



Escuela
Politécnica
Superior

Sistema de gestión logística y de usuarios para economatos sociales



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Diego Maroto García

Tutor/es:

José García Rodríguez

John Alejandro Castro Vargas

Mayo 2021



Universitat d'Alacant
Universidad de Alicante

Sistema de gestión logística y de usuarios para economatos sociales

Arquitectura y software lowcost

Autor

Diego Maroto García

Tutor/es

José García Rodríguez

Dtic

John Alejandro Castro Vargas

Dtic



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2021

Preámbulo

La pobreza en España es un problema social que ha afectado a miles de familias a lo largo de los años, muy duramente por la crisis de 2008 y, actualmente, por la pandemia de COVID-19. En esta situación y a pesar de las ayudas económicas del gobierno, muchas personas se verán dramáticamente perjudicadas, afectando incluso a la adquisición de alimentos básicos. Los economatos sociales son proyectos que suelen ser llevados en comunidad, con pocos recursos económicos, y que permiten el acceso a recursos básicos por un precio inferior al de mercado. Sin embargo, estos proyectos tienen un volumen considerable de usuarios que se magnifica en épocas de crisis como la que ha ocasionado la pandemia.

El desarrollo del proyecto se ha llevado a cabo a través de un estrecho feedback directamente con la gente que trabajará con este software; habiendo creado desde cero una solución adhoc que busca encajar lo mejor posible con las necesidades que tienen los voluntarios que atienden a las familias que acuden al economato.

Se ha pretendido en todo momento mantener una infraestructura basada en la alta disponibilidad, automatización de procesos manuales y de bajo coste económico.

Se ha buscado la máxima simplicidad de la experiencia de usuario, haciendo uso de diseños minimalistas e interacciones cortas, debido a la posibilidad de que haya voluntarios con poca educación digital y la necesidad que éstos puedan ser capaces de usar la aplicación o aprender a usarla con facilidad.

Agradecimientos¹

Este trabajo no habría sido posible sin el apoyo y el estímulo de mi compañera y amiga, Vanesa, cuya amabilidad y dedicación me ha permitido mantener la concentración y motivación que un proyecto como este necesita para llegar a buen puerto.

No puedo dejar de agradecer a mis tutores John y a José la predisposición y disponibilidad mostradas desde el primer día que les contacté para empezar a trabajar en este TFG. Una flexibilidad que me han permitido expresar mis conocimientos y habilidades para conseguir construir algo de lo que sentirme orgulloso; y que, con suerte, permitirá facilitarle la labor lo suficiente a los economatos sociales para que su preocupación sea ayudar a las personas y no qué tecnología usar para ello.

Es a estas personas a quien dedico este trabajo.

¹Por si alguien siente curiosidad, lo que me une a Vanesa es algo más que un anillo, es un vínculo y una complicidad que aún no han conocido límites.

Índice general

1	Introducción	1
1.1	Motivaciones	2
1.2	Objetivos	3
1.3	Relación con asignaturas	3
2	Estado de arte	5
2.1	La importancia de la tecnología en el campo social	6
2.2	El impacto de la adopción tecnológica en bancos de alimentos	8
2.3	Estudio de mercado	9
2.3.1	Las bases del proyecto	10
2.3.1.1	ERP's y el Free Open Source Software	11
2.3.2	Decidiendo arquitectura	13
2.3.2.1	Raspberry pi (DIY)	13
2.3.2.2	Computación elástica bajo demanda (Cloud)	15
2.3.2.3	Serverless (Cloud)	19
2.3.2.4	¿Cómo funciona?	19
2.3.2.5	Vista general de ventajas y desventajas	20
2.3.2.6	Cuándo se usa el principio Serverless	20
2.3.2.7	Ejemplos de uso con heroku	21
2.3.3	Stack FrontEnd/BackEend	23
3	Metodología	25
3.1	Requisitos mínimos	25
3.2	Fases del desarrollo	25
3.3	Metodología ágil utilizada	26
3.4	Herramientas hardware	27
3.5	Herramientas software	27
3.5.1	Documentación	28
3.5.2	Planificación y cumplimiento de las etapas	28
3.5.3	Desarrollo	28
4	Estudio viabilidad	31
4.1	Planificación temporal	31
4.2	Gestión de riesgos	33
5	Especificación de requisitos	37
5.1	Objetivos del sistema	37
5.2	Casos de uso	39

6	Desarrollo	45
6.1	Funciones	45
6.2	Pantallas	46
6.2.1	Login	48
6.2.2	Pedidos	51
6.2.3	Caja	56
6.2.4	Almacén	57
6.3	Implementación	58
6.3.1	Base de datos	58
6.3.2	Esquemas y servicios	60
6.3.2.1	Login y registro	60
6.3.2.2	Productos	62
6.3.2.3	Pedidos	64
6.3.2.4	Caja	67
6.3.2.5	Almacén	69
6.3.2.6	Roles y controladores	70
6.4	Conclusiones, patrones y revisión del código	73
6.5	Resultado final	74
7	Conclusiones	75
7.1	Revisión de los objetivos marcados	75
7.2	Conclusiones	76
	Bibliografía	77

Índice de figuras

2.1	Compra de módulos de Odoo	11
2.2	Comunicación entre módulos de Odoo	12
2.3	Raspberry Pi 4 Modelo B	13
2.4	Raspberry Pi integrada en el interior de un teclado	13
2.5	Raspberry Pi protegida con funda	14
2.6	Arquitectura simplificada	16
2.7	Presupuesto simplificado	17
2.8	Configurando proyecto en Heroku	21
2.9	CI/CD en Heroku	22
2.10	Ejemplo de uso de Heroku por línea de comandos	22
4.1	Milestones cerradas en Github	32
6.1	Ejemplo colores de la aplicación	47
6.2	Login esperando respuesta del backend	48
6.3	Login error credenciales	49
6.4	Dashboard	50
6.5	Pedidos: Step 1 - Búsqueda de familia	51
6.6	Pedidos: Step 2 - Resumen familia en gestión del pedido	52
6.7	Pedidos: Step 2 - Gestión del pedido	52
6.8	Pedidos: Step 3 - Resumen del pedido	53
6.9	Pedidos: Vista previa factura	54
6.10	Pedidos: Imprimir factura	55
6.11	Caja	56
6.12	Almacén - Resumen de pedidos por despachar	57
6.13	Almacén - Despachando un pedido	57

Índice de Códigos

6.1	Inyección proveedores ddbb	58
6.2	Gestión de roles para acceder a datos	59
6.3	User schema	60
6.4	Creación de un usuario	61
6.5	Login de un usuario	61
6.6	Esquema de producto	62
6.7	Product service: Añadir producto nuevo	63
6.8	Esquema de pedido	65
6.9	Invoice service: Crear nuevo pedido	66
6.10	Websocket service: Difusión de mensajes	66
6.11	Invoice service: Resolver pedido	68
6.12	Invoice service: Despachar pedido	69
6.13	User Controller: Guards en el controlador de usuario	70
6.14	Guard JWT: Verificación de un json web token e inyección de datos en la request	71
6.15	Guard Roles: Verificación de roles necesarios para el acceso	71
6.16	Jwt service: Creación de token	72

1 Introducción

En este apartado se va a presentar el trabajo, se introducirá la situación social actual que ha hecho de este trabajo una necesidad, qué lo ha motivado y, por último, se relacionará el alcance del trabajo y las asignaturas cursadas en el grado de Ingeniería Informática de la Universidad de Alicante.

El objetivo de este proyecto consiste en aportar una solución informática a los economatos sociales, dado que usan procedimientos manuales que les lastran en tiempo y confianza. Como por ejemplo el seguimiento de un expediente para mantener ciertas limitaciones o los cierres de caja y seguimiento de precios de los productos. Este proyecto toma como referencia aplicaciones de gestión de stock, facturación o erp's (Enterprise Resource Planning).

Los economatos sociales, como bancos de alimentos, se apoyan fuertemente en el voluntariado de la gente, habitualmente éstos son vecinos del mismo barrio en el que se encuentra. Dado que se dedican a ayudar a familias con pocos o ningún recurso, es común encontrarlos en zonas de clase media/baja e incluso en barrios con familias en peligro de exclusión social; esta es una de las razones por las que suelen carecer de equipos o procedimientos informatizados lo suficientemente desarrollados como para que el buen funcionamiento de éstos deje de consistir una carga y una preocupación.

Los economatos sociales suelen ser sedes de distribución de alimentos y productos de higiene, que usan el dinero del que les provee alguna ONG para mantener el stock en sus almacenes. El economato social para el que se ha realizado este TFG está bajo la dirección de Caritas y, a priori, la aplicación dará solución exclusivamente a este economato, hasta que se haya refinado y adoptado lo suficiente como para dar el salto a otros economatos de la ciudad de Alicante.

La entrada de soluciones de software desarrolladas adhoc para esta clase de actividades empuja en la buena dirección, ayudando a que la gente que se preocupa de poder atender a las familias lo mejor posible, realmente se dedique a ello; sin tener que preocuparse de poner parches procedimentales a la forma en la que trabajan, que si bien ayudan, no solucionan. Normalmente hacen un uso principal del elemento humano, confiando en el buen hacer y organización personal de cada uno de los y las voluntarias. Algo de lo que se aprende en el grado de informática es que el elemento humano da pie inequívocamente a errores de procedimiento y datos; de esta forma, la automatización de recogida, procesamiento, almacenamiento y distribución de datos se hace cargo de una de las obligaciones manuales que estaban llevando a cabo en persona.

Del economato para el que hemos trabajado este proyecto hemos visto algunos de estos procedimientos manuales, hojas de excel que se deben guardar apropiadamente y almacenar

en carpetas concretas manualmente para su posterior uso, dado que una de las necesidades que tiene este economato es el seguimiento de expedientes. Pues cada familia tiene ciertas limitaciones en función de su situación social, limitaciones que decide Caritas, como son saldo máximo a usar en el mes en curso y en relación a este, stock máximo de productos concretos que se puede retirar. Hacer uso de seguimientos manuales mientras, en el mismo momento en que se está atendiendo a esta familia, hay una cola interminable en la calle esperando; es cuanto menos estresante y, además, ineficiente y propenso a errores.

1.1 Motivaciones

Este país y nuestra sociedad, históricamente, ha sido social, dedicando recursos económicos y personales en amparo de los más vulnerables. Que aún haya actividades tan importantes, como las que se llevan a cabo en los economatos sociales, que estén rozando la superficie del uso de la capacidad informática y tecnológica de la que disponemos actualmente; empieza a hacer sentir urgente la necesidad de integraciones hechas a medida, que les brinden reducciones de tiempos en sus tareas, automatización de tareas repetitivas y la seguridad e instantaneidad de obtención y cálculo de datos veraces.

A día de hoy, la falta de recursos económicos no es excusa ni debería ser impedimento para tener soluciones de software de libre acceso, código abierto y de infraestructura en servicios en la nube. Una de las primeras premisas, bajo las cuales, se estableció la base de actuación y diseño de la aplicación para este trabajo fue ésta: la falta de recursos económicos. No siempre por no tener dinero, si no que a menudo, la burocracia hace tan tediosa la adopción de alguna solución informática, que por no ver claro su uso y beneficio, o la falta de conocimiento o capacidad de aplicación, se acaba dejando de lado; de ésta manera, un proyecto libre y gratuito, saltaría de golpe un muro que aparentemente se había convertido en uno francamente duro de salvar.

Vista esta problemática, es un ingeniero de software la mejor carta a jugar para investigar y construir una solución que abarque estas premisas y así, facilitar y mejorar el trato y la calidad que reciben estas familias sin recursos; que lo único que quieren es poder comer en el mes en curso y no preocuparse de que se haya perdido un fichero o de que no encuentran el historial de su expediente y que les hagan esperar mientras llaman por teléfono a la sede para informarse. Esto, a su vez, facilita el trabajo de los voluntarios; que pudiendo confiar en una herramienta que les va a brindar un proceso limpio y automático para realizar su función en el economato, se verán aliviados, aunque sea un poco, de la realidad que viven cada día que levantan la persiana del almacén; pues se habrán quitado de encima el peso de la incertidumbre y la preocupación de hacer mal su función como voluntarios.

Todo esto es lo que ha motivado a mis tutores a proponer éste como trabajo de fin de grado y a mí, como estudiante de informática, a solicitar llevarlo a cabo y defenderlo. Si bien he tenido la suerte de no necesitar nunca acudir a un banco de alimentos, sí he vivido de cerca la ansiedad y la carga que supone la falta de recursos. Ser voluntario puede ser, además de dar tu tiempo en trabajo manual, dar tiempo y conocimiento para crear una solución que les facilite el trabajo.

1.2 Objetivos

En este proyecto se pretende dar una solución de software mediante la consecución de una serie de objetivos propuestos, que permitan alcanzar la meta final de este trabajo:

- Investigar la viabilidad de selfhosting con una raspberry pi
- Investigar la viabilidad de servicios en la nube y precios
- Comparar y decidir la arquitectura final que dará accesibilidad al servicio
- Analizar las necesidades del economato y decidir el alcance y funcionalidades
- Investigar capacidades y herramientas de leer códigos de barras (hardware, webcam)
- Diseño de una base de datos adecuada a las necesidades del economato
- Decidir tecnología para llevar a cabo la solución de software
- Implementar solución de software del backend
- Diseñar e implementar frontend SPA
- Reuniones con los implicados de forma ocasional para obtener feedback
- Entregar solución software cerrada y funcionando, infraestructura, backend y frontend

1.3 Relación con asignaturas

Los estudiantes de la Ingeniería Informática de la Universidad de Alicante son preparados como profesionales, haciendo uso de una sólida y amplia formación que los encamine a ser capaces de dirigir y realizar las tareas de que son propias de todas las fases del ciclo del software, arquitectura de hardware y gestión de proyectos. Los estudiantes se preparan para resolver problemas de cualquier ámbito de la tecnología de la información y las comunicaciones, aplicando conocimiento científico y métodos y técnicas propios de la ingeniería. En el aspecto personal, me he decantado siempre por el software y la administración de sistemas; motivo de que haya cursado el grado de Ingeniería Informática.

Una recopilación de las asignaturas cursadas que ayudan a que este trabajo se pueda realizar en tiempo y forma, son:

- Diseño de bases de datos
 - Herramientas avanzadas para el desarrollo de software
 - Diseño de sistemas de software
 - Ingeniería web
 - Programación
 - Administración de Sistemas Operativos y Redes de computadores
 - Gestión proyectos informáticos
-

2 Estado de arte

En este apartado se va a presentar el impacto y el beneficio de la tecnología entrando, cada vez más, en el campo práctico de los servicios sociales.

La tecnología ha revolucionado nuestra forma de consumir, de relacionarnos y de informarnos. Donde tampoco han sido ajenos a la evolución de la tecnología, es en el ámbito de los servicios sociales. Sin embargo, comparándolo con otros sectores mucho más maduros y con más presupuesto como la banca o el comercio, el sector social parece no haber sido capaz de adoptar o tener acceso a toda la tecnología que ya está desarrollada, afectando negativamente al servicio humano que se pretende dar.

La tecnología en los servicios sociales tiene un papel cada vez mayor, éste puede adoptar muchas formas, como el uso de inteligencia artificial, los sistemas de gestión de casos y/o servicios especialmente desarrollados para hacer uso de la tecnología de asistencia. Como se estaba explicando, éstos avances tecnológicos pueden ayudar a mejorar la planificación, gestión y prestación de servicios sociales, pero también es importante comprender los desafíos que plantea la digitalización, como la falta de conocimiento sobre las nuevas tecnologías, su costo y cómo garantizar la protección de la privacidad y la seguridad.

Aquí entraría en juego la Universidad de Alicante con su plan formativo de Ingeniería Informática, formando personal con los conocimientos y habilidades necesarias para ayudar estudiar, acortar y amortiguar los miedos que suponen en este sector la aproximación a la tecnología.

2.1 La importancia de la tecnología en el campo social

Puede que la crisis del COVID-19 pueda resultarnos nueva, sin embargo las crisis medioambientales y de salud han sido una constante en nuestro planeta. Éstas siempre han superado las agendas planificadas y los objetivos establecidos de las organizaciones comunitarias para continuar ejerciendo labores de manera tradicional y más importante aún, para coordinar asistencia social y llegar a los más necesitados. Por ejemplo en Puerto Rico se han experimentado fenómenos que han derrotado cualquier intento de gestión, desde el Huracán María hasta el terremoto de 2020, resultando en una falta de atención a las necesidades sociales.

Como consecuencia de este tipo de crisis, las organizaciones de impacto comunitario se ven obligadas a redirigir sus esfuerzos hacia nuevas maneras de alcanzar sus objetivos. Esta problemática, a su vez, se extiende a los individuos que más lo necesitan, dado que encuentran dificultades añadidas para encontrar ayuda en estas circunstancias.

Esta realidad no sólo atañe a las organizaciones sin ánimo de lucro, nos debería afectar a todos como sociedad, a las corporaciones, entidades privadas y gubernamentales.

Los servicios sociales son la herramienta principal para una integración apropiada de las comunicaciones desfavorecidas y de los individuos en desventaja, resultando en un fortalecimiento del tejido social. Frente a este tipo de situaciones los métodos tradicionales parecen no ser suficientes, las organizaciones comunitarias necesitan poder medir datos de forma veraz y con precisión, asegurar la continuidad de sus servicios, atender de forma rápida y eficaz, dedicar más esfuerzo a las personas que a los procesos.

Es por eso que en una sociedad cada vez más activa en el mundo digital, es necesario adquirir nuevas herramientas que sean contemporáneas y relevantes para tener resultados eficientes. La tecnología permite anticiparse al impacto de situaciones futuras, las organizaciones sociales pueden tener información suficiente para no necesitar realizar un estudio de necesidad para cada desastre que pueda suceder, por el contrario, pueden usar la tecnología para tener identificadas las necesidades previamente y, así, atendiendo de antemano los casos y agilizando la ayuda social.

Para beneficiarse de los aspectos digitales mencionados, se debe integrar la tecnología en los procesos de coordinación social. Ejemplos podrían ser:

1. El uso de la tecnología para agilizar la asistencia social
2. La conexión entre el individuo que necesita ayuda y la organización que puede asistirle
3. Conexión entre organizaciones aliadas, de interés o de apoyo
4. Mapas interactivos para ubicar a las organizaciones por región y pueblo
5. Manejo de casos y administración de recursos

Las crisis siempre revelan oportunidades, oportunidades que bien gestionadas resultarán en comunidades más resilientes. La tecnología como inversión social permite visibilidad, transparencia, medición, agilidad y eficiencia en los procesos de atención social. Uniendo esfuerzos

se podrá maximizar el impacto social de todas las organizaciones comunitarias y obtener mejores resultados.

2.2 El impacto de la adopción tecnológica en bancos de alimentos

A la par que la tecnología está en pleno auge, la transformación digital está permitiendo a empresas y organizaciones a mejorar los procesos de innovación y automatización, implicando directamente cambios sustanciales en la manera de ofrecer sus soluciones a la sociedad.

Un ejemplo del potencial que tiene la adopción y desarrollo de las nuevas tecnologías, haciendo más fácil llegar a realizar labores tan importantes para la sociedad, es el de FESBAL (Federación Española de Bancos de Alimentos). Que ha conseguido adoptar un sistema avanzado de gestión que les ha permitido ganar agilidad y eficacia en su enorme actividad, al digitalizar la gestión y coordinación de alimentos.

Este tipo de software hecho a medida, puede ayudar a los economatos a mejorar la eficacia en campos como recursos humanos, logística, financiera y jurídica; además de facilitar trámites y agilizar los acuerdos de donación, colaboración y ayuda.

Veamos en qué estado ha estado el economato para el que se ha desarrollado este proceso, en qué estado está actualmente y en qué estado estará tras la entrega y adopción del software:

El economato no digitalizado : Soporte papel y procesos manuales

- Procesos manuales para todo.
- Formato papel para documentación, facturas y comunicación entre departamentos.
- Uso de calculadoras, tiempos tediosos para seguir expedientes del mes.
- Problemas de espacio y orden para almacenar documentación y facturas.
- Tensión y falta de confianza con aquellos voluntarios que son menos disciplinados.

El economato digitalizado 1.0 : Soporte excel, papel y procesos manuales

- Procesos manuales para todo.
- Formato papel para las facturas y comunicación manual entre departamentos.
- Búsquedas tediosas para el seguimiento de expedientes del mes, un excel por familia al mes.
- Problemas de orden para almacenar los excel y por el uso incorrecto de elementos compartidos en red.
- Uso de calculadoras, errores en datos, de seguimiento y procedimiento. Falta de confianza.

El economato digitalizado 2.0 : Soporte web, papel y procesos automáticos

- Todo proceso de almacenamiento, distribución y cálculo está automatizado.
 - Facturas en formato papel para la familia, los departamentos usan la aplicación.
 - Seguimiento familiar imperceptible, totalmente automatizado y flexible.
 - Almacenamiento y orden imperceptible, totalmente automatizado.
 - Incapacidad de alterar el comportamiento de las restricciones en las compras.
-

2.3 Estudio de mercado

En esta sección vamos a navegar por el mercado actual del software de gestión y planificación de recursos. Las posibilidades de Free Open Source Software y los lenguajes de programación en los que nos tendríamos que mover.

Para llevar a cabo este proyecto necesitamos un software que resuelva los problemas que cualquier negocio tiene. Esto no quiere decir que haya que gastarse cientos de euros en un sistema de planificación de recursos empresariales (ERP - Enterprise resource planning). El mercado de los ERP está dominado por Oracle (2021) y SAP (2021), pero son soluciones tan completas y complejas que necesitan personal especialmente formado para poder usarlos; habiendo incluso titulaciones de diferentes niveles al respecto.

Hoy día las soluciones open source son bastante comunes, en bastantes lenguajes de programación, vivimos en una sociedad cuya comunidad informática desarrolla de forma altruista soluciones genéricas, libres de restricciones comerciales para que cualquiera con pocos o ningún recurso tenga, al menos, la posibilidad de informatizar sus procesos sin necesitar grandes inversiones.

Como puntilla, cabe recalcar que los ERP se consideran un dolor de cabeza para ajustar, configurar y adoptar, hemos de encontrar una solución sencilla.

Artículos como el siguiente (Jutras, 2019) habla sobre cómo adoptar un ERP puede convertir un negocio en algo miserable, haciendo hincapié en cómo el hecho de sobrevalorar la necesidad de implementar ciertas soluciones pueden superar con creces los costes de llevarlo a cabo correctamente e, incluso, acabar fracasando en el intento.

2.3.1 Las bases del proyecto

Hemos de tener en cuenta que este proyecto tiene ciertas especificaciones de base, todo lo que se construya debe ser alrededor de una premisa muy clara. Cuyo resumen sería:

- Low cost
- Adopción sencilla y rápida
- Bajo mantenimiento
- Alta disponibilidad
- Experiencia de usuario simple

El proyecto debe ser lo más barato posible o, si es posible, ser gratuito. En caso de que hayan elementos que cuesten dinero, deberían suponer pagos únicos para poder adquirirlos en un pago y donarlos al economato social.

La burocracia es tan larga y sube tantos niveles que conseguir que se nos acepte un presupuesto puede resultar especialmente tedioso; y además, nada nos asegura que vaya a ser aprobado.

La escalabilidad y el mantenimiento también es una necesidad, el economato para el que realizamos el proyecto no tiene soporte técnico, son los propios voluntarios los que ofrecen sus conocimientos para ayudar e intentar mejorar los procesos y sistemas. Por lo que la solución final debería ser open source, para que cualquiera con conocimientos pudiese aportar al proyecto; haciendo uso de una arquitectura o hardware que necesite poco o ningún mantenimiento, para que cuanto menos interacción humana haya para su funcionamiento, mejor.

2.3.1.1 ERP's y el Free Open Source Software

El mercado de los ERP es amplio y con mucha variedad de soluciones. Una pequeña búsqueda desde el buscador favorito de cada uno devolverá infinidad de listados y comparaciones con votaciones. Sólo podemos estar seguros de una cosa, y es que el Software ERP mueve mucho dinero. Es muy difícil encontrar alguna solución que sea 100% libre y/o sin coste asociado. Aquí podemos ver un ejemplo de listado bastante completo (Quirk, 2019).

Uno de los ERP más conocidos en el mundo del desarrollo es (Odoo, 2021), es Open Source, aunque no de licencia libre ni con capas gratuitas. Este ERP está basado en módulos para hacer configurable su uso y su tarifa y, además, tiene capacidad para que desarrolles Python construyan módulos y los pongan a la venta en su app store, o si no quisieran venderlos, engancharlos a su solución Odoo y no compartirlos con nadie.

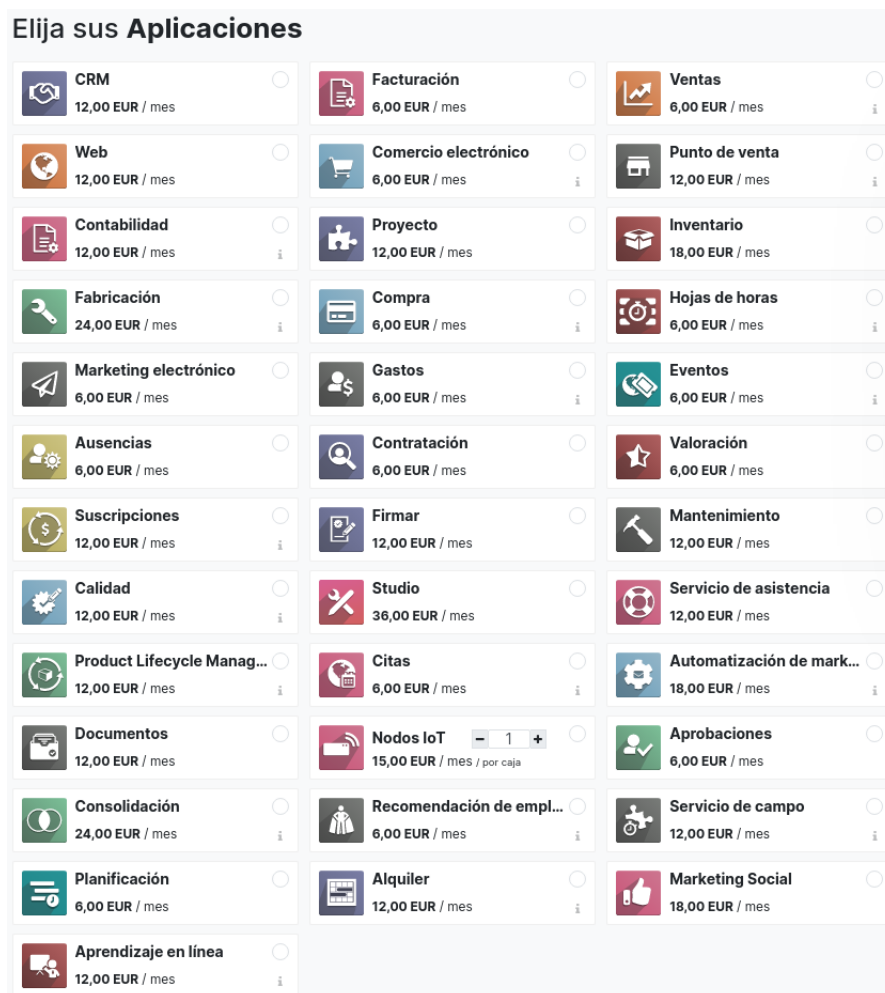


Figura 2.1: Compra de módulos de Odoo

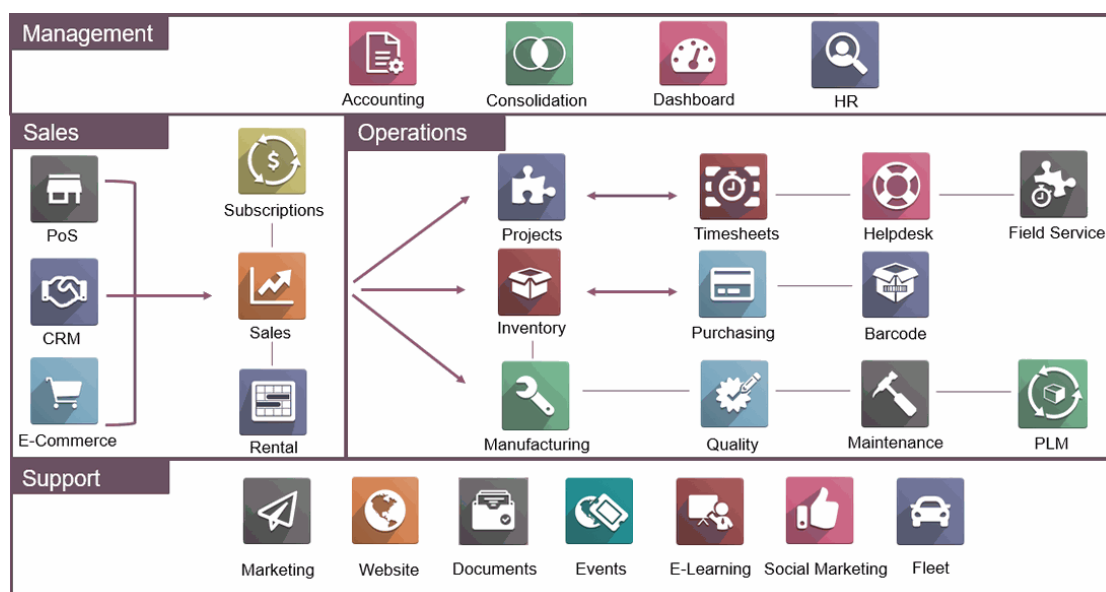


Figura 2.2: Comunicación entre módulos de Odoo

Podríamos listar las siguientes ventajas:

- Bajo coste
- Open source
- Capacidad de operar en la nube
- Inyección de módulo hechos a mano

Y las siguientes desventajas:

- Tiene coste
- Necesita mantenimiento informático avanzado
- Curva de aprendizaje y estudio de módulos para la adopción

A pesar de tener ventajas en la instalación y puesta en marcha frente a otras soluciones como (Oracle, 2021) o (SAP, 2021), y a pesar de ser open source con una comunidad amplia de desarrolladores y con miles de aplicaciones; sigue siendo demasiado complejo para lo que necesitamos, la propia plataforma recomienda contactar con un distribuidor para que ayude al negocio a implantar la solución dado que hacen falta conocimientos informáticos extensos. Recordamos que pretendemos una solución simple de usar y poner en marcha.

Por lo que terminamos por descartar las soluciones ERP que el mercado pone a nuestra disposición. Las necesidades del economato social son muy concretas y decidimos que hacer ad hoc una solución verdaderamente libre nos resuelve el problema que tenemos, además de permitirnos escalar y mantener la aplicación de forma libre si la solución saltase a otros bancos de alimentos.

2.3.2 Decidiendo arquitectura

Dadas las premisas de arquitectura, para este proyecto optamos por estudiar las siguientes posibilidades:

- Raspberry pi y Do It Yourself
- Computación elástica bajo demanda
- Serverless

2.3.2.1 Raspberry pi (DIY)



Figura 2.3: Raspberry Pi 4 Modelo B

La Raspberry Pi fue nuestra primera idea, debido a su condición de software de código abierto, bajo coste de adquisición, reducido espacio y poco consumo.

La Raspberry Pi es una serie de ordenadores de placa reducida de bajo coste desarrollado en el Reino Unido. Su objetivo principal ha sido siempre poner a disposición de todo el mundo el poder de la informática con un ordenador reducido en espacio y coste. Cuando se creó, se pretendía promocionar la enseñanza informática en las escuelas, pero terminó siendo mucho más popular de lo que se esperaba. Como se muestra en la figura 2.3, no trae ningún tipo de periférico, ni carcasa; aunque ya hay packs de todo tipo para ayudar a la adopción del hardware.



Figura 2.4: Raspberry Pi integrada en el interior de un teclado



Figura 2.5: Raspberry Pi protegida con funda

En cuanto a la distribución del hardware, la Raspberry Pi Foundation no indica expresamente que el hardware sea libre o con derechos de marca, aunque sí explican en su web oficial que cualquiera puede convertirse en revendedor o redistribuidor de las tarjetas Raspberry Pi, dando a entender que es un producto con propiedad registrada pero permitiendo el uso libre tanto a nivel educativo como particular.

Su software sí es código abierto, basando su sistema operativo en Debian y llamándose Raspberry Pi OS; el procesador es ARM por lo que la paquetería debianita tiene que pasar un proceso de adaptación y recompilación para poder usarse en la Pi OS.

Todas estas características fueron las que nos incitó a considerarla como una buena solución a lo que se no estaba planteando. Pero el conocimiento de sistemas informáticos iba a ser una carga muy elevada en caso de avería, mantenimiento o escalabilidad. Además, una última idea nos hizo deshacernos de las Pis como solución de hardware, la disponibilidad del software.

Una de nuestras premisas era tener en cuenta la capacidad de salto a otros bancos de alimentos, tanto de la zona, como de la ciudad; haber resuelto el host de la aplicación web en una raspberry pi nos habría planteado problemas de arquitectura mucho más elevados más adelante.

- ¿Dónde deberá localizarse el servidor una vez haya más de una sede haciendo uso?
- ¿Deberíamos distribuir su ejecución por si el nodo principal se queda sin luz?

En seguida caímos en que si bien es una problemática resoluble, teníamos soluciones al alcance de la mano y por muy bajo coste: Los servicios de computación en la nube bajo demanda.

2.3.2.2 Computación elástica bajo demanda (Cloud)

Dado que la arquitectura hardware mantenida por nosotros no iba a ser viable a la larga, pusimos la vista en la computación elástica.

Los servicios Cloud nacen para dar un recurso muy simple de explicar y no tan fácil de proveer, capacidad de procesamiento infinita. Un servicio cloud no es simplemente un software que corre en el ordenador de otro en lugar del propio, es la escalabilidad (casi) sin límites y bajo demanda, lo que le da potencia como idea. Hoy en día hay múltiples proveedores de servicios cloud, aunque los jugadores más fuertes son indiscutiblemente Aws, Azure y Digital Ocean.



Dado que yo tengo experiencia laboral y personal como arquitecto cloud en servicios aws y me estoy preparando el examen oficial, decidimos sin pensárnoslo mucho en estudiar nuestra solución como nativa cloud haciendo uso de los servicios Aws. Si bien Aws tiene capas gratuitas, éstas son temporales en caso de la computación y son circulares en caso de la transmisión de datos. Dado nuestro cliente y nuestro estudio de casos de uso, sabíamos que podríamos usar las capas gratuitas de transferencia de datos sin problema, pero el hosting sí podría suponer una traba. Aún así, proseguimos.

En concreto, con Aws, usaríamos una de las máquinas de computación elástica más pequeña que tiene, para montar nuestro servidor web, haciendo uso de imágenes dockerizadas y distribuidas por el propio registro de imágenes de Aws. En todo momento hemos querido mantener el estándar y el open source, esto facilitaría la recepción de voluntarios informáticos. Las máquinas que Aws provee, servicio llamado (EC2-AWS, 2021), es descrito como ‘Capacidad informática segura y de tamaño ajustable que admite prácticamente cualquier carga de trabajo’.

La parte buena de usar este servicio es la capacidad de usar sus servicios bajo demanda, podemos hacer crecer y disminuir máquinas conforme necesitemos, balancear la carga entre varios nodos si quisiéramos o incluso aumentar nuestro porcentaje de disponibilidad usando la Route 53 para que si una región deja de estar disponible, automáticamente el tráfico quedase redirigido a la región más cercana. Habríamos usado una base de datos adhoc RDS para evitar configuración y mantenimiento, teniéndolo todo centralizado en una misma red para minimizar la espera por latencia y habríamos protegido el acceso y ddos de nuestra máquina usando el servicio de ApiGateway. El frontend podría ser desatendido, almacenado en un S3 de bajo coste y distribuido y cacheado por el servicio de CloudFront.

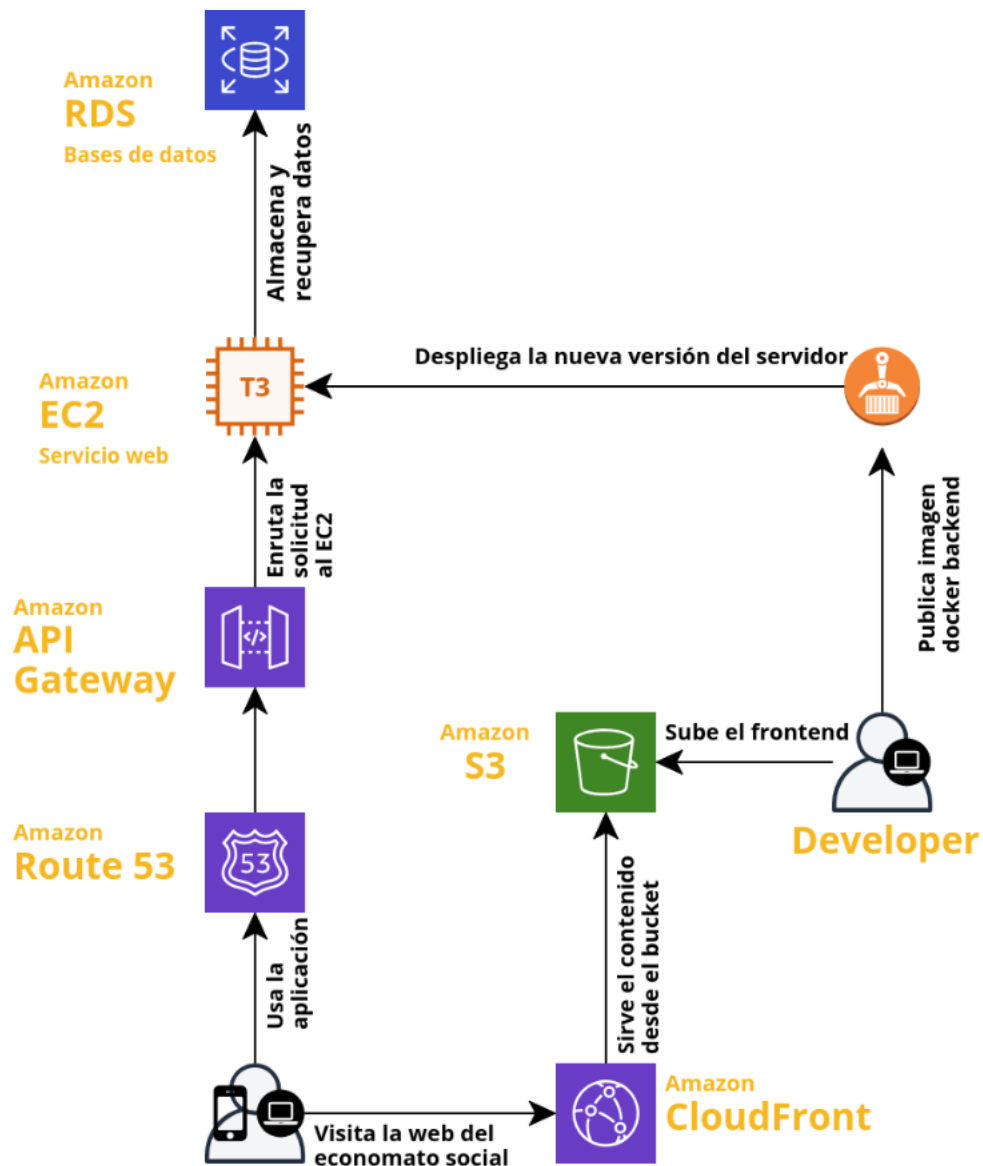







Figura 2.6: Arquitectura simplificada

Presupuesto Economato Social 	
Compute	€6,25 / mo 
Storage	€0,09 / mo 
Database	€11,42 / mo 
App services	€4,82 / mo 
<hr/>	
Total	€22,59 / mo

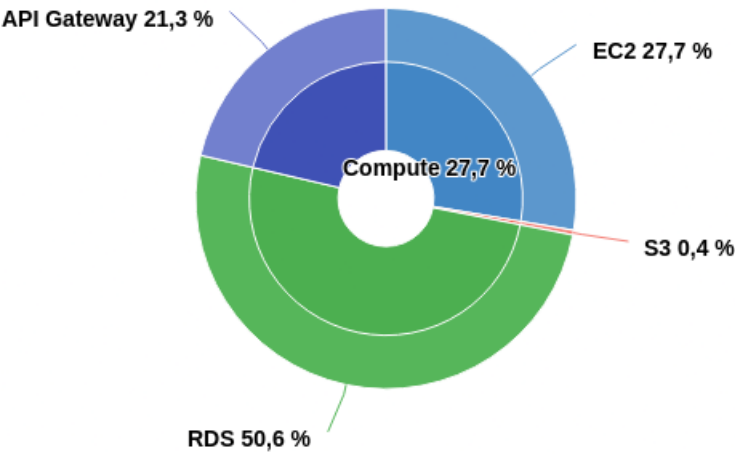


Figura 2.7: Presupuesto simplificado

Todo esto hace de la arquitectura un proyecto ideal, una estructura sencilla de manejar para alguien con conocimientos, pero aunque hubiésemos podido aceptar el presupuesto, volvimos a toparnos la misma problemática que con la Raspberry Pi. ¿Qué ocurriría si una máquina deja de estar accesible? ¿Y si hay un problema en la configuración de red o de acceso por grupos de seguridad? ¿Y si alguien toca sin querer algo que no debe en un despliegue? Nuestro objetivo de no ceder el mantenimiento a la posibilidad de tener siempre conocimientos informáticos disponibles en el voluntariado, era más fuerte que intentar idear una arquitectura digna de proyectos modernos.

Sólo nos quedaba una carta que jugar, una forma de explotar el cloud que ya había usado en proyectos anteriores, y que podría ser lo que nos permitiese centrar el conocimiento informático voluntario en el desarrollo y mantenimiento de la aplicación; permitiéndonos ignorar la disponibilidad del servicio y configuración de las máquinas: el Serverless.

2.3.2.3 Serverless (Cloud)

Como ya hemos ido hablando, el sector privado tiende cada vez hacia la nube, sobretodo el sector comercial; dada la capacidad de contratar y configurar soluciones a medida de las necesidades de cada uno. Dentro de la oferta de servicios cloud existe el uno llamado Serverless Computing, una de las prácticas más recientes, y comúnmente conocido como FaaS (Function as a Service).

El primer concepto que nos atrajo para adoptar esta solución es la propia descripción del Serverless Computing o Arquitectura Serverless. Éste viene a ser un modelo que permite a los usuarios crear y ejecutar aplicaciones y procesos sin entrar en contacto con el servidor subyacente. Dado que es justo lo que queremos, un entorno en la nube en el que es el propio proveedor el que se ocupa del suministro, gestión y escalado del servicio. Ésto además tiene un punto fuerte para nuestra intención de adopción futura, podemos centrar toda nuestra atención y esfuerzo en el desarrollo y en la ejecución del software, ahorrando tareas y conocimientos al equipo implicado.

También, normalmente los proveedores de Serverless no sólo son responsables de que los recursos de servidor estén siempre disponibles, si no también lo son de garantizar la disponibilidad del servicio; es decir, seguridad anticáida. Normalmente este tipo de servicios tienen un modelo de pago por uso, el cual abordaremos después.

2.3.2.4 ¿Cómo funciona?

En una infraestructura serverless, la gestión del hardware por parte del proveedor es esencial. El único desafío con el que tienen que lidiar los usuarios es integrar su software o su lógica, incluidas las funciones adecuadas, en el espacio alquilado en la nube. El acceso a estas funciones se puede realizar de dos maneras:

- de forma asíncrona, a través de eventos
- de forma síncrona, según el modelo Cliente-Servidor clásico

La forma de uso asíncrona ofrece la ventaja de evitar acoplamiento entre las funciones y mantener la demanda de recursos en un nivel bajo durante la ejecución. Un ejemplo de función asíncrona sería que al cargar una imagen cree también un icono en miniatura. Son funciones atómicas, que permiten mucho control en el código, simplicidad y reducción de gastos, dado que sólo se paga cuando se usa; problemas de optimización también son más fáciles de acotar y se puede decidir poner recursos a optimizar una función que funcione deficientemente pero no necesariamente a todo el sistema.

El uso que nosotros vamos a darle al Serverless es la función síncrona, vamos a usar el servicio para poder mantener levantado un servidor web sin tener que estar pendientes de mantener el sistema, actualizarlo o configurarlo. En concreto, vamos a usar la solución que provee Heroku, un proveedor de servicios Serverless con una generosa capa gratuita que a éste trabajo le vendrá de perlas. La única limitación que pueda importarnos, de las que tiene Heroku en su capa gratuita, es que los servicios mueren tras 5 minutos sin uso; haciendo que

la siguiente petición tarde un poco más. Esto encaja a la perfección con nuestras necesidades pues, a priori, se le va a dar uso al sistema un par de días a la semana.

A diferencia de una infraestructura de plataforma como servicio, el proveedor Serverless no facilita para ello un entorno de trabajo duradero para todo el tiempo de trabajo, si no que aporta de manera puntual y en tiempo real aquellos recursos que se requieren durante el tiempo de ejecución de la llamada a la función. A pesar de llamarse Serverless, obviamente sí hay servidores detrás del telón, aunque el usuario nunca lo perciba.

2.3.2.5 Vista general de ventajas y desventajas

Las infraestructuras convencionales en la nube permiten al usuario administrar y eliminar el hardware que se requiere pero, a menudo, ésto requiere grandes esfuerzos a nivel de administración y microgestión. Lo que pretende el Serverless es minimizar eso al máximo.

Podríamos considerar las siguientes ventajas y desventajas

- Ventajas
 - Escalamiento y administración de los recursos necesarios por parte del proveedor
 - Suministro ágil de los recursos en tiempo real, en función del consumo en cada momento
 - Cobro únicamente por uso al milisegundo
 - Gran tolerancia a errores gracias a la infraestructura flexible de hardware
- Desventajas
 - Acceso restringido a los sistemas e incapacidad de configuración de éstos
 - Grandes proyectos basados en funciones asíncronas pueden suponer un gran esfuerzo de implementación
 - Gran dependencia del proveedor, posibles dificultades de migración
 - Procesos de monitorización y depuración de errores más complejos

2.3.2.6 Cuándo se usa el principio Serverless

El serverless computing ha sido concebido principalmente para el intercambio efímero de datos de aplicaciones web y de negocios en la nube, aunque acabó abarcando servicios más complejos y con una vida de ejecución más larga pero basándose en las mismas premisas: escalabilidad elástica en tiempo real y adecuándose al uso, sin que el usuario tenga que hacerse cargo de configuraciones de sistemas y redes.

Los escenarios más comunes del Serverless son los siguientes:

- Proxy API: muchas aplicaciones comerciales antiguas cuenta con interfaces complejas y lentas. Mediante el serverless se puede crear una capa de abstracción alternativa para poder acceder a estas aplicaciones mediante una API REST mucho más simple
-

- Serverless Backend: el serverless computing también se usa cada vez más para construir y soportar todo el backend de una aplicación en la nube. Backend as a Service.
- Procesamiento de datos no estructurados: hoy en día, es imposible imaginar un entorno de negocios sin big data. En este contexto, el serverless se muestra como un gran aliado para procesar este tipo de información.
- Ejecución de tareas según horario: una gran cantidad de demanda de este tipo de funciones en un horario definido, un disparador que ejecuta alguna automatización aislada como backups de bases de datos, reorganización de datos, etc.
- Aplicación de asistentes y chat bots: Normalmente este tipo de aplicaciones son sin estado y conectan con servicios de modelos inteligentes más complejos, estas interfaces en serverless están en auge.

2.3.2.7 Ejemplos de uso con heroku

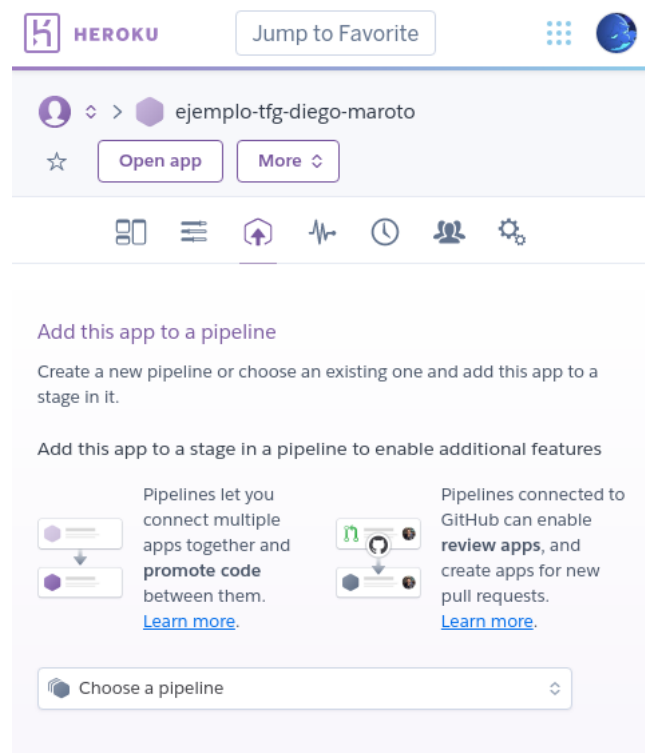


Figura 2.8: Configurando proyecto en Heroku

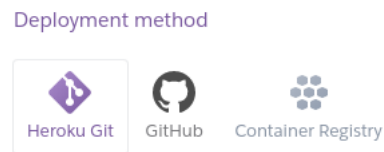


Figura 2.9: CI/CD en Heroku

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku CLI

Download and install the [Heroku CLI](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/  
$ git init  
$ heroku git:remote -a ejemplo-tfg-diego-maroto
```

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .  
$ git commit -am "make it better"  
$ git push heroku master
```



You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).

Existing Git repository

For existing repositories, simply add the `heroku` remote

```
$ heroku git:remote -a ejemplo-tfg-diego-maroto
```

Figura 2.10: Ejemplo de uso de Heroku por línea de comandos

2.3.3 Stack FrontEnd/BackEend

Una de nuestras principales preocupaciones, como se lleva diciendo toda la exposición, es la escalabilidad del proyecto en cuanto a la facilidad de colaboración técnica voluntaria. Desde esta premisa estuvimos estudiando qué lenguajes de programación podrían encajar en esta dinámica.

Del mundo del desarrollo, la plataforma más usada de largo es Stack Overflow; en esencia es un foro tecnológico donde usuarios escriben sus dudas y otros usuarios intentan resolverlas. Stack Overflow hace una encuesta todos los años para extraer datos demográficos, económicos y tecnológicos sobre la comunidad tecnológica mundial. Decidimos basarnos en la encuesta de 2020 (survey StackOverflow, 2020) para valorar el stack a usar.

Como se aprecia en la encuesta, javascript es el más usado tanto profesional como amateur; además, encaja en nuestra idea inicial, en la que considerábamos usar el mismo lenguaje para frontend y backend una ventaja competitiva. Dado que va a ser un proyecto dilatado en el tiempo, un lenguaje de tipado estático sería una buena forma de agilizar la curva de aprendizaje del proyecto y recortar la barrera de entrada para los aspirantes; así pues, dado que Typescript está en el top 10 y podemos considerarlo javascript, optamos por usarlo.

Una vez decidido el lenguaje, queda la decisión de si usar frameworks o no. No usar frameworks puede ser potente en un inicio pues permite la flexibilidad de montar el sistema como se quiera y definir unas pautas; unas pautas que preferiríamos que pudiesen ponerse en tela de juicio lo menos posible, así que optaremos por frameworks pues así la forma de hacer las cosas será siempre la misma independientemente de quién trabaje en el proyecto.

En javascript es muy común el stack MEAN (Mongo, Express, Angular y Nodejs), y para acomodar el backend y el frontend, usaremos para el backend con Express el framework Nestjs; un framework disponible en typescript cuya estructura MVC es muy parecida a Angular, de esta forma, un desarrollador frontend podrá pivotar al backend y viceversa con muy poco problema.

Para la persistencia de datos se había propuesto MongoDB desde un inicio, y éste encaja con Javascript a la perfección, dado que los documentos de ambos se anotan en JSON, la integración entre ambos es intuitiva y sencilla. Además, MongoDB tiene capa gratuita en su servicio cloud Compass y es justo lo que queremos.

Y de esta forma queda decidido, se usará Typescript con Nestjs para el backend, Typescript con Angular para el frontend y MongoDB para la persistencia de datos.

3 Metodología

En este apartado se van a describir las herramientas necesarias para el correcto desarrollo del proyecto. Éstas herramientas están comprendidas en lo que se puede llamar ingeniería del software, disciplina que comprende análisis del problema, estudio de la solución, el diseño del proyecto, el desarrollo del software, las pruebas que faciliten la integración continua y la adopción del sistema. Éste proyecto se ha llevado a cabo mediante etapas como las detalladas a continuación:

3.1 Requisitos mínimos

Los requisitos mínimos que se deben cumplir para poder desarrollar el proyecto, probarlo y ponerlo en marcha son los siguientes:

- Pc, smartphone o tablet de cualquier sistema operativo que posea un navegador moderno
- Si Pc, lector de código de barras independiente por usb o webcam
- Si smartphone o tablet, cámara integrada para poder leer códigos de barras
- Conexión estable a internet

3.2 Fases del desarrollo

Aquí se van a nombrar y explicar de forma introductoria las fases de desarrollo que se han seguido para llevar a cabo el proyecto:

- Análisis de requisitos: es la primera etapa en la que se identificarán los conceptos importantes sobre el proyecto, tales como las necesidades del proyecto, las funcionalidades requeridas y los requisitos para su buen funcionamiento
- Diseño del proyecto: en esta etapa se redactan los casos de uso y se realizan los mockups para una primera impresión visual de la aplicación
- Desarrollo del software: en esta etapa se realiza la aplicación usando las etapas anteriores como guía
- Pruebas: en esta etapa, que va muy ligada al desarrollo del software, se preparan pruebas automatizadas que permitan cerciorarse de que las funcionalidades desarrolladas efectivamente funcionan, y lo siguen haciendo tras refactorizaciones o nuevos desarrollos en la aplicación.

3.3 Metodología ágil utilizada

En este proyecto se ha seguido la metodología ágil denominada SCRUM. Ésta se define como un proceso de gestión que reduce la complejidad en el desarrollo de productos para satisfacer las necesidades de los clientes.

En este proyecto en particular, se ha desarrollado el software mediante sprints, sprints review y milestones, que se detallan a continuación:

Sprint: Etapa de 3 semanas en las que se desarrollan los objetivos establecidos para el mismo

Sprint review: Reunión en la que se realiza una revisión de lo desarrollado, bloqueos, problemas y soluciones, además de una retrospectiva y planificación del siguiente.

Milestone: Conjunto de objetivos de desarrollo que marcan metas en el progreso. Se han usado junto a fechas, para poder estimar y considerar el alcance en función del tiempo disponible.

Gracias a la utilización de metodologías ágiles es posible obtener grandes beneficios como indica la empresa (Bravent, 2019) en un artículo:

- Satisfacción del cliente a través de la entrega temprana y continua del software de valor.
 - Proximidad del cliente y constante iteración con él, es parte del equipo y está presente en la toma de decisiones.
 - Gestión regular de las expectativas del cliente y basada en resultados tangibles.
 - Resultados anticipados (time to market).
 - Flexibilidad y adaptación respecto a las necesidades del cliente, cambios en el mercado, etc.
 - Gestión sistemática del Retorno de la Inversión (ROI).
 - Capacidad para abordar los requisitos cambiantes, incluso si llegan tarde en el proceso de desarrollo.
 - Equipo implicado y motivado ya que pueden usar su creatividad para resolver problemas y pueden decidir organizar su trabajo.
 - Auto-superación: de forma periódica se evalúa el producto que se está desarrollando
 - Priorizar los requerimientos de acuerdo a su valor.
 - Se proporciona la mínima funcionalidad, de forma que solo se desarrolla lo necesario. Evita escribir código innecesario.
 - Calidad del producto obtenido. El software que funciona es la principal medida del progreso.
-

- Pruebas continuas durante todo el desarrollo.
- Mejora la productividad y el control del tiempo requerido para realizar el proyecto.
- Permite dividir el trabajo en módulos minimizando los fallos y el coste.
- Si surge cualquier error, se sabe rápido, disminuyendo riesgos.
- Permite solucionar rápidamente los problemas que impiden que los equipos progresen.

3.4 Herramientas hardware

Este apartado trata de los dispositivos de hardware necesarios para el buen desarrollo del proyecto, a excepción del ordenador utilizado para realizar el proyecto, del cual se asume que es obligatorio pero no exclusivo.

Dado que una de las necesidades del proyecto es leer códigos de barras, podemos hacerlo de dos formas en función de qué dispositivos tengamos a mano:

- Lector manual de código de barras por usb, al usar Pc
 - Es la forma más cómoda de leer un código de barras cuando alguien trae una donación al banco de alimentos. Es la forma más fiable, teniendo el dispositivo a mano, sin importar el tamaño del producto, el tamaño del código de barras ni si es transparente o con poco contraste de colores.
- Webcam al usar Pc
 - Es un poco más tedioso que el anterior, se debe coger el producto y sostenerlo delante de la webcam, que en función de la calidad de ésta no enfoque bien automáticamente. Sin contar que la calidad y contrastes de la impresión del código de barras es un factor que puede hacerlo ilegible frente a una cámara.
- Cámara trasera del dispositivo móvil
 - Es un poco más tedioso que el hardware adhoc, normalmente los voluntarios no estarán usando el móvil, deben hacer login expresamente para esto e ir a la edición del producto para añadir el EAN. Sin contar, como en la anterior, que la calidad y contrastes de la impresión del código de barras es un factor que puede hacerlo ilegible frente a una cámara.

3.5 Herramientas software

En este apartado se va a realizar una descripción del propósito de cada una de las herramientas de software utilizadas y que han sido necesarias para la realización del proyecto.

3.5.1 Documentación

En este apartado se define el software necesario para crear la documentación y memoria, y distribuirla.

- Dropbox: Gestor de archivos en la nube, permite documentos simples y colaborativos. Los primeros bocetos de las ideas a desarrollar fueron en esta plataforma.
- Overleaf: Gestor de documentos \LaTeX en la nube. La memoria se ha creado completamente en \LaTeX y online
- Github: Control de versiones en la nube, las versiones de la memoria se han ido publicando en el repositorio que contiene el proyecto.

3.5.2 Planificación y cumplimiento de las etapas

En este apartado se va a definir el software necesario para poder realizar una planificación temporal del proyecto, definiendo etapas y marcas temporales y una estructuración de tareas.

- Github: Panel de proyecto. Github contiene un panel en los repositorios que permite crear y manejar proyectos, al estilo de un tablero canvan. Las tareas se pueden mover de forma automática siendo disparadas por commits o pull requests.
- Github: Issues. Github permite listar y crear tareas, dándole etiquetas y relacionándolas con proyectos y milestones para acomodar su gestión.
- Github: Milestones. Github permite crear milestones con fecha de inicio y fin. Las milestones pueden estar relacionadas con proyectos y contener tantas issues como se crea conveniente en la planificación.

3.5.3 Desarrollo

En este apartado se va a comentar el software que se ha utilizado para la realización del backend y el frontend dentro del proyecto.

- Git: Control de versiones que permite conectar con gestores remotos, en este caso Github.
 - Heroku-cli: Control de Continuous Delivery integrado con Git, permite la entrega continua y el despliegue automático completamente integrado con el uso de las ramas del proyecto.
 - Docker: Se ha usado docker para virtualizar MongoDB en local y así desarrollar y lanzar las pruebas automáticas contra la máquina local y no afectar a los datos de producción en compass.
 - Jest: Librería de testing para javascript que facilita muchísimo el uso de mocks, stubs y espiarlos.
 - Yarn: Gestor de paquetería para javascript, facilita el manejo de dependencias del proyecto.
-

- WebStorm: IDE de JetBrains, contiene todo lo necesario para desarrollar javascript, nodejs y typescript. Trae integración con git para facilitar el uso del control de versiones, comparación de ramas, secciones de código, code blame y resolución de conflictos; además integra gestión de baterías de testing con Jest.
-

4 Estudio viabilidad

En este punto del proyecto se va a presentar una planificación temporal del proyecto, la planificación de los recursos y requisitos, además de una planificación de riesgos.

4.1 Planificación temporal

Para llegar a buen puerto en un proyecto es necesario intentar una buena estimación, que sabiendo que no es tarea fácil y difícil de cumplir, teniendo fechas de entrega continua al cliente podemos establecer prioridades y tomar decisiones estructurales en función de las necesidades temporales y de entrega. Como puede ser, por ejemplo, la entrada de nuevos requisitos que hacen retroceder otras funcionalidades en prioridad o que, incluso, las hacen quedar fuera del backlog; o que por problemas técnicos o por dificultad de implementación una tarea se retrase tanto que arrastre otras relacionadas.

Cada hito realizado se asemeja bastante a las fases de la ingeniería de software y, a grandes rasgos, son los siguientes:

- Hito 0 - Análisis: En este primer hito se realiza la primera ronda de contacto con los temas relacionados del proyecto. En este proyecto han sido el estudio de la disponibilidad web low-cost y acercarnos al banco de alimentos a ver en vivo el trabajo y necesidades de los voluntarios.
- Hito 1 - Especificación: En este hito se realizan los estudios y la documentación tanto del estado del arte como de las herramientas a utilizar. Además, se especifican los requisitos, los riesgos y se documentan los casos de uso y los diagramas.
- Hito 2 - Fase de desarrollo: Éste es el hito más largo y complicado, consiste en realizar el desarrollo de la aplicación backend, diseño e implementación de bases de datos y desarrollo del frontend.
- Hito 3 - Fase de pruebas: Último hito, en este se realizan pruebas de integración y satisfacción además de revisar y finalizar la memoria.

En este proyecto la planificación temporal empezó una vez llegados al hito 2, se ha usado la funcionalidad de Canvan, Issues y Milestone que proporciona Github en su plataforma.

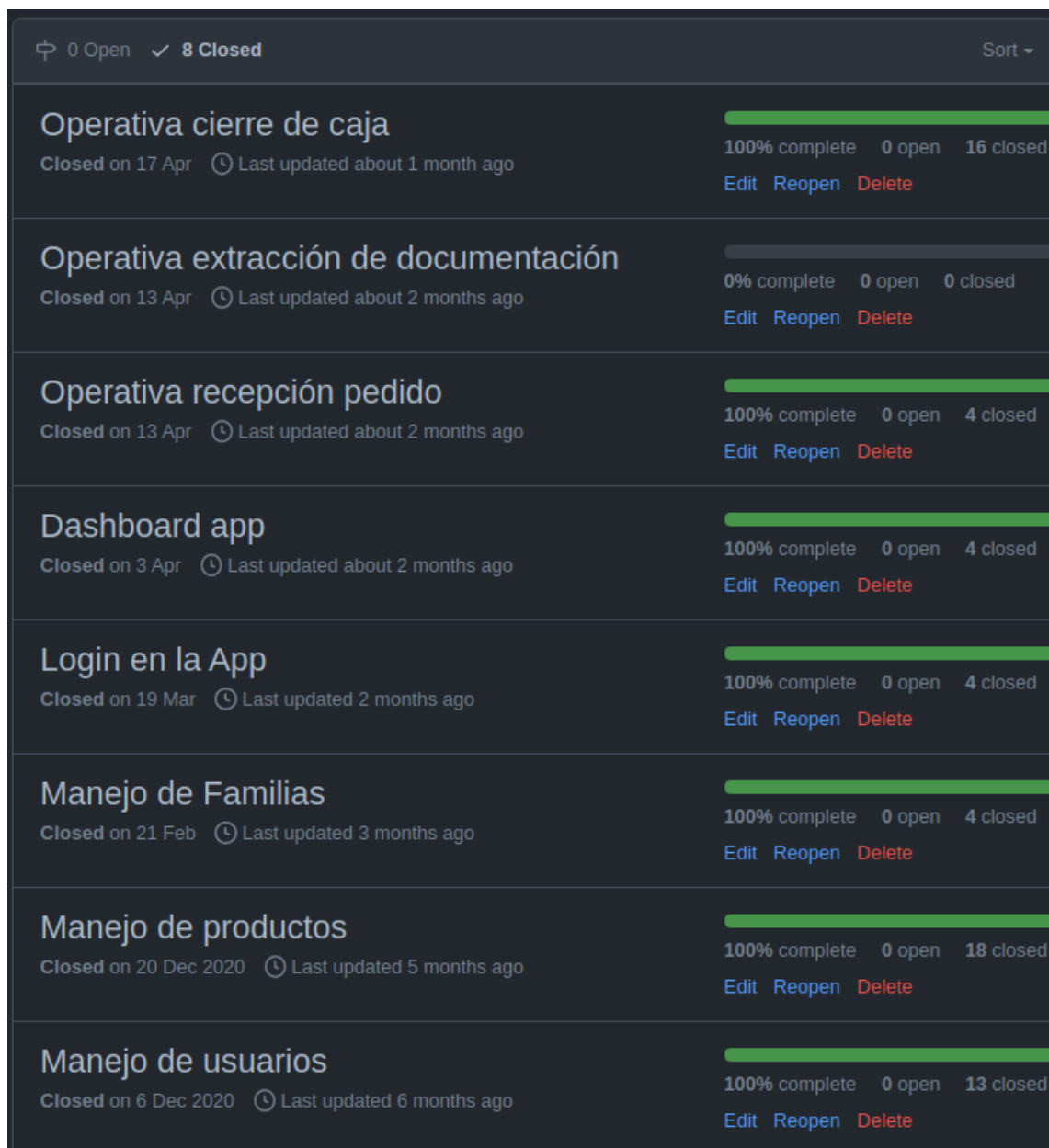


Figura 4.1: Milestones cerradas en Github

4.2 Gestión de riesgos

Una de los aspectos más importantes de la planificación de un proyecto es una buena prevención de los riesgos y contratiempos que se puedan ocasionar durante la realización del proyecto. Tanto si deben ser mitigables sin falta o si se pueden asumir. Para ello se documentan y se establecen pautas que especifiquen cómo se resuelve cada riesgo, éstos se identifican de la siguiente manera:

- Identificador: código de identificación
- Nombre: Palabra o conjunto de palabras breves y descriptivas
- Probabilidad: posibilidad de que ocurra dicho riesgo
- Consecuencia: nivel de importancia de que ocurra el riesgo
- Estrategia: pautas a seguir para resolver el riesgo

Identificador	R1
Nombre	No disponer de hosting
Tipo	Tecnología
Probabilidad	Media
Efectos	Serio
Estrategia	Obtener presupuestos

Identificador	R2
Nombre	Caída del hosting
Tipo	Tecnología
Probabilidad	Baja
Efectos	Serio
Estrategia	Obtener presupuestos alta disponibilidad o esperar a su recuperación

Identificador	R3
Nombre	Conocimientos innecesarios
Tipo	Persona
Probabilidad	Media
Efectos	Serio
Estrategia	Revisar documentación y vídeo tutoriales entregados

Identificador	R4
Nombre	Acumulación de tareas
Tipo	Organizacional
Probabilidad	Media
Efectos	Serio
Estrategia	Buena planificación y gestión de tiempos

Identificador	R5
Nombre	Licencias de herramientas
Tipo	Herramientas
Probabilidad	Media
Efectos	Tolerable
Estrategia	Asumir costes o adoptar herramientas libres

Identificador	R6
Nombre	Bugs
Tipo	Herramientas
Probabilidad	Media
Efectos	Serio
Estrategia	Seguir metodología para arreglar y entregar rápido

Identificador	R7
Nombre	Nueva funcionalidad
Tipo	Requerimientos
Probabilidad	Alta
Efectos	Serio
Estrategia	Seguir metodología para desarrollar y entregar rápido

Identificador	R8
Nombre	Cambios en las bases de datos
Tipo	Requerimientos
Probabilidad	Alta
Efectos	Tolerable
Estrategia	Mongodb minimiza el impacto por su flexibilidad

Identificador	R9
Nombre	Pérdida de trabajo
Tipo	Herramientas
Probabilidad	Baja
Efectos	Serio
Estrategia	Hacer copias de seguridad de forma recurrente

Identificador	R10
Nombre	Modificaciones en el frontend
Tipo	Requerimientos
Probabilidad	Alta
Efectos	Tolerable
Estrategia	Seguir metodología para desarrollar y entregar rápido

Identificador	R11
Nombre	Infraestimar tiempo de desarrollo
Tipo	Estimación
Probabilidad	Media
Efectos	Serio
Estrategia	Organizar tareas con margen suficiente

Identificador	R12
Nombre	Falta de realización de alguna parte del proyecto
Tipo	Estimación
Probabilidad	Media
Efectos	Serio
Estrategia	Documentar funcionalidades pendientes y continuar desarrollo

5 Especificación de requisitos

En esta sección se van a detallar los objetivos del sistema para, a partir de ahí, obtener los casos de uso y así los requisitos del proyecto.

5.1 Objetivos del sistema

Para poder comenzar a establecer los requisitos se deben establecer los objetivos que tiene el sistema, para ello vamos a mostrarlo en formato de tabla en la que se detalla cada objetivo con su descripción y prioridad.

Identificador	O1
Nombre	Gestión de usuarios
Descripción	Se debe ser capaz de crear y loguear usuarios
Prioridad	Alta

Identificador	O2
Nombre	Gestión de almacenes
Descripción	Se debe ser capaz de crear y editar almacenes
Prioridad	Alta

Identificador	O3
Nombre	Gestión de productos
Descripción	Se debe ser capaz de crear y editar productos
Prioridad	Alta

Identificador	O4
Nombre	Gestión de roles de usuario
Descripción	Se debe diferenciar permisos por roles de usuario
Prioridad	Alta

Identificador	O5
Nombre	Gestión de familias
Descripción	Se deben recordar las familias por limitaciones en el mismo mes
Prioridad	Alta

Identificador	O6
Nombre	Recepción de pedidos de compra
Descripción	Se debe gestionar el proceso de compra con la memoria por familia
Prioridad	Alta

Identificador	O7
Nombre	Gestión de cobros y caja
Descripción	Se deben poder marcar pedidos como cobrados o erróneos
Prioridad	Alta

Identificador	O8
Nombre	Gestión de expedición de pedidos
Descripción	Se deben poder ver pedidos a entregar y marcarlos entregados
Prioridad	Alta

Identificador	O9
Nombre	Dashboard con estadísticas
Descripción	Se debe añadir información relevante de forma resumida y legible
Prioridad	Media

Identificador	O10
Nombre	Imprimir facturas
Descripción	Se deben mostrar las facturas maquetadas para imprimirlas
Prioridad	Alta

Identificador	O11
Nombre	Anonimización de facturas
Descripción	Se debe poder desvincular facturas de las familias
Prioridad	Media

Identificador	O12
Nombre	Productos con información nutricional
Descripción	Se debe tener información automatizada de los productos
Prioridad	Baja

Identificador	O13
Nombre	Leer EAN de los productos
Descripción	Se debe poder vincular EAN al producto mediante cámara o lector
Prioridad	Baja

5.2 Casos de uso

Los casos de uso sirven para ilustrar el comportamiento del sistema con un agente externo. Permite ver el alcance y límites del sistema y su relación con el entorno. Para poder realizar los casos de uso se deben identificar los agentes externos.

En este sistema hay varios agentes, éstos son:

- Usuario con uno o más roles:
 - Recepción pedidos
 - * Búsqueda de familias
 - * Recepción de pedidos y visualización de facturas
 - Caja
 - * Gestión de caja y facturas
 - Almacén
 - * Expedición de pedidos
 - * Gestión de productos, añadir, editar y borrar
 - Administrador
 - * Todo lo anterior
 - * Añadir usuarios y editarlos
 - * Activar y desactivar usuarios
 - * Dar o quitar roles
 - * Anonimizar facturas
 - Usuario súper administrador
 - Igual que administrador pero no puede perder su rol
 - Se deben crear a mano por un desarrollador con acceso al sistema
-

Ahora se van a clasificar cada caso de uso de una forma similar a los objetivos, con los siguientes datos: identificador, nombre, descripción, flujo básico, flujos alternos, pre-condiciones, post-condiciones.

Identificador	CU1
Nombre	Iniciar
Descripción	Iniciar sesión
Pre-condiciones	Debe estar registrado
Flujo básico	Carga la página principal, escribe sus datos en el formulario y envía el formulario
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none"> • Login correcto: Se muestra el dashboard • Login incorrecto: Se muestra mensaje de datos incorrectos

Identificador	CU2
Nombre	Crear usuario
Descripción	Permite al administrador crear un usuario nuevo
Pre-condiciones	Debe tener rol Administrador
Flujo básico	Login correcto y navega a creación de usuarios. Introduce los datos nuevos y pulsa crear.
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none"> • Crear correcto: Se muestra mensaje verde de confirmación • Crear incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU3
Nombre	Editar usuario
Descripción	Permite al administrador cambiar los datos del usuario
Pre-condiciones	Debe tener rol Administrador
Flujo básico	Login correcto y navega al detalle de un usuario. Cambia los datos y pulsa guardar.
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none"> • Edición correcto: Se muestra mensaje verde de confirmación • Edición incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU4
Nombre	Buscar familia
Descripción	Permite al usuario buscar familia por identificación o por expediente
Pre-condiciones	Debe tener rol Pedidos
Flujo básico	Login correcto y navega a pedidos y usa el formulario de búsqueda de familias
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none"> • Búsqueda con coincidencias: Se muestra en la página de pedidos información sobre la familia ese mes • Búsqueda sin coincidencias: En la página de pedidos no aparece información sobre la familia ese mes

Identificador	CU5
Nombre	Crear pedido
Descripción	Permite al usuario crear un pedido nuevo teniendo en cuenta limitaciones del expediente
Pre-condiciones	Debe tener rol Pedidos
Flujo básico	Login correcto y navega a pedidos, usa el formulario de búsqueda de familias y comienza el proceso de pedido hasta la generación de la factura
Flujo alternativo	-
Post-condiciones	-

Identificador	CU6
Nombre	Imprimir factura
Descripción	Permite al usuario ver la factura maquetada para impresión
Pre-condiciones	Debe estar registrado
Flujo básico	Login correcto y en la tabla de facturas del día, click en el ojo de la que quiera ver
Flujo alternativo	Flujo de pedido y al dar click a generar factura se abre una pestaña nueva con ésta lista para imprimir
Post-condiciones	-

Identificador	CU7
Nombre	Crear almacén
Descripción	Permite al usuario crear un almacén nuevo para separar stocks, facturas y expedición
Pre-condiciones	Debe tener rol Administrador
Flujo básico	Login correcto y navega a gestión de almacenes, pulsa crear y completa el formulario de creación
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none">• Crear correcto: Se muestra mensaje verde de confirmación• Crear incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU8
Nombre	Editar almacén
Descripción	Permite al usuario cambiar los datos del almacén
Pre-condiciones	Debe tener rol Administrador
Flujo básico	Login correcto y navega a gestión de almacenes, pulsa editar el almacén de la lista y edita los datos en el formulario
Flujo alternativo	-
Post-condiciones	<ul style="list-style-type: none">• Editar correcto: Se muestra mensaje verde de confirmación• Editar incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU9
Nombre	Crear producto
Descripción	Permite al usuario crear un producto
Pre-condiciones	Debe tener rol Almacén
Flujo básico	Login correcto y navega a gestión de productos, pulsa crear y completa el formulario de creación
Flujo alterno	Login correcto y navega a gestión de productos, pulsa crear y completa el formulario de creación, además usa la webcam para leer el código EAN y se recupera la información nutricional pidiéndola a OpenFoodFacts
Post-condiciones	<ul style="list-style-type: none">• Crear correcto: Se muestra mensaje verde de confirmación• Crear incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU10
Nombre	Editar producto
Descripción	Permite al usuario editar un producto
Pre-condiciones	Debe tener rol Almacén
Flujo básico	Login correcto y navega a gestión de productos, pulsa editar en el producto que quiera de la lista y edita los datos en el formulario
Flujo alterno	-
Post-condiciones	<ul style="list-style-type: none">• Editar correcto: Se muestra mensaje verde de confirmación• Editar incorrecto: Se muestra mensaje rojo con información del error

Identificador	CU11
Nombre	Cobrar un pedido
Descripción	Permite al usuario marcar un pedido pendiente como cobrado
Pre-condiciones	Debe tener rol Caja
Flujo básico	Login correcto y navega a caja, pulsa en cobrar la factura que considere
Flujo alterno	-
Post-condiciones	-

Identificador	CU12
Nombre	Rechazar un pedido
Descripción	Permite al usuario marcar un pedido pendiente como anulado
Pre-condiciones	Debe tener rol Caja
Flujo básico	Login correcto y navega a caja, pulsa en anular la factura que considere
Flujo alternativo	-
Post-condiciones	-

Identificador	CU13
Nombre	Expedir un pedido
Descripción	Permite al usuario marcar un pedido sin expedir como expedido
Pre-condiciones	Debe tener rol Almacén
Flujo básico	Login correcto y navega a almacén, pulsa en el pedido que considere, marca los productos conforme los va preparando y lo marca como expedido
Flujo alternativo	-
Post-condiciones	-

Identificador	CU14
Nombre	Anonimizar facturas
Descripción	Permite al usuario, por rango de fechas, anonimizar facturas para desvincular datos de compra a datos personales
Pre-condiciones	Debe tener rol Administrador
Flujo básico	Login correcto y navega a caja, busca facturas en el rango que considere y pulsa el botón anonimizar facturas
Flujo alternativo	-
Post-condiciones	-

6 Desarrollo

Este apartado va a ser el que contenga los detalles sobre la implementación de la aplicación, tanto frontend como backend, mostrando código, capturas de la misma y acceso a dos vídeos de las funcionalidades de una forma resumida.

6.1 Funciones

Una de las primeras cosas que se hizo antes de empezar el desarrollo, fue una tormenta de ideas para decidir las funcionalidades y el alcance de éstas en el proyecto; estableciendo de esta manera un producto mínimo viable que entregar.

Las funcionalidades que se decidió entregar como mínimo fueron:

- Gestión de usuarios: Capacidad de crear usuarios, activarlos y desactivarlos, cambiarles la contraseña y atomizar los roles y permisos.
- Gestión de almacenes: Capacidad de crear y editar almacenes, los almacenes son la entidad más alta de la aplicación, sólo comparten súper administradores entre sí.
- Leer EAN por hardware: Capacidad de leer los códigos de barras de los productos mediante un lector específico o la webcam.
- Recepción de pedidos: Capacidad de recibir pedidos de productos con limitaciones muy concretas configuradas en la creación del producto, limitar al máximo la capacidad humana de fallar en estas limitaciones.
- Recepción del pedido restricciones familiares: Las familias tienen restricciones de cuánto y cuántas veces pueden consumir productos del banco de alimentos. Capacidad automática de gestionar estas limitaciones, con la mínima interacción humana.
- Gestión de productos: Capacidad para añadir y editar productos. Éstos productos deben poder tener un EAN asociado y solicitar información pública de OpenFoodFacts para añadir información nutricional a su ficha. Hay bancos de alimentos que cuentan con personal sanitario al que puede resultarles útil tener esta información a mano.
- Facturación y caja: Capacidad de cobrar facturas y buscarlas por rango de fechas para consultas posteriores.
- Impresión de facturas: Capacidad de ver las facturas maquetadas para imprimirlas fácilmente.
- Expedición de pedidos: Capacidad de ver qué pedidos están pendientes de despachar para que almacén pueda prepararlos con antelación.

- Dashboard con gráficas: Capacidad de ver intuitivamente datos relevantes que faciliten al personal la previsión de trabajo.

Además de ciertas solicitudes y cambios puntuales en la forma en la que la aplicación iba funcionando, que entraban dentro de la prevención de riesgos y las estimaciones; se solicitaron funcionalidades nuevas casi a última hora. Como la gestión temporal fue un éxito y se iba holgado en tiempo, se aceptaron las solicitudes y se implementaron.

Las nuevas funcionalidades fueron:

- Anonimizar facturas: Capacidad de desvincular familias y facturas, además de la LOPD, Cáritas manda, y al no tener, en el banco de alimentos, muy clara la política de datos personales; se solicitó una funcionalidad capaz de anonimizar facturas de forma permanente, por si acaso.
- Productos de higiene especial: Capacidad de configurar ciertos productos como especiales, éstos productos podrían impactar o no en las limitaciones de la familia, pudiendo ser ésta (en el proceso de pedido) la que decida si quiere que éstos productos afectasen a los límites de las facturas de su expediente, o no.

6.2 Pantallas

Una vez definidas las funcionalidades que deberá atender la aplicación, se pasó a organizar las diferentes pantallas que tendrá. La gama de colores usadas se dejaron a discreción del tema por defecto de Angular Material, debido a la falta de conocimientos de diseño. El menú usa escalas de grises por decisión estética personal.

Los colores permiten una clara diferenciación de qué implicaciones tendrán las acciones asociadas al uso de éstos botones. Listémoslos:

- Color azul:
 - Acciones que activan, aceptan o navegan a subsecciones o funcionalidades relacionadas.
 - Combinaciones entre vacíos con iconos o textos azules, o rellenos azules con contenido blanco.
 - Color rojo:
 - Acciones que desactivan, cancelan, vacían datos o navegan a secciones anteriores.
 - Combinaciones entre vacíos con iconos o textos rojos, o rellenos rojos con contenido blanco.
 - Mensajes relacionados con errores o acciones que son irreversibles.
 - Color verde:
 - Mensajes relacionados con acciones realizadas correctamente por el servidor.
-

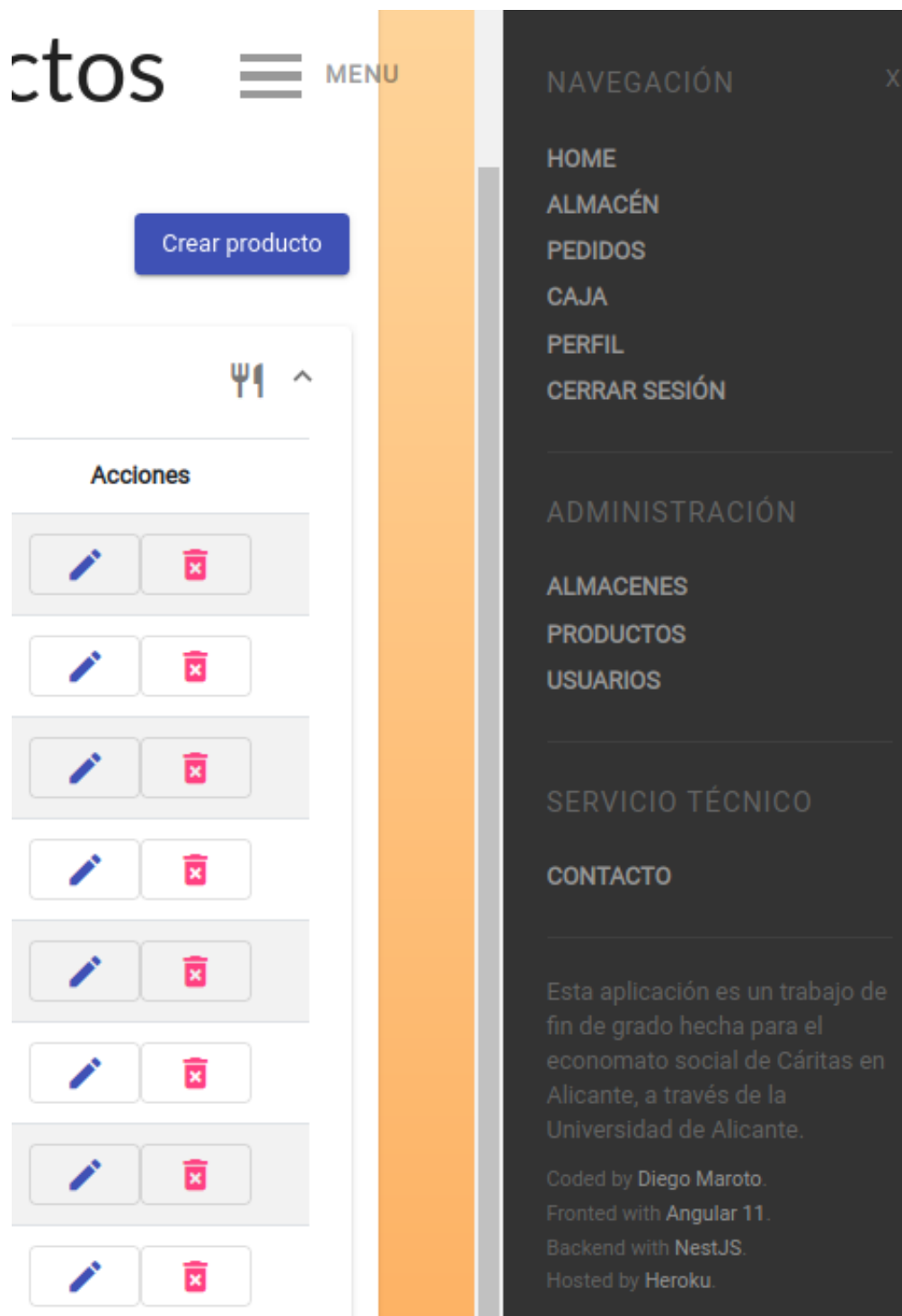


Figura 6.1: Ejemplo colores de la aplicación

6.2.1 Login

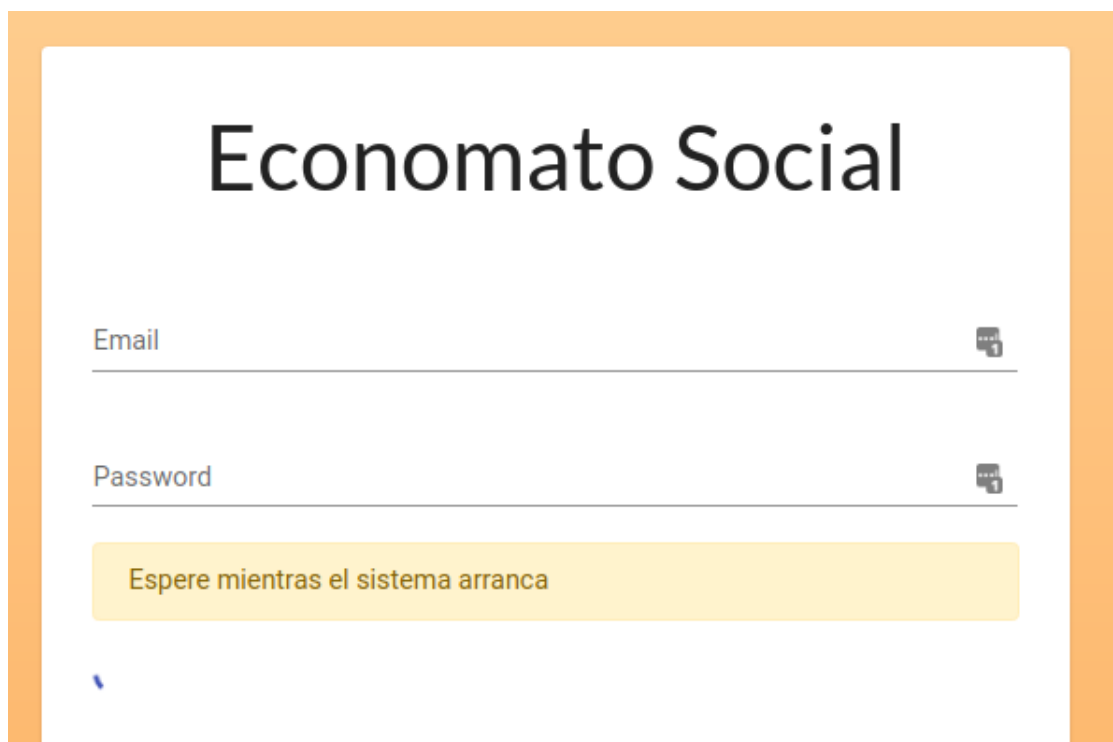
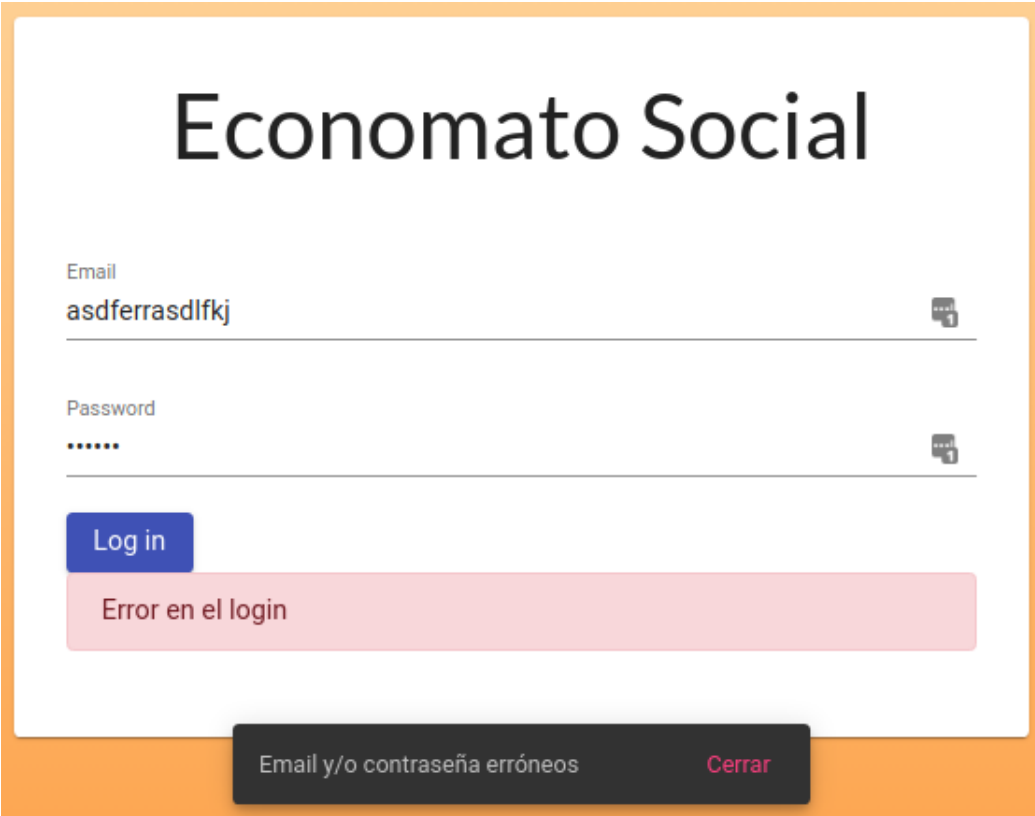


Figura 6.2: Login esperando respuesta del backend

En esta figura se muestra cómo el frontend, para evitar frustración del usuario, avisa de que el backend aún tienen que arrancar y debe esperar. Esto es dado por tener el frontend y el backend alojados de forma independiente y habiendo usando heroku como hosting.

En heroku cuando la aplicación no recibe peticiones en 5 minutos, muere, esto hace que la siguiente petición tarde más en responder, dado que debe arrancar el sistema; Pudiendo tardar entre 20 segundos y un minuto. Esta pantalla es la que el primer voluntario que acceda a la aplicación verá cada día.



The screenshot shows a login interface for 'Economato Social'. It features two input fields: 'Email' with the text 'asdferrasdlfkj' and 'Password' with masked characters '.....'. Both fields have a small icon of a document with a red '1' in the bottom right corner. Below the fields is a blue 'Log in' button. A pink error message box displays 'Error en el login'. At the bottom, a dark grey toast notification shows the text 'Email y/o contraseña erróneos' and a red 'Cerrar' button.

Figura 6.3: Login error credenciales

Este sería el comportamiento de un login incorrecto. Una posible mejora, que sería crítica en caso de ataque, sería añadir un captcha y/o un rate limiter a este tipo de vectores de ataque.

Propuse añadir un rate limiter a este endpoint, dado que es el único que no requiere un jwt debidamente firmado y en vigor, pero como habían tareas más acuciantes pendientes y llegaron otras nuevas, se decidió dejar para una versión posterior.



Figura 6.4: Dashboard

Tras un login correcto, se muestra el dashboard.

6.2.2 Pedidos

La sección de pedidos es un stepper, es decir, una única página que se mueve adelante y atrás de forma dinámica y a discreción del usuario y con alguna automatización. Esta sección es la que crea las facturas y para ello se obliga a que el voluntario pase por 3 subsecciones:

- Búsqueda de la familia que está atendiendo, para que el sistema tenga en cuenta su histórico del mes en las restricciones de la compra
- La gestión de los productos que desea adquirir
- Resumen del pedido y generación de la factura

Después de generar la factura, ésta se abre en una pestaña nueva; maqueta de forma limpia e independiente y lista para imprimir.

La imagen muestra una maqueta de la interfaz de usuario para la creación de pedidos, específicamente el primer paso: 'Búsqueda de familia'. El título principal es 'Creación de pedidos'. Debajo, hay un progreso de tres pasos: 1. Buscar familia (activo), 2. Añadir productos, y 3. Revisión del pedido. El formulario principal contiene tres campos de texto: 'Nombre' (con un ícono de lupa), 'Nº Expediente' y 'Credencial'. Debajo de estos, hay un grupo de opciones para 'Límite de la familia' con botones de 10€, 13€, 17€, 22€, 27€ y 32€. A la derecha de este grupo hay un interruptor para 'Es especial'. En la parte inferior, hay un botón gris con el texto 'Buscar'.

Figura 6.5: Pedidos: Step 1 - Búsqueda de familia

Creación de pedidos

✓ Buscar familia
2 Añadir productos
3 Revisión del pedido

Diego
987987
Exp: 123123
0 visitas
Límite 17€

Vaciar pedido

Figura 6.6: Pedidos: Step 2 - Resumen familia en gestión del pedido

Acumulado 12.85€ en 13 productos

#	Código	Nombre		Cantidad		Máximo	Precio	Total
☞	A01	ACEITE G.	-1	<input type="text" value="2"/>	+1	2	0.80€	1.60€
☞	A02	ACEITE O.	-1	<input type="text" value="2"/>	+1	2	1.20€	2.40€
☞	A03	ALUBIA 1KG	-1	<input type="text" value="2"/>	+1	2	1.00€	2.00€
☞	A04	ALUBIA FR	-1	<input type="text" value="1"/>	+1	2	0.30€	0.30€
☞	A05	ARROZ L.	-1	<input type="text" value="1"/>	+1	2	0.40€	0.40€
☞	A06	ARROZ R.	-1	<input type="text" value="1"/>	+1	2	0.40€	0.40€
☞	A07	ATUN3U	-1	<input type="text" value="0"/>	+1	3	0.90€	0.00€
☞	A08	AZÚCAR	-1	<input type="text" value="1"/>	+1	2	0.35€	0.35€
☞	A22	HARINA	-1	<input type="text" value="2"/>	+1	3	0.20€	0.40€
☞	A27	MAGDALENAS	-1	<input type="text" value="0"/>	+1	1	0.45€	0.00€
☞	A28	MELOCOTON MIT	-1	<input type="text" value="0"/>	+1	2	0.65€	0.00€
☞	A35	SARDINAS ACEITE	-1	<input type="text" value="0"/>	+1	3	0.40€	0.00€
☞	A36	SARDINAS TOM.	-1	<input type="text" value="0"/>	+1	3	0.20€	0.00€
🛒	P01	PAÑAL T2 ★	-1	<input type="text" value="1"/>	+1	1	5.00€	5.00€

Cancelar
Siguiente

Figura 6.7: Pedidos: Step 2 - Gestión del pedido

Creación de pedidos

✓ Buscar familia

✎ Añadir productos

3 Revisión del pedido

Diego

987987

Exp: 123123

0 visitas

Límite 17€

Total acumulado: 12.85€

#	Código	Ean	Producto	Cantidad	Precio	Total
🛒	A01		ACEITE G.	2	0.80€	1.60€
🛒	A02		ACEITE O.	2	1.20€	2.40€
🛒	A03		ALUBIA 1KG	2	1.00€	2.00€
🛒	A04		ALUBIA FR	1	0.30€	0.30€
🛒	A05		ARROZ L.	1	0.40€	0.40€
🛒	A06		ARROZ R.	1	0.40€	0.40€
🛒	A08	5601370145015	AZÚCAR	1	0.35€	0.35€
🛒	A22	8480024754509	HARINA	2	0.20€	0.40€
🛒	P01		PAÑAL T2	1	5.00€	5.00€

Anterior

Generar factura

Figura 6.8: Pedidos: Step 3 - Resumen del pedido

Nº 2021-D31DF46

Diego - 987987

Lunes, 31 mayo 2021

Expediente 123123

Total: 12.85€

#	Código	Ean	Producto	Cantidad	Precio	Total
🍴	A01		ACEITE G.	2	0.80€	1.60€
🍴	A02		ACEITE O.	2	1.20€	2.40€
🍴	A03		ALUBIA 1KG	2	1.00€	2.00€
🍴	A04		ALUBIA FR	1	0.30€	0.30€
🍴	A05		ARROZ L.	1	0.40€	0.40€
🍴	A06		ARROZ R.	1	0.40€	0.40€
🍴	A08	5601370145015	AZÚCAR	1	0.35€	0.35€
🍴	A22	8480024754509	HARINA	2	0.20€	0.40€
👤	P01		PAÑAL T2	1	5.00€	5.00€

Figura 6.9: Pedidos: Vista previa factura

N° 2021-D31DF46
Diego - 987987
Lunes, 31 mayo 2021
Expediente 123123

Total: 12.85€

#	Código	Ean	Producto	Cantidad	Precio	Total
¶	A01		ACEITE O.	2	0.80€	1.60€
¶	A02		ACEITE O.	2	1.20€	2.40€
¶	A03		ALUBIA 1KG	2	1.00€	2.00€
¶	A04		ALUBIA FR	1	0.30€	0.30€
¶	A05		ARROZ L.	1	0.40€	0.40€
¶	A06		ARROZ R.	1	0.40€	0.40€
¶	A08	5601370145015	AZÚCAR	1	0.35€	0.35€
¶	A22	8480224754509	HARINA	2	0.20€	0.40€
✶	P01		PAÑAL T2	1	5.00€	5.00€

Imprimir1 página

Destino

Guardar como PDF

Páginas

Todo

Páginas por hoja

1

Márgenes

Predeterminados

Opciones

☐ Encabezado y pie de página

☒ Gráficos de fondo

Imprimir utilizando el cuadro de diálogo del sistema (Ctrl+Shift+P)

Cancelar

Guardar

Figura 6.10: Pedidos: Imprimir factura

6.2.3 Caja

La sección de caja es un conjunto de acordeones que se complementan entre sí para mostrar las facturas abiertas, cobradas y cerradas (anuladas) y, por último, un resumen de caja.

Las facturas que se muestra, por defecto son del día en curso, pero se puede cambiar el rango de fechas para consultar otras facturas. Además, en caso de que el usuario sea administrador, éste podrá anonimizar las facturas que se estén incluidas en el rango de fechas especificado.

Caja

Anonimizar estas facturas

Búsqueda por rango de fechas:

Desde
31/5/2021

Hasta
31/5/2021

Buscar facturas

Facturas abiertas1 facturas x 12,85 €

#	Código	Nombre	Especial	Precio	Acciones
	2021-D31DF46	Diego	No	12,85 €	

Facturas cobradas0 facturas x 0,00 €

Facturas cerradas0 facturas x 0,00 €

Cierre de cajaResúmenes

Figura 6.11: Caja

6.2.4 Almacén

La sección de almacén es un conjunto de acordeones que se complementan entre sí para mostrar los pedidos sin despachar.

Cuando se quiere despachar un pedido, aparecerá un listado de sus productos con la capacidad de marcarlos para saber si se ha preparado ya o no, esta funcionalidad se ha realizado a petición expresa de voluntarios de almacén del banco de alimentos al que se entregará el proyecto.

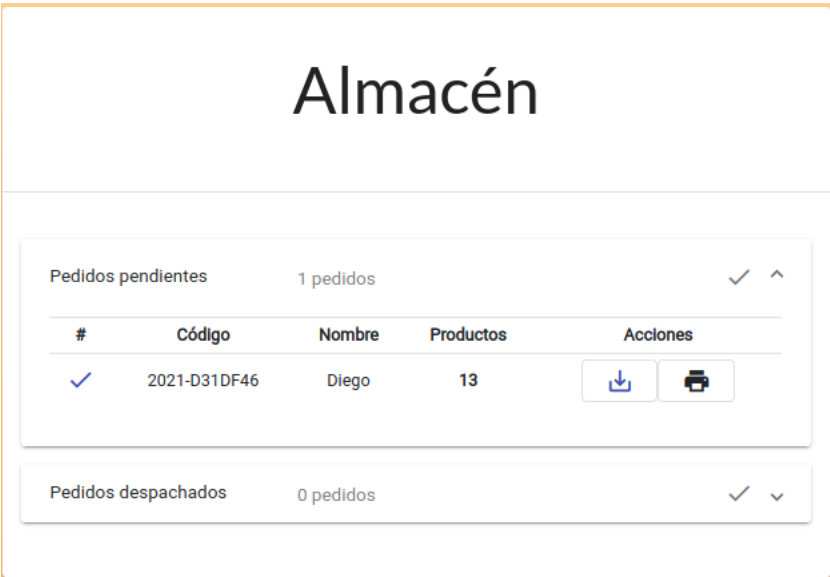


Figura 6.12: Almacén - Resumen de pedidos por despachar

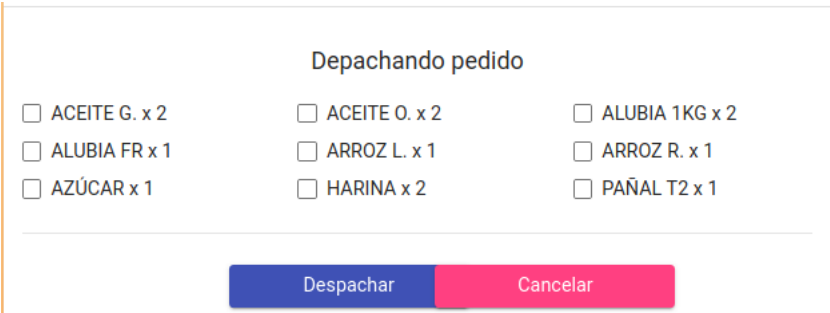


Figura 6.13: Almacén - Despachando un pedido

6.3 Implementación

En este apartado se detallarán implementaciones que tengan relevancia en el desarrollo del proyecto, dándole un trasfondo y sentido a las pantallas que se acaban de mostrar.

6.3.1 Base de datos

Para la realización de este proyecto se ha decidido trabajar con una base de datos no relacional, en concreto con MongoDB que es una base de datos basada en documentos. Usar MongoDB como motor de base de datos ha venido especialmente bien al proyecto pues, además de usar JSON como esquema base de los documentos, hemos podido usar Compass y su capa gratuita de cluster en la nube.

```
1 export const databaseProviders = [  
2   {  
3     provide: 'MONGODB_CONNECTION',  
4     imports: [ConfigModule],  
5     inject: [ConfigService, Logger],  
6     useFactory: (  
7       configService: ConfigService<AppConfig>,  
8       logger: LoggerService,  
9     ): Promise<typeof mongoose> => {  
10      logger.log('CONNECTION MADE', 'ProvidersModule');  
11      return mongoose.connect(configService.get('mongoUrl'), {  
12        useNewUrlParser: true,  
13        useUnifiedTopology: true,  
14      });  
15    },  
16  ],  
17];
```

Código 6.1: Inyección proveedores ddbb

Dado que para el buen funcionamiento de la aplicación nos había solicitado atomizar los permisos que tiene cada usuario para acceder a los datos, se decidió hacer un mapa que incluye una relación entre el rol que se requiere y la lista de roles que podría tener el usuario para que el permiso fuese concedido. Por ejemplo, para realizar acciones de almacén, se debe tener cualquiera de los roles relacionados, que en esta caso son cualquiera de los 3 de administrador ó el propio de almacén. Este cruce se ideó con el pensamiento a futuro de que la aplicación pudiese crecer, complicarse los permisos y que hubiese algunos que colisionasen, por ejemplo, un permiso que fuese imprimir facturas y que éste fuese únicamente permitido a administradores, caja y pedidos; con esta implementación sólo habría que añadir un nuevo elemento al mapa con los permisos permitidos y toda la aplicación se vería afectada inmediatamente.


```
1 export class Role {
2   static permissionMap: RolePermissionMap;
3   static initPermissionMap(): void {
4     const map = new Map<RoleName, RoleName[]>();
5     map.set(RoleName.SUPERADMIN, [RoleName.SUPERADMIN]);
6     map.set(RoleName.ADMIN, [RoleName.SUPERADMIN, RoleName.ADMIN]);
7     map.set(RoleName.ADMINLOCAL, [
8       RoleName.SUPERADMIN,
9       RoleName.ADMIN,
10      RoleName.ADMINLOCAL,
11    ]);
12    map.set(RoleName.ALMACEN, [
13      RoleName.SUPERADMIN,
14      RoleName.ADMIN,
15      RoleName.ADMINLOCAL,
16      RoleName.ALMACEN,
17    ]);
18    map.set(RoleName.RECEPCION, [
19      RoleName.SUPERADMIN,
20      RoleName.ADMIN,
21      RoleName.ADMINLOCAL,
22      RoleName.RECEPCION,
23    ]);
24    map.set(RoleName.FAMILIAR, [
25      RoleName.SUPERADMIN,
26      RoleName.ADMIN,
27      RoleName.ADMINLOCAL,
28      RoleName.FAMILIAR,
29    ]);
30    map.set(RoleName.CAJA, [
31      RoleName.SUPERADMIN,
32      RoleName.ADMIN,
33      RoleName.ADMINLOCAL,
34      RoleName.CAJA,
35    ]);
36    Role.permissionMap = map;
37  }
38  static hasNeededRole(input: RoleName[], needed: RoleName): boolean {
39    if (!Role.permissionMap) {
40      Role.initPermissionMap();
41    }
42    const allowed = Role.permissionMap.get(needed);
43    return allowed && input.some((role) => allowed.includes(role));
44  }
45 }
```

Código 6.2: Gestión de roles para acceder a datos

6.3.2 Esquemas y servicios

6.3.2.1 Login y registro

```
1 export type UserDocument = User & Document;
2
3 export class UserComment {
4   @Prop() author: string;
5   @Prop() comment: string;
6 }
7
8 export class UserAction {
9   @Prop() date: Date;
10  @Prop() action: string;
11 }
12
13 @Schema()
14 export class User {
15   @Prop() name: string;
16   @Prop() email: string;
17   @Prop() password: string;
18   @Prop() active: boolean;
19   @Prop() permissions: RoleName[];
20   @Prop() warehouses: string[];
21   @Prop() accessHistory: Date[];
22   @Prop() actionsHistory: UserAction[];
23   @Prop() comments: UserComment[];
24
25   constructor(o: User) {
26     Object.assign(this, o);
27   }
28
29   static fromUserDto(user: UserDto): User {
30     return new User({
31       name: user.name,
32       email: user.email,
33       active: user.active,
34       permissions: user.permissions.map((p: RoleName) => p),
35       warehouses: user.warehouses.map((w: string) => w),
36       accessHistory: user.accessHistory.map((h: string) => new Date(h)),
37       actionsHistory: user.actionsHistory.map((h: UserActionDto) => ({
38         action: h.action,
39         date: new Date(h.date),
40       })),
41       comments: user.comments.map((c: UserCommentDto) => ({
42         author: c.author,
43         comment: c.comment,
44       })),
45     });
46   }
47 }
48
49 export const UserSchema = SchemaFactory.createForClass(User);
```

Código 6.3: User schema

El documento usuario contiene su propia contraseña, la cual está resumida con bcrypt y 10 rondas, veamos cómo se crea un usuario y cómo se hace login:

```
1 @Injectable()
2 export class UserService {
3   private hashRounds = 10;
4
5   async createUser(input: UserDto): Promise<UserDto> {
6     this.userValidations(input);
7     const repeated: UserDocument = await this.userMongo.findOneBy(
8       'email',
9       input.email,
10    );
11    if (repeated)
12      throw new ConflictException(`The email ${input.email} is in use`);
13    input.password = hashSync(input.password, this.hashRounds);
14    const user = User.fromUserDto(input);
15    return UserService.userMapper(await this.userMongo.create(user));
16  }
17 }
```

Código 6.4: Creación de un usuario

```
1 @Injectable()
2 export class LoginService {
3   private genericErrorMessage = 'Login error';
4
5   async login(input: LoginDto): Promise<UserDto> {
6     const user = (await this.userMongo.findBy('email', input.email))[0];
7     if (!user) throw new ForbiddenException(this.genericErrorMessage);
8     if (!compareSync(input.password, user.password))
9       throw new ForbiddenException(this.genericErrorMessage);
10    if (!user.active) {
11      user.actionsHistory.push({
12        date: new Date(),
13        action: 'Attempt to login when inactive',
14      });
15      await user.save();
16      throw new ForbiddenException(this.genericErrorMessage);
17    }
18    user.accessHistory.push(new Date());
19    await user.save();
20    return UserService.userMapper(user);
21  }
22 }
```

Código 6.5: Login de un usuario

El mapeo de salida de un usuario limpia datos sensibles, en este caso es exclusivamente la contraseña.

6.3.2.2 Productos

El documento de productos contiene los datos necesarios para identificarlos correctamente, además de aquellos datos relacionados con OpenFoodFacts. Cada producto está relacionado con su almacén, por lo que almacenes con el mismo producto tendrán duplicidad de datos, algunos innecesarios, como los nutricionales. Una posible mejora sería sacar datos nutricionales a una colección distinta y relacionarlos por ean para evitar duplicidad, como ahora mismo sólo se tiene vista a usar un único almacén, ésta implementación se decidió dejar para versiones posteriores.

```
1 export type ProductDocument = Product & Document;
2
3 export class ProductLimits {
4   @Prop() price: number;
5   @Prop() quantity: number;
6 }
7
8 @Schema()
9 export class Product {
10   @Prop() ean: string;
11   @Prop() name: string;
12   @Prop() alias: string;
13   @Prop() quantity: string; // amount or weight
14   @Prop() categories: string;
15   @Prop() ingredients: string;
16   @Prop() allergens: string;
17   @Prop() labels: string;
18   @Prop() imageUrl: string;
19   @Prop() limits: ProductLimits[];
20   @Prop() pvp: number;
21   @Prop() code: string; // Inner code for every warehouse
22   @Prop() type: string;
23   @Prop() chargeableOutBudget: boolean;
24   @Prop() warehouse: string;
25
26   constructor(o: Product) {
27     Object.assign(this, o);
28   }
29 }
30
31 export const ProductSchema = SchemaFactory.createForClass(Product);
```

Código 6.6: Esquema de producto

Veamos cómo se crea un producto y, en caso de tener EAN, se consulta con el api pública de OpenFoodFacts para recuperar los datos nutricionales de éste.

```

1 @Injectable()
2 export class ProductService {
3   private openFoodUrl =
4     'https://world.openfoodfacts.org/api/v0/product/{ean}.json';
5   async create(
6     input: CreateProductResponseDto,
7   ): Promise<ReadProductResponseDto> {
8     let product = {} as Product;
9     product.alias = input.alias;
10    product.limits = input.limits;
11    product.pvp = input.pvp;
12    product.code = input.code;
13    product.type = input.type;
14    product.chargeableOutBudget = input.chargeableOutBudget;
15    product.ean = input.ean;
16    let openFoodResponse = {};
17    if (product.ean)
18      openFoodResponse = await this.readByEan(product.ean);
19    product = {
20      ...ProductService.fromOpenFoodToProduct(openFoodResponse),
21      ...product,
22    };
23    return ReadProductResponseDto.fromProductDocument(
24      await this.productsMongo.create(product),
25    );
26  }
27  async readByEan(ean: string): Promise<any> {
28    const response: AxiosResponse = await this.httpClient
29      .get(this.openFoodUrl.replace('{ean}', ean), {
30        headers: { 'User-Agent': this.config.get('openFoodUserAgent') },
31      }).toPromise();
32    if (!response.data.product)
33      throw new InternalServerErrorException('Error requesting to openfood');
34    return response.data.product;
35  }
36  private static fromOpenFoodToProduct(response: any): Product {
37    return {
38      name: response.product_name_es || response.product_name,
39      ean: response.code,
40      quantity: response.quantity,
41      categories: response.categories,
42      labels: response.labels,
43      allergens: response.allergens,
44      ingredients: response.ingredients_text_es || response.ingredients_text,
45      imageUrl: response.image_front_url,
46    } as Product;
47  }
48 }

```

Código 6.7: Product service: Añadir producto nuevo

6.3.2.3 Pedidos

El documento de pedidos contiene los datos que hacen falta para dar completa funcionalidad a lo que se nos pidió que era necesario para procesar los pedidos de las familias.

Una de las claves, es que aparecen varios identificadores de usuario, dado que es un sistema muy centrado en permitir quién puede hacer qué, es necesario almacenar quién ha hecho qué; en este caso, quién ha creado el pedido, quién lo ha cobrado y quién lo ha despachado, además de cuándo se han realizado estas acciones.

Dado que la información sobre las familias está almacenada, mantenida y expedida por Cáritas, en este documento se relaciona el pedido con el documento identificativo de la familia y su expediente de Cáritas, no almacenamos datos de ningún tipo en ninguna otra colección, dado que no se sabía si tendríamos permiso, además, no resulta necesario para dar ninguno de los servicios propuestos. Esta forma de relacionar cada pedido con la familia origen, facilitó la búsqueda de qué familias han hecho visitas el mes en curso (por las restricciones) y, además, la anonimización de los pedidos, pues únicamente hay que vaciar esos campos en los pedidos del rango de fechas que se especifica en la llamada.

Uno de los quebraderos de cabeza que se tuvo en el proyecto hasta llegar a este esquema final, fue que en principio se quería restringir los pedidos cuando las familias ya habían superado sus límites, pero en las reuniones de sprint en el economato llegamos a saber que habían problemas de colisión entre expedientes y documentos de identificación, que debían resolver personalmente llamando por teléfono a la sede de Cáritas. Por lo que acabamos haciendo el sistema flexible a las colisiones en este sentido y, si se revisan los vídeo-tutoriales adjuntos, se podrá apreciar que en caso de una colisión es el voluntario el que puede decidir manualmente si esta colisión afecta o no a los límites que impone el sistema.

Además de lo anterior, dado que las órdenes es algo que se está consultando continuamente por diferentes secciones de la aplicación, se incluyó caché en la misma aplicación, de ésta forma salvo que la lista de pedidos del día corriente cambie, siempre que se consulten se devolverá el caché en lugar de acceder a base de datos. De ésta forma ahorramos ancho de banda y, si fuese una capa de pago, coste por procesamiento y transferencia de las consultas innecesarias.

Estas peticiones tan repetitivas fue sacada a la luz por una necesidad que apareció conforme iba creciendo la aplicación, hay diferentes pantallas desde donde se está esperando que el listado de pedidos se actualice y, además, pueden ser diferentes personas/dispositivos las que estén a espensas de estos cambios. Así, pues, se hizo uso de la tecnología de websockets para poder comunicar en tiempo real cambios en los pedidos, ya sea una adición nueva, un cobro, una cancelación o haberlo despachado. Si se revisan los vídeo-tutoriales adjuntos se podrá apreciar cómo un usuario recibe en tiempo real los cambios ejecutados por otro en otro navegador.

Veamos el esquema de los pedidos:

```
1 export type OrderDocument = Order & Document;
2
3 export class ProductResume {
4   @Prop() id: string;
5   @Prop() ean: string;
6   @Prop() code: string;
7   @Prop() name: string;
8   @Prop() amount: number;
9   @Prop() pvp: number;
10 }
11
12 @Schema({ strict: false })
13 export class Order {
14   @Prop({ index: true }) code: string; // Code for easy human identification
15   @Prop() headquarter: string; // foreign key headquarter collection
16   @Prop() familyName: string;
17   @Prop({ index: true }) expedient: string;
18   @Prop({ index: true }) credential: string;
19   @Prop() special: boolean;
20   @Prop() products: ProductResume[]; // list of products
21   @Prop() pvp: number;
22   @Prop() type: string;
23   @Prop() chargeableOutBudgetSelected: boolean;
24   @Prop() createdAt: Date;
25   @Prop() updatedAt: Date;
26   @Prop() paid: boolean;
27   @Prop() deleted: boolean;
28   @Prop() origin: string; // USER ID
29   @Prop() resolver: string; // USER ID
30   @Prop() resolvedAt: Date;
31   @Prop() dispatcher: string; // USER ID
32   @Prop() dispatched: boolean;
33   @Prop() dispatchedAt: Date;
34 }
35
36 export const OrderSchema = SchemaFactory.createForClass(Order);
```

Código 6.8: Esquema de pedido

La creación de pedidos es una de las implementaciones que hace uso de los websockets, pues hay secciones de la aplicación que deben actualizarse en tiempo real cuando aparecen nuevos datos. Veamos cómo se ha resuelto esto:

```

1 @Injectable()
2 export class InvoiceService {
3   async create(input: Order, jwt: JWTToken): Promise<any> {
4     input.createdAt = new Date();
5     input.updatedAt = new Date();
6     input.code = `${new Date().getFullYear().toString()}-${uid(7).toUpperCase()}
7     `;
8     input.paid = false;
9     input.deleted = false;
10    input.origin = jwt.id;
11    const order: OrderDocument = await this.mongo.create(input);
12    this.broadcastInvoiceResolved(order.id, ResolveInvoiceAction.CREATED);
13    await this.saveUserAction(jwt, `Invoice ${order.id} created by ${order.id}`,);
14    await this.refreshCache();
15    return order.toObject();
16  }
17  private broadcastInvoiceResolved(
18    invoiceId: string,
19    action: ResolveInvoiceAction,
20  ): void {
21    this.wsService.broadcastInvoice({ invoiceId, action });
22  }
23 }

```

Código 6.9: Invoice service: Crear nuevo pedido

```

1 @Injectable()
2 @WebSocketGateway()
3 export class WebsocketsService
4   implements OnGatewayConnection, OnGatewayDisconnect {
5   private wsClients: any[] = [];
6   broadcastInvoice(message: InvoiceMessage): void {
7     const data: WsInvoiceMessage = { event: WsTopics.INVOICES, data: message };
8     this.broadcast(data, { id: null } as Socket);
9   }
10  broadcast(message: WsMessage | string, sender: Socket): void {
11    const payload: WsMessage = { event: WsTopics.UNKNOWN, data: '' };
12    if (typeof message === 'string') payload.data = message;
13    else {
14      payload.event = message.event;
15      payload.data = message.data;
16    }
17    for (const c of this.wsClients) {
18      if (c.id === sender.id) return;
19      c.send(JSON.stringify(payload));
20    }
21  }
22 }

```

Código 6.10: Websocket service: Difusión de mensajes

6.3.2.4 Caja

La gestión de caja hace uso del documento de pedidos, editando lo que necesita para dar funcionalidad a lo solicitado.

Recordemos que la caja debe permitir dos acciones, cobrar pedidos o cerrarlos por erróneos. Esto afectará al listado de pedidos del día corriente, por lo que debe actualizar la caché y, además, difundir el mensaje por websocket para que todos los clientes sepan que ha habido un cambio.

Ésta implementación es susceptible de mejora, ahora mismo cuando el servidor cambia un pedido, actualiza la caché y avisa por difusión a los clientes de que ha habido un cambio, esto provoca que inmediatamente haya un aluvión de solicitudes para recuperar la lista de pedidos actualizada. Dado que está cacheada, el servidor tan cual recibe la solicitud despacha los datos directamente de la ram, sin pasar por base de datos, pero aún así, es ineficiente; sobretodo si hay muchos clientes conectados. La mejora podría consistir en difundir el nuevo estado del pedido con su identificador, de ésta forma, los clientes pueden actualizar ese pedido en concreto en sus datos locales sin más. Dado que todo lo relacionado con esta funcionalidad no era una necesidad y se hizo como mejora, se decidió que este último punto se delegase a versiones futuras, ya que impactaría en la forma en la que se difunden los mensajes y en la que los clientes deciden qué hacer con éstos.

Veamos la sección de código en la que se cambia el estado de un pedido de pendiente a cobrado/cerrado:

```
1 export enum ResolveInvoiceAction {
2   CLOSE,
3   PAY,
4   CREATED,
5 }
6 @Injectable()
7 export class InvoiceService {
8   async resolveInvoice(
9     id: string,
10    action: ResolveInvoiceAction,
11    jwt: JWTToken,
12  ): Promise<void> {
13    const order: OrderDocument = await this.mongo.findById(id);
14    if (!order) throw new NotFoundException(`Invoice with id ${id} not found`);
15    if (order.paid || order.deleted) {
16      throw new PreconditionFailedException('This invoice is already resolved');
17    }
18    switch (action) {
19      case ResolveInvoiceAction.CLOSE:
20        order.deleted = true;
21        break;
22      case ResolveInvoiceAction.PAY:
23        order.paid = true;
24        break;
25      default:
26        throw new InternalServerErrorException('Action against invoice not
27        implemented');
28    }
29    this.broadcastInvoiceResolved(id, action);
30    order.updatedAt = new Date();
31    order.resolvedAt = new Date();
32    order.resolver = jwt.id;
33    await order.save();
34    await this.saveUserAction(
35      jwt,
36      `Invoice ${order.id} updated by ${jwt.id} with action ${action}`,
37    );
38    await this.refreshCache();
39  }
40 }
```

Código 6.11: Invoice service: Resolver pedido

6.3.2.5 Almacén

La gestión de almacén también hace uso del documento de pedidos, editando lo que necesita para dar funcionalidad a lo solicitado.

De una forma similar a la caja, el almacén tiene una funcionalidad muy concreta, ésta es despachar pedidos y nada más. Como bien se estuvo comentando antes, esto cambia el listado de órdenes del día y refrescará caché además de difundir el mensaje por websocket.

```
1 @Injectable()
2 export class InvoiceService {
3   async dispatchOrder(id: string, jwt: JWTToken): Promise<OrderDocument> {
4     const order: OrderDocument = await this.mongo.findById(id);
5     if (!order) throw new NotFoundException(`Invoice with id ${id} not found`);
6     if (order.dispatched)
7       throw new PreconditionFailedException('This order has been dispatched
8       already');
9     order.dispatched = true;
10    order.dispatcher = jwt.id;
11    order.dispatchedAt = new Date();
12    order.updatedAt = new Date();
13    await order.save();
14    await this.saveUserAction(jwt, `Order ${order.id} dispatched by ${jwt.id}`);
15    this.deleteCache();
16    return order;
17  }
18 }
```

Código 6.12: Invoice service: Despachar pedido

6.3.2.6 Roles y controladores

Por último, se procede a explicar cómo se han implementado la salvaguarda de roles en Nestjs, y dónde y cómo afecta a la aplicación.

Como ya se ha comentado, uno de los requisitos indispensables es la gestión de roles de forma atómica, con solape o sin él, para manejar los permisos de los usuarios lo más estrictamente posible. De esta forma, se optó por un patrón de seguridad que se inició en los sistemas linux y se puso muy de moda en la gestión de usuarios en los servicios de infraestructura cloud. Éste es el del permiso más restrictivo por defecto, es decir, no se tendrá acceso a no ser que explícitamente se diga lo contrario.

Para poder simular este patrón en el proyecto, hicimos 2 implementaciones haciendo uso del patrón Guard de Nestjs, éste viene a ser un servicio que se ejecuta antes de cualquier solicitud y que decide si se tiene acceso o no a ese recurso. Para este proyecto hay 2 guards, haciendo su magia respectivamente, en función de:

- Token Bearer con un Json Web Token
- Roles permitidos en la ruta

Su uso en Nestjs sería el siguiente:

```
1 @Controller('user')
2 @UseGuards(JwtGuard, RolesGuard)
3 export class UserController {
4   constructor(private service: UserService) {}
5
6   @Get('/:id')
7   @Roles(RoleName.OWNER, RoleName.ADMINLOCAL)
8   async readUserById(@Param('id') id: string): Promise<UserDto> {
9     const result: UserDto = await this.service.readUserById(id);
10    return result;
11  }
12 }
```

Código 6.13: User Controller: Guards en el controlador de usuario

Mediante estos decoradores damos lógica automáticamente a esta clase, se convierte en un controlador de la ruta '/user' para el servidor web en Express, hace uso a nivel de controlador de los 2 guards y, en la ruta '/:id', escribe expresamente qué roles son necesarios para acceder a esa ruta.

El orden de los guards es importante, primero se debe manejar el jwt ya que ahí está incluido el identificador del usuario y, entre otros datos, su lista de permisos.

```

1 @Injectable()
2 export class JwtGuard implements CanActivate {
3   constructor(private configService: ConfigService<AppConfig>) {}
4   canActivate(context: ExecutionContext): Promise<boolean> {
5     const req: Request = context.switchToHttp().getRequest();
6     const bearer: string = req.header('Authorization');
7     let token: JWTToken;
8     try {
9       const regex = /^bearer (.+)\$/i;
10      const g = regex.exec(bearer);
11      token = jwt.verify(g[1], this.configService.get('jwtSecret'));
12    } catch (err) {
13      throw new UnauthorizedException('Bad bearer token');
14    }
15    req['jwt'] = token;
16    return true;
17  }
18 }

```

Código 6.14: Guard JWT: Verificación de un json web token e inyección de datos en la request

Este guard por defecto siempre devuelve true, ya que el único caso en el que debería impedir pasar es cuando el jwt no se puede verificar (no coincide el secreto de la firma o ha caducado), caso en el que se lanza una excepción más intuitiva. Dado que los datos extraídos del jwt deber ser accesibles más adelante, es común en Express pasar información entre middlewares inyectándolos directamente en la request, tal y como se hace en la línea 15 del código 6.14.

Una vez verificada la identidad del usuario, se procede a revisar su rol, pasando por la implementación del guard de roles:

```

1 @Injectable()
2 export class RolesGuard implements CanActivate {
3   constructor(private reflector: Reflector) {}
4   canActivate(context: ExecutionContext): boolean {
5     const roles = this.reflector.get<RoleName[]>('roles', context.getHandler());
6     if (!roles) throw new InternalServerErrorException('Roles needed in route');
7     const req: Request = context.switchToHttp().getRequest();
8     const token: JWTToken = req['jwt'];
9     if (roles.includes(RoleName.OWNER)) {
10      if (roles.length === 1) return req.params.id === token.id;
11      if (req.params.id === token.id) return true;
12    }
13    return roles
14      .filter((role) => role !== RoleName.OWNER)
15      .some((role) => Role.hasNeededRole(token.roles, role));
16  }
17 }

```

Código 6.15: Guard Roles: Verificación de roles necesarios para el acceso

Esta implementación nos indica lo siguiente del guard de roles:

- Es completamente obligatorio que la ruta que está siendo solicitada tenga el decorador de roles asignado.
 - Línea 6 del código 6.15
- Si la ruta tiene un rol y es exclusivamente OWNER, no es cuestión de rol si no de ser dueño de los datos, si no se da el caso se corta el acceso.
 - Línea 10 a 13 del código 6.15
- Por último, se verifica que de los roles que tiene el usuario, haya al menos uno compatible con las necesidades de la ruta.
 - Línea 14 a 16 del código 6.15

De esta forma tenemos las rutas protegidas, primero, obligando a que haya que acreditarse siempre; y segundo, obligando a que el desarrollador explícitamente indique qué permisos se deben complacer en las rutas que expone en la aplicación.

A esta implementación le hace falta una mejora, ésta consistiría en asegurarse de poder hacer caducar los tokens, ahora mismo estos caducan a las 8 horas de su creación, para facilitar el trabajo de los voluntarios y no tener que estar logueando durante la jornada. El problema de esto es que no se le puede retirar el acceso a alguien sin ir a su dispositivo a cerrar la sesión, habría que esperar a que le caduque el token.

Veamos cómo se crean los tokens en el login de los usuarios:

```
1 createJwt(user: UserDto): string {
2   const secret = this.configService.get('jwtSecret');
3   const payload: Partial<JWTToken> = {
4     id: user.id,
5     exp: moment().add(8, 'h').toDate().getTime(),
6     roles: user.permissions,
7     warehouses: user.warehouses,
8   };
9   return jwt.sign(payload, secret);
10 }
```

Código 6.16: Jwt service: Creación de token

6.4 Conclusiones, patrones y revisión del código

Para terminar esta subsección, se procede a un pequeño análisis del código desarrollado, patrones utilizados y el framework elegido para mejorar la productividad.

El uso del framework Nestjs ha permitido usar diferentes patrones de forma intuitiva y sin apenas codificar.

- Patrón MVC clásico, con la salvedad de que la vista es mapeo e impresión de datos en json.
- Inyección de dependencias, Nestjs permite una configuración de módulos muy intuitiva que permite inyectar dependencias en el constructor de las clases con muy poca configuración, de esta forma la inicialización de las clases de las que depende la lógica de negocio es completamente agnóstica a la misma lógica de negocio.
- Patrón singleton, los servicios inyectables en Nestjs que se importan desde módulos independientes tienen la peculiaridad de que sólo se instancian una vez por módulo, ahorrando memoria y errores de implementación en caso de querer hacerlo manualmente.
- Patrón factory, Nestjs usa una factory por defecto para crear el model de MongoDB a partir de una clase decorada como esquema. Además, en el proyecto se ha usado el patrón factory para construir los dto de salida a partir de un documento de MongoDB.

Nestjs tiene muchos más beneficios, como el enrutado automático del servidor web o la configuración automática del servidor de documentación basado en swagger.

Para el frontend se ha usado Angular 11, del cual se ha expuesto menos código ya que es muy similar al backend. Ésta fue una de las razones de usar la combinación Nestjs y Angular, ambos son MVC con módulos configurables e inyectables y usan una sintaxis muy similar, de ésta forma ser conocedor de uno de los dos framework hace el acercamiento al otro mucho más sencillo y con muy poca curva de aprendizaje.

Para ahorrar esfuerzo y tiempo en diseño se han usado paquetes de Angular que traen animaciones y maquetaciones, Angular Material. Y para la estructuración y el responsive se ha usado Bootstrap 4, éste no es de Angular pero hay integraciones libres de la comunidad que integran a la perfección las dependencias de Bootstrap con las de Angular para hacer funcionar sin problema este framework.

Más allá de los frameworks, en el desarrollo del backend se ha hecho especial hincapié en desarrollar teniendo en mente los principios SOLID (Óscar Blancarte, 2018):

- S: Responsabilidad única.
 - División del proyecto en módulos y servicios independientes, intentando siempre que cada clase y/o función se haga cargo exclusivamente de lo que le concierne.
- O: Principio abierto/cerrado.
 - Cuando se ha tenido que añadir o cambiar lógica, se ha intentado siempre minimizar el impacto que esto tiene sobre las funcionalidad actuales. Las factories han servido de ayuda, ya que aíslan funcionalidades concretas de forma estática e independiente de clases relacionadas.
- L: Sustitución de Liskov.
 - Cuando hay herencia, las clases hijas deben servir para lo mismo que para los padres. En este proyecto se ha usado herencia de una clase abstracta para dar funcionalidad genérica a servicios de la capa de datos, las funcionalidad no cambian entre servicios, sólo las colecciones a las que hacen referencia.
- I: Segregación de interfaces.
 - Se ha intentado hacer prevalecer la alta cohesión frente al acoplamiento, la modularización de Nestjs ha ayudado en este menester.
- D: Inversión de dependencias
 - Principalmente en la capa de acceso a datos, se ha implementado de forma que los detalles dependen de las abstracciones y no al revés.

Para finalizar, se ha pretendido mantener el código todo lo seco posible: DRY (Don't Repeat Yourself), haciendo uso de herencias, funciones y extrayendo a módulos inyectables las codificaciones que se han detectado como generalizaciones.

6.5 Resultado final

Para finalizar este capítulo sabe señalar que hay mucho más código y muchos más detalles en la aplicación de lo que se puede expresar en un documento. Por lo que se va a aprovechar este cierre para incluir enlaces directamente a los servicios desplegados en producción, vídeo-tutoriales que se han entregado junto al proyecto al economato social y el enlace al proyecto en el repositorio remoto Github.

- Enlace al Backend (Maroto, 2021a)
 - Enlace al Frontend (Maroto, 2021b)
 - Enlace al Repositorio público en Github (Maroto, 2021c)
 - Enlace a los Vídeo-tutoriales (Maroto, 2021d)
-

7 Conclusiones

Una vez finalizado el proyecto, no está de mas recapitular y revisar el trabajo realizado. Es una buena forma de revisar si se han logrado cumplir los objetivos que se marcaron en un inicio.

7.1 Revisión de los objetivos marcados

Los objetivos que se pretendían conseguir son los siguientes:

- Investigar y analizar soluciones hardware lowcost
 - Se ha realizado un estudio de diferentes tipos de sistemas a tener en cuenta, pasando desde el self-hosting con una raspberry pi a terminar decidiendo usar servicios serverless cloud en heroku.
- Investigar y analizar el estado de los bancos de alimentos en el mundo y su relación con la tecnología
 - Se han investigado proyectos y necesidades, quedando patente las potenciales implicaciones positivas que tendría adoptar en mayor medida la tecnología en el campo social
- Estudiar el mercado para decidir el stack tecnológico en el que desarrollar la aplicación
 - Se ha realizado un estudio centrado en encuestas de StackOverflow, líder en el de la comunidad online tecnológica, para decidir lenguajes a utilizar.
- Estudiar alternativas de software en el mercado
 - Se ha investigado si la solución que se pretende con este proyecto podría ser solventada por uno de los CRM que ya existen en el mercado, si bien hay mucha comunidad tecnológica a penas hay proyectos libres adhoc para este tipo de necesidades.
- Diseñar la aplicación web conforme a los requisitos recogidos del economato social
 - Se estudiaron y documentaron los requisitos fundamentales que la aplicación debería cubrir y no sólo se han cumplido en su totalidad si no que se han implementado mejoras a lo largo del proyecto y, cuando ya se había finalizado el desarrollo, se aceptaron dos nuevas implementaciones y se llevaron a cabo en tiempo y forma.
- Uso de metodología ágil con entrega continua
 - Se ha adoptado con éxito una metodología ágil basada en sprints de 3 semanas, con reuniones en vivo con el economato para recoger feedback y requisitos; habiendo construido así una solución totalmente a medida.

- Entregar la aplicación web en su versión acabada
 - Se ha entregado la aplicación finalizada y usable desplegada en un host gratuito, además de guía de uso y vídeo tutoriales para facilitar la adopción de los voluntarios del economato social.

7.2 Conclusiones

Para concluir esta memoria, cabe destacar que ha sido un auténtico placer desarrollar una aplicación de estas características debido al uso que se le pretende dar. Ha sido un desafío, personal y académico, tanto por el alcance del proyecto como por la situación de pandemia en la que nos encontramos. Haber sido parte de un proyecto como este hace mella en uno, saber que los conocimientos y habilidades pueden ser puestas en pos de los más necesitados y de que sólo hacen falta ganas y no grandes presupuestos para mejorar la calidad laboral de aquellos que se ponen en primera línea para ayudar a los más necesitados. Ha sido una experiencia enriquecedora y si puedo, seguiré colaborando y mejorando este proyecto a lo largo del tiempo hasta que, por la razón que sea, no me sea posible aportar nada más. Deseo que la adopción de la tecnología en el campo social no deje de crecer y que, efectivamente, algún día los bancos sociales tengan voluntarios o financiación suficiente como para que la tecnología deje de ser un muro tan insalvable como lo es hoy.

Además de lo personal, profesionalmente ha sido un reto. He tenido que desempolvar conocimientos de la carrera, sacar las mejores herramientas y dar lo mejor de mí para llegar a buen puerto. No es mi primera experiencia como full-stack pero sí puedo decir sin miedo que es la primera vez que desarrollo yo sólo un proyecto de éste alcance y del que hay gente que espera tanto, me he visto necesitado de sacar a relucir patrones de diseño y conocimientos avanzados en tecnología y arquitectura cloud que he adquirido personal y profesionalmente en mis años de experiencia. Siento que ha sido una gran experiencia y que, aunque esté terminando de escribir la memoria de este proyecto, aún queda mucho por desarrollar y mejorar en la entrega que terminar aquí supone.

Al principio no las tenía todas conmigo en la decisión de volcar los esfuerzos en desarrollar una solución cloud native, dado que la idea de usar el self-hosting tenía mucho peso inicial. Pero por suerte no hubo arrepentimientos, las automatizaciones hicieron su magia para facilitarme el trabajo en el CI/CD y las capas gratuitas de hosting y base de datos nos han permitido entregar un producto de calidad que funciona a coste cero y que, dada la naturaleza inicial de configuración y desarrollo, podríamos migrar a cualquier otro servicio cloud o proveedor de base de datos con muy poco esfuerzo; en caso de ser necesario.

Para finalizar me gustaría comentar que aún hay mucho por recorrer en la tecnología centrada en el campo social y, con los retos que tenemos encima ahora mismo por la pandemia, seguro que hay muchísimas ideas que podemos llevar a cabo. Desarrollos y avances que no pueden hacer otra cosa que mejorar nuestra sociedad justo donde más falta hace, donde no hay recursos informáticos ni educación tecnológica.

Bibliografía

- Bravent. (2019). *Bravent beneficios metodologías ágiles*. Descargado de <https://www.bravent.net/beneficios-de-las-metodologias-agiles/>
- EC2-AWS. (2021). *Amazon ec2*. Descargado de <https://aws.amazon.com/es/ec2/>
- Jutras, M. (2019). *Erp implementation survey shows real facts about erp project outcomes*. Descargado de <https://ultraconsultants.com/wp-content/uploads/2021/02/Real-Facts-About-ERP-Implementation-final-rev-2.12.19.pdf>
- Maroto, D. (2021a). *Backend proyecto*. Descargado de <https://economato-social-api.herokuapp.com/>
- Maroto, D. (2021b). *Frontend del proyecto*. Descargado de <https://economato-social.herokuapp.com>
- Maroto, D. (2021c). *Repositorio github del proyecto*. Descargado de <https://github.com/DiegoMGar/TFG>
- Maroto, D. (2021d). *Vídeo-tutoriales resumiendo funcionalidades*. Descargado de <https://drive.google.com/drive/folders/1pv-6iishQkM29StiSHUyY74lWEVx0t6q>
- Odoo. (2021). *Odoo erp y crm*. Descargado de https://www.odoo.com/es_ES/
- Oracle. (2021). *Oracle enterprise resource planning*. Descargado de <https://www.oracle.com/es/erp/>
- Quirk, E. (2019). *Top 15 free and open source erp solutions*. Descargado de <https://solutionsreview.com/enterprise-resource-planning/top-15-free-and-open-source-erp-solutions/>
- SAP. (2021). *Sap erp*. Descargado de <https://www.sap.com/spain/products/enterprise-management-erp.html>
- survey StackOverflow, T. (2020). *Stack overflow developer survey 2020*. Descargado de <https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages>
- Óscar Blancarte. (2018). *Principios solid y patrones de diseño*. Descargado de <https://www.oscarblancarteblog.com/2018/08/15/principios-solid-patrones-diseno/>