se inicializan a los valores de respuesta con el servidor, con el objetivo de resultar accesible en todo momento en la ejecución de la aplicación.
- **Test:** Cuando el usuario accede a realizar el test, se inicializa un objeto de esta clase, en la que se crean los diferentes objetos de tipo Pregunta y Respuesta, almacenando también el registro de respuestas y correcciones del usuario. También contiene métodos para comprobar si la respuesta es correcta, llevando así además el control de la puntuación del test.
- **Question:** Clase para las preguntas del test:
- **Answer:** Clase para cada una de las respuestas del test, contiene además un campo que permite definir desde ese momento si dicha respuesta es correcta o no.
- **BasicRecipe:** Dado que las recetas básicas tienen campos diferentes a las recetas que se leen del servidor, y que además su lógica es diferente, se decidió crear una clase aparte para establecer una correcta diferenciación. Las recetas que se leen del servidor se cargan directamente en la vista del controlador.
- **BasicRecipeList:** Colección de recetas básicas que es utilizada en el controlador BasicsViewController.
- **PyramidContent:** Contenido para el apartado de "Desglosando la pirámide".

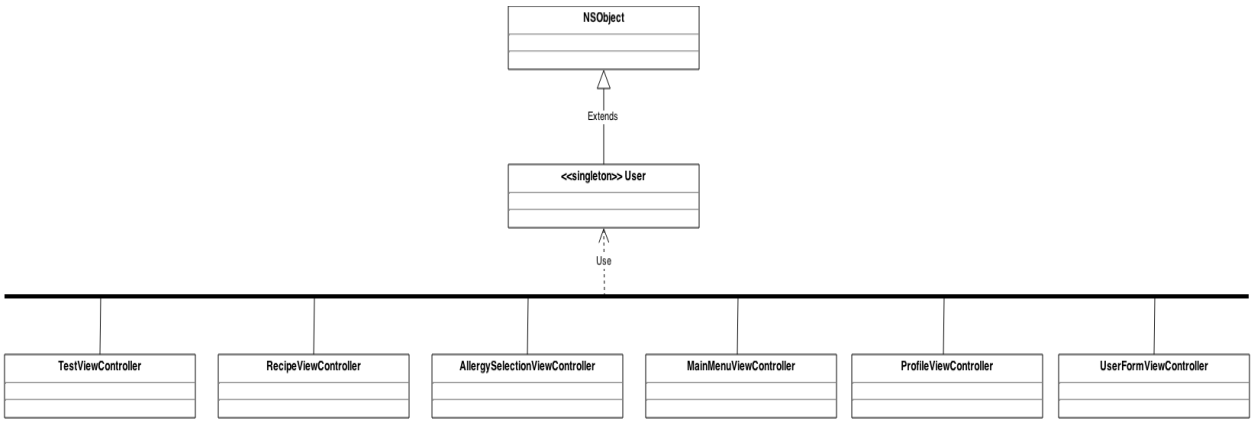
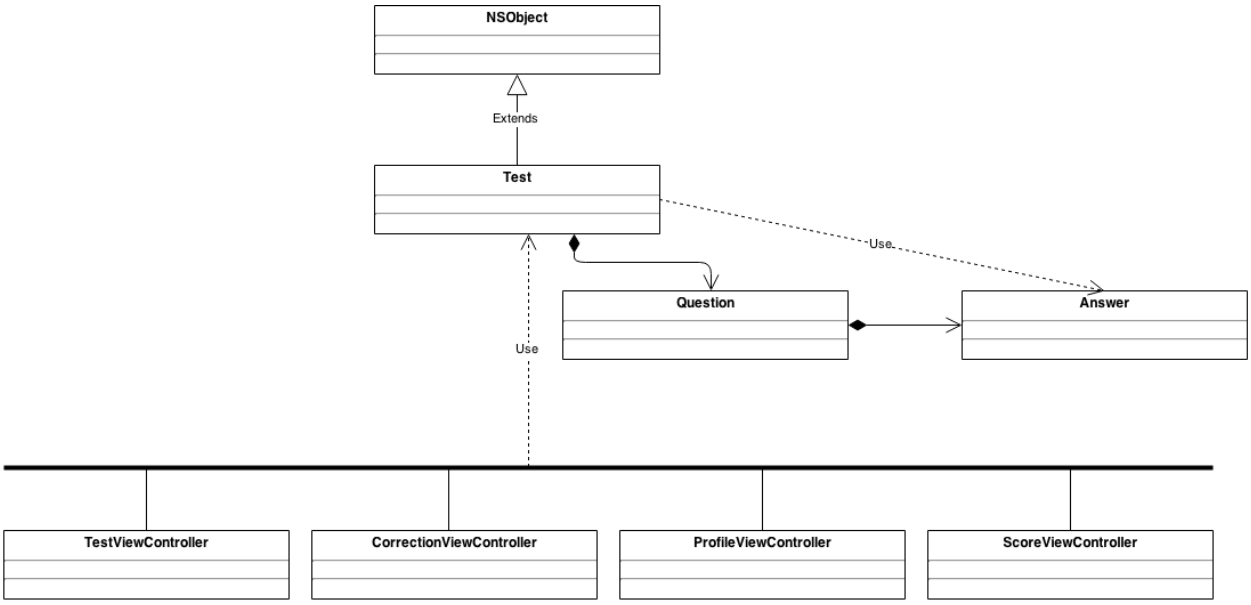
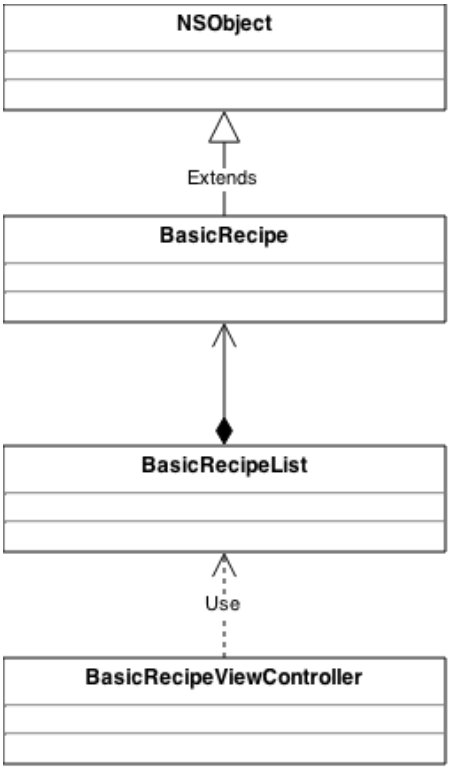


Imagen con las dependencias de User



Test y sus dependencias



BasicRecipe y sus dependencias.

4.4 PROGRAMACIÓN

A continuación, se procede a describir la secuencia que se ha seguido a la hora de realizar la programación de la aplicación.

Dada la importancia de que tiene la aplicación de administración para poder alimentar de contenidos a la aplicación móvil, además de afectar directamente a la lógica de algunas funciones que se van a implementar en el código del software para el dispositivo, la construcción del código en si comenzó en la aplicación *back-end*.

CREANDO LA LÓGICA BÁSICA

Dada la flexibilidad que ofrece Ruby on Rails, así como su principio "convención sobre configuración", construir el CRUD básico de las categorías se basó en ejecutar la sentencia desde la consola de comandos:

```
rails generate scaffold <nombre> <atributos>
```

De esta manera, Rails genera automáticamente las vistas, el modelo y controlador correspondiente al nombre que le hemos asignado.

La ejecución de este comando, realiza además las siguientes modificaciones:

- Declara como recurso la categoría que se acaba de crear. Esto crea las urls /nombredecategoria, /nombredecategoria/new, nombredecategoria/id y nombredecategoria/id/edit. Estos recursos corresponden a los métodos índice, crear, ver y editar. Además, crea un método destroy para la eliminación de registros de la base de datos.
- Crea una vista básica de cada una de estas funciones, proporcionando la funcionalidad básica.
- Genera automáticamente ficheros jbuilder para elaborar las respuestas JSON.

Exceptuando el caso de las recetas y los ingredientes, cuya lógica es algo diferente; para las categorías, tipos de plato y alergias su funcionamiento se corresponde íntegramente con el del *scaffolding*.

En cuanto a las recetas, es importante destacar el proceso de asociación de los ingredientes a la receta.

Cuando se añade una receta, se llama al método "New" de Recetas. En este caso, se ha modificado la lógica para que cada vez que se llama a esta función se cargue una lista actualizada de cada ingrediente con su ID correspondiente.

Esta lista se carga en el formulario en forma de lista de selección junto con otros dos campos que permiten añadir una cantidad y una unidad (tal y como se había especificado en el apartado anterior).

Cuando el usuario añade un nuevo ingrediente en la receta, se ejecuta un Javascript que incluye como variables la unidad, la cantidad y el ID que se obtiene de la lista de selección.

Estas variables son recogidas en la función “Create”, con el objetivo de crear una nueva entrada de Ingredient_Recipe, la tabla intermedia de la interrelación entre Recetas e Ingredientes.

```
ingredients=params["ingredients"]
if ingredients
for i in 0 .. ingredients.length - 1
ingid = ingredients[i.to_s()]['id']
qty = ingredients[i.to_s()]['quantity']
unity = ingredients[i.to_s()]['unity']
@recipe.ingredient_recipes.create(ingredient_id: ingid, quantity: qty, unity: unity)
end
end
```

Un caso similar ocurre con las categorías y los tipos de plato con respecto a la receta. Como se especificaba en el apartado anterior, estas relaciones incluyen un campo para la posición. En este caso, el valor del campo de la posición viene determinado por el orden en el que se reciben las variables, como se puede apreciar en el código que se adjunta:

```
course_types = params[:course_type]
@recipe.ct_recipes.create(course_type_id:course_types['0'],recipe_id:@recipe.id,posit
ion:1)
@recipe.ct_recipes.create(course_type_id:course_types['1'],recipe_id:@recipe.id,posit
ion:2)
categories = params[:category]
@recipe.category_recipes.create(category_id:categories['0'],recipe_id:@recipe.id,posi
tion:1)
@recipe.category_recipes.create(category_id:categories['1'],recipe_id:@recipe.id,posi
tion:2)
```

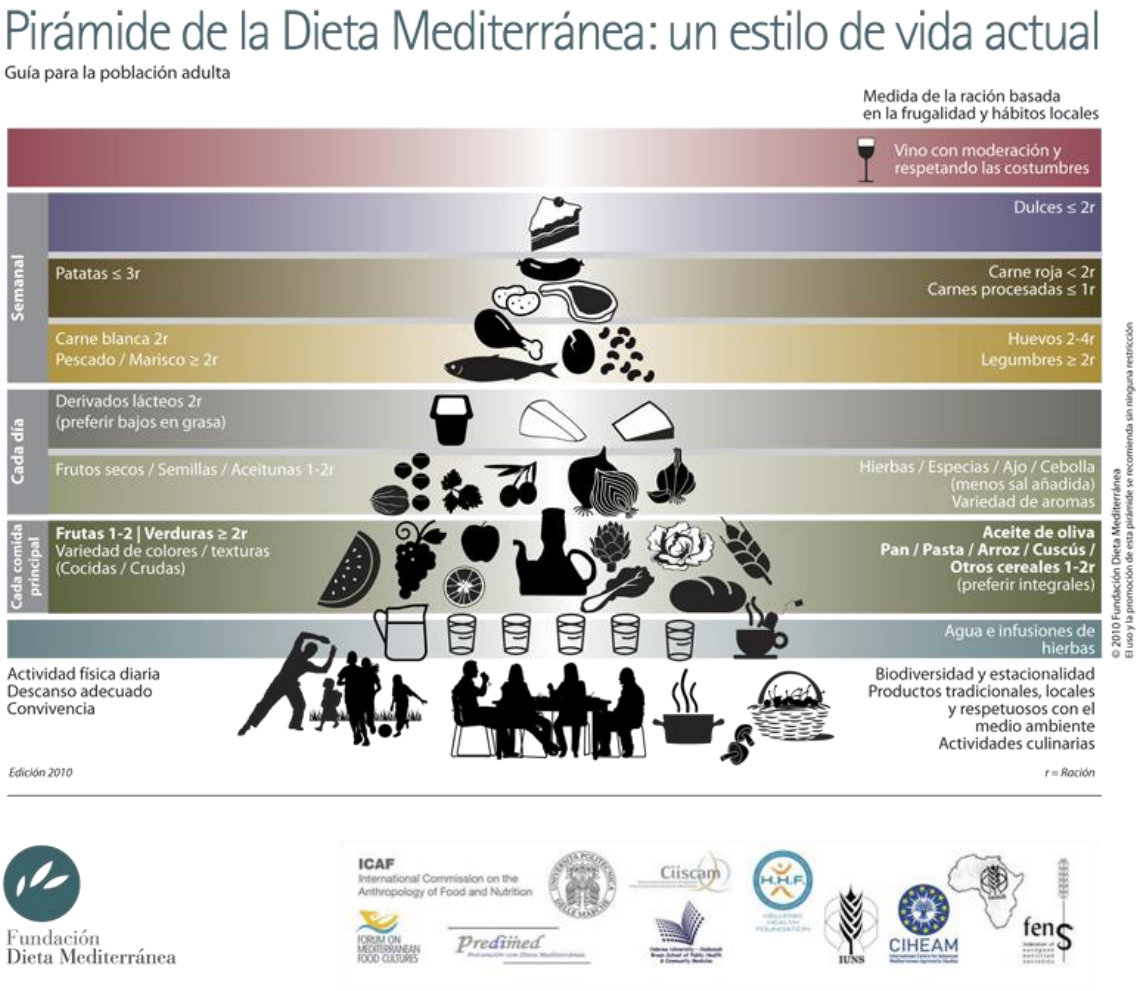
Dado que estos elementos crean una relación entre ambas entidades, cada vez que se quiera realizar una actualización de dicho registro, es necesario realizar una eliminación de dichas categorías para volver a asginarlas:

```
course_types = params[:course_type]
@recipe.ct_recipes.delete_all
@recipe.ct_recipes.create(course_type_id:course_types['0'],recipe_id:@recipe.id,posit
ion:1)
```

```
@recipe.ct_recipes.create(course_type_id:course_types['1'],recipe_id:@recipe.id,posit
ion:2)
categories = params[:category]
@recipe.category_recipes.delete_all
@recipe.category_recipes.create(category_id:categories['0'],recipe_id:@recipe.id,posi
tion:1)
@recipe.category_recipes.create(category_id:categories['1'],recipe_id:@recipe.id,posi
tion:2)
```

ALGORITMO DE GENERACIÓN DEL MENÚ

Uno de los puntos clave del desarrollo de la aplicación consistía en cómo se iba a realizar la generación del menú. En un principio, el menú se debe ajustar lo máximo posible a las cantidades recomendadas especificadas en la siguiente pirámide:



Podemos observar como la categorización del alimento es fundamental para que se genere un menú correcto y respetuoso con los parámetros que se detallan en esta pirámide.

Por tanto, era necesario tener una entidad que permitiese agregar cada una de esas categorías a las recetas en concreto. Esta entidad corresponde a Categories, como se ha especificado con anterioridad. Por tanto, se asignaba una categoría a la receta, que era la que se iba a tener en cuenta para la elaboración de la receta.

Al comenzar el desarrollo, el grupo formado por los estudiantes de Medicina expresaron que, con el objetivo de que el usuario tuviese una información más correcta, se debían especificar dos categorías: la categoría principal, que es la que se tendría en cuenta a la hora de realizar el desarrollo y la categoría secundaria, que estaría únicamente como información.

Sin embargo, esto causaba que la muestra de alimentos fuese demasiado pequeña, por lo que los platos se repetían con demasiada asiduidad. Por tanto, se ha considerado que la posición (elemento que se ha comentado en el bloque anterior) tendría validez únicamente a título informativo, por lo que ambas categorías son válidas a la hora de hacer el filtrado para obtener las recetas.

Otra de las decisiones que pasaron por un cambio fue la del tratamiento de las alergias a la hora de realizar el algoritmo. En un principio, las alergias se asignarían a las recetas, por lo que el filtrado de recetas válidas se haría por ingredientes. No obstante, ello ocasionaba falta de información a la hora de obtener información de las recetas, ya que resultaba muy complicado establecer cada una de las alergias dadas a una receta (había que tener en cuenta todos y cada uno de los ingredientes que la formaban).

La solución radicó en cambiar cómo se producía la asignación de las alergias, pasando a añadirse a los ingredientes. Por tanto, la búsqueda de las recetas adecuadas para una persona dada, conllevaba en identificar qué ingredientes no son los adecuados para su dieta para posteriormente obtener qué recetas debían ser eliminadas. Aun así, sigue existiendo una relación entre recetas y alergias para algunos casos muy puntuales en los que se quisiese establecer una exclusión solo a la receta.

Por tanto, para obtener el menú, hay que tener en cuenta lo siguiente:

- Obtener las recetas permitidas para el usuario, teniendo en cuenta sus alergias.
- Buscar las recetas que se van a añadir teniendo en cuenta su categoría.
- Configurar el menú para que sea variado y se respete al máximo la pirámide especificada anteriormente.

El método que se encarga de configurar todo el menú se denomina getMenu y se llama desde el controlador Recipe.

1. **Obtención de las recetas que puede consumir el usuario.**

```
allergyids = params[:aid]
    if allergyids
      validRecipes = get_recipes_without_allergies(allergyids)
    else
      validRecipes = Recipe.all
    end
```

Las alergias que padece el usuario que está realizando la petición llegará via POST en un parámetro llamado “aid” (más información en el apartado “Comunicación entre cliente y servidor” de este mismo punto).

En este caso, se comprueba si el usuario tiene alguna alergia diagnosticada. En caso negativo las recetas que consideramos válidas son todas aquellas que estén registradas en la base de datos. Por contrario, si existen alergias se llama al siguiente método:

```
def get_recipes_without_allergies(allergies)
    results = []
    allergy = Allergy.find(allergies)
    allergy.each do |al|
      al.ingredients.each do |ing|
        ing.recipes.each do |re|
          results.push(re)
        end
      end
    end
    @recipes = Recipe.all - results
    return @recipes
  end
```

Este método obtiene los ingredientes que están asociados a cada una de las alergias. Posteriormente, consulta qué recetas están realizadas con ese ingrediente y las almacena iterativamente en un vector.

La función devolverá el conjunto de recetas que no contienen esos ingredientes, pudiendo asegurar que el usuario no corre riesgo de consumir dichas recetas.

2. **Buscar las recetas que se van a añadir teniendo en cuenta su categoría.**

Para este caso en concreto, y con el objetivo de facilitar la lectura para el grupo colaborador, se definió como variable cada una de las categorías, igualándolas a su ID correspondiente, realizando la llamada con dicha variable a la siguiente función:

```
      def getRecipes(validRecipes,category_id,course_type_id)
          if (category_id)
            if (course_type_id)
              result = validRecipes & Category.find(category_id).recipes &
      CourseType.find(course_type_id).recipes
              return result.dup
            else
              result = result = validRecipes & Category.find(category_id).recipes
              return result.dup
            end
          else
            result = validRecipes & CourseType.find(course_type_id).recipes
            return result.dup
          end
  end
```

Esta función tiene como parámetros:

- validRecipes: el vector cargado anteriormente con el conjunto de recetas válidas.
- category_id: el id de la categoría de alimento por la que se va a realizar el filtrado. Puede ser nulo en casos como los desayunos o las meriendas, en cuyo caso se obvia la categoría dado que estas recetas están pensadas para aportar los nutrientes y vitaminas necesarios para esa hora.
- course_type_id: id correspondiente al tipo de ración.

La función se encarga de buscar las recetas correspondientes a la categoría y/o el tipo de plato que se pasa como parámetro y las concatena con las recetas válidas para obtener aquellas correctas. Para evitar duplicados, se llama al método .dup, que elimina las repeticiones de elementos.

3. Configurar el menú para que sea variado y respete al máximo la pirámide

En este caso, se plantearon varias alternativas. Inicialmente, se intentaba respetar al máximo las categorías, obteniendo las recetas necesarias y configurando el menú con aleatoriedad en aquellos casos más flexibles (fundamentalmente comidas y cenas).

Sin embargo, los resultados no eran lo suficientemente satisfactorios. En muchas ocasiones, se repetían recetas con el mismo ingrediente incluso en un mismo día, lo que rompía con uno de los principios de la dieta mediterránea. Después de probar con varias heurísticas, se contó con la ayuda de los expertos (los estudiantes de Medicina y su tutor) para que el resultado fuese lo más correcto posible.

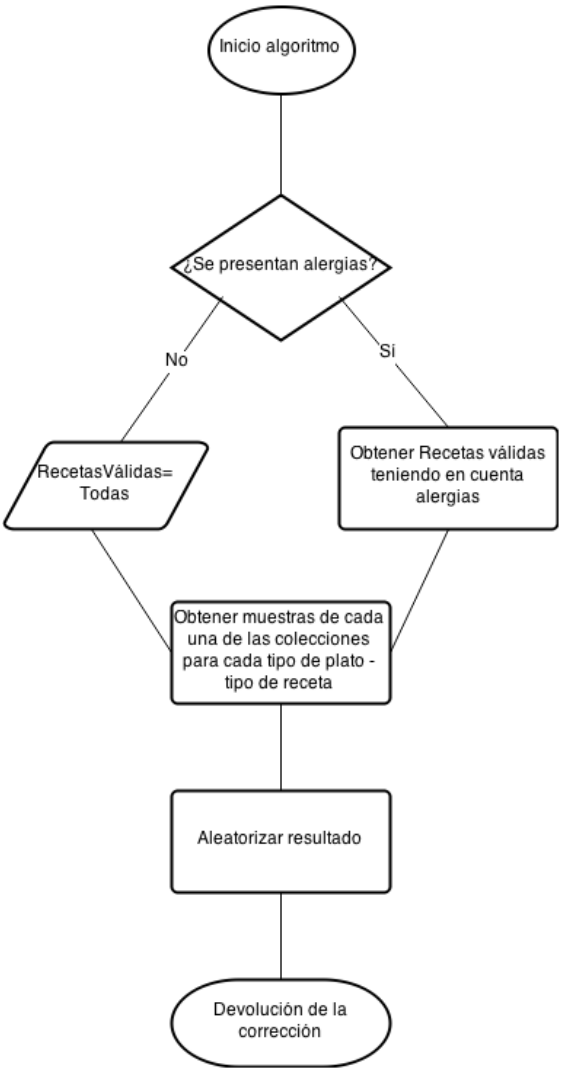
Para solucionar este problema, se optó por recoger una muestra fija para cada tipo de alimento-ración, de tal manera que se iban a seleccionar una mayor cantidad de las relaciones más frecuentes, aportando más variedad. Por ejemplo, las verduras son más comunes como primer plato de la comida, por lo que se recoge una muestra de 3 recetas. Asimismo, el pescado nunca se selecciona como primer plato del almuerzo, por lo que directamente no se toma muestra. La configuración final se puede observar en el siguiente código:

```
    @breakfast = getRecipes(validRecipes,nil,breakfast).sample(7)
    #mediaMañana
    @beforeLunch = getRecipes(validRecipes,nil,beforeLunch).sample(7)
    #Almuerzo - Primer plato
    vegetablesLunchFirstCourse =
getRecipes(validRecipes,vegetables,lunchFirstCourse).sample(3)
    pastaLunchFirstCourse = getRecipes(validRecipes,pasta,lunchFirstCourse).sample(2)
    legumesFirstCourse = getRecipes(validRecipes,legumes,lunchFirstCourse).sample(2)
    lunchFirstCourse = vegetablesLunchFirstCourse + pastaLunchFirstCourse +
legumesFirstCourse
    @lunchFirstCourse = lunchFirstCourse.sample(7)
    #Almuerzo - Segundo plato
    fishLunchSecondCourse =
getRecipes(validRecipes,fishAndShellfish,lunchSecondCourse).sample(3)
    leanMeatSecondCourse =
getRecipes(validRecipes,leanMeat,lunchSecondCourse).sample(2)
```

```
    legumesSecondCourse =
getRecipes(validRecipes,legumes,lunchSecondCourse).sample(2)
    lunchSecondCourse = fishLunchSecondCourse + leanMeatSecondCourse +
legumesSecondCourse
    @lunchSecondCourse = lunchSecondCourse.sample(7)
    #Postre
    desserts = getRecipes(validRecipes,nil,dessert).sample(1)
    fruit = getRecipes(validRecipes,fruit,dessert).sample(6)
    lunchDesserts = desserts + fruit
    @dessert = lunchDesserts.sample(7)
    #Merienda
    @snack = getRecipes(validRecipes,nil,snack).sample(7)
    #Cena
    @vegetablesDinner = getRecipes(validRecipes,vegetables,dinner).sample(2)
    pastaDinner = getRecipes(validRecipes,pasta,dinner).sample(2)
    fishDinner = getRecipes(validRecipes,fishAndShellfish,dinner).sample(1)
    leanMeatDinner = getRecipes(validRecipes,leanMeat,dinner).sample(1)
    legumesDinner = getRecipes(validRecipes,legumes,dinner).sample(1)
    dinner = @vegetablesDinner + pastaDinner + fishDinner + leanMeatDinner +
legumesDinner
    @dinner = dinner.sample(7)
```

Cabe destacar el uso del método sample() en este algoritmo. Este método toma el número *n* especificado como parámetro de muestras dentro de una colección aleatoriamente. Por tanto, en este código tiene dos usos:

- En el caso de las colecciones de tipos de plato, selecciona la cantidad que se ha considerado oportuna mediante la aplicación de la heurística. Al tomar una muestra aleatoria, se asegura una cierta variedad.
- Cuando se obtiene la colección para cada tipo de plato, se llama a este método con el número total de elementos de la colección. Esto causa una reordenación aleatoria de los elementos de la colección, con el objetivo de evitar lo máximo posible la repetición de patrones de categorías.



MÉTODO PARA OBTENER RECETAS SIMILARES

En el caso de que el usuario no quisiese seguir una receta, era recomendable obtener las recetas recomendadas dada una en concreto.

Para ello se ha creado un método en la clase Receta que permite obtener las recetas similares a otra en base a sus ingredientes, estableciendo el número como parámetro.

```
def similar(n=4)
    ingredients = self.ingredients
    recipes = ''
    ingredients.each do |ing|
      recipes = ing.recipes
    end
    similar = []
      recipes.each do |r|
        size = (r.ingredients & ingredients).size
        if (size >= n && r.id != self.id)
```

```
          similar << r
        end
      end
      return similar
  end
```

El método realiza:

1. Selecciona los ingredientes propios.
2. Obtiene las recetas de cada uno de esos ingredientes.
3. Añade a una colección aquellas recetas cuya coincidencia de ingredientes sea mayor al parámetro *n* dado.

Este método del modelo es llamado desde el controlador de Recetas, que también tiene en cuenta las alergias asociadas al usuario con objeto de ajustar las recomendaciones lo máximo posible.

```
def getsimilarrecipes
    recipeid = params[:recipeid]
    allergyids = params[:aid]
    if allergyids
      recipes = get_recipes_without_allergies(allergyids)
    else
      recipes = Recipe.all
    end
    @similarrecipes = recipes && Recipe.find(recipeid).similar()
  end
```

UTILIZACIÓN DE CLOUDINARY

Como está descrito en el apartado de herramientas utilizadas, se ha empleado Cloudinary para realizar la manipulación de imágenes. En este caso, tras la instalación de la gema correspondiente de Rails, hubo que especificar qué campo se iba a utilizar para la carga de las imágenes:

```
mount_uploader :image, ImageUploader
```

A su vez, fue necesaria realizar la configuración del servidor de Rails para que se comunique con la API de Cloudinary. Esta configuración está recogida en un fichero llamado *cloudinary.yml* dentro del directorio /config:

```
development:
  cloud_name: dbzd6ba7w
  api_key: '447364498759748'
  api_secret: 5OGPpORCge3UNPgWO1row8N7NbA
  enhance_image_tag: true
  static_image_support: false
production:
  cloud_name: dbzd6ba7w
  api_key: '447364498759748'
  api_secret: 5OGPpORCge3UNPgWO1row8N7NbA
  enhance_image_tag: true
  static_image_support: true
test:
```

```
cloud_name: dbzd6ba7w
api_key: '447364498759748'
api_secret: 5OGPpORCge3UNPgWO1row8N7NbA
enhance_image_tag: true
static_image_support: false
```

La declaración de las diferentes configuraciones de las imágenes se realizan desde el directorio /app/uploaders.

Una vez vistos los métodos más destacables de la aplicación del lado del servidor, se desarrolló la lógica de la aplicación móvil.

MÉTODOS BÁSICOS

Como se detalló en el apartado de diseño, para buscar una mayor facilidad de programación, especialmente en cuanto a interfaces se refiere, se recomienda la creación de un *ViewController* para cada pantalla. Por tanto, hay una serie de métodos que se implementan en numerosas ocasiones y cuyo funcionamiento es similar, cambiando en cada caso la lógica aplicable.

viewDidLoad: Este método se ejecuta cada vez que se ha cargado completamente la vista. En esta función se realiza la inicialización de los elementos de la interfaz, especialmente los textos e imágenes.

Métodos relacionados con la interfaz: Xcode y Objective-C utilizan las acciones (IBActions) para realizar la comunicación entre la interfaz y los controladores. Para ello se crean métodos que toman como parámetro el id del elemento de interfaz que ha enviado el mensaje. Estas acciones se asignan desde el *Interface Builder* de Xcode para que reaccione al evento deseado. Por ejemplo, para crear una acción que muestre un saludo por consola al pulsar sobre un botón:

```
-(IBAction)saludar:sender(id){
      NSLog(“¡Hola mundo!”);
}
```

Y posteriormente, de forma gráfica, asignar la acción al evento correspondiente. En este caso, al evento TouchDown.

Cabe destacar que para que se puedan ejecutar las acciones, los elementos de interfaz han tenido que ser declarados previamente y haber sido declarados como Referencing Outlet. Para ello hay que declarar el fichero cabecera.

```
-(IBOutlet) UILabel* etiqueta;
```

Cambiando la clase UILabel por la correspondiente dependiendo del tipo de elemento de interfaz que sea.

Por último, queda registrar el elemento declarado como *Referencing Outlet* del objeto de interfaz.

Cambio de pantallas: Al seguir esta convención, el cambio de pantalla se realiza invocando al ViewController correspondiente.

La aplicación tiene un controlador, el *NavigationController*, que se encarga de guardar en su pila las llamadas sucesivas a los controladores. Esto nos permite controlar además el comportamiento cuando se realizan acciones que implican una gestión de los controladores de vista. A continuación vemos un ejemplo de cambio de pantalla programáticamente.

```
EjemploViewController = [[EjemploViewController alloc]
initWithNibName:@”EjemploViewController” bundle:nil];

[self.navigationController pushViewController:ejemploViewController animated:YES];
```

Se observa que la llamada a la pantalla se realiza de la siguiente manera:

- Crear una instancia del *ViewController* correspondiente. Esta instancia requiere que se importe la clase que se va a llamar, que es la causante de las relaciones entre los controladores que están descritas en la sección “Diseño” de este documento.
- Opcionalmente, puebla aquellas propiedades que estén definidas en el fichero de cabecera correspondiente.
- Manda un mensaje del tipo “pushViewController” al navigationController, de tal manera que coloca como cabeza de la pila dicho controlador. Llamar al método “popViewController” tendría como consecuencia volver al ViewController actual.

MÉTODOS DE TABLAS

Las tablas son uno de los elementos básicos de interfaz de las aplicaciones iOS. No obstante, su implementación es algo diferente con respecto al resto de elementos de interfaz. A continuación se describen sus métodos básicos y su funcionamiento. Para ello utilizaremos como referencia la clase que muestra el menú de una semana personalizado para el usuario.

Los métodos que requieren implementación son:

- ***NumberOfSectionsInTableView***: Este método deuvelve un entero con la cantidad de secciones que va a tener la tabla. Por ejemplo, en el caso de la aplicación que nos ocupa, la pantalla del menú mostrará una tabla con 7 secciones, una para cada día de la semana:

```
-(NSInteger) numberOfSectionsInTableView(UITableView *)tableView{
   return 7;
}
```

  En caso de que se especifique un entero mayor que 1, se debe implementar el método *titleForHeaderInSection*, que devuelve una cadena de caracteres con el título que se desea mostrar.

- ***NumberOfRowsInSection:*** Esta función devuelve el número de celdas que va a tener cada sección de las especificadas anteriormente. En este caso, cada día va a tener 7 platos diferentes para crear el menú:

```
-(NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
return 7;
}
```

- ***heightForRowAtIndexPath*:** Se especifica la altura de cada una de las celdas que se van a pintar. No obstante, este valor puede ser sobrescrito editando el fichero de interfaz .xib:

```
-(NSInteger) tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath{
   return 60.0f;
}
```

- ***WillSelectRowAtIndexPath***: Esta implementación especifica el comportamiento que van a tener las celdas con respecto a la selección. En el ejemplo que se muestra, se desactiva la selección de las celdas:

```
-(NSInteger) tableView:(UITableView *)tableView
willSelectRowAtIndexPath:(NSIndexPath *)indexPath{
      return NO
}
```

- ***CellForRowAtIndexPath*:** Este método es el encargado de definir qué contenido va a incluirse en cada una de las celdas. Para ello, requiere la creación de una clase que represente a la celda (que herede de ViewCell). Las ocasiones en las que ocurre este hecho están recogidas en la sección "Diseño" de esta memoria. El procedimiento es el siguiente:
    1. Se crea una instancia de la celda.
    2. Se llama a los métodos de la celda, que cambian el contenido. En la mayor parte de los casos de esta aplicación, el método se encarga de colocar el texto a las etiquetas.
    3. Devuelve la celda.

```
-(NSInteger) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{
      MenuTableViewCell cell = [[MenuTableViewCell alloc] init];
      [cell setTítulo:@"Ejemplo"];
}
```

  Cabe destacar que para este caso, es muy importante invocar a *dequeueReusableCellWithIdentifier*. Este método indica qué clase reutiliza las celdas, de tal manera que el número de celdas creadas sea siempre las que se muestran por pantalla y su contenido sea lo que cambie. Esto supone un ahorro de memoria importante, especialmente teniendo en cuenta que se trata de un dispositivo móvil.

- ***DidSelectRowAtIndexPath*:** Este método describe qué comportamiento se sigue cuando el usuario selecciona una de las celdas de la tabla. Este método es el que más varía su implementación para adaptarse a cada caso. Por ejemplo, si deseamos que muestre por consola el número de la celda dentro de la sección que ha sido pulsado, el código sería:

```
-(NSInteger) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{
      NSLog(indexPath.row);
}
```

LA ENCUESTA

Una de las partes con mayor procesamiento dentro de la aplicación móvil corresponde a la lógica del test. Como se indicó en el apartado de diseño, en la encuesta se emplean tres clases:

- **Respuesta(Answer)**: Esta clase está compuesta por una string que representa el texto de la respuesta y una variable booleana, indicando si dicha respuesta es correcta o no.
  Se proporciona además el método *esCorrecta* que devuelve el valor de dicha variable booleana.
- **Pregunta(Question):** Esta clase contiene un texto para indicar la pregunta, dos cadenas para indicar el texto de la corrección (uno cuando la pregunta se ha respondido correctamente y la otra para cuando dicha respuesta es errónea), y una colección de objetos Respuesta.
- **Encuesta(Test):** Por su parte, este objeto Test se encarga de inicializar cada una de las preguntas que van a conformar el cuestionario. Por tanto, la clase Test consta de tres vectores: el primero almacena cada uno de los objetos Pregunta que conforman el test, el segundo array almacenará en cada uno de sus valores un entero que representan el valor que el usuario ha marcado, y una tercera colección para almacenar si cada una de las respuestas han sido contestadas correctamente o no (resultado que se obtiene consultando si esCorrecta la respuesta seleccionada en cada una de las preguntas). Por último, también se almacena la puntuación del test actual, permitiendo consultar fácilmente todos los datos del test con solo pasar este objeto creado.

  Este funcionamiento permite que no sea necesario comprobar de nuevo la corrección de cada una de las preguntas del test cuando se accede a la pantalla de corrección, ahorrando tiempo y la repetición de ejecución de métodos.

  Por tanto, el proceso de contestación a la encuesta se divide en los siguientes pasos:

  1. El usuario acepta que va a iniciar el test.
  2. Se crea un objeto de tipo Test. Actualmente, esa información está codificada directamente en el inicializador de dicha clase.
  3. El TestViewController se encarga de mostrar la información de cada una de las preguntas. Para ello, toma como variable el número de pregunta y accede en cada momento a la posición de las preguntas de Test correspondiente.
  4. Cuando el usuario pulsa en “Siguiente Pregunta”, se almacena en la posición del array correspondiente de OpcionesMarcadas un 0 o un 1 dependiendo del índice de la tabla que haya marcado (se registra en el *didSelectRowAtIndexPath*). A la vez, en la variable

registroDeRespuestas se le añade al vector un booleano indicando si la respuesta final es la correcta o no, consultando al método esCorrecta de la respuesta.

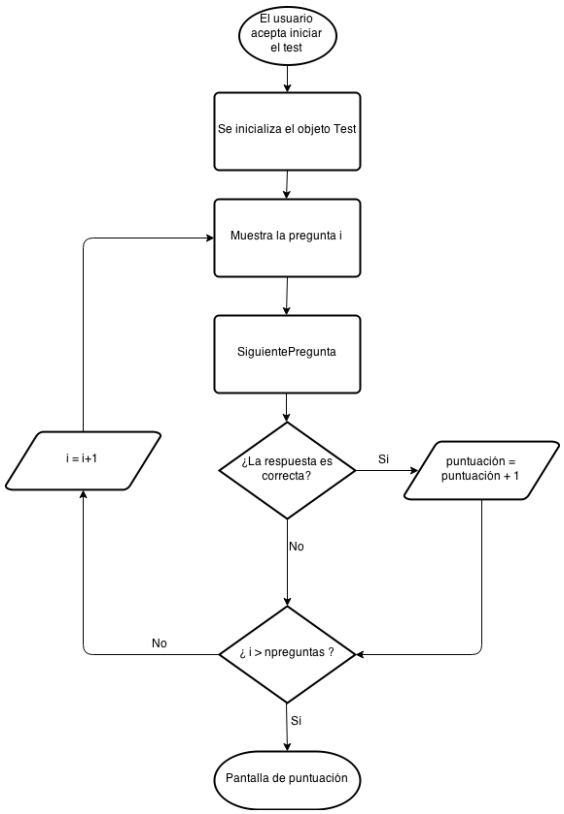
5. Si la respuesta es correcta, se suma un punto al resultado de la encuesta y se avanza a la siguiente pregunta.
6. Al finalizar el test, se realiza el envío de las opciones y las correcciones al servidor, con el objetivo de mantener un registro con respecto al usuario.
7. El usuario puede acceder entonces a la corrección del test. En ese caso, se le pasará como parámetro al controlador de esa vista correspondiente la puntuación del test junto con el registro de opciones marcadas y el de correcciones.

EL MENÚ

En esta aplicación se recibirán los datos del menú personalizado siguiendo el algoritmo descrito anteriormente en este desarrollo. Con el objetivo de que el algoritmo de generación del menú no tuviese un tiempo de ejecución excesivamente alto, se decidió que la función del servidor devolviese el menú correspondiente a una semana. Esto además concuerda con el criterio de mostrado seguido en la aplicación móvil, ya que el usuario selecciona primero la semana de la que quiere obtener el menú y posteriormente se le muestra el resultado.

En el código anterior vemos que la llamada al servidor de produce cuando el usuario selecciona la semana de la que desea obtener la información. No obstante, se comprueba primero la existencia de un fichero en el dispositivo que contenga el menú para esa semana, con el objetivo de cargar esa información en vez de generar una nueva cuando el usuario entra en sucesivas ocasiones. De esta manera, la creación del menú se realiza únicamente cuando se ha solicitado y no de forma completa en cada ocasión.

Por otro lado, en el menú existe la opción de seleccionar recetas correspondientes a los platos del almuerzo y la cena. Para obtener esa información, se realiza una llamada a *RecipeViewController* enviando como parámetro la URL que permitirá obtener una respuesta en formato JSON de la información de la receta. Vemos el flujo a continuación:



USUARIO

La última parte en desarrollarse correspondió a la creación del usuario, tanto en el acceso del servidor como en la aplicación móvil. Antes de analizar este desarrollo hay que tener las siguientes consideraciones:

- En un principio, el acceso a la plataforma del servidor se iba a producir también con una gestión de usuarios, permitiendo acceder al usuario administrador al apartado de gestión de contenidos y a la corrección de datos por parte del resto de los usuarios. Sin embargo, por decisión del grupo, se eliminó la idea de que los usuarios accediesen a la interfaz web para mostrar información. Esto hace que, si bien hay una gestión de sesiones en la parte del servidor, no se utilicen usuarios para acceder al *backend*.

- Para la versión del prototipo, la identificación del usuario entre el servidor y la aplicación móvil se realiza mediante el id del usuario, utilizando una API REST. En el caso del paso a producción final del software, sería necesario mejorar esta identificación, como se explica en el apartado 5 de este documento.

El *login* cuenta con los siguientes pasos:

1. Cuando el usuario pulsa la opción para iniciar sesión, se realiza una llamada al servidor con el usuario y contraseña.

```
-(IBAction)login:(id)sender{
   NSURL *url = [NSURL URLWithString:@"http://localhost:3000/login"];
```

```
    NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
    NSMutableString *senddata = [[NSMutableString alloc]
initWithString:@"email="];
    [senddata appendString:emailTextField.text];
    [senddata appendString:@"&password="];
    [senddata appendString:passwordTextField.text];
    NSLog(passwordTextField.text);
    [senddata appendString:@""];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"application/x-www-form-urlencoded; charset=utf-8"
forHTTPHeaderField:@"Content-Type"];
    [request addValue:@"" forHTTPHeaderField:@"X-CSRFToken"];
    [request setHTTPBody:[senddata dataUsingEncoding:NSUTF8StringEncoding]];
    NSLog(@"Prueba");
    [NSURLConnection connectionWithRequest:request delegate:self];
    NSURLResponse *response = [[NSURLResponse alloc] init];

    [NSURLConnection sendSynchronousRequest:request returningResponse:&response
error:nil];
}
```

2. En el servidor, se busca por la existencia del nombre de usuario para posteriormente comprobar los resúmenes de la contraseña. En caso de que estos coincidan, el sistema devuelve como respuesta el ID del usuario. Si se produce algún error en la autenticación, se devolverá un entero con valor -1.

```
def login
      email = params[:email]
      not_encrypted_password = params[:password]
      Session.create(email, not_encrypted_password)
end
```

En el controlador de sesión, se recoge:

```
def create
      if user = User.authenticate(params[:email], params[:password])
            session[:user_id]=user.id
            @user = User.find(session[:user_id])
            aids = []
            @user.allergies.each do |allergy|
                  aids.push(allergy.id)
            end
            aidsstring = aids.join(",")
            render :json => {:id => @user.id, :name => @user.name, :surname =>
@user.surname, :email => @user.email, :aidsstring => aidsstring}
      else
            render :json => {:id => -1}
      end
end
```

3. La aplicación recoge ese valor. En caso de que la autenticación haya sido exitosa, crea la instancia del singleton User, cargando con la información correspondiente.

```
-(void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data{
   [_responseData appendData:data];
   NSError *error = nil;
   NSDictionary* responseDictionary = [NSJSONSerialization
JSONObjectWithData:_responseData options:0 error:&error];
   int userid = [[responseDictionary objectForKey:@"id"] intValue];
   if (userid > 0){
       User *loggedInUser = [User sharedUser];
       loggedInUser.nombre = [responseDictionary objectForKey:@"name"];
       loggedInUser.apellidos = [responseDictionary objectForKey:@"surname"];
       loggedInUser.email = [responseDictionary objectForKey:@"email"];
       loggedInUser.userid = userid;
       NSMutableArray* listastrings = [[[responseDictionary
objectForKey:@"aidsstring"] componentsSeparatedByString:@","] mutableCopy];
       NSMutableArray* listaAlergias = [[NSMutableArray alloc] init];
       for (int i=0; i<listastrings.count; i++) {
           NSNumber* number = [NSNumber numberWithInt:[[listastrings
objectAtIndex:i] intValue]];
           [listaAlergias insertObject:number atIndex:i];
       }
       loggedInUser.listaAlergias = listaAlergias;
       InicioViewController* inicioViewController = [[InicioViewController alloc]
initWithNibName:@"InicioViewController" bundle:nil];
       [self.navigationController pushViewController:inicioViewController
animated:YES];
   }else{
       UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Usuario
incorrecto" message:@"Ese usuario no está en nuestra base de datos. Revise su
información o regístrese." delegate:nil cancelButtonTitle:@"Cancelar"
otherButtonTitles:@"OK",nil];
       [alert show];
   }
}
```

Para el registro:

1. Se rellenan los datos y se envían al servidor.

```
NSURL *url = [NSURL URLWithString:@"http://localhost:3000/createuser"];
   NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
   NSMutableString *senddata = [[NSMutableString alloc]
initWithString:@"email="];
   [senddata appendString:eMailTextField.text];
   [senddata appendString:@"&password="];
   [senddata appendString:passwordField.text];
   [senddata appendString:@"&name="];
   [senddata appendString:nombreTextField.text];
   [senddata appendString:@"&surname="];
   [senddata appendString:apellidosTextField.text];
   [senddata appendString:@"&location=0"];
   [request setHTTPMethod:@"POST"];
   [request setValue:@"application/x-www-form-urlencoded; charset=utf-8"
forHTTPHeaderField:@"Content-Type"];
   [request setHTTPBody:[senddata dataUsingEncoding:NSUTF8StringEncoding]];
   [NSURLConnection connectionWithRequest:request delegate:self];
```

2. Se comprueba por la no existencia del nombre de usuario y si la contraseña cumple con los criterios definidos.

```
def create
            email = params[:email]
            not_encrypted_password = params[:password]
            location = params[:location]
            name = params[:name]
            surname = params[:surname]
            @user =
User.new(email:email,password:not_encrypted_password,location:location,name:na
me,surname:surname)
            if @user.save
                  cookies[:user_id] = @user.id
                  render :json => {:id => @user.id, :name =>
@user.name, :surname => @user.surname, :email => @user.email}
            else
                  render :json => {:id => -1}
            end
end
```

Los criterios de validación se pueden consultar en el modelo User y son:

```
validates :email, uniqueness: {case_sensitive: false}
validates :password, :length => {:within => 4..30}
```

3. En caso afirmativo, se cifra la contraseña y se almacenan los datos en la base de datos. El sistema devuelve la ID del usuario. Como ocurría con el caso del login, se devuelve -1 en caso de que se haya producido un error, no creando los datos.
4. Se crea el usuario con los datos correspondientes, pero con la lista de alergias vacía. Se continúa al procedimiento de alta de las alergias.

```
-(void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data{
   [_responseData appendData:data];
   NSError *error = nil;
   NSDictionary* responseDictionary = [NSJSONSerialization
JSONObjectWithData:_responseData options:0 error:&error];
   int userid = [[responseDictionary objectForKey:@"id"] intValue];
   if (userid > 0){
       User *signedUpUser = [User sharedUser];
       signedUpUser.email = [responseDictionary objectForKey:@"email"];
       signedUpUser.nombre = [responseDictionary objectForKey:@"name"];
       signedUpUser.apellidos = [responseDictionary objectForKey:@"surname"];
       signedUpUser.userid = userid;
       /*User *signedUpUser = [[User alloc] initWithEmail:[responseDictionary
objectForKey:@"email"] andNombre:[responseDictionary objectForKey:@"name"]
andApellidos:[responseDictionary objectForKey:@"surname"] andUserid:userid];*/
       AllergySelectionViewController *allergySelectionViewController =
[[AllergySelectionViewController
alloc]initWithNibName:@"AllergySelectionViewController" bundle:nil];
       [self.navigationController
pushViewController:allergySelectionViewController animated:YES];
   }else{
```

```
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
message:@"Ha ocurrido un error al almacenar el usuario, por favor, revise su
información y vuelva a intentarlo" delegate:nil cancelButtonTitle:@"Cancelar"
otherButtonTitles:@"OK",nil];
        [alert show];
    }
}
```

5. Cuando el usuario ha seleccionado los elementos de la lista de las alergias, se realiza la llamada al método que asigna las alergias al usuario. En caso de que se produzca un error estas no son añadidas a la instancia de User hasta que se produzca un caso de éxito.

```
-(IBAction)registrarAlergias:(id)sender{
    NSString *stringAlergias = [listaAlergias componentsJoinedByString:@","];
    NSURL *url = [NSURL URLWithString:@"http://localhost:3000/registerallergies"];
    NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url];
    NSMutableString *senddata = [[NSMutableString alloc]
initWithString:@"allergies="];
    [senddata appendString:stringAlergias];
    [senddata appendString:@"&userid="];
    [senddata appendString:[NSString stringWithFormat:@"%d",signedUpUser.userid]];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"application/x-www-form-urlencoded; charset=utf-8"
forHTTPHeaderField:@"Content-Type"];
    [request setHTTPBody:[senddata dataUsingEncoding:NSUTF8StringEncoding]];
    [NSURLConnection connectionWithRequest:request delegate:self];
}
```

Una vez el usuario haya accedido al menú principal, sus datos permanecen en tiempo de ejecución en la instancia de User. Los controladores que acceden a la información de User se pueden comprobar consultando el diagrama de dependencias de User de la sección “Diseño”.

CONEXIONES APP DISPOSITIVO-*BACKEND*

En el código de la aplicación móvil, se ha utilizado en numerosas ocasiones los métodos que nos permiten realizar la conexión con el servidor que aloja la administración del contenido. A continuación se resume el funcionamiento básico de dichos métodos y se adjunta una tabla en donde se recogen todas las llamadas que se realizan entre las aplicaciones.

Métodos GET

Las llamadas con GET de la aplicación se realiza cuando se desean obtener datos que no van a modificar el estado de los registros del servidor. En este caso, basta con realizar la siguiente llamada:

```
data = [NSData dataWithContentsOfURL:url];
```

Este método carga, dada una URL, un objeto del tipo NSData con esa información. Con el objetivo de poder acceder fácilmente a esa información con los pares clave-valor, se realiza una llamada al siguiente método, que serializa los datos a JSON:

```
datosJSON = [NSJSONSerialization JSONObjectWithData:data options:kNilOptions
error:nil];
```

Métodos POST

Los métodos POST son utilizados cada vez que queremos realizar cambios en el estado del servidor, especialmente cuando se desea introducir información nueva en la base de datos. En este caso, el funcionamiento varía sensiblemente con respecto a GET:

El controlador que vaya a encargarse de realizar la conexión debe heredar de *NSURLConnectionDelegate*. Esta herencia causa que sea necesario implementar los siguientes métodos:

- ***didRecieveResponse:*** Método que se ejecuta cuando se ha recibido una respuesta del servidor. Normalmente se realiza la inicialización de los elementos que van a alojar los datos.
- ***didRecieveData:*** Este método se ejecuta una vez que se han recibido los datos del servidor. En este método se realiza la manipulación de datos, implementando la lógica adecuada en cada caso.
- ***willCacheResponse:*** empleado para los casos en los que se desee alojar en una cache la respuesta obtenida. En la aplicación objeto de este proyecto, no realizamos uso de la caché, por lo que se devuelve nulo.
- ***connectionDidFinishLoading:*** Cuando el proceso de establecimiento de la conexión ha finalizado, se envía un mensaje al que este método responde con la lógica correspondiente.
- ***didFailWithError:*** Función encargada de controlar los casos en los que la conexión haya resultado fallida.

Un ejemplo de esta implementación se encuentra en el registro/inicio de sesión del usuario descrito con anterioridad en esta memoria.

Se presenta a continuación una tabla que contiene información sobre las llamadas entre las aplicaciones con objetivo de servir como referencia para comprender qué datos son pedidos en cada caso:

Nombre función	Tipo de petición	Recibe como parámetro	Devuelve	Funcionalidad
getSimilarRecipes	GET	- ID de la receta - ID de las alergias	- Nombres de recetas - URLs de petición	Obtener las recetas similares dada una receta
createuser	POST	- Email - Contraseña - Nombre - Apellidos	- ID del usuario creado o -1	Registrar un usuario

registerallergies	POST	- ID del usuario - Alergias del usuario	- OK o KO	Asociar alergias a un usuario
registrartest	POST	- ID del usuario - Cadena de texto son sucesión de respuestas - Cadena de texto con correcciones	- OK o KO	Almacenar el test del usuario
gettest	GET	- ID del usuario	- Cadena de texto con sucesión de respuestas - Cadena de texto con sucesión de correcciones	Obtener el último test del usuario
login	POST	- Nombre de usuario - Contraseña	- ID del usuario o -1	Autenticar a un usuario
getMenu	GET	- Alergias del usuario	- Recetas categorizadas por tipo de plato con sus campos	Obtener el menú personalizado para el usuario
tableIndex	GET	- ID de la categoría	- Nombre de las recetas - Url de las recetas	Obtener el listado de recetas perteneciente a una categoría
recipe	GET	- ID de la receta	- Campos de la receta - Nombre de categorías - Nombre de tipos de ración	Obtener todos los datos de la receta

5 CONCLUSIONES Y TRABAJOS FUTUROS

Teniendo en cuenta los objetivos que planteamos en el inicio, y el desarrollo explicado en el punto 4, concluimos qué:

- La persona que acceda a este prototipo va a ser capaz de evaluar su grado de adecuación a la dieta mediterránea realizando un test cuya exactitud está científicamente probada en la documentación.
- El usuario va a disponer de una herramienta que le permitirá obtener un menú adaptado a sus necesidades, teniendo en cuenta las alergias que puede padecer y en parte sus gustos (ya que siempre dispone de recetas alternativas).
- Los administradores de la aplicación tienen a su disposición una base de datos que permite categorizar correctamente los alimentos, con el objetivo de obtener los resultados más fiables posibles.
- Se dispone de diferentes secciones que aportan un conocimiento contrastado sobre los beneficios de adherirse a la dieta mediterránea.

Teniendo en cuenta que hemos superado los objetivos de este prototipo (recogidos en el apartado 1 de este documento) así como los requisitos funcionales y no funcionales de la fase de análisis, y que la fase de pruebas ha sido satisfactoria ya que tanto los administradores como los usuarios de prueba han tenido una experiencia positiva con la aplicación, pronosticamos que la futura aplicación final logrará el impacto que se esperaba, especificado en el punto 3 de esta memoria.

No obstante, al presentar como resultado del proyecto un prototipo, existe un margen de mejora, especialmente en los puntos que comentamos a continuación:

- Tener una interfaz gráfica más atractiva tanto en la aplicación del dispositivo como en la del servidor.
- Mejora de las cargas de las pantallas de la aplicación del dispositivo móvil, comprobando el estado de la conexión e informando en todo momento al usuario de que se está produciendo la carga de los contenidos.
- Mejorar la sincronización del perfil del usuario: si bien en la actualidad, los perfiles de usuario se sincronizan con el servidor, hay elementos como los menús que no se quedan registrados en la base de datos, sino en el propio dispositivo. Esto causa, entre otros comportamientos, que un usuario que acceda con otro dispositivo móvil obtenga un menú personalizado distinto al que obtuvo con el primer dispositivo.
- Proporcionar estadísticas a los administradores sobre el test: actualmente las respuestas del test se almacenan en la base de datos del *back-end*. Sin embargo, con el objetivo de obtener unas estadísticas mejor sectorizadas, se necesitarían obtener datos adicionales, como por ejemplo la localización, elemento que no se cubría en este prototipo.
- Nuevas secciones: en la fase de captura de requisitos, surgió la idea de incluir juegos para facilitar la comprensión de alguna de las ideas que se querían reflejar en la aplicación. Por cuestiones de complejidad (el juego en sí podría ser otro TFG) se decidió aplazar para el producto final.

- Mejora de la sesión del usuario. A la hora de realizar el cambio final a producción, es trivial realizar algunos cambios en el manejo de usuarios relativo a la sesión, con objetivo de mejorar en niveles de seguridad y fiabilidad.
- Control del estado de la conexión: es vital en una aplicación de estas características, en donde gran parte de la información se obtiene mediante la consulta de información a servicios en Internet, llevar un control sobre el estado de la conexión, controlando los errores provenientes de los cortes de conexión, bastante comunes en entornos de redes 3G/4G. No obstante, dado que el proceso de desarrollo se ha realizado con simuladores, no ha sido posible realizar las pruebas en todos esos supuestos, por lo que sería necesaria una fase de pruebas adicional.
- Adaptar la aplicación para que se ajuste a la legislación vigente, como se recoge en la siguiente sección.

6 LEGISLACIÓN

Aunque se han tomado medidas de seguridad básicas a la hora de registrar la información del usuario, hay que tener en cuenta el marco legislativo que afecta al proyecto.

La legislación vigente que afecta a gran parte del uso de la aplicación se centra en la **LOPD**, especialmente en lo relativo a los datos personales.

Por tanto, el usuario debe tener derecho de acceso, modificación y cancelación de sus datos personales, preferiblemente desde el servidor desde donde se están registrando los datos. Adicionalmente, esto conllevaría que el usuario tuviese que aceptar los **Términos y condiciones** de uso de la aplicación a la hora de crear una nueva cuenta, lo que requiere una redacción por parte de expertos en legislación.

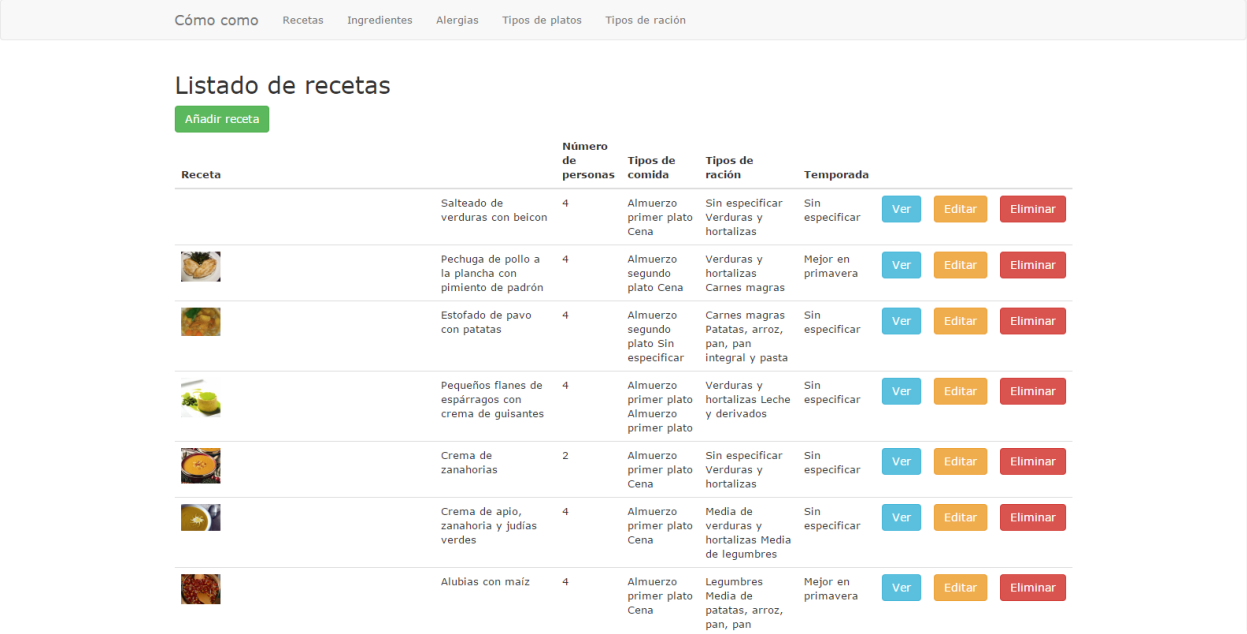
Además en este caso se almacenan datos de carácter médico (en este caso alergias) que están catalogados en dicha ley como de nivel **alto**, por lo que se ven afectados por los artículos del 101 al 104 de la legislación vigente. Entre otras medidas, se establece la necesidad de cifrar dichos datos, además de elaborar ficheros de registro que permitan comprobar qué personas han accedido a dicha información.

Teniendo en cuenta el contexto en el que se desarrolla este proyecto, cuyo fin es presentar un prototipo que muestre una idea del funcionamiento de la aplicación, se ha obviado el cumplimiento completo de la normativa de esta ley, dado que conllevaría también la creación de un documento de seguridad en donde se especifiquen qué medidas se van a llevar a cabo.

7 MANUAL DE USUARIO

El objetivo de este manual de usuario es el de facilitar la utilización de la aplicación tanto a los usuarios administradores como a los usuarios de la aplicación.

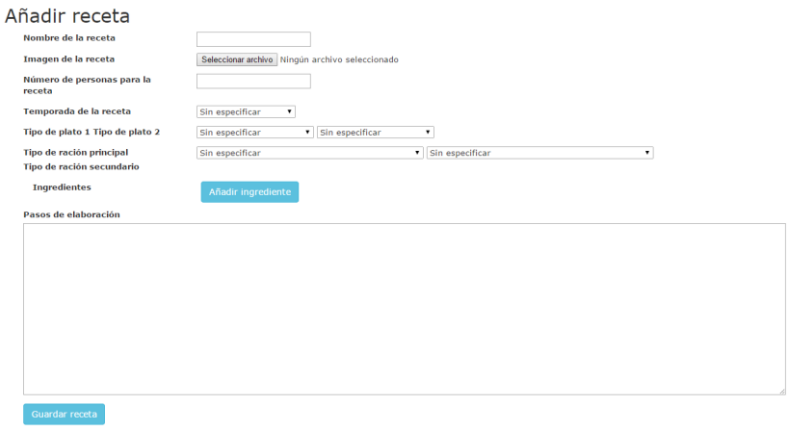
APLICACIÓN DE ADMINISTRACIÓN



Este es el aspecto de la página principal de la aplicación de administración. Desde aquí se puede acceder a los diferentes tipos de contenido haciendo click en el correspondiente apartado de la barra superior.

Recetas

Para añadir una nueva receta, haz click en “Añadir receta”. Se abrirá un formulario donde se añadirá la información de la receta que se está creando:



En ese formulario, se debe proporcionar:

- Nombre de la receta: El título de la receta que se está creando.
- Imagen: Permite subir una imagen de la receta. El servidor las procesará automáticamente con el tamaño y calidad adecuados.
- Número de personas de la receta: especifique con un número la cantidad de personas correspondiente a las cantidades que se están especificando.
- Temporada de la receta: Seleccione la temporada recomendada. Si no desea indicar una en concreto, puede dejar el valor por defecto.
- Tipos de plato: seleccione el tipo de plato principal y secundario. Para que aparezcan opciones adicionales en esa lista desplegable, debe haber creado antes tipos de plato.
- Tipo de ración: seleccione el tipo de ración principal y secundario, debe haber añadido antes los tipos de ración a la base de datos.
- Pasos de elaboración. Redacte el texto que acompaña a la receta para indicar su elaboración.

Para añadir ingredientes, haga click en “Añadir ingrediente”:

Debe especificar una cantidad, la unidad de medida y el ingrediente. Así, si por ejemplo desea guardar “2 cucharadas de aceite balsámico de Módena”, debe asignar “2” como cantidad, “cucharadas” como unidad de medida y “aceite balsámico de Módena” como ingrediente.

Para que se muestren ingredientes en esa vista, debe haberlos agregado antes desde la correspondiente pestaña.

Pulse el botón guardar. Será redirigido al índice de recetas donde podrá observar que su receta ha sido agregada.

Ingredientes

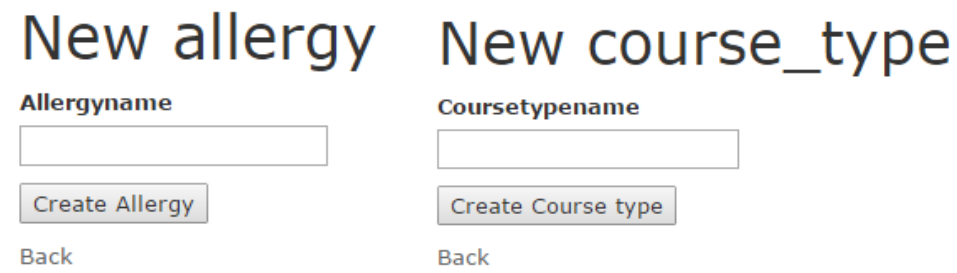
Para añadir ingredientes, acceda a la pestaña “Ingredientes” y haga click en “Añadir nuevo ingrediente”

Añada el nombre del ingrediente. Para añadir alergias, haga click en “Añadir alergia”

Se le mostrará una lista de selección con las alergias que haya agregado anteriormente en la categoría “Alergias”. Seleccione y pulse “Guardar”.

Pulse en “Guardar ingrediente” para almacenar el ingrediente. Podrá comprobar que ha sido agregado y sus alergias han sido asociadas consultando el índice de la tabla.

Alergias, tipos de recetas, tipos de ración



Cada una de estas categorías consta de un campo dónde se indica el nombre de dicha entidad. Al completar ese campo, pulse el botón de guardado. Será redirigido al índice de la categoría, donde se ha agregado una nueva entrada de la tabla.

Ver

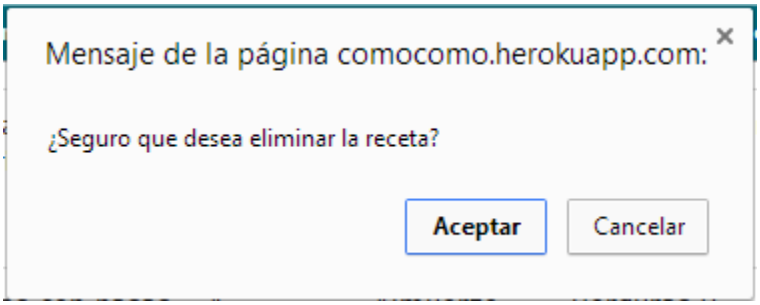
Cada entrada de la tabla de cada categoría cuenta con un botón “Ver” situado a la derecha del elemento correspondiente. Este botón le permitirá ver el contenido de cada uno de los elementos registrados en el sistema:

Editar

Junto al botón “Ver” de cada categoría, se encuentra un botón “Editar”. Este botón le llevará a un formulario con los mismos campos que en la creación, permitiendo realizar modificaciones. Al haber realizado modificaciones, pulse en “Guardar” para que dichos cambios queden almacenados.

Eliminar

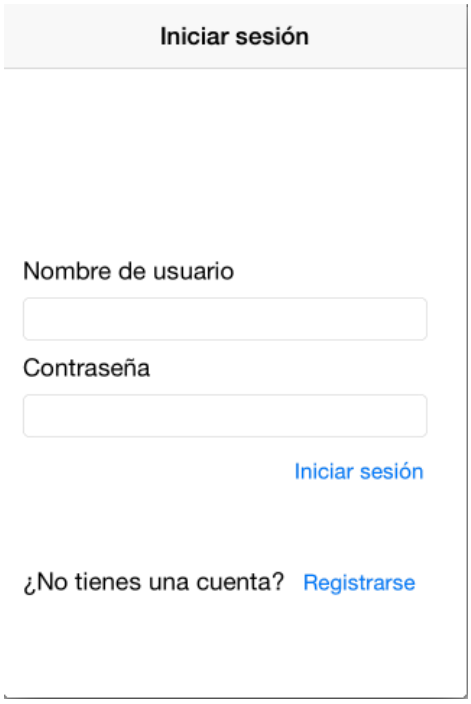
Si desea eliminar definitivamente un contenido, haga click sobre el botón “Eliminar”. Se le abrirá una ventana de advertencia en su navegador preguntando por la confirmación de la eliminación del contenido. Si pulsa en “aceptar”, el contenido se eliminará definitivamente.



APLICACIÓN MÓVIL

Pantalla de bienvenida

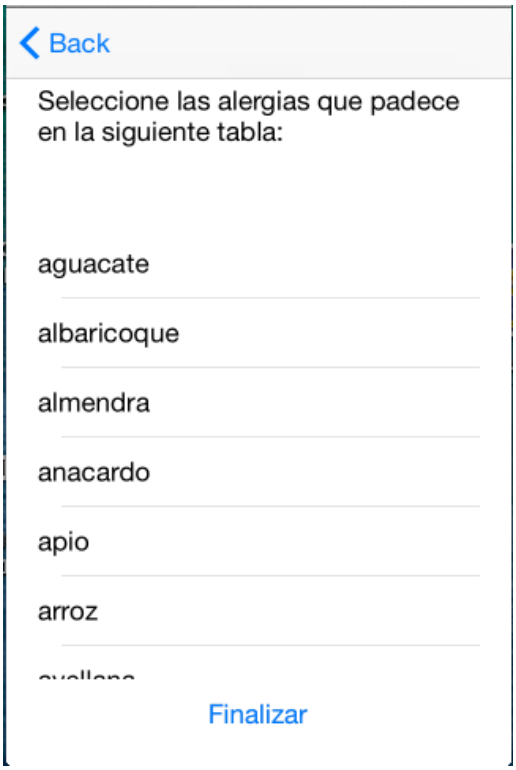
Al iniciar la aplicación, le será presentada la pantalla de bienvenida. Desde esa pantalla puede realizar el inicio de sesión en la aplicación.



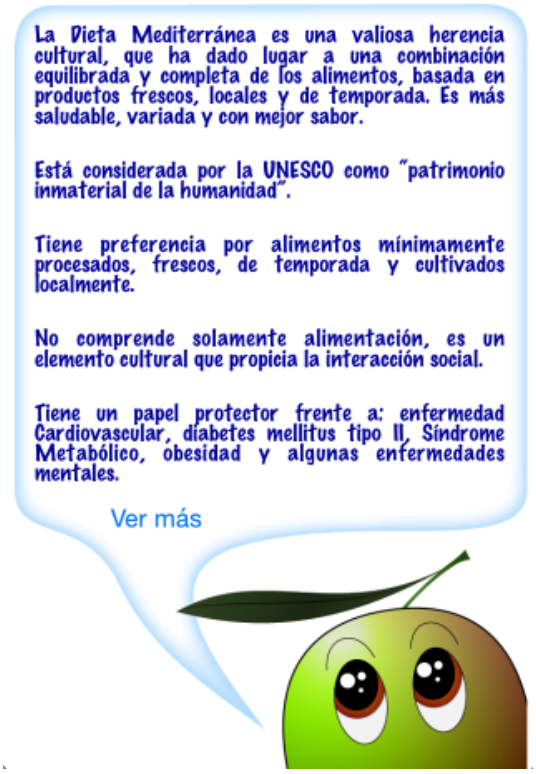
Registro

En caso de que no tenga una cuenta, debe pulsar sobre “Registrarse”. Será dirigido a una pantalla en donde entrará sus datos de creación de la cuenta:

Una vez rellene esos datos, haga click en “Siguiente paso”. En caso de que aparezca una ventana de advertencia, léala atentamente y vuelva a intentar acceder al siguiente paso:



En esta pantalla, seleccione las alergias que padece, si padece alguna, y pulse “Finalizar”. En caso de que aparezca una pantalla de advertencia, lea el contenido y vuelva a intentarlo.



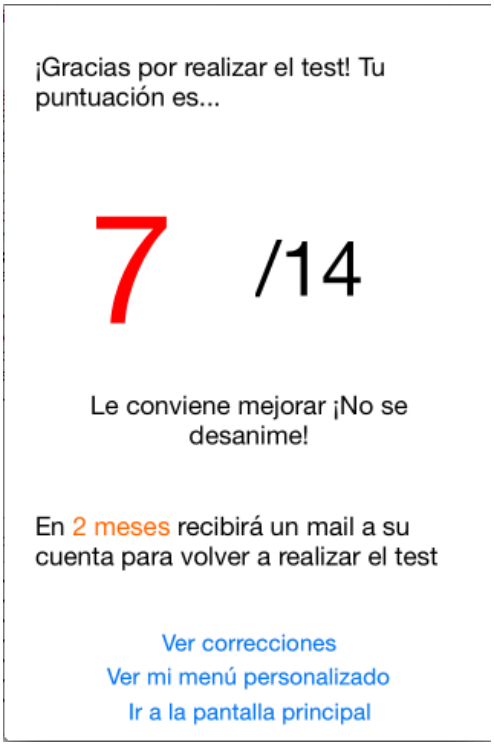
Una vez haya completado el registro de su perfil, verá información sobre la aplicación y sus contenidos. Cuando haya terminado de leer, continúe pulsando sobre "Ver más".

Test

Será informado de que va a hacer un test de adecuación inicial. Acepte el mensaje para comenzar el test:

Responda a cada una de las preguntas con su propia experiencia. Al seleccionar una opción, se le mostrará un botón que le permitirá cambiar a la siguiente pregunta.

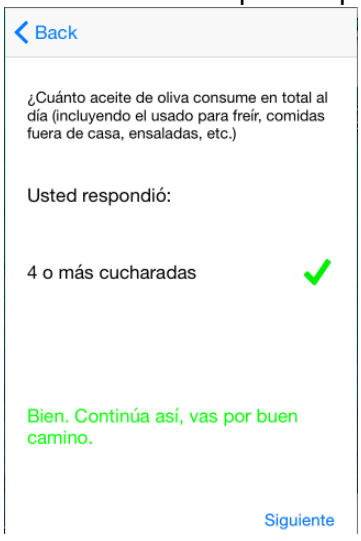
Una vez haya terminado el test, se le mostrará su puntuación:



Desde esa pantalla, usted puede:

- Ir al menú principal, desde el que podrá acceder a todas las opciones de la aplicación.

- Ver la corrección del test: En esta pantalla usted puede comprobar pregunta por pregunta cuál es la respuesta correcta y el por qué. En cualquier momento puede volver atrás pulsando el botón Atrás que se muestra en la esquina superior izquierda:



- Visualizar el **menú personalizado:** En esta sección puede obtener un menú basado en la dieta mediterránea y personalizado teniendo en cuenta las alergias que padece:

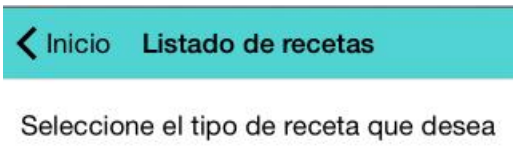
Se le presentará un menú en donde podrá seleccionar la semana de la que quiere obtener la información. Al seleccionar una semana, obtendrá el menú separado por día y tomas de alimentos:

En el caso de los alimentos del almuerzo y las cenas, podrá acceder a la receta pulsando sobre la celda correspondiente:

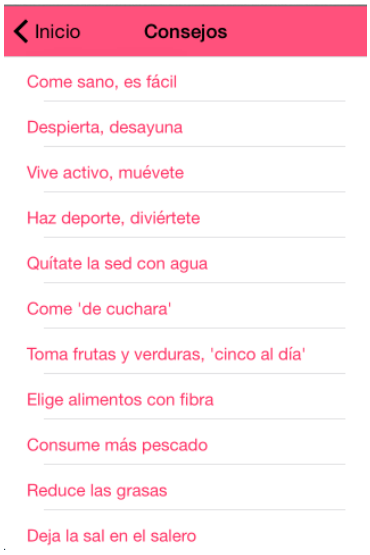
Menú principal

Desde el menú principal podrá acceder a todas las secciones que esta aplicación le ofrece:

- Introducción: le permite volver a ver la introducción que se mostró en el registro de la aplicación.
- Mi menú personalizado: le muestra la pantalla de menú personalizado, tratado con anterioridad en este manual.
- Listado de recetas: le permite acceder a la base de datos de recetas separadas por categorías:



- Los 10 básicos de cocina: donde se muestra una tabla con la receta de los 10 alimentos básicos que se ajustan a la dieta mediterránea.
- Consejos: puede ver los consejos para mejorar sus hábitos tanto alimenticios como del día a día:



- Desglosando la pirámide: aquí puede obtener información en detalle sobre cada uno de los niveles de la pirámide alimenticia, obteniendo recomendaciones para cada caso.
- Alimentos de temporada: seleccionando la estación deseada, podrá obtener información sobre los alimentos que son recomendables consumir en cada caso, dado que están en su mejor momento y además al mejor precio.
- Medidas caseras: contiene material gráfico que le permitirá conocer con mayor exactitud a qué cantidad se corresponden medidas como “cucharadas”, “puñados” etcétera.
- Cantidades recomendadas: Aquí puede obtener, cuáles son las medidas más usuales para los alimentos de consumo diario.

8 BIBLIOGRAFÍA

-RAY, John JOHNSON, Sean. *Desarrollo de aplicaciones para iPhone*: Editorial Anaya,2010. ISBN:978-84-415-2795-9

-RailsGuides;Active Record Basis; http://guides.rubyonrails.org/active_record_basics.html; acceso:10/12/2014

-PEREZ SERNA,Jesús;Medidas de seguridad de nivel alto; http://www.ayudaleyprotecciondatos.es/2011/03/23/medidas-seguridad-nivel-alto;acceso:8/12/2014

- Agencia Española de Protección de datos;Obligaciones; https://www.agpd.es/portalwebAGPD/canalresponsable/obligaciones/index-ides-idphp.php;acceso:8/12/2014

- Apple, Inc.; NSURLConnection Class Reference; https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSURLConnection_Class/index.html; acceso:10/9/2014

-Fundación Dieta Mediterránea;http://dietamediterranea.com;acceso:4/6/2014;

-Cloudinary, Inc;Ruby On Rails integration; http://cloudinary.com/documentation/rails_integration ;acceso:4/9/2014

-Heroku, Inc;Getting Started with Rails 4.x on Heroku; https://devcenter.heroku.com/articles/getting-started-with-rails4 ;acceso:17/10/2014

-Apple, Inc; The Foundation Framework; https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/ObjC_classic/index.html#//apple_ref/doc/uid/20001091 ; acceso 2/12/2014

9 ANEXO: RELACIONES ENTRE CONTROLADORES APLICACIÓN IOS