



This is the **published version** of the bachelor thesis:

Colao Cintas, Pablo. *Cercador intern per a una botiga online*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334411>

under the terms of the  license.

Cercador Intern per a una botiga online

Pablo Colao Cintas

juny 2026

Resum— Aquest treball presenta el disseny, desenvolupament i implementació d'un Cercador Intern per a botigues online, amb l'objectiu de ser comercialitzat com a SaaS (Software as a Service). L'objectiu principal és oferir una eina multillogater potent, robusta i fàcilment integrable mitjançant una API que millori l'experiència de l'usuari final i incrementi les conversions en el comerç electrònic. El sistema integra funcionalitats clau com l'autocompletat predictiu, la cerca tolerant a errors tipogràfics, el filtrat avançat per atributs i la gestió de subscripcions automatitzada. A més, compta amb un mòdul d'administració que permet personalitzar productes i analitzar el comportament de cerca a través d'un tauler de control. El sistema s'ha desenvolupat utilitzant tecnologies eficients com Django per a la creació d'APIs, Meilisearch com a motor de cerca d'alt rendiment per la indexació de dades, PostgreSQL per a la gestió de la base de dades relacional, i Vue per a una interfície dinàmica.

Paraules clau— Cercador Intern, E-commerce, SaaS, Multi-llogater, API, Motor de Cerca, Experiència d'Usuari.

Abstract— This work presents the design, development, and implementation of an internal search engine for online stores, aimed at being commercialized as a SaaS (Software as a Service). The main objective is to offer a powerful, robust, and easily integrable multi-tenant tool through an API that enhances the end-user experience and increases conversions in e-commerce. The system integrates key features such as predictive autocomplete, typo-tolerant search, advanced filtering by attributes, and automated subscription management. Furthermore, it includes an administration module that allows product customization and search behavior analysis through a dashboard. The system has been developed using efficient technologies such as Django for API creation, Meilisearch as a high-performance search engine for data indexing, PostgreSQL for relational database management, and Vue for a dynamic interface.

Index Terms— Internal search engine, E-commerce, SaaS, Multi-tenant, API, Search engine, User experience.



1 INTRODUCCIÓ - CONTEXT DEL TREBALL

1.1 Motivació i problemàtica

El comerç online s'ha convertit en una de les tendències que marquen tant el present com el futur d'empreses grans, petites o mitjanes [1]. Aquest fenomen, exigeix mecanismes de venda ràpids i eficaços. En aquest entorn digital, l'experiència d'usuari (UX) esdevé el factor més determinant per a l'èxit del negoci: una interacció satisfactòria no només facilita l'adquisició immediata dels productes, sinó que optimitza la taxa de conversió i minimitza la taxa de rebot. Dins d'aquest context, aquest treball es centra en un mecanisme concret d'interacció amb l'usuari: el cercador intern. Aquest mecanisme és essencial per a un comerç, ja que permet als clients trobar els productes o informació pels quals estan interessats. El seu potencial és evident, actualment prop del 30% dels usuaris utilitzen el cercador per navegar i gràcies a això, es pot arribar a multiplicar per 1,8 la taxa de conversió de la botiga. Malgrat aquesta rellevància, s'observa una problemàtica generalitzada en el sector: més del 60% dels comerços electrònics ofereixen cercadors tradicionals o per defecte molt limitats. Aquests sistemes convencionals són

incapaços de gestionar sinònims o detectar errors tipogràfics, generant d'aquesta manera una reacció negativa en cadena on l'usuari, en no trobar el que busca, abandona la web. Per tant, utilitzar-los no sempre aporta una millor experiència d'usuari.

Per donar resposta a aquestes mancances, la investigació analitza la integració de cercadors avançats com a solució tecnològica viable. Aquests sistemes intel·ligents milloren significativament la fluïdesa i la precisió de les cerques mitjançant algorismes i funcionalitats clau com l'autocompletat, el filtrat personalitzat, la comprensió d'errors i els mòduls de recomanació.

La implementació d'aquestes eines d'última generació aporta un doble benefici estratègic:

- Per al client: Garanteix una experiència òptima en oferir els productes exactes amb claredat i rapidesa, fet que augmenta el temps de permanència a la web (millorant d'aquesta manera el SEO) i reforça la fidelització cap a la marca.

- Per als gestors: Proveeix eines avançades d'optimització comercial, tals com l'obtenció de resultats analítics en temps real i la programació de resultats estratègics segons les necessitats del negoci i els clients.

1.2 Objectius

L'objectiu principal del projecte és el disseny, desenvolupament i implementació d'una solució de cerca interna per a botigues online, que s'haurà de comercialitzar com a SaaS. Es vol definir un sistema potent i que sigui fàcilment integrable a qualsevol comerç electrònic per millorar l'experiència d'usuari i incrementar les taxes de conversió.

Els objectius d'aquest projecte, es poden definir en categories, segons importància o profundització amb caràcter incremental. Es compta amb objectius principals, objectius secundaris i objectius tècnics.

Objectius principals: Oferir als comerços online un servei de cerca interna. Implementar el sistema com un SaaS multi-tenant: Ha de ser d'ús independent, aïllat i escalable per a cada client. A més, ha de garantir un cert nivell d'integritat mitjançant una API robusta i fàcilment integrable.

Objectius secundaris: Millorar l'experiència d'usuari (UX) oferint resultats precisos, tolerància a errors, reconeixement de sinònims, sistema d'autocompletat i suggeriments. Augmentar la taxa de conversió, igualment que reduir la taxa de rebot mitjançant sistemes d'ordenació per rellevància o preu. Implementar funcionalitats avançades de cerca com filtrat dinàmic o rànquings. Desenvolupar un sistema de subscripcions per controlar l'abast de productes i les peticions realitzades. A més, dissenyar un dashboard analític que permet als administradors de les botigues fer un seguiment de les cerques populars, control de mètriques i detecció de cerques sense resultats.

Objectius tècnics: Definir i implementar l'arquitectura més òptima per al suport de la plataforma. Avaluar i seleccionar les tecnologies del projecte, comparant diferents motors de cerca i justificant l'elecció del backend, la base de dades i els frameworks utilitzats.

2 ESTAT DE L'ART

Actualment, el comerç online s'ha convertit en un entorn digital altament competitiu que exigeix mecanismes de venda ràpids i eficaços, on l'experiència d'usuari (UX) esdevé el factor més determinant per a l'èxit d'un negoci. Les dades mostren la rellevància del cercador intern en aquesta interacció: prop del 30% dels usuaris el fan servir per navegar i això permet multiplicar per 1,8 la taxa de conversió de la botiga [2], [3].

Malgrat aquesta importància, s'observa una problemàtica generalitzada al sector: més del 60% dels comerços online

ofereixen cercadors tradicionals o molt limitats. Aquests sistemes basats en cerques simples de bases de dades són incapaces de gestionar funcionalitats avançades, generant una reacció negativa per l'usuari. Per portar una solució a aquest inconvenient, el mercat tecnològic compta amb una sèrie de propostes orientades a la indexació i recuperació de dades, tot i que presenten limitacions segons el tipus de negoci:

Algolia: És el líder en el mercat de cerques com a SaaS gestionat al núvol. Ofereix gran velocitat i eines avançades tant visuals com de personalització basades en IA per a grans corporacions. Igualment, el seu codi és tancat, per la qual cosa impedeix l'autoal·lotjament i el seu model de preus basat en el volum de peticions s'escala de manera molt agressiva. Això la converteix en una opció poc accessible per a negocis en creixement.

Elasticsearch: És una opció molt popular, robusta i dissenyada per a la recopilació, anàlisi de logs i consultes de grans volums de dades en entorns distribuïts. Tot i així, és una eina que requereix una extensa i complexa configuració per obtenir la mateixa rellevància que altres cercadors ofereixen de sèrie. La seva corba d'aprenentatge i la necessitat d'una arquitectura exigent penalitzen el temps de desenvolupament en projectes modulars.

Typesense: És l'alternativa de codi obert que destaca per la seva velocitat i configuració senzilla. Malgrat això, té una gran dependència en els recursos físics, la qual cosa encareix els costos d'infraestructura.

Solucions natives de BDs (Node.js): Aquestes requereixen molta personalització des de zero per oferir seguretat en entorns multi-tenants. A més, els cercadors basats en models no relacionals tenen limitacions a l'hora de gestionar entitats interconnectades.

Aquesta situació de mercat és la que motiva el desenvolupament d'aquest projecte. Amb aquest Cercador, busquem oferir als comerços una solució en format SaaS multi-tenant potent, robusta i fàcilment integrable mitjançant API RESTful. En lloc d'haver de desenvolupar codi nou o enfrontar-se a configuracions exigents, la nostra plataforma es recolza en Meilisearch com a motor de cerca d'alt rendiment [21]. Això permet oferir de manera nativa funcionalitats com l'autocompletat, la tolerància a errors, rànquings i filtratge per facetes, garantint respostes ultraràpides en menys de 50 mil·lisegons.

3 METODOLOGIA

Després de valorar múltiples opcions, la que millor encaixa en el desenvolupament d'aquest projecte és la metodologia Agile [4]. Aquest enfocament de treball de tipus incremental i iteratiu, orientat al disseny d'un producte mínim viable (MVP), s'ajusta perfectament a les necessitats i a la tipologia del treball. Comptem amb un desenvolupament dividit en fases successives (Anàlisi i planificació, Desenvolupament, Validació i proves i Avaluació final per tal d'ordenar el

projecte segons el seu estat actual [5]. Això ens permet tenir un control total sobre el punt en el qual es troba el projecte en cada part de l'aplicació i validar progressivament els resultats obtinguts.

Aquest enfocament no es limita a una planificació tancada, sinó al control d'un flux modular per garantir un ritme de treball uniforme. Permet ajustar la planificació, introduir canvis o millores en qualsevol punt del desenvolupament si cal. D'aquesta manera, és molt més flexible que altres metodologies tradicionals al no comptar amb cicles estructurats prèviament, cosa que permet establir una base sòlida d'anàlisi de requisits i cercadors abans de començar amb el disseny detallat de l'arquitectura.

En tenir una arquitectura per mòduls en el desenvolupament (on es defineix l'API, els motors de cerca, les bases de dades, el dashboard, les subscripcions), permet que cada component sigui provat i validat de forma independent abans de ser aplicat finalment al sistema general.

Així mateix, incloure tècniques de verificació com les proves funcionals, les simulacions i el control d'aïllament entre tenants, permet poder dur a terme les proves pertinents per a veure si tot està alineat amb els requisits definits, avaluant al final del procés el rendiment i la capacitat d'escalar en el futur sense dependre d'un calendari rígid o de canvis que puguin afectar la planificació.

4 JUSTIFICACIÓ EINES ESCOLLIDES

Per la realització del projecte, es presenten una sèrie d'eines per al desenvolupament de backend, frontend, bases de dades i motor de cerca. En aquest apartat es presenta una valoració i justificació de les eines seleccionades pel projecte prioritzant l'eficiència, la seguretat, la facilitat d'integració i l'adequació al model SaaS multi-tenant. Es divideix l'apartat entre eines backend, frontend, motors de cerca i bases de dades.

Eines Backend: Es selecciona Django (Python) com a framework de desenvolupament web en lloc d'un entorn d'execució com Node.js (JavaScript) [6]. Tot i que Node.js ofereix un gran rendiment gràcies al seu motor V8 i una gestió d'operacions d'E/S no bloquejants ideal per a alta concurrència, requereix una configuració i personalització manual excessiva per dotar el sistema d'una infraestructura multi-tenant segura. Django redueix la complexitat del codi gràcies a la sintaxi d'alt nivell de Python i aporta de sèrie una infraestructura robusta amb bateries incloses indispensable per al projecte, com ara el mòdul d'autenticació (Django Auth), la gestió de models de dades (ORM) i defenses integrades contra vulnerabilitats XSS i CSRF. Aquesta arquitectura estalvia temps i codi, adaptant-se nativament al disseny modular d'una API SaaS amb control d'accés rígid [32].

Eines Frontend: Per a la interfície d'administració i el dashboard analític, s'ha escollit Vue.js enfront de React [7]. React és una alternativa potent basada en Virtual DOM i components reutilitzables que facilita aplicacions altament escalables, però presenta una corba d'aprenentatge més exigent i requereix configuracions complexes de llibreries d'ecosistema que alenteixen el desenvolupament d'un MVP. Vue.js és un framework progressiu més senzill de programar que garanteix la sincronització bidireccional immediata entre el model de dades i la interfície visual. Incorpora també l'ús de plantilles HTML base molt accessibles. Això permet oferir reactivitat en temps real sobre estadístiques de cerca i analítiques amb un flux d'integració molt àgil mitjançant l'API REST.

Bases de dades: La persistència del sistema s'ha delegat en una base de dades relacional com PostgreSQL, descartant solucions NoSQL basades en documents JSON com MongoDB [8]. Tot i que MongoDB simplifica l'emmagatzematge flexible i la replicació ràpida de documents, resulta molt limitat a l'hora de gestionar models de negoci modulars amb múltiples entitats altament interconnectades. El projecte requereix dades estructurades que representin botigues, usuaris, productes, subscripcions i logs en un entorn multi-tenant consistent. PostgreSQL garanteix un suport ACID complet, operacions estables pel canvi de pla de subscripció i seguretat referencial en les consultes complexes. A més, l'ORM de Django es troba completament optimitzat de forma nativa per a PostgreSQL, fet que simplifica l'aïllament dels tenants i minimitza el risc de filtracions de dades.

Motors de Cerca: El nucli tecnològic de la cerca s'ha resolt mitjançant Meilisearch en lloc d'Elasticsearch, adaptant-se estrictament als requisits d'experiència d'usuari (UX) [9].

D'una banda, Elasticsearch destaca en entorns distribuïts per a la recopilació i anàlisi massiva de logs, però penalitza els temps de desenvolupament modular a causa de la seva corba d'aprenentatge i la necessitat d'una extensa i exigent configuració manual per aconseguir resultats optimitzats. D'altra banda, Meilisearch és un motor dissenyat de manera nativa per a aplicacions web de comerç electrònic. Proporciona directament i sense configuració prèvia algoritmes de tolerància a errors tipogràfics, autocompletat predictiu, rànquings de rellevància ajustables i filtratge facetat. Meilisearch assegura respostes ultraràpides en menys de 50 mil·lisegons, reduint costos d'infraestructura i oferint suport natiu per a entorns multi-tenant de forma eficient i assequible [30], [31].

5 ARQUITECTURA DEL SISTEMA

Amb el funcionament ja definit, el següent pas consisteix a estructurar l'arquitectura lògica que seguirà el sistema. S'ha optat per una arquitectura client-servidor, la qual permet separar responsabilitats, facilitant el manteniment independent, l'escalabilitat i el desplegament de cada capa.

D'una banda, el backend s'encarrega de gestionar tota la lògica de negoci, la seguretat, l'accés a les dades i l'exposició dels diferents serveis. D'altra banda, el frontend gestiona la interfície d'usuari i l'experiència d'usuari, consumint els servidors mitjançant peticions HTTP i intercanviant dades en format JSON.

5.1 Arquitectura Interna del Backend

Tot i que Django utilitza un patró de disseny anomenat MVT (Model - View- Template), en aquest projecte, al tractar-se d'una arquitectura desacoblada on el frontend renderitza les pantalles, la capa de Templates s'elimina del backend.

En el seu lloc, mitjançant l'ús de Django REST Framework, el sistema s'estructura sota el patró Model - Serializer - View (MSV), que funciona de la següent manera:

- **Models:** És la representació en codi Python de les taules de la base de dades. Defineix l'estructura de les entitats (usuari, botigues, productes, subscripcions) i les seves relacions, utilitzant l'ORM per a interactuar amb el motor de cerca sense escriure SQL directament [10].
- **Serializers:** Actuen com a traductors o ponts de dades. S'encarreguen de convertir les instàncies dels models en format JSON per a poder ser enviades al frontend, validant que les dades rebudes siguin correctes abans de guardar-les [15].
- **Views:** Contenen la lògica de negoci de l'API. Reben les peticions HTTP del frontend, executen les accions pertinents, interactuen amb els serializers i el motor de cerca, i finalment retornen la resposta HTTP corresponent.

Aquest disseny modular garanteix que el cercador estigui completament aïllat de la interfície visual. Si en un futur es decidís canviar el frontend per una app mòbil o un altre framework, el backend i el motor de cerca continuarien funcionant exactament igual sense haver de modificar la lògica.

5.2 Modelatge de Dades

Per al desenvolupament del sistema sobre la base de dades relacional s'ha definit un model de dades modular que ha evolucionat des d'un nucli bàsic fins a una estructura preparada per un entorn SaaS multi-tenant. Inicialment basat en tres entitats clau, el model s'ha ampliat per permetre gestionar subscripcions i el control analític de les cerques.

El sistema final es compon de cinc models de dades simplificats i optimitzats:

- **User:** Gestiona les dades de credencials i accés dels clients del SaaS.
 - Camps: id, username, email, first_name, last_name, password
- **Shop:** Representa la unitat de negoci (tenant) dins del sistema.
 - Camps: id, name, address, subscription_plan, owner
- **Product:** Conté els articles que indexarà el motor de cerca per a cada botiga.
 - Camps: id, name, price, description, brand, shop
- **SubscriptionPlan:** Defineix els límits comercials de l'aplicació.
 - Camps: name, max_searches, max_products_per_search, price
- **SearchLog:** Emmagatzema les mètriques i l'historial del cercador per a l'anàlisi del dashboard.
 - Camps: shop, query, timestamp, results_count

Hi ha una sèrie de relacions entre entitats que formalitzen la lògica del sistema multi-tenant.

Aquestes relacions són:

- **User - Shop:** Un usuari pot gestionar múltiples botigues en un entorn multi-tenant, però cada botiga està vinculada exclusivament a un sol usuari administrador (owner).
- **Shop - Product:** Una botiga pot allotjar un catàleg d'un o més productes, mentre cada producte pertany de forma unívoca a una única botiga.
- **SubscriptionPlan - Shop:** Per limitar l'ús de l'API segons el model de negoci, un pla de subscripció pot estar assignat a diverses botigues simultàniament, però cada botiga està només pot tenir un únic pla actiu.
- **Shop - SearchLog:** Per alimentar el panell d'anàliques, una botiga genera múltiples registres de cerca, de manera que cada log fa referència a una única botiga d'origen.

Cardinalitats:

User 1 - * Shop
 Shop 1 - * Product
 SubscriptionPlan 1 - * Shop
 Shop 1 - * SearchLog

6 ESTRUCTURA DEL BACKEND

Per al desenvolupament del backend del projecte s'ha escollit

el framework Django amb Django Rest Framework per la construcció d'una API RESTful eficient. Un dels pilars de Django és la seva filosofia de disseny basat en aplicacions, la qual cosa permet estructurar el sistema en mòduls independents. Aquest tipus d'arquitectura és essencial en un SaaS multi-tenant, ja que facilita l'escalabilitat del codi, simplifica el manteniment i assegura una separació neta de les responsabilitats de la lògica de negoci.

A continuació, es defineix l'organització del sistema amb una visió funcional dels mòduls i una anàlisi tècnica dels fitxers.

6.1 Aplicacions

Per complir completament amb els requisits i objectius predefinits anteriorment, s'ha dividit el sistema en sis aplicacions independents, on cadascuna respon a un problema específic i resol una funcionalitat requerida pel SaaS. Aquestes aplicacions són:

Users

Aquesta aplicació s'encarrega de manera centralitzada de la gestió dels usuaris de la plataforma i del cicle de vida de l'autenticació. Des d'un punt de vista funcional, aquest mòdul permet controlar de forma segura el registre de nous comerciants, l'inici i tancament de sessió, i la consulta o modificació del perfil i la informació rellevant associada a cada compte.

Shops

Aquesta aplicació és el component que fa possible el model de negoci multi-tenant de la plataforma. Permet als comerciants registrar, configurar i controlar diferents botigues de manera centralitzada. Des de la vista de l'usuari, l'aplicació ofereix un entorn de gestió on es pot visualitzar el llistat complet de botigues existents sota el teu compte, donant accés a altres àrees com la creació de productes o la visualització de les analítiques. El flux de creació permet introduir les dades bàsiques d'una nova botiga i li assigna per defecte el pla de subscripció inicial, assegurant que les accions d'un comerciant s'executin exclusivament dins del seu entorn propi.

A la imatge 3 de l'Anex B es pot veure representat l'entorn de les botigues.

Products

Aquesta aplicació representa la gestió de l'inventari i el

catàleg de cada botiga, essent el nivell més específic de la jerarquia del sistema. Funcionalment, els usuaris disposen d'un panell de control on poder mantenir el seu catàleg. Permet donar d'alta productes de manera individual, així com llistar-los. A més, en cas de requerir càrrega massiva d'actius, l'aplicació incorpora una funcionalitat avançada d'importació massiva de productes, accepta col·leccions estructurades de dades per introduir centenars de productes en una sola operació.

Search

Aquesta aplicació té l'objectiu de gestionar les cerques sobre el catàleg de productes internament dins de la plataforma d'administració. Proporciona les eines de cerca necessàries perquè el comerciant pugui localitzar els productes dins del seu panell de forma àgil i amb un bon rendiment. El sistema garanteix reactivitat en temps real a la interfície d'usuari, a mesura que l'usuari escriu a la barra de cerca, el catàleg es filtra de manera dinàmica. A més, garanteix que les cerques estiguin filtrades per botiga, evitant un encreuament d'informació confidencial. A la imatge 4 de l'Annex B es pot veure representat el cercador d'administració.

SaaS

Aquesta aplicació constitueix el producte final i el motor SaaS en oferir el cercador com a un servei directament consumible per tercers. El seu objectiu és subministrar un cercador d'alt rendiment completament desenvolupat i llest per ser integrat en qualsevol lloc web d'un comerç electrònic extern dels clients, sense que aquests hagin de desenvolupar codi nou de cerca. Exposa característiques funcionals avançades que milloren l'experiència dels usuaris finals de les botigues:

- **Tolerància a errors:** Corregeix automàticament errors d'escriptura de l'usuari per oferir resultats encertats.
- **Autocompletat i suggeriments:** Anticipa la intenció de cerca de l'usuari a mesura que tecleja.
- **Filtratge combinat i dinàmic:** Permet a l'usuari final de l'e-commerce acotar els resultats instantàniament per rang de preu o per marca específica (capturades automàticament pel sistema).
- **Ordenació intel·ligent:** Habilita la reordenació del llistat de productes tant per criteris de preu ascendent com descendent de manera immediata [24].

A més de garantir l'experiència de cerca externa, s'encarrega de controlar la lògica comercial dels plans de subscripció

(Basic, Premium), mitjançant credencials de seguretat úniques, verificant que la botiga no superi la quota màxima de productes permesos ni el volum de cerques contractat.

A les imatges 2 i 5 de l'Anex B es poden veure representades una cerca amb tolerància a errors i una cerca per filtratge i ordenança.

Dashboard

Aquesta aplicació té l'objectiu de capturar les interaccions dels consumidors finals amb el cercador de l'aplicació SaaS i transformar aquests registres en mètriques per ser analitzades. Aquest model proporciona als administradors de les botigues un entorn analític estructurat amb dues àrees de coneixement:

- **Cerques populars:** Mostra quins són els termes o productes més sol·licitats pels consumidors, permetent al comerciant identificar tendències del mercat i demanar més estoc d'un article concret.
- **Cerques sense resultats:** Identifica quines paraules o conceptes s'estan introduint a la barra de cerca i no retornen cap producte. Aquesta mètrica ajuda als comerciants a detectar defectes com errors comuns a l'hora de teclejar i així poder oferir sinònims, o bé, descobrir demandes de productes que haurien d'introduir per incrementar la taxa de conversió.

A la imatge 1 de l'Anex B es veu representat el Dashboard.

6.2 Implementació Tècnica

Per construir una API REST orientada al servei, s'ha adoptat el framework Django REST Framework (DRF), adaptant el patró natiu MTV de Django cap a una estructura desacoblada Model-Serializer-View (MSV) que delega el renderitzat de la interfície a Vue.js mitjançant peticions asíncrones amb la llibreria Axios [26], [28]. Cadascun dels sis mòduls independents de l'arquitectura manté una estructura homogènia de fitxers dins del backend: la carpeta migrations/ per a l'historial de canvis a la base de dades PostgreSQL, apps.py per a la configuració i registre de components, models.py com a capa de persistència, serializers.py per a la validació i transformació de dades JSON, signals.py per a les reaccions a esdeveniments, urls.py per als endpoints de l'API i views.py per als controladors basats en APIView. A més, per realitzar les cerques de manera externa simulant l'entorn multi-tenant, s'ha creat el fitxer tech_components.html. Aquest fitxer actua com a una simulació de la web d'un comerç electrònic extern que consumeix directament els endpoints de

l'API, demostrant la capacitat del backend per servir dades a múltiples clients independents.

A continuació es detalla la lògica de disseny i les particularitats tècniques implementades a cada aplicació:

Users: Controla el cicle de vida de l'autenticació centralitzada (registre, inici i tancament de sessió, i consulta de perfil) definint un model d'usuari que hereta de la classe AbstractUser de Django, aprofitant la seva infraestructura nativa i emmagatzemant els camps bàsics d'ID, username, email, first_name, last_name i password. La lògica de negoci es divideix en quatre endpoints basats en APIView [16] i es recolza en dos serialitzadors: UserSerializer (que encapsula la informació de l'usuari autenticat i exposa les botigues associades via clau forana) i UserRegisterSerializer (que valida les dades d'entrada durant el procés de creació). L'arquitectura de seguretat utilitza sessions del servidor [12], cookies identificatives, permisos IsAuthenticated per rebutjar peticions anònimes i una protecció obligatòria contra atacs de falsificació de peticions CSRF, comunicant-se amb els components de Vue.js RegisterUser, LoginUser i UserProfile [20], [27], [29].

Shops: Regeix el model de negoci multi-tenant de la plataforma permetent registrar, configurar i controlar diferents comerços de manera aïllada mitjançant el model Shop, que conté els atributs d'ID, name, address, subscription_plan i owner. Estableix una restricció de cardinalitat d'un a molts mitjançant una clau forana cap al model User (owner) [11], garantint que cada botiga pertanyi unívocament a un sol comerciant. L'aplicació exposa dos punts d'accés gestionats per APIView: el llistat de botigues, on s'aplica un filtre sobre el queryset segons l'usuari en sessió per impedir que es vegin dades de tercers, i la creació de botigues, on el camp owner s'introdueix automàticament des del backend (owner = request.user) per evitar manipulacions malicioses. Per optimitzar el rendiment de la base de dades, el component ShopSerializer valida la informació i transforma les instàncies fent ús de camps de tipus StringRelatedField per mostrar textualment el propietari i els productes, vinculant-se al frontend amb els components ShopList i createShop.

Products: Representa la gestió de l'inventari i el catàleg aïllat de cada tenant mitjançant el model Product, que encapsula els atributs d'ID, name, price, description, brand i shop. S'utilitza una clau forana cap a Shop que imposa una relació un a molts per marcar l'aïllament del tenant, de manera que els endpoints d'aquesta aplicació (llistat, creació individual i creació massiva) requereixen la injecció del paràmetre shop_id a les peticions perquè el backend comprovi que la botiga pertany realment al comerciant autenticat, evitant exposar dades confidencials. El component ProductSerializer s'encarrega de la transformació i validació definint el camp shop com a

`StringRelatedField` de només lectura per mostrar una representació simplificada de la botiga sense revelar la seva estructura interna. Per als fluxos de càrrega massiva d'actius, s'incorpora el paràmetre `many=True` al serializer, actuant sobre matrius JSON en una sola transacció i governant la interfície amb els components `productsList`, `createProduct` i `createMore`.

Search: Encarregada de la cerca interna dins del panell d'administració per localitzar articles de forma àgil, aquesta aplicació es caracteritza per no comptar amb taules relacionals a PostgreSQL. La seva funció és acoblar directament el motor de cerca Meilisearch per obtenir alta velocitat de resposta i, per garantir la consistència de les dades, implementa el patró Observer amb senyals nadius de Django ([signals.py](#)) [13], [14]. El senyal `post_save` intercepta les creacions o modificacions a PostgreSQL per replicar-les a Meilisearch, mentre que `post_delete` elimina instantàniament el document indexat del motor. Quan s'executa una cerca, l'endpoint de la vista consulta Meilisearch incloent l'ID de la botiga per preservar el model multi-tenant i evitar encreuaments d'informació; el motor retorna un vector amb els identificadors dels productes rellevants que s'utilitzen per recuperar els objectes directament de la base de dades relacional mitjançant el seu ORM. Al frontend, el component Cerca en Vue fa servir la directiva `v-model` vinculada a la variable `query` per llançar peticions asíncrones que actualitzen el DOM en temps real i eliminen les recàrregues de pàgina.

SaaS: Constitueix el nucli de programari com a servei que subministra el cercador d'alt rendiment directament consumible per tercers i integrable en webs d'e-commerce externs sense necessitat de desenvolupar codi nou. Estén el model Shop amb camps de control clau: una clau forana cap a `SubscriptionPlan`, un camp `apiKey` xifrat per identificar el comerç exterior i un acumulador numèric `searches_count`. La configuració inicial de l'índex es recolza en el script `search/meilisearch.py` per definir de antemà els atributs de filtratge i ordenació [22], [23]. A diferència del nucli intern, els endpoints de l'aplicació SaaS són stateless (sense sessions, cookies ni CSRF), resolent la seguretat mitjançant la interceptió de la capçalera personalitzada `X-API-KEY` validada contra PostgreSQL [17], [18]. Els controladors `APIView` separen la Cerca de productes de l'API (que proveeix funcionalitats de tolerància a errors, autocompletat, filtratge combinat per preu/marca i ordenació intel·ligent) i l'Assignació de marques, utilitzant un middleware addicional que comprova que no es superi la quota màxima de productes permesos ni el volum de cerques contractat en els plans Basic o Premium.

Dashboard: Mòdul analític dissenyat com a registre històric basat en la classe `SearchLog`, el qual té l'objectiu de capturar les interaccions dels consumidors finals amb el cercador de l'aplicació SaaS. Per cada cerca processada amb èxit, es genera automàticament una entrada a PostgreSQL que emmagatzema els camps de shop, la cadena exacta teclejada (`query`), la marca de temps (`timestamp`) i el nombre de resultats obtinguts (`results_count`). Mitjançant estructures `APIView`, el mòdul realitza consultes d'agregació sobre el model per oferir mètriques de negoci a través de dos endpoints: Cerques

populars, que agrega les consultes i fa un recompte de freqüència ordenat de forma descendent per identificar tendències de mercat i preveure l'estoc, i Cerques sense resultats, que filtra les consultes fallides per detectar errors comuns de tecleig, programar sinònims o mitigar la taxa de rebot. L'accés al panell requereix autenticació global per sessions i vincula obligatòriament totes les consultes a l'ID del comerciant actiu mitjançant la restricció `owner = request.user`, garantint un aïllament independent que impedeix filtracions d'informació confidencial entre botigues.

7 RESULTATS

7.1 Aplicació Desenvolupada

El desenvolupament del projecte del Cercador Intern ha donat lloc a una aplicació web funcional en format SaaS orientada principalment a la millora de la cerca i la taxa de conversió per a les botigues online.

Primerament, i com a resultat principal, s'ha desenvolupat una plataforma robusta amb una arquitectura desacoblada que permet als comerciants gestionar de manera centralitzada el seu inventari, analitzar el comportament dels seus usuaris i oferir un motor de cerca extern altament eficient per als seus e-commerce. A més, per al desenvolupament s'han utilitzat tecnologies i frameworks àmpliament fets servir actualment en el sector industrial i amb un ampli recorregut comercial, com Django per a la creació d'APIs RESTful, Meilisearch com a motor de cerca d'alt rendiment per a la indexació de dades, PostgreSQL per a la gestió de la base de dades relacional i Vue per a una interfície dinàmica i reactiva al frontend.

S'ha aconseguit assolir tots els objectius plantejats a l'inici de la planificació del projecte. Això ha estat possible gràcies a la metodologia aplicada i al disseny modular escollit. Per una banda, l'adopció d'una metodologia Agile ha permès un desenvolupament incremental i iteratiu, facilitant la detecció i prevenció d'errors i l'adaptació continua dels requisits. Per altra banda, l'arquitectura desacoblada i el disseny modular han sigut uns pilars fonamentals per garantir la independència entre el backend i el frontend.

Totes les funcionalitats definides en el primer informe han estat implementades i validades amb èxit. Aquestes funcionalitats són les següents:

Gestió d'usuaris i entorn multi-tenant: S'ha implementat un control d'accés segur, on cada comerciant compta amb un entorn complet aïllat on poder administrar el conjunt de les seves botigues. S'ha validat que les dades de configuració, productes i analítiques d'una botiga no són accessibles per a cap altre comerciant. En concret s'han dissenyat les

funcionalitats de registre d'usuaris, log-in un cop registrat, log-out i visualització de perfils.

Administració de catàleg i inventari: S'ha implementat les funcionalitats necessàries per permetre als administradors donar d'alta botigues i dins d'aquestes inserir productes de forma individual. A més, s'ha validat amb èxit la importació massiva de productes mitjançant col·leccions de dades estructurades. Gràcies als senyals nadius implementats en el backend, qualsevol canvi en l'inventari es sincronitza instantàniament amb l'índex del motor de cerca.

Cercador intern d'administració: S'ha implementat un cercador integrat a la plataforma d'administració perquè els comerciants puguin localitzar els productes del seu propi catàleg de forma àgil. Al tractar-se d'una solució stateful (amb estat), el sistema requereix l'autenticació prèvia de l'usuari. Per preservar el model multi-tenant i evitar qualsevol encreuament d'informació confidencial entre botigues, les consultes es filtren rigidament al backend mitjançant l'identificador de l'administrador autenticat (`owner = request.user`).

- **Motor de cerca extern i integració amb l'API:** A diferència del cercador d'administració, aquest component constitueix el producte final del SaaS per a ser consumit per e-commerces de tercers de manera stateless. La seguretat s'ha resolt mitjançant la injecció de la capçalera personalitzada `X-API-KEY`. Qualsevol e-commerce extern pot realitzar consultes al cercador de forma senzilla. Per la validació del funcionament del cercador i la consumició de l'endpoint de cerca de manera externa en l'àmbit de SaaS multi-tenant, s'ha creat el fitxer `tech_components.html` que consumeix els endpoints per permetre a les botigues utilitzar el cercador.
- **Experiència de cerca intel·ligent en temps real (UX):** S'ha integrat i validat amb èxit el component destinat a les cerques de l'usuari final. Aquest incorpora les funcionalitats definides com l'autocompletat predictiu immediat i els suggeriments en temps real. A més, s'afegeixen els sistemes de filtrat avançat per facetes (marca o preu) o ordenació per preu. Així mateix, s'ha comprovat la robustesa del motor gràcies a la tolerància a errors tipogràfics (typo-tolerance), garantint que el sistema retorni els productes encara que es cometin omissions o inversions de lletres.
- **Control de subscripcions:** S'ha dissenyat un sistema de subscripcions per botiga amb dos plans principals. Aquests plans compten amb dues mètriques que controlen l'ús dels endpoints de cerca. En primer lloc,

el pla Basic s'ha inicialitzat amb un total de 100 cerques. A més, només permet la visualització d'un total de 10 productes per cerca. En segon lloc, el pla Premium disposa de 1000 cerques i 50 productes per cerca.

Un cop el sistema detecta que s'arriba al màxim permès, les peticions no són acceptades.

- **Revisió d'anàliques (Dashboard):** S'ha desenvolupat una interfície dinàmica que llegeix el registre històric de cerques. Aquests registres es van guardant de manera automàtica per cada cerca realitzada. Gràcies a aquest dashboard, els administradors de les botigues tenen accés a dues mètriques rellevants per controlar les seves botigues. Aquestes són, les cerques populars (per preveure la demanda d'estoc) i les cerques que no donen resultats (per detectar possibles necessitats o l'addició de nous sinònims).

7.2 Proves de rendiment

Per tal d'avaluar la viabilitat i la capacitat d'escalabilitat del cercador, s'ha dissenyat un banc de proves simulant un entorn de producció d'alta demanda. L'objectiu principal és comparar el rendiment del motor de cerca especialitzat (Meilisearch, utilitzant l'endpoint de cerca real) enfront de les consultes tradicionals de la base de dades relacional (PostgreSQL, fent ús de l'operador `ILIKE`) [25].

Per dur a terme les proves, es va iniciar primerament amb un total de 1000 productes, però aquesta quantitat era insuficient per veure diferències rellevants en el rendiment, ja que per a un volum tan reduït, qualsevol base de dades com PostgreSQL és capaç de mantenir pràcticament el catàleg sencer en la memòria del sistema. És per això que finalment es van afegir un total de 100000 productes ficticis a una sola botiga, forçant cerques a gran escala. Les mètriques mostrades són el resultat de calcular el temps mitjà obtingut al llarg de 50 iteracions consecutives per cada tipologia de cerca.

Cal destacar una diferència en l'àmbit d'estructura en el comportament dels motors: mentre que PostgreSQL intenta retornar totes les coincidències en brut, Meilisearch autolimita els resultats a un màxim de 1000 registres amb la directiva `maxTotalHits`.

Execució de Proves

Es van plantejar cinc escenaris crítics que reflecteixen el comportament d'un usuari final en un e-commerce:

A) Cerca exacta (terme: "iphone")

PostgreSQL (Temps mitjà): 295,41 ms | Productes trobats: 16.620

Meilisearch + ORM (Temps mitjà): 26,35 ms | Productes trobats: 1.000 (límit estructural)

PostgreSQL es veu obligat a recórrer les 100000 files, mentre Meilisearch inclou la cerca i la rehidratació de l'ORM.

B) Cerca amb error tipogràfic ("ihone")

PostgreSQL (Temps mitjà): 203,45 ms | Productes trobats: 0

Meilisearch + ORM (Temps mitjà): 26,86 ms | Productes trobats: 1.000

En aquest cas, PostgreSQL inverteix bastant temps de processament per oferir 0 resultats, ja que no es capaç de tolerar fallades. Meilisearch activa la seva política de tolerància a errors basada en la distància de Levenshtein.

C) Cerca per Prefix / Mode (Suggeriments) ("iph")

PostgreSQL (Temps mitjà): 306,96 ms | Productes trobats: 16.620

Meilisearch + ORM (Temps mitjà): 30,06 ms | Productes trobats: 1.000

Si s'escriu un terme incomplet, el cost de cerca en PostgreSQL augmenta. Meilisearch ofereix una experiència fluida.

D) Cerca combinada amb error ("iphone ultre")

PostgreSQL (Temps mitjà): 199,25 ms | Productes trobats: 0

Meilisearch + ORM (Temps mitjà): 29,96 ms | Productes trobats: 1.000

En cercar múltiples paraules on una d'elles inclou un error PostgreSQL torna a fallar. Meilisearch separa els termes i aplica la tolerància a errors mantenint l'eficiència i oferint una bona experiència als usuaris.

E) Cerca sense coincidències ("producteX")

PostgreSQL (Temps mitjà): 220,97 ms | Productes trobats: 0

Meilisearch + ORM (Temps mitjà): 5,24 ms | Productes trobats: 0

Encara que el producte no existeixi, PostgreSQL ha de recórrer totes les files per verificar-ho. Meilisearch detecta de forma immediata que no existeix. En retornar 0 hits, evitem el procés de rehidratació de l'ORM.

Les proves demostren que davant de catàlegs realistes a gran escala en entorns SaaS multi-tenants, PostgreSQL pateix una degradació de rendiment constant, necessitant sempre entre

200 i 300 ms de temps de resposta.

D'altra banda, l'arquitectura implementada amb Meilisearch i l'ORM de Django es manté competitiva i robusta. Gràcies a les seves funcionalitats avançades i a la seva velocitat, el temps de resposta no excedeix els 30ms, confirmant l'èxit del motor de cerca.

8 CONCLUSIONS

Amb la finalització d'aquest treball de final de grau, es considera que s'ha assolit amb èxit tots els objectius i requisits plantejats a l'inici del projecte. S'ha dissenyat, desenvolupat i validat un motor de cerca intern multi-tenant sota el model de Software as a Service (SaaS), demostrant que és possible implementar una solució personalitzada d'alt rendiment sense haver de recórrer a plataformes complexes ni grans inversions.

A través de la combinació d'un motor de cerca especialitzat com Meilisearch i un sistema relacional com PostgreSQL, s'ha validat la viabilitat d'una arquitectura on Meilisearch ofereix un rendiment excel·lent. Les proves realitzades a gran escala (100000 productes) han aportat el suport de la hipòtesi principal: mentre que les consultes tradicionals realitzades amb la base de dades PostgreSQL pateixen una degradació de rendiment, el sistema desenvolupat amb Meilisearch manté un temps de resposta per sota dels 30ms aportant un rendiment correcte.

Des d'un punt de vista de disseny i experiència d'usuari (UX), el projecte s'ha centrat a prevenir la frustració habitual dels cercadors tradicionals, on els errors tipogràfics i la pèrdua de velocitat perjudiquen la taxa de conversió i augmenten la taxa de rebot. Per tant, l'ús de la tolerància a errors tipogràfics, l'autocompletat, els suggeriments i la resta de funcionalitats avançades esdevenen un punt important. En casos en els quals els cercadors tradicionals retornen 0 resultats, el nostre cercador ofereix una resposta immediata, guiant fluidament la interacció de l'usuari i millorant la taxa de conversió.

Pel que fa a la metodologia, haver adoptat un model Agile ha permès afirmar els avantatges d'aplicar un procediment incremental i modular. Gràcies a la construcció d'un backend base i un modelatge sòlid durant les primeres instàncies del projecte, la resta de treball i funcionalitats han sigut fàcils d'implementar.

En l'àmbit acadèmic i personal, aquest treball m'ha permès explorar i aprendre sobre el desenvolupament web. Gràcies a aquest treball he entès com es construeix una aplicació d'inici a fi, passant per tots els components essencials. Aquesta immersió m'ha fet comprendre a fons el funcionament i la lògica de les comunicacions basades en serveis i la creació d'una API RESTful. He après a dissenyar i estructurar els diferents endpoints de l'aplicació, entenent el cicle complet de com es consumeixen aquests serveis mitjançant peticions HTTP des de la interfície d'usuari i com respon el servidor en temps real. Més enllà, la gestió de les queries i el funcionament dels motors de cerca ha estat essencial per la realització del

sistema.

Així mateix, el projecte m'ha aportat una visió pràctica de com es dissenya un entorn SaaS multi-tenant. He hagut d'aprendre a implementar mecanismes de seguretat en l'intercanvi de dades, utilitzant capçaleres personalitzades amb claus d'autenticació. Aconseguir coordinar tots aquests components i verificar que l'aplicació és capaç de mantenir diferents clients aïlladament m'ha aportat una gran seguretat i unes habilitats tècniques considerables.

9 ÚS DE LA IA GENERATIVA

En el desenvolupament d'aquest treball s'ha fet servir la IA generativa com a eina de suport per tasques auxiliars, relacionades amb l'organització i estructuració del contingut. A més, s'ha utilitzat per realitzar aprenentatge autònom sobre les diferents eines aplicades durant el projecte. S'ha fet servir més per aquelles parts que no eren estrictament requerides en el projecte, com les proves, la visualització, etc. Més concretament, s'ha fet servir per:

- Generació de l'entorn de proves per validar el rendiment del projecte.
- Adaptacions en el disseny de la interfície d'usuari
- Resolució d'errors de configuració durant el projecte

AGRAÏMENTS

M'agradaria agrair al meu tutor, en Marc Ortega, per la seva ajuda durant la realització del projecte i la resolució de dubtes. També vull agrair a la meua família per donar-me suport i ajudar-me en aquest procés.

10 BIBLIOGRAFIA

[1] Solmicro, "¿Qué es comercio online?", Solmicro, [Online]. Available: <https://www.solmicro.com/blog/e-business/que-es-comercio-online> [Accessed: 23 Feb. 2026].

[2] SE Ranking, "Sistema de búsqueda interno en ecommerce," SE Ranking Blog, [Online]. Available: <https://seranking.com/es/blog/sistema-de-busqueda-interno-ecommerce/> [Accessed: 23 Feb. 2026].

[3] Product Hackers, "Buscador avanzado en ecommerce," Product Hackers, [Online]. Available: <https://producthackers.com/es/blog/buscador-avanzado-ecommerce/> [Accessed: 23 Feb. 2026].

[4] UNIR Revista, "¿Qué es la metodología Agile y cómo funciona? Ejemplos," UNIR Revista: Ingeniería y Tecnología, [Online]. Available: <https://www.unir.net/revista/ingenieria/metodologia-agile/>

[Accessed: 25 Feb. 2026].

[5] GooApps, "Las 5 mejores metodologías de desarrollo de software," GooApps, Oct. 27, 2022. [Online]. Available: <https://gooapps.es/2022/10/27/las-5-mejores-metodologias-de-desarrollo-de-software/> [Accessed: 25 Feb. 2026].

[6] Dongee, "Django vs Node.js," Dongee, [Online]. Available: <https://www.dongee.com/tutoriales/django-vs-node-js/> [Accessed: 3 Mar. 2026].

[7] Tokio School, "React JS vs Vue JS," Tokio School, [Online]. Available: <https://www.tokioschool.com/noticias/react-js-vs-vue-js/> [Accessed: 3 Mar. 2026].

[8] Amazon Web Services (AWS), "The difference between MongoDB and PostgreSQL," AWS, [Online]. Available: <https://aws.amazon.com/es/compare/the-difference-between-mongodb-and-postgresql/> [Accessed: 3 Mar. 2026].

[9] Meilisearch, "Meilisearch vs Elasticsearch," Meilisearch Blog, [Online]. Available: <https://www.meilisearch.com/blog/meilisearch-vs-elasticsearch> [Accessed: 3 Mar. 2026].

[10] Django Software Foundation, "Models," Django Documentation, [Online]. Available: <https://docs.djangoproject.com/en/6.0/topics/db/models/> [Accessed: 16 Apr. 2026].

[15] Django REST Framework, "Serializers," Django REST Framework Documentation, [Online]. Available: <https://www.django-rest-framework.org/api-guide/serializers/> [Accessed: 22 Apr. 2026].

[21] Meilisearch, "What is Meilisearch?," Meilisearch Documentation, [Online]. Available: https://www.meilisearch.com/docs/learn/getting_started/what_is_meilisearch [Accessed: 3 Mar. 2026].

[24] Meilisearch, "Sort results," Meilisearch Documentation, [Online]. Available: https://www.meilisearch.com/docs/capabilities/filtering_sorting_faceting/how_to/sort_results [Accessed: 16 May-2026].

[26] Vue.js Team, "Component Basics," Vue.js Documentation, [Online]. Available: <https://vuejs.org/guide/essentials/component-basics.html> [Accessed: 16 Apr. 2026].

[28] Axios, "Request Config," Axios Documentation, [Online]. Available: https://axios-http.com/docs/req_config [Accessed: 8 Apr. 2026].

[32] YouTube, "Curso Django REST Framework," YouTube, [Online]. Available: https://www.youtube.com/watch?v=ydlwzwOUkyE&list=PLkVpKYNT_U9cl3hhVg_ROISY33uuBWZ_h [Accessed: 1 Mar. 2026].

ANEX A: REFERÈNCIES COMPLEMENTÀRIES

[11] Django Software Foundation, "ForeignKey," Django Documentation, [Online]. Available: <https://docs.djangoproject.com/en/6.0/ref/models/fields/#foreignkey> [Accessed: 18 Apr. 2026].

[12] Django Software Foundation, "Session Management," Django Documentation, [Online]. Available: <https://docs.djangoproject.com/en/6.0/topics/http/sessions/> [Accessed: 4 May 2026].

[13] Django Software Foundation, "Signals," Django Documentation, [Online]. Available: <https://docs.djangoproject.com/en/6.0/topics/signals/> [Accessed: 8 Apr. 2026].

[14] PyWombat, "Signals in Django," PyWombat, [Online]. Available: <https://pywombat.com/articles/signals-django> [Accessed: 8 Apr. 2026].

[16] Django REST Framework, "APIView," Django REST Framework Documentation, [Online]. Available: <https://www.django-rest-framework.org/api-guide/views/> [Accessed: 25 Apr. 2026].

[17] Django REST Framework, "Requests and Authentication (Custom API Keys)," Django REST Framework Documentation, [Online]. Available: <https://www.django-rest-framework.org/api-guide/authentication/#custom-authentication> [Accessed: 29 Apr. 2026].

[18] Django REST Framework, "Permissions," Django REST Framework Documentation, [Online]. Available: <https://www.django-rest-framework.org/api-guide/permissions/> [Accessed: 8 Apr. 2026].

[19] Hektor Profe, "Sistema de autenticación en Django REST," Hektor Profe, [Online]. Available: <https://hektorprofe.github.io/django/api-django-rest/sistema-autenticacion-registro/> [Accessed: 8 Apr. 2026].

[20] Django Software Foundation, "Cross Site Request Forgery Protection," Django Documentation, [Online]. Available: <https://docs.djangoproject.com/en/6.0/ref/csrf/> [Accessed: 8 Apr. 2026].

[22] Meilisearch, "Settings API Specification," Meilisearch Specifications, [Online]. Available: <https://specs.meilisearch.dev/specifications/text/0123-settings-api.html> [Accessed: 7 May 2026].

[23] Meilisearch, "List all settings," Meilisearch Documentation, [Online]. Available: <https://www.meilisearch.com/docs/reference/api/settings/list-all-settings> [Accessed: 14 May 2026].

[25] Meilisearch, "Search with POST," Meilisearch

Documentation, [Online]. Available: <https://www.meilisearch.com/docs/reference/api/search/search-with-post> [Accessed: 8 Apr. 2026].

[27] Vue.js Team, "Vue Router Guide," Vue Router Documentation, [Online]. Available: <https://router.vuejs.org/guide/> [Accessed: 8 Apr. 2026].

[29] Stack Overflow, "How to use CSRF token in Django integrated with VueJS," Stack Overflow, [Online]. Available: <https://stackoverflow.com/questions/67033262/how-to-use-csrf-token-in-django-integrated-with-vuejs> [Accessed: 8 Apr. 2026].

[30] YouTube, "Laravel VueJS Meilisearch Instant Modal Search," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=57JOt0tHzuo> [Accessed: 21 Apr. 2026].

[31] YouTube, "Meilisearch + Django Tutorial," YouTube, [Online]. Available: <https://www.youtube.com/watch?v=kAORL2KgK04> [Accessed: 8 Apr. 2026].

ANEX B: APLICACIÓ FINAL

A continuació s’inclouen una sèrie d’imatges referents a l’aplicació final desenvolupada.

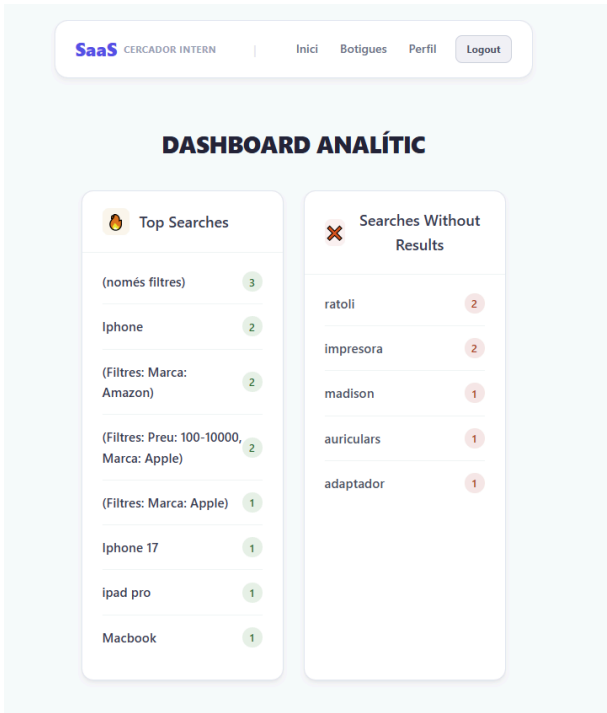


Fig. 1: Secció del Dashboard de la botiga Tech Components



Fig. 2: Secció del cercador final amb cerca tolerant a errors

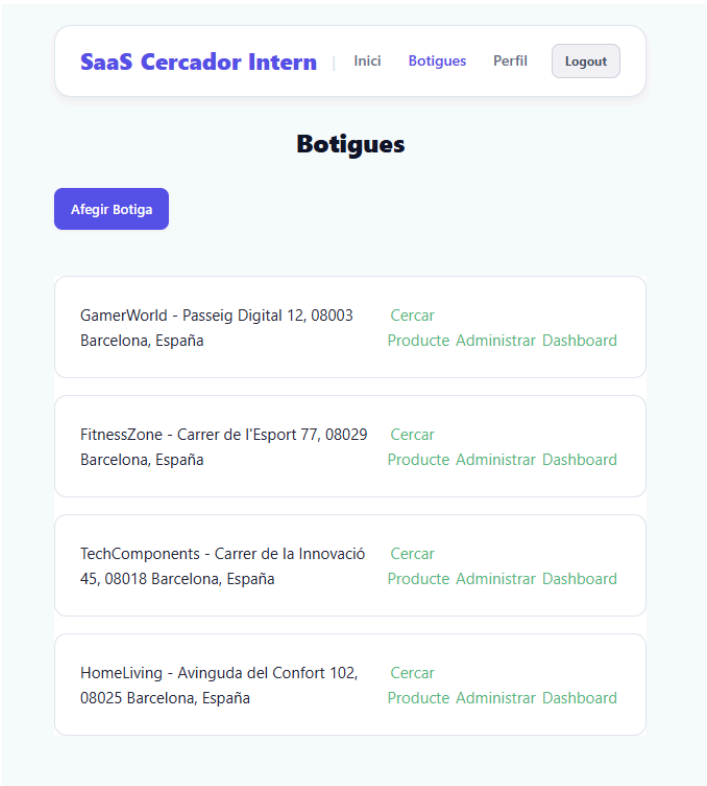


Fig. 3: Secció de botigues d’un administrador

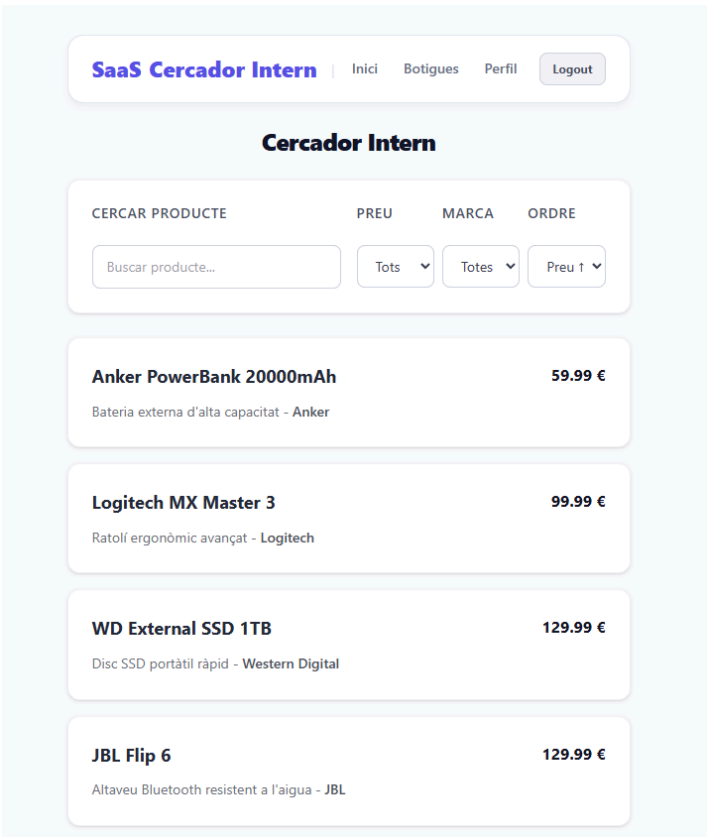


Fig. 4: Secció del cercador intern per administració

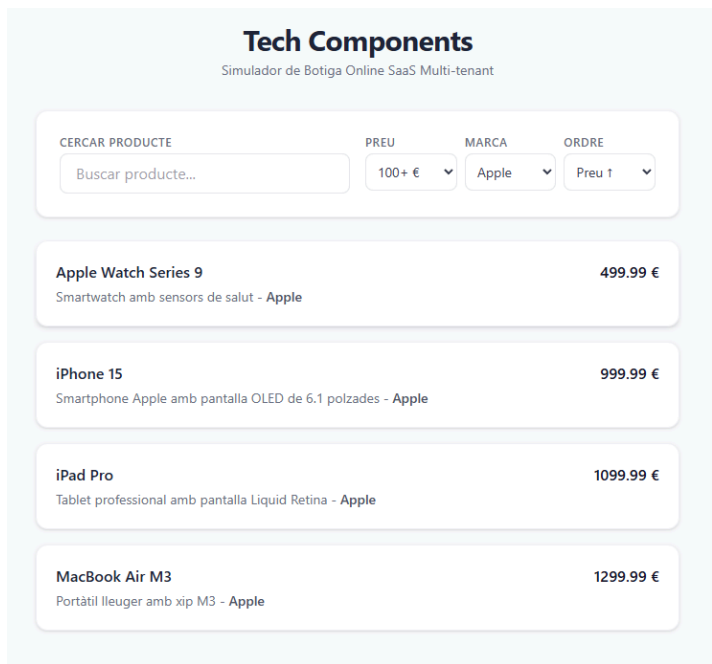


Fig. 5 : Secció del cercador final amb una cerca realitzada amb filtratge per marca i preu