Joel Teodoro Gómez

HTMX: La Revolució Senzilla del Desenvolupament Web

TREBALL DE FI DE GRAU

dirigit per Marc Sánchez

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Resum.

Aquest projecte es centra en l'anàlisi d'HTMX com una alternativa revolucionària als frameworks tradicionals com React en el desenvolupament web. S'ha desenvolupat una aplicació de comerç electrònic utilitzant HTMX per al frontend, demostrant la seva capacitat per simplificar la creació de pàgines web dinàmiques sense la necessitat de JavaScript addicional. La metodologia inclou la integració d'HTMX amb Go i PostgreSQL, destacant com HTMX permet una interactivitat eficient i una reducció significativa de la complexitat en el desenvolupament frontend. Els resultats mostren que HTMX és una eina potent per a projectes que requereixen interacció moderada, oferint una experiència de desenvolupament més fluida i lleugera.

Resumen.

Este proyecto se centra en el análisis de HTMX como una alternativa revolucionaria a los frameworks tradicionales como React en el desarrollo web. Se ha desarrollado una aplicación de comercio electrónico utilizando HTMX para el frontend, demostrando su capacidad para simplificar la creación de páginas web dinámicas sin la necesidad de JavaScript adicional. La metodología incluye la integración de HTMX con Go y PostgreSQL, destacando cómo HTMX permite una interactividad eficiente y una reducción significativa de la complejidad en el desarrollo frontend. Los resultados muestran que HTMX es una herramienta poderosa para proyectos que requieren interacción moderada, ofreciendo una experiencia de desarrollo más fluida y ligera.

Abstract.

This project focuses on analyzing HTMX as a groundbreaking alternative to traditional frameworks like React in web development. An e-commerce application was developed using HTMX for the frontend, showcasing its ability to simplify the creation of dynamic web pages without the need for additional JavaScript. The methodology includes integrating HTMX with Go and PostgreSQL, highlighting how HTMX enables efficient interactivity and significantly reduces complexity in frontend development. The results demonstrate that HTMX is a powerful tool for projects requiring moderate interaction, offering a smoother and lighter development experience.

Índex

Índex de taules

Índex de figures

1 Introducció

1.1 Context i Motivacions

Acabo de finalitzar la carrera d'Enginyeria Informàtica, i després de dedicar molt de temps a l'estudi de diverses tecnologies, em vaig sentir atret pel món del desenvolupament web, especialment pel frontend[1]. Vaig escoltar parlar d'un terme que em va cridar molt l'atenció: HTMX. Aquest terme em va resultar molt curiós perquè generava opinions molt polaritzades entre els desenvolupadors. Mentre alguns el criticaven severament, altres en parlaven amb gran entusiasme; semblava que no hi havia terme mig: o l'odiaves, o l'amaves.

A més, vaig notar que avui dia la web sembla estar dominada per un monoparadigma, on gairebé tot es fa amb React. Això em va portar a plantejar-me si realment era necessari seguir aquest camí o si existien alternatives que podien ser igualment efectives. Vaig veure en HTMX una oportunitat per explorar un nou paradigma en el desenvolupament web, un que prometia simplificar el procés de creació de pàgines web dinàmiques sense la necessitat d'una llibreria o framework[2] tan robust i complexe.

La meva curiositat em va portar a voler comprovar si realment serveix per construir webs de manera eficient. Volia saber si era tan fàcil d'utilitzar com es deia, si era tan ràpid i si podia ser una eina poderosa capaç de competir amb React en certs contextos. Segons moltes opinions, la gran majoria de les webs, potser un 80% o fins i tot un 90%, es podrien fer només amb HTMX. Així doncs, vaig decidir posar-ho a prova en aquest projecte, aprendre alguna cosa nova que em pogués ser útil en el futur, i al mateix temps veure si podia ser una alternativa viable en el desenvolupament web modern.

1.2 Objectius del Projecte

L'objectiu principal d'aquest projecte és desenvolupar una aplicació de comerç electrònic per avaluar l'eficàcia i l'eficiència de l'ús de HTMX en el frontend com a alternativa de React, en el context d'una aplicació web funcional.

Els objectius específics són els següents:

1. **Explorar i avaluar HTMX**: Comprendre com pot simplificar el desenvolupament de frontends interactius sense necessitat de llibreries o frameworks pesats, i avaluar si és una solució viable per a la majoria de projectes web.
2. **Implementació d'una aplicació de comerç electrònic**: Desenvolupar una aplicació funcional amb Go com a backend i PostgreSQL com a base de dades, que inclou funcionalitats essencials com la gestió de productes, cistella de la compra, processament de comandes i autenticació d'usuaris.
3. **Integració i seguretat**: Assegurar que l'aplicació sigui segura mitjançant l'ús de JWT[3] per a l'autenticació, la implementació de middleware[4] per gestionar les rutes

[1] Frontend: és la part de l'aplicació amb la qual interactua directament l'usuari

[2] Framework: Estructura de suport per desenvolupar aplicacions, que inclou eines i biblioteques

[3] JWT: JSON web tokens

protegides, i l'ús de pràctiques segures en la manipulació de dades i l'execució de consultes SQL.

Avaluació del rendiment: Analitzar el rendiment de l'aplicació, comparant la velocitat i la capacitat de resposta d' HTMX en comparació amb altres solucions més convencionals.

[4] Middleware: Programari intermedi que connecta diferents components d'una aplicació, facilitant la comunicació i la gestió de dades entre ells.

2 Descripció General del Projecte

2.1 Entorn

Per a la part pràctica del projecte he desenvolupat una aplicació web de comerç electrònic (e commerce) especialitzada en la venda de productes electrònics, un sector en constant creixement i evolució. Com ja explicaré en el següent apartat, l'entorn tecnològic en el qual s'ha desenvolupat el projecte inclou HTMX per al frontend, i el backend[5] està implementat en Go.

L'entorn de desenvolupament també inclou altres eines modernes com Git per al control de versions, Docker per al desplegament de l'aplicació, i Visual Studio Code com a editor principal. Aquests elements han facilitat el desenvolupament col·laboratiu i la integració contínua durant tot el procés.

2.2 Necessitats

El projecte respon a diverses necessitats tant tècniques com d'usuari:

- **Necessitats dels usuaris**: Els usuaris necessiten una plataforma de comerç electrònic senzilla, ràpida i segura, on puguin navegar per una àmplia gamma de productes electrònics, afegir-los a la cistella, i realitzar compres de manera àgil i eficient. A més, "user experience" ha de ser fluida, amb temps de resposta ràpid i interaccions dinàmiques que no requereixin la recàrrega completa de la pàgina.
- **Necessitats tècniques**: Des del punt de vista del desenvolupament, calia una solució que permetés implementar aquestes funcionalitats amb el mínim cost de complexitat possible. HTMX ofereix la possibilitat de crear aplicacions web dinàmiques sense la sobrecàrrega de frameworks més pesats, mentre que Go em proporciona un backend eficient i fàcil de mantenir, capaç de gestionar la complexitat de les operacions de comerç electrònic de manera ràpida.
- **Seguretat**: Amb la gestió de dades personals, la seguretat és una prioritat absoluta. Era necessari implementar sistemes d'autenticació i autorització robustos, assegurar la integritat de les dades i protegir l'aplicació contra possibles vulnerabilitats com XSS[6], CSRF[7], i SQL[8] Injection, més endavant explicaré com he conseguir mitigar aquestos atacs.

2.3 Previsions d'Ús

L'aplicació està dissenyada per ser utilitzada per un ampli ventall d'usuaris, des de compradors ocasionals fins a clients habituals que realitzin compres freqüents de productes electrònics. Amb aquesta finalitat, s'han previst els següents usos i escenaris:

---

[5] Backend: infraestructura que permet que l'aplicació funcioni.

[6] XSS: Cross-site scripting

[7] CSRF: Cross-site request forgery

[8] SQL: Structured Query Language

- **Compra de productes electrònics**: Els usuaris poden navegar per les diferents categories de productes electrònics, com ara ordinadors, smartphones, televisors i altres dispositius, afegir articles a la seva cistella i realitzar la compra de manera ràpida i segura.
- **Gestió d'usuari**: Els usuaris poden registrar-se, iniciar sessió, i gestionar les seves dades personals i històric de compres. L'autenticació es gestiona mitjançant JWT, oferint una experiència fluida sense necessitat de mantenir sessions al servidor.
- **Escalabilitat**: Tot i que el projecte inicial està pensat per a un nombre limitat d'usuaris, la solució està dissenyada amb l'objectiu de poder escalar fàcilment per a un ús més massiu. L'arquitectura basada en Go i PostgreSQL permet adaptar-se a un major volum de dades i usuaris sense comprometre el rendiment, fet que és essencial en un mercat de comerç electrònic on la demanda pot créixer significativament en períodes curts de temps.

3 Tecnologies i Eines Utilitzades

Aquest apartat estarà dedicat a mencionar les eines que he utilitzat per al desenvolupament de l'ecommerce. Considero que és essencial incloure una breu introducció teòrica, especialment sobre HTMX i les plantilles (*templates*). Tot i que aquest contingut és senzill, proporciona les bases necessàries per entendre correctament la part pràctica que explicaré a continuació. D'aquesta manera, serà més fàcil seguir i aplicar els conceptes pràctics que es presentaran posteriorment.

3.1 HTMX

3.1.1 Què és HTMX?

HTMX és una biblioteca de javascript que permet estendre les capacitats d'HTML[9], donant-li accés a funcionalitats avançades com AJAX[10], transicions de CSS[11], WebSockets i Server-Sent Events mitjançant l'ús d'atributs HTML. A diferència d'altres biblioteques o frameworks, no es preocupa de transformar JSON[12], mapar dades o gestionar plantilles. És per això que és recomanable utilitzar HTMX en combinació amb un sistema de plantilles que pugui generar HTML de manera eficient i segura.

L'objectiu principal d'HTMX és reduir l'ús de JavaScript en els projectes, permetent crear webs dinàmiques de manera més senzilla. És aquest un dels motius principal de la meva tria per al desenvolupament d'aquest projecte.

3.1.2 Funcionament d'HTMX

Funciona principalment a través de la manipulació d'atributs HTML, que defineixen les accions que l'element ha de realitzar quan l'usuari interacciona amb ell. A continuació s'expliquen les passes que es segueixen en una operació típica, il·lustrat amb els diagrames següents:

3.1.2.1 Procés de Sol·licitud i Resposta

Quan un element HTML conté un atribut hx-verb (on verb pot ser get, post, put, etc.), aquest defineix quin tipus de sol·licitud HTTP[13] s'enviarà quan l'usuari interaccioni amb l'element. Per exemple, en l'exemple següent:

```
<el hx-post="/path"></el>
```

Quan l'usuari fa una acció que activa aquest element (com un clic), HTMX enviarà una sol·licitud POST al camí /path del servidor. El servidor processa la sol·licitud i retorna

[9] HTML: Hypertext Markup Language

[10] AJAX: Asynchronous JavaScript And XML

[11] CSS: Cascading Style Sheets

[12] JSON: JavaScript Object Notation

[13] HTTP: Hypertext Transfer Protocol

una resposta en format HTML. Si la resposta té un codi d'estat 200 (èxit), el contingut retornat reemplaçarà el contingut interior de l'element que va originar la sol·licitud.

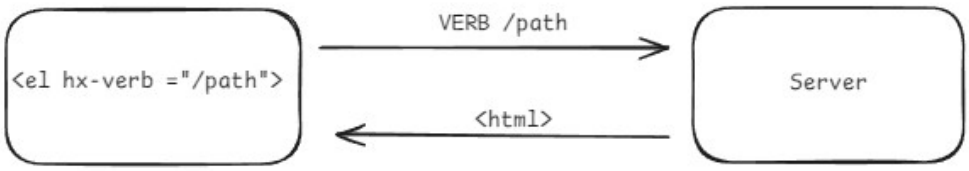

Figura 1: Procés de Sol·licitud i Resposta

3.1.2.2 Actualització de Contingut

En aquest exemple, si un conté l'atribut hx-post="/foo", i inicialment conté quatre quadres verds, quan el servidor respon amb dos quadres blaus, el contingut del s'actualitzarà per mostrar només aquests dos quadres blaus.

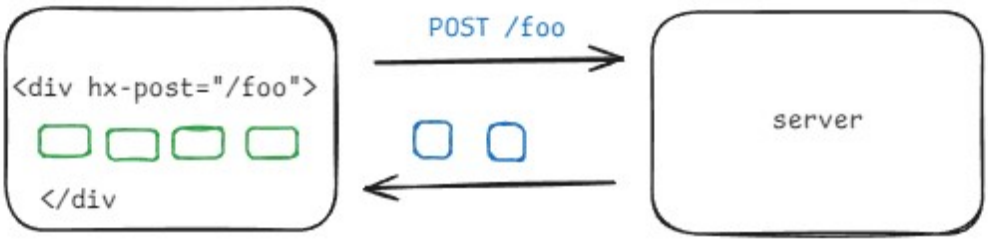

Figura 2: Exemple petició

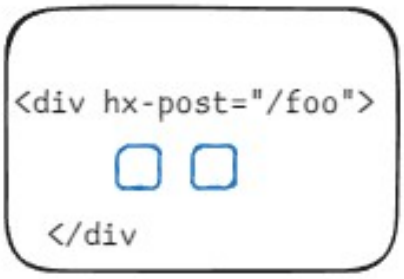

Figura 3: Actualització de Contingut

3.1.2.3 Esdeveniments i Redireccions

També emet diversos esdeveniments durant el procés de sol·licitud i resposta, permetent als desenvolupadors afegir comportaments personalitzats mitjançant JavaScript si és necessari o sense javascript amb algunes etiquetes que ens proporciona. A més, ens ofereix maneres de reduir la quantitat de HTML retornat o redirigir el contingut a altres parts de l'aplicació. Això ens proporciona una gran flexibilitat en la gestió del flux de dades i la interfície d'usuari.

3.1.3 Elements d'HTMX

En aquest subapartat es farà un breu resum de les etiquetes d'HTMX que considero més útils i que s'han fet servir en el projecte. Amb aquestes eines és possible cobrir pràcticament totes les necessitats en el desenvolupament d'una web.

3.1.3.1 Etiquetes AJAX

Les etiquetes AJAX són essencials perquè permeten la interacció entre el client i el servidor sense necessitat de recarregar tota la pàgina. Les etiquetes més comunes són:

- **hx-get**: Envia una sol·licitud GET al servidor i actualitza l'element amb la resposta.
- **hx-post**: Envia una sol·licitud POST, sovint utilitzada per enviar dades com en els formularis.
- **hx-put** i **hx-delete**: Enviament de sol·licituds PUT i DELETE, respectivament, per actualitzar o eliminar recursos.

3.1.3.2 hx-target

L'etiqueta hx-target especifica on s'ha d'inserir el contingut retornat per la sol·licitud HTTP dins del DOM[14]. Per defecte, HTMX reemplaça el contingut de l'element que va

[14] DOM: Document Object Model

originar la sol·licitud, però amb hx-target podem especificar un altre element on es vol fer la inserció. Això permet un control més gran sobre l'estructura de la pàgina i sobre com es volen mostrar les dades.

Exemple:

```
<form hx-post="/submit" hx-target="#result"></form>
<div id="result"></div>
```

En aquest cas, la resposta del servidor es mostrarà en l'element amb l'id #result. A més, si estem treballant amb templates que s'utilitzen en diferents parts de la pàgina, podem aprofitar hx-target per referenciar elements d'altres templates.

3.1.3.3 hx-swap

L'etiqueta hx-swap controla com s'insereix el contingut retornat pel servidor al DOM. Es poden especificar diferents modes de substitució, però les més útils en la meva opinió són:

- **innerHTML**: Reemplaça el contingut interior de l'element objectiu.
- **outerHTML**: Reemplaça completament l'element objectiu.
- **delete**: Elimina l'element objectiu del DOM.
- **none**: No realitza cap canvi en el DOM, útil per a operacions on la resposta no necessita visualització.
- **beforeend**: Insereix el contingut al final d'una llista o altre tipus d'element contenedor.

Per exemple, per afegir nous elements a una llista, beforeend és molt pràctic. A més, per fer que les transicions siguin més fluïdes, es pot habilitar l'opció transitions:true, que utilitza la tecnologia de *view transitions* disponible a partir de la versió 1.9.0 d'HTMX. Això permet efectes com el *fade in* i *fade out*, que també es poden personalitzar segons les necessitats.

3.1.3.4 hx-trigger

L'etiqueta hx-trigger especifica l'esdeveniment que ha de disparar la sol·licitud HTTP. Alguns dels esdeveniments més comuns són:

- **click**: Activa la sol·licitud quan l'usuari fa clic a l'element.
- **change**: Activa la sol·licitud quan el valor d'un input canvia.
- **keyup**: Activa la sol·licitud quan l'usuari apreta una tecla.

Es poden aplicar retards (delay) per controlar el temps abans que s'executi la sol·licitud o fins i tot reaccionar a l'esdeveniment de càrrega (load). Aquesta flexibilitat permet que HTMX sigui altament adaptable a diferents situacions i necessitats de l'aplicació.

També podem reaccionar a events, més endavant exposare algun exemple sobre l'ús d'aquests.

3.1.3.5 Headers de Request i Response

Finalment, HTMX també permet inclou *headers* específics tant en les sol·licituds com en les respostes HTTP. Especialment els de response m'han resultat els més útils, entre ells:

- **HX-Location**: Permet canviar la localització del navegador després d'una sol·licitud, fent que el navegador es redirigeix-hi a una nova pàgina o secció sense haver de carregar la pàgina.
- **HX-Trigger-After-Swap**: Permet activar esdeveniments del costat del client després del pas de "swap". Per mi, aquest és el header més útil, ja que permet desencadenar *triggers* d'altres elements després que s'hagi actualitzat el contingut del DOM.

Un exemple pràctic de l'ús del header HX-Trigger-After-Swap en el meu projecte de comerç electrònic és la implementació d'un comptador dinàmic per al carret de compra. Aquest comptador es representa amb un icona que mostra el nombre de productes actuals dins del carret. Més endavant explicaré la implementació d'aquesta funcionalitat.

3.2 Backend amb Go (Golang)

En el desenvolupament d'aplicacions web dinàmiques, és fonamental disposar d'un llenguatge que suporti sistemes de plantilles (templating) per a generar contingut HTML de manera dinàmica des del servidor. En aquest projecte, he triat Go (Golang) com a llenguatge de programació per al backend, tant per la seva capacitat per gestionar plantilles de manera eficient com per les seves característiques tècniques i popularitat creixent.

Go és un llenguatge de programació de codi obert, dissenyat i desenvolupat per Google, que es destaca per la seva velocitat, simplicitat i eficiència. Combina una sintaxi clara i concisa amb una alta capacitat de processament, la qual cosa el converteix en una opció ideal per al desenvolupament de sistemes backend escalables i de rendiment elevat.

Els motius que m'han portat a triar Go per aquest projecte són diversos:

- **Velocitat i Eficiència**: És molt ràpid, gràcies a la seva compilació directa a codi màquina i la seva gestió eficient de la concurrència. És un aspecte crucial en un projecte ecommerce, on la velocitat de resposta i la capacitat de manejar múltiples peticions simultànies són essencials per oferir una bona experiència d'usuari.
- **Simplicitat**: Go ha estat dissenyat per ser simple, amb una sintaxi neta que facilita el desenvolupament i manteniment del codi. Aquesta simplicitat no només redueix la corba d'aprenentatge, sinó que també permet escriure codi més segur i fiable amb menys probabilitat d'errors.
- **Popularitat**: També ha guanyat molta popularitat en els últims anys, amb una comunitat activa i en creixement. Això significa que hi ha una gran quantitat de recursos, biblioteques i eines disponibles, cosa que facilita el desenvolupament i la resolució de problemes.

Finalment, un altre motiu per triar Go és la meva intenció de masteritzar aquest llenguatge durant aquest any. No només s'ha convertit en una opció preferida per a

projectes de gran escala, sinó que també ofereix oportunitats d'aprenentatge i creixement professionals molt valuoses.

3.2.1 Dependències Utilitzades en el Backend

En el desenvolupament del backend, he utilitzat dues biblioteques principals per a la gestió del servidor i la generació de plantilles: net/http i html/template. En aquest projecte, he optat per mantenir el mínim de dependències possibles, aprofitant aquestes dues biblioteques natives de Go que ofereixen tot el que necessito sense afegir complexitat innecessària.

3.2.1.1 Biblioteca net/http

Aquesta biblioteca és clau per al funcionament del servidor, ja que permet gestionar de manera eficient totes les peticions i respostes entre el client i el servidor, garantint un servei web robust i escalable. Al ser nativa de Go, no es necessita instal·lar cap paquet extern per començar a utilitzar-la.

Funcions Bàsiques de net/http:

- **http.HandleFunc**: Aquesta funció permet associar una ruta específica del servidor amb una funció que gestionarà les peticions que arribin a aquesta ruta. Per exemple, si es defineix http.HandleFunc("/", homeHandler), totes les peticions a la ruta / seran gestionades per la funció homeHandler.
- **http.ListenAndServe**: És la funció que posa en marxa el servidor HTTP. Aquesta funció escolta les peticions entrants en un port especificat (per exemple, http.ListenAndServe(":8080", nil)) i les deriva a les funcions corresponents segons la ruta.
- **http.Request i http.ResponseWriter**: Aquests són dos tipus essencials dins de net/http. http.Request conté tota la informació de la petició entrant, com ara el mètode HTTP, els headers i el cos de la petició. http.ResponseWriter és utilitzat per enviar una resposta HTTP al client, incloent-hi l'estatus, els headers i el cos de la resposta.

3.2.1.2 Biblioteca html/template

Sóc conscient que les plantilles no són atractives en el món actual. Són simples però poden arribar a ser molestes. Per la raó esmentada anteriorment, al ser senzilles la ventatja que tenen és requereixen gairebé cap comprensió per a seguir-les. En resum, aquesta biblioteca permet combinar dades del servidor amb plantilles HTML, facilitant la interacció amb HTMX i la creació de pàgines web complexes d'una manera eficient i sobretot segura.

Funcions Bàsiques de html/template:

- **template.ParseFiles**: Aquesta funció permet carregar un o més fitxers de plantilles HTML i preparar-los per a la seva execució. Per exemple,

template.ParseFiles("index.html") carregarà la plantilla index.html per ser utilitzada més endavant.

- **template.Execute**: Aquesta funció s'utilitza per combinar una plantilla carregada amb dades i escriure el resultat al http.ResponseWriter, que serà enviat al client com a resposta. Per exemple, tmpl.Execute(w, data) renderitza la plantilla tmpl amb les dades contingudes a data i envia el resultat al client.
- **template.ExecuteTemplate:** Similar a Execute, però es fa servir quan es treballa amb un conjunt de plantilles carregades. ExecuteTemplate permet especificar quina plantilla dins del conjunt es vol renderitzar. Això és útil quan es carreguen múltiples plantilles que comparteixen dades o quan es vol renderitzar diferents vistes en funció d'una mateixa estructura de dades. En aquest exemple, tmpl.ExecuteTemplate(w, "nomPlantilla", data) selecciona i renderitza la plantilla anomenada nomPlantilla dins del conjunt de plantilles carregades, utilitzant les dades contingudes a data i enviant el resultat al client.
- **template.New i template.Funcs**: Són utilitzats per crear noves plantilles des de codi o per afegir funcions personalitzades a les plantilles, com funcions de format o manipulació de dades abans de ser renderitzades.

Expressions i Control de Flux en les Plantilles:

Una de les característiques més potents que té és la seva capacitat per incloure expressions i control de flux dins de les plantilles, cosa que permet construir interfícies dinàmiques i adaptatives.

- **{{if ...}} i {{else}}:** Aquestes expressions permeten afegir condicions dins de les plantilles. Per exemple, es pot mostrar un bloc de HTML només si es compleix una condició determinada
- **{{range ...}}:** Aquesta expressió permet iterar sobre un *slice* o un *array* dins de la plantilla, generant HTML per a cada element de la col·lecció. Amb range es poden mostrar tots els productes en una llista.
- **{{define ...}} i {{template ...}}:** Amb {{define}}, es poden definir blocs de plantilles que es poden reutilitzar en altres llocs de la plantilla amb {{template}}. Això facilita molt la reutilització de codi i el manteniment de les plantilles.

A més a més, quan executem una plantilla, li podem passar dades, normalment en format d'estructura (*struct*), per poder gestionar una quantitat més gran d'informació de manera organitzada. Aquestes dades es poden accedir de manera natural dins de la plantilla, permetent iterar, condicionar i mostrar informació dinàmica de forma senzilla i intuïtiva, utilitzant les expressions descrites anteriorment.

3.3 PostgreSQL: Gestió i Consulta de Dades

PostgreSQL és un sistema de gestió de bases de dades relacionals de codi obert, conegut per la seva robustesa, flexibilitat i conformitat amb els estàndards SQL. L'he escollit com a base de dades per al projecte per diverses raons clau:

- **Popularitat i Comunitat Activa:** Actualment és una de les bases de dades més populars i compta amb una gran comunitat, oferint una gran quantitat de recursos, documentació i eines que faciliten el desenvolupament i manteniment.
- **Simplicitat i Familiaritat:** També és fàcil d'utilitzar, amb una sintaxi SQL estàndard que és accessible per a tots els nivells de desenvolupadors. A més, ja tenia experiència prèvia amb aquesta base de dades, la qual cosa ha simplificat la seva integració en el projecte.
- **Robustesa i Fiabilitat:** PostgreSQL és conegut per la seva capacitat de gestionar grans volums de dades i executar consultes complexes de manera eficient, sent una opció fiable per a aplicacions que requereixen estabilitat i escalabilitat.

Alternatives Considerades: Tot i haver considerat altres opcions com SQLite, que és més lleugera i adequada per a projectes més petits, finalment m'he decantat per PostgreSQL. Com he esmentat anteriorment, ja tenia experiència prèvia amb aquesta base de dades, i ja que l'objectiu principal del treball és utilitzar HTMX, no m'he volgut complicar massa amb la tria.

3.4 Altres eines de Treball

Finalment, per al desenvolupament i validació del projecte, he utilitzat diverses eines que han facilitat el procés i han garantit la qualitat del resultat final. A continuació, es descriuen les eines principals que han estat emprades i els motius pels quals les he triat:

- **Postman:** Per a la validació de diverses APIs desenvolupades en el backend, he utilitzat Postman. Aquesta eina m'ha permès realitzar proves exhaustives de les peticions i respostes HTTP, assegurant que les APIs funcionen correctament. L'he triada perquè ofereix una interfície intuïtiva per construir, provar i documentar les APIs, la qual cosa ha simplificat molt el procés de validació.
- **Visual Studio Code (VSCode):** VSCode ha estat l'entorn de desenvolupament integrat (IDE[15]) que he utilitzat per a aquest projecte. La meva tria es deu a les múltiples extensions disponibles per a Golang i HTMX, que han facilitat enormement el desenvolupament tant del backend com del frontend. Un altre motiu de pes es que proporciona funcions com autocomplete i descripcions, que ajuden a comprendre millor el codi i prevenir errors.

[15] IDE: integrated development environment

- **Excalidraw:** Per a la creació de gràfiques i diagrames que il·lustren diferents parts del projecte, he utilitzat Excalidraw. Aquesta eina m'ha permès generar representacions visuals de l'arquitectura, fluxos de dades i altres aspectes importants del sistema de manera clara i comprensible. La he triat perquè és la més senzilla d'utilitzar.
- **Tailwind CSS:** Per al disseny i els styles en el frontend, he utilitzat Tailwind CSS, un framework CSS que facilita la creació de dissenys responsius i moderns sense haver d'escriure fulls d'estil des de zero. Malgrat que els meus coneixements de CSS són limitats, m'he decantat per aquesta opció per la seva popularitat i la seva facilitat d'ús.
- **Icones SVG:** L'ús d'icones és essencial en un e-commerce per millorar l'experiència d'usuari al frontend, aportant elements visuals que complementen la interfície d'usuari. He triat icones en format SVG[16] perquè són lleugers, escalables i compatibles amb tots els navegadors moderns, la qual cosa assegura que els elements visuals es veuran bé en qualsevol dispositiu.
- **Docker:** Docker m'ha facilitat la gestió de les dependències del projecte i el seu desplegament. Tot el sistema, incloent el backend en Go, la base de dades PostgreSQL, i el frontend, es pot empaquetar en contenidors lleugers que es poden executar en qualsevol entorn compatible amb Docker.
- **GitHub:** Per al control de versions he utilitzat GitHub. Aquesta plataforma em permet mantenir un registre complet de tots els canvis realitzats en el codi i gestionar branques de desenvolupament de manera eficient. En cas que hi hagi un error o problema amb alguna implementació em permet fer un rollback. A més, tenir el projecte penjat a GitHub és un avantatge de cara a trobar feina, ja que permet mostrar el meu treball a futurs ocupadors o col·laboradors de manera accessible i professional.

[16] SVG: Scalable Vector Graphics

4 Requisits

Els requisits funcionals i no funcionals són especificacions que descriuen les característiques i criteris que un sistema, aplicació o producte ha de complir. Aquestes especificacions són essencials en el procés de desenvolupament de programari i sistemes, ja que guien desenvolupadors, dissenyadors i tots els involucrats en la creació i avaluació del projecte.

4.1 Requisits funcionals

4.1.1 Gestió d'Usuaris

- **RF1: Iniciar sessió:** Els usuaris podran iniciar sessió introduint les seves credencials vàlides.
- **RF2: Registrar-se:** Els nous usuaris podran crear un compte omplint un formulari de registre.
- **RF3: Navegació entre registre i login:** Els usuaris podran navegar entre els dos formularis via enllaç.
- **RF4: Fer log out:** Els usuaris podran tancar la seva sessió de manera segura.
- **RF5: Restablir contrasenya:** Els usuaris podran restablir la seva contrasenya seguint l'enllaç proporcionat.

4.1.2 Gestió de Productes

- **RF6: Visualitzar productes:** Els usuaris podran veure un producte en concret disponible amb la seva informació detallada.
- **RF7: Filtrar productes:** Els usuaris podran filtrar els productes per nom des del camp de cerca (search) o per tipus des de la barra de navegació (navbar).
- **RF8: Afegir productes al carret:** Els usuaris podran afegir productes al carret de compra.
- **RF9: Modificar quantitat de productes al carret:** Els usuaris podran augmentar o disminuir la quantitat de productes al carret.
- **RF10: Eliminar productes del carret:** Els usuaris podran eliminar productes del carret de compra.

4.1.3 Gestió del Carret i Comandes

- **RF11: Visualitzar el carret:** Els usuaris podran veure el contingut del seu carret de compra, incloent-hi productes, quantitats i preus.
- **RF12: Procedir al pagament:** Els usuaris podran procedir al pagament dels productes al carret i completar la compra.
- **RF13: Visualitzar historial de comandes:** Els usuaris podran veure un historial de les seves compres anteriors, incloent-hi els detalls de cada factura.

4.2 Requisits no funcionals

4.2.1 Rendiment i Escalabilitat

- **RNF1**: Temps de resposta: El sistema ha de respondre ràpidament a qualsevol acció de l'usuari per garantir una experiència fluida. L'ús de *templating* i HTMX permet renderitzar només les parts de la pàgina que canvien, evitant la necessitat de recarregar tota la pàgina.
- **RNF2: Suport de càrrega:** L'aplicació ha de poder gestionar bastantes transaccions simultànies sense degradació del rendiment (per això l'ús de Go), tenint en compte l'escala de l'e-commerce.

4.2.2 Seguretat

- **RNF3: Encriptació de dades:** Totes les dades sensibles, com les credencials d'usuari i la informació personal, han d'estar encriptades utilitzant la llibreria bcrypt per garantir la confidencialitat.
- **RNF4: Autenticació segura:** El sistema utilitzarà JSON Web Tokens per a gestionar sessions d'usuari de manera segura juntament amb un middleware.
- **RNF5: Protecció contra atacs:** El sistema ha d'implementar mesures de seguretat per protegir-se contra atacs comuns com SQL injection, Cross-Site Scripting i Cross-Site Request Forgery.

4.2.3 Usabilitat

- **RNF6: Compatibilitat de navegador:** L'aplicació ha de ser totalment funcional en les últimes versions de Chrome, Firefox, Safari, i Edge, assegurant una experiència d'usuari consistent independentment del navegador utilitzat.
- **RNF7: Disseny responsive:** L'aplicació ha de ser totalment responsive, garantint que els usuaris puguin accedir-hi i utilitzar-la des de dispositius mòbils, tablets i ordinadors de sobretaula sense problemes. Això es pot implementar amb tailwind.

4.2.4 Mantenibilitat

- **RNF8: Revisió constant del codi:** El codi de l'aplicació ha de ser revisat regularment per detectar possibles errors i millorar la qualitat del codi. Aquesta pràctica permet identificar problemes abans que es converteixin en errors greus i garanteix que el sistema es mantingui estable i segur.
- **RNF9: Simplicitat i claredat en el codi:** L'aplicació segueix un model de codi simple i ben estructurat, amb noms de variables intuïtives i clares, facilitant així la comprensió i el manteniment del codi. Aquest enfocament redueix el temps necessari per trobar i corregir errors, millorant l'eficiència del desenvolupament i manteniment a llarg termini.

4.2.5 Interoperabilitat

- **RNF10: Integració amb sistemes de pagament:** El sistema en un futur ha de ser capaç d'integrar-se fàcilment amb passarel·les de pagament com Stripe,

tant a través d'API com de webhooks[17], per gestionar transaccions de manera segura.

4.3 Casos d'ús

Els casos d'ús són una descripció detallada dels processos específics que un usuari realitza amb un sistema per assolir un objectiu concret. En el context del desenvolupament del meu projecte de comerç electrònic, aquests casos d'ús representen una sèrie d'interaccions entre l'usuari i l'aplicació, com per exemple, navegar per productes, afegir articles al carret de compra, o completar una transacció.

Els casos d'ús són fonamentals per al desenvolupament d'un ecommerce perquè ajuden a definir clarament les funcionalitats que el sistema ha de proporcionar des del punt de vista dels usuaris finals. A més, permeten establir un full de ruta clar per als desenvolupadors i dissenyadors, assegurant que totes les necessitats operatives es cobreixin adequadament. També serveixen com a base per a les proves del sistema, ja que cada cas d'ús proporciona un escenari concret per a verificar el comportament de l'aplicació en situacions reals.

En aquest apartat, descriuré els casos d'ús que considero més importants per al correcte funcionament de la meva aplicació de comerç electrònic. Aquests reflecteixen les funcionalitats essencials que el sistema ha de suportar per a complir amb els objectius establerts i proporcionar una experiència d'usuari òptima.

És important destacar que, per la simplicitat que tenen alguns, com per exemple mostrar els detalls d'un producte, no és necessari descriure'ls detalladament, ja que la seva implementació és directa i no presenta grans reptes tècnics. Per aquesta raó, he optat per centrar-me en aquells casos d'ús que considero més rellevants i complicats de programar, com ara el procés d'inici de sessió, el registre d'usuaris o el filtratge de productes.

4.3.1 01. Iniciar sessió

Resum de la funcionalitat: Permetre als usuaris existents accedir al sistema introduint les seves credencials vàlides.

Paràmetres d'entrada:

- **Nom d'usuari:** El nom d'usuari registrat.
- **Contrasenya:** La contrasenya associada a l'usuari.

Paràmetres de sortida: Cap directament, però si ja estem loguejats, l'usuari serà redirigit a la home Page.

Actors:

- **Usuari:** Persona que ja té un compte al sistema i desitja accedir-hi.

Precondicions:

[17] Webhook: Mètode per lliurar notificacions automàtiques entre aplicacions, enviant dades en temps real quan es produeix un esdeveniment específic.

- L'usuari ha de tenir un compte ja creat al sistema.

Postcondicions:

- Si les credencials són correctes, el sistema permetrà a l'usuari accedir al seu compte i el redirigirà a la home Page.
- Si les credencials són incorrectes, el sistema mostrarà un missatge d'error i l'usuari no podrà accedir.

Procés normal principal:

1. L'usuari obre la pàgina d'inici de sessió.
2. El sistema presenta un formulari amb camps per al nom d'usuari i la contrasenya.
3. L'usuari introdueix el seu nom d'usuari i contrasenya en els camps corresponents.
4. L'usuari fa clic al botó de Login.
5. El sistema valida les credencials contra la base de dades i verifica si l'usuari existeix.
6. Si les credencials són correctes, el sistema crea una sessió per a l'usuari i el redirigeix a la home Page.
7. Si les credencials no són correctes, el sistema mostra un missatge d'error i permet a l'usuari intentar-ho de nou.

Alternatives de procés i excepcions:

- **Camps buits:**
    - Si el nom d'usuari o la contrasenya no es proporcionen, el sistema mostra un missatge d'error indicant que tots els camps són obligatoris.
- **Credencials incorrectes:**
    - Si el nom d'usuari o la contrasenya no són correctes, el sistema mostra un missatge d'error indicant que les credencials són invàlides.

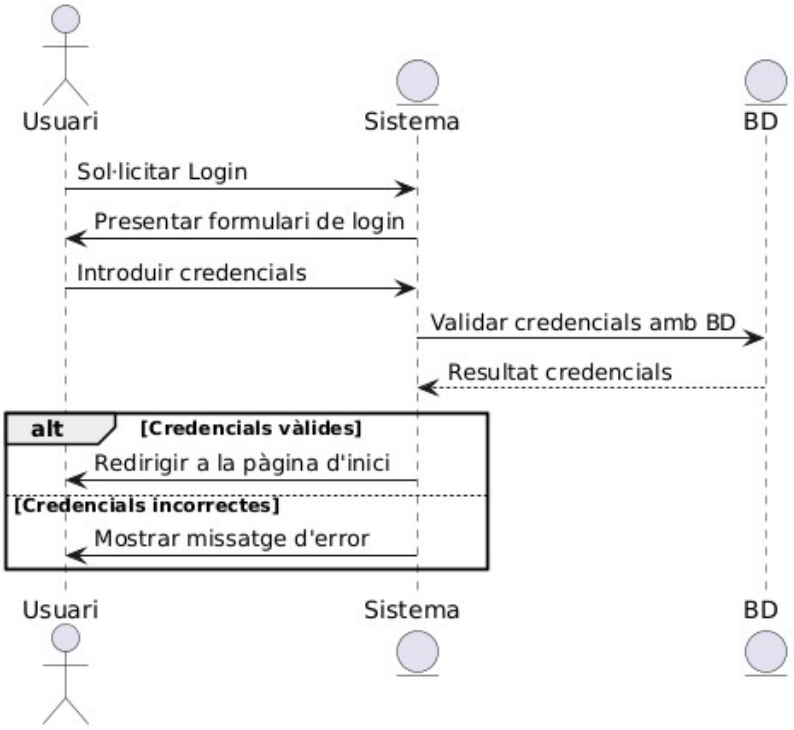

Figura 4: Cas d'ús de login

4.3.2 02. Registrar-se

Resum de la funcionalitat: Permetre als nous usuaris crear un compte al sistema, omplint un formulari de registre.

Paràmetres d'entrada:

- **Nom:** Nom complet de l'usuari.
- **Contrasenya:** Contrasenya escollida per l'usuari.
- **Confirmació de contrasenya:** Repetició de la contrasenya per a verificació.

Paràmetres de sortida: Cap, però es redirigeix a login automàticament.

Actors:

- **Usuari:** Persona que desitja crear un compte al sistema.

Precondicions:

- L'usuari no ha de tenir un compte existent en el sistema.

Postcondicions:

- Un nou compte d'usuari és creat.

Procés normal principal:

1. L'usuari accedeix a la pàgina de registre del sistema.
2. El sistema presenta un formulari amb els camps necessaris: nom, contrasenya i confirmació de contrasenya.
3. L'usuari omple el formulari amb les seves dades i les introdueix al sistema.

4. L'usuari envia el formulari.
5. El sistema valida les dades introduïdes, assegurant-se que la contrasenya i la confirmació coincideixen.
6. Si la validació és correcta, el sistema crea el nou compte.
7. El sistema activa el compte i redirigeix al usuari a Login.

Alternatives de procés i excepcions:

- **Contrasenya i confirmació no coincideixen:**
  - El sistema mostra un missatge d'error i sol·licita que les contrasenyes siguin reintroduïdes.
- **Usuari ja en ús:**
  - El sistema informa a l'usuari que el nom d'usuari ja està registrat i ofereix opció d'iniciar sessió.

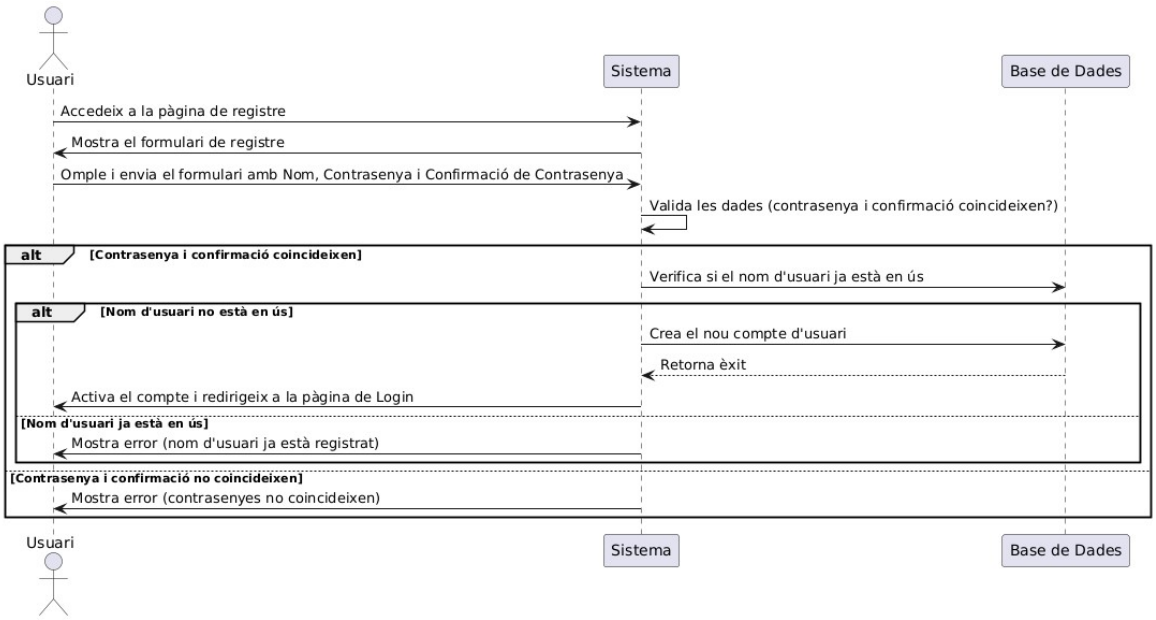

Figura 5: Cas d'ús de registrar

4.3.3 03. Filtrar productes

Resum de la funcionalitat: Permetre als usuaris filtrar els productes disponibles en l'ecommerce per nom mitjançant un camp de cerca o per tipus utilitzant la barra de navegació (navbar).

Paràmetres d'entrada:

- **Nom del producte:** Cadena de text introduïda al camp de cerca (search).
- **Tipus de producte:** Categoria seleccionada des de la barra de navegació (navbar).

Paràmetres de sortida: Llista de productes que coincideixen amb el criteri de filtratge (nom o tipus).

Actors:

- **Usuari:** Persona que desitja cercar o filtrar productes dins l'ecommerce.

Precondicions:

- El sistema ha de tenir productes disponibles en el catàleg.

Postcondicions:

- Es mostren a l'usuari els productes que compleixen amb els criteris de cerca o filtratge seleccionats.

Procés normal principal:

1. L'usuari accedeix a la pàgina de productes.
2. El sistema presenta la llista de productes disponibles amb opcions per cercar o filtrar.
3. L'usuari introdueix un nom al camp de cerca o selecciona un tipus de producte des de la barra de navegació.
4. El sistema envia la sol·licitud de filtratge a la base de dades.
5. La base de dades retorna els productes que coincideixen amb el nom o tipus seleccionat.
6. El sistema mostra la llista de productes filtrats a l'usuari.

Alternatives de procés i excepcions:

- **Cap producte coincideix amb la cerca:**
    - El sistema mostra un missatge indicant que no s'han trobat productes que coincideixin amb els criteris de filtratge.
- **Cap tipus de producte seleccionat:**
    - El sistema mostra tots els productes disponibles si no es filtra per cap tipus específic.

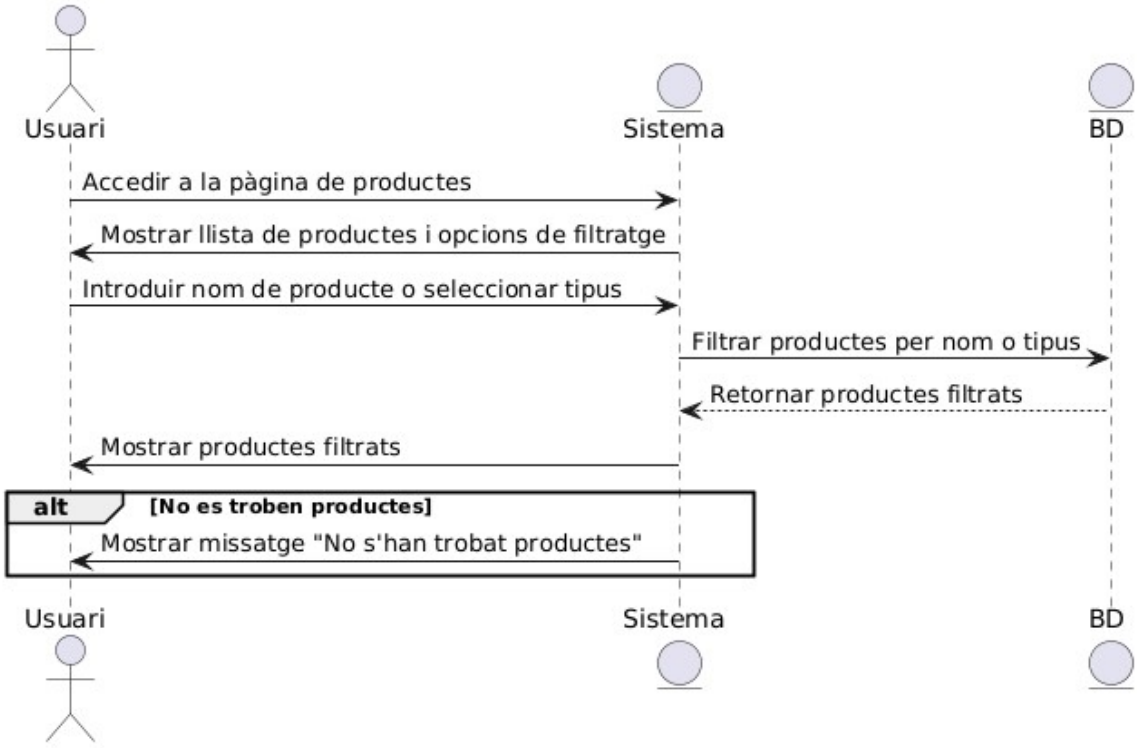

Figura 6: Cas d'ús de filtratge de productes

4.3.4 04. Afegir productes al carret

Resum de la funcionalitat: Permetre als usuaris afegir productes al seu carret de compra, amb una validació prèvia de l'autenticació i la disponibilitat de l'stock. Tot i que en la meva aplicació actual no he implementat la gestió de l'stock, en aquest cas d'ús he decidit incloure'n la validació, ja que tinc previst afegir aquesta funcionalitat en futures actualitzacions. El sistema assegura que el carret està correctament associat amb l'usuari que ha iniciat sessió i que només es poden afegir productes disponibles en l'inventari. Si el producte té stock suficient, es procedeix a afegir-lo al carret de compra.

Paràmetres d'entrada:

- **ID del producte:** Identificador únic del producte que es desitja afegir al carret.
- **Quantitat:** Quantitat del producte que es vol afegir.

Paràmetres de sortida: Missatge de confirmació que el producte ha estat afegit al carret o un error en cas que l'usuari no estigui autenticat.

Actors:

- **Usuari:** Persona que vol afegir productes al seu carret de compra.

Precondicions:

- L'usuari ha d'estar autenticat al sistema.
- El sistema ha de tenir productes disponibles en el catàleg.

Postcondicions:

- El producte seleccionat s'afegeix al carret de compra associat a l'usuari loguejat.

Procés normal principal:

1. L'usuari selecciona un producte a la pàgina de productes i fa clic a "Afegir al carret".
2. El sistema aplica el middleware per verificar si l'usuari està loguejat.
3. **Si l'usuari no està autenticat:**
    - El middleware retorna una resposta 402 Unauthorized i el procés es deté.
4. **Si l'usuari està autenticat:**
    - El sistema obté les credencials de l'usuari per identificar el carret de compra.
    - El sistema envia una sol·licitud a la base de dades per afegir el producte seleccionat al carret de compra de l'usuari.
5. La base de dades actualitza el carret de compra amb el producte afegit.
6. El sistema retorna un missatge de confirmació a l'usuari indicant que el producte ha estat afegit correctament al carret.

Alternatives de procés i excepcions:

- **Usuari no autenticat:**
    - El middleware bloqueja la sol·licitud i retorna una resposta 402 Unauthorized.
- **Producte no disponible:**
    - El sistema mostra un missatge indicant que el producte no està disponible en stock.
- **Error en la base de dades:**
    - El sistema retorna un missatge d'error i no s'afegeix el producte al carret.

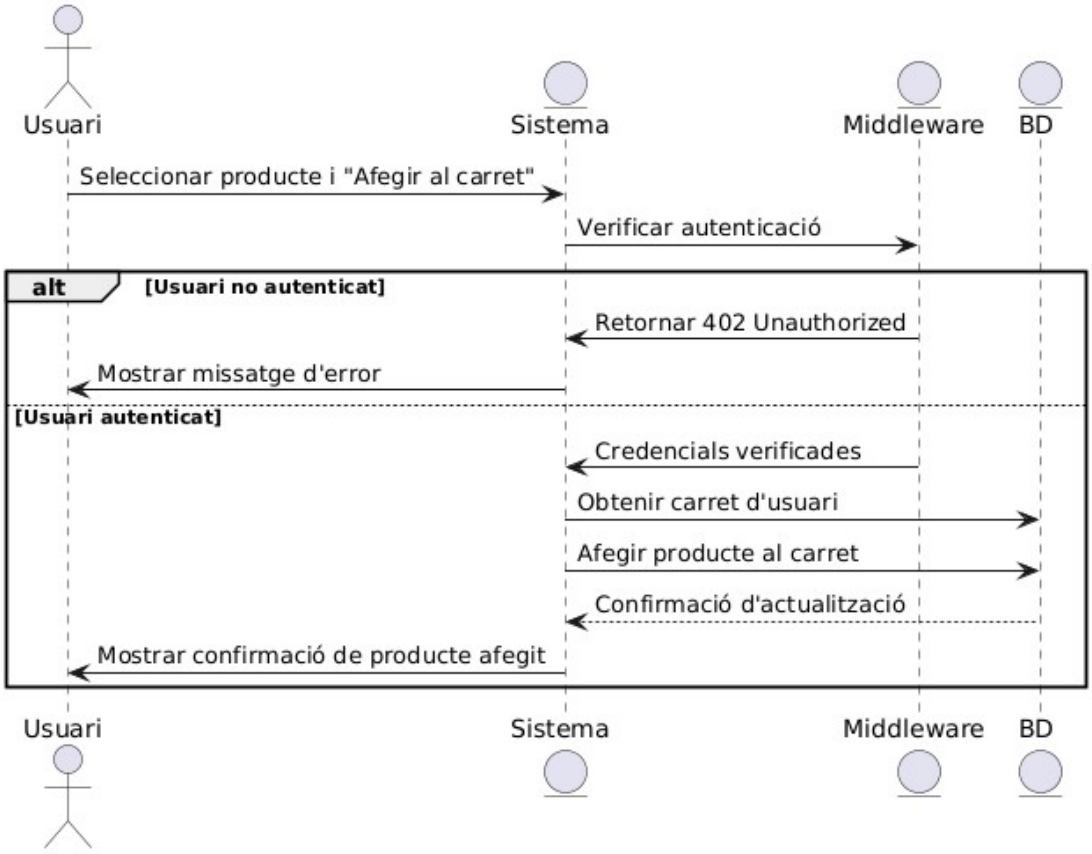

Figura 7: Cas d'ús de afegir producte al carret

4.3.5 05. Modificar quantitat de producte del carret

Resum de la funcionalitat: Permetre als usuaris modificar la quantitat d'un producte ja existent en el seu carret de compra, amb validació d'autenticació i associació del carret amb l'usuari que està loguejat.

Paràmetres d'entrada:

- **ID del producte:** Identificador únic del producte que es desitja modificar.
- **Nova quantitat:** Nova quantitat del producte que es vol establir en el carret.

Paràmetres de sortida: Missatge de confirmació que la quantitat del producte ha estat modificada correctament o un error en cas que l'usuari no estigui autenticat.

Actors:

- **Usuari:** Persona que vol modificar la quantitat de productes en el seu carret de compra.

Precondicions:

- L'usuari ha d'estar autenticat al sistema.
- El carret de compra ha de contenir el producte que es desitja modificar.

Postcondicions:

- La quantitat del producte seleccionat en el carret de compra és modificada segons la nova quantitat introduïda per l'usuari.

Procés normal principal:

1. L'usuari accedeix al seu carret de compra i selecciona el producte del qual vol modificar la quantitat.
2. L'usuari introdueix la nova quantitat desitjada.
3. El sistema aplica el middleware per verificar si l'usuari està loguejat.
4. **Si l'usuari no està autenticat:**
    - El middleware retorna una resposta 402 Unauthorized i el procés es deté.
5. **Si l'usuari està autenticat:**
    - El sistema obté les credencials de l'usuari per identificar el carret de compra.
    - El sistema envia una sol·licitud a la base de dades per actualitzar la quantitat del producte seleccionat en el carret de compra.
6. La base de dades actualitza el carret de compra amb la nova quantitat del producte.
7. El sistema retorna un missatge de confirmació a l'usuari indicant que la quantitat del producte ha estat modificada correctament.

Alternatives de procés i excepcions:

- **Usuari no autenticat:**
    - El middleware bloqueja la sol·licitud i retorna una resposta 402 Unauthorized.
- **Producte no existeix en el carret:**
    - El sistema mostra un missatge indicant que el producte no està present en el carret de compra.
- **Error en la base de dades:**
    - El sistema retorna un missatge d'error i no s'actualitza la quantitat del producte en el carret.

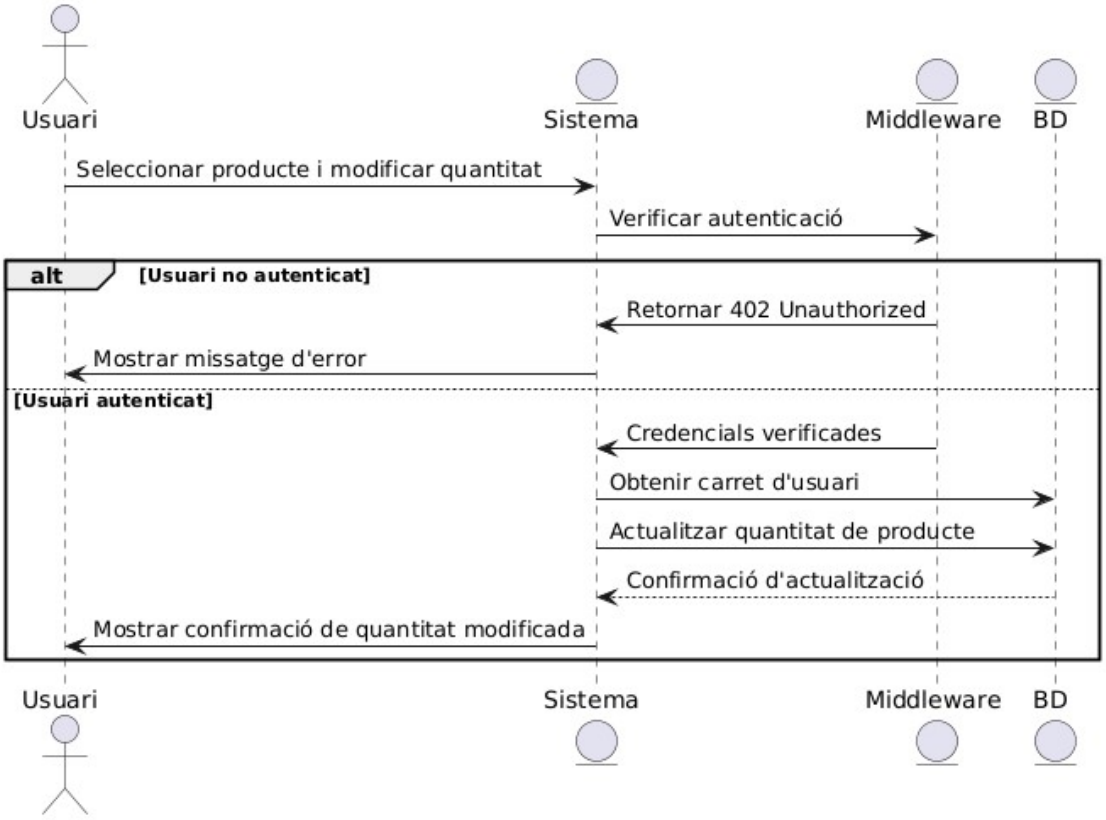

Figura 8: Cas d'ús de modificar quantitat producte al carret

4.3.6 06. Eliminar producte del carret

Resum de la funcionalitat: Permetre als usuaris eliminar un producte del seu carret de compra, amb validació d'autenticació i associació del carret amb l'usuari que està loguejat.

Paràmetres d'entrada:

- **ID del producte:** Identificador únic del producte que es desitja eliminar del carret.

Paràmetres de sortida: Missatge de confirmació que el producte ha estat eliminat correctament o un error en cas que l'usuari no estigui autenticat.

Actors:

- **Usuari:** Persona que vol eliminar un producte del seu carret de compra.

Precondicions:

- L'usuari ha d'estar autenticat al sistema.
- El carret de compra ha de contenir el producte que es desitja eliminar.

Postcondicions:

- El producte seleccionat és eliminat del carret de compra associat a l'usuari loguejat.

Procés normal principal:

1. L'usuari accedeix al seu carret de compra i selecciona el producte que vol eliminar.
2. L'usuari fa clic a l'opció d'eliminar producte.

3. El sistema aplica el middleware per verificar si l'usuari està loguejat.
4. **Si l'usuari no està autenticat:**
    - El middleware retorna una resposta 402 Unauthorized i el procés es deté.
5. **Si l'usuari està autenticat:**
    - El sistema obté les credencials de l'usuari per identificar el carret de compra.
    - El sistema envia una sol·licitud a la base de dades per eliminar el producte seleccionat del carret de compra.
6. La base de dades actualitza el carret de compra eliminant el producte.
7. El sistema retorna un missatge de confirmació a l'usuari indicant que el producte ha estat eliminat correctament.

Alternatives de procés i excepcions:

- **Usuari no autenticat:**
    - El middleware bloqueja la sol·licitud i retorna una resposta 402 Unauthorized.
- **Producte no existeix en el carret:**
    - El sistema mostra un missatge indicant que el producte no està present en el carret de compra.
- **Error en la base de dades:**
    - El sistema retorna un missatge d'error i no s'elimina el producte del carret.

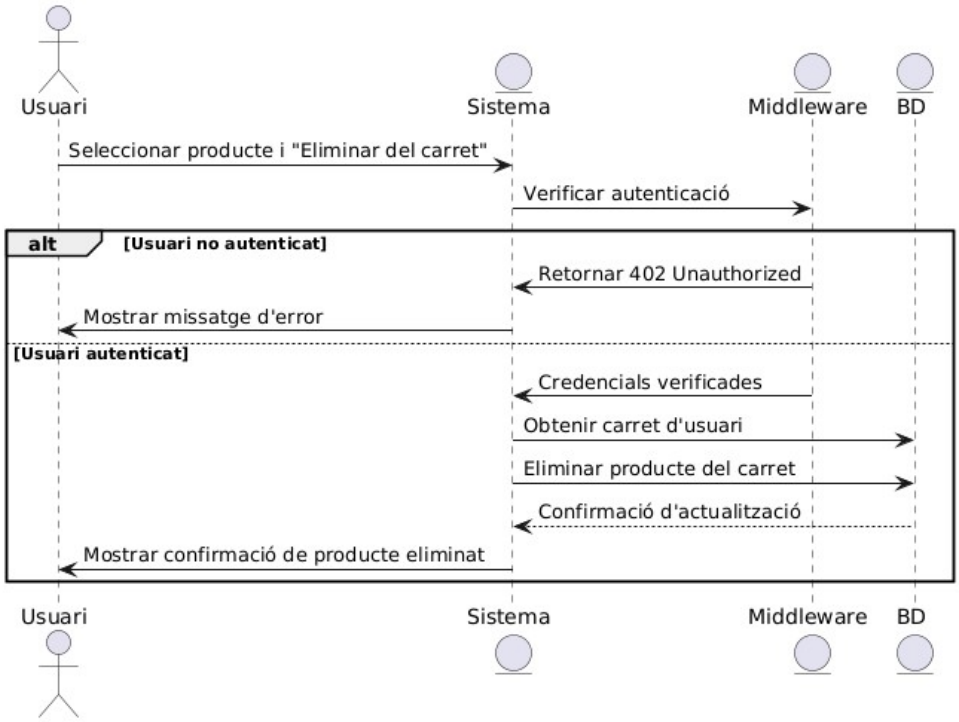

Figura 9: Cas d'ús de borrar producte del carret

4.3.7 07. Procedir al pagament (Checkout)

Resum de la funcionalitat: Permetre als usuaris procedir al pagament dels productes en el seu carret de compra, amb validació d'autenticació, verificació del contingut del carret, processament del pagament i generació d'una factura (invoice). En aquest cas d'ús he afegit un pas addicional que encara no tinc integrat en l'aplicació final. En concret, he inclòs un sistema de pagament per defecte (Stripe[18]) que preveig implementar en futures actualitzacions.

Paràmetres d'entrada:

- **Detalls de pagament:** Informació necessària per processar el pagament (targeta de crèdit, PayPal, etc.).
- **Direcció d'enviament:** Adreça a la qual s'enviaran els productes comprats.

Paràmetres de sortida:

- **Factura (Invoice):** Document que resumeix la transacció realitzada, incloent els productes comprats, el preu total i la informació de facturació.

Actors:

- **Usuari:** Persona que vol completar la compra dels productes en el seu carret de compra.

Precondicions:

- L'usuari ha d'estar autenticat al sistema.
- El carret de compra ha de contenir almenys un producte.
- Els detalls de pagament han de ser vàlids.

Postcondicions:

- El pagament es processa correctament i els productes es marquen com a comprats.
- Es genera una factura que es retorna a l'usuari.

Procés normal principal:

1. L'usuari accedeix al seu carret de compra i revisa els productes.
2. L'usuari fa clic a "Procedir al pagament".
3. El sistema aplica el middleware per verificar si l'usuari està loguejat.
4. **Si l'usuari no està autenticat:**
    - El middleware retorna una resposta 402 Unauthorized i el procés es deté.
5. **Si l'usuari està autenticat:**
    - El sistema obté les credencials de l'usuari per identificar el carret de compra.

[18] Stripe: Plataforma que ofereix serveis de pagament.

- El sistema verifica el contingut del carret i valida la informació de pagament.
- El sistema envia una sol·licitud a la base de dades per processar el pagament i actualitzar l'estat del carret.

6. La base de dades actualitza el carret, processa la transacció i marca els productes com a comprats.
7. El sistema genera una factura (invoice) que inclou els detalls de la transacció.
8. El sistema retorna la factura a l'usuari com a confirmació de la compra.

Alternatives de procés i excepcions:

- **Usuari no autenticat:**
  - El middleware bloqueja la sol·licitud i retorna una resposta 402 Unauthorized.
- **Error en el pagament:**
  - El sistema mostra un missatge indicant que hi ha hagut un problema amb el pagament i no es completa la compra.
- **Productes no disponibles:**
  - Si algun dels productes ja no està disponible, el sistema avisa l'usuari i no processa el pagament fins a resoldre la situació.

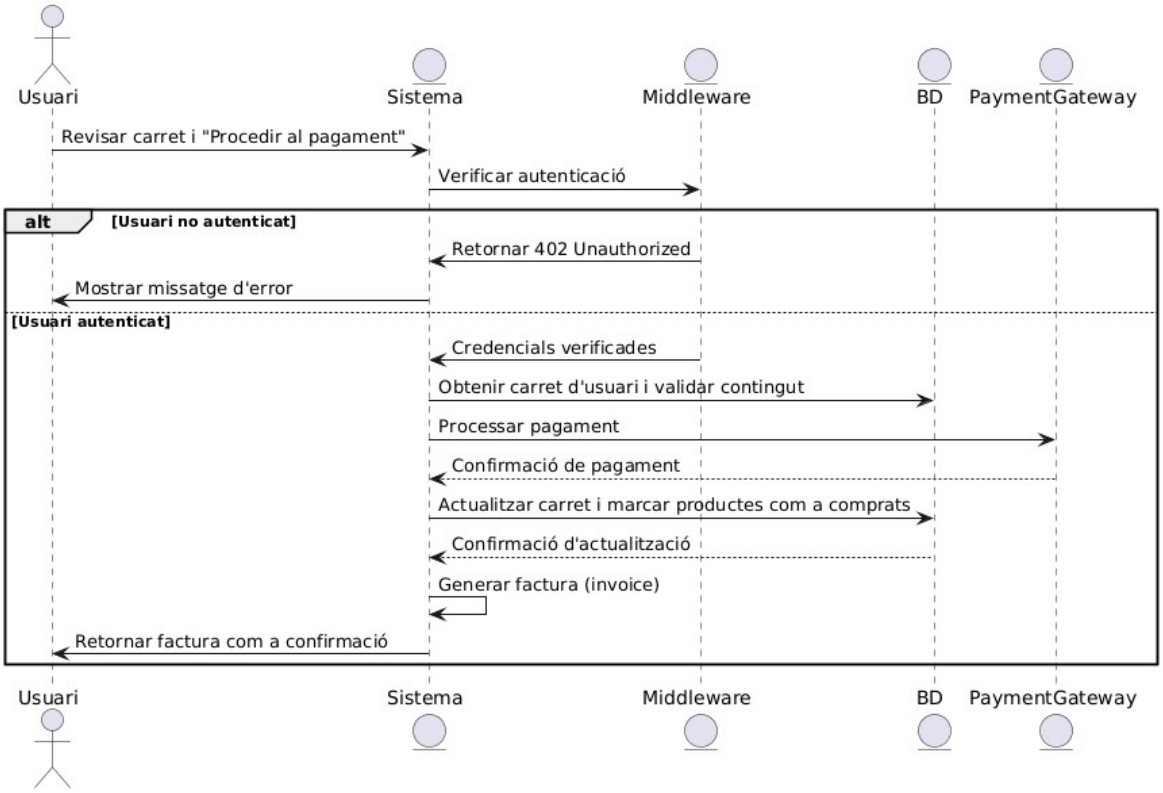

Figura 10: Cas d'ús checkout del carret

4.3.8 08. Tancar sessió (Logout)

Resum de la funcionalitat: Permetre als usuaris tancar la seva sessió de manera segura, la qual cosa implicarà esborrar les cookies d'autenticació i qualsevol altra informació relacionada amb la sessió de l'usuari al navegador.

Paràmetres d'entrada:

- Cap. L'usuari simplement fa clic al botó de "Tancar sessió".

Paràmetres de sortida:

- Cap redirigeix a la home page.

Actors:

- **Usuari:** Persona que desitja tancar la seva sessió en el sistema.

Precondicions:

- L'usuari ha d'estar loguejat en el sistema.

Postcondicions:

- Les cookies[19] d'autenticació esborren del navegador de l'usuari.
- La sessió de l'usuari es finalitza al sistema, prevenint l'accés a àrees restringides fins que es torni a iniciar sessió.

Procés normal principal:

1. L'usuari fa clic al botó de "Tancar sessió" (Logout).
2. El sistema aplica el middleware per verificar si l'usuari està loguejat.
3. **Si l'usuari està loguejat:**
    - El sistema esborra les cookies d'autenticació i qualsevol altra informació relacionada amb la sessió del navegador.
    - El sistema invalida la sessió de l'usuari al servidor, prevenint l'accés posterior sense iniciar sessió.
4. El sistema retorna un missatge de confirmació indicant que la sessió s'ha tancat correctament.
5. L'usuari és redirigit a la home Page o a una pàgina de confirmació de tancament de sessió.

Alternatives de procés i excepcions:

- **Usuari no autenticat:**
    - Si l'usuari intenta tancar sessió sense estar loguejat, el sistema simplement el redirigeix a la home Page.

[19] Cookie: Petits fitxers emmagatzemats al navegador que contenen dades per recordar informació de sessió o preferències de l'usuari.

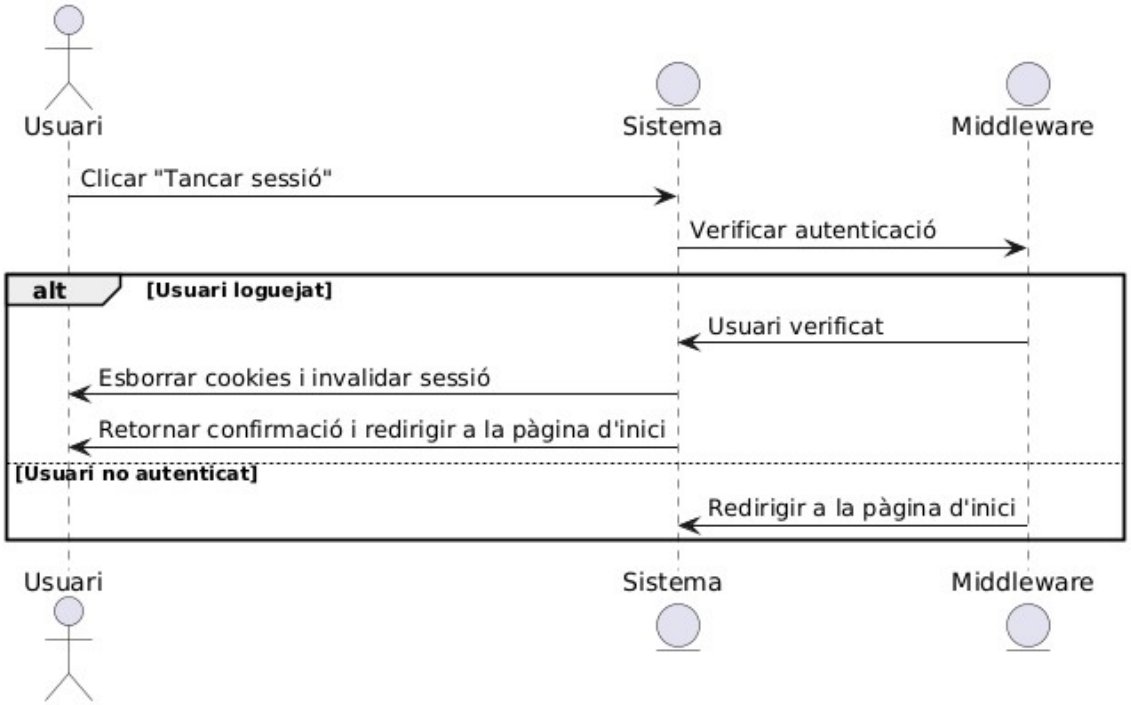

Figura 11: Cas d'ús de logout

4.4 Cas d'Ús General

El cas d'ús general del sistema de comerç electrònic descriu el flux complet d'interacció que un usuari pot tenir amb l'aplicació. Aquest flux comença des del moment en què l'usuari accedeix a l'aplicació i continua fins que l'usuari completa una transacció o tanca la sessió. A diferència dels casos d'ús individuals descrits anteriorment, aquest cas d'ús general cobreix la seqüència completa de passos i les dependències entre ells.

4.4.1 Flux General de l'Usuari

1. **Accés a l'Aplicació**:
    - L'usuari pot començar la seva interacció accedint a la home Page, si encara no esta loguejat pot iniciar sessió, registrar-se si no en té, o simplement navegar pels productes disponibles mitjançant l'opció de filtrat.
2. **Autenticació**:
    - Si l'usuari decideix iniciar sessió, ha d'introduir les seves credencials. Aquest pas és crucial, ja que només els usuaris autenticats poden accedir a funcions més avançades, com afegir productes al carret, modificar la quantitat de productes o procedir al pagament. Alternativament, l'usuari pot optar per registrar-se si no té un compte previ.
    -

3. **Navegació i Selecció de Productes**:
    - L'usuari pot navegar pel catàleg de productes, utilitzant les opcions de filtratge a la home Page per trobar els articles d'interès. Aquest pas és essencial per personalitzar la vista del catàleg i facilitar la selecció dels productes que l'usuari desitja comprar.
4. **Gestió del Carret de Compra**:
    - L'usuari pot afegir productes,al carret, un cop autenticat. Una acció que és fonamental per avançar cap al procés de compra. Després d'afegir un producte, l'usuari també té l'opció de modificar la quantitat d'articles al carret o eliminar-los completament. Aquestes opcions estan disponibles només si hi ha productes prèviament afegits al carret.
5. **Procés de Pagament**:
    - Un cop l'usuari ha finalitzat la selecció de productes, pot procedir al pagament. Aquest és un dels passos finals i requereix que el carret contingui productes. Durant aquest procés, l'usuari revisa la comanda i completa la transacció.
6. **Finalització de la Sessió**:
    - Després de completar la compra, o en qualsevol moment que l'usuari decideixi, pot tancar la sessió de manera segura. Aquest pas garanteix que les credencials de l'usuari no queden actives en el dispositiu, protegint així la seva informació personal (ha de estar autenticat previament).

7. Dependències entre Accions

El sistema està dissenyat de manera que algunes accions depenen d'altres prèvies per garantir un flux lògic i segur dins de l'aplicació. Per exemple, l'usuari no pot modificar o eliminar productes del carret sense haver afegit prèviament algun article. De la mateixa manera, no és possible procedir al pagament sense haver afegit productes al carret. Aquestes dependències asseguren que l'usuari segueixi una seqüència d'accions coherent, evitant errors o incoherències en l'experiència de compra.

Aquest cas d'ús general proporciona una visió integral de com l'usuari interactua amb l'aplicació, cobrint tots els passos des de l'accés fins a la finalització de la sessió. El diagrama que segueix il·lustra aquest flux d'interacció, destacant les relacions de prerequisit i les dependències entre les diferents accions disponibles per a l'usuari.

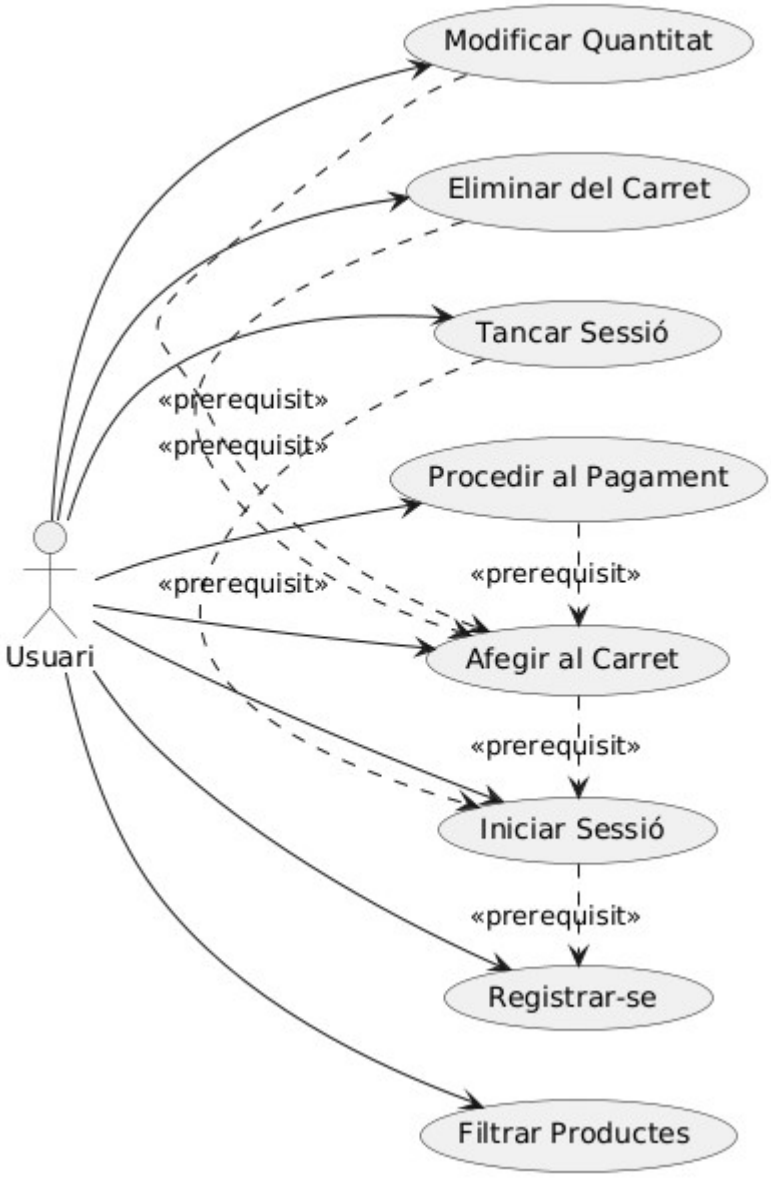

Figura 12: Diagrama de casos d'ús general

5 Disseny

El disseny d'una aplicació és un aspecte fonamental en el món del desenvolupament de software. No és suficient amb tenir un sistema que funcioni i es pugui fer servir; hem de garantir que el nostre sistema sigui robust i escalable. Per això, aspectes com un correcte disseny de la base de dades o la comunicació entre client i servidor són fonamentals en aquest projecte. En aquest apartat, explicaré l'arquitectura general de l'aplicació.

5.1 Arquitectura General de l'Aplicació

L'aplicació segueix un model d'arquitectura basat en **HTML-Over-the-Wire Dynamic Applications (HDA)**, que ofereix una combinació equilibrada d'eficiència i dinamicitat. En aquest model, el servidor és el responsable de generar el contingut HTML, mentre que el client pot actualitzar parts específiques de la pàgina de manera dinàmica, sense necessitat de recarregar-la completament, gràcies a l'ús de **HTMX**. A diferència del model **Server-Side Rendering (SSR)** clàssic, on cada interacció requereix una recàrrega completa de la pàgina, HDA permet que només les seccions necessàries siguin actualitzades, millorant l'eficiència i oferint una experiència d'usuari més fluida i interactiva.

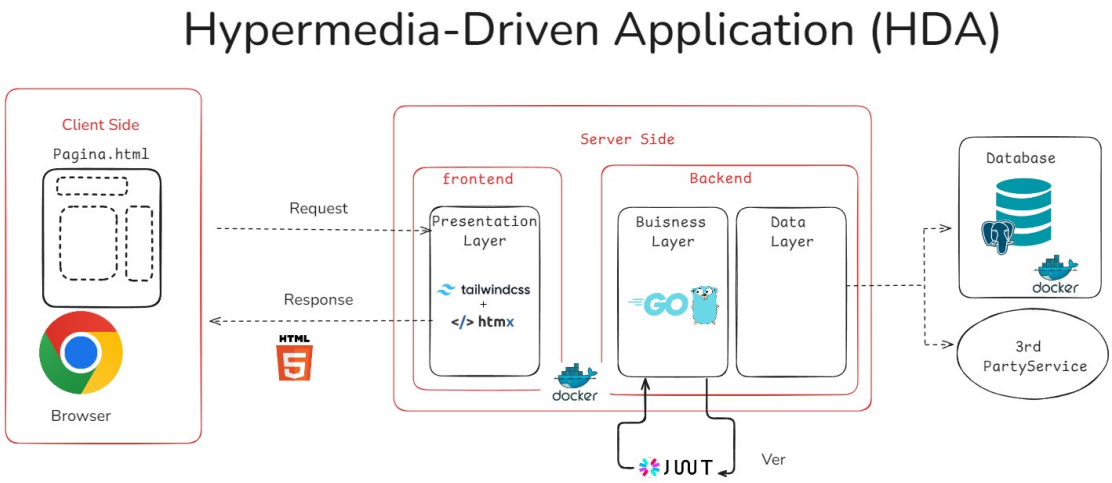

Figura 13: Arquitectura aplicació

5.1.1 Components de l'Arquitectura

5.1.1.1 Client Side (Frontend)

- **Client**: El frontend de l'aplicació s'executa en el navegador web del client, que és responsable de mostrar la interfície d'usuari generada pel servidor. El client envia sol·licituds al servidor i rep fragments d'HTML que actualitzen parts específiques de la pàgina, gràcies a la integració de HTMX, com ve explico en la nota del diagrama.
- **HTMX**: Permet que el frontend sigui dinàmic, gestionant les sol·licituds AJAX per a actualitzar seccions específiques de la pàgina sense necessitat de recarregar-la

completament. Això redueix els temps de càrrega i fa que la interfície sigui més responsiva, millorant l'experiència de l'usuari.

- **Tailwind CSS**: Tailwind CSS s'utilitza per estilitzar la interfície d'usuari de manera coherent i efectiva, seguint un enfocament utilitari que facilita la creació de dissenys nets i responsius.

5.1.1.2 Server Side (Backend)

- **Servidor**: El backend de l'aplicació està desenvolupat en **Golang**, ideal per a la gestió de sol·licituds concurrents i de gran volum. El servidor és responsable de processar les sol·licituds del client, aplicar la lògica de negoci i interactuar amb la base de dades, generant el contingut HTML necessari per a l'aplicació.
- **Lògica de Negoci**: Aquesta capa gestiona totes les operacions essencials, com ara l'autenticació dels usuaris, el maneig de les transaccions i la generació de contingut dinàmic. Les operacions segueixen un model RESTful, en el qual cada sol·licitud HTTP és gestionada de manera individual per endpoints específics.
- **Autenticació i Seguretat**: L'autenticació es gestiona mitjançant **JSON Web Tokens (JWT)**. Després que un usuari s'autentica amb èxit, el servidor genera un token JWT que el client envia amb cada sol·licitud posterior que requereix autenticació. Aquest procés assegura que només els usuaris verificats poden accedir a recursos protegits dins de l'aplicació.

5.1.1.3 Database

- **Base de Dades (PostgreSQL)**: La base de dades està gestionada per **PostgreSQL** i s'encarrega d'emmagatzemar de manera segura totes les dades de l'aplicació, incloent-hi informació d'usuaris, productes, comandes, etc. El servidor interactua amb la base de dades per a realitzar operacions CRUD (Crear, Llegir, Actualitzar, Esborrar) segons sigui necessari.
- **Integració amb Docker**: Tant el servidor com la base de dades estan containeritzats utilitzant **Docker**, la qual cosa permet un desplegament fàcil i replicable de l'aplicació en diferents entorns. Aquesta configuració també facilita l'escalabilitat i el manteniment del sistema.

Nota: A sota de Database tenim "3rd Party Service", es refereix a serveis externs que no són part del sistema principal de l'aplicació, però amb els quals aquesta pot interactuar. Aquests serveis poden incloure passarel·les de pagament, sistemes d'autenticació de tercers, APIs de terceres parts per obtenir dades externes, entre altres. Aquests serveis són accedits pel backend de l'aplicació per complementar les funcionalitats de l'e-commerce, proporcionant serveis especialitzats que serien costosos o complexos de desenvolupar i mantenir internament..

5.1.1.4 Flux de Dades

- **Sol·licitud i Resposta**: El navegador (Client Side) envia sol·licituds HTTP al servidor (Server Side), que les processa, interactua amb la base de dades si cal, i retorna una resposta al client. Aquesta resposta pot consistir en una pàgina HTML completa o en fragments d'HTML que HTMX utilitza per actualitzar parts de la pàgina existent, proporcionant així una experiència d'usuari dinàmica i interactiva.

- **Autenticació amb JWT**: Quan un usuari autenticat envia una sol·licitud que requereix accés a dades protegides, el servidor valida el token JWT adjunt a la sol·licitud abans de processar-la. Si el token és vàlid, el servidor processa la sol·licitud i envia la resposta corresponent. En cas contrari, l'accés és denegat, i l'usuari és redirigit a la pàgina d'inici de sessió.

5.1.2 Arquitectura RESTful

El backend de l'aplicació exposa una API RESTful que gestiona les operacions CRUD (Crear, Llegir, Actualitzar, Esborrar) a través de sol·licituds HTTP. Aquesta API[20] és consumida pel frontend, que envia sol·licituds per obtenir dades o enviar informació de manera dinàmica, especialment gràcies a l'ús d'HTMX per a l'actualització parcial de la pàgina. Això permet que l'aplicació mantingui una experiència d'usuari fluida i responsiva, evitant la necessitat de recarregar la pàgina completa en cada interacció.

5.1.2.1 Endpoints Principals de l'API

1. **Autenticació i Gestió d'Usuaris:**
    - **POST /login:** Aquest endpoint permet als usuaris autenticar-se en l'aplicació. Utilitza el middleware LogAuthMiddleware per a redirigir.
    - **POST /register:** Aquest endpoint permet registrar nous usuaris. També està protegit per LogAuthMiddleware, assegurant que redirigeixi en cas d'estar registrat o loguejat.
    - **GET /logout:** Aquest endpoint finalitza la sessió de l'usuari, eliminant el token JWT actiu i redirigint l'usuari fora de les àrees protegides.
2. **Gestió de Productes:**
    - **GET /products:** Aquest endpoint retorna una llista de productes disponibles. Els usuaris poden veure tots els productes sense necessitat d'estar autenticats.
    - **GET /products/search:** Permet cercar productes segons criteris especificats pels usuaris, com ara el nom del producte o la categoria.
    - **GET /product:** Mostra els detalls d'un producte concret, incloent informació com el nom, la descripció i el preu.
    - **GET /products/filter:** Aquest endpoint filtra els productes segons certs paràmetres definits, com la categoria, el rang de preus, etc.

[20] API : Application Programming Interface

3. **Gestió del Carret de Compra (Protegida per Autenticació):**
    - **GET /cart:** Retorna el contingut del carret de compra de l'usuari autenticat. Aquest endpoint està protegit per AuthMiddleware, assegurant que només l'usuari autenticat pugui veure i gestionar el seu carret.
    - **POST /cart/add:** Permet afegir un producte al carret de l'usuari. Aquest endpoint està protegit per AuthMiddleware.
    - **POST /cart/remove:** Permet eliminar un producte del carret de compra. Igual que els altres endpoints del carret, està protegit per AuthMiddleware.
    - **POST /cart/checkout:** Processa la finalització de la compra, generant una factura i buidant el carret de l'usuari. Està protegit per AuthMiddleware.
    - **POST /cart/addHome:** Afegeix un producte al carret des de la pàgina principal (Home). Aquest endpoint està protegit per AuthMiddleware.
    - **GET /invoices:** Retorna totes les factures associades amb l'usuari autenticat. Està protegit per AuthMiddleware.
4. **Altres Rutes API:**
    - **GET /api/cart-count:** Aquest endpoint retorna el recompte total de productes en el carret de l'usuari autenticat. Protegit per AuthMiddleware, assegura que només l'usuari autenticat pugui accedir a aquesta informació.

5.1.3 Comunicació a través de JSON Web Tokens (JWT)

Per a l'autenticació i l'autorització dels usuaris en l'aplicació, s'utilitzen **JSON Web Tokens (JWT)**. Aquest sistema permet que, un cop l'usuari s'ha autenticat correctament, el servidor generi un token JWT que s'envia al client. Aquest token conté informació codificada sobre l'usuari i les seves autoritzacions, i s'envia amb cada sol·licitud que requereix autenticació.

5.1.3.1 Funcionament

El següent diagrama il·lustra el procés complet de generació, enviament i validació d'un token JWT en l'aplicació:

1. **Autenticació de l'Usuari**:
    - L'usuari envia les seves credencials (nom d'usuari i contrasenya) al servidor mitjançant una sol·licitud POST /login.
    - El servidor valida les credencials amb la base de dades. Si són correctes, el servidor genera un token JWT.
2. **Enviament del JWT al Client**:
    - El token JWT generat pel servidor es retorna al client en la resposta HTTP, habitualment emmagatzemat en una cookie segura o en local storage.
3. **Emmagatzematge i Ús del JWT**:
    - El client emmagatzema el token JWT i el reutilitza per a futures sol·licituds que requereixin autenticació.

- Cada vegada que el client fa una sol·licitud a una ruta protegida, envia el token JWT en la capçalera de la sol·licitud HTTP (normalment en la capçalera Authorization: Bearer <token>).

4. **Validació del JWT pel Servidor**:
   - El servidor valida el token JWT rebut amb cada sol·licitud. Aquesta validació implica verificar la signatura del token i assegurar-se que no ha expirat.
   - Si el token és vàlid, el servidor processa la sol·licitud i retorna la resposta desitjada al client.
   - Si el token no és vàlid o ha expirat, el servidor retorna un error d'autorització (401 Unauthorized), i el client és redirigit a la pàgina d'inici de sessió per obtenir un nou token.
5. **Finalització de la Sessió**:
   - Quan l'usuari decideix tancar la sessió, el client simplement elimina el token JWT. Això assegura que futures sol·licituds no estiguin autenticades fins que l'usuari torni a iniciar sessió i obtingui un nou token.

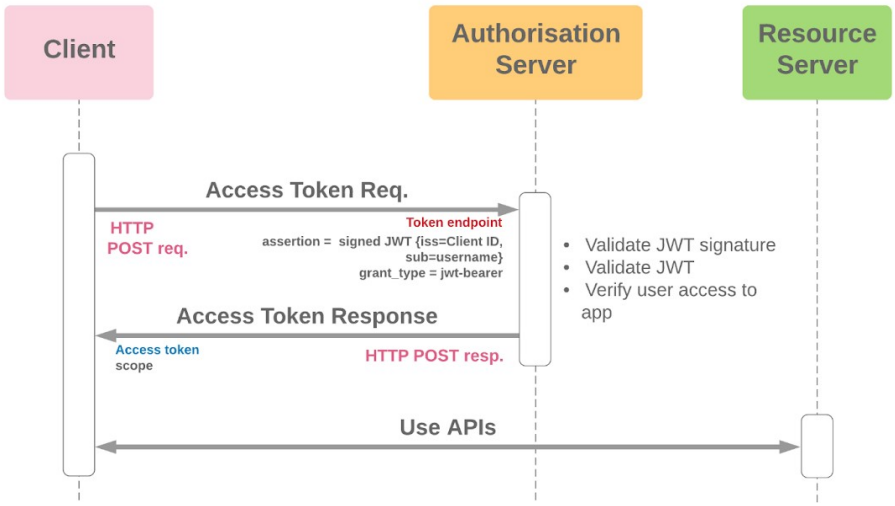

Figura 14: Cicle vida JWT. Font: https://cloudsundial.com/salesforce-identity/jwt-bearer

5.1.4 Seguretat i Middlewares

Per assegurar la integritat i seguretat de l'aplicació, s'han implementat diverses mesures de protecció que inclouen l'ús de middlewares per a la verificació de la validesa de les sol·licituds, l'escapat de dades a través de plantilles (templating) i l'ús de consultes SQL parametritzades. Aquestes mesures treballen conjuntament per defensar l'aplicació contra

vulnerabilitats comunes que poden ser explotades per atacants per comprometre la seguretat d'una aplicació web.

A continuació, es descriuen alguns dels atacs més comuns i les tècniques generals que utilitzem per prevenir-los:

5.1.4.1 Cross-Site Scripting (XSS)

XSS és un tipus d'atac que permet a un atacant injectar codi maliciós (habitualment en JavaScript) dins d'una pàgina web que altres usuaris poden veure. Aquest codi maliciós pot ser utilitzat per robar informació sensible com les cookies de sessió o per redirigir l'usuari a llocs web maliciosos.

Prevenció:

- **Escapat de Dades d'Entrada amb Plantilles**: En l'aplicació, s'utilitzen sistemes de templating que automàticament escapen qualsevol codi potencialment maliciós abans de ser inserit en les pàgines HTML, evitant així la seva execució al navegador dels usuaris.
- **HTTP-Only Cookies**: Les cookies que contenen tokens JWT estan configurades com a "HTTP-Only", la qual cosa significa que no són accessibles des de JavaScript, prevenint així que un atacant pugui accedir a aquestes cookies mitjançant un script XSS.

5.1.4.2 Cross-Site Request Forgery (CSRF)

CSRF és un atac en el qual un atacant enganya a un usuari autenticat perquè faci una sol·licitud maliciosa a un servidor en què l'usuari està autenticat, utilitzant la seva sessió activa per a executar accions no autoritzades.

Prevenció:

- **Tokens CSRF**: Per a protegir l'aplicació contra CSRF, generem i validem tokens CSRF únics per a cada sessió d'usuari. Aquest token s'inclou en les sol·licituds que poden modificar dades (com ara formularis POST), i el servidor verifica que el token enviat pel client coincideixi amb el que s'espera, assegurant que la sol·licitud és legítima.

5.1.4.3 SQL Injection

SQL Injection és un atac que permet a un atacant introduir codi SQL maliciós en una consulta que el servidor executa a la base de dades. Això podria permetre a l'atacant veure, modificar o esborrar dades en la base de dades.

Prevenció:

- **Consultes Parametritzades**: Per a prevenir SQL Injection, l'aplicació utilitza consultes SQL parametritzades en lloc de construir consultes SQL dinàmiques amb dades d'entrada proporcionades per l'usuari. Aquest enfocament assegura que les dades d'entrada són tractades com a literals dins de la consulta, evitant l'execució de codi SQL maliciós.

En aquest subapartat, he explicat de manera general com l'e-commerce es protegeix d'aquests atacs i quins són els perills que representen aquests atacs. A més, cal destacar que en l'apartat d'implementació es proporcionarà una explicació més detallada de com s'han

implementat aquestes mesures de seguretat i els mecanismes específics que s'han utilitzat per garantir la protecció contra aquestes vulnerabilitats.

5.1.5 Futures Implementacions

De cara a futures ampliacions del projecte, es podrien considerar les següents millores:

1. **Integració amb una passarel·la de pagaments**: Es podria implementar una passarel·la de pagaments com **Stripe** per gestionar transaccions de manera segura i eficient. Aquesta integració es podria realitzar mitjançant la seva API o mitjançant l'ús de webhooks per respondre a esdeveniments de pagament en temps real.
2. **Millora del sistema de caching**: La integració de **Redis** podria millorar significativament el sistema de caching de l'aplicació, optimitzant el rendiment en la gestió de sessions, l'emmagatzematge de dades temporals, i la reducció de la càrrega en la base de dades.
3. **Desplegament i Escalabilitat**: Per garantir la seguretat i eficiència en un entorn de producció real, es podria considerar desplegar l'aplicació en contenidors **EC2**[21] a Amazon Web Services (AWS), la qual cosa permetria una escalabilitat flexible segons la demanda. A més, es podria utilitzar un **reverse proxy** (com **Nginx** o **HAProxy**) per gestionar millor el tràfic, especialment en casos d'augments significatius del volum de visites.
4. **Microserveis**: Una altra ampliació possible seria la reestructuració de l'aplicació en microserveis. Aquest enfocament permet una major modularitat, escalabilitat i mantenibilitat, a més de facilitar el desplegament independent de components del sistema.
5. **Implementació de CI/CD**: La implementació d'un pipeline de **CI/CD (Continuous Integration/Continuous Deployment)** amb eines com **Jenkins**, **GitLab CI**, o **GitHub Actions** automatitza els processos de prova i desplegament, millorant l'eficiència i reduint els errors humans en les actualitzacions de l'aplicació.

5.2 Model Relacional

La decisió d'optar per un model relacional en comptes d'un model NoSQL va ser clau durant el disseny del sistema. Després de considerar els pros i contres de cadascun, vaig arribar a la conclusió que el model relacional oferia millor rendiment i avantatges a l'hora d'executar consultes complexes. El disseny inclou moltes taules i algunes relacions, i la majoria de les consultes no involucren només una entitat sinó diverses relacionades entre si. A més, el principi ACID[22] i la capacitat d'executar transaccions són crucials per mantenir la integritat de les dades.

[21] EC2: Amazon Elastic Compute Cloud

[22] ACID: Atomicity, Consistency, Isolation, and Durability

A continuació, es mostra una imatge del model relacional que representa les taules principals de l'aplicació, dissenyada amb PostgreSQL. La Figura 15 mostra la relació entre les diferents taules:

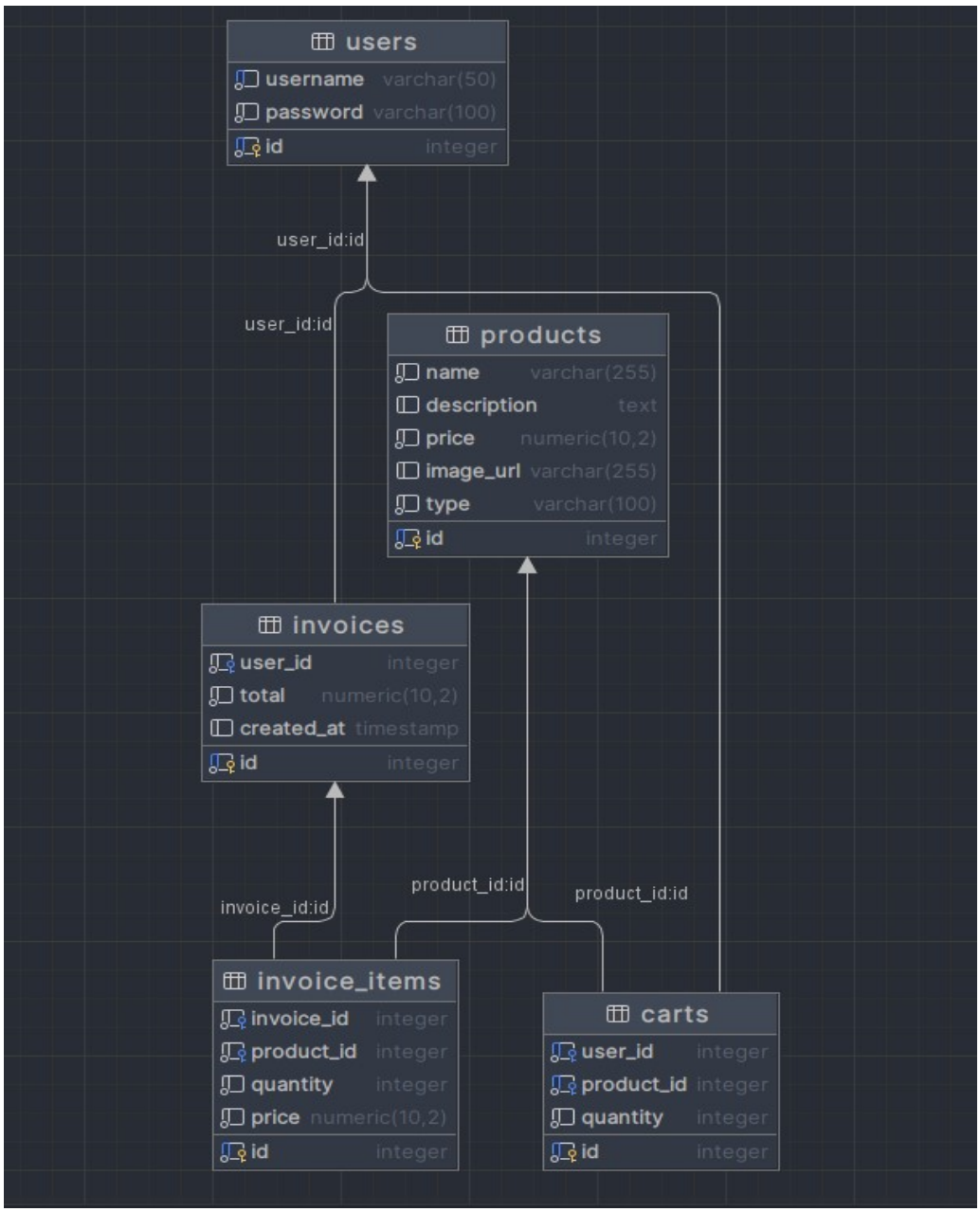

Figura 15: Taules i relacions de l'aplicació

5.2.1 Taula Users

Aquesta taula conté la informació dels usuaris registrats a l'aplicació. A continuació, es detallen alguns dels camps més destacables:

- **id**: Identificador únic per a cada usuari. Es genera automàticament (serial).
- **username**: Nom d'usuari, que actua com a identificador únic dins del sistema.
- **password**: Contrasenya de l'usuari, emmagatzemada de manera segura mitjançant hashing.

Aquestes columnes són fonamentals per gestionar l'autenticació i autorització dels usuaris dins de l'aplicació. He optat per simplificar els camps de la taula users per mantenir un disseny minimalista i eficient, enfocant-me només en la informació essencial per a l'autenticació. Tot i que podria haver afegit camps addicionals, com ara correus electrònics o números de telèfon, he preferit centrar-me en la seguretat i la facilitat de gestió de les dades dels usuaris. Aquest enfocament facilita la gestió de la privacitat i redueix la superfície d'atac potencial de l'aplicació.

5.2.2 Taula Products

Aqui s'emmagatzema la informació dels productes disponibles a la botiga. Els camps més importants són:

- **id**: Identificador únic per a cada producte.
- **name**: Nom del producte.
- **description**: Descripció del producte.
- **price**: Preu del producte.
- **type**: Categoria o tipus de producte.

Aquesta informació és clau per gestionar l'inventari i presentar els productes als usuaris. Per simplificar l'aplicació i centrar-me en l'objectiu principal del treball, que és treballar amb HTMX, he decidit no afegir funcionalitats com la gestió de stock o la possibilitat de tenir diferents variants de productes, com ara diversos colors o mides. Sóc conscient que aquestes funcionalitats són habituals en una aplicació de comerç electrònic completa, però he preferit evitar perdre massa temps en aquestes funcionalitats.

5.2.3 Taula Carts

La taula carts s'encarrega de gestionar els articles que els usuaris han afegit al seu carret de compra, però que encara no han comprat. Els camps més importants inclouen:

- **user_id**: Identifica l'usuari propietari del carret.
- **product_id**: Referencia al producte afegit al carret.
- **quantity**: Indica la quantitat de cada producte en el carret.

He decidit persistir-la per oferir una millor experiència d'usuari, ja que els articles afegits al carret no es perdran si l'usuari tanca la sessió o abandona la pàgina. Això permet als usuaris reprendre la seva compra en qualsevol moment sense haver de tornar a afegir els productes al carret, la qual cosa millora la comoditat i usabilitat de l'aplicació.

5.2.4 Taula Invoices

La taula invoices registra les comandes finalitzades pels usuaris. Els camps destacats són:

- **id**: Identificador únic per a cada factura.
- **user_id**: Identifica l'usuari que ha realitzat la comanda.
- **total**: Import total de la comanda.

- **created_at**: Registra la data i hora de la compra.

Aquesta informació és essencial per la gestió de les transaccions de comerç electrònic dins de l'aplicació.

5.2.5 Taula Invoice_Items

La taula **invoice_items** s'encarrega de desglossar els articles inclosos en cada comanda finalitzada pels usuaris. Aquesta taula emmagatzema la informació detallada sobre els productes que han estat comprats i s'associa directament amb les factures generades (taula invoices). Els camps més importants són:

- **invoice_id**: Clau forana que enllaça cada article amb la seva factura corresponent. Aquesta relació permet identificar tots els productes associats a una comanda específica.
- **product_id**: Clau forana que referencia el producte que ha estat comprat. Això permet obtenir tots els detalls del producte des de la taula products.
- **quantity**: Indica la quantitat de cada producte comprat en la factura.
- **price**: Registra el preu del producte en el moment de la compra. Això assegura que, fins i tot si el preu del producte canvia posteriorment, la factura reflecteixi l'import correcte en el moment de la transacció.

Aquesta taula és essencial per gestionar el detall de cada comanda realitzada pels usuaris. Gràcies a **invoice_items**, és possible mantenir un registre precís dels articles comprats, cosa que facilita les operacions de facturació. A més, permet un historial de compres detallat, que pot ser útil tant per a l'usuari com per a l'administració de l'aplicació en la gestió de l'inventari i l'anàlisi de vendes.

5.2.6 Resum de les Relacions en el Model Relacional

Per acabar amb la temàtica del model de relacions fare un resum de les relacions més importants entre les taules:

- **Relació entre users i carts**: Un usuari pot tenir un o més productes al seu carret de compra. La taula carts conté una clau forana user_id que enllaça cada producte afegit al carret amb l'usuari propietari. Això permet a l'usuari mantenir el seu carret de compra actiu, fins i tot després de tancar la sessió.
- **Relació entre products i carts**: Cada article afegit al carret de compra està associat amb un producte específic. La clau forana product_id a la taula carts enllaça cada article amb el seu corresponent registre a la taula products.
- **Relació entre users i invoices**: Quan un usuari finalitza una compra, es genera una factura (invoice) que està enllaçada amb l'usuari que ha realitzat la transacció. Aquesta relació es gestiona a través de la clau forana user_id a la taula invoices, la qual connecta la factura amb l'usuari.
- **Relació entre invoices i invoice_items**: Cada factura pot contenir múltiples articles. La taula **invoice_items** conté una clau forana invoice_id que enllaça cada article comprat amb la seva factura corresponent. Això permet desglossar el contingut de cada factura i registrar els detalls dels productes comprats.

- **Relació entre products i invoice_items**: Cada article en una factura està associat amb un producte específic. La clau forana product_id a la taula **invoice_items** enllaça cada article comprat amb el seu producte corresponent a la taula products.

5.3 Disseny de la Interfície Gràfica

Com veurem més endavant he optat per un enfocament minimalista, utilitzant **Tailwind CSS,** com ja havia dit el meu coneixement de CSS és pràcticament nul, m'ha sigut molt útil com a eina principal per a els syles. L'objectiu ha estat proporcionar una experiència d'usuari (UX) senzilla, intuïtiva i eficient. He escollit un estil fosc i senzill que manté la interfície neta i fàcil d'usar, assegurant que l'usuari pugui navegar per l'aplicació de manera fluida i sense distraccions.

5.3.1 Disseny de la Pàgina de Login

El Login segueix un enfocament minimalista i funcional, utilitzant **Tailwind CSS** per garantir una estètica coherent i clara. Aquest inclou camps bàsics que totes les pàgines de login tenen, el nom d'usuari i la contrasenya, amb feedback visual per a millorar l'experiència d'usuari. També he afegit un enllaç centrat per a la recuperació de contrasenya. El botó d'enviament, destacat en verd, inclou les **view transicions** activades per a una millor interacció. Aquest disseny busca una experiència senzilla i eficient, centrada en l'accés ràpid a l'aplicació. A més, es mostrarà a sota de la contrasenya un missatge personalitzat indicant el tipus d'error en cas que no sigui vàlid algun dels camps de login.

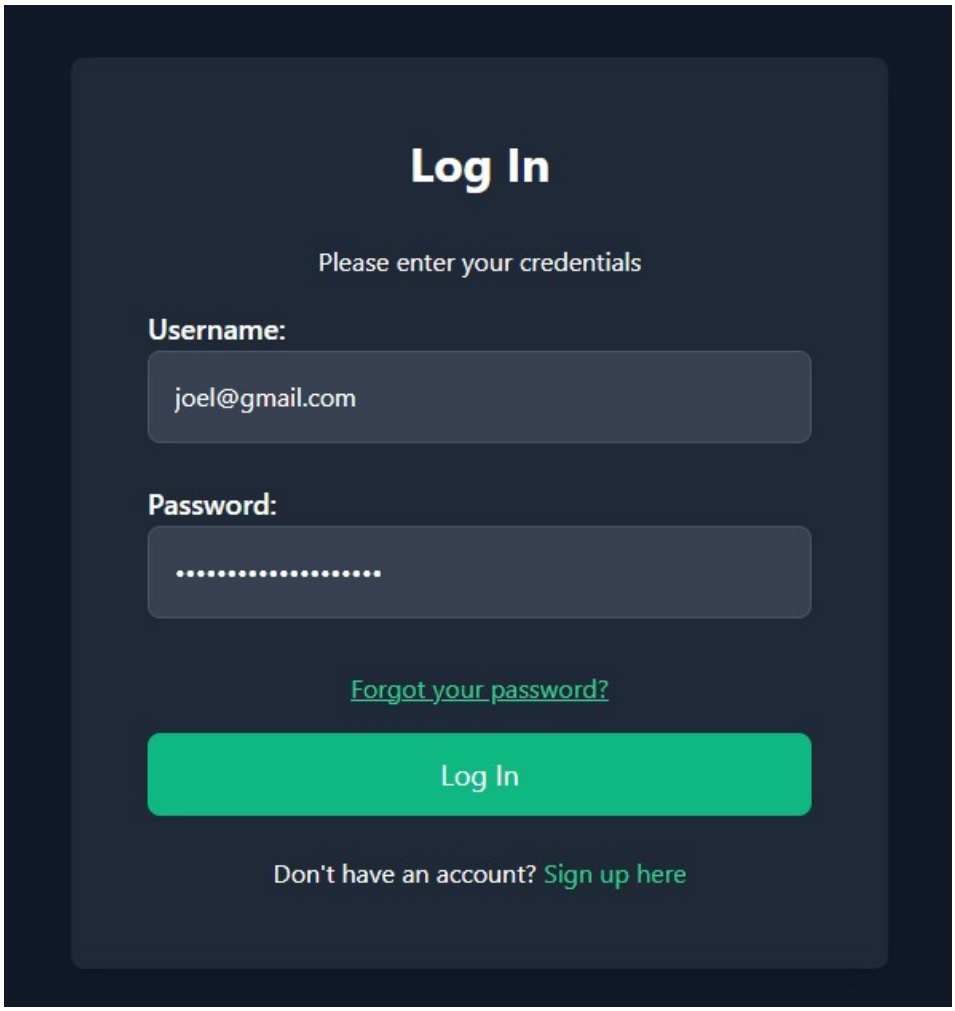

Figura 16: Formulari Login

5.3.2 Disseny de la Pàgina de Registre

Per a la pàgina de registre, he creat un formulari bàsic que recull el nom d'usuari i la contrasenya, mantenint la coherència visual amb la resta de l'aplicació. A més, he reutilitzat el mateix *template* que s'utilitza per mostrar els missatges d'error, assegurant una experiència d'usuari consistent. També es disposa d'un enllaç a la pàgina d'inici de sessió, que permet als usuaris redirigir-se fàcilment si ja tenen un compte. A més, durant el registre, se sol·licitarà una segona confirmació de la contrasenya a través d'un altre *textbox* que apareixerà, per garantir que la contrasenya introduïda és correcta.

Finalment, un cop completat el registre, l'usuari és redirigit automàticament a la pàgina de login sense necessitat de recarregar la pàgina sencera.

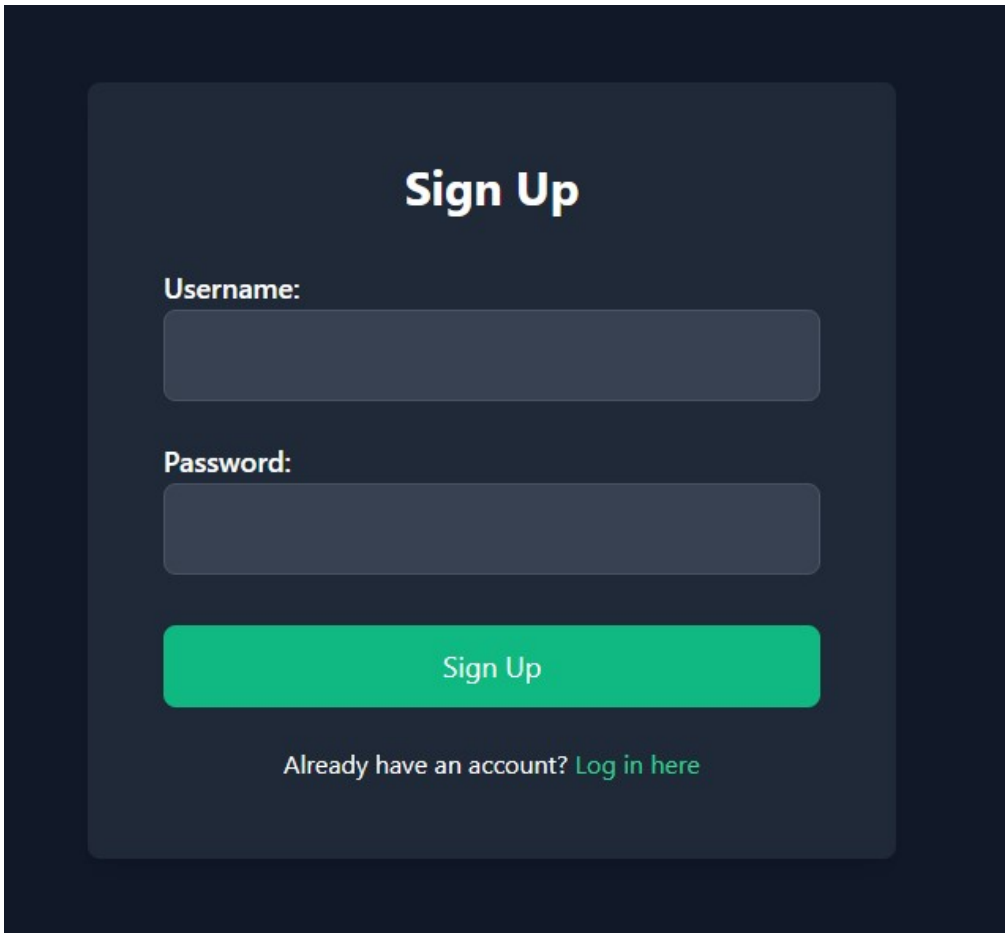

Figura 17: Formulari register

5.3.3 Pàgina Principal

La pàgina principal de l'ecommerce ha estat la més complexa de programar, ja que és la que conté la major quantitat d'elements i interaccions. Aquí es mostren i es filtren tots els productes disponibles, per la qual cosa ha estat necessari un ús intensiu de la plantilla **products**.

Aquesta plantilla itera sobre la llista de productes amb {{range .}} i mostra camps clau com el nom, la imatge i el preu de cada producte. Això permet als usuaris veure i seleccionar fàcilment els productes des de la pàgina principal, facilitant així una experiència d'usuari intuïtiva.

A més, he implementat un carret de la compra desplegable que es pot obrir i tancar amb un petit script per millorar l'experiència d'usuari. El carret també es desplega automàticament quan s'afegeix un producte, gràcies a la integració amb HTMX. He utilitzat l'esdeveniment htmx:afterRequest per obrir el carret després que l'usuari afegeixi un producte, assegurant que l'experiència de compra sigui fluida i eficient sense tenir que renderitzar la pàgina sencera. Si no estem loguejats el carret no s'obrirà.

També he configurat diversos icones interactius, com el de CART, que mostra la quantitat de productes al carret en temps real, i el desplegable de ACCOUNT, que s'activa

en passar el ratolí per sobre. Aquest menú desplegable inclou opcions com veure factures, configurar el compte i tancar la sessió. El menú es desplega suaument i es tanca automàticament quan l'usuari deixa d'interactuar-hi, proporcionant una interfície d'usuari neta i funcional.

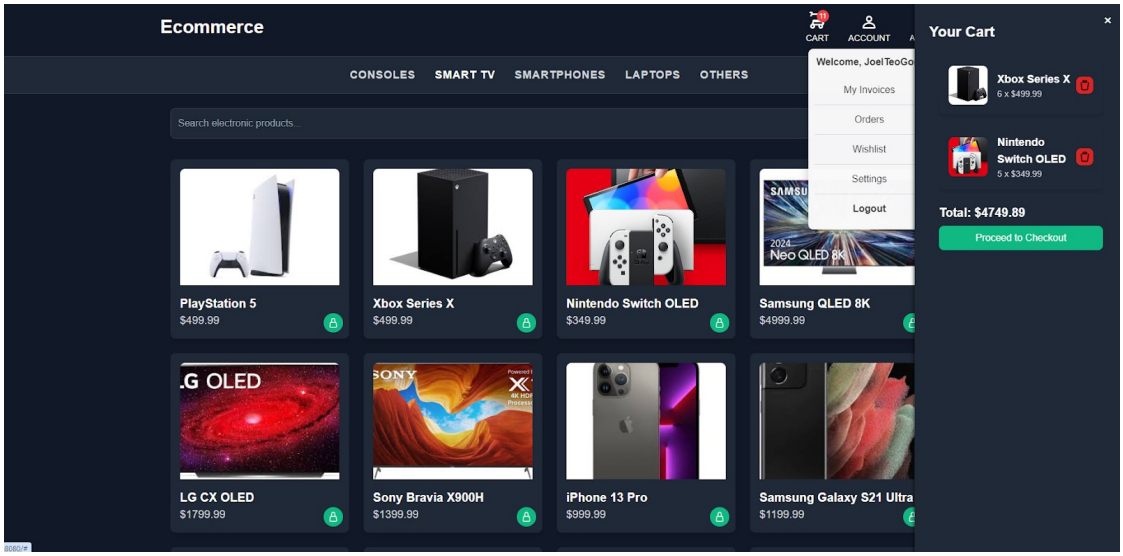

Figura 18: Formulari home page

5.3.4 Detall de Producte

La pàgina de detall de producte ofereix una descripció més completa i visual del producte seleccionat. Aquí, l'usuari pot veure una imatge gran del producte, el seu nom, una descripció detallada, i el preu. A més, he inclòs un botó prominent que permet afegir el producte al carret.

El disseny d'aquesta pàgina és molt simple. Al final, no volia perdre massa temps en una pàgina que simplement mostra la informació del producte. He seguit el mateix estil minimalista i visualment coherent amb la resta de l'aplicació, assegurant una experiència d'usuari intuïtiva i agradable.

Figura 19: Formulari Product detail

5.3.5 Pàgina del Carret de la Compra

La pàgina del carret de la compra és essencial per revisar i gestionar els articles abans de finalitzar la compra. Cada producte es mostra amb la seva imatge, nom, preu i quantitat, i he afegit botons "+" i "-" per ajustar fàcilment la quantitat directament des del carret. Un botó de "paperera" permet eliminar productes, amb una actualització automàtica del total del carret després de cada acció.

Per procedir al pagament, l'usuari pot clicar al botó "Proceed to Checkout", destacat en verd. Aquest disseny segueix un estil minimalista, assegurant una experiència d'usuari simple i eficient.

Tant en aquest carret com en la versió de la home Page al fer checkout es mostra com a resultat la invoice.

Figura 20: Formulari Cart

Figura 21: Formulari Invoice

5.3.6 Pàgina de Factures

No m'he volgut complicar massa amb el disseny de la pàgina de factures, ja que al final només serveix com a excusa per mostrar la llista de factures en una pàgina real. Si fos un projecte més gran, probablement dedicaria més temps a fer-la més atractiva. Tot i això, la pàgina permet als usuaris veure un resum detallat de les seves compres anteriors. Cada factura mostra el número d'identificació, la data de la compra, els productes comprats, la quantitat de cada producte, el preu individual i el total de la compra. Aquest disseny segueix la mateixa línia minimalista i coherent amb la resta de l'aplicació.

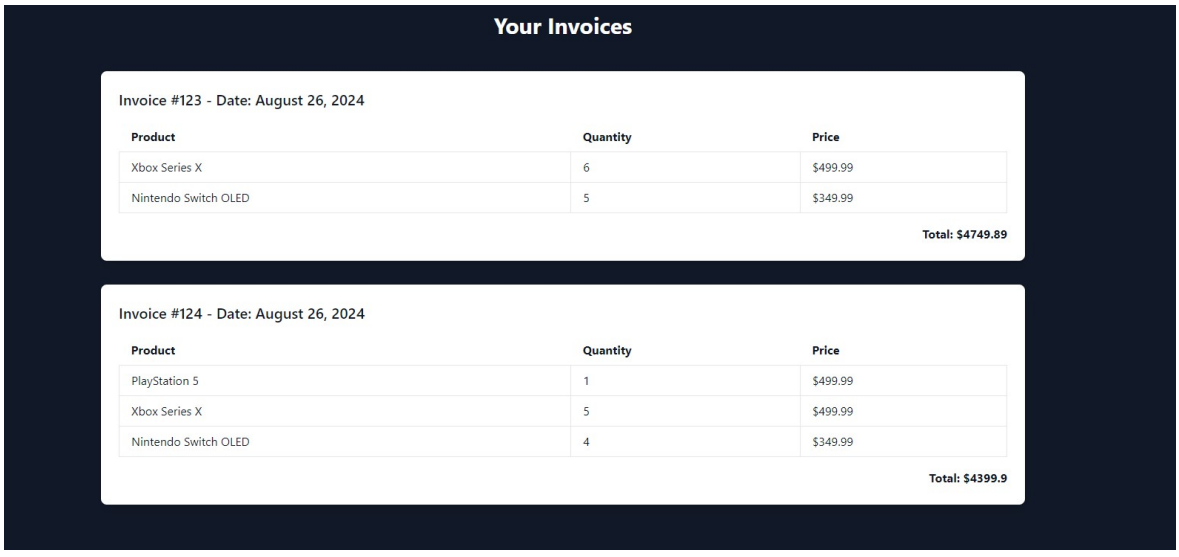

Figura 22: Formulari Invoices

6 Implementació

En aquest apartat, descriuré com s'ha realitzat la implementació tècnica del projecte. Servirà per mostrar el procés de configuració del servidor, la connexió amb la base de dades, i com es gestionen les diferents funcionalitats a través de handlers i models. També s'inclourà un exemple complet del funcionament d'una funcionalitat des del frontend amb HTMX fins al backend, així com les mesures de seguretat implementades.

6.1 Estructura del projecte

He optat per mantenir una jerarquia clara i intuïtiva, amb directoris separats per als templates, handlers, models, middleware i database. Aquesta estructura permet que el projecte sigui fàcilment comprensible per altres desenvolupadors que puguin treballar-hi en el futur. La jerarquia de carpetes es pot veure en la figura adjunta, on es mostren els diferents directoris i fitxers que conformen el projecte.

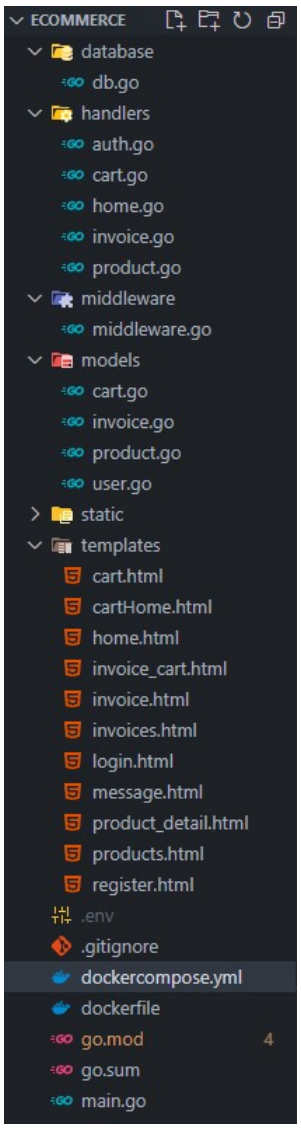

Figura 23: Jerarquia de directoris del ecommerce

6.2 Configuració del Servidor

En aquest subapartat, es descriu com he configurat el servidor utilitzant Go. Es detallarà el procés d'inicialització del servidor HTTP, incloent la configuració de les rutes i handlers que gestionen les peticions dels usuaris. Tota aquesta configuració es fa en main.go.

La configuració del servidor en l'aplicació es fa utilitzant Go. Primer, es connecta a la base de dades amb database.InitDB() i s'assegura que la connexió es tanqui correctament al final.

```
func main() {
    database.InitDB()
    defer database.DB.Close()
```

Seguidament, he configurat un servidor d'arxius estàtics per servir recursos com les imatges dels productes i CSS des del directori ./static.

```
    fileServer := http.FileServer(http.Dir("./static"))
    http.Handle("/static/", http.StripPrefix("/static/", fileServer))
```

Les rutes de l'aplicació es defineixen utilitzant http.HandleFunc, i s'assignen a diferents handlers que gestionen funcionalitats com l'inici de sessió, registre, gestió de productes, i carret de compra. Algunes rutes les he protegit amb uns middleware per garantir que només els usuaris autenticats hi tinguin accés. En l'últim subapartat explicaré com estan implementats els middlewares.

Com es disposa de moltes rutes en ficaré només algunes d'exemple:

1. **Ruta sense autenticació**

   Aquesta ruta és accessible per tots els usuaris i mostra la pàgina principal.
   ```
   http.HandleFunc("/", handlers.HomeHandler())
   ```

2. **Ruta de login amb middleware de redirecció**

   Aquesta ruta utilitza un middleware que redirigeix els usuaris ja autenticats.
   ```
   http.HandleFunc("/login",
   middleware.LogAuthMiddleware(handlers.LoginHandler()))
   ```

3. **Ruta protegida amb autenticació**

   Aquesta ruta requereix que l'usuari estigui autenticat per accedir-hi.
   ```
   http.HandleFunc("/cart",
   middleware.AuthMiddleware(handlers.CartHandler()))
   ```

Finalment, el servidor HTTP s'inicia escoltant en el port 8080, permetent l'accés a l'aplicació des del navegador.

```
    log.Println("Servidor iniciat en :8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
```

```
}
```

6.3 Connexió a la Base de Dades

Aqui detallaré com s'estableix la connexió, com es gestiona la configuració de les credencials, i com s'inicia la connexió des del servidor. En si, la creació de la base de dades ha sigut molt simple i ràpid de verificar.

El codi per a la inicialització de la connexió a la base de dades es troba encapsulat en la funció InitDB(). Tota la configuració que després es carregarà a l'apartat anterior es troba al fitxer de database/db.go i .env.

La separació de les credencials i altres detalls en un fitxer .env ajuda a mantenir la seguretat i facilita la gestió de la configuració en diferents entorns de desenvolupament.

- **Imports Necessaris:**

```
"github.com/joho/godotenv"
_ "github.com/lib/pq"
```

- **github.com/joho/godotenv**: Aquest paquet s'utilitza per carregar les variables d'entorn des d'un fitxer .env. Això permet gestionar de manera segura les credencials i altres configuracions sensibles sense incloure-les directament en el codi font.
- **_ "github.com/lib/pq"**: Aquest import es refereix al controlador de PostgreSQL per a Go. La notació amb el guió baix (_) indica que l'import s'utilitza només per inicialitzar el paquet, que en aquest cas és necessari per poder utilitzar PostgreSQL amb database/sql.
- **Variable Global:**

```
var DB *sql.DB
```

- **DB *sql.DB**: He declarat una variable global DB que és un punter a sql.DB. Aquesta variable representa la connexió a la base de dades i s'utilitza a tota l'aplicació per executar consultes SQL i interactuar amb la base de dades. Com podem comprovar Go juga molt amb els punters que funcionen d'una manera similar a C.

Explicació Resumida del Codi

1. **Càrrega de variables d'entorn**:
    - Amb godotenv.Load(), es carreguen les variables d'entorn definides en el fitxer .env. Aquestes variables inclouen detalls com el nom de la base de dades, l'usuari, la contrasenya, l'amfitrió i el port.
    - Si hi ha algun error en carregar el fitxer .env, el programa es deté i mostra un missatge d'error.
2. **Obtenció de detalls de connexió**:
    - Les variables d'entorn es llegeixen utilitzant os.Getenv() per obtenir els valors necessaris per connectar-se a la base de dades.

3. **Construcció de la cadena de connexió**:
    - Es crea una cadena de connexió (dataSourceName) que conté tots els detalls necessaris per establir la connexió amb PostgreSQL.

```
dataSourceName := fmt.Sprintf("host=%s port=%s user=%s dbname=%s
password=%s sslmode=disable",
        dbHost, dbPort, dbUser, dbName, dbPassword).
```

4. **Inicialització de la connexió a la base de dades**:
    - Es fa servir sql.Open("postgres", dataSourceName) per obrir una connexió a la base de dades amb la cadena de connexió creada anteriorment, aquesta funció retorna o un error o un punter que s'assigna el valor a la base de dades creada.
    - Després, es verifica que la connexió sigui vàlida amb DB.Ping(). Si la connexió falla, el programa es deté i es mostra un missatge d'error.
5. **Missatge de confirmació**:
    - Si la connexió és exitosa, es registra un missatge de confirmació indicant que la connexió amb la base de dades s'ha establert correctament.

6.4 Exemple de Handler

Aquest subapartat descriu un exemple de com es defineixen els handlers en Go. Els handlers són les funcions encarregades de gestionar les peticions HTTP entrants i proporcionar respostes adequades. A continuació es mostra un exemple d'un handler que gestiona la llista de productes disponibles en l'aplicació.

```
func ProductFilterHandler() http.HandlerFunc {
 return func(w http.ResponseWriter, r *http.Request) {
        productType := r.URL.Query().Get("type")
        log.Println("Filtrando productos por tipo:", productType)
        products,   err   :=   models.GetProductsByType(database.DB,
productType)
        if err != nil {
              http.Error(w,"Error al buscar productos",
http.StatusInternalServerError)
              return
        }
        log.Println(products)
        tmpl                                                    :=
template.Must(template.ParseFiles("templates/products.html"))
        tmpl.ExecuteTemplate(w, "products", products)
 }
}
```

Explicació del Codi:

1. **Definició del *Handler***:
    - Aquesta funció ProductFilterHandler() retorna una funció de tipus http.HandlerFunc. Això permet que la funció retornada pugui ser utilitzada directament com a *handler* per gestionar les peticions HTTP dins del fitxer main.go.
2. **Obtenció de Paràmetres de la Petició**:
    - El codi comença recuperant el valor del paràmetre type de la URL, que especifica el tipus de producte pel qual es volen filtrar els resultats. Això es fa amb r.URL.Query().Get("type").
3. **Interacció amb el Model**:
    - Un cop obtingut el tipus de producte, es crida la funció models.GetProductsByType, la qual recupera de la base de dades els productes que coincideixen amb el tipus especificat.
    - Aquesta funció accepta la connexió a la base de dades com a paràmetre (database.DB) juntament amb el tipus de producte. Si hi ha un error en aquesta operació, es retorna un missatge d'error al client amb http.Error, i la funció acaba la seva execució per evitar continuar amb una llista de productes no vàlida.
4. **Renderització de la Resposta**:
    - Si la consulta de productes és exitosa, la llista de productes es passa a la plantilla HTML products.html per ser renderitzada.
    - La plantilla s'encarrega de generar el codi HTML que es retornarà al navegador de l'usuari, mostrant els productes filtrats segons el tipus especificat.
5. **Plantilles HTML**:
    - La funció utilitza el paquet template de Go per carregar i executar la plantilla products.html. Aquesta plantilla està ubicada dins la carpeta templates i conté el disseny de la pàgina on es mostraran els productes filtrats.

Aquest *handler* exemplifica el flux típic de treball en una aplicació web amb Go: des de la recepció d'una petició HTTP, passant per la interacció amb el model de dades per obtenir informació, la gestió d'errors, fins a la renderització d'una resposta que s'envia de nou al client. Els *logs* que he inclòs en el codi m'han servit principalment per a depurar i verificar que el *handler* funciona correctament.

En altres *handlers*, com els relacionats amb el carret de compra o les factures (*invoices*), és necessari obtenir les credencials de l'usuari a través de les *cookies*. Aquestes *cookies* es gestionen automàticament pel navegador i permeten al servidor verificar la identitat de l'usuari per assegurar que només pot accedir a la informació o realitzar accions que li corresponen.

Cookies

Enviament de *Cookies*: Quan un client fa una petició HTTP al servidor, el navegador inclou automàticament totes les *cookies* associades amb el domini. A Go, aquestes *cookies* es poden accedir amb r.Cookie("token"), permetent al servidor utilitzar aquesta informació per a diverses finalitats, com identificar l'usuari o mantenir l'estat de la sessió.

Gestió de *Cookies*: Al servidor, es pot processar la informació continguda en les *cookies* per prendre decisions, com verificar si l'usuari està autenticat o si cal generar una nova *cookie,* com és el cas de login.

Setejat de Noves *Cookies*: Després de processar la petició, el servidor pot establir noves *cookies* creant un objecte http.Cookie amb els detalls necessaris (nom, valor, durada, etc.). Aquesta *cookie* es retorna al client amb http.SetCookie(w, &cookie). El navegador emmagatzema aquesta *cookie* i la inclourà automàticament en futures peticions, mantenint així la continuïtat de la sessió.

Per tant, les *cookies* s'envien i es gestionen automàticament; l'únic del que m'he de preocupar és de fer-ne un ús correcte, assegurant-me que siguin segures, tal com explicaré en l'apartat de seguretat.

6.5 Exemple de Model

En aquest subapartat, no em centraré en excés en els detalls tècnics, però sí que proporcionaré una visió general del funcionament dels models a l'aplicació, utilitzant com a exemple la funció GetCartItems.

La funció GetCartItems s'encarrega d'obtenir tots els articles que un usuari té al seu carret de compra. Aquesta funció rep com a paràmetre la connexió a la base de dades (db *sql.DB) i l'ID de l'usuari (userID int).

```
func GetCartItems(db *sql.DB, userID int) ([]Cart, error) {
    query := `
      SELECT carts.id, carts.user_id, carts.product_id,
carts.quantity, products.name, products.price, products.image_url
        FROM carts
        JOIN products ON carts.product_id = products.id
        WHERE carts.user_id = $1
    `
rows, err := db.Query(query, userID)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var cartItems []Cart
    for rows.Next() {
        var cartItem Cart
```

```
            var product Product
            err := rows.Scan(&cartItem.ID, &cartItem.UserID,
&cartItem.ProductID, &cartItem.Quantity, &product.Name, &product.Price,
&product.ImageURL)
            if err != nil {
                return nil, err
            }
            cartItem.Product = product
            cartItems = append(cartItems, cartItem)
        }
        return cartItems, nil
    }
```

El procediment és el següent:

1. **Consulta SQL:** He resaltat la part de la sentencia SQL que es la més important, aquesta selecciona els detalls dels articles del carret de la taula carts i els enllaça amb la taula products per obtenir la informació addicional del producte, com el nom, preu i URL[23] de la imatge.
2. **Execució de la consulta:** La consulta es realitza mitjançant db.Query, passant l'ID de l'usuari com a paràmetre.
3. **Processament dels resultats:** Es recorren les files resultants (rows.Next()), i es mapegen als camps de les estructures Cart i Product. Després, cada article es va afegint a una llista d'articles del carret (cartItems).
4. **Retorn dels resultats:** Finalment, es retorna la llista d'articles o un error si n'hi ha hagut algun durant el procés.

Aquest mateix esquema s'aplica a la majoria de funcions que interactuen amb la base de dades a l'aplicació: es construeix una consulta SQL, s'executa, es processen els resultats i es retornen les dades corresponents. Així, la gestió de dades i operacions CRUD[24] es realitza de manera consistent i eficient.

6.6 Flux de Funcionament Complet

En aquest subapartat, es presentarà un exemple complet del flux de treball d'una funcionalitat a l'aplicació, il·lustrant com HTMX interactua amb el servidor per processar una sol·licitud.

1. **Enviament de la Sol·licitud amb HTMX**: Quan un usuari interactua amb l'aplicació, per exemple, afegint un producte al carret, HTMX s'encarrega d'enviar una sol·licitud AJAX al servidor sense recarregar tota la pàgina.

[23] URL: Uniform Resource Locator

[24] CRUD: Create, Read, Update, and Delete

Aquesta sol·licitud es fa a través d'una petició HTTP específica al handler corresponent al servidor.

2. **Gestió de la Sol·licitud per part del Handler**: Un cop el servidor rep la sol·licitud, aquesta és gestionada per un handler en Go. El handler és responsable de processar la petició, validant les dades i, si cal, autenticant l'usuari a través de les cookies o altres mecanismes de seguretat. Després, el handler crida a una funció del model per realitzar les operacions necessàries, com per exemple afegir el producte al carret de compra de l'usuari.

3. **Interacció amb la Base de Dades**: El handler delega la interacció amb la base de dades a una funció del model. Aquesta funció s'encarrega de realitzar una consulta SQL o una altra operació rellevant a la base de dades, com inserir, actualitzar o eliminar dades. Per exemple, pot buscar un producte específic, verificar la seva disponibilitat i actualitzar la quantitat al carret de compra.

4. **Retorn de la Resposta al Frontend**: Un cop s'ha completat la consulta o l'operació a la base de dades, la funció del model retorna el resultat al handler. El handler, al seu torn, prepara una resposta HTTP que pot incloure una actualització de l'HTML, un missatge de confirmació, o qualsevol altra informació necessària. Aquesta resposta es retorna al frontend on HTMX s'encarrega de processar-la i actualitzar dinàmicament la interfície d'usuari sense recarregar tota la pàgina.

La majoria dels casos, he utilitzat un format similar al següent per gestionar les interaccions entre el frontend i el backend:

```
hx-get="/products/search"
hx-trigger="input changed delay:500ms"
hx-target="#product-list"
hx-swap="innerHTML transition:true"
```

Aquest enfocament utilitza una petició GET o POST combinada amb un hx-trigger com ara input changed o click. Quan es produeix l'esdeveniment, es substitueix el contingut del innerHTML de l'element objectiu, sovint amb efectes de transició per millorar l'experiència visual. Un altre exemple és el següent codi:

```
        hx-post="/cart/addHome"
        hx-vals='{"id": {{.ID}} }'
        hx-trigger="click"
        hx-target="#cart-items-panel"
        hx-on="htmx:afterRequest: openCart()"
        hx-swap="innerHTML"
        class="absolute bottom-2.5 right-2.5 bg-green-500 text-white p-2
rounded-full hover:bg-green-600 transition">
```

En aquest cas, s'utilitza hx-on, que permet associar una acció JavaScript a un esdeveniment específic. Aquí, després de fer una petició POST per afegir un producte al carret, hx-on="htmx:afterRequest: openCart()" s'encarrega d'obrir automàticament el carret desplegable una vegada que la petició ha estat completada. En el següent apartat, explicaré també com podem usar HTMX per a la compartició d'estats.

6.7 Compartició d'Estats entre Components Web

Un dels majors reptes en el desenvolupament web és la compartició d'estats entre components, especialment en aplicacions complexes on múltiples components han de mantenir-se sincronitzats. En frameworks com React, aquesta tasca es sol gestionar amb biblioteques com Redux, que permeten centralitzar l'estat de l'aplicació. Tot i això, l'ús de Redux pot complicar significativament el codi, introduint una corba d'aprenentatge elevada i problemes addicionals en la gestió d'estats.

Amb HTMX, la compartició d'estats es simplifica molt. Tal com he explicat en la part teòrica, permet generar events de manera molt senzilla utilitzant headers com hx-trigger-after-swap, que es pot activar després d'un intercanvi (swap) de contingut. A més, amb l'atribut hx-trigger, podem fer que una altra petició respongui a aquest event creat. Així, amb només dues línies de codi, podem gestionar la sincronització d'estats entre diferents components de la pàgina de manera fàcil i eficient, evitant la complexitat que sovint es troba en altres solucions com Redux. Aquest enfocament ofereix una manera intuïtiva de mantenir els estats dels components compartits sense necessitar un estat global complicats.

En la meva aplicació, he implementat una funcionalitat per actualitzar dinàmicament el número de productes al carret de compra, que es mostra en un icona a la interfície d'usuari. L'element HTML que representa aquest icona fa ús de HTMX per gestionar l'actualització del comptador de manera automàtica quan es realitzen operacions al carret, com afegir, eliminar o completar una compra (checkout).

```
{{if .IsAuthenticated}}
<a href="/cart" class="relative flex flex-col items-center">
    <svg xmlns="http://www.w3.org/2000/svg" class="h-8 w-8 text-white" fill="none" viewBox="0 0 24 24" stroke="currentColor">
        <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M3 3h2l.4 2M7 13h10l4-8H5.4L4 7M7 13l-1 5h11.6a2 2 0 002-2h-1a2 2 0 00-2-2H7zm-1 5a2 2 0 102 2 2 2 0 00-2-2zm10 0a2 2 0 102 2 2 2z"/>
    </svg>
    <span class="text-sm text-white mt-1">CART</span>
    <span id="cart-count"
        class="absolute top-0 right-0 bg-red-500 text-white text-xs rounded-full h-5 w-5 flex items-center justify-center"
        hx-get="/api/cart-count"
        hx-trigger="load, updateCart from:body"
        hx-swap="innerHTML">
        0
    </span>
</a> {{end}}
```

6.7.1 Funcionament

1. **Icona del Carret**: L'element span amb l'ID cart-count és responsable de mostrar el número de productes al carret. Aquest element fa una petició HTTP GET a l'endpoint /api/cart-count quan es carrega la pàgina o quan es dispara l'esdeveniment updateCart.
2. **Trigger HTMX**: El hx-trigger="load, updateCart from:body" fa que aquest element escolti l'esdeveniment updateCart, que es dispara des de qualsevol part del document (body). També es carregarà al iniciar la pàgina amb l'atribut load.
3. **Actualització del Comptador**: Quan es realitza alguna acció sobre el carret (afegir, eliminar, fer checkout), els handlers corresponents envien una resposta HTTP amb un header especial que conté l'esdeveniment updateCart. Quan l'element cart-count detecta aquest esdeveniment, fa automàticament una petició a l'API per actualitzar el comptador amb el número actual de productes.

Exemple de codi eliminar del carret:

```
func RemoveFromCartHandler() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        userID, err := GetUserIDFromCookie(r)
        if err != nil || userID == 0 {
            http.Error(w, "No se pudo obtener el usuario de la sesión", http.StatusUnauthorized)
            return
        }
        productID, _ := strconv.Atoi(r.FormValue("id"))
        log.Println(productID)
        err = models.RemoveFromCart(database.DB, userID, productID)
        if err != nil {
            http.Error(w, "No se pudo eliminar el producto del carrito", http.StatusInternalServerError)
            return
        }
        w.Header().Set("HX-Trigger-After-Swap", "updateCart")
    }
}
```

6.8 Seguretat

La seguretat és un pilar fonamental en el desenvolupament d'aplicacions web, especialment en el context d'un projecte de comerç electrònic on gestiono dades sensibles, com les credencials dels usuaris i les transaccions financeres. He adoptat diverses mesures per assegurar que les dades dels usuaris es mantinguin segures i que l'aplicació sigui resilient davant possibles atacs. A continuació, detallo les mesures de seguretat implementades, així com les vulnerabilitats potencials i les recomanacions per millorar la seguretat del sistema.

6.8.1 Autenticació amb JWT

He gestionat l'autenticació en aquesta aplicació mitjançant **JSON Web Tokens (JWT)**. Els tokens JWT permeten autenticar els usuaris de manera segura, emmagatzemant el token en una cookie després que l'usuari hagi iniciat sessió correctament. Cada vegada que un usuari intenta accedir a una ruta protegida, un middleware verifica la validesa del token. Si el token és vàlid, l'usuari pot accedir a la ruta; si no ho és, se li denega l'accés.

Aquí mostro com he configurat les cookies per a l'emmagatzematge del token JWT:

```
func setTokenCookie(w http.ResponseWriter, token string) {
    cookie := &http.Cookie{
        Name:     "token",
        Value:    token,
        HttpOnly: true,
        Secure:   os.Getenv("NODE_ENV") == "production",
        SameSite: http.SameSiteStrictMode,
        Path:     "/",
        Expires:  time.Now().Add(1 * time.Hour),
    }
    http.SetCookie(w, cookie)
}
```

He configurat les cookies com a HttpOnly i Secure per impedir que el token sigui accessible des del client mitjançant JavaScript i per assegurar que només es transmeti a través de connexions HTTPS. A més, amb el mode SameSite=Strict, limito l'enviament de cookies a peticions del mateix lloc, prevenint atacs CSRF (Cross-Site Request Forgery). Aquestes cookies es creen cada cop que l'usuari fa login, o s'eliminen cada cop que fa logout ficant el expire a zero.

6.8.2 Prevenció d'Atacs XSS (Cross-Site Scripting)

Una de les grans crítiques o el principal argument en contra d'HTMX és que suposadament promou el risc d'atacs XSS (Cross-Site Scripting). No obstant això, aquesta preocupació no té gaire sentit, ja que, independentment de la tecnologia utilitzada, sigui HTMX o React, mai no ens hauríem de refiar de les dades que ens arriben des de fora del servidor. Per això, és fonamental validar i sanititzar sempre les dades que entren al sistema, assegurant-nos que no contenen contingut maliciós que pugui comprometre la seguretat de l'aplicació.

Per prevenir atacs XSS, he utilitzat la llibreria html/template de Go per a la generació de plantilles HTML. Aquesta llibreria és segura per defecte contra XSS, ja que escapa automàticament els caràcters especials en les variables que s'inserten en el HTML. Això

garanteix que qualsevol entrada de l'usuari que contingui codi maliciós no s'executi en el navegador, evitant així que es produeixin aquests atacs.

Per exemple, a l'hora de mostrar una llista de productes, he utilitzat html/template d'aquesta manera:

```
func ProductListHandler() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        products, err := models.GetAllProducts(database.DB)
        if err != nil {
            http.Error(w, "Error al obtenir productes", http.StatusInternalServerError)
            return
        }
        tmpl := template.Must(template.ParseFiles("templates/products.html"))
        tmpl.ExecuteTemplate(w, "products", products)
    }
}
```

Aquí, template.ParseFiles carrega la plantilla HTML, i tmpl.ExecuteTemplate escapa automàticament qualsevol contingut dinàmic abans de renderitzar-lo, assegurant així que no es pugui injectar codi maliciós.

6.8.3 Prevenció d'Atacs SQL Injection

Totes les consultes SQL en la meva aplicació utilitzen placeholders ($1, $2, etc.) conjuntament amb funcions preparades com db.QueryRow, db.Exec i db.Query. Aquestes funcions asseguren que els valors proporcionats per l'usuari es gestionin de manera segura, evitant que puguin manipular les consultes SQL i provocar un atac d'injecció SQL.

Per exemple, a l'hora de recuperar un usuari per nom d'usuari, he utilitzat la següent consulta preparada:

```
func GetUserByUsername(db *sql.DB, username string) (*User, error) {
    query := "SELECT id, username, password FROM users WHERE username=$1"
    user := &User{}
    err := db.QueryRow(query, username).Scan(&user.ID, &user.Username, &user.Password)
    if err != nil {
        if err == sql.ErrNoRows {
            return nil, nil // Usuari no trobat
        }
        return nil, err
    }
    return user, nil
}
```

Aquesta implementació utilitza placeholders per evitar injeccions SQL, assegurant que qualsevol dada proporcionada per l'usuari es tracti com a valor i no com a part de la consulta SQL.

6.8.4 Mitigació de CSRF (Cross-Site Request Forgery)

Per protegir l'aplicació contra atacs CSRF, he configurat la cookie de token JWT amb l'opció SameSite=Strict. Això impedeix que la cookie s'enviï en peticions creuades entre diferents dominis, assegurant que només les sol·licituds originades des del mateix lloc puguin incloure la cookie d'autenticació. Aquesta configuració, combinada amb l'ús de cookies HttpOnly, redueix significativament el risc d'atacs CSRF.

6.8.5 Middlewares Implementats

He implementat diversos *middlewares* que actuen com a capes de protecció addicionals dins de l'aplicació. En Go, una característica poderosa és la capacitat de passar funcions com a paràmetres a altres funcions. Això permet construir *middlewares* de manera eficient i reutilitzable. Com podem veure en aquestos dos middlewares després de processar la lògica dins del middleware, com pot ser la verificació d'autenticació o la manipulació de capçaleres HTTP, es pot cridar la funció next(w, r) per continuar amb l'execució del *handler* original. Això permet que la petició segueixi el seu curs normal cap al *handler* que s'encarrega de la funcionalitat específica de la ruta.

LogAuthMiddleware: En si, aquest middleware simplement té función de redirigir als usuaris autenticats que intenten accedir a les pàgines de **login** o **registre**. Si un usuari ja ha iniciat sessió, és redirigit automàticament a la pàgina principal.

```
func LogAuthMiddleware(next http.HandlerFunc) http.HandlerFunc {
        return func(w http.ResponseWriter, r *http.Request) {
            cookie, err := r.Cookie("token")
                if err == nil && token.Valid {
                    if   r.URL.Path   ==   "/login"   ||   r.URL.Path   ==
"/register" {
                        http.Redirect(w, r, "/", http.StatusSeeOther)
                        return
                    }
                }
            }
            next.ServeHTTP(w, r)
        }
    }
```

- **AuthMiddleware**: Aquest middleware protegeix les rutes que només han de ser accessibles per usuaris autenticats. Verifica la presència i validesa del token JWT abans de permetre l'accés a aquestes rutes, garantint que només els usuaris autoritzats puguin accedir a dades sensibles.

```
func AuthMiddleware(next http.HandlerFunc) http.HandlerFunc {
```

```
        return func(w http.ResponseWriter, r *http.Request) {
            cookie, err := r.Cookie("token")
            if err != nil || cookie.Value == "" {
                http.Error(w, "Unauthorized", http.StatusUnauthorized)
                return
            }
            token, err := jwt.Parse(cookie.Value, func(token *jwt.Token)
(interface{}, error) {
                return jwtKey, nil
            })
            if err != nil || !token.Valid {
                http.Error(w, "Unauthorized", http.StatusUnauthorized)
                return
            }
            next.ServeHTTP(w, r)
        }
    }
```

6.8.6 Conclusió i Recomanacions

He desenvolupat l'aplicació amb un alt nivell de seguretat, implementant defenses efectives contra atacs comuns com XSS, SQL Injection i CSRF. Les configuracions de cookies, l'ús de consultes SQL preparades i la gestió adequada dels tokens JWT són elements clau que asseguren la protecció de les dades dels usuaris i la integritat del sistema.

Recomanacions per a Millorar la Seguretat:

- **Rate Limiting**: Una de les millores que podria implementar per augmentar encara més la seguretat és la limitació de la velocitat de les sol·licituds (rate limiting). Això ajudaria a prevenir atacs de força bruta (DDOS) en les rutes d'autenticació, limitant el nombre de sol·licituds que un usuari o IP pot fer en un període de temps determinat.
- **Refresh Tokens**: A més, és recomanable l'ús de *refresh tokens* per augmentar la seguretat en la gestió de sessions llargues. Un *refresh token* podria ser utilitzat per renovar el token d'accés cada 24 hores, assegurant que, fins i tot si un token JWT és compromès, l'atacant només tindrà un accés limitat en el temps.

7 Avaluació

La validació de qualsevol projecte és essencial per garantir la qualitat i el correcte funcionament del sistema abans del seu desplegament a producció. Per assegurar-ho, he realitzat una sèrie de proves exhaustives, utilitzant eines i tecnologies que ofereixen una cobertura completa i una execució eficient. Aquest procés de validació és crucial per identificar i corregir possibles errors, assegurant que el sistema compleixi amb els requisits i funcioni de manera òptima.

7.1 Objectius Complerts

Un dels objectius principals del projecte era garantir que les API desenvolupades funcionessin correctament i oferissin els resultats esperats. Per assolir aquest objectiu, he realitzat proves exhaustives.

Mitjançant Postman, he verificat el correcte funcionament de cada endpoint, assegurant-me que les peticions HTTP (GET, POST, PUT, DELETE) es gestionessin adequadament i que les respostes fossin les esperades. A més, he validat que les dades enviades i rebudes compleixen amb els formats esperats i que els errors es gestionen correctament. Aquest procés de prova ha estat crucial per assegurar la fiabilitat i la robustesa de les API, així com per garantir que els components frontend i backend s'integrin de manera efectiva.

Aquestes proves m'han permès identificar i corregir possibles inconsistències o errors abans de la implementació final, contribuint així a l'assoliment dels objectius del projecte.

7.2 Validació dels Requisits No Funcionals

En aquesta secció es descriuen les proves i validacions realitzades per assegurar que l'aplicació compleix amb els requisits no funcionals establerts durant la planificació del projecte. Aquests requisits són essencials per garantir que l'aplicació és segura, eficient, fàcil de mantenir i escalable. A continuació, es presenten els resultats de les validacions realitzades.

7.2.1 Rendiment i Escalabilitat

Temps de Resposta (RNF1): Per avaluar el rendiment de l'aplicació, he realitzat proves utilitzant eines com el navegador per mesurar el temps de càrrega de les pàgines principals. Els resultats indiquen que el temps de resposta mitjà per a les operacions més comunes, com ara iniciar sessió, afegir productes al carret i completar compres, es va mantenir dins dels límits acceptables, amb una latència mitjana de 69ms i un temps de resposta de filtrat de productes de 10.36ms, com demostrare en el següent apartat amb la figura 25. Això demostra que l'ús de **HTMX** i **Golang** com a tecnologies ha permès mantenir una resposta ràpida i eficient.

Suport de Càrrega (RNF2): Gràcies a l'eficiència de Golang, conegut per la seva capacitat de manejar concurrència de manera eficient, l'aplicació ha mostrat un rendiment robust sota càrrega. S'ha verificat que l'aplicació pot gestionar un nombre elevat de sol·licituds simultànies sense degradació significativa del rendiment, confirmant que és escalable i preparada per a escenaris d'ús real amb un nombre creixent d'usuaris.

7.2.2 Seguretat

Encriptació de Dades (RNF3): Totes les dades sensibles, com les contrasenyes dels usuaris, es gestionen mitjançant la llibreria **bcrypt**, assegurant que es mantenen encriptades en la base de dades. S'han realitzat proves de seguretat per confirmar que l'encriptació és resistent a atacs de força bruta, garantint la seguretat de la informació.

Autenticació Segura (RNF4): L'autenticació es gestiona mitjançant **JSON Web Tokens (JWT)**, assegurant que cada sessió d'usuari és segura. He validat que els tokens JWT són generats de manera única per a cada sessió i que expiren correctament, protegint així l'aplicació contra accessos no autoritzats.

Protecció Contra Atacs (RNF5): S'han dut a terme simulacions d'atacs per validar que l'aplicació és resistent a vulnerabilitats comunes, com **SQL Injection**, **Cross-Site Scripting (XSS)** i **Cross-Site Request Forgery (CSRF)**. El middleware desenvolupat protegeix l'aplicació, assegurant que les sol·licituds malicioses no puguin comprometre la seguretat del sistema.

7.2.3 Usabilitat

Compatibilitat de Navegador (RNF6): L'aplicació ha estat provada en diverses plataformes i navegadors, incloent Chrome, Firefox, Safari i Edge, per assegurar que funciona correctament en totes elles. Els resultats confirmen que no hi ha problemes significatius de compatibilitat i que l'experiència d'usuari és consistent en tots els navegadors.

Disseny Responsive (RNF7): El disseny de l'aplicació s'ha implementat utilitzant **Tailwind CSS**, que assegura una experiència d'usuari responsive i adaptable a diferents mides de pantalla. S'han realitzat proves en dispositius de diferents tipus (ordinadors, tablets i smartphones), i els resultats mostren que la interfície d'usuari s'adapta correctament sense comprometre la funcionalitat.

7.2.4 Mantenibilitat

Revisió Constant del Codi (RNF8): Durant tot el desenvolupament del projecte, s'ha seguit un procés rigorós de revisió de codi, utilitzant **Git** com a eina de control de versions. Aquest procés ha permès identificar i corregir errors de manera ràpida, assegurant que el codi es manté net i fàcil de mantenir.

Simplicitat i Claredat en el Codi (RNF9): El codi s'ha mantingut simple i ben estructurat seguint les millors pràctiques de **Golang**. S'ha usat un linter per assegurar que el codi compleix els estàndards de qualitat, garantint que sigui fàcilment mantenible i comprensible per altres desenvolupadors.

7.2.5 Interoperabilitat

Integració amb Sistemes de Pagament (RNF10): Encara que en aquesta fase del projecte no s'ha implementat la integració amb passarel·les de pagament com **Stripe**, l'arquitectura actual està dissenyada per permetre aquesta integració en el futur. S'ha verificat que l'aplicació és compatible amb la implementació d'aquest tipus de sistemes, facilitant futures actualitzacions que requereixin processament de pagaments.

8 Avantatges i Desavantatges de l'ús d'HTMX

8.1 Introducció

Aquest apartat està enfocat a analitzar les avantatges i desavantatges d'HTMX en comparació amb frameworks moderns de JavaScript. Es posarà especial èmfasi en les diferències entre HTMX i el framework més popular actualment que és React. Finalment, es discutirà com aquestes diferències poden impactar el desenvolupament d'aplicacions web.

8.2 Avantatges HTMX

HTMX adopta un enfocament centrat en el servidor amb capacitats avançades d'hipermèdia, mentre que React intenta cobrir tot l'espectre dins del model de components. Cada un d'aquests enfocaments té els seus propis avantatges i inconvenients, i la seva utilitat dependrà del tipus d'aplicació que es vulgui construir.

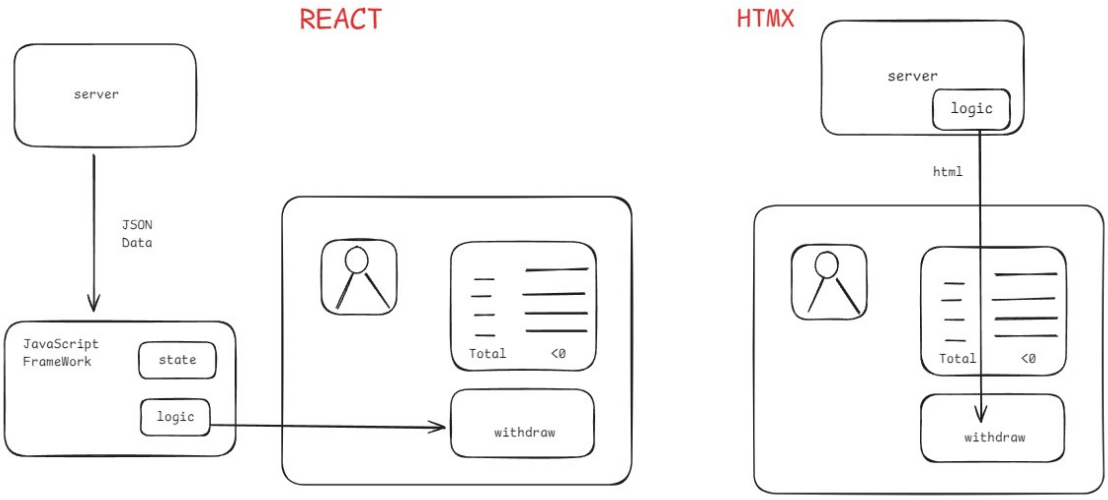

Figura 24: Comparació gràfica d'enfocaments

Enfocament Actual amb Frameworks de JavaScript

1. **El servidor coneix l'estat i produeix una "vista"** en funció d'aquest (el JSON és el format més popular).
2. **Aquesta vista es transfereix a través de les capes** de comunicació (per exemple, mitjançant AJAX o WebSockets).
3. **Aquesta vista és decodificada pel client**, normalment mitjançant JSON.parse, per tal de transformar el JSON en dades manipulables pel framework.
4. **El client reconcilia l'estat actual amb el nou estat**, actualitzant només les parts de la interfície d'usuari que han canviat.
5. **Es determinen quines vistes s'han d'actualitzar**, en funció de l'estat del client.

Aquest enfocament implica una complexa gestió de l'estat al costat del client, on el framework ha de controlar què ha canviat i actualitzar les vistes de manera precisa. Això pot ser potent i flexible, però també introdueix una certa complexitat, amb més possibilitats d'introduir errors en la lògica de negoci, especialment quan es manipula l'estat i es reconcilia.

Enfocament d'HTMX

1. **El servidor coneix l'estat i produeix una "vista"** en format HTML.
2. **Aquesta vista es transfereix a través de les capes** de comunicació, similar a l'enfocament tradicional.
3. **Aquesta vista és decodificada per HTMX** i col·locada segons les regles establertes a l'element HTML que va originar la sol·licitud.

Es simplifica el procés perquè no cal gestionar l'estat al client. El servidor envia directament la vista en format HTML, i HTMX s'encarrega d'inserir aquest contingut al DOM segons les especificacions de l'element HTML. Això redueix la complexitat del codi, ja que elimina la necessitat de reconciliar estats o de processar dades JSON al client. No obstant, aquesta simplicitat pot suposar menys flexibilitat per a projectes grans o aplicacions que necessiten una gestió avançada de l'estat al client.

Nota: cal assenyalar que aquesta comparació conté una lleugera simplificació, ja que en certs casos HTMX pot necessitar mecanismes addicionals per gestionar determinades interaccions complexes.

1. Rendiment en la Generació de Contingut i velocitat

Un dels avantatges més destacats d'HTMX és la seva eficiència en la generació de contingut HTML, com he demostrat abans. Tot i que podria semblar que produir HTML des del servidor podria ser més lent en comparació amb la generació de JSON, això no és necessàriament cert. Generar JSON implica recórrer objectes, descobrir totes les claus, concatenar cadenes i navegar per tots els valors, la qual cosa pot ser una operació lenta i costosa en termes de temps de processament.

D'altra banda, la generació d'HTML és una simple interpolació de cadenes, que en molts casos pot ser més ràpida. De fet, un estudi d'Intel de l'any 2012 o 2013 indicava que més del 60% del temps dels servidors es dedica a copiar dades d'un format a un altre. Tenint en compte que JSON consisteix en copiar dades d'un format (objectes) a un altre (cadenes de text), és evident que aquesta operació no és tan ràpida com es podria pensar. Així doncs, HTMX pot oferir un rendiment comparable, o fins i tot superior, en la generació de contingut HTML en comparació amb la generació de JSON.

A demés combinat amb un backend com Go, que destaca per l'alta velocitat que té en procesar peticions podem extreure encara més rendiment a HTMX. És un altre de les grans avantatges que té, pot utilitzar qualsevol llenguatge per al backend.

2. HTMX per a desenvolupadors de backend

HTMX és especialment interessant per a desenvolupadors de backend que volen construir aplicacions web amb comportaments interactius sense haver d'aprendre frameworks de frontend com React, ja que HTMX és molt més senzill d'aprendre i utilitzar. Per exemple, en construir un clon de Twitter, HTMX permet que un tweet nou

aparegui immediatament sense necessitat de recarregar tota la pàgina, cosa que abans requeria l'ús de React.

3. Simplicitat d' HTMX

Aquesta facilitat d'ús es tradueix en un temps de desenvolupament més ràpid i una menor probabilitat d'errors. Els desenvolupadors no han d'aprendre una nova sintaxi o paradigma per gestionar l'estat al client, ja que HTMX delega gran part d'aquesta tasca al servidor. Això també significa que no cal preocupar-se per la reconciliació de l'estat o la gestió de la complexitat associada a la renderització del DOM des del client.

A més, programar la lógica de compartició d'estats entre components es simplifica molt amb HTMX, com he explicat en l'apartat anterior. Per exemple, mitjançant l'ús de headers especials com HX-Trigger-After-Swap i la funcionalitat hx-trigger, és possible gestionar l'estat de l'aplicació de manera eficient, com en el cas de l'actualització del comptador del carret. Amb HTMX, es pot obtenir el 80% de les funcionalitats d'una aplicació de pàgina única (SPA[25]) amb molta menys complexitat i sense la necessitat de dependre completament d'un framework de JavaScript.

Aquest aspecte fa que sigui ideal per a projectes petits o mitjans, com ara la creació d'un e-commerce senzill com el que he programat. En resum, és possible construir una aplicació de comerç electrònic funcional sense necessitat de gestionar un stack complet de JavaScript com he demostrat en el transcurs d'aquest treball.

4. Reducció del Bundle size

Aquesta simplicitat no només facilita el desenvolupament, sinó que també redueix la bundle size de l'aplicació. Les aplicacions construïdes amb React solen pesar bastant, degut a la necessitat de carregar diverses biblioteques i dependències per gestionar l'estat i renderitzar el DOM. Això pot afectar negativament el temps de càrrega inicial de la pàgina, especialment en dispositius amb recursos limitats o connexions a Internet lentes. En canvi, les aplicacions construïdes amb HTMX tenen un bundle molt més petit, ja que es basa en l'HTML i no necessita carregar grans quantitats de codi JavaScript addicional.

Aquest aspecte fa que HTMX sigui ideal per a projectes petits o mitjans, com ara la creació d'un e-commerce senzill com el que he programat. En resum, és possible construir una aplicació de comerç electrònic funcional sense necessitat de gestionar un stack complet de JavaScript.

5. Reduccio "Verbositat"

La verbositat es refereix a la quantitat de codi necessari per implementar una funcionalitat determinada. En aquest apartat, compararem la verbositat entre HTMX i React a través de dos exemples: una crida AJAX i la gestió d'esdeveniments.

---

25 SPA: Single-Page Application

Exemple 1. Crida AJAX:

React

```
import React, { useState } from 'react';

function MyComponent() {
    const [data, setData] = useState('');

    const fetchData = () => {
        fetch('/endpointX')
            .then(response => response.text())
            .then(data => setData(data));
    };

    return (
        <div>
            <button onClick={fetchData}>Click me</button>
            <div>{data}</div>
        </div>
    );
}

export default MyComponent;
```

HTMX

```
<button hx-get="/endpointY " hx-target="#targetX">Click me</button>
<div id="targetX"></div>
```

Exemple 2. Gestió d'Esdeveniments:

React

```
import React from 'react';

function MyComponent() {
    const handleClick = () => {
        alert('Button clicked!');
    };

    return (
        <button onClick={handleClick}>Click me</button>
```

```
    );
}
export default MyComponent;
```

HTMX

```
<button hx-on="click: alert('Button clicked!')">Click me</button>
```

Conclusions

En comparació, HTMX és l'opció menys verbosa, ja que permet a un desenvolupador implementar funcionalitats interactives amb molt menys codi. Això és especialment avantatjós en projectes més petits o mitjans, on la simplicitat i la rapidesa de desenvolupament són crucials.

React, tot i ser més eficient que JavaScript pur, encara requereix una estructura de codi més detallada, amb components i hooks per gestionar l'estat i les interaccions. Aquesta verbositat pot ser justificada en projectes grans i complexes, però per a aplicacions més senzilles, HTMX ofereix una alternativa molt més àgil i manejable.

4. KPIs (Key PerformanceMeasures) de l'aplicació

En aquest apartat, es realitzaran diverses proves per avaluar el rendiment de la meva aplicació utilitzant diferents **KPI** (Key Performance Indicators, Indicadors Clau de Rendiment). Els KPI són mètriques utilitzades per mesurar l'eficiència i l'eficàcia de l'aplicació en termes de rendiment. Aquests indicadors proporcionen dades objectives que ajuden a comprendre com es comporta l'aplicació en diferents aspectes, com ara el temps de càrrega, el consum de memòria i la resposta a les interaccions de l'usuari.

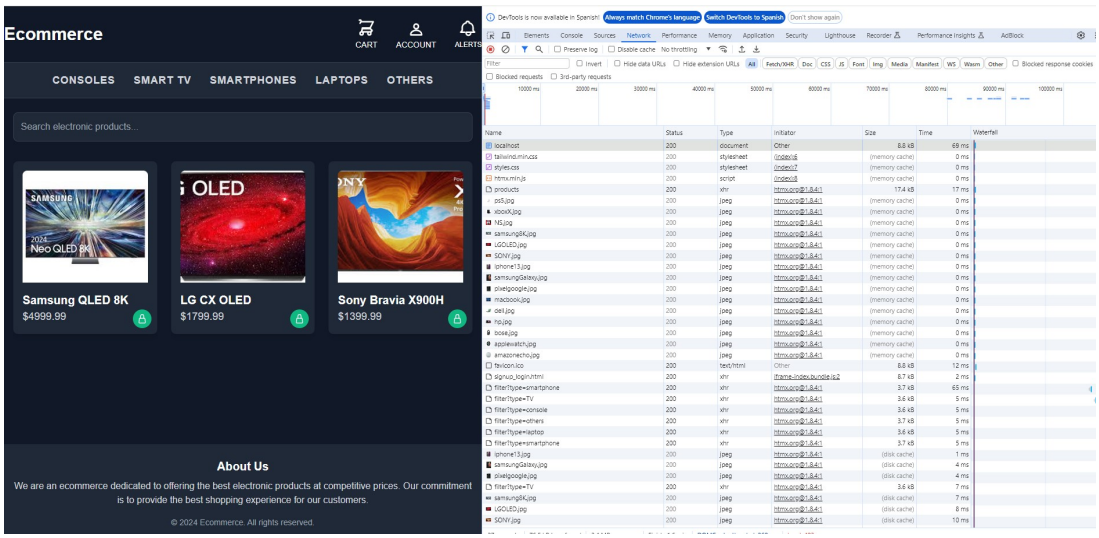

Figura 25: KPIs (Key PerformanceMeasures) del ecommerce pàgina principal

KPI	HTMX	Descripció
Bundle Size	8.8 KB	Mida del bundle (JS i CSS minimitzat).
Page Load Time (PLT)	69ms	Temps mitjà que triga la pàgina a carregar completament (en un navegador típic).
DOM Content Loaded	100ms	Temps fins que el DOM inicial està completament carregat i processat.
Time to Interactive (TTI)	150ms	Temps fins que la pàgina és completament interactiva després de carregar
Número de Sol·licituds HTTP	2	Nombre total de sol·licituds HTTP necessàries per carregar completament la pàgina.
Consum de Memòria	154 MB	Quantitat de memòria utilitzada pel navegador durant la càrrega de la pàgina.
Temps per Filtrar Productes	10.36ms	Temps de resposta mitjà per a l'acció de filtrar productes.

Taula 1: Taula KPI home Page

Explicació dels Resultats:

1. **Bundle Size (8.8 KB)**: El tamany del bundle és molt petit, la qual cosa és positiva, ja que permet una càrrega inicial molt ràpida de la pàgina, especialment en dispositius amb limitacions de recursos o connexions a Internet més lentes.
2. **Page Load Time (69ms)** i **DOM Content Loaded (100ms)**: Aquests temps indiquen una càrrega molt ràpida de la pàgina i del contingut del DOM, el que resulta en una experiència d'usuari fluida. Aquestes mètriques són excel·lents i mostren l'eficiència d'HTMX en la gestió de la càrrega inicial.
3. **Time to Interactive (150ms)**: El temps fins que la pàgina és completament interactiva és molt curt, la qual cosa és molt bona. Això vol dir que els usuaris poden començar a interactuar amb la pàgina gairebé immediatament després de carregar-la.

4. **Número de Sol·licituds HTTP (2)**: Un nombre molt baix de sol·licituds HTTP és beneficiós, ja que redueix la latència associada amb la càrrega de la pàgina i millora el rendiment general.
5. **Consum de Memòria (154 MB)**: El consum de memòria és relativament alt en comparació amb el tamany del bundle, però és normal en aplicacions modernes que mostren imatges i altres recursos dinàmics. Tot i que aquest valor és acceptable, es podria optimitzar en funció de la mida dels recursos carregats.
6. **Temps per Filtrar Productes (10.36ms)**: El temps de resposta per filtrar productes és molt ràpid, la qual cosa proporciona una experiència d'usuari molt bona quan es tracta de buscar o filtrar productes.

En general, aquestes mètriques mostren que la meva aplicació està molt ben optimitzada en termes de rendiment i eficiència, especialment pel que fa al temps de càrrega i la interactivitat. Això significa que els usuaris poden començar a interactuar amb la pàgina gairebé immediatament després de carregar-la, la qual cosa millora considerablement l'experiència d'usuari.

No obstant això, hi ha marge per millorar en termes de consum de memòria, que actualment és de 154 MB. Encara que aquest valor és acceptable per l'escala actual de la meva aplicació, és important tenir en compte que, en aquest moment, la meva aplicació no gestiona una gran quantitat de productes. Si la quantitat de productes augmentés significativament, el consum de memòria podria incrementar-se, afectant així el rendiment general de l'aplicació.

Per fer front a aquesta situació, es podria considerar implementar tècniques d'optimització com el lazy loading. Aquesta tècnica permet carregar només les imatges o elements visibles per a l'usuari en el moment, diferint la càrrega dels altres elements fins que siguin necessaris (per exemple, quan l'usuari fa scroll). D'aquesta manera, es redueix significativament el consum de memòria inicial i es millora l'eficiència en entorns amb recursos limitats. Aquesta seria una millora essencial en el cas que la meva aplicació creixés en volum de productes, garantint així que continuï oferint una experiència d'usuari òptima independentment de la seva escala.

8.3 Desavantatges d'HTMX

Tot i que HTMX pot assumir gran part del treball que abans només es podia fer amb frameworks com React, té les seves limitacions. No és adequat per a aplicacions que requereixen una gran interactivitat, com ara aplicacions de dibuix en canvas o interfícies d'usuari molt interactives com Figma. No obstant això, HTMX pot ser suficient per a moltes aplicacions, oferint una solució molt més senzilla i directa.

Un dels principals inconvenients que he trobat amb HTMX és la qualitat i l'abast de la documentació oficial. Tot i que és un tema molt "googlejable" (és a dir, si saps exactament el que busques, pots trobar molta informació) la profunditat i claredat de la documentació encara són limitades. A diferència de React, HTMX està en una fase de

maduració, i això es reflecteix en la seva documentació oficial. Aquesta manca de detall pot obligar els desenvolupadors a recórrer a fonts externes per trobar respostes a problemes específics o per comprendre millor com implementar certes funcionalitats.

Tot i que la comunitat d'HTMX és activa i es poden trobar recursos com fòrums, blogs i tutorials, la informació disponible sovint pot ser inconsistent o no estar actualitzada. Això pot crear dificultats per als desenvolupadors, especialment quan es tracta de resoldre problemes complexos o desplegar funcionalitats més avançades. Així, malgrat la seva relativa facilitat d'ús, pot presentar alguns reptes quan es busca aprofundir en el seu ús o abordar casos d'ús més sofisticats.

9 Conclusió

Durant aquest el transcurs del projecte, vaig tenir algunes dificultats, sobretot a l'inici, especialment amb el tema del "swap". Al començament, vaig experimentar problemes on la resposta HTML es solapava i em duplicava els formularis. Va ser una experiència frustrant, però em vaig adonar que la causa principal era que estava intentant programar amb HTMX amb la mateixa mentalitat que faria servir amb React. Requeria d'una manera diferent de pensar, i cal oblidar algunes de les pràctiques apreses amb frameworks més tradicionals per poder aprofitar al màxim el seu potencial.

Un cop vaig canviar el meu enfocament i vaig començar a comprendre millor la filosofia darrere d'HTMX, vaig trobar que és una eina relativament fàcil de dominar. Pot ser una solució excel·lent per a aplicacions que no requereixen tant dinamisme, on la simplicitat i l'eficiència són prioritaris.

També vaig tenir una mica de dificultat inicial amb el sistema de plantilles (templating), ja que em resultava una mica estrany en comparació amb el que estava acostumat amb React. Amb React, gaudeixo de la separació clara entre frontend i backend, especialment per la comoditat que això ofereix quan es treballa en equips separats. Aquesta separació facilita la col·laboració i pot fer que el codi es vegi més net i organitzat, tot i que, al final, això depèn dels gustos i preferències de cada desenvolupador.

En resum, no és necessàriament una substitució per a React en tots els projectes, però sí que ofereix una alternativa atractiva per a aquells que busquen una solució menys complexa i més directa per a aplicacions web senzilles o moderadament complexes. És fàcil de dominar un cop es comprèn el seu enfocament diferent, i pot ser una opció potent per a determinats tipus de projectes on el dinamisme extrem no és un requisit.

10 Referències

[1] React-Htmx. Article. https://bobaekang.com/blog/react-solid-htmx/

[2] Htmx sucks. Article. https://htmx.org/essays/htmx-sucks/

[3] JSON conversion. Documentació de intel. https://www.intel.com/content/dam/support/xl/es/documents/software/manageability-products/intel-ema-javascript-libraries.pdf

[4] HDA. Documentació. https://hypermedia.systems/more-htmx-patterns/

[5] Htmx. Documentació oficial. https://htmx.org/docs/

[6] Github. Repositori htmx https://github.com/bigskysoftware/htmx

[7] Htmx. Youtube. https://youtu.be/x7v6SNIgJpE?si=xNjXIgZuyT5LilV3

[8] Github. Repositori htmx. https://github.com/bigskysoftware/htmx/blob/master/src/htmx.js

[9] htmx vs react. Article. https://www.linkedin.com/pulse/ctos-perspective-htmx-vs-react-replace-saurabh-barot-1bmwf/

[10] Tailwind. Documentació. https://tailwindcss.com.

[11] HATEOAS. Documentació. https://en.wikipedia.org/wiki/HATEOAS

[12] Github. Repositori ecommerce. https://github.com/JoelTeoGom/htmx-golang-ecommerce

[13] JWT. Article. https://cloudsundial.com/salesforce-identity/jwt-bearer

[14] Visual Studio Code. Home page. https://code.visualstudio.com/.