



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Aplicación de dietas adaptativas

Adaptando tu alimentación a tus necesidades

Autor

Alejandro Olivares del Rey Pierres

Directores

Juan Julián Merelo Guervós



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Junio de 2023

Aplicación de dietas adaptativas: Adaptando tu alimentación a tus necesidades

Alejandro Olivares del Rey Pierres

Palabras clave: frontend, framework, git, backend, dataset, project management, crawler, design thinking

Resumen

A todo el mundo nos ha ocurrido el problema de tener muchos ingredientes en la despensa, al final acabas olvidando algunos, como tomate escondido en el frigorífico o el típico tarro que tenemos al fondo de la despensa detrás de ingredientes que se terminan usando mucho más. Cuando se va a utilizar un ingrediente que sabemos que tenemos, y lo encontramos caducado e inutilizable sentimos una frustración por no recordar su existencia, además de tener que volver a comprarlo en el supermercado. Por ello, en este proyecto se analizará una posible solución a esta problemática por la que todos hemos pasado. Para un correcto desarrollo del proyecto, se utilizará una metodología ágil permitiendo obtener las ventajas del uso de este tipo de metodología. Se necesitarán perfilar los usuarios a los que va dirigida la solución del problema, haciendo uso del diseño orientado al usuario, pensando específicamente que quiere dicho usuario. Para guardar el progreso y llevar una correcta gestión se utilizarán herramientas de control de versiones. Además de analizar los diversos lenguajes que se pueden utilizar para el desarrollo del software, seleccionando el más adecuado dependiendo del tipo de implementación que se realice en el proyecto.

Los contenidos incluidos en la memoria son:

1. Introducción: Una breve introducción al los problemas alimentarios que existen en nuestro país y la descripción del problema a resolver.
2. Estado del arte: Un vistazo a aplicaciones similares que puedan ser competencia directa con la solución que se va a desarrollar.
3. Metodología: Descripción de la metodología a seguir en el proyecto, con las principales claves de la metodología ágil que se buscan seguir en el proyecto, las historias de usuario que han servido de guía a lo largo del mismo, user journeys, perfiles de usuarios objetivo de la aplicación y una descripción detallada de los objetivos a seguir. Además se incluye una descripción de design thinking que recoge el proceso seguido para el desarrollo del proyecto.
4. Planificación. Se recoge la planificación del proyecto desde el inicio del proyecto hasta la elección de la metodología, se expone la dificultad de

la planificación siguiendo una mentalidad ágil centrada en las necesidades del usuario, poniendo como punto de partida el desarrollo de la infraestructura del proyecto y como siguiente objetivo la funcionalidad para resolver el problema del usuario.

5. Implementación: Sección dedicada a la justificación de cada decisión técnica tomada, desde el lenguaje del proyecto hasta el tipo de solución planteada, añadiendo también las decisiones relativas a la infraestructura del proyecto.
6. Recetas. Explicación de la obtención de recetas, su la manera de guardarlas y acceder a ellas y explicación de la solución diseñada para encontrar recetas en la base de datos, añadiendo consideraciones iniciales de la funcionalidad que resuelve el problema a tratar, algoritmo utilizado y breve explicación de la implementación.
7. Prototipo. Explicación del uso de prototipo, junto con el diseño de cada parte que lo compone desde la interfaz, con un diseño inicial y la explicación de la implementación de la misma añadiendo pequeñas funcionalidades para aumentar la usabilidad de la misma, hasta el desarrollo del backend haciendo uso de un framework, estableciendo las conexiones entre el frontend y el backend.
8. Pruebas. Descripción de los test realizados a las vistas de la aplicación que ven los usuarios para encontrar recetas y viajes por la aplicación de los usuarios descritos.
9. Análisis del despliegue. Análisis de como desplegar la aplicación para que sea accesible para todos los usuarios desde el sofá de su casa y explicación del diseño de la infraestructura que se utilizará en el despliegue.
10. Costes: Desglose de gastos en la producción de la aplicación
11. Conclusiones: Consideraciones finales del proyecto y trabajos futuros.

Adaptative diet app: Adapting your diet to your needs

Alejandro, Olivares del Rey Pierres

Keywords: frontend, framework, git, backend, dataset, project management, crawler, design thinking

Abstract

Everyone has had the problem of having a lot of ingredients in the pantry, in the end you end up forgetting some, like tomatoes hidden in the fridge or the typical jar we have at the back of the pantry behind ingredients that end up being used much more. When we are going to use an ingredient that we know we have, and we find it expired and unusable we feel frustrated for not remembering its existence, in addition to having to buy it again at the supermarket. Therefore, this project will analyze a possible solution to this problem that we have all gone through. For a correct development of the project, an agile methodology will be used, allowing to obtain the advantages of the use of this type of methodology. It will be necessary to profile the users to whom the solution to the problem is directed, making use of the user-oriented design, thinking specifically about what the user wants. Version control tools will be used to keep the progress and to manage it correctly. In addition to analyzing the various languages that can be used for software development, selecting the most appropriate depending on the type of implementation to be carried out in the project.

The contents included in the memory are:

1. Introduction: A brief introduction to the food problems that exist in our country and the description of the problem to be solved.
2. State of the art: A look at similar applications that could be in direct competition with the solution to be developed. : Description of the methodology to be followed in the project, with the main keys of the agile methodology to be followed in the project, the user stories that have served as a guide throughout the project, user journeys, profiles of target users of the application and a detailed description of the objectives to be followed. It also includes a description of design thinking that reflects the process followed for the development of the project.
3. Planning. The planning of the project from the beginning of the project until the choice of the methodology, the difficulty of the planning is exposed following an agile mentality focused on the user's needs, putting as a starting point the development of the project infrastructure and as the next objective the functionality to solve the user's problem.

4. Implementation: Section dedicated to the justification of each technical decision taken, from the language of the project to the type of solution proposed, adding also the decisions related to the project infrastructure.
5. Recipes. Explanation of how to obtain recipes, how to save and access them and explanation of the solution designed to find recipes in the database, adding initial considerations of the functionality that solves the problem to be addressed, algorithm used and brief explanation of the implementation.
6. Prototype. Explanation of the use of prototype, along with the design of each part that composes it from the interface, with an initial design and explanation of the implementation of the same adding small features to increase the usability of it, to the development of the backend using a framework, establishing the connections between the frontend and backend.
7. Tests. Description of the tests performed to the views of the application that users see to find recipes and journeys through the application of the described users.
8. Deployment analysis. Analysis of how to deploy the application so that it is accessible to all users from the sofa at home and explanation of the design of the infrastructure to be used in the deployment.
9. Costs: Breakdown of expenses in the production of the application.
10. Conclusions: Final considerations of the project and future work.

Yo, **Alejandro Olivares del Rey Pierres**, alumno de la titulación Grado en ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77166527D, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandro Olivares del Rey Pierres

Granada a 11 de Noviembre de 2023.

D. **Juan Julián Merelo Guervós**, Profesor del Área de ATC del Departamento ICAR de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Aplicación de dietas adaptativas, Adaptando tu alimentación a tus necesidades*, ha sido realizado bajo su supervisión por **Juan Julián Merelo Guervós**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de Noviembre de 2023.

Los directores:

Juan Julián Merelo Guervós

Agradecimientos

A toda la gente que me ha apoyado en el desarrollo de este trabajo. Empezando por mis padres, que se leyeron la primera versión de esta memoria allá por Julio solo para ayudarme a corregir frases que no entendiesen o errores tipográficos. A mis amigos de Almería que siempre me han acompañado en los momentos difíciles. A los nuevos amigos que he hecho por el camino en el transcurso del desarrollo del proyecto. A Marien, que ha aguantado cada bloqueo mental que he tenido y sobretodo a Celia, la persona que más me ha apoyado en este viaje, escuchando todas mis quejas, aguantándome cuando algo no salía bien y había que rehacerlo, animándome cada vez que me encontraba derrotado.

Índice general

1. Introducción	19
1.1. Motivación	19
1.2. Descripción del problema	20
1.3. Objetivos	20
2. Estado del arte	21
3. Metodologías	23
3.1. Marco ágil del proyecto	23
3.2. Design thinking	23
3.2.1. Empatizar - User journeys	25
3.2.2. Definir - Perfiles de usuarios	25
3.2.3. Idear - Soluciones propuestas	28
3.2.4. Prototipo - Bocetos	28
3.3. Uso de git en el proyecto	31
3.4. Planificación	32
3.4.1. Milestones	33
4. Planificación	35
4.1. Introducción	35
4.2. Estado del arte	35
4.3. Metodologías	36
5. Implementación	39
5.1. Herramientas de control de versiones	39
5.2. Herramientas de integración continua	40
5.3. Lenguaje del proyecto	40
5.4. Gestor de dependencias	40
5.5. Gestor de tareas	41
5.6. Fuente de recetas	41
5.7. Base de datos	42
5.8. <i>Test framework</i>	43
5.9. <i>Framework web</i>	43
5.10. Herramienta desacoplar la Base de datos	44

5.11. Web Server Gateway Interface	45
5.12. Servidor Web	45
6. Recetas	47
6.1. Fuente de recetas	47
6.2. Gestión de las recetas en la aplicación	49
6.3. Encontrar recetas	52
6.3.1. Consideraciones iniciales	52
6.3.2. Algoritmo	52
6.3.3. Implementación	53
7. Prototipo	55
7.1. Interfaz	55
7.1.1. Diseño	55
7.1.2. Implementación	57
7.2. Framework	63
7.2.1. Endpoints	65
8. Demostración	69
8.1. Pruebas unitarias	69
8.2. Prueba de la aplicación	70
8.3. Prueba de carga	73
9. Despliegue	75
9.1. Web Server Gateway Interface	75
9.2. Servidor Web	75
9.3. Infraestructura en la nube	76
10. Costes	79
10.1. Infraestructura	79
10.2. Proyecto	79
11. Conclusiones	81
11.1. Futuro del proyecto	81
11.2. Conclusiones	81
Bibliografía	82
A. Base de datos	83
B. Plataforma Web	95
B.1. Interfaz	95
B.2. Framework	104
C. Pruebas	109

Glosario

111

Índice de figuras

3.1. Pasos en el design thinking	24
3.2. Diseño para escritorio	29
3.3. Diseño para tableta horizontal	30
3.4. Diseño para tableta vertical	31
6.1. Función para extraer enlaces	49
6.2. Función para extraer recetas	50
6.3. Identificar los pasos de la receta	51
6.4. Esquema de la Base de datos	51
7.1. Diseño para escritorio	56
7.2. Diseño para tableta horizontal	57
7.3. Diseño para tableta vertical	58
7.4. Diseño del buscador	59
7.5. Diseño del buscador con resultados	60
7.6. Añadir ingredientes a la interfaz del buscador	61
7.7. Resultados del filtrado por ingrediente	62
7.8. Resultados del filtrado por ingrediente en una tablet	62
7.9. Tarjeta de la receta “Lasaña de carne”	63
7.10. Tarjeta de la receta “Budín de bacalao” en una tablet	63
7.11. Código de la tarjeta de la receta	64
7.12. Comandos para la instalación de Django	64
7.13. Estructura del proyecto	65
7.14. Puntos de acceso de la aplicación	66
7.15. Endpoint buscar	66
8.1. Comando para lanzar los test unitarios	70
8.2. Primeros pasos de Daniel	71
8.3. Pasos finales de Daniel	72
8.4. Resultados a la búsqueda de María Encarnación	73
8.5. Resultados de la prueba de carga	74
9.1. Diagrama de infraestructura	77

10.1. Desglose del coste del proyecto	80
A.1. Script para insertar los datos en la Base de datos	84
A.2. Continuación del Script para insertar los datos en la Base de datos	85
A.3. Método para buscar ingredientes por nombre en la Base de datos	85
A.4. Método para buscar ingredientes por ID en la Base de datos	86
A.5. Metodo para obtener los ingredientes de una receta por ID en la Base de datos	87
A.6. Metodo para buscar una lista de recetas por nombre en la Base de datos	88
A.7. Metodo para buscar una lista de recetas por ID en la Base de datos	89
A.8. Metodo para buscar una lista de recetas por su numero de ingredientes y que contengan un ingrediente de la lista en la Base de datos	90
A.9. Metodo para recuperar los ingredientes de una receta por su identificador de la Base de datos	91
A.10. Metodo para recuperar recetas aleatorias de la Base de datos	92
A.11. Clase configuracion para las variables de entorno	93
B.1. Template de los ficheros fuente de la interfaz	95
B.2. Template del navegador de la interfaz	96
B.3. Template del contenedor principal la base de la interfaz	97
B.4. Bloque HTML del buscador	98
B.5. Bloque HTML de la receta	99
B.6. Bloque HTML de la lista de resultados	100
B.7. Evento del buscador	101
B.8. Función para buscar dinámicamente los ingredientes	101
B.9. Función para seleccionar el ingrediente de la lista	102
B.10. Función para obtener la cookie	102
B.11. Función para crear la cookie	103
B.12. Función para eliminar un ingrediente de seleccionados	103
B.13. Función para eliminar un ingrediente de seleccionados	103
B.14. Fichero con los modelos de	104
B.15. Fichero con los endpoint de	104
B.16. Punto de acceso a la página principal	105
B.17.	105
B.18. endpoint para buscar recetas por ingredientes	106
B.19. Método para buscar recetas aleatorias	107
B.20. endpoint para ver detalles de una receta	107
B.21. endpoint para ver las recetas guardadas del usuario	108

C.1. Pruebas de la funcionalidad encontrar recetas	110
C.2. Script para la prueba de esfuerzo de Locust	111

Capítulo 1

Introducción

1.1. Motivación

En la , debido a una gran concienciación impartida por diferentes organismos orientados a la salud, ha aumentado la preocupación por mantener una alimentación saludable y equilibrada. Sin embargo, la gran mayoría de personas no tienen los conocimientos suficientes como para elaborar una dieta adaptada a sus necesidades individuales. Ya no solo estamos hablando de personas que buscan bajar peso, sino de personas que tienen algún tipo de intolerancia o alergia alimentaria.

Según la Fundación de Seguridad Alimentaria (FSA), en el mundo existen unas 520 millones de personas con este tipo de problemas alimentarios. Solo en España se estima que entre el 1-3 % de los adultos y un 4-6 % de los niños padecen algún inconveniente al consumir determinados alimentos. Además, debemos diferenciar a las personas con intolerancias alimentarias de las que sufren alergias. Ya que no tienen los mismos efectos aunque a primera vista sean parecidos. Las alergias se producen cuando el sistema inmune entra en contacto con el alérgeno, mientras que las intolerancias están ligadas a la dificultad para digerir determinados alimentos.**[FSA]**

Cualquier persona sabe qué es una dieta, ya que todos nos alimentamos cada día. La palabra “dieta” se refiere al conjunto de alimentos que ingerimos. Sin embargo, no todo el mundo lleva una dieta equilibrada. Existen múltiples tipos de dietas, como omnívora, vegetariana, vegana, paleo, mediterránea, entre otras, las cuales se diferencian por los alimentos que se consumen. Es un error común pensar que simplemente al comer de manera saludable se está siguiendo una dieta equilibrada. Para lograr una dieta equilibrada, es necesario consumir todos los nutrientes esenciales que el cuerpo necesita para mantener una salud óptima. Una dieta equilibrada se caracteriza por ser variada, moderada y proporcionada.

Cuando hablamos de variedad, nos referimos a la inclusión de alimentos de todos los grupos: frutas, verduras, cereales, legumbres, carnes, pescados,

etc. Por supuesto, hay dietas que excluyen ciertos alimentos; en esos casos, se debe buscar una fuente alternativa de nutrientes. Por otro lado, una dieta debe ser moderada, lo que significa que se deben ingerir las cantidades necesarias de nutrientes sin excederse.

1.2. Descripción del problema

A todo el mundo nos ocurre el tener un montón de ingredientes en la despensa sin saber que hacer con ellos, que se terminan acumulando o caducando, haciendo que sea un derroche de dinero. El problema que se quiere resolver con el desarrollo de esta aplicación es el mal aprovechamiento de los ingredientes que se tienen por casa y el desconocimiento de determinadas recetas que solventen esa mala gestión.

El proyecto va dirigido a usuarios que cuentan con recursos económicos limitados, como un estudiante con un presupuesto ajustado, o personas con movilidad reducida que no puedan permitirse acudir al supermercado cada vez que les falte un ingrediente.

Pero el uso de esta aplicación no está limitado a estos usuarios descritos, sino que puede ser utilizada por cualquier persona que quiera aprovechar los ingredientes que tengan sueltos por la cocina.

En primer lugar, necesitamos conocer a fondo el problema descrito, para comprender toda la complejidad que pueda tener. Pensando un poco es posible encontrar mucha profundidad a la hora de resolver el problema, por ejemplo, que se entiende por ingrediente o que ingredientes tiene todo el mundo en su casa. Estas preguntas irán surgiendo a medida que avance el desarrollo del proyecto.

Para resolver el problema descrito se utilizarán las mejores prácticas dentro de un enfoque ágil.

1.3. Objetivos

1. Búsqueda básica de recetas. El usuario debe ser capaz de encontrar recetas utilizando los ingredientes que desee.

Capítulo 2

Estado del arte

Después de una investigación inicial en este campo buscando en diversos blogs de nutrición como *HealthLine*[Ans22] y revistas como *Bussines Insider*[Sto21] y *The Guardian*, donde se habla de aplicaciones que permiten desde desarrollar una dieta utilizando recetas publicadas por la comunidad o simplemente con los ingredientes que tenemos en nuestro frigorífico.

Una de las más interesantes de las que se trata es *Yummly*, una aplicación disponible tanto en *iOS* como en *Android* que se adapta a las necesidades individuales de cada persona, incluyendo intolerancias alimentarias[Dre16]. Esta ofrece un plan gratuito que permite buscar recetas, recomendaciones personalizadas, una lista de la compra. Además cuenta con un plan de pago que cuesta 4,99\$/mes, que incluye la creación de dietas personalizadas, información nutricional de las recetas, un motor de búsqueda por ingredientes que permite filtrar por ingredientes, alergias o tipo de dieta. Pero, desafortunadamente, no está disponible en España.

Una aplicación quizá más conocida es *Cookpad*, esta es una de las aplicaciones más descargadas de *Apple Store* y *Google Play* en la sección de cocina. Una de sus características principales es su comunidad, donde es posible compartir recetas y fotos de tus creaciones. Es posible usar esta aplicación de manera gratuita pero cuenta con algunas limitaciones. Por otra parte, la versión pro cuesta 2,99\$/mes y entre las funcionalidades que ofrece se encuentran planes de menús semanales elaborados por nutricionistas.

Otra de las aplicaciones más interesantes que se encontraron fue *Eat This Much*. Esta es una aplicación web, también disponible para *Android* e *iOS*, que permite generar una dieta de manera automática basándose en:

- Las calorías que se quieren ingerir al día.
- El número de comidas que se desea realizar.
- El tipo de dieta que se quiere generar.

Eat This Much tiene una versión gratuita abierta al usuario. Al generar la dieta es posible cambiar el plato aleatoriamente entre los que se recomien-

dan. Pero, el principal problema se basa en que la personalización de la dieta es muy general, sin tener en cuenta las intolerancias del usuario. De la misma manera, ofrecen una suscripción orientada a profesionales y entrenadores permitiendo que estos recomiendan generen las dietas a sus clientes de manera totalmente personalizada. Esta suscripción cuesta 79,00\$/mes y solo incluye servicio para diez clientes, cada cliente extra costará 4\$/mes.

La última aplicación de la que se hablará es *HappyForks*, que se trata de una herramienta que permite monitorizar una gran cantidad de valores nutricionales. Contando con una gran cantidad de utilidades para controlar una dieta ajustándose a las necesidades del usuario y una gran base de datos llena de recetas. Una de las utilidades que ofrecen permite analizar recetas y productos mostrando los ingredientes para no causar alergias al usuario.

El proyecto que se busca desarrollar tiene como objetivo crear un software que permita a los usuarios encontrar recetas de manera sencilla utilizando los ingredientes que ya tienen en su despensa. Esto facilita tener todas las recetas reunidas en un solo lugar, organizadas y disponibles en todo momento. Además, el software también puede ayudar a los usuarios a aprender nuevos estilos de cocina y experimentar con nuevos ingredientes. Sobre el modelo de negocio, se debería analizar detenidamente si contará con una suscripción mensual con diversas ofertas o si se liberará añadiendo anuncios a la aplicación con la posibilidad de eliminarlos mediante una pequeña transacción.

Capítulo 3

Metodologías

En todo proyecto es necesaria una metodología que ayude a guiar el siguiente paso, ya que en realidad el paso más importante es el siguiente y no el primero. Por esta razón, en este capítulo se describirá la metodología utilizada así como la planificación del proyecto.

3.1. Marco ágil del proyecto

Debido a las limitaciones de la gestión de proyectos, a finales de los 90 se empezó a utilizar una metodología ágil. Pero no fue hasta 2001 que un grupo de expertos desarrolla el “Manifiesto por el Desarrollo Ágil de Software”. Donde se especificaron los doce principios clave para el desarrollo de software. A partir de ese punto, cada vez han ido surgiendo más modelos ágiles. Los 12 conceptos clave[Pér16] a seguir en este proyecto son

1. Los cambios son bienvenidos
2. Software funcional en un periodo corto de tiempo
3. Facilidad para medir el progreso
4. Desarrollo sostenible
5. Excelencia técnica y buen diseño
6. Simplicidad
7. Adaptación a circunstancias cambiantes

3.2. Design thinking

En el proyecto, se aplicará el enfoque del *Design Thinking*, en el que sus primeros pasos implican realizar un análisis del problema empatizando con

los usuarios y para la definición del usuario se utilizará la metodología de personas. Para empatizar con el usuario y comprender a fondo el problema que están experimentando, se debe asumir el rol de usuario y ponerse en su lugar para identificar los posibles obstáculos que podrían tener al cocinar con ingredientes específicos sin conocer una receta adecuada. En este caso, el proceso de *Design Thinking* nos ayudará a resolver el problema del usuario al permitirle encontrar recetas que se adapten a los ingredientes de su elección, abordando así sus necesidades de manera efectiva.

El concepto nació a finales del siglo XX como un compuesto que se basa en disciplinas como el diseño industrial y gráfico, además de la ingeniería y arquitectura. Este se refiere a estrategias creativas que utilizan los diseñadores durante el proceso de diseño. También es utilizado tanto como para abordar problemas como para desarrollar una solución, no solo en el ámbito de la ingeniería sino en ámbitos como empresas o cuestiones sociales. El *Design Thinking* aprovecha los métodos a disposición del diseñador para hacer coincidir las necesidades de las personas con lo tecnológicamente factible, que a su vez se puede convertir en valor de mercado para una empresa.

Se compone de cinco estados que se deben usar en el proceso. Es muy importante no saltarse ningún paso

1. Empatizar
2. Definir
3. Idear
4. Crear prototipos
5. Pruebas

Figura 3.1: Pasos en el design thinking

Se determinan las características de los usuarios objetivo de los que se van a beneficiar de la aplicación. Así es posible obtener toda la información posible sobre los usuarios del producto y sus necesidades. Este paso ya se ha realizado anteriormente y se han especificado los clientes de la aplicación.

La siguiente etapa del Design Thinking es la definición del usuario, es el paso más difícil ya que normalmente el desarrollador del proyecto tiende a idear soluciones familiares para él, pero se tiene que poner al usuario por delante de la comodidad del desarrollador. Se perfilarán los usuarios del proyecto, detallando su entorno, conocimientos, círculos sociales, motivaciones, etc... Para comprender qué necesidades tienen de la aplicación. Basándonos en esas necesidades irán surgiendo nuevos objetivos y metas que alcanzar.

El paso de idear involucra la parte creativa del diseño, se debe intentar

generar la mayor cantidad de ideas posibles, teniendo en cuenta solo las más importantes o realistas en el proyecto. Y una vez que se ha realizado una lluvia de ideas se seleccionarán las más relevantes y se creará un prototipo de aplicación, o varios dependiendo del proyecto. Para la fase de pruebas, donde se debería llegar al mayor público posible para obtener la mayor cantidad de retroalimentación de cara a futuras versiones de la aplicación o el descubrimiento de nuevas necesidades del usuario. [Wol17]

Para realizar los pasos futuros del se tendrá en cuenta lo explicado en este apartado, en la etapa de ideas se anotarán las ideas más relevantes para la solución al problema de los usuarios. Posteriormente se escogerán las ideas más relevantes y razonables para crear prototipos de la aplicación planteada en el proyecto. Y, por último, se presentará el prototipo a determinados usuarios que puedan aportar retroalimentación útil para posibles mejoras o nuevas necesidades que se deban cubrir.

3.2.1. Empatizar - User journeys

En primer lugar se tratará de resolver el problema de un estudiante que cuenta con un presupuesto ajustado al mes y no puede permitirse el lujo de comprar en el supermercado todos los días. Dicho estudiante se siente frustrado por no ser capaz de controlar sus gastos en el supermercado y que ciertos ingredientes que no utiliza asiduamente le caduquen en la despensa. Por casualidad descubre el software desarrollado en este proyecto y se lo descarga. Una vez descargado abre la aplicación e introduce los ingredientes que tiene en su despensa. El estudiante podrá encontrar a su disposición diferentes recetas que realizar con los ingredientes introducidos. Una vez seleccionada la receta, el usuario puede comprobar los detalles de la misma.

Otro problema que se tratará de resolver es el que tiene una persona mayor que no se encuentra en disposición de ir al supermercado cada vez que le faltan ingredientes y tiene una despensa limitada. De pronto escucha la existencia de la aplicación y se dispone a probarla. Después de descargarla, con dificultad debido a sus achaques de la edad y a su dificultad para entender las nuevas tecnologías, la abre. En primer lugar debe introducir los ingredientes que tiene por casa y aunque encuentra cierta dificultad sustituir una nota de papel, con esfuerzo consigue hacer un inventario de sus ingredientes. La aplicación en este momento le podrá empezar a sugerir recetas con los ingredientes que tiene en su despensa. Después de elegir una receta, puede comprobar los detalles de la misma con detenimiento.

3.2.2. Definir - Perfiles de usuarios

Para definir a los usuarios se utiliza la metodología de personas.

El concepto surge del intento de comprender coherentemente fue muy utilizado en los 90. Antiguamente en marketing para definir audiencias se

utilizaba un método menos preciso. Se pensaba en personas que pudiesen ser un *target* para el producto, teniendo en cuenta pocas características. En un ejemplo de productos de bebé, se consideraba la edad de los padres, la del bebé y opcionalmente el poder adquisitivo. Con la metodología de persona es posible profundizar mucho más a la hora de identificar audiencias.[Ani19]

Para construir personas se detallan una serie de preguntas que permiten perfilar el cliente del producto. Se responderán las siguientes preguntas:

1. ¿Cómo se llama?
2. ¿Qué edad tiene?
3. ¿A qué se dedica?
4. ¿Qué idiomas domina?
5. ¿Cómo es la persona?
6. ¿Qué dispositivos tiene?
7. ¿Cuál es su entorno social?
8. ¿Qué le motiva?
9. ¿Cuál es la frase que le inspira?

Perfil de un estudiante

- Nombre: Daniel Pérez
- Edad: 22 años
- Ocupación: Estudiante de Bioquímica
- Idiomas: Español, inglés y francés
- Descripción general: Daniel cursa el tercer año de su carrera en Bioquímica en una universidad pública, ansioso por frenar el hambre en el mundo, investigando una manera de crear cultivos que arraiguen en climas extremadamente secos. Divide su tiempo entre las clases, trabajos y amigos. Lee multitud de investigaciones científicas actuales, estando muy interesando en este mundo.
- Dispositivos: Daniel utiliza asiduamente el ordenador y vive enganchado al teléfono.
- Entorno social: Las amistades de Daniel son también estudiantes que están acabando sus estudios y se plantean iniciarse en el mundo laboral. Se dedican a diferentes campos no solo científicos. Pero todos están muy concienciados con las causas sociales. En el ámbito familiar, Daniel tiene una relación cercana con sus padres

y su hermano Jorge. Realizan actividades habituales en familia y están todo el tiempo que pueden juntos. Aunque él viva en un piso para estudiantes con tres personas más.

- Motivación: Daniel busca reducir el derroche de alimentos que se ponen malos en su piso. Ya que en ocasiones se olvidan de algún ingrediente que se termina poniendo malo. Además, cuenta con unos recursos realmente limitados.
- Cita: Erradicar el hambre en el mundo es una responsabilidad compartida.

Perfil de una persona mayor

- Nombre: María Encarnación Sánchez
- Edad: 85 años
- Ocupación: Jubilada
- Idiomas: Español
- Descripción general: María Encarnación vive en un pequeño barrio al norte de Granada, donde las calles son demasiado empinadas para ir tirando del carrito de la compra por los inmensos escalones hasta el supermercado y vuelta. Muchos días piensa que ya no tiene edad para estar yendo a comprar cada dos o tres días, aunque no tenga mucho más que hacer. Se pasa las mañanas realizando tareas del hogar. Aunque cuando encuentra un hueco, se arregla para ir a su parroquia, ubicada a tan solo unas pocas calles de su casa. Además, con la edad cada vez es más olvidadiza y cada poco se tiene que poner a buscar el papel de la receta para comprobar los ingredientes o pasos que no recuerda.
- Dispositivos: Aunque María Encarnación tiene un ordenador que le regaló su hijo, no sabe usarlo más que para buscar cosas en internet. Y el móvil que tiene es tan pequeño que apenas ve lo que pone en la pantalla. Pero el dispositivo que más utiliza se trata de una tablet muy antigua que le regalaron por navidad y usa todos los días mientras se mece en la butaca.
- Entorno social: El círculo más cercano de María Encarnación son sus amigas de la iglesia, su hijo voló del nido hace más de cincuenta años y su marido falleció hace tan solo 2 años. Cuando ocurrió se refugió en la Fe cristiana para sobrellevar la pérdida. Por ello acude cuando puede a rezar por el alma de su marido y tiene un círculo cercano de señoras de su edad.
- Motivación: María Encarnación ya no tiene edad para ir a comprar cada vez que le falta algún ingrediente y su memoria no es lo que era, no recuerda muchas recetas de su juventud.

- Cita: La edad puede nublar la memoria de nuestras recetas favoritas, pero nunca desvanece el sabor de los momentos compartidos

3.2.3. Idear - Soluciones propuestas

Para pasar de un problema a una solución, necesitamos idear una solución. En este paso del design thinking. Es un proceso en el que se combina el entendimiento al problema del usuario con nuestra imaginación para diseñar la solución. Uno de los principios fundamentales de este proceso es la mentalidad de “No hay malas ideas”, pero hay que tener en cuenta las ideas que se pueden realizar, teniendo en cuenta las necesidades del usuario. [con]

Después de reunir algunas ideas, se deben descartar aquellas que no sea posible realizar en este momento o que no se alineen con las necesidades del usuario. Siendo las personas mayores uno de los grupos objetivo para solucionar su necesidad, es necesario tener en cuenta sus limitaciones.

Las ideas razonables técnicamente que se han recogido son las siguientes:

- Realizar una biblioteca para filtrar recetas desde una fuente
- Realizar una API que permita recoger recetas a través de peticiones web
- Realizar una aplicación web completa que permita al usuario obtener recetas a través de una interfaz web

Ya que estas tres ideas no son incompatibles, se implementarán poco a poco estas ideas para ofrecer la solución a la mayor cantidad de usuarios posibles. Comenzando por ofrecer la solución a los usuarios perfilados en la etapa anterior. La mejor manera de ofrecerla directamente es permitiendo que estos usuarios puedan interactuar con ella, la idea más razonable para este pensamiento es la implementación de una aplicación web. Pero más adelante se pueden ofrecer las otras dos ideas para otros tipos de usuarios, como desarrolladores de *software*.

3.2.4. Prototipo - Bocetos

El objetivo de esta fase es producir versiones mínimas o baratas del producto, para investigar las soluciones exploradas. En primer lugar, se desarrollará una aplicación web con una interfaz sencilla que permita al usuario interactuar y dar *feedback*.

No solo es un mecanismo para probar la funcionalidad del servicio, además se trabajan en distintos niveles:

- Ganar empatía
- Explorar otros conceptos para testear en paralelo
- Inspirar a otras personas al mostrar la visión

El prototipo comienza con un diseño de baja fidelidad que imprimir al prototipo para luego mejorar la solución iterativamente. Se utilizarán los siguientes bocetos para la solución web, tanto para computadores como para dispositivos móviles.

Se ha creado un mockup de la aplicación web. Además se han generado diagramas de bajo nivel para cada tipo de dispositivo para acceder a la aplicación web.

Computadora

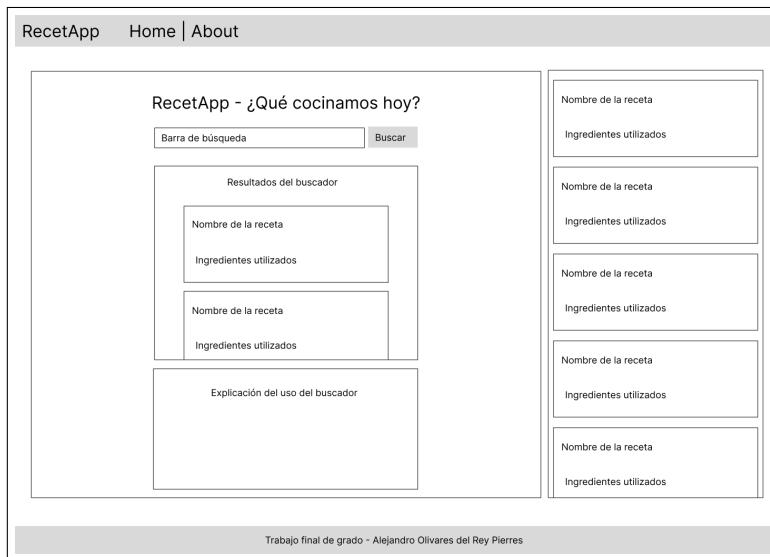


Figura 3.2: Diseño para escritorio

En la figura superior se presenta la interfaz con un diseño para escritorio. Se utiliza una barra superior a modo de navegador de la página web, con un botón para volver a la página principal y un buscador que permite buscar recetas por nombre.

En cuanto al diseño central de la aplicación, se puede observar una clara división en dos zonas. El buscador de recetas por ingredientes y una columna a la derecha con algunas recetas aleatorias para inspirar al usuario, estas irán cambiando mientras el usuario utiliza la aplicación. Los resultados del buscador aparecerán bajo la barra de búsqueda.

Dispositivos móviles

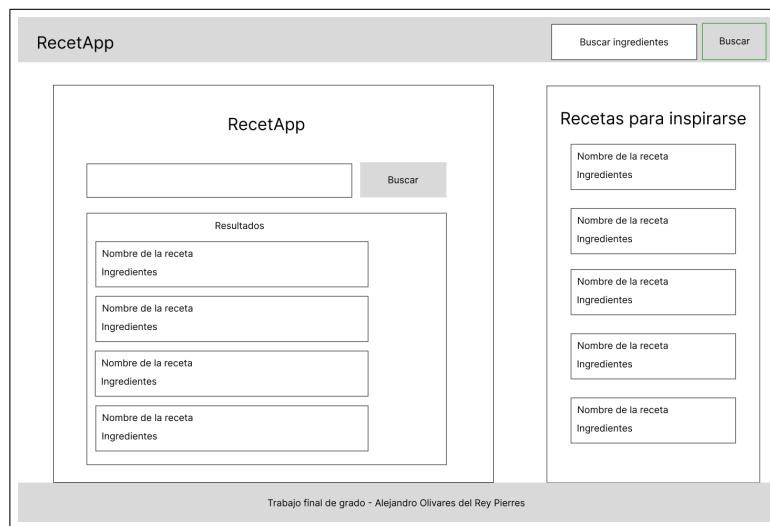


Figura 3.3: Diseño para tableta horizontal

La figura superior muestra un diseño para una tableta en modo horizontal, siendo muy parecida al diseño de escritorio, excepto el navegador que colapsaría mostrando el botón para desplegarlo y la desaparición del cuadro que explicaría el uso del buscador.



Figura 3.4: Diseño para tableta vertical

3.3. Uso de git en el proyecto

Git se trata de una herramienta de control de versiones que surge de la necesidad de llevar un seguimiento del código fuente y sus actualizaciones. Es el núcleo de herramientas como *GitHub* y *GitLab*. Cualquier equipo de desarrolladores, que cuente con más de un trabajador, debe hacer uso de una de estas herramientas.

Ambas herramientas proporcionan un marco para la gestión de proyectos. Permitiendo que diferentes equipos implementen concurrentemente *features* en el software creando ramas a partir del código principal. Cualquier cambio realizado debe reflejarse con su correspondiente *issue*, dotando de coherencia dichos cambios. Estos no solo sirven para reflejar cambios en el código o correcciones, en un proyecto muy grande también permiten llevar un seguimiento de en qué trabaja cada miembro. Todos los *issues* deben estar vinculados a un objetivo o *milestone*. Estos últimos constituyen los entregables (producto mínimo viable) a alcanzar o, lo que es lo mismo, la planificación del proyecto.

Al comenzar todo proyecto, la orientación del mismo se reflejará como una serie de historias de usuario especificadas en *issues* y la planificación haciendo uso de *milestones*.

Para añadir código e ir cerrando *issues*, se utilizan (o PR para abrir). Estas son solicitudes para mezclar código de una rama a otra. Permiten cerrar problemas que pueda tener el código, reflejados en *issues*. Puede darse el caso en el que se quieran mezclar *commits* (o cambios) para cerrar *issues* pero no todos los cambios porque algunos no deben pasar de un *milestone* al siguiente.

En el proyecto ha sido necesario el uso de *cherry-picking* para extraer *commits* de una rama extensa que contenía cambios que no debían mezclarse con la rama principal. Que han sido extraídos a una nueva rama y mezclados con éxito en la rama principal.

cherry-picking es un proceso por el cual se escogen *commits* manualmente de una rama para introducirlos en otra. El problema es la dificultad de conocer que cambios contiene cada *commits* y en cual deberían ser introducidos. Se utiliza normalmente en ramas de desarrollo muy largas que no pueden ser mezcladas, los desarrolladores escogen los *commits* haciendo uso de *cherry-pick* y reusar el código en otras ramas.[BP17]

Es posible usar *cherry-pick* desde la línea de comandos de *Git* pero es muy lioso, en el proyecto se ha utilizado la herramienta de Github Desktop para extraer los *commits* de manera sencilla.

3.4. Planificación

En este proyecto se usará una mentalidad ágil para el desarrollo del mismo, esto significa que no se sabe a largo plazo que objetivos se van a cumplir, ya que solo se puede planificar como mucho un objetivo o dos por delante. Pero hasta que no se llega al final de un objetivo no se puede saber realmente por donde se va a seguir avanzando.

Para dar un poco más de contexto, la metodología es la que nos guía para definir el objetivo que debemos alcanzar en cada etapa del proyecto. Una vez que alcanzamos un objetivo, también nos ayuda a establecer el siguiente. En este caso particular, se ha definido únicamente el primer objetivo, que es satisfacer la necesidad del usuario de encontrar recetas.

Debido a que el desarrollo ágil está centrado en el cliente, no es posible planificar los objetivos que se van a cumplir en el desarrollo del proyecto de antemano.

En el proyecto se utilizarán *milestones* para medir el desarrollo del progreso, estos son puntos de referencia que indican cuando ha concluido una iteración del desarrollo. También representan un producto mínimo viable o un entregable para el cliente.[Tha22]

Antes de echarse a definir una solución, el proyecto también requiere de una infraestructura mínima que se debe tener en cuenta para el correcto transcurso del desarrollo ágil. Esta infraestructura constituye como el primer *milestone* en el proyecto.

Como mínimo, se espera que se elija razonadamente un lenguaje de programación, un gestor de tareas para automatizar llevar a cabo pequeñas labores como la generación de la memoria del proyecto. Por último, se debe añadir un gestor de dependencias que permita fácilmente reproducir el entorno de desarrollo en cualquier computadora que sea necesaria.

El siguiente milestone especificado constituye permitir al usuario encontrar recetas con ciertas restricciones, para ello se establecerán pequeños subobjetivos para alcanzar el milestone y se resolverán cuestiones relativas al objetivo, por ejemplo, que se entiende como ingrediente o receta. Se desestimarán las consideraciones que no avancen el objetivo de cubrir la necesidad del usuario hasta que se complete el milestone relativo a encontrar recetas.

Inicialmente, se especifican los *milestones* mencionados, pero debido a la mentalidad ágil es imposible planificar *milestones* más allá, porque no se sabe qué necesidades tendrá el usuario a lo largo del desarrollo del proyecto o qué nuevos problemas se pueden plantear y por tanto se especificarán en la sección implementación

3.4.1. Milestones

Para resumir, la lista de *milestones* iniciales mencionados anteriormente son los siguientes

- a) Despliegue de la infraestructura mínima
- b) Encontrar recetas

Capítulo 4

Planificación

En este tercer capítulo se narrará la planificación seguida en el proyecto, desde la definición del problema a resolver hasta la especificación de las metodologías que se utilizarán en el proyecto.

4.1. Introducción

En el apartado de introducción se detalla la motivación del proyecto, exponiendo los motivos que llevan al desarrollo del proyecto, además se determina que se considera como “dieta equilibrada”, ya que no todo el mundo tiene el mismo concepto y, adicionalmente, que se considera una “dieta variada”.

Por otra parte, se describe el problema desde el punto de vista de los clientes de la aplicación. El problema que se va a resolver es la necesidad que tiene el usuario para encontrar nuevas recetas, o recordar antiguas, utilizando los ingredientes a su disposición.

Una vez descrito el problema, fue necesario describir a quien se dirige la solución, se determinó que existen dos tipos de cliente a los que se resuelve la necesidad:

- a)* Estudiante
- b)* Persona mayor

4.2. Estado del arte

En este capítulo se definen las principales soluciones existentes similares a lo que se busca desarrollar. Se realizó una búsqueda de aplicaciones que pudieran ser parecidas o con una funcionalidad que sea una solución al problema planteado.

Se encontraron algunas aplicaciones, en blogs de nutrición y revistas [Ans22][Dre16][Sto21], que cumplen los requisitos especificados antes, la mayoría son de pago o imponen restricciones a la hora de escoger ingredientes que se utilicen en la búsqueda de recetas.

Se han detallado las más relevantes o parecidas que se han encontrado.

4.3. Metodologías

En esta sección inicialmente se describe el concepto de marco ágil y las ventajas que se aprovecharán en el desarrollo del proyecto.

Para el proyecto se plantea el uso de *Design Thinking*, realizando una pequeña investigación leyendo diversos artículos académicos[Wol17][TM+11] que han servido de guía para obtener un entendimiento del concepto de *Design Thinking* y sus etapas en el desarrollo. Además, se ha especificado el avance en las primeras etapas del proyecto.

La etapa de empatizar con el usuario objetivo se completa pensando en las necesidades que hay que cubrir del usuario. Especificando el viaje del usuario () en la aplicación. Un es una herramienta que permite representar paso a paso el proceso que necesita el usuario para alcanzar su objetivo, en este caso encontrar una receta que se adapte a él. [Gas15]

Para definir a los usuarios se utiliza la metodología de persona, que permite definir al usuario por medio de una serie de información planeada por medio de preguntas para conocer a fondo al cliente, tanto sus necesidades como sus motivaciones.

Las preguntas propuestas para el perfilado del usuario permiten entender las necesidades de los clientes potenciales y abasteciéndolos de las funcionalidades o servicios que necesitan.

El proyecto más complejo que el desarrollar una aplicación, es muy importante llevar una correcta gestión del mismo. Para llevar a cabo esa gestión se utiliza *Git*, una herramienta de control de versiones que permite trabajar siguiendo una mentalidad ágil resolviendo poco a poco *issues*.

Existen diferentes herramientas que permiten el uso de *Git*, se utiliza *GitHub* por ser la herramienta con la que he trabajado y tengo mayor experiencia en su uso. Pero existen otras herramientas como *Jira* o *GitLab* en las que tengo menos experiencia.

En la gestión del proyecto se han afrontado algunos desafíos como la correcta creación de *issues* que definan de manera corta un problema

o desafío que se necesita resolver para continuar el desarrollo. La definición de *milestones* y que se espera de cada hito también ha supuesto un desafío en esta etapa del proyecto, debido a la poca experiencia que poseía al abordar el proyecto con una mentalidad ágil.

En un punto del desarrollo del *milestone* se desarrolló una parte del proyecto de manera prematura, utilizando una rama para ello. Para afrontar el problema de manera organizada, se abrieron lo más atómicos posibles con el objetivo de cerrar los *issues* relativos a este milestone, dejando pendientes los prematuros. Una vez completados todos los *issues* se tomó la decisión de dejar la rama abierta sin mezclar y extraer los cambios realizados para cerrar los problemas abordados. Para ello, se utiliza la operación *cherry-pick*, siendo mejor opción que eliminar la rama y tener que repetir los cambios para solventar los *issues*. Desde una nueva rama con los cambios extraídos de la problemática, se mezclaron con la rama principal sin inconvenientes.

En la planificación del proyecto se tomó la decisión de especificar únicamente los primeros *milestones* del proyecto debido a la imposibilidad de conocer el camino que tomará el desarrollo siguiendo una mentalidad ágil.

Capítulo 5

Implementación

En todo proyecto existen una serie de decisiones que marcan el avance del proyecto. En este capítulo se describirán, detalladamente, las justificaciones de cada decisión que se tome durante el desarrollo del mismo. Otorgando tanto al lector como a cualquier desarrollador que busque continuar este proyecto, un entendimiento de porqué se han tomado dichas decisiones.

5.1. Herramientas de control de versiones

Como se explicó en capítulos anteriores, *Git* es una herramienta de control de versiones muy poderosa que permite mantener una organización en los cambios realizados en el código de un proyecto por medio de *issues* y *commits*.

Es posible trabajar con *Git* localmente, pero se consiguen mejores resultados al utilizar una plataforma en línea para almacenar estos cambios. Para mantener la coherencia en los cambios del proyecto es necesario elegir una plataforma para el control de versiones. Las dos plataformas consideradas son *GitHub* y *GitLab*.

Dichas plataformas son muy parecidas, dificultando la elección de una de ellas objetivamente. Ambas permiten la gestión del proyecto por medio de un repositorio remoto y están basadas en *Git*. La mayor diferencia que existe entre ambas es el objetivo para el que se utilizan. GitHub es una plataforma colaborativa que ayuda a revisar y gestionar el código remotamente. Mientras que GitLab está más enfocado a proyectos de DevOps y CI/CD. [Vat22]

En el proyecto se utiliza GitHub por ser la única herramienta de control de versiones

5.2. Herramientas de integración continua

La integración continua (CI) es un proceso de desarrollo de software que permite al equipo de desarrollo realizar *builds* consistentes, añadiendo test que permitan comprobar que las funcionalidades desarrolladas funcionen como se espera. Los tres pasos esenciales de la integración continua son: Construir, testear y mezclar.

Después de una investigación, de diferentes artículos *online*, en el cambio de los *framework* de integración continua más populares actualmente se deben valorar diferentes soluciones posibles para la implementación de la integración continua. [Kum23][Tay23][Rod23]

Se consideran las siguientes herramientas como herramientas de CI:

- a) GitHub Actions
- b) CircleCI

Las mencionadas anteriormente cumplen los dos requisitos, CircleCI cuenta con un plan regido por créditos de ejecución en el primer caso, mientras que GitHub Actions es completamente gratuito. Se utilizará CircleCI por salir de la zona de confort y aprender una nueva herramienta de CI, y adicionalmente, GitHub Actions de ser necesario su uso como soporte, en caso de agotar los créditos mensuales de CircleCI.

5.3. Lenguaje del proyecto

Para el desarrollo de la solución, de cualquier tipo, es necesario definir un lenguaje de programación.

Python es un lenguaje interpretado, es decir, es un interprete quien traduce línea a línea en tiempo de ejecución el programa a lenguaje máquina. Es un lenguaje flexible, con una sintaxis legible, siendo muy parecida al inglés. Además, cuenta con gran cantidad de módulos indexados y de fácil acceso que le dan esa flexibilidad mencionada anteriormente.

Se elige este lenguaje por ser con el que más experienciauento para realizar cualquier tipo de implementación que necesite el proyecto de manera eficaz.

5.4. Gestor de dependencias

En el proyecto es necesario utilizar un gestor de dependencias que resuelva la problemática de la gestión de módulos y bibliotecas externas

del proyecto, para facilitar la colaboración en el proyecto y brindar una reproducibilidad en diferentes sistemas y momentos.

Como opciones se considera Poetry, ya que el lenguaje en el que se implementará el proyecto es Python, y son sencillos de utilizar. Las mejores prácticas en la gestión de dependencias de Python es el uso de pip. Esta herramienta instala las dependencias de manera global, por ello se habla también de aislar los entornos de Python para separar las dependencias de un proyecto de otro, teniendo que añadir herramientas extra.

Se escoge Poetry por ser un gestor de dependencias sencillo que permite gestionar las dependencias de manera sencilla dentro de un entorno virtual por proyecto, para asegurar la reproducibilidad.

5.5. Gestor de tareas

Se considera Poetry e Invoke como gestor de tareas por su simplicidad y completitud. Poetry es mucho más simple y está enfocado a la gestión de dependencias, aunque pueden ejecutar tareas con la librería “subprocess” de Python. Sigue siendo un gestor de dependencias.

En conclusión, se elige Invoke como gestor de dependencias por su simplicidad y potencial. Se ajusta mejor a las necesidades del proyecto para lanzar test, siendo también muy intuitivo implementar las tareas, ya que se incluyen en un fichero “tasks.py” escrito en Python.

5.6. Fuente de recetas

Para ser capaces de resolver la necesidad del usuario de obtener recetas con unos ingredientes limitados, es necesario tener un conjunto de recetas. Para ello, como se comentó en el capítulo anterior.

Después de una investigación *on-line*, no se ha encontrado ninguna fuente válida de recetas. Existen algunas API en inglés que permiten filtrar por ingredientes, pero no existe ninguna que lo ofrezca gratuitamente, ni que tenga un plan gratuito de prueba. No se ha encontrado ningún conjunto de recetas que puedan ser insertadas en una base de datos.

Para formar una base de datos lo más realista posible, se plantean dos soluciones para recoger recetas de una página web real:

- Utilizar un *crawler*
- Crear un robot que recoja las recetas y las organice

Un *crawler* o rastreador, es un programa que analiza los documentos HTML de sitios web. Su objetivo principal es crear una base de datos, se utilizan sobretodo en motores de búsqueda, con el objetivo de extraer información que necesitan para evaluar sitios web y el posicionamiento en las búsquedas web.

Otra solución planteada es utilizar una automatización para extraer recetas de un sitio web, que sirva para obtener una base de datos inicial con la que trabajar.

La ventaja que determina utilizar una herramienta de “Robotic Process Automation” (RPA) es que la mayoría de sitios web añaden un fichero “robots.txt” que impiden el acceso a *crawlers* a la página, aunque es posible esquivar esta medida de protección de manera no muy lícita. Mientras que un robot hace uso de la interfaz gráfica de un navegador común, pudiendo completar la tarea sin ningún inconveniente.

Para automatizar el proceso de búsqueda de recetas, se utilizará la herramienta BluePrism, aunque es una herramienta de pago, cuenta con una licencia para su uso. Existen otras opciones como UIPath, robocorp o python RPA. Pero BluePrism es la herramienta con la que más experiencia tengo y contar con una licencia es un aliciente para la elección de esta herramienta.

5.7. Base de datos

Para contener las recetas y servirlas al usuario a la hora de responder a las peticiones que hagan en la aplicación, es necesario servir las recetas en una base de datos. Se elige una base de datos de tipo relacional, para aprovechar la relación que existe entre una receta y una base de datos, además de valerse de la condición necesaria de unicidad a la hora de guardar ingredientes para que no se repitan. Esta condición no existe de manera natural en una base de datos no relacional. Se consideran MySQL y PostgreSQL como bases de datos, siendo ambas gratuitas y relacionales.

Para contener las recetas y servirlas al usuario a la hora de responder a las peticiones que hagan en la aplicación, es necesario servir las recetas en una base de datos. Se elige una base de datos de tipo relacional, para aprovechar la relación que existe entre una receta y una base de datos, además de valerse de la condición necesaria de unicidad a la hora de guardar ingredientes para que no se repitan. Esta condición no existe de manera natural en una base de datos no relacional. Se consideran MySQL, PostgreSQL y SQLite como bases de datos, siendo todas las opciones gratuitas y relacionales.

Se escoge SQLite sobre PostgreSQL y MySQL por ser más ligero, estar incluido en Python y no depender de un contenedor docker para gestionar la base de datos, además del coste que supone en los recursos del sistema, teniendo toda la información contenida en un único fichero. Si bien es cierto, es posible contar con un servidor SQLite Server de bases de datos local. En un posterior despliegue en la nube, es posible migrar la base de datos a una instancia que ejecute este motor, para permitir mayor concurrencia, fiabilidad. [ost14]

5.8. *Test framework*

Para asegurar que las características de la aplicación siguen funcionando, es necesario incluir un *framework* para ejecutar los test. Se busca crear test de integración, basados en comprobar diferentes componentes de la aplicación, por ahora solo se comprueba la funcionalidad de encontrar recetas, pero a medida que avance el desarrollo de la aplicación y se implementen más funcionalidades, será necesario contar con sus respectivos test.

Se elige pytest como *test runner*, entre otros como unittest, nose o nose2. Su configuración no tiene complicación, siendo perfecto para este proyecto, y su uso está muy extendido para realizar las pruebas en el código desarrollado no solo en pequeños proyectos, sino también en proyectos a gran escala. Se han añadido algunos test para asegurar la funcionalidad del buscador de recetas. Además esta herramienta contiene un plugin para realizar test en las aplicaciones de Django.

5.9. *Framework web*

Para que el usuario medio (HU01 y HU02) pueda encontrar recetas por medio de uno de sus dispositivos, ya sea móvil o computadora. Es necesario componer la solución y envolverla con una interfaz de usuario agradable a la vista, que le permita encontrar recetas. Antes de añadir esa capa de interfaz, es necesario una herramienta que conecte la capa de interfaz con la funcionalidad de encontrar recetas.

Un *framework* es una estructura software compuesta de componentes personalizados, existiendo una infinidad. No solo relacionadas con el desarrollo web, sino que también hay *framework* orientados a desarrollo de aplicaciones médicas, desarrollo de videojuegos o para cualquier contexto que se nos ocurra. Dicho en otras palabras, se puede definir como una aplicación incompleta a la que tenemos que añadirle las últimas pinceladas para completar el desarrollo.

Se consideran Django y Flask como “*framework*” web para Python.

Flask es un *microframework* que permite el desarrollo de aplicaciones web minimalistas, de manera sencilla. Entre sus ventajas se encuentran:

- a) Ligeró
- b) Sistema de test unitarios
- c) Extensión por medio de *plugins*
- d) Compatibilidad con *Web Server Gateway Interface* (WSGI)

Django por su parte es un *framework* web completo que contiene muchas funcionalidades en su núcleo, sin tener que hacer uso de extensiones como Flask. Entre sus ventajas principales se encuentran:

- a) Seguridad robusta
- b) Se adapta a proyectos con un gran volumen de carga
- c) *SEO-Friendly*
- d) Gran variedad de extensiones
- e) Compatibilidad con *Web Server Gateway Interface* (WSGI)

En la experiencia, Flask ha sido más difícil de configurar ya que se encontró menos documentación para los fallos surgidos. No tiene soporte para aplicaciones multi-página, carece de una capa de seguridad incluida y no permite multiplataforma de manera nativa.

Sobre el papel, Django es más configurable, requiere de un poco más de configuración pero merece la pena por la gran cantidad de configuraciones posibles que permiten adaptarse completamente a cualquier aplicación web. Además, aporta características de seguridad como CSRF, Structured Query Language y XSS, parcheando cualquier falla identificada por un equipo comprometido. [22][Bah23]

Se elige Django como *framework* web por la facilidad del desarrollo de la aplicación en pocas líneas, siendo más rápida en la creación de un producto mínimo viable que Flask. Django permite implementar un *gl frontend* de manera más rápida para recibir *feedback* del usuario, además de poder implementar de manera sencilla cualquier funcionalidad futura.

5.10. Herramienta desacoplar la Base de datos

Para desacoplar la base de datos, se utiliza una herramienta *Object-Relational Mappers* (ORM). Que permite acceder a una base de datos por medio de modelos.

ORM tiene tanto ventajas como desventajas, es complicado relacionar los datos de una tabla con un objeto. En el apartado relacional, los elementos se representan en tablas con tuplas y los atributos de un objeto se referencian formando un grafo.

En primer lugar, se utilizó SQLAlchemy por ser una herramienta con un uso sencillo, aunque con una curva de aprendizaje elevada, para el proyecto no se necesitan consultas demasiado complejas. SQLAlchemy es compatible con multitud de conectores de bases de datos, encontrándose SQLite entre ellos y su compatibilidad con *frameworks* web de Python [Mak]

Después de la elección de Django como *framework* web, para no tener redundancia en las herramientas, se ha migrado el uso de SQLAlchemy al uso de modelos en django.

5.11. Web Server Gateway Interface

Antes de lanzarse a elegir un *Web Server Gateway Interface* (WSGI), considero necesario que se explique que es y para qué sirve. Para establecer una base sobre porqué se necesita este módulo en la aplicación. Si bien es cierto que, en teoría, sería posible subir el proyecto a una instancia en la nube y exponer el puerto HTTP creando un servicio poco seguro.

Un servidor web no sabe ejecutar aplicaciones en python, por eso Grisha Trubetskoy creó un modulo para Apache conocido como *mod_python*. Era simple y llanamente una implementación que permitía ejecutar código de python en un servidor. Poco después, fruto de la aparición de vulnerabilidades la comunidad de python comenzó a implementar WSGI, que realmente se trata de un estándar escrito en <https://peps.python.org/pep-3333/PEP 3333>. Al ser un estándar, permite definir de manera organizada por medio de reglas, que se requiere de la aplicación.

El WSGI que más se utiliza en la comunidad de Python es Gunicorn ya que es muy ligero, sencillo de implementar, escalable, y se ajusta perfectamente a las necesidades de cualquier proyecto. uWSGI es muy popular también pero ofrece demasiadas funcionalidades que se solapan entre ellas. Por las ventajas mencionadas de Gunicorn, se escoge este WSGI.

5.12. Servidor Web

Para exponer la aplicación a internet de forma segura, es necesario utilizar un servidor web que permita desplegar la aplicación para re-

coger peticiones del usuario. Aunque se pueda desplegar la aplicación, tal y como está, para añadir escalabilidad y seguridad, se utilizará un servidor web. Existen otros servidores web muy populares como Apache o LiteSpeed, se escoge Nginx como servidor web por su flexibilidad, facilidad de configuración y modularidad, permitiendo extender la funcionalidad hasta el punto deseado, cubriendo las necesidades de la aplicación a medida.

Capítulo 6

Recetas

En este capítulo se detallará todo lo relativo al *milestone* “Encontrar recetas”, que tiene como objetivo solucionar al usuario la necesidad de encontrar recetas utilizando exclusivamente los ingredientes seleccionados por el usuario.

6.1. Fuente de recetas

Para la obtención de recetas, se obtendrán de la famosa página de recetas Cookpad. Donde la comunidad puede subir cualquier receta. Habiendo mucha variedad de recetas, permitiendo obtener una base de datos variada. Estas recetas se deberán cargar en una base de datos.

El robot de BluePrism, utiliza un reconocimiento del código fuente, similar a un *crawler*, de la página permitiendo seleccionar los elementos deseados de cada una. Inspeccionando la página se pueden analizar una serie de atributos para identificar inequívocamente algunos elementos.

Los pasos que se han seguido para obtener recetas son:

- a) Buscar diferentes recetas de diferentes tipos (pescado, carne, verduras, etc...)
- b) Obtener los enlaces de dichas recetas
- c) Filtrar los enlaces para evitar los repetidos
- d) Acceder a cada receta y recuperar
 - 1) Nombre
 - 2) Ingredientes
 - 3) Pasos
- e) Formar un fichero en formato JSON (JavaScript Object Notation)

Para la función de buscar recetas de diferentes tipos, se lanza un navegador web con la dirección de inicio de Cookpad, al buscar cualquier cosa utilizando el cuadro de búsqueda. La dirección a la que se consulta la petición es la siguiente: `buscar/verduras?event=search.historyorder=recent`, con un simple vistazo se puede identificar que la petición en este caso fue “verduras”. Sabiendo esto, para buscar por tipo de receta se han compuesto las direcciones url de manera dinámica, evitando pasos extra a la hora de realizar esta búsqueda inicial.

En el paso de obtención de la receta, se identifica el nombre de la receta. Utilizando la herramienta de modelado de aplicación que trae incorporada BluePrism. Con ella se puede seleccionar el elemento de la página que se desea extraer y automáticamente se extrae información relevante para la identificación como el identificador del elemento, las clases CSS del mismo, su ruta XPATH, etc... El nombre de las recetas como elemento no es único, al ser una lista de recetas habrá muchas coincidencias que se quieran extraer. Para ello se utiliza el atributo `match;index, si se selecciona el primer resultado de la página se obtendrá el primer nombre.Blue`

Una vez obtenidos todos los enlaces, puede que haya repetidos por coincidir en la categoría de carne y verduras a la vez, es necesario hacer un filtrado muy sencillo que elimine los enlaces repetidos.

La función que extrae la receta de la página, obtiene el nombre teniendo en cuenta que es el primer `h1` de la página. Los ingredientes y los pasos se tratan de listas ordenadas, fáciles de extraer. Cada ingrediente tienen un identificador `ingredient` seguido de una cadena de números. BluePrism puede utilizar `wildcards` para identificar elementos de la página web, de la misma manera que al extraer una lista de enlaces, se itera por los ingredientes hasta que no haya más. Con los pasos se sigue el mismo razonamiento, valiéndose de un selector CSS para extraer los pasos iterando sobre ellos hasta que no hay más.

Para que constituya una fuente de información uniforme, la colección de recetas se guarda como un fichero JSON que puede ser fácilmente recuperado y transformado en cualquier otro formato.

El proceso de ejecución del robot lleva tiempo, ya que el identificar los ingredientes a través de una expresión regular toma entre diez y veinte segundos. Los demás pasos son relativamente menos costosos en cuestión de tiempo. El robot tardó aproximadamente cuatro horas en obtener sesenta y dos recetas con todos los elementos mencionados. La ventaja de haber utilizado este proceso de obtención de recetas es que se puede ampliar en el futuro a conveniencia, aunque hay que tener en cuenta el tiempo que tarda el robot en obtener recetas.

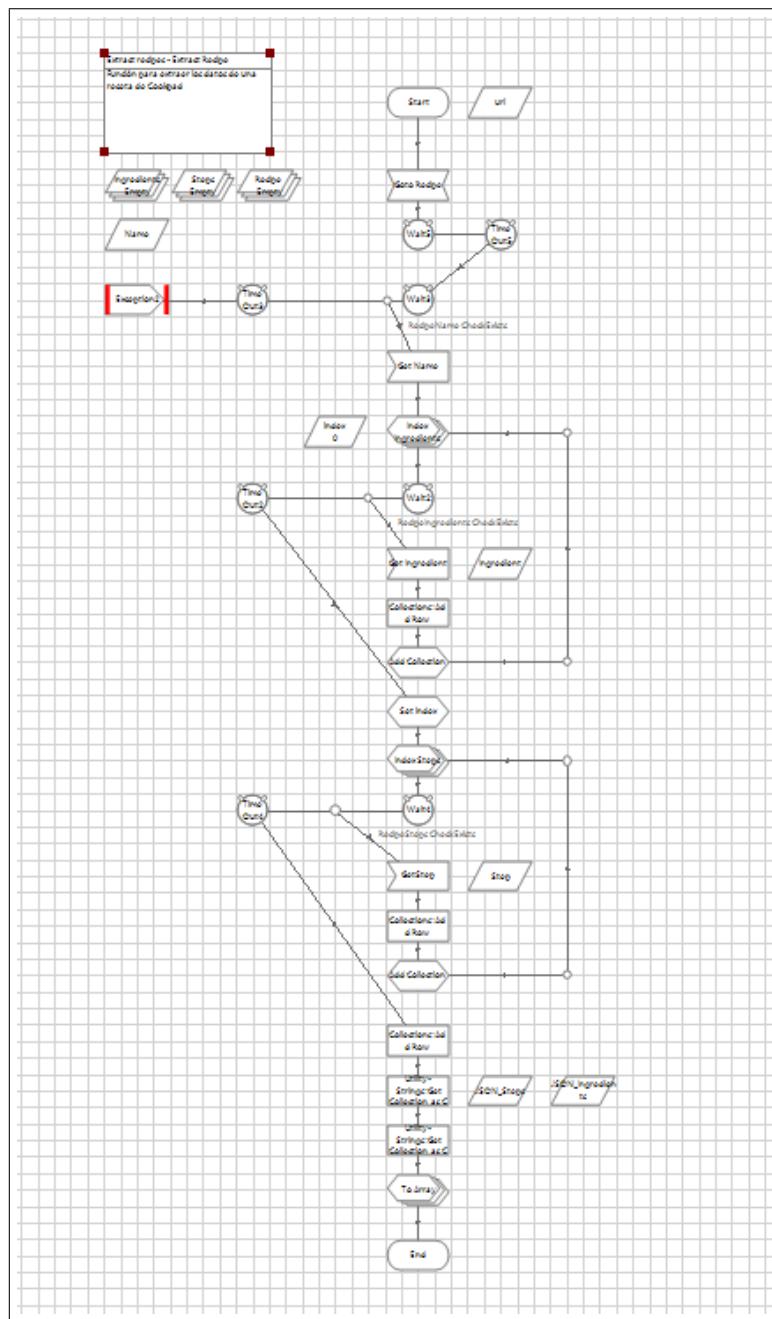


Figura 6.1: Función para extraer enlaces

6.2. Gestión de las recetas en la aplicación

Para insertar los datos en la aplicación, se insertan en una base de datos relacional para aprovechar la relación que existe entre una receta y

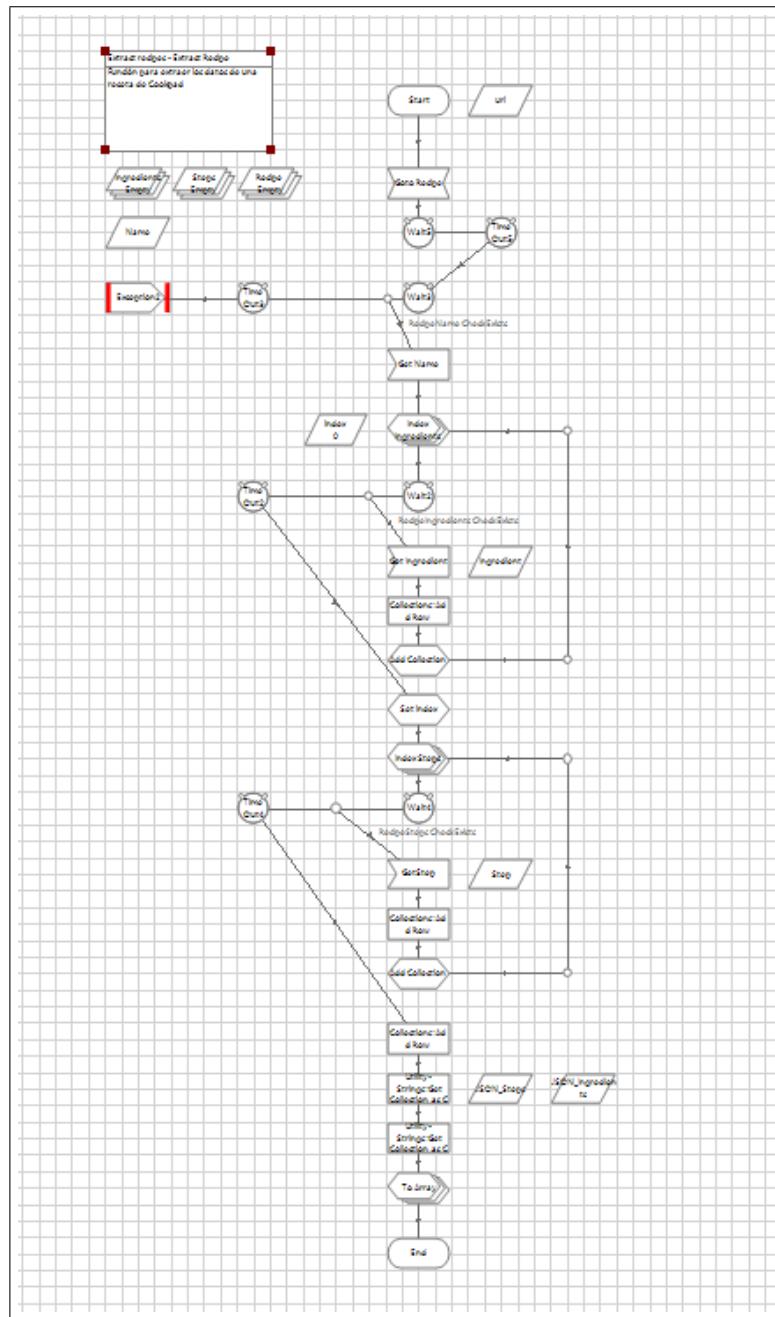


Figura 6.2: Función para extraer recetas

los ingredientes que contiene. Permitiendo identificar rápidamente que ingredientes lleva una receta, pudiendo gestionar de manera sencilla los requisitos para realizar una receta.

Se utiliza un pequeño script para insertar los datos desde sus respec-

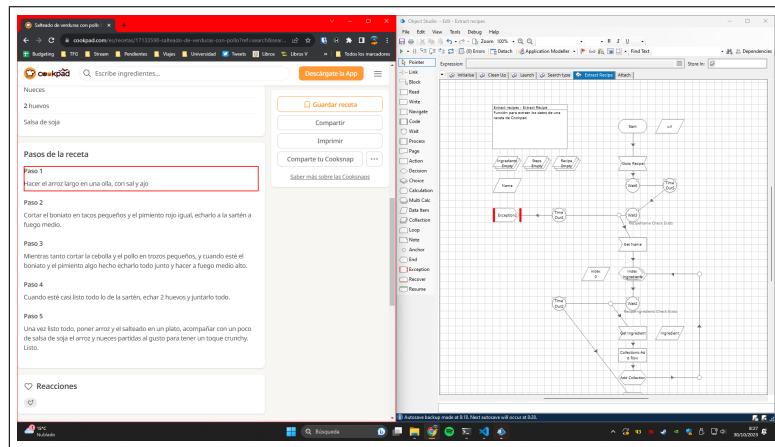


Figura 6.3: Identificar los pasos de la receta

tivos ficheros a la base de datos siguiendo el siguiente esquema

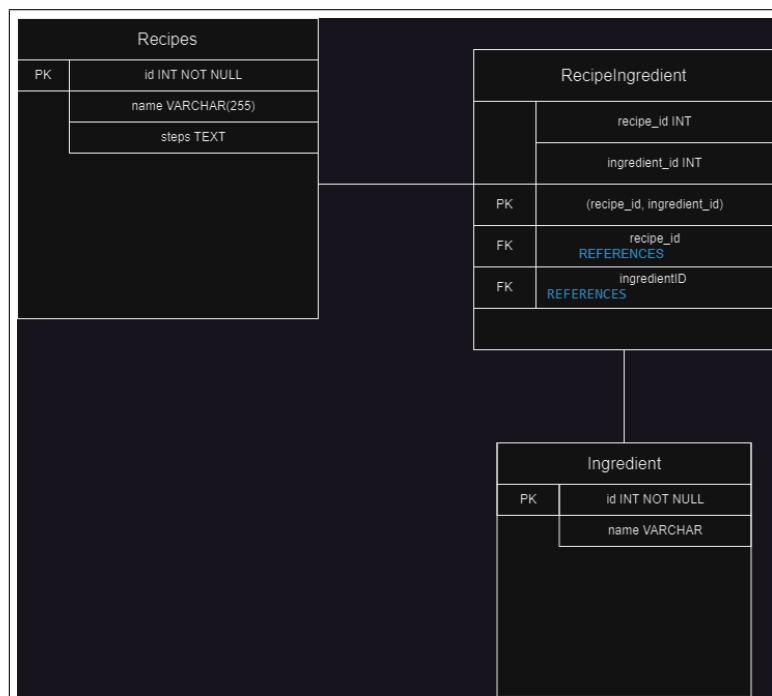


Figura 6.4: Esquema de la Base de datos

6.3. Encontrar recetas

El primer problema que se pretende resolver con la publicación del proyecto es permitir a los usuarios encontrar recetas, ya sea por aprovechar los ingredientes de su despensa o porque no recuerdan con exactitud los ingredientes de una receta o sus pasos.

6.3.1. Consideraciones iniciales

Antes de explicar como se ha resuelto el problema debemos hacer unas consideraciones iniciales que se han tomado en cuenta en la etapa de modelización del problema.

Por ejemplo, existen ciertos tipos de ingredientes que todo el mundo tiene en casa, por ello no se consideran como restricciones a la hora de encontrar una receta. Estos son

- a) Agua
- b) Sal
- c) Pimienta
- d) Aceite
- e) Vinagre

Otra consideración importante es que no se tiene en cuenta la cantidad de cada ingrediente que tiene el usuario en casa. Por ejemplo, puede introducir en el buscador que cuenta con macarrones y tomate frito, obtendrá la receta de “Macarrones con tomate”. Pero a la hora de cocinar, puede que no tenga pasta suficiente como para una ración completa.

6.3.2. Algoritmo

La funcionalidad se puede implementar de muchas maneras. Quizá la más sencilla sea obtener todas las recetas de la base de datos e ir comprobando una por una las que pueda hacer el usuario con los ingredientes seleccionados. Pero el orden de la función sería $O(f(n)) = n*m$ siendo n el número de recetas de la base de datos y m el número de ingredientes que contiene cada una. En este proyecto, al ser una modelización de una solución al problema no se utilizan unos datos demasiado extensos, pero si se extiende el catálogo de recetas la eficiencia se verá afectada gravemente.

Debido a la poca eficiencia en un *dataset* muy grande, es necesario utilizar algunos ajustes previos para reducir lo máximo posible el número

de recetas inicial. El primer filtro que se utiliza es el razonamiento de que una receta no será candidata cuando tenga más ingredientes que los seleccionados por el usuario, pues significará que existe un ingrediente que falta.

Una vez que se ha reducido la búsqueda a un subconjunto de recetas que cuentan con un número de ingredientes menor o igual a los introducidos por el usuario. Existen recetas del subconjunto que no tienen ningún ingrediente de los que se tienen en la despensa, analizar si es una receta candidata será un gasto de tiempo. Del anterior razonamiento nace el segundo filtro aplicado: Obtener solo recetas que contengan al menos un ingrediente de los que ha seleccionado el usuario.

Al aplicar los dos filtros conjuntamente, se está obteniendo un subconjunto mucho menor y más manejable, solo se necesitaría comprobar que el usuario tenga en su haber los demás ingredientes de la receta.

6.3.3. Implementación

De manera técnica se ha implementado toda la funcionalidad en una misma clase denominada *DBConnector*, que contiene todos los métodos para buscar en la base de datos utilizada.

La mayoría de métodos son accesos a objetos triviales, con el objetivo de recuperar un objeto por su identificador o por nombre. Se puede ver la información detallada de esta clase en el apéndice de la memoria.

Los dos métodos más relevantes de esta clase son las relativas a buscar recetas: *buscarRecetasNombre*, *buscarRecetasID* y *buscarRecetasIngredientes*.

Al buscar recetas se obtienen también los ingredientes que contienen, con el objetivo de que el usuario pueda comprobar que contiene la receta, o qué ingredientes necesitará antes de comprobar los detalles de la misma.

El método para buscar recetas por su nombre, *buscarRecetasNombre*, hace uso de la sentencia Structured Query Language que selecciona las recetas en base a su nombre, utilizando la función *LIKE*, ignorando las mayúsculas a la hora de buscar la tupla en la Base de datos, devolviendo el identificador y el nombre de las recetas que cumplan la condición. Para cada receta, se incluyen sus ingredientes, de manera que el usuario tenga en cuenta la posibilidad de que le falte algún ingrediente antes de comprobar los detalles de la receta, iterando los ingredientes que contiene cada receta e introduciéndolos en una lista que será añadida a la tupla devuelta.

Por otra parte, *buscarRecetasID*, usa un algoritmo similar a la función anterior. Se recupera la receta de la Base de datos así como sus

ingredientes de manera similar, posteriormente se añade la lista de ingredientes a la tupla que será devuelta como resultado de la operación.

El método *buscarRecetasIngredientes*, utiliza el algoritmo descrito en la sección anterior. Con el método *buscarRecetasIngredientesNumber*, se seleccionan con una sentencia Structured Query Language todos las las recetas que tengan un número menor de ingredientes que la selección del usuario y tengan al menos un ingrediente de la lista proporcionada.

Una vez reducido el número de recetas se comprueba si cada ingrediente de la receta está en la lista de ingredientes del usuario. En caso de que no esté, no se debería seguir comprobando esa receta así que se sale del bucle para no perder tiempo. Si la receta cumple, se le añade la lista de ingredientes que se utilizan y a su vez este conjunto se añade al resultado a devolver.

```
def buscarRecetasIngredientes(self, id: int): try: receta,ingredientes = RecipeIngredient.objects.filter(recipe_id = id).values('ingredient_id', 'ingredient_name') F('ingredient_id_name'))resultado = []for receta in receta,ingredientes: resultado.append((receta['ingredient_id'], r
```

Clase conector de la Base de datos

Capítulo 7

Prototipo

Una vez creado el núcleo de la aplicación, para llevarlo al usuario final se debe encapsular en dos capas: Interfaz y framework. La primera capa es totalmente visual, lo que el usuario va a ver y con lo que va a interactuar, por eso es tan importante contar con una interfaz sencilla e intuitiva que permita a cualquier persona entender como utilizar la aplicación. La segunda capa se puede definir como el pegamento que une la interfaz con la funcionalidad para encontrar recetas, gestiona las llamadas al núcleo y da formato a la interfaz que ve el usuario.

7.1. Interfaz

En el apartado de *Design Thinking* se detallaba la interfaz de baja fidelidad para el prototipo, esta se ha llevado a cabo como una primera versión de interfaz para conocer las opiniones de los usuarios y poder mejorarlade cara al futuro.

7.1.1. Diseño

Siguiendo los mockup presentados de la aplicación web, se intenta realizar en mayor medida una interfaz sencilla que una persona mayor pueda entender, si el diseño está muy recargado es muy probable que se confunda. En la versión final de la aplicación se puede añadir un banner que actúe como footer con anuncios, que no entorpezcan la visualización o la experiencia del usuario.

En la figura superior se presenta la interfaz con un diseño para escritorio. Se utiliza una barra superior a modo de navegador de la página web, con un botón para volver a la página principal y un buscador que permite buscar recetas por nombre.

En cuanto al diseño central de la aplicación, se puede observar una clara división en dos zonas. El buscador de recetas por ingredientes y una columna

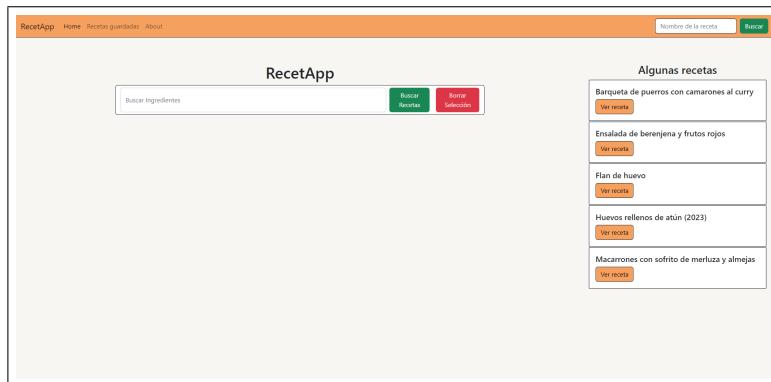


Figura 7.1: Diseño para escritorio

a la derecha con algunas recetas aleatorias para inspirar al usuario, estas irán cambiando mientras el usuario utiliza la aplicación. Los resultados del buscador aparecerán bajo la barra de búsqueda.

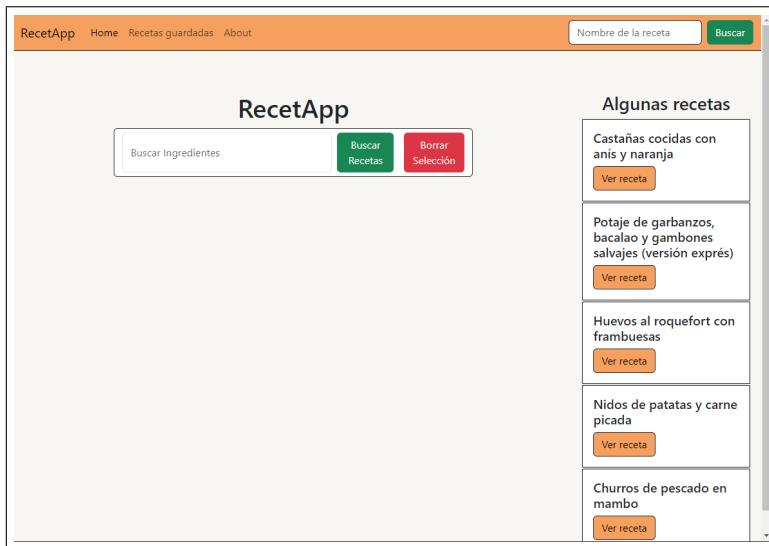


Figura 7.2: Diseño para tableta horizontal

La figura superior muestra un diseño para una tableta en modo horizontal, siendo muy parecida al diseño de escritorio, excepto el navegador que colapsaría mostrando el botón para desplegarlo y la desaparición del cuadro que explicaría el uso del buscador.

Por último, se muestra el diseño para una tableta vertical más estrecha. Se puede observar que la zona central que antes estaba dividida en dos, se ha pasado a una sola columna con el objetivo de que los resultados sean lo más legibles posible. Este mismo diseño es el utilizado para dispositivos móviles.

7.1.2. Implementación

Para la parte de implementación de la interfaz se ha utilizado Boostrap, un *framework* de estilos orientado a la facilitar una mejora visual al código HTML sin tener que entrar a CSS (Cascade Style Sheet). Existen otros frameworks de estilo, pero Boostrap es muy completo, no solo incluyendo estilos CSS sino también algunas funcionalidades en JavaScript, para mejorar la interfaz, haciéndola dinámica.

Para instalar el framework, existen dos maneras diferentes:

- Descargar el código fuente
- Utilizar un Content Delivery Network (CDN)

Para incluirlo en el proyecto se debe añadir el “link” con la referencia al fichero descargado o el enlace al CDN. Por otra parte Boostrap usa elementos dinámicos que necesitan de unas funciones escritas en JavaScript incluidas en el mismo fichero comprimido descargable, que será necesario incluir dentro de un “script”, una vez añadidos los ficheros necesarios, el estilo se puede

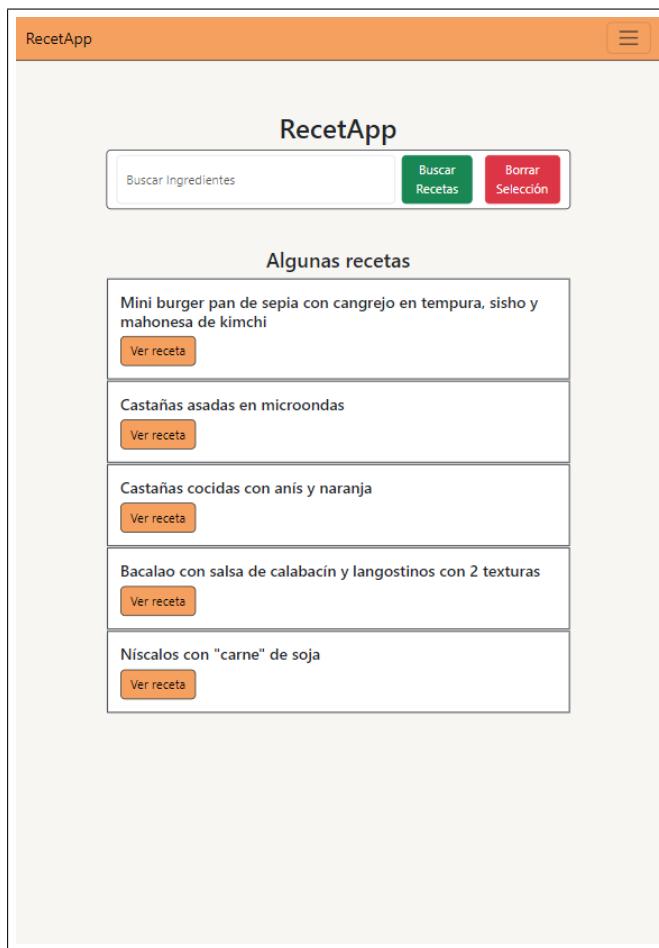
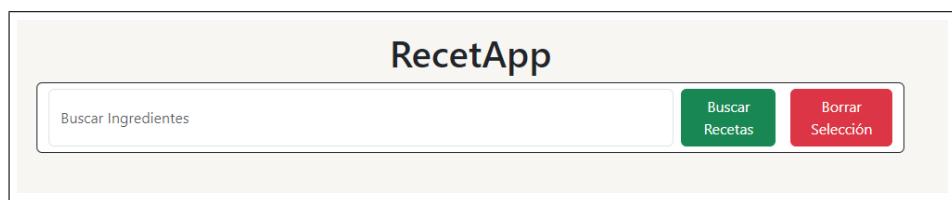


Figura 7.3: Diseño para tableta vertical

aplicar añadiendo clases a los elementos HTML, como se haría con un CSS normal.

El buscador principal contiene una tabla en principio oculta, esperando a ser utilizado. Cuando se escribe una letra, se empieza a buscar el resultado de la cadena en la base de datos, y los resultados se escriben dinámicamente en una tabla para que el usuario pueda seleccionar el ingrediente que desea usar. Esta función se realiza con JQuery, usando un evento *input* en el campo de búsqueda. Cada vez que se desencadena el evento se genera, con AJAX, una petición a un endpoint de la aplicación web, que devuelve una lista de ingredientes coincidentes.



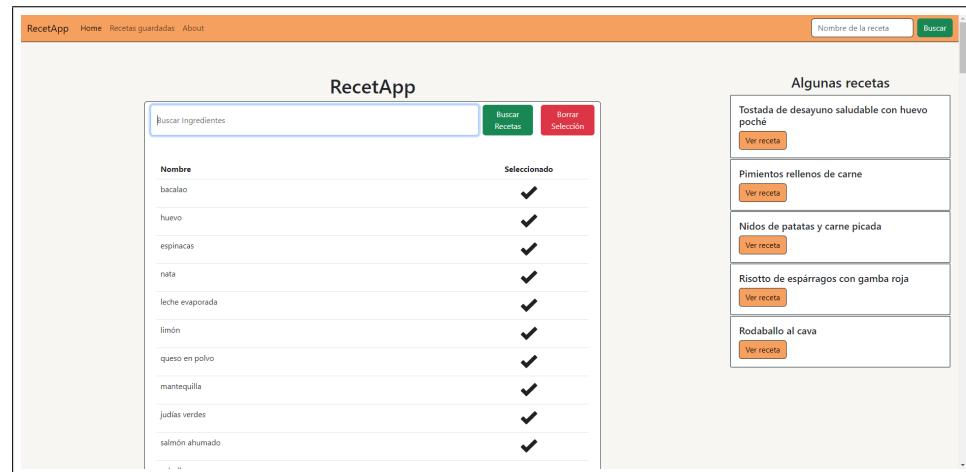
The screenshot shows a user interface for a search application named "RecetApp". At the top center is the title "RecetApp". Below it is a horizontal input field containing the placeholder "Buscar Ingredientes". To the right of the input field are two buttons: a green button labeled "Buscar Recetas" and a red button labeled "Borrar Selección".

```
<div class="d-inline-flex w-100">
    <input id="buscador-ingredientes" class="form-control me-2"
        type="search" placeholder="Buscar Ingredientes" aria-label="Search">
    <button id="enviarIngredientes" class="btn btn-success" type="submit">Buscar Recetas</button>
    <div id="eliminarSeleccion" class="btn btn-danger ms-3">Borrar Selección</div>
```

Figura 7.4: Diseño del buscador

La función que recupera los ingredientes de la base de datos se ha codificado en JavaScript y es llamada cada vez que el endpoint devuelve una lista de ingredientes encontrados, es la función que da forma a la tabla de manera dinámica. *buscarIngrediente* obtiene como parámetro la respuesta con la lista de ingredientes. En primer lugar, se crea la cookie que contendrá la lista de ingredientes seleccionados. Y se elimina el atributo *hidden*, para que se visualicen los resultados. Posteriormente por cada ingrediente de la lista se crea una nueva fila a la que se le añaden los atributos del ingrediente recuperado, y un disparador de eventos que al pulsar la fila añade el identificador del ingrediente a la lista almacenada en la cookie mencionada. Si se selecciona el elemento, se muestra un *tick* que permite al usuario conocer si cuenta con ese ingrediente en su lista, al volver a seleccionar el ingrediente se elimina de la “despensa” del usuario y se vuelve a esconder el *tick*.

Por simplicidad se ha añadido solo el código HTML relativo a una sola entrada de ingrediente. En este, como se mencionaba anteriormente, se puede observar como cada fila de la tabla cuenta con un atributo *data-id*, que contiene el identificador del ingrediente. Cuando es seleccionado, se llama a



```
<table id="tabla" class="table mt-5 container ingredientes">
  <thead>
    <tr>
      <th>Nombre</th>
      <th class="text-center">Seleccionado</th>
    </tr>
  </thead>
  <tbody id="tabla-ingredientes" class="">
    <tr data-id="6" onclick="SeleccionarIngrediente(this)">
      <td>canela</td>
      <td>especias</td>
      <td class="text-center">
        <svg hidden=""></svg>
      </td>
    </tr>
  </tbody>
</table>
```

Figura 7.5: Diseño del buscador con resultados

la función *SeleccionarIngrediente* con la fila seleccionada como argumento. Cabe destacar que no es el único método de añadir un disparador de eventos a un elemento.

```
var resultados = response.resultados;
tabla.empty();
for (var i = 0; i < resultados.length; i++) {
    var fila = $('<tr data-id="'+resultados[i].id+'"
                onclick="SeleccionarIngrediente(this)">');
    fila.append($('<td>').text(resultados[i].nombre));
    var td = $('<td class="text-center">');
    var svg = '<svg>';
    if(isSeleccionado(resultados[i].id) == false){
        svg += " hidden"
    }
    svg += '<codigo del svg>';
    td.append(svg);
    fila.append(td);
    tabla.append(fila);
}
```

Figura 7.6: Añadir ingredientes a la interfaz del buscador

SeleccionarIngrediente selecciona el elemento SVG (Scalable Vector Graphics) de la fila con el ingrediente seleccionado, y lo muestra. Además, obtiene la cookie donde se almacena la lista de elementos seleccionados por el usuario y añade el identificador. El principal problema es que JavaScript almacena las cookies como una sola cadena de texto con un separador, es decir, antes de poder usar la lista hay que obtener el trozo de cadena que pertenece a dicha cookie, haciendo una función que permita darle el formato correcto.

Los dos botones del buscador tienen la utilidad de eliminar ingredientes seleccionados y buscar recetas, como su propio nombre indica. El primero limpia la lista de ingredientes, eliminando toda la selección que ha realizado el usuario permitiéndole cambiar de parecer en los ingredientes sin tener que buscarlos uno a uno. El segundo botón manda los ingredientes al backend para hacer la búsqueda de recetas utilizando dichos ingredientes, la gestión se explicará en la siguiente sección.

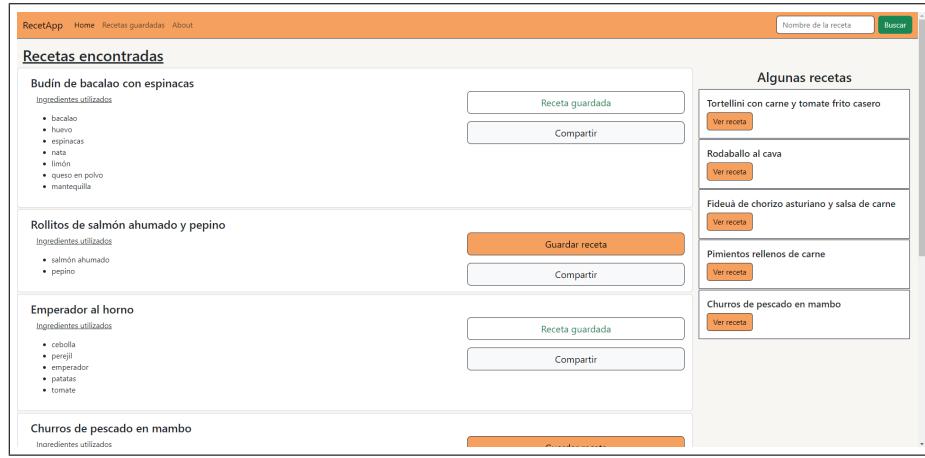


Figura 7.7: Resultados del filtrado por ingrediente



Figura 7.8: Resultados del filtrado por ingrediente en una tablet

Esta interfaz es muy clara, ya que solo muestra recetas que se pueden realizar con los ingredientes proporcionados. Se obtiene el identificador de la receta, los nombres y sus ingredientes, para luego formatearlos en una columna de tarjetas agradable al usuario. Al seleccionar una tarjeta concreta se recupera toda la información relativa a la receta y se muestra de manera organizada. Cabe destacar que en la página de la receta no se ha añadido la columna de recetas aleatorias para enfocar la atención en la propia tarjeta de la receta, además de añadir información nutricional estimada por ración.

El código HTML de la receta se genera dinámicamente obteniendo los datos del backend, que se explicará en la siguiente sección.

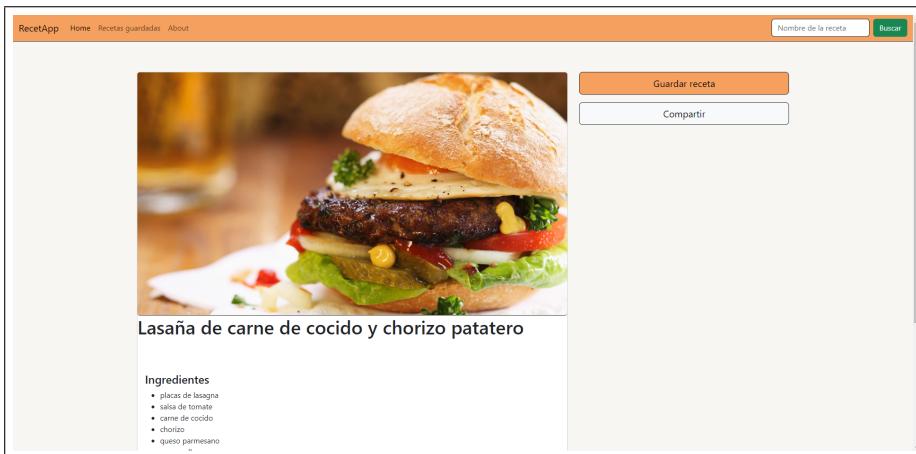


Figura 7.9: Tarjeta de la receta “Lasaña de carne”



Figura 7.10: Tarjeta de la receta “Budín de bacalao” en una tablet

7.2. Framework

Para pegar las dos capas desarrolladas hasta ahora: El núcleo para encontrar recetas y la capa de frontend, se utiliza un framework que actúa como intermediario estableciendo unos endpoint para el usuario y algunas funcionalidades ya vistas en la sección anterior.

Como ya se explicó en el apartado de implementación del proyecto, se ha elegido Django como framework para Python. Este permite crear las rutas de manera sencilla y ofrece un modelo Model-View-Controller (Modelo-Vista-Controlador), que gestiona tanto la base de datos como la generación del HTML que se ofrece al usuario. Además, permite crear plantillas HTML

```

<div class="row d-flex-inline bg-light rounded rounded-sm">
    <div class="col-lg-8">
        <div class="ms-5 mt-5 w-100">
            <h1>{{receta_nombre}}</h1>
            <div class="ms-2 ps-3 mt-5">
                <h4>Ingredientes</h4>
                <ul>
                    {% for item in receta_ingredientes %}
                        <li>{{item}}</li>
                    {% endfor %}
                </ul>
                <h4 class="mt-5">Pasos</h4>
                <ol>
                    {% for item in receta_pasos %}
                        <li>{{item}}</li>
                    {% endfor %}
                </ol>
            </div>
        </div>
    </div>

```

Figura 7.11: Código de la tarjeta de la receta

dinámicas con los datos recuperados del modelo, y generar formularios que encajen con la generación de tuplas para la Base de datos.

La instalación es muy sencilla, pudiéndose insertar en el entorno de Poetry añadiendo las dependencias según lo explicado. Django usa por defecto una Base de datos SQLite 3 llamada “db.sqlite3” se puede cambiar el nombre mediante el parámetro “*DB NAME*” del fichero de configuración de entorno. Además es necesario

Iniciar una aplicación genera una plantilla de ficheros de donde Django obtendrá los datos para montar la lógica de la página web.

```

poetry run django-admin startproject <nombre del proyecto>
poetry run Python3 <carpeta del proyecto>/manage.py startapp <
    nombre de la aplicación>
invoke migrate --app=<nombre del proyecto>

```

Figura 7.12: Comandos para la instalación de Django

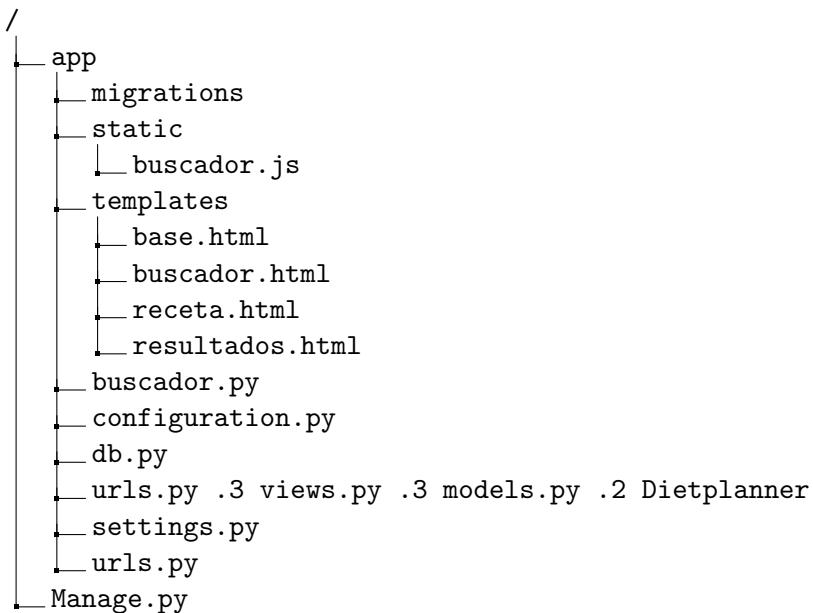


Figura 7.13: Estructura del proyecto

En la figura superior se muestra el árbol de directorios final que tiene el proyecto generado por Django. En la carpeta “app” es donde se contiene toda la información de la aplicación.

7.2.1. Endpoints

La definición de endpoint tiene muchos significados diferentes dependiendo del contexto, en un entorno de programación el significado se refiere a una URL de una API o backend que se encarga de responder a una petición.

Para el desarrollo del proyecto se han creado algunos endpoint que sirven como punto de acceso a los datos de recetas almacenado en la base de datos. En Django se definen en el archivo “app urls.py”, pero para que Django pueda encontrarlos debemos incluir este fichero en las URL a nivel de proyecto, desde “Dietplanner urls.py”.

```
urlpatterns = [
    path("", views.index, name="index"),
    path("buscar/", views.buscar, name="buscar_resultados"),
    path("resultados/", views.resultados, name="resultados"),
    path("receta/<int:id>", views.detalle_receta, name="detalle_receta")
]
```

Figura 7.14: Puntos de acceso de la aplicación

```
resultados = buscador.buscarIngredientesNombre(query)
for resultado in resultados:
    resultado_dict = {
        'id': resultado.id,
        'nombre': resultado.nombre,
    }
    resultados_lista.append(resultado_dict)
```

Figura 7.15: Endpoint buscar

Dentro del fichero “views.py” se implementa la lógica de cada endpoint. Se destacan los cuatro endpoints creados

- índice
- Buscar
- Resultados
- Detalle receta

Índice

Este punto de acceso se utiliza por defecto al acceder a la página web inicialmente, y devuelve una vista de la plantilla “buscador.html” con el buscador principal y una columna de recetas aleatorias.

Buscar

Anteriormente, en el apartado de interfaz, se explicó que el buscador de ingredientes principal, que se actualizaba la lista de ingredientes con cada letra que se introducía haciendo uso de un event listener, dicha funcionalidad utiliza este endpoint.

Al llegar una petición se comprueba el método, si viene de una solicitud “GET” se obtiene la cadena pasada como argumento. Posteriormente, por simplicidad, se utiliza un filtro en Django para obtener todos los ingredientes que existen en la Base de datos. Para cada uno de los ingredientes encontrados como resultado se crea una cadena de texto en formato JSON con

toda la información relativa de los ingredientes, y en el lado de la interfaz se gestionan los datos como se explicó anteriormente.

Resultados

Este endpoint se especifica para obtener los resultados de las recetas que puede hacer el usuario con los ingredientes que desea utilizar en la cocina. En la aplicación existen dos tipos de búsqueda de recetas

- Ingredientes
- Nombres

Dependiendo del tipo de búsqueda se añade un parámetro que lo especifica. En caso de buscar por ingredientes, se obtienen de la cookie guardada en el navegador, convertida a una lista de enteros que se pasa como argumento a la función *buscarRecetasIngredientes* del buscador, explicada en capítulos anteriores.

Esta devuelve la lista de recetas que se pasa como contexto al render de la plantilla HTML, para darle el formato final.

Detalle receta

Este endpoint recibe como argumento el identificador de la receta a la que se está intentando acceder.

Se obtiene la receta haciendo uso del método *buscarRecetasID* de la clase *Buscador*. Posteriormente, se les da el formato a los pasos de la receta para listarlos de manera sencilla, por último, se añade la información al contexto que se pasa al render.

Capítulo 8

Demostración

En este capítulo se mostrará una pequeña demostración del uso de la aplicación que haría cualquier usuario de la aplicación. Además de establecer un test para comprobar que la lógica de los algoritmos del buscador está correctamente implementada.

8.1. Pruebas unitarias

Para una correcta metodología, es necesario contar con una serie de test unitarios que permitan comprobar cada vez que se añade una nueva funcionalidad, la correcta integración de la misma. En el proyecto actual solo existe una funcionalidad, pero se deja la puerta abierta para el desarrollo de nuevas utilidades en el futuro, como añadir cualquier restricción alimentaria que pueda tener el usuario, cambiando los ingredientes problemáticos por unos permitidos para el usuario.

Para los test, se añaden los pertinentes que permiten realizar comprobar que lo que ve el usuario es lo correcto en cada momento, no solo comprobando si se devuelve un código de respuesta correcto (2xx) sino que realmente sea lo que el usuario tiene que estar viendo, haciendo los test a nivel de vista.

En el desarrollo de las pruebas se ha realizado con la herramienta pytest, incluida en Django. Se ha generado un test para cada vista de la aplicación.

1. Índex
2. Resultados de recetas
3. Detalles de recetas
4. Recetas guardadas

Para ejecutar los test se hace uso de la tarea especificada para ello en Invoke, el código de las pruebas se incluye en el anexo de la documentación.

```
invoke test
```

Figura 8.1: Comando para lanzar los test unitarios

8.2. Prueba de la aplicación

En esta sección se hará una prueba completa a la aplicación, desde el punto de vista del usuario, relatando el viaje por la aplicación de uno de los usuarios: Daniel Pérez.

Daniel encuentra un anuncio de la aplicación en internet y se anima a probarla ya que se le suelen caducar algunos ingredientes de la despensa. Al entrar en la aplicación, desde su ordenador personal, se encuentra directamente con el buscador principal. Echa un vistazo a la interfaz, pensando en las recetas aleatorias que aparecen a la derecha. Entrando en una porque parece que tiene buena pinta, lamentablemente no cuenta con los ingredientes para realizarla y vuelve al buscador principal.

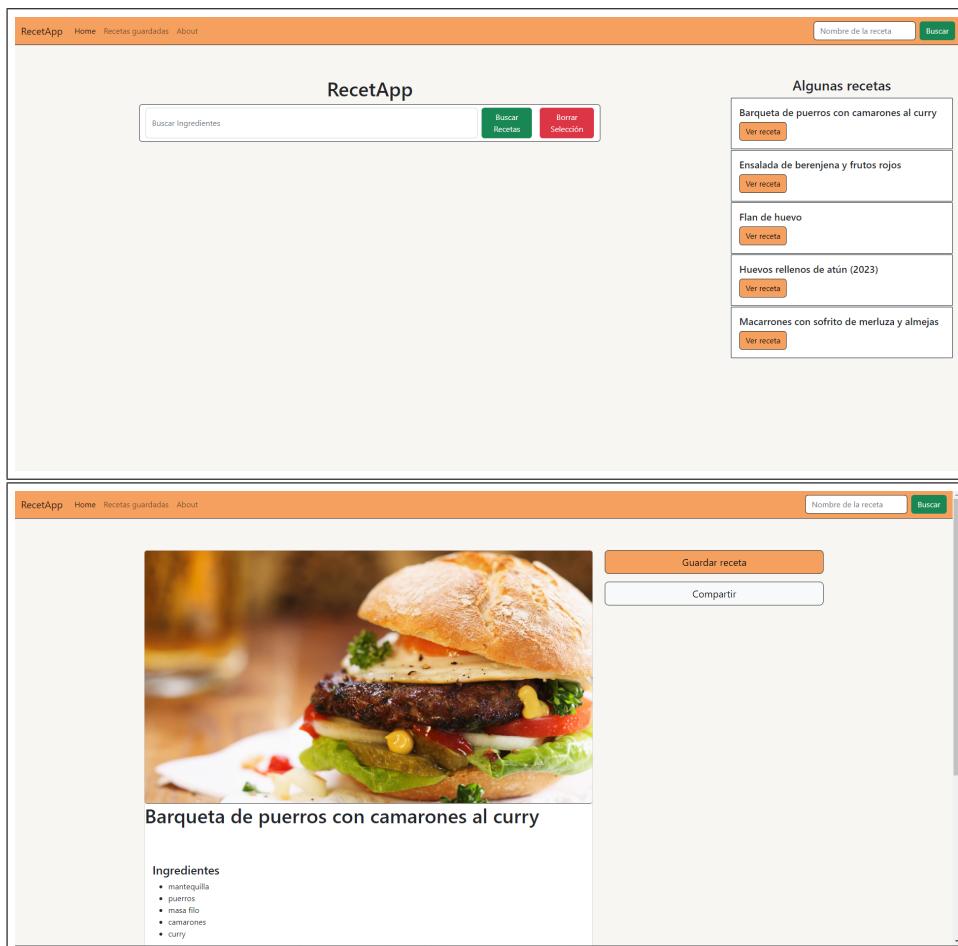


Figura 8.2: Primeros pasos de Daniel

Una vez que vuelve a la página principal, empieza a introducir los ingredientes que quiere gastar. Como últimamente quiere comer sano, le apetece cenar una pescado, gastando las espinacas que están a punto de caducar y añadiendo otros ingredientes que le sobran.

1. Bacalao
2. Huevos
3. Espinacas
4. Nata
5. Leche evaporada
6. Limón
7. Queso en polvo
8. mantequilla

El buscador debería encontrar las recetas que usan estos ingredientes, mostrándolas al usuario de manera organizada y clara.

Daniel pulsa el botón para buscar ingredientes con el botón verde habilitado para ello. Los resultados que ha obtenido son los esperados, ha obtenido todas las recetas que se esperaba que devolviera la aplicación.

Daniel mira los resultados devueltos y elige comer un budín de bacalao, seleccionando dicha receta desde el buscador, leyendo la receta y reuniendo los ingredientes para realizar los pasos detallados.

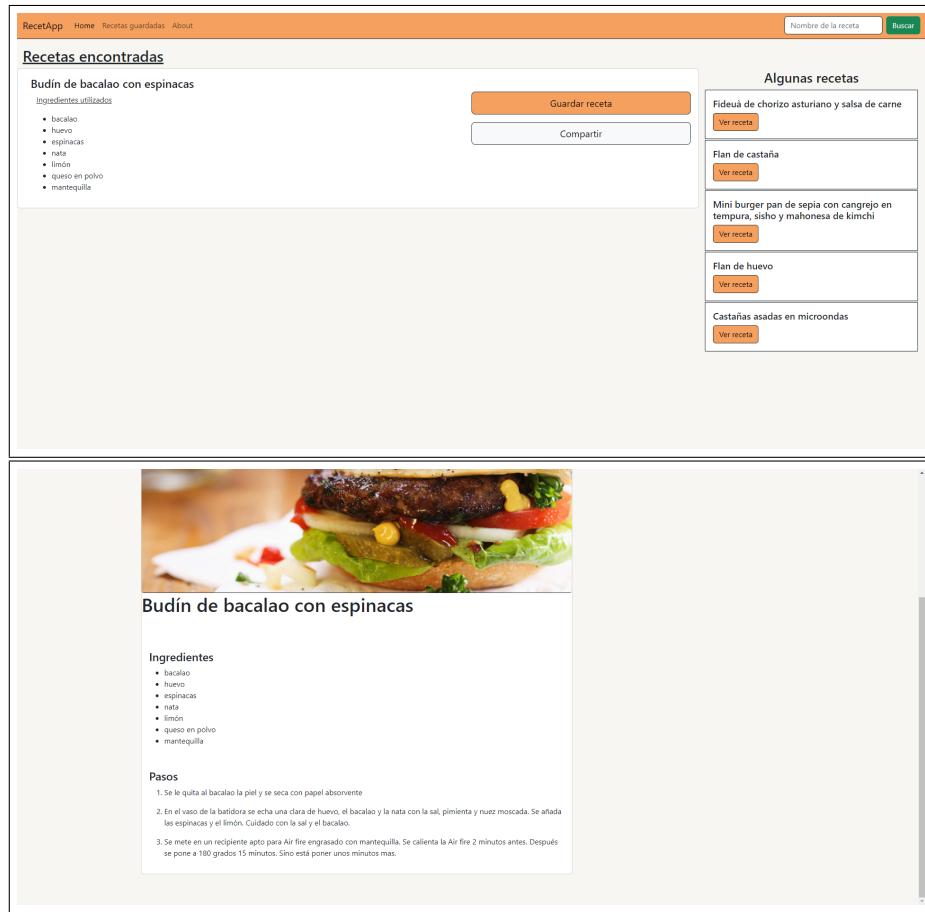


Figura 8.3: Pasos finales de Daniel

Otro viaje por la aplicación es el de María Encarnación, al que su hijo recomienda usar la aplicación y se la muestra en su tableta, enseñándola a usarla. Observa el buscador inicial, incluyendo algunos ingredientes, pero no quiere buscar recetas por ingredientes, así que los elimina todos con el botón “Eliminar selección”. Para buscar recetas por nombre su hijo le dijo que tenía que utilizar la barra de arriba, así que busca “Bizcocho”.

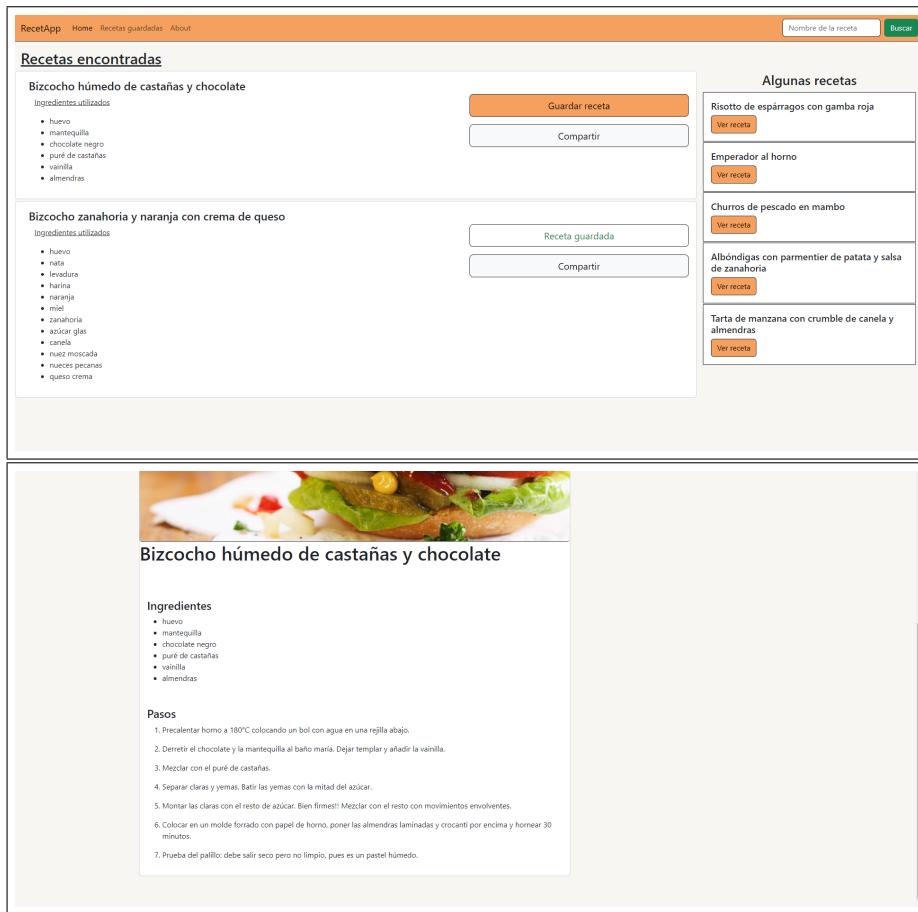


Figura 8.4: Resultados a la búsqueda de María Encarnación

Decide escoger el bizcocho húmedo de castaña y guardar el de zanahoria para otro día.

Con esta interfaz se recogerán las opiniones de los usuarios tanto para mejorar la funcionalidad como para mejorar esta interfaz final. En conclusión, la interfaz funciona correctamente y de la manera esperada para el uso de los usuarios, tanto para estudiantes que necesitan restringir la búsqueda de recetas para incluir solo recetas que puedan hacer con los ingredientes que tienen como para usuarios mayores que no recuerdan las recetas con exactitud.

8.3. Prueba de carga

Para comprobar el rendimiento de la aplicación propuesta como solución al problema del usuario, es necesario utilizar una herramienta de *load testing*. Se ha elegido Locust como herramienta. Esta es muy sencilla de utilizar y

permite realizar las pruebas escribiendo las tareas que realiza un usuario en código de Python, se puede revisar en el anexo de pruebas de la memoria.

Como prueba de carga, se ha definido un camino para el usuario en el que accede a la página principal de la web, visita una receta de las aleatorias que se muestran en la columna derecha de la pantalla y por último busca una receta a partir de unos ingredientes que tiene en su despensa.

Por ser una prueba de carga, se busca que los usuarios realicen estas tres acciones de manera continua, cada vez escalando más el número de usuarios que realizan esta acción repetitiva, con el objetivo de comprobar cuando el servidor se ve sobrepasado y empieza a perder rendimiento.

Locust cuenta con una interfaz gráfica para lanzar los test, que permite configurar el número de usuarios que tendrá la prueba y el ratio de *spawn* de los usuarios, permitiendo incrementar gradualmente el nivel de estrés. Además, genera en tiempo real un gráfico que permite observar el número de usuarios, cuantas peticiones fallidas se han detectado y el tiempo de respuesta de la página web.

La prueba se ha realizado con mil usuarios, incrementándolos gradualmente de diez en diez cada uno realizando los pasos mencionados anteriormente.

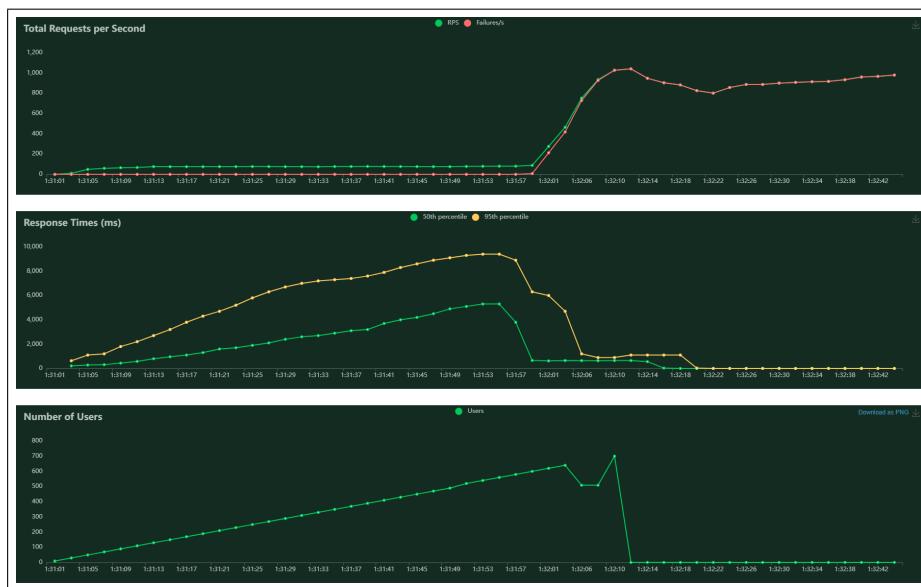


Figura 8.5: Resultados de la prueba de carga

En la imagen se observa como la aplicación aguanta hasta setecientos usuarios, teniendo problemas entre seiscientos y setecientos usuarios, hasta que al final el servidor no puede más y deja de poder responder a las peticiones correctamente.

Capítulo 9

Despliegue

En este capítulo se analizará el despliegue de la aplicación y la manera en la que se puede realizar de manera teórica.

9.1. Web Server Gateway Interface

Django ya cuenta con una capa *middleware* que permite el uso de un WSGI y por defecto el fichero *wsgi.py* ya contiene lo necesario para utilizar un WSGI, simplemente se tiene que lanzar el comando del gestor de tareas para que Gunicorn comience a escuchar peticiones HTTP.

Gunicorn se puede ejecutar de manera manual cada vez que se reinicie el servidor, pero se recomienda encarecidamente crear un servicio utilizando el *Systemd*, para ello es necesario crear un fichero con extensión *.service* y habilitarlo para que se ejecute automáticamente en el arranque de sistema.

9.2. Servidor Web

La configuración básica de Nginx es muy sencilla. Siendo necesario instalar el paquete y cambiar el fichero de configuración *default* en la ruta */etc/nginx/sites-available/default*, es completamente necesario cambiar la ruta de archivos estáticos y llevársela a Nginx, cambiando el parámetro *STATIC_URL* del fichero de ajustes de django.

Adicionalmente se necesita un host, donde alojar el servidor web, que se elegirá *dietplanner.com* como dominio de prueba en el campo teórico, sin poder registrar el nombre en algún sitio de para el registro de dominios como GoDaddy.

Para aplicar el cambio de dominio, se debe cambiar el *server_name* de Nginx y el parámetro *Allowed hosts* de Django, con esta información.

Se utilizará Nginx como reverse proxy para añadir más seguridad a la infraestructura del proyecto.

9.3. Infraestructura en la nube

Como servicio de infraestructura en la nube se utilizará AWS, aunque existen otros IaaS como Azure y Google Cloud. Pero AWS ofrece un gran abanico de servicios que pueden ser útiles en el desarrollo del proyecto, no solo en seguridad, sino en escalabilidad.

Para el despliegue del proyecto se pueden seguir varias alternativas, la más razonable es utilizar una instancia basada en Linux. En AWS, las instancias se conocen como Elastic Cloud Computing (E2) y tienen diferentes tamaños dependiendo de que componentes se quieran potenciar. Para el proyecto se utilizarán E2 de la familia *T3*, orientada a un propósito general, en cuanto al tamaño no se esperan inicialmente demasiadas peticiones simultáneas así que se puede utilizar una instancia *t3.medium* con 2vCPU y 4GB de memoria. De manera normal se utilizarán dos instancias E2, pero se incluye una plantilla para escalar la instancia y ejecutar la aplicación, dándose a conocer al balanceador de carga para añadir escalabilidad a la aplicación. Cuando no haya mucha carga se escalaran hacia abajo automáticamente, ahorrando costes.

Actualmente se tiene SQLite en un fichero ubicado en la raíz del proyecto, pero para añadir escalabilidad, se utilizará una instancia del servicio de base de datos relacional (RDS) de AWS. Actualmente no se permite el uso de SQLite en AWS, con lo cual existen dos alternativas, confiar en SQLite3 ya que permite la lectura simultánea y no se realizará ninguna escritura o bien migrar la base de datos a una Aurora con el motor MySQL o PostgreSQL, que son menos costosas que utilizar una base de datos “pura”. Al igual que ocurría con las E2, se debe elegir el tipo de instancia para la base de datos, se elegirá la más pequeña, una Aurora equilibrada. Para una alta disponibilidad de los datos, se migrarán a una Aurora, ya que la migración no incurre mucho esfuerzo, simplemente se cambia la dirección de la base de datos y se cargan los datos a la nueva RDS, El *ORM* de Django se encargará del resto.

Se han definido tanto la base de datos como el tipo de instancia E2 que se utilizará, pero no existe una infraestructura de red. Para contener la infraestructura en la nube de manera privada, se utiliza una *Virtual Private Cloud* a la que se le asigna un bloque *CIDR*, permitiendo que la infraestructura interna cuente con una dirección privada para cada recurso. La red de una VPC se subdivide en *subnets*, partiendo ese bloque *CIDR* en pequeñas redes orientadas al uso por parte de diferentes servicios. Para este proyecto solo serán necesarias una *subnet* para la E2 y dos *subnets* para una RDS con alta disponibilidad, dividiéndose en varias zonas de disponibilidad de Amazon.

En cuestión de seguridad, las *subnet* cuentan con una *Access Control List* (ACL) que por defecto deja pasar cualquier tráfico en la red, para proteger las instancias normalmente se utilizan grupos de seguridad con

reglas explícitas. Por ejemplo, se debe poder permitir la gestión de una instancia a través de Secure Shell solo para determinado grupo de personas. Para ello se puede utilizar un *site-to-site-VPN* ofrecido por Amazon, que conecta el equipo local con las instancias a través de un túnel VPN. Y configurar el grupo de seguridad para permitir solo las conexiones Secure Shell a esa instancia. De la misma manera será necesario crear un grupo de seguridad para las conexiones a través del puerto 5432 para PostgreSQL o del puerto 3306 para MySQL.

Para permitir el acceso a la aplicación de manera segura, se utilizará un *Elastic Load Balancer* (ELB) que actúe como una capa de seguridad, transportando únicamente las peticiones HTTPs a la instancia E2 con nuestra aplicación, sin abrir dicha instancia a internet. El balanceador de carga, desechará toda petición que no sea HTTP, las conexiones no seguras en el puerto 80 se pueden redirigir al puerto 443, pero para ello es necesario un certificado. Amazon ofrece su propio servicio de certificados, solo aplicables al entorno de AWS, pero por ahora es suficiente. Este balanceador tiene una dirección pública que se registrará como dominio, al resolver el dominio en un servidor DNS, se obtendrá esta dirección pública.

Para conectar la instancia a internet y ser capaces de descargar actualizaciones es necesarias de seguridad y mantenimiento es necesario una *Internet Gateway*, que tiene su coste a parte.

En conclusión, la infraestructura descrita quedaría así:

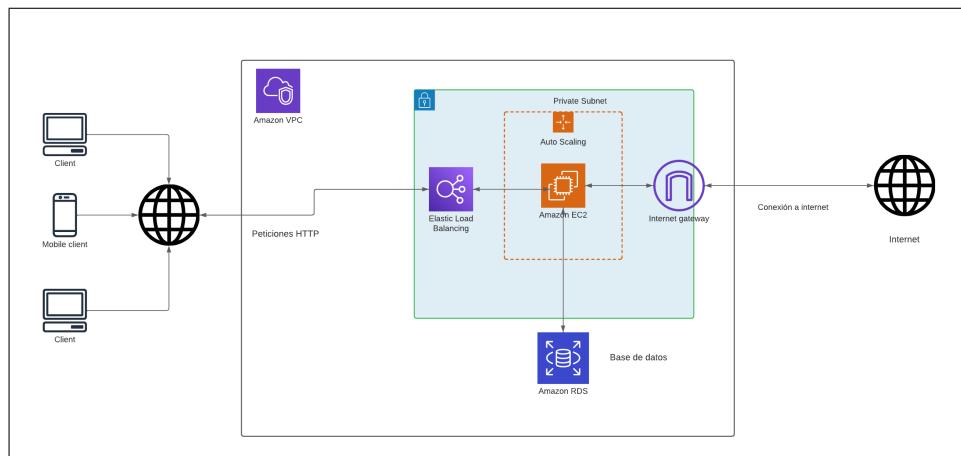


Figura 9.1: Diagrama de infraestructura

Capítulo 10

Costes

En este capítulo se definirán los costes del desarrollo de la aplicación, tanto los costes de IaaS como los costes derivados de la manufactura del proyecto.

10.1. Infraestructura

Un despliegue en la nube es costoso, aunque depende de los servicios que se utilicen en él, para detallar los costes de manera sencilla, AWS ofrece una calculadora de precios. Para resumir, en este despliegue se utilizan los siguientes servicios de AWS:

1. Instancias E2 t3.medium con 2vCPU y 4GB de memoria: 58.86 - 117.72 USD/mes
2. Instancia Aurora PostgreSQL 2vCPU, 4GB de memoria y *network performance: low to moderate*: 121.86 USD/mes
3. Balanceador de carga para una sola aplicación suponiendo 10GB/hora de peticiones: 80.64 USD/mes

En conclusión, la suma total es de 231.93 - 291.09 USD/mes. 2783.16 - 3493.08 USD/año. Coste que no puedo aportar actualmente de mi bolsillo, por ello se deja planteada teóricamente el despliegue de la aplicación en la nube de AWS. Cabe destacar que la instancia EC2 es parte de un grupo de escalado, que permite añadir instancias automáticamente cuando se necesiten y eliminarlas cuando haya poca carga de trabajo. Se estiman que haya normalmente unas 2 instancias en ejecución y como máximo 4 instancias.

10.2. Proyecto

El coste del equipo técnico se calcula teniendo en cuenta el ordenador utilizado, un Lenovo ThinkPad P15v de segunda generación, de un coste estimado de 2.497,00 euros, teniendo en cuenta una vida útil de cinco años

y una amortización de un 25 %, con un uso de unos 4 meses de proyecto, habiendo trabajado alrededor de unas 500 horas en el proyecto en un formato de jornada partida de cinco horas. Cabe destacar que al haberse combinado un trabajo a jornada completa, se ha intentado compatibilizar de la mejor manera, intentando cumplir con los horarios, aunque no siempre ha sido posible.

1. Equipo técnico
 - a) Hardware
 - Ordenador Lenovo ThinkPad P15v 2da Gen: 520e
 - b) Software
 - Todo el software utilizado es de código abierto.
2. Equipo de trabajo: 2219e/mes
3. Total del proyecto: 5458e

Figura 10.1: Desglose del coste del proyecto

Capítulo 11

Conclusiones

11.1. Futuro del proyecto

El proyecto no acaba con esta memoria, sino que se han marcado objetivos futuros para aumentar la utilidad de la aplicación, se han planteado dos objetivos futuros.

- Desplegar la infraestructura en la nube de manera práctica
- Sustituir ingredientes dinámicamente para personas con alérgenos en las recetas

Si bien es cierto que desplegar la infraestructura supone un coste elevado, supone una ventaja frente a contar con los servidores de la aplicación on-premises. Pero habría que analizar la manera en la que se puede obtener beneficio de la aplicación para costear su mantenimiento en la nube.

El segundo objetivo a alcanzar pretende solucionar la problemática que tienen algunos usuarios de alterar las recetas originales debido a alguna intolerancia o alergia alimentaria. La aplicación permitirá que al seleccionar los ingredientes deseados por el usuario, las recetas que contengan ingredientes problemáticos se sustituirán por ingredientes coherentes en la receta, pero también se buscarán recetas a las que se le puedan sustituir ingredientes originales por sustitutos seleccionados por el usuario.

11.2. Conclusiones

En conclusión, este trabajo de fin de grado ha abordado la gestión de un proyecto siguiendo una mentalidad ágil, en el que se ha llevado a cabo el diseño y desarrollo de una solución completamente orientada al usuario desde la definición del problema hasta la entrega de una solución final. Para ello se ha tenido que realizar una investigación en la que se ha necesitado investigar las metodologías actuales para este tipo de trabajos.

Se superó el problema de la ausencia de unos datos de recetas en un formato correcto, analizando diferentes soluciones para extraer los datos. Desarrollando un robot que permitiera extraer las recetas de una fuente fiable y almacenándolas en un fichero estandarizado para ser cargado en una base de datos.

Además, se ha diseñado una solución a la problemática que tenían los usuarios perfilados en el apartado de metodología. Una vez completada la solución, no se quiso dejar como una biblioteca de funciones que recuperen información de una base de datos sino que al estar el proyecto tan orientado al usuario final, era necesario llevar la solución hasta ellos.

Para que el usuario final pudiera acceder y utilizar la aplicación se ha creado una interfaz como prototipo, que se irá mejorando en futuras iteraciones, después de escuchar el feedback que tengan que aportar. La interfaz actual es simple, con colores vivos que acentúen los rasgos minimalistas de la página web. Se utiliza una paleta de colores anaranjados, para dar calidez a la página.

También era necesario conectar la interfaz diseñada con la funcionalidad desarrollada inicialmente. Por ello, se eligió un framework que facilitó la implementación de los puntos de acceso a la aplicación y la posibilidad de gestionar la aplicación web como conjunto.

Por último, se ha analizado el despliegue en una infraestructura en la nube de AWS y proponiendo una solución de infraestructura que permita mantener una alta disponibilidad, sin escatimar en seguridad. Aunque el coste inicial que se tiene que aportar para el despliegue en esta infraestructura es demasiado alto como para poderse probar.

Entre los desafíos que he encontrado durante el viaje que supone el desarrollo del proyecto, sin duda el más complicado ha sido seguir una metodología orientada al usuario, ya que, durante el grado no se ha abarcado este tema en profundidad o con un enfoque práctico, a excepción de una asignatura. En el desarrollo de código, me he valido de los conocimientos adquiridos a lo largo de estos cinco años en el grado, aprovechando el contenido de las asignaturas orientadas al desarrollo web y al diseño de software.

Para terminar, a nivel personal, este proyecto ha sido una experiencia enriquecedora que ha fortalecido tanto mi conocimiento como mis habilidades en los ámbitos de gestión de proyectos y desarrollo de aplicaciones web.

Bibliografía

- [TM+11] Katja Thoring, Roland M Müller y col. “Understanding design thinking: A process model based on method engineering”. En: *DS 69: Proceedings of E&PDE 2011, the 13th International Conference on Engineering and Product Design Education, London, UK, 08.-09.09. 2011.* 2011, págs. 493-498.
- [ost14] Mark Drake ostezer. *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. Feb. de 2014. URL: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>.
- [Gas15] Andrea Gasparini. “Perspective and use of empathy in design thinking”. En: *ACHI, the eight international conference on advances in computer-human interactions*. 2015, págs. 49-54.
- [Dre16] Stuart Dredge. *10 of the best cookery apps for iPhone, iPad and Android*. Jun. de 2016. URL: <https://www.theguardian.com/technology/2016/jun/02/10-best-cookery-apps-iphone-ipad-android>.
- [Pér16] Anna Pérez. *Metodología agile: ¿Cuáles son los 12 principios de su modelo?* Ago. de 2016. URL: <https://www.obsbusiness.school/blog/metodologia-agile-cuales-son-los-12-principios-de-su-modelo>.
- [Pol16] Denis Polkhovskiy. “Comparison between continuous integration tools”. Tesis de mtría. 2016.
- [BP17] Panuchart Bunyakiati y Chadarat Phipathananunth. *Cherry-picking of code commits in long-running, multi-release software*. 2017.
- [Wol17] Radosław Wolniak. “The Design Thinking method and its stages”. En: *Systemy Wspomagania w Inżynierii Produkcji* 6.6 (2017), págs. 247-255.
- [Ani19] Anita. *¿De qué se trata la metodología de persona?* Nov. de 2019. URL: <https://proteina.marketing/lametodologiadepersona/>.

- [Sto21] Andrea Núñez-Torrón Stock. *Las mejores apps de dieta y nutrición totalmente gratis: recetas, consejos y escáneres de alimentos para tener a mano en el móvil.* Jun. de 2021. URL: <https://www.businessinsider.es/mejores-apps-dieta-nutricion-gratis-877177>.
- [Ans22] LD Ansley Hill RD. *The 11 Best Meal Planning Apps to Help You Lose Weight.* Jun. de 2022. URL: <https://www.healthline.com/nutrition/best-meal-planning-apps>.
- [22] [22] *Flask vs. Django: una comparativa de los frameworks de Python.* Nov. de 2022. URL: <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/flask-vs-django/>.
- [Tha22] Jenny Thai. *¿Qué son los hitos de un proyecto y como gestionarlos?* Nov. de 2022. URL: <https://asana.com/es/resources/project-milestones>.
- [Vat22] Rohan Vats. *GitHub vs GitLab: Difference Between GitHub and GitLab.* Sep. de 2022. URL: <https://www.upgrad.com/blog/github-vs-gitlab-difference-between-github-and-gitlab/>.
- [Bah23] Ahmed Bahgat. *Flask vs Django: Let's Choose Your Next Python Framework.* Sep. de 2023. URL: <https://kinsta.com/blog/flask-vs-django/#:~:text=Flask%20provides%20support%20for%20only,management%20and%20support%20for%20sessions>.
- [Kum23] Pawan Kumar. *Top 8 Continuous Integration Tools in 2023.* Ago. de 2023. URL: <https://www.browserstack.com/guide/continuous-integration-tools>.
- [Rod23] Stephen Roddewig. *10 Best Continuous Integration Tools for DevOps in 2022.* Jul. de 2023. URL: <https://blog.hubspot.com/website/continuous-integration-tools>.
- [Tay23] David Taylor. *20 Best CI/CD Tools (2023 Update).* Sep. de 2023. URL: <https://www.guru99.com/top-20-continuous-integration-tools.html>.
- [con] laboratorio de contenidos. *Design Thinking: la fase de idear.* URL: <https://laboratoriodecontenidos.cl/design-thinking-como-idear/>.
- [FSA] FSA. *Incidencia de la alergia alimentaria.* URL: <https://funsap.org/alergia-alimentaria/incidencia/>.
- [Mak] Matt Makai. *Object-relational Mappers (ORMs).* URL: <https://www.fullstackpython.com/object-relational-mappers-orms.html>.

Apéndice A

Base de datos

```
1     RecipeIngredient.objects.all().delete()
2     Recipes.objects.all().delete()
3     Ingredients.objects.all().delete()
4
5     configuration = Configuration()
6     # Carga de los datos del CSV
7     with open(configuration.data_ingredients, "r") as f
8         :
9         reader = csv.reader(f, delimiter=",")
10        next(reader)
11        for row in reader:
12            ingredient = Ingredients(id=row[0], name=row
13                [1])
14            ingredient.save()
15
16        ## Cargar las recetas en formato JSON
17        with open(configuration.data_recipes, "r", encoding
18            ="utf-8") as f:
19            recipes = json.load(f)
20
21        for recipe in recipes:
22            _steps = ";" .join(recipe['Steps'])
23            row = Recipes(name=recipe['Name'], steps=_steps,
24                number_of_ingredients=None)
25            row.save()
26
27        for recipe in recipes:
28            ingredients = recipe['Ingredients_ID']
29            name = recipe['Name']
30            resultado = Recipes.objects.get(name=name)
31            id = resultado.id
32
33            for ingredient in ingredients:
34                result_ingredient = Ingredients.objects.get(
35                    id=ingredient)
36                recipeingredient = RecipeIngredient(
37                    recipe_id=resultado, ingredient_id=
38                        result_ingredient)
39                recipeingredient.save()
```

Figura A.1: Script para insertar los datos en la Base de datos

```
1     ## Actualizar tabla para incluir el numero de
2     # ingredientes de la receta
3     for recipe in recipes:
4         name = recipe['Name']
5         resultado = Recipes.objects.get(name=name)
6         id = resultado.id
7         num_ingredients = RecipeIngredient.objects.
              filter(recipe_id=id).count()
8         Recipes.objects.filter(id=id).update(
9             number_of_ingredients=num_ingredients)
```

Figura A.2: Continuación del Script para insertar los datos en la Base de datos

```
1      def buscarIngredienteNombre(self ,name: str):
2          """
3              Metodo que recupera una lista de ingredientes de la
4                  base de datos
5
6          Parameters
7          -----
8              name : str
9                  Nombre del ingrediente a buscar.
10
11         Raises
12         -----
13             Error_DB : No hay conexion con la base de datos.
14
15         Returns
16         -----
17             resultados : List<(id, name)>
18                 Lista de ingredientes encontrados
19             """
20
21     try:
22         results = Ingredients.objects.filter(
23             name__icontains=name).all()
24         format_results = []
25         for result in results:
26             format_results.append((result.id, result.
27                 name))
28     return format_results
29 except:
30     raise Error_DB("Error de base de datos: No hay
31                 conexion con la base de datos")
```

Figura A.3: Método para buscar ingredientes por nombre en la Base de datos

```
1     def buscarIngredienteID(self ,id: int):
2         """
3             Metodo que busca un ingrediente por identificador
4                 en la base de datos.
5
6             Parameters
7             -----
8                 id : Int
9                     Identificador del ingrediente.
10
11            Raises
12            -----
13                Error_DB : No hay conexion con la base de datos.
14
15            Returns
16            -----
17                results : Tuple(id, name)
18                    Ingrediente encontrado
19
20        try:
21            results = Ingredients.objects.get(id=id)
22            return (results.id, results.name)
23        except:
24            raise Error_DB("Error de base de datos: No hay
25                            conexion con la base de datos")
```

Figura A.4: Método para buscar ingredientes por ID en la Base de datos

```
1      def getIngredientes(self, id: int):
2          """
3              Metodo para obtener los ingredientes de una
4                  receta por ID
5
6          Parameters
7          -----
8          id : int
9              Identificador de la receta
10
11         Raises
12         -----
13             Error_DB : No hay conexion con la base de datos
14
15
16         Returns
17         -----
18             list_ingredients : list<str>
19                 Lista de nombres de ingredientes que lleva
20                     la receta
21
22         """
23     try:
24         list_ingredients = Ingredients.objects.
25             filter(recipeingredient_recipe_id=id).
26             select_related('recipeingredient').
27             values_list('name', flat=True)
28         return list_ingredients
29     except:
30         raise Error_DB("Error de base de datos: No
31                         hay conexion con la base de datos")
```

Figura A.5: Metodo para obtener los ingredientes de una receta por ID en la Base de datos

```
1     def buscarRecetasNombre( self , name : str ):  
2         """  
3             Metodo que busca una lista de recetas por nombre en  
4                 la base de datos.  
5  
6             Parameters  
7             -----  
8             name : str  
9                 Nombre de la receta.  
10            Raises  
11            -----  
12            Error_DB : No hay conexion con la base de datos.  
13            Returns  
14            -----  
15            resultados : List<(Recipes)>  
16            Lista de recetas encontradas  
17            """  
18            try:  
19                return Recipes.objects.filter(name__icontains=  
20                                                name).all()  
21            except:  
22                raise Error_DB("Error de base de datos: No hay  
                    conexion con la base de datos")
```

Figura A.6: Metodo para buscar una lista de recetas por nombre en la Base de datos

```
1     def buscarRecetasID(self , id: int):
2         """
3             Metodo que busca una receta por identificador en la
4                 base de datos.
5
6             Parameters
7             -----
8
9                 ID : Int
10                    Identificador de la receta.
11
12            Raises
13            -----
14
15                Error_Buscador : La sentencia esta mal escrita.
16
17            Returns
18            -----
19
20                resultados : Tuple(id,name,steps,
21                    number_of_ingredients)
22                    Lista de recetas encontradas
23
24            """
25
26            try:
27                result = Recipes.objects.get(id=id)
28                return (result.id , result.name , result.steps ,
29                        result.number_of_ingredients)
30            except:
31                raise Error_DB("Error de base de datos: No hay
32                                conexion con la base de datos")
```

Figura A.7: Metodo para buscar una lista de recetas por ID en la Base de datos

```
1     def buscarRecetasIngredientesNumber(self , id: list [
2         int]):
3         """
4             Metodo que busca una lista de recetas por numero de
5                 ingredientes y que contengan al menos un
6                 ingrediente de la lista en la base de datos.
7
8             Parameters
9             -----
10            id : List<Int>
11                Identificadores de ingredientes a utilizar.
12
13            Raises
14            -----
15            Error_DB : No hay conexion con la base de datos.
16
17            Returns
18            -----
19
20            resultados : List<(Recipes)>
21                Lista de recetas que cumplen la condicion
22
23        """
24        try:
25            recipes = Recipes.objects.filter(
26                Q(number_of_ingredients__lte=len(id)) & Q(
27                    recipeingredient__ingredient_id__in=id)
28            ).distinct()
29            return recipes.all()
30        except:
31            raise Error_DB("Error de base de datos: No hay
32                            conexion con la base de datos")
```

Figura A.8: Metodo para buscar una lista de recetas por su numero de ingredientes y que contengan un ingrediente de la lista en la Base de datos

```
1     def buscarRecetasIngredientes(self , id: int):
2         """
3             Metodo que recupera los ingredientes de una receta
4                 por identificador.
5
6             Parameters
7             -----
8                 id : Int
9                     Identificador de receta
10
11            Raises
12            -----
13                Error_DB : No hay conexion con la base de datos.
14
15            Returns
16            -----
17                resultados : List<(ingredient_id,ingredient_name)>
18                    Lista de recetas que cumplen la condicion
19
20        try:
21            receta_ingredientes = RecipeIngredient.objects.
22                filter(recipe_id=id).values('ingredient_id',
23                    'ingredient_name=F('ingredient_id__name')')
24            resultado = []
25            for receta in receta_ingredientes:
26                resultado.append((receta['ingredient_id'],
27                    receta['ingredient_name']))
28        return resultado
29    except:
30        raise Error_DB("Error de base de datos: No hay
31            conexion con la base de datos")
```

Figura A.9: Metodo para recuperar los ingredientes de una receta por su identificador de la Base de datos

```
1     def getRecetasRandom( self , number: int ):
2         """
3             Metodo para obtener recetas aleatorias de la base
4                 de datos
5
6             Parameters
7             -----
8                 number : int
9                     Numero de recetas a obtener
10
11            Raises
12            -----
13                Error_DB : No hay conexion con la base de datos.
14
15            Returns
16            -----
17                recetas : list<(Recipes)>
18                """
19
20                try:
21                    items = list( Recipes . objects . all () )
22                    random_items = random . sample( items , number )
23                    return random_items
24                except:
25                    raise Error_DB( "Error de base de datos: No hay
26                        conexion con la base de datos" )
```

Figura A.10: Metodo para recuperar recetas aleatorias de la Base de datos

```
1     from decouple import config
2
3     class Configuration:
4         """Clase configuracion para modularizar el uso de
5            la base de datos"""
6         _instance = None
7
8         def __new__(cls):
9             """Metodo para asegurar el singleton"""
10            if not cls._instance:
11                cls._instance = super().__new__(cls)
12            return cls._instance
13
14        def __init__(self):
15            """Constructor de clase"""
16            self._db_name = config("DB_NAME")
17            self._data_recipes = config("DATA_RECIPES")
18            self._data_ingredients = config(
19                "DATA_INGREDIENTS")
20
21        @property
22        def db_name(self):
23            """Getter nombre base de datos"""
24            return self._db_name
25
26        @property
27        def data_recipes(self):
28            """Getter ruta al fichero de recetas"""
29            return self._data_recipes
30
31        @property
32        def data_ingredients(self):
33            """Getter ruta al fichero de ingredientes"""
34            return self._data_ingredients
```

Figura A.11: Clase configuracion para las variables de entorno

Apéndice B

Plataforma Web

B.1. Interfaz

```
<head>
    <title>RecetApp</title>

    <!--Custom style css for more spaces-->
    <link rel="stylesheet" href="{% static 'css/custom.css' %}">

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/
        bootstrap@5.2.3/dist/css/bootstrap.min.css" integrity="
        sha384-
        rbsA2VBKQhggwzxH7pPCaAq046Mgn0M80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65
        " crossorigin="anonymous">

    <!-- Latest compiled and minified JavaScript -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist
        /js/bootstrap.min.js" integrity="sha384-
        cuYeSxntonZOPPNlHhBs68uyIAVpII0ZZ5JeqqvYYIcEL727kskC66kF92t6X12V
        " crossorigin="anonymous"></script>

    <!-- Latest compiled JQuery-->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery
        /3.7.1/jquery.min.js"></script>
</head>
```

Figura B.1: Template de los ficheros fuente de la interfaz

```
<body class="">
    <nav class="navbar navbar-expand-lg navbar-light background-
        nav border-bottom border-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="/">RecetApp</a>
            <button class="navbar-toggler" type="button" data-bs-
                toggle="collapse" data-bs-target="#
                    navbarSupportedContent" aria-controls="
                    navbarSupportedContent" aria-expanded="false" aria-
                    label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="
                navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active" aria-current="page"
                            href="#">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link " aria-current="page" href="/
                            guardadas/">Recetas guardadas</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link" href="#">About</a>
                    </li>
                </ul>
                <form class="d-flex" action="/resultados/" method="
                    POST">
                    {% csrf_token %}
                    <input class="form-control me-2 border border-dark"
                        type="search" placeholder="Nombre de la receta"
                        aria-label="Search" name="search-recetas">
                    <button class="btn btn-success" type="submit">Buscar
                    </button>
                    <input type="hidden" name="metodo" value="recetas">
                </form>
            </div>
        </div>
    </nav>
```

Figura B.2: Template del navegador de la interfaz

```
<main class="background">
    <div class="container-fluid pt-3">
        <div class="row justify-content-center">
            <div class="col-lg-9 col-md-9 container-fluid">
                {%
                    block content %
                %}
                {%
                    endblock %
                </div>
                <div class="col-lg-3 col-md-9 mt-5 pe-lg-3 d-flex flex-column overflow-auto">
                    {%
                        if top %
                    <h3 class="w-100 text-center">Algunas recetas</h3>
                    {%
                        for item in top %
                            <div class="card border rounded w-100" style="width: 18rem;">
                                <a class="link-unstyled" href="#">
                                    <div class="card-body border border-dark background-card">
                                        <h5 class="card-title">{{item.name}}</h5>
                                        <a href="/receta/{{item.id}}" class="btn button-bg border border-dark">Ver receta</a>
                                    </div>
                                </a>
                            </div>
                        {%
                            endfor %
                        }
                        {%
                            endif %
                        </div>
                    </div>
                </div>
            </main>
        </body>
        <script src="{% static 'js/ buscador-ingredientes.js' %}"></script>
</html>
```

Figura B.3: Template del contenedor principal la base de la interfaz

```
{% extends 'base.html' %}  
{% block content %}  
    <div class="row mt-5">  
        <div class="col-lg-2 col-none"></div>  
        <div class="col-lg-8 col-12 justify-content-center">  
            <h1 class="text-center">RecetApp</h1>  
            <form class="form-control border border-dark" action  
                  ="/resultados/" method="POST">  
                {% csrf_token %}  
                <div class="d-inline-flex w-100">  
                    <input id="buscador-ingredientes" class="form-  
                        control me-2" type="search" placeholder="  
                        Buscar Ingredientes" aria-label="Search">  
                    <button id="enviarIngredientes" class="btn btn-  
                        -success" type="submit">Buscar Recetas</  
                        button>  
                    <div id="eliminarSeleccion" class="btn btn-  
                        danger ms-3">Borrar Seleccion</div>  
                </div>  
                <div class="container">  
                    <table id="tabla" hidden class="table mt-5  
                        container ingredientes">  
                        <thead class="container">  
                            <tr class="justify-content">  
                                <th>Nombre</th>  
                                <th class="text-center">  
                                    Seleccionado</th>  
                            </tr>  
                        </thead>  
                        <tbody id="tabla-ingredientes" class  
                              ="">  
                            </tbody>  
                    </table>  
                </div>  
                <input type="hidden" name="metodo" value="  
                    ingredientes">  
            </form>  
        <div class="col-lg-2 col-none"></div>  
    </div>  
    </div>  
    {% endblock %}
```

Figura B.4: Bloque HTML del buscador

```
{% extends 'base.html' %}  
{% load static %}  
{% block content %}  
<div class="container-fluid h-100 background">  
    <div class="row d-flex inline bg-light rounded rounded-sm  
        background">  
        <div class="col-lg-8 background">  
            <div class="card mt-5 w-100">  
                <img class="card-img border-bottom border-dark" src  
                    ="{{ static 'img/img-template.jpg' }}" width  
                    ="100%">  
                <h1 class="card-title">{{receta_nombre}}</h1>  
                <div class="ps-3 mt-5 card-body">  
                    <h4>Ingredientes</h4>  
                    <ul>  
                        {% for item in receta_ingredientes %}  
                            <li>{{item}}</li>  
                        {% endfor %}  
                    </ul>  
                    <h4 class="mt-5">Pasos</h4>  
                    <ol>  
                        {% for item in receta_pasos %}  
                            <li><p>{{item}}</p></li>  
                        {% endfor %}  
                    </ol>  
                </div>  
            </div>  
        </div>  
        <div class="col-lg background">  
            <div class="container-fluid">  
                <div class="row rounded-sm d-flex flex-column justify-  
                    content-center-100">  
                    <button id="guardarReceta" data-id="{{receta_id}}" class="btn  
                        save-btn flex-grow button-bg btn-outline-success  
                        btn-lg mt-5 mb-3 border border-dark" saved={{saved}}  
                        origin="receta">Guardar receta</button>  
                    <div id="compartirReceta" data-id="{{receta_id}}" class="btn  
                        share-btn flex-grow btn-lg btn-light border border-  
                        dark">Compartir</div>  
                </div>  
            </div>  
        </div>  
    </div>  
</div>  
{% endblock %}
```

Figura B.5: Bloque HTML de la receta

```

{% extends 'base.html' %}
{% block content %}
    {% if resultados %}
        <h2><u>Recetas encontradas</u></h2>
        {% for resultado in resultados %}
            <div id="resultados" class="mt-1">
                <div class="card mb-1 d-flex-inline flex-row row">
                    <div class="col-lg-8">
                        <a class="link-unstyled w-100" href="/
                            receta/{{resultado.0}}">
                            <div class="card-img"></div>
                            <div class="card-body">
                                <div class="card-title"><h4>{{
                                    resultado.1 }}</h4></div>
                                <div class="card-text">
                                    <div class="container d-flex-
                                        inline">
                                        <p class=""><u>
                                            Ingredientes
                                            utilizados</u></p>
                                    <ul>
                                        {% for item in
                                            resultado.2%}
                                            <li>{{item}}</li>
                                        {% endfor %}
                                    </ul>
                                </div>
                            </div>
                        </a>
                    </div>
                    <div class="col-lg container">
                        <div class="row rounded-sm d-flex flex-
                            column justify-content-center-100 pe
                            -3">
                            <button data-id={{resultado.0}} class=""
                                save-btn btn flex-grow button-bg btn
                                -outline-success btn-lg mt-5 mb-3
                                border border-dark" saved="{{
                                resultado.3}} origin="resultado">
                                Guardar receta</button>
                            <button id="compartirReceta" data-id={{{
                                resultado.0}} class="btn share-btn
                                flex-grow btn-lg btn-light border
                                border-dark">Compartir</button>
                        </div>
                    </div>
                </div>
            {% endfor %}
            {% else %}
                <h2>No se han encontrado recetas</h2>
            {% endif %}
        {% endblock %}

```

Figura B.6: Bloque HTML de la lista de resultados

```

$(document).ready(function() {
    $('#buscador-ingredientes').on('input', function() {
        var query = $(this).val();
        $.ajax({
            url: '/buscar/',
            data: {'query': query},
            dataType: 'json',
            success: function(response) {
                BuscarIngrediente(response)
            }
        });
    });
});

```

Figura B.7: Evento del buscador

```

function BuscarIngrediente(response) {
    if(ObtenerCookie("IDs") == null){
        var json = {
            seleccionados: []
        };
        crearCookie("IDs",JSON.stringify(json),1);
    }
    $('#tabla').removeAttr('hidden');
    var tabla = $('#tabla-ingredientes');
    if(response == null){
        tabla.empty();
    } else{
        var resultados = response.resultados;
        tabla.empty();
        for (var i = 0; i < resultados.length; i++) {
            var fila = $('<tr data-id="'+resultados[i].id+'" onclick="'
                SeleccionarIngrediente(this)">');
            fila.append($('<td class="">').text(resultados[i].nombre))
            ;
            var td = $('<td class="text-center">');
            var svg = '<svg';
            if(isSeleccionado(resultados[i].id) == false){
                svg += " hidden"
            }
            svg += <tag del SVG>
            td.append(svg);
            fila.append(td);
            tabla.append(fila);
        }
    }
}

```

Figura B.8: Función para buscar dinámicamente los ingredientes

```

function SeleccionarIngrediente(row){
    var svg = row.querySelector('svg');
    if(svg.hasAttribute('hidden')){
        svg.removeAttribute('hidden');
        var json = ObtenerCookie("IDs");
        if(json != null){
            var lista = JSON.parse(json);
            lista.seleccionados.push(row.getAttribute('data-id'));
            crearCookie("IDs",JSON.stringify(lista));
        }
    }else{
        svg.setAttribute('hidden','true');
        var json = ObtenerCookie("IDs");
        if(json != null){
            var lista = JSON.parse(json);
            lista = eliminarID(row.getAttribute('data-id'),lista);
            crearCookie("IDs",JSON.stringify(lista));
        }
    }
}

```

Figura B.9: Función para seleccionar el ingrediente de la lista

```

function ObtenerCookie(cookieName) {
    var cookieValue = null;
    var cookieString = document.cookie;
    var cookies = cookieString.split(';');

    for (var i = 0; i < cookies.length; i++) {
        var cookie = cookies[i].trim();

        if (cookie.startsWith(cookieName + '=')) {
            cookieValue = cookie.substring(cookieName.length + 1);
            break;
        }
    }

    return cookieValue;
}

```

Figura B.10: Función para obtener la cookie

```
function crearCookie(nombre, valor, diasExpiracion) {
    var fechaExpiracion = new Date();
    fechaExpiracion.setDate(fechaExpiracion.getDate() + diasExpiracion
    );
    var cookieString = nombre + '=' + valor + '; expires=' +
        fechaExpiracion.toUTCString() + '; path=/';
    document.cookie = cookieString;
}
```

Figura B.11: Función para crear la cookie

```
function eliminarID(id, lista){
    var indice = lista.seleccionados.indexOf(String(id));
    if (indice !== -1) {
        lista.seleccionados.splice(indice, 1);
    }
    return lista
}
```

Figura B.12: Función para eliminar un ingrediente de seleccionados

```
function eliminarID(id, lista){
var indice = lista.seleccionados.indexOf(String(id));
if (indice !== -1) {
    lista.seleccionados.splice(indice, 1);
}
return lista
}
```

Figura B.13: Función para eliminar un ingrediente de seleccionados

B.2. Framework

```

class Ingredients(models.Model):
    id = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=255)

    class Meta:
        db_table = 'ingredient'

class Recipes(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255)
    steps = models.TextField()
    number_of_ingredients = models.IntegerField(null=True)

    class Meta:
        db_table = 'recipes'

class RecipeIngredient(models.Model):
    recipe_id = models.ForeignKey(Recipes, on_delete=models.CASCADE)
    ingredient_id = models.ForeignKey(Ingredients, on_delete=models.
        CASCADE)

    class Meta:
        db_table = 'recipeingredient'
        constraints = [
            models.UniqueConstraint(fields=['recipe_id', ,
                'ingredient_id'], name='unique_recipe_ingredient'),
        ]

```

Figura B.14: Fichero con los modelos de

```

from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
    path("buscar/", views.buscar, name="buscar_resultados"),
    path("resultados/", views.resultados, name="resultados"), #
        Endpoint para buscar recetas
    path("receta/<int:id>", views.detalle_receta, name="detalle_receta"
        ), #Endpoint para buscar detalles de una receta
    path("guardadas/", views.guardadas, name="guardadas")
]

```

Figura B.15: Fichero con los endpoint de

```
def index(request):
    """
    Endpoint para el home
    """
    context = {
        "top": buscador.getRandom(5)
    }
    return render(request, 'buscador.html', context)
```

Figura B.16: Punto de acceso a la página principal

```
def buscar(request):
    """
    Endpoint para buscar los ingredientes en base a una query
    """
    resultados_lista = []
    if(request.method == 'GET'):
        query = request.GET.get('query', '')
        resultados = buscador.buscarIngredienteNombre(query)
        for resultado in resultados:
            resultado_dict = {
                'id': resultado[0],
                'nombre': resultado[1]
            }
            resultados_lista.append(resultado_dict)
    return JsonResponse({'resultados': resultados_lista})
```

Figura B.17:

```

def resultados(request):
    """
    Endpoint para buscar recetas en base a los ingredientes
    seleccionados
    """
    if request.method == 'POST':
        metodo = request.POST.get("metodo")
        if request.COOKIES.get("recetas"):
            guardadas_json = json.loads(request.COOKIES.get(
                "recetas"))
            guardadas = [eval(i) for i in guardadas_json]
        else:
            guardadas = []
        if metodo == "ingredientes":
            if request.COOKIES.get("IDs"):
                seleccionados_json = json.loads(request.
                    COOKIES.get("IDs"))
                if seleccionados_json:
                    IDs = list(map(int, seleccionados_json[""
                        seleccionados"]))
                    lista_recetas = buscador.
                        buscarRecetasIngredientes(IDs)
                    recetas = []
                    for receta in lista_recetas:
                        recetas.append((receta[0], receta[1],
                            receta[2], receta[0] in guardadas))
                    context = {
                        "top": buscador.getRandom(5),
                        "resultados": recetas
                    }
                else:
                    context = {
                        "top": buscador.getRandom(5)
                    }
            else:
                return redirect("index")
        if metodo == "recetas":
            query = request.POST.get('search-recetas')
            lista_recetas = buscador.buscarRecetasNombre(query
                )
            recetas = []
            for receta in lista_recetas:
                recetas.append((receta[0], receta[1], receta[2],
                    receta[0] in guardadas))
            context = {
                "top": buscador.getRandom(5),
                "resultados": recetas
            }
            return render(request, 'resultados.html', context)
        else:
            return redirect("index")
    
```

Figura B.18: endpoint para buscar recetas por ingredientes

```
def getRandom(numero : int):
    return buscador.getRandom(numero)
```

Figura B.19: Método para buscar recetas aleatorias

```
def detalle_receta(request, id):
    """
    Endpoint para ver los detalles de una receta
    """
    try:
        receta = buscador.buscarRecetasID(id)
        formateado = re.sub("Paso (\d+)\n\n", "", receta[2]).split(";")
        IDs = []
        if request.COOKIES.get("recetas"):
            guardadas_json = json.loads(request.COOKIES.get("recetas"))
            IDs = [eval(i) for i in guardadas_json]

        saved = id in IDs
        context = {
            "receta_id": receta[0],
            "receta_nombre": receta[1],
            "receta_pasos": formateado,
            "receta_ingredientes": receta[4],
            "saved": saved
        }
        return render(request, 'receta.html', context)
    except:
        return redirect("index")
```

Figura B.20: endpoint para ver detalles de una receta

```
def guardadas(request):
    """
    Endpoint para ver las recetas guardadas
    """
    context = {}
    if request.COOKIES.get("recetas"):
        guardadas_json = json.loads(request.COOKIES.get(
            "recetas"))
        IDs = [eval(i) for i in guardadas_json]
        recipes = []
        for id in IDs:
            recipe = buscador.buscarRecetasID(id)
            saved = recipe[0] in IDs
            recipes.append((recipe[0], recipe[1], recipe[4],
                            saved))

        context = {
            "top": buscador.getRandom(5),
            "resultados": recipes
        }

    return render(request, 'resultados.html', context)
```

Figura B.21: endpoint para ver las recetas guardadas del usuario

Apéndice C

Pruebas

```

def setUp(self) -> None:
    self.buscador = Buscador()
    self.factory = RequestFactory()
    recipe1 = Recipes(name="test1", steps="test1")
    recipe2 = Recipes(name="test2", steps="test2")
    recipe3 = Recipes(name="test3", steps="test3")
    recipe4 = Recipes(name="test4", steps="test3")
    recipe5 = Recipes(name="test5", steps="test5")
    recipe1.save()
    recipe2.save()
    recipe3.save()
    recipe4.save()
    recipe5.save()
    ingredient1 = Ingredients(id=0, name="test1")
    ingredient1.save()
    contains = RecipeIngredient(recipe_id=recipe2,
                                 ingredient_id=ingredient1)
    contains.save()

def test_call_view_index(self):
    response = self.client.get(reverse("index"))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'buscador.html')

def test_call_view_resultados_no_exists(self):
    response = self.client.get(reverse("resultados"), follow=True)
    self.assertRedirects(response, '/')

def test_call_view_resultados(self):
    self.client.cookies['IDs'] = []
    response = self.client.post(reverse("resultados"), data={'',
                                                             'metodo': 'ingredientes'})
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'resultados.html')

def test_call_view_guardadas(self):
    request = self.client.get(reverse("guardadas"))
    self.assertEqual(request.status_code, 200)
    self.assertTemplateUsed(request, 'resultados.html')

def test_call_view_detalle(self):
    request = self.client.get(reverse("detalle_receta", args
                                     ="1"))
    self.assertEqual(request.status_code, 200)
    self.assertTemplateUsed(request, 'receta.html')

def test_data_in_detalle(self):
    request = self.client.get(reverse("detalle_receta", args
                                     ="1"))
    self.assertTemplateUsed(request, 'receta.html')
    self.assertEqual(request.context['receta_id'], 1)

```

```
1     from locust import HttpUser, task
2     class MyLocustUser(HttpUser):
3
4         def __init__(self, *args, **kwargs):
5             super().__init__(*args, **kwargs)
6             self.csrf_token = None
7
8         def on_start(self):
9             response = self.client.get("/")
10            self.csrf_token = response.cookies[""
11                csrf_token"]
12            print(self.csrf_token)
13
14        def post_with_csrf_token(self, url, data):
15            headers = {"X-CSRFToken": self.csrf_token}
16            response = self.client.post(url, headers=
17                headers, data=data)
18            return response
19
20        @task
21        def resultado(self):
22            self.on_start()
23            cookie_value = json.dumps({"seleccionados":
24                ["0", "1", "2", "3", "4", "5", "6", "7"
25                ]})
26            self.client.cookies.set("IDs", cookie_value
27                )
28            response = self.post_with_csrf_token("/
29            resultados/", {'metodo': "ingredientes"})
30
31            # Valida la respuesta
32            if response.status_code == 200:
33                print("validada")
34
35        @task
36        def detalle(self):
37            self.on_start()
38            self.client.get("/receta/44")
```

Figura C.2: Script para la prueba de esfuerzo de Locust

Glosario

ACL Access Control List. Lista de control de acceso que permite definir permisos de acceso a recursos.

AJAX Asynchronous JavaScript and XML. Técnica que permite a las páginas web actualizar su contenido de forma asíncrona.

API Application Programming Interface. Conjunto de funciones y datos que permiten a los desarrolladores interactuar con un sistema.

AWS Amazon Web Services. Plataforma de computación en la nube de Amazon.

backend Parte de una aplicación web que se ejecuta en el servidor.

Base de datos Repositorio de datos estructurados que se utilizan para almacenar información.

CDN Content Delivery Network. Red de distribución de contenido que almacena y sirve contenido web desde servidores distribuidos.

cherry-picking Práctica de seleccionar cambios específicos de una rama de código para fusionarlos con otra.

CIDR Classless Inter-Domain Routing. Método de direccionamiento IP que permite asignar bloques de direcciones IP más grandes.

commit Cambio de código que se realiza en un repositorio.

cookie Pequeña pieza de información que un servidor web almacena en el navegador del usuario.

crawler Programa que navega por la web para recopilar información.

CSRF Cross-Site Request Forgery. Ataque que permite a un atacante realizar acciones en nombre de un usuario sin su conocimiento.

CSS Cascading Style Sheets. Lenguaje de diseño para páginas web.

dataset Conjunto de datos estructurados.

Design Thinking Metodología de diseño centrada en las necesidades del usuario.

django Framework web para Python.

DNS Domain Name System. Sistema de nombres de dominio que traduce los nombres de dominio en direcciones IP.

E2 Elastic Compute Cloud. Servicio de Amazon Web Services que proporciona instancias de servidores virtuales.

ELB Elastic Load Balancing. Servicio de Amazon Web Services que distribuye el tráfico entre varias instancias de EC2.

endpoint Punto final de una API.

event listener Función que se ejecuta cuando ocurre un evento.

flask Microframework web para Python.

framework Biblioteca o conjunto de herramientas que proporciona una estructura para el desarrollo de software.

frontend Parte de una aplicación web que se ejecuta en el navegador del usuario.

Git Sistema de control de versiones distribuido.

GitHub Plataforma de alojamiento de repositorios de código.

GitLab Plataforma de alojamiento de repositorios de código con funciones de gestión de proyectos.

HTML HyperText Markup Language. Lenguaje de marcado para la creación de páginas web.

HTTP HyperText Transfer Protocol. Protocolo de comunicación que se utiliza para transferir datos entre un servidor web y un cliente.

IaaS Infrastructure as a Service .Modelo de computación en la nube que permite a las empresas alquilar recursos informáticos, como servidores, almacenamiento y redes, a un proveedor de servicios en la nube.

interfaz Espacio de comunicación entre dos sistemas o usuario-máquina.

issue Problema o error que se encuentra en un software.

Jira Software de gestión de proyectos y tareas.

JQuery Librería JavaScript para facilitar el desarrollo de AJAX.

JSON JavaScript Object Notation. Formato de datos ligero basado en JavaScript.

middleware Software que se ejecuta entre el servidor web y la aplicación web.

milestone Objetivo o hito importante en el desarrollo de un proyecto.

mockup Prototipo de un diseño que se utiliza para probar la funcionalidad y el aspecto de un producto o servicio.

Model-View-Controller Model-View-Controller. Patrón de diseño que divide una aplicación web en tres componentes: modelo, vista y controlador.

on-premises Se refiere a los recursos informáticos que se encuentran en las instalaciones de una empresa.

ORM Object Relational Mapping. Arquitectura que permite a los desarrolladores interactuar con bases de datos relacionales utilizando objetos.

Poetry Gestor de paquetes para Python.

Pull Request Solicitud para fusionar cambios de un repositorio de código a otro.

python Lenguaje de programación interpretado de alto nivel.

RDS Relational Database Service. Servicio de Amazon Web Services que proporciona bases de datos relacionales.

render Método que se utiliza para generar la respuesta HTML de una vista Django.

script Código que se ejecuta en el navegador del usuario.

Secure Shell Secure Shell .Protocolo de red seguro que permite conectarse a un servidor de forma remota.

SEO Search Engine Optimization. Conjunto de técnicas para mejorar el posicionamiento de un sitio web en los resultados de búsqueda.

singleton Diseño de patrón que garantiza que solo exista una instancia de una clase.

Structured Query Language Structured Query Language. Lenguaje de consulta estructurado para bases de datos relacionales.

SVG Scalable Vector Graphics. Formato de imagen vectorial escalable.

tag Etiqueta que define el contenido y el comportamiento de un elemento en una página web.

test Proceso de verificar el funcionamiento correcto de un software.

tupla Colección de datos inmutable.

VPC Virtual Private Cloud. Red virtual aislada que se ejecuta en la nube.

VPN Virtual Private Network. Red privada virtual que permite conectar dispositivos de forma segura a través de Internet.

WSGI Web Server Gateway Interface. Interfaz que permite a los servidores web interactuar con frameworks web.

XSS Cross-Site Scripting. Inyección de código malicioso en una página web.

