UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCIÓN - CHILE

"DESARROLLO DEL BACKEND PARA LA APLICACIÓN MÓVIL *NOT WASTE*"

MARTIN LUCIANO FONSECA RUBIO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO EN INFORMÁTICA

Profesor Guía: José Luis Carrasco Sáez

Enero - 2026

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☑ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Desarrollo del backend para la aplicación móvil Not Waste

Nombre del candidato(a): Martin Luciano Fonseca Rubio

Carrera / Grado: Ingeniería en Informática

Campus: Concepción **Departamento: Departamento de Electrónica e Informática.**

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, **José Luis Carrasco Sáez**, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☑ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis: José Luis Carrasco Sáez

Fecha: 09/01/2026 **Firma:** 

Estudiante o Candidato(a): Martín Luciano Fonseca Rubio

Fecha: 09/01/2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Dedicado a mi persona que de alguna forma se las arregló para terminar esta memoria y a mi familia siempre me ha apoyado en todos los aspectos de mi vida.

AGRADECIMIENTOS

Tomando en cuenta en que este apartado está dedicado a que el alumno escritor del documento de sus agradecimientos voy a ignorar una de las instrucciones para la escritura de este documento y procederé a hablar en primera persona y a escribir no de una forma tan formal como en el resto del texto, pero aun así siguiendo estándares de decencia al hablar. Hola soy Martin Fonseca, el estudiante (al momento de escribir esta página) que escribió toda esta memoria, escribir esta parte se siente extraño debido a que, aunque tengo mucho por agradecer a mucha gente, aun así, no quiero hacerlo no porque me cueste expresarme o algo parecido (cosa que, si sucede hablando, pero escribiendo disminuye) si no porque la gente recordar y me lo mencionará y me dará vergüenza, pero bueno me tengo que obligar a mí mismo a escribir esto.

Para empezar, agradecer a mi familia que ha cuidado y me ha ayudado durante todos mis años de vida, a mi profesor guía el cual fue el encargado de revisar varios aspectos de esta memoria y guiarme en la escritura de esta , a mis amigos con los cuales disfrute el tiempo que pasamos juntos en todos estos años que los conozco y finalmente le tengo que agradecer a mis compañeros de trabajo de *Not Waste*, que los considero amigos, pero aun así quiero separarlos del otro grupo por agradecerles por todo el esfuerzo hecho para todo lo que involucro este proyecto, a toda la gente mencionada en este párrafo, muchas gracias por todo lo hecho por mi persona.

Como palabras finales de este apartado tengo que indicar que dejé esta sección para lo último así que, aunque sigue siendo la tercera página de la memoria, esto al mismo tiempo se puede considerar el final del documento, así que los dejo para que disfruten su lectura y en caso de que no la disfruten, no sé qué quieres que te diga sigue siendo una memoria, ve a ver una serie ,jugar algún juego o simplemente ir a caminar , no me extrañaría que fuera más disfrutable que leer un documento hecho por mí, pero en fin me despido y al mismo tiempo los saludo ya que ahora toca el resto de la memoria así que eso *chao*.

PD: ¿Entiendo que una memoria tenga que seguir un formato y todo, pero no les parece un poco exagerado limitar los agradecimientos a una página?, quiero suponer que realmente si el estudiante se sobrepasa tampoco habría quejas, pero aun así realmente alguien debería considerar modificar eso.

RESUMEN

Resumen—El documento aquí presentado muestra el desarrollo del backend de la aplicación móvil "*Not Waste*", la cual busca solucionar el problema del desperdicio de alimentos. El proceso de elaboración se hizo usando la metodología scrum para la organización del equipo y para tener entregas incrementales de la aplicación.

Los resultados la construcción de la aplicación se demuestran en los últimos capítulos de la memoria donde se evidencia las opiniones de los posibles usuarios cuando se les presentó la APP para que tuvieran una primera impresión y dieran una retroalimentación. Estos resultados ayudan a tener una vista a futuro para el desarrollo de la aplicación.

Palabras Clave—*Backend*; *Api*; Alimento; Desperdicio; *Flutter*.

ABSTRACT

Abstract— *The document presented here shows the development of the backend of the mobile application "Not Waste", which aims to address the problem of food waste. The development process was carried out using the Scrum methodology for team organization and to ensure incremental deliveries of the application.*

The results of the application's construction are demonstrated in the final chapters of this report, where the opinions of potential users are presented after being introduced to the app, allowing them to provide first impressions and feedback. These results contribute to a forward-looking perspective on the continued development of the application.

Keywords— *Backend; Api; Food; Waste; Flutter.*

GLOSARIO

- *Api*: *Application Programming Interface.*
- *APP: Application.*
- *AWS*: *Amazon Web Service.*
- *BDD: Base De Datos.*
- *DBMS: Data Base Managment System.*
- *EC2*: *Elastic Compute Cloud.*
- *HTTP: Hypertext Transfer Protocol.*
- *Id:* identificador.
- *IDE: Integrated Development Enviroment.*
- *JSON: JavaScript Object Notation.*
- *PC*: Personal Computer.
- RAE: Real Academia Española.
- *RDS*: *Relational Database Service.*
- *URL: Uniform Resource Locators.*
- *UTFSM*: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN 4

ABSTRACT 4

ÍNDICE DE TABLAS 14

INTRODUCCIÓN 15

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA 16

1.1 Contexto 16

1.2 Problema 16

1.3 Investigaciones personales 17

1.4 Solución 19

1.5 Objetivo de la tesis 20

1.5.1 Objetivo principal 20

1.5.2 Objetivos específicos 20

CAPÍTULO 2: MARCO CONCEPTUAL 21

2.1 Etapas del proyecto 21

2.1.1 Investigación inicial 21

2.1.2 investigación de grupo 21

2.1.3 Planificación de proyecto 22

2.1.4 Creación de proyecto 22

2.1.5 Verificaciones finales del proyecto 22

2.1.6 Presentación final 22

2.2 Metodología 22

2.2.1 Definición 22

2.2.2 Tipos de metodología 22

2.2.2.1 Metodología tradicional 23
2.2.2.2 Metodología ágil 23
2.2.3 Metodología elegida 23
2.2.3.1 Scrum 24
2.2.3.2 Uso de *Scrum* 25
2.2.3.2.1 Roles 25
2.2.3.2.1.1 Product Owner 25
2.2.3.2.1.2 Scrum Master 25
2.2.3.2.1.3 Equipo *Scrum* 25
2.2.3.2.2 Sprints 25
2.3 Not Waste 26
2.4 Modelo de datos de *Not Waste* 26
2.5 Arquitectura de *Not Waste* 27
2.6 Backend 28
2.6.1 Trabajo en el *backend* 29
2.6.1.1 Planificación de la funcionalidad 29
2.6.1.2 Codificación la funcionalidad 30
2.6.1.3 *Test* unitario de la función 30
2.6.1.4 *Test* de compatibilidad con el resto de las funciones 30
2.7 Lenguajes y herramientas para el desarrollo del *backend* 30
2.7.1 Api 30
2.7.2 Herramientas 31
2.7.2.1 Visual studio code 31
2.7.2.2 Postman 31

2.7.2.3 pgAdmin 32
2.7.3 Lenguajes 32
2.7.3.1 Framework 32
2.7.3.2 Dart 32
2.7.3.3 Flutter 32
2.7.3.4 Python 32
2.7.3.5 Django 32
2.7.3.6 SQL 32
2.7.4 Alojamiento de los servicios 33
2.7.4.1 Explicación 33
2.7.4.2 Servicios escogidos 33
2.7.4.2.1 Amazon Relational Database Service (Amazon RDS) 33
2.7.4.2.2 Amazon Elastic Compute Cloud (Amazon EC2) 34
2.7.4.2.3 Amazon Simple Storage Service (Amazon S3) 34
2.8 *Api* externa 34
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN 35
3.1 Solución 35
3.2 Funcionalidades de la aplicación 35
3.2.1 Ingreso de productos en la aplicación 35
3.2.2 Inventario 36
3.2.3 Recetas 36
3.2.4 Tienda 36
3.3 Consideraciones que tomar en cuenta dentro de la memoria 36
3.3.1 Base de datos 36

3.3.2 Funcionamiento de la *Api* 37
3.4 Base de datos 37
3.5 Funcionamiento de la *Api* 39
3.5.1 Explicación de cómo funciona la Api 39
3.5.2 Formato *JSON* 41
3.5.3 Acciones o métodos 41
3.5.4 Url 42
3.5.5 Respuesta 43
3.5.6 Funcionalidades en la *Api* 45
3.5.6.1 Ingreso de productos 45
3.5.6.1.1 Ingreso manual 45
3.5.6.1.1.1 Explicación 45
3.5.6.1.1.1.1 Funcionamiento 46
3.5.6.1.2 Ingreso con código de barras 47
3.5.6.1.2.1 Explicación 47
3.5.6.1.2.2 *Api* de *Open Foods Facts* 48
3.5.6.1.2.3 Funcionamiento 48
3.5.6.2 Inventario 50
3.5.6.2.1 Obtener productos del inventario 50
3.5.6.2.1.1 Explicación 50
3.5.6.2.1.2 Funcionamiento 51
3.5.6.2.2 Ver información de un producto 52
3.5.6.2.2.1 Explicación 52
3.5.6.2.2.2 Funcionamiento 52

3.5.6.3 Recetas 54
3.5.6.3.1 Obtener recetas 54
3.5.6.3.1.1 Explicación 54
3.5.6.3.1.2 Funcionamiento 54
3.5.6.3.2 Obtener ingredientes de las recetas 55
3.5.6.3.2.1 Explicación 55
3.5.6.3.2.2 Funcionamiento 55
3.5.6.4 Tienda 56
3.5.6.4.1 Ver publicaciones de la tienda 57
3.5.6.4.1.1 Explicación 57
3.5.6.4.1.2 Funcionamiento 57
3.5.6.4.2 Crear publicación 58
3.5.6.4.2.1 Explicación 58
3.5.6.4.2.2 Funcionamiento 58
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN 60
4.1 Validación 60
4.2 Flujo de pruebas 60
4.3 *Tests* unitarios 62
4.3.1 Cantidad de *tests* 62
4.3.2 Validaciones revisadas en las pruebas 62
4.3.3 Respuesta esperada 63
4.3.4 Agregar manualmente 63
4.3.4.1 Caso exitoso – Datos correctos 63
4.3.4.2 Caso fallido – Datos incorrectos 64

4.3.5 Agregar productos usando el código de barra 65
4.3.5.1 Caso exitoso - Código de barras correcto 65
4.3.5.2 Caso fallido - Código de barras no registrado en *Open* 66
4.3.5.3 Caso fallido – Código de barras faltante 66
4.3.6 Crear publicaciones 67
4.3.6.1 Caso correcto – información correcta 67
4.3.6.2 Caso incorrecto – información incorrecta 68
4.3.7 Resultados 69
4.4 *Test* de compatibilidad con el resto de las funciones 69
4.4.1 Cantidad de casos 69
4.4.2 Prueba exitosa 69
4.4.3 Agregar alimento y ver alimentos en el inventario 70
4.4.4 Agregar producto y ver recetas 70
4.4.5 Crear publicación y ver publicaciones 71
4.4.6 Resultados 72
4.5 Resumen de las pruebas 72
CAPÍTULO 5: CONCLUSIONES 73
5.1 Conclusiones respecto a *Not Waste* 73
5.2 Conclusiones respecto a objetivos 73
5.3 Limitaciones 74
5.4 Trabajo futuro 74
REFERENCIAS BIBLIOGRÁFICAS 76

ÍNDICE DE FIGURAS

Figura 1: Hallazgos de la investigación. Fuente: Maggi e Ipsos 17

Figura 2: Gráfico edad-género. Fuente: Grupo *Beholders* 18

Figura 3: Razón para tirar comida. Fuente: Grupo *Beholders* 18

Figura 4: Preocupación frente al desperdicio de alimentos. Fuente: Grupo Beholders 19

Figura 5: *Sprints* de la aplicación *Not Waste*. Fuente: Grupo *Beholders* 26

Figura 6: Tareas claves. Fuente: Elaboración propia. 26

Figura 7: Modelo de datos. Fuente: Elaboración propia. 27

Figura 8: Arquitectura de la aplicación Not Waste. Fuente: Elaboración propia. 28

Figura 9: Ciclo trabajo en el backend. Fuente: Elaboración Propia. 29

Figura 10: Logo de *Not Waste* Fuente: Grupo *Beholders* 35

Figura 11: Ejemplo de una tabla dentro de una base de datos. Fuente: Elaboración propia 36

Figura 12: Esquema relacional de la base de datos de *Not Waste*. Fuente: Elaboración propia 38

Figura 13: Funcionamiento de la Api en forma visual. Fuente: Elaboración propia. 40

Figura 14: Sección de la respuesta de la Api de Open Foods Facts en formato JSON Fuente: Elaboración Propia 41

Figura 15: Proceso de ingreso manual a la aplicación. Fuente: Elaboración propia. 47

Figura 16: Sección de la respuesta de Open Foods Facts. Fuente: Elaboración propia. 48

Figura 17: Proceso del ingreso con código de barra. Fuente: Elaboración propia. 50

Figura 18: Proceso de obtener productos del inventario. Fuente: Elaboración propia. 52

Figura 19: Proceso ver información de un alimento. Fuente: Elaboración propia. 53

Figura 20: Proceso de obtener recetas. Fuente: Elaboración propia. 55

Figura 21: Proceso de obtener los ingredientes de la receta. Fuente: Elaboración propia. 56

Figura 22: Proceso de ver publicaciones del usuario. Fuente: Elaboración propia. 58

Figura 23: Proceso de creación de una publicación. Fuente: Elaboración propia. 59

Figura 24: Flujo de pruebas. Fuente: Elaboración propia. 61

Figura 25: Caso correcto - información correcta. Fuente: Elaboración propia. 64

Figura 26: Caso fallido - Datos incorrectos. Fuente: Elaboración propia. 65

Figura 27: Caso exitoso - Barcode correcto. Fuente: Elaboración Propia 66

Figura 28: Caso fallido – Código de barras no incluida en *Open*. Fuente: Elaboración Propia. 66

Figura 29: Caso fallido - Código de barras faltante. Fuente: Elaboración propia..... 67

Figura 30: Caso correcto - información correcta. Fuente Elaboración propia. 68

Figura 31: Caso incorrecto - información incorrecta. Fuente: Elaboración propia. 69

Figura 32: Compatibilidad Agregar alimento - Ver inventario. Fuente: Elaboración propia. 70

Figura 33: Compatibilidad Agregar producto - Ver recetas. Fuente: Elaboración propia... 71

Figura 34: Compatibilidad Crear recetas - Ver recetas. Fuente: Elaboración propia..... 72

ÍNDICE DE TABLAS

Tabla 1: Tabla comparativa de metodologías. Fuente: Parafraseó de tabla hecha por Lecciones aprendidas 23

Tabla 2: Url en la *Api* de *Not Waste Fuente: Elaboración propia* 42

Tabla 3: Códigos de respuesta positiva. Fuente: Elaboración propia. 44

Tabla 4: Códigos de respuesta negativa. Fuente: Elaboración propia. 45

INTRODUCCIÓN

Teniendo una gran cantidad de producción de comida en el mundo, se produce un problema adyacente, el cual es: ¿qué hacer con los residuos de los alimentos? Al no tener un plan bien estructurado sobre qué hacer con los restos se genera el inconveniente que se intenta solucionar en esta memoria: El desperdicio de alimentos. Disponiendo de esta complicación en mente se creó *Not Waste,* una aplicación que busca ayudar a reducir la cantidad de alimento desperdiciado en los hogares. Está se creó y desarrolló usando una metodología ágil para tener entregas incrementales de la *APP* y principalmente por el hecho que la metodología también toma en consideración la opinión del usuario, también se usaron herramientas que permiten hacer más ágil el desarrollo de la aplicación.

Este documento presenta diferentes temas relacionados con el desarrollo del *backend* de *Not Waste*, estos temas abarcan desde el problema original hasta cual es el futuro de la aplicación, todo esto se escribió durante el año 2024. A continuación, se dará un breve resumen sobre los capítulos que se incluyen en esta memoria. Los primeros dos capítulos fueron escritos durante el transcurso del primer semestre del año 2024, el capítulo 1 nombrado “Definición del problema” mostrará todo el contexto de la problemática y el por qué se toma la decisión de crear la aplicación, el capítulo 2 llamado “Marco Conceptual” presenta la arquitectura de la *APP* incluyendo la explicación de las herramientas, lenguajes y metodologías usadas para la construcción de esta. Luego de estos dos primeros capítulos vienen los siguientes que fueron escritos durante la duración del segundo semestre del año 2024, estos son los últimos 3 de la memoria. El capítulo 3 “Propuesta de solución” abarca el cómo funciona la solución buscando resolver la problemática planteada en el primer capítulo, el capítulo 4 de nombre “Validación de la solución” trata sobre validar el correcto uso de la solución presentada haciendo uso de diferentes pruebas a las funcionalidades de la aplicación y el capítulo 5 llamado “Conclusiones” como su nombre indica trata de los pensamientos finales respecto del trabajo realizado y también se describe las consideraciones finales para su mejora y el futuro de la aplicación.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Contexto

En chile y en el mundo en general existe una gran producción de comida, esto debido a dos características en específico: una gran cantidad de población que es necesaria alimentar (Más de 8 mil millones de personas en el mundo (División de Población de las Naciones Unidas, 2024), y la otra razón la cual es todo el mercado que se genera relacionado al mercado de alimentos (Este en 2022 alcanzó un valor 147.700 millones (Statista, 2024)). Ahora con la gran producción viene un problema, el cual se trata en esta memoria, el cual es el desperdicio de estos. El desperdicio de alimentos es un problema global que es causado debido a múltiples razones entre las cuales se encuentran: el olvidar que se compró un alimento y que este llegue a su fecha de vencimiento, comprar una cantidad mayor de alimento de lo que se va a consumir, entre otras razones.

1.2 Problema

Ahora enfocándose solamente Chile, tomando como base un estudio hecho por las empresas Maggi e Ipsos (2023). En este se muestra que, de las 600 personas encuestadas, el 92% de estas, declara desperdiciar comida, ahora sin enfocarnos en la razón, el estudio declara que 9 de cada 10 hogares efectivamente desperdician alimento, esto solamente logra demostrar lo grave que es este problema. A continuación, se muestra un resumen de la investigación:

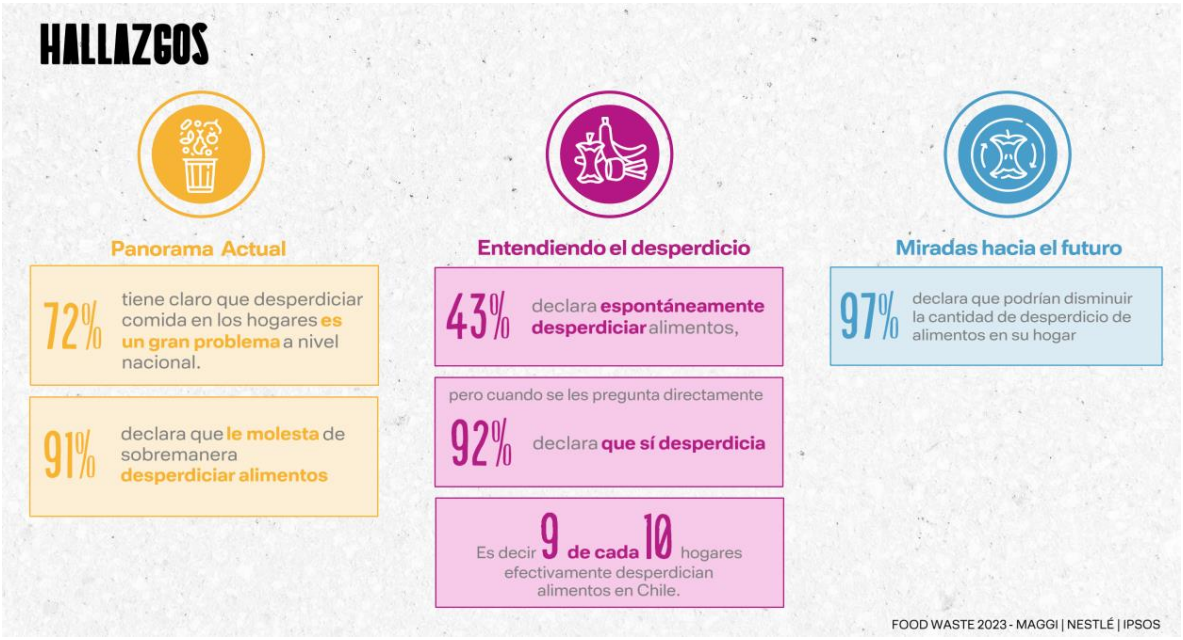

Figura 1: Hallazgos de la investigación.
Fuente: Maggi e Ipsos

La imagen anterior refleja un resumen con los hallazgos encontrados por la investigación mencionada anteriormente, con esta información se tiene que, aunque existe el problema de que la gente desperdicia alimento, al mismo tiempo estas mismas personas que producen el problema les molesta el desperdicio de este. Lo que evidencia que crear una solución para reducir el porcentaje de desperdicio es viable, considerando la inmensa cantidad de gente que podría llegar a interesarse en una solución para este problema.

1.3 Investigaciones personales

Ya con la problemática y la motivación planteada por una investigación inicial del problema, el grupo se dispuso a hacer indagaciones propias para conocer más el conflicto y tener más conocimiento de parte de las personas preocupadas por esta situación, todo esto con el objetivo de que se conozca más de primera mano la opinión de la gente con respecto al problema, él por qué terminan desechando comida y buscar el enfoque para la construcción de la solución. A continuación, se muestran los gráficos resultantes de la encuesta.

En esta siguiente imagen se muestra las edades y géneros de las personas encuestadas:

Figura 2: Gráfico edad-género.
Fuente: Grupo *Beholders*

En esta imagen se muestra el por qué la gente encuestada tira comida a la basura, en la cual se evidencia que la mayor cantidad de comida desperdiciada fue culpa en su mayor parte de comida que no se terminó consumiendo:

Figura 3: Razón para tirar comida.
Fuente: Grupo *Beholders*

Y ahora esta muestra la opinión de los encuestados frente al problema del desperdicio de alimentos, que refleja que, aunque existe gente que el problema no le causa mayor interés admite que realmente es un problema:

Figura 4: Preocupación frente al desperdicio de alimentos.
Fuente: Grupo Beholders

Teniendo toda esta información proporcionada por la encuesta, se pensó en los problemas que tiene la gente y se ideó ciertas soluciones que podrían funcionar para estos casos.

Entre las funciones que se plantean existen estas:

- Alarma o notificación para dar un aviso al usuario respecto a que se acerca la fecha de vencimiento de algún producto en específico.
- Tener un apartado en la aplicación donde aparezcan todos los alimentos que tiene el usuario.

1.4 Solución

Teniendo la información presentada por las empresas y ampliada mediante las investigaciones del grupo, se decidió enfocarse en este problema y crear su solución, esta terminó siendo la aplicación llamada *Not Waste.*

Not Waste es una aplicación móvil ideada con el objetivo de ayudar en la reducción del desperdicio de alimentos en los hogares. Esta solución busca solucionar el problema mediante varias funcionalidades que posee la aplicación.

1.5 Objetivo de la tesis

1.5.1 Objetivo principal

Desarrollar el *backend* de la aplicación móvil *Not Waste*, la cual busca cumplir con el objetivo de la reducción de comida usando una metodología ágil para entregas incrementales de la aplicación, y empleando los lenguajes de programación *Dart* y *Flutter* para la construcción de la aplicación *móvil* y la *Api*.

1.5.2 Objetivos específicos

Como objetivos específicos para desarrollar el *backend*, se propuso estos objetivos:

- Crear las funcionalidades de la aplicación.
- Realizar pruebas de las funcionalidades para la comprobación de que estas funcionen de la forma en la que se planificó.
- Verificar y corregir el código de la aplicación y de la *Api* de esta, para comprobar que cumplen con los estándares de código limpio.

CAPÍTULO 2: MARCO CONCEPTUAL

Para la creación de la aplicación es necesario primero que se defina las metodologías y herramientas necesarias usadas en la construcción de *Not Waste*.

2.1 Etapas del proyecto

El proyecto se dividió en 6 etapas principales las cuales se pueden apreciar mejor en la imagen a continuación:

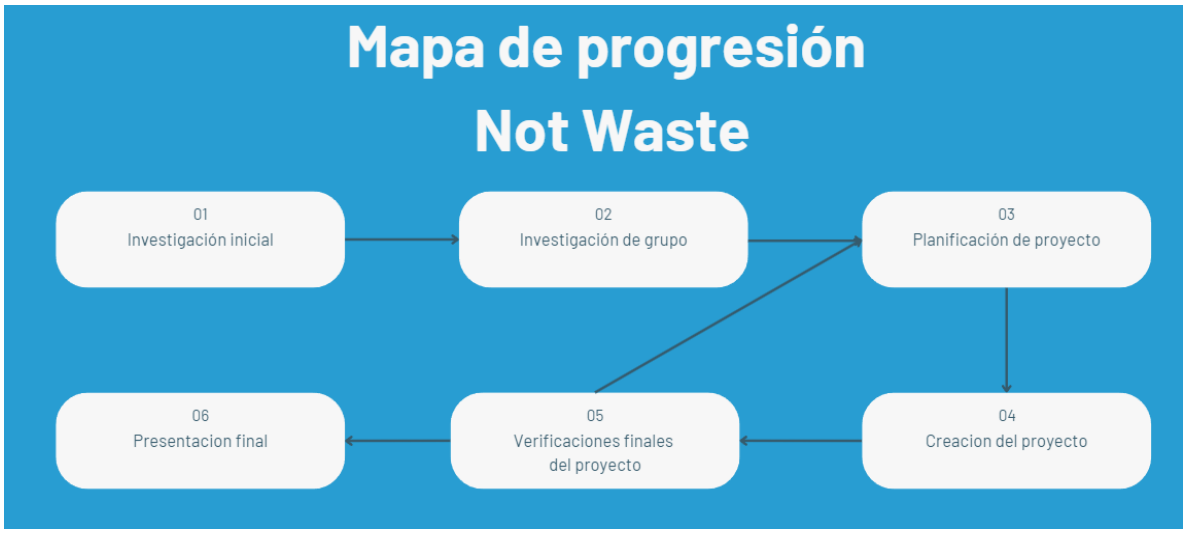

Figura 5: Mapa de progresión de *Not Waste*.
Fuente: Elaboración propia.

A continuación, se explicarán estas 6 etapas que, a excepción de la etapa final, se muestran a través de esta memoria.

2.1.1 Investigación inicial

Presentado en el capítulo 1, una investigación inicial hecha para elegir un problema el cual finalmente terminó siendo el desperdicio de alimento.

2.1.2 investigación de grupo

Presentado en el capítulo 1, investigaciones personales del grupo hechas para expandir el conocimiento adquirido durante la investigación inicial y empezar a pensar en una posible solución.

2.1.3 Planificación de proyecto

Presentado durante el capítulo 1 y 2, en esta etapa se decide por una solución y se empieza a planificar que necesita tener esta solución para ayudar con el problema, con que herramientas se creara y planificar el desarrollo de esta.

2.1.4 Creación de proyecto

Presentado durante el capítulo 3, en esta etapa se crea el proyecto siguiendo las directrices propuestas durante la etapa anterior.

2.1.5 Verificaciones finales del proyecto

Presentado durante el capítulo 4, dado que se completó la solución en la etapa anterior, en esta sección toca validar la propuesta a través de diversos *tests* para comprobar el rendimiento de esta y además validar con gente externa del desarrollo del proyecto para comprobar las opiniones respecto a la solución.

2.1.6 Presentación final

Esta etapa no está presentada durante la memoria, pero igual es necesaria mencionarla considerando que es una parte importante del proyecto, dado que es la conclusión del proyecto donde se presentó la solución final a un jurado.

Cabe destacar que las etapas 3,4 y 5 funcionan de forma cíclica dado que se planifica una función, se desarrolla y luego se verifica así hasta que el proyecto quede en un buen estado para la etapa final. Ahora esto no siempre es así, ya que puede pasar que se plantee una función en la etapa 3 pasee al desarrollo al capítulo 4 y esa funcionalidad se descarte haciendo que vayamos a la etapa 3 nuevamente.

2.2 Metodología

2.2.1 Definición

La definición de metodología es “Conjunto de métodos o procedimientos que se usan para hacer algo” (RAE ,2024).

2.2.2 Tipos de metodología

Con respecto a los tipos de metodología existen dos que son las más conocidas, estas dos son la metodología tradicional y la ágil, las cuales se presentan a continuación:

2.2.2.1 Metodología tradicional

La metodología tradicional, también conocida como *waterfall (*o cascada en español) se basa en una serie de ciclos de vida los cuales son: Inicio, planificación, ejecución, seguimiento y cierre. El lado positivo que posee esta metodología es que gracias a que es necesario tener una gran cantidad de documentación del proyecto, es fácil definir dónde llega el límite de este, pero esto mismo genera una desventaja ya que la metodología tradicional es poco flexible y si fuera necesario un cambio, este provocaría problemas tales como un retraso (Ginzo Tech,2024).

2.2.2.2 Metodología ágil

Esta metodología usada principalmente en entornos de desarrollo de software tiene un enfoque flexible y adaptable en entregas incrementales. Esta metodología es flexible buscando la colaboración entre los miembros del equipo, las partes interesadas y el cliente. La metodología enfatiza en la entrega de un producto de calidad y la satisfacción del cliente. El lado positivo de esta es que, al ser usado mediante entregas pequeñas, esta metodología permite más flexibilidad que la anterior, ahora por el lado negativo es que al tener poca cantidad de documentación del proyecto es complicado calcular los límites y costos de este (*Red Hat*,2022).

2.2.3 Metodología elegida

Con los dos tipos de metodologías explicadas, se puede hacer una tabla comparativa para ver lo positivo y negativo de cada una de estas.

Tabla 1: Tabla comparativa de metodologías.
Fuente: Parafraseó de tabla hecha por Lecciones aprendidas

	Tradicional	Ágil
Flexibilidad al momento de desarrollar el proyecto.	Posee una menor flexibilidad.	Posee una mayor flexibilidad.
Documentación del proyecto.	Todo el proyecto tiene que documentarse desde el inicio.	Esta metodología al ser como es no se documenta todo desde un inicio y se hace por plazos.

Requisitos	Los requisitos tienen que detallarse desde el inicio del proyecto.	Los requisitos pueden cambiar a través del desarrollo del proyecto.
Costo	El costo del proyecto queda definido desde el inicio.	El costo del proyecto no se define desde un inicio lo que hace que luego puedan existir problemas monetarios como por ejemplo que se agote el presupuesto del proyecto.
Iteraciones	Pocas iteraciones que generan una gran cantidad de desarrollo del proyecto.	Bastantes que son usadas para la construcción y evolución del proyecto.

Tomando los puntos de comparación mostrados en la tabla se decidió que se usaría la metodología ágil, debido a la flexibilidad que ofrece esta metodología y a la gran cantidad de iteraciones que se hacen cuando se usa esta metodología, y para ser especifico se usará la metodología *Scrum*.

2.2.3.1 Scrum[1]

Esto funciona para el desarrollo de proyectos complejos, el cual se enfoca en la entrega incremental para cumplir con las necesidades del cliente. Esta metodología fue elegida para el desarrollo del proyecto debido a la flexibilidad que esta brinda y la participación que tiene el cliente dentro de la construcción de la aplicación, el cual en este caso serían los propios usuarios de la aplicación.

[1] Para más información se puede ingresar a su sitio web: https://www.scrum.org/

2.2.3.2 Uso de *Scrum*

Primero para el uso de *Scrum* fue necesario definir los roles principales de este, aunque existen roles compartidos debido a que finalmente todos los integrantes del grupo terminaron trabajando en el desarrollo de la aplicación.

2.2.3.2.1 Roles

2.2.3.2.1.1 Product Owner

Persona encargada de darle valor al proyecto, responsable de que el producto final sea lo mejor posible, esto se hace mediante la definición de características y en este caso dando una visión de cómo debería funcionar la aplicación.

2.2.3.2.1.2 Scrum Master

Persona encargada de que el equipo siga la metodología *Scrum* manteniendo al equipo enfocado en los principios de este, se encarga de organizar reuniones para la planificación del proyecto y comprobando el avance de este.

2.2.3.2.1.3 Equipo *Scrum*

Es el equipo de desarrollo que trabajan juntos en el desarrollo del proyecto, estos trabajan juntos para entregar los incrementos del producto final.

2.2.3.2.2 Sprints

Scrum se usa mediante *Sprints*, los cuales son periodos de tiempo cortos en los que el equipo scrum trabaja para entregar un incremento, en el caso del proyecto se definieron 3 de estos, los cuales se encuentran repartidos durante el año, los cuales se pueden apreciar en la imagen.

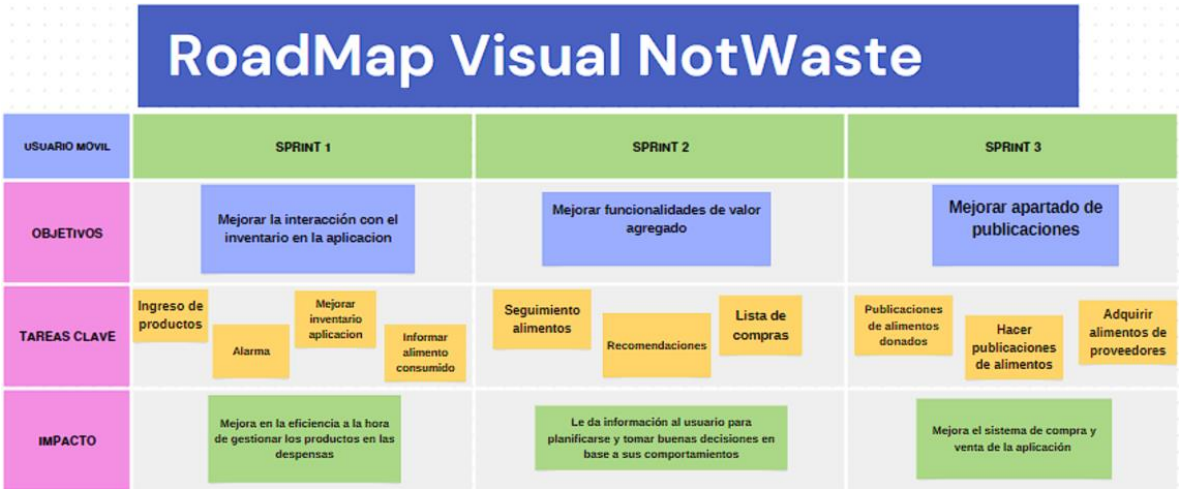

Figura 5: *Sprints* de la aplicación *Not Waste*.
Fuente: Grupo *Beholders*

Dentro de la memoria se hablará de las siguientes tareas claves mostradas en la figura 6, se hablará más a fondo de las funcionalidades relacionadas a estas tareas claves en el capítulo 3, las tareas claves mencionadas son las que están visibles, en contraste las tareas que están con un cuadrado rojo encima de ellas no se mencionan por motivos tales como: finalmente no se hicieron, se descartaron o eran tareas que no forman parte de lo mostrado durante el capítulo 3.

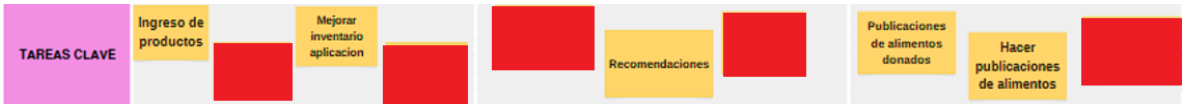

Figura 6: Tareas claves.
Fuente: Elaboración propia.

2.3 Not Waste

Not Waste es la aplicación móvil que se creó para dar una solución al problema propuesto durante el capítulo 1, el concepto de esto se explicara nuevamente a más profundidad dentro del capítulo 3, pero es necesario mencionarlo ahora debido a los siguientes subcapítulos.

2.4 Modelo de datos de *Not Waste*

Un modelo de base de datos muestra la estructura lógica de la base, incluida las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se

accede a ellos (Lucidchart,2024) o en palabras simples es un mapa con los datos y sus relaciones, este se usará luego para la construcción de la base de datos.

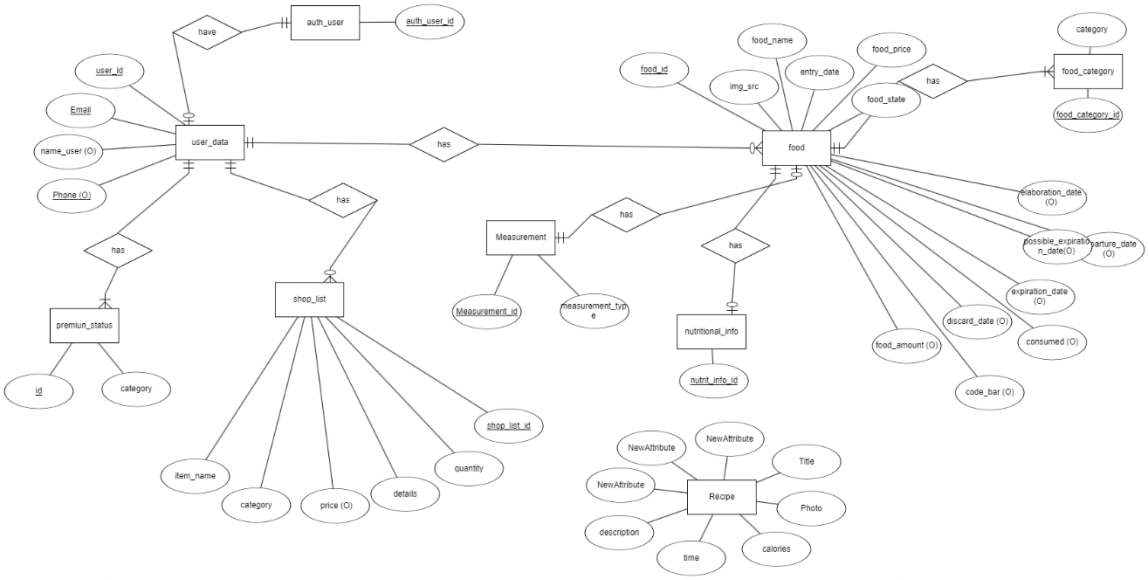

Figura 7: Modelo de datos.
Fuente: Elaboración propia.

2.5 Arquitectura de *Not Waste*

La palabra “arquitectura” en el contexto de la construcción de una aplicación se refiere a la estructura y componentes que se usan para el armado de ésta. Dentro de la arquitectura de la aplicación de *Not Waste*, lo importante para esta memoria son los componentes que se muestran en la fotografía:

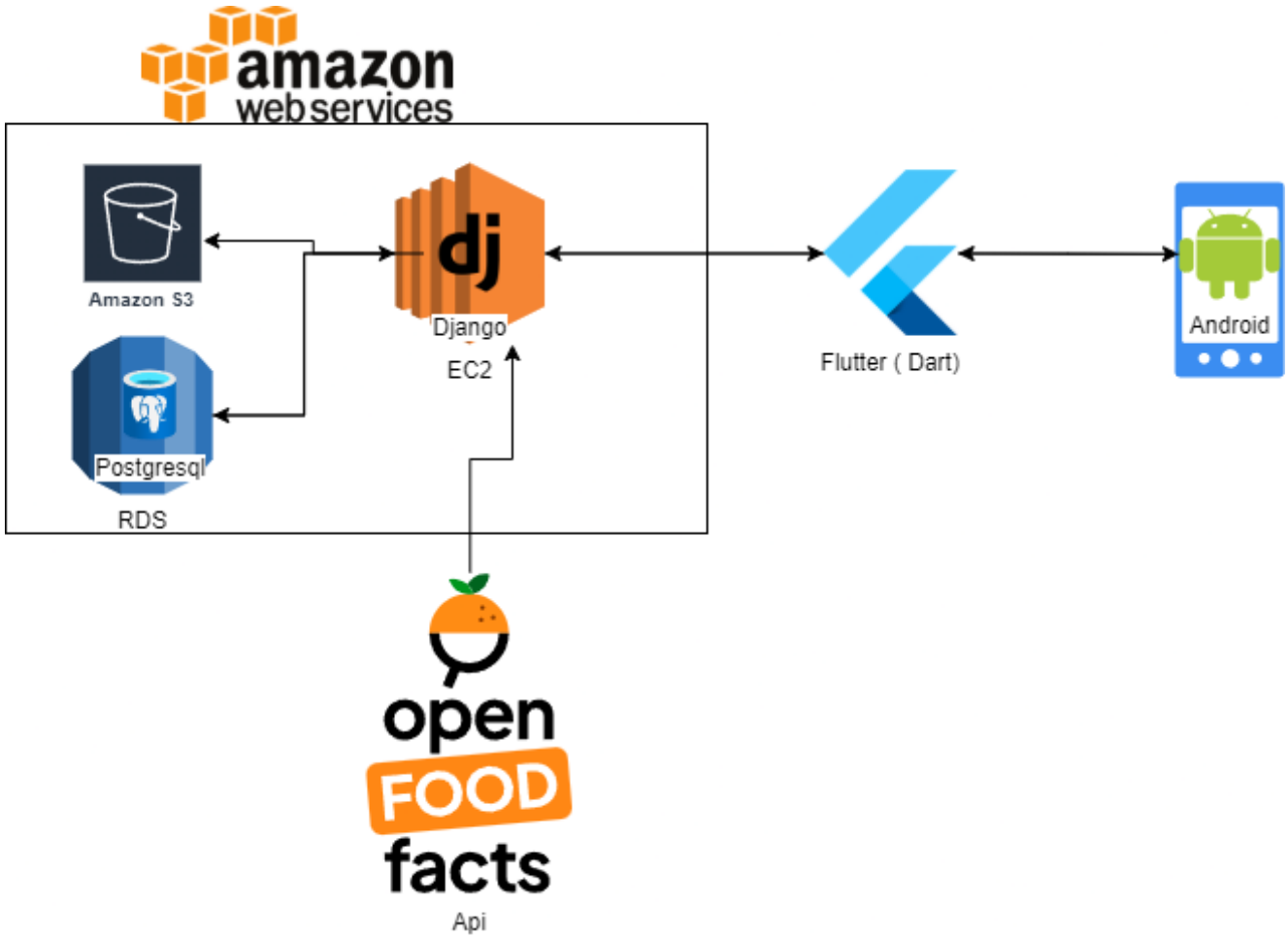

Figura 8: Arquitectura de la aplicación Not Waste.
Fuente: Elaboración propia.

La explicación de esta arquitectura es la siguiente: Se tiene la aplicación móvil para *Android* la cual se programó con el lenguaje de programación *Dart* en conjunto con el *framework Flutter,* esta aplicación se conecta con una *Api* que usa el *framework Django* la cual se encuentra funcionando gracias a en un *EC2*, esta *Api* se encuentra conectada a su vez con un *RDS* que usa la base de datos *PostgreSQL* y un *S3.* Está *Api*, RDS y *S3* se encuentran alojadas en los servicios *web* de Amazon. La *Api* de *Django* también se encuentra conectada con una Api externa para obtener información de los alimentos.

2.6 Backend

La palabra *backend* en el contexto de la memoria se refiere a cómo funciona la aplicación por detrás, el tema de lógica de *Not Waste* y que hace este para que las funcionalidades de la aplicación funcionen como es debido. Tomando la figura 1 se

podría separar en 2, la parte del backend se enfocaría en la parte izquierda (*EC2, S3, RDS, Api*).

2.6.1 Trabajo en el *backend*

Para la construcción en el backend se sigue el siguiente bucle:

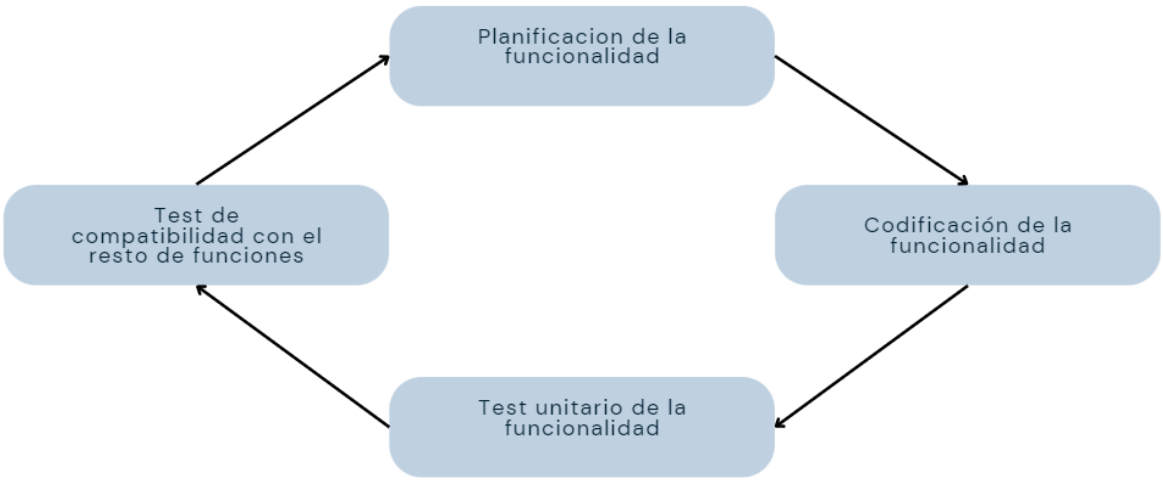

Figura 9: Ciclo trabajo en el backend.
Fuente: Elaboración Propia.

Este bucle se encuentra dividido en 4 secciones, es necesario que una etapa pase satisfactoriamente para continuar con la siguiente sección, estas se explicarán a continuación:

2.6.1.1 Planificación de la funcionalidad

Durante esta etapa se planifica la función, aquí se enfoca en una parte del problema y que lo puede solucionar teniendo 3 preguntas.

- ¿Qué quiere solucionar esta función? Cuál es el problema que quiere solucionar la función que se está planteando.
- ¿Qué va a hacer? Qué función se propone para solucionar el problema.
- ¿Cómo lo va a hacer? Pasos que debe hacer la función para lograr el "que" de la pregunta pasada.

Teniendo estar 3 preguntas se puede a la siguiente sección

2.6.1.2 Codificación la funcionalidad

Aquí se codifica la funcionalidad dentro de la aplicación y la *Api*, está se programa en base a lo planteado en la sección anterior.

2.6.1.3 *Test* unitario de la función

Se llaman pruebas o *test* unitarios porque descomponen las funciones del programa en comportamientos comprobables discretos que se pueden probar como unidades individuales (Microsoft,2023) en palabras simples dividir el programa completo en múltiples funcionalidades y hacer *test* para la comprobación de cada una de estas.

2.6.1.4 *Test* de compatibilidad con el resto de las funciones

Dado que la funcionalidad por si sola funciona de una forma correcta, es momento de comprobar que el resto de funcionan entre sí, dando ejemplo es que, si tenemos una funcionalidad de "crear usuario" y otra de "ver usuarios", aquí se comprueba si creando un usuario durante la primera funcionalidad este se ve reflejada en la ejecución de la segunda.

2.7 Lenguajes y herramientas para el desarrollo del *backend*

Ahora que está explicada la arquitectura general de la aplicación es necesario explicar cada herramienta y lenguaje que se mencionó para un mejor entendimiento de parte del lector.

2.7.1 Api

La Interfaz de Programación de Aplicaciones o *Api* es un conjunto de definiciones y protocolos que permite la conexión entre dos aplicaciones, que en este caso es la aplicación móvil con la base de datos. El uso de esta en el proyecto se fundamenta con estas 2 razones:

- Agilizar la creación de las funcionalidades de la aplicación mediante código dentro de la *Api.*
- Crear una barrera intermedia entre el usuario y la *BDD* debido a que no es seguro que el usuario tenga una conexión directa con esta.

Los siguientes puntos se dividen en 3 sectores herramientas, lenguajes y servicios *cloud*, todo esto se puede resumir en la siguiente imagen:

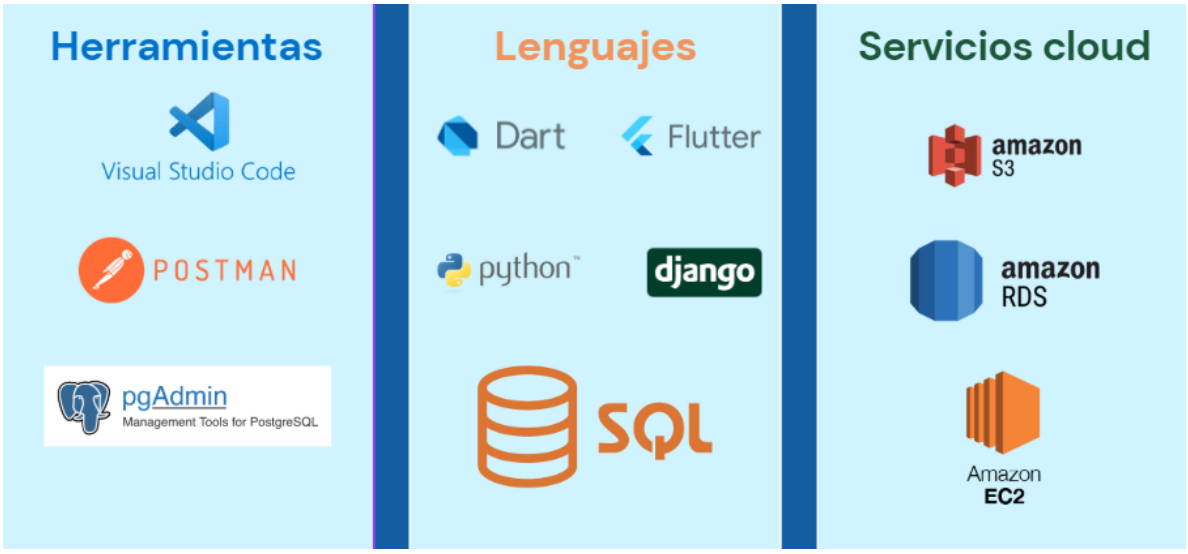

Figura: Herramientas, lenguajes y servicios cloud.
Fuente: Elaboración propia.

2.7.2 Herramientas

Empezando por el sector de las herramientas, aquí se exponen algunos programas usados durante la aplicación, aunque cabe mencionar que las nombradas a continuación son preferencias personales del estudiante que escribe esto y existen alternativas a estas.

2.7.2.1 Visual studio code

Este es un *IDE* o en español "entorno de desarrollo integrado". Este permite la ejecución y escritura del código de la aplicación y de la *Api,* gracias a ciertas extensiones que el programa permite que se instalen en este. Existen alternativas tales como: Pycharm y Intellij IDEA, programas que haciendo un par de configuraciones pueden usarse para el mismo objetivo.

2.7.2.2 Postman

Herramienta dedicada a la construcción y uso de *Api*. *Postman* tiene el uso en la aplicación de verificar el funcionamiento de las peticiones de la *Api* que usa el *backend* de la aplicación. Como alternativa para este programa existe: *Insomnia* y *Hoppscotch*.

2.7.2.3 pgAdmin

Esta herramienta permite gestionar la base de datos *PostgreSQL*, usando una interfaz visual para la gestión de la *BDD*. Como alternativas para esta aplicación existen: *DBeaver* y *DataGrip.*

2.7.3 Lenguajes

2.7.3.1 Framework

Antes de la explicación de los lenguajes que se utilizaron durante el desarrollo del *backend*, es necesario explicar que es un *framework.*

Framework es un conjunto de herramientas y estructuras que ayudan a organizar y desarrollar aplicaciones más eficientemente. Estos son usados en conjunto los lenguajes de programación proporcionando funcionalidades que ayudan en tareas específicas agilizando el tiempo de desarrollo del producto hecho.

2.7.3.2 Dart

Dart es un lenguaje de programación que se usa para aplicaciones web y móviles. Esté es el lenguaje principal que se utiliza en *Not Waste*.

2.7.3.3 Flutter

Es el *framework* usado en la aplicación, permite la creación de aplicaciones multiplataforma, este es usado en conjunto con *Dart* para la creación de la aplicación.

2.7.3.4 Python

Python es un lenguaje de programación usado para diferentes situaciones tales como desarrollo de software y análisis de datos.

2.7.3.5 Django

Django es un *framework* escrito en *Python* el cual usualmente se usa para la creación de Webs, a este hay que sumarle la librería: *Django Rest framework* el cual en esta situación es beneficioso ya que es un *toolkit* usado para la creación de la *Api* que utiliza la aplicación móvil para conectarse con la base de datos.

2.7.3.6 SQL

SQL (*Structured Query Language*) es un lenguaje usado en las bases de datos relacionales. Este lenguaje permite hacer operaciones con los datos que poseen

estas bases de datos, esto es usado en conjunto con pgAdmin para el manejo de la base de datos.

2.7.4 Alojamiento de los servicios

2.7.4.1 Explicación

En la sección de arquitectura se mencionó que se alojaría la *Api* y la base de datos en *AWS*. Así que es necesario explicar a qué se refiere con el alojar y qué es *AWS*.

Alojamiento o también conocido como *hosting* consiste en almacenar una aplicación o sitio web en un servicio de alojamiento web para que se pueda acceder a esta desde diferentes dispositivos vía internet.

AWS (Amazon Web Services[2]) es una sección de la empresa de Amazon, esta se encarga del alojamiento de servicios en la nube. Está empresa posee varios servicios que son de utilidad para la construcción y que la aplicación permanezca operativa luego de finalizada la construcción de esta.

Se decidió que se usará los servicios de alojamiento prestados por esta empresa para los componentes explicados en la arquitectura debido a dos razones, una mucho más importante que la otra. La primera razón la cual es importante, pero de menor relevancia, es debido a los beneficios que ofrece *AWS* en comparación con sus rivales, ahora la razón principal del por qué se eligió esta empresa, fue los costos que tiene los servicios que ofrece, estos son más razonables que la competencia.

2.7.4.2 Servicios escogidos

Aquí se explicará cuáles son los servicios escogidos, que son y cuál es su uso en el proyecto:

2.7.4.2.1 Amazon Relational Database Service (Amazon RDS)

Amazon Relation Database Service es un servicio de base de datos relacional fácil de administrar, optimizada para el costo total de propiedad (Amazon,2024) este servicio se usa para la base de datos funcionando en la nube.

[2] Para una mayor explicación de los servicios de *AWS* se puede dirigir a su página: https://aws.amazon.com/es/

2.7.4.2.2 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon Elastic Compute Cloud (Amazon *EC2*) proporciona capacidad de computación escalable bajo demanda en la nube de *Amazon Web Services* (Amazon,2024). Este servicio de *AWS* en palabras simples *son* máquinas virtuales[3], este servicio se eligió para *host* y el uso de las *Apis*.

2.7.4.2.3 Amazon Simple Storage Service (Amazon S3)

Amazon *S3* es un servicio de almacenamiento de objetos creado para almacenar y recuperar cualquier volumen de datos desde cualquier ubicación (Amazon,2024). Es un servicio de almacenamiento de objetos el cual es usado para guardar las imágenes que son ocupadas en la aplicación.

2.8 *Api* externa

Para la construcción de la aplicación *Not Waste* se usa una *Api* externa propiedad de "*Open Foods Facts"*.

Open Food Facts es una base de datos de productos alimenticios con ingredientes, alérgenos, información nutricional y todos los detalles que podemos encontrar en las etiquetas de los productos (Open Foods Facts,2024).

Esta *Api*[4] toma el código de barras de un producto alimenticio y regresa diversa información respecto al producto. Se dará más contexto con respecto a la información pedida a la *Api* externa en el capítulo 3 de esta memoria.

[3] Las máquinas virtuales son un entorno de software que simula ejecutar un sistema operativo y aplicaciones como si fuera una computadora física, pero todo esto lo hace a través de la nube.

[4] Para más información respecto a la *Api* de *Open Foods Facts*, se puede visitar su página: https://es.openfoodfacts.org/data

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN

3.1 Solución

Considerando el problema propuesto y las diversas problemáticas que tienen las personas con respecto a este dilema se propuso *Not Waste*.

Not Waste es una aplicación móvil que busca ayudar a las personas en sus hogares para que controlen mejor sus alimentos y con esto conseguir reducir el nivel de desperdicio de alimentos.

Figura 10: Logo de *Not Waste*
Fuente: Grupo *Beholders*

3.2 Funcionalidades de la aplicación

Ahora es momento de hablar de las funcionalidades que posee la aplicación *Not Waste.* Estas nacieron de los problemas presentados por la gente en el capítulo 1/punto 1.4.2.

3.2.1 Ingreso de productos en la aplicación

Trata de poder ingresar productos en la aplicación, en esta existen varios modos de añadir productos, en esta memoria se hablará de las siguientes 2 formas, las cuales son: ingreso manual e ingreso con el código de barras del producto.

Estos métodos nacieron dado que durante las entrevistas con posibles usuarios se comentó al grupo que: aunque existiera la aplicación no les gustaría usarla si el ingresar alimentos a esta fuera muy lento, debido a que ingresar alimentos 1 por uno manualmente a la aplicación sería un trabajo muy largo y tedioso.

3.2.2 Inventario

Como su nombre indica, esto es un inventario que posee todos los productos que tiene un usuario, estos productos mostrados son los que el usuario ingresa en la aplicación anteriormente.

Esta funcionalidad nació del problema de que a la gente se le puede olvidar que tienen ciertos productos, esto sumado a que facilita manejar sus productos a través de la aplicación.

3.2.3 Recetas

Actualmente esto no se encuentra implementado al 100% dentro de la aplicación debido a que es una parte extra dentro de la aplicación, pero lo que hace actualmente es buscar recetas basándose en los productos que tiene el usuario en su inventario

Esto nace debido a que cierto porcentaje de gente encuestada comentó durante las entrevistas que: les gustaría que la aplicación les dijera que pueden hacer con ciertos restos de comida.

3.2.4 Tienda

Dentro de la aplicación existe un apartado para que los usuarios puedan crear publicaciones donde se venden productos que estos tengan.

3.3 Consideraciones que tomar en cuenta dentro de la memoria

Antes de pasar a explicar los siguientes puntos es necesario dejar de forma explícita que dentro de esta memoria se tocará el *backend* de la aplicación *Not Waste.* Esto indica que los siguientes a puntos a tocar dentro del capítulo 3 serán:

3.3.1 Base de datos

Se hablará de la base de datos, específicamente sus tablas y sus campos. Para un mejor entendimiento de esto se dará una explicación de que es una tabla y a que se refieren con los campos usando de ejemplo la fotografía presentada a continuación:

	id [PK] integer	nombre character varying	apellido character varying
1	1	Martin	Fonseca
2	2	Ramon	Montalves

Figura 11: Ejemplo de una tabla dentro de una base de datos.
Fuente: Elaboración propia

Una tabla dentro de una *BDD* se puede entender como una entidad y los campos se pueden comprender como las características de una entidad. En este ejemplo el nombre de la tabla es: personas, haciendo alusión a que hablan de personas y sus campos son: *id* (el cual es identificador), nombre (nombre de la persona) y apellido (apellido de la persona). Dentro de las bases de datos, las tablas se dividen en filas y columnas, columnas para los campos de la tabla y las filas son instancias de la entidad (en este caso en la tabla personas, cada fila es una persona). Se hablará más de esto en el punto 3.4.

3.3.2 Funcionamiento de la *Api*

Otro de los puntos de los que se enfocará en el capítulo 3 es el funcionamiento de la *Api*. Esta conecta la aplicación móvil con la base de datos, la mayor razón de la existencia de esta es debido a que es necesario tener una capa intermedia entre estas 2 (*APP* y *BDD*). Durante este capítulo se explicará qué valores se envían desde la *APP* y que envía devuelta la *Api.* Se hablará más de esto en el punto 3.5.

3.4 Base de datos

Empezando con la sección de la base de datos debemos presentar las tablas y sus atributos que se usarán en la aplicación móvil. Cabe aclarar que estas no son todas las tablas, ni campos que se tienen dentro de la base de datos, pero el resto que se omitió en la fotografía no son necesarias para la explicación del funcionamiento de la *Api* [5].

[5] El resto de las tablas y campos omitidos no son usados para lo que se va a presentar dentro de la memoria o son tablas para la configuración del Django.

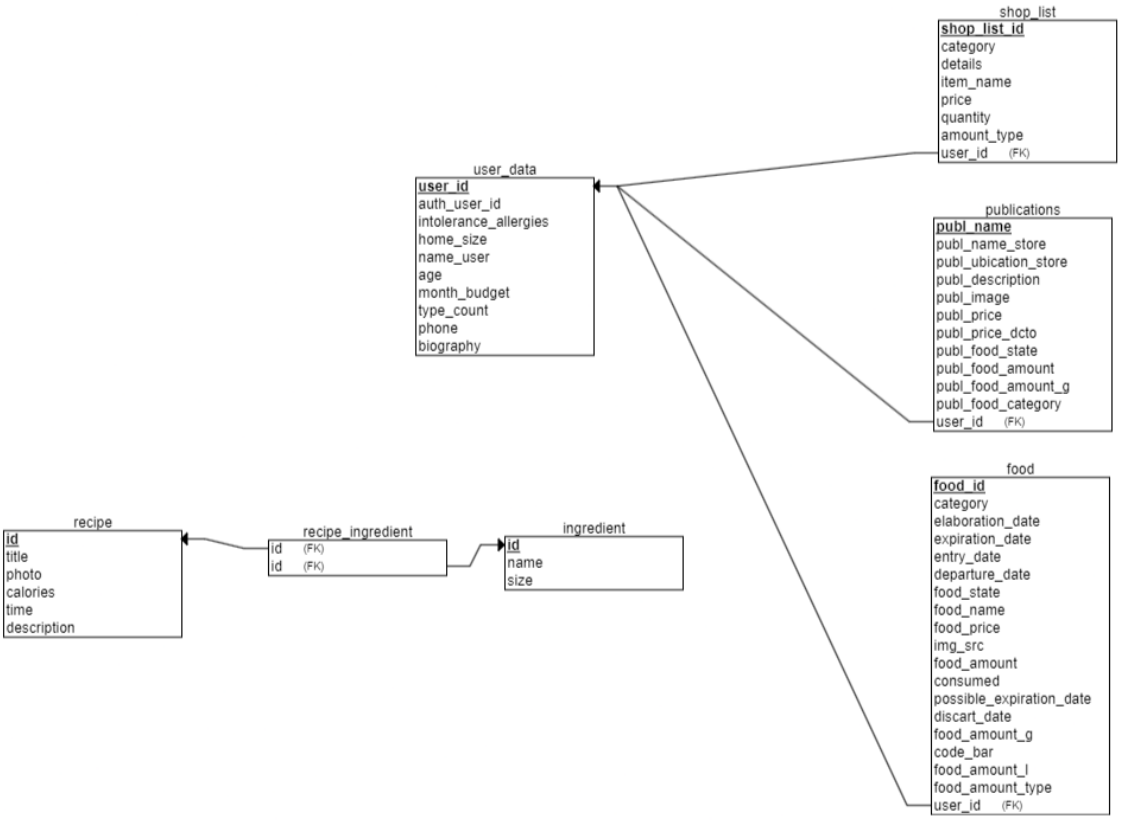

Figura 12: Esquema relacional de la base de datos de *Not Waste*.
Fuente: Elaboración propia

Se puede apreciar en la imagen las tablas que se encuentran dentro de la *BDD* de *PostgreSQL,* ahora es necesario explicar qué representa cada tabla en la fotografía.

- *user_data:* Tabla que se usa para los usuarios, aquí se guarda diferente información del usuario que puede ocuparse en otras partes dentro de la aplicación. Aquí se tiene el identificador con el nombre de: *user_id*, el cual se usa para tener relación con otras 3 tablas llamadas: *shop_list, publications y food.*

- *shop_list*: Tabla en la que se guardan los alimentos que tienen relación con la lista de compra del usuario. Aquí se guardan las características fundamentales para los alimentos, la categoría, el nombre, la cantidad, entre otras características.

- *publications*: Esta tabla dentro de la *BDD* se usa para las publicaciones de la tienda de la aplicación. Usa el identificador "*publ_name*" y guarda algunas

características de las publicaciones, tales como: su nombre, su ubicación, el precio del producto, entre otros.

- *food*: Tabla que se usa para guardar los alimentos de los usuarios. Tiene su identificador con el nombre de "*food_id*" y se guardan varias características del alimento como sus fechas (fecha de entrada a la aplicación, fecha de vencimiento, entre otras), categoría, cantidad, etc.

- *recipe*: Aquí se guardan las recetas con algunas de sus características. De identificador tiene su "*id*", el cual se usa en la tabla de *recipe_ingredient*.

- *Ingredient:* Una tabla donde se guardan varios ingredientes los cuales poseen un identificador bajo el nombre de "*id*".

- *recipe_ingredient*: Tabla que se usa para crear conexión entre la tabla de *ingredient* y *recipe* a través del identificador de cada tabla.

3.5 Funcionamiento de la *Api*

3.5.1 Explicación de cómo funciona la Api

Es necesario dar una breve explicación de cómo funciona la *Api* antes de explicar el funcionamiento de esta dentro de la aplicación *Not Waste*. La *Api* recibe información enviada por el usuario desde la *APP* y esta genera una acción y una respuesta. La siguiente imagen muestra cómo funciona.

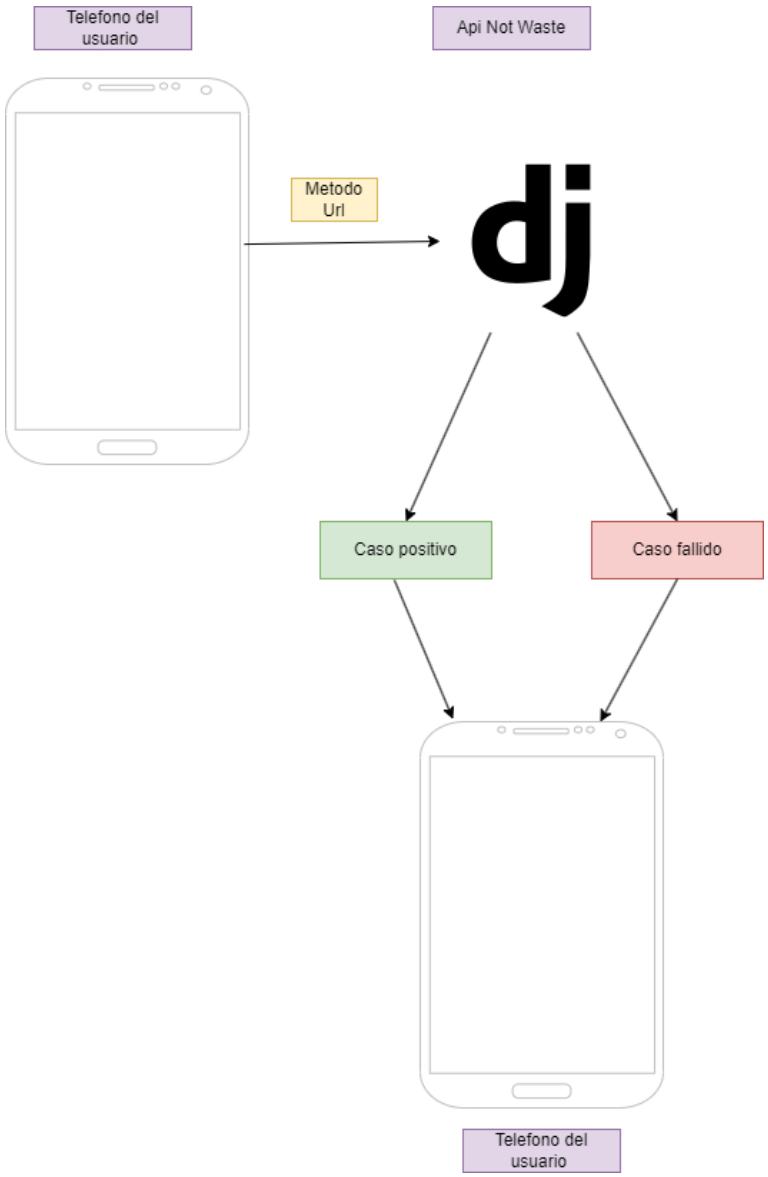

Figura 13: Funcionamiento de la Api en forma visual.
Fuente: Elaboración propia.

Todo lo de la imagen será mejor explicado en los siguientes subcapítulos, pero por ahora se dará una breve explicación: El teléfono del usuario hace una petición a una cierta dirección web, con esa dirección y con un método se hace una petición a la *Api* de *Not Waste*, esto puede terminar de dos formas con un caso correcto o un caso incorrecto, en ambos casos se termina enviado el resultado de vuelta al celular del usuario.

3.5.2 Formato *JSON*

JavaScript Object Notation también o *JSON* es un formato de texto estándar para la representación de datos estructurados, en palabras simples es un texto con una estructura estándar, esta se usa aquí para él envió de información de parte de las acciones y de las respuestas se muestra un ejemplo del formato.


```
{
    "product": {
        "_id": "7802215505300",
        "_keywords": [
            "costa",
            "imbis",
            "kekse",
            "kuchen",
            "snack",
            "susser",
            "und",
            "vino"
        ]
    }
}
```


Figura 14: Sección de la respuesta de la Api de Open Foods Facts en formato JSON
Fuente: Elaboración Propia

3.5.3 Acciones o métodos

Dentro del contexto de la *Api*, la palabra 'acciones' se refieren a los métodos *HTTP*[6], estos métodos son las acciones que va a hacer la *Api,* las cuales también son conocidos como petición *HTTP*, a continuación, se nombran las que son usadas y se darán ejemplos de cómo funcionan usando tablas expuestas en la Figura 13*:*

- *Get:* Este método se utiliza para conseguir datos de un recurso a un servidor. Tomando de ejemplo la tabla *food*, usando el método *Get* se intenta obtener información de una comida que esté dentro de la tabla.
- *Post:* Este método es lo contrario al método *Get*, mientras el anterior se enfoca en traer recursos del servidor, este se usa cuando se quiere subir recursos al servidor. Tomando la tabla *food*, el método *Post* permite crear una nueva comida para la tabla.

[6] Para mayor explicación se puede visitar esta página *web* que habla de los métodos *HTTP* (La *web* se encuentra en inglés): https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

- *Put:* Se usa cuando se quiere modificar toda una instancia de un objeto. Siguiendo con los ejemplos de la tabla *food*, teniendo un alimento elegido, se cambiará todas las características de este con las nuevas características.

- *Patch:* Similar al método *Put*, pero este es solamente usado para una modificación parcial. Para cambiar tomemos la tabla *ingredient*, eligiendo solamente un ingrediente y se le cambió el nombre, pero el resto de las características del alimento quedan intactas.

- *Delete: Delete* es usado para eliminar un recurso. Tomando la tabla *ingredient*, se elige un ingrediente y este se elimina de la base de datos.

3.5.4 Url

Uniform Resource Locators por sus siglas *URL* es una dirección web de un recurso[7],dentro del contexto de la *Api*, estas *URL* permiten llegar a se usan para llegar a los *Endpoint*, los cuales son la dirección donde se ejecuta la petición *HTTP*. A continuación, se presenta una tabla con los Endpoint de la Api que se presentarán en la memoria:

Tabla 2: Url en la *Api* de *Not Waste*
Fuente: Elaboración propia

Método	*Endpoint*	Descripción
Post	*api/food/user*	Ingreso manual de alimentos a la aplicación.
Post	*api/food/barcode*	Ingreso de alimentos usando el código de barras.
Get	*api/food/user*	Obtener el inventario de un usuario.

[7] Un recurso en este contexto se puede definir como información, archivó o servicio que se puede acceder a través de una *URL*.

Get	*api/food/user/<int:pk>*	Obtener información de un producto de un usuario (*pk* hace referencia al identificador de producto dentro de la base de datos).
Get	*api/recipes/ingredient*	Obtener las recetas que tengan en sus ingredientes alimentos que se encuentren entre los productos del usuario.
Get	*api/recipes/ingredients/<int:recipe_id>/*	Obtener los ingredientes de una receta.
Get	*api/publications/user*	Obtener las publicaciones hechas por el propio usuario.
Post	*api/publications/user*	Publicar una nueva publicación en la tienda de la aplicación.

3.5.5 Respuesta

Como se comentó anteriormente la *Api* genera una acción y una respuesta, dado que ya se explicó la acción, es momento de explicar la respuesta. La *Api* intenta generar la acción la cual puede terminar en 2 situaciones que son: se consiguió hacer o no se consiguió hacer. Ahora dependiendo cada caso el sistema envía una respuesta que puede variar si la acción fue correcta o no,

estas respuestas son códigos de estado *HTTP*[8], estos son respuestas que envía el servidor para indicar el resultado de una petición *HTTP,* la respuesta positiva y negativa del servidor junto a sus códigos de cada una se explica a continuación:

Acción que genera una respuesta positiva: Si la acción que se hizo se ejecutó de forma correcta, la *Api* genera una respuesta positiva junto a un código, este es de la numeración 2xx (200 -299) el cual se usa para una respuesta positiva de parte del servidor.

Tabla 3: Códigos de respuesta positiva.
Fuente: Elaboración propia.

Numeración	Nombre	Explicación
200	*Ok*	Este código es uno de respuesta positivo por defecto. Simplemente indica que la petición se hizo de forma correcta.
201	*Created*	201 se usa junto a los métodos Post para indicar la creación exitosa de un recurso.
204	*No content*	El 204 es usado para decir que la petición fue correcta pero que no es necesario que el usuario cambie de página.

Acción que genera una respuesta negativa: Si la petición que se hizo al servidor se ejecutó de forma incorrecta, la *Api* genera una respuesta negativa junto a un código, este usa la enumeración 4xx (400-499), la cual se usa para una respuesta negativa de parte del servidor.

[8] Existen más códigos de los hablados en la memoria para más información puede ingresar a la página *web*: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Tabla 4: Códigos de respuesta negativa.
Fuente: Elaboración propia.

Numeración	Nombre	Explicación
400	*Bad request*	Indica que la petición no se pudo procesar mayormente por error de parte del usuario.
404	*Not Found*	Este código indica que no se encontró el recurso buscado por la petición.

3.5.6 Funcionalidades en la *Api*

Dado que ya se explicó cómo funciona la *Api* es momento de explicar las funcionalidades dentro de esta, cuáles son las peticiones y respuestas en el caso exitoso y caso negativo. Cabe aclarar que en algunas funcionalidades no existe un caso fallido dentro de la *Api*, pero que pueden llegar a fallar por problemas externos de esta tales como: falla de conexión entre la *Api* y la aplicación móvil.

Ahora en funciones donde se hace una petición *Post* para crear algo y este algo tiene una fotografía asociada puede pasar el caso que la fotografía no se incluya en la información de este algo, lo que hace la *Api* en estos casos es usar una imagen por defecto que se tiene para estas situaciones, se explica esto ahora ya que, aunque no está totalmente correcto esto no provoca un fallo en la función.

3.5.6.1 Ingreso de productos

2 funcionalidades que tratan acerca de cómo un usuario puede ingresar productos a la aplicación.

3.5.6.1.1 Ingreso manual

3.5.6.1.1.1 Explicación

Un usuario a través de la aplicación ingresa de forma manual las características de un alimento, cuando el usuario guarda el alimento en

la aplicación, esta información se envía a la *Api* de la aplicación, está guarda el producto en la *BDD* y envía una respuesta a la aplicación.

3.5.6.1.1.1.1 Funcionamiento

Caso Exitoso: Desde la aplicación se envía la información del producto ingresado en formato *JSON* en conjunto con el usuario que envió la petición, esto es recibido por la *Api*, está verifica cual usuario mando la petición, si toda la información es correcta guarda el producto en la base de datos y su fotografía en el *S3*, envía un código 201 devuelta a la aplicación.

Caso Fallido: Debido a que desde la propia aplicación se verifica los datos para que estos sean correctos la funcionalidad no debería fallar, ahora puede darse el caso de que llegue a fallar la funcionalidad en un determinado caso, ahora si esto llegara a pasar se provocar un error, en respuesta la *Api* envía a la aplicación un código de error 400.

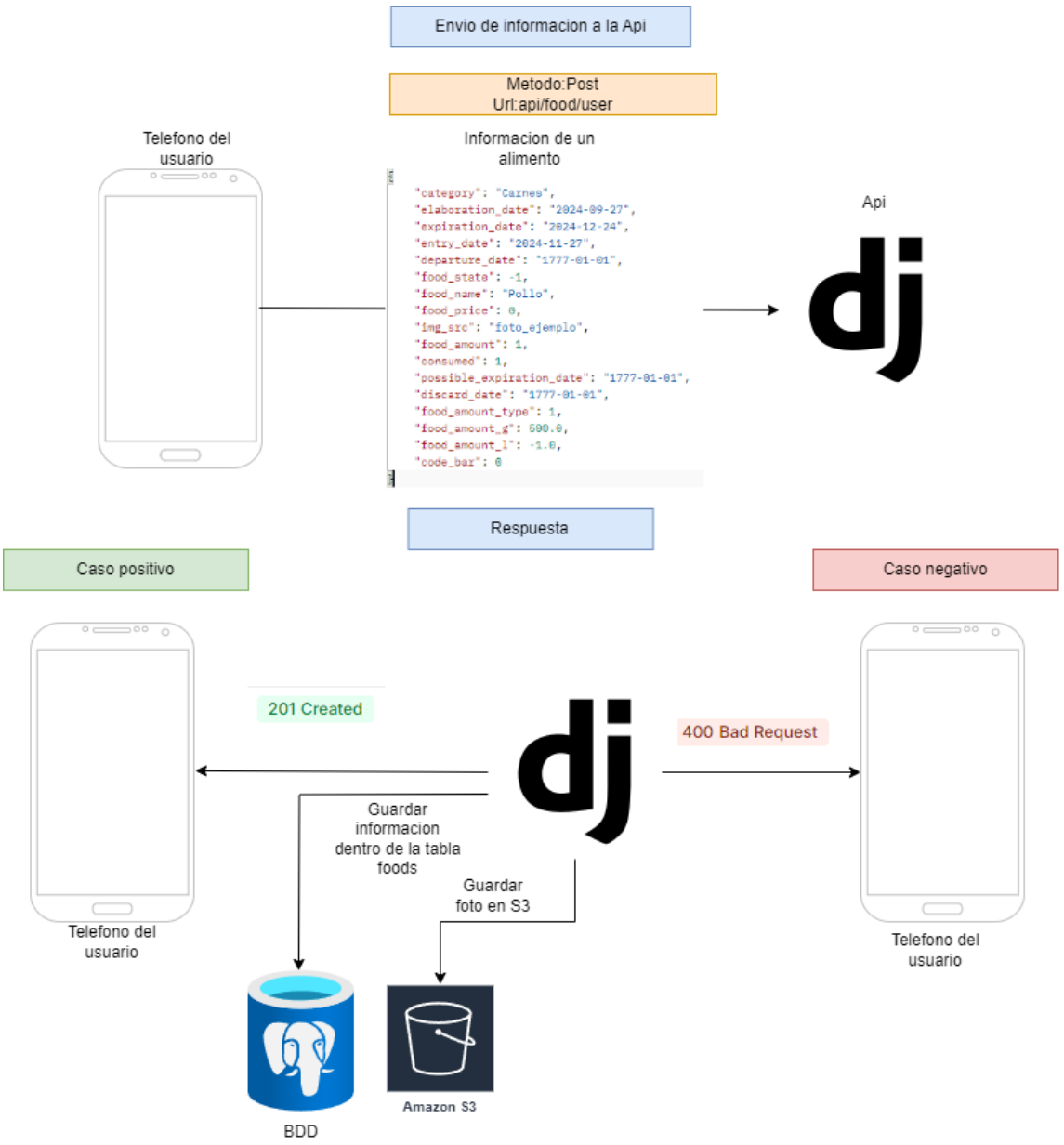

Figura 15: Proceso de ingreso manual a la aplicación.
Fuente: Elaboración propia.

3.5.6.1.2 Ingreso con código de barras

3.5.6.1.2.1 Explicación

Un usuario puede ingresar un producto alimenticio a la aplicación usando el código de barras del producto. Este código de barras se envía a la *Api* de *Not Waste* y se usa en conjunto con una *Api* externa propiedad de *Open Foods Facts*.

3.5.6.1.2.2 *Api* de *Open Foods Facts*

La *Api* de *Open (*Como se referirá a *Open Foods Facts* de ahora en adelante) ofrece diversa información de un alimento. Esta se usa en conjunto al código de barras del producto para detectar el alimento. De la información proporcionada por la *Api* se requieren ciertos aspectos del alimento tales como:

- Nombre
- Cantidad (Kilogramo o litro)
- Fotografía del producto
- Categoría del producto

A continuación, se muestra una imagen donde se muestran los campos que son usados en la *Api* de *Not Waste*, estos campos son en el caso de que la respuesta de parte de la *Api* de *Open* las contenga, en caso de no ser así se buscan otros campos que posean información similar o se usa información por defecto.


```
{
    "product_name": "Céréales Mini Pack Kellogg's Variety",
    "product_quantity": "215",
    "product_quantity_unit": "g",
    "image_url": "https://images.openfoodfacts.org/images/products/315/947/000/5347/front_fr.61.400.jpg",
    "categories": "Aliments et boissons à base de végétaux, Aliments d'origine végétale, Céréales et pommes de terre, Céréales et dérivés"
}
```


Figura 16: Sección de la respuesta de Open Foods Facts.
Fuente: Elaboración propia.

3.5.6.1.2.3 Funcionamiento

Caso exitoso: El Usuario ingresa un producto en la aplicación usando una foto del código de barras del producto, la aplicación envía a la *Api* de *Django* el código de barras y esta solicita información a la *Api* de *Open*. La *Api* externa toma el código del producto y busca la información del producto y la devuelve a *Django*, luego esta toma la información del producto y la guarda en la tabla de *food* dentro de la *BDD*, guarda la foto en el *S3*, y envía un código 201 a la aplicación como respuesta.

Caso Fallido: Existen 3 casos en los que se puede producir un error, estos son:

- Se envía un código de barras a la *Api* de *Not Waste*, pero esta detecta que hubo un problema con el código y envía un código de error 400 a la aplicación. Ejemplo: Que el número del código de barras enviado sea muy pequeño o no se haya enviado ninguno código de barras a la *Api*.

- Se envía el código a la *Api* externa, pero este no se encuentra dentro de la base de datos de *Open* así que retorna error, en este caso la *Api* de *Not Waste* detecta un problema y envía a la aplicación un código 404.

- Al momento de convertir los datos para guardarlo en la base de datos sucede un problema, así que la *Api* envía a la aplicación un error de código 400.

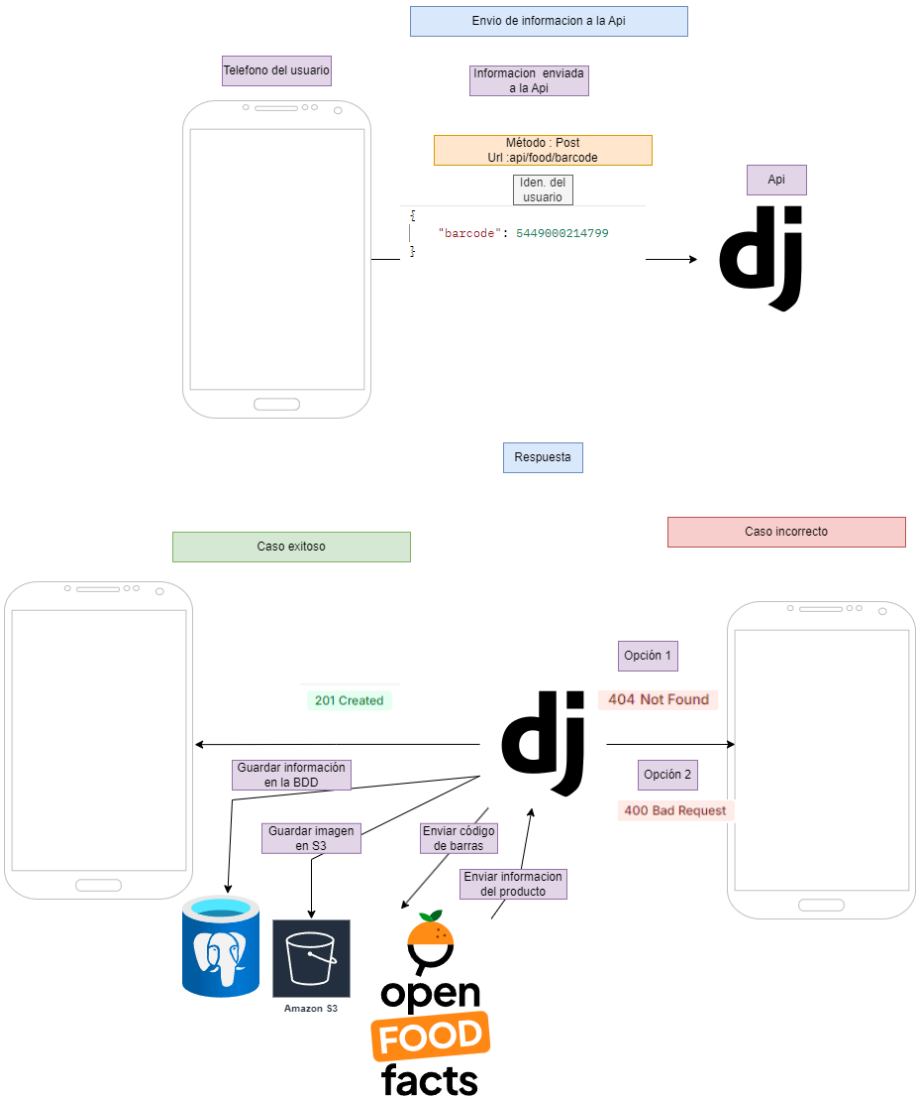

Figura 17: Proceso del ingreso con código de barra.
Fuente: Elaboración propia.

3.5.6.2 Inventario

Funcionalidades relacionadas al inventario de cada usuario.

3.5.6.2.1 Obtener productos del inventario

3.5.6.2.1.1 Explicación

Un usuario entra a la sección de inventario dentro de la aplicación y puede ver los alimentos que tiene registrados, esto se hace de la siguiente forma: al momento de que el usuario entre a la sección del inventario se envía una petición *Get* a la *Api* en conjunto con el identificador del usuario, este identificador se usa para hacer una

consulta a la *BDD* acerca de los alimentos que tienen ese id asociado, luego de esto devuelve todos los alimentos a la aplicación.

3.5.6.2.1.2 Funcionamiento

Caso exitoso: El usuario ingresa al inventario en la aplicación, este envía la petición a la *Api* y se devuelve a la aplicación la lista de comidas del usuario en conjunto con el código de respuesta 200.

Caso fallido: Por el funcionamiento de la petición no se puede fallar, ya que la única forma en que pueda fallar son situaciones externas tales como: mal funcionamiento de la base de datos o un fallo de conexión entre la aplicación y la *Api*, ambas situaciones van más lejos de las capacidades de esta funcionalidad.

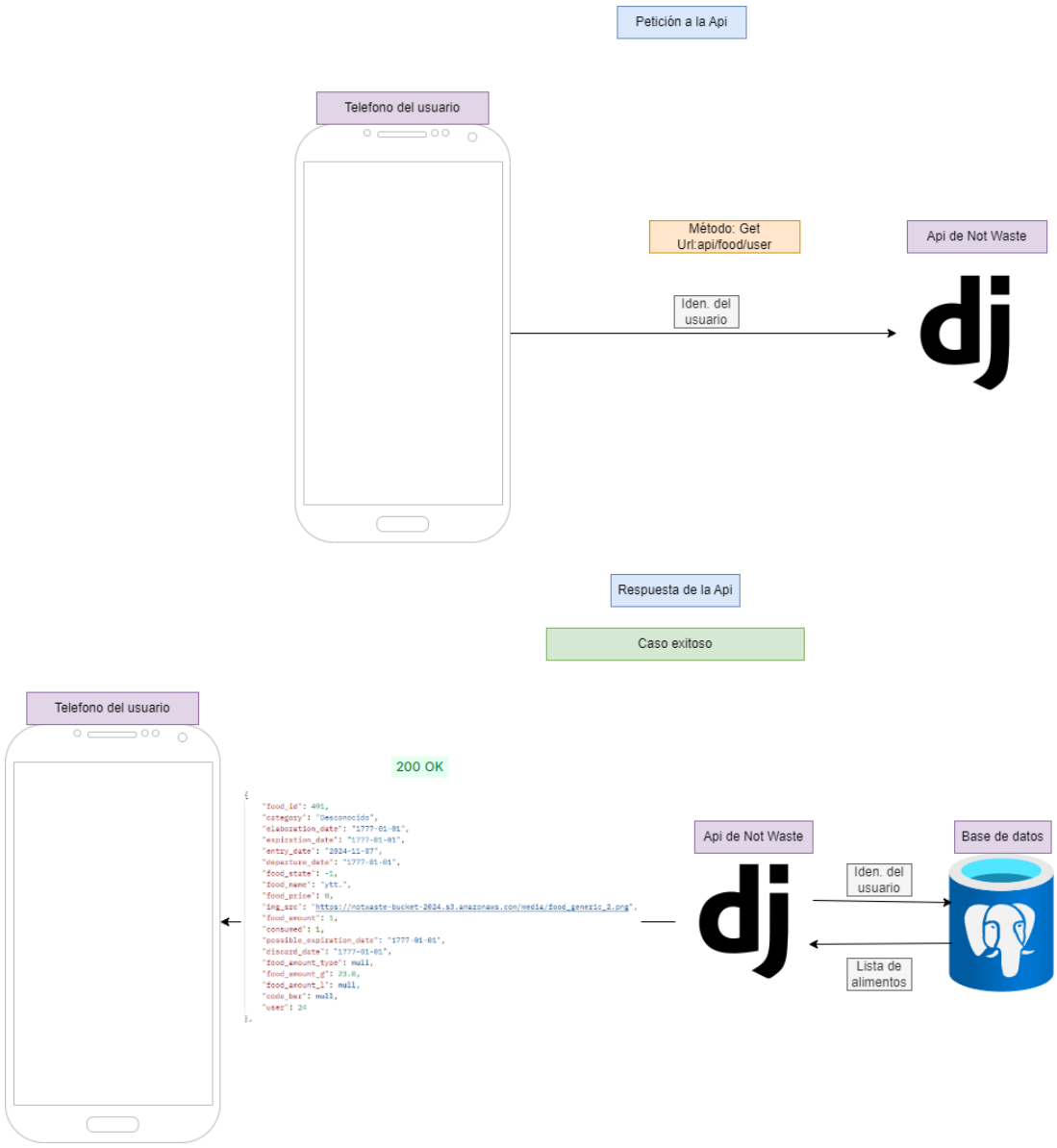

Figura 18: Proceso de obtener productos del inventario.
Fuente: Elaboración propia.

3.5.6.2.2 Ver información de un producto

3.5.6.2.2.1 Explicación

Un usuario dentro del inventario elige un producto para ver más información del producto elegido.

3.5.6.2.2.2 Funcionamiento

Caso exitoso: Un usuario elige un producto dentro del inventario en la aplicación, al hacer esto se envía una petición *Get* a la *Api, esta* petición

envía un número a la *Api* el cual es el identificador del producto dentro de la base de datos. Toma la información del producto y lo envía de vuelta a la aplicación en conjunto a un código de respuesta 200.

Caso fallido: La única forma en que este apartado puede fallar es si el producto no existiera en la base de datos, ahora esto no es posible debido a que primero es necesario ingresar a un alimento dentro del inventario para ejecutar esta petición, y si el producto no existe no es posible ingresar a este dentro de la aplicación, haciendo que no se pueda ejecutar esta petición.

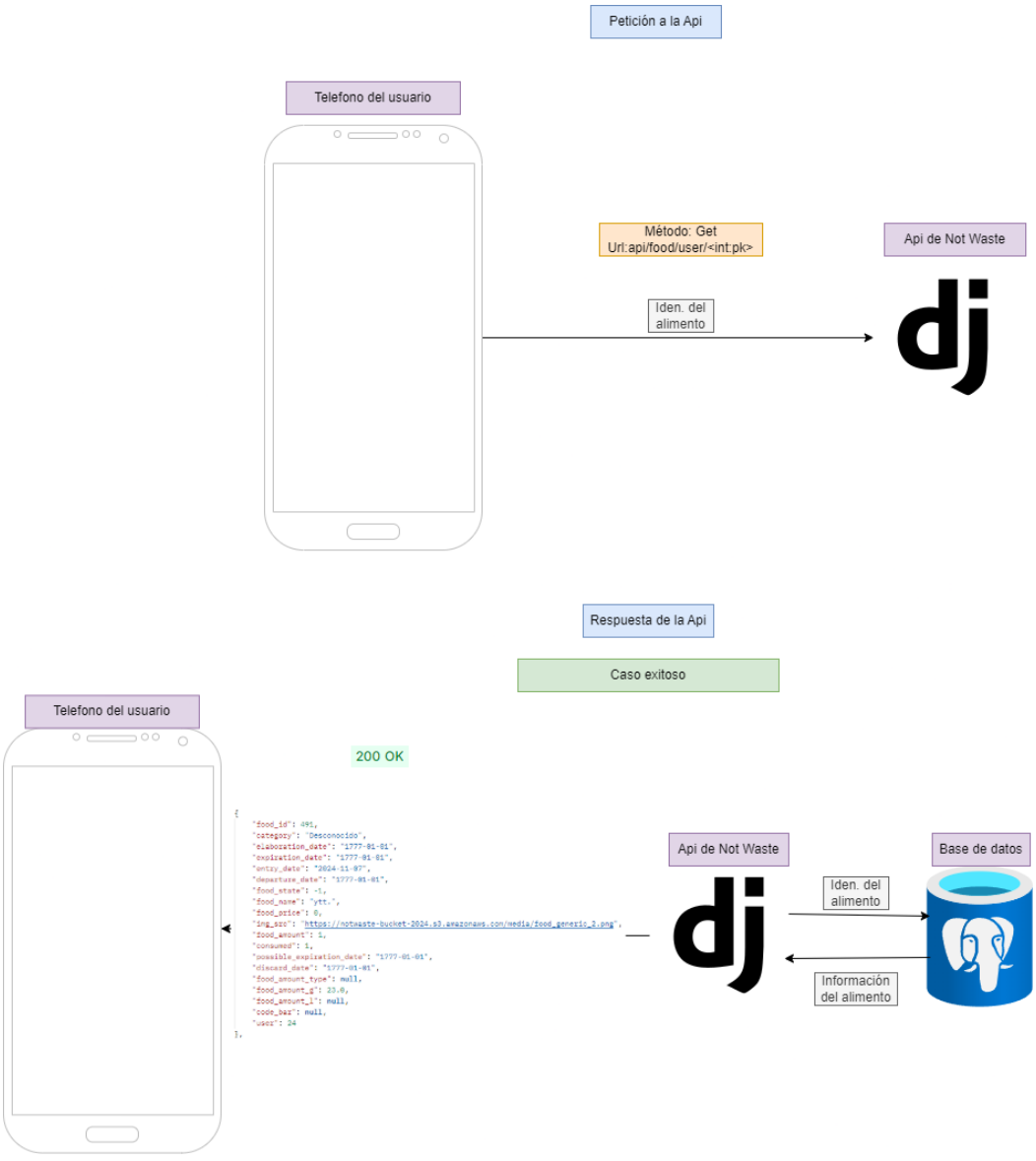

Figura 19: Proceso ver información de un alimento.
Fuente: Elaboración propia.

3.5.6.3 Recetas

Funcionalidades relacionadas a la sección de las recetas dentro de la aplicación.

3.5.6.3.1 Obtener recetas

3.5.6.3.1.1 Explicación

Un usuario ingresa a la sección de recetas, dentro de esta existe una sección que muestra recetas en base a los alimentos que tiene el usuario.

3.5.6.3.1.2 Funcionamiento

Caso exitoso: El usuario al ingresar a la sección de recetas se envía una petición *Get* a la *Api*, esta ve los productos del usuario y los compara con los ingredientes en la tabla *ingredient*, al encontrar una similitud envía las recetas de la tabla *recipe* que usen esos ingredientes en conjunto un código de respuesta 200.

Caso fallido: En el caso de que dentro de los alimentos del usuario no exista ningún ingrediente que tenga el mismo nombre, esta envía un error con el código 404.

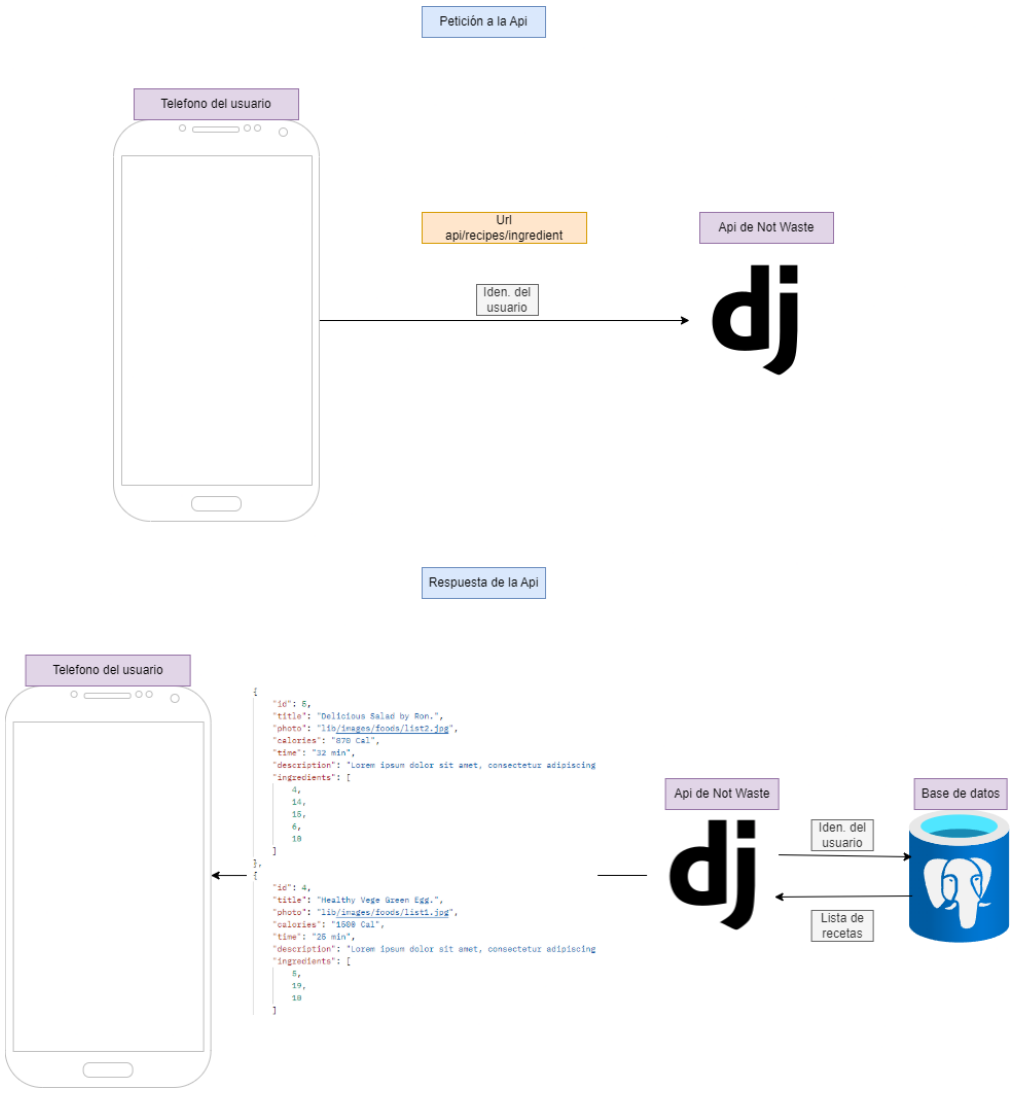

Figura 20: Proceso de obtener recetas.
Fuente: Elaboración propia.

3.5.6.3.2 Obtener ingredientes de las recetas

3.5.6.3.2.1 Explicación

Dentro de la sección de las recetas, al elegir una de estas, se cargan los ingredientes de esa receta en específico.

3.5.6.3.2.2 Funcionamiento

Caso exitoso: El usuario ingresa a una receta, al hacerlo la aplicación envía una petición *Get* a la *Api* la cual busca los ingredientes de esa receta. Al encontrarlas envía de vuelta a la aplicación los ingredientes y manda un código de respuesta 200.

Caso fallido: No existe un caso de fallo en la *Api* ya que al existir una receta siempre se tendrá ingredientes asociados a esta.

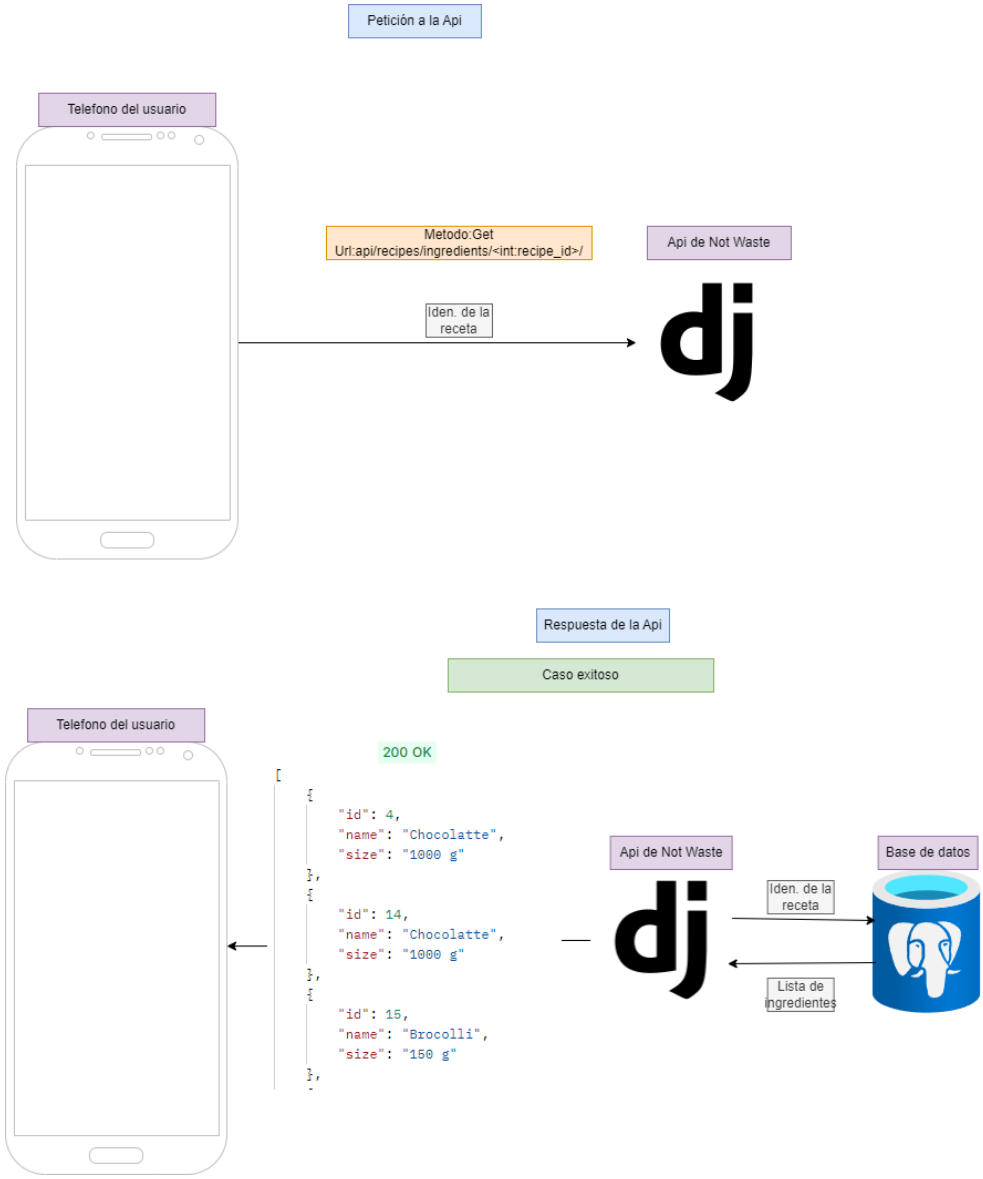

Figura 21: Proceso de obtener los ingredientes de la receta.
Fuente: Elaboración propia.

3.5.6.4 Tienda

Funcionalidades respecto a la tienda en la aplicación.

3.5.6.4.1 Ver publicaciones de la tienda

3.5.6.4.1.1 Explicación

En la sección de la tienda, un usuario puede ver todas las publicaciones que hizo.

3.5.6.4.1.2 Funcionamiento

Caso exitoso: Él usuario ingresa a la sección de la tienda para ver sus publicaciones, aquí se manda una petición *Get* a la *Api*, dentro de la *Api* se buscan las publicaciones dentro de la tabla *publications* que comparta el *user_id* con el usuario que realizó la petición, al encontrarla o encontrarlas procede a enviar las publicaciones al teléfono en conjunto a un código de respuesta 200.

Caso fallido: Dentro de la *Api* no existe un caso en que esta funcionalidad pueda fallar esto debido a que, aunque el usuario no tenga publicaciones se mandara algo de vuelta haciendo que la aplicación no falle.

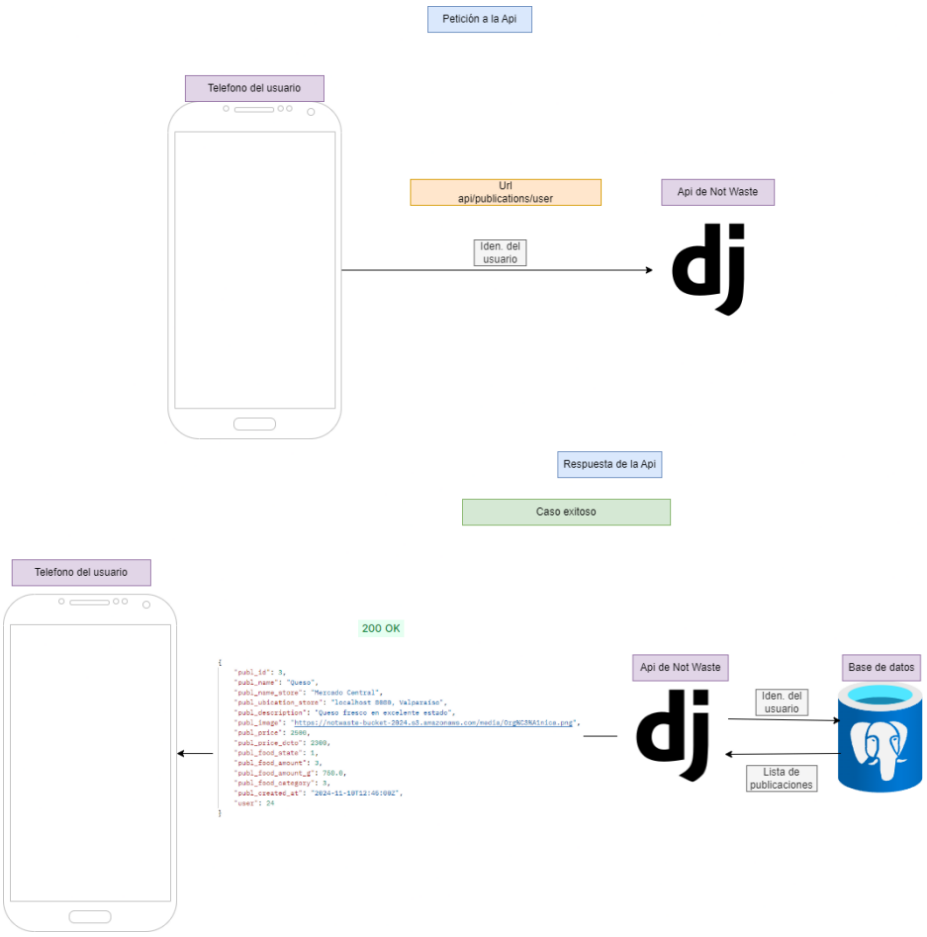

Figura 22: Proceso de ver publicaciones del usuario.
Fuente: Elaboración propia.

3.5.6.4.2 Crear publicación

3.5.6.4.2.1 Explicación

Que un usuario pueda crear su propia publicación dentro de la sección de la tienda.

3.5.6.4.2.2 Funcionamiento

Caso exitoso: Un usuario dentro de la sección de publicaciones en la tienda crea una publicación esto manda una petición *Post* a la *Api* con la información de la publicación creada en formato *JSON*, la *Api* verifica el usuario que mandó la información y la información que mando si toda la información mandada es correcta, ya que toda la información junto a la imagen es correcta, guarda la imagen en el *S3* y guarda la información

dentro de la base de datos, siguiente a esto manda un código de respuesta 201 de vuelta a la *APP*.

Caso fallido: En el caso de que la información enviada desde la aplicación presente problemas, la funcionalidad dentro de la *Api* falla y envía un código de respuesta 400 a la aplicación indicando que hubo un error.

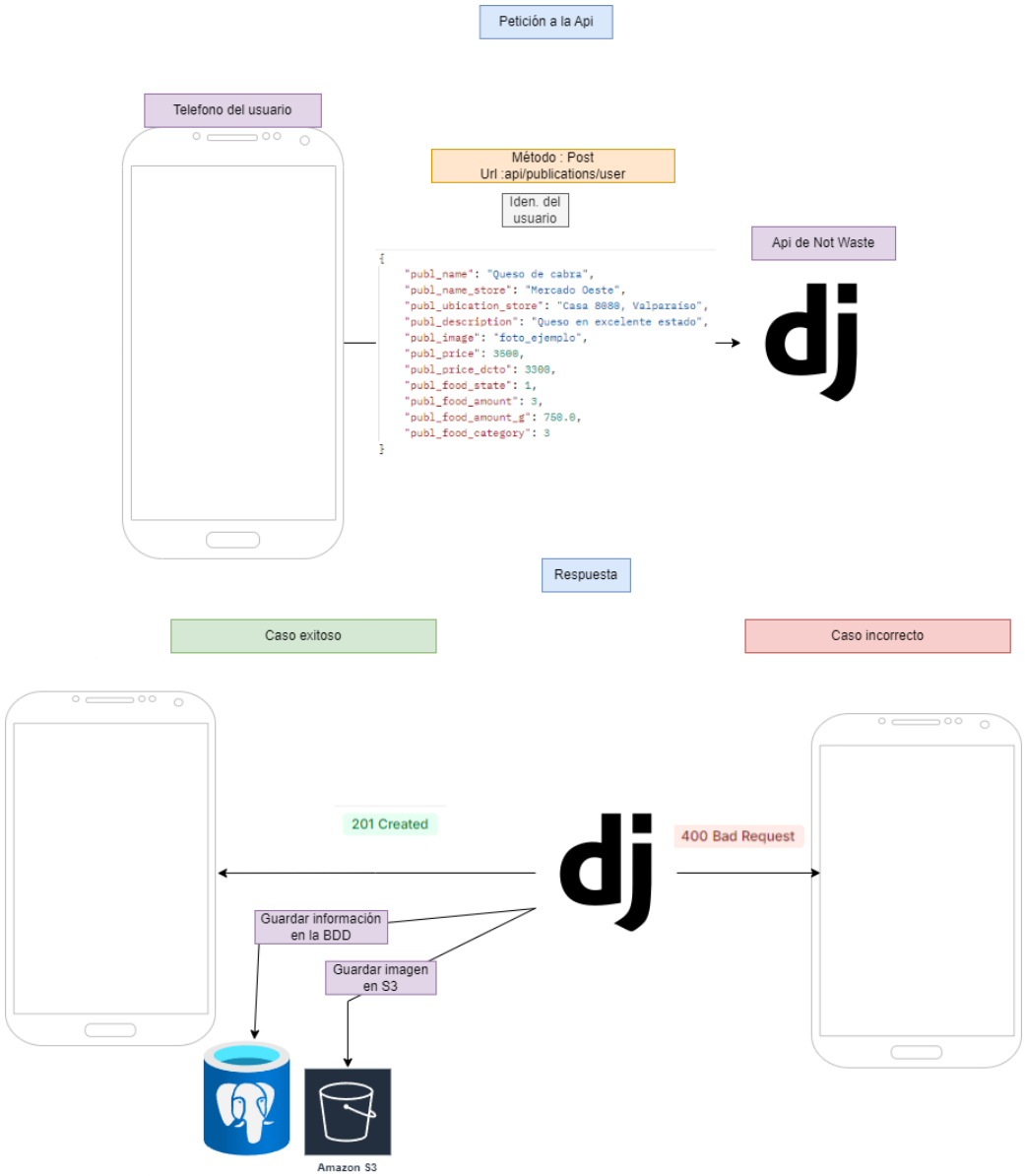

Figura 23: Proceso de creación de una publicación.
Fuente: Elaboración propia.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

Dado que ya se explicó cómo funciona la *Api* durante el capítulo pasado en momento de validar que todo lo que se propuso funciona como debería esto mediante 2 procesos mencionados durante el capítulo 2, los "*test* unitarios" y los "*test* de compatibilidad con el resto de las funciones".

4.1 Validación

Es necesario comentar algunos detalles de la validación de la solución, los encargados de estas validaciones fueron los mismos participantes del grupo, las pruebas hechas fueron en 2 formatos: mandando a la *Api* solicitudes desde un *PC*, y desde la aplicación móvil de *Not Waste*, pero debido a que solamente estoy hablando de la *Api*, se mostraran las pruebas hechas a esta desde un *PC*.

4.2 Flujo de pruebas

Antes de entrar en los *tests*, se hará una explicación breve del flujo para el desarrollo de una función y para las pruebas.

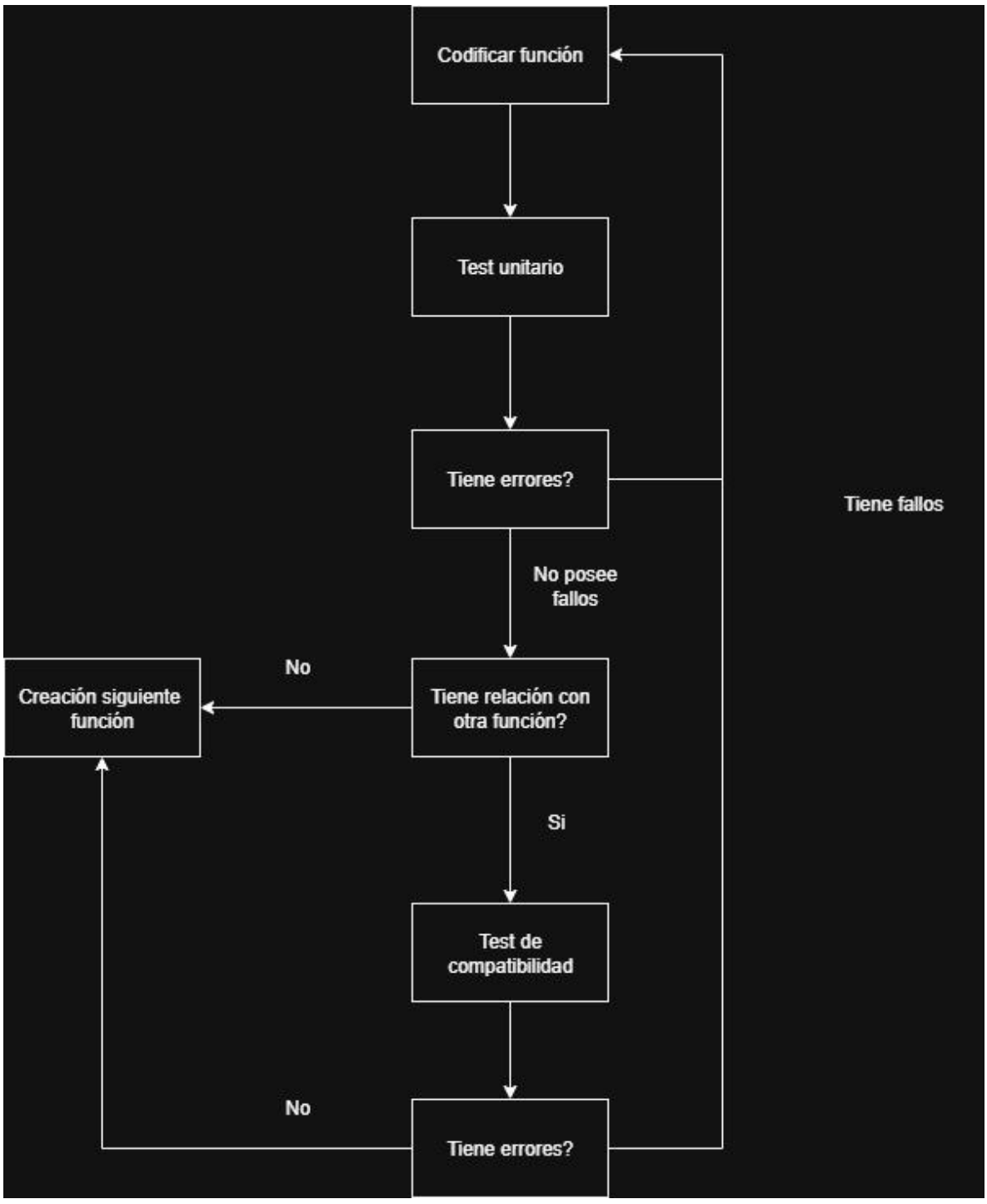

Figura 24: Flujo de pruebas.
Fuente: Elaboración propia.

Se dará una explicación paso a paso de la imagen.

1. Se codifica una función.
2. Se hace una prueba de esta, si esta posee errores va nuevamente al punto 1, si no tiene va al siguiente punto.
3. Se comprueba si esta característica de la aplicación móvil tiene relación con otra, en otras palabras, si la ejecución de esta se ve afectada o afecta otras funcionalidades. Si no lo hace se procede a la creación de la siguiente función, ahora si tiene relación con otra función entra al punto 4.
4. *Test* de compatibilidad donde se comprueba si las funciones que interactúan entre si funcionan perfectamente y sin errores, si no los posee igual que en el punto 3 se continua con otra funcionalidad, ahora si durante esta prueba hay errores, se reinicia desde el punto 1 donde se corrigen estos errores.

Explicado ya la imagen se explorará más a fondo en los siguientes puntos.

4.3 *Tests* unitarios

En estos *tests* o pruebas, se comprueba que las funcionalidades actúan de forma correcta en ambos casos, negativo y positivo, dando como resultados los tipos de respuesta anteriormente explicados. En estas pruebas el objetivo es siempre que los resultados de estas obtengan *Passed* o aprobado, lo que da a entender que el código se ejecuta de la forma que se tiene pensado. Estas pruebas solamente se harán con funcionalidades que puedan fallar y no las cuales solamente tengan el caso correcto y que no tengan una forma de fallar. Se decidió usar este tipo de *test* ya que es fácil comprobar cuando algo funciona debido a que se puede ver mejor que se envía y que se recibe de vuelta.

4.3.1 Cantidad de *tests*

En este apartado se hacen *tests* de 3 funcionalidades de la aplicación y se muestran 7 casos en total, donde 2 de estas funcionalidades tienen 2 casos y 1 funcionalidad tiene 3 casos posibles, estos 7 casos constan de 3 para el funcionamiento correcto de la funcionalidad y el resto para un funcionamiento incorrecto de la funcionalidad.

4.3.2 Validaciones revisadas en las pruebas

Con el objetivo de que las funcionalidades sean correctamente revisadas, se tiene que definir validaciones para cada prueba:

- Código de respuesta: Verificar que el código de respuesta de parte de la *Api* sea el correcto para cada contexto, tanto en los casos negativos como en los casos positivos.

- Respuesta enviada desde la *Api*: Verificar que la respuesta enviada desde la *Api* sea correcta.

Con esto explicado es momento de pasar a las pruebas que se harán a las funcionalidades.

4.3.3 Respuesta esperada

Los resultados esperados para los *tests* unitarios de las diferentes funcionalidades que se mostraran a continuación son que todas estas deberían funcionar como se explayo en el desarrollo del capítulo anterior, esto haciendo referencia a que deben reaccionar como se indicó, si la información con la que se probó la funcionalidad es correcta esta tiene que responder con el caso correcto , en caso de que la información sea incorrecta o no exista la función tiene que responder con el caso incorrecto.

4.3.4 Agregar manualmente

Con respecto a esta funcionalidad se hacen 2 *tests, uno en el caso de que la información sea correcta y otra en el caso de que la información sea incorrecta*

4.3.4.1 Caso exitoso – Datos correctos

En este caso se envían datos correctos para analizar la reacción que tiene el sistema frente este caso.

Validaciones para esta prueba:

- Código de respuesta.

- Respuesta enviada desde la *Api*.

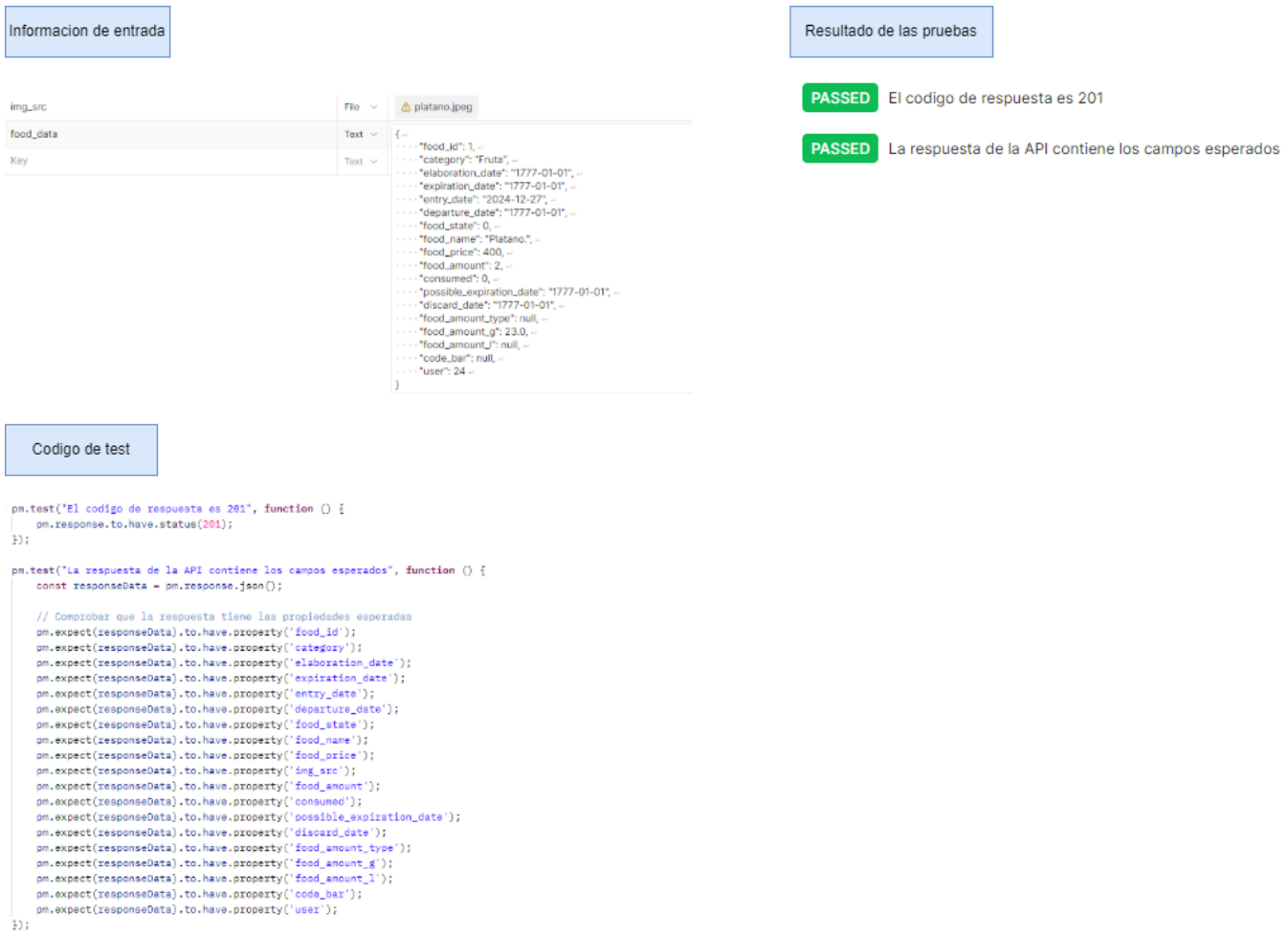

Figura 25: Caso correcto - información correcta.
Fuente: Elaboración propia.

4.3.4.2 Caso fallido – Datos incorrectos

Se envían datos incorrectos (en este caso una de las fechas se envió de forma incorrecta) para verificar la interacción con la *Api.*

Validaciones para esta prueba:

- Código de respuesta.

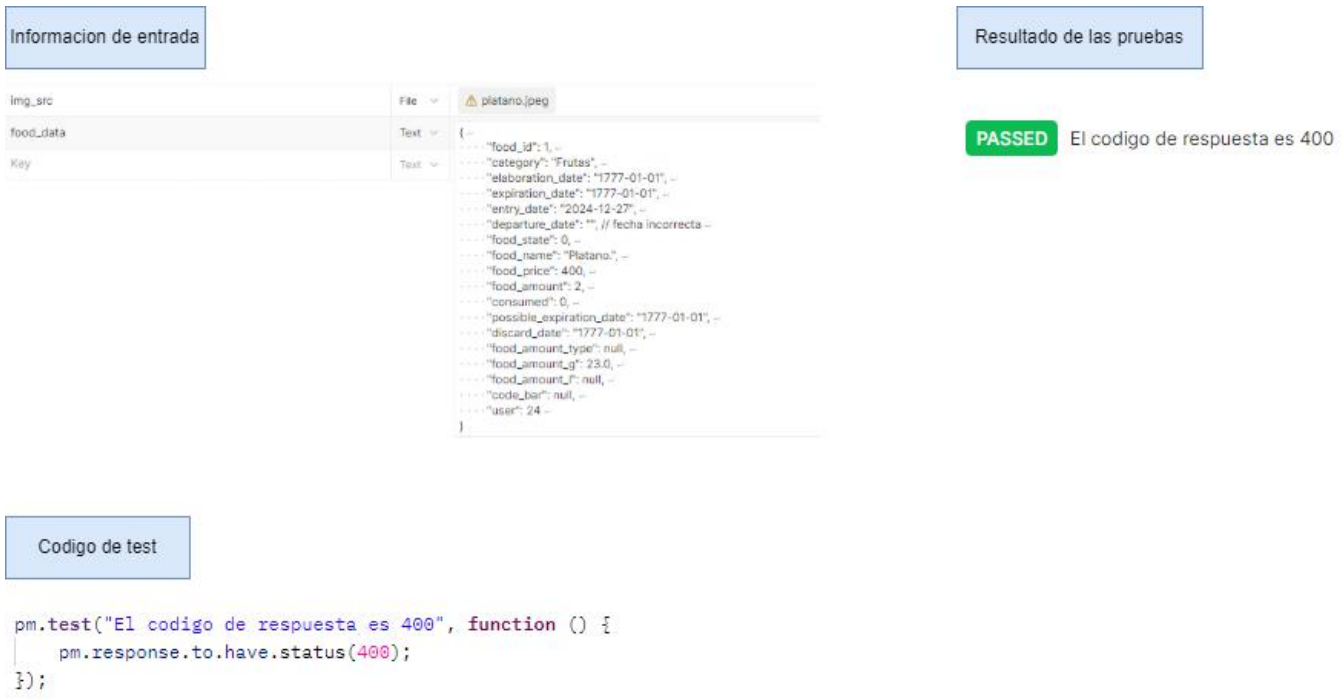

Figura 26: Caso fallido - Datos incorrectos.
Fuente: Elaboración propia.

4.3.5 Agregar productos usando el código de barra

Con esta funcionalidad se hacen 3 *tests* relacionados con el *barcode* o código de barra, uno en el caso de que el código de barra sea correcto y los otros 2 para los casos en que los códigos de barra sean incorrectos (Falta de números/Código de barras no registrado en *Open*).

4.3.5.1 Caso exitoso - Código de barras correcto

Se envió un código de barras el cual existe dentro de la *BDD* de *Open* para comprobar el funcionamiento de la *Api.*

Validaciones para esta prueba:

- Código de respuesta.
- Respuesta enviada desde la *Api*.

Figura 27: Caso exitoso - Barcode correcto.
Fuente: Elaboración Propia

4.3.5.2 Caso fallido - Código de barras no registrado en *Open*

Se envía un código de barras el cual no existe dentro de la *BDD* de *Open* para comprobar la interacción de las dos *Apis.*

Validaciones para esta prueba:

- Código de respuesta.

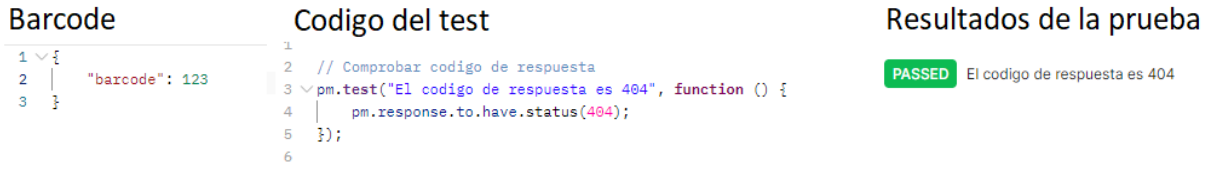

Figura 28: Caso fallido – Código de barras no incluida en *Open*.
Fuente: Elaboración Propia.

4.3.5.3 Caso fallido – Código de barras faltante

En este caso se envía un código de barras con fallos (en este caso no se envía nada) para comprobar que el sistema funciona de la forma que se explicó anteriormente.

Validaciones para esta prueba:

- Código de respuesta.

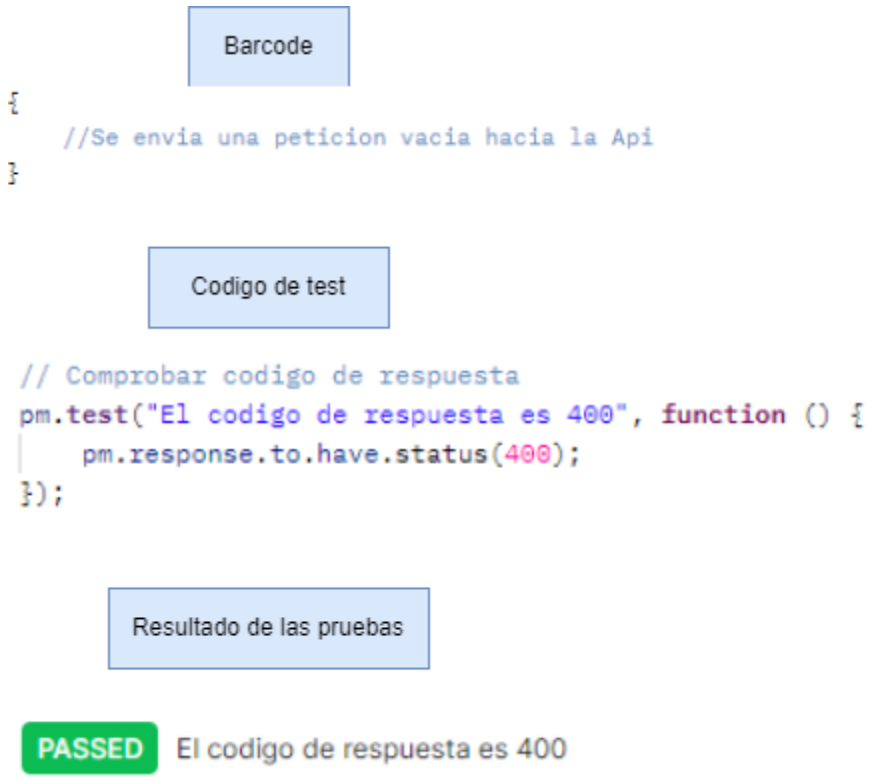

Figura 29: Caso fallido - Código de barras faltante.
Fuente: Elaboración propia.

4.3.6 Crear publicaciones

Para esta funcionalidad se hacen 2 pruebas, una prueba en el caso de que la información sea correcta y otra prueba para el caso incorrecto en el cual la información contiene un fallo.

4.3.6.1 Caso correcto – información correcta

Se envía información la cual se sabe de antemano que el sistema acepta, esto con el objetivo de comprobar que el sistema actúe de la forma planeada.

Validaciones para esta prueba:

- Código de respuesta.
- Respuesta enviada por la *Api*:

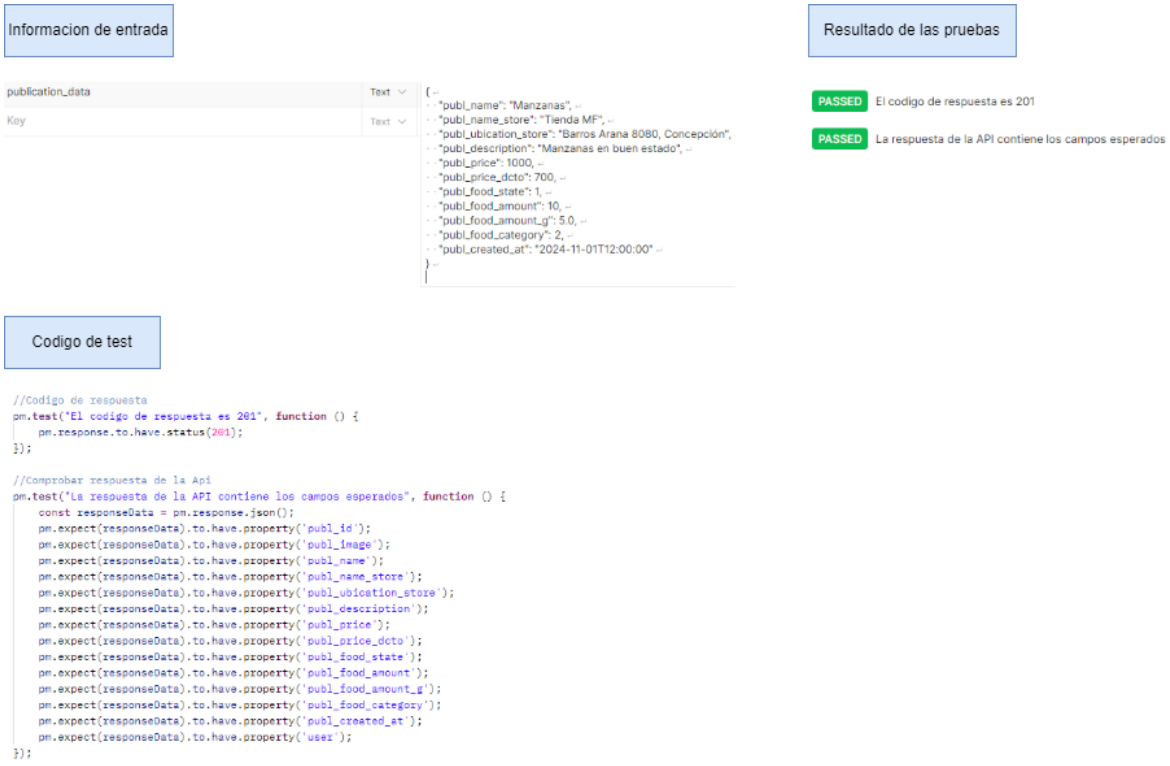

Figura 30: Caso correcto - información correcta.
Fuente Elaboración propia.

4.3.6.2 Caso incorrecto – información incorrecta

Se envía información incorrecta a la Api con el objetivo de comprobar que la *Api* actúe de la forma en la que se dijo que actuaría.

Validaciones para esta prueba:

- Código de respuesta.

Figura 31: Caso incorrecto - información incorrecta.
Fuente: Elaboración propia.

4.3.7 Resultados

Como se mostró en el anterior apartado, cada uno de los *tests* fueron exitosos debido a que el programa respondía de la forma planteada.

4.4 *Test* de compatibilidad con el resto de las funciones

Se comprueba que las funcionalidades sean capaces de usarse unas con otras, dando ejemplo de esto es que, si tenemos una funcionalidad para ver productos y otra de agregar productos, cuando se agregue un producto este se vea en la de ver productos. Sé uso este tipo de pruebas ya que es necesario comprobar las interacciones entre las diferentes funcionalidades.

4.4.1 Cantidad de casos

Este apartado consta de 3 casos donde se demuestra que ciertas funcionalidades hacen cambios, estos pueden ser visibles con el uso de otras funcionalidades dentro de la aplicación.

4.4.2 Prueba exitosa

Para que una prueba de esta clase se considere exitoso es necesario que las funcionalidades que se están probando interactúen de la forma en que se tiene planteada, en otras palabras, que los cambios hechos por una funcionalidad se puedan ver reflejadas en otra.

4.4.3 Agregar alimento y ver alimentos en el inventario

Tenemos la funcionalidad dedicada a agregar un alimento y otra dedicada a ver el inventario del usuario, el plan es agregar un alimento (en este caso se usará la funcionalidad de agregar productos con código de barras) y luego comprobar si se puede ver ese alimento mediante la funcionalidad de obtener productos.

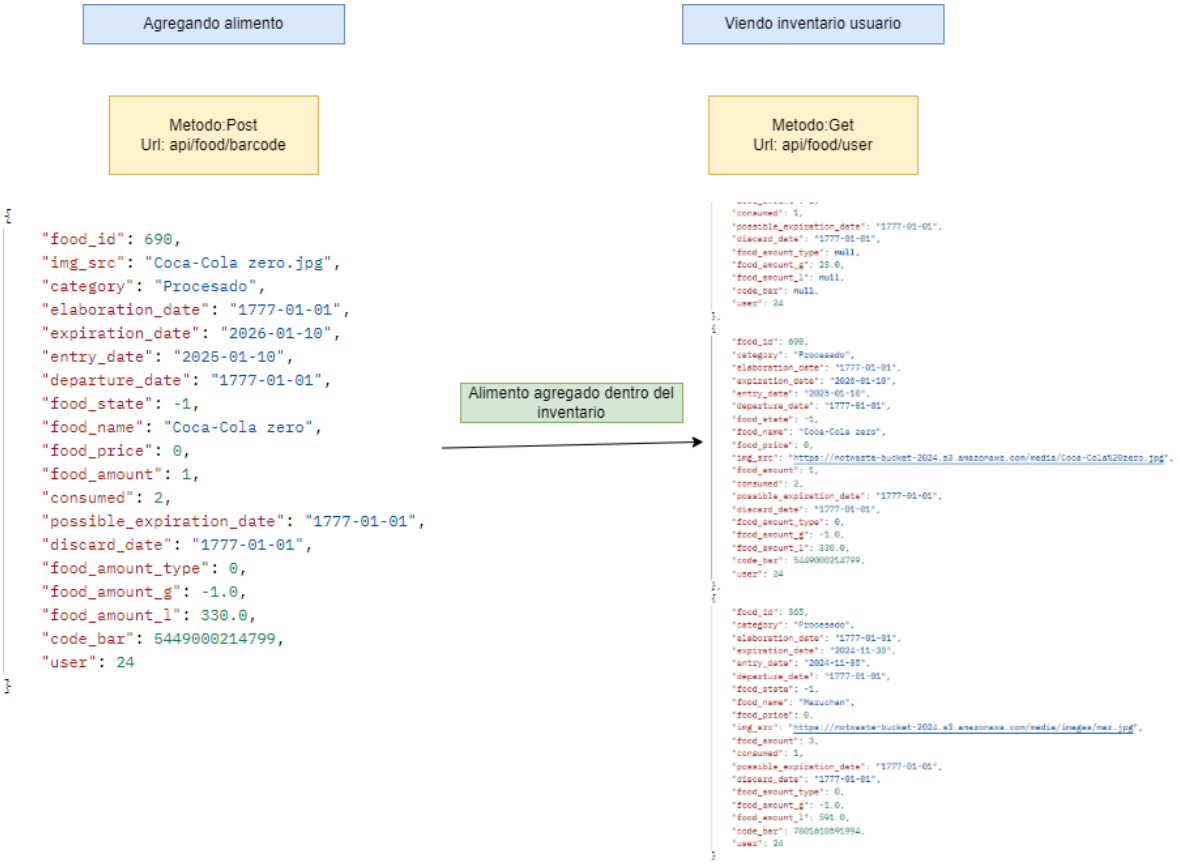

Figura 32: Compatibilidad Agregar alimento - Ver inventario.
Fuente: Elaboración propia.

4.4.4 Agregar producto y ver recetas

Como se explicó anteriormente las recetas van relacionadas a los productos que tiene un usuario dentro de su inventario, haciendo que si dentro de los alimentos de un usuario se encuentre un alimento que sea un ingrediente para una receta, esta receta ahora se le mostrará al usuario. Pará esta prueba se eliminará los alimentos del usuario, luego de esto se agrega un alimento que se usa como ingrediente en una receta, luego de esto se comprobará que se pueda ver la receta usando la funcionalidad de ver recetas.

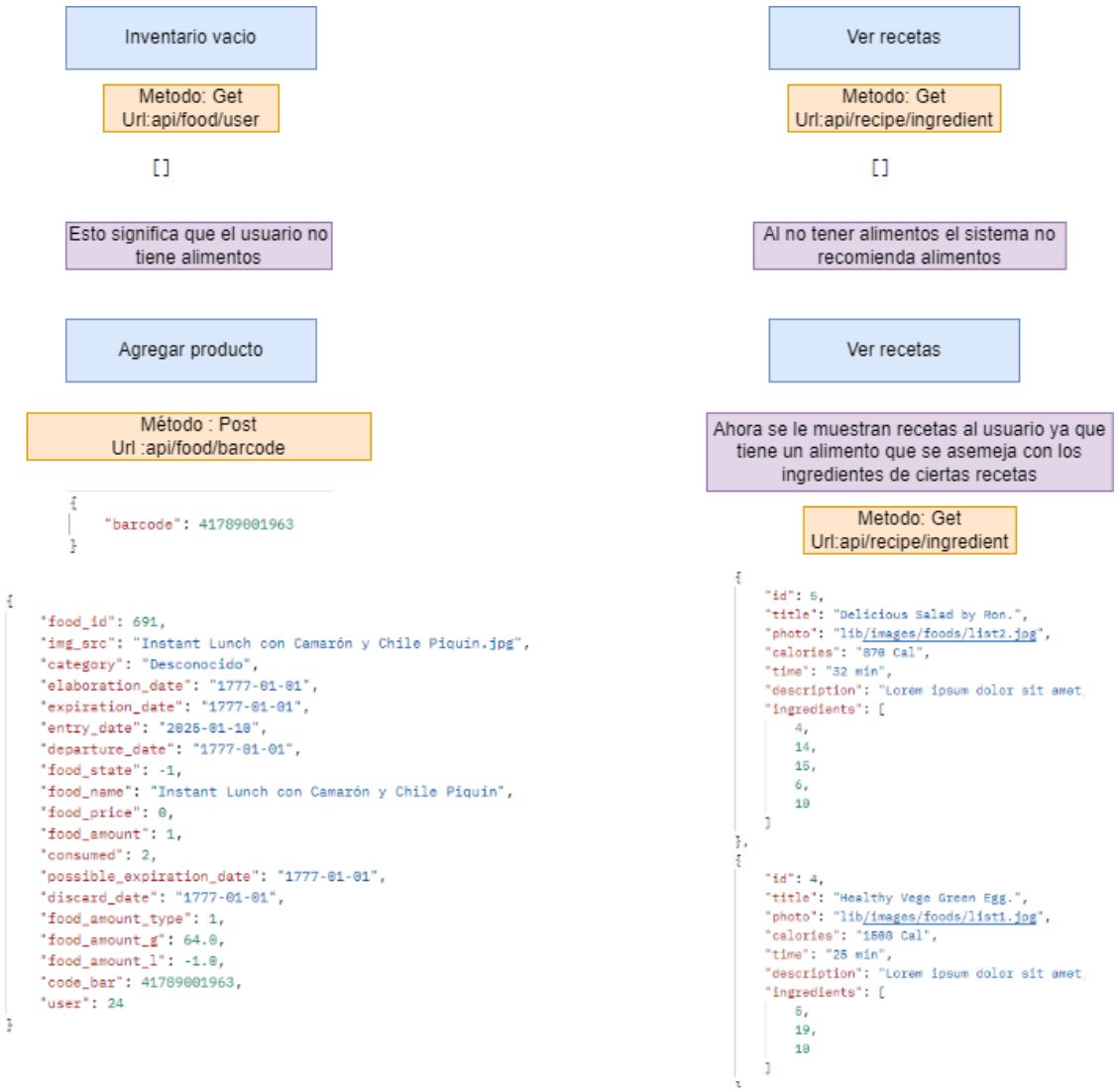

Figura 33: Compatibilidad Agregar producto - Ver recetas.
Fuente: Elaboración propia.

4.4.5 Crear publicación y ver publicaciones

Parecido a la prueba hecha en el punto 4.3.3 solamente que ahora se hace con las funcionalidades relacionadas con las publicaciones. Crear una publicación y luego comprobar que esta se pueda apreciar con la funcionalidad de ver publicaciones.

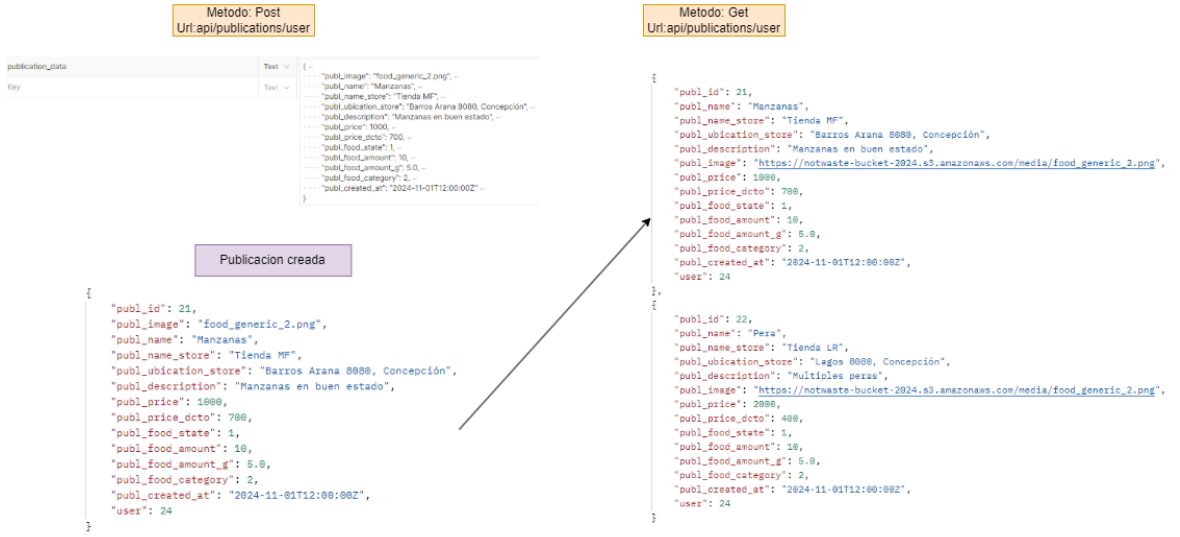

Figura 34: Compatibilidad Crear recetas - Ver recetas.
Fuente: Elaboración propia.

4.4.6 Resultados

Se indica que los *tests* de compatibilidad fueron exitosos debido a que se verifico que las diferentes funciones pueden interactuar entre ellas.

4.5 Resumen de las pruebas

Dado que se mostraron los 2 tipos de pruebas hechas para la verificación del funcionamiento del sistema es momento de dar un fin a este capítulo de esta memoria con un resumen de este.

Todas las pruebas terminaron dando resultados satisfactorios mostrando que el sistema funciona como se diseñó, en la primera sección del capítulo 4 en la cual se enfoca sobre los *test* unitarios , se demostró que todos estos cumplieron con los objetivos necesarios de sus pruebas respectivas ya que todos obtuvieron el aprobado, y en la segunda parte del capítulo el cual se enfoca en *test* colaborativos, todos estos fueron exitosos ya que como se vio en las imágenes cuando se hace algo con una funcionalidad, esté cambio se ve reflejado en otras funcionalidades.

CAPÍTULO 5: CONCLUSIONES

Finalmente, luego de explicar pasar por todos los apartados de la creación del *backend* de la aplicación móvil, llego el momento de concluir esta memoria con su capítulo final, en este se presentarán diversos puntos e ideas respecto al desarrollo de *Not Waste* y que podría pasar en un futuro con respecto a este proyecto.

5.1 Conclusiones respecto a *Not Waste*

Como conclusión a todo lo que es *Not Waste*, este fue un trabajo que enseño bastante pasando por diferentes etapas, fases iniciales como la investigación del problema hasta apartados más avanzados como la construcción de una aplicación móvil, todo esto fue un trabajo largo y complicado pero que dejo bastante entendimiento respecto a varios temas, conceptos que igualmente fueron reforzados durante la escritura de esta memoria y que con la lectura de esta, se espera que los posibles lectores tengan más informaciones de los distintos temas tocados en este documento y se invita a que si algún tema llamo su atención, investigue a profundidad por cuenta propia.

5.2 Conclusiones respecto a objetivos

Ahora es momento de recordar los objetivos de esta memoria presentados durante el capítulo 1, así que se dará una conclusión alrededor de estos.

- Objetivo Principal: Este objetivo fue cumplido, y se puede apreciar esto viendo todo el desarrollo del *backend* mostrado aquí en este documento, pero en puntos simples es esto.
    - Creación de la *BDD*.
    - Creación de la *API.*
    - Conexión entre estas 2.
    - Conexión con la aplicación móvil.
    - Pruebas de las funcionalidades comprobando que funcionan de forma correcta.
- Crear las funcionalidades de la aplicación: Este objetivo se puede dar por cumplido, ya que como se explicó durante el capítulo 3, se crearon las diferentes funcionalidades de la aplicación móvil, a continuación, se nombrarán algunas.
    - Obtener productos del inventario.

  - Obtener recetas.
  - Ingreso con código de barras.

- Realizar pruebas de las funcionalidades para la comprobación de que estas funcionen de la forma en la que se planificó: Fue concluido satisfactoriamente como se pudo comprobar en el capítulo 4 de este documento, sección que trata acerca de los diversos *test* que se hicieron en las funciones los cuales fueron 2.
  - *Test* unitario.
  - *Test* de compatibilidad.
- Verificar y corregir el código de la aplicación y de la *Api* de esta, para comprobar que cumplen con los estándares de código limpio: Esto fue hecho, pero ya que es algo más del código no se entró en profundidad dentro de la memoria.

5.3 Limitaciones

Actualmente la aplicación se encuentra con limitaciones debido a varias razones en las que se encuentran la arquitectura de la aplicación y la *Api* externa de *Open.* Con respecto a primer problema este se genera debido a que si repentinamente llega una gran cantidad de datos a secciones de la arquitectura (usuarios , comidas de los usuarios ,etc.) los sistemas dentro de esta van a empezar a fallar debido a que no soportan una gran cantidad de operaciones, y con respecto al segundo problema es cual es la *Api* externa , este se origina debido a que la aplicación *Not Waste* tiene una fuerte dependencia de esta , ya que al necesitarla para una funcionalidad dentro de la aplicación ,dependemos de que la *Api* funcione como es previsto y que tenga el producto que se le pide. Las posibles soluciones para estos dos problemas se mencionan en el siguiente apartado.

5.4 Trabajo futuro

Con la naturaleza de *Not Waste* de ser una aplicación hecha solamente para la universidad y que este proyecto no se seguiría desarrollando finalizado, sería ineficiente pensar en el futuro de esta, pero en el caso de que por alguna razón se volviera a continuar la aplicación con vistas a ser lanzada al público se debería considerar estas ideas.

- Base de datos propia: Se debería considerar la idea de tener una *BDD* propia con los alimentos que se pueda tener en la aplicación esto con la idea de dejar de depender de *una Api* externa que no se pueda controlar

internamente haciendo que se dependa de un tercero para que esta funcione.

- Escalar los sistemas involucrados en la arquitectura de la aplicación: Crear un escalamiento horizontal y vertical con respecto a la arquitectura para poder soportar una mayor cantidad de usuarios en la aplicación. Él escalamiento vertical mejora los recursos de un equipo, y el escalamiento horizontal se refiere a aumentar la cantidad de servidores.

- Modificación de funcionalidades actuales: Se debe considerar dar una revisada a las funcionalidades actuales para plantear una mejora en el rendimiento, efectividad o de plano cambiar totalmente la funcionalidad para enfocar desde un punto de vista nuevo, sé toma este punto a futuro debido a que actualmente no se plantea cambios a las funcionalidades actuales.

- Creación de nuevas funcionalidades: Añadir nuevas funciones a la aplicación las cuales no se añadieron debido a falta de tiempo o que directamente no se imaginaron para agregar.

- Creación de un nuevo plan de pruebas para las funcionalidades: Dado que en el caso de que se cumplan varios puntos anteriores es necesario crear un nuevo plan de pruebas para la aplicación, esto tomando en cuenta las nuevas características del proyecto.

- Portabilidad a nuevos dispositivos: La aplicación móvil solamente permite sistemas *Android*, en el caso de continuar con el desarrollo de la aplicación es un buen punto agregar configuraciones para el lanzamiento en dispositivos *iOS*.

REFERENCIAS BIBLIOGRÁFICAS

Amazon Web Services (AWS). (2024). *Amazon Relational Database Service (RDS)*. Recuperado de https://aws.amazon.com/es/rds/#:~:text=Amazon%20Relational%20Database%20Service%20(Amazon,el%20costo%20total%20de%20propiedad.

Amazon Web Services (AWS). (2024). *Conceptos básicos de Amazon EC2*. Recuperado de https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html

Amazon Web Services (AWS). (2024). *Preguntas frecuentes sobre Amazon S3*. Recuperado de https://aws.amazon.com/es/s3/faqs/

División de Población de las Naciones Unidas. (2024). *World Population Prospects*. Recuperado de https://population.un.org/wpp/

Ginzo Tech. (2024). *Metodología clásica de desarrollo de software*. Recuperado de https://ginzo.tech/metodologia-clasica-desarrollo-software/

Ipsos. (2023). *Maggi e Ipsos presentan el primer estudio sobre desperdicio de alimentos en Chile*. Recuperado de https://www.ipsos.com/es-cl/maggi-e-ipsos-presentan-el-primer-estudio-sobre-desperdicio-de-alimentos-en-chile

Lecciones Aprendidas. (2014). *Tabla comparativa entre metodologías ágiles*. Recuperado de https://www.lecciones-aprendidas.info/2014/07/tabla-comparativa-entre-metodologias.html

Lucidchart. (2024). *¿Qué es un modelo de base de datos?* Recuperado de https://www.lucidchart.com/pages/es/que-es-un-modelo-de-base-de-datos/

Microsoft. (2024). Conceptos básicos de pruebas unitarias en Visual Studio. Recuperado de https://learn.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2022

Open Food Facts. (2024). *A food products database*. Recuperado de https://world.openfoodfacts.org/discover

Real Academia Española. (2024). Metodología. En *Diccionario de la lengua española* (23.ª ed.). Recuperado de https://dle.rae.es/metodolog%C3%ADa

Red Hat. (2022). *¿Qué es la metodología ágil?* Recuperado de https://www.redhat.com/es/topics/devops/what-is-agile-methodology

Statista. (2024). *Tamaño del mercado mundial de ingredientes alimentarios especiales en 2022.* Recuperado de https://es.statista.com/estadisticas/1411229/tamano-del-mercado-mundial-de-ingredientes-alimentarios-especiales/#:~:text=El%20tama%C3%B1o%20de%20mercado%20de%20la%20industria,d%C3%B3lares%20estadounidenses%20a%20nivel%20mundial%20en%202022.