

Taperfy

Memoria de Proyecto Final de Grado/Máster
Máster Universitario en Desarrollo de Sitios y Aplicaciones Web
Informática, Multimedia y Comunicación

Autor: Juan Jesús Gutiérrez Ramos

Consultor: César Pablo Córcoles Briongos / Miguel Calvo Matalobos
Profesor: César Pablo Córcoles Briongos / Miguel Calvo Matalobos

Junio 2021

Créditos/Copyright



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

A mí madre, que sin tener ni idea de informática me enseñó todo lo necesario para llevar a buen puerto este proyecto y este máster.

Abstract

El proyecto surge con la idea de crear una plataforma que pueda poner en contacto a personas dispuestas a cocinar para los demás (y vender porciones) con potenciales consumidores de táperes, a los que les gusta la comida hecha en casa, pero no tienen el tiempo o la habilidad necesaria para hacerla. Será pues, un sistema de intercambio comercial de un producto específico que pueda proporcionar una fuente de ingresos a algunas personas (productores) y pueda ayudar a otras (consumidores) en dos necesidades especialmente presentes en la juventud: comer bien y ahorrar tiempo.

Palabras clave: comida casera, táper, angular, laravel.

Abstract (english version)

The main goal of this project is offer a tool to people who want sell his home-made food to other people who don't have time to cooking (or not know how-to cook). It's pretend to offer a way to monetize his knowledge to that persons who make good meals and offer to the costumers the possibility of save time and get healthy food.

Keywords: homemade food, tuppens, angular, lavavel.

Índice

1. Introducción	10
1.1 Contexto y justificación del trabajo	11
1.2 Objetivos del trabajo	12
1.3 Planificación del trabajo	13
2. Descripción	16
3. Objetivos	20
3.1 Principales.....	20
3.2 Secundarios.....	20
4. Escenario	22
5. Metodología.....	25
6. Arquitectura de la aplicación/sistema/servicio	27
7. Plataforma de desarrollo.....	31
8. Planificación	32
9. Frameworks, APIs y herramientas utilizadas.....	33
10. Diagramas.....	34
10.1 Base de datos, diagrama entidad relación.....	35
10.1 User StoryMap	36
11. Prototipos.....	37
11.1 Lo-Fi	37
11.2 Hi-Fi.....	41
12. Perfiles de usuario	49
13. Usabilidad/UX	52
14. Seguridad	56
15. Tests.....	61
16. Versiones de la aplicación.....	64
17. Consideraciones de implementación	66
17.1 Uso del patrón Redux y NgRx	66
17.2 Tailwind y Diseño web responsive	68
17.3 Animaciones.....	70
17.4 PWA	71
17.5 Rendimiento, lazy loading y precarga de módulos	73
17.6 Chat: polling vs websockets	76

17.7 Envío de correos	80
18. Requisitos de instalación	82
19. Instrucciones de instalación	83
19.1 Instalación del backend	83
19.2 Instalación del frontend.....	88
20. Instrucciones de uso.....	90
21. Proyección a futuro.....	91
22. Presupuesto.....	94
23. Conclusiones.....	96
Anexo 1. Entregables del proyecto.....	97
Anexo 2. Código fuente (extractos).....	98
A2.1 Backend	98
A2.1 Frontend	102
Anexo 3. Librerías/Código externo utilizado	109
Anexo 4. Capturas de pantalla.....	110
Anexo 5. Guía de usuario.....	114
A5.1 Cocinero.....	116
A5.2 Comprador.....	119
Anexo 6. Libro de estilo	124
Anexo 7. Bibliografía	125
Anexo 8. Vita.....	127

Figuras, tablas e ilustraciones

Índice de figuras

Figura 1: Comida casera, difícil de encontrar en aplicaciones	10
Figura 2: PECs propuestas en la asignatura del PFM.....	13
Figura 3: Tipos de comida más demandados por provincia en la plataforma JustEat.....	23
Figura 4: Metodología de desarrollo en cascada	25
Figura 5: Arquitectura de la aplicación	27
Figura 6: Elementos de una aplicación angular.....	29
Figura 7: Arquitectura framework angular.....	30
Figura 8: Patrón de diseño REDUX	30
Figura 9: Planificación del proyecto.....	32
Figura 10: Modelo entidad relación de la aplicación.....	35
Figura 11: User StoryMap con las historias de usuario de Taperfy	36
Figura 12: Boceto de la pantalla de login	37
Figura 13: Boceto de la pantalla de registro de usuarios	38
Figura 14: Boceto de la pantalla de inicio en escritorio y móvil.....	38
Figura 15: Boceto de la pantalla de inicio en escritorio y móvil.....	39
Figura 16: Boceto de la pantalla que muestra el detalle de un taper.	39
Figura 17: Boceto de chats entre usuarios	40
Figura 18: Boceto de la pantalla de peticiones realizadas por el comprador.....	40
Figura 19: Boceto de la pantalla de solicitudes al cocinero.....	41
Figura 20: Prototipo Hi-Fi de la pantalla de login (desktop y móvil)	42
Figura 21: Prototipo Hi-Fi de la pantalla de registro (desktop y móvil)	43
Figura 22: Prototipo Hi-Fi de la pantalla principal(desktop y móvil)	44
Figura 23: Prototipo Hi-Fi de la pantalla de nuevo taper(desktop y móvil).....	45
Figura 24: Prototipo Hi-Fi de la pantalla de detalle de taper(desktop y móvil)	46
Figura 25: Prototipo Hi-Fi de la pantalla de chat entre usuarios	47
Figura 26: Prototipo Hi-Fi de la pantalla de peticiones de reserva del comprador	47
Figura 27: Prototipo Hi-Fi de la pantalla de solicitudes recibidas por el cocinero (desktop y móvil)	48
Figura 28: El usuario puede pasar de cocinero a comprador y viceversa desde la edición de su perfil.....	49
Figura 29: Campo de BBDD que indica si el usuario es cocinero o comprador.....	50
Figura 30: Parte del estado que nos indica si es cocinero, en el NgRx Dev Tools	51
Figura 31: Varias capturas que muestran información del estado de la aplicación.....	53
Figura 32: Token almacenado en el state de la aplicación angular	57
Figura 33: Constraseñas cifradas en base de datos.....	58
Figura 34: Página indicando al usuario que no tiene permisos para ejecutar la ruta	60
Figura 35: Informe tras la ejecución de test unitarios, web.....	63
Figura 36: Informe tras la ejecución de test unitarios, consola	63
Figura 37: Redux / Ngrx.....	67
Figura 38: Interfaz adaptada a diferentes pantallas	69
Figura 39: Instalando Taperfy como una PWA.....	71
Figura 40: Taperfy como PWA, instalada en un teléfono móvil.....	72
Figura 41: Taperfy como PWA, instalada en un MacBook.....	72
Figura 42: Tabla para almacenar los mensajes	76
Figura 43: Notificaciones dentro de la aplicación	80
Figura 44: Correos enviados por la plataforma	81
Figura 45: Versión de Angular	82
Figura 46: git clone del proyecto backend.....	83
Figura 47: Instalación de dependencias del backend con composer	84
Figura 48: Configuración de la conexión del backend con la BBDD	84
Figura 49: Base de datos de la aplicación, vista en phpmyadmin	85
Figura 50: Ejecución de la migración para crear el modelo de datos	86
Figura 51: db:seed para poblar la base de datos con datos de prueba	86
Figura 52: Visualización en el navegador de los resultados devueltos de la API	87
Figura 53: git clone del proyecto frontend	88
Figura 54: ng serve para arrancar el proyecto front.....	88
Figura 55: Acceso al front-end desde el equipo local.....	89
Figura 56: Capturas de pantalla de la aplicación en su versión móvil.....	110
Figura 57: Capturas de pantalla de la aplicación en su versión tablet.....	111
Figura 58: Capturas de pantalla de la aplicación en su versión de escritorio	113
Figura 59: Paleta de colores de la aplicación.....	124

Índice de tablas

Tabla 2: Planificación del proyecto	15
Tabla 1: Fases de la metodología en cascada	26
Tabla 3: Recursos tecnológicos usados durante la creación del proyecto	31
Tabla 5: Listado de Frameworks, apis y herramientas utilizadas	33
Tabla 5: Versiones de la aplicación	65
Tabla 6: Tabla de costes de la construcción del proyecto	94
Tabla 7: Coste total de Taperfy	95

Ilustraciones (fragmentos de código)

Ilustración 1: Obtención de un token JWT a partir de credenciales del usuario	57
Ilustración 2: Inclusión del token JWT en las cabeceras http enviadas por el cliente.....	58
Ilustración 3: Route Guard para proteger rutas que necesitan autenticación.....	59
Ilustración 4: Asociación de un Route Guard a una ruta de la aplicación	59
Ilustración 5: Test unitarios en Angular	62
Ilustración 5: Personalización del framwork Tailwind.....	68
Ilustración 6: Implementación de diseño responsive en tailwind.....	69
Ilustración 7: Definición de animación con angular animations	70
Ilustración 8: Ejemplo de aplicar una animación angular a un elemento	70
Ilustración 9: Registro del service worker.....	71
Ilustración 10: Carga de los módulos de la aplicación	74
Ilustración 11: Pre-loading de módulos	75
Ilustración 12: Parte de la API relacionada con el intercambio de mensajes	76
Ilustración 13: Consulta al backend por mensajes nuevos del usuario.....	77
Ilustración 14: Configuración del servidor de correos en el backend	80
Ilustración 15: Envío de emails en el backend	80
Ilustración 16: Definición de la api rest del backend	98
Ilustración 17: Controlador que gestiona las reservas	101
Ilustración 18: Models, factories, seeders y migrations	102
Ilustración 19: Componente angular app-taper-card	103
Ilustración 20: Ejemplo de construcción de un formulario reactivo.....	104
Ilustración 21: Fragmentos de código REDUX / ngrx	105
Ilustración 22: Prefijos tailwind.....	106
Ilustración 23: Invocación a la api con BackendService	107
Ilustración 24: Operaciones con observables	107

1. Introducción

Es un tema manido, un lugar común, una expresión que casi ha perdido su significado de tanto usarse, pero que no deja de ser verdad: la dieta mediterránea es oro. Un patrimonio que tenemos a nivel de país que nos da calidad y nos alarga la vida, un tesoro en forma de recetas tradicionales que ha ido pasando de padres a hijos (quizá más bien de madres a hijas) durante generaciones en una cadena que la falta de tiempo, la aceleración a la que nos somete constantemente la vida moderna y las influencias de otras cocinas quizás no tan sanas, está a punto de romperse. No quiero pecar de agorero, pero veo cierto desapego de los más jóvenes por los sabores de siempre, la comida de cuchara, la más lenta, más sosegada, más sana. Quizás entre otros muchos factores es que no estamos ofreciendo un acceso lo suficientemente fácil y atractivo a este tipo de comida.

Mientras escribo esto se va acercando la hora de cenar, son las 20.00 de la tarde. Vivo en una ciudad mediana, abro una popular aplicación de repartos a domicilio y esto es lo que ve portada: dos conocidas cadenas de hamburgueserías, un restaurante especializado en pollo frito, 2 italianos y un turco. Esta muy bien tener todas esas opciones y sin duda es atractivo para según qué público o según que momento puntual, pero estoy convenido de que junto a esta ‘colonización gastronómica’ hay un hueco para los platos tradicionales, los que te prepararía tu madre o tu abuela, la slow-food si se quiere hablar en esos términos. Creo firmemente que podría tener un mercado que podría llenar el proyecto que aquí se propone.

Porque la falta de tiempo es un problema real que nos afecta a todos los jóvenes y no parece que vaya a ir a mejor. De nuevo hablo desde un punto de vista estrictamente personal: yo intento hacer batch-cooking u otras técnicas de optimización para tratar de cocinar menos tiempo, pero aún así, hay semanas que no llego. Con niños, trabajo y estudios, muchas veces es imposible sacar el tiempo necesario para la compra y elaboración de comida para la semana, y no creo estar en una situación especial en absoluto, muchas personas de mi entorno y generación andan más o menos igual. En esencia, esta plataforma persigue hacer posible intercambiar dinero por tiempo y salud, algo a lo que pienso que estaría dispuesta mucha gente.

Claro que existen comidas y platos preparados en las grandes cadenas de supermercados, pero el ‘toque casero’ es algo que pienso que no ha sido conseguido por ninguna marca, por no hablar de la sensación de estar comprándole directamente a alguien cercano, revirtiendo en la economía de tu localidad.



Figura 1: Comida casera, difícil de encontrar en aplicaciones

Otra dimensión que podría tener este proyecto que me gustaría destacar en esta introducción: el punto de vista de los usuarios ‘productores de táperes’ (utilizaré en esta memoria la palabra táper una vez fue introducida por la rae como una acepción válida en castellano). Lo plantearé de nuevo desde mi visión personal: en mi pueblo, Chiclana (en Cádiz) en los años 60/70 surgió una empresa con un modelo de negocio que puede ser discutido, pero fue a su manera innovador: las muñecas de Marín. Estas muñecas (normalmente vestidas de flamenca que servían de decoración, no entraremos en consideraciones estéticas aquí), y concretamente sus trajes, eran tejidos por una red de amas de casa que trabajaban desde su salón: confeccionando y cosiendo pequeños vestidos. La empresa por un lado se servía de un conocimiento que hasta el momento nadie había sabido usar a esa escala: todas las amas de casa de la época sabían coser. Por otro lado, estas trabajadoras (que de otra forma no hubieran trabajado con un horario estricto, desplazándose, etc..) podían monetizar un conocimiento adquirido y que a todas luces tenía valor.

Esa misma filosofía y principio podría latir en nuestra aplicación: convertir en dinero algo que tiene muchísimo valor y que rara vez ha sido monetizado: el saber hacer en cuanto a la cocina tradicional de mucha gente. Por ejemplificarlo de forma resumida y clara y sin ningún tipo de ironía, para mí, el conocimiento que tiene, por decir alguien, mi suegra, en lo referente a su recetario personal, trucos de cocina, saber hacer y el toque que imprime a lo que hace, tiene valor. La idea que subyace en Taperfy es explotar ese valor y proporcionar una herramienta que permita convertirlo en dinero.

Todo lo mencionado anteriormente en esta introducción es una defensa del proyecto desde un punto de vista personal. Se debe entender el alcance real y las posibilidades de hacer algo completo y ‘real’ dentro del ámbito de un proyecto de fin de máster. En cualquier caso, pienso que podría ser un proyecto interesante, relevante y que me ayude a poner en práctica todo lo aprendido durante estos meses.

1.1 Contexto y justificación del trabajo

Aunque en el texto anterior se ha hecho una defensa de la utilidad del proyecto y su viabilidad, a continuación, se listará de una forma más esquemática el porqué considero que tiene sentido la implementación del proyecto, así como su adecuación como trabajo de fin de Máster.

Contexto actual, en el que pienso que tiene cabida una aplicación de intercambio de comida casera:

- En la actualidad, muchas personas tienen problemas de tiempo debido a las exigencias de la vida diaria: trabajo, niños, actividades extraescolares, estudios, etc. En este sentido, cualquier plataforma que nos ofrezca algo que permita ahorrar tiempo (en nuestro caso, el tiempo de la compra y cocinar) creo que puede ser bienvenida.
- Pienso que las tradiciones gastronómicas y las recetas que pasan de generación en generación se están de alguna forma perdiendo. Este cambio está motivado en parte porque ahora es más habitual que las nuevas generaciones no residan en el mismo sitio que sus padres, porque marchan a estudiar o a trabajar.
- En su mayor parte, las plataformas más conocidas que ofrecen comida suelen ofertar opciones de comida rápida, o en el mejor de los casos, comida de fuera. Es muy raro ver comida de ese otro tipo y es innegable que hay un interés real por la alimentación saludable, movimientos como el reciente real food lo demuestran. Por ejemplificarlo de alguna forma, es muy difícil encontrar unas lentejas por Glovo, o platos con legumbres, etc.
- Como se comentó en el punto introductorio, esta web puede ser, a su escala, una buena herramienta para proporcionar unos ingresos (aunque sean modestos) a cierto perfil de

personas que tienen muy difícil el acceso al mercado laboral por otras vías y sin embargo cuentan con un saber hacer y conocimiento perfectamente valioso y monetizable.

Por otro lado, cambiando el punto de vista para centrarnos en la parte más técnica del proyecto, creo que podría adecuarse perfectamente como proyecto de fin de máster ya que me permitirá poner en práctica buena parte de los conocimientos adquiridos en las diferentes asignaturas que componen estos estudios.

- Por un lado, habrá que desarrollar un backend que ofrezca una API Rest y pueda explotar la base de datos de la aplicación. Esto se podría desarrollar con el framework de desarrollo phpLaravel aplicando lo estudiado en la asignatura Desarrollo Backend con PHP.
- El front se podría desarrollar con Angular aplicando los conocimientos sobre todo de las asignaturas ‘Desarrollo front-end’ y ‘Desarrollo front-end avanzado con frameworks’. En principio, casi todo lo necesario está estudiado en el master: consumo de una api rest, formularios, búsquedas, filtrado, etc aunque será necesario realizar cierta investigación para ver como se realiza la autenticación de las llamadas a la api, de que forma se geolocalizan los productos, etc.
- Respecto a la interfaz, se intentarán aplicar los conocimientos aprendidos en las asignaturas de diseño de interfaces y HTML, y se propone usar el frameworkcssTailWind tal y como se vio en el curso ‘Herramientas HTML y CSS II’. Adicionalmente, se intentará usar en la medida de lo posible el boilerplate propuesto por la UOC así como las diferentes técnicas de optimización de imágenes, adaptación de interfaces web a móviles, etc. vistas durante el Master.

1.2 Objetivos del trabajo

Con lo descrito anteriormente, podemos enumerar una serie de objetivos que definen nuestra aplicación tanto desde un punto de vista funcional como en otros aspectos igualmente importantes en la consecución de un producto final de calidad.

Objetivos funcionales de nuestro sistema:

- Ofrecer la posibilidad a usuarios registrados de vender su propia comida preparada en casa, en forma de raciones individuales, generalmente conocidas como tupperes.
- Permitir a esos ofertantes de comida preparada describir su producto, subir fotos y establecer el precio que consideren oportuno.
- Posibilitar a un usuario (registrado o no) buscar comida preparada casera en su zona, mostrando un formulario de búsqueda por diferentes criterios y pudiendo aplicar filtros sobre los resultados obtenidos.
- Permitir a un usuario registrado reservar un tupper de su elección.
- Una vez que se haya realizado la transacción y el vendedor de la comida marque el producto como vendido, posibilitar al comprador que valore la calidad del producto, así como que marque al cocinero como favorito.
- Posibilitar el envío de mensaje de usuarios registrados a sus cocineros favoritos, con comentarios, peticiones, sugerencias, etc.

Otros objetivos, no tan relacionados con la funcionalidad como con otros aspectos igualmente importantes para que el resultado final sea un producto de calidad:

- La aplicación debe ofrecer una interfaz amigable para el usuario y fácil de usar.
- Debe ser completamente funcional tanto en su versión escritorio, como cuando se visualice en una tablet o teléfono móvil.
- La aplicación debe responder en un tiempo aceptable tanto a la subida de platos como en la búsqueda y el filtrado de estos.

Para una descripción más detallada de la funcionalidad que define el sistema, así como de los otros aspectos de diseño, rendimiento, etc. se debe acudir al documento anexo *PACI_rec_Gutiérrez_Ramos_Juan_Jesús* donde se detallan, en forma de historias de usuario, todas las características de la aplicación.

1.3 Planificación del trabajo

En la planificación de la asignatura se propone subdividir el trabajo en cuatro grandes bloques: uno inicial más corto relacionado con la exposición y descripción del proyecto y otros tres de aproximadamente un mes de duración.

Esta distribución me parece acertada porque el hecho de tener fases más o menos largas de un mes (con entregables definidos al final de cada fase) me permite adaptar la carga de trabajo a mi día a día. En mi caso, que estoy trabajando (y mi carga de trabajo en mi empresa no es para nada uniforme) y tengo que cuidar de un niño pequeño, no es muy realista pensar que puedo comprometer entregas cada semana. De hecho, una previsión más real es que compagine semanas de no poder casi avanzar con el proyecto con semanas de dedicación exclusiva.

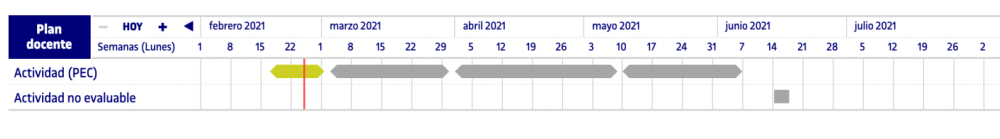


Figura 2: PECs propuestas en la asignatura del PFM

A continuación, se detalla como concibo cada una de las fases propuestas en la asignatura respecto al desarrollo del proyecto, qué espero hacer en cada una de estas fases y cual es el objetivo principal de las mismas.

Al igual que en el punto anterior, espero que esta planificación me sirva de guion para orientar y fasear el esfuerzo, pero soy consciente de que habrá mucho trabajo y fases en paralelo. Por ejemplo, el testing, aunque esté especialmente presente en una fase, quiero que sea continuo a lo largo del proyecto, idealmente el modelo de datos estará definido en la segunda fase, pero la experiencia me dice que este tipo de definiciones nunca suelen estar cerradas del todo, etc.

PEC 1
Periodo: del 17/02 al 02/03 de 2021
Descripción
Esta primera fase servirá como toma de contacto, para presentar el proyecto, moldear la idea, así como hacer una descripción general del sistema que se quiere desarrollar.
Objetivos
Tener claro lo que se quiere construir, el alcance del proyecto y validar la idea con distintas personas para recibir feedback y aportes. A nivel de software no se va a dar ningún paso, pero si se pretende tener una base sólida sobre qué tecnologías, herramientas y frameworks se van a usar durante la construcción del proyecto.

Se redactarán los primeros puntos de la documentación destinados a la exposición y justificación de la idea elegida.
--

PEC 2
Periodo: del 03/03 al 31/03 de 2021
Descripción
<p>Este segundo periodo, que abarca un mes, considero que de alguna forma será el más complicado de abordar. Resumiendo, podríamos decir que en la PEC 2 ‘echaremos a andar el proyecto’.</p> <p>Por un lado, se describirá con absoluto detalle la funcionalidad y requisitos de otro tipo que debe cumplir la aplicación, desarrollando wireframes de las pantallas y teniendo una primera versión del modelo de datos de la aplicación.</p> <p>Tras la decisión de qué tecnologías usar y qué herramientas, se definirá un esqueleto de aplicación que sirva de base para el desarrollo del proyecto. Este esqueleto o boilerplate deberá tener configurada las herramientas que se vayan a incluir y debe existir uno tanto para el front en Angular como el back en Laravel.</p> <p>A nivel de diseño, sin entrar en maquetación aún si que me gustaría hacer una pequeña investigación que me ayude a decidir líneas de diseño, paleta de colores, componentes de interfaz que me podrían ser útiles, etc.</p> <p>El último punto importante de esta fase es la investigación técnica. En los primeros compases de la aplicación, sé que la implementación de la plataforma debe abordar una serie de problemas técnicos en los que no tengo experiencia y que me requerirán cierta labor investigación.</p> <p>Tendremos que abordar estos problemas técnicos buscando en internet, proponiendo soluciones y haciendo alguna prueba de concepto. ¿Cómo se van a autenticar las peticiones REST que realiza el front al back? ¿Qué sistema utilizaremos para determinar la localización de un producto y del usuario (para mostrar tappers en la zona)? ¿Cómo gestionaremos la subida y custodia de las fotos en una plataforma que potencialmente debe almacenar miles y miles de imágenes?</p> <p>Todas estas preguntas deben plantearse previamente y tener una investigación y análisis hecha que den algo de luz para cuando estemos en la fase de implementación.</p>
Objetivos
<p>Tener definido perfectamente el alcance del proyecto, qué pantallas hay que implementar.</p> <p>Al haber decidido las herramientas y frameworks a utilizar, podríamos tener un esqueleto de aplicación ya preparado para la agilizar en la medida de lo posible la fase de implementación y no perder tiempo con la configuración del entorno / herramientas.</p> <p>Tras la conclusión de esta fase, aun sin tenerlo todo resuelto 100% pretendo haber abordado las cuestiones técnicas más peliagudas, habiendo realizado una investigación de cada uno de los problemas y teniendo alguna aproximación válida a la solución de cada uno de ellos.</p> <p>Aún siendo algo que estará ‘vivo’ durante todo el ciclo de desarrollo del proyecto, al finalizar esta fase me gustaría en la medida de lo posible tener cerrado el modelo de datos de la aplicación y una definición bastante exacta de qué métodos deberá ofrecer nuestra API Rest.</p>

PEC 3
Periodo: del 01/04 al 09/05 de 2021
Descripción
<p>Esta fase es la más larga y con mucha probabilidad la más intensa, ya que partiremos de nuestra base y esqueleto del proyecto para ir implementando una tras otra las funcionalidades definidas en la aplicación. El trabajo es doble, por un lado, debemos implementar un backend en Laravel que explote la base de datos definida en la fase anterior, así como ofrezca los métodos auxiliares necesarios que consideremos oportuno. Afortunadamente, como vi en la asignatura de</p>

<p>Backend, Laravel pone las cosas más o menos fáciles y tanto hacer un CRUD de una base de datos como implementar una apiRest son dos tareas muy comunes, bien documentadas y sencillas de llevar a cabo.</p> <p>Por otro lado, debemos implementar el backend con los conocimientos adquiridos en Angular, javascript así como hacer una maquetación del producto basado en las líneas de diseño y wireframes definidos en la fase anterior.</p> <p>Adicionalmente, habrá que abordar los problemas técnicos planteados previamente y los que vayan surgiendo. Estimo que esta fase será la de mayor carga y precisamente por eso es muy importante hacer un trabajo preciso en las fases anteriores para tener todo bien definido y no dar demasiados pasos en falso.</p> <p>En paralelo, se irán ampliando la memoria del proyecto, así como documentando todos los obstáculos y hechos reseñables que nos vayamos encontrando en el camino.</p>
<p>Objetivos</p> <p>El objetivo de esta fase es claro: tener una versión beta de nuestro proyecto, totalmente funcional, parcialmente testada y con el diseño más o menos definitivo.</p>

<p>Entrega final</p>
<p>Periodo: del 10/05 al 07/06 de 2021</p>
<p>Descripción</p> <p>En el último mes del proyecto, si todo va bien partiremos de una versión beta funcional de la aplicación que debemos probar en profundidad, tanto a nivel funcional como a nivel de interfaz, compatibilidad con dispositivos, posibles problemas de rendimiento, etc.</p> <p>Tras la prueba de la aplicación será necesario re-escribir algunas partes del código para corregir errores, hacer refactoring para aumentar la calidad del código escrito y pulir el diseño hasta que tenga una calidad aceptable para ser entregado y presentado.</p> <p>Mi idea es dar a probar el producto por distintos usuarios y que esos testers me reporten posibles problemas, mejoras a nivel de usabilidad, etc.</p> <p>Por último, en esta fase se terminará de preparar la documentación, así como la preparación de la exposición del proyecto.</p>
<p>Objetivos</p> <p>El objetivo principal de esta fase es hacer un buen trabajo de testeo para lograr tener una aplicación depurada, libre de bugs y con un buen diseño, compatible con la mayor cantidad de dispositivos posibles.</p> <p>Cuando se llegue al final de esta fase debemos contar con la memoria acabada y la exposición del proyecto preparada.</p>

Tabla 1: Planificación del proyecto

2. Descripción

En el punto introductorio del presente documento ya se han descrito a grandes rasgos la funcionalidad, contexto y enfoque del proyecto, sin embargo, en este apartado daremos una descripción un poco más amplia de la visión general y misión del sistema, y trataremos que la funcionalidad quede perfectamente descrita a través de la redacción de historias de usuario. Para una comprensión a un nivel más técnico, podemos acudir al punto 6 donde se describe la arquitectura de la aplicación o los diferentes diagramas explicativos, la mayoría concentrados en el punto 10, pero también repartidos a lo largo de todo el documento.

A grandes rasgos, **Taperfy es una plataforma de compra venta de comida casera**, esto es, usuarios cocineros guisando y vendiendo a través de la web sus preparaciones en dosis individuales generalmente conocidas como tupperes, y otros usuarios dispuestos a obtener este tipo de productos por la plataforma ya que no tienen tiempo (o conocimiento) para hacer su propia comida casera.

La motivación principal y la idea que subyace en el proyecto es doble: por un lado, hacer una reivindicación y defensa de la comida tradicional y local, ya que el auge de la comida preparada y las nuevas plataformas de distribución parecen estar dejando fuera a este tipo de productos. Por otro lado, la segunda razón de ser de la plataforma es ofrecer una herramienta para explotar un conocimiento valioso que tienen muchas personas: su buen hacer en la cocina.

Como se ha comentado anteriormente, básicamente **Taperfy** será una plataforma de compraventa de productos, por lo que sus funcionalidades, opciones e interfaz de usuario será similar a las decenas y decenas de webs que hay en el mercado para productos especializados (ropa, muebles, juguetes, etc). Aun siendo una plataforma mucho más generalista, una buena referencia para hacernos una idea de como será el proyecto es pensar en **Wallapop**: usuarios ofreciendo cosas, subiendo fotos, gente negociando la compra en pequeños chats previos a la reserva, filtrado de búsqueda, pantalla de detalle del producto, etc.

Nuestra aplicación deberá mostrar al usuario la comida preparada disponible en su localidad y este usuario, al interesarse por alguna preparación en concreto podrá acceder al detalle de ese taper, conocer la reputación al cocinero y tener un pequeño chat con él donde pueda resolver dudas, negociar.. Si llegan a un acuerdo, el usuario comprador le mandará una solicitud de reserva al cocinero (por una cantidad concreta de raciones) y esta solicitud deberá ser aceptada. Todas las partes del proceso recibirán notificaciones en la aplicación con cada cambio en el proceso de reserva. Una vez finalizada la transacción el comprador valorará el producto y esto repercutirá en la reputación del cocinero.

Adicionalmente, la aplicación implementará funcionalidades usuales en este tipo de productos: edición del perfil, búsquedas, filtrado de resultados, etc. Como valor añadido, se pretende crear un vínculo entre usuarios y cocineros con el concepto de ‘Cocinero favorito’, que ofrecerá al usuario un mejor acceso y comunicación directa con aquellos vendedores que haya catalogado como favorito.

En nuestro proyecto tendremos dos tipos de usuarios. Por un lado, los **cocineros**, que deben poder publicar sus guisos de una forma sencilla y tener en la interfaz un mecanismo para atender los mensajes y las solicitudes de los compradores.

Por el otro lado, **compradores** de comida. En este caso, la aplicación pondrá el foco en mostrar de forma atractiva la comida disponible, ofrecer filtros para localizar preparaciones por tipo, nombre, precio, etc. e implementar pantallas para la negociación previa a la compra y seguimiento de solicitudes. Sería lógico pensar en un tercer tipo de usuario: **administrador**, que tenga acceso a todos los datos, administrar tablas paramétricas, resolver disputas, etc. pero este perfil se ha quedado fuera del desarrollo del PFM por ser ya un proyecto con una carga de trabajo muy grande y ser el tipo de usuario más genérico y no directamente vinculado con el núcleo de la aplicación.

Para concretar lo anteriormente escrito, a continuación se presentan un conjunto de historias de usuario que describen como debe comportarse la aplicación y cuales son sus principales funcionalidades. En el apartado de Diagramas se pueden ver estas historias de usuario agrupadas en un User StoryMap.

Historias de usuario funcionales

Como **usuario**, quiero ver un listado de comida casera que ofrecen otros usuarios cerca de mi lugar de residencia.

Como **usuario**, quiero poder filtrar los platos ofertados para poder visualizar únicamente las opciones que se adecuen a mis gustos y necesidades.

Como **usuario**, quiero poder registrarme en la aplicación con un correo electrónico y contraseña de mi elección.

Como **usuario**, quiero poder recuperar mi contraseña en caso de haberla olvidado.

Como **vendedor** de tappers, quiero poder ofrecer la comida que he preparado, asociar fotos a mi producto, describirlo y establecer el precio que considere oportuno para sacar unos ingresos extras.

Como **vendedor** de tappers, quiero poder actualizar mi producto, retirarlo del mercado o, si llega el caso, marcarlo como vendido.

Como **vendedor** de tappers, quiero recibir valoraciones de lo que he vendido para tener feedback de mis clientes y ver cuales son mis puntos fuertes y en qué puedo mejorar.

Como **vendedor** de tappers, quiero poder responder mensajes de mis clientes para que ellos tengan perfectamente claro en qué consiste mi producto y antes de comprarlo.

Como **vendedor** de tappers, quiero poder recibir mensajes de aquellos usuarios que me han marcado como cocinero favorito para recibir sugerencias, que ellos me pidan comidas y, en general, tener un canal de comunicación abierto con mis mejores clientes.

Como **comprador** de tappers, quiero preguntar al vendedor de un producto cualquier cosa que se me ocurra respecto al producto ofertado, medio de entrega, precio, etc. para no tener ninguna duda antes de decidirme a comprar o no.

Como **comprador** de tappers, quiero poder reservar una comida ofertada y que esta no pueda ser vendida a otro usuario hasta que se complete la transacción.

Como **comprador** de tappers, junto al detalle de cada comida ofertada en el sistema quiero poder leer las valoraciones que otros usuarios tienen sobre el cocinero de dicha comida.

Como **comprador** de tappers, quiero poder calificar al vendedor de cada uno de los productos que compre en la plataforma para ayudar a otros usuarios en sus decisiones y tener un mecanismo que me permita expresar mi satisfacción como cliente.

Como **comprador** de tappers, quiero poder marcar a cocineros como favoritos para comunicarme directamente con ellos y tener un acceso más rápido a los productos que ellos ofrecen.

Como **comprador** de tappers, quiero poder escribir directamente a mis cocineros favoritos para solicitarles algo, preguntar dudas y tener un canal abierto de comunicación con ellos.

Historias de usuario no funcionales

Como **usuario** de la aplicación, quiero poder utilizar la web desde cualquiera de mis dispositivos: ordenador, tablet o móvil para poder encargar mis platos favoritos desde cualquier sitio y en cualquier momento.

Como **usuario** de la aplicación, quiero que sea sencilla de usar y pueda realizar las acciones de forma intuitiva y natural.

Como **usuario** de la aplicación, quiero poder ver los platos preparados en mi zona y filtrar esos platos sin tener que esperar demasiado tiempo para obtener la información.

Para implementar la funcionalidad anteriormente descrita desarrollaremos dos proyectos: un backend con la capa de acceso a datos y una api, y un front-end que mostrará todos los datos al usuario y modelará la interacción de este con el sistema. Este proyecto doble nos va a permitir poner en práctica los diferentes conocimientos adquiridos durante el desarrollo del máster, utilizando los lenguajes principales: typescript, html, css así como las herramientas y frameworks más destacados estudiados durante estos años: node, angular, ngrx, laravel, etc.

3. Objetivos

A continuación, se detallan los objetivos que pretende alcanzar el sistema una vez esté desarrollado y puesto en producción. Los dividiremos en principales, que describen la misión principal y la razón de ser de la web, y los secundarios, que son aquellas características deseables en nuestra plataforma pero que no determinan el objetivo principal de la aplicación ni pertenecen al núcleo de esta.

Desde un punto de vista muy general y sin comentar nada de la funcionalidad de la aplicación, el objetivo y filosofía del proyecto es doble: por un lado reivindicar el tipo de comida tradicional, luchando contra la 'colonización culinaria' a la que nos vemos sometido a veces por otras plataformas y poner en valor los múltiples beneficios que aportan este tipo de comidas en términos de salud.

Por otro lado, Taperfy nace con la ilusión de dar la posibilidad a muchas personas de obtener unos ingresos extras aprovechando un conocimiento y saber hacer pocas veces reconocido, explotado y monetizado: su buen hacer en la cocina. Junto a cocineros nóveles, gente en situación de desempleo y otros perfiles, la plataforma alcanzaría su mayor éxito si consiguiera ayudar a personas generalmente fuera del mercado laboral, que no generarían ingresos de otra forma.

3.1 Principales

Objetivos clave del proyecto:

- Que los usuarios registrados que así lo deseen, puedan poner a la venta su comida casera cocinada por ellos, con el precio que estimen oportuno.
- Ofrecer la posibilidad de los que accedan a la web de buscar comida preparada en su localidad, filtrar los resultados en base a sus preferencias y si así lo estiman oportuno, reservar raciones de algunas de las comidas ofertadas en la web.
- Proveer un sistema de valoración donde el usuario que ha comprado pueda puntuar su satisfacción con el producto obtenido y que esa información que se vaya recopilando sirva a otros usuarios para identificar los cocineros más reputados de su zona.
- Permitir a los usuarios registrados marcar algunos cocineros como favoritos para estar al día de las nuevas comidas que vayan sacando a la plataforma y tener un mejor acceso a sus novedades.

3.2 Secundarios

Junto a los objetivos clave de la aplicación anteriormente descritos, hay una serie de aspectos que el sistema persigue y que también son deseables para desarrollar un producto de calidad.

- Ofrecer una interfaz amigable, limpia y fácil de usar a todos los usuarios. Donde tanto aquellas personas que estén acostumbradas a utilizar aplicaciones web, como otras con menos experiencia, se sientan cómodas en la aplicación y la puedan utilizar sin problemas.
- Que la web se adapte a cualquier dispositivo donde se visualice y el resultado siga siendo igualmente funcional y usable. Debe ser igual de sencillo comprar un táper desde el móvil que desde un ordenador de sobremesa.

- Que la aplicación sea fluida y tenga un rendimiento aceptable. Ningún usuario debería esperar a que el sistema actualice los filtros, se suba una nueva comida o se envíe una solicitud de reserva.
- Ofrecer un sistema de notificaciones donde, tras registrarse, el usuario pueda acceder fácilmente a las novedades que le son relevantes en el contexto de la aplicación: un cocinero ha recibido una reserva, hay nuevos tappers de un cocinero favorito, le han aceptado la solicitud de reserva, etc.
- Posibilitar comunicación entre usuarios: tanto en un chat previo a la reserva de la comida donde el usuario comprador puede exponer sus dudas o acordar el sistema de entrega como la comunicación directa de los usuarios registrados con sus cocineros favoritos.

4. Escenario

Las aplicaciones de comida a domicilio han tenido un auge exponencial en los últimos tiempos. Quizás sea un síntoma relativo de lo comentado en la introducción: la falta de tiempo de las nuevas generaciones y también por la ruptura de la cadena de transmisión que hacía que una generación enseñara a cocinar a la siguiente. Falta de interés o falta de tiempo, el hecho es que cada vez se cocina menos en casa y se consume más comida cocinada por otros. A continuación, un par de artículos que respaldan el incremento de esta forma de consumo en los últimos años:

[Hostelería digital hablando del crecimiento del sector](#) [1]

[Sobre el auge de la comida a domicilio](#) [2]

Por tanto, parece que es un sector nuevo pero con un buen crecimiento y buenas expectativas de futuro, hecho que podría respaldar al idea de que en un sector que está creciendo a ese ritmo hay hueco para una web más si se sabe diferenciar y ofrecer un servicio original y de calidad.

Dentro de la comida a domicilio, creo identificar dos grandes grupos de aplicaciones / empresas: las que sirven para conectar con restaurantes en tu localidad y las que cocinan para ti.

En el primer grupo se encuentran las archiconocidas **just-eat**, **glovo** (aunque esta es una plataforma con un propósito más amplio), **ubereats**, etc. Estas aplicaciones han pasado en muy pocos meses de no existir a formar parte del día a día de millones de personas y su éxito es innegable. Funcionan perfectamente, son útiles y el éxito que tienen demuestra la calidad de producto que ofrecen. Sin embargo, como contrapunto, diría que están enfocadas a un tipo de comida muy particular: la comida rápida. En una espiral que se retroalimenta, la aplicación es el punto de entrada perfecto a pizzas, hamburguesas, etc y consiguen introducir en nuestro día a día estos alimentos, poniendo las cosas extremadamente sencillas para hacerse con ellos.

Just-eat tiene un informe muy completo e ilustrativo, llamado **Gastrómetro de JustEAT** [3], que muestra el estado actual del sector (actualizado en 2020) y revela algunos datos interesantes

Por ejemplo, el tipo de comida más solicitado según las provincias (a nivel nacional, era la pizza):



Figura 3: Tipos de comida más demandados por provincia en la plataforma JustEat

Italiana, oriental, turca, americana... a eso me refería en los puntos introductorios del documento cuando hablaba de la colonización gastronómica a la que nos vemos sometidos a veces y que las nuevas generaciones están desacostumbrando el paladar a los sabores de siempre. Entiendo que no es excluyente elegir una cosa u otra y que ambos tipos de comida se pueden compaginar perfectamente, pero digamos que un joven no lo tiene fácil ni puede conseguir comida tradicional de forma tan atractiva y este es el hueco que pretende llenar Taperfy.

Por otro lado, existe algunas webs de reparto de tappers a domicilio, algunos ejemplos:

<https://www.nococinomas.es/>

<https://www.menudiet.es/>

<https://www.miplato.es/>

No las he probado como usuario, pero parecen webs consolidadas, fiables y que comparten mucho de la esencia del proyecto que aquí se expone. Básicamente, todas podrían compartir el mismo lema: *cocinamos por ti*. Además, parecen más centradas en la comida saludable y tradicional. El hecho de que en una búsqueda rápida se puedan encontrar tantas alternativas remarca la expansión del sector y confirma los problemas anteriormente citados de falta de tiempo y conocimiento.

Respecto a estas últimas alternativas, aún estando mucho más cercanas a la filosofía del Taperfy todavía veo algunas diferencias importantes respecto a la idea de este proyecto. Todas las webs citadas anteriormente, junto a la logística y algún servicio adicional, actúan como cocineros. Ellos realizan la cocina y la compra por ti. Sin embargo, Taperfy es un sistema distribuido que pone en contacto a cocineros y compradores, lo que permite sin duda que la comida ofrecida sea más local, porque los cocineros registrados venden en su localidad.

Por ejemplificarlo, cuando este verano vaya al pueblo de mi pareja a pasar 15 días de vacaciones. Si busco en nuestra plataforma me podré poner en contacto con cocineros locales que me permitan conocer los platos de allí: potaje trujillano, perrunillas, gazpacho de huevo, etc.

Este factor local es diferenciador en Taperfy que se propone posibilitar a los cocineros registrados sacar ‘sus ases en la manga’ culinarios y compartirlos con sus vecinos. Al ser un sistema totalmente distribuido, la misma plataforma irá premiando a las recetas más aceptadas, cocineros mejor valorados, etc. No hay que olvidar que el sistema pretende crear un vínculo entre los clientes y los vendedores con el concepto de cocinero favorito.

Por otro lado, el segundo favor que echo en falta cuando reviso las alternativas es la distribución de las ganancias y a quien puede beneficiar una transacción en la plataforma. De alguna forma, cuando compramos por glovo o ubbereats estamos pidiendo a un restaurante ya existente, que ha creado un nuevo y excelente canal de distribución. Pidiendo a algunas de las empresas que preparan tupperes por ti, estamos premiando a una única empresa, que podrá ser socialmente responsable y eso es estupendo, pero el valor añadido de nuestra plataforma es que cuando estamos haciendo alguna transacción por Taperfy tenemos la certeza de que estamos dejando el dinero en alguien cercano en nuestra comunidad, a la que estamos ayudando a sacar unos ingresos extras y que probablemente nunca ha sido recompensada monetariamente por cocinar. Esta última dimensión de la aplicación, la ambición de empoderar a gente que sabe cocinar bien y quiere tener unos ingresos extras, es el verdadero factor diferenciador de la plataforma.

5. Metodología

A la hora de la elección de una metodología para implementar este proyecto, hay que tener en cuenta el contexto en el que se va a desarrollar. Aunque considero que podría tener una proyección comercial en el mundo real, esta implementación no deja de ser un proyecto académico enmarcado en un proyecto de fin de máster que será desarrollado de forma unipersonal en el espacio de unos meses.

Es una tendencia en el mundo del desarrollo el uso de metodologías ágiles como Scrum o Kanban que han demostrado adaptarse mejor a equipos de desarrollo para la construcción de productos software, consiguiendo una mayor implicación y mejor participación del cliente, mejor comunicación en el equipo y minimización de los riesgos. Sin embargo, como comentaba anteriormente, aquí no tenemos cliente como tal, ni tenemos equipo y el hecho de ser un proyecto unipersonal es determinante para decidir usar una metodología algo más clásica pero que creo que se adaptará mejor a nuestro desarrollo: el desarrollo en cascada.

La metodología waterfall o desarrollo en cascada divide la elaboración de un producto software en una serie de fases de tal forma que, idealmente, no se debe pasar a la siguiente fase hasta tener concluida la anterior. Podemos usar la variante ‘cascada con retroalimentación’ en la que la que podemos volver a alguna fase anterior en caso de encontrar algún error o necesitar replantear alguna parte del proyecto.

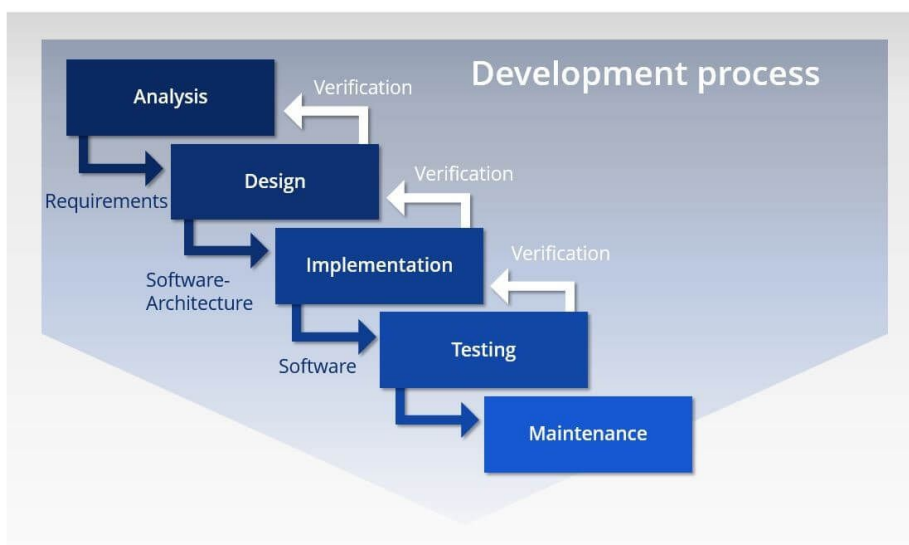


Figura 4: Metodología de desarrollo en cascada

Las diferentes fases que define la metodología de por las que de alguna forma u otra vamos a pasar son las siguientes:

Fase	Descripción
Análisis	En esta fase se definirán con detalle los requisitos funcionales y no funcionales de la aplicación y se podrá planificar con precisión el esfuerzo necesario del proyecto. En nuestro caso, este trabajo corresponde a la primera PEC aunque como he mencionado anteriormente, si durante el proceso de desarrollo el proyecto crece o muta, se estudiaría la posibilidad de incorporar estas mejoras al proyecto siempre y cuando se puedan garantizar la calidad y corrección de lo implementado.
Diseño	Detallaremos estos requisitos hasta que tengamos claro el modelo de datos de la aplicación, que entidades intervienen con que atributos y crearemos casos de uso, así como alguna forma de prototipado de pantallas que nos permita tener una idea clara de que es lo que debemos implementar.
Implementación	Es generalmente la parte más extensa. En ella escribiremos el código de la aplicación y haremos pruebas unitarias para garantizar la corrección. Dividiremos esta implementación en diferentes módulos que nos permita subdividir esta fase y versiones incrementales totalmente funcionales de nuestro sistema.
Testing	Haremos pruebas integrales de toda la aplicación desarrollada. En caso de encontrar errores, incompatibilidades de dispositivos, problemas de rendimiento... volveremos al desarrollo para subsanar cualquier incidencia. Aún así, es buena idea que el testing esté presente durante todo el desarrollo del proyecto y durante el máster se han estudiado algunas técnicas para probar el código implementado automáticamente.
Mantenimiento	Una vez el sistema está en producción, una última fase en los desarrollos tradicionales es gestionar el mantenimiento de la misma: con bugs detectados por los usuarios, evolutivos, etc. Esta última fase no tiene del todo sentido en este proyecto académico a no ser que termine convirtiéndose en un producto real.

Tabla 2: Fases de la metodología en cascada

Nos tomaremos estas definiciones de fases y orden de estas como una directiva o guía para orientar nuestro desarrollo, pero no como un esquema rígido de obligado cumplimiento. Por ejemplo, mi intención es que el testing esté presente durante todo el desarrollo del proyecto (aunque por último dedique unos días exclusivamente a pruebas). Tampoco pienso que sea una buena idea tener absolutamente cerrado los requisitos funcionales porque si durante el desarrollo del proyecto surge una idea que pudiera aportar al producto y nuestra planificación ‘se la pueda permitir’ considero que sería una buena idea incluirla.

6. Arquitectura de la aplicación/sistema/servicio

Nuestro sistema consta de dos partes bien diferenciadas: por un lado una aplicación backend encargada de centralizar todos los datos del sistema en una base de datos y ofrecer una api de servicios REST y en el otro extremo un frontend, implementado en angular, que se ejecuta en el navegador del usuario, muestra la interfaz de la aplicación y se comunica con el back consumiendo los servicios REST anteriormente mencionados.

Por tanto, estamos ante una arquitectura cliente servidor ‘clásica’, donde contamos con un servidor central con todos los datos de tappers, usuarios, interacciones, etc. y una serie de clientes consumiendo y actualizando estos datos y mostrándolos en pantalla de la mejor forma posible para el usuario.

Esta comunicación entre front-end y back-end se realizará mediante peticiones HTTP, haciendo uso de servicios REST, un protocolo de comunicación ampliamente extendido y utilizado en el desarrollo web moderno que utiliza los ‘verbos’ de HTTP (GET, POST, PUT, etc.) como forma de comunicación entre ambos extremos. Estas peticiones normalmente devuelven objetos codificados en formato json y para que cada cliente se identifique se ha utilizado el estándar de autenticación JWT, consistente en la inclusión de un token en cada petición REST realizada por el cliente.

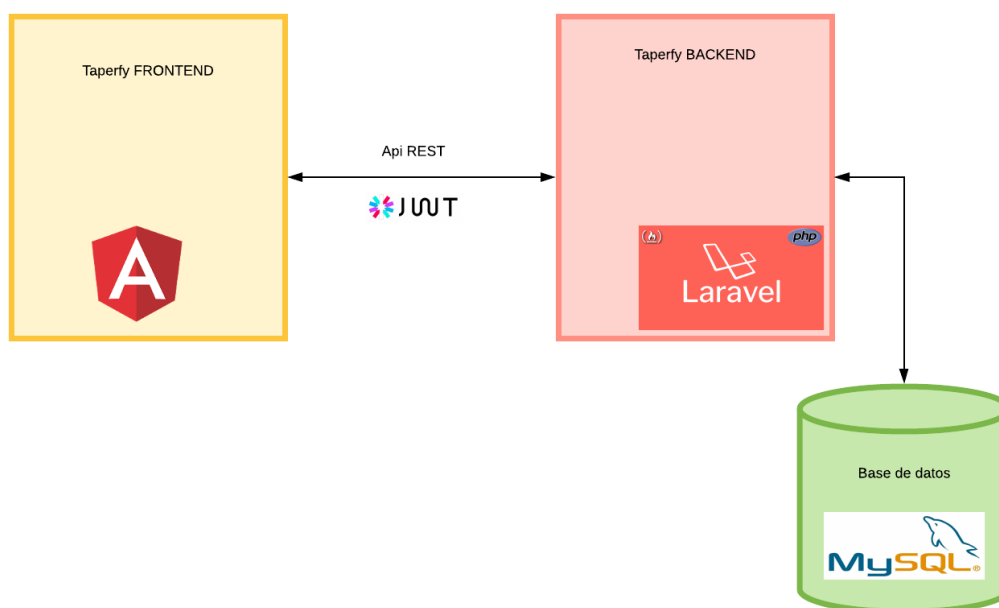


Figura 5: Arquitectura de la aplicación

Backend

El backend se ha implementado haciendo uso del framework de desarrollo **Laravel**. Esta herramienta, en la que se profundizó en una de las asignaturas del máster, nos permite hacer un

desarrollo rápido en proyectos php, ofreciéndonos a los programadores una serie de funcionalidades que suelen requerirse en cualquier desarrollo web: integración rápida con la base de datos a través de un ORM, clases de soporte para implementar controladores, parseo de datos etc.

Este framework sigue el patrón de arquitectura MVC, que persigue independizar los datos de la representación de los mismos. De esta forma, tendremos un modelo (la capa de acceso a datos en sí), una serie de vistas (en el caso de una aplicación que ofrece una api REST, la representación de los datos es en formato json) y un ‘puente’ que nos ayuda a conectar el modelo y las vistas: los controladores. Por tanto, buena parte de la implementación del backend ha sido tanto definir el modelo (y la capa de acceso a datos) como implementar una serie de controllers que nos permite atender a las diferentes peticiones del cliente (métodos que ofrece la api REST). Afortunadamente, Laravel proporciona una serie de herramientas que facilitan las labores comunes en este tipo de desarrollo:

- Un ORM para implementar la capa de acceso a datos
- Mecanismo para devolver objetos json a partir de objetos que pertenecen al modelo.
- Mecanismos para crear controladores, validación de datos, etc.
- Una serie de utilidades y comandos para trabajar con la base de datos: creación y actualización del modelo de datos, factorías de objetos, creación de datos de prueba, etc.

En mi día a día soy desarrollador backend principalmente en tecnologías java y, aún llevando unas semanas utilizando laravel con regularidad sigue sorprendiéndome la facilidad y potencia de esteframeworksphp. No hay comparación posible en cuanto a velocidad respecto al mundo de aplicaciones web Java donde suelo trabajar, y la mayoría de los cambios no requiere reiniciar el servidor de aplicaciones, algo que agiliza mucho el trabajo

Front-end

El front-end de Taperfy está escrito en **Angular**, uno de los frameworks de desarrollo front modernos que mas auge ha tenido en los últimos años y en que hemos visto en mayor profundidad durante el transcurso del master. Angular es un framework de desarrollo web creado por google que persigue facilitar el desarrollo web moderno y construir aplicaciones del tipo SPA (single page application), es decir, una aplicación que se ejecuta en el navegador que no va de una página a otra, si no donde todo normalmente transcurre en la misma página y solo se producen recargas parciales y automáticas de los datos necesarios. Este tipo de páginas suele ofrecer una experiencia más fluida para los usuarios y las aplicaciones desarrolladas en angular se suelen caracterizar por su gran rendimiento y velocidad.

La arquitectura de las aplicaciones en Angular, al igual que nos ocurría en el backend, también suele ser MVC. En este caso, se podría definir la arquitectura de Angular como MVC basado en componentes:

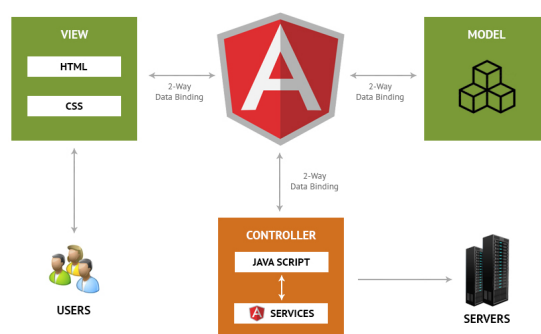


Figura 6: Elementos de una aplicación angular

Al igual que ocurría con Laravel, aquí también tenemos un modelo (en este caso es prácticamente una transposición en TypeScript de lo que teníamos definido en el backend), una serie de plantillas o vistas y unos elementos que se encargan de conectar ambas cosas: los controladores y servicios, posibilitando el data-binding o enlazo de los datos del modelo y la representación de los mismos. En medio de todo y orquestando estos elementos están las librerías de Angular que nos proporcionan mecanismos para la construcción de componentes, manejar su ciclo de vida, servicios de negocios, modelado de datos, inyección de dependencias, etc.

Angular es un campo amplio, complejo, rico en matices y entiendo que una explicación profunda del framework excede del ámbito de esta memoria, pero conocer los principales conceptos que componen la herramienta nos puede dar una visión general de como opera y cual es la filosofía de desarrollo que hay detrás de esta librería:

- **Componentes** → es la pieza básica de construcción de una aplicación angular. Cada componente es una ‘parte’ de nuestra SPA (pensar por ejemplo en la barra de navegación superior, el menú lateral, una tarjeta informativa...). Cada componente puede tener a su vez uno o varios subcomponentes y constan de una serie de datos, plantillas HTML, funcionalidad escrita en typescript y un ciclo de vida que será manejado por el framework y que nos permite codificar su comportamiento escribiendo typescript y haciendo uso de servicios de negocio, otros componentes y utilidades comunes que Angular inyecta por nosotros.
- **Módulos** → nos permiten empaquetar un conjunto de componentes y servicios de negocio dentro del mismo contexto. Podemos utilizar los módulos angular para agrupar los diferentes elementos que estén relacionados entre sí a nivel de funcionalidad. En nuestra aplicación se han definido varios módulos para dividir funcionalmente la parte cliente y poder aplicar la carga perezosa de módulos, lo que nos permitirá optimizar los tiempos de carga en nuestro sistema [4].
- **Servicios de negocio** → tendrán la lógica de negocio de la aplicación, generalmente las invocaciones a los backends, llamadas directas a la BBDD, lógica propia de negocio, etc. En este sentido, no me costo demasiado adquirir este concepto porque en el mundo backend java del que procedo siempre se ha perseguido mantener la lógica de negocio en una capa diferente a la más superficial de representación de los datos. Las dependencias entre los servicios de negocios son manejadas por Angular automáticamente haciendo Inyección de Dependencias.
- **Directivas** → Las directivas son un mecanismo que nos proporciona angular para darle algún tipo de comportamiento o funcionalidad extra a las vistas que escribimos, por ejemplo incluir un elemento o no en base a una condición lógica evaluada en el

componente, repetir una serie de veces un bloque de código o hacer una vinculación de datos entre la vista y una variable de nuestro modelo.

Lo anteriormente expuesto está explicado con mis palabras, a alto nivel y no persigue ser una definición académica de la arquitectura de Angular. Afortunadamente, la documentación de la librería es excelente y tiene una gran comunidad de desarrollo, con lo que es fácil encontrar soporte para cualquier problema o duda en los sitios habituales: stackoverflow, github, etc.

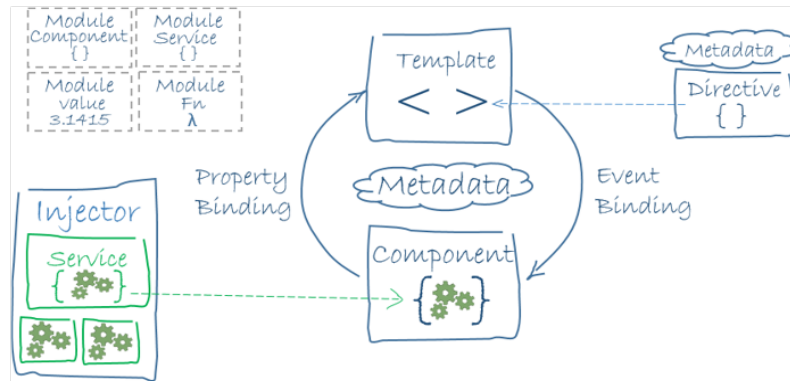


Figura 7: Arquitectura framework angular

Adicionalmente a la arquitectura resultante de las aplicaciones angular: MVC, con componentes, agrupados en módulos, con servicios inyectados.. En la implementación de este proyecto se ha hecho un especial esfuerzo por implementar el patrón de diseño **REDUX**, un patrón de arquitectura que pretende dotar de ‘estado’ a la aplicación, contar con un punto único en el código para los cambios de estado y tener definidas de una forma precisa las acciones que puedan provocar un cambio de estado en la aplicación:

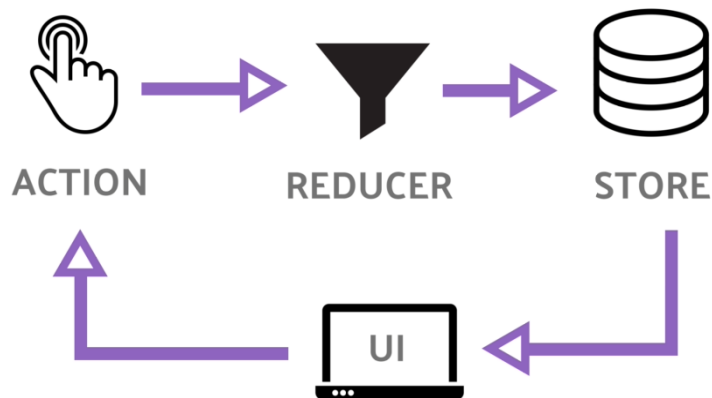


Figura 8: Patrón de diseño REDUX

Esto en Angular toma forma con **NGRX** (<https://ngrx.io/>), una librería que nos permite implementar dicho patrón de diseño en aplicaciones Angular. La curva de aprendizaje fue especialmente dura en los primeros momentos pero afortunadamente ya se hizo una PEC en la asignatura de Desarrollo con Frameworks donde se introducía el tema y los beneficios de la aplicación del patrón han sido grandes: mayor trazabilidad, control del estado de la aplicación, mejor debugging y un mayor control del flujo de datos.

7. Plataforma de desarrollo

A continuación se muestra las herramientas tanto software como hardware que se han utilizado durante el análisis, implementación y pruebas del proyecto:

Tipo	Recursos utilizados
Servidor web	Apache
Base de datos	MySql
Interfaz de BBDD	PhpMyAdmin
Lenguaje backend	PHP
Backend	Laravel
Lenguajes frontend	Typescript, HTML, SCSS
Frontend	Angular, ngrx, node, npm
Control de versiones	Git, GitHub
Equipo de desarrollo	MacBook PRO (Apple Silicon M1 2010, 16GB Ram, 1TB HDD)
Sistema operativo	macOS Catalina
Diagramas	miro, lucidchart, google drive
Editor	Visual Studio Code
Pruebas API	PostMan
Video	Loom

Tabla 3: Recursos tecnológicos usados durante la creación del proyecto

8. Planificación

En el [apartado 1.3](#) del presente documento se hizo una planificación detallada del trabajo de análisis y desarrollo del proyecto, qué entregas se definen y que se espera de cada una de las fases en las que se dividió el PFM. Para tener un resumen con lo más relevante y acceder de una forma más visual a la planificación del trabajo, se muestra a continuación un diagrama donde en el eje vertical están los principales hitos del proyecto y horizontalmente se encuentran cada una de las 4 PECs en las que repartiremos el trabajo:

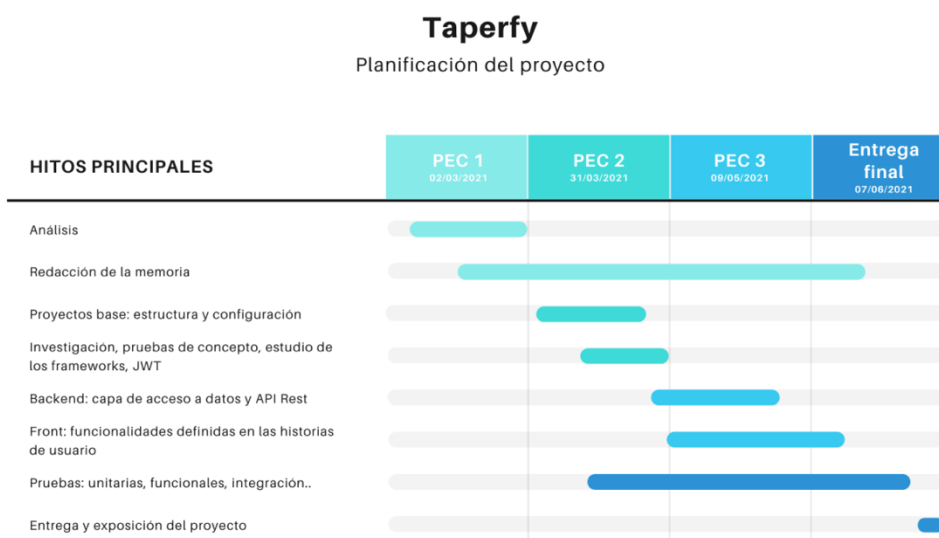


Figura 9: Planificación del proyecto

Como se puede apreciar, la planificación del proyecto se ajusta mucho a la metodología de desarrollo en cascada que propusimos en la parte introductoria de la memoria. Aún así, hay dos tareas transversales que estarán presentes en casi todas las fases del proyecto: la redacción de la memoria y la realización de pruebas de distinto tipo (unitarias, funcionales, de aceptación...) al código implementado.

9. Frameworks, APIs y herramientas utilizadas

En este apartado se expone una relación de todas las tecnologías en las que nos hemos apoyado para el análisis e implementación del proyecto. Dentro de lo posible, he tratado usar aquellos componentes y técnicas que he cursado durante el Máster, aunque en algunas áreas específicas he tenido que profundizar un poco más o incluir alguna librería o componente externo tras una investigación personal.

Componentes	Tipo	Descripción
Angular	Framework de desarrollo	Framework de desarrollo front-end
Laravel	Framework de desarrollo	Framework de desarrollo backend
rxjs	Librería	Librería para trabajar con observables en angular
ngrx	Librería	Librería para implementar el patrón de arquitectura REDUX
tymon/jwt-auth	Librería	Librería php para gestionar tokens jwt en las peticiones REST
tailwind.css	Framework css	Framework css que utiliza la filosofía utility-first para proporcionar clases que faciliten en maquetado, responsive, etc. de la aplicación
font awesome	Iconos	Iconos visualizados en la aplicación, distribuidos en forma de fuente
ng-select	Componente angular	Implementar un selector con databinding con el componente y multitud de opciones de configuración y eventos disponibles
pselect	Librería javascript	Convierte dos input de tipo select en dos selectores con las provincias y municipio españoles, encargándose la librería de la recarga del municipio en base a la provincia elegida.
ngx-slider	Componente angular	Componente angular para implementar la selección del precio máximo en modo barra deslizadora
spatie/laravel-http-logger	Librería php	Middleware para laravel que permite visualizar en el log la petición http completa de las peticiones de la api que seleccionemos trazar

Tabla 4: Listado de Frameworks, apis y herramientas utilizadas

10. Diagramas

Para ahondar en la comprensión tanto a nivel técnico como funcional del proyecto, en esta sección se expondrán dos diagramas utilizados usualmente en el desarrollo de software: el diagrama entidad relación y un *userstorymap*. Con el primero de ellos, tendremos una visión general del modelo de datos de la aplicación, que entidades intervienen y cual es la relación entre estas. Como se ha comentado en otros puntos dentro de este documento, el objetivo es hacer una versión casi definitiva del modelo de entidades en los primeros compases del desarrollo, ya que esto clarifica mucho la visión global y el alcance de la aplicación, pero por otro lado no queremos cerrar esta definición 100% ya que en principio estamos abiertos a, si nos da tiempo, ampliar nuevas funcionalidades o afinar elementos del modelo de datos para un mejor funcionamiento o rendimiento.

A continuación, mostramos un *userstorymap*, que es una herramienta de gestión visual que nos permite visualizar los requisitos en forma de historias de usuario y agrupar estos elementos en épicas, que son definiciones de alto nivel de partes de la aplicación. Este sería un buen punto de partida si aplicáramos alguna metodología ágil e iterativa ya que dividiríamos estas historias de usuario en diferentes releases y en cada iteración iríamos liberando un producto terminado con una funcionalidad cada vez más amplia. En nuestro caso, prácticamente solo tenemos una iteración de desarrollo (la correspondiente a la PEC 3) por lo que no tiene demasiado sentido definir releases. Aún así, el *userstorymap* nos da una visión global de la funcionalidad acordada para el proyecto, como se agrupan y cuales son las prioridades.

10.1 Base de datos, diagrama entidad relación

El diagrama ERD de Taperfy es el siguiente:

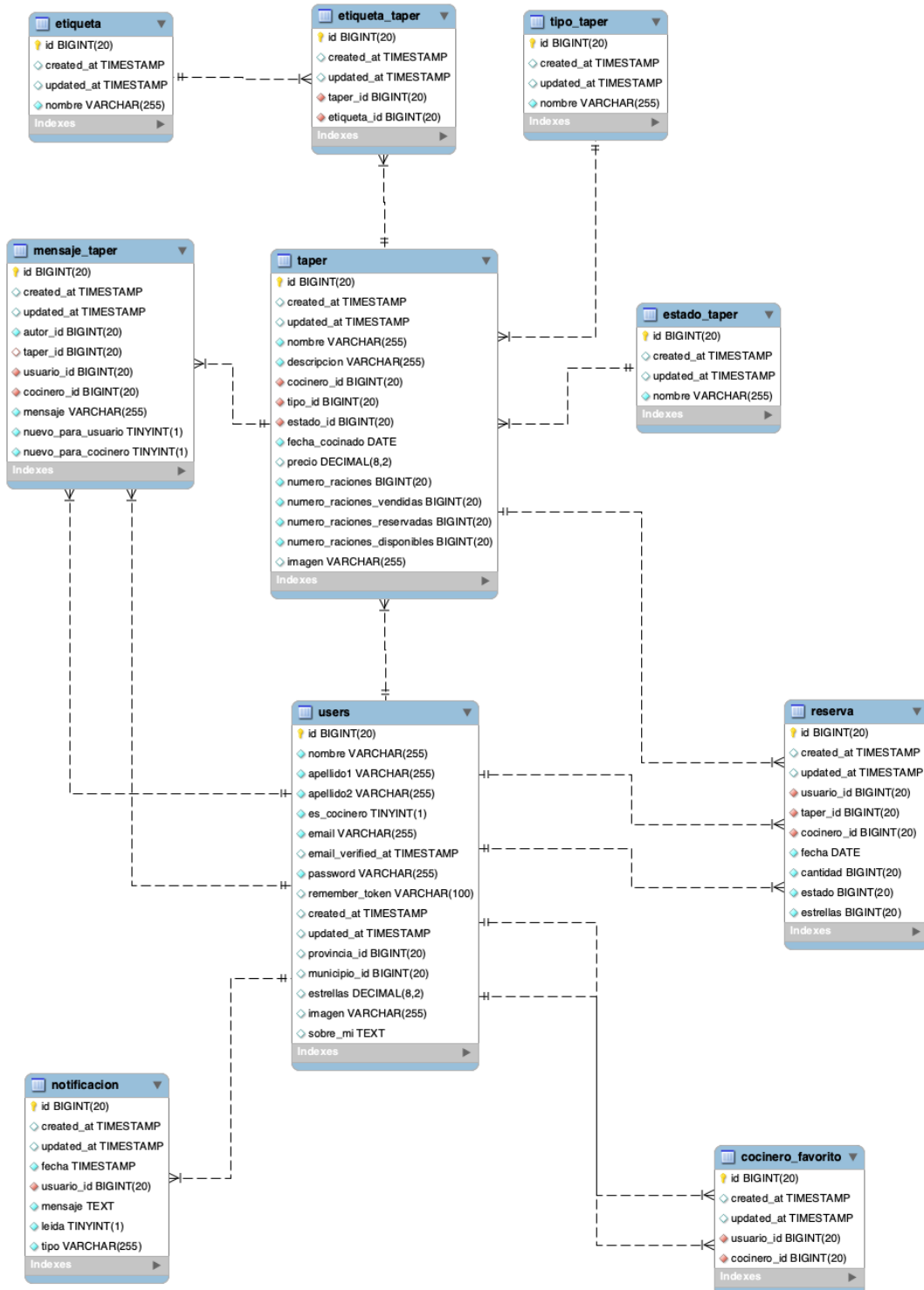


Figura 10: Modelo entidad relación de la aplicación

10.1 User StoryMap

Podemos ver agrupadas las historias de usuario presentadas en el [punto 2](#) en el siguiente User StoryMap:

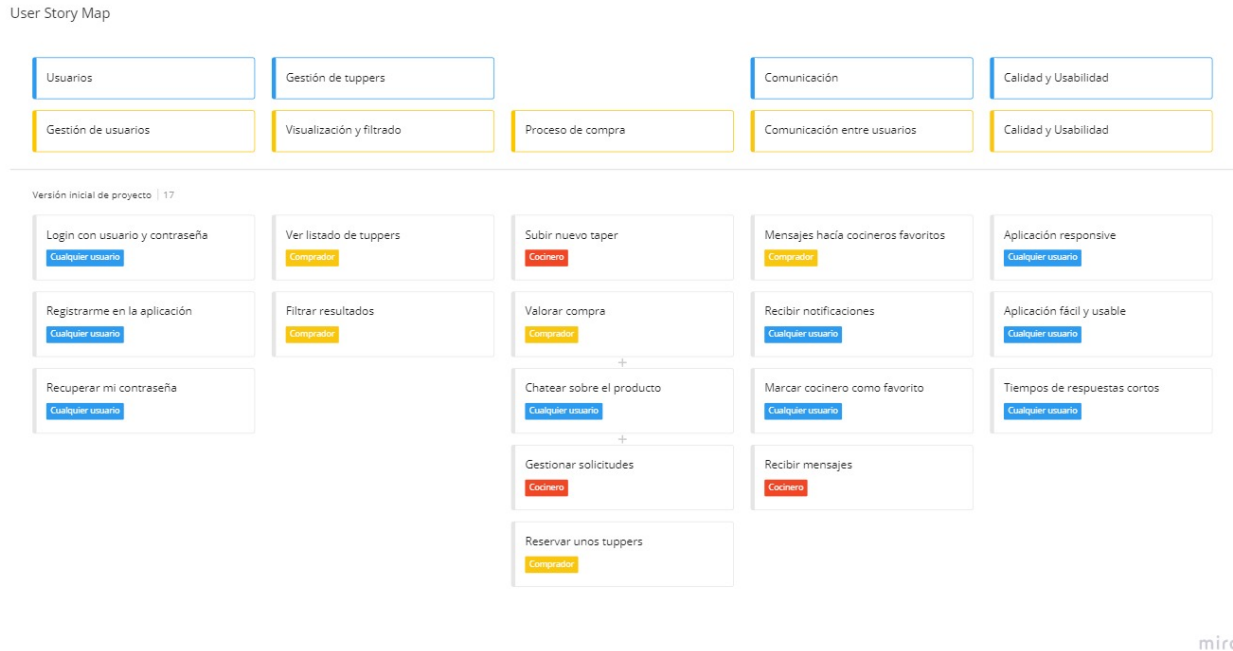


Figura 11: User StoryMap con las historias de usuario de Taperfy

11. Prototipos

En este apartado se detallan una serie de bocetos para tratar de determinar como será la interfaz de usuario, que elementos deben tener y cuales serán las pantallas más representativas de la aplicación. Se dividirán en prototipos Lo-Fi, cuyo principal objetivo es mostrar la idea de la pantalla y cual es su cometido básico y los Hi-Fi, con el que nos podemos hacer una idea más exacta de que incluye la pantalla y cómo será su representación visual. Estos últimos están desarrollados ya en HTML, con una apariencia similar al producto final, aunque no deben tomarse como una descripción exacta de cada pantalla, más bien como una herramienta que nos ayude a pensar cuales son las partes principales de esa funcionalidad, cual es el contenido más relevante que aspecto tendrá a grandes rasgos.

11.1 Lo-Fi

Estos bocetos los hice a mano alzado en un primer estadio del proyecto, cuando aún le estaba dando forma a la idea. Dibujar las pantallas me ayudó a concretar muchos aspectos, definir como podrían ser los flujos y navegación de usuario y a validar la idea con muchos amigos y familiares, ya que enseñándoles los wireframes se hacían una idea mucho más precisa de cómo podría ser la aplicación y podían realizar mejores aportes. A continuación, expongo las pantallas más representativas:

Pantalla de login:

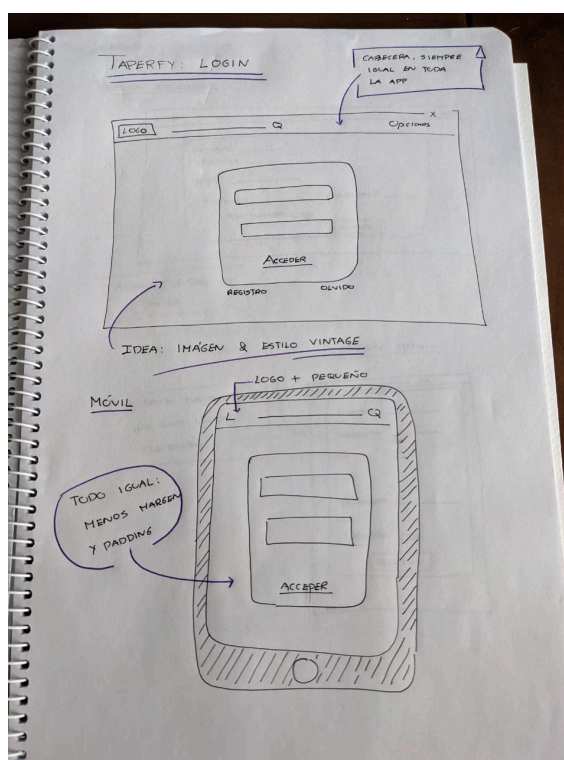


Figura 12: Boceto de la pantalla de login

Pantalla de registro de usuarios:

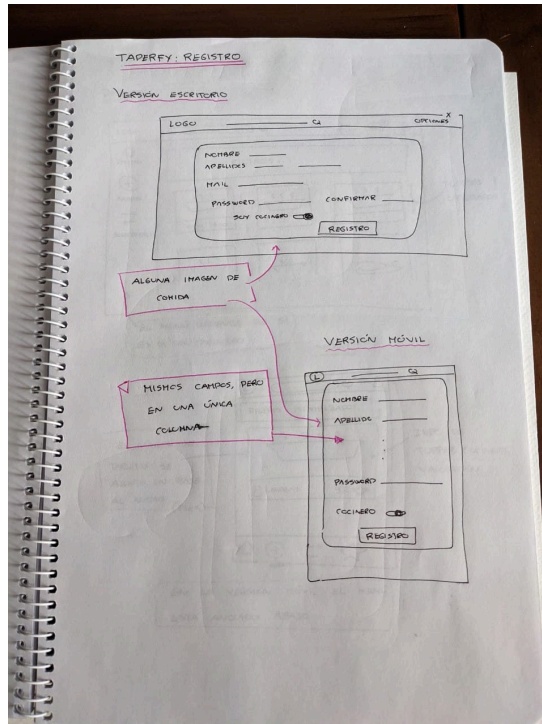


Figura 13: Boceto de la pantalla de registro de usuarios

Pantalla principal de la aplicación

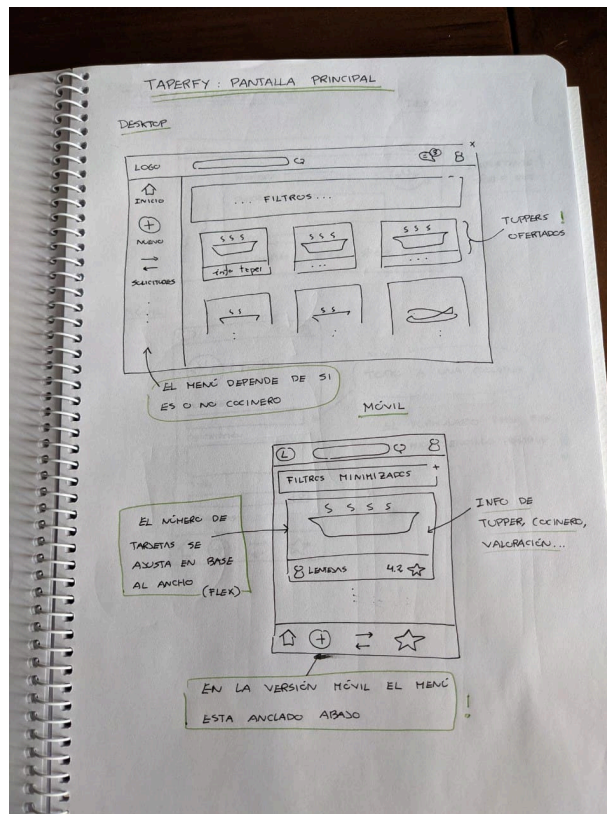


Figura 14: Boceto de la pantalla de inicio en escritorio y móvil

Pantalla de nuevo taper

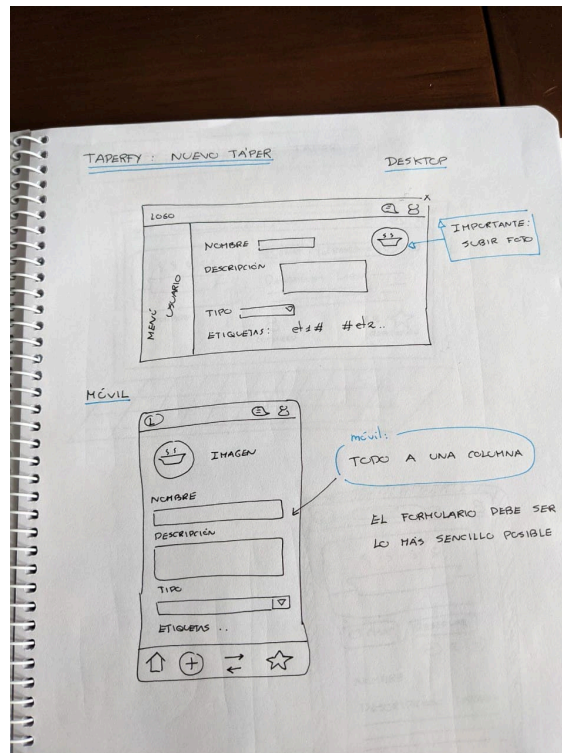


Figura 15: Boceto de la pantalla de inicio en escritorio y móvil

Pantalla de detalle de taper

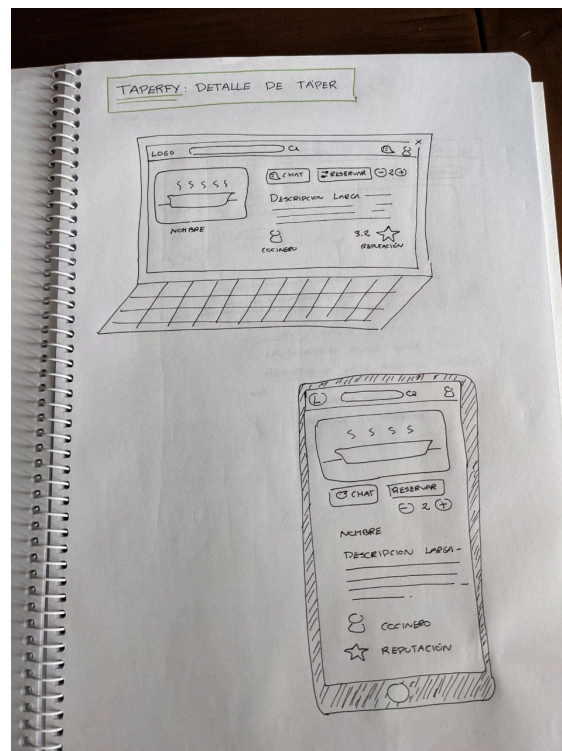


Figura 16: Boceto de la pantalla que muestra el detalle de un taper.

Pantalla de chats entre usuarios

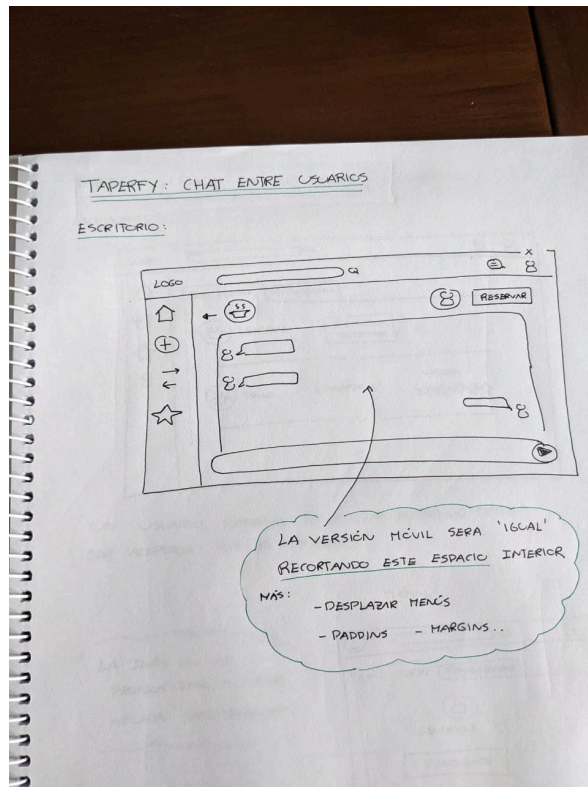


Figura 17: Boceto de chats entre usuarios

Pantalla de peticiones (comprador)

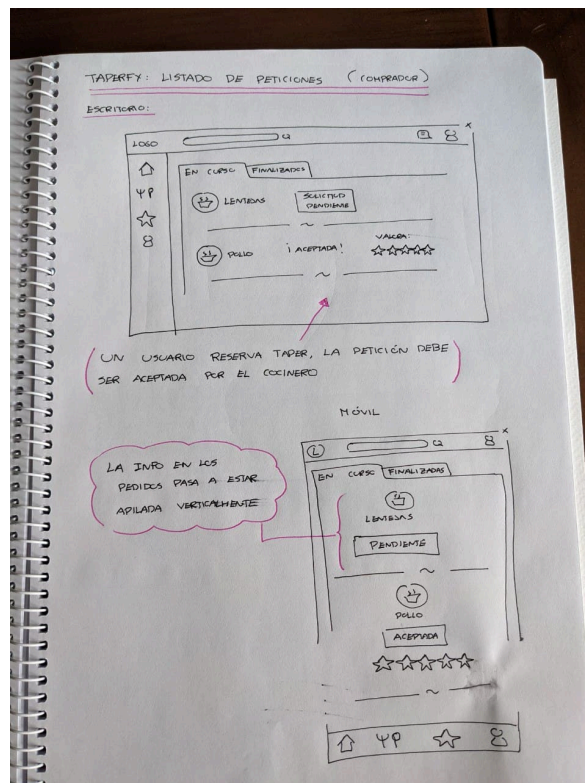


Figura 18: Boceto de la pantalla de peticiones realizadas por el comprador

Pantalla de listado de solicitudes (cocinero)

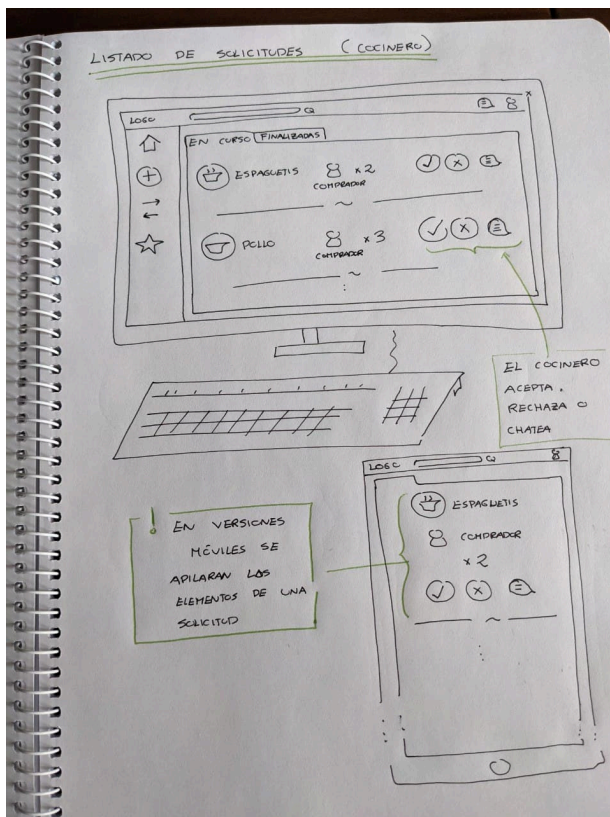


Figura 19: Boceto de la pantalla de solicitudes al cocinero

11.2 Hi-Fi

En esta sección incluiremos las mismas pantallas de los bocetos anteriormente expuestos, pero con un aspecto muy parecido a como presentaremos el producto final. Para ello, hemos maquetado con html y css básico nuestras pantallas y hemos incluido la mayoría de componentes angular aunque sea sin darles aún comportamiento. Junto a describir visualmente las características principales de la aplicación, trabajar de esta forma nos ha permitido ir tomando algunas decisiones de diseño, como por ejemplo la decisión de incluir un menú lateral en versiones de escritorio e inferior en las versiones móviles, o el aspecto general de la aplicación, imágenes que enlazamos y paleta de colores. Para nuestra aplicación, hemos escogido tonalidades pastel, con colores muy poco saturados para perseguir un aspecto vintage. Las pantallas aquí expuestas nos dan una idea aproximada de como va a ser la aplicación, aunque algún campo puede variar en la versión final del sistema, ya que como comentaba en los puntos introductorios estaremos abiertos hasta el final a incorporar nuevas ideas, funcionalidades y que el proyecto vaya evolucionando durante todo el ciclo de desarrollo.

Pantalla de login:

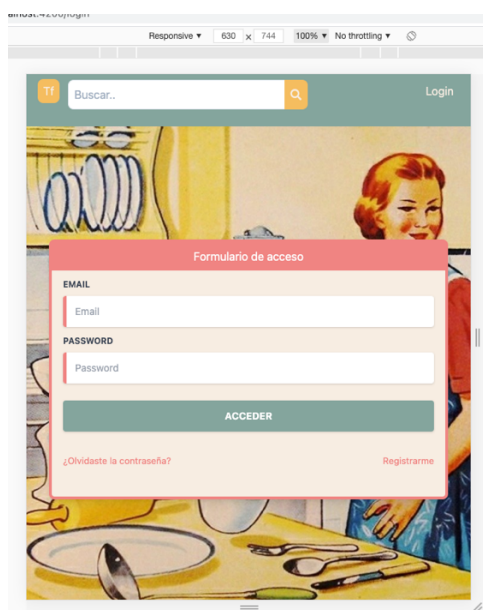
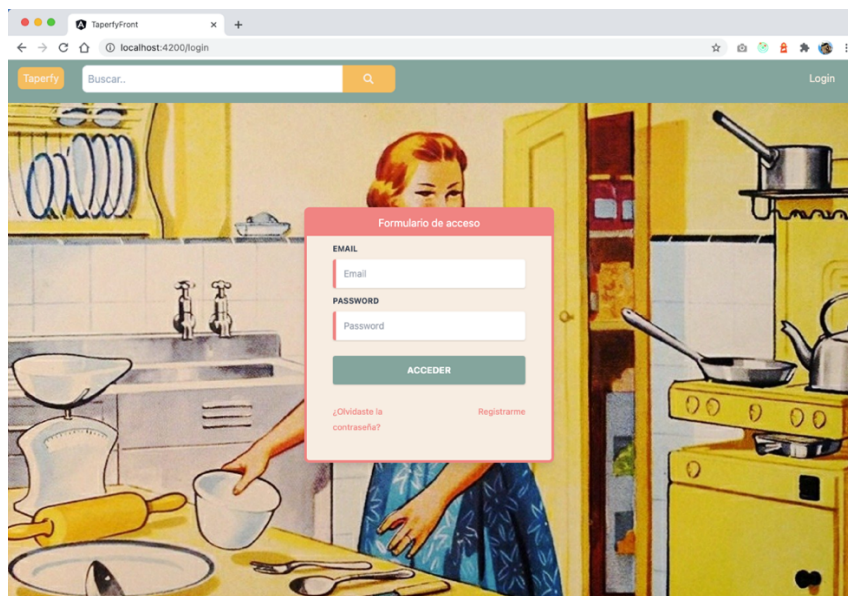


Figura 20: Prototipo Hi-Fi de la pantalla de login (desktop y móvil)

Pantalla de registro de usuarios:

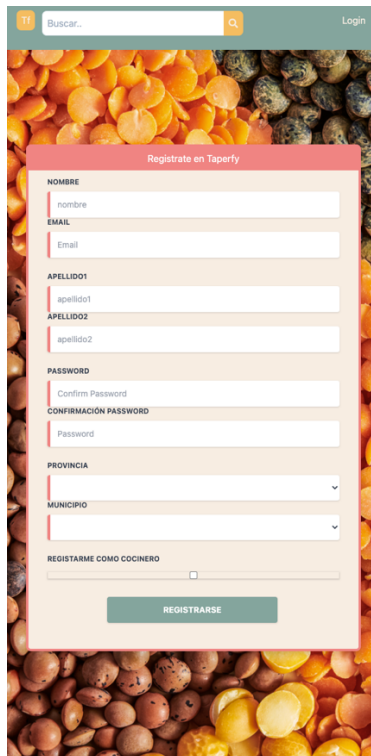
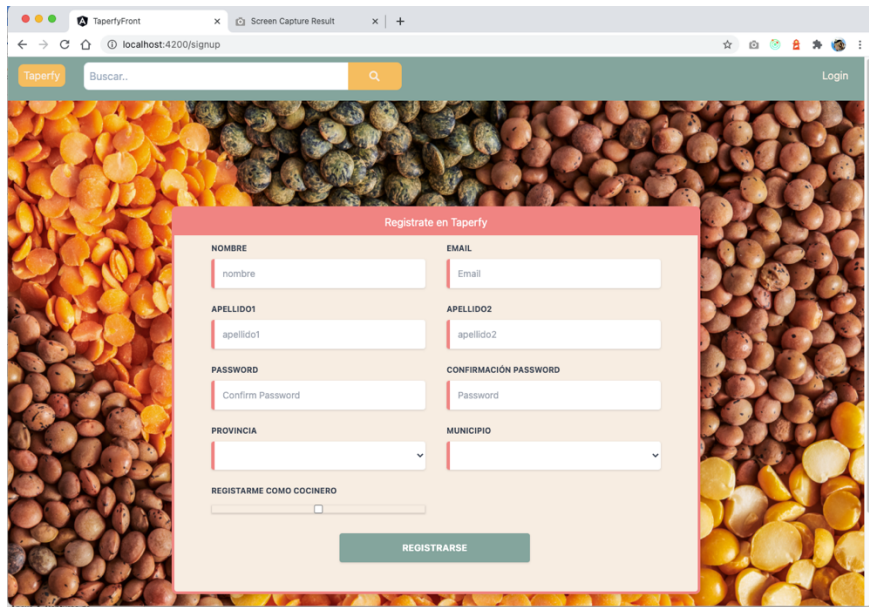


Figura 21: Prototipo Hi-Fi de la pantalla de registro (desktop y móvil)

Pantalla principal de la aplicación

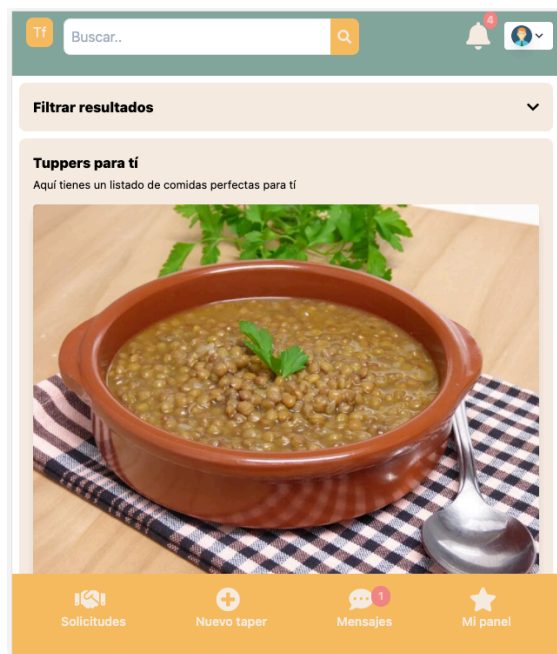
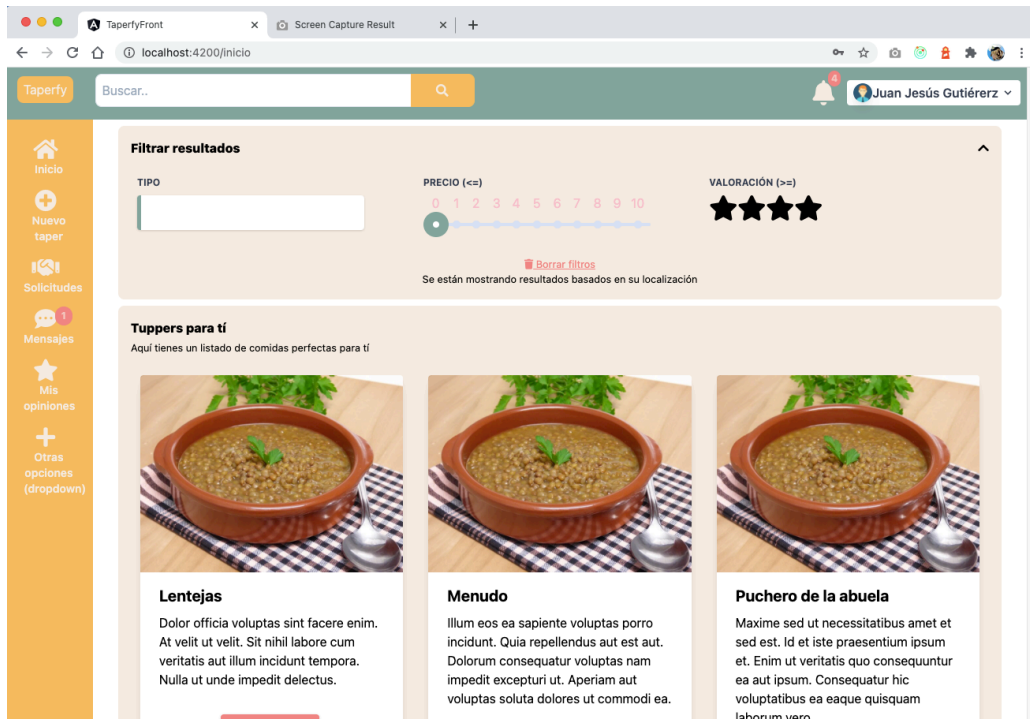


Figura 22: Prototipo Hi-Fi de la pantalla principal(desktop y móvil)

Pantalla de nuevo taper

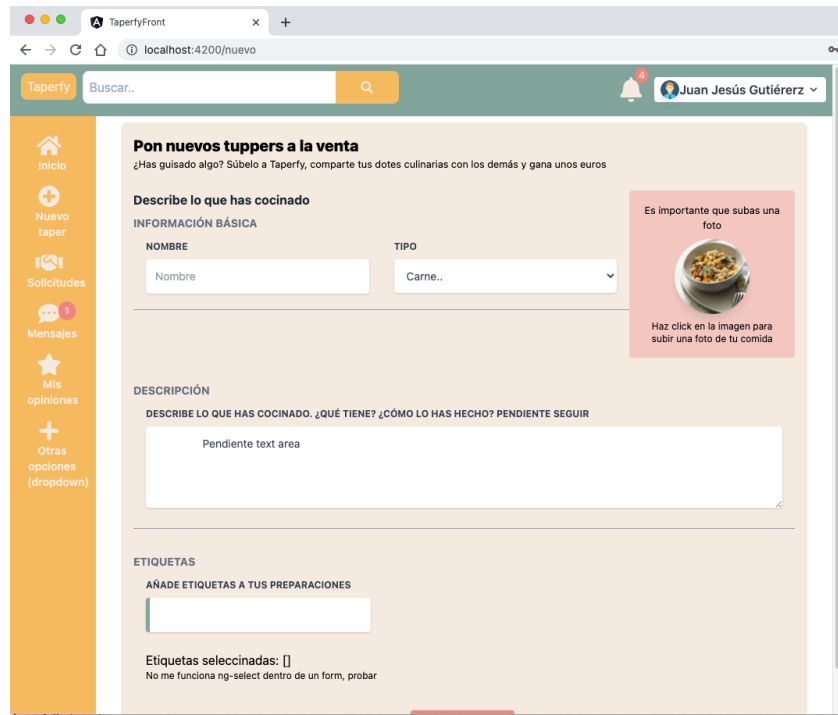


Figura 23: Prototipo Hi-Fi de la pantalla de nuevo taper(desktop y móvil)

Pantalla de detalle de taper

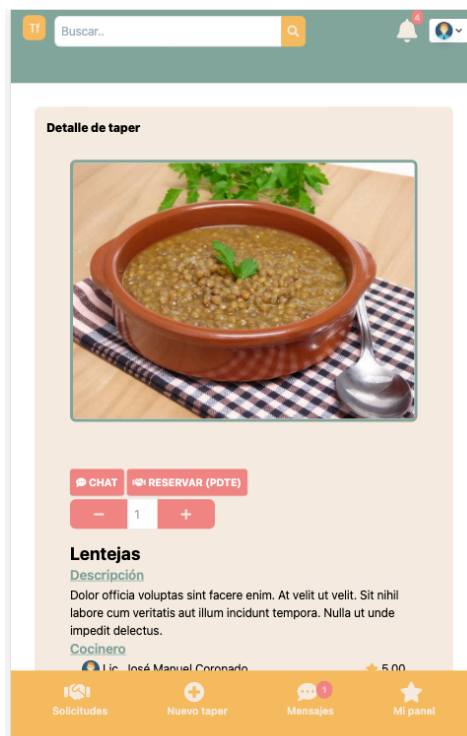
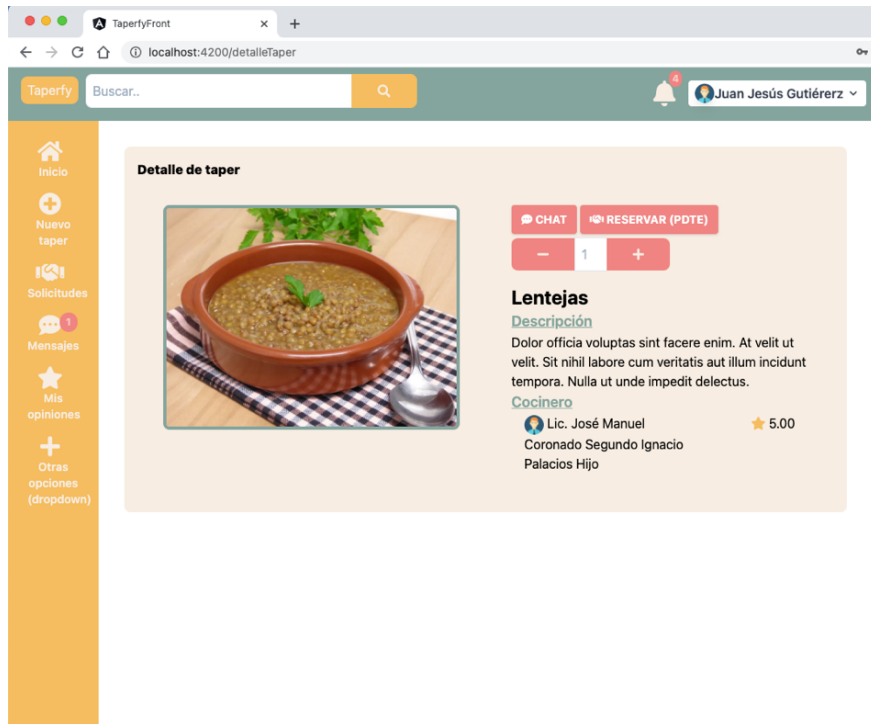


Figura 24: Prototipo Hi-Fi de la pantalla de detalle de taper(desktop y móvil)

Pantalla de chats entre usuarios

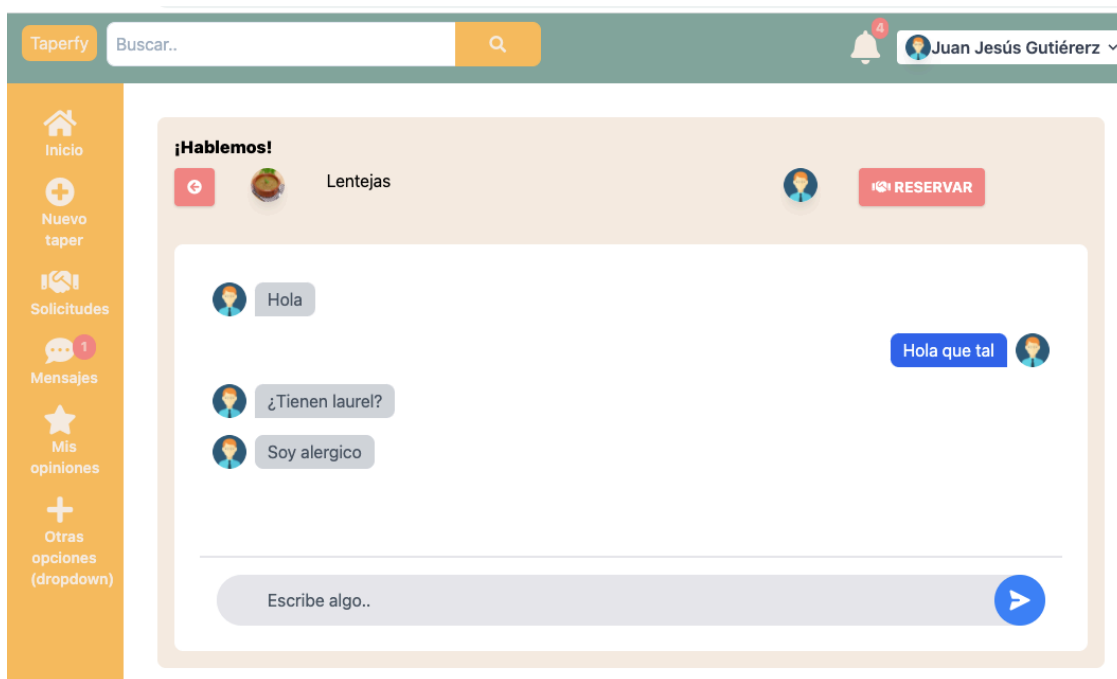


Figura 25: Prototipo Hi-Fi de la pantalla de chat entre usuarios

Pantalla de peticiones (comprador)

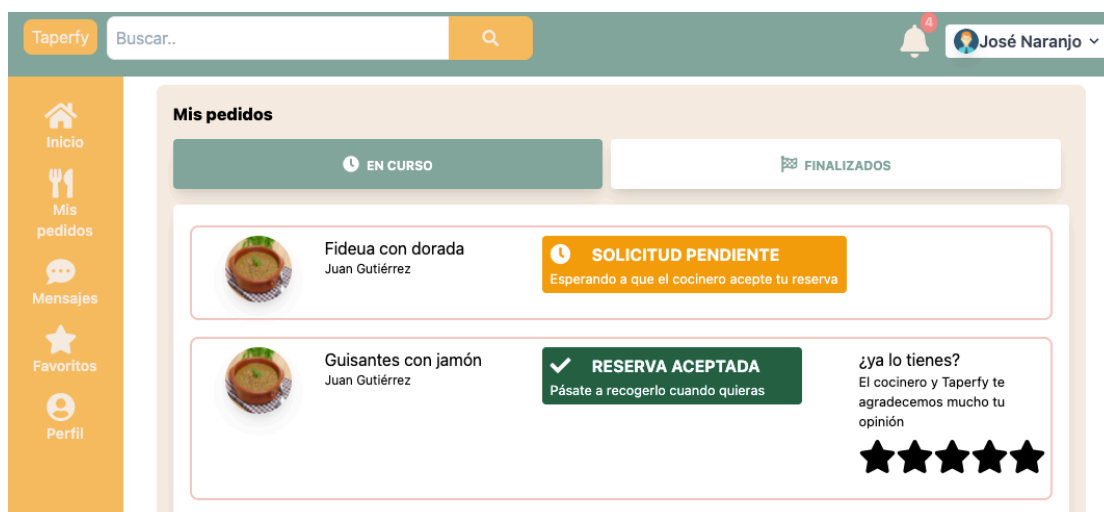


Figura 26: Prototipo Hi-Fi de la pantalla de peticiones de reserva del comprador

Pantalla de listado de solicitudes (cocinero)



Figura 27: Prototipo Hi-Fi de la pantalla de solicitudes recibidas por el cocinero (desktop y móvil)

12. Perfiles de usuario

La aplicación tiene dos perfiles de usuario bien diferenciados: el cocinero y el comprador de tupper. En la pantalla de registro de usuario existe un control de tipo toggle button donde el usuario puede determinar si se quiere registrar para vender o para comprar comida. Una vez registrado, el usuario podrá cambiar su rol en la aplicación desde la página donde se modifican los datos del perfil de usuario.



Figura 28: El usuario puede pasar de cocinero a comprador y viceversa desde la edición de su perfil

Estos perfiles de usuario representan roles totalmente diferentes en nuestra aplicación y nos han servido para escribir las distintas historias de usuario en las que hemos basado en análisis y la planificación del trabajo de implementación del proyecto.

Las funciones de los diferentes perfiles son las siguientes:

- Perfil cocinero
 - Publica nueva comida en la plataforma: categoriza la comida, le asocia etiquetas, le da una descripción y una opcionalmente una foto asociada. También debe indicar cuantas raciones están a la venta y a que precio se vende cada ración.
 - Revisar las solicitudes de reserva para aprobar o rechazar estas solicitudes a los usuarios.
 - Chatear con los usuarios compradores para solucionar cualquier duda, negocia...
 - Consultar sus estadísticas, tanto información relativa a los tupper que tiene publicados en cada momento como estadísticas globales: dinero generado por la aplicación, número total de ventas, etc.
 - Chatear con aquellos usuarios que le han seleccionado como cocinero favorito. Este canal de comunicación puede servir para hacer avisos, recibir solicitudes personalizadas, encargos, resolver dudas tras la venta, etc.
- Perfil comprador
 - Busca tupper en la plataforma. Una vez se autentique en la aplicación, el cocinero comprador solo verá tupper de aquellos cocineros que pertenezcan a su localidad, pero puede hacer un filtrado de los resultados que obtiene por defecto haciendo uso del formulario que aparece en la parte superior de la pantalla.

- Acceder al detalle de un taper. En principio, en la pantalla principal el usuario comprador verá previsualizaciones de los productos en forma de tarjetas, un patrón de diseño muy habitual en los e-commerce. En estas tarjetas solo aparecerá la parte más relevante del producto, si quiere acceder a toda la información el usuario podrá pinchar en la tarjeta y acceder a una página específica de detalle de tupper, donde el cocinero tendrá que decidir si el producto le convence (e inicia una reserva), si necesita información adicional (en ese caso, podría chatear con el cocinero) o si quiere buscar otro producto.
- Solicitar una reserva. Una vez que el comprador decide que quiere uno de los tupper ofertados, deberá hacer una solicitud de reserva al cocinero, indicando el número de raciones que desea reservar. Esta reserva podrá ser aceptada o rechazada por parte del cocinero.
- Cuando una compra se formaliza en la aplicación, el usuario comprador tiene la posibilidad de valorar el producto, con el habitual uso de estrellas presente en multitud de sistemas reputacionales. Esta valoración nos sirve para calcular la puntuación media del cocinero, un campo por el que se puede filtrar productos en la aplicación.
- Al igual que para el perfil cocinero, una de las acciones que pueden realizar los usuarios con perfil comprador es chatear, en este caso, con cocineros que ofrecen tupper o cocineros que han marcado como favoritos.
- Por último, la última acción asociada al perfil comprador es asociar (o eliminar la asociación) a cocineros como favoritos. Esto le permitirá un mejor acceso a estos cocineros y sus productos ofertados.

A nivel de implementación, no ha sido necesario incluir una tabla específica de perfiles en el modelo de datos de la aplicación puesto que únicamente tendremos dos tipos de usuarios (cocinero o no) y nos ha bastado añadir un campo booleano en la tabla usuario (es_cocinero) con el que determinaremos si el usuario puede publicar tupper o no.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	bigint(20)		UNSIGNED	No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
2	nombre	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
3	apellido1	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
4	apellido2	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
5	es_cocinero	tinyint(1)			No	Ninguna			Cambiar Eliminar Más
6	email	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
7	email_verified_at	timestamp			Sí	NULL			Cambiar Eliminar Más
8	password	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar Eliminar Más
9	remember_token	varchar(100)	utf8mb4_unicode_ci		Sí	NULL			Cambiar Eliminar Más
10	created_at	timestamp			Sí	NULL			Cambiar Eliminar Más
11	updated_at	timestamp			Sí	NULL			Cambiar Eliminar Más
12	provincia_id	bigint(20)		UNSIGNED	Sí	NULL			Cambiar Eliminar Más
13	municipio_id	bigint(20)		UNSIGNED	Sí	NULL			Cambiar Eliminar Más
14	estrellas	decimal(8,2)			Sí	4.00			Cambiar Eliminar Más
15	imagen	varchar(255)	utf8mb4_unicode_ci		Sí	NULL			Cambiar Eliminar Más
16	sobre_mi	text	utf8mb4_unicode_ci		Sí	NULL			Cambiar Eliminar Más

Figura 29: Campo de BBDD que indica si el usuario es cocinero o comprador

En el front, el campo específico que determina si el usuario es cocinero está mapeado en el modelo de angular en una variable de la clase *User* llamada *es_cocinero*, también de tipo booleano. Esta información, y todos los campos del usuario que devuelve la api Rest, es almacenada en el estado

de la aplicación (recordemos que Taperfy está implementado haciendo uso del patrón Redux) y en múltiples componentes tenemos un observable apuntando a esa parte del estado porque muchas acciones e interfaz de usuario son dependientes del tipo de usuario logado.

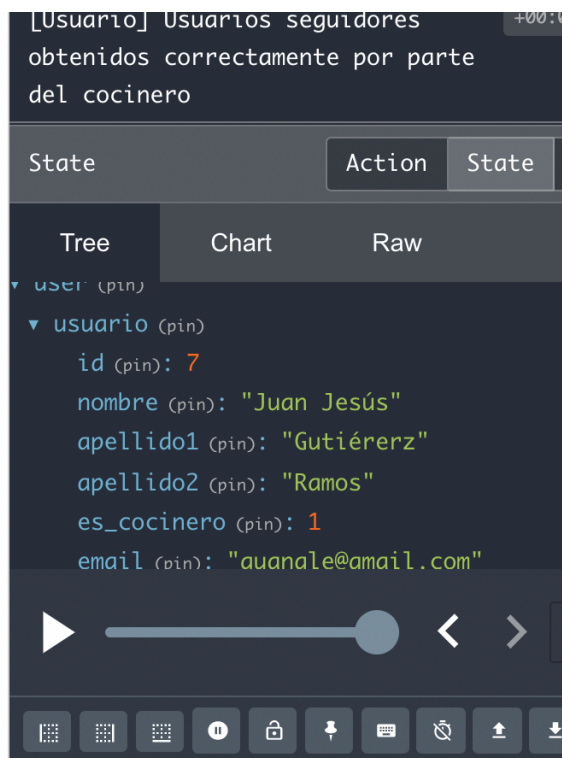


Figura 30: Parte del estado que nos indica si es cocinero, en el NgRx Dev Tools .

Si Taperfy fuera una aplicación real y no un proyecto académico, necesitaría al menos incluir un nuevo perfil de usuario: **Administrador**. Este perfil debería tener acceso y las diferentes tablas paramétricas de la aplicación (tipos de tupperes, etiquetas) y poder hacer CRUD de estos datos. Sería deseable que este rol también pudiera retirar contenido inapropiado publicado en la aplicación, resolver disputas, tener una pequeña gestión de usuarios para incidencias y demás.

Es muy probable que en una empresa real se planteara además que este que este perfil tuviera acceso a estadísticas globales de la aplicación para hacer una explotación estadística del negocio: cuantos usuarios registrados, tupperes publicados, etc. La implementación de un perfil administrador y sobre todo de todas las funcionalidades anteriormente mencionadas no se ha podido abordar porque excedería con mucho de las 300 horas que se ha dedicado al proyecto aproximadamente, pero es una de las primeras mejoras que propondremos en el punto de **Proyección a futuro**.

13. Usabilidad/UX

Durante construcción de Taperfy, se han seguido una serie de principios de usabilidad que suelen ser comunes en el diseño e implementación de las aplicaciones web. Estos principios son independientes de la tecnología concreta con la que se ha implementado la aplicación y hacen referencia a soluciones y patrones de diseño relativos a la interacción que se han demostrado eficaces para tener experiencias de usuarios satisfactorias.

Como comentaba en la parte introductoria de este documento, donde describía el proyecto y quien se podría beneficiar de él, he construido la web con la idea en mente de que fuera una plataforma perfecta para dar una oportunidad de negocio a personas de mediana / avanzada edad, que considero que son las que atesoran mayor valor en términos de sabiduría culinaria y buen hacer en la cocina. Pensando en estos usuarios, he tenido especialmente presente la necesidad de hacer un diseño sencillo de usar, no pedir más datos de los necesarios, no saturar al usuario con información y, en general, hacer una interfaz limpia e intuitiva.

A nivel de usabilidad y experiencia de usuario, no hay demasiadas innovaciones, de hecho, diría que la interfaz es muy parecida en términos de componentes y disposición de los elementos a otras plataformas de compraventa de productos como puedan ser Vinted o Wallapop, por lo que el usuario que conozca y esté familiarizado con este tipo de plataforma encontrará la aplicación familiar.

Se han utilizado patrones de diseño a nivel de usabilidad como la presentación de los productos en tarjetas, que muestran la información más relevante y son ‘clickables’, para acceder a la página de detalle. También se han usado otros elementos comunes en las aplicaciones web como las listas, pestañas, tablas y se ha hecho especial hincapié en mostrar en todo momento una interfaz sencilla, no sobrecargada de opciones para el usuario y tener jerarquizado los diferentes elementos para que estos estén en una posición y tamaño acorde a su importancia.

El diseño es totalmente responsive y desde que se elaboraron los bocetos a la implementación de la aplicación, se ha seguido una filosofía mobile-first, esto es, pensar primero en la versión móvil de la pantalla y luego ir adaptando a pantallas cada vez mas grandes. Para conseguir este objetivo, se ha utilizado un framework css como es TailWind, que pone las cosas realmente sencillas para el diseño responsive, brindando al desarrollador la posibilidad de aplicar clases únicamente a determinados anchos de pantalla. Durante el proceso de desarrollo se ha utilizado constantemente la herramienta de google Chrome que permite visualizar la página como si estuviéramos en un dispositivo móvil y periódicamente se ha publicado la aplicación en un servidor apache en mi equipo para poder probar en móvil y tablets.

Durante el mismo semestre en el que he estado desarrollando el proyecto de fin de máster, he cursado la asignatura de *Diseño de Interfaces Interactivas*, por lo que tengo especialmente presente este punto y he podido aplicar directamente algunas técnicas y principios conforme los he ido aprendiendo.

Los principios de usabilidad que he aplicado y he tenido en mente en el diseño y desarrollo han sido los siguientes:

- **Visibilidad de estado del sistema.** El usuario tiene acceso a buena parte de la información almacenada en el estado del sistema, como el nombre del usuario arriba a la derecha, si estamos esperando una respuesta del servidor, o las burbujas sobre los iconos de notificaciones o mensajes indicando si tenemos elementos pendientes de leer.

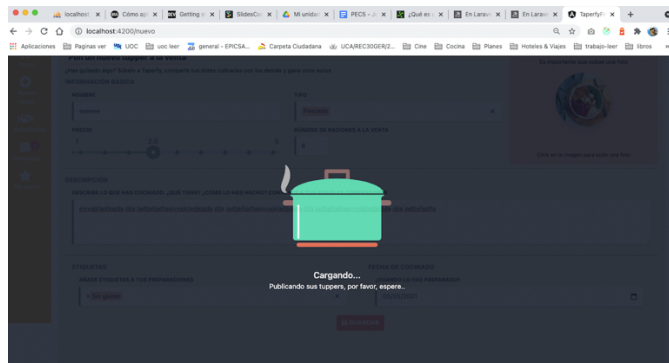
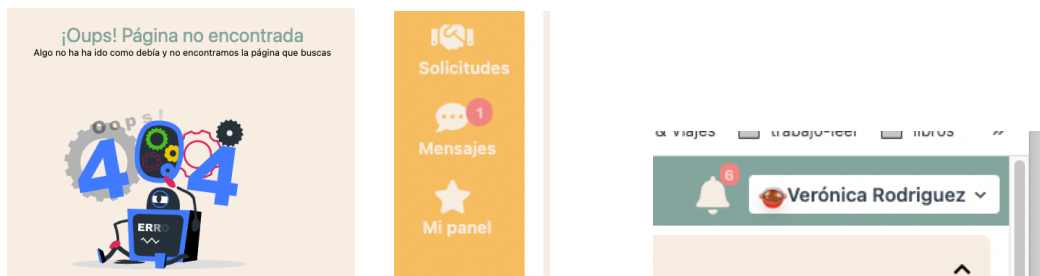
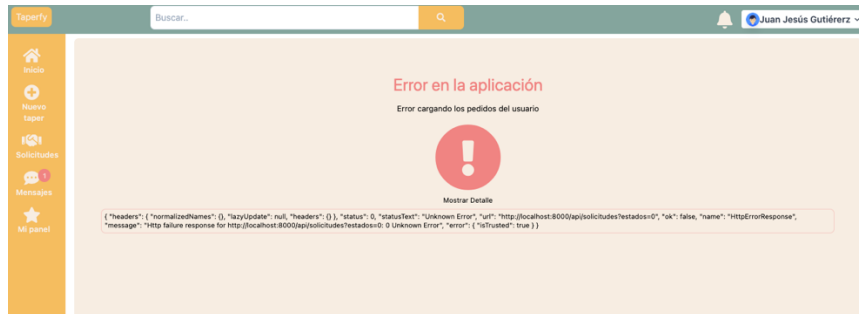


Figura 31: Varias capturas que muestran información del estado de la aplicación

- Se ha utilizado un **lenguaje coloquial**, evitando tecnicismos y estableciendo relación con objetos y conceptos del mundo real: taper, reserva, raciones, etc. En la medida de lo posible, siempre se han seleccionado iconos de la librería Font-awesome para reforzar visualmente cada una de las acciones que puede efectuar el usuario y estos iconos referencian conceptos que recalcan al usuario la idea de lo que va a pulsar: apretón de mano, estrellas, bocardillos, campanas, etc.
- **Consistencia y normas.** Se aplica un estilo consistente en todas las paginas que componen la web, tanto a nivel de distribución de elementos como el tamaño y espaciado de estos. Un

icono que aparece en dos sitios diferentes de la aplicación referencia la misma cosa y todos los títulos y subtítulos tienen la misma apariencia y tamaño.

- Se ha pretendido, en la medida de lo posible, aplicar un **diseño y estética minimalista**, sin caer en ningún exceso (algo que puede resultar contraproducente). Se ha perseguido no tener una interfaz excesivamente sobrecargada y no mostrar ni solicitar más información de la estrictamente necesaria.
- **Estándares:** como se ha comentado en el párrafo introductorio, para garantizar una buena experiencia de usuario y una usabilidad óptima se han seguido estándares y patrones de diseño de usabilidad ampliamente extendidos en el mundo web y en particular en los e-commerce: productos en tarjetas, detalle de productos, información del usuario logado arriba a la izquierda, etc. El uso de este tipo de patrones facilita enormemente la usabilidad ya que un usuario ve en la aplicación elementos y patrones de interacción que le son familiares.

En la asignatura de Diseño de Interfaces Interactivas que mencionaba anteriormente se resalta la utilidad e importancia de la redacción de *User Personas*, esto es, arquetipos que representan usuarios que pueden utilizar la aplicación. La redacción de estas *User Personas* hace tener más presentes a los usuarios cuando se esté prototipando y maquetando la aplicación. Dos usuarios arquetípicos de nuestro sistema podrían ser los siguientes:

- *Carmen es una mujer de mediana edad, no está habituada al uso de nuevas tecnologías y no es especialmente diestra en este campo, pero si tiene una habilidad especial: atesora multitud de recetas tradicionales y sabe guisar estupendamente. Sus hijos se han ido este año a la universidad y tiene más tiempo que nunca. No ha tenido continuidad en el mundo laboral, pero si le vendrían bien unos ingresos extras que podría conseguir cocinando para vecinos de su localidad, que vivan cerca de ella.*
- *Luis acaba de empezar la carrera en Sevilla, una ciudad nueva para él. Ha alquilado una habitación cerca de su facultad y es la primera vez que vive solo. Tras empezar el curso, se da cuenta de que, entre clases, exámenes, trabajos... no tiene tiempo ni energías para aprender a cocinar, ni hacer la compra. Aún así, está preocupado por comer sano y echa mucho de menos las comidas que le preparaba su madre en su pueblo natal, al que casi no puede volver.*

El objetivo principal de la redacción de estos *User Personas* es mostrar al equipo de desarrollo y diseño el tipo de usuarios que utilizará la aplicación, y cuales son sus motivaciones y principales 'pain points' que podría solucionar la plataforma.

Adicionalmente, otra técnica de usabilidad que recomiendan en la asignatura es redactar uno o un par de **trigger points**, escenarios de entrada que ejemplifican como acceden a nuestra aplicación estos usuarios. En nuestro caso, puede ayudarnos a comprender las futuras interacciones entre usuarios y sistema este par de ejemplo:

- **Trigger Point 1:** *Carmen, que está acostumbrada a cocinar para sus tres hijos pero que ahora vive sola, acaba de guisar lentejas, no ha calculado bien las cantidades y tiene 6 raciones de sobra. Además, estamos a final de mes y le vendría bien un refuerzo a su*

pensión para afrontar los gastos de la semana, por lo que coge su móvil, entra en Taperfy y vende a sus vecinos cada una de las raciones excedentes a 3 euros.

- **Trigger Point 2:** *Luis tiene 2 exámenes y que entregar un trabajo esta semana, no tiene tiempo para comprar y cocinar y ya está un poco cansado de comer todos los días de bocadillo y comida rápida, por lo que accede desde su portátil a Taperfy para encargar comida sana, tradicional y rica para toda la semana y centrarse en sus estudios.*

14. Seguridad

La seguridad en la aplicación web ha sido tenida en cuenta desde el primer momento del proceso de análisis y diseño ya que es una aplicación con usuarios registrados, distintos perfiles y se guardan múltiples datos de estos (reservas, notificaciones, mensajes, etc.). Se han desarrollado algunos mecanismos de seguridad tanto en la comunicación entre el cliente y el servidor como en la ejecución de rutas protegidas en la aplicación Angular.

La comunicación entre el front y el backend de la aplicación se hace mediante una API Rest. En la mayoría de los casos, cada una de las peticiones a los métodos de la API deberán estar protegidos y autenticados, es decir, un usuario anónimo no tiene permisos para invocar a determinadas funciones de la API. Esto lo hemos conseguido con **JWT (Json Web Token)**, que es un estándar abierto, basado en json, para la creación de tokens de autenticación entre cliente y servidor.

El cliente, cuando hace un login satisfactorio, obtiene un token JWT válido (generado por el servidor tras comprobar sus credenciales) y utiliza este token en todas las llamadas posteriores a la API. Este token generado por el servidor tiene toda la información necesaria relativa a la autenticación: usuario, validez, etc. y es almacenado en la aplicación cliente, concretamente en el estado de la aplicación Angular.

Una vez obtenido el token por parte del cliente, nuestro servidor Laravel comprueba, haciendo uso de un middleware, que las peticiones cuentan con un token JWT antes de ejecutar los métodos de la api securizados. Para la implementación del mecanismo JWT en el backend se ha utilizado la librería **jwt auth** [5] que hemos instalado haciendo uso de composer.

Para la instalación de la librería se han seguido los pasos descritos en su web [6] . Tras la instalación del componente, la generación de un *secret key* y la configuración del middleware en Laravel para que se utilice la autenticación JWT, tenemos a nuestra disposición algunos métodos para generar tokens, determinar si el token es válido, descifrarlo para obtener datos relativos a la autenticación, etc.

La configuración anteriormente mencionada y las primeras pruebas de concepto relativas a la autenticación en la API REST fue el primer trabajo de investigación que se realizó en la PEC 2 ya que es la base de la comunicación entre cliente y servidor y uno de los primeros pasos importantes que debimos abordar para tener una base sobre la que construir las funcionalidades de la aplicación. Para la consecución de este objetivo me fue especialmente útil esta serie de videos [7]

https://www.youtube.com/playlist?list=PLe30vg_FG4OSbizS6Gpw_LICp9zBcmjZU

donde se explica cómo montar la comunicación angular-laravel basado en llamadas JWT autenticadas y el mecanismo de creación de tokens, configuración de los entornos, etc.

A continuación, se muestran algunas pantallas y fragmentos de código que ilustran las partes fundamentales en el mecanismo de comunicación implementado:

- La generación de un token JWT válido por parte del servidor la realizamos en el método de **login**, siempre y cuando se reciban en la *request* unas credenciales

válidas. Esto lo conseguimos con el método **attempt** que nos proporciona la librería (<https://jwt-auth.readthedocs.io/en/develop/auth-guard/>)

```
public function login()
{
    Log::info('Petición de login');
    $credentials = request(['email', 'password']);

    if (! $token = auth()->attempt($credentials)) {
        Log::info('El usuario no tiene credenciales..');
        return response()->json(['error' => 'No existe email o password'], 401);
    }
    Log::info('Login realizado correctamente');
    return $this->respondWithToken($token);
}
```

Ilustración 1: Obtención de un token JWT a partir de credenciales del usuario

- Una vez obtenemos un token mediante el proceso de login, el cliente angular almacena dicho token en el estado de la aplicación

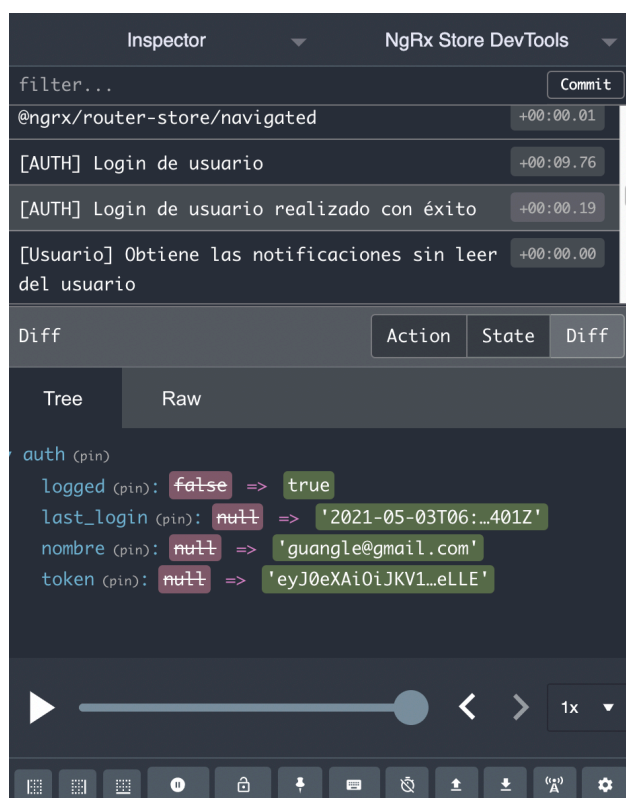


Figura 32: Token almacenado en el state de la aplicación angular

- Por último, una vez que el cliente tiene un token válido, puede incluirlo en la cabecera de las peticiones http que realice al backend

```

this.authInfo$.subscribe((login_state) => {
  //Estamos atentos a si cambia algo en el estado del login, para
  //actualizar el token incluido en la cabecera de las peticiones
  //http
  this.token = login_state.token;

  this.httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + this.token })
  };
});

```

Ilustración 2: Inclusión del token JWT en las cabeceras http enviadas por el cliente

Hay que destacar que JWT básicamente es un mecanismo de autenticación / autorización. JWT es un estándar de codificación y se puede descodificar fácilmente la información que almacena cada uno de los tokens (hay páginas para ello [8]). Si queremos tener una seguridad completa en la comunicación entre los clientes angular y el servidor sería absolutamente necesario que la web se publicara bajo HTTPS, ya que con dicho protocolo la comunicación entre cliente y servidor está cifrada y nadie que esté ‘al medio’ de esa conversación puede descifrar el contenido de los mensajes intercambiados.

Hay otros aspectos fundamentales de la seguridad que se han tenido en cuenta, por ejemplo, guardar las contraseñas almacenadas en la base de datos para que ni el administrador del sistema pueda acceder a una información tan sensible de los usuarios:

The screenshot shows a database management interface with a table named 'users'. The table has columns: id, nombre, apellido1, apellido2, es_cocinero, email, email_verified_at, and password. The 'password' column contains long, alphanumeric strings, indicating that the passwords are encrypted. A red box highlights the 'password' column header and the first row's password value.

id	nombre	apellido1	apellido2	es_cocinero	email	email_verified_at	password
1	Poi Villalobos Jimémez Segundo	Manuela	Dr. Angeles Anguiano	0	aitana84@example.net	2021-04-23 22:29:15	\$2y\$10\$/3/EHFdP06cx1P71cys3Od5ReGllk4z0TvBOPgA1QB...
2	Josefa Baca Hijo	Sr. Santiago	Poi Carreón	0	balderas.ignacio@example.org	2021-04-23 22:29:15	\$2y\$10\$/0/Kg6gJhNqW1dTHzoleRs1rz6SFTYxy41w80EU7L...
3	Srta. Altana Sauceda	Srta. Juana Fajardo	Cristian Arriaga	0	rios.lucia@example.org	2021-04-23 22:29:15	\$2y\$10\$/stx/3pVSTsy8yAplbvq2r;ymvSEsD.eQJqUpI7G7Cmf...
4	D. Dario Haro Segundo	Lola Rincón	D. Izan Escobar Segundo	1	jordi12@example.net	2021-04-23 22:29:15	\$2y\$10\$/vqyEIZVC/SQvVldXyYb.111BpO.4HBdIEQdgIFUq...
5	Berta Salgado	Jorge Ávila Tercero	Ing. Noa Mendoza	1	enrique84@example.net	2021-04-23 22:29:15	\$2y\$10\$/mYLDQzEjMqleL1anXH0v.y1rNo8gkH2WXHL9P7Tj...
6	Mario Vicente	Jaime Carrero	Lic. Alicia Vergara Segundo	0	veronica.castellanos@example.net	2021-04-23 22:29:15	\$2y\$10\$/SFA2iFqzHz4ed10Ls40iOj97R7gHCL31268hmjg...
7	Juan Jesús	Gutiérrez Ramos		1	guangle@gmail.com	2021-04-23 22:29:15	\$2y\$10\$/mVlug1Dydm.pxs55RSORBejMBCBISWmiPIUtrdHK70IK...

Figura 33: Contraseñas cifradas en base de datos

Por último, se han aplicado los mecanismos de seguridad en las aplicaciones angular vistos en la asignatura de ‘*Javascript con Framewroks*’: los **route guards**.

Las guardas en Angular son un tipo especial de componentes cuyo propósito es determinar si un usuario puede activar o no una determinada ruta. En nuestro caso, como se ha mencionado antes, hemos implementado el patrón de diseño REDUX por lo que tenemos un estado de la aplicación, en

el que tenemos información precisa relativa al usuario autenticado (si está autenticado, perfil, token JWT...).

Por tanto, hemos incluido en la aplicación un *Guard* llamado **UsuarioLogadoGuardService** que en el método *canActivate* (que determina si la ruta que recibe como parámetro se puede ejecutar o no) comprueba el estado de la aplicación para devolver *true* únicamente cuando verifiquemos, a través del estado, que la ruta la está ejecutando un usuario autenticado.

```
canActivate(route: ..): boolean.. {
  return this.store$.select(appState => appState.auth.logged)
    pipe(map(logged => {
      if (!logged) {
        this.router.navigate(['no_autenticado'])
      }
      return logged;
    }));
}
```

Ilustración 3: Route Guard para proteger rutas que necesitan autenticación

Una vez implementado el Guard podemos asociarlo a aquellas rutas que únicamente pueda ejecutar un usuario logado:

```
..
{
  path : 'nuevo',
  component : NuevoTaperComponent,
  canActivate: [UsuarioLogadoGuardService]
},
..
```

Ilustración 4: Asociación de un Route Guard a una ruta de la aplicación

De esta forma, si intentamos ejecutar sin estar autenticado una de las rutas protegidas por la guarda (por ejemplo, escribiendo directamente la url) somos redirigidos al componente **NoAutenticado**, que muestra un mensaje informativo al usuario indicando que no tiene los permisos suficientes.

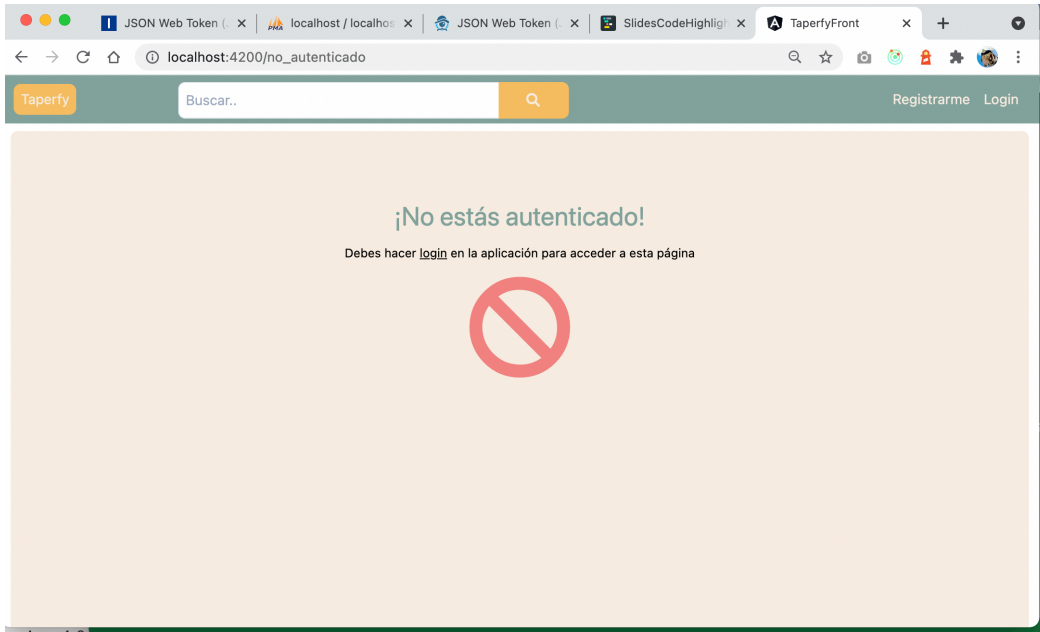


Figura 34: Página indicando al usuario que no tiene permisos para ejecutar la ruta

15. Tests

Junto al análisis y la implementación del código, otra parte fundamental en la construcción del producto es la realización de pruebas de diferentes tipos con el fin de garantizar la corrección de lo implementado, y tener una herramienta para garantizar que cambios futuros en la implementación no alteren la funcionalidad previamente desarrollada.

En este sentido, los test son de vital importancia en el proceso de desarrollo y durante la planificación del proyecto se tuvo en cuenta que se debe dedicar un tiempo considerable a la escritura de pruebas automáticas de código, así como tiempo para pruebas de usuario, aceptación, etc.

Aunque las pruebas han estado presentes durante todo el ciclo de construcción del producto (cada vez que implementaba un componente o servicio, realizaba pruebas de usuario sobre él), hemos dedicado buena parte del tramo final (PEC 4) a la escritura de test unitarios con el objetivo de tener una batería de unit-tests que poder ejecutar ante cualquier nuevo cambio en el código o nuevo entorno donde se despliegue la aplicación.

Por tanto, en la última iteración de la construcción de la aplicación se han implementado los test unitarios haciendo uso de los mecanismos que proporciona Angular (Jasmine, añadiendo Mocha reporte para una mejor visualización de los test) [9] [10]. La creación de pruebas tiene una curva de aprendizaje un tanto elevada al principio y la sintaxis me ha resultado un tanto ardua en un primer momento, pero una vez se entiende la dinámica y se consigue hacer mocks y doubles de los datos del store y los servicios de negocio, la inclusión de nuevos test que aumenten la cobertura de código probado es algo sencillo y repetitivo.

Los test unitarios se han desarrollado esencialmente para los componentes, probando algunos elementos de la interfaz y el comportamiento de estos antes determinados datos de entrada. También se han escrito test unitarios relacionados con la implementación del patrón redux: acciones, efectos y reducers.

A modo de ejemplo, se muestra a continuación un test unitario en Angular. La casuística es diversa, pero en la mayoría de los casos se deben establecer unas condiciones previas a la ejecución del test, emular los datos del estado y los servicios de negocio afectados para posteriormente hacer aseveraciones referentes a la interfaz o datos almacenados tras la ejecución del alguna función o invocación de un servicio de negocio.

```

//Ejemplos de test unitarios en Angular
//Inicialización del test
describe('TaperCardComponent', () => {
  let component: TaperCardComponent;
  let fixture: ComponentFixture<TaperCardComponent>;
  //Creamos un store para los tests
  let store: MockStore;
  const initialState = estado_inicial_mock;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ TaperCardComponent ],
      imports: [RouterTestingModule.withRoutes([]), HttpClientTestingModule, BrowserAnimationsModule],
      providers: [
        provideMockStore({ initialState })
      ]
    })
    .compileComponents();
    //Utilizamos un store mockeado
    store = TestBed.inject(MockStore);
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(TaperCardComponent);
    component = fixture.componentInstance;
    //Datos de entrada del componente..
    component.taper = taper_test_1;

    fixture.detectChanges();
  });
  //..

  //Probamos algún elemento de la interfaz
  it('El precio de tupper se visualiza correctamente', () => {
    let precio = fixture.debugElement.query(By.css('#card-precio_taper')).nativeElement;
    expect(precio.textContent).toContain( taper_test_1.precio );
  });

  //Ejemplo de test de comportamiento de un componente
  it('Se invoca el método valorarReserva() cuando se hace click en una de las estrellas', fakeAsync(() => {
    spyOn(component, 'valorarReserva');
    let button = fixture.debugElement.nativeElement.querySelector('#reserva-valorar-1');
    button.click();
    tick();
    expect(component.valorarReserva).toHaveBeenCalled();
  }));
  //..

  //Ejemplo de test sobre un efecto ngrx
  //Cuando se invoca a una actualización de filtros, esperamos que se despache la acción para listar los tappers
  //cuando el backendService obtenga una respuesta del método getTuppers
  it('Efecto actualizarFiltros$', () => {
    let actions$ = of( new TaperActions.ActualizarFiltros({estrellas: 1, tipo_tuper: 2, precio_ma...}), );
    let backendService = jasmine.createSpyObj('BackendService', ['getTupper']);
    let getTupperSpy = backendService.getTupper.and.returnValue(of(taper_test_1));

    const effect = new TaperEffects(actions$, backendService, null);
    effect.loadTuppers$.subscribe(a => {
      expect(a.type.toEqual(TaperActions.TaperActionTypes.LISTAR) );
    });
  });
}

```

Ilustración 5: Test unitarios en Angular

Se han implementado un total de 149 test unitarios sobre los diferentes elementos de la aplicación. Ejecutando el comando `ng test`, angular-cli nos mostrará un informe de los resultados de la ejecución de los test tanto en el navegador como por consola:

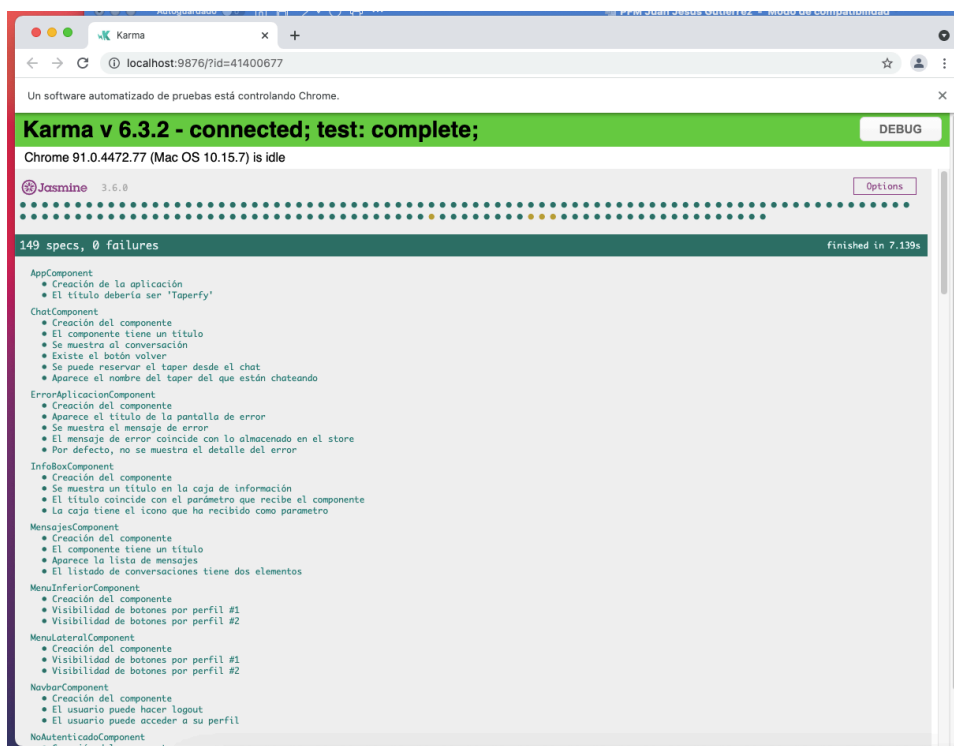


Figura 35: Informe tras la ejecución de test unitarios, web

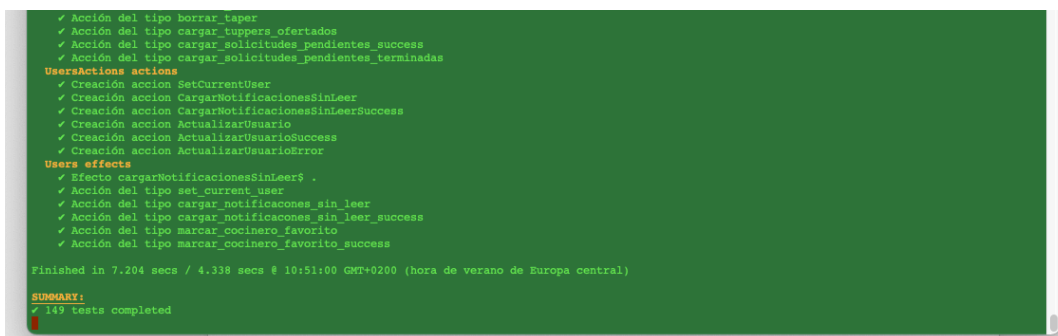


Figura 36: Informe tras la ejecución de test unitarios, consola

Por último, una vez que hemos dispuesto del servidor de producción AWS donde desplegar la aplicación y que esté disponible a través de internet, he pedido a diferentes amigos y familiares que hicieran pruebas en sus dispositivos y me dieran feedback. Al tratarse de un proyecto académico, en el que no tenemos cliente, podemos considerar estas pruebas como pruebas de aceptación ya que han sido usuarios reales, sin tener nada que ver con el mundo del desarrollo ni conocer la aplicación previamente, los que me han transmitido el feedback y han validado las funcionalidades.

16. Versiones de la aplicación

Como se indicó en la fase de planificación del proyecto, la construcción del sistema se ha dividido en 4 grandes bloques, coincidiendo estos con las entregas de las diferentes PEC. En la primera PEC se hizo una exposición de la idea y un análisis inicial muy a alto nivel, y en las siguientes entregas hemos tenido versiones de la aplicación que han ido incluyendo sucesivamente características y mejoras.

Versión Alpha	
Fecha de entrega	31 de Marzo de 2021
PEC	2
Descripción	<p>Esta versión prácticamente carecía de funcionalidades: solo implementaba el login y un listado básico de tappers, pero durante este periodo se trabajó en la estructura de la aplicación, análisis, definición de los componentes que se iban a usar y se hicieron las investigaciones necesarias para desarrollar las historias de usuario que surgieron en el análisis.</p> <p>La versión Alpha ya incluía un diseño muy parecido al final, aunque quedaba mucho trabajo para hacer la aplicación responsive y muchos aspectos de usabilidad.</p> <p>Esta versión también incluye una primera versión del modelado de la aplicación, así como los primeros métodos REST ofrecidos por la API.</p>

Versión Beta	
Fecha de entrega	09 de Mayo de 2021
PEC	3
Descripción	<p>En esta versión se implementan casi el 100% de las funcionalidades inicialmente propuestas en el análisis:</p> <ul style="list-style-type: none"> - Login - Registro - Recuperación de contraseña - (Cocinero) Publicar nuevo taper - (Comprador) Listar y filtrar resultados - (Comprador) Solicitar una reserva - (Comprador) Puntuar producto comprado - (Comprador) Marcar cocinero como favorito - (Cocinero) Mi panel - (Cocinero) Aceptar reservas - Notificaciones en la aplicación - Chat entre usuarios <p>El modelo de datos de esta versión ya es el definitivo, pero aún queda como tarea pendiente la generación automática de datos de prueba ‘de más calidad’ para la presentación y lanzamiento inicial</p>

	<p>del proyecto.</p> <p>Aún estando casi totalmente acabado funcionalmente, la versión Beta de la aplicación aún no ha sido suficientemente probada y quedan pulir muchos detalles a nivel de interfaz.</p> <p>Adicionalmente, el código no está todo lo refactorizado y bien estructurado como debería en una versión final de la aplicación.</p>
--	--

Versión 1.0	
Fecha de entrega	07 de Junio de 2021
PEC	4
Descripción	<p>En la versión final de la aplicación, se incluyen todas las funcionalidades inicialmente definidas en el análisis, la aplicación es totalmente responsive, adaptándose a diferentes tamaños de pantalla y se ha hecho una revisión del rendimiento así como una refactorización de las partes más importantes del código.</p> <p>Adicionalmente, se han realizado pruebas tanto a nivel de código como pruebas con usuarios finales para garantizar la corrección de lo implementado y recibir feedback.</p> <p>Si se implementaran futuras versiones de la aplicación, el RoadMap sería definido con las mejoras propuestas en el punto ‘Proyección a futuro’ de este documento.</p>

Tabla 5: Versiones de la aplicación

Las diferentes versiones se han ido distribuyendo comprimidas en las entregas de cada una de las PEC, pero se ha proporcionado acceso al repositorio de código al tutor de la asignatura con el objetivo de que este pudiera ir viendo el progreso en tiempo real, sin tener que esperar a cada entrega de código.

17. Consideraciones de implementación

En este apartado se expondrán algunas consideraciones que han surgido durante la implementación del proyecto y que a mi juicio son reseñables para entender el código desarrollado y otras características de la aplicación, no directamente relacionadas con la funcionalidad de la misma, que van en pos de la calidad final del producto. Me ha sido posible aplicar la mayor parte de estas características adicionales y mejoras gracias a los conocimientos adquiridos en diferentes asignaturas del máster.

17.1 Uso del patrón Redux y Ngrx

Cómo se ha comentado anteriormente en el punto relativo a la arquitectura de la aplicación, durante la fase de análisis de Taperfy se decidió usar el patrón de diseño **REDUX**, que en los proyectos Angular toma forma gracias a la librería **Ngrx** [11].

REDUX es un patrón de diseño que dota a una aplicación front-end de estado. Los datos del estado o storage son ‘la única fuente de verdad’ en nuestra aplicación y están almacenados en memoria (aunque gracias al componente **ngrx-storage-localstorage** [12] lo mantenemos sincronizado con el local storage del navegador.

El estado solo puede ser cambiado en la aplicación mediante unas funciones especiales denominadas *reducers*, que deben tomar como parámetro una acción realizada por el usuario y el estado actual y devolver un nuevo estado. Así pues, tenemos perfectamente tipificado los datos que componen el estado de la aplicación, que acciones se pueden lanzar en el sistema y dónde se modifican estos datos (*reducers*), por lo que la trazabilidad es total.

Otro concepto incluido en el patrón de diseño REDUX – Ngrx es el de efecto. Un *effect* se dispara cuando se ejecuta una de las acciones pre-establecidas y puede (o no) disparar otra acción. Normalmente se utilizan para llamar a proveedores externos de datos, como el backend de la aplicación.

El funcionamiento básico del REDUX-Ngrx se ilustra con un ejemplo:

- El usuario submite el formulario de login, y nuestro componente dispara la acción LOGIN.
- En el reducer, el disparo de la acción LOGIN implica en el estado de la aplicación los siguientes cambios:
 - o La variable *cargando* pasa a tener el valor *true*
 - o La variable *mensaje* pasa a tener el valor ‘*Realizando login, espere..*’
- Aquellos componentes que estuvieran observando las variables *cargando* y *mensaje* de nuestro estado, se actualizan automáticamente en consecuencia. Por ejemplo, poniendo un spinner o mostrando el nuevo mensaje de espera en la cabecera.
- Se ejecuta un efecto asociado a la acción LOGIN, consistente en invocar al backend con las credenciales. Si el login es satisfactorio y obtenemos un token JWT, desde el efecto se invoca a la acción LOGIN_SUCCESS, pasándole el token como parámetro
- En el reducer la acción LOGIN_SUCCESS cambia nuestro estado de la siguiente forma
 - o *cargando* → *false*
 - o *mensaje* → ‘*Login realizado correctamente*’
 - o *usuario_logado* → el procedente de la acción

- Todas los componentes que estuvieran observando esa parte del estado, se actualizan en consecuencia. Por ejemplo, aparece el nombre del usuario logado en la barra de navegación.

En el ejemplo anterior queda expuesta la mecánica de REDUX y la principal ventaja es que tenemos todo perfectamente trazado, el punto de cambios está localizado y tenemos potentes herramientas de depuración con las que podemos consultar el estado e incluso navegar por las acciones ejecutadas hacia atrás y hacia adelante.

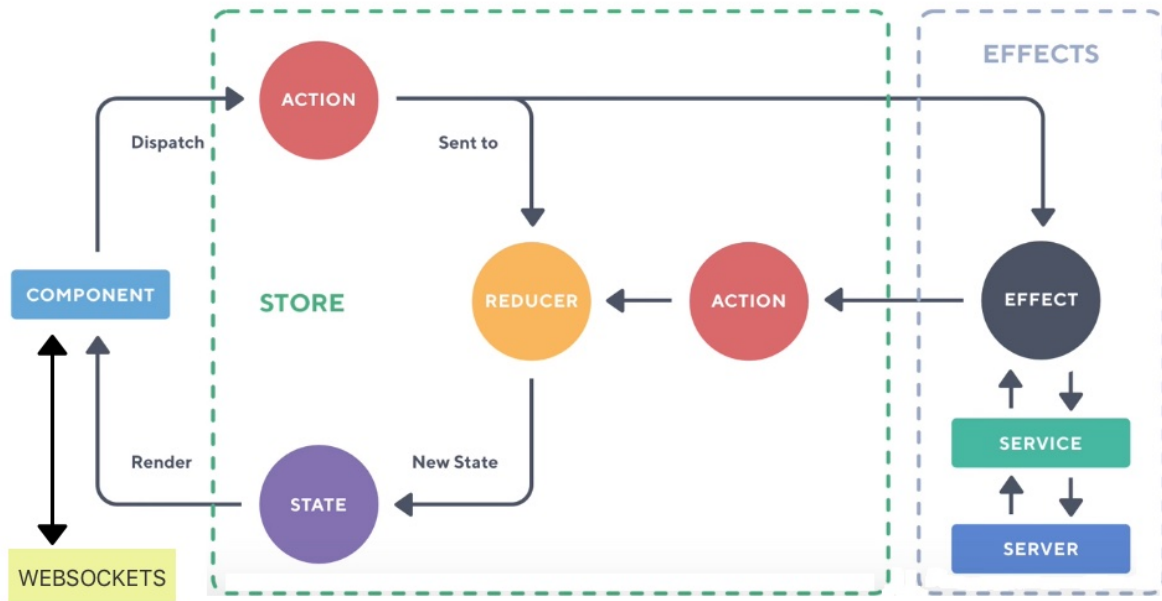


Figura 37: Redux / NgRx

El cambio a Redux fue inicialmente costoso, diría que la curva de aprendizaje es elevada y que es una técnica que tiene sentido para proyectos medianos o grandes, pero en lo personal quería afrontar y conocer en profundidad el patrón y una vez implantado se obtienen múltiples ventajas como una trazabilidad total de lo que sucede en la aplicación, mayor orden, control y sobre todo un mayor desacoplamiento entre los elementos, ya que en este modelo los componentes no conocen nada del backend de la aplicación, ni a que fuente de datos se conectan, ni siquiera los servicios de negocio que se utilizan. Básicamente emiten acciones y observan la parte de la aplicación que les interesa (los selectores de partes del estado son observables) para mostrar los datos en la pantalla.

17.2 Tailwind y Diseño web responsive

Como herramienta CSS para maquetar la web se ha utilizado **Tailwind CSS** [13] [14], un framework que se estudió en profundidad en la asignatura **‘Herramientas HTML y CSS II’** del máster.

Tailwind CSS es una librería relacionada con los estilos que cambia un poco el paradigma respecto a las librerías de componentes más usadas (por ejemplo, *Bootstrap*) ya que aquí tenemos una herramienta que sigue la filosofía **utility-first**, es decir, proporcionar ‘pequeñas’ clases css que tienen un único propósito y que puedes aplicar en diferentes partes del código.

Esto difiere un poco respecto a Bootstrap, ya que este framework se trata más bien de una librería ‘de componentes’, donde ya nos dan piezas hechas que podemos insertar en nuestro código para solucionar problemas comunes.

En principio, esta aproximación hace que trabajemos un poco más a ‘bajo nivel’, por lo que tenemos un mayor control y personalización. Además, me ha dado la impresión de que evitamos un efecto poco deseable que suele pasar con bootstrap, que es que todas las páginas que utilizan dicho framework acaban pareciéndose entre sí.

Tailwind ofrece multitud de pequeñas clases de utilidad (para añadir márgenes, padding, modificar fuentes, colores, etc.) con las que prácticamente he podido implementar todo lo que he ideado a nivel de interfaz de usuario. Aún así, el framework es extensible y permite definir medidas propias, paleta de colores, animaciones...

Toda la configuración de tailwind reside en el fichero **tailwind.config.js**:

```
...
theme: {
  extend: {
    //definimos nuestra paleta de colores personalizada
    //Probamos con la paleta vintage
    //https://coolors.co/f6bd60-f7ede2-f5cac3-84a59d-f28482
    colors: {
      paleta_tf_amarillo: '#F6BD60',
      paleta_tf_beige: '#F7EDE2',
      paleta_tf_rosa1: '#F5CAC3',
      paleta_tf_verde: '#84A59D',
      paleta_tf_rosa2: '#F28482',
    },
  },
  height: {
    "8porciento": "8%",
    ...
  }
}
```

Ilustración 6: Personalización del framework Tailwind

Uno de los aspectos más potentes de esta librería es el mecanismo que incluye para trabajar con diferentes medidas de pantalla. Tailwind define una serie de prefijos (sm, md, lg y xl) que representan diferentes anchos máximos de pantalla de forma que podemos asociar estos prefijos a cada clase de utilidad y dicha clase solo sería aplicada a los anchos de pantallas representados por el prefijo. Ejemplos:

```
<!--  
Ancho del 100% por defecto  
y ancho de 9/12 para pantallas tipo LG  
-->  
<div class="w-full lg:w-9/12">  
  
<!--  
mt = margin top  
Diferentes margin-top en función del ancho  
-->  
<div class="sm:mt-1 md:mt-2 lg:mt-3 xl:mt-4">
```

Ilustración 7: Implementación de diseño responsive en tailwind

El resultado ha sido que hemos implementado de una forma sencilla una aplicación que se adapta al ancho de diferentes pantallas:

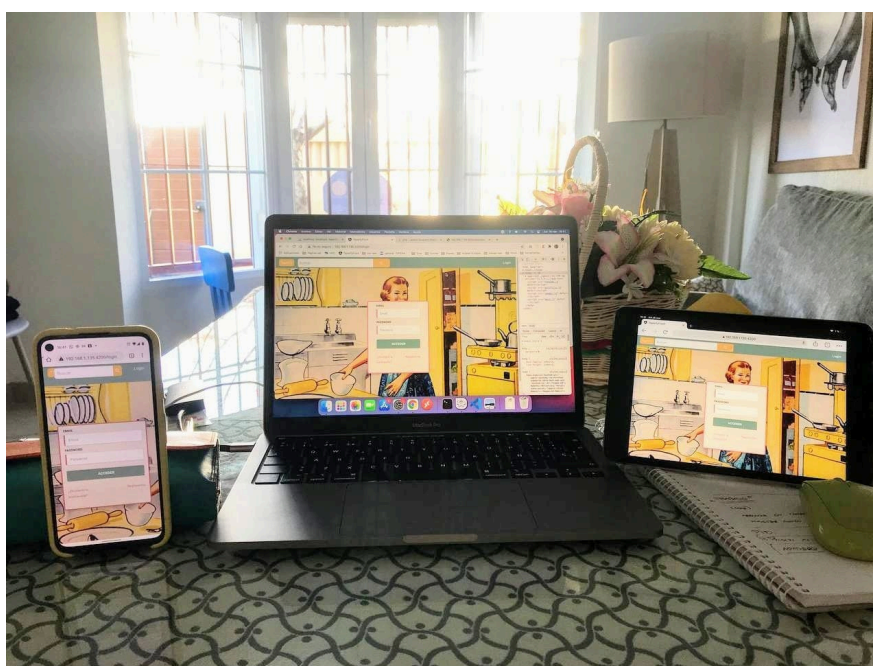


Figura 38: Interfaz adaptada a diferentes pantallas

En la fase de construcción del proyecto para producción se eliminan el CSS que no es utilizado por nuestro sistema y el peso de los estilos queda ampliamente reducido en la versión de producción.

17.3 Animaciones

Con el objetivo de dar cierta sensación de fluidez a la interfaz, mejorar la experiencia de usuario y para reforzar los conceptos relativos a las animaciones CSS vistos en diferentes asignaturas del máster, se han incluido una serie de animaciones en el proyecto.

Para ello, se ha hecho uso del paquete **Angular Animations** [15] que permite añadir animaciones mediante código en algunos elementos de nuestros componentes. Para instalar la dependencia se ha ejecutado:

npm i @angular/animations

y una vez importados los componentes en los módulos que lo requieran se pueden definir animaciones que podemos aplicar sobre componentes completos o sobre elementos de estos. Por ejemplo, definiendo la animación:

```
export const fadeAnimation_configurable = trigger('fadeAnimation_configurable', [
  transition(':enter', [
    style({ opacity: 0 }), animate('{{ time }}ms', style({ opacity: 1 })))
  ],
  transition(':leave',
    [style({ opacity: 1 }, animate('300ms', style({ opacity: 0 })))
  ]
);
```

Ilustración 8: Definición de animación con angular animations

Conseguimos aplicar un fade a un elemento y que dure el número de milisegundos que se recibe en el parámetro *time*. Una vez definida la aplicación e importada en el componente, podemos aplicarla directamente en el html:

```
<div class=" rounded overflow-hidden shadow-lg my-2 bg-white relative"
  [@fadeAnimation_configurable]="{value: ':enter', params: { time: 200}}">
  ...
```

Ilustración 9: Ejemplo de aplicar una animación angular a un elemento

Con lo que conseguimos aplicar una animación al div consistente en que pase de opacidad 0 a 1 durante 300 milisegundos

Se han aplicados otras animaciones a diferentes partes del código y, adicionalmente, también se muestra alguna animación ‘lottie’ [16] en tiempos de espera, páginas de error... con el objetivo doble de dotar a la aplicación de más dinamismo y aprender algunas técnicas relativas a animaciones en proyectos angular.

17.4 PWA

Durante el proceso de desarrollo, se han realizado las acciones necesarias para que el proyecto implementado sea una **Progressive Web Application**, una tecnología que, entre otras ventajas, permite añadir aplicaciones web a nuestros dispositivos (ya sean pcs, tablets o móviles) como si estuvieran instaladas en el mismo, dando al usuario la sensación de estar utilizando una aplicación ‘nativa’ sin tener realmente nada instalado en su equipo.

Buenos ejemplos de aplicaciones web progresivas son Twitter o Facebook, en ambos casos si abrimos la página desde el móvil (por ejemplo, con Chrome) se nos permite añadir a la pantalla de inicio un enlace y desde ese momento podremos acceder a dichos servicios como si estuviéramos utilizando las apps nativas del Google Play o el App Store. Adicionalmente, las aplicaciones web progresivas también pueden instalarse en ordenadores personales, otros ejemplos de PWAs pueden ser Google Drive o Postman.

Esta difuminación de la barrera entre web y aplicación es posible gracias a tecnologías recientes como los *Service Workers* y afortunadamente Angular proporciona los mecanismos para que convertir una web desarrollada con este framework en una PWA sea un proceso sencillo.

Basta con ejecutar el comando:

ng add @angular/pwa

y se crearán y actualizarán todos los ficheros necesarios (iconos, configuración del service worker en angular.json, configuración de la pwa en manifest.webmanifest...). También se registra el service worker (scripts que el navegador puede ejecutar en segundo plano, aunque no se esté mostrando la web) de la PWA en el módulo principal de la aplicación:

```
ServiceWorkerModule.register('ngsw-worker.js', {
  enabled: environment.production,
  registrationStrategy: 'registerWhenStable:30000'
})
```

Ilustración 10: Registro del service worker

Tras instalar una pequeña herramienta para la generación de los iconos necesarios [17] y configurar algunos aspectos de la visualización de la Progressive Web Application en el fichero manifest, si servimos la aplicación en un servidor http el navegador nos ofrece la posibilidad de ‘instalarla’ en nuestro equipo, tanto en el dispositivo móvil como en el portátil:

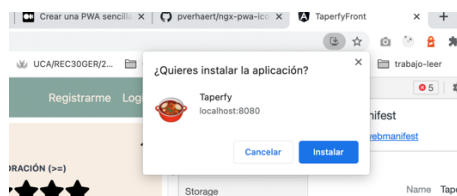


Figura 39: Instalando Taperfy como una PWA



Figura 40: Taperfy como PWA, instalada en un teléfono móvil

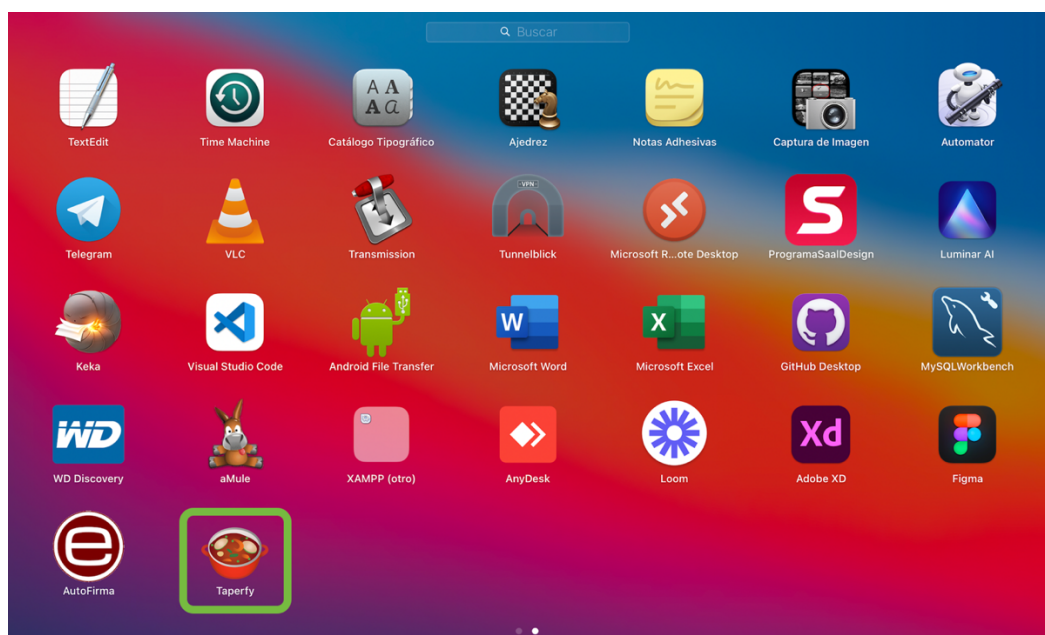


Figura 41: Taperfy como PWA, instalada en un MacBook

Nota: la aplicación no se muestra como una PWA durante el proceso de desarrollo, para que se muestren las opciones propias de la PWA se debe compilar una versión para producción del producto y servir el resultado en un servidor http. En nuestro caso, para la ejecución de pruebas se ha instalado el paquete **http-server**.

17.5 Rendimiento, lazy loading y precarga de módulos

Junto a los conocimientos y habilidades necesarias para la construcción de una aplicación Angular, en la asignatura *Desarrollo FrontEnd con Frameworks (I y II)* se han estudiado diferentes técnicas de optimización y configuración para el entorno de producción que se han aplicado durante la implementación de este proyecto de fin de máster.

El *Lazy Loading* es uno de los principales mecanismos de optimización que se pueden aplicar a la hora de poner una página desarrollada con Angular en producción. Usando esta técnica, los diferentes componentes, guardas, rutas... (todo lo que se puede agrupar en un módulo) se irán cargando en la aplicación cliente a medida que se vaya necesitando.

Por el contrario, si no aplicamos *Lazy Loading* y añadimos todas las rutas (y sus componentes asociados) en el módulo principal de la aplicación, en la primera *request* que realice el cliente para cargar nuestra aplicación, el servidor tendrá que enviar todo el javascript (aunque sea minificado y agrupado) de todos los módulos que componen el sistema, lo que puede ser potencialmente muy costosos en términos de tráfico de red.

Quizás en nuestra aplicación y en las prácticas realizadas durante el máster no se aprecia con suficiente claridad la mejora, ya que Taperfy es una aplicación mediana, con únicamente 4 módulos, unos 30 componentes y unos pocos servicios de negocio. El *Lazy Loading* va cobrando sentido conforme va creciendo el tamaño de la aplicación ¿Por qué cargar en la primera petición los 400 componentes de una web si únicamente se van a utilizar 3? La filosofía detrás del lazy-loading es modularizar la aplicación e ir sirviendo esos módulos bajo demanda, con la consiguiente mejora en la percepción de velocidad de carga en el usuario final.

A grandes rasgos, para aplicar el lazy-loading en un proyecto hay que hacer los siguientes pasos:

1. Dividir el proyecto, creando módulos que agrupen funcionalmente los componentes, interceptores, servicios de negocio, guardas, etc. Existirán elementos que no se puedan separar ya que serán compartidos por todos los módulos de la aplicación.

En el desarrollo de Taperfy la modularización de la aplicación se ha realizado en el intervalo que va desde la entrega de la PEC 2 a la entrega de la PEC 3, ya que en ese momento ya teníamos definidos todos los componentes necesarios y fue posible agruparlos.

Para generar módulos por línea de comandos se utilizó la orden:

ng generate module nombre_modulo --routing

Y en nuestro caso se han agrupado los elementos que componen la aplicación en cuatro grandes bloques: común, usuarios, tapers y reservas.

2. Reordenar las rutas para que cada módulo cuente con su propio fichero de routing y en el routing principal de la aplicación se realicen los import a los diferentes enrutamientos de cada módulo. Aplicándolo a nuestro proyecto:

```

..
{
  path: "usuario",
  loadChildren: () =>
    import("../app/modules/usuario/usuario.module").then((m) => m.UsuarioModule),
},
{
  path: "reserva",
  loadChildren: () =>
    import("../app/modules/reserva/reserva.module") then((m) => m.ReservaModule),
},
..

```

Ilustración 11: Carga de los módulos de la aplicación

En el ejemplo mostrado anteriormente, si un usuario invitado navega por la aplicación, pero no visualiza ni ejecuta ninguna funcionalidad relacionada con las reservas (no lanza ninguna de las rutas relacionadas ni muestra ningún componente) el módulo no se descargará en el navegador con el consiguiente ahorro en el tráfico de red y tiempo de carga de la página.

En las primeras versiones de la web (el código entregado en la PEC 2) todos los elementos desarrollados se incluyeron dentro del código principal de la aplicación y en la iteración de la PEC 2 a la PEC 3 se dividió en módulos el proyecto. En principio no he notado grandes cambios el rendimiento ya que incluso la versión ‘no optimizada’ tenía muy buenos tiempos de carga de página y de respuesta. Como se comento anteriormente, este tipo optimizaciones va cobrando sentido conforme va creciendo el proyecto. En cualquier caso, tener los elementos divididos en módulos también implica una mejor organización del código fuente, mantenibilidad y desacoplamiento.

La información necesaria para implementar y configurar la carga perezosa de los módulos se encuentra y sigue con mucha claridad en la documentación oficial del framework [18].

Hay una optimización adicional que se estudió en las asignaturas del máster que versaban sobre Angular y que hemos aplicado en Taperfy. Una vez tenemos la aplicación dividida en módulos y hemos descargado los componentes fundamentales para mostrar únicamente lo que está utilizando el usuario... ¿Podríamos anticipar la siguiente descarga y tenerla preparada para cuando el usuario la necesite?

De esta forma, ya tendríamos ‘el siguiente módulo’ necesario disponible en el cliente y no tendríamos descargarlo cuando el usuario lo quiera utilizar. Eso sí, la descarga de lo que ‘todavía no se ha solicitado’ se debe hacer únicamente cuando la aplicación esté inactiva, no esté ejecutando ningún proceso ni descargando nada, de esta forma nos aseguramos que sea transparente para el usuario.

Esta manera de cargar los elementos se denomina en Angular **Pre-Loading** y para utilizarla basta con añadir la opción **preloadingStrategy: PreloadAllModules** en el fichero donde tengamos la carga principal de rutas de la aplicación (en mi caso fichero *app-routing.module.ts*)

```
...
RouterModule.forRoot(
  appRoutes ,
  {preloadingStrategy: PreloadAllModules,}
)
..
```

Ilustración 12: Pre-loading de módulos

De nuevo, el cambio no es significativo en una aplicación del tamaño de este proyecto, pero es una configuración que merece la pena tener activada porque mejorará el rendimiento de la aplicación conforme esta vaya creciendo, incorporando módulos y añadiendo más elementos a los módulos ya implementados.

17.6 Chat: polling vs websockets

Durante la fase de análisis y primeros pasos de la implementación de Taperfy, surgió la idea de posibilitar en la aplicación una comunicación entre usuarios cocineros y compradores a través de un pequeño chat, ya que en el contexto de nuestra web puede tener valor un mecanismo para resolver dudas sobre los productos, hacer peticiones, negociaciones, etc.

La implementación de esta funcionalidad no resulta demasiado problemática a nivel de base de datos y métodos necesarios en el api rest. Basta con añadir una tabla de mensajes intercambiados, que tenga un par de claves foráneas que permitan enlazar con los usuarios asociados al mensaje, una fecha de envío y unas banderas para determinar si se ha leído.



Figura 42: Tabla para almacenar los mensajes

A su vez, nuestro proyecto Laravel debe ofrecer las operaciones necesarias para operar sobre la tabla de mensajes anteriormente descrita: nuevo mensaje, consultar mensajes, marcar mensajes como leídos, etc.

```
//métodos de la api relacionados con el chat
Route::get('mensajes/conversacion', 'MensajesController@conversacion');
Route::get('mensajes/conversacionTaper', 'MensajesController@conversacionTaper');
Route::post('mensajes/nuevo', 'MensajesController@nuevoMensaje');
Route::get('mensajes/conversaciones', 'MensajesController@conversaciones');
Route::get('mensajes/obtenerNumMensajesNuevos', 'MensajesController@obtenerNumMensajesNuevos');
```

Ilustración 13: Parte de la API relacionada con el intercambio de mensajes

Implementar lo anteriormente expuesto en el backend así como el consumo de los servicios en el front no fue difícil, pero la inclusión de un chat en tiempo real entre usuarios en nuestra aplicación plantea un problema serio de uso de recursos y escalabilidad.

El problema en esencia es el siguiente: si un cliente tiene abierto un chat con otro usuario ¿Cómo sabe si el otro usuario le ha escrito un mensaje nuevo? Recordemos que nuestra pretensión es que sea un chat en tiempo real y por tanto esperamos que el retraso entre el envío y la recepción del mensaje por la otra parte sea el menor posible.

En las aplicaciones web clásicas, si un cliente requiere cierta información del servidor, le envía una solicitud (*request*) para obtener el recurso deseado (en nuestro caso, una solicitud para obtener los

mensajes nuevos). En este escenario, el inicio de la comunicación siempre es desde el cliente al servidor, en forma peticiones - respuesta que se ejecutan sobre el protocolo HTTP.

El problema es que nuestro cliente quiere obtener esa información ‘constantemente’ y estar continuamente preguntando al servidor (un mecanismo denominado polling) es muy costoso en términos de rendimiento, consumo de recursos, tráfico de red, sobrecarga, etc. Además, hay que tener en cuenta que potencialmente el servidor debe estar respondiendo a cientos o miles de clientes por lo que si todos estos le están lanzado peticiones continuas, con total seguridad se llegará al punto de no poder atender todas las peticiones y el rendimiento se degradará o se provocará la caída del sistema.

Con las dudas y la incertidumbre que me provocaba este posible problema, pregunté al tutor de la asignatura y me confirmó que efectivamente es un problema complejo, que el uso de polling tiene grandes dificultades en términos de escalabilidad y costes en el servidor y la construcción de un chat en tiempo real sería en si mismo un proyecto de fin de máster. Adicionalmente, me sugirió la idea de explicar el problema en esta memoria y explorar algunas alternativas para la implementación que serán apuntadas a continuación.

En cualquier caso, como toda la parte del backend la tenía hecha, únicamente con el objetivo de mostrar el funcionamiento y sabiendo que no tendremos problemas de rendimiento por la limitada cantidad de usuarios simultáneos que tendremos en nuestra aplicación, he dejado en la versión final entregada la implementación con polling (cada 10 segundos) sabiendo que no es una versión definitiva ni escalable y no sería apta para una aplicación real.

```
//Observable que emite cada 10 segundos..
checkChat$ : Observable<any> = timer(1000, 10000);
..
//Cuando checkChat$ emita (cada 10s),
//cambiamos el observable para devolver los
//mensajes consultados al backend
polling() {
  this mensajes$ = this checkChat$.pipe(
    switchMap( () =>
      this backendService.getMensajes(...))
  );
}
```

Ilustración 14: Consulta al backend por mensajes nuevos del usuario

Afortunadamente, para solucionar este problema y en general el de aquellas aplicaciones que requieran actualización de los datos desde el servidor en tiempo real, existen una serie de técnicas y tecnologías más apropiadas que la comunicación clásica petición – respuesta de la mayoría de las aplicaciones web: los **websockets**.

WebSockets es una tecnología que permite la creación de un canal bidireccional de comunicación entre cliente y servidor, posibilitando el intercambio de mensajes en ambas direcciones de una forma fiable y eficiente. Esta ampliamente soportado por los navegadores [19] y con esta API, una vez establecido el canal de comunicación, podemos conseguir que sea el servidor quien envíe un mensaje con información al cliente sin que este ‘se lo solicite’.

En el caso de nuestro chat en tiempo real, la solución óptima pasaría por utilizar esta tecnología para establecer dos canales abiertos de comunicación: entre cada uno de los usuarios que estén chateando y el servidor. De esta forma, cuando un usuario envía un mensaje al servidor, este lo almacenaría en su base de datos e informaría al usuario destinatario del mensaje (a través del websocket que tiene establecido con él) que tiene un mensaje nuevo. Así nos evitaríamos que ambos extremos estuvieran continuamente preguntando si hay mensajes pendientes de leer y solo se ejecutarían las comunicaciones estrictamente necesarias, con el consiguiente ahorro en términos de procesamiento y tráfico de red.

Como indicaba anteriormente y confirmaba el tutor de la asignatura, la implementación de este mecanismo no es trivial y supondría un sobre-esfuerzo que excedería con mucho al tiempo previsto para la implementación del proyecto (la implementación de un chat en tiempo real sería un proyecto en si mismo) por lo que se distribuirá la versión inicial (no óptima, con polling) de la aplicación y a continuación se hacen algunos apuntes sobre que librerías o frameworks podrían ayudarnos a conseguir el propósito de una aplicación en tiempo real eficiente y escalable:

- **socket.io** es una potente API WebSockets multiplataforma para Node.js. La mayoría de los tutoriales que he encontrado en internet sobre la implementación de websockets y la inclusión de esta tecnología en proyectos Angular referencian a socket.io como librería a utilizar. El problema que tenemos es que la parte servidor corre en node.js por lo que no podríamos incluirlo en nuestro backend que funciona en Laravel, o tendríamos que levantar un segundo backend que trabaje conjuntamente con Laravel y sea exclusivo para la comunicación con websockets.

Ejemplos de tutorial de implementación de websockets con Angular y socket.io:

<https://www.digitalocean.com/community/tutorials/angular-socket-io>

- Para la inclusión de la tecnología de websockets en proyectos Laravel existen varias alternativas. La mayoría de la documentación que he encontrado referencia a servicios de terceros, que son de pago, como Pusher [20] o Ably [21]. Afortunadamente, también existen algunas alternativas open-source como el paquete **laravel-websockets** (<https://github.com/beyondcode/laravel-websockets>).

En la siguiente página se muestra como incluir websockets en un proyecto Laravel haciendo uso del alternativa gratuita laravel-websockets:

<https://freak.dev/1228-introducing-laravel-websockets-an-easy-to-use-websocket-server-implemented-in-php>

Toda la documentación sobre la inclusión de websockets en un proyecto Laravel, así como las diferentes alternativas de implementación, la he encontrado en la página oficial del producto [22]

- Otra opción sería usar Firebase como componente separado de lo ya implementado cuya única objetivo sería proveer soporte para la comunicación de usuarios en tiempo real. Firebase, entre otras muchas funcionalidades, ofrece una especie de base de datos en tiempo real y la comunicación y envío de mensajes entre partes es bastante transparente para el

programador. Existen multitud de tutoriales sobre como construir una aplicación de chat en tiempo real con Firebase, por ejemplo, este CodeLab en la documentación oficial de la herramienta [23].

17.7 Envío de correos

Cuando se produce un evento de interés para un usuario dentro de la aplicación, el sistema registra una notificación asociada al usuario para informarle, de forma que en el navbar superior de la web se muestra un listado con aquellas notificaciones asociadas al usuario (que marca como leídas cuando pasa el ratón sobre ellas).



Figura 43: Notificaciones dentro de la aplicación

Este sistema es suficiente para reclamar la atención del usuario cuando este está navegando por la aplicación, pero como mecanismo alternativo para hacer volver a entrar al usuario y que este no tenga que estar continuamente pendiente del sistema se ha implementado en envío de correos electrónicos.

Estos correos electrónicos se producen cuando un usuario inicia una solicitud de reserva de un taper a un cocinero o cuando a un comprador se le acepta o rechaza una solicitud de reserva que tenía pendiente y la implementación de esta funcionalidad no ha sido nada compleja en Laravel, siguiendo los pasos indicados en la siguiente web [24].

El primer paso es configurar la cuenta de Gmail de la UOC para permitir el envío de correos mediante aplicaciones externas y tras esta configuración se debe establecer en el fichero .env los datos necesarios para el envío de mails:

```
MAIL_MAILER=smt
MAIL_HOST=smt.gmail.com
MAIL_PORT=587
MAIL_USERNAME=guangle@uoc.edu
MAIL_PASSWORD=. . .
MAIL_ENCRYPTION=tl
MAIL_FROM_ADDRESS=admin@tupperfy.com
MAIL_FROM_NAME="{APP_NAME}"
```

Ilustración 15: Configuración del servidor de correos en el backend

Con estos simples pasos podemos usar la clase Illuminate\Support\Facades\Mail para invocar al método que envía los correos electrónicos:

```
//Enviamos emails al cocinero y comprador
Mail::to($usuario->email)->send(new ReservaRealizadaMail($reserva));
Mail::to($reserva->cocinero->email)->send(new SolicitudReservaMail($reserva));
```

Ilustración 16: Envío de emails en el backend

El método **send** debe recibir como parámetro una clase que extienda de **Illuminate\Mail\Mailable** y en dicha clase, en su método **build**, se configura el contenido del correo electrónico que se debe enviar. En nuestro caso hemos utilizado el motor de plantillas **Blade** incluido en Laravel para componer el contenido de estos correos y, adicionalmente, se ha seguido una guía [25] para personalizar los estilos de estas plantillas de emails e incluir los colores de la paleta de la aplicación

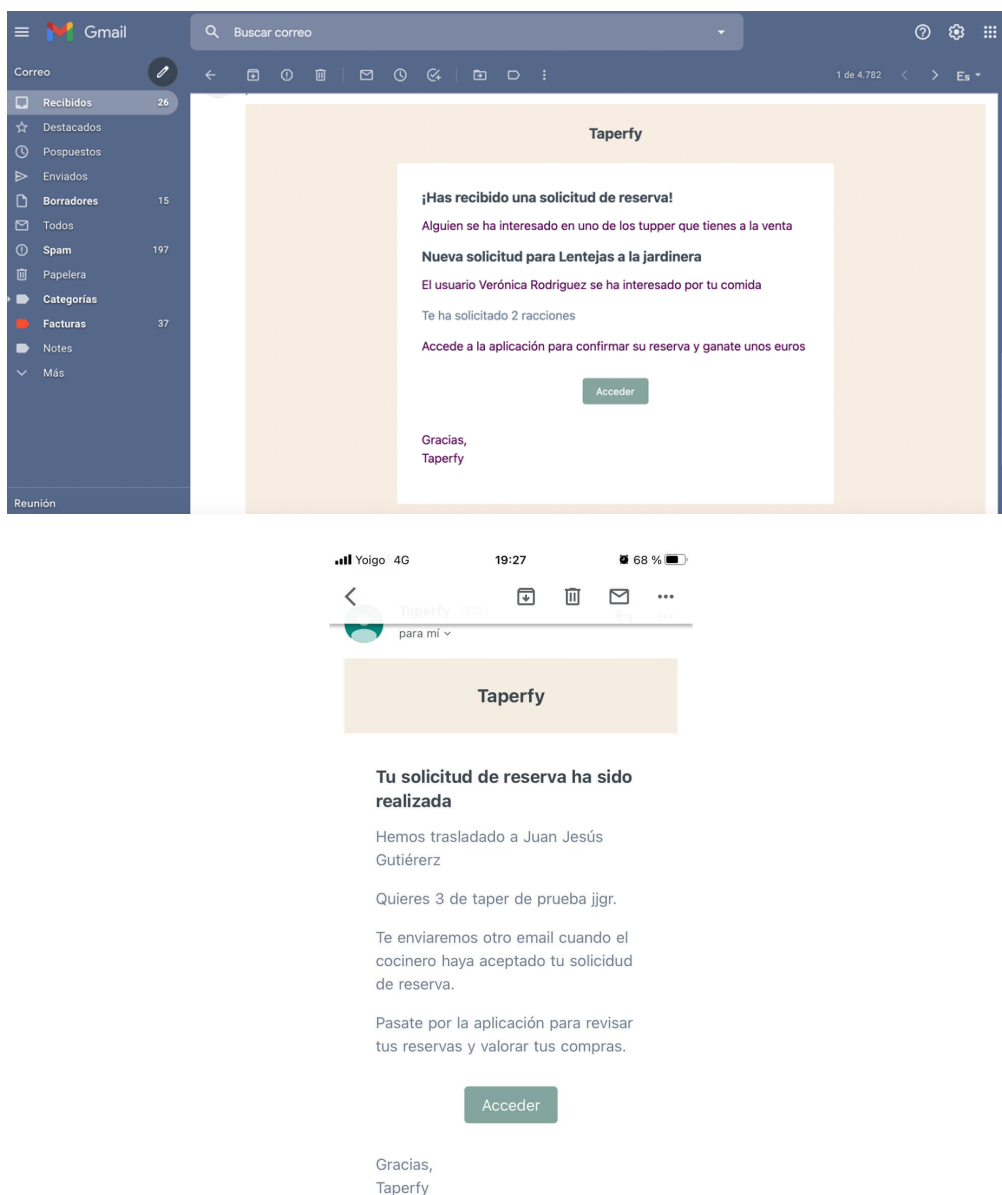


Figura 44: Correos enviados por la plataforma

18. Requisitos de instalación

Para desplegar el producto una vez construido para producción necesitamos un servidor que tenga instalado este software base:

- Servidor Apache, versión 2.4+. En mi caso he desplegado la aplicación para producción en la versión 2.4.46
- Base de datos MySQL
- PHP versión 7.3 o posterior. Este requisito nos viene dado por la versión de Laravel.

Podemos conectar a una base de datos remota y, en principio, Laravel y su ORM Eloquent está preparado para conectar con múltiples sistemas de gestión de base de datos: Mysql, Postgre, Sqlite y Sql server, pero solo se han realizado pruebas con MySQL como motor de base de datos.

Si lo que queremos es desplegar el entorno de desarrollo en un equipo local, necesitamos instalar algunas herramientas adicionales:

- Node & npm para la gestión de dependencias, ejecución de scripts, etc. He tenido actualizado ambos componentes todo el tiempo, siendo actualmente la versión de Node.js la 14.15.5 y la del gestor de paquetes npm 7.6.1
- Angular CLI: como puede verse en la siguiente imagen, el proyecto ha sido desarrollado con la versión 11 de Angular

```
quangle@MacBook-Pro-de-Juan taperfy-back % ng --version
Angular CLI
Angular CLI: 11.2.2
Node: 14.15.5
OS: darwin x64
Angular: undefined
...
Ivy Workspace: <error>

Package                       Version
-----
@angular-devkit/architect     0.1102.2 (cli-only)
@angular-devkit/core          11.2.2 (cli-only)
@angular-devkit/schematics    11.2.2 (cli-only)
@schematics/angular           11.2.2 (cli-only)
@schematics/update            0.1102.2 (cli-only)
```

Figura 45: Versión de Angular

- Git, para descargar el código del repositorio, subir cambios, crear ramas, hacer pull request, etc.
- PHP, en su versión 7.3 o posterior
- Composer, para instalar las dependencias del backend
- Tener una instancia de Mysql y phpmyadmin o similar para la administración de datos.

19. Instrucciones de instalación

Para desplegar en local la aplicación necesitamos obtener el código de ambos proyectos, descargar sus dependencias, configurar la base de datos y arrancar ambos entornos.

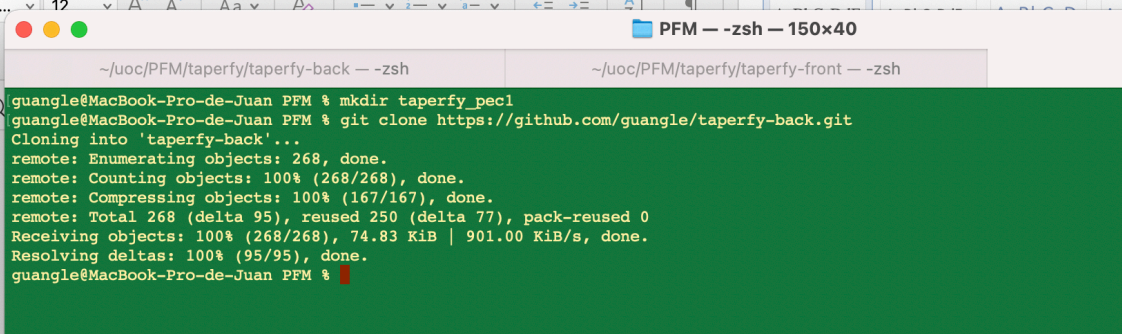
Debemos tener instalado el siguiente software base en nuestro equipo antes de proceder a la instalación del entorno de desarrollo de los proyectos:

- Node / npm
- Angular cli
- Git
- Php (versión 7.3 o posterior)
- Composer
- Tener una instancia de mysql y phpmyadmin o similar para crear un usuario

19.1 Instalación del backend

1) Abrimos una consola, y hacemos un git clone para obtener el código del proyecto:

git clone https://github.com/guangle/taperfy-back.git



```

guangle@MacBook-Pro-de-Juan PFM % mkdir taperfy_pec1
guangle@MacBook-Pro-de-Juan PFM % git clone https://github.com/guangle/taperfy-back.git
Cloning into 'taperfy-back'...
remote: Enumerating objects: 268, done.
remote: Counting objects: 100% (268/268), done.
remote: Compressing objects: 100% (167/167), done.
remote: Total 268 (delta 95), reused 250 (delta 77), pack-reused 0
Receiving objects: 100% (268/268), 74.83 KiB | 901.00 KiB/s, done.
Resolving deltas: 100% (95/95), done.
guangle@MacBook-Pro-de-Juan PFM %

```

Figura 46: git clone del proyecto backend

O si se prefiere, podemos utilizar el código distribuido en la entrega final del proyecto, concretamente la carpeta taperfy-back.

2) Entramos en la carpeta del proyecto (*cd taperfy-back*) e instalamos las dependencias con composer

composer install

```
guangle@MacBook-Pro-de-Juan taperfy-back % composer install
No lock file found. Updating dependencies instead of installing from lock file. Use composer update over compose
le.
Loading composer repositories with package information
Updating dependencies
Lock file operations: 111 installs, 0 updates, 0 removals
- Locking asm89/stack-cors (v2.0.3)
- Locking brick/math (0.9.2)
- Locking dnoegel/php-xdg-base-dir (v0.1.1)
- Locking doctrine/inflector (2.0.3)
- Locking doctrine/instantiator (1.4.0)
- Locking doctrine/lexer (1.2.1)
- Locking dragonmantank/cron-expression (v3.1.0)
- Locking egulias/email-validator (2.1.25)
- Locking facade/flare-client-php (1.4.0)
- Locking facade/ignition (2.6.0)
- Locking facade/ignition-contracts (1.0.2)
- Locking fakerphp/faker (v1.13.0)
- Locking fideloper/proxy (4.4.1)
- Locking filp/whoops (2.11.0)
- Locking fruitcake/laravel-cors (v2.0.3)
- Locking graham-campbell/result-type (v1.0.1)
- Locking guzzlehttp/guzzle (7.3.0)
- Locking guzzlehttp/promises (1.4.1)
- Locking guzzlehttp/psr7 (1.8.1)
- Locking hamcrest/hamcrest-php (v2.0.1)
- Locking laravel/framework (v8.34.0)
- Locking laravel/helpers (v1.4.1)
- Locking laravel/sail (v1.4.9)
- Locking laravel/tinker (v2.6.1)
- Locking lcobucci/jwt (3.3.3)
- Locking league/commonmark (1.5.7)
```

Figura 47: Instalación de dependencias del backend con composer

3) Creación de un esquema de base de datos y un usuario

En el fichero `.env` (que se encuentra en la raíz del proyecto) se encuentra la conexión con la base de datos de la aplicación

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=taperfy
DB_USERNAME=taperfy
DB_PASSWORD=taperfy
```

Figura 48: Configuración de la conexión del backend con la BBDD

Por tanto, debemos crear una instancia de base de datos en el mysql local que se llame **taperfy** y un usuario con credenciales **taperfy / taperfy** para acceder a este esquema. En mi caso utilizo xampp y php myadmin pero podemos hacer este paso con cualquier gestor de base de datos o por consola.

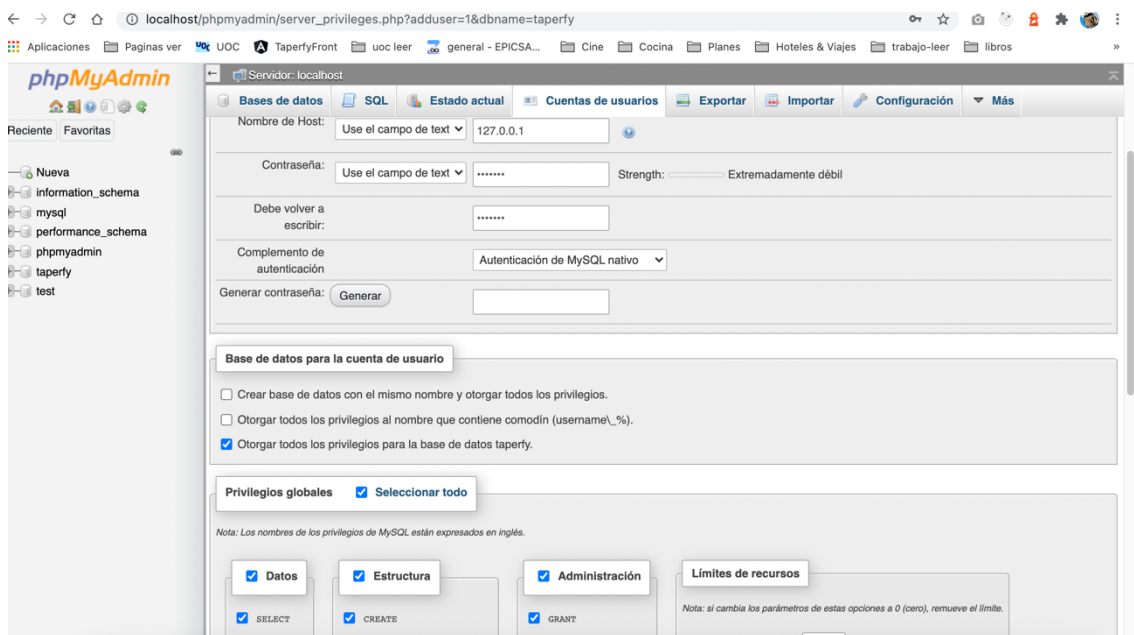
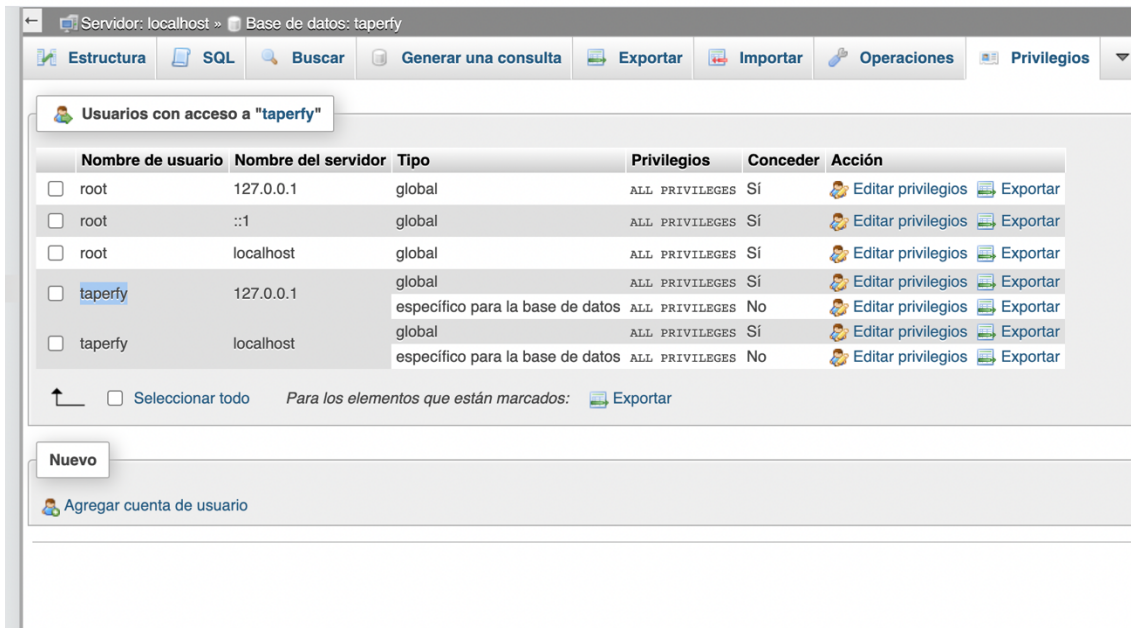


Figura 49: Base de datos de la aplicación, vista en phpmyadmin

4) Creación del modelo de datos y datos de prueba

Ejecutamos los comandos necesarios para tener el modelo de datos definido, así como un conjunto de datos iniciales con los que trabajar.

Para la creación de tablas y relaciones, ejecutamos el siguiente comando en la consola:

php artisan migrate:fresh

```

~/.uoc/PPM/taperfy/taperfy-back -- -zsh
~/.uoc/PPM/taperfy/taperfy-back

guangle@MacBook-Pro-de-Juan taperfy-back % php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (64.77ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (63.17ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (55.29ms)
Migrating: 2021_03_10_085819_create_tipo_taper_table
Migrated: 2021_03_10_085819_create_tipo_taper_table (25.79ms)
Migrating: 2021_03_10_095345_create_estado_taper_table
Migrated: 2021_03_10_095345_create_estado_taper_table (27.31ms)
Migrating: 2021_03_10_106720_create_taper_table
Migrated: 2021_03_10_106720_create_taper_table (201.42ms)
Migrating: 2021_03_10_111008_create_mensaje_taper
Migrated: 2021_03_10_111008_create_mensaje_taper (192.85ms)
Migrating: 2021_03_11_202505_create_notificacion
Migrated: 2021_03_11_202505_create_notificacion (83.87ms)
Migrating: 2021_03_13_115611_create_etiqueta_table
Migrated: 2021_03_13_115611_create_etiqueta_table (28.13ms)
Migrating: 2021_03_13_115621_create_etiqueta_taper_table
Migrated: 2021_03_13_115621_create_etiqueta_taper_table (136.65ms)
Migrating: 2021_03_13_115637_create_reserva_table
Migrated: 2021_03_13_115637_create_reserva_table (221.76ms)
guangle@MacBook-Pro-de-Juan taperfy-back %

```

Figura 50: Ejecución de la migración para crear el modelo de datos

Nota: para ejecutar el comando debemos tener arrancada la base de datos, el esquema y el usuario creado como se indica en el punto anterior.

Generamos los datos de prueba con:

php artisan db:seed

```

~/.uoc/PPM/taperfy/taperfy-back -- -zsh
~/.uoc/PPM/taperfy/taperfy-back

guangle@MacBook-Pro-de-Juan taperfy-back % php artisan db:seed
Seeding: Database\Seeders\TipoTaperSeeder
Ejecutando el seed para la tabla de tipos_taper
-----
Seeded: Database\Seeders\TipoTaperSeeder (10.12ms)
Seeding: Database\Seeders\EstadoTaperSeeder
Ejecutando el seed para la tabla de estados_taper
-----
Seeded: Database\Seeders\EstadoTaperSeeder (2.08ms)
Seeding: Database\Seeders\EtiquetaSeeder
Ejecutando el seed para la tabla de etiqueta
-----
Seeded: Database\Seeders\EtiquetaSeeder (3.41ms)
Seeding: Database\Seeders\UserSeeder
Ejecutando el seed para la tabla de usuarios
-----
Seeded: Database\Seeders\UserSeeder (627.19ms)
Seeding: Database\Seeders\TaperSeeder
Ejecutando el seed para la tabla de tappers
-----
Seeded: Database\Seeders\TaperSeeder (19.04ms)
Seeding: Database\Seeders\ReservaSeeder
Ejecutando el seed para la tabla de reserva
-----
Seeded: Database\Seeders\ReservaSeeder (11.49ms)
Seeding: Database\Seeders\NotificacionSeeder
Ejecutando el seed para la tabla de notificacion
-----
Seeded: Database\Seeders\NotificacionSeeder (35.34ms)
Seeding: Database\Seeders\EtiquetaTaperSeeder
Ejecutando el seed para la tabla de etiquetas_taper
-----
Seeded: Database\Seeders\EtiquetaTaperSeeder (10.22ms)
Seeding: Database\Seeders\MensajeTaperSeeder
Ejecutando el seed para la tabla de mensaje_taper
-----
Seeded: Database\Seeders\MensajeTaperSeeder (19.20ms)
Database seeding completed successfully.
guangle@MacBook-Pro-de-Juan taperfy-back %

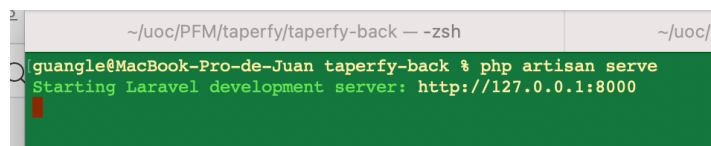
```

Figura 51: db:seed para poblar la base de datos con datos de prueba

5) Arranque del servidor y prueba

Una vez que tenemos el código, descargadas las dependencias, y configurada y poblada nuestra base de datos, arrancamos el servidor local con el siguiente comando:

```
php artisan serve
```



```
~uoc/PFM/taperfy/taperfy-back -- zsh ~uoc/
guangle@MacBook-Pro-de-Juan taperfy-back % php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
```

Si todo ha ido bien, podemos probar que el backend está funcionando correctamente abriendo un navegador y ejecutando una llamada get a la api, concretamente a uno de los métodos que se puede ejecutar sin autorización. Abrimos un navegador y escribimos:

<http://localhost:8000/api/taper>

Deberíamos ver, en formato json, información de los datos de prueba generados

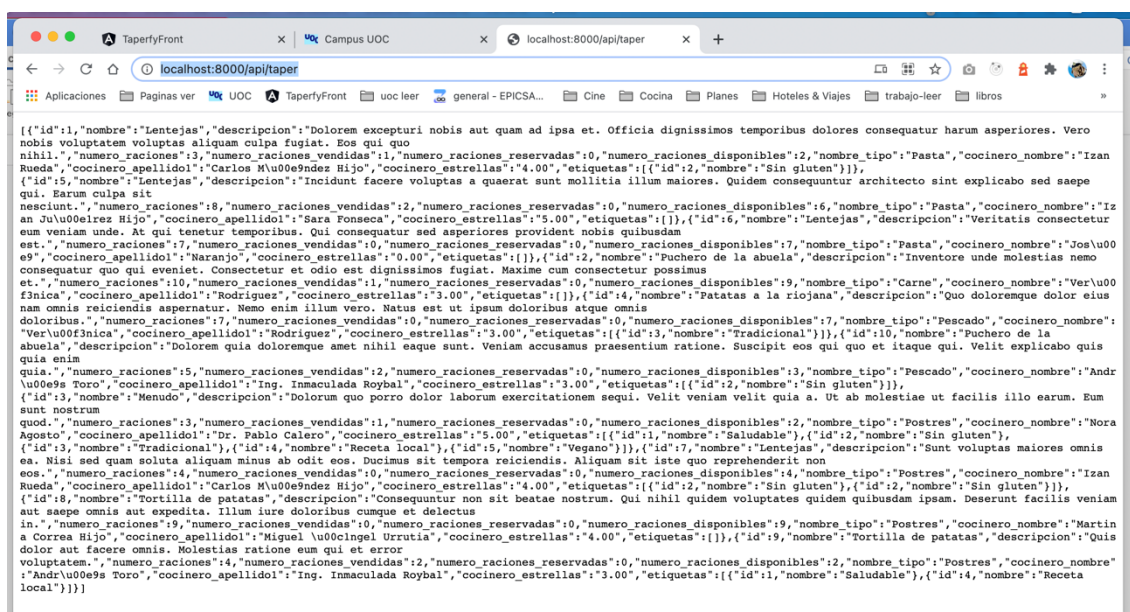


Figura 52: Visualización en el navegador de los resultados devueltos de la API

4) Pruebas

Si todo ha ido bien deberíamos ver el front funcionando en la siguiente dirección:

<http://localhost:4200/>

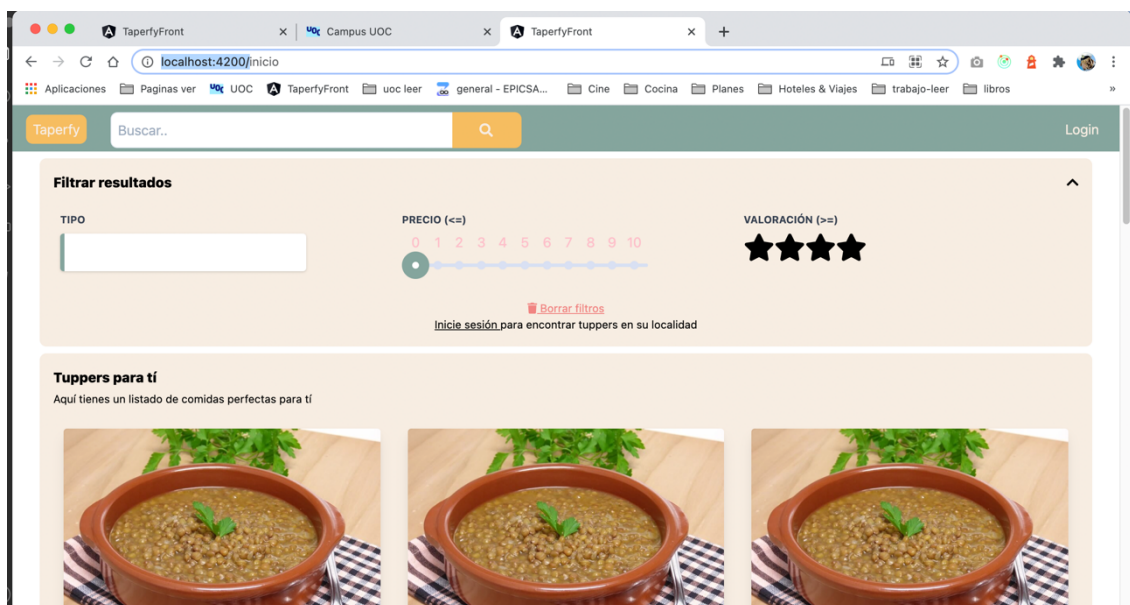


Figura 55: Acceso al front-end desde el equipo local

Sin autenticarnos, podremos visualizar el listado inicial de tuppers y aplicar los filtros en la parte superior. Si queremos utilizar las funcionalidades con usuarios logados podemos usar los siguientes usuarios de prueba:

cocinero@gmail.com / 1111
comensal@gmail.com / 1111

O si se prefiere podemos irnos a la pantalla de registro para crear un nuevo usuario (cocinero o comprador) y empezar a realizar publicaciones de tuppers, reservas, valoraciones, conversaciones y demás.

20. Instrucciones de uso

En el Anexo 5 del presente documento se puede encontrar una guía de usuario, tanto de los aspectos generales de la aplicación como cada uno de los perfiles definidos en la plataforma: cocinero y comprador.

Aunque se han detallado las instrucciones de uso de prácticamente todas las pantallas de la aplicación, se ha perseguido que la interfaz de la misma sea intuitiva y usable, apoyándonos en multitud de patrones de diseño de UX y consiguiendo que el producto tenga un aspecto y comportamiento similar al de otras plataformas e-commerce o de compra-venta, siendo el resultado una aplicación sencilla de usar, al que el usuario se acostumbra rápidamente, un hecho que he tenido la ocasión de comprobar en las diferentes pruebas con testers reales que he hecho durante el desarrollo del producto.

21. Proyección a futuro

Taperfy no deja de ser una aplicación que se ha desarrollado en un ámbito académico, forma parte de un proyecto de fin de máster y aunque me he excedido un poco, aproximadamente he invertido en su desarrollo las 300 horas estipuladas para este tipo de proyectos.

Esto quiere decir que hay algunas funcionalidades de la aplicación que se han quedado fuera por falta de tiempo, o por exceder del ámbito del proyecto, pero que serían deseables incluirlas si esta aplicación se pusiera en producción en un entorno real o se quisiera hacer una empresa a partir de la idea inicial.

A continuación, se detallan aquellas mejoras que podría tener la aplicación en futuras versiones. Algunas de estas nuevas funcionalidades o cambios estuvieron presentes ya en tiempo de análisis del sistema y otras han ido surgiendo durante el proceso de desarrollo.

- Como se ha comentado en el punto 12 de esta memoria, la aplicación en su estado actual contempla dos tipos de usuarios: cocineros y compradores. Se hace patente la necesidad de al menos un **perfil Administrador** que pudiera gestionar las tablas paramétricas, revisar contenido, mediar en disputas y hacer una explotación estadística de todos los datos de la plataforma.

La inclusión de este perfil sería crucial en una empresa real y la implementación implicaría cambios tanto a nivel de modelo de datos como en el desarrollo de muchas pantallas y componentes adicionales.

- En la versión actual de la aplicación, la **localización** del usuario es determinada mediante dos selectores que permiten seleccionar provincia y municipio, por lo que lo más precioso que tenemos para localizar a un cocinero, un comprador o un producto, es un municipio.

Esto puede implicar un problema para usuarios que vivan en municipios grandes, y sería mucho más valioso disponer de un mecanismo para geolocalizar a usuarios y tappers.

Actualmente, los navegadores web y los últimos estándares de HTML y Javascript permiten conocer la localización del usuario que está visitando una web [26], siempre y cuando este usuario conceda permisos para la obtención de esa información.

En nuestro contexto, parece factible que el usuario cocinero y comprador quieran compartir la localización dentro de la plataforma porque permitiría aplicar búsquedas mucho más precisas (tappers a menos de 1 kilómetro, por ejemplo) o mostrar los datos en un mapa de forma mucho más atractiva para el usuario.

- Aunque la aplicación es una PWA que se puede ‘instalar’ en los móviles de nuestros usuarios compradores, es factible que algunas personas prefieran usar una **aplicación móvil** para comprar y/o publicar comida en nuestra plataforma. La creación de dicha app no dejaría de ser otro cliente que consumiría exactamente los mismos servicios que tenemos implementado en el backend y además utilizaría el mismo protocolo de comunicación y mecanismo de seguridad implementados.

- En Taperfy, cada usuario tiene la posibilidad de subir su propia foto de perfil, algo que siempre añade confianza en la interacción entre usuarios. Por supuesto, cada tupper tiene asociada una foto, que debe subir el cocinero para atraer a posibles compradores.

Todas estas imágenes son subidas al servidor mediante un método de la api destinado a tal efecto, y los ficheros se guardan dentro de una carpeta en el sistema de ficheros en el servidor de aplicaciones. A medida que se van consumiendo estas imágenes por distintos usuarios, el servidor de aplicaciones sirve cada uno de los ficheros solicitados.

Este modo de funcionamiento es aceptable y funciona correctamente con un número restringido de usuarios, pero pensando en la escalabilidad, en el momento que tengamos, por ejemplo, 20.000 usuarios cocineros publicando tupper (y fotos) cada día, no parece que sea la opción más acertada guardar toda esta cantidad de datos directamente en el servidor y cobra sentido la inclusión de un proveedor especializado en este tipo de contenido, como Amazon S3, Google Cloud Storage o incluso Firebase.

Aún sin haberlo abordado, intuyo que el cambio no sería demasiado costoso porque he encontrado en internet multitud de tutoriales sobre como conectar Laravel con un proveedor de almacenamiento cloud. Por ejemplo en el siguiente enlace [27] se explica como almacenar los ficheros subidos a un backend Laravel en el servicio de almacenamiento Amazon S3

- Si la aplicación tuviera algún ánimo de lucro, que no es el caso, sería interesante añadir la posibilidad de realizar los pagos directamente en la aplicación, ya que actualmente la transacción económica se produce fuera de la web.

Habría que implementar un proceso en el que el comprador realice el pago y este llegue al cocinero cuando ambos confirmen que el intercambio del taper se ha realizado satisfactoriamente, un mecanismo análogo al que implementan plataformas como Blablacar o Amovens, por citar dos ejemplos.

En este sentido, stripe.com sería la plataforma de pagos a investigar, pero este análisis, la integración con la paralela de pagos y implementación de esta funcionalidad excede con mucho el alcance de este proyecto.

- Por último, y aunque hay un punto en la memoria de la aplicación exclusivo para el chat analizar implementado, sus problemas y alternativas, sería crucial la implementación de **websockets** para implementar la funcionalidad de intercambio de mensajes en tiempo real. La inclusión de esta tecnología excede el ámbito del proyecto si tenemos en cuenta el resto del código implementado pero conseguiríamos que la aplicación fuera escalable y el rendimiento no se viera excesivamente afectado conforme vayan creciendo los usuarios chateando en el sistema.

En la medida de lo posible, se han aplicado en el desarrollo principios de arquitectura como la inclusión del patrón de diseño MVC en ambos proyectos, el patrón de diseño REDUX en Angular, la modularización de los componentes para conseguir un bajo acoplamiento, etc.

La inclusión de todas estas técnicas de ingeniería del software persiguen una mejor calidad del desarrollo, aumentar la mantenibilidad y sobre facilitar la ampliación del sistema por lo que a nivel

de arquitectura de las aplicaciones se cuenta con una buena base para desarrollar las nuevas funcionalidades anteriormente citadas.

22. Presupuesto

Taperfy es un proyecto académico y en ese sentido no se va a hacer un estudio especialmente profundo acerca de los costes relacionados con la creación y el mantenimiento del proyecto ya que no es un proyecto con ánimo de lucro y los costes no son reales ya que todo el proyecto ha sido desarrollado por la misma persona, sin una remuneración de tipo económico. Aún así, se hará una pequeña estimación sobre cual sería el presupuesto de este producto si ‘lo vendiéramos’ o fuera un encargo de una empresa o persona que estuviera dispuesto a pagar por él.

Cómo se ha indicado en la parte de planificación del presente documento, la creación del producto se ha dividido básicamente en 3 grandes fases y hay tres grandes tipologías de trabajo que se han realizado durante este periodo: análisis del producto, implementación y pruebas. Tal y como se consensuó con el tutor, del proyecto el esfuerzo estimado en horas para un proyecto de fin de máster son aproximadamente 300, contando el tiempo de redacción de la memoria y documentación adicional, correos, trabajos auxiliares, etc.

Durante el proceso de creación del proyecto se ha excedido ligeramente estas 300 horas propuestas y sería más aproximado decir que solo el análisis, implementación y pruebas del producto ha ocupado esta cantidad de tiempo. Por tanto, una buena aproximación del coste de la creación del sistema podría ser 300 horas hombre.

Al ser un proyecto personal y académico, todas las tareas relacionadas las ha desarrollado una sola persona. En un entorno empresarial sería más oportuno y óptimo contar con perfiles especializados para cada uno de los tipos de trabajo. Por tanto, en la siguiente tabla se muestra el número de horas estimadas de cada uno de los trabajos, qué perfil realizaría la tarea y cual sería el coste, de nuevo, haciendo una estimación del coste/hora de cada uno de los perfiles.

Trabajo	Nº aproximadas	Horas	Coste / Hora	Coste total
Análisis	80		45 €	3600 €
Implementación	200		35 €	7000 €
Pruebas	20		25 €	500 €
Total	300			11100 €

Tabla 6: Tabla de costes de la construcción del proyecto

Por tanto, con la estimación anteriormente mencionada tendríamos un coste de 11100 euros para la construcción del producto. A este coste habría que sumar otros costes derivados del desarrollo e implementación del sistema: el equipo de desarrollo, así como los costes de puesta en producción (hosting y dominio).

Respecto al equipo de desarrollo, durante estos meses he utilizado un portátil Apple Macbook Pro, con un valor de unos 2000 euros aproximadamente, aunque he desplegado y compilado el proyecto en otros equipos de inferior coste y características, teniendo también un buen rendimiento.

Analizando alguna de las alternativas de hosting y dominio, por ejemplo, *digitalocean*, podemos encontrar planes de alojamiento que podrían ser válidos para nuestra aplicación desde 10 euros al mes, por lo que podríamos deberíamos sumar 120 euros al coste para alojar la aplicación el primer año. En este sentido, sería interesante contratar algún servicio de Paas, como Google Cloud, Amazon AWS o Windows Azure para escalar automáticamente (y pagar por ello) en base a la demanda de la aplicación. Este tipo de servicios son ‘elásticos’ en el sentido de que permiten redimensionar de forma dinámica los recursos conforme nuestra aplicación (y su demanda de cpu, disco y memoria) vaya creciendo.

Por último, un dominio (*taperfy.es* y *taperfy.com* están libres) nos costaría aproximadamente 10 euros al año.

Por tanto, con los importes citados anteriormente tendríamos el siguiente coste para la creación y puesta en producción del Taperfy:

	Coste
Construcción	11100
Equipo de desarrollo	2000
Hosting (1 año)	120
Dominio (1 año)	10
Total	13230 €

Tabla 7: Coste total de Taperfy

En cualquier caso, este coste es una estimación y un análisis profundo de todos los costes del proyecto y estudio de la viabilidad excede del ámbito y la temática del máster, que tiene un carácter totalmente técnico.

Sería particularmente interesante un análisis sobre las vías para rentabilizar el proyecto, que probablemente pasaría por la inclusión de una pasarela de pagos en la aplicación, posibilidad de publicaciones promocionadas y usuarios Premium.

23. Conclusiones

A nivel general, estoy muy satisfecho con el trabajo desarrollado durante estos meses, ya que me ha permitido poner en práctica muchos de los conocimientos adquiridos en las asignaturas del máster y considero que el resultado final es un producto de buena calidad, sólido técnicamente y robusto.

Desde la fase de análisis de la aplicación, se ha tenido como premisa utilizar, en la medida de lo posible, el mayor número de técnicas, herramientas y frameworks aprendidos durante estos años de máster y trabajar durante tantos meses en el mismo proyecto y con todas estas herramientas a la vez me ha dado una visión con conjunto de la que carecía hasta ahora. Adicionalmente, se han profundizado en algunas de las materias cursadas (sobre todo en algunas partes de Angular, ngrx, operaciones con observables, creación de pruebas unitarias y maquetación) y se han explorado otras nuevas como pueden ser la autenticación de apis con JWT, el envío de mails en Laravel o las animaciones.

El hecho de desarrollar un proyecto completo difiere un poco del trabajo con PECs. Un trabajo de un mayor volumen, como es el caso, me ha dado el tiempo necesario para afianzar conceptos, ser plenamente consciente de lo que estaba haciendo en cada momento y, por último, adquirir cierta soltura y naturalidad con el manejo de componentes, servicios de negocio, guardias, etc. Por tanto, considero totalmente alcanzado uno de los objetivos principales de este trabajo de fin de máster: consolidar lo aprendido y adquirir experiencia y soltura en un proyecto real. En definitiva, con el paso de estos meses me siento mucho más ágil, habilidoso y consciente en este mundo de las tecnologías web.

A grandes rasgos, se ha cumplido la planificación inicialmente estipulada en los primeros compases del proyecto. Comenté con el tutor que mis obligaciones laborales y familiares quizás me impidieran tener el ritmo constante que sería deseable, pero en cualquier caso se han cumplido todos los hitos propuestos en el tiempo y las entregas previstas, por lo que no hemos tenido ningún desvío significativo en este sentido.

Por último, diría que estoy satisfecho con producto final, la documentación asociada, así como con la vehemente defensa que realizado del proyecto en las presentaciones y videos anexos. He disfrutado mucho durante el proceso y considero que, de alguna forma, eso se filtra en los diferentes entregables que he ido produciendo. Taperfy es un proyecto que siempre recordaré y del que estoy particularmente orgulloso, tanto a nivel puramente técnico como la idea y valores que defiende.

Anexo 1. Entregables del proyecto

Los documentos, software y material complementario que componen la entrega final del proyecto son los siguientes:

- Documentación:
 - PAC_FINAL_mem_GutierrezRamos_JuanJesus → memoria del proyecto (este documento) que describe todas las fases del proceso: concepción, análisis y desarrollo del proyecto. El documento tiene información, a nivel funcional, a nivel técnico (arquitectura, consideraciones técnicas, ejemplos de código...), así como a nivel de usuario, con guías, instrucciones de instalación, etc.
 - PAC_FINAL_prs_GutierrezRamos_JuanJesus → presentación escrita-visual, destinada a un público general, donde se expone la idea de la aplicación, motivaciones y orientada a la difusión del proyecto.
 - PAC_FINAL_informe_GutierrezRamos_JuanJesus → informe de autoevaluación, con algunas consideraciones y reflexiones sobre el trabajo final entregado, así como una auto calificación en distintos ámbitos del esfuerzo realizado durante estos meses.

- Código del proyecto:
 - PAC_FINAL_prj_GutierrezRamos_JuanJesus → código fuente del proyecto. El backend de la aplicación se encuentra bajo el directorio /taperfy-back y el front-end angular en el directori /taperfy-front. Las instrucciones de instalación del proyecto pueden encontrarse en el punto 19 del presente documento. En nuestro caso, como hemos utilizado los mecanismos de Larvael para la generación de la base de datos, modelos y datos de prueba, no necesitamos ningún script sql para la creación y configuración de base de datos.

- Multimedia:
 - PAC_FINAL_vid_GutierrezRamos_JuanJesus → video de presentación del proyecto, donde se cuenta en qué consiste la aplicación funcionalmente y se hace una demostración de la versión final del producto. Adicionalmente, el video contiene algunos detalles sobre el proceso de desarrollo y algunos apuntes sobre la arquitectura de la aplicación implementada. El video se ha subido a la plataforma Present@ y adicionalmente está disponible en el siguiente enlace:

<https://www.loom.com/share/597d251549a341008c786cb934140c91>

Anexo 2. Código fuente (extractos)

En este apartado se muestran algunos fragmentos del código fuente de la aplicación que pretenden ser una muestra significativa de los elementos que se han implementado. He tratado de incluir una muestra representativa y variada de las partes más importantes del código. Adicionalmente, como adjunto a esta memoria se incluye todo el código fuente, así como los enlaces a los repositorios donde se encuentra alojado el mismo.

El código fuente generado durante el periodo de implementación está ampliamente comentado con el objetivo de hacer más fácil la inclusión al proyecto a nuevos programadores, facilitar la corrección del programa y en muchos casos también han sido comentarios que me han ayudado en la digestión y comprensión de conceptos aprendidos para abordar la creación de Taperfy.

A2.1 Backend

Aunque se hayan utilizado diferentes controllers para una mejor modularización y desacoplamiento de la api, lo métodos ofrecidos por nuestro backend pueden consultarse en el fichero api.php, donde se describen los métodos de la api, a qué url responde y cual es la clase que las implementa:

```

.
.
.
//Métodos relacionados con el usuario
Route::put('/usuario', 'App\Http\Controllers\UsuarioController@actualizaUsuario')
->middleware(\Spatie\HttpLogger\Middlewares\HttpLogger::class);
Route::post('/usuario/uploadImagen', 'App\Http\Controllers\UsuarioController@uploadImagen')
->middleware(\Spatie\HttpLogger\Middlewares\HttpLogger::class);
Route::get('/usuario/cocinerosFavoritos', 'App\Http\Controllers\UsuarioController@cocinerosFavoritos');
Route::post('/usuario/nuevoCocineroFavorito', 'App\Http\Controllers\UsuarioController@nuevoCocineroFavorito');
Route::post('/usuario/borrarCocineroFavorito', 'App\Http\Controllers\UsuarioController@borrarCocineroFavorito');
Route::get('/usuario/cocinero/estadisticasGlobales',
'App\Http\Controllers\UsuarioController@estadisticasGlobalesCocinero');
Route::get('/usuario/cocinero/seguidores', 'App\Http\Controllers\UsuarioController@seguidoresCocinero');
Route::get('/usuario/detalle/{id}', 'App\Http\Controllers\UsuarioController@obtenerUsuarioById');

//Métodos relacionados con la recuperación de la contraseña
Route::post('enviarPassReset', 'App\Http\Controllers\ResetPasswordController@enviarPassResetEmail')
->middleware(\Spatie\HttpLogger\Middlewares\HttpLogger::class);
Route::post('cambiarPassword', 'App\Http\Controllers\ChangePasswordController@cambiarPassword');

//Métodos relacionados con los tappers
Route::get('taper', 'App\Http\Controllers\TaperController@todos');
Route::get('taper/{id}', 'App\Http\Controllers\TaperController@obtenerTaperById');
Route::post('taper', 'App\Http\Controllers\TaperController@nuevoTaper');
//etc
.
.
//sigue el resto de métodos..

```

Ilustración 17: Definición de la api rest del backend

Como curiosidad, hemos utilizado la librería Spatie\HttpLogger como middleware en algunas peticiones ya que este componente nos permite imprimir, en el log de la aplicación, la petición http que se recibe antes de ser procesada, una utilidad que nos ayuda bastante cuando estamos depurando las llamadas a la api del backend.

Junto a la definición de la api, el desarrollo en el proyecto Laravel ha consistido esencialmente en la implementación de controladores y las clases de utilidades necesarias en la capa de acceso a datos (factories, seeders...). Respecto a los controladores, todos reciben un objeto request como parámetro y de forma general hacen una o varias operaciones contra la base de datos antes de devolver un objeto json de respuesta. A modo de ejemplo, mostramos el controlador encargado de implementar los métodos de la api relacionados con la reserva de tappers:

```
namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;
use App\Http\Controllers\Controller;
//resto de imports..

/** Métodos de la api relacionados con la gestión de reservas en la aplicacion */
class ReservaController extends Controller {
    public function __construct() {
        //Indicamos en el constructor que las peticiones deben estar autenticadas
        $this->middleware('auth:api', ['except' => []]);
    }

    /** Nueva solicitud de reserva de un comprador a un cocinero */
    public function nuevaReserva( Request $request ) {
        Log::info('Creación de una reserva');

        $usuario = auth()->user();
        $reserva = new Reserva;

        $taper = Taper::find( $request->taper['id'] );
        $cocinero = User::find( $request->cocinero['id'] );

        $reserva->usuario()->associate( User::find($usuario->id) );
        $reserva->taper()->associate( $taper );
        $reserva->cocinero()->associate( $cocinero);

        $reserva->fecha = Carbon::now();
        $reserva->cantidad = $request->cantidad;
        $reserva->estado = $request->estado;
        $reserva->estrellas = 0;

        $reserva->save();
        Log::info('Reserva almacenada en la base de datos');

        //Actualizamos el número de raciones disponibles del taper
        $taper->numero_raciones_reservadas = $taper->numero_raciones_reservadas + $request->cantidad;
        $taper->numero_raciones_disponibles = $taper->numero_raciones_disponibles - $request->cantidad;
        $taper->save();
        Log::info('Actualizado el taper asociado a la reserva');

        //Creamos una nueva notificacion para el cocinero
        $notificacion_cocinero = new Notificacion();
        $notificacion_cocinero->fecha = Carbon::now();
        $notificacion_cocinero->leida = false;
        $notificacion_cocinero->tipo = 'solicitudes';
    }
}
```

```

$notificacion_cocinero->mensaje = "Tienes una nueva solicitud de reserva para " . $taper->nombre;
$notificacion_cocinero->usuario()->associate( $cocinero );
$notificacion_cocinero->save();

//Enviamos emails al cocinero y comprador
Mail::to($usuario->email)->send(new ReservaRealizadaMail($reserva));
Mail::to($reserva->cocinero->email)->send(new SolicitudReservaMail($reserva));

}

/** Devuelve la reserva cuyo identificador se pasa como parametro */
public function obtenerReservaById($id) {
    Log::info('Obteniendo la reserva con identificador: ' . $id);
    $reserva = Reserva::find($id);
    return new ReservaResource($reserva);
}

/**
 * Devuelve las reservas pendientes del usuario (comprador) que invoca el método
 */
public function reservasPendientes( Request $request ) {
    Log::info('Usuario comprador: reservas pendientes');

    $usuario = auth()->user();
    $consulta_reservas = Reserva::where('usuario_id', '=', $usuario->id)
->where('estado', '=', 0)
->orWhere(function($query)
    {
        $query->where('estado', '=', 1)
->where('estrellas', '=', 0);
    });
    $consulta_reservas->orderBy('id', 'DESC');
    $reservas_pendientes = $consulta_reservas->get();
    return ReservaResource::collection($reservas_pendientes);
}

/** Acepta la reserva cuyo id se pasa como parámetro en la request */
public function aceptarReserva( Request $request ) {
    Log::info("Llega una aceptación de la reserva");
    $usuario = auth()->user();
    $reserva = Reserva::find($request->id);
    $reserva->estado = 1; //Aceptada
    $reserva->save();
    $cantidad = $reserva->cantidad;
    $taper = $reserva->taper;
    $taper->numero_raciones_vendidas += $cantidad;
    if( $taper->numero_raciones_reservadas > 0 ) {
        $taper->numero_raciones_reservadas -= $cantidad;
    }
    //nos ha producido algún error con los datos de prueba (faker)
    //asi que nos aseguramos que num_raciones_reservadas
    //nunca baje de cero
    if( $taper->numero_raciones_reservadas < 0 ) {
        $taper->numero_raciones_reservadas = 0;
    }
    $taper->save();
    Log::info('Reserva aceptada');
    //Enviamos un mail al comprador
    Mail::to($reserva->usuario->email)->send(new ReservaConfirmadaMail($reserva));
}

```

```

//Creamos una nueva notificacion para el comprador
$notificacion_cocinero = new Notificacion();
$notificacion_cocinero->fecha = Carbon::now();
$notificacion_cocinero->leida = false;
$notificacion_cocinero->tipo = 'mis_pedidos';
$notificacion_cocinero->mensaje = 'Tu reserva para ' . $taper->nombre . ' ha sido aceptada';
$notificacion_cocinero->usuario()->associate( $reserva->usuario );
$notificacion_cocinero->save();

return $reserva;
}

/** El comprador asigna una puntuación a reserva */
public function valorarReserva( Request $request ) {
    /** ...
}

//FALTAN MÉTODOS, OMITIDOS PARA UNA MEJOR LEGIBILIDAD DEL DOCUMENTO
//(TODO EL CÓDIGO ESTA DISPONIBLE EN EL ANEXO A ESTA MEMORIA)
}

```

Ilustración 18: Controlador que gestiona las reservas

En el ejemplo anterior se han omitido algunos métodos para no extender innecesariamente la memoria. Por último, se incluyen algunos fragmentos de código backend encargado de la gestión del modelo de datos y acceso a los mismos. Se han utilizado los mecanismos que proporciona el framework para la implementación de la capa de acceso a datos y generación de la estructura de la BBDD y datos de prueba:

```

//Migraciones:
class CreateTaperTable extends Migration{
    //Crea la tabla taper cuando se ejecuta una migración..
    public function up()
    {
        Schema::create('taper', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string('nombre')->nullable(false);
            $table->string('descripcion')->nullable(false);
            $table->unsignedBigInteger('cocinero_id')->nullable(false);
            $table->unsignedBigInteger('tipo_id')->nullable(false);
            //sigue..
        });
    }
}

//Factory:
class TaperFactory extends Factory {
    protected $model = Taper::class;

    /**
     * Creación de objetos de prueba de tipo taper,
     * util para poblar la base de datos en el entorno
     * de desarrollo
     */
    public function definition() {
        //Usamos la librería faker para los datos de prueba..
        return [
            'nombre' => $this->faker->text(),
            'descripcion' => $this->faker->text(),
        ];
    }
}

```

```

        'fecha_cocinado' => $this->faker->dateTimeBetween('-3 days','now','Europe/Madrid'),
        //sigue..

//Modelo:
class Taper extends Model {
    use HasFactory;

    //Nombre de la tabla que mapea
    protected $table = 'taper';

    //activamos created_at, updated_at
    public $timestamps = true;

    //Métodos de acceso a las relaciones one to many o many to many
    public function cocinero(){
        return $this->belongsTo(User::class,'cocinero_id');
    }
    //sigue..

//Seeder:
class TaperSeeder extends Seeder {
    /** Pueba la base de datos de tappers con datos de prueba */
    public function run(){
        //Creaos 10 tappers de prueba
        Taper::factory()->times(10)->create();
    }
}

```

Ilustración 19: Models, factories, seeders y migrations

A2.1 Frontend

La parte de código del front es más extensa y, al igual que hemos hecho con el backend, solo se expondrán los puntos más significativos por no extender demasiado la memoria.

Angular es un framework front-end orientado a la creación de componentes, por tanto, la implementación de la aplicación ha consistido en ir construyendo todos los componentes necesarios (en taperfy, finalmente más de 30) así como los elementos asociados para dotar al componente de comportamiento, estado e interfaz (guardas, servicios de negocio, validadores, etc.).

Por citar un ejemplo, las tarjetas con información de tappers que componen el listado principal son componentes:

```

//imports..

@Component({
  selector: 'app-taper-card',
  templateUrl: './taper-card.component.html',
  styleUrls: ['./taper-card.component.scss'],
  animations: [
    Animaciones.fadeAnimation_configurable
  ]
})
/** Tarjeta para previsualizar un taper, mostrado en el listado
 * principal de la aplicación

```

```

...
*/
export class TaperCardComponent implements OnInit {

  //Observamos los cocineros favoritos del usuario para poder determinar
  //si el cocinero de la tarjeta pertenece a ellos
  public cocinerosFavoritos$: Observable<any> = this.store$.select(
    UserSelectors.cocinerosFavoritosSelector
  );

  //Taper que será representado en la tarjeta. Es un parámetro de entrada
  //del componente.
  @Input() taper: Taper;
  //indica cuanto esperaremos para ver el elemento
  @Input() aparicion:number = 0;

  cocinado_por_favorito : boolean = false;

  public hoy = new Date();
  public url_base_img = environment.BACKEND_IMG_URL;

  constructor(
    private store$: Store<AppStore>,
    private router: Router
  ) {}

  ngOnInit(): void {
    this.cocinerosFavoritos$.subscribe((cocineros)=>{
      this.cocinado_por_favorito = cocineros.some(c => c.id == this.taper.cocinero.id);
    });
  }

  detalleTaper(id) {
    console.log('Se va a acceder al detalle de un taper ' + id);
    this.store$.dispatch(new TaperActions.Detalle( id ));
  }
}

```

Ilustración 20: Componente angular app-taper-card

En el ejemplo anterior se puede ver que el componente solo inyecta en el constructor el estado de la aplicación, la lógica implementada consiste en ‘despachar’ acciones y el componente ‘está atento’ a los cambios en el estado de la aplicación haciendo uso de selectores. Este es un buen ejemplo para ver que con el patrón de diseño REDUX (ngrx) los diferentes elementos están muy desacoplados y la creación de componentes se simplifica una vez tienes las acciones y selectores definidos.

En todos los diferentes formularios de la aplicación se ha utilizado la librería de angular *FormBuilder* para la construcción de formularios reactivos. Este tipo de formularios permite la validación de los datos en tiempo real, construcción de formularios en el componente, aplicar validaciones personalizadas y la obtención inmediata del valor de los diferentes campos submitios en formato json.

```

//inicializa el formulario reactivo
crearFormulario() {
  this.perfilForm = this.fb.group({
    nombre : [ this.usuario.nombre, [Validators.required, Validators.minLength(3)] ],
    email : [ this.usuario.email, [Validators.required, Validators.email] ],
    apellido1 : [ this.usuario.apellido1, [Validators.required, Validators.minLength(3)] ],
    apellido2 : [ this.usuario.apellido2, [Validators.required, Validators.minLength(3)] ],

    provincia : [ null, [Validators.required] ],
    municipio : [ null, [Validators.required] ],

    es_cocinero: [this.usuario.es_cocinero?1:0, []],
    sobre_mi : [ this.usuario.sobre_mi, [ Validators.required, Validators.minLength(10) ] ]
  });
}

```

Ilustración 21: Ejemplo de construcción de un formulario reactivo

La aplicación utiliza el patrón redux para utilizar el estado de la aplicación como única fuente de datos a la que estarán suscritos multitud de componentes. Una descripción detallada de como funciona el patrón de diseño puede encontrarse en el punto 17.1 del presente documento. A continuación, algunos fragmentos de código relacionados con la ejecución de tareas, uso de selectores de estado, reducir y efectos:

```

//Actions
//Definición de acciones:
export class LoginUser implements Action {
  readonly type = AuthActionTypes.LOGIN_USER;
  constructor(public email: string, public password: string) {
  }
}

//Llamada a las acciones:
detalleTaper(id) {
  console.log('Se va a acceder al detalle de un taper ' + id);
  this.store$.dispatch(new TaperActions.Detalle( id ));
}

//Reducers:
//authReducer = Única función donde se cambia el estado relacionado con la
//información de autenticación
export function authReducer(
  state: AuthState = authInitialState(),
  action: AuthActions
): AuthState {
  switch (action.type) {
    //Ante un login correcto, actualizamos los datos del estado relacionados
    //con la autenticación
    case AuthActionTypes.LOGIN_USER_SUCCESS:
      return {
        ...state,
        logged: true,
        last_login: new Date(),
        nombre : action.token.user.email ,
        token : action.token.access_token,
        mensaje : null
      };
    //Sigue..

```



```

//Selectores de estado (en componentes):
public usuarioLogado$: Observable<User> = this.store$.select(
  UserSelectors.usuarioSelector
);
//Los componentes están suscrito a cambios de estado
...
this.authInfo$.subscribe((login_state) => {
  this.loggedIn = login_state.loggedIn;
  if(this.loggedIn) {
    //Cargamos las notificaciones del usuari (si estamos logados)
    this.store$.dispatch(new UserActions.CargarNotificacionesSinLeer());
    this.store$.dispatch( new UserActions.CalcularNumMensajesNuevos() );
  }
});

//Efectos:
...
//Cuando se lanza una acción de aceptación de reserva, invocamos
//al backend para actualizar la reserva y el tupper
@Effect()
aceptarReserva$: Observable<any> = this.actions$.pipe(
  ofType(TaperActions.TaperActionTypes.ACEPTAR_RESERVA),
  switchMap(( action : TaperActions.AceptarReserva ) =>
    this.backendService.aceptarReserva( action.reserva ).pipe(
      switchMap((reserva) => [
        new TaperActions.AceptarReservaSuccess(reserva),
        new TaperActions.CargarSolicitudesPendientes(),
        new TaperActions.CargarSolicitudesTerminadas(),
      ]),
      catchError((error) => of(new TaperActions.SolicitarReservaError(error)))
    )
  )
);
//.. sigue

```

Ilustración 22: Fragmentos de código REDUX / ngx

Otro de los puntos de la implementación que más esfuerzo me ha supuesto, en cuanto al número de horas que me ha costado, es que la web sea responsive, esto es, que se adapte a cualquier tamaño de dispositivo. Para conseguir este objetivo nos hemos apoyado en la librerías css tailwind, que nos ofrece una serie de mecanismos para aplicar una clase u otra en base a una serie de anchos de pantalla pre-establecidos. A continuación, se muestra un ejemplo tanto de scss como de clases aplicadas en el html donde se utilizan los prefijos definidos por el framework para que las clases únicamente tengan efecto en los tamaños s, m, lg...

```

//Ejemplo de como se aplican prefijos Tailwind:

//mx = margen horizontal
//mx-1 (1 rem) por defecto
//aumentamos a 2em (mx-2) para pantallas 'm'
//mx-3 para pantallas de tamaño lg
//mx-4 (4rem) para pantallas 'xl' en adelante
.contenedor-principal {
  @apply mx-1 md:mx-2 lg:mx-3 xl:mx-4
  pt-4 px-4 pb-20 mt-2 md:mb-2 lg:mb-2
  bg-paleta_tf_beige rounded-lg
  min-h-full;
}

```

```

}

<!-- Ejemplo Tailwind en el html -->
<!--
  El div ocupará un ancho u otro (w-..) en base
  al tamaño de la pantalla ç
-->
<div class="w-full md:w-2/12 mt-2 pl-4 md:pr-4 lg:pr-4 lg:pr-4" >
  <i class="fas fa-star fa-2x text-paleta_tf_amarillo"></i>
  {{ (taper$ | async)?.cocinero_estrellas }}
</div>

```

Ilustración 23: Prefijos tailwind

Otro aspecto fundamental para tener una visión general del código de la aplicación es la comunicación entre front y back. Para invocar a los métodos rest ofrecidos por nuestro proyecto Laravel, en el front de taperfy se ha implementado un servicio de negocio que hace uso de la librería *HttpClient* de `@angular/common/http`. Esta librería nos permite hacer todo tipo de llamadas http (post, get, put...), devuelve observables tipados a una de las clases de nuestro modelo y nos permite incluir en la cabecera de las peticiones el token JWT. Este backendService que hemos implementado para realizar las llamadas es inyectado en diferentes clases de nuestra aplicación y generalmente se usa en los efectos asociados a las acciones ngrx. La clase es muy extensa por lo que a continuación se muestran un par de ejemplos de invocación al backend:

```

import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http'
import { Observable, of, throwError } from "rxjs";
import { tap, catchError, map } from "rxjs/operators";
..

export class BackendService {

  //inyectamos el objeto HttpClient que será el encargado de hacer las
  //peticiones http
  constructor( private http:HttpClient ,
  private store$: Store<AppState> ) {

    this.authInfo$.subscribe((login_state) => {
      //Estamos atentos a si cambia algo en el estado del login, para
      //actualizar el token incluido en la cabecera de las peticiones
      //http
      this.token = login_state?.token;

      this.httpOptions = {
        headers: new HttpHeaders({ 'Content-Type': 'application/json',
          'Authorization': 'Bearer ' + this.token })
      };
    });
  }

  //métodos get, post, put...
  ...
  /** Devuelve un observable del tupper cuyo identificador se pasa como parámetro */
  getTaper(id) : Observable<Taper> {
    console.log('Se va a consultar el taper con id ' + id);
    return this.http.get<Taper>(` ${this.baseUrl}/taper/${id}` ).pipe(
      tap( this.imprimirResultados ),
      catchError(this.handleError)
    )
  }
}

```

```

);
}
..
/* Inserta en el back la reserva que se pasa como parametro */
postReserva(reserva: Reserva) : Observable<Reserva> {

    return this.http.post<Reserva>(`${this.baseUrl}/reserva`, reserva, this.httpOptions).pipe(
        tap(data => console.log(data)),
        catchError(this.handleError)
    );
}
//Sigue..

```

Ilustración 24: Invocación a la api con BackendService

Un último elemento necesario para comprender el código de la aplicación son las operaciones con observables. Como se ha indicado en el ejemplo anterior, el servicio de negocio que invoca al backend devuelve observables (un flujo de datos que puede ‘llegar’ en cualquier momento) y estas estructuras de datos se manejan en angular con la librería rxjs. El trabajo con observables ha sido uno de los retos que he afrontado con una curva de aprendizaje más elevada [28] y la mayoría de utilizaciones en nuestro código pueden encontrarse en los efectos implementados y en las clases que ejecutan test unitarios:

```

//Ejemplo de uso de observables en un efecto

//pipe --> permite encadenar varias operaciones sobre un obs

//ofType --> operador de apoyo ngrx que permite filtrar
//el observable con las acciones emitidas para quedarnos únicamente
//con un tipo concreto

//switchMap --> ante la llegada de un TaperAction, hace un map
//y permite devolver otro observable (backendservice.aceptar)

//catchError --> invocado si en el observable llega un error

//of --> permite construir un observable a partir de un objeto

@Effect()
aceptarReserva$: Observable<any> = this.actions$.pipe(
    ofType(TaperActions.TaperActionTypes.ACEPTAR_RESERVA),
    switchMap(( action : TaperActions.AceptarReserva ) =>
        this.backendService.aceptarReserva( action.reserva ).pipe(
            switchMap((reserva) => [
                new TaperActions.AceptarReservaSuccess(reserva),
                new TaperActions.CargarSolicitudesPendientes(),
                new TaperActions.CargarSolicitudesTerminadas(),
            ]),
            catchError((error) => of(new TaperActions.SolicitarReservaError(error)))
        )
    )
);

```

Ilustración 25: Operaciones con observables

En este anexo se han omitido muchos elementos que podrían ser reseñables y se han acortado al máximo los ejemplos expuestos para quedarnos únicamente con lo esencial para ilustrar el concepto que se quería destacar. Puede encontrarse todo el código de la aplicación tanto en los anexos de la entrega del proyecto como en el repositorio git.

Anexo 3. Librerías/Código externo utilizado

Las librerías externas incorporadas al proyecto y utilizadas durante el desarrollo son las siguientes:

Interfaz:

- **tailwind.css:** es un framework de maquetado css que sigue la filosofía utility-first, esto es, proporciona al programador multitud de clases con una función única y concreta. No se ha seguido ningún template en el desarrollo de la interfaz y esta librería nos ha ayudado enormemente en que taperfy sea responsive.
- **Font-awesome:** nos ha proporcionado los distintos iconos que se muestran en la aplicación. Estos iconos son realmente una tipografía por lo que estos elementos se ven afectados por los modificadores css aplicados a texto, con la versatilidad que eso significa.
- **LottieFiles:** animaciones basadas en json con la que hemos amenizado algunas pantallas de espera, así como el error 404 de la aplicación que indica que el recurso no se ha encontrado

Librerías angular:

- **ng-select:** librería angular para mostrar un input type select en el componente, con un alto número de opciones de configuración, databinding, eventos, etc.
- **ngx-slider:** permite incluir un slider (selector horizontal de rango de valores) en un componente angular. Lo hemos utilizado para seleccionar el rango de precios en los filtros, así como para que el cocinero establezca el precio al que publica un taper.
- **pselect:** convierte dos input de tipo select en dos selectores con las provincias y municipio españoles, encargándose la librería de la recarga del municipio en base a la provincia elegida.
- **ngrx-store-localstorage:** sincronización entre el store ngrx y el localstorage del navegador.

Backend:

- **tymon/jwt-auth:** para la generación y validación de tokens jwt con los que el cliente puede autenticar las peticiones REST que realiza al backend.
- **spatie/laravel-http-logger:** esta librería nos ha ayudado en el debugging, ya que permite imprimir por consola la petición completa http recibida por el backend antes de ser procesada por un controlador.
- **faker:** para la generación de los datos de prueba. Esta librería, utilizada en los Factories y Seeders implementados, nos ha ahorrado mucho tiempo en la creación de usuarios, tapers, reservas, etc, ya que ejecutando un comando generábamos un conjunto de datos coherente con el que probar las funcionalidades que se iban implementando.

Anexo 4. Capturas de pantalla

En este apartado se expondrán algunas imágenes de la aplicación en funcionamiento, con los datos de pruebas generados automáticamente por la aplicación así como los incorporados durante el proceso de pruebas realizado por algunos usuarios. No se exponen todas las pantallas de la aplicación pero sí porque extendería mucho la memoria pero sí se muestran algunos screenshots significativos, tanto en la versión de escritorio de la aplicación como cuando esta se visualiza en un teléfono móvil o una tablet.

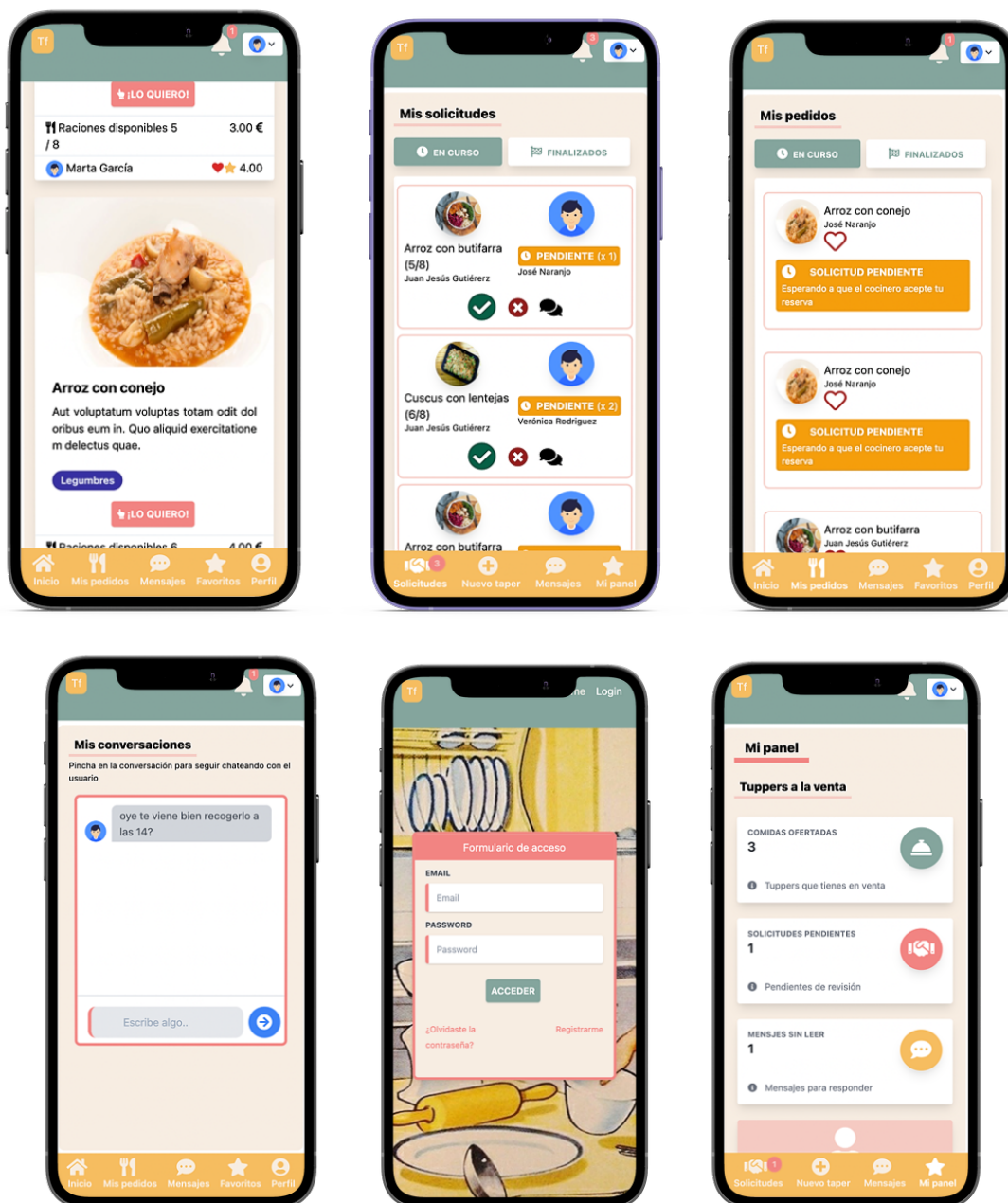


Figura 56: Capturas de pantalla de la aplicación en su versión móvil

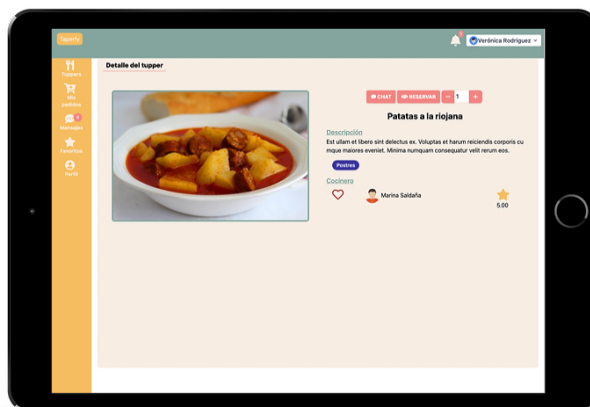
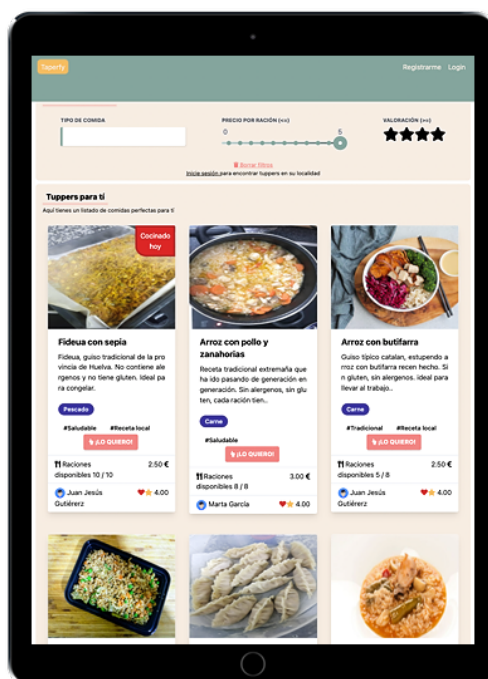
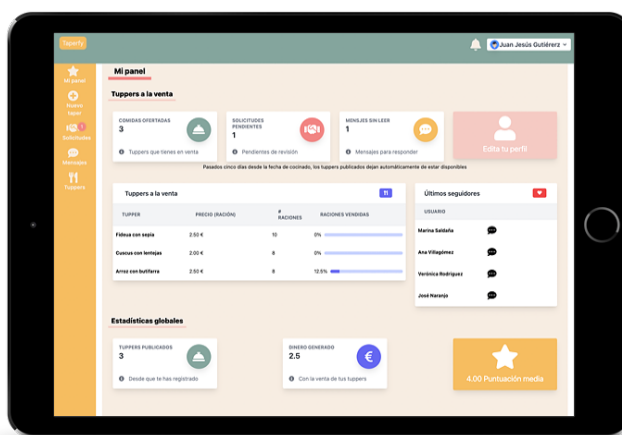
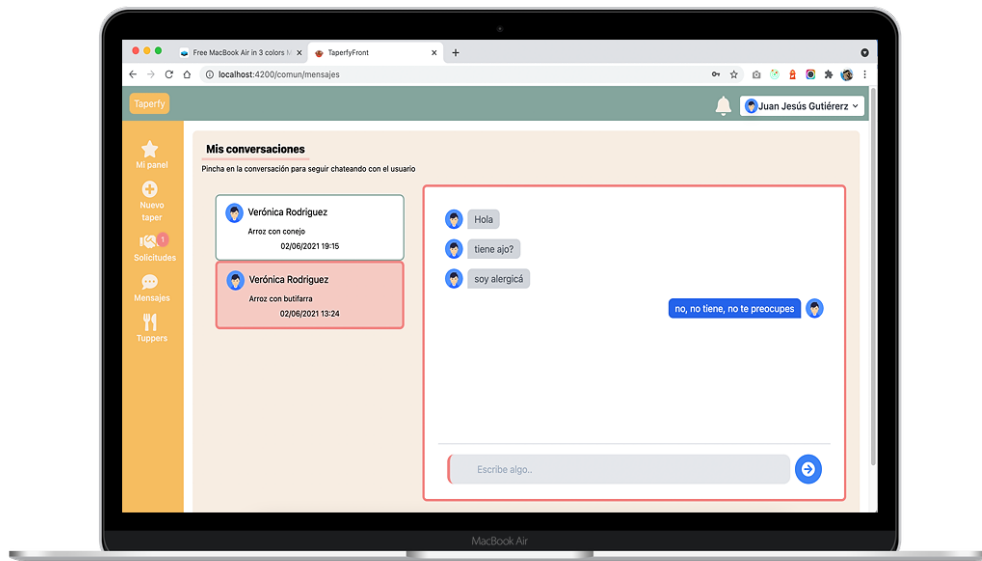
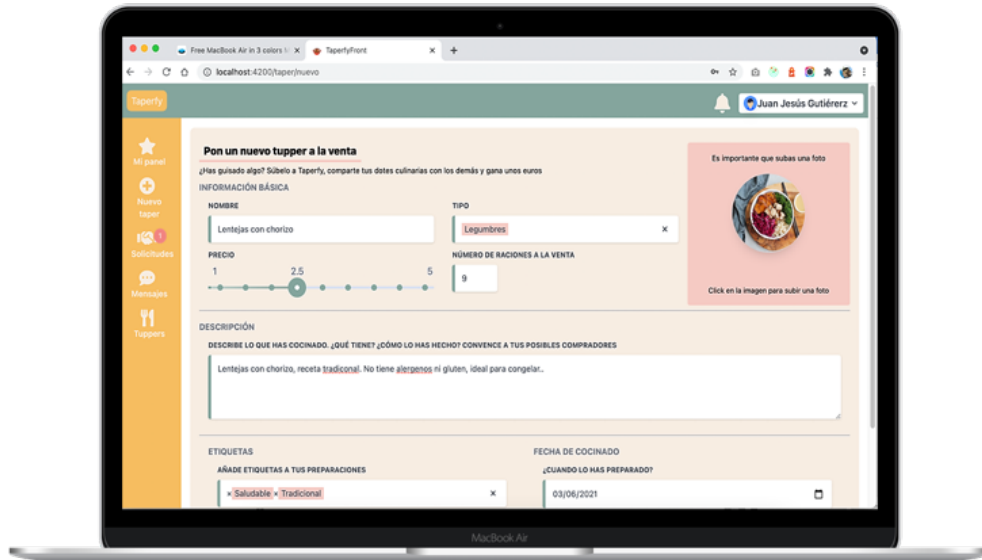


Figura 57: Capturas de pantalla de la aplicación en su versión tablet



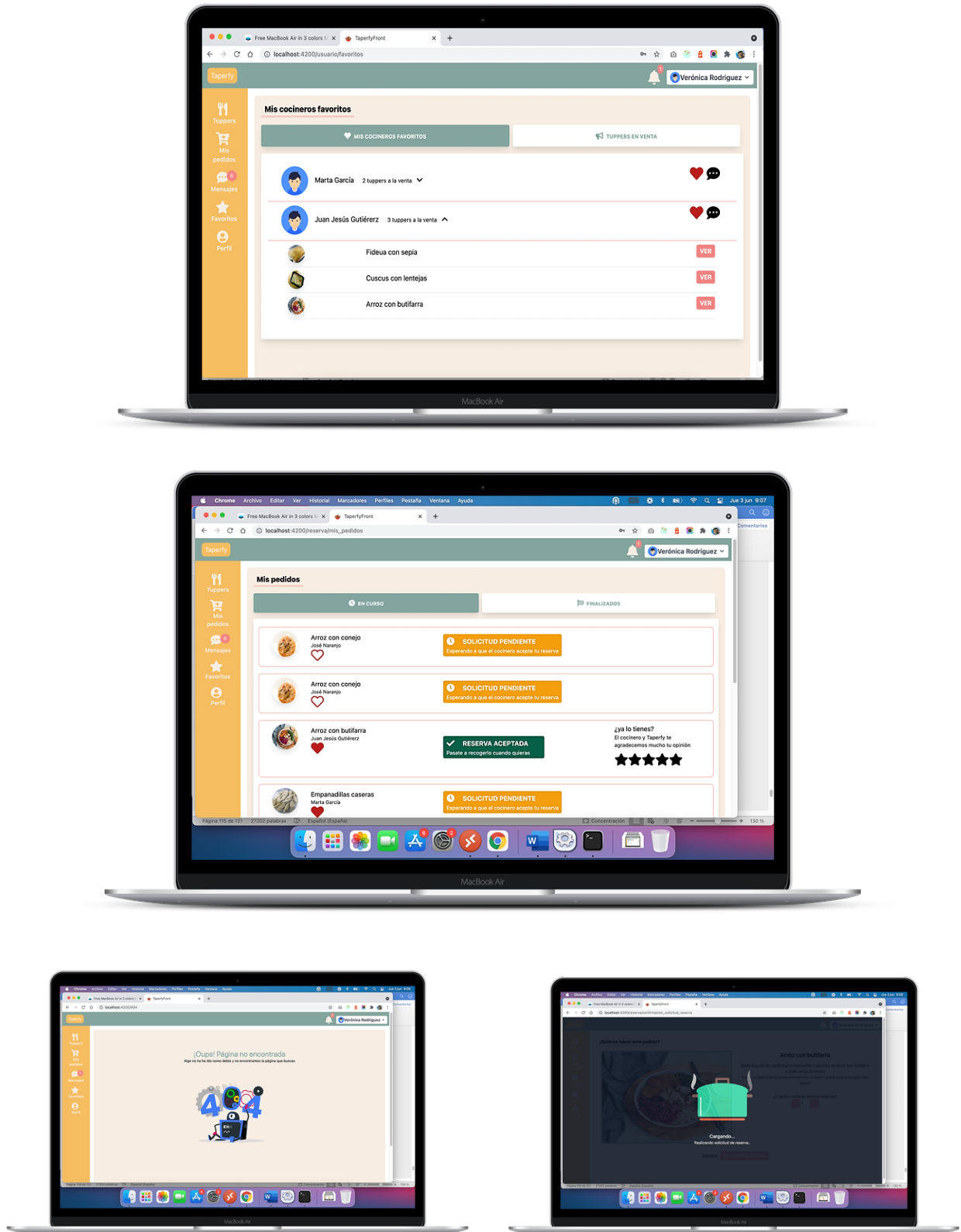
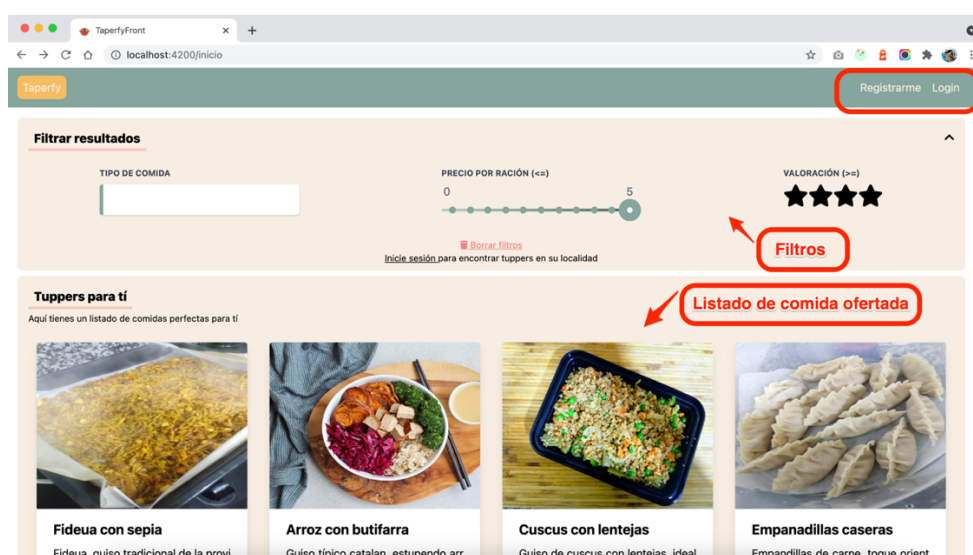


Figura 58: Capturas de pantalla de la aplicación en su versión de escritorio

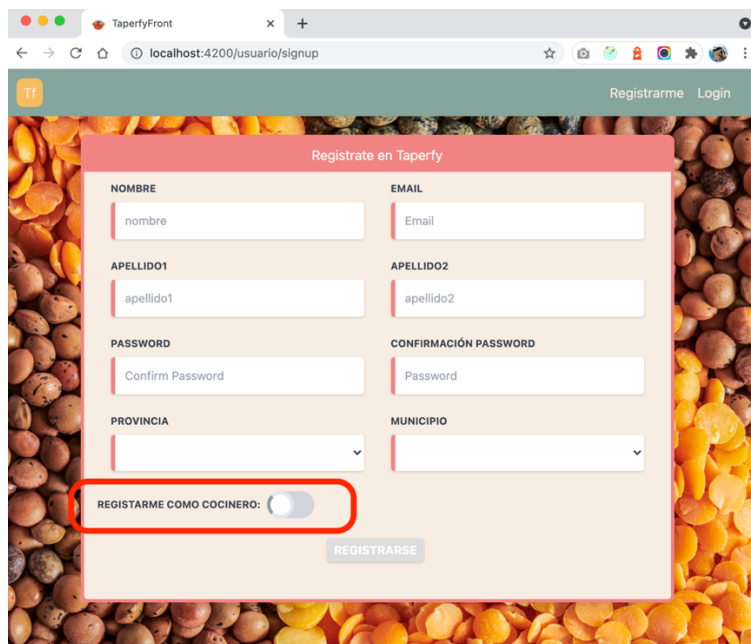
Anexo 5. Guía de usuario

Aunque la web es sencilla de usar y se parece mucho a otros e-commerce y plataformas de compra venta entre usuarios, en este apartado se expondrán unas instrucciones de uso de la aplicación, tanto para el usuario que se registre como cocinero como el perfil de comprador.

Cuando accedemos a la web vemos la página principal de la misma, que muestra una serie de filtros en la parte superior y un listado de las comidas ofrecidas en la parte central. Antes de estar autenticados en la aplicación, el sistema mostrará todos los tupperes existentes en la plataforma que aún tengan raciones disponibles, y cuya fecha de cocinado es posterior a 5 días atrás desde la fecha actual, ya que carece de sentido ofrecer una comida preparada hace mucho tiempo. Cuando el usuario se autentique, se actualizará el listado para visualizar únicamente aquellas comidas preparadas por cocineros de su localidad.

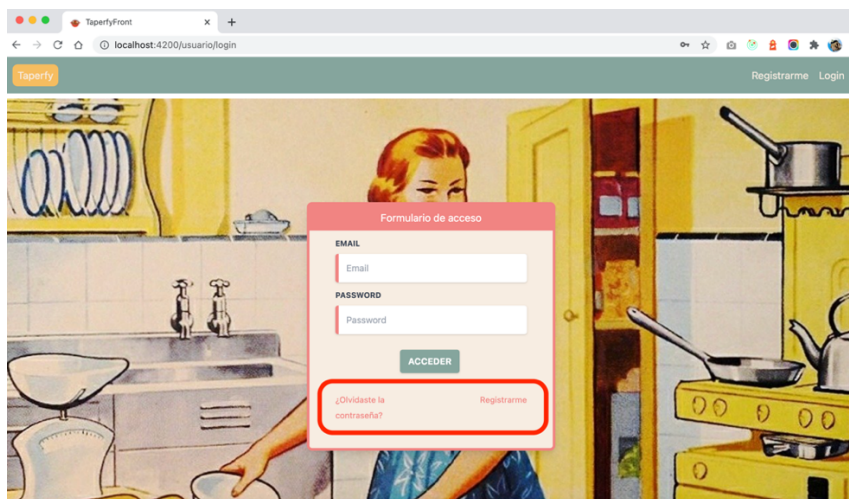


En la barra de navegación superior aparecen las opciones tanto para iniciar sesión como para registrarse en la aplicación. Si pulsamos sobre la opción de registros accederemos al formulario, donde el usuario debe indicar los datos necesarios para completar el registro: nombre, apellidos, email, contraseña, provincia y localidad.



En la misma pantalla de registro, el usuario debe decidir si quiere darse de alta como cocinero o como comprador. En función de lo seleccionado en este *toggle button*, cuando se registre la aplicación le mostrará unas opciones u otras.

Si el usuario ya está registrado en la aplicación deberá autenticarse en ella a través de la pantalla de login, accesible en la parte superior derecha de la pantalla. En esta pantalla deberá introducir su email y contraseña:



Desde la pantalla de login el usuario puede acceder a la de registro visto anteriormente, así como a una pequeña utilidad que le permitirá cambiar de contraseña en caso de haberla olvidado. Se le pedirá un email al usuario y se le mandará un correo electrónico con un enlace donde puede cambiar la password.

A5.1 Cocinero

Cuando un usuario registrado como cocinero se autentique en la aplicación, en la parte derecha (o en la parte inferior si está visualizando la web desde un móvil) se le muestran las opciones disponibles:



- **Mi panel** → donde el cocinero tiene una visión general de lo que tiene publicado en la plataforma, las solicitudes que tiene pendientes de gestionar, mensajes, usuarios que le siguen, así como una pequeña explotación estadística de sus datos.
- **Nuevo taper** → le permite publicar una nueva comida en la aplicación, describiendo el guiso, subiendo una foto, aplicando etiquetas, etc.
- **Solicitudes** → gestión de las diferentes solicitudes que provienen de los usuarios compradores
- **Mensajes** → conversaciones con los diferentes compradores para resolver dudas, negociaciones, acordar puntos de encuentro, etc. Estas conversaciones pueden estar dentro del ámbito de un taper publicado o ser directas, con los usuarios que le han marcado como cocinero favorito.
- **Tuppers** → acceso al listado principal, donde el cocinero puede ver los productos que ha publicado, así como los de su competencia.

Pulsando sobre Mi Panel accedemos a diferentes opciones y estadísticas de los tapers actualmente publicados así como algunos datos históricos:

Mi panel

Tuppers a la venta

COMIDAS OFERTADAS: 4
 SOLICITUDES PENDIENTES: 1
 MENSAJES SIN LEER: 1

Tuppers que tienes en venta
 Pendientes de revisión
 Mensajes para responder

Peasados cinco días desde la fecha de cocinado, los tupperts publicados dejan automáticamente de estar disponibles

TUPPER	PRECIO (RACIÓN)	# RACIONES	RACIONES VENDIDAS
Lentijas con chorizo	2.50 €	10	0%
Fideus con sepia	2.50 €	10	0%
Casaca con lentijas	2.00 €	8	0%
Arroz con bullabarra	2.50 €	8	25%

Últimos seguidores

USUARIO
Marina Sanjaña
Ana Villagómez
Verónica Rodríguez
José Naranjo


Edición del perfil

Chatear con seguidores

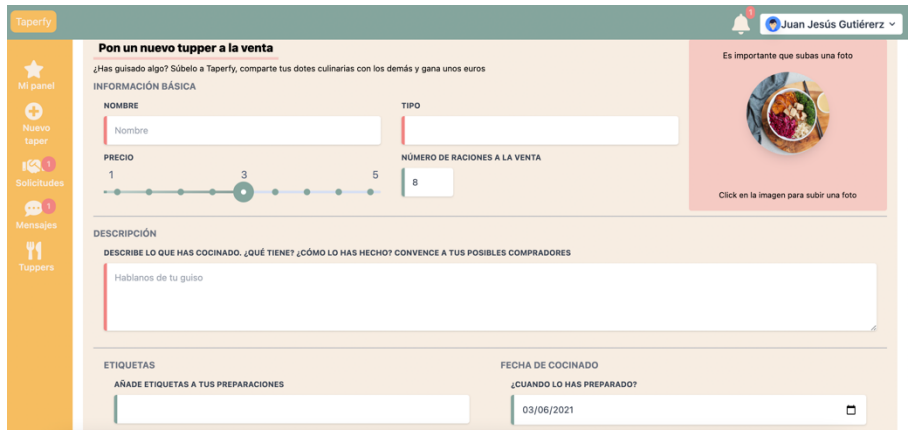
Estadísticas globales

TUPPERS PUBLICADOS: 4
 DINERO GENERADO: 5 €
 4.00 Puntuación media

Desde la pantalla anterior, que tiene un carácter informativo, el usuario cocinero puede editar su perfil o chatear directamente con sus seguidores.

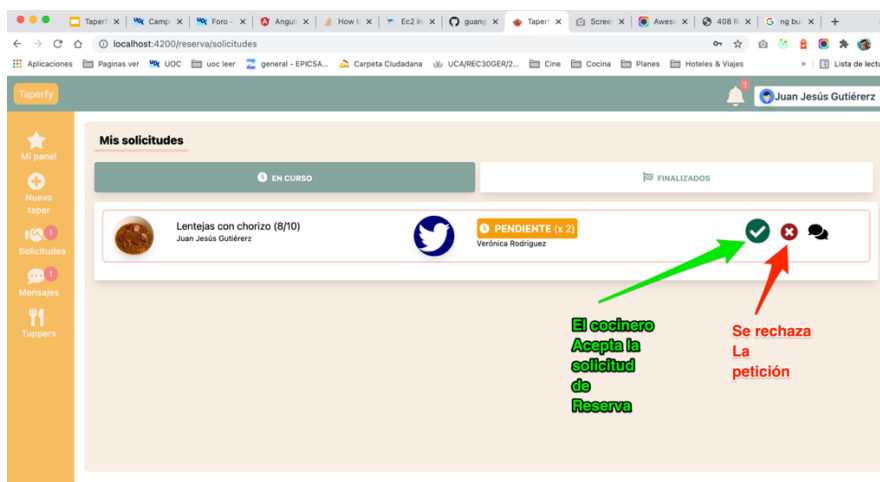
Para vender un nuevo Taper, el usuario deberá pulsar sobre la opción  que le llevará al formulario de publicación. En dicho formulario el usuario cocinero deberá rellenar de forma obligatoria los siguientes campos:

- **Nombre** del taper.
- **Tipo** de comida: carne, pescado, legumbre, etc.
- **Descripción** de lo que ha cocinado. Esta descripción debería ser atractiva para los posibles compradores, indicar que ingredientes ha usado, si tiene alérgenos, de dónde viene la receta, etc.

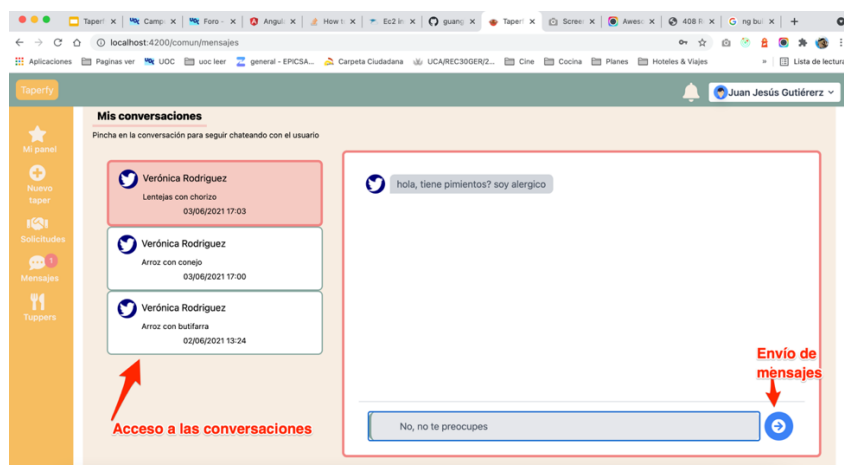


- Una o varias **etiquetas** que añaden información adicional sobre el producto: si es una receta local, si es saludable, si tiene gluten, tradicional, etc.
- **Fecha** en la que ha cocinado la comida.
- Adicionalmente, el vendedor tiene la posibilidad de incluir una **foto** de su comida pulsando sobre la foto que aparece por defecto en la parte superior derecha del formulario.

En la opción de **solicitudes** el usuario cocinero recibe las diferentes peticiones de reserva de tuppers de los compradores y decide si acepta o rechaza dichas peticiones. Adicionalmente, tiene un acceso directo para acceder a la conversación con el posible comprador:



Por último, si accede a la sección de **mensajes**, se visualiza en la parte izquierda de la pantalla todas las conversaciones con otros usuarios y pulsando sobre cada una de ellas se visualiza en la parte derecha el contenido de la conversación, posibilitando al usuario a escribir nuevos mensajes utilizando el cuadro de texto inferior:



A5.2 Comprador

Al igual que ocurría con los cocineros, cuando hace login un usuario que está registrado como comprador, en la parte derecha (o en la parte inferior para dispositivos móviles) de la aplicación aparecen opciones específicas para su perfil



- **Inicio** → donde el comprador puede ver las comidas que se ofrecen en la plataforma y aplicar filtros para encontrar los tupperes que más se adecuen a sus gustos y necesidades.
- **Mis pedidos** → puede revisar todos los pedidos que haya realizado a los distintos cocineros y se encuentren pendientes o en algún estado finalizado. Adicionalmente, desde esta pantalla valorará la experiencia de compra y producto de aquellos tupperes que ya haya adquirido.
- **Mensajes** → conversaciones con los distintos cocineros para establecer una negociación, resolver dudas, pactar un punto de recogida, etc.
- **Favoritos** → acceso directo a la información de sus cocineros favoritos. Desde esta pantalla el comprador puede ver que comida tienen publicado sus cocineros de confianza y chatear directamente con ellos.

En la pantalla principal, el usuario ve la comida publicada, así como (en la parte superior) una serie de filtros que se puede aplicar a ese conjunto de tarjetas. De forma automática, se aplican dos criterios de filtrado: tupperes cuya fecha de cocinado no es anterior a 5 días y comida de cocineros de la localidad del usuario. Adicionalmente, el usuario puede aplicar más criterios de búsqueda para acotar los resultados en el listado principal

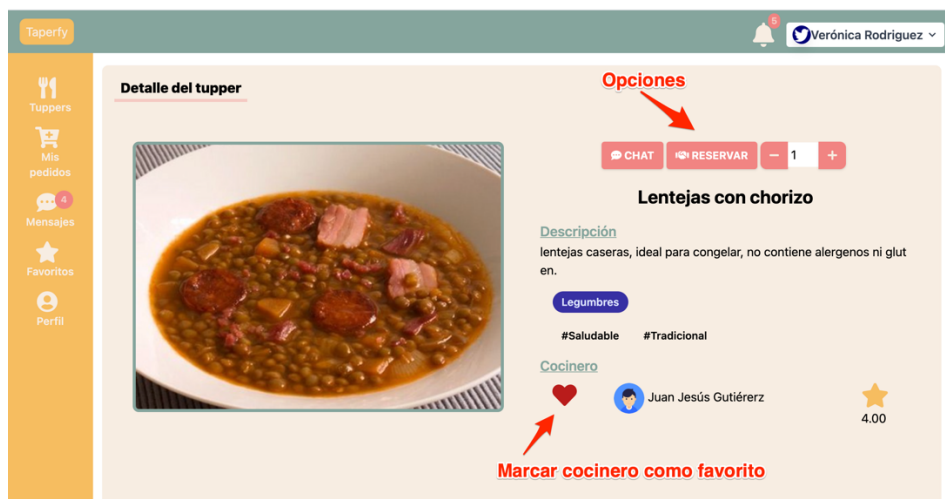


La comida ofrecida puede filtrarse por:

- **Tipo de plato:** carnes, pescados, legumbres, etc.
- **Precio máximo** por ración. Todas las raciones de taperfy tienen un precio que está en el rango desde los 50 céntimos a los 5 euros. Con este filtro podremos visualizar raciones que tengan el precio por debajo de un determinado valor.
- **Reputación del cocinero**, por ejemplo, mostrar comida de cocineros con al menos 3 estrellas.

Proceso de reserva

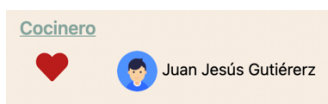
Una vez el usuario encuentre una comida que se ajuste a sus necesidades, este puede pulsar sobre la imagen o sobre el botón para acceder a la página de detalle:



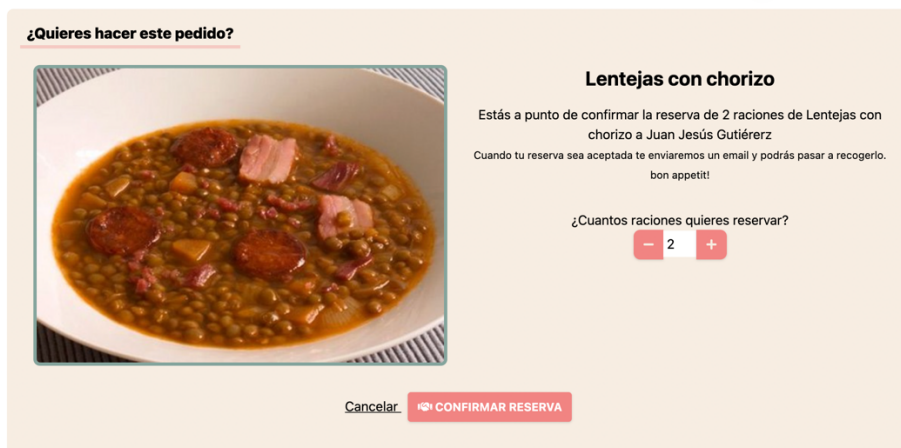
Desde esta página, el usuario puede empezar una conversación con el cocinero para resolver dudas, negociar, establecer un punto de entrega... para ello, deberá pulsar el botón **CHAT**.

Una vez haya decidido comprar el producto, deberá realizar una solicitud de reserva al cocinero pulsando sobre el botón **RESERVAR**, lo que le llevará a una pantalla de confirmación

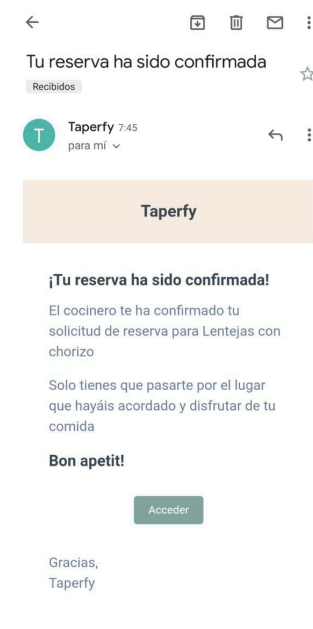
Además, desde la pantalla de detalle el usuario comprador tiene la posibilidad de marcar al cocinero como favorito pulsando el corazón que aparece en la parte inferior:



Tras seleccionar el número de raciones que quiere solicitar y pulsar la opción para hacer la reserva, la aplicación muestra una pantalla de confirmación:

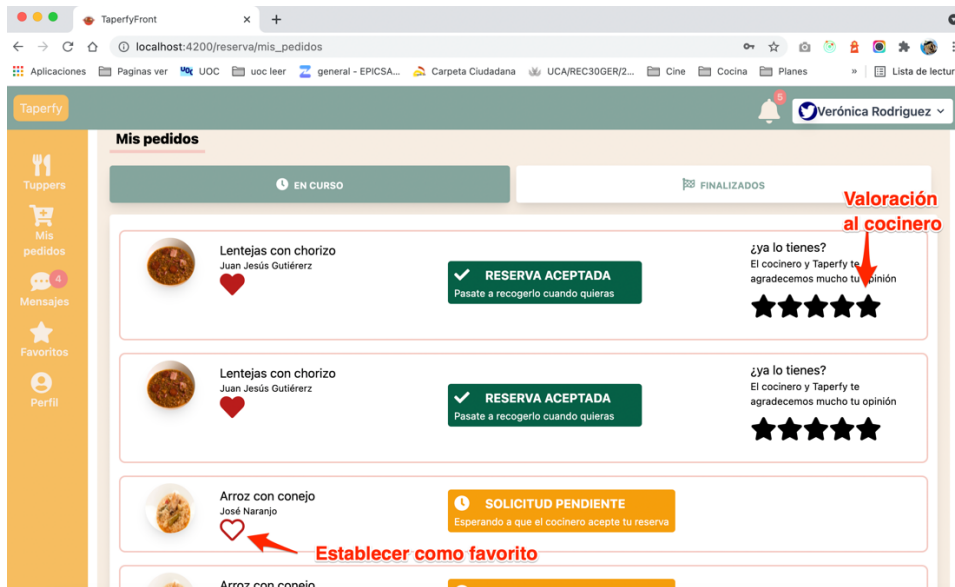


Donde el usuario deberá confirmar el número de raciones que quiere solicitar. Tras la confirmación de la solicitud de reserva, el usuario recibe un correo electrónico informativo y cuando el cocinero acepte o rechace dicha reserva se recibirá otro correo indicándolo.



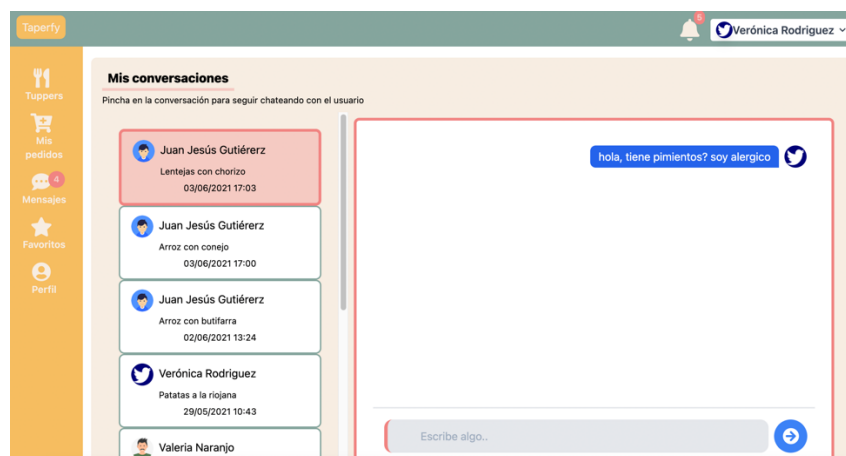
Otras opciones de menú

En la opción de menú **Mis Reservas** el usuario comprador puede ver aquellas solicitudes de reservas, tanto pendientes como terminadas que haya realizado en la plataforma:



Una vez que la reserva sea aceptada por el cocinero, el comprador tiene la opción de valorar el producto pulsando sobre una de las estrellas, con la puntuación que considere más oportuna. Adicionalmente, desde esta pantalla también puede establecer los cocineros como favoritos (o que dejen de serlo), para ella basta con que pulsar el corazón que aparece junto al nombre del cocinero.

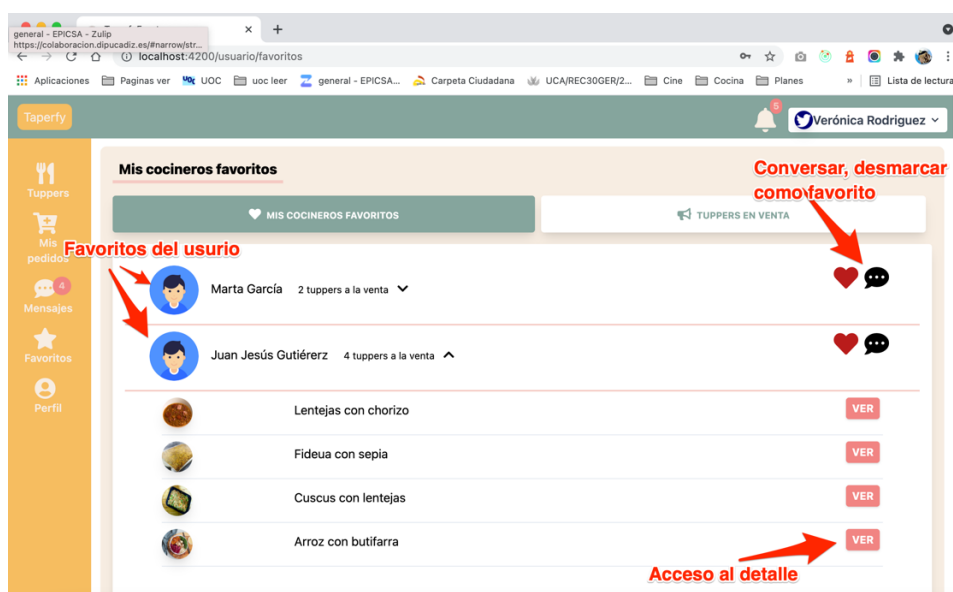
De forma análoga a lo que ocurría con los vendedores, en la sección **Mensajes** el usuario puede acceder a las conversaciones que tiene iniciadas con los diferentes cocineros de los tupper así como con sus favoritos. La pantalla es igual a la que aparecía en los cocineros, con las conversaciones mostrándose a la izquierda y el chat a la derecha.





Por último, en la opción **Favoritos** se tiene un acceso directo al listado de cocineros que el usuario haya marcado como favorito, con la posibilidad de desplegar cada uno de los elementos del listado y ver una sub-lista con los tupperes que tiene publicado ese cocinero.

En este listado, el usuario puede iniciar una conversación con su cocinero de confianza, desmarcarlo para que deje de ser favorito o acceder al detalle de cada una de sus comidas publicadas. Adicionalmente, la pantalla muestra una segunda pestaña donde se muestra un conjunto de tarjetas similar a la pantalla principal con la particularidad de que dichas tarjetas solo corresponden a guisos de los cocineros que el comprador tiene establecidos como favoritos.



Anexo 6. Libro de estilo

A nivel visual, se ha pretendido conseguir un estilo vintage, con la elección de una imagen de fondo en la pantalla de login y el uso de una paleta de colores pastel y poco saturados. Algunos elementos de la línea gráfica que define el trabajo:

- **Logotipos:** aparece tanto cuando se instala la aplicación en un dispositivo (Taperfy es una PWA) como en el favicon cuando se muestra en el navegador



- **Paleta de colores:** se ha usado la web **colors.co** para obtener la paleta de colores que se ha incluido en la aplicación.

<https://colors.co/f6bd60-f7ede2-f5cac3-84a59d-f28482>

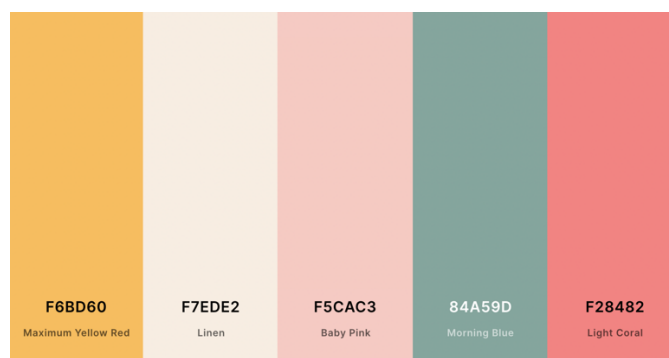


Figura 59: Paleta de colores de la aplicación

Para disponer de la paleta de colores en las clases de utilidad proporcionadas por Tailwind se ha tenido que extender la configuración del framework para incluir los códigos hexadecimales de nuestros colores en el fichero **tailwind.config.js**

- **La Tipografía la tengo pendiente – Tengo que estudiar si usar una Font de google Font repercute demasiado al rendimiento.**
- **Botones,** de algunos de los colores de la paleta, con bordes ligeramente redondeados y un padding que cambia ligeramente en función del ancho de la pantalla.



Anexo 7. Bibliografía

- [1] H. digital, «Hostelería digital,» [En línea]. Available: <https://www.hosteleriadigital.es/2020/07/24/la-comida-a-domicilio-online-crece-a-doble-digito-anual-y-alcanza-los-740-millones-de-euros/>.
- [2] «El País,» [En línea]. Available: https://elpais.com/economia/2017/12/01/actualidad/1512125659_853869.html.
- [3] «Gastrómetro de JustEat,» [En línea]. Available: <https://www.just-eat.es/explora/gastrometro>.
- [4] «Módulos en Angular,» [En línea]. Available: <https://medium.com/@yonem9/angular-qu%C3%A9-son-los-m%C3%B3dulos-y-c%C3%B3mo-se-refactoriza-una-aplicaci%C3%B3n-9457550e8e9>.
- [5] «JWT Auth,» [En línea]. Available: <https://jwt-auth.readthedocs.io/en/develop/>.
- [6] «Instalación JWT Auth,» [En línea]. Available: <https://jwt-auth.readthedocs.io/en/develop/laravel-installation/>.
- [7] «Tutorial JWT Angular + Laravel,» [En línea]. Available: https://www.youtube.com/watch?v=32b9o6e_LBg&list=PLe30vg_FG4OSbizS6Gpw_LICp9zBcmjZU&index=2.
- [8] «JsonWebToken,» [En línea]. Available: <https://www.jsonwebtoken.io/>.
- [9] «Test unitarios en Angular,» [En línea]. Available: <https://angular.io/guide/testing>.
- [10] «Mocha Reporter en los test unitarios Angular,» [En línea]. Available: <https://juristr.com/blog/2018/02/add-mocha-reporter-angular-cli-tests/>.
- [11] «Guía Ngrx,» [En línea]. Available: <https://ogomez.medium.com/guia-rapida-para-entender-el-patron-redux-y-angular-con-ngrx-e60d39d35f1b>.
- [12] «Sincronización ngrx-store local-storage,» [En línea]. Available: <https://www.npmjs.com/package/ngrx-store-localstorage>.
- [13] «Framework CSS Tailwind,» [En línea]. Available: <https://tailwindcss.com/>.
- [14] «Tailwind en Angular,» [En línea]. Available: <https://www.amadousall.com/how-to-add-tailwind-css-to-your-angular-application/>.
- [15] «Angular Animations,» [En línea]. Available: <https://angular.io/guide/animations>.
- [16] «LottieFiles,» [En línea]. Available: <https://lottiefiles.com/>.
- [17] «Herramienta de creación de iconos para PWA,» [En línea]. Available: <https://github.com/pverhaert/ngx-pwa-icons>.
- [18] «Angular Lazy Loading,» [En línea]. Available: <https://angular.io/guide/lazy-loading-ngmodules>.
- [19] «Can I Use - Websockets,» [En línea]. Available: <https://caniuse.com/websockets>.
- [20] «Pusher,» [En línea]. Available: <https://pusher.com/>.
- [21] «Abbyly,» [En línea]. Available: <https://ably.com/>.
- [22] «Broadcasting en Laravel,» [En línea]. Available: <https://laravel.com/docs/8.x/broadcasting>.
- [23] «CodeLab Websockets Firebase,» [En línea]. Available: <https://firebase.google.com/codelabs/firebase-web#0>.
- [24] «Envío de correos desde Laravel,» [En línea]. Available: <https://programacionymas.com/blog/como-enviar-mails-correos-desde-laravel>.

- [25] «Personalizar emails Laravel,» [En línea]. Available: <https://laravel.com/docs/7.x/mail#customizing-the-components>.
- [26] «Geolocalización con HTML 5,» [En línea]. Available: https://www.w3schools.com/html/html5_geolocation.asp.
- [27] «Amazon S3 Storage en Laravel,» [En línea]. Available: <https://dev.to/aschmelyun/getting-started-with-amazon-s3-storage-in-laravel-5b6d>.
- [28] «Explicación observables,» [En línea]. Available: <https://desarrolloweb.com/articulos/introduccion-teorica-observables-angular.html>.
- [29] «Inclusión de Tailwind en Angular,» [En línea]. Available: <https://www.amadousall.com/how-to-add-tailwind-css-to-your-angular-application/>.

Anexo 8. Vita

Juan Jesús Gutiérrez Ramos es natural de Cádiz, licenciado en Ingeniería Informática por la Universidad de Sevilla en el año 2007, ha trabajado en diferentes empresas del sector TIC, casi siempre como desarrollador backend, colaborando en la construcción de múltiples aplicaciones en su mayoría implementadas con tecnologías Java.

Actualmente trabaja como desarrollador en la empresa pública provincial de la Diputación de Cádiz, EPICSA.

La principal motivación para emprender el máster al que pertenece este trabajo es renovar el background tecnológico, y actualizarse en el cada vez más apasionante y complejo mundo de tecnologías web.