Ferran Ballesté Solsona

Aplicació web de gestió d'inventari pel sector de la restauració

TREBALL DE FI DE GRAU

dirigit per Marc Sánchez Artigas

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Resum.

El projecte té com a objectiu dissenyar i implementar una aplicació web de tipus SaaS (*Software as a Service*) per al control d'inventari en els magatzems d'aliments del sector de la restauració. L'aplicació permetrà als usuaris gestionar els seus productes, receptes i magatzems de forma senzilla i eficaç. El projecte també inclou el desenvolupament d'una pàgina web de màrqueting per promocionar l'aplicació.

L'objectiu principal d'aquest projecte és millorar la gestió dels recursos i la rendibilitat dels restaurants. Per al desenvolupament de l'aplicació s'ha utilitzat Vue com a *framework* per l'aplicació i PostgreSQL com a sistema de gestió de bases de dades.

Resumen.

El proyecto tiene como objetivo diseñar e implementar una aplicación web de tipo SaaS (*Software as a Service*) para el control de *stock* en los almacenes de alimentos del sector de la restauración. La aplicación permitirá a los usuarios gestionar sus productos, recetas y almacenes de forma sencilla y eficaz. El proyecto también incluye el desarrollo de una página web de marketing para promocionar la aplicación.

El objetivo principal consiste en mejorar la gestión de los recursos y la rentabilidad de los restaurantes. Para el desarrollo de la aplicación se ha utilizado Vue como *framework* para la aplicación y PostgreSQL como sistema de gestión de bases de datos.

Abstract.

The project aims to design and implement a web application of SaaS (*Software as a Service*) type for stock control in food warehouses of the catering sector. The application will allow users to manage their products, recipes and warehouses in a simple and efficient way. The project also includes the development of a marketing website to promote the application.

The main objective of this project is to improve the management of resources and profitability of restaurants. For the development of the application, Vue has been used as a frontend framework and PostgreSQL as a database management system.

Índex

1 INTRODUCCIÓ 5

2 DESCRIPCIÓ GENERAL DEL PROJECTE 6

2.1 PROBLEMES 6
2.2 PROPOSTA DE VALOR I MODEL DE NEGOCI 7
2.3 PREVISIONS D'ÚS 8
2.4 LÍNIA TEMPORAL DEL PROJECTE 9

3 REQUISITS 10

3.1 REQUISITS FUNCIONALS 10
3.1.1 Aplicació Web 10
3.1.2 Pàgina web de màrqueting 11
3.2 REQUISITS NO FUNCIONALS 11
3.3 DIAGRAMA DE CASOS D'ÚS 12

4 ANÀLISI DELS REQUISITS FUNCIONALS 13

4.1 DIAGRAMA DE CLASSES 13
4.2 DIAGRAMES DE SEQÜÈNCIA DELS CASOS D'ÚS 14

5 ELECCIÓ D'EINES DE TREBALL I PLATAFORMES DE *HOSTING* 17

5.1 APLICACIÓ WEB 17
5.2 BACKEND DE L'APLICACIÓ 18
5.3 PÀGINA WEB DE MÀRQUETING 18

6 DISSENY 19

6.1 ARQUITECTURA DE L'APLICACIÓ 19
6.2 DISSENY DE LA INTERFÍCIE GRÀFICA 20
6.3 DISSENY DE LA BASE DE DADES 21

7 IMPLEMENTACIÓ 23

7.1 APLICACIÓ WEB 23
7.1.1 Estructura de l'aplicació 23
7.1.2 Estat de l'aplicació i sincronització amb el backend 25
7.1.3 Responsivitat 26
7.1.4 Reactivitat amb Vue 27
7.1.5 Anàlisis de dades 27
7.2 BACKEND 28
7.2.1 Supabase API 28
7.2.2 PostgreSQL RPCs (Remote Procedure Call) 29
7.2.3 Node Mailer 31
7.3 PÀGINA WEB 32
7.3.1 Estructura de l'aplicació 32
7.3.2 SEO 33
7.3.3 Google Analytics i Cookies 34

8 SEGURETAT 35

8.1 AUTENTICACIÓ 35
8.2 AUTORITZACIÓ EN EL BACKEND (ROW LEVEL SECURITY) 36
8.3 PROTECCIÓ DE COLUMNES 39

9 AVALUACIÓ 40

9.1 DISSENY DELS CASOS DE PROVA 40
9.1.1 Aplicació Web 40
9.1.2 Backend 40
9.1.3 Pàgina de màrqueting 40
9.2 RESULTATS 41
9.2.1 Frontend 41

9.2.2 Backend 42
9.2.3 Pàgina de màrqueting 43
10 CONCLUSIONS 46
11 RECURSOS UTILITZATS 47

Índex de taules

TAULA 1. TAULA COMPARATIVA DE PREUS ENTRE NETLIFY I FIREBASE [6] [12] 17
TAULA 2. PERMISOS PELS DIFERENTS ROLS D'USUARI 38
TAULA 3. TESTS DELS CASOS D'ÚS 41
TAULA 4. TESTS FUNCIONS POSTGRESQL 43
TAULA 5. TESTS REQUISITS FUNCIONALS DE LA PÀGINA DE MÀRQUETING 43

Índex de figures

FIGURA 1. LÍNIA TEMPORAL DEL PROJECTE 9
FIGURA 2. DIAGRAMA DE CASOS D'ÚS 12
FIGURA 3. DIAGRAMA DE CLASSES 13
FIGURA 4. DIAGRAMA DE SEQÜÈNCIES EN LA CREACIÓ D'UN PRODUCTE 14
FIGURA 5. DIAGRAMA DE SEQÜÈNCIES PER INTRODUIR UNA ARRIBADA 15
FIGURA 6. DIAGRAMA DE SEQÜÈNCIES PER REALITZAR UNA COMANDA 16
FIGURA 7. GRÀFIC COMPARATIU DE PREUS ENTRE NETLIFY I FIREBASE 17
FIGURA 8. ARQUITECTURA DE L'APLICACIÓ [2] 19
FIGURA 9. UI DISPOSITIU MÒBIL 20
FIGURA 10. UI DISPOSITIU AMB PANTALLA GRAN 20
FIGURA 11. POSTGRESQL DATABASE SCHEMA [18] 21
FIGURA 12. VISTA PRINCIPAL DE L'APLICACIÓ PER DISPOSITIU DE PANTALLA GRAN 26
FIGURA 13. CARPETA DE PROJECTE DE LA PÀGINA WEB AMB ASTRO 32
FIGURA 14. VISITES PER TÍTOL DE PANTALLA EN COOKLEDGE.COM 33
FIGURA 15. DIFERENTS FONTS DE LA PÀGINA COOKLEDGE.COM 34
FIGURA 16. RESULTATS LIGHTHOUSE (DESKTOP) 44
FIGURA 17. RESULTATS LIGTHOUSE (MOBILE) 44
FIGURA 18. RESULTAT DE CERCA A GOOGLE 45

1 Introducció

El sector de la restauració és un dels més importants i dinàmics de l'economia espanyola. No obstant això, també és un sector que es troba davant de grans reptes i dificultats, com la competència creixent, la variabilitat de la demanda, els canvis en els hàbits dels consumidors o la gestió eficient dels recursos.

Un dels aspectes clau per al bon funcionament i la rendibilitat dels restaurants és el control d'inventari en els magatzems d'aliments. Aquest control implica planificar, organitzar i supervisar les entrades i sortides dels productes, així com el seu emmagatzematge, conservació i distribució. Un control inadequat pot provocar pèrdues econòmiques per deteriorament o caducitat dels aliments, falta d'aprovisionament o excés d'inventari.

El projecte consisteix en dissenyar i implementar una aplicació web de tipus SaaS que faciliti i millori el control d'inventari als magatzems d'aliments del sector de la restauració. L'aplicació permetrà als usuaris gestionar els seus productes, productes semielaborats, proveïdors i magatzems de forma senzilla i eficaç. A més, el projecte també inclou el desenvolupament d'una pàgina web de màrqueting per promocionar l'aplicació i captar clients potencials.

Aquest treball ha suposat aplicar els coneixements i les competències adquirides al llarg del grau en informàtica a un projecte real i innovador. També ha permès conèixer i utilitzar eines i tecnologies actuals per al desenvolupament web, com Vue, Quasar, Astro i PostgreSQL, i aprendre els seus avantatges i desavantatges i les diferents opcions de *hosting* al Cloud.

2 Descripció general del projecte

Aquest treball de fi de grau forma part de Cookledge, un projecte d'empresa de logística B2B que permet digitalitzar i optimitzar tota la cadena de subministrament i operacions dels restaurants. Actualment, participa en la 3a edició de la Incubadora TIC de REDESSA.

2.1 Problemes

Els restaurants conviuen diàriament amb dos problemes logístics importants: problemes d'emmagatzematge i problemes de gestió dels productes:

- **Problemes d'emmagatzematge**

Al voltant del 70% dels restaurants lloguen un local que utilitzen com a magatzem de productes tant acabats com no acabats. El cost mitjà d'aquest local puja fins als 700 euros mensuals, comptabilitzant tant costos fixos com costos variables (electricitat, aigua, gas, etc.). Aquest cost, si estudiem les previsions macroeconòmiques dels pròxims anys (2022-2027), tendirà a pujar, ja que els estudis anuncien l'augment continu del cost de l'energia, disparant la taxa d'inflació per sobre del 12% interanual a la zona euro.

Tenint en compte la inelàstica oferta del mercat immobiliari espanyol, els costos fixos del lloguer dels immobles, de la mateixa manera, continuaran augmentant. Així, els costos fixos i variables que les empreses de restauració suporten als magatzems es menjaran una gran part de la seva estructura financera.

- **Problemes de gestió de producte.**

a) Ineficient gestió de l'inventari

La digitalització del sector restauració continua sent, en paraules clares, nul·la. En la majoria dels casos, l'únic sistema digital utilitzat pels restaurants són softwares TPV (Terminal Punt de Venda) i, ocasionalment, softwares de gestió de reserves.

b) Pèrdua d'informació financera rellevant:

La competició en el sector de la restauració és molt alta, i converteix a aquesta indústria en la menor quant a marges de benefici. Degut a això resulta imprescindible per als restaurants ajustar i conèixer les econòmiques del negoci. Tot i això, la majoria dels restaurants no realitzen aquests càlculs, estimant els preus de les seves elaboracions.

c) No disponibilitat de productes:

La ineficient gestió de l'inventari comporta, sempre, falta de productes i, en conseqüència, impossibilita la venda d'aquests mateixos. A més, a causa dels terminis de lliurament dels proveïdors majoristes del sector restauració, el restaurant pot no tenir disponibilitat del determinat producte durant dies i afectar la valoració del restaurant per part dels clients.

d) Perill de pèrdua de producte:

Una pèrdua d'un producte és una pèrdua de diners. Aquesta pèrdua de productes és un problema costós, ja que reduirà el marge i la rendibilitat de les vendes. Els nivells de pèrdua de l'inventari varien segons la indústria, però, generalment, és ideal mantenir-los en un rang entre l'1% i el 3% de les vendes. Aquest rang, en els restaurants, s'eleva fins a una mitjana de 12%.

Les causes més comunes de pèrdua de *stock* en el sector restauració són:

- Robatori: El robatori intern i extern constitueix gran part de la pèrdua d'inventari.
- Accions administratives: Inclosos errors en els enviaments, variacions en el magatzem, materials extraviats i documentació detallada incorrectament.
- Deteriorament de l'inventari: Pot succeir durant el transport o al local i pot ser conseqüència d'un mal embalatge, sistema d'emmagatzematge incorrecte, etc.
- Estafes o comissions il·legals: Quan els proveïdors, intencionalment o no, cobren de més per béns de menor valor o directament aquests béns no són lliurats.

2.2 Proposta de valor i model de negoci

Cookledge proposa una solució en els diferents problemes detallats anteriorment basada en l'externalització de tota la cadena logística, fent focus en la gestió de magatzems d'aliments per restaurants.

Cookledge posseeix magatzems d'última milla repartits estratègicament en zones geogràfiques densament poblades. Aquests magatzems tenen una capacitat d'emmagatzematge mínima i suficient per donar servei a una mitjana de 15 restaurants per local, doncs aquests no poden posseir, generalment, una àrea superior als 150 m^2. Aquest fet és conseqüència de la imperativa necessitat d'establir els magatzems a una distància màxima, en termes temporals, no superior als 30 minuts de viatge en automòbil al punt de venda.

Els magatzems de Cookledge, en si mateixos, solucionen el primer problema de la cadena logística esmentat anteriorment: Els costos de l'emmagatzematge del *stock*. Basats en l'economia col·laborativa, l'immobilitzat i els treballadors dels magatzems son compartits pel total del nombre de clients establerts als locals de Cookledge, fet que implica una reducció dràstica dels costos i l'àrea total d'emmagatzematge.

De mitjana, l'optimització proposada genera una disminució del 30% de l'àrea total ocupada pels productes emmagatzemats i, de la mateixa manera, també genera una disminució del 40% en els costos totals soferts pels restaurants. Això és degut, detalladament, pel repartiment de la inversió en CapEx (*Capital Expenditures*), OpEx (*Operational Expenditures*), arrendament, amortitzacions, etc., entre el total de clients establerts en aquell únic magatzem.

Aquests magatzems són, també, el centre d'operacions de la cadena logística dels restaurants. Tots els esdeveniments logístics necessaris fora del núvol són gestionats per Cookledge als seus magatzems (Arribades del transport de mercaderies, accions d'inventariat i emmagatzematge de producte, manipulació de mercaderies, Picking, Packing i transport final fins al punt de venda).

El personal del restaurant ja no perd més temps en aquests processos costosos i burocràtics i és capaç de centrar aquest període de la jornada laboral en la generació de valor pels seus clients.

Això no serviria de res (o de molt) si aquests locals no poguessin disposar de connectivitat en línia amb el restaurant. És necessària, per a la diferenciació respecte a altres magatzems tradicionals anomenats *self storatge*, la implementació d'un software al núvol capaç de comunicar al restaurant tota informació generada pels moviments operatius de la cadena logística.

A través del software, el director del restaurant pot controlar tota la cadena logística en temps real. Es poden realitzar les següents accions:

- Observació de nova informació financera (normalment no controlada pels restaurants). Alguns exemples, per no esplaiar-me massa, poden ser la rotació de producte, històrics de preus, control de proveïdors i distribuïdors, escandalls, incidències, etc.

- Control d'arribades de compres realitzades als proveïdors, juntament amb l'observació de factures i albarans digitalitzats facilitats per aquests mateixos.

- Realització de comandes als operaris del magatzem de Cookledge, que portaran els productes que el restaurant necessiti en una durada màxima de 30 minuts de 08.00 a 23.00 h.
- Realització de receptes de productes semielaborats a partir de les relacions gramatge-cost introduïdes al software amb l'arribada de mercaderia.

La relació simbiòtica *magatzem físic / magatzem virtual* soluciona el segon problema de la cadena logística: La gestió del *stock*, la rapidesa, l'agilitat i l'experiència d'usuari de Cookledge juntament amb l'externalització logística són els punts clau causants del canvi del clàssic "paper i bolígraf" cap a l'era digital al sector HORECA.

2.3 Previsions d'ús

El nombre d'usuaris inicialment no serà tant elevat com altres plataformes de tipus SaaS. Això és degut al fet que en el model de negoci incorporem un component físic de lloguer compartit de magatzems. De totes maneres, l'aplicació s'ha de dissenyar de forma escalable necessita ser accessible en tot moment.

Respecte a la utilització per cada usuari, un cop l'usuari tingui tots els seus productes creats i els proveïdors introduïts, s'estima que cada restaurant utilitzarà l'aplicació dos o tres cops dia almenys per fer una comanda i diverses vegades per realitzar consultes. Si es tracta d'un magatzem personal i no un gestionat per Cookledge també haurà d'utilitzar-la per introduir els productes que li arriben.

Per la part de l'administrador, aquest tindrà un ús més elevat de l'aplicació ja que haurà de revisar l'aplicació quan rebi la notificació d'una comanda o arribin productes al magatzem.

Per la part de previsió d'emmagatzematge, un usuari pot tenir entre 100 i 1000 productes, depèn molt del tipus de restaurant i si tenen carta de vins o varien constantment de productes. S'espera que la taula de moviments sigui la que ocupa més espai, ja que cada producte tindrà múltiples moviments que s'aniran acumulant en el temps.

2.4 Línia temporal del projecte

En la **Figura 1** tenim representades les diferents activitats que s'han realitzat en el projecte i les que s'han previst a curt termini. Per buscar possibles clients ens vam trobar amb la necessitat de tenir una pàgina web de màrqueting i una aplicació mínimament funcional per poder convèncer a la gent de la serietat del projecte, aquesta web va ser millorada posteriorment després de les sessions de formació on vam acabar de definir el model de negoci.

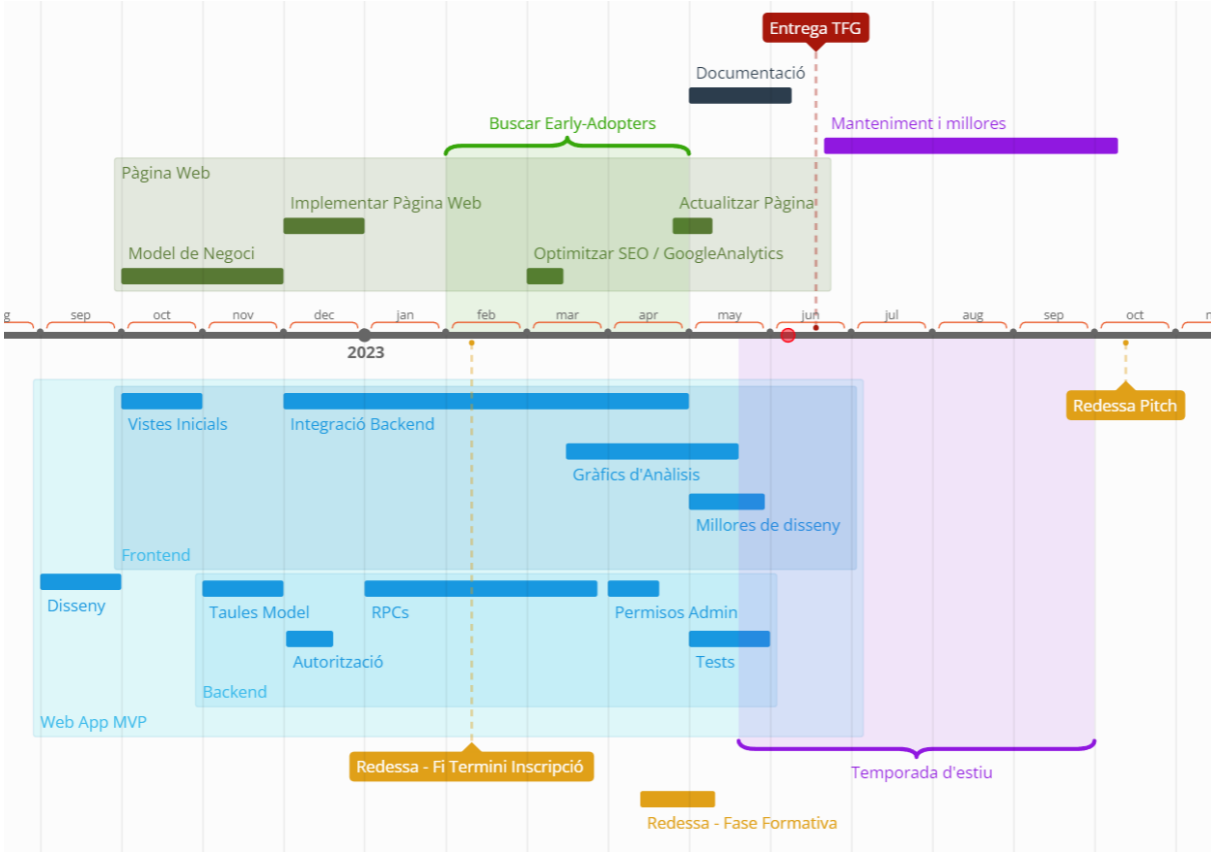

Figura 1. Línia temporal del projecte

3 Requisits

En aquest apartat s'especifiquen els diferents requisits funcionals i no funcionals que s'han de complir en la realització del projecte, així com els casos d'ús derivats.

3.1 Requisits funcionals

3.1.1 Aplicació Web

El projecte consisteix en el disseny i implementació d'una aplicació web de tipus SaaS pel control d'inventari en magatzems d'aliments del sector de la restauració i la seva pàgina de màrqueting. La idea principal és digitalitzar el procés logístic i millorar l'eficiència per fer front als problemes d'emmagatzematge dels restaurants.

- **R1:** L'aplicació ha de permetre que cada usuari pugui crear els seus productes, assignant-los un nom, una descripció, una categoria, un preu, proveïdor i una unitat de mesura.
- **R2:** L'aplicació ha de possibilitar la creació i gestió de múltiples magatzems per part dels usuaris, indicant el seu nom i el seu tipus (privat o gestionat per Cookledge).
- **R3:** L'aplicació ha de donar suport a la creació de receptes de productes elaborats o semielaborats a partir d'altres productes, especificant els ingredients, les quantitats, els passos i el temps necessari per a la seva elaboració.
- **R4:** L'aplicació ha de permetre el control del inventari i els moviments dels productes i receptes dels usuaris, registrant les entrades i sortides dels magatzems.
- **R5:** L'aplicació ha de calcular l'evolució del preu dels productes i receptes dels usuaris en funció dels preus dels ingredients i dels marges de benefici.
- **R6:** L'aplicació ha de facilitar les incidències de les arribades dels productes als magatzems, identificant els proveïdors responsables i les causes possibles.
- **R7:** L'aplicació ha de mostrar diferents anàlisis de dades sobre els seus productes, receptes, magatzems, restaurants i proveïdors, mitjançant gràfics interactius en la pantalla principal.
- **R8:** L'aplicació ha d'enviar avisos per correu electrònic d'entrades, sortides i alertes relacionades amb els productes, receptes, magatzems, restaurants i proveïdors dels usuaris.
- **R9:** Una *demo* amb dades fictícies ha de ser accessible per usuaris no identificats, on es puguin modificar les dades per adaptar-les a les necessitats dels clients potencials.
- **R10:** L'aplicació ha d'incorporar vistes i permisos d'administrador per poder gestionar les comandes dels usuaris i supervisar el funcionament de l'aplicació.

3.1.2 Pàgina web de màrqueting

- **R11:** La pàgina web ha de presentar el nom, el logotip i el lema de l'aplicació Cookledge, així com una breu introducció sobre la seva finalitat i els seus beneficis per als usuaris.
- **R12:** La pàgina web ha de mostrar una captura de pantalla o un vídeo demostratiu de l'aplicació, destacant les seves principals funcionalitats i característiques.
- **R13:** La pàgina web ha de proporcionar un formulari de contacte per a aquells usuaris que vulguin rebre més informació sobre l'aplicació o sol·licitar un servei personalitzat.
- **R14:** La pàgina web ha d'incloure un botó o un enllaç perquè els usuaris puguin accedir a la *demo* pública de l'aplicació i provar-la ells mateixos.
- **R15:** La pàgina web ha de tenir una secció amb preguntes freqüents (FAQ) on es resolguin els dubtes més comuns sobre l'aplicació i el seu funcionament.
- **R16:** La pàgina web ha de tenir una secció amb testimonis i opinions de clients satisfets amb l'aplicació, així com logos de restaurants o empreses que l'utilitzen.
- **R17:** La pàgina web ha de tenir una secció amb les dades de contacte de l'empresa Cookledge, com el seu correu electrònic, telèfon, adreça i xarxes socials.
- **R18:** La pàgina web ha d'estar optimitzada per al posicionament SEO (*Search Engine Optimization*), fent servir paraules clau rellevants, etiquetes meta, títols i subtítols adequats i contingut original i actualitzat.

3.2 Requisits no funcionals

- **R19:** L'aplicació i la pàgina web han de ser compatible amb els navegadors més utilitzats, com Chrome, Firefox, Safari i Edge.
- **R20:** L'aplicació i la pàgina web han de tenir una interfície d'usuari intuïtiva, atractiva i adaptable a diferents dispositius i resolucions.
- **R21:** S'ha de garantir la seguretat i la privacitat de les dades dels usuaris, aplicant mesures com l'encriptació, l'autenticació i l'autorització.
- **R22:** S'han de complir amb les normatives legals i ètiques vigents, respectant la privacitat dels usuaris i la protecció de dades.
- **R23:** L'aplicació de tenir un alt nivell de disponibilitat, fiabilitat i rendiment, evitant errors, caigudes i retards en el servei.
- **R24:** Tot el sistema ha de ser escalable i mantenible, permetent afegir noves funcionalitats i corregir possibles defectes sense afectar el funcionament global.
- **R25:** L'aplicació ha de ser testejada, fent servir tècniques i eines adequades per assegurar la qualitat del codi i del producte final.

3.3 Diagrama de casos d'ús

En la **Figura 2**, hi ha il·lustrats els diferents casos d'ús de l'aplicació. Hi ha tres actors principals: usuaris de l'aplicació (propietaris o treballadors de restaurants), usuaris no autenticats que volen provar l'aplicació i administradors de Cookledge. Els usuaris no autenticats tindran accés a l'anàlisi en el *Dashboard* amb dades d'un usuari de prova.

En el diagrama s'ha simplificat els casos relacionats amb la gestió d'entitats. Cada cas inclou totes les operacions CRUD (*Create, Read, Update, Delete*). Els casos d'introduir arribades o realitzar comandes són més complexos i impliquen més elements, com el sistema de missatgeria per correu electrònic.

Figura 2. Diagrama de casos d'ús

4 Anàlisi dels requisits funcionals

Un cop s'han establert els requisits, s'han analitzat per tenir una visió i una guia sobre el què s'ha de fer a l'hora d'implementar l'aplicació.

4.1 Diagrama de Classes

En el diagrama de la **Figura 3** tenim quatre entitats principals: **Product**, **Provider**, **Warehouse** i **Restaurant**. Pel que fa a productes, n'hi ha de dos tipus: Producte i Recepta. La Recepta o producte semielaborat hereta de producte, ja que aquesta pot actuar com a tal. Una recepta està composta per diversos productes. Existeix una relació entre Producte i Magatzem on es guarden els diversos grups de productes per data de caducitat i localització dins el magatzem.

Per poder tenir constància dels diferents moviments d'entrada i sortida, existeix l'entitat **Movement** que registra la quantitat, el preu, i l'instant que s'ha realitzat. Els moviments es duen a terme en grups, **Arrival** i **Delivery** són les entitats que agrupen aquests moviments.

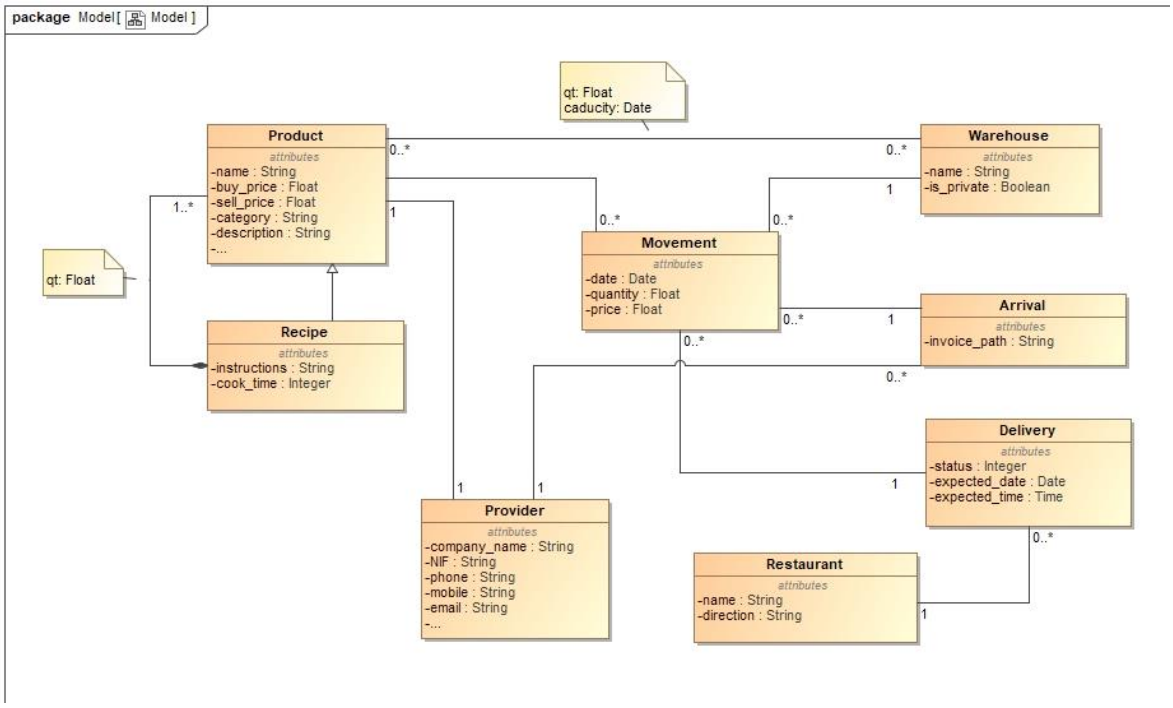

Figura 3. Diagrama de classes

4.2 Diagrames de seqüència dels casos d'ús

Entre les diferents operacions que l'usuari ha de poder realitzar mitjançant l'aplicació, s'ha realitzat els diagrames de seqüències dels casos d'ús de les dues operacions més recurrents: la introducció de dades quan arriben els productes al magatzem i la realització de comandes.

Crear Producte

A l'hora de crear un producte, es necessita tenir una llista de productes per indicar si el producte està repetit o no a la validació. També necessitem saber quins proveïdors tenim per poder assignar-ne un al producte creat (o editat). Posteriorment es crida a la API i s'actualitza la llista de productes. Els detalls es poden visualitzar en la **Figura 4**.

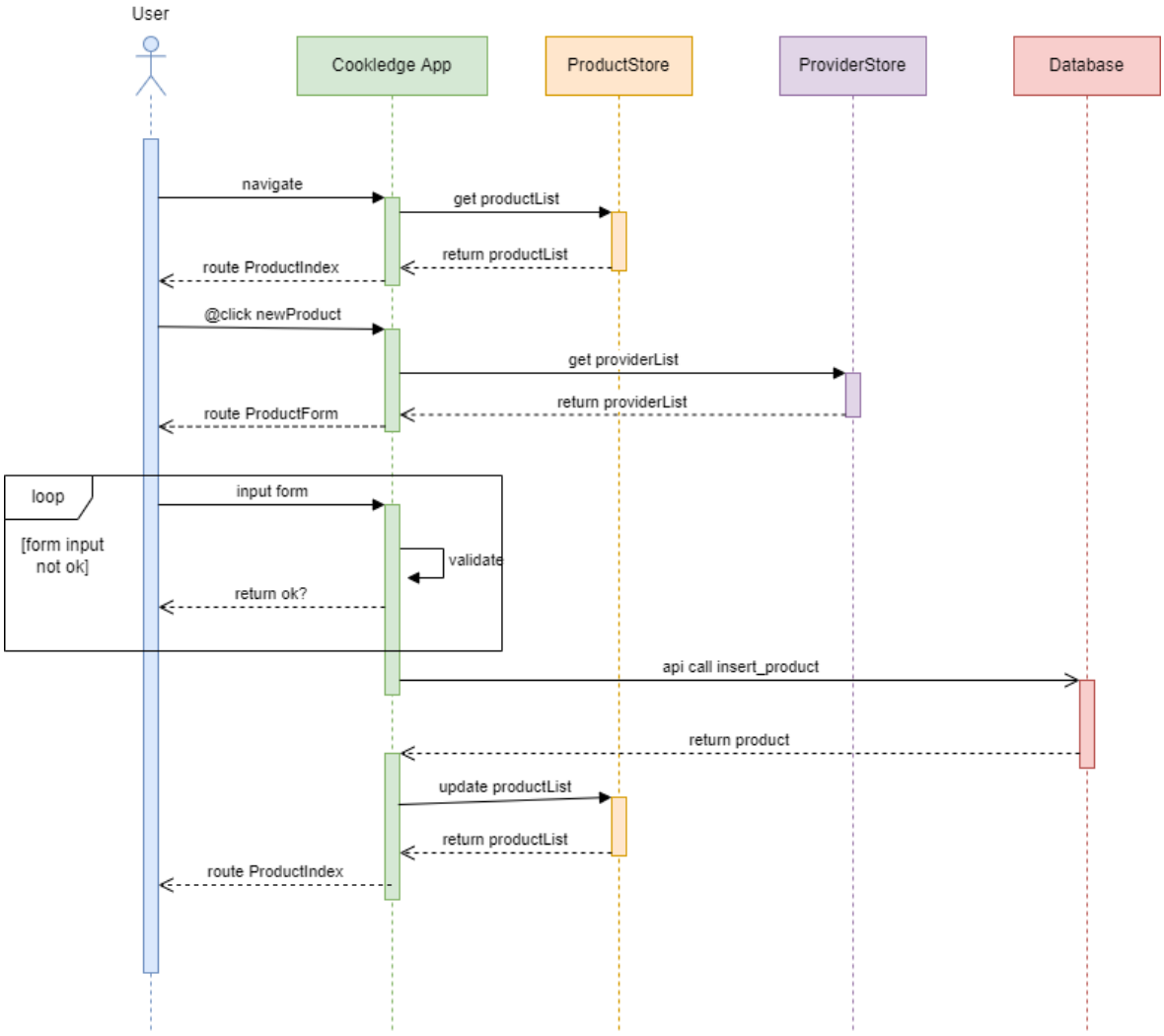

Figura 4. Diagrama de seqüències en la creació d'un producte

Introduir arribada

En la **Figura 5** tenim el diagrama de seqüencies per introduir arribades. Depenent de si el magatzem és propietat de l'usuari o gestionat, l'actor principal és l'usuari mateix o un treballador. Una arribada succeeix en un magatzem en un proveïdor en concret, si filtrem per proveïdor facilitem la feina de l'usuari a l'hora de seleccionar els productes específics.

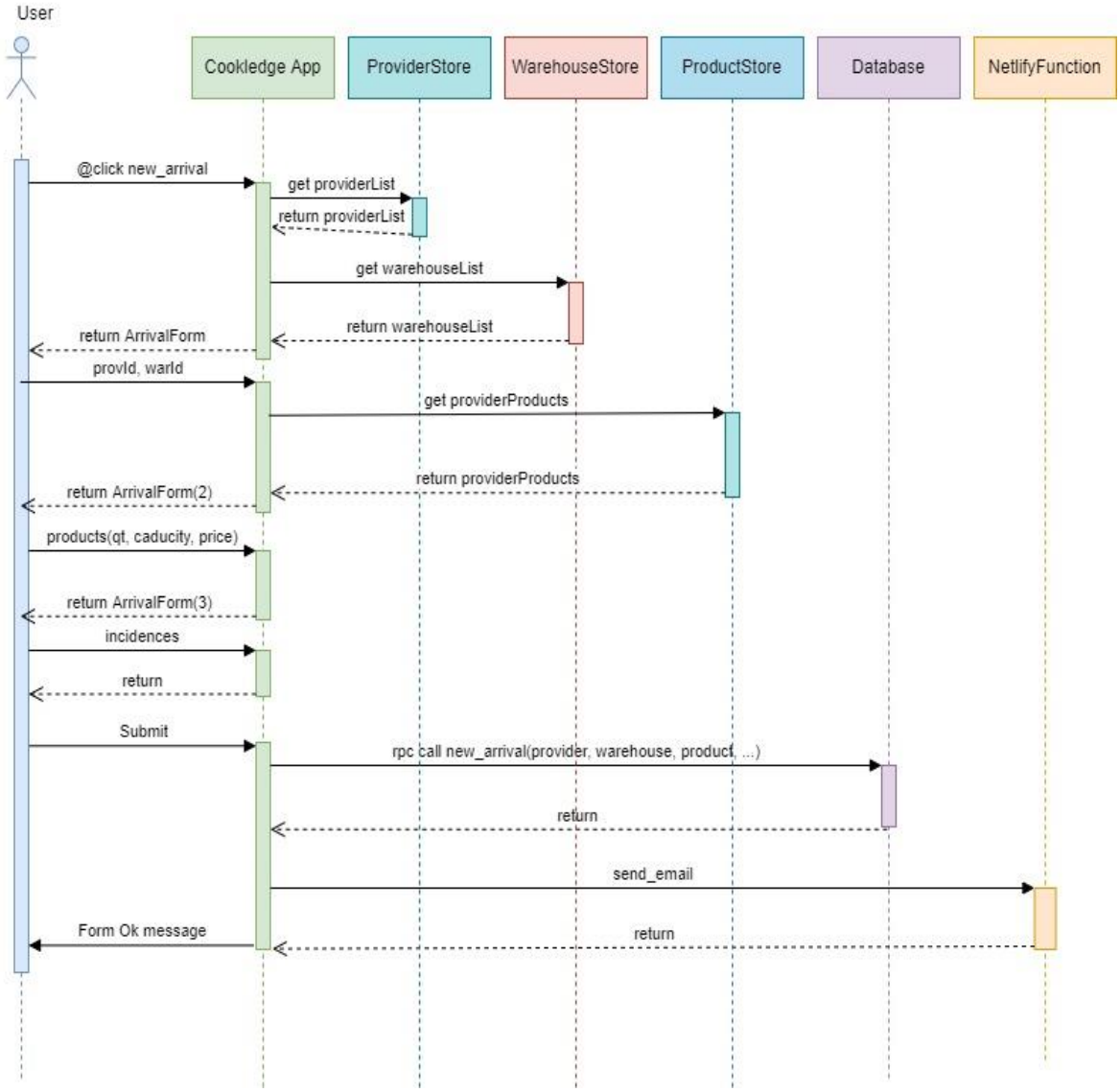

Figura 5. Diagrama de seqüències per introduir una arribada

Realitzar comanda

En la **Figura 6** tenim el diagrama de seqüencies per realitzar comandes. Una comanda succeeix en un magatzem i és per un restaurant en concret. En realitzar comanda, a diferència d'arribada no necessitem saber els proveïdors en concret, simplement filtrem els productes que no tenen quantitat.

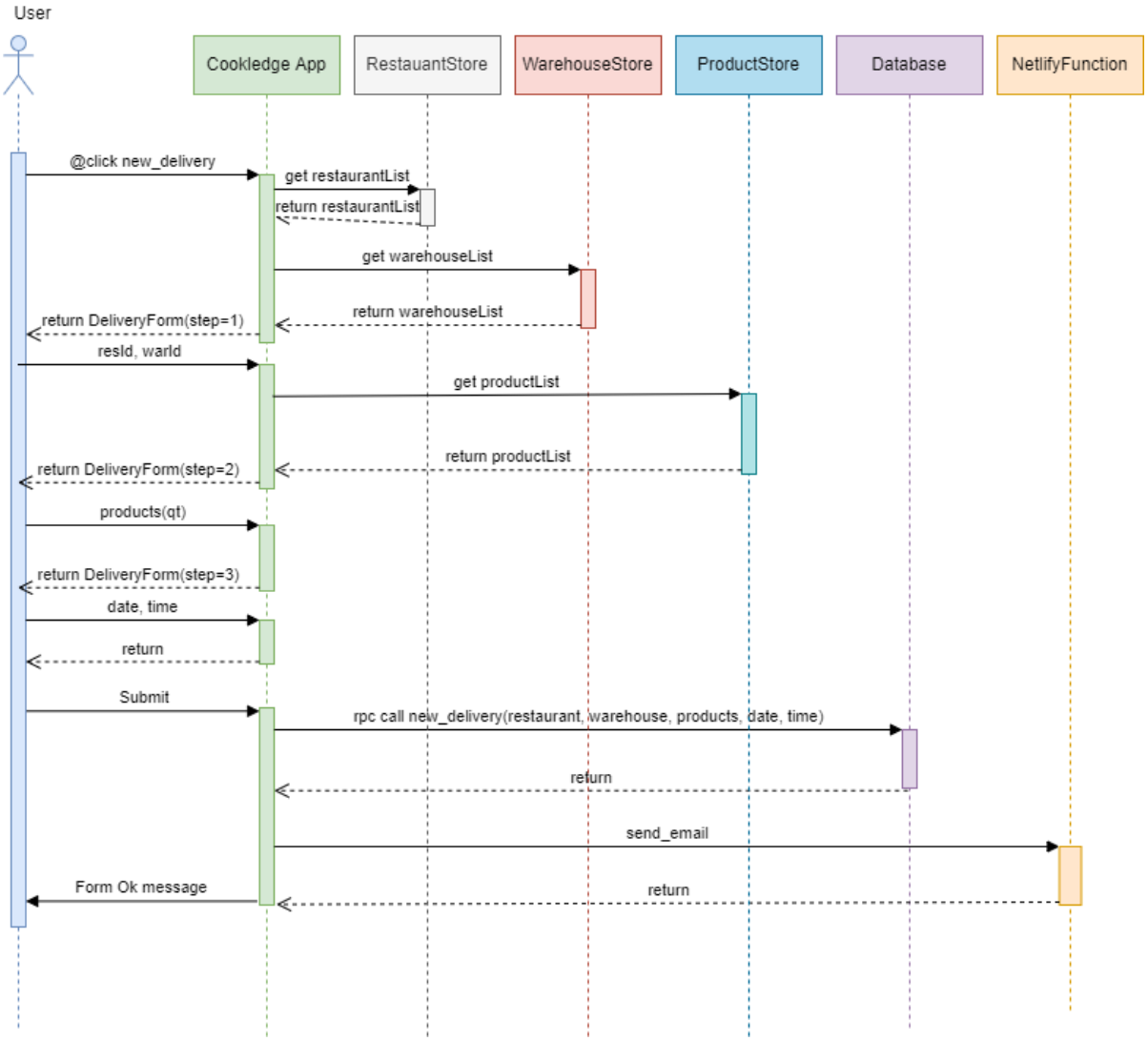

Figura 6. Diagrama de seqüències per realitzar una comanda

5 Elecció d'eines de treball i plataformes de *hosting*

Per decidir quines eines utilitzar per desenvolupar l'aplicació es van analitzar les possibles opcions i es va triar la que més adequada era en funció dels requisits, suport de la comunitat, la facilitat i el preu de *hosting*.

5.1 Aplicació web

Pel desenvolupament de l'aplicació s'ha fet servir **Vue**, un *framework* de Javascript que facilita la creació interfícies web reactives. Amb Vue es poden generar aplicacions web estàtiques de tipus **SPA** (*Single-Page Application*), que redueixen la càrrega al servidor i milloren l'experiència de l'usuari. També permet organitzar l'aplicació de manera modular, mitjançant components reutilitzables.

Per accelerar el procés de desenvolupament s'ha utilitzat **Quasar**, un *framework* basat en Vue que ofereix una llibreria de components UI adaptables a diferents dispositius i una integració amb **Capacitor** per a generar aplicacions natives per Android i iOS. [5]

A l'hora de triar *hosting* per l'aplicació s'ha comparat els preus de Netlify i Firebase. S'ha descartat GitHub Pages pel fet que el repositori ha de ser públic en el pla gratuït. S'ha tingut en compte principalment el preu de transferència de dades.

	Netlify	**Firebase**
Pla gratuït	Fins a 100 GB / mes	Fins a 10 GB/mes
Pagament	$19 → Fins 1000 GB/mes + $0.55/GB	10 GB + $0.15/GB

Taula 1. Taula comparativa de preus entre Netlify i Firebase [6] [12]

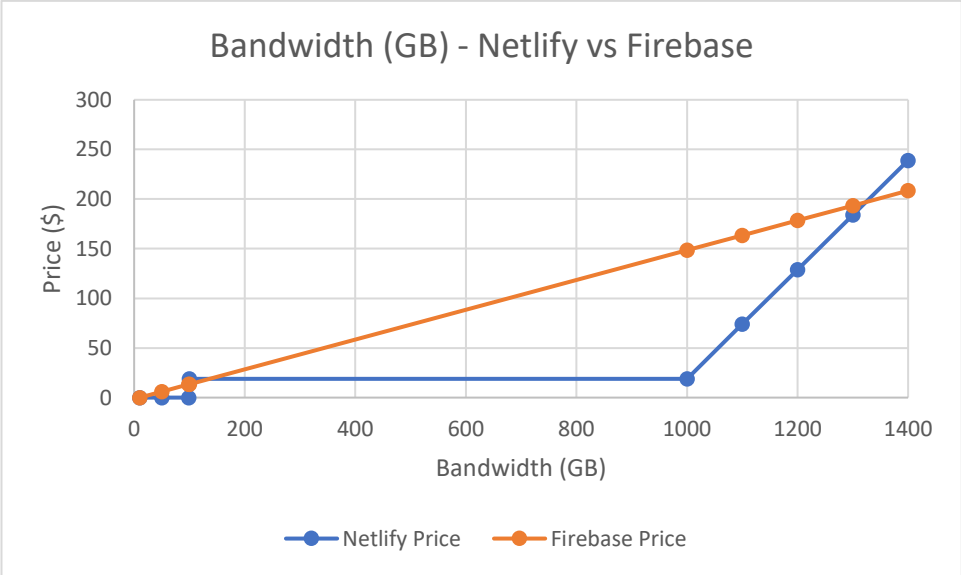

Figura 7. Gràfic comparatiu de preus entre Netlify i Firebase

S'ha triat **Netlify** perquè el seu pla gratuït ofereix 10 vegades més comparat amb el de Firebase. És cert que el preu per GB transferit és més barat a Firebase, però, com podem observar en la **Taula 1** el gràfic de la **Figura 7**, només compensa a partir de $197 o 1323.75 GB mensuals.

Durant el mes que s'ha estat realitzant actualitzacions i proves de forma contínua només s'ha gastat 100 MB de 1000 GB disponibles. Amb aquesta dada es podria extrapolar que al voltant de 10000 usuaris deixaria de ser gratuït.

5.2 Backend de l'aplicació

A l'hora d'escollir plataforma per *backend*, es va realitzar una comparativa de característiques i preus de diferents plataformes BaaS (*Backend as a Service*). Es va tenir en compte el preu i si el que oferia s'adaptava a les necessitats del projecte.

Després d'analitzar les diferents alternatives disponibles, com Firebase, AWS Amplify i MongoDB Atlas, es va optar per **Supabase** com a plataforma per al desenvolupament d'aquest projecte. Les raons principals d'aquesta elecció van ser les següents:

- Utilitza una base de dades relacional (**PostgreSQL**) que s'adapta més al model del projecte.
- Té un preu transparent i es pot estimar fàcilment.
- Incorpora un sistema d'autenticació i una API per interactuar amb les taules de la base de dades.
- Ofereix un servei d'emmagatzematge d'arxius i un pla gratuït amb recursos suficients per a la fase inicial d'aquest projecte.

Cal destacar que un dels casos d'ús de Supabase és la utilització del *backend* mitjançant RLS (*Row Level Security*), cosa que permet estalviar l'ús de *Middleware* entre l'aplicació web i la base de dades. [17]

5.3 Pàgina web de màrqueting

Pel que fa a la pàgina de màrqueting, s'ha triat **Astro**. Les pàgines de tipus SPA no tenen un bon SEO perquè els bots no tenen capacitat de llegir la pàgina correctament. També es realitza el *hosting* de amb Netlify.

Astro és un *framework* per crear pàgines estàtiques **MPA** (*Multi-page Application*) amb JavaScript basades en contingut, sense perdre la flexibilitat que aporten altres *frameworks* com Next, Nuxt o Gatsby i millorant la velocitat. Funciona mitjançant components reutilitzables que es combinen a l'hora de construir la pàgina. També conté integracions amb React o Vue per si un dels components vols que sigui reactiu.

6 Disseny

En aquest apartat es detallen els diferents aspectes referents l'arquitectura de l'aplicació, el disseny de la interfície de l'usuari i el disseny de la base de dades.

6.1 Arquitectura de l'aplicació

Vue és un *framework* basat en el patró **MVVM** (*Model-View-ViewModel*). Aquest patró permet separar la lògica de negoci (*Model*), la presentació (*View*) i la interacció entre ells (*ViewModel*). [7] Aquest es basa en el concepte de components, que són unitats reutilitzables de codi que poden contenir HTML, CSS i JavaScript. Els components es poden combinar per formar aplicacions complexes i dinàmiques.

Per gestionar l'estat compartit entre components s'ha utilitzat **Pinia**, una biblioteca per Vue.js que permet crear i accedir a *stores*, instàncies **Singleton** que contenen les dades i la lògica de l'aplicació. [8] Aquestes instàncies són reactives i s'hi pot accedir des de qualsevol component de l'aplicació. Des de les *stores* de Pinia es fan les peticions al *backend* a través de l'API de **Supabase**. Quan es rep la resposta, les dades s'emmagatzemen al *store* corresponent per evitar fer peticions innecessàries.

En el moment que es produeix una arribada o una comanda, també es fa una crida a una funció *serverless* de Netlify, on mitjançant la llibreria **NodeMailer** s'envia un correu electrònic al client amb les dades pertinents. En la **Figura 8** es poden apreciar com es troben connectats els diferents components.

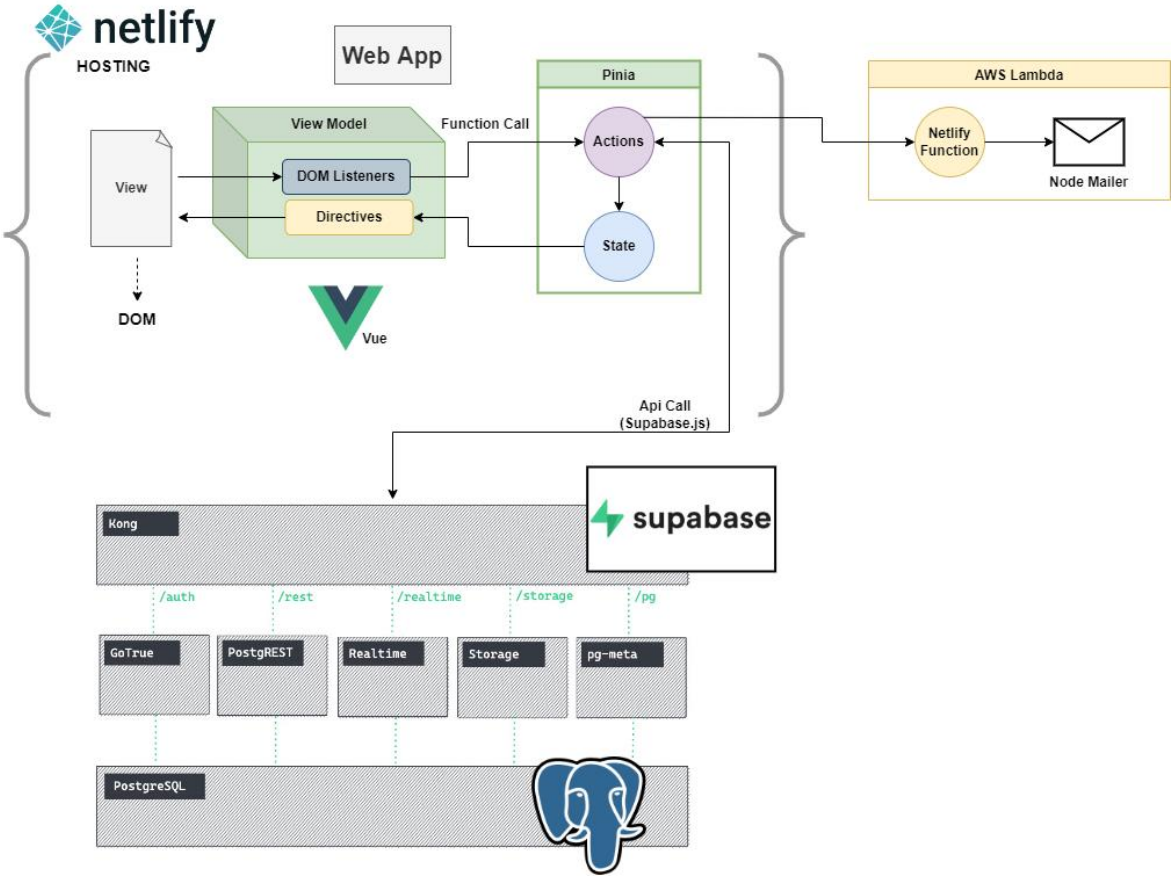

Figura 8. Arquitectura de l'aplicació [2]

6.2 Disseny de la interfície gràfica

A l'hora de dissenyar l'aplicació s'ha tingut en compte que ha de poder ser utilitzada per dispositius mòbils amb pantalles de diferents. S'han realitzat esbossos sobre les principals vistes de l'aplicació, aquests es poden apreciar en la **Figura 9** i la **Figura 10**. Hi han vistes com els índex dels diferents elements que són similars (Producte, Proveïdor...) i no s'han duplicat.

Figura 9. UI dispositiu mòbil

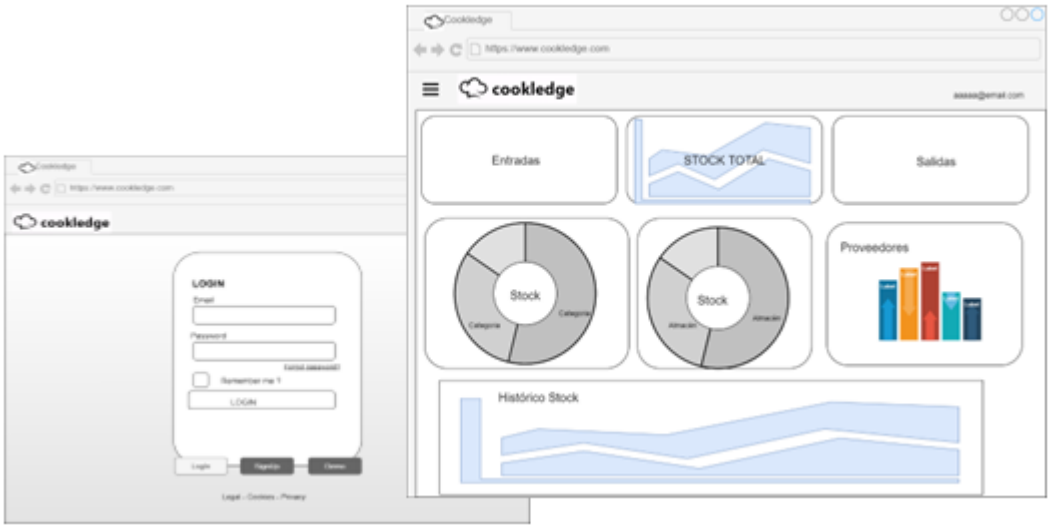

Figura 10. UI dispositiu amb pantalla gran

Una de les principals dificultats que s'ha trobat a l'hora de fer un disseny responsiu han sigut les taules. Aquestes es veuen molt bé en pantalles grans, però no caben en pantalles petites, sobretot en orientació vertical. Aquest fet requereix que s'hagi de realitzar *scrolling* horitzontal.

Per reduir la incidència d'aquest fenomen s'ha reduït la informació en pantalles petites, i, en el moment de fer clicar en una fila, aquesta es desplega o es redirigeix l'usuari a la pantalla específica de l'element.

6.3 Disseny de la base de dades

Seguint el diagrama de classes, s'ha utilitzat el diagrama de la **Figura 11** a l'hora d'implementar les diferents taules de la base de dades.

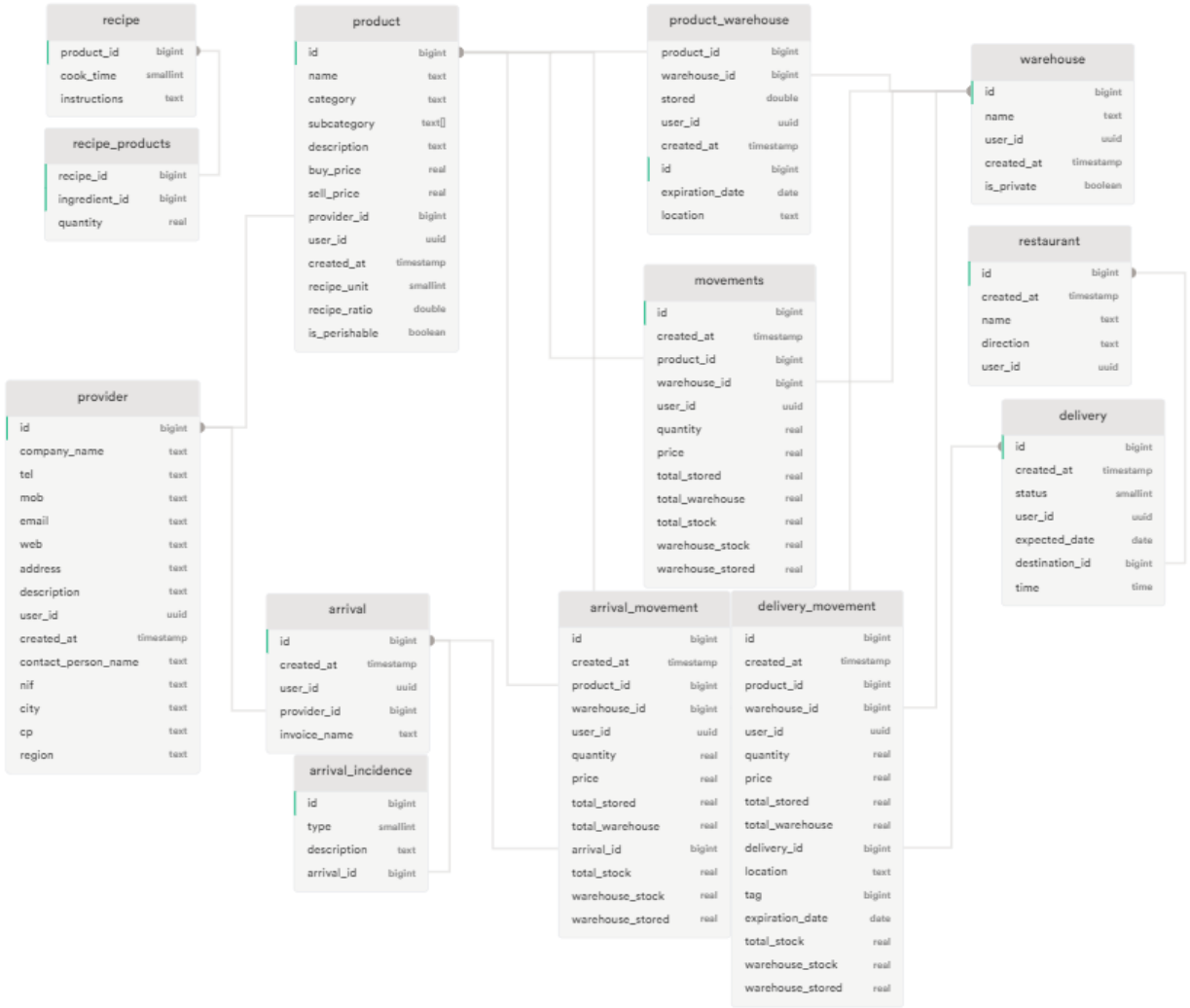

Figura 11. PostgreSQL Database Schema [18]

Supabase disposa d'una API que es crea automàticament en crear una taula i permet realitzar operacions CRUD. Aquesta API va molt bé per a taules senzilles, però de moment encara no suporta insercions en múltiples taules de forma transaccional. Aquesta limitació afecta les decisions preses a l'hora de dissenyar la base de dades.

En producte categoria i subcategoria no es troben normalitzats. Això s'ha decidit per simplificar el procés de creació de productes i receptes. La recepta s'ha normalitzat seguint el patró **Class Table Inheritance**, en el que la taula pare (*product*) i la taula filla (*recipe*) tenen una relació **One-to-One**. L'id de la taula filla és el mateix que la taula pare i serveix com a clau forana. El desavantatge principal és la necessitat d'implementar un RPC específic per crear noves receptes. [4]

Tenim principalment dos tipus de moviments: moviments d'arribada i moviments de sortida. Els moviments d'arribada requereixen tenir una columna amb l'id de la taula d'arribades mentre que els moviments de sortida necessiten guardar certes dades com la localització o caducitat del producte específic per ajudar al treballador a localitzar-lo en el magatzem. S'ha utilitzat la nomenclatura d'herència pròpia de PostgreSQL, que segueix el patró **Concrete Table Inheritance** on la taula pare (*movements*) s'utilitza com a *template* que les filles hereten.

L'herència en PostgreSQL té unes certes limitacions, ja que les taules filles no hereten les restriccions de la taula pare. Això implica que un moviment d'entrada i un de sortida puguin tenir ids repetits per defecte. També implica que s'hagin de reescriure els fk per cada taula filla. [15]

En el diagrama no es veu la taula d'usuaris, ja que pertany a un altre *schema*. La majoria de les taules tenen una columna amb *user_id* per aplicar polítiques d'accés RLS detallades en l'apartat de seguretat.

7 Implementació

En aquest apartat es detallen els diferents aspectes de la implementació de l'aplicació, el seu *backend* i la pàgina web de màrqueting.

7.1 Aplicació web

7.1.1 Estructura de l'aplicació

Les aplicacions complexes de Vue, i en extensió Quasar s'estructuren en vistes o pàgines gestionades per **vue-router**. Cada pàgina té un *path* únic amb el que el *router* pot dirigir l'usuari a la vista en concret. Les pàgines no funcionen de la mateixa manera que en una MPA sinó que es carreguen en segments específics del *template* anomenats <router-view />, d'aquesta manera existeix la possibilitat de tenir un *layout* comú amb diferents pàgines o mostrar diferents components.

En l'element de configuració *router* de l'aplicació que es mostra a continuació es poden observar tres *layouts* diferents (LoginLayout, MainLayout, Adminlayout) i les rutes filles de cada. Es realitza la importació de la pàgina de forma *lazy-loaded*, això vol dir que no es carrega tota l'aplicació sencera, sinó que es fa una petició dels diferents components de la pàgina en el moment d'accedir a la ruta.

```
const routes = [
  {
    path: '/:tab?',
    component: () => import('layouts/LoginLayout.vue'),
    children: [
      { path: '', component: () => import('pages/IndexPage.vue'),
name:'Login' },
      { path: 'recover', component: () =>
import('pages/auth/PasswordRecovery.vue') },
      { path: 'update', component: () =>
import('pages/auth/PasswordUpdate.vue') }
    ]
  },
  {
    path: '/app',
    component: () => import('layouts/MainLayout.vue'),
    children: [
      { path: '', component: () => import('pages/DashboardPage.vue'),
name: 'dashboard' },
      { path: 'products', component: { render: () => h(RouterView) },
children: [
        { path: '', component: () =>
import('pages/product/ProductIndex.vue'), name:'products' },
```

```
        { path: 'product:product_id?', name: 'product', component: () =>
import('pages/product/ProductView.vue') },
        { path: 'new-product', name:'new-product', component: () =>
import('pages/product/NewProduct.vue') }
      ]},
      { path: 'recipes', component: { render: () => h(RouterView) },
children: [
        { path: '', component: () =>
import('pages/recipe/RecipeIndex.vue') },
        { path: 'new-recipe', name:'new-recipe', component: () =>
import('pages/recipe/NewRecipe.vue') }
      ]},
      . . .
      { path: 'deliveries', component: { render: () => h(RouterView) },
children: [
        { path: '', component: () =>
import('pages/delivery/DeliveryIndex.vue') },
        { path: 'new-delivery', name:'new-delivery', component: () =>
import('pages/delivery/NewDelivery.vue') }
      ]},
    ]
  },
  {
    path: '/admin',
    component: () => import('layouts/AdminLayout.vue'),
    children: [
      { path: '', component: () => import('pages/admin/UserIndex.vue'),
        meta: { requiresAdmin: true }, name: 'users'},
      { path: 'deliveries', component: () =>
import('pages/delivery/DeliveryIndex.vue'),
        meta: { requiresAdmin: true } },
    ]
  },
  // Always leave this as last one, but you can also remove it
  {
    path: '/:catchAll(.*)*',
    component: () => import('pages/ErrorNotFound.vue')
  }
]
```

7.1.2 Estat de l'aplicació i sincronització amb el backend

Com s'ha vist en l'apartat d'arquitectura de l'aplicació **(6.1)**, per compartir l'estat de l'aplicació entre les diferents vistes s'ha utilitzat **Pinia Store**, que contenen les dades i la lògica de l'aplicació. A l'hora de mostrar elements en una vista existeix l'opció de fer crides desde els components directament al *backend* o carregar prèviament els elements a la *store* i cridar els mètodes de la *store*. Cada *store* conté una llista amb els elements corresponents de l'usuari (productes, magatzems, restaurants...), diferents funcions auxiliars i les crides corresponents a la seva API REST (*Representational State Transfer*).

S'ha de tenir en compte que si la pàgina es recarrega, el **localStorage** s'elimina i la informació guardada en Pinia es perd. Una solució per realitzar la sincronització consisteix a controlar si s'han carregat prèviament les dades a través d'una variable anomenada *loaded* en cada *store*. Cada cop que s'invoca **useStore()**, es cridarà a la seva funció de fetch corresponent. Si les dades ja estan carregades llavors no es farà res, en cas contrari es realitza la petició.

```
const loaded = ref(false)
fetchProducts()
 async function fetchProducts(reset = false){
   if(!loaded.value || reset){
     const { data, error } = await supabase.from('product')
     .select('*,' +
     'product_warehouse(id, warehouse_id, stored, expiration_date),' +
     'recipe(*, recipe_products(ingredient_id, quantity))'
     )
     if(error) console.log(error)
     if(data){
       addProducts(mapProducts(data))
       for(const product of productList.value){
         if(product.recipe) updateRecipePrice(product, productList.value)
       }
       //console.log(productList.value)
       loaded.value = true
     }
   }
 }
```

Una altra qüestió important és el tema de quins moviments portar al *frontent*. Ja s'ha vist en la previsió de l'ús de l'aplicació que serà la taula més llarga i no és eficient portar-los tots cada cop que s'inicia la sessió d'un usuari. L'estratègia que es podria seguir és portar els moviments del últim mes i/o limitar-ne el nombre.

7.1.3 Responsivitat

Per realitzar una aplicació responsiva per diferents dispositius s'ha utilitzat les diferents classes CSS que porta incorporades **Quasar**. Aquestes funcionen de forma similar al sistema de 12 columnes de Bootstrap. El lloc més clar on es pot apreciar és en el **Dashboard**:

```
<div class="row q-col-gutter-md">
  <div class="col-12 col-md-7 q-col-gutter-md">
    <div class="col-12">
      <SparkStock></SparkStock>
    </div>
    <div class="col-12">
      <ProductStockBar></ProductStockBar>
    </div>
    <div class="col-12">
      <PendingDeliveries></PendingDeliveries>
    </div>
  </div>
  <div class="col-12 col-md-5 q-col-gutter-md">
    <div class="col-12">
      <NearExpiration></NearExpiration>
    </div>
    <div class="col-12">
      <PieWarehouse></PieWarehouse>
    </div>
    <div class="col-12">
      <PieCategory></PieCategory>
    </div>
  </div>
</div>
```

En el fragment de codi anterior tenim dues agrupacions, una amb les classes "col-12 col-md-7" i l'altra amb la classe "col-12 col-md-5". Per a dispositius mòbils els diferents gràfics ocuparan tota l'amplitud de la pantalla (col-12). Pels dispositius majors a 1024px, tal com es pot apreciar a la **Figura 12**, la primera columna ocuparà 7/12 de l'espai disponible mentre que l'altra n'ocuparà 5/12.

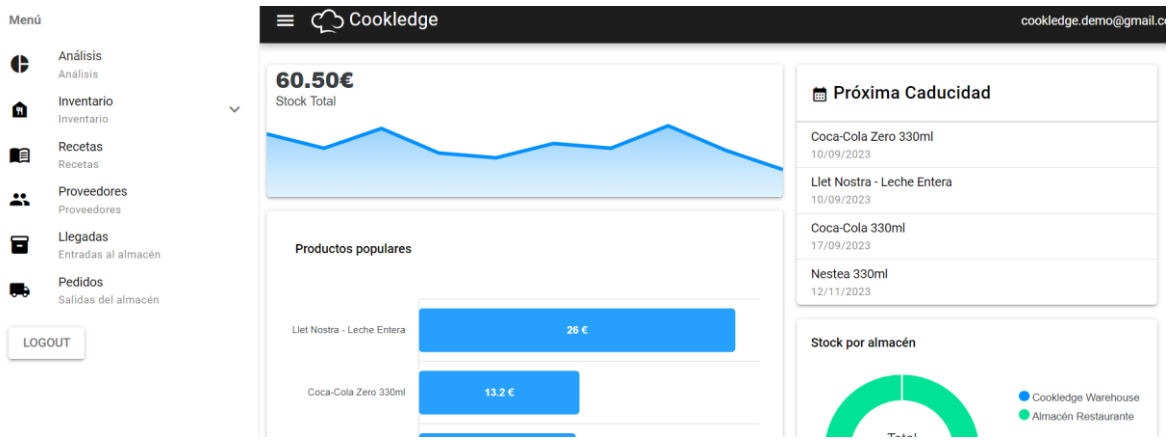

Figura 12. Vista principal de l'aplicació per dispositiu de pantalla gran

7.1.4 Reactivitat amb Vue

El sistema reactiu de Vue es basa convertint objectes Javascript senzills *proxies* reactius, els quals intercepten els *getters/setters* de les propietats d'aquests objectes. En el **Composition API** que s'utilitza en Vue3 existeixen principalment tres elements: *ref*, *computed* i *watchers*. [16]

- **ref**: S'utilitza per crear una referència reactiva d'un objecte.
- **computed**: Valor reactiu derivat d'altres dades reactives. Es recalcula quan un *ref* o un altre *computed* que el compon canvia de valor.
- **watcher**: funció que s'executa quan el *ref* que està observant canvia de valor.

Aquests elements són útils a l'hora de crear *forms* reactius. Cada valor d'un input és un referència que es pot controlar mitjançant un *watcher* per realitzar la validació al moment. També s'utilitzen per renderització condicional dels diferents elements del *template*. Les *stores* de Pinia també són reactives i també poden tenir referències i elements computats.

7.1.5 Anàlisis de dades

Per implementar els gràfics que apareixen en l'aplicació s'han provat dues llibreries **Chart.js** i **ApexCharts**. En una primera versió es va utilitzar Chart.js, però en refer l'aplicació amb Quasar, es va utilitzar ApexCharts per la seva simplicitat, exemples i documentació. [1]

Els anàlisis es basen principalment en les taules de moviments que registren l'històric de preus dels productes, quantitats i *stock*. El *dashboard* és la vista principal de l'aplicació de l'usuari que es veurà cada cop que entri, per aquesta raó s'han calculat amb anterioritat els diferents valors:

- **total_stock**: preu del *stock* total de tots els productes
- **warehouse_stock**: preu del *stock* per magatzem
- **total_stored**: unitats totals d'un producte en concret total en un instant de temps
- **warehouse_stored**: unitats d'un producte en concret per magatzem.

Guardar aquestes dades no segueix realment els principis de la normalització, però s'ha vist necessari per conservar un històric robust. Si només guardem la quantitat de producte per cada moviment, per tenir el gràfic s'hauria d'agafar la quantitat total i anar sumant o restant els moviments per cada producte i per cada magatzem. Les dades també serien fràgils en el sentit que si un moviment és erroni o s'elimina, tots els càlculs anteriors a aquella data serien erronis. S'ha de tenir en compte que realitzar aquests càlculs quan s'insereixen moviments també implica augmentar el temps d'inserció d'arribades i comandes. A l'hora de realitzar el disseny es va preferir tenir un temps d'inserció més llarg i tenir un temps de lectura més curt.

Haver calculat amb anterioritat aquests valors facilita molt la feina de creació dels gràfics. Entre ells destaquen els següents:

- **Històric del cost total de l'inventari**: Mostra l'evolució del *stock* en un gràfic de línia contínua per magatzem. S'utilitza **warehouse_stock**.
- **Històric de l'inventari d'un producte:** Mostra l'evolució de les unitats d'un producte per magatzem. S'utilitza **warehouse_stored**
- **Evolució del Preu d'un producte**: Mostra l'evolució del preu d'un producte en un gràfic lineal.

En alguns gràfics es realitzen els càlculs al *frontend* com els gràfics circulars amb *stock* per categoria o per magatzem. També s'han realitzat altres càlculs com el càlcul del *stock* total o la rotació de productes (*stock turnover rate*), que indica el temps que es tarda per vendre un *stock* determinat. La rotació es calcula amb la següent fórmula:

$$IR(t) = \frac{Cost\ of\ Goods\ Sold(t)}{Average\ Inventory} \quad (1)$$

Depenent del temps que s'agafi com a referència pot variar el seu valor, un període massa petit pot ser inexacte mentre que període massa gran no té en compte les diferents temporades. Un valor elevat indica que un producte és popular i està pocs dies en un magatzem, per tant ens estalviem costos de deteriorament de productes o emmagatzematge. També indica que necessitarem un sistema de logística més potent per gestionar el volum de vendes. El valor de l'inventari (**total_stock**) ja es troba precalculat, per tant, per calcular la rotació, haurem de sumar els costos dels moviments de sortida per un període determinat.

7.2 Backend

7.2.1 Supabase API

Supabase incorpora una **API REST** per realitzar operacions a les diferents taules. Un exemple seria el d'un proveïdor, on a inserir o actualitzar l'element s'introdueix un objecte Javascript amb els paràmetres adients:

```
const { data, error } = await supabase.from('provider').select('*')
```

```
const { data, error } = await
supabase.from('provider').insert(newProvider).select()
```

```
const { error } = await
supabase.from('provider').update(updatedProvider).eq('id', provider.id)
```

```
const { error } = await supabase.from('provider').delete().match({ 'id':
provId})
```

Es poden realitzar consultes complexes especificant el nom de la taula si les claus foranes han estat especificades prèviament a la base de dades:

```
const { data, error } = await supabase.from('product')
      .select('*,' +
      'product_warehouse(id, warehouse_id, stored, expiration_date),' +
      'recipe(*, recipe_products(ingredient_id, quantity))'
      )
```

7.2.2 PostgreSQL RPCs (Remote Procedure Call)

En el moment de realitzar aquest treball, aquesta API no suporta operacions d'inserció atòmiques en múltiples taules. Com a conseqüència s'han hagut de crear funcions que posteriorment es criden com a RPC des de l'aplicació. S'han programat amb el llenguatge **plpgsql**.

Crear recepta

En crear una recepta necessitem omplir la taula **product**, **recipe** i **recipe_products.** Per a fer-ho primer es crea l'element a la taula de producte per a tenir el seu id. Posteriorment, s'omple la taula de receptes i es fa un bucle amb els ingredients per omplir l'última taula.

Nova Arribada

A l'introduir una arribada d'un proveïdor s'ha de guardar informació en diferents taules. Els passos que es segueixen són els següents:

- Afegir l'arribada en **arrival.**
- Afegir les quantitats i dates de caducitat dels grups de productes en **product_movements** i afegir l'històric de moviments que conformen l'arribada en **arrival_movements,**
- Afegir incidències en **arrival_incidences.**
- Actualitzar el preu del producte en la taula **product.**

```
create or replace function new_arrival2(prov_id bigint, document
text,  movements arrival_item[], incidences incidence[] DEFAULT
'{}'::incidence[])
returns void as $$
declare
  arrival_id int;
  mov arrival_item;
  inc incidence;
begin
  insert into arrival (user_id, provider_id, invoice_name)
    values (auth.uid(), prov_id, document)
    returning id into arrival_id;
  foreach mov in array movements loop
    --new_item (prod_id, war_id, qt, exp_date, locat, price, arr_id)
    perform new_item(mov.prod_id, mov.war_id, mov.qt, mov.exp_date,
mov.location, mov.price, arrival_id);
  end loop;
  foreach inc in array incidences loop
    insert into arrival_incidence(arrival_id, type, description)
    values(arrival_id, inc.type, inc.description);
  end loop;
end;
$$ language plpgsql;
```

Per simplificar la funció dins del bucle de moviments es crida a una funció **new_item** que s'encarrega de fer les operacions per cada producte (moviments, quantitats i preu).

```
create or replace function new_item(
  prod_id bigint, war_id bigint,
  qt real, exp_date date, locat text,
  price real default null, arr_id int default null)
returns void as $$
declare
  tot_w real;
  tot real;
begin
  --Check if can access product (Row Level Security)
  insert into product_warehouse (
    product_id, warehouse_id, user_id,
    stored, expiration_date, location)
  values (
    prod_id, war_id, auth.uid(),
    qt, exp_date, locat);

  --New Movement & Update price
  select sum(stored) into tot_w from product_warehouse where
product_id=prod_id and warehouse_id=war_id;
  select sum(stored) into tot from product_warehouse where
product_id=prod_id;
  if price is not null then
    update product set buy_price = price where id = prod_id;
    insert into arrival_movement (
      user_id, product_id, warehouse_id, arrival_id,
      quantity, price, total_warehouse, total_stored
    )
    values(
      auth.uid(), prod_id, war_id, arr_id,
      qt, price, tot_w, tot
    );
  else
    insert into arrival_movement (
      user_id, product_id, warehouse_id, arrival_id,
      quantity, price, total_warehouse, total_stored
    )
    values(
      auth.uid(), prod_id, war_id, arr_id,
      qt, (select buy_price from product where id = prod_id), tot_w, tot
    );
  end if;
end;
$$ language plpgsql;
```

Nova Comanda

De forma similar a nova arribada, es necessita afegir informació en diferents taules. El codi és similar al de nova arribada, les diferències principals són els paràmetres d'entrada i el bucle que s'ha de realitzar al anar restant quantitats de product_warehouse. Te algun pas menys ja que no es registren incidències. Es segueixen els següents passos.

- Afegir comanda en **delivery**
- Treure elements d'un producte específic en **product_warehouse** i afegir moviments en **delivery_movements** a partir del id del delivery.

Altres

A part dels RPC anteriors, també existeixen altres funcions auxiliars com el càlcul del *stock* total, càlcul del *stock* per magatzem o el cost dels moviments de sortida en un període determinat.

```
CREATE OR REPLACE FUNCTION total_cost(start_date DATE, end_date DATE
DEFAULT CURRENT_DATE, product_id BIGINT DEFAULT NULL)
RETURNS NUMERIC AS $$
DECLARE
  sum_cost NUMERIC;
BEGIN
  IF product_id IS NULL THEN
    SELECT SUM(price * quantity) INTO sum_cost
    FROM delivery_movement
    WHERE created_at BETWEEN start_date AND end_date;
  ELSE
    SELECT SUM(price * quantity) INTO sum_cost
    FROM delivery_movement
    WHERE created_at BETWEEN start_date AND end_date
    AND product_id = delivery_movement.product_id;
  END IF;
  RETURN sum_cost;
END;
$$ LANGUAGE plpgsql;
```

7.2.3 Node Mailer

Per enviar correus als usuaris s'ha utilitzat una funció "serverless" de Netlify que corren sobre AWS (AWS Lambda). Les funcions es defineixen en la carpeta /funcions i es compilen de forma simultània amb l'aplicació.

Per enviar el missatge s'utilitza la llibreria NodeMailer utilitzant el compte de Gmail de Cookledge com a emissor, Per fer servir un compte de correu de Gmail es necessita habilitar el 2FA (Two Factor Auhtentication) en el compte i crear una "*Third Party App Password*". En la funció s'ha utilitzat un *string template* amb estils CSS per donar estil al correu que s'envia a l'usuari. S'envien correus rebre arribades de productes o realitzar comandes.

7.3 Pàgina web

7.3.1 Estructura de l'aplicació

Astro és un *framework* dissenyat per ser ràpid que permet construir pagines web a traves de components. La pàgina web es renderitza a un HTM estàtic en el moment consumint el codi Javascript de la part del servidor a les capçaleres respectives dels diferents components, enviant menys Javascript al navegador per millorar-ne la velocitat. Es poden utilitzar arxius de tipus **md** (*markdown*) que s'utilitzen per incorporar diferents pàgines que han de seguir un estil similar i contenen majoritàriament text.

En construir la pàgina, el nom del fitxer dins la carpeta */src/pages*, que es pot apreciar en la **Figura 13**, serà el nom que apareix al *path* del navegador de les diferents pàgines del projecte. Cada pàgina pot estar englobat per un *layout*, però a diferència de Vue, aquest s'importa directament dins el *template* de cada pàgina.

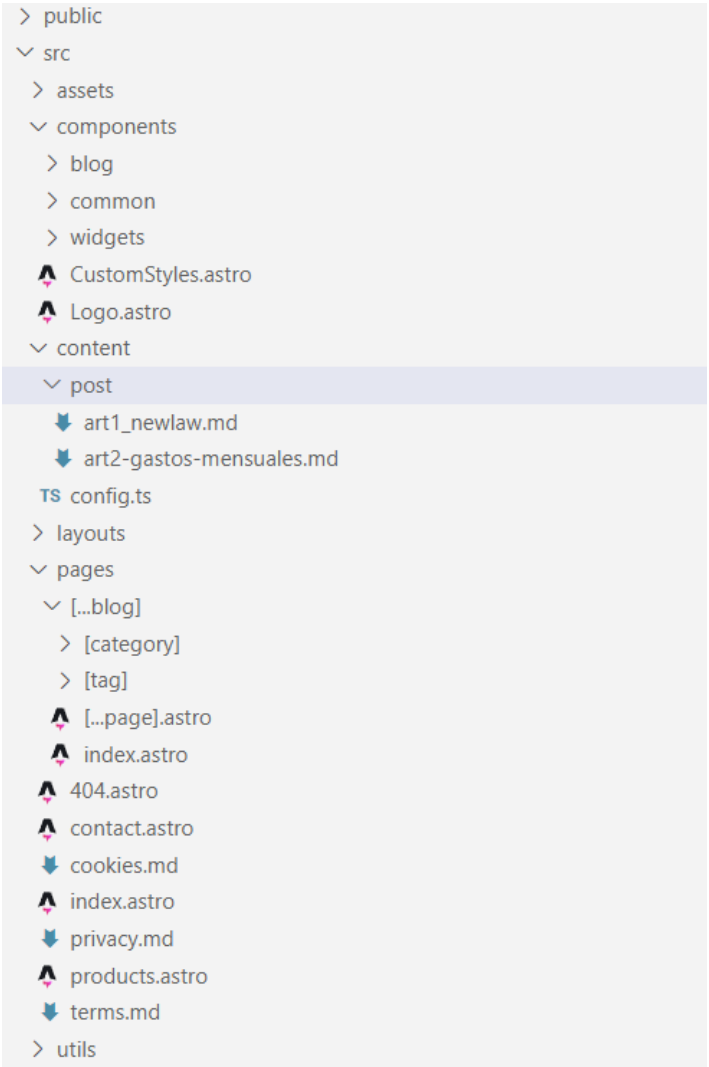

Figura 13. Carpeta de projecte de la pàgina web amb Astro

Respecte als arxius de blog, dins l'arxiu **/src/pages/[...blog]/[...page].astro** s'utilita una funció anomenada **getStaticPaths(),** aquesta genera una pàgina per cada arxiu md de la carpeta **/content/post** amb un estil específic indicat al *template*. Les pàgines que indexen els blogs funcionen de forma similar, realitzant un bucle dins el *template* afegint els articles corresponents.

7.3.2 SEO

Per incorporar totes les definicions de metadades s'ha utilitzat la extensió **@astrolib/seo**, incorporant el component **AstroSeo** dins del component **MetaTags** que s'incorpora al *layout* de cada pàgina. [13] D'aquesta manera es poden carregar dinàmicament els títols i les descripcions al fer el *build* mitjançant comentaris a la capçalera en els diferents arxius *markdown* o una variable meta a les capçaleres de les diferents pàgines.

Això aporta l'avantatge de que cada pàgina pugui tenir un títol i una descripció especifica. Tenir diferents títols també va bé a l'hora d'observar el número de visites per cada pàgina en **Google Analítics**, aquests es poden apreciar a la **Figura 14**.

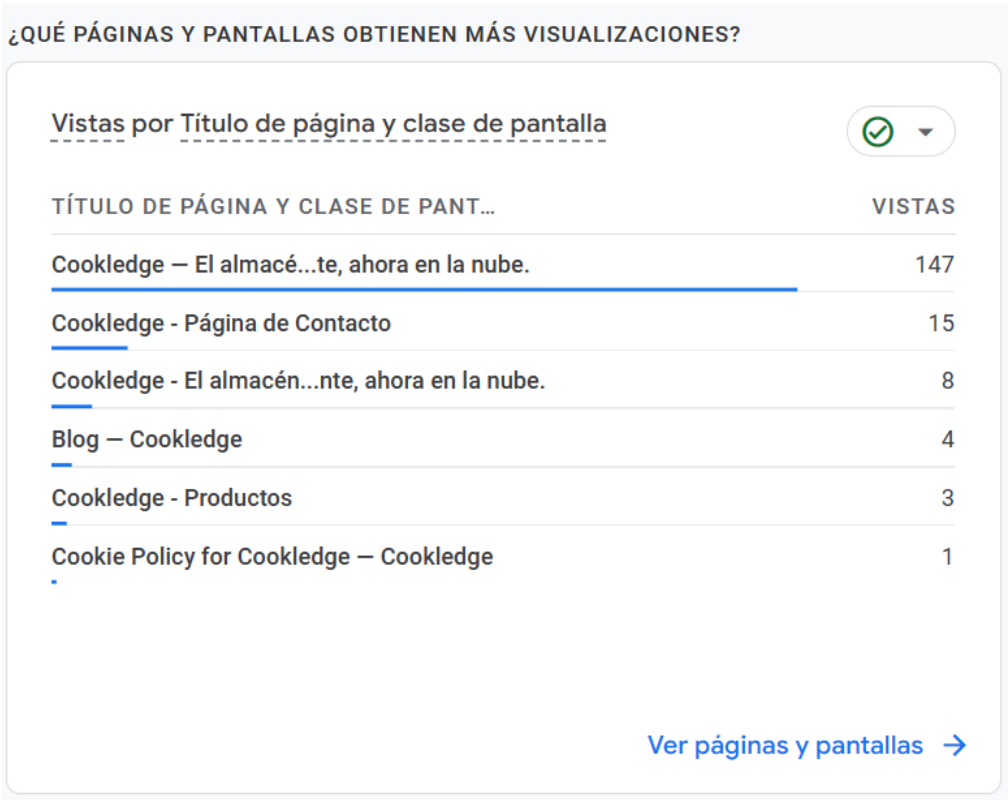

Figura 14. Visites per títol de pantalla en cookledge.com

7.3.3 Google Analytics i Cookies

Per seguir el reglament europeu GDPR (*General Data Protection Regulation*) s'ha implementat un *banner* de *cookies* per activar-les només quan aquestes siguin acceptades. [11] Per a fer-ho s'ha utilitzat la funció **loadScript** per carregar l'script **gtag** de Google en el cas que l'usuari accepti les *cookies*. També es guarda a *localStorage* una variable anomenada **accept_cookies** per saber si l'usuari ha acceptat anteriorment o no i no aparegui el *banner* en cada pàgina.

```
  const gtagString = 'https://www.googletagmanager.com/gtag/js?id=' + id
  function loadScript({ src, type }) {
    return new Promise((resolve, reject) => {
      const script = document.createElement("script");
      script.src = src;
      script.type = type;
      script.onload = resolve;
      script.onerror = reject;
      document.head.appendChild(script);
    });
  document.getElementById('accept-cookie')?.addEventListener("click", ()
=> {
    cookieBanner?.classList.add('hidden');
    localStorage.setItem("accept_cookies", "true")
    loadScript({ src: gtagString, type: "text/partytown" });
  })
```

Per millorar la eficiència s'utilitza @astrojs/partytown, que trasllada tots els scripts de analítics a un **web worker** enlloc d'executar-los al *thread* principal. [10] En la **Figura 15** s'aprecia com es poden visualitzar les diferents fonts de la pàgina a través de les eines de desenvolupador de Google Chrome.

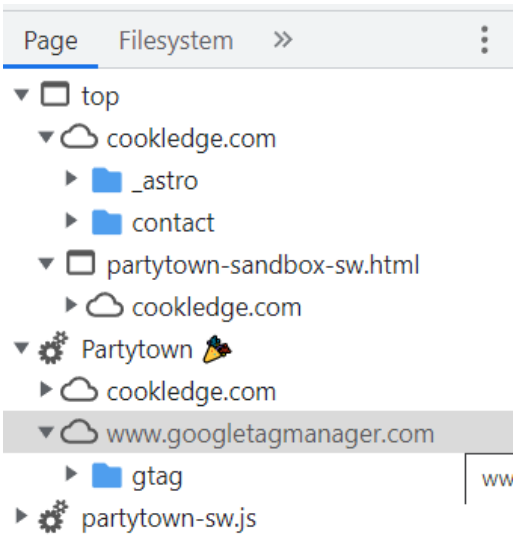

Figura 15. Diferents fonts de la pàgina cookledge.com

8 Seguretat

El *backend* de Supabase té incorporat un sistema d'autenticació on hi ha dues parts diferenciades: autenticació i autorització. Autenticació consisteix en saber si algú ha de poder entrar, i saber qui és. Autorització es refereix a què pot fer aquesta persona un cop ja està a dins.

8.1 Autenticació

Hi ha vàries formes d'autenticar usuaris, en aquest projecte s'ha implementat l'autenticació per correu electrònic.

Quan es fa una petició d'iniciar sessió, Supabase et retorna un esdeveniment amb la informació del usuari, aquesta es guardada en **userStore.js** (Pinia). Existeixen diferents esdeveniments, aquests es tracten en la següent *callback* definit al component principal App.vue

```
supabase.auth.onAuthStateChange((event, session) => {
  console.log(event)

  if ((event == "SIGNED_IN") || (event == "INITIAL_SESSION" && session != 
null && !store.user)) {
    store.setUser(session)
    loadData()
  }
 (event == "INITIAL_SESSION" && session == null) {
    store.setDemo()
    loadData()
  }
  else if (event == "PASSWORD_RECOVERY") {
    //Show Password Reset Screen
  }
  else if (event == "SIGNED_OUT") {
    store.user = undefined
    resetStores()
    router.push('/')
  }

})
```

En el sistema d'autenticació s'ha activat la opció de requerir confirmació per correu. Quan es crea un usuari, aquest no tindrà accés fins que accedeixi al enllaç de confirmació. D'aquesta manera verifiquem que es una persona real o prevenim abusos del sistema.

Per redirigir l'usuari a l'hora que inicia l'aplicació s'han aplicat **Route Guards**, on el Router de Vue realitza una sèrie de comprovacions abans de dirigir l'usuari a una vista en concret:

```
  Router.beforeEach(async (to, from) => {
    const userStore = useUserStore()
    const signedIn = userStore.user
    const demo = userStore.isDemo
    const admin = userStore.superUser

    //Access without login or demo
    if( !signedIn && !demo && to.name !== 'Login') return { name: 'Login'
}

    //Reload page when user logged in
    if( signedIn && to.name == 'Login'){
      if(admin) return { name: 'users'}
      else return { name: 'dashboard'}
    }

    if( to.meta.requiresAdmin && !admin) return { name: 'dashboard'}
  })
```

RouteGuards no ofereixen cap tipus de seguretat ja que pertany al *frontend* i un usuari pot intentar modificar el *localStorage* o el codi en sí. Es podria fer una crida verificant l'usuari en cada canvi de vista enlloc d'accedir al *userStore*, però tampoc és un mètode infal·lible. Tots els aspectes de seguretat relacionats amb l'autorització s'ha d'implementar en el *backend*, si un usuari atacant no administrador intenta entrar a la vista del administrador d'usuaris, veurà la pàgina amb els components però simplement tindrà una llista buida.

Supabase també incorpora mètodes relacionats amb l'administració d'usuaris a través de **supabase.auth.admin** que requereixen una clau secreta. Aquests mètodes només s'han de cridar en un servidor conegut de confiança i mai s'ha d'exposar aquesta clau al navegador. De moment no s'ha implementat control d'usuaris en l'aplicació de l'administrador ja que no es troba en la llista de requisits inicial.

8.2 Autorització en el backend (Row Level Security)

Per saber quines operacions pot realitzar un usuari s'ha utilitzat polítiques RLS de PostgreSQL. [3] Funciona de la següent manera:

1. Es registra un usuari. Aquest s'incorpora a la taula auth.users,
2. Supabase retorna un nou JWT, que conté l'UUID de l'usuari.
3. Cada petició del usuari al *backend*, s'envia el JWT.
4. PostgreSQL inspecciona el JWT per determinar l'usuari que fa la petició.
5. L'UID del usuari es pot utilitzar en polítiques per restringir l'accés a les files

Un exemple senzill de política d'accés implementat és restringir l'accés només al usuari propietari d'un producte, per realitzar-ho la taula ha de tenir una columna amb el uuid del usuari.

```
CREATE POLICY "Enable all operations by user_id" ON "public"."product"
AS PERMISSIVE FOR ALL TO public
USING (auth.uid = user_id)
```

Per realitzar la demostració, s'ha creat un usuari nou i donat permís de lectura a qualsevol usuari no autenticat. D'aquesta manera, quan un usuari autenticat vulgui visualitzar els seus productes, només obtindrà els seus i no els de demostració.

```
CREATE POLICY "Enable read access for DEMO" ON "public"."product"
FOR SELECT TO public
USING ((user_id = 'XXXX'::uuid) AND (auth.uid() IS NULL))
```

Per l'administrador s'ha fet de forma similar, però donant tots els permisos. La única diferència és que saber qui és administrador és un procés més complex ja que no hi ha cap funció similar a auth.uid().

```
CREATE POLICY "Enable all operations to admin." ON "public"."arrival"
AS PERMISSIVE FOR ALL
TO public
USING ((auth.is_admin(auth.uid())))
```

Per indicar quin usuari és admin s'ha creat una funció per llegir la columna is_super_user de la taula auth.users. Aquesta funció està definida amb **SECURITY DEFINER**, un mode que indica que la funció s'executarà amb els privilegis del usuari propietari de la funció, i no qui la crida. Pot ser útil si es vol que usuaris no privilegiats tinguin un accés privilegiat a certes taules, en específic el *schema* **auth** de Supabase

```
CREATE or REPLACE FUNCTION auth.is_admin (IN user_id uuid)RETURNS boolean
LANGUAGE plpgsql SECURITY DEFINER AS $$
DECLARE is_super_user boolean;
BEGIN
  SELECT is_super_admin INTO is_super_user FROM auth.users WHERE id =
user_id;
  IF is_super_user THEN RETURN true;
  ELSE RETURN false;
  END IF;
END;
$$;
```

Afegir regles significa afegir consultes WHERE addicionals a cada taula, cosa que impacta el rendiment. S'han atorgat els permisos detallats en la **Taula 2** en les diferents taules de la base de dades:

	Operació	Demo	User	Admin
Product / Recipe	Select	✓	✓	✓
	Insert		✓	✓
	Update		✓	✓
	Delete		✓	✓
Provider	Select	✓	✓	✓
	Insert		✓	✓
	Update		✓	✓
	Delete		✓	✓
Movement / Arrival / Delivery	Select	✓	✓	✓
	Insert		✓	✓
	Update		*	✓
	Delete		*	✓
Warehouse / Restaurant	Select	✓	✓	✓
	Insert		✓	✓
	Update		✓	✓
	Delete		✓	✓

Taula 2. Permisos pels diferents rols d'usuari

*Un usuari no hauria de poder modificar les quantitats en un magatzem gestionat per Cookledge.

8.3 Protecció de columnes

Les regles RLS anteriors protegeixen l'accés a una fila en concret, però no determina quines columnes pot llegir o modificar l'usuari un cop hi tingui accés. Existeixen vàries maneres de protegir columnes:

- **Mitjançant Trigger:** Una manera de protegir la columna és mitjançant *trigger* que s'executa BEFORE UPDATE.

```
CREATE OR REPLACE FUNCTION protect_created_at()
RETURNS trigger AS $$
BEGIN
  IF OLD.created_at IS DISTINCT FROM NEW.created_at THEN
    RAISE EXCEPTION 'Cannot modify created_at column';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER protect_created_at_trigger
BEFORE UPDATE ON your_table
FOR EACH ROW EXECUTE PROCEDURE protect_created_at();
```

- **Mitjançant Policy Check:** Una altra manera és mitjançant POLICY WITH CHECK al fer UPDATE.

```
CREATE POLICY "test" ON posts FOR UPDATE
  USING (true)
  WITH CHECK (old.created_at = new.created_at);
```

- **Mitjançant Vistes o Revoke/Grant:** Una altra manera seria crear una vista parcial de la taula i no fer pública la taula sencera o utilitzar revoke and grant en columnes específiques per rols d'usuari específics.

S'ha decidit utilitzar **Policy Check**. Les vistes costen de mantenir si les columnes de les taules originals canvien freqüentment, cosa que pot passar sovint quan encara s'està desenvolupant. La *policy* RLS pot ser més lenta que l'accés amb vistes, ja que incorpora comprovacions addicionals a cada fila, però és més fàcil de mantenir. La diferència principal entre Trigger i RLS és que Trigger llança error mentre que RLS simplement no et dona accés a la fila a l'hora de fer *l'update*.

9 Avaluació

9.1 Disseny dels casos de prova

9.1.1 Aplicació Web

El funcionament de l'aplicació depèn de que totes les parts funcionin i estiguin connectades. Principalment s'ha de mirar que funcionin els diferents **casos d'ús** i la implementació del sistema d'autenticació. També s'ha de revisar que les redireccions en els diferents tipus d'usuari funcionin i comprovar que es visualitzi bé en diferents dispositius.

Per avaluar l'experiència d'usuari i trobar possibles errors s'ha contactat amb possibles **early adopters** per provar l'aplicació i que donin **feedback**.

9.1.2 Backend

API REST

La API de Supabase no fa falta testejar-la per si sola, però la seva correcta execució depèn de com es crida des de el *frontend*, si existeixen claus foranes a l'hora de seleccionar múltiples taules o si les policies s'han aplicat correctament. Una de les maneres en que pot fallar és quan es realitza una petició però l'objecte que es passa té algun paràmetre de més o un nom diferent al del *backend*. Per aquesta raó s'han de realitzar les proves utilitzant la API en cada taula per cada rol, tenint en compte la taula de permisos de l'apartat de RLS (8.2).

RPCs

Les funcions creades manualment s'han de testejar, ja sigui mitjançant tests unitaris o comprovant la integració amb el *frontend*. Posteriorment s'ha de repetir el procés anterior de comprovació de permisos.

9.1.3 Pàgina de màrqueting

En la pàgina web s'ha de comprovar els diferents aspectes:

- **Rendiment**: Les imatges estan optimitzades? TTL, ...
- **Accessibilitat**: Les diferents pàgines es vegi de forma correcta per dispositius mòbils?
- **SEO**: Com apareix la pàgina en diferents navegadors? S'han aplicat les metadades correctament?
- **Bones pràctiques de programació**, per facilitar el manteniment de la pàgina.

Per realitzar aquestes comprovacions es pot utilitzar Google Lighthouse. Una eina automàtica de Google per millorar la qualitat de la teva pàgina. Quan s'executa Lighthouse, aquest genera un informe amb una valoració i una llista dels aspectes a millorar.

També s'ha de comprovar que s'hagi integrat Google Analytics correctament i revisar el funcionament del *banner* de *cookies*. Les *cookies* de Google només han d'aparèixer quan l'usuari les hagi acceptat i no abans.

9.2 Resultats

9.2.1 Frontend

Els casos d'ús i els seus requisits funcionals associats, establerts en l'apartat **3.1.1**, s'han testejat de forma manual abans de publicar-se. Posteriorment, s'ha corregit qualsevol classe d'error reportat pels *early adopters* que actualment estan utilitzant l'aplicació i s'ha tingut en compte les seves valoracions. Es poden visualitzar els resultats en la **Taula 3**.

Cas d'ús	Comentari	Test superat
01. SignUp		✓
02. LogIn		✓
03. ResetPassword	El missatge de recuperació a vegades arriba a la bústia de *spam*. S'ha afegit un missatge avisant al usuari.	✓*
05. ManageProducts **(R1)**		✓
06. ManageProviders **(R2)**	Un restaurant demana poder tenir varis proveïdors per un mateix producte. Es tindrà en compte per millores futures.	✓
07. ManageRecipes **(R3)**	Falta implementar l'RPC d'edició de receptes.	1/2
08. ManageWarehouse **(R2)**		✓
09. ManageRestaurant		✓
10. WatchAnalysis **(R7)**	El zoom d'algun gràfic no funciona correctament.	3/4
11. OrderDeliveries **(R4/R8)**		✓
12. IntroduceArrivals **(R4/R5/R8)**	Falta adaptar RPC per administrador	3/4

Taula 3. Tests dels casos d'ús

S'ha implementat la *demo* en el *backend* a partir d'un compte específic d'usuari. És relativament senzill modificar les dades per adaptar-les a les necessitats dels clients (**R9**).

L'aplicació incorpora vistes i permisos d'administrador (**R10**) però falta implementar alguna funcionalitat del *backend*. No s'ha prioritzat aquest apartat degut a que aquest estiu només es treballarà amb un client a través del seu compte.

Pel que fa els requisits no funcionals, l'adaptació de l'aplicació a diferents mides de dispositiu (**R19, R20**) s'ha testejat mentre es desenvolupava l'aplicació mitjançant les eines de desenvolupador de Google Chrome. Posteriorment s'ha provat a través del telèfon mòbil, tauleta, ordinador portàtil i pantalla gran de sobretaula.

9.2.2 Backend

Per realitzar tests del *backend* s'ha utilitzat **pgTAP**, una extensió de PostgreSQL que permet realitzar Unit Tests. La extensió et permet realitzar tests de taules, columnes, polítiques RLS i funcions. [14] (**R25**)

API REST

Pel bon funcionament dels permisos, s'ha provat de realitzar les diferents operacions CRUD per cada tipus d'usuari i comprovant els permisos especificats en la **Taula 2**. (**R21**) Un test interessant que es pot realitzar mitjançant **pgTAP** és comprovar l'existència d'una política RLS.

```
begin;
select plan( 1 );
select policies_are('public','product', ARRAY [
    'Enable all operations for admin', # Admin policy exists.
    'Enable all operations for users based on user_id', # User policy
exists.
    'Enable read access for DEMO' # Demo policy exists
  ]
);
select * from finish();
rollback;

ok 1 - Table public.product should have the correct policies
```

Si un usuari que no ha iniciat sessió intenta efectuar una operació d'inserció de forma maliciosa, PostgreSQL retorna un missatge d'error. De totes maneres, des de l'aplicació s'han deshabilitat els botons per crear elements de la demostració.

```
begin;
  select plan( 1 );
  SET ROLE anon;
  PREPARE my_thrower AS INSERT INTO product (name) VALUES ('AAAA');
  -- Try to insert on product and expect an error
  SELECT throws_ok(
    'my_thrower',
    '42501',
    'new row violates row-level security policy for table "product"'
  );
  select * from finish();
rollback;
```

RPCs

Per comprovar el bon funcionament de les funcions s'han realitzat tests unitaris amb **pgTAP** i posteriorment s'ha comprovat la seva integració amb el *frontend*, comprovant els diferents casos d'ús involucrats. Els resultats es poden veure en la **Taula 4**.

Funció	Comentari	Test Superat
total_warehouse_stock		✓
total _stock		✓
total_cost		✓
new_delivery	User_id no es passa per paràmetre, s'aconsegueix amb **auth.uid().** La funció no és correcta si l'executa un administrador.	✓*
new_arrival2	En la versió anterior, en **product_warehouse** s'agrupaven quantitats per data de caducitat. Es va canviar per poder localitzar el grup en concret dins el magatzem.	✓*
new_recipe		✓
is_admin		✓

Taula 4. Tests funcions PostgreSQL

9.2.3 Pàgina de màrqueting

S'ha revisat que la pàgina de màrqueting compleix amb els diferents requisits funcionals establerts en l'apartat **3.1.2**.

Requisit Funcional	Comentari	Test Superat
R11 - Nom, logotip, lema, introducció		✓
R12 - Vídeo demostratiu	S'ha encarregat a una altra persona.	✗
R13 - Formulari de Contacte	Els missatges arriben al compte de Netlify.	✓
R14 - Enllaç a la demo		✓
R15 - Preguntes Freqüents		✓
R16 - Testimonis / Opinions	S'introduirà l'apartat quan es tingui més usuaris.	✗
R17 - Dades de Contacte / Xarxes Socials		✓
R18 – SEO Metadades		✓

Taula 5. Tests requisits funcionals de la pàgina de màrqueting

Els resultats de Lighthouse per dispositius mòbils i ordinador es poden observar en les **Figures 16** i **17**. Quan s'executa, es genera un informe amb el diagnòstic detallant les diferents mètriques i si s'han superat o no, alguns no es proven valorar de forma automàtica i et recomana comprovar-los de forma manual. (**R23**)

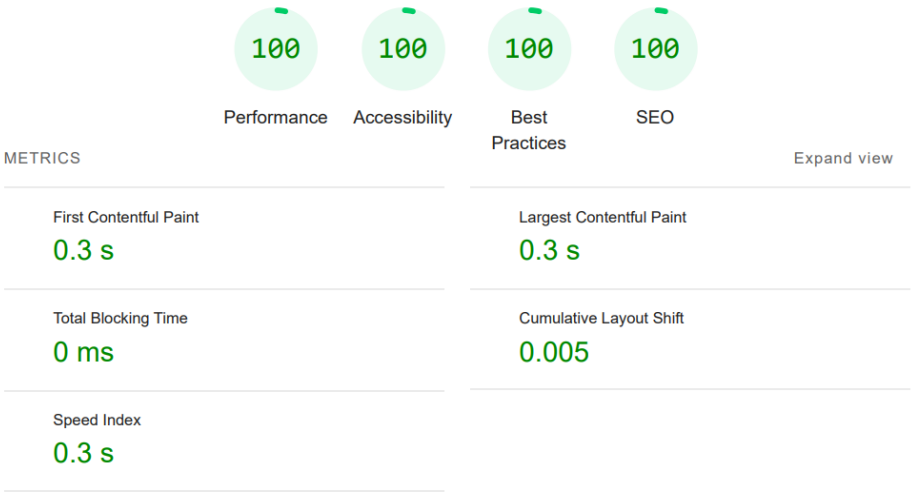

Figura 16. Resultats Lighthouse (Desktop)

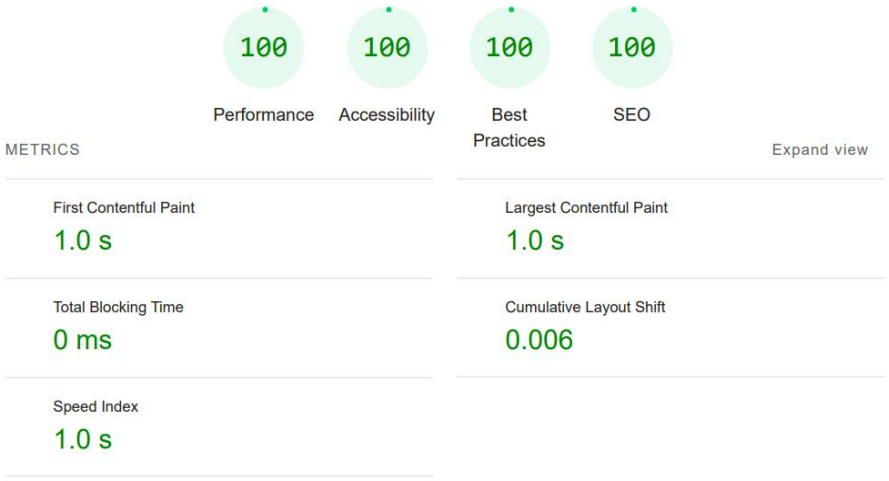

Figura 17. Resultats Ligthouse (Mobile)

També s'ha comprovat el funcionament del banner de cookies i la incorporació dels diferents aspectes legals en la pàgina web. La *cookie* de Google Analytics només ha d'aparèixer quan es acceptada per l'usuari i no abans. (**R22**)

La raó principal per disposar de pàgina web era obtenir reconeixement a l'hora de contactar amb els clients. (**R18**) Per això, és imprescindible que la pàgina web aparegui al cercar el nom del domini a Google. En la **Figura 18** es pot comprovar el resultat d'aquesta cerca.

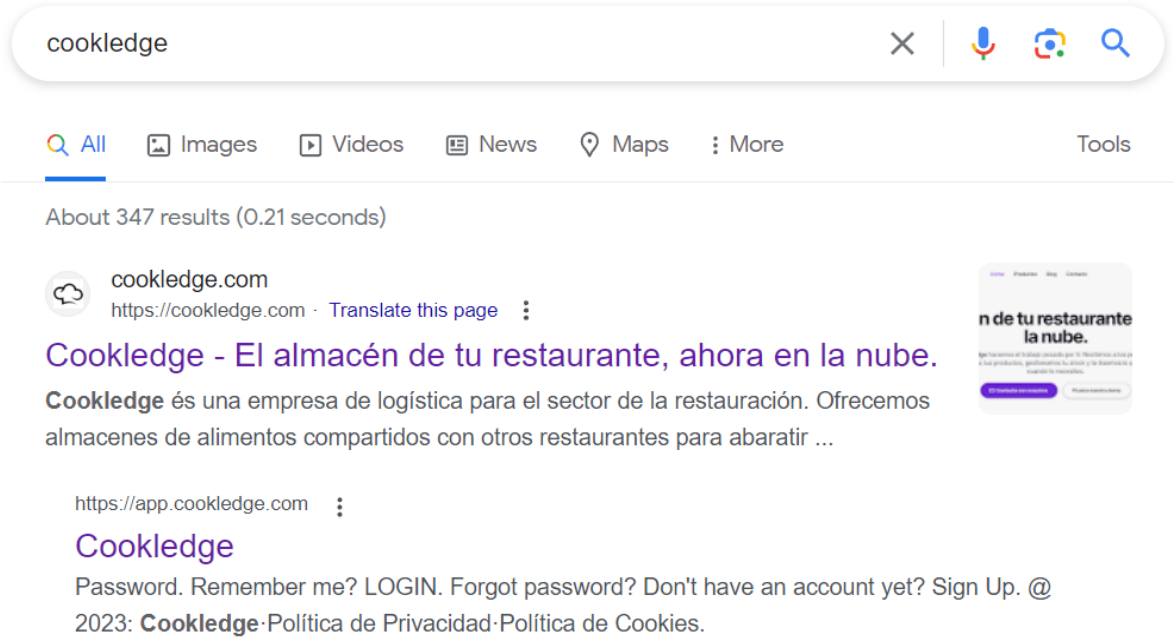

Figura 18. Resultat de cerca a Google

10 Conclusions

En aquest treball s'ha dissenyat i implementat una aplicació web de tipus SaaS que facilita i millora el control de l'inventari als magatzems d'aliments del sector de la restauració. Amb aquesta aplicació, els usuaris poden gestionar de forma senzilla i eficaç els seus productes, productes semielaborats, proveïdors i magatzems, així com els seus moviments, preus, incidències i anàlisis de dades. A més, s'ha desenvolupat una pàgina web de màrqueting per promocionar l'aplicació i captar clients potencials.

L'objectiu principal d'aquest projecte era desenvolupar una solució tecnològica que resolgués les necessitats i els problemes dels restaurants en relació amb el control de *stock* als seus magatzems d'aliments. Aquest objectiu s'ha assolit satisfactòriament, demostrant la validesa i la funcionalitat de l'aplicació web. Així mateix, s'han assolit els objectius específics relacionats amb el disseny i la implementació de l'aplicació web i la pàgina web de màrqueting, utilitzant eines i tecnologies actuals per al desenvolupament web, com Vue, Quasar, Astro i PostgreSQL.

Durant el desenvolupament d'aquest projecte s'han trobat algunes limitacions o dificultats que s'han pogut solucionar o superar amb èxit. Algunes d'aquestes són: la falta d'experiència prèvia amb algunes eines o tecnologies, la indecisió de quin camí seguir, quines eines utilitzar quan hi ha diverses opcions i totes són correctes. Hi ha certes tasques que no saps quina és la millor fins que no l'has implementat com per exemple:

- On faig els càlculs i l'anàlisi, *frontend* o *backend*? Què és més eficient? Què és més fàcil? Quin tarda menys?
- Quina base de dades utilitzar, què és més assequible? Vaig mirar AWS, però tenir una aproximació del preu final és complicat, ja que hi ha moltes opcions i et cobren per tot.
- Quina forma és més segura i eficient per limitar l'accés a certes columnes, RLS, *trigger* o vista? Fins al moment que vaig implementar la vista i em va canviar el nom d'una columna no havia pensat en el manteniment de la vista.

Com a possibles millores futures que es podrien fer a partir d'aquest projecte, es poden plantejar: incorporar funcionalitats addicionals a l'aplicació web, integrar la plataforma amb algun TPV d'un restaurant o fer servir intel·ligència artificial per realitzar la lectura de factures en lloc de fer-ho manualment.

Aquest projecte em va proporcionar l'oportunitat entrar en la 3a edició de la incubadora TIC de REDESSA on s'han realitzat diverses sessions sobre el funcionament de les *startup* en sector empresarial digital. Això ha suposat una gran oportunitat per aprendre sobre aspectes com el model de negoci, el pla de màrqueting o com gestionar una campanya de vendes. També m'ha permès conèixer altres emprenedors amb els quals he pogut compartir experiències i idees.

11 Recursos utilitzats

[1] ApexCharts. (2020, January 16). *Vue Chart Examples & Samples Demo – ApexCharts.js*. ApexCharts.js. https://apexcharts.com/vue-chart-demos/

[2] *Architecture | Supabase Docs*. (2023, June 8). https://supabase.com/docs/guides/getting-started/architecture

[3] *Auth | Supabase Docs*. (2023, June 8). https://supabase.com/docs/guides/auth

[4] *Data Modeling – Table Inheritance*. (2021, November 23). https://ruheni.dev/writing/sql-table-inheritance/

[5] *Documentation | Quasar Framework*. (n.d.). Quasar Framework. https://quasar.dev/docs

[6] Firebase Pricing. (n.d.). *Firebase*. https://firebase.google.com/pricing

[7] *Getting Started*. (n.d.). Astro Documentation. https://docs.astro.build/en/getting-started/

[8] *Introduction | Pinia*. (n.d.). https://pinia.vuejs.org/introduction.html

[9] *Introduction | Vue.js*. (n.d.). https://vuejs.org/guide/introduction.html

[10] *Introduction - Partytown*. (n.d.). Partytown. https://partytown.builder.io/

[11] Koch, R. (2019). Cookies, the GDPR, and the ePrivacy Directive. *GDPR.eu*. https://gdpr.eu/cookies/

[12] *Netlify Pricing and Plans*. (n.d.). Netlify. https://www.netlify.com/pricing/

[13] Onwidget. (n.d.). *astrolib/packages/seo at main · onwidget/astrolib*. GitHub. https://github.com/onwidget/astrolib/tree/main/packages/seo

[14] *pgTAP: Unit Testing | Supabase Docs*. (2023, June 8). https://supabase.com/docs/guides/database/extensions/pgtap#overview

[15] *PostgreSQL 15.3 Documentation*. (2023, May 11). PostgreSQL Documentation. https://www.postgresql.org/docs/current/

[16] *Reactivity in Depth | Vue.js*. (n.d.). https://vuejs.org/guide/extras/reactivity-in-depth.html#how-reactivity-works-in-vue

[17] *Row Level Security | Supabase Docs*. (2023, June 8). https://supabase.com/docs/guides/auth/row-level-security

[18] *Supabase Schema Visualizer*. (n.d.). https://supabase-schema.vercel.app/

ANNEX

Software Generat

Els repositoris en Github són privats. Per compartir els arxius sense haver de fer-los públics s'ha utilitzat gitfront.io, que utilitza "*Deploy Keys*" per crear una connexió SSH amb el repositori:

- Pàgina Web: https://gitfront.io/r/FerranBalleste/kdKA4sQBK1fF/cookledge-astro/
- Aplicació: https://gitfront.io/r/FerranBalleste/hZGB6C18RZxx/cookledge-quasar/
- Funcions RPC: https://gitfront.io/r/FerranBalleste/Ysrc8dWR29e1/cookledge-sql/

Si es necessita accés al codi i els links anteriors no funcionen contactar amb:

ferran.balleste@estudiants.urv.cat

Posada en marxa

1. Obrir consola en la carpeta de projecte (**cookledge-astro** o **cookledge-quasar)**
2. Executar "**npm install**" (S'ha d'haver instal·lat npm amb anterioritat)
3. Per engegar el servidor local, executar "**npm run dev**"
4. Obrir el navegador i anar a l'adreça indicada per consola (localhost:9000, ...)

Alternativament es poden visitar les pàgines següents:

- **Pàgina web:** https://cookledge.com/
- **Aplicació:** https://app.cookledge.com/