Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament d'una eina interactiva d'aprenentatge sobre les Marine Protected Areas (MPA)

Document: Memòria

Alumne: Aleix Alvarez Calafell

Tutor: Anton Bardera i Xaquín Veira

Departament: Informàtica, matemàtica aplicada i estadística.

Àrea: Llenguatges i sistemes informàtics.

Convocatòria (mes/any): 09/2022

ÍNDEX

1. Introducció **6**

2. Motivacions **7**

3. Propòsit i Objectius **8**

4. Estudi de viabilitat **9**
   4.1 Viabilitat Tecnològica 9
   4.2 Viabilitat Econòmica 11

5. Metodologia **12**

6. Planificació **16**

7. Marc de treball i conceptes previs **20**
   7.1 Personal involucrat 20
      7.1.1 UNEP 20
      7.1.2 Fundació Visualització per a la Transparència (ViT) 21
      7.1.3 Tutors 21
      7.1.4 Desenvolupadors 22
      7.1.5 Dissenyadors 23
   7.2. Conceptes previs 24
      7.2.1 Estructura de la pàgina 24
         7.2.1.1 CMS 24
         7.2.1.2 Pages 25
         7.2.1.3 Tags 26
         7.2.1.4 Users 27
         7.2.1.5 Chapters 27
         7.2.1.6 Case studies 28
         7.2.1.7 Lifecycle 28
         7.2.1.8 Milestones 30
         7.2.1.9 Collection page 30
      7.2.2 Tecnologies bàsiques 30
         7.2.2.1 Serveless 30
            7.2.2.1.1 FaaS 31
            7.2.2.1.2 BaaS 31
            7.2.2.1.3 Arquitectura serverless 31
         7.2.2.2 Sveltekit 33
            7.2.2.2.1 Routing 34
               7.2.2.2.1.1 Pages 34
            7.2.2.2.2 Endpoints 35
               7.2.2.2.2.1 Page endpoints 36
         7.2.2.3 Vectors de cerca 37

7.2.2.4 Gestió de la cache amb CDNs 38

8. Requisits del sistema **42**
8.1 Requisits funcionals 42
8.1.1 CMS 42
8.1.2 Front-end 48
8.1.2.1 Landing page 48
8.1.2.2 Chapters 49
8.1.2.3 Case Studies 50
8.1.2.4 Collection pages 51
8.2 Requisits no funcionals 52

9. Estudis i decisions **53**
9.1 Programari 53
9.1.1 Svelte 53
9.1.2 SvelteKit 55
9.1.3 TypeScript 56
9.1.4 Copilot 57
9.1.5 Prisma 59
9.1.6 PostgreSQL 61
9.1.7 Fastly 62
9.1.8 Visual Studio Code 63
9.1.9 Stylus 63
9.1.10 Llibreries 63
9.1.10.1 Svelte-splide 63
9.1.10.2 Svelte-multiselect 63
9.1.10.3 Textfit 64
9.1.10.4 Svelte-scrollto 64
9.1.10.5 MenuSpy 64
9.2 Maquinari 64

10. Anàlisi i disseny del sistema **64**
10.1 Anàlisi i disseny de la Landing Page 65
10.1.2 Header 65
10.1.2 Collection Cards 66
10.1.3 MPA Management Life Cycle 68
10.1.3.1 Circle Menu 68
10.1.3.2 Circular Segment 71
10.1.3.3 Cards 73
10.1.3.4 Carousel Dots 75
10.1.3.5 Dot 77
10.1.4 Madlib 79
10.1.5 Footer 80

10.2 Anàlisi i disseny de les pàgines de contingut 80
10.2.1 Header 81
10.2.2 LifeCycle 82
10.2.3 Contingut 87
10.2.3.1 Sticky Menu 88
10.2.3.2 Recommended pages 89
10.3 Anàlisi i disseny Collection Pages 91
10.4 Anàlisi i disseny Pages 92
10.5 Anàlisi i disseny Tag Editor 95
10.6 Anàlisi i disseny Page Ordering 97
10.7 Anàlisi i disseny Authors 98
10.8 Anàlisi i disseny Users 99

11. Implementació, proves i resultats **99**
11.1 Components 99
11.1.1 CircularSegment 99
11.1.2 CircleMenu 103
11.1.3 LifeCycle 104
11.1.4 HelpPopup 114
11.1.5 Dot 115
11.1.6 CarouselDots 117
11.1.7 Cards 119
11.1.8 CardBody 123
11.1.9 ContentCarousel 126
11.1.10 StickyMenu 128
11.1.11 TagsEditor 131
11.1.12 TagEditor 135
11.1.13 PageOrdering 136
11.1.14 Altres 139
11.2 Endpoints i base de dades 139
11.2.1 Base de dades 140
11.2.2 Tags 141
11.2.3 Page Ordering 155
11.2.4 Recommended Pages 160

12. Conclusions **161**

13. Treball futur **163**

14. Bibliografia **164**

15. Manual d'usuari i/o instal·lació **168**
15.1 Manual d'instal·lació 168

15.2 Manual d’usuari. 171

1. Introducció

No podem sobreviure sense els oceans. Els ecosistemes oceànics han estat malmesos degut a les activitats humanes durant molts anys. La sobrepesca, l'explotació de recursos, el turisme, la recreació, el desenvolupament de la costa i la contaminació són les principals causes que destrossen els hàbitats i a conseqüència les espècies marines redueixen la seva població.

Es pot observar de forma visible com el canvi climàtic avança molt ràpidament a pitjor. Actualment, s'han perdut la meitat dels esculls de coral del món a causa d'un ús massiu dels recursos de l'oceà, provocant que el seu ús sigui major del que es poden recuperar naturalment.

Les MPA (Àrees Marines Protegides), que són regions clarament definides i reconegudes dels quals se'n vol conservar l'ecosistema són una de les millors opcions per poder mantenir els nostres oceans i evitar una destrossa major. A més de proporcionar beneficis ecològics, socials i econòmics.**[1]**.

Per aquesta raó es vol construir una eina interactiva d'aprenentatge que proporcioni informació sobre les MPA per a millorar la formació dels gestors d'aquestes àrees. Aquesta eina és un projecte entre la UNEP (Programa de les Nacions Unides per al Medi Ambient) que és un programa que coordina tot tipus de activitats relacionades amb el medi ambient, oferint ajuda a altres països per implementar polítiques mediambientals, la Universitat de Girona i la Fundació Visualització per a la Transparència que pretén promoure l'alfabetització en l'accés a les dades per tal de fomentar una ciudadanía més informada.

Es pretén desenvolupar un lloc web accessible i interactiu on es pugui trobar tota la informació i resoldre les qüestions que es puguin tenir sobre les MPA.

El desenvolupament del projecte inclou la creació d'un front-end fent ús de frameworks i llenguatges com Svelte, Javascript, HTML, CSS, Typescript, entre altres. Aquesta eina permet als usuaris gestors del lloc web elaborar, publicar i actualitzar gran part del contingut. Això implica tenir un CMS integrat amb el front-end i back-end amb una API amb sistemes d'autorització que permetin accedir al CMS de forma segura i que garanteixi la confidencialitat, integritat, disponibilitat de la informació.

2. Motivacions

No és la primera vegada que treballo en un entorn de desenvolupament web, en altres ocasions he pogut participar en altres projectes dels quals les meves tasques han consistiten solucionar errors informàtics o bé el desenvolupament de noves funcionalitats dins d'una aplicació ja creada. Penso que el fet de poder participar en un projecte des de zero dins d'un entorn laboral és una gran oportunitat nivell professional i personal. Ja que durant aquest procés podré aprendre moltes de les bases, tècniques i tecnologies actuals utilitzades per al desenvolupament d'aplicacions webs.

En aquest projecte espero aprofundir els meus coneixements actuals dels llenguatges de programació que farem servir, veure quins són els procediments a seguir i quines dificultats es presenten per al camí així com quines són les solucions més bones i perquè ho són.

A més també aprendré a fer ús de noves tecnologies amb les quals no havia treballat abans, així com control de la memòria cau, cercadors de contingut, el SEO (Search Engine Optimization) i altres.

Espero poder acabar aquest projecte i sentir que he adquirit els coneixements necessaris per poder desenvolupar qualsevol aplicació web. Saber els passos que he de seguir i com estructurar i organitzar l'aplicació. Saber com tots els mòduls estan relacionats entre si i sentir-me còmode quan s'hagin de fer qualsevol mena de modificacions en el programa.

3. Propòsit i Objectius

Les MPA són molt importants per al medi ambient, per aquesta raó és necessària una eina d'aprenentatge on formar-se. El propòsit és crear una eina web que facilita l'aprenentatge, oferint nou contingut que podrà ser gestionat pels administradors i editors de contingut responsables del lloc web, és a dir, podran afegir, editar o eliminar contingut. Aquest contingut podrà ser creat des d'un CMS creat exclusivament per satisfer les opcions i particularitats de la interfície gràfica del lloc web.

L'Objectiu principal és que aquest lloc web serveixi per formar a totes les persones del sector del manteniment i conservació de les MPA i poder donar una visió de les MPA a totes les persones que estiguin interessades. Per tal de poder complir aquests objectius a continuació faré menció de les funcionalitats de l'aplicació per tal que pugui complir amb la finalitat desitjada.

A continuació faré un desglossament dels punts a assolir durant aquest projecte:

- Crear un front-end de l'aplicació amb l'ajuda de tot l'equip.
    - Realitzar un disseny atractiu del front-end mitjançant eines col·laboratives com Figma.
    - Implementar el lloc web dissenyant usant les tecnologies Svelte, HTML, CSS, Javascript i TypeScript.
    - Implementar el lloc web perquè sigui accessible i compleixi els corresponents estàndard.
- Aprendre a fer ús de l'eina de control de versions Github.
- Crear un back-end de l'aplicació amb l'ajuda de tot l'equip.
    - Realitzar un bon disseny d'una base de dades mitjançant tecnologies com Prisma i PostgreSQL.
    - Implementar un bona gestió de la memòria cau a l'aplicació amb tecnologies com Fastly.
    - implementar un sistema d'autenticació d'usuaris.

- Implementar un sistema de cerca per poder buscar diversos elements dins del lloc web amb l'ajuda dels vectors de cerca.
- Realitzar un disseny d'una API mitjançant la tecnologia cloud serverless.

Aquest projecte es durà a terme amb tot l'equip de treball de la fundació ViT, per aquest projecte l'equip estarà format per 7 membres, es treballarà amb la modalitat de teletreball, el projecte s'iniciarà des de zero sense partir de cap base. La meva contribució en aquest projecte serà el desenvolupament de nous components i funcionalitats de l'aplicació així com la modificació dels components, correcció d'errors que puguin aparèixer i el testeig. Hauré d'aprendre com funciona l'aplicació al complet, ja que per dur a terme la meva feina requereix d'aquest coneixement.

4. Estudi de viabilitat

A l'inici del projecte, cal analitzar la viabilitat d'aquest projecte per tal d'estar segurs que el seu desenvolupament es pot dur a terme sense problemes. Haurem d'analitzar si el projecte és viable tecnològicament i per altra banda si també ho és econòmicament.

4.1 Viabilitat Tecnològica

Per dur a terme l'estudi de la viabilitat tecnològica de l'aplicació estudiarem les principals característiques i funcionalitats que requereix pel seu funcionament i veure si tecnològicament hi ha alguna solució ja disponible al mercat. Fent referència a l'apartat anterior podem identificar les necessitats tecnològiques que tindrà el projecte. Així doncs, a continuació faig un llistat de totes les necessitats i amb el candidat o metodologia potencialment indicada per la seva implementació.

- Per al disseny de les interfícies de l'aplicació requereix una eina que permeti fer Mockups de cada una de les seccions visuals i que alhora sigui col·laborativa, és a dir, que tots els membres de l'equip puguem treballar en ella paral·lelament. Com veurem més endavant hem escollit Figma ja que permet treballar simultàniament amb tots els membres de l'equip.
- Per al desenvolupament de la interfície gràfica necessitarem fer ús d'un framework tal que permeti un nivell d'abstracció major respecte el que

tenim amb HTML, JS, i CSS. Com veureu més endevant, em escollit fer ús de Svelte perquè és el framework utilitzat dins de la fundació i les seves prestacions compleixen amb els requisits de l'aplicació.

- Molts dels membres de l'equip estarem treballant simultàniament en diferents parts del projecte, per la qual cosa necessitem una eina que faciliti el control de versions i que permeti realitzar correccions de les tasques executades. Hem escollit fer ús de GitHub perquè ofereix un sistema de control de versión ideal per el treball en equip.
- Per desenvolupar l'aplicació també serà necessari un entorn de desenvolupament integrat (IDE). En aquest cas és bastant personal l'elecció, únicament requereix que tingui compatibilitat amb el control de versions GitHub. He escollit VSCode, ja que apart de tenir compatibilitat amb GitHub també ofereix gran quantitat de plugins que faciliten el desenvolupament, cal destacar Copilot, un plugin que usa intel·ligència artificial per fer suggeriments de codi.
- Per emmagatzemar les dades serà necessari fer ús d'un sistema d'administració de bases de dades. Hem escollit PostgreSQL que és un sistema de codi obert gratuït.
- Per al sistema d'autenticació farem servir SevelteKitAuth, una llibreria personalitzable que permet gestionar totes les autentificacions.
- Per al desenvolupament de l'API, necessitem un sistema encarregat d'obtenir totes les dades de la base de dades. En el nostre cas hem optat per utilitzar un sistema serverless que proporciona SvelteKit.
- Els editors poden crear les seves pròpies tags (etiquetes), però es donarà l'opció d'autogenerar paraules clau a partir del contingut escrit. Per realitzar això farem servir una eina anomenada TextRazor que disposa d'una API gratuïta amb certes restriccions.
- Per tal de millorar l'experiència de l'usuari i a la mateixa vegada reduir costos en el servidor també serà necessària una eina que permeti gestionar la memòria cau del lloc web a través d'una CDN (Content Delivery Network). S'ha decidit utilitzar Fastly.

4.2 Viabilitat Econòmica

La viabilitat econòmica del projecte és definida per l'estudi de costos, aquest es veu afectat per la viabilitat tecnològica, ja que, algunes de les tecnologies utilitzades poden tenir alguns costos associats. Cal tenir en compte que aquest projecte està finançat per l'ONU, per la qual cosa es disposa d'un cert pressupost en el desenvolupament de l'aplicació.

En referència els costos associats a les tecnologies estudiades en la viabilitat tecnològica específiques d'aquest projecte, tenim:

- Fastly té una versió de prova fins el que correspon a 50 euros de tràfic**[2]**, aquests costos estan relacionats amb el nombre de peticions realitzades i la ubicació de les peticions. És complicat preveure si serà suficient la versió gratuïta, ja que fins que l'aplicació no estigui en marxa no podrem estimar ni el nombre d'usuaris que l'utilitzaran ni des d'on l'utilitzaran.
- TextRazor té una versió gratuïta, que s'ha estimat que en un principi no se superarà el nombre de peticions diàries que permet la versió gratuïta (500 peticions)**[3]**.
- Svelte, Sveltekit, TypeScript, PostgreSQL, Prisma, SvelteKitAuth i VSCode no tenen cap cost associat.

Un altre factor a tenir present és el cost per programador del projecte. El sou rebut com a programador en aquest entorn és de 10 €/hora per un total de 25 hores setmanals, el que dona lloc a un sou mensual aproximat de 1000 €/mes. El temps total d'aquest projecte és de 6 mesos. Per al que el cost total per programador puja als 6000 €.

Durant el procés de desenvolupament el repositori ha estat allotjat a Github de forma privada, el qual té un cost aproximat de 45 €/any**[4]**. A més també s'ha fet ús de l'eina Slack per dur a terme la comunicació amb els membres de l'equip, la llicència utilitzada és la Pro la qual té un cost de 6,25 €/mes per persona**[5]**. Finalment, també fem ús de Notion, que serveix per planificar, gestionar, organitzar projectes... El pla emprat és el Team el qual té un cost de 8 $/mes per usuari**[6]**. Com que totes aquestes plataformes

són d'ús habitual, és a dir, no específiques d'aquest projecte i que alhora també es fan servir per a altres projectes considerem que són un 10% de costos indirectes.

El maquinari fet servir pel desenvolupament és el meu ordinador personal, per al que no el considerem com un cost pel projecte.

5. Metodologia

Per el desenvolupament del projecte hem utilitzat la metodologia SCRUM. En Aquest procés s'apliquen un conjunt de bones pràctiques per treballar en equip i obtenir el millor resultat possible del projecte.
Amb aquest mètode es realitzen entregues parcials i regulars del projecte final. En SCRUM es duen a terme cicles temporals curts i de duració fixa**[7]**. En el nostre cas aquest cicles temporals són setmanals, on cada dilluns tenim una reunió conjunta amb tots els membres de l'equip, la seva duració aproximada són d'uns 45 minuts.
S'expliquen totes les novetats relacionades amb els projectes actuals o possibles projectes futurs. A més, cada membre exposa el que ha fet durant la setmana anterior i explica el que farà durant la setmana. A la reunió també s'assignen noves tasques a dur a terme (veure *Figura 1*).

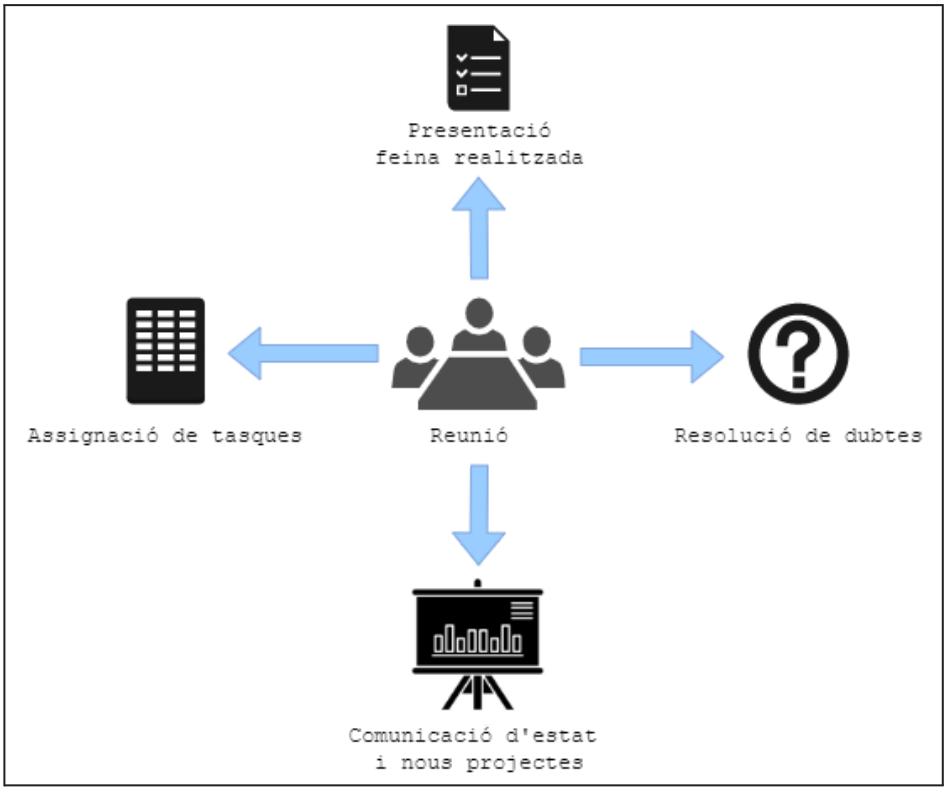


Figura 1. Esquema reunions setmanals.

Durant la setmana utilitzem la plataforma Slack per comunicar-nos amb el membres de l'equip. Normalment ho fem a través d'un chat dedicat exclusivament al projecte actual, (veure *Figura 2*). El fem servir per exposar dubtes, mostrar resultats i altres temes relacionats amb el projecte. Si per alguna raó a través del xat no podem obtenir la claredat o fluïdesa desitjada que ens permeti resoldre els problemes sorgits, llavors fem una trucada per solucionar els problemes que tenim. Aquestes trucades també les fem a través de Slack.

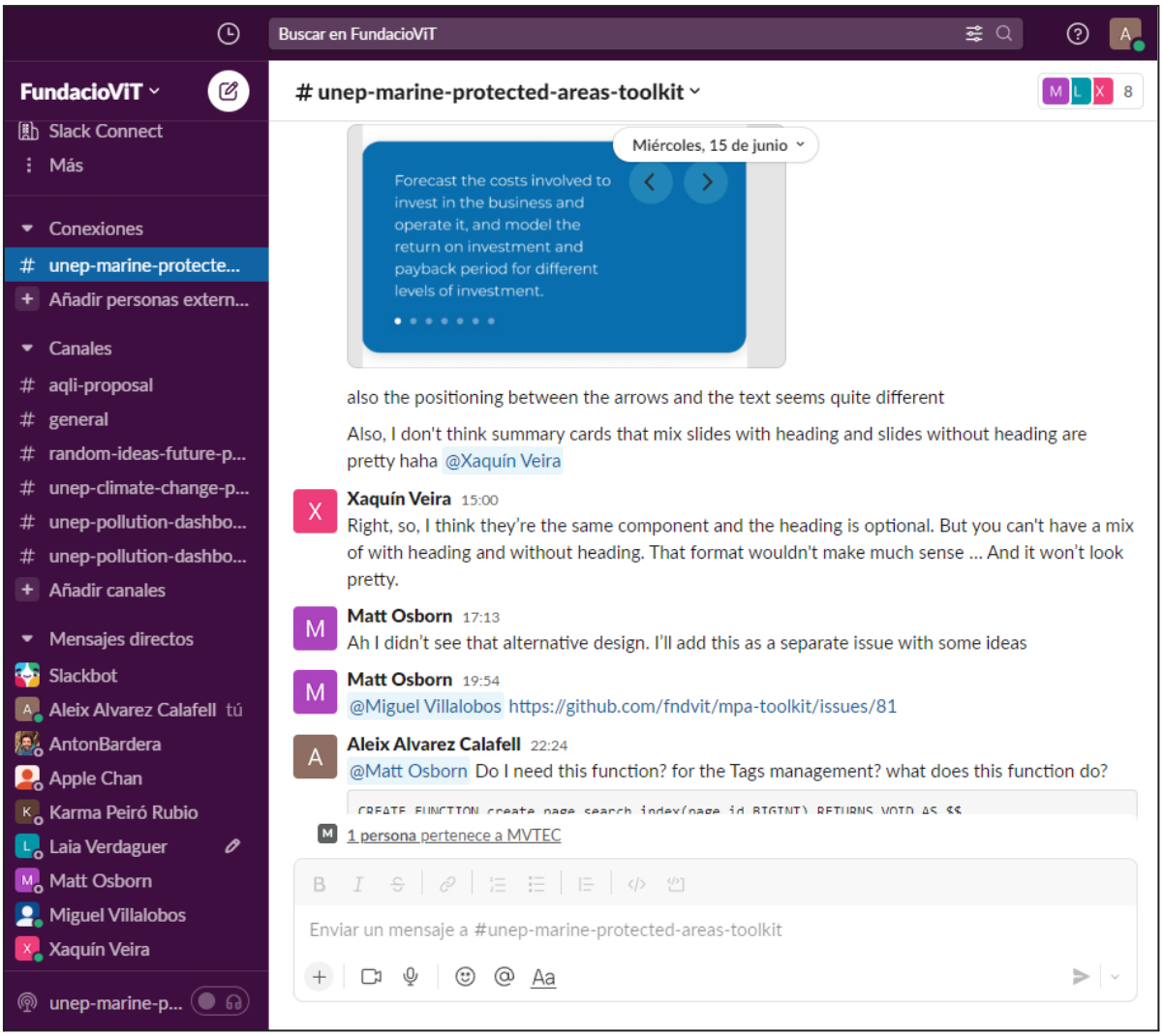


Figura 2. Canal de comunicació de Slack.

En el procés de desenvolupament fem servir l'eina Github, la qual permet treballar amb un sistema de versions molt òptim. Quan es desenvolupa una nova funcionalitat, es creen noves branques i cada membre del grup treballa en la branca corresponent. Llavors quan s'han finalitzat les tasques corresponent a la branca de treball es crea una nova Pull Request i el technical lead la revisa. A vegades es requereixen canvis en el codi, on es comuniquen a través de comentaris dins de la mateixa Pull Request. En algunes ocasions el canvis que solicita estan fora dels meus coneixements i llavors el que fem es una videotrucada on el technical lead m'ensenya com fer-ho. Quan tot està correcta es procedeix a fusionar la branca creada amb la branca principal.

També disposem d'una secció de problemes dins del propi Github. El technical lead identifica les tasques restants i ens assigna a cada membre en una (veure *Figura 3*).

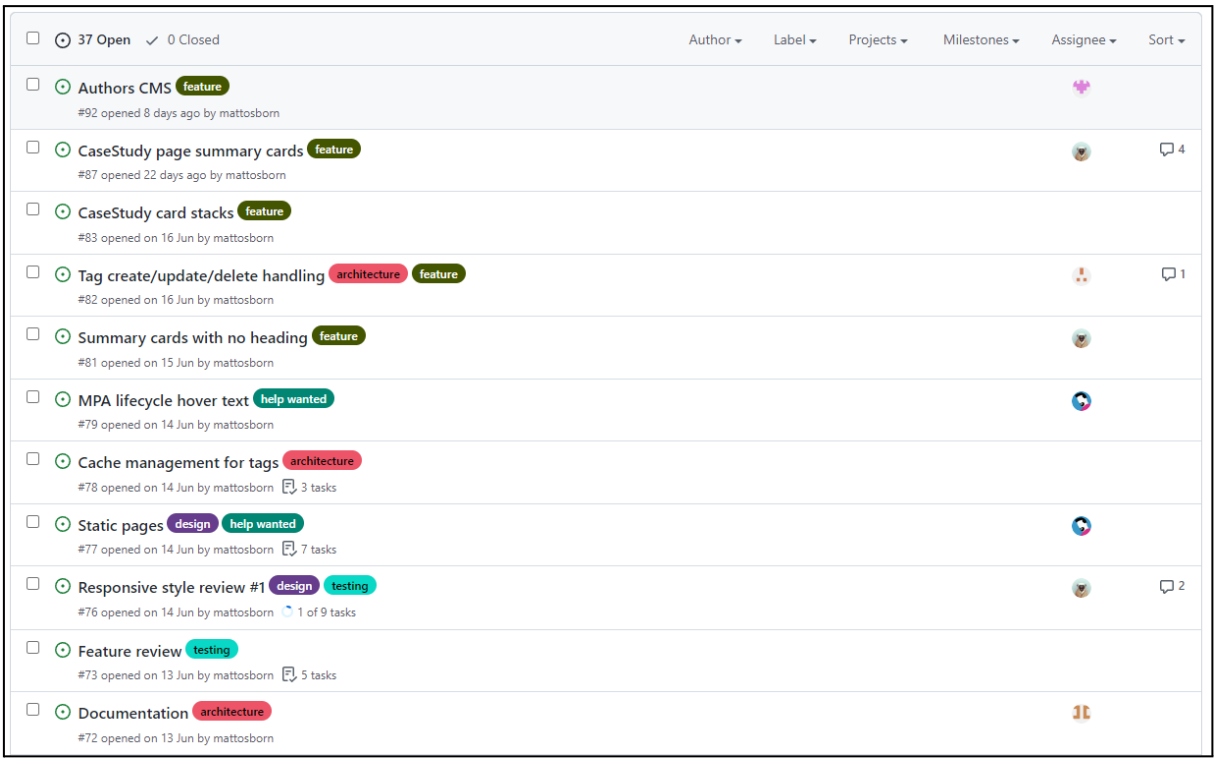


Figura 3. Secció de problemes a Github.

Per complementar el procés de desenvolupament, concretament la part de “front-end”, fem ús de Figma. Figma es una plataforma que permet dissenyar interficies d’usuari de forma col·lectiva a més de que permet exportar els estils en format CSS. En auqesta plataforma dispossem de tots els disseny actuals, per tenir referències de com te que ser la visualització. Tots els canvis en el disseny es mostren aquí (veure *Figura 4*).

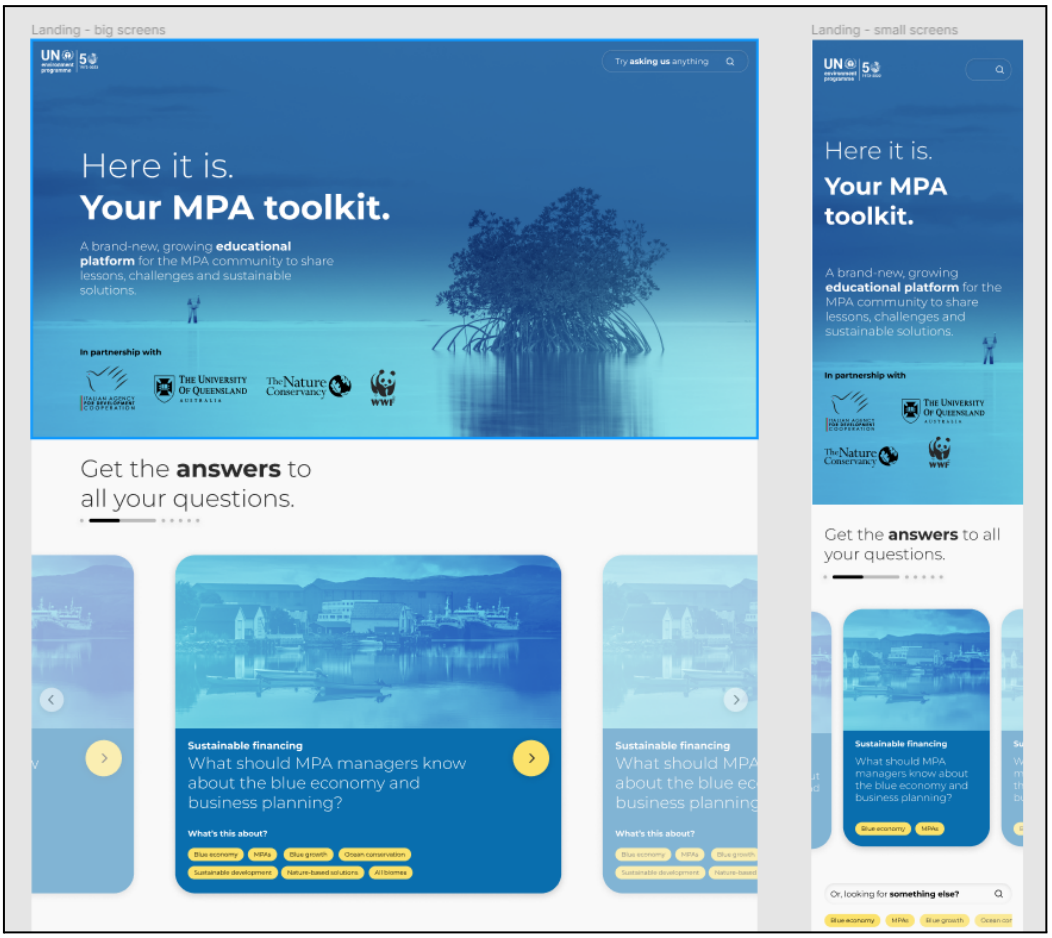


Figura 4. Visualització Landing Page dins de Figma.

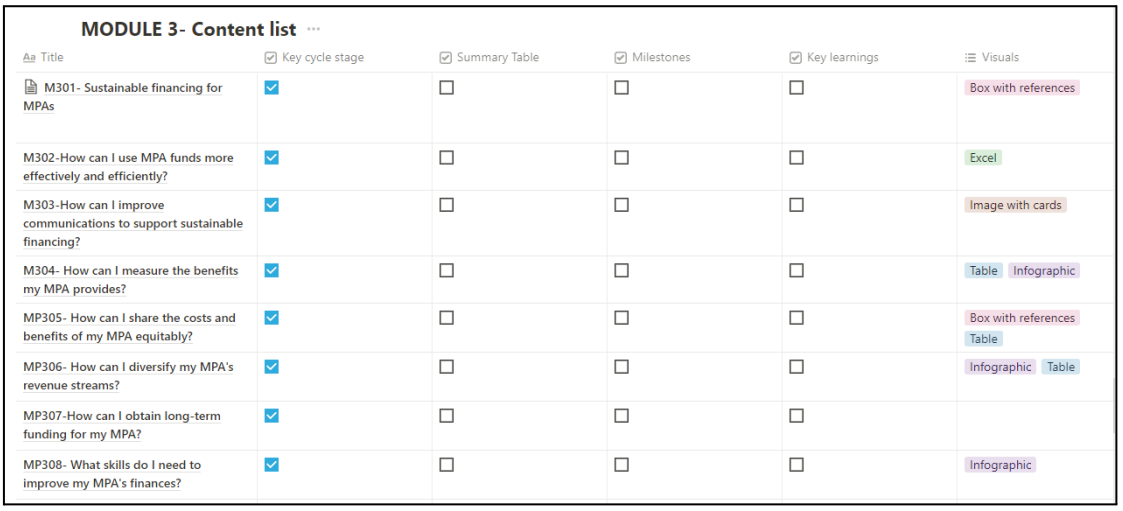
6. Planificació

Aquest projecte s'inicia al novembre del 2021 i té programada una durada de 6 mesos des de la data d'inici. Inicialment estava prevista la entrega del PFG al juny de 2022, però vaig decidir posteriorment fer l'entrega del projecte al setembre de 2022 ja que d'aquesta manera el projecte estaria més complet.

A mitjans de l'any 2021 des de la fundació ja se'ns va introduir el projecte, no es van especificar els detalls i a més se'ns va proposar aquest projecte com a projecte de final de grau.

Primerament es desenvoluparà una eina per poder realitzar una enquesta a diferents perfils d'usuari per donar una ressenya sobre els elements a més rellevants i menys rellevants.

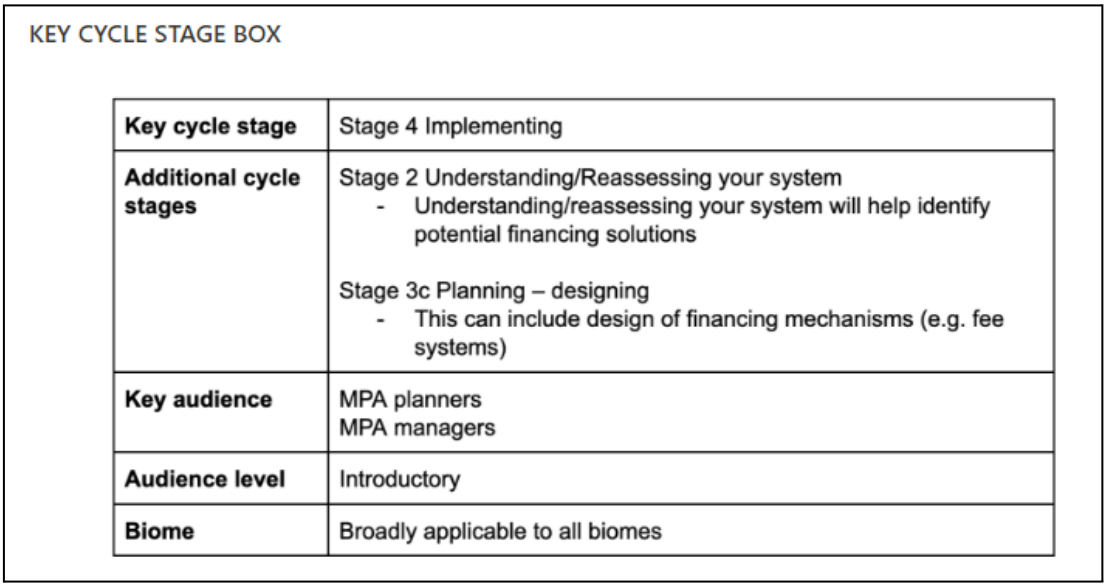
Després es fa la fase de disseny on s'identifiquen totes les funcionalitats necessàries a partir de les peticions del client. En aquest cas se'ns va facilitar una llista de tot el contingut inicial que es volia introduïr en el CMS. A través de notion es va facilitar una llista, i altres requisits que el sistema havía de mostrar (veure *Figures 5 i 6*), corresponen a algun exemple de la informació proporcionada.



MODULE 3- Content list

Title	Key cycle stage	Summary Table	Milestones	Key learnings	Visuals
M301- Sustainable financing for MPAs	☑	☐	☐	☐	Box with references
M302-How can I use MPA funds more effectively and efficiently?	☑	☐	☐	☐	Excel
M303-How can I improve communications to support sustainable financing?	☑	☐	☐	☐	Image with cards
M304- How can I measure the benefits my MPA provides?	☑	☐	☐	☐	Table Infographic
MP305- How can I share the costs and benefits of my MPA equitably?	☑	☐	☐	☐	Box with references Table
MP306- How can I diversify my MPA's revenue streams?	☑	☐	☐	☐	Infographic Table
MP307-How can I obtain long-term funding for my MPA?	☑	☐	☐	☐	
MP308- What skills do I need to improve my MPA's finances?	☑	☐	☐	☐	Infographic

Figura 5. Llista de contingut del mòdul 3.



KEY CYCLE STAGE BOX

Key cycle stage	Stage 4 Implementing
Additional cycle stages	Stage 2 Understanding/Reassessing your system - Understanding/reassessing your system will help identify potential financing solutions Stage 3c Planning – designing - This can include design of financing mechanisms (e.g. fee systems)
Key audience	MPA planners MPA managers
Audience level	Introductory
Biome	Broadly applicable to all biomes

Figura 6. Key cycle *stage* box.

D'aquesta part se'n encarregaran els dissenyadors. Paral·lelament a mesura de que nous components siguin dissenyats els desenvolupadors començarem a fer la seva implementació a un storybook, una plataforma que té integració amb Svelte que permet visualitzar els components de forma ràpida i amb possibilitat de testeig, és a dir,

modificar les seves propietats i comprovar que el seu funcionament és l'adequat (veure *Figura 7*).

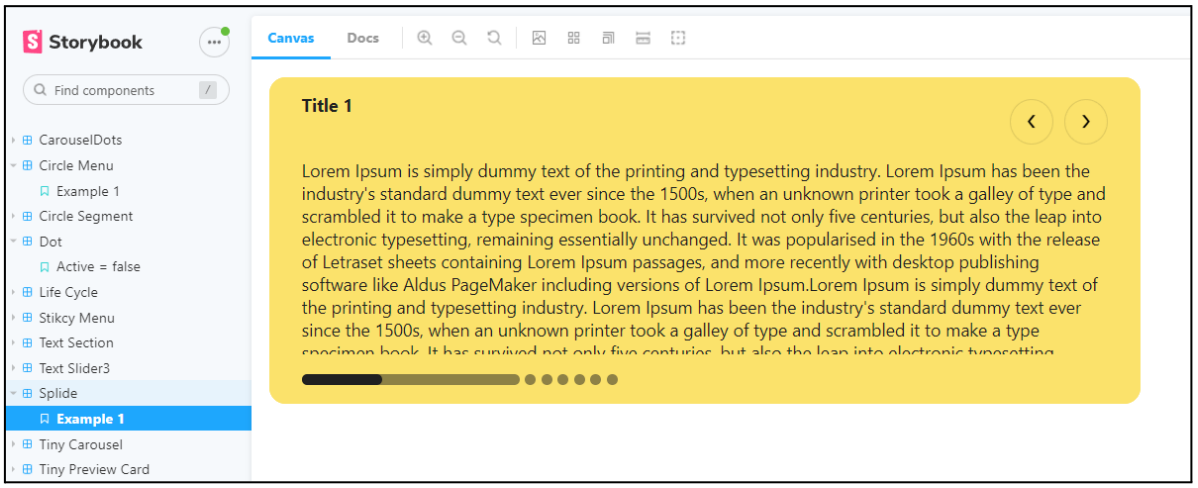


Figura 7. Previsualització dels components amb StoryBook.

Un cop fet el disseny cal identificar les tasques a realitzar i una possible estimació d'aquestes. Aquestes són estimades en dies i corresponen a 3 parts de l'aplicació: API, CMS i front-end.

API

Nom	**Estimació (dies)**
Architecture, setup, devops	4
User endpoints	4
Author endpoints	2
Content endpoints	4
Landing page endpoints	2
Authorization	2
Media endpoints	2
Search	10

CMS

Nom	**Estimació (dies)**
Hotspot editor	4

Nom	Estimació (dies)
Landing page	4
Summary cards	4
Case-study metadata fields	2
Inline images	4
Architecture, setup, devops	4
User login	2
Content listing / browser	4
Authors (names / images)	2
Content skeleton	2
Metadata fields (headline, summary, lifecycle, keywords, authors)	4
Cover images	2
Basic editor integrations	2
Metadata tags	2
Content tagging (metadata tags)	4
Case-study skeleton	2
Globe viz	4

Front-end

Nom	**Estimació (dies)**
Related links	2
Landing page skeleton	2
Collection page	4
Content cards	2
Inline madlib	2
Inline images	2
Case-study head	2

Globe viz	2
Milestones	2
Architecture, setup, devops	4
Content page skeleton	2
Conent head	2
Lifecycle viz	2
Summary cards	2
Content text	2
Menu	2
Search	2

Alhora també es crea el repositori i es crea dins de la plataforma GitHub una llista dels elements restants. El projecte s'inicia i setmanalment es fan revisions i afegint noves característiques al repositori.

7. Marc de treball i conceptes previs

7.1 Personal involucrat

Aquest projecte ha estat realitzat dins un entorn professional, per al que a part de jo també han estat involucrats altres professionals del sector. En aquest punt presentaré a tots ells.

7.1.1 UNEP

El client del nostre projecte és l'ONU, aquest projecte formarà part del programa pel medi ambient de les Nacions Unides (UNEP). Durant el transcurs del projecte algunes persones d'UNEP han estat fent el seguiment del projecte.

7.1.2 Fundació Visualització per a la Transparència (ViT)

La Fundació Visualització per a la Transparència (ViT) va ser fundada el 2020 per la Karma Peiró i en Xaquín Veira. La fundació pretén promoure l'alfabetització en l'accés a les dades, la responsabilitat pública i l'ús de les dades i visualització per fomentar una ciutadania més informada, participant i empoderada**[8]**.

Els seus objectius generals són:

- Desenvolupar alfabetització visual de dades per combatre la desinformació en tots els àmbits de la societat.
- Aconseguir una bona transparència en les dades públiques per tal de responsabilitzar les administracions públiques.
- Convertir-se en un referent de bones pràctiques en la visualització i la transparència de dades per a l'administració pública, instituts d'investigació i mitjans de comunicació.

Per tal d'aconseguir aquests objectius, la fundació es focalitza en diferents activitats en les següents tres àrees:

- Activitats educatives en la visualització i la ciència de dades per l'anàlisi i la comunicació.
- Un laboratori multidisciplinari per construir i desenvolupar projectes públics de dades obertes i eines d'investigació.
- Una col·lecció d'eines tecnològiques per a ciutadans, administracions públiques i entitats privades.

7.1.3 Tutors

Els tutors d'aquest projecte han estat l'Anton Bardera i en Xaquín Veira, els quals s'han encarregat d'orientar i fer un seguiment de la memòria i del projecte. Inicialment, vam acordar fer una reunió cada dues setmanes on es presentarien els avenços fets i es discutiria sobre els pròxims passos a seguir, posteriorment vam continuar amb reunions setmanals de seguiment del projecte general i per temes més concrets del projecte,

aquestes reunions es feien a través de canals de comunicació digitals. Cada vegada que tenia algun avanç o necessitava la seva ajuda els hi comunicava i posteriorment si era necessari realitzàvem una tutoria en línia.

NOM	Anton Bardera
ROL	Tutor
CATEGORIA PROFESSIONAL	Membre departament d'Informàtica, Matemàtica Aplicada i Estadística
RESPONSABILITATS	Seguiment del projecte
CONTACTE	anton.bardera@imae.udg.edu, 972418831

NOM	Xaquín Veira
ROL	Tutor
CATEGORIA PROFESSIONAL	Periodista visual i expert en visualització de dades
RESPONSABILITATS	Seguiment del projecte
CONTACTE	xaquin.veira@udg.edu

7.1.4 Desenvolupadors

Els desenvolupadors hem set els encarregats de desenvolupar el projecte, el qual un cop definits els requisits, conjuntament, hem analitzat i implementat el programari. Aquest equip en formem part 4 membres, en Matt Osborn (líder tecnològic), en Miguel Villalobos (programador júnior), Laia Verdaguer (programadora júnior) que es va incorporar més endavant i jo mateix (programador júnior).

NOM	Matt Osborn
ROL	Líder tecnològic
CATEGORIA PROFESSIONAL	Enginyer de Software
RESPONSABILITATS	Desenvolupador i assessor tecnològic del projecte

NOM	Miguel Villalobos
ROL	Programador júnior
CATEGORIA PROFESSIONAL	Dissenyador i desenvolupador de videojocs
RESPONSABILITATS	Desenvolupament i disseny del projecte

NOM	Laia Verdeguer
ROL	Programadora júnior
CATEGORIA PROFESSIONAL	Dissenyadora i desenvolupadora de videojocs
RESPONSABILITATS	Desenvolupament i disseny del projecte

7.1.5 Dissenyadors

Els dissenyadors han set els encarregats de tot el disseny visual de l'aplicació i de la interacció amb tots els components, també van fer una anàlisis de tota la informació aportada per part del client amb la finalitat de trobar amb una solució visual.

NOM	Xaquín Veira
ROL	President de la fundació i dissenyador
CATEGORIA PROFESSIONAL	Periodista visual i expert en visualització de dades
RESPONSABILITATS	Disseny i comunicació amb els membres de la ONU
CONTACTE	xaquin@fundaciovit.org

NOM	Rebecca Pazos
ROL	Dissenyadora

CATEGORIA PROFESSIONAL	Periodista de gràfics interactius
RESPONSABILITATS	Disseny

7.2. Conceptes previs

A l'inici de la memòria s'ha explicat que es realitzarà una eina interactiva, destinada a la informació i a la formació per a totes les persones de l'àmbit de les MPA. Tota aquesta informació haurà de ser creada pels experts del sector. Per això faran ús de les eines que desenvoluparem, a més, per tal de poder-nos comunicar amb tot l'equip de treball es van assignar diversos noms per facilitar el desenvolupament. En aquest punt detallaré els conceptes utilitzats per tal de poder entendre amb més facilitat quan es faci referència a un d'ells. També es detallaran altres tècniques usades d'interès.

7.2.1 Estructura de la pàgina

7.2.1.1 CMS

El Gestor de continguts o bé CMS (Content Management System) és la base del projecte, en ella es poden crear nous *Chapters*, *Case Studies*, administrar els usuaris i les tags. El CMS disposa d'un editor de text i altres eines que permeten crear tot el contingut del lloc de manera fàcil i intuïtiva (veure *Figura 8*).

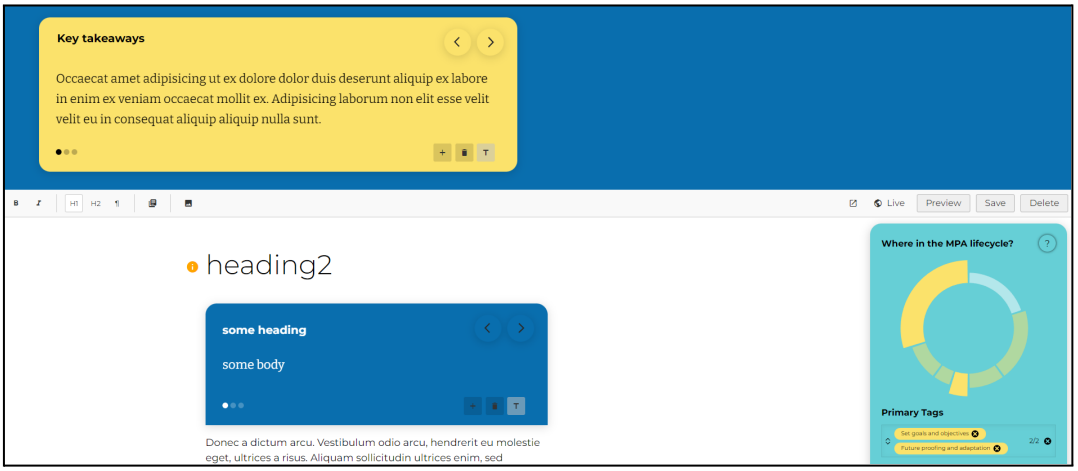


Figura 8. Visió general de l'editor de capítols.

En el mercat hi han diverses opcions ja desenvolupades, algunes de les més populars poden ser Wordpress, el qual es caracteritza per la seva facilitat per crear pàgines web i

la seva gran quantitat de plugins (*veure Figura 9*). En aquest projecte no s'ha optat per l'ús de WordPress ni cap CMS semblant, ja que hi ha components específics i els CMS del mercat actual no estan preparats per satisfer les necessitats del projecte actual, pel fet que són generalistes.

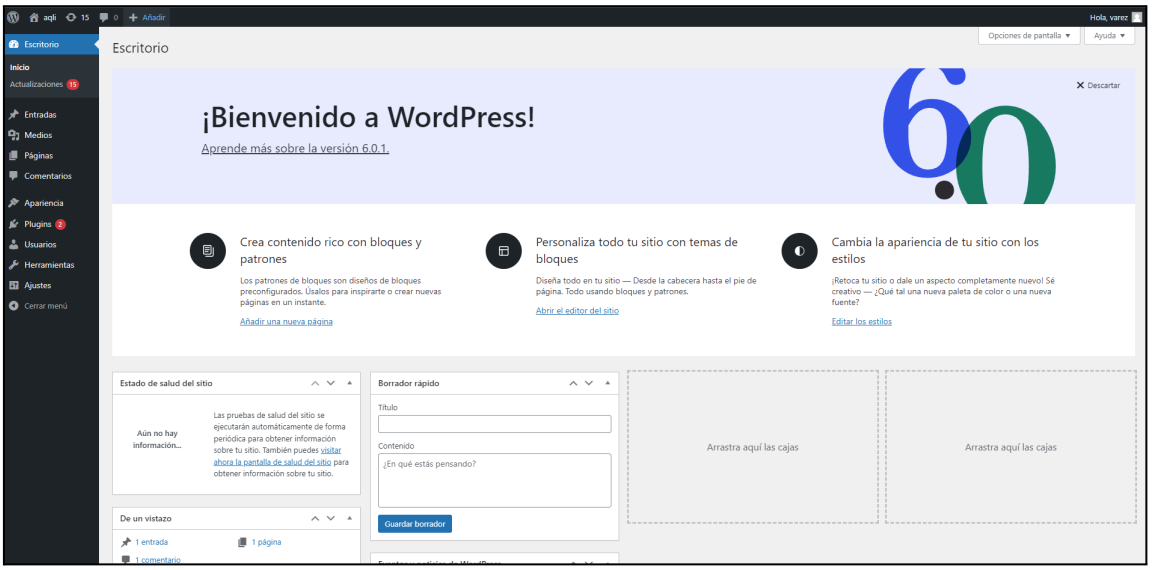


Figura 9. Visió del dashboard de WordPress.

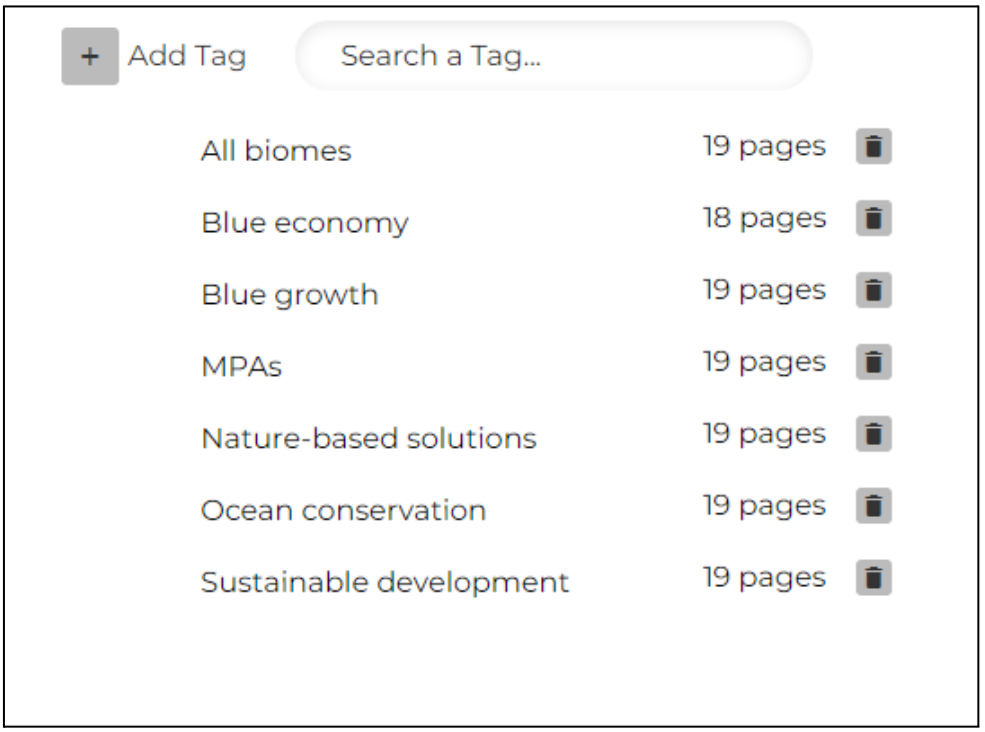
7.2.1.2 Pages

La pàgina de pages, ubicada en el CMS és el mòdul que permet crear tot el contingut del lloc web. Et permet crear nous Chapters i nous Case Studies. La *Figura 8* que s'ha vist anteriorment correspon a l'editor de Chapters i Case Studies. Es pot observar com està formada per una secció superior que permet gestionar la capçalera de l'article, un cos, corresponent a l'editor de text i una taula ubicada a la dreta la qual serveix per gestionar les tags del Chapter o Case Study. Aquesta és una visió molt general del CMS, més endavant s'aprofundirà més en detall. Si veiem la *Figura 10* es pot veure com és la seva visualització.

Figura 10. Visualització de les pages dins del CMS.

7.2.1.3 Tags

Les tags són paraules clau relacionades amb cada Chapter o Case Study. Es poden distingir 3 tipus de tags, les *stage* que corresponent a quina part del cicle de vida de les MPA el Chapter o Case Study està situat, les *topic* que resumeix sobre els temes dels quals parlen els Chapters i el Case Studies i per últim les *user* que orienten sobre quin tipus de lector el Chapter o Case Study està enfocat. A continuació, si observem la *Figura 11,* podem veure com la pàgina del CMS destinada a les tags es veuria.



Figrua 11. Editor de tags.

7.2.1.4 Users

Els *users* corresponen a les persones registrades a l'aplicació, cal distingir els diferents tipus d'usuaris que pot haver-hi, els quals anomenem rols. El rol *user* correspon a un usuari bàsic, per altra banda, el rol *content manager* el qual correspon als usuaris encarregats de l'edició i la creació del contingut i finalment el rol *admin* que és el que disposa de més privilegis. Més endavant ja profunditzarem més en detall dels privilegis de cada rol. Si mirem la *Figura 12* podem veure com es mostraria la secció users del CMS. En ella es poden editar els permisos dels usuaris i modificar les dades relacionades amb ells.

Figura 12. Editor d'usuaris.

7.2.1.5 Chapters

Els Chapters són articles que estan ubicats en un punt de les MPA, tracten temes específics que són importants explicar i treballar sobre les MPA. En els capítols es presenta un títol, biografia dels autors i un petit resum. Els Chapters segueixen una estructura: estan formats per una introducció, un cos on es desenvolupa el Chapter, les conclusions, i les referències bibliogràfiques.

7.2.1.6 Case studies

Els Case Studies són articles que estan ubicats a un punt de les MPA, a diferència dels Chapters, se centren en un cas pràctic real. En el Case Studies, es presenta un títol, l'any que va ser establert, la superficie que comprèn, la governança, els treballadors, el pressupost, el nivell del pressupost, una ubicació i les milestones. Els Case Studies

segueixen una estructura: estan formats per una introducció, un cos on es desenvolupa i finalment unes conclusions.

7.2.1.7 Lifecycle

Aquest punt va destinat a explicar els punts on pot estar ubicat un Chapter o Case Study, recordem que hi ha un tipus de tag destinat a complir aquesta funció, les tags de tipus *stage*. Dins del cicle de vida d'una MPA es poden diferenciar 5 escenaris diferents:

- **Escenari 1 - Identificar el problema:** Quin és el repte? Una MPA és la solució adequada? Si sabem quin és el problema al qual l'usuari vol fer front els articles que estan en aquest escenari ajudaran a determinar si les MPA són les solucions correctes al problema i et permetrà dissenyar una MPA o una xarxa de MPA realment efectiva per al propòsit desitjat.

- **Escenari 2 - Entendre el teu sistema**: Quin és el context social ecològic? Sabent quina informació existeix per donar suport al disseny de la MPA i al seu desenvolupament de gestió de plans efectius farà que la planificació d'una MPA sigui més efectiva.

- **Escenari 3 - Planificació**
    - **Escenari 3A - Implicar les parts interessades**: Com implicar millor les parts interessades? La implicació de les parts interessades al principi del procés de planificació d'una MPA, fins i tot abans que comenci la planificació real, pot garantir que hi haurà una acceptació duradora de la MPA i un millor compliment, especialment si es fixen metes i objectius realistes mitjançant l'establiment d'objectius participatius.
    - **Escenari 3B - Com establir/revisar millor una visió i objectius**: La implicació de les parts interessades en l'establiment d'objectius participatius genera confiança mitjançant la transparència i estableix el marc per poder avaluar i adaptar el rendiment de la MPA. Això s'aconsegueix mitjançant un procés de planificació que permet la cocreació d'una visió per a les MPA i l'articulació d'unes metes i propòsits molt específics, sovint decidits per negociació i compromís.

    - **Escenari 3C - Com dissenyar millor una MPA i ajustar el disseny d'una MPA**:
        - El disseny i la regulació/gestió òptims de la MPA reflectiran les condicions, les amenaces i les capacitats del lloc concret. No hi ha una recepta única per a fer una MPA efectiva, tot i que hi ha processos provats per aprofitar la informació disponible (científica, social i basada en l'usuari) per crear dissenys òptims de MPA per adaptar-se a les circumstàncies.
        - Les MPA realment efectives són aquelles aptes per al seu propòsit, amb objectius clars i mesurables. Aquests propòsits poden ser ecològics o ambientals, però també poden ser socials. Els millors dissenys i plans de gestió reflectiran, per tant, la finalitat específica de la MPA, sigui quina sigui aquesta.

- **Escenari 4 - Implementació, com gestionar efectivament i eficientment les MPA.**

- **Escenari 5 - Futures proves i adaptacions**: Com avaluar/revaluar el rendiment de les MPA i ajustar la gestió per assolir els objectius: Les MPA amb més èxit en el futur seran aquelles que s'adaptin a les condicions socials i ambientals que canvien ràpidament; serà essencial poder anticipar-se a aquestes condicions i ajustar la gestió àgilment.

7.2.1.8 Milestones

Les milestones fan referència als esdeveniments temporals en el desenvolupament d'un Case Study, aquestes només es mostra en els Case Studies, ja que en els Chapters per la seva naturalesa no tenen milestones (veure *Figura 13*).

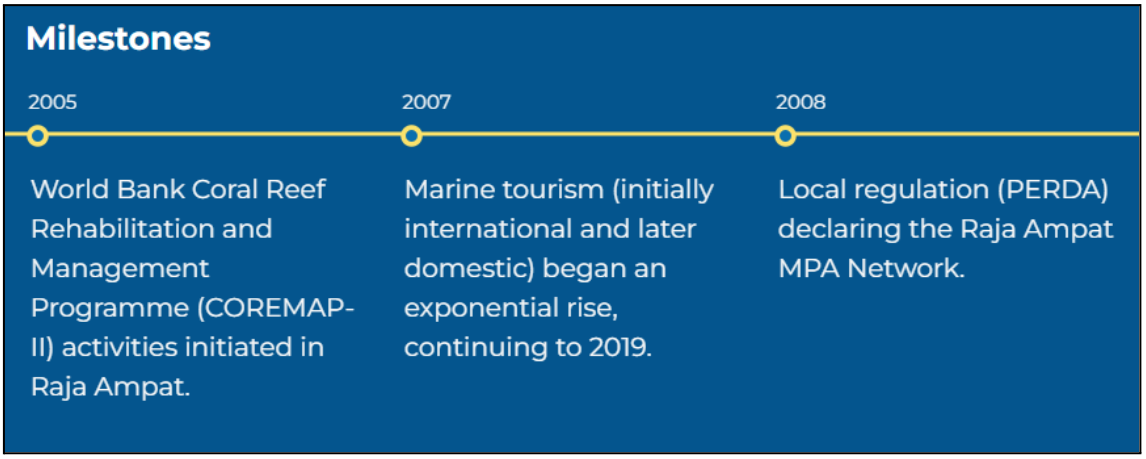


Figura 13. Exemple de les milestones.

7.2.1.9 Collection page

La collection page és la pàgina de cerca on es mostra tot el contingut, en ella es mostren els resultats de cerca. Es poden fer cerques per nom d'autor, tags i text.

7.2.2 Tecnologies bàsiques

En aquest punt em centraré a introduir algunes de les tecnologies bàsiques del sistema desenvolupat. En l'apartat 9 aprofitaré algunes de les explicacions realitzades en aquest punt per justificar l'elecció d'aquestes tecnologies.

7.2.2.1 Serveless

La tecnologia serverless és un model natiu del cloud, que et permet desenvolupar aplicacions sense cap necessitat de preocupar-te per la gestió del servidor. Els servidors existeixen i continuen sent-hi, però són abstrets de l'aplicació per tal que no hagis de preocupar-te durant el desenvolupament. Els proveïdors de sistemes cloud controlen el treball del servidor, és a dir, gestionen els recursos i ofereixen un escalat en funció de les necessitats de l'aplicació. Un cop llançada, les aplicacions serverless responen automàticament a la demanda i escalen en funció de les seves necessitats**[9]**.

7.2.2.1.1 FaaS

Els fonaments de Faas o Function as a Service són executar el codi del back-end sense necessitat de gestionar els servidors. Tampoc has de programar amb un llenguatge de programació o framework específic. Per al que les funcions que s'implementen poden haver estat programades amb Javascript, python, Go, qualsevol llenguatge JVM o

qualsevol llenguatge .NET. És un model d'execució que es basa en esdeveniments i s'executa en contenidors sense estat. Les funcions són les que gestionen la lògica i l'estat del servidor a través de l'ús de serveis d'un proveïdor de FaaS (AWS Lambda). També es pot implementar de forma local**[10][11]**.

Els avantatges de FaaS són:

- Increment de la productivitat dels desenvolupadors i més facilitat amb la implementació.
- No tens la necessitat de preocupar-te per la gestió dels servidors.
- És fàcil d'ampliar i la mateixa plataforma gestiona la incorporació d'equips.
- Només s'utilitzen els recursos quan és necessari i es paga només per al seu ús.
- Les funcions es poden escriure gairebé en tots els llenguatges de programació

7.2.2.1.2 BaaS

BaaS o com les seves sigles indiquen Backend as a Service és un model que proporciona als desenvolupadors web i d'aplicacions mòbils una forma de vincular les aplicacions a l'emmagatzematge al núvol. En el nostre cas fem ús d'aquest servei per la gestió d'usuaris**[12]**, concretament fem servir l'autenticació d'usuaris amb Google OAuth 2.0.

7.2.2.1.3 Arquitectura serverless

A diferència d'altres models de computació al núvol, en un model serverless és el proveïdor el responsable de controlar tant com la infraestructura cloud com l'escalabilitat de les aplicacions. Les aplicacions serverless són desplegades en contenidors que s'inicien automàticament quan se sol·liciten.
Si pensem amb els sistemes tradicionals three-tier client-oriented system amb la lògica server-side. La nostra aplicació es veuria com una cosa semblant al diagrama de sota (veure *Figura 14*).

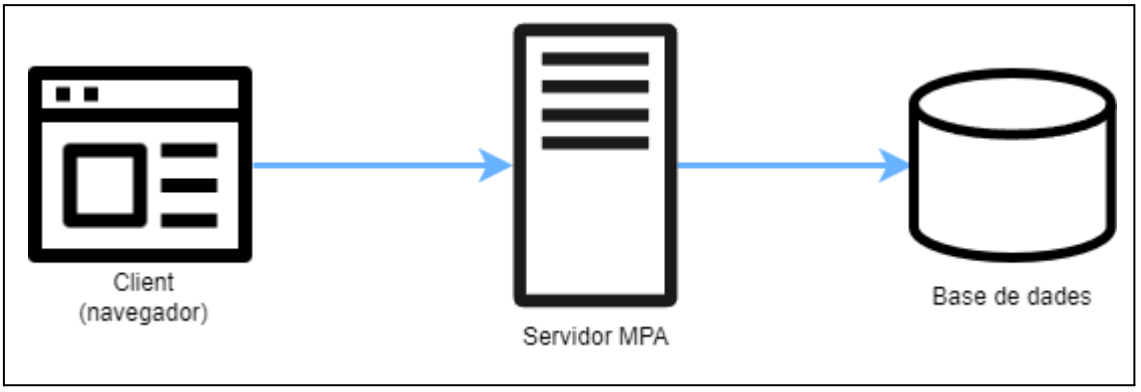


Figura 14. Estructura tradicional aplicació web.

Amb aquesta arquitectura el client pot ser relativament poc “intel·ligent” on gran part de la lògica del sistema (autentificació, navegació, cerca…) estan implementades en el servidor.

Si observem el diagrama de sota podem veure com l’aplicació es veuria més o menys amb l’arquitectura serverless (veure *Figura 15*).

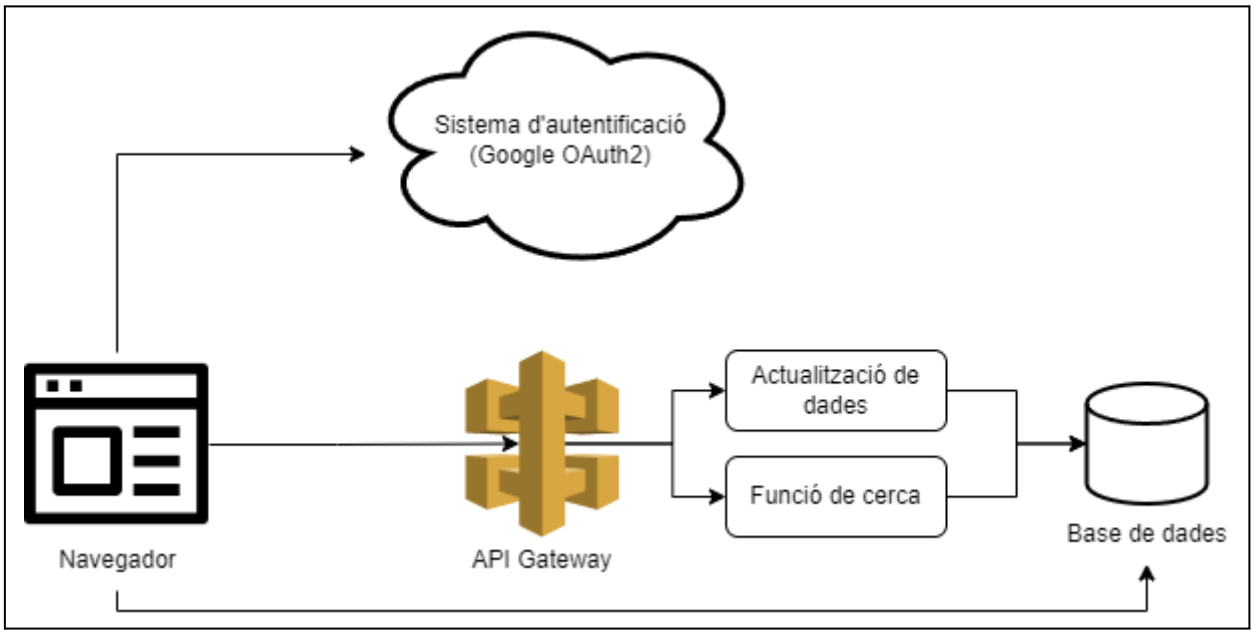


Figura 15. Diagrama simplificat arquitectura serverless.

El diagrama situat a sobre és una vista simplificada de l'arquitectura serverless. Es poden observar una sèrie de diferències respecte a l'arquitectura tradicional. Primerament, es pot veure com tot el sistema d'autenticació ha estat substituït per un servei de tercers BaaS (Backup as a Service), en el nostre cas amb Google OAuth2. Un

altre exemple de BaaS és que ara el client té l'opció d'obtenir dades de la base de dades. La base de dades està allotjada a un sistema de tercers, en el nostre cas AWS (Amazon Web Services). Aquests punts mencionats anteriorment també impliquen a un tercer. Part de la lògica que en l'arquitectura tradicional es troba en el servidor ara es troba en el client (navegador). Ara hem de tenir un control en les sessions dels usuaris, llegir de la base de dades i transformar les dades per tal que siguin visibles, etc.

Per altra banda, pot ser que algunes funcionalitats que requereixen el processament de grans quantitats de dades estiguin en el servidor per tal de millor l'experiència de l'usuari. Un exemple en la nostra aplicació és la cerca de Chapters o Case Studies. En comptes de tenir un servidor que sempre està en marxa podem tenir un FaaS (Function as a Service) que respongui a les peticions HTTP a través de l'API gateway (GET, PATCH, DELETE, etc.). Finalment, podem observar que a través de l'API podem tenir una sèrie de funcions que estan en el servidor per diverses raons, una d'elles podria ser per temes de seguretat. Aquestes funcions consisteixen en l'actualització de les dades (CMS) a les quals es pot accedir a elles a través de l'API, per tant, utilitzant FaaS**[10]**.

7.2.2.2 Sveltekit

SvelteKit és un framework dissenyat per construir aplicacions web d'alt rendiment. Al moment de desenvolupar una aplicació hi ha un conjunt de pràctiques per millorar el rendiment que poden resultar ser complicades. Amb SvelteKit no cal que t'en preocupis, ja que internament ho controla tot per oferir el millor rendiment**[13]**.

7.2.2.2.1 Routing

Aquest Framework és un filesystem-based-router. Això significa que l'estructura de l'aplicació és definida per l'estructura del codi. Hi ha dos tipus de routes: pages i endpoints.

Les pages normalment generen el codi HTML, CSS i JS per mostrar a l'usuari. Aquestes pàgines per defecte són renderitzades tant a client com a servidor, però aquest comportament pot ser configurat.
Per altra banda, els endpoints només s'executen en el servidor, per tant, és el lloc on s'accedeix a la base de dades o a l'API. Les pages poden demanar dades a través dels

endpoints. Els endpoints retornen les dades en format JSON, tot i que també poden retornar les dades en altres formats**[14]**.

7.2.2.2.1.1 Pages

Les pages són components de Svelte escrits en fitxers amb extensió .svelte. Per defecte quan un usuari visita l'aplicació per primera vegada l'hi serà entregada una versió de la pàgina renderitzada per al servidor més codi JavaScript que "hidrata" la pàgina i inicialitza l'encaminament en el client. A partir d'aquest punt tota la navegació a les altres pàgines és gestionada per al client donant a lloc una sensació semblant al d'una aplicació on les parts comunes del disseny no necessiten ser renderitzades una altra vegada.

Els noms de fitxer determinen la ruta, anem a veure un exemple:

Dins de l'estructura bàsica d'un projecte amb SvelteKit hi ha un subdirectori (src/routes) on tot el que hi hagi dins d'ell representarà l'estructura de la pàgina web. Si tenim, aquests directoris i fitxers**[14]**:

- src
    - routes
        - index.svelte **(1)**
        - pagina1.svelte **(2)**
        - user
            - index.svelte **(3)**

Si accedim al navegador a l'URL domini.com el contingut que es mostrarà és el del fitxer index.svelte **(1)**, ja que aquest està situat a l'arrel del directori routes. En canvi, si accedim a domini.com/user el contingut que aquest mostrarà és el del fitxer index.svelte **(3)**. No és obligatori que el fitxer s'anomeni index.svelte, quan s'utilitza aquest nom de fitxer implícitament estem indicant que fa referència a la ruta del directori on es troba. També podem tenir fitxers amb altres noms com és el cas de pagina1.svelte, per accedir a ell, si observem la seva ubicació podem veure que està a l'arrel, per tant, per accedir a ell podem fer servir el seu nom de fitxer sense extensió en el navegador. En aquest cas domini.com/pagina1.

Aquest sistema de rutes que ofereix SvelteKit ens dona un conjunt d'avantatges respecte al SEO, ja que amb una aplicació de Svelte l'encaminament es veu afectat negativament perquè no ofereix cap sistema d'encaminament.

7.2.2.2.2 Endpoints

Els endpoints són mòduls escrits en JavaScript o TypeScript que exporten les funcions de control de les sol·licituds dels mètodes HTTP (GET, POST, DELETE...). Aquests controladors fan possible llegir i escriure dades que només estan disponibles en el servidor (DB o sistema de fitxers).

La seva feina és retornar un objecte ({ status, headers, body }) que representa la resposta**[14]**.

- status és un codi d'estat HTTP (2xx, 3xx, 4xx, etc).
- headers pot ser un objecte o una instància de la classe Headers.
- body pot ser un objecte o si alguna cosa va malament un Error.

7.2.2.2.2.1 Page endpoints

Quan un endpoint té el mateix nom de fitxer que la pàgina (excepte l'extensió), la pàgina obté les propietats de l'endpoint a través del fetch durant la navegació en el client o a través d'una crida a una funció directa durant el SSR (Server Side Rendering)**[14]**.

Per exemple, si tenim el subdirectori src/routes on a dins hi ha els fitxers [slug].svelte i [slug].ts podem veure on els seus noms coincideixen excepte la seva extensió, per tant, el fitxer [slug].ts actuarà com a endpoint de la pàgina [slug].svelte. Mirar *Figura 16 i 17*.

```
import type { RequestHandler } from "@sveltejs/kit";
import { getPageComponentProps } from "$lib/prisma/wrappers";

export const get: RequestHandler<{slug: string}> = async ({ locals, params: {
slug } }) => {
  const pageProps = await getPageComponentProps(slug, false);

  if ('error' in pageProps.body) return pageProps;
```

```
  const pageId = pageProps.body.page.id;
  const pageTags = pageProps.body.page.tags;

  locals.cacheKeys.add(`page-${pageId}`);
  pageTags.forEach(tag => locals.cacheKeys.add(`tag-${tag.tag.id}`));

  return pageProps;
};
```

Figura 16. Fitxer de l'endpoint [slug].ts

```
<script lang="ts">
  import type { SubTypes } from "$lib/types";
  import Page from "$lib/components/pages/Page.svelte";

  export let page: SubTypes.Page.Full;
  export let recommendedPages: SubTypes.Page.ContentCard[];

</script>

<Page {page} {recommendedPages} />
```

Figura 17. Fitxer de la pàgina [slug].svelte

Els fitxers de la *Figura 16 i 17* estan emparellats, si ens fixem tenen el mateix nom i estan ubicats en el mateix directori (src/routes). En el fitxer [slug].ts definim la funció get. Aquesta retorna pageProps que és el resultat d'executar la funció getPageComponentsProps(). El contingut retornat correspon a un objecte com el que s'ha mencionat en el punt 7.2.2.2.2 Endpoints. Dins de la propietat body hi ha 2 objectes que són page i recommendedPages que aquest són els que seran compartits amb la pàgina slug.svelte corresponent a la *Figura 17*.

SvelteKit a part de les característiques explicades anteriorment en té moltes altres que explicaré més endavant quan es requereixi el seu ús. He considerat que aquestes són les més interessants per tal de fer una introducció i poder donar una visió de com aquest framework es comporta.

7.2.2.3 Vectors de cerca

Els motors de cerca tradicionals estan limitats a la coincidència de les paraules clau i no tenen en compte el context. Una sola paraula pot tenir molts significats, per exemple si busques la paraula "gat" podria tenir diversos significats associats. Podria o bé ser l'animal gat o bé l'eina que s'utilitza per a la reparació de les rodes dels cotxes.

Aquest problema es pot solucionar amb els vectors de cerca. En la cerca vectorial, el text es codifica de forma que els textos semànticament semblants estan a prop uns dels altres dins de l'espai vectorial, mentre que els textos diferents estan molt separats. Per tal de codificar conjunts de dades en representacions vectorials la cerca vectorial fa servir models d'aprenentatge profund (deep learning models).

Els vectors de cerca ens permeten representar objectes desordenats en un espai vectorial d'altres dimensions com a vectors. Comparant les distàncies i semblances entre els vectors es pot trobar el contingut relacionat.

Per tal de fer la cerca en comptes de comparar cada vector un per un amb algoritmes com el KNN (K-Nearest Neighbor) que pot fer que el temps de cerca sigui molt elevat, els motors de cerca fan servir l'algorisme ANN (Approximate Nearest Neighbor). Alguns algorismes ANN divideixen l'espai del vector en diversos grups, assignant etiquetes a cada grup, i busquen només per les etiquetes assignades al grup. Aquest mètode millora el temps de consultes substancialment. Alguns dels avantatges de la cerca per vectors són**[15]**:

- Resultats més rellevants.
- Control d’errors.
- Bons resultats en els sistemes de recomanació.
- Models entrenats fàcilment disponibles.
- Els models entrenats treballen eficientment.

7.2.2.4 Gestió de la cache amb CDNs

La caché és una ubicació que guarda les dades temporalment per tal d'aconseguir una obtenció de les dades més ràpida. Per tal d'entendre com funciona el caching, necessitem entendre com funciona les peticions i respostes HTTP. Quan un usuari envia una petició d'una URL a través del navegador (client) al servidor on està allotjat el contingut. Posteriorment el servidor processa la petició i envia la resposta al client (veure *Figura 18*).

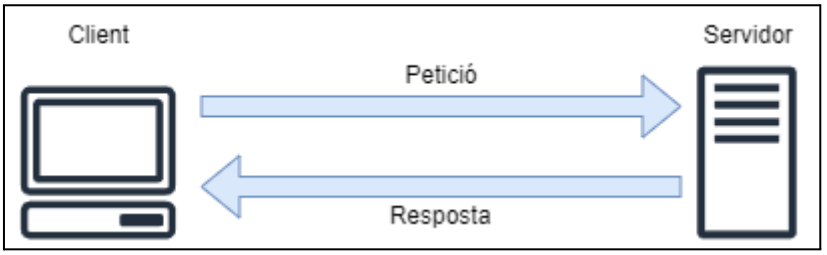


Figura 18. Peticions HTTP entre client i servidor.

Tot i que sembla que les peticions siguin instantànies, realment comporta cert temps processar la petició en el servidor, generar la resposta i enviar la resposta. A més a més algunes respostes requereixen que es renderitzi el contingut en la web.

En aquest procés és quan la caché local pren importància. Els navegadors poden cachejar alguns recursos estàtics, com imatges, logos, els estils de la pàgina, etc. en l'ordinador personal de l'usuari per tal de millorar el rendiment la pròxima vegada que s'accedeixi en el lloc web. Tot i això, aquest tipus de caché és només útil per a un usuari.

Per tal de cachejar recursos per tots els usuaris que visiten la pàgina web, pot ser útil utilitzar un sistema de caché remot com un reverse proxy. Un reverse proxy és una aplicació que es situa entre el client i el servidor i respon les peticions en nom teu. Aquest reverse proxy pot ser instal·lat en el mateix servidor on està allotjada la web o a un servidor diferent. La funció del reverse proxy serà cachejar el contingut de l'origen per tal de poder respondre les peticions des de la caché (veure *Figura 19*).

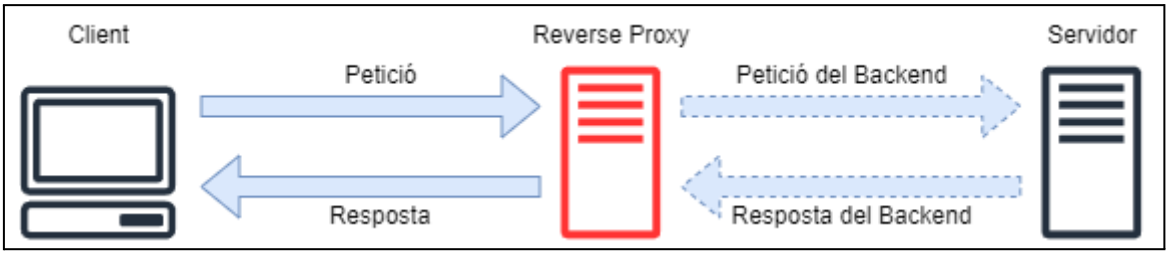


Figura 19. Peticions HTTP amb un reverse proxy.

Les capçaleres HTTP s'utilitzen per passar informació entre el client i el servidor origen, tant en la petició com en la resposta. Es poden definir certes capçaleres HTTP en la resposta per tal de controlar quin contingut està cachejat i per quant de temps. El reverse proxy actua com un intermediari, modificant algunes capçaleres enviades per al servidor d'origen i afegint d'altres que s'envien al client. El propòsit és la gestió de com els objectes estan cachejats. Els beneficis d'utilitzar un reverse proxy són**[16]**:

- Entrega de les peticions al moment sense necessitat d'esperar que siguin generades.
- El contingut que ja està cachejat allibera de feina extra en el servidor origen.
- Incrementa l'experiència de l'usuari, ja que aconsegueixes més rapidesa.
- Permet comprimir les dades HTTP abans d'enviar una resposta, fent que l'entrega sigui més ràpida.
- Balanceja la demanda del contingut.
- Prevenció d'onades de peticions en el servidor origen.
- Si el servidor origen deixa de donar servei, es pot continuar entregant els recursos que estan cachejats.

Ara hem vist com funciona un reverse proxy i tots els beneficis que tenia. Es pot incrementar l'eficiència dels reverse proxy si a més a més fem ús d'una CDN (Content Delivery Network). Una CDN és una xarxa de servidors distribuïts a través del món que treballen conjuntament per entregar contingut als usuaris de forma més ràpida i eficient.

La ubicació dels usuaris és una variable important quan parlem de la rapidesa i el temps d'entrega de les peticions, si tenim el servidor origen ubicat a Espanya el temps de resposta no és el mateix per un usuari ubicat a França que per un altre ubicat al Japó.

En fer servir una CDN podem millorar el temps de resposta per als usuaris que estan ubicats a una distància considerable del servidor origen. Això s'aconsegueix guardant còpies temporals en ubicacions properes als usuaris. Els serveis de CDN tenen serveis de caché al voltant del món, per tant, quan els usuaris fan una petició, la CDN entrega el contingut cachejat des del servidor que està ubicant més a prop de l'usuari. Això fa que el temps d'entrega sigui reduït i conseqüentment la pàgina web carregui molt més ràpid (veure *Figura 20*).

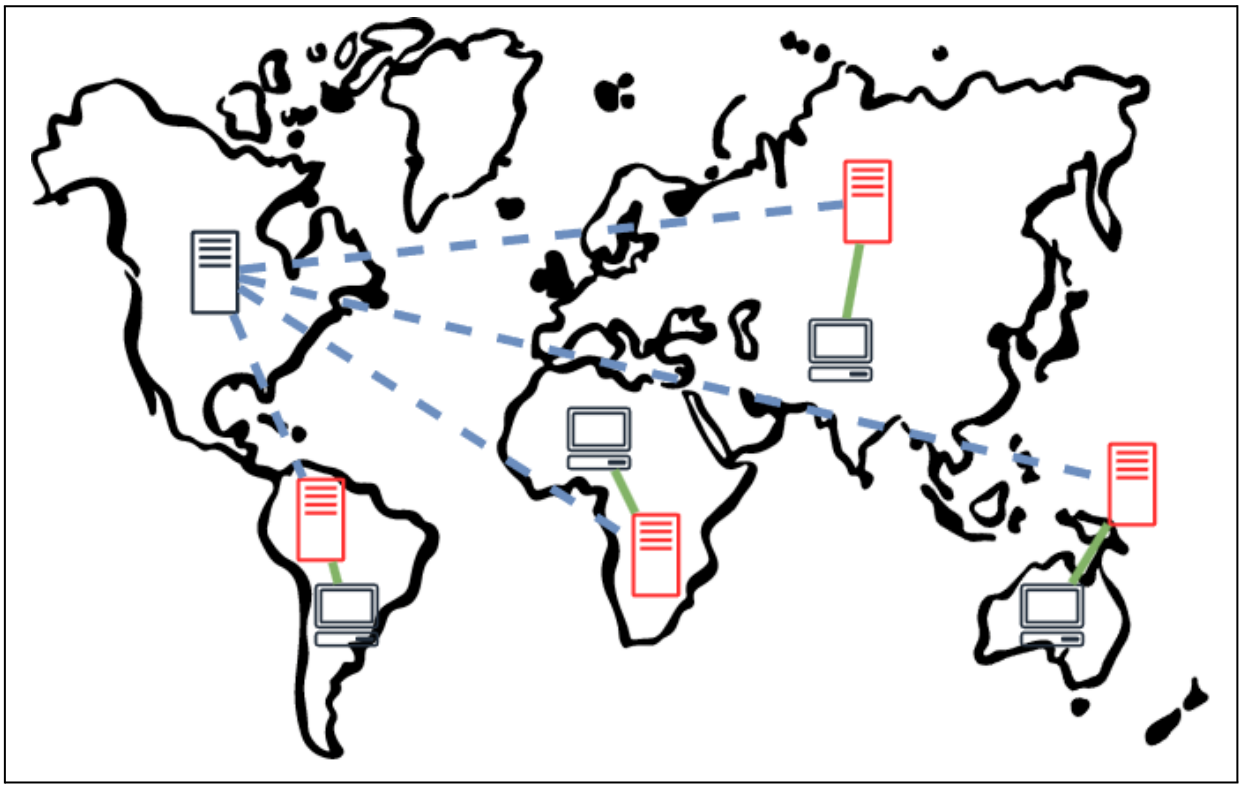


Figura 20. Representació d'una CDN.

Els avantatges d'utilitzar una CDN són les següents**[17]**:

- Menor latència.
- Disponibilitat: Es pot continuar servint contingut encara que el servidor origen no estigui disponible.
- Escalabilitat: Permet que la CDN escali automàticament quan major és la demanda.
- Seguretat: Permet prevenir atacs DDoS, ja que la CDN pot controlar la càrrega de tràfic i ajuda al bloqueig de l'atac i continuar amb l'encaminament del tràfic vàlid.

- Reducció de costos en el servidor origen: En fer ús d'una CDN implica que el tràfic del teu servidor origen es veurà reduït per al que els costos de sortida disminuiran.

8. Requisits del sistema

En aquest punt explicaré tots els requisits del sistema, tant funcionals com no funcionals, això també permet donar una visió del comportament i de les funcionalitats que el sistema ofereix als usuaris de l'aplicació.

8.1 Requisits funcionals

8.1.1 CMS

- L'usuari s'ha de poder autentificar.
- L'usuari s'ha de poder registrar.
- L'usuari s'ha de poder desconnectar.
- Crear/Modificar/Eliminar un Chapter: A través del CMS el gestor de contingut ha de ser capaç de crear/modificar/eliminar un Chapter.
    - Afegir/Eliminar Tags associades a una Chapter: Quan s'edita un Chapter s'han de poder afegir i eliminar les tags associades a ells, dins de les quals es poden distingir 3 categories:
        - Afegir/Eliminar tags tipus *stage*.
        - Afegir/Eliminar tags tipus topic.
        - Afegir/Eliminar tags tipus *user*.
    - Modificar el títol del Chapter.
    - Afegir/Eliminar autors a un Chapter: Cada Chapter pot tenir un o diversos autors.
    - Modificar slug al Chapter: S'ha de poder triar quina és la slug (nom de l'URL) del Chapter.
    - Modificar la descripció del Chapter.
    - Modificar la imatge del Chapter.
    - Afegir/Eliminar Cards en la secció Key takeaways.
    - Editar contingut del Chapter:

        - Afegir/Modificar/Eliminar cards dins del contingut del Chapter.
        - Afegir/Modificar/Eliminar títols dins del contingut del Chapter.
        - Afegir/Modificar/Eliminar paràgrafs dins del contingut del Chapter.
        - Afegir/Modificar/Eliminar diagrama dins del contingut del Chapter.
    - Previsualitzar Chapter.

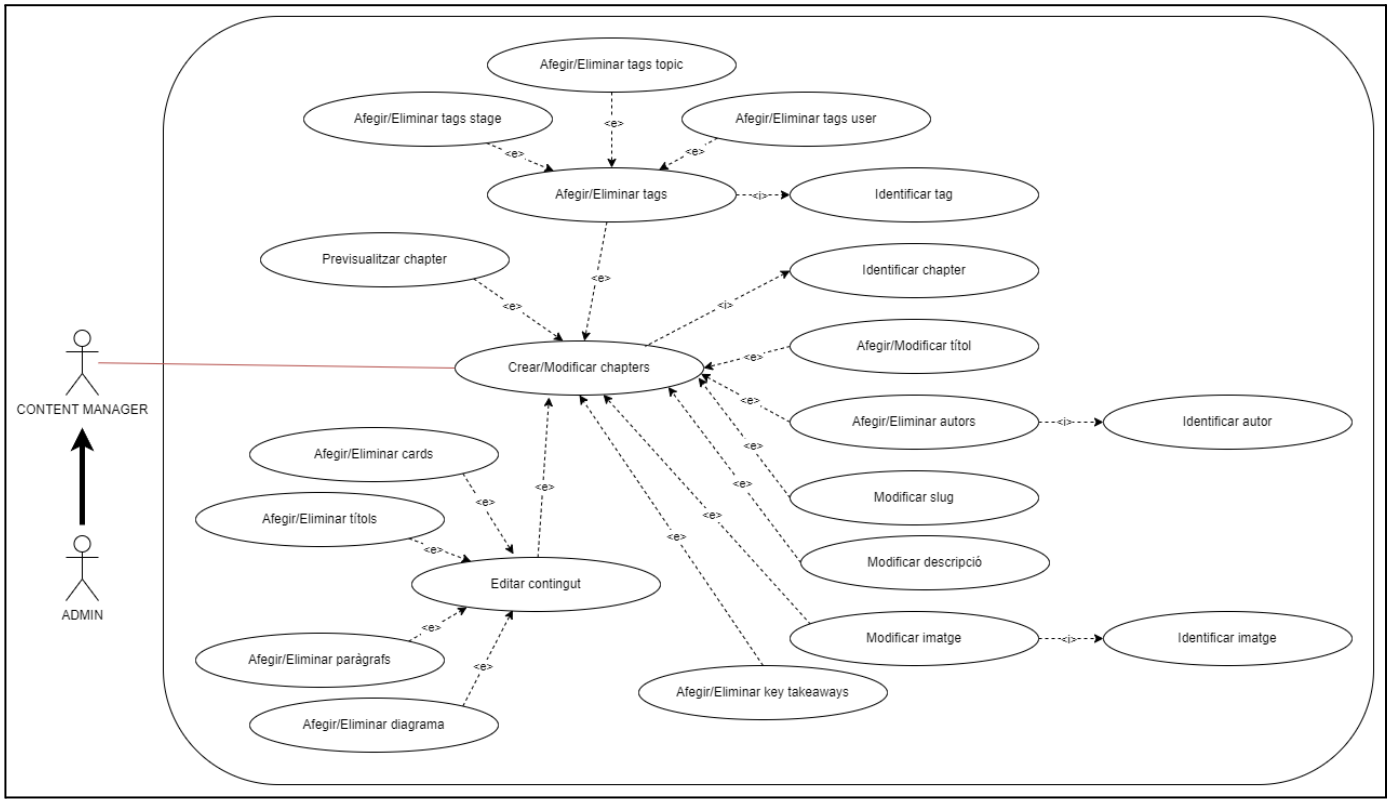


Figura 21. Diagrama casos d'ús crear/modificar un Chapter.

- Crear/Modificar/Eliminar un Case Study: A través del CMS el gestor de contingut ha de ser capaç de crear/modificar/eliminar un Case Study.
    - Afegir/Eliminar Tags associades a un Case Study: Quan s'edita un Case Study, s'han de poder afegir i eliminar les tags associades a ells, dins de les quals es poden distingir 3 categories:
        - Afegir/Eliminar tags tipus *stage*.
        - Afegir/Eliminar tags tipus *topic*.
        - Afegir/Eliminar tags tipus *user*.
    - Modificar el títol del Case Study.

- Modificar slug al Case Study: S'ha de poder triar quina és la slug (nom de l'URL) del Case Study.
- Modificar la imatge del Case Study.
- Afegir/Eliminar Cards en la card principal.
- Editar contingut del Case Study:
  - Afegir/Modificar/Eliminar cards dins del contingut del Case Study.
  - Afegir/Modificar/Eliminar títols dins del contingut del Case Study.
  - Afegir/Modificar/Eliminar paràgrafs dins del contingut del Case Study.
  - Afegir/Modificar/Eliminar diagrama dins del contingut del Case Study.
- Editar contingut de les Milestones:
  - Afegir/Modificar/Eliminar any.
  - Afegir/Modificar/Eliminar milestone.
- Modificar el GlobeViz.
- Previsualitzar Case Study.

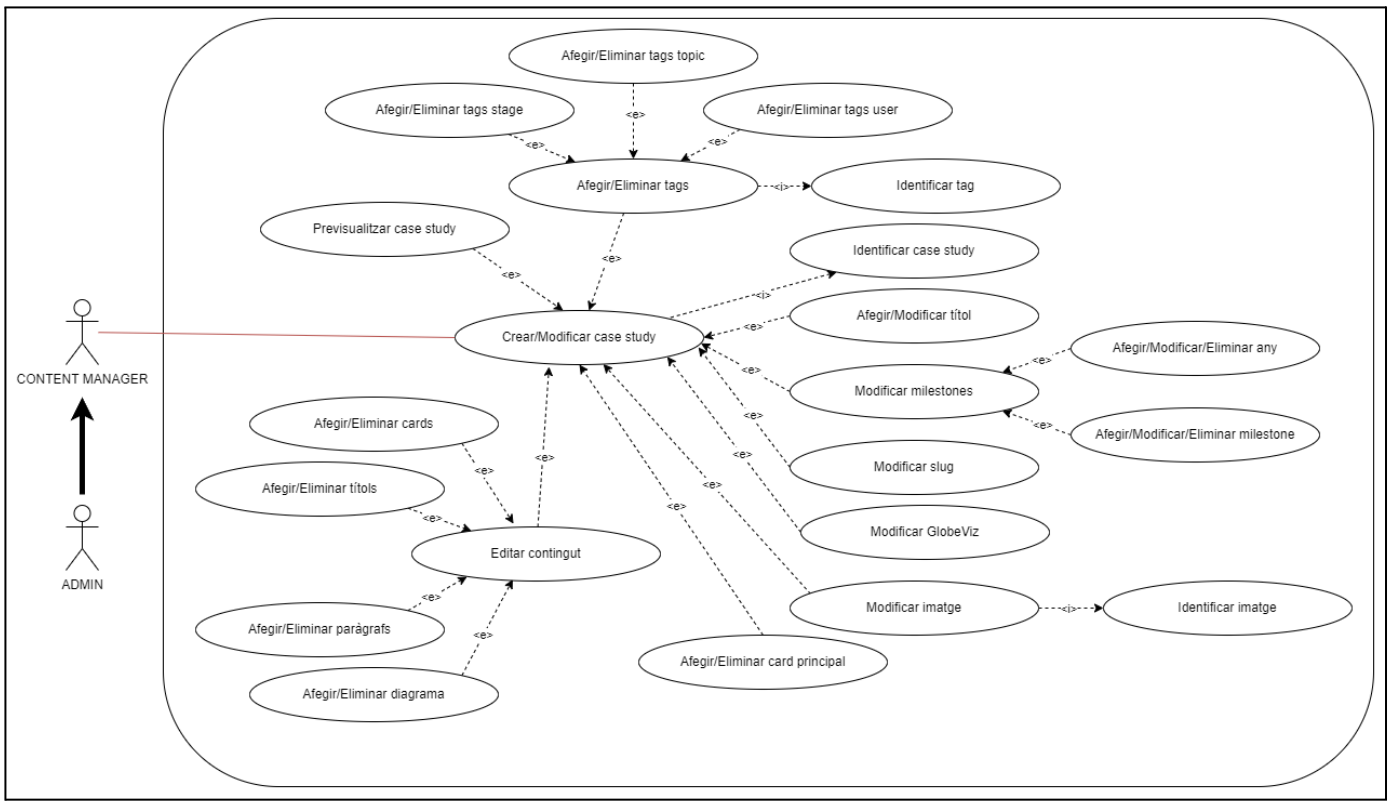


Figura 22. Diagrama casos d'ús crear/modificar un Case Study.

- Filtrar contingut de les pages:
    - Cerca per títol.
    - Cerca per tags.
- Afegir/Modificar/Eliminar tags del tipus topic.

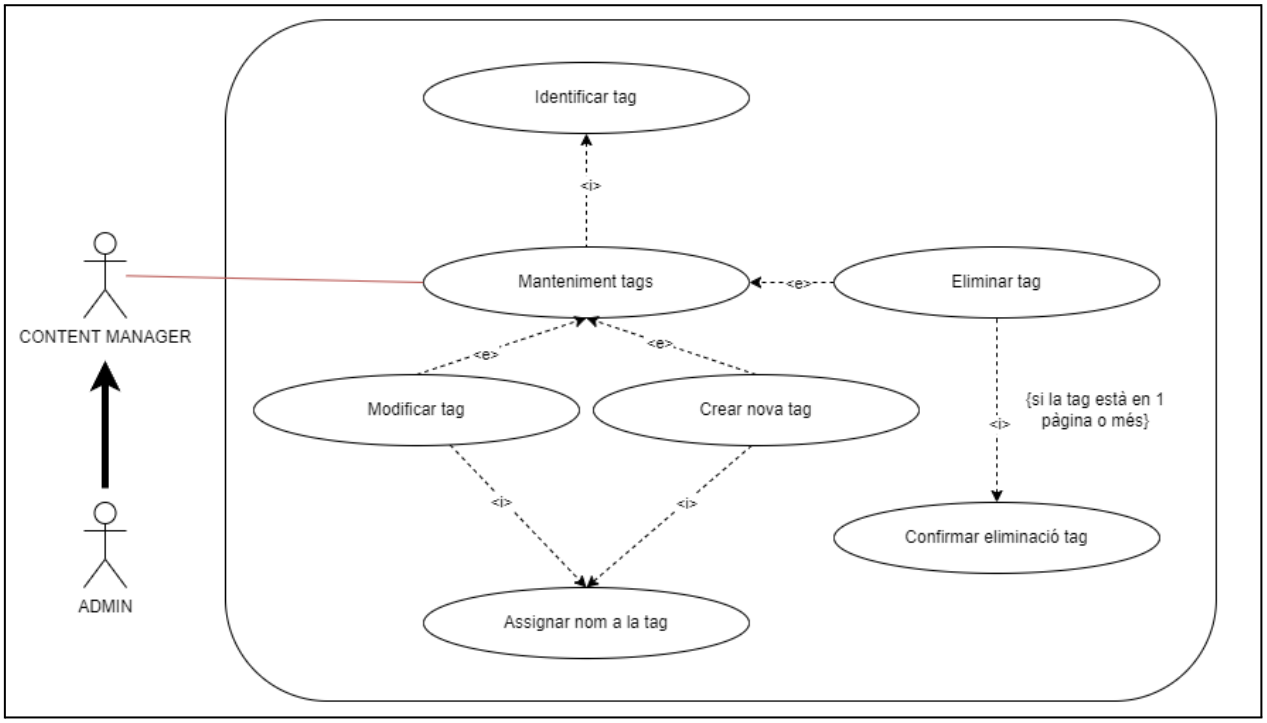


Figura 23. Diagrama casos d'ús manteniment tags.

- Afegir/Modificar/Eliminar autors:
    - Canviar imatge de l'autor.
    - Canviar nom de l'autor.
    - Canviar biografia de l'autor.
    - Cercar autors per nom d'autor.

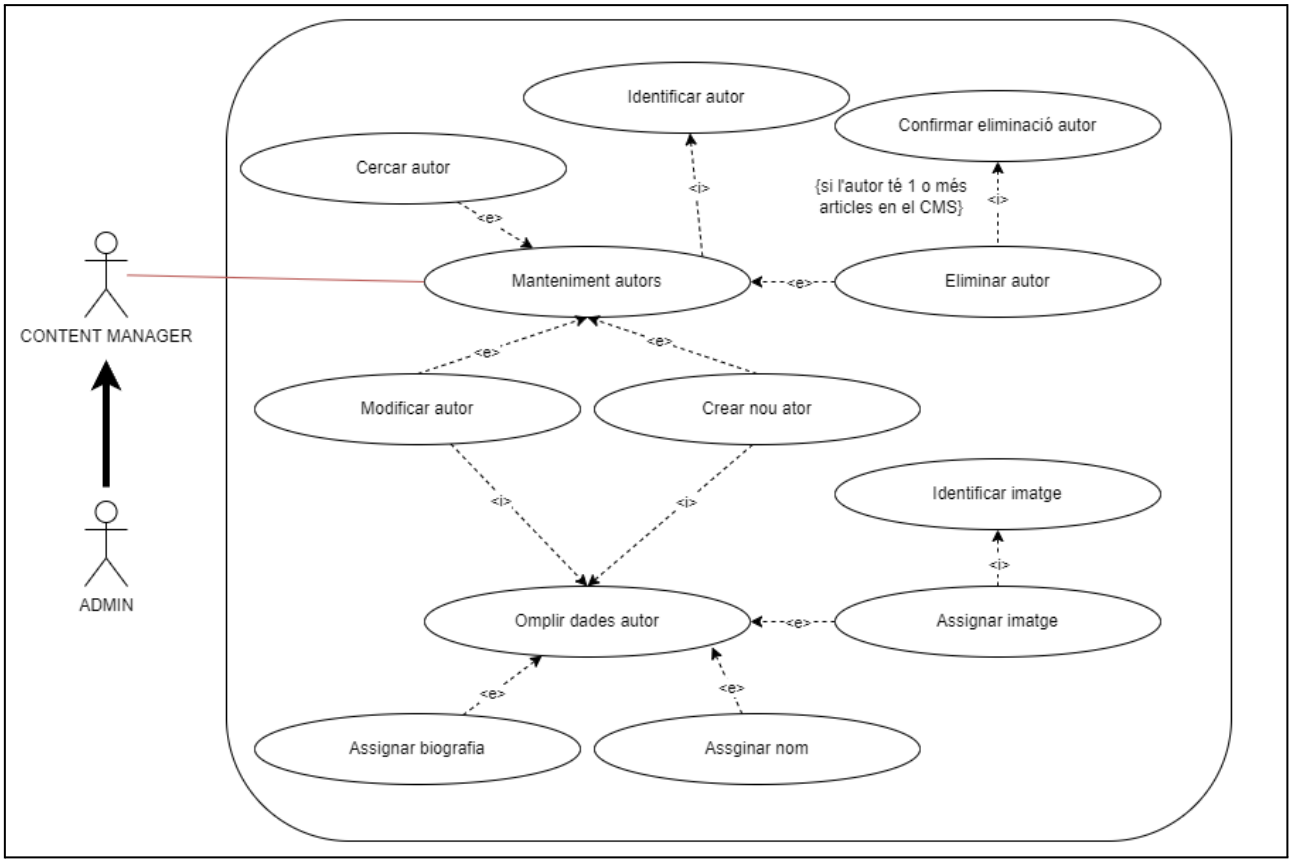


Figura 24. Diagrama casos d'ús manteniment de autors.

- Modificar/Eliminar usuaris: Els administradors del lloc han de poder modificar i eliminar els usuaris.
    - Canviar el nom de l'usuari.
    - Canviar el rol de l'usuari.

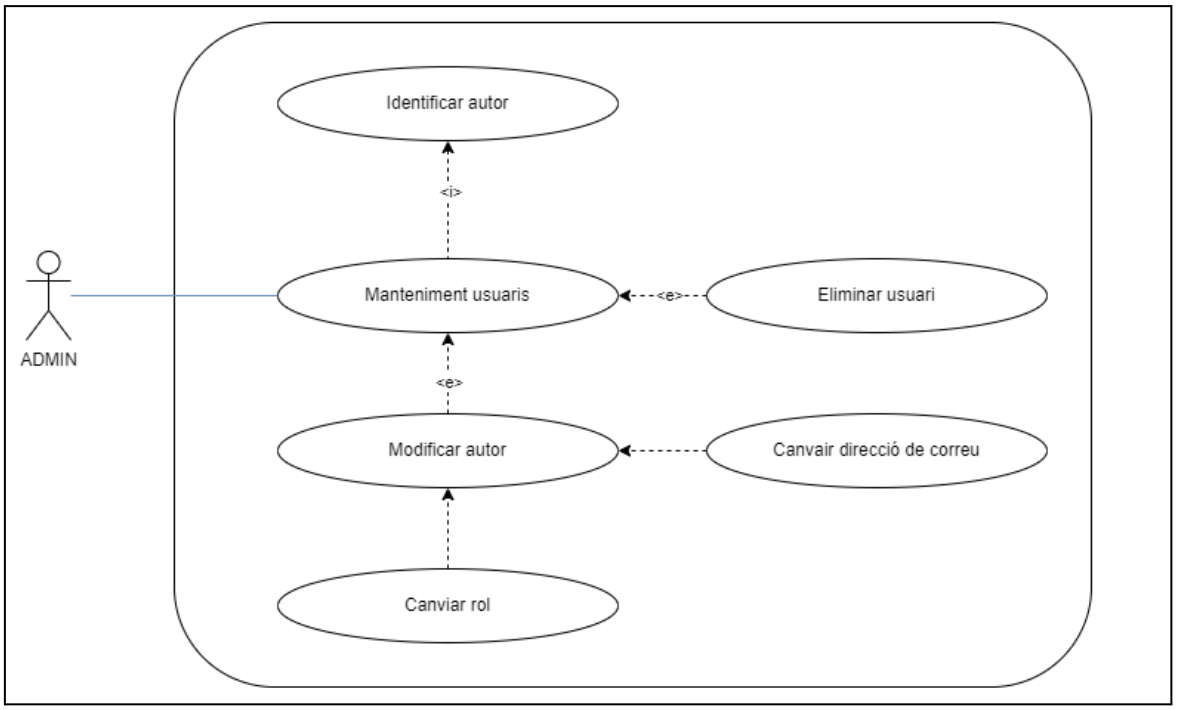


Figura 25. Diagrama casos d'ús manteniment usuaris.

- Modificar l'ordre de visualització dels components de la landing page: Els administradors han de poder escollir quin son els components que es situen al principi de la pàgina principal.

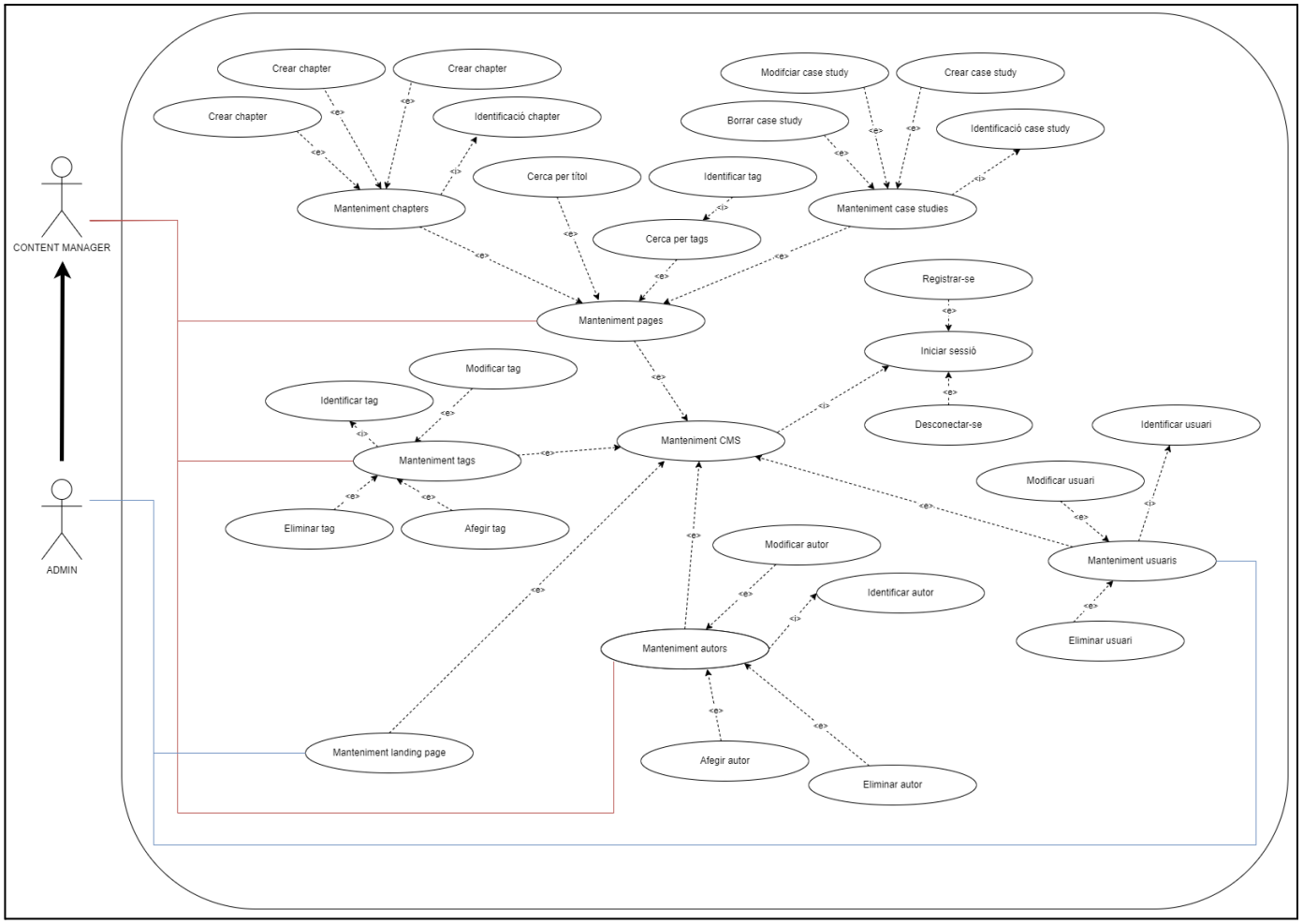


Figura 26. Diagrama de casos d'ús manteniment del CMS.

8.1.2 Front-end

8.1.2.1 Landing page

- Navegació entre Chapters recomanats.
- Navegació entre el Case Studies recomanats.
- Cercar contingut.
    - Cercar per nom d'autor.
    - Cercar per tag.
    - Cercar per context.
- Navegació en el madlib
    - Cercar per tipus d'usuari.
- Navegació informativa d'on se situa en el cicle de vida d'una MPA.

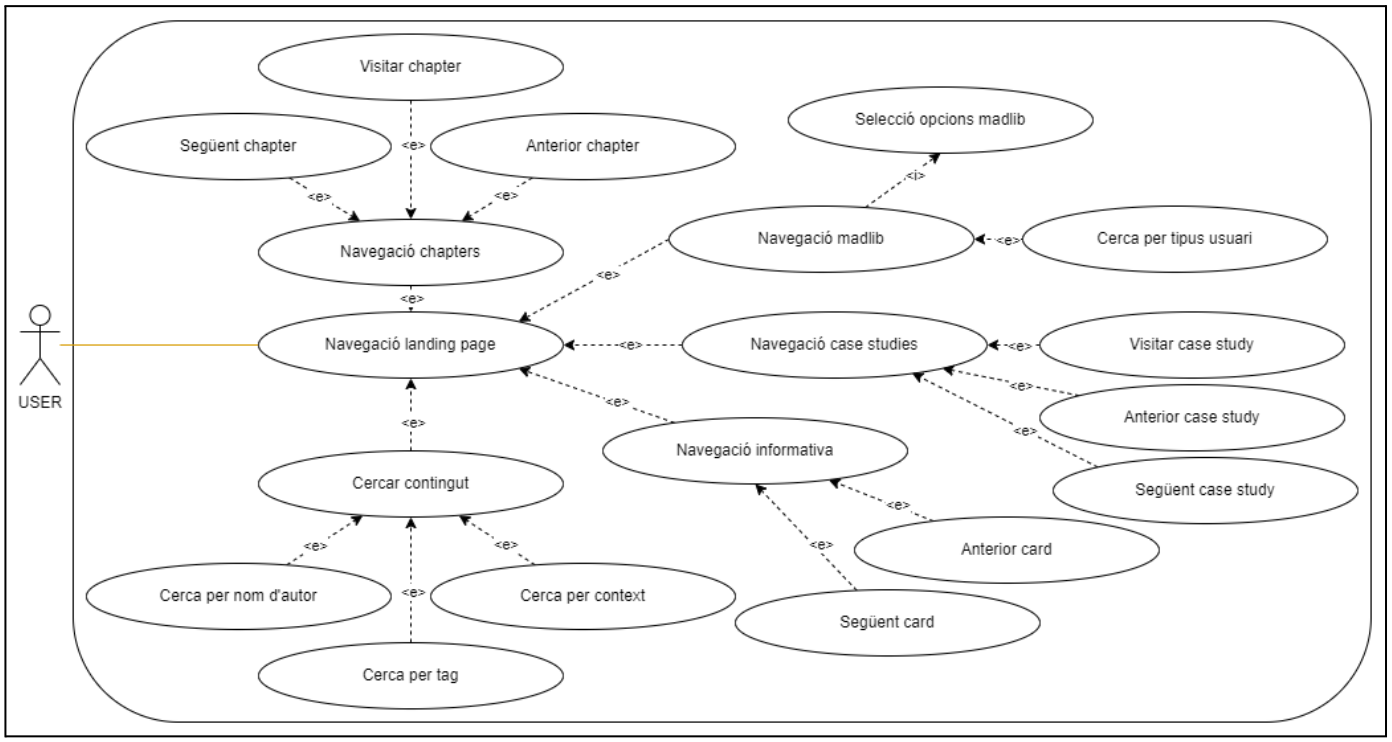


Figura 27. Diagrama casos d'ús landing page.

8.1.2.2 Chapters

- Navegació a través de les key takeaways.
- Navegació a través del menú.
- Navegació a través del lifecycle.
- Navegació a través de les cards.
- Navegació a través del madlib.
- Navegació a través de les recomanacions.

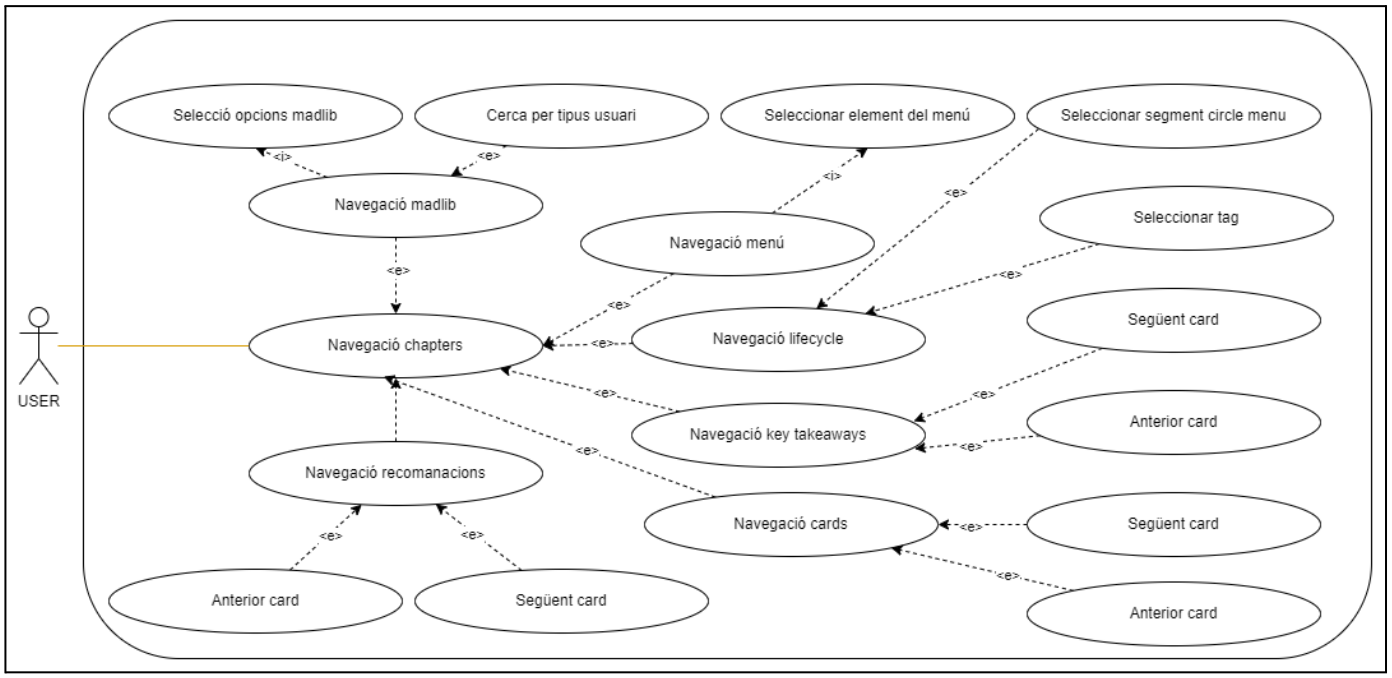


Figura 28. Diagrama casos d'ús Chapter front-end.

8.1.2.3 Case Studies

- Navegació a través de les key takeaways.
- Navegació a través del menú.
- Navegació a través del lifecycle.
- Navegació a través de les cards.
- Navegació a través del madlib.
- Navegació a través de les recomanacions.

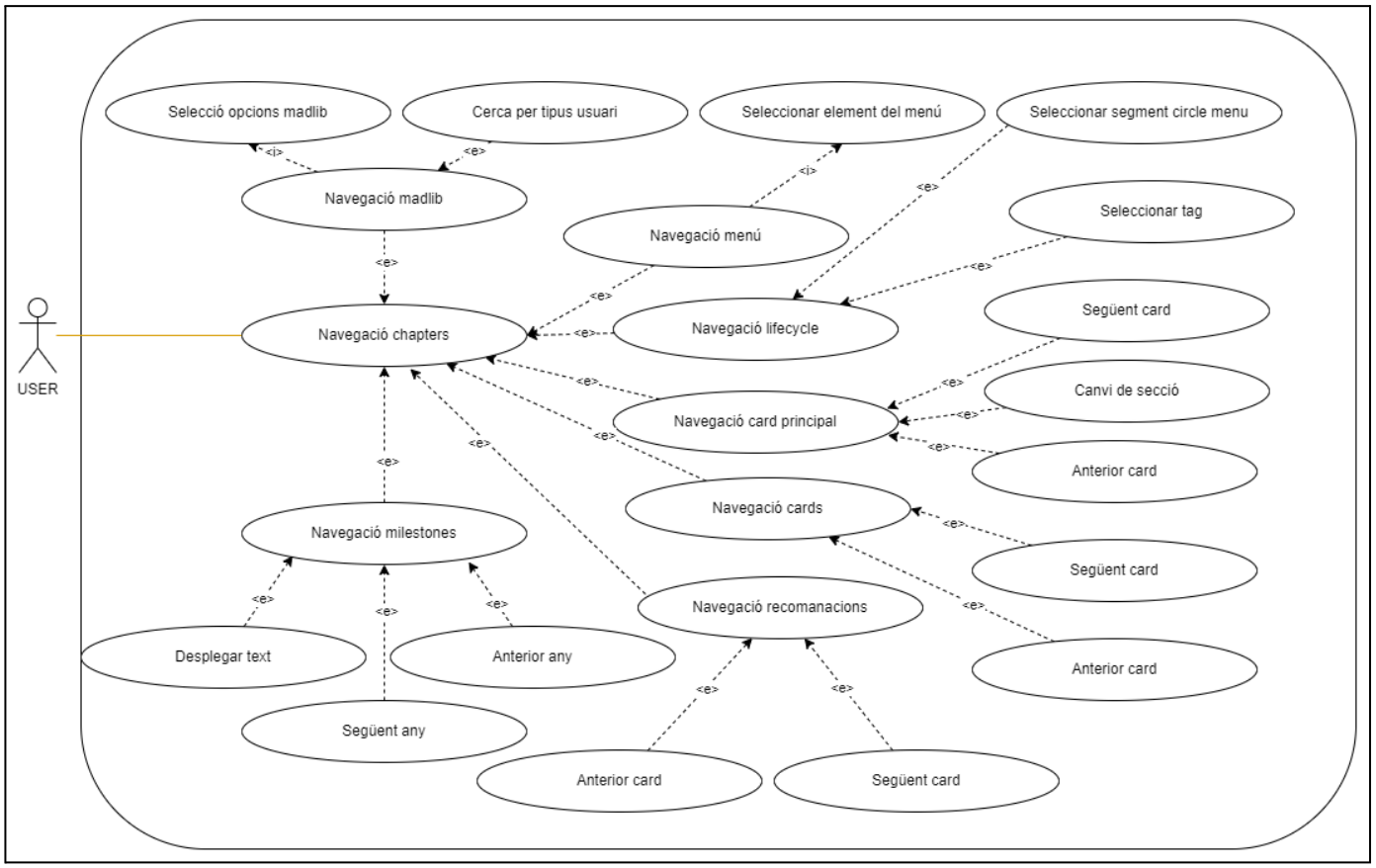


Figura 29. Diagrama casos d'ús Case Study front-end.

8.1.2.4 Collection pages

- Cercar contingut.
    - Cercar per nom d'autor.
    - Cercar per tag.
    - Cercar per context.
- Seleccionar entrada.
- Seleccionar tag.
- Seleccionar author.

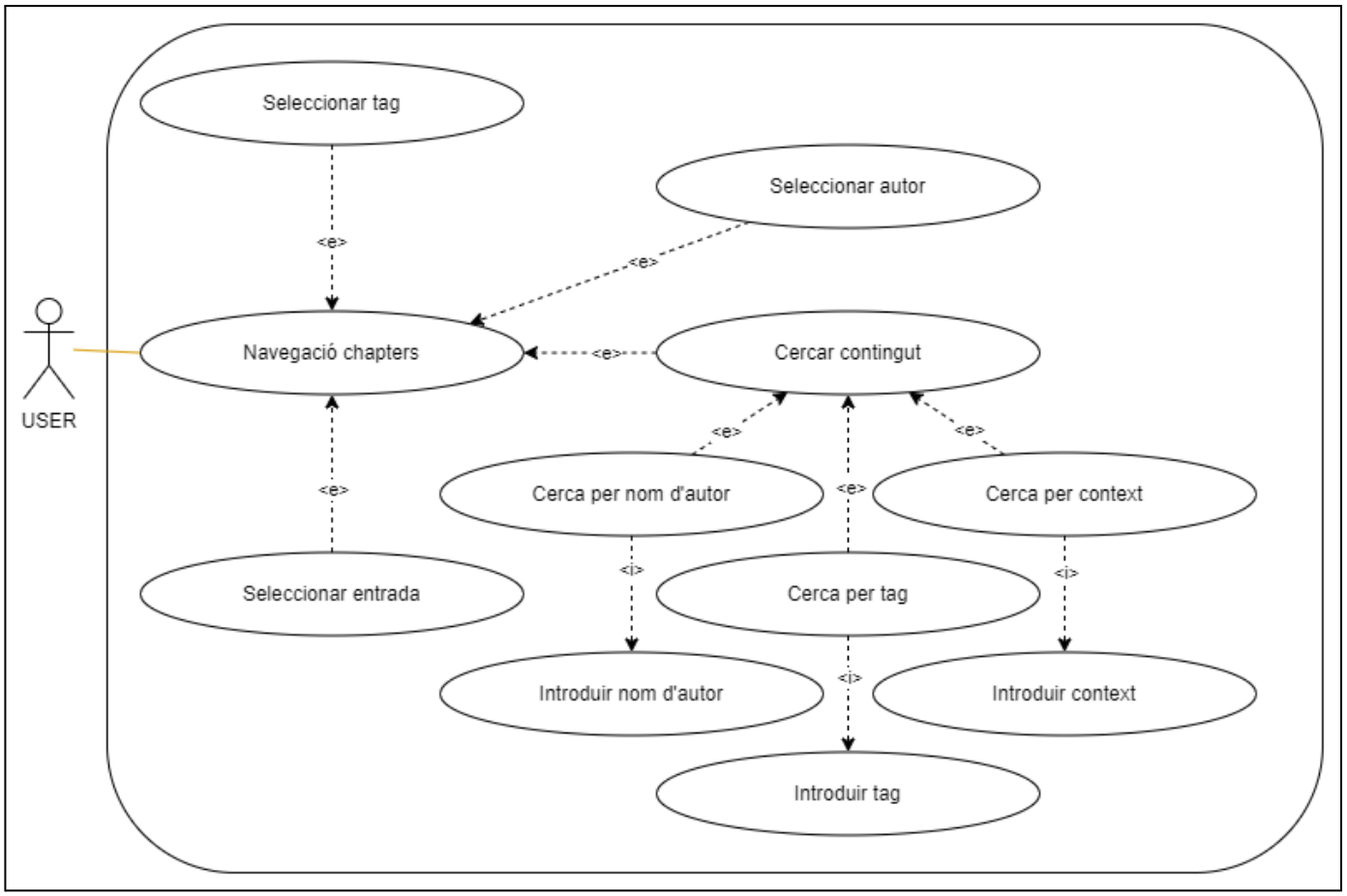


Figura 30. Diagrama casos d'ús collection pages.

8.2 Requisits no funcionals

- Rendiment
    - El temps de renderització de la pàgina ha de ser ràpid.
    - El temps d'espera en les connexions a la base de dades per l'obtenció i modificació de les dades ha de ser ràpid.
- Seguretat
    - Control de permisos en el CMS.
    - Control d'accés (autenticació via google OAuth2).
- Accessible.
    - Diferents visualitzacions en funció de la pantalla del dispositiu.
        - Disseny en dispositius mòbils.
        - Disseny d'escriptori.
    - Compatibilitat entre els principals navegadors (Google Chrome, Firefox, Safari).

    - La visualització de la pàgina ha de ser adaptada per diferents discapacitats, per tal d'oferir accessibilitat a tota classe d'usuaris.
- Manteniment: El CMS ha de proporcionar els recursos necessaris per a poder mantenir el contingut del lloc web per part dels responsables sense intervenció de canvis en el codi de programa (excepte possibles errors informàtics).

9. Estudis i decisions

Per la realització d'aquest projecte s'han fet servir un conjunt de tecnologies, és per això que en aquest apartat s'analitzaran les capacitats de cada una d'elles i es reflexionarà el perquè s'ha decidit fer-les servir.

9.1 Programari

9.1.1 Svelte

Svelte és un nou framework dedicat a la construcció de les interfícies d'usuari. A diferència d'altres frameworks com React i Vue que fan la major part del treball en el navegador, Svelte canvia part del treball fent una prèvia compilació del codi quan es construeix l'aplicació.

En comptes d'utilitzar tècniques com el virtual DOM diffing, que consisteix en la comparació del nou DOM virtual amb l'anterior quan es produeixen canvis a la pàgina. Svelte construeix codi que quirúrgicament s'actualitza quan l'estat de l'aplicació canvia.

A més Svelte permet tenir Scoped CSS, és a dir, disposem de CSS propi a cada component sense que interfereixi amb la resta, si un component modifica l'estil HTML de totes les etiquetes <h1> només es veurà afectat en el fitxer actual i no tindrà cap efecte en la resta de components. A continuació es pot veure un exemple d'aplicació (veure *Figures 30, 31 i 32*).

```
<script>
  import Component2 from './Component2.svelte';
```

```
</script>

<h1>Color Vermell</h1>
<Component2/>

<style>
  h1 {
    color: red;
  }
</style>
```

Figura 30. Component1.svelte.

```
<h1>Color per defecte</h1>
```

Figura 31. Component2.svelte.

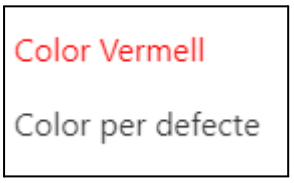


Figura 32. Resultat del Scoped CSS

Una altra propietat interessant de Svelte és la Reactivitat, és a dir, quan una variable és modificada el DOM s'actualitza automàticament amb els nous valors. A més la seva activació és eficient, amb actualitzacions granulars mitjançant l'ús de variables locals. La resta de la feina la fa el compilador.

A continuació podem observar un exemple d'un comptador amb svelte. Quan es fa clic al botó automàticament els valors s'actualitzen**[19]** (veure *Figura 33*).

```
<script>
  let contador = 0;
  $: doble = contador * 2;

  function handleClick() {
    contador += 1;
  }
</script>

<button on:click={handleClick}>
```

```
  Clicked {contador} {contador === 1 ? 'vegada' : 'vegades'}
</button>

<p>{contador} doble és {doble}</p>
```

Figura 33. Contador.svelte.

Des de que vaig entrar a la fundació s'ha utilitzat aquest framework com a element principal de treball. Tot l'equip de treball està familiaritzat amb ell i permet un desenvolupament molt més fluid.

Altres frameworks que també serien viables per la construcció de l'aplicació podrien ser React o Vue. Els quals tenen un bon reconeixement en la comunitat de desenvolupament web per les seves capacitats per a la construcció d'interfícies d'usuari**[20][21]**.

9.1.2 SvelteKit

SvelteKit és un framework per desenvolupar aplicacions web de totes les mides. Amb "filesystem-based routing". A diferència d'altres aplicacions d'una sola pàgina com podria ser Svelte. SvelteKit no es veu afectat negativament pel SEO (Search Engine Optimization), la millora progressiva o l'experiència de càrrega inicial. A diferència de les aplicacions tradicionals que són renderitzades en el servidor, la navegació és instantània**[22]**. No hi ha gaires alternatives al mercat que et permetin tenir aquest benefici en el SEO que Svelte en si no ofereix (utilitzant Svelte com a framework principal). Aquest framework és dut a terme pels propis membres de Svelte així que es pot considerar com una pròpia extensió de Svelte. Tot i això, es podria emular el sistema d'encaminament a Svelte amb algunes llibreries creades per la comunitat, algunes d'elles són svelte-routing o svelte-router**[23][24]**.

Altres de les funcionalitats les quals fa que SvelteKit sigui una bona opció són les que s'han mencionat en el punt 7.2.2.2 SvelteKit, el sistema d'endpoints, de routing i moltes més que es veuran més endavant quan requereixi el seu ús.

9.1.3 TypeScript

TypeScript és un llenguatge de programació lliure i de codi obert, és un superconjunt de JavaScript, que principalment afegeix tipus estàtics i objectes basats en classes, a més permet una integració més bona amb l'editor. Normalment, TypeScript s'utilitza per desenvolupar aplicacions que s'executen al client, el servidor o extensions per a programes (p.e Node.js).
TypeScript estén la sintaxi de JavaScript i internament és transformat a JavaScript, per tant, tot el codi de JavaScript és compatible amb TypeScript**[24][26]**.

La raó principal per la qual s'ha decidit utilitzar TypeScript és per la seva utilitat de tipatge de les dades, la qual permet definir una estructura de dades més sòlida i un manteniment molt més senzill. A continuació ensenyaré algunes de les seves utilitats amb alguns exemples.

TypeScript té un verificador de tipus estàtic. Permet detectar errors en el codi sense la necessitat d'executar-se. Per això s'anomena verificador de tipus estàtic. Per exemple, si observem la *Figura 34*, podem veure com hem fet un error a l'hora d'escriure "alcada" i gràcies a TypeScript podem veure l'error, i inclús, ens suggereix el canvi**[27]**.

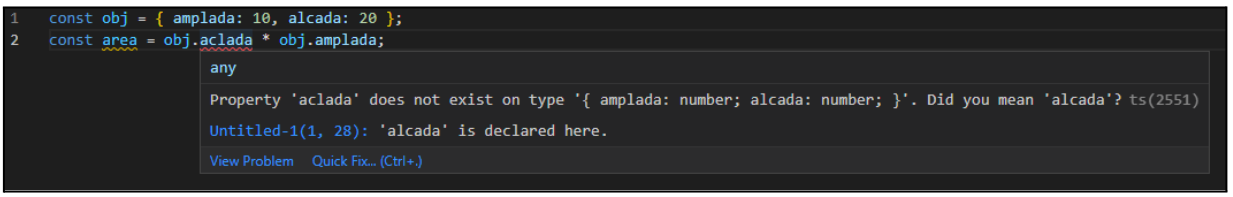


Figura 34. Verificador de tipus estàtic de TypeScript.

També et permet definir tipus de dades i verificar que l'element que s'utilitza és del tipus que realment esperes a diferència de JavaScript que ignora el tipus en l'assignació de variable o bé en els elements que es passa com a paràmetres d'una funció (veure *Figures 35 i 36*).

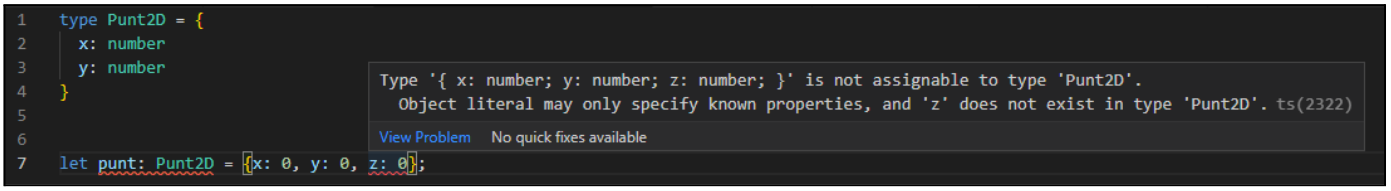


Figura 35. Correcció de TypeScript, assignació de tipus incorrecte.

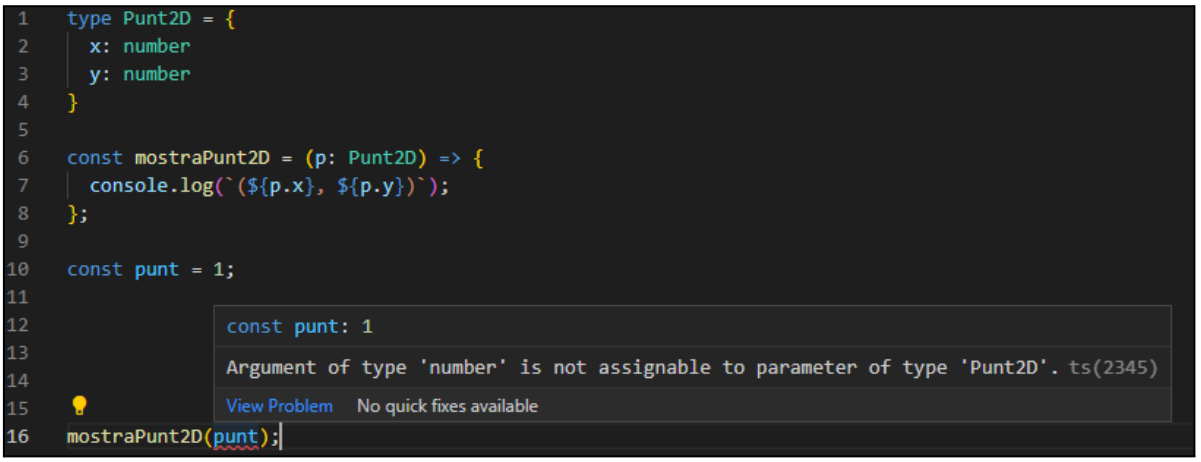


Figura 36. Correcció de TypeScript, tipus incorrecte en el paràmetre d'una funció.

9.1.4 Copilot

Copilot és una IA que et suggereix codi en temps real en l'editor. És una IA que ha estat entrenada amb bilions de línies de codi i és capaç de transformar les indicacions del llenguatge natural a suggeriments de codi. Gràcies a Copilot podem focalitzar la nostra feina en solucionar problemes més grans i oblidar-nos de problemes menors i tasques repetitives**[28]**.

A continuació podem veure com gràcies a copilot podem crear amb menys de 5 segons la funció d'ordenació bubble sort només comentant "bubblesort" (veure *Figura 37*).

```
//bubblesort
const bubbleSort = (arr: number[]) => {
  for (let i = 0; i < arr.length; i++) {
    for (let j = 0; j < arr.length; j++) {
      if (arr[i] < arr[j]) {
        const temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
```

Figura 37. Generació bubble sort amb copilot.

A més també funciona molt bé per la generació del codi que s'està creant. Si recordem l'exemple anterior, en el punt 9.3 TypeScript, havíem definit un tipus Punt2D. Amb copilot podem rebre suggeriments de codi molt precisos tal com podem observar a la *Figura 38*.

Figura 38. Suggeriment de copilot.

Gràcies a les funcionalitats comentades fa que el rendiment de treball incrementi de forma considerable, ja que, moltes vegades és necessari donar un cop d'ull a la documentació, cercar per internet alguns algorismes, fer feines repetitives com el de la *Figura 38*, etc.

A part de Copilot hi ha altres opcions en el mercat com podrien ser gtp-code-clippy, kite, TabNine, entre altres**[29][30][31]**. He decidit fer ús de Copilot, ja que se'm va concedir accés a la beta i des de llavors ja he estat familiaritzat amb el seu funcionament, a més té una molt bona integració amb Visual Studio Code.

9.1.5 Prisma

Prisma es un generador de consultes a les bases de dades que es genera automàticament a partir del esquema creat, a més, permet treballar amb tipus de dades Prisma és un generador de consultes a les bases de dades que es genera automàticament a partir de l'esquema creat, a més, permet treballar amb tipus de dades adaptats a l'aplicació. A més té integració amb TypeScript i PostgreSQL. Prisma et permet modelar les dades en un esquema. Per exemple si definim l'esquema de la *Figura 39*:

```
model Author {
  id         Int      @id @default(autoincrement())
  name       String   @unique
  bio        String?
  img        String?
  chapter    Chapter[]
}

model Chapter {
  summary      String
  authors      Author[]
  keyTakeaways String[]
  pageId       Int      @id
}
```

Figura 39. Esquema Prisma de l'aplicació simplificat.

L'esquema de la *Figura 39* correspondria a una relació N:M entre l'entitat Author i Chapter, podem observar que s'han definit dos models, Author i Chapter el qual correspondrien a les entitats, és a dir, una taula de la base de dades. Per poder relacionar els models entre ells només és necessari definir un camp amb el tipus del model que desitgem. En aquest cas per crear la relació N:M hem definit un camp en el model Author anomenat chapter que el seu tipus és Chapter[], és a dir, un array de Chapters, i, per altra banda, en el model Chapter hem definit un camp anomenat authors amb el tipus Author[]. D'aquesta forma Prisma ja sap que ens estem referint a una

relació de molts a molts. El diagrama de la base de dades que correspondria el que es mostra a continuació (veure *Figura 40*).

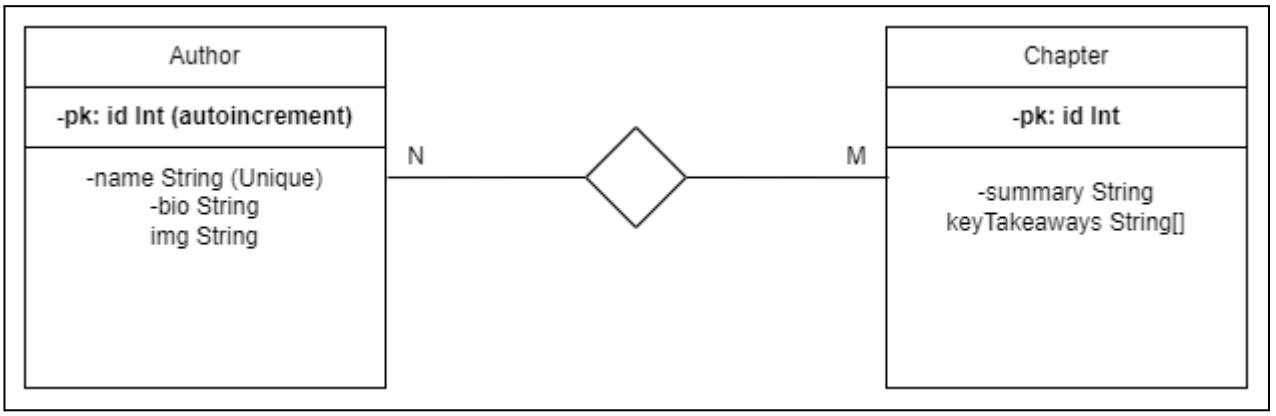


Figura 40. Esquema Entitat Relació Author Chapter simplificat.

A més també ofereix una llibreria per tal de poder obtenir els elements de la base de dades compatible amb TypeScript. Per exemple, si volem aconseguir tots els authors d'un Chapter, podem executar la següent funció dins del codi de la nostra aplicació (veure *Figura 41*).

```
const authors = await prisma.chapter.findUnique({
      where: {
        id: 1,
      },
      include: {
        authors: true,
      }
    });
```

Figura 41. Consulta per obtenir tots els authors del Chapter amb id 1.

A part del que s'ha mencionat anteriorment també permet fer migracions de la base de dades automàticament a partir de l'esquema, és a dir, podem generar les taules de la base de dades a partir del codi de la Figura 39 de forma automàtica.
Finalment, cal dir que té compatibilitat amb PostgreSQL per al que facilitat un entorn pel desenvolupament de tot el back-end. Per aquestes raons s'ha decidit utilitzar Prisma.

9.1.6 PostgreSQL

PostgreSQL és la més professional de les bases de dades relacionals de codi obert i premiada com a sistema de bases de dades de l'any múltiples vegades. Per exemple, Instagram fa ús de PostgreSQL per a la seva aplicació. PostgreSQL és una potent base de dades objecte-relacional de codi obert que utilitza i amplia el llenguatge SQL combinat amb diverses funcionalitats que permeten guardar i escalar de forma segura complicades càrregues de treball**[32]**. A part de les funcions bàsiques de qualsevol llenguatge SQL també ens deixa fer ús d'unes funcions que permeten codificar el contingut desitjable per tal que sigui compatible per fer cerques vectorials com les que s'ha mencionat en el punt 7.2.2.3 Vectors de cerca.

En el mercat existeixen algunes alternatives que també serien viables per aquest projecte com poden ser MySQL o SQLite**[34][35]**.

PostgreSQL ens ofereix dos tipus de dades que estan destinats a donar suport a la cerca per text. El contingut de la nostra aplicació en gran part serà text que estarà distribuït entre els Chapters i els Case Studies. Disposa del tipus tsvector el qual representa un document en el format adequat per la cerca per text, i també, del tipus tsquery el qual representa una consulta de text**[33]**.

El tipus tsvector és utilitzat per la cerca per text. Combina diferents variants de la mateixa paraula i eliminar duplicats per crear llistes ordenades de paraules anomenades lexemes. Per exemple, la frase "Todo pasa y todo queda pero lo nuestro es pasar, pasar haciendo caminos, caminos sobre la mar" podem veure que eliminar connectors de paraules i intel·ligentment escull les paraules més rellevants. A la frase les paraules "caminos" i "pasa" estan repetides, però en el format tsvector només hi ha una entrada en el vector, tal com es pot observar en la *Figura 42*.

```
SELECT to_tsvector('pg_catalog.spanish', 'Todo pasa y todo queda pero lo
nuestro es pasar, pasar haciendo caminos, caminos sobre la mar');
-- 'camin':13,14 'hac':12 'mar':17 'pas':2,10,11 'qued':5
```

Figura 42. Conversió a tsvector.

El tipus tsquery guarda lexemes que buscarà per ells en la cerca, a més també es poden combinar alguns lexemes amb els operadors AND, OR i NOT. Per exemple, en cas que volguéssim cercar per algun element que el seu contingut fos la frase de la *Figura 42*, segurament el que escriuriem seria una cosa semblant a "pasar caminos caminos sobre la mar". Primer de tot s'hauria de convertir el text de cerca en tsquery i posteriorment realitzar la cerca. Si observem el resultat de la *Figura 43*, podem veure que el resultat del text de cerca és "'pas' & 'camin' & 'camin' & 'mar'", Podem veure que ha transformat el text en pla en un conjunt de lexemes concatenats amb l'operador AND on posteriorment podem utilitzar per dur a terme la cerca.

```
SELECT websearch_to_tsquery('pasar caminos caminos sobre la mar')
-- 'pas' & 'camin' & 'camin' & 'mar'
```

Figura 43. Conversió de text pla a tsquery.

9.1.7 Fastly

Fastly és eina destinada a la gestió de la memòria cau dels llocs web. La seva finalitat és servir als clients el contingut de la millor forma i rapidesa possible. Això es duu a terme gràcies a uns serveis de CDN (Content Delivery Network) que actuen com un reverse proxy. Consta de diversos POPs (Points Of Presence) situats estratègicament de tal forma que podem ubicar les memòries cau a prop de diferents usuaris arreu del món independentment d'on està situat el servidor. Internament, utilitza Varnish Cache reverse proxy com arquitectura subjacent la qual no només és ràpida sinó que permet una gran personalització, cosa que en certs punts pot augmentar la complexitat de gestió de memòria cau.

Un altre dels beneficis que té és que permet optimitzar el temps de resposta balancejant la càrrega de peticions. En cas que hi hagués un gran nombre de peticions amb fastly podem servir el contingut des de la memòria cau prevenint que el servidor, en cas que tingui una sobrecàrrega de peticions i arribi a col·lapsar o caure, podem continuar donant servei als usuaris oferint les dades guardades a la memòria cau.

Normalment, els servidors estan hostatjats a servidors que tenen costos associats al tràfic d'entrada i sortida. Amb Fastly com que moltes de les peticions poden ser entregades sense haver de consultar al servidor permet reduir aquests costos**[36]**.

Per aquestes raons s'ha escollit Fastly com una eina necessària per a l'aplicació final, ja que com s'ha explicat anteriorment ofereix una sèrie d'avantatges que són molt convenients per una aplicació que està destinada a un gran nombre de persones i distribuïdes per tot el globus. Més endavant contaré en més detall el funcionament de la memòria cau en l'aplicació, no obstant hi ha algunes alternatives a fer menció, aquestes són Cloudflare CDN, Amazon CloudFront i Microsoft Azure CDN **[37][38][39]**.

9.1.8 Visual Studio Code

Visual Studio Code és un editor de codi font desenvolupat per Microsoft, Linux i macOS. Que inclou suport per a la depuració, control integrat de Git, ressaltat de sintaxi, finalització intel·ligent de codi, fragments i refactorització de codi font. He decidit utilitzar aquest editor, ja que és el que porto fent servir des de fa molt de temps i amb el que em sento a gust. En el mercat existeixen altres alternatives com Sublime Text, Coda, Notepad++, Atom, etc.**[40]**.

9.1.9 Stylus

Stylus és un llenguatge preprocessador de fulls d'estil dinàmics que es compila a CSS. El seu disseny està influenciat per Sass i LESS. A diferència d'altres alternatives com Sass o bé CSS pla, Stylus n'ofereix un conjunt de característiques que són interessants. Els selectors poden ser utilitzats sense el caràcter "{" i a més no és necessari utilitzar ":" i "," ja que són opcionals. També permet definir variables sense la necessitat de fer servir

cap caràcter especial. Igualment, permet definir mixins i funcions. Finalment, també es poden introduir variables en els arguments i identificadors**[41]**.

9.1.10 Llibreries

En el desenvolupament de components hem necessitat paquets de nodejs per tal de facilitar la creació de nous components. Aquest punt es destina a definir tot els que jo he fet servir amb una petita explicació de la seva finalitat.

9.1.10.1 Svelte-splide

Svelte-splide és un paquet de nodejs que permet utilitzar un component que habilita a crear sliders de forma ràpida i molt personalitzable**[42]**.

9.1.10.2 Svelte-multiselect

Svelte-multiselect és un paquet de nodejs que permet utilitzar un component al qual pots afegir i treure elements donada una llista**[43]**.

9.1.10.3 Textfit

Textfit és un paquet de nodejs que permet redimensionar de forma dinàmica un element en funció de la mida disponible**[44]**.

9.1.10.4 Svelte-scrollto

Svelte-scrollto és un paquet que permet fer scroll a una ubicació concreta de la pàgina amb una animació**[45]**.

9.1.10.5 MenuSpy

MenuSpy és una llibreria que permet saber en quina posició de la pàgina et trobes. S'utilitza pel menú dels Chapters i Case Studies**[46]**.

9.2 Maquinari

Aquest projecte no necessita maquinari específic. Es pot realitzar des de qualsevol ordinador, sempre que tingui els requisits mínims per l'execució del programari necessari. A més també es fan servir eines en el núvol, que estan en algun servidor. Jo he fet servir el meu ordinador personal. A continuació descriuré les característiques principals del maquinari que té.

Component	Descripció
RAM	16GB DDR4
Processador	Ryzen 5 3600
Targeta Gràfica	Nvidia GTX 1070

10. Anàlisi i disseny del sistema

En aquest apartat s'explicarà com serà l'aplicació desenvolupada, a més també s'explicarà de forma conceptual els elements més importants de l'aplicació. No totes les coses detallades aquí han estat dissenyades per mi. Procuraré mencionar en tot moment la feina realitzada per mi i entraré en més detall en cada una d'elles, en canvi, seré més breu contant la feina realitzada pels meus companys. El procediment que seguiré és desglossar les principals pàgines del lloc web i analitzar i explicar en detall les necessitats de cada una. Explicant els seus propòsits generals.

El lloc web està format per la Landing Page, la qual correspon a la benvinguda, les pàgines de lectura dels Chapters i els Case Studies, la pàgina de cerca de contingut que s'anomena Collection Page i finalment el CMS el qual està format per una Homepage, que té 2 visualitzacions diferents. Una que s'utilitza per realitzar el login, i l'altre que s'utilitza un cop l'usuari s'ha loggejat. També requereix un índex de tot el contingut creat amb el seu corresponent editor, Page Editor. Hi ha dues pàgines molt semblants que estan destinades a la gestió dels autors i dels usuaris, on pots canviar paràmetres de cada un d'ells. També hi ha una que està destinada a modificar, crear o eliminar tags del tipus *topic*. Finalment, una última que s'usa per ordenar els components de la Landing Page.

A més cada una de les pàgines disposa de 2 dissenys diferents, un adaptat a pantalles grans, és a dir, d'ordinador i l'altre adaptat a dispositiu mòbils.

10.1 Anàlisi i disseny de la Landing Page

La pàgina principal, *Landing Page*, té la finalitat de donar la benvinguda en els nous usuaris, proporcionant una breu descripció i algunes recomanacions per iniciar amb les lliçons, reptes i solucions sostenibles. Es poden diferenciar 7 mòduls diferents, cada un destinat a una funció específica. A continuació explicaré en detall la finalitat de cada un.

10.1.2 Header

L'objectiu d'aquest mòdul és donar la benvinguda a l'usuari, proporcionant una breu informació de per a què està destinat el lloc web. Aquest Mòdul està format per un títol i una breu descripció de l'eina, a més també proporciona informació de tots els col·laboradors i una barra de cerca perquè l'usuari pugui buscar per a les seves necessitats (veure *Figura 44*).

Figura 44. Header

10.1.2 Collection Cards

Les Collection Cards mostren un carrusel amb un nombre determinat d'elements. Cada element consisteix en una recomanació per l'usuari. A la pàgina principal hi ha 2 seccions, una destinada per als Chapters i un altre per als Case Studies. Si resulten atractius per l'usuari, s'hi podrà accedir de forma ràpida. A més també proporciona

informació rellevant sobre el que va destinat, ja que tots els Chapters o Case Studies disposen de Tags per a poder classificar la seva temàtica (veure *Figures 45 i 46*).

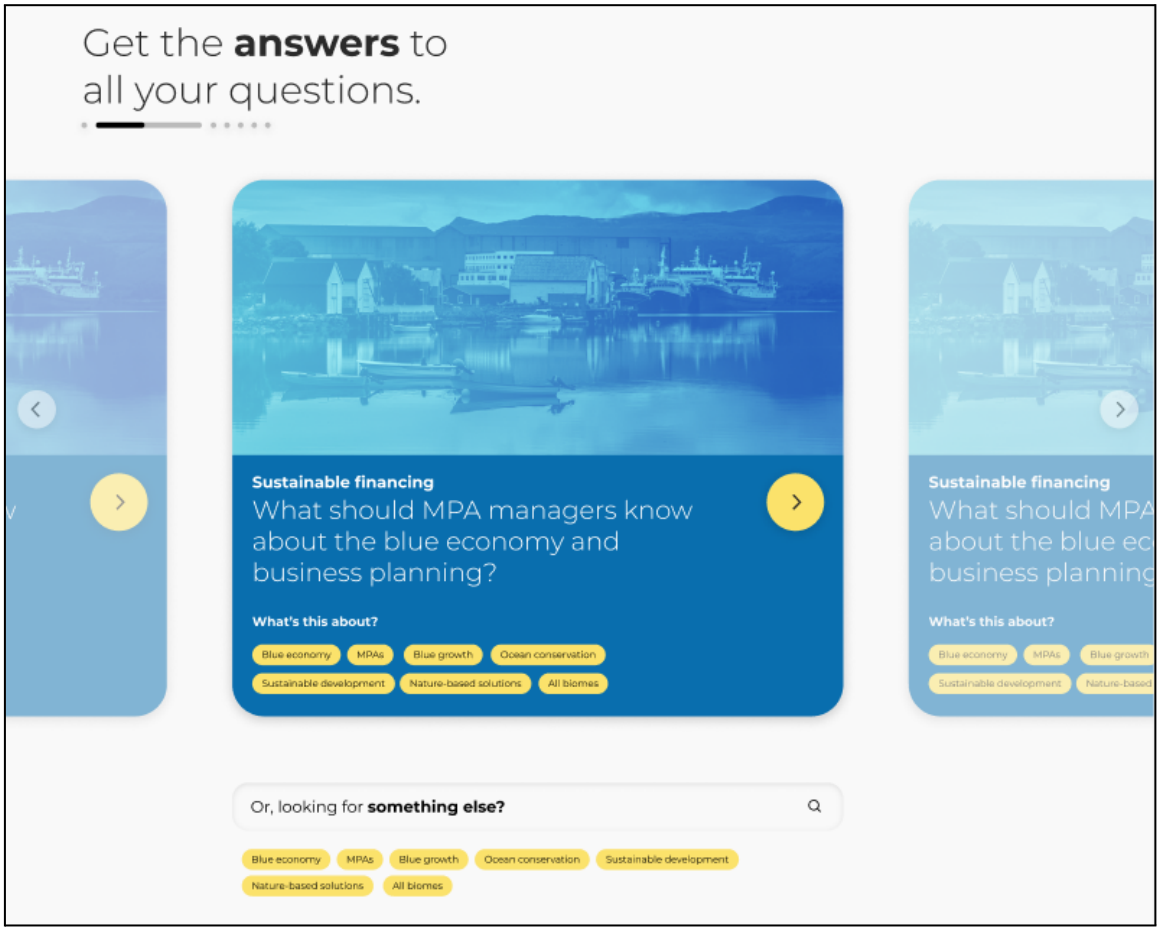


Figura 45. *Collection Cards* per els *Chapters*.

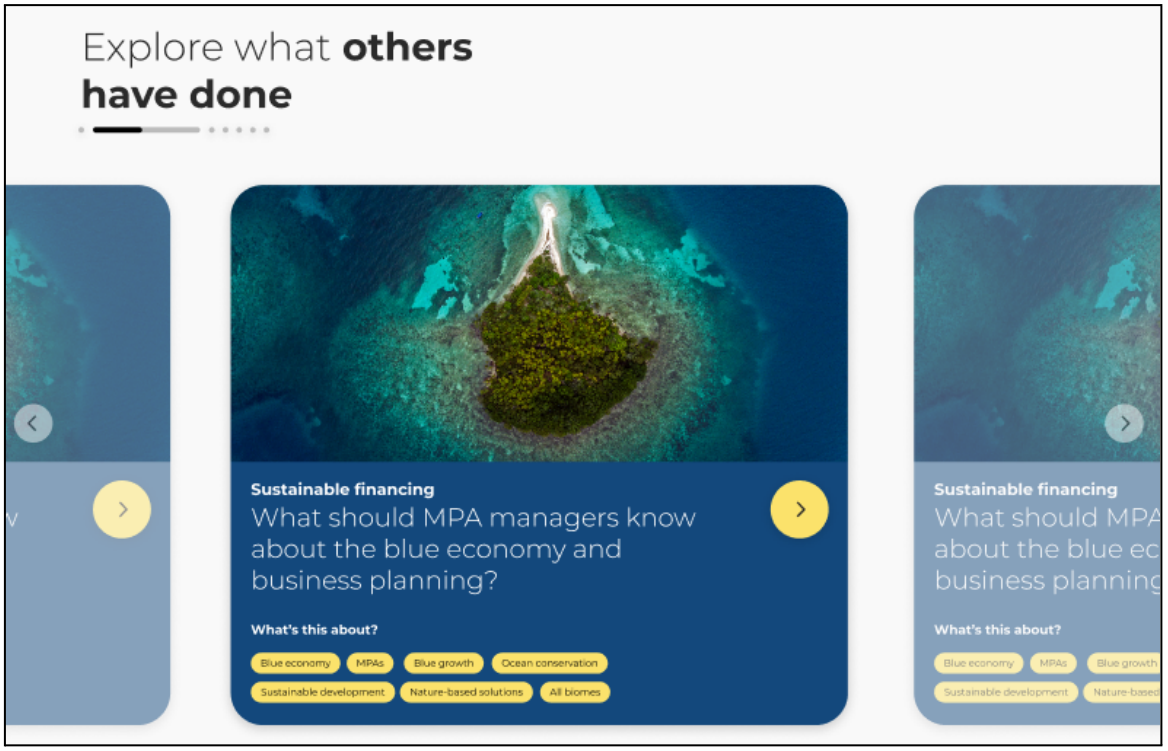


Figura 46. *Collection Cards* per els *Case Studies*.

10.1.3 MPA Management Life Cycle

El cicle de gestió dels MPA segueix uns passos estipulats i els diferents Chapters i Case Studies es centren en alguns passos en concret. Cada una de les entrades del lloc web està vinculat a diversos moments d'aquest cicle. L'objectiu d'aquest mòdul és informar els usuaris sobre quin moment del cicle es troben els articles. Aquests punts són els que s'han explicat en el punt 7.2.1.7 Lifecycle (veure *Figura 47*).

Figura 47. Component MPA Management Life Cycle.

Per la realització d'aquest component podem observar dos elements principals. Els TextSlides i el CircleMenu. Primer explicaré el funcionament del Circle Menu.

10.1.3.1 Circle Menu

Els Circle Menu és un component que interacciona amb l'usuari, les seves formes d'interacció són fer "hover" amb el cursor per sobre un segment o bé fer clic amb el cursor a un segment. A continuació podem veure la fitxa de casos d'ús i un diagrama de flux del component CircleMenu.svelte (veure *Figures 48 i 49*).

CircleMenu.svelte	
DESCRIPCIÓ	Són un conjunt de segments dels quals poden tenir 3 estats diferents, unselected, main selected o secondary selected.
ACTORS	Usuari
PRECONDICIÓ	El component està vinculat amb el *TextSlider* o algun altre component de funcionament semblant.
FLUX PRINCIPAL	1. Si usuari fa hover amb el cursor a un segment

	1.1. Canviar estat segment a seleccionat 2. Si usuari fa clic a un segment 2.1. Canviar estat del segment a main selected o secondary selected, depenent de la configuració de cada segment. 2.2. Actualitzar índex seleccionat.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	El segment seleccionat ha estat modificat

Figura 48. Fitxa casos d'ús.

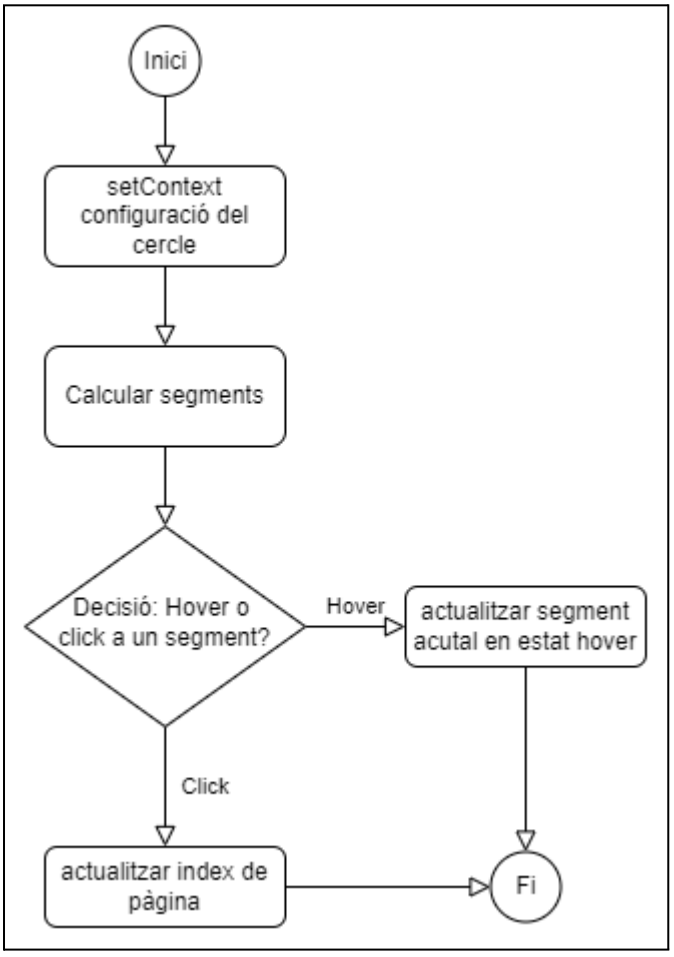


Figura 49. Diagrama de flux de CircleMenu.svelte

En el càlcul dels segments es requereix definir per cada segment, angle d'inici, angle fi i el tipus de segment. En el següent punt entrarem en detall, ja que són característiques del component CircularSegment.svelte.

Si observem el diagrama de flux, es defineix la configuració del cercle. El cercle té certs paràmetres de configuració que permeten la reutilització i personalització d'aquest component, si analitzem l'estructura de dades que el defineix (veure *Figura 50*). Veiem que es pot definir un radi, una mida del cercle generat, l'espai entre segments i l'amplada de cada segment depenent del seu tipus. El tipus de cada segment venen definits pel tipus SegmentType (veure *Figura 51*).

```
type CircleConfig = {
  radius: number;
  size: number;
  gap: number;
  thicknesses: {
      main: number;
      secondary: number;
      unselected: number;
      hover: number;
  };
}
```

Figura 50. Paràmetres de configuració de CircularSegment.svelte

```
export type SegmentType = 'main' | 'secondary' | 'unselected';
```

Figura 51. Tipus de segments.

10.1.3.2 Circular Segment

El Circular Segment és un component que s'encarrega de generar un segment que ve definit per un angle inici i un angle fi, a més també es pot definir el tipus de segment que és, tal com hem vist en la *Figura 51* del punt anterior. A continuació si observem les *Figures 52 i 53* corresponents a la fitxa de casos d'ús i diagrama de flux podrem veure les tasques que ha de realitzar aquest component.

CircularSegment.svelte	
DESCRIPCIÓ	És un segment d'un cercle el qual interacciona quan es fa hover o bé es clica sobre ell.
ACTORS	Usuari
PRECONDICIÓ	El component està vinculat amb el CircleMenu i té una configuració vàlida
FLUX PRINCIPAL	1. Si usuari fa hover amb el cursor al segment 1.1. Canviar estat hovered 2. Si usuari fa clic a un segment 2.1. Propagar event click
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	El segment ha canviat d'estat

Figura 52. Fitxa casos d'ús del component CircularSegment.svelte.

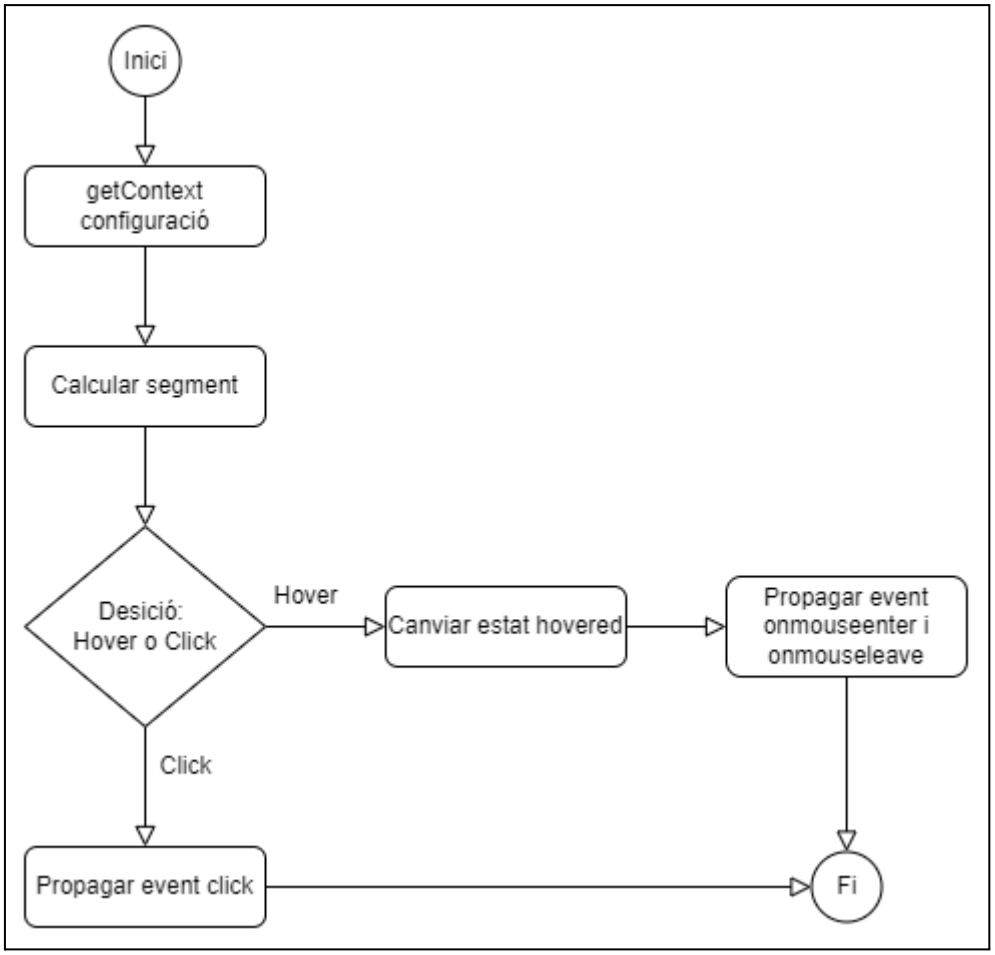


Figura 53. Diagrama de flux component CircularSegment.svelte.

Aquest component s'encarrega de propagar els esdeveniments necessaris per gestionar en el circle menu el comportament desitjat. A més també genera els segments conforme als paràmetres de configuració passada per la variable de context "circleConfig". Els paràmetres de configuració venen definits per la interfície Segment la qual et permet identificar l'id de la tag amb el que està vinculat <number>, l'angle d'inici <number>, l'angle de fi <number> i el tipus de Segment <SegmentType> (veure *Figura 54*).

```
export interface Segment {
  tagId: number;
  startAngle: number;
  endAngle: number;
  type: SegmentType;
}
```

Figura 54. Interfície Segment.

10.1.3.3 Cards

Aquest component inicialment va estar dissenyat per mi, però posteriorment s'hi van afegir noves funcionalitats, com permetre la seva edició o bé introduir nous apartats. Em centraré a explicar el funcionament bàsic sense tenir en compte les modificacions fetes pels meus companys.

La finalitat d'aquest component és mostrar cards que es van movent automàticament o causa d'una interacció de l'usuari. Cada card està format per un títol i un cos. Anem a veure la fitxa de casos d'ús i el diagrama de flux per entendre amb més detall el seu funcionament (veure *Figura 55 i 56*).

Cards	
DESCRIPCIÓ	Permet a l'usuari visualitzar una sèrie de cartes amb text informatiu. La transició de les cartes és automàtic, també es pot forçar la seva transició.
ACTORS	Usuari
PRECONDICIÓ	Hi ha més d'una carta de text.
FLUX PRINCIPAL	1. Es visualitza la primera *slide* de text. 2. Si la carta conté més text del que pot ser vist LLAVORS: 2.1. S'aplica un gradient per indicar que hi ha més text per llegir. 2.2. Si l'usuari situa el cursor sobre el cos de la carta i fa *scroll down* LLAVORS: 2.2.1. Apareix una *scroll bar* indicant la posició actual. 2.2.2. Si l'usuari arriba al final del text LLAVORS: 2.2.2.1. S'elimina el gradient per indicar que s'ha arribat al final del text. 3. Si l'usuari fa clic als botons de controls LLAVORS:

	3.1. Si clica el botó de següent *slide* LLAVORS: 3.1.1. Mostra la següent *slide* (en cicle). 3.2. Si clica el botó de anterior *slide* LLAVORS: 3.2.1. Mostra l'anterior *slide* (en cicle). 4. Si el temps per *slide* es supera LLAVORS: 4.1. Es mostra la següent *slide* (en cicle).
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	S'han mostrat les *slides* al usuari.
COMENTARIS	En cas de que només es disposi d'una sola *slide* només es mostrarà aquesta i l'única interacció amb el component serà el *scroll down*.

Figura 55. Fitxa casos d'ús del component TextSlider.svelte

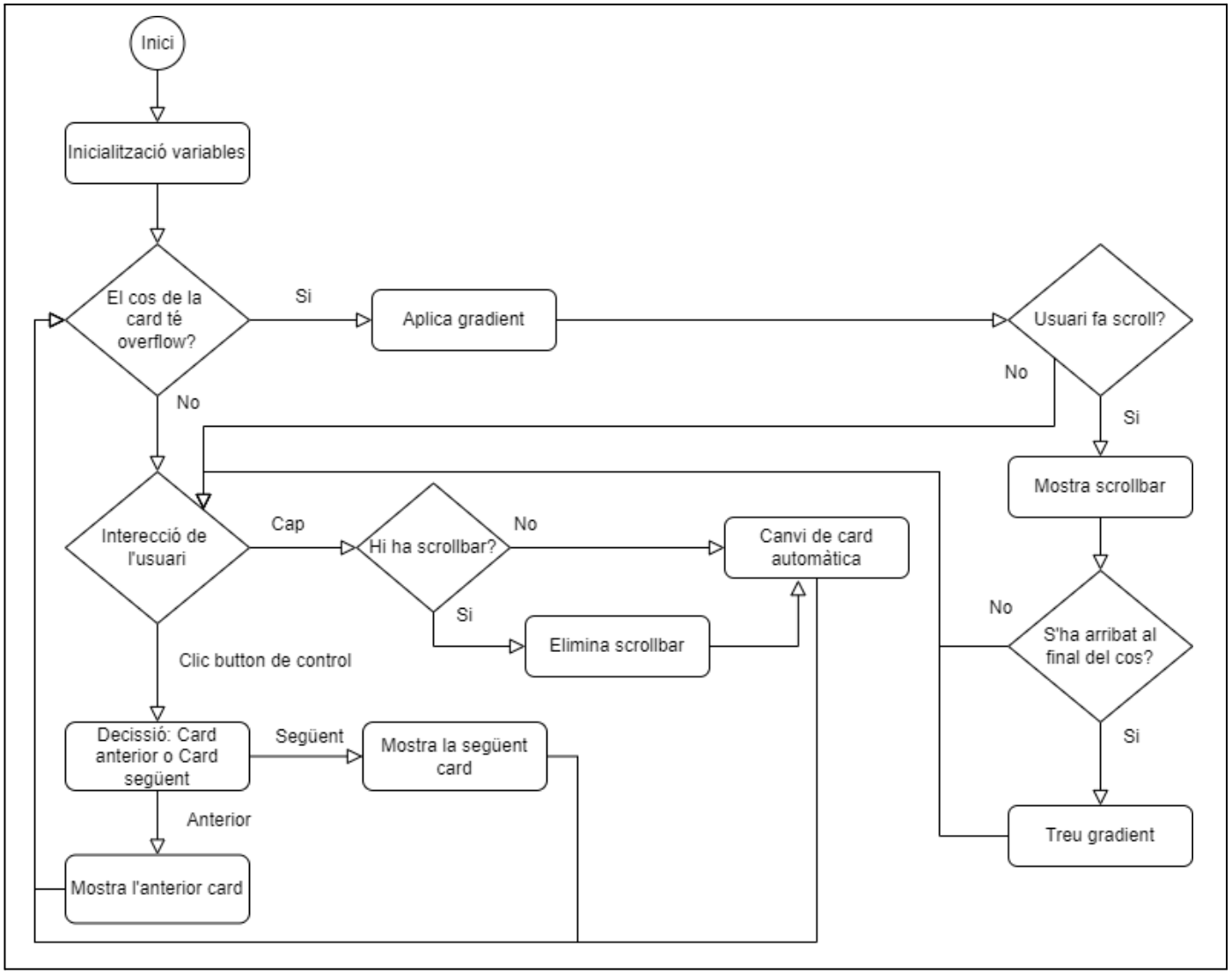


Figura 56. Diagrama de flux del component Cards.

10.1.3.4 Carousel Dots

Aquest component està dissenyat per treballar amb sincronia amb el Text Slider. La seva finalitat és tenir un control de l'índex de pàgina de cada text slider. Pot tenir dues formes de treball, per una banda, la forma dinàmica, la qual consisteix en una transició entre tots els punts del carousel amb una animació. Aquesta animació defineix el temps que cada card del Text Slider serà mostrada per l'usuari. La segona forma de funcionament és de manera estàtica on l'animació i el canvi de punt seleccionat no serà automàtic sinó que requereix interacció de l'usuari per modificar l'índex de pàgina. A continuació, si observem les *Figures 57 i 58*, corresponents a la fitxa de casos d'ús i el diagrama de flux es podrà veure una visió més detallada del seu funcionament.

CarouselDots.svelte	
DESCRIPCIÓ	Són una sèrie de punts dels quals només un d'ells es l'actiu, quan aquest punt està actiu es mostre diferent els altres i arbitràriament pot tenir un temporitzador.
ACTORS	Usuari
PRECONDICIÓ	El component està vinculat amb el *TextSlider* o algun altre component de funcionament semblant.
FLUX PRINCIPAL	1. Si l'usuari clica a qualsevol punt que no sigui l'actiu LLAVORS: 1.1. El punt clicat és el nou punt actiu.
FLUX ALTERNATIU (amb animació)	1. Si l'usuari clica a qualsevol punt que no sigui l'actiu LLAVORS: 1.1. El punt clicat és el nou punt actiu. 1.2. Inicia el temporitzador amb una animació. 2. Si la barra de progress arriba al final LLAVORS: 2.1. Actualitza l'índex de pàgina.
POSTCONDICIÓ	El punt actiu s'ha modificat per el flux principal, per el flux alternatiu la funció arbitrària s'ha executat.
COMENTARIS	La funció arbitrària executada en el flux alternatiu sol estar vinculada amb l'índex dels elements que està senyalitzant p.e: quina *slide* es l'actual.

Figura 57. Fitxa casos d'ús de CarouselDots.svelte

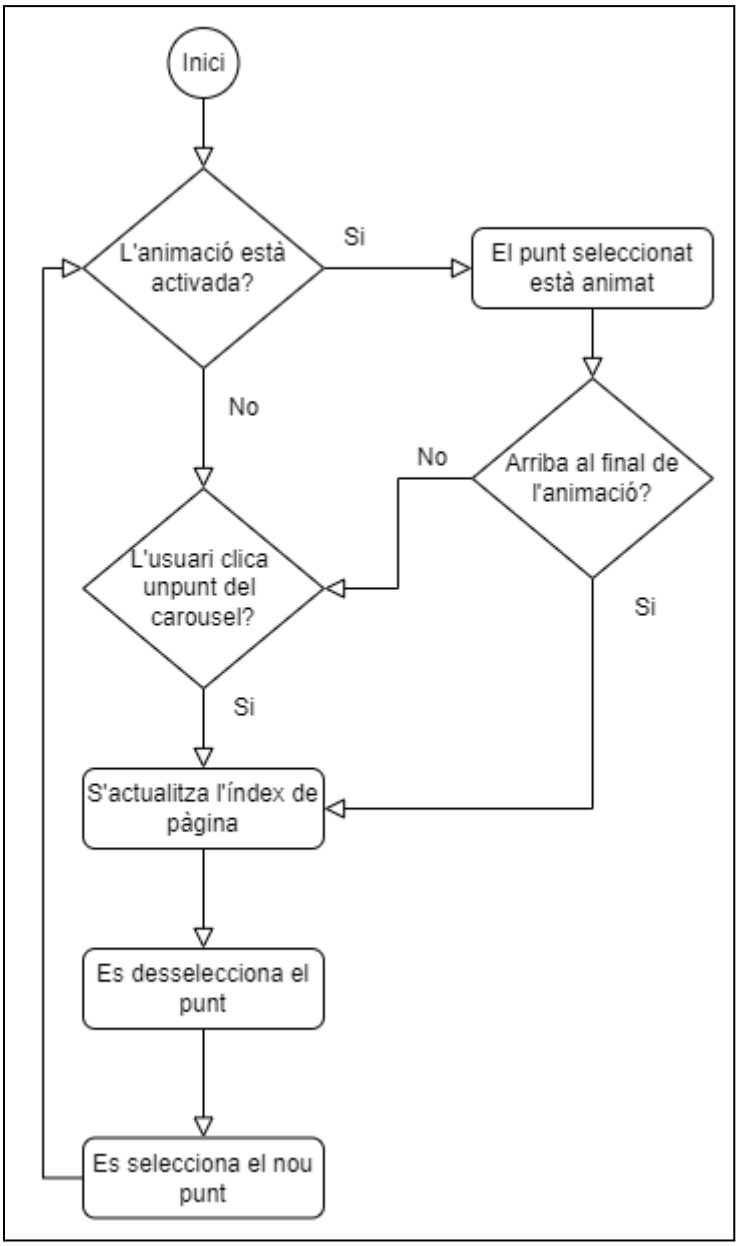


Figura 58. Diagrama de flux de CarouselDots.svelte

Aquest component internament funciona amb un altre anomenat Dot, el qual representa cada un dels punts de la barra de punts d'aquest component. En el següent punt explico en detall el seu funcionament.

10.1.3.5 Dot

El component Dot és la representació d'un punt, aquest punt pot ser estàtic o dinàmic en funció d'un sol paràmetre. La representació dinàmica consisteix en una barra que té un progrés i en quant finalitza emet un esdeveniment de finalització d'animació per tal que el component pare el pugui gestionar. La representació estàtica actua quan l'usuari fa clic en ell emetent un esdeveniment per gestionar el seu clic en el component pare. A

continuació podem observar la fitxa de casos d'ús i el diagrama de flux del seu comportament (veure *Figures 59 i 60*).

Dot.svelte	
DESCRIPCIÓ	És un sol punt que pot està seleccionat o deseleccionat, arbitràriament pot tenir una animació que al finalitza emet un event.
ACTORS	Usuari
PRECONDICIÓ	El component està vinculat amb CarouselDots.svelte o algún component de funcionalitat similar.
FLUX PRINCIPAL	1. Si l'usuari clica al punt i no està actiu LLAVORS: 1.1. S'emet event de click
FLUX ALTERNATIU (amb animació)	1. Si la barra de progress arriba al final LLAVORS: 1.1. S'emet event de finalització d'event
POSTCONDICIÓ	El punt no actiu ha emès un event per la seva gestió.

Figura 59. Fitxa de casos d'ús del component Dot.svelte.

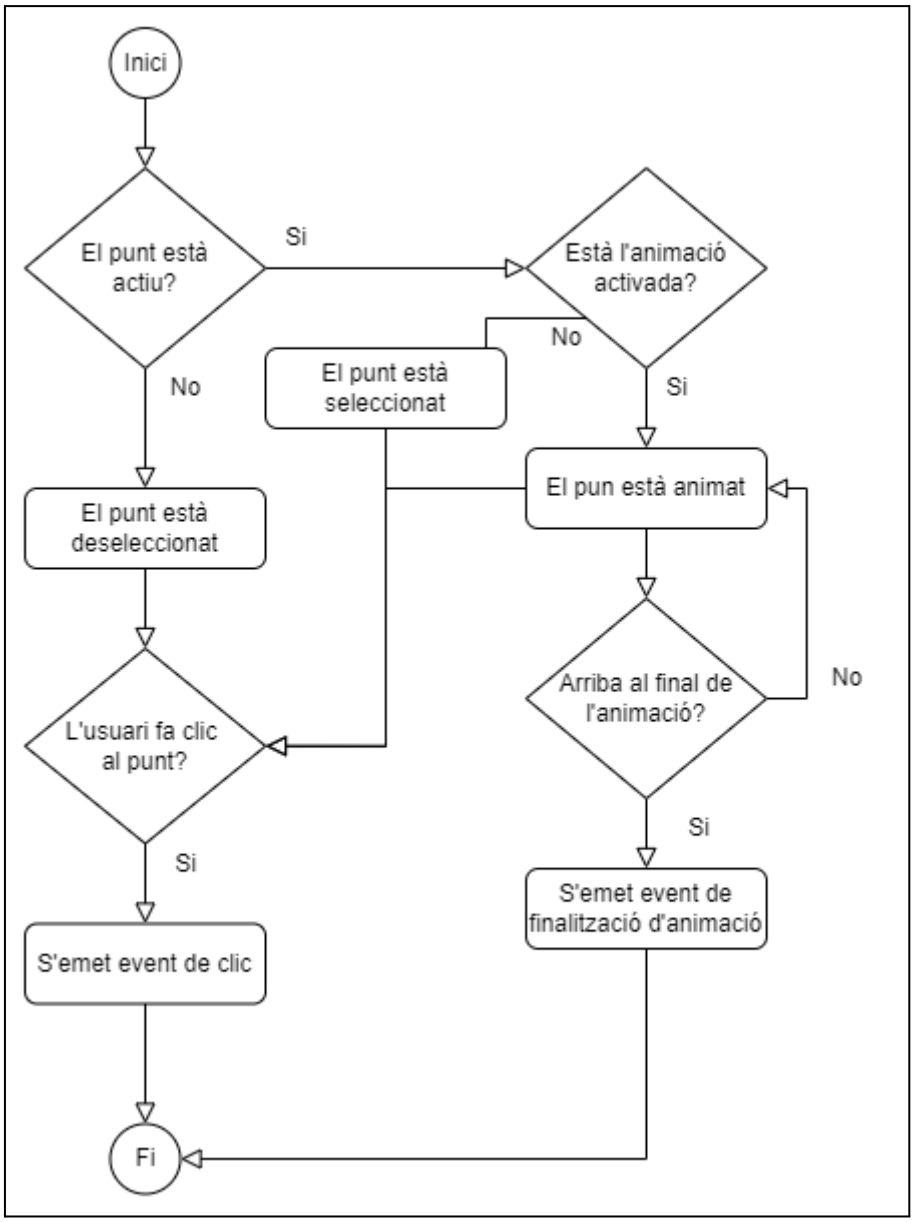


Figura 60. Diagrama de flux Dot.svelte.

10.1.4 Madlib

El madlib és un component realitzat pels meus companys que la seva finalitat és ajudar a l'usuari a buscar contingut en funció del tipus d'usuari que és. Consisteix en una sèrie de dropdowns amb opcions per tal de poder saber quines són les seves necessitats i a partir d'aquí assignar un tipus d'usuari i mostrar-li contingut en funció (veure *Figura 61*).

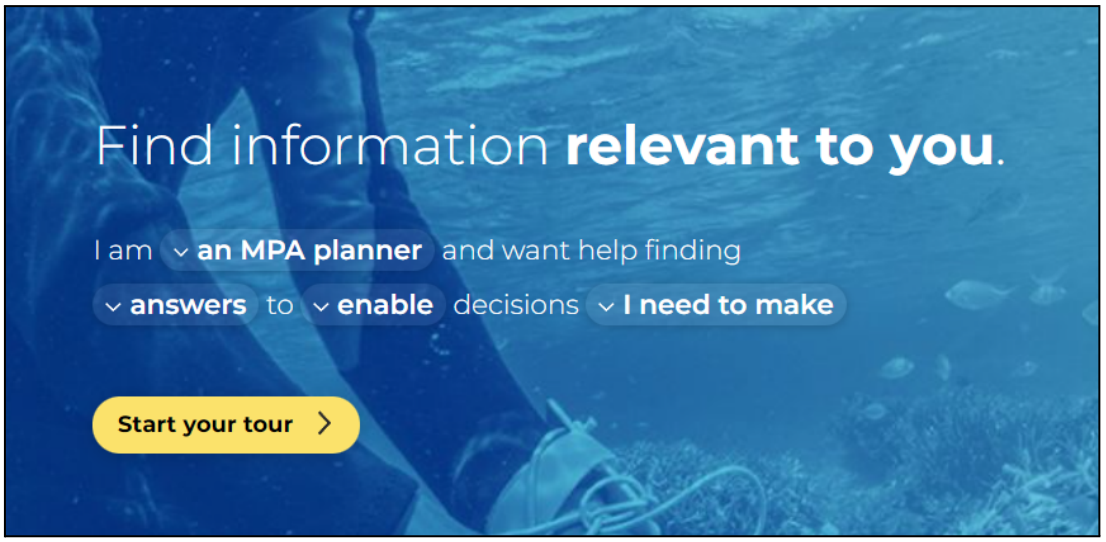


Figura 61. Component Madlib.

10.1.5 Footer

Aquest component ha estat realitzat pels meus companys, és el peu de pàgina on es mostra informació de les associacions que han col·laborat en el projecte. A més també dona accés a pàgines informatives i de contacte (veure *Figura 63*).

Figura 62. Component Footer.

10.2 Anàlisi i disseny de les pàgines de contingut

Les pàgines de contingut són les que corresponen a les entrades dels Chapters o Case Studies. Tant per Case Studies o per Chapters el contingut és el mateix excepte la capçalera de cada un.

10.2.1 Header

L'objectiu del header és ubicar a l'usuari dins de cada capítol i Case Study. Proporcionar una descripció de l'article de forma específica, clara, exacta, breu i concisa, possibilitant al lector la identificació del tema amb facilitat i proporcionar una indexació precisa de l'article. A més també mostra una estimació del temps de lectura.

La capçalera dels Chapters i els Case Studies no és la mateixa. Per una banda, en els Chapters està formada per un títol, un resum, els autors i les key takeaways, que correspon al component que s'ha explicat en el punt 10.1.3.2 TextSlider. Consisteix en uns cards que mostren els punts claus del Chapter (veure *Figura 63*).

Figura 64. Capçalera dels Chapters.

La capçalera dels Case Studies, està formada per un títol, un resum explicant algunes característiques del Case Study i les milestones (veure *Figura 65*).

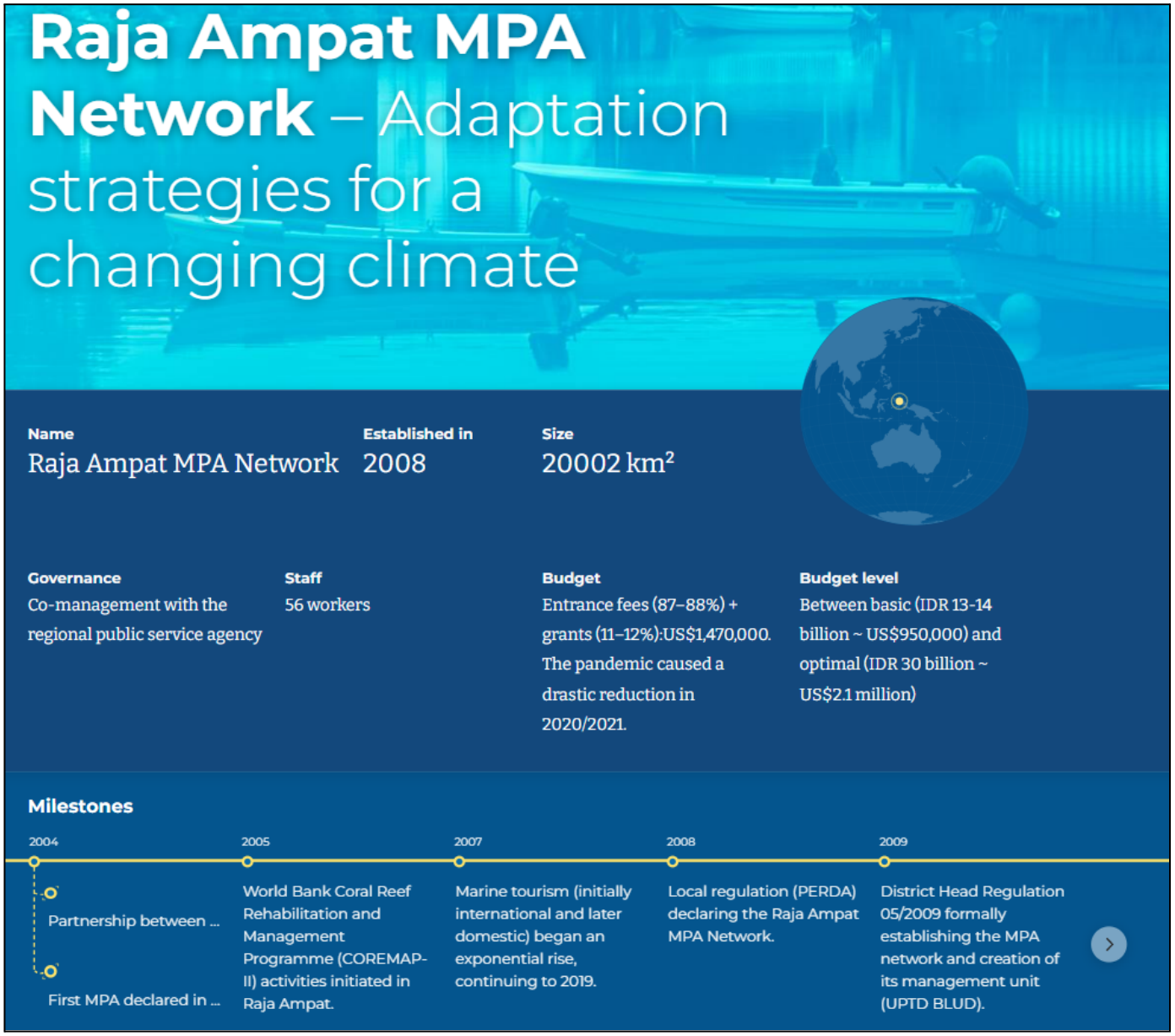


Figura 65. Capçalera dels Case Studies.

10.2.2 LifeCycle

El component LifeCycle és utilitzat per ubicar el Chapter o Case Study, també s'utilitza en el CMS per afegir o treure les tags associades a un Chapter o Case Study. En ell es mostren totes les tags associades. A més, les tags de tipus *stage* quan es passa el cursor per sobre visualitzen una descripció.

A continuació si mirem les *Figures 66 i 67* podem veure la fitxa de casos d'ús i el diagrama de flux del component en el mode visualització.

LIFE CYCLE	
DESCRIPCIÓ	Permet a l'usuari visualitzar totes les tags relacionades amb el contingut.
ACTORS	Usuari
PRECONDICIÓ	Hi ha com a mínim una tag.
FLUX PRINCIPAL	1. Si l'usuari fa clic sobre una tag, LLAVORS: 1.1. L'usuari es redirigit a la pàgina collection page amb resultats de cerca de la tag clicada. 2. Si l'usuari fa hover a una tag del tipus *stage*, LLAVORS: 2.1. Es mostra text descriptiu de la tag. 3. Si l'usuari fa hover sobre l'interrogan, LLAVORS: 3.1. Es mostra un text d'ajuda explicant el funcionament.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L'usuari ha set redirigit a les collection page o informat sobre el funcionament del component o informat sobre les tags del tipus *stage*.

Figura 66. Fitxa de casos d'ús del component LifeCycle.

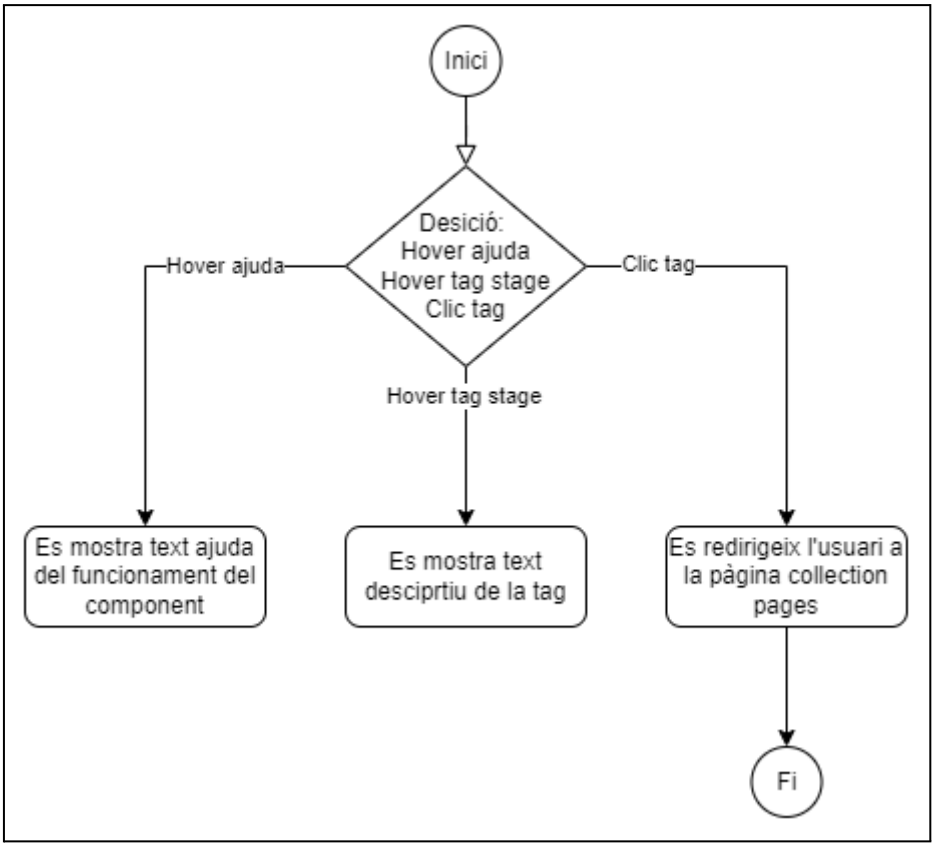


Figura 67. Diagrama de flux del component LifeCycle.

Les *Figures 68 i 69* corresponen a la fitxa de casos d'ús en mode edició.

LIFE CYCLE	
DESCRIPCIÓ	Permet a l'usuari visualitzar totes les tags relacionades amb el contingut i afegir o treure'n.
ACTORS	Usuari
PRECONDICIÓ	Cap.
FLUX PRINCIPAL	1. Si l'usuari fa hover a una tag del tipus *stage*, LLAVORS: 1.1. Es mostra text descriptiu de la tag. 2. Si l'usuari fa hover sobre l'interrogan, LLAVORS: 2.1. Es mostra un text d'ajuda explicant el funcionament. 3. Si l'usuari fa clic sobre el multiselect de les tag *stage*

	primary, LLAVORS: 3.1. Es visualitzen totes les tags *stage* disponibles. 3.2. Si l'usuari clica una de les tags disponibles, LLAVORS: 3.2.1. S'afegeix la tag en el multiselect. 3.2.2. S'actualitza el CircleMenu 4. Si l'usuari clica la creu del multiselect de les tags tipus *stage* primary, LLAVORS: 4.1. S'eliminen totes les tags tipus *stage* primary. 4.2. S'actualitza el CircleMenu 5. Si l'usuari clica la creu d'una tag *stage* primary, LLAVORS: 5.1. La tag clicada és eliminada. 5.2. S'actualitza el CircleMenu 6. Si l'usuari fa clic sobre el multiselect de les tag *stage* secondary, LLAVORS: 6.1. Es visualitzen totes les tags *stage* disponibles. 6.2. Si l'usuari clica una de les tags disponibles, LLAVORS: 6.2.1. S'afegeix la tag en el multiselect. 6.2.2. S'actualitza el CircleMenu 7. Si l'usuari clica la creu del multiselect de les tags tipus *stage* secondary, LLAVORS: 7.1. S'eliminen totes les tags tipus *stage* secondary. 7.2. S'actualitza el CircleMenu 8. Si l'usuari clica la creu d'una tag *stage* secondary, LLAVORS: 8.1. La tag clicada és eliminada. 8.2. S'actualitza el CircleMenu 9. Si l'usuari fa clic sobre el multiselect de les tag *topic*, LLAVORS: 9.1. Es visualitzen totes les tags *topic* disponibles.

	9.2. Si l'usuari clica una de les tags disponibles, LLAVORS: 9.2.1. S'afegeix la tag en el multiselect 10. Si l'usuari clica la creu del multiselect de les tags tipus *topic*, LLAVORS: 10.1. S'eliminen totes les tags tipus *topic*. 11. Si l'usuari clica la creu d'una tag *topic*, LLAVORS: 11.1. La tag clicada és eliminada. 12. Si l'usuari fa clic sobre el multiselect de les tag *user*, LLAVORS: 12.1. Es visualitzen totes les tags *user* disponibles. 12.2. Si l'usuari clica una de les tags disponibles, LLAVORS: 12.2.1. S'afegeix la tag en el multiselect 13. Si l'usuari clica la creu del multiselect de les tags tipus *user*, LLAVORS: 13.1. S'eliminen totes les tags tipus *user*. 14. Si l'usuari clica la creu d'una tag *user*, LLAVORS: 14.1. La tag clicada és eliminada.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L'usuari ha pogut afegir i treure tags.

Figura 68. Fitxa de casos d'ús del component LifeCycle en mode edició.

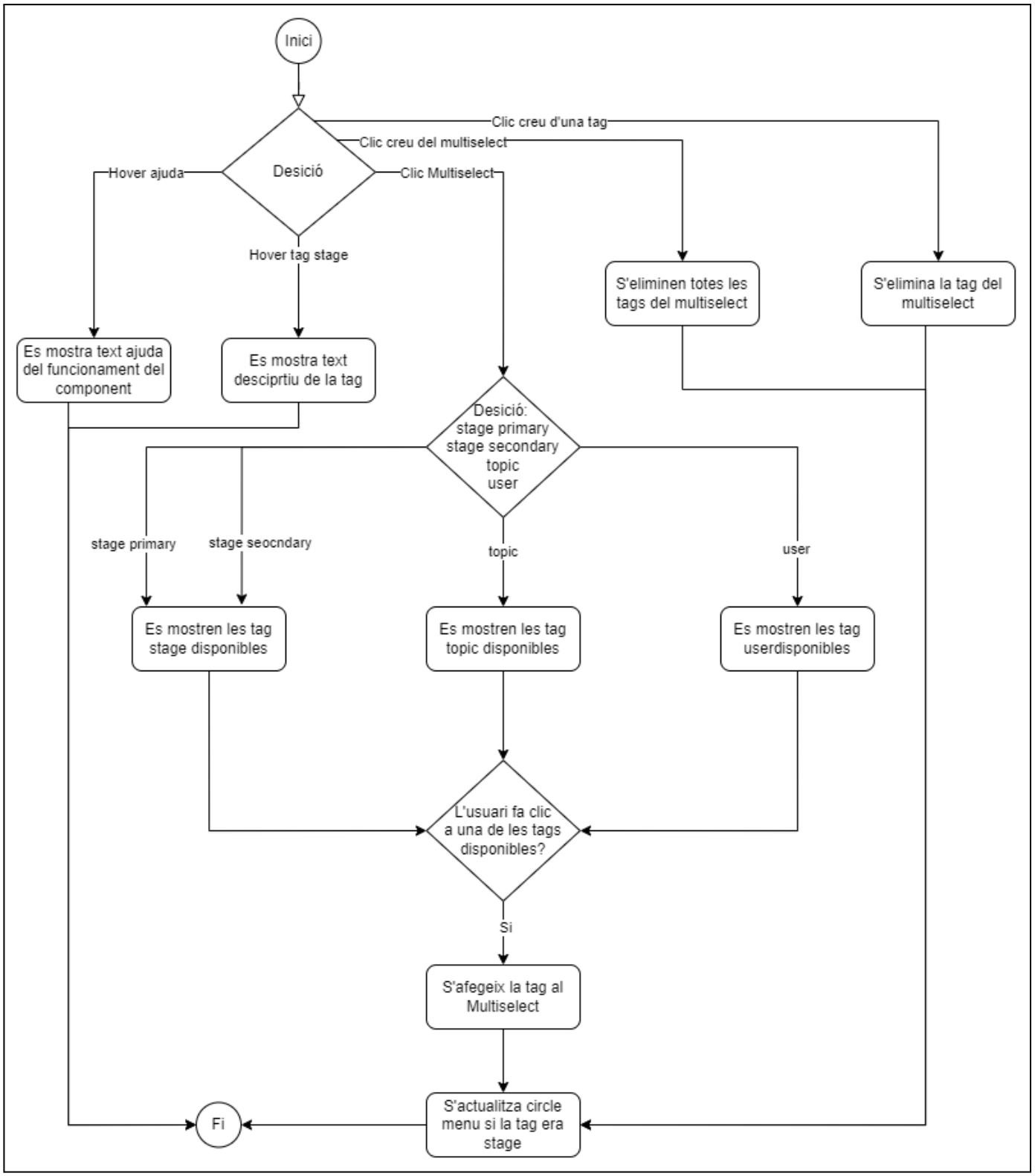


Figura 69. Diagrama de flux del component LifeCycle en mode edició.

10.2.3 Contingut

El contingut majoritàriament és text, però a més hi ha tres components que també estan presents. Aquests són els Cards que s'ha explicat anteriorment en el punt 10.1.3.2 Cards, StickyMenu i RecommendedPages. En aquest punt detallaré els dos últims.

10.2.3.1 Sticky Menu

El component StickyMenu ha de permetre a l'usuari a navegar a través d'un Chapter o d'un Case Study. Té dues interaccions diferents, una quan es clica a sobre un element del menú, que llavors, ha de redirigir a l'usuari a la secció clicada, i un segon que s'actualitza a mesura que l'usuari navega per la pàgina. Quan ha fet scroll fins a una secció nova, l'element seleccionat del menú ha de canviar i ser la secció actual. A continuació si mirem les *Figures 70 i 71* podem veure la seva fitxa de casos d'ús i el seu diagrama de flux.

STICKY MENU	
DESCRIPCIÓ	Permet a l'usuari navegar a través del contingut.
ACTORS	Usuari
PRECONDICIÓ	Hi ha com a mínim una entrada en el menú
FLUX PRINCIPAL	1. Si l'usuari fa clic sobre un element del menú, LLAVORS: 1.1. Es fa scroll automàticament a la secció clicada. 1.2. L'element clicat es marca com a seleccionat. 1.3. L'element anterior es desselecciona. 2. Si l'usuari fa scroll a la pàgina, LLAVORS: 2.1. L'element del menú corresponen a la secció ubicada es marca com a seleccionat. 2.2. L'anterior element del menú es deselecciona.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L'usuari ha set informat del punt on es troba o bé redirigit al punt desitjat.

Figura 70. Fitxa de casos d'ús del component StickyMenu.

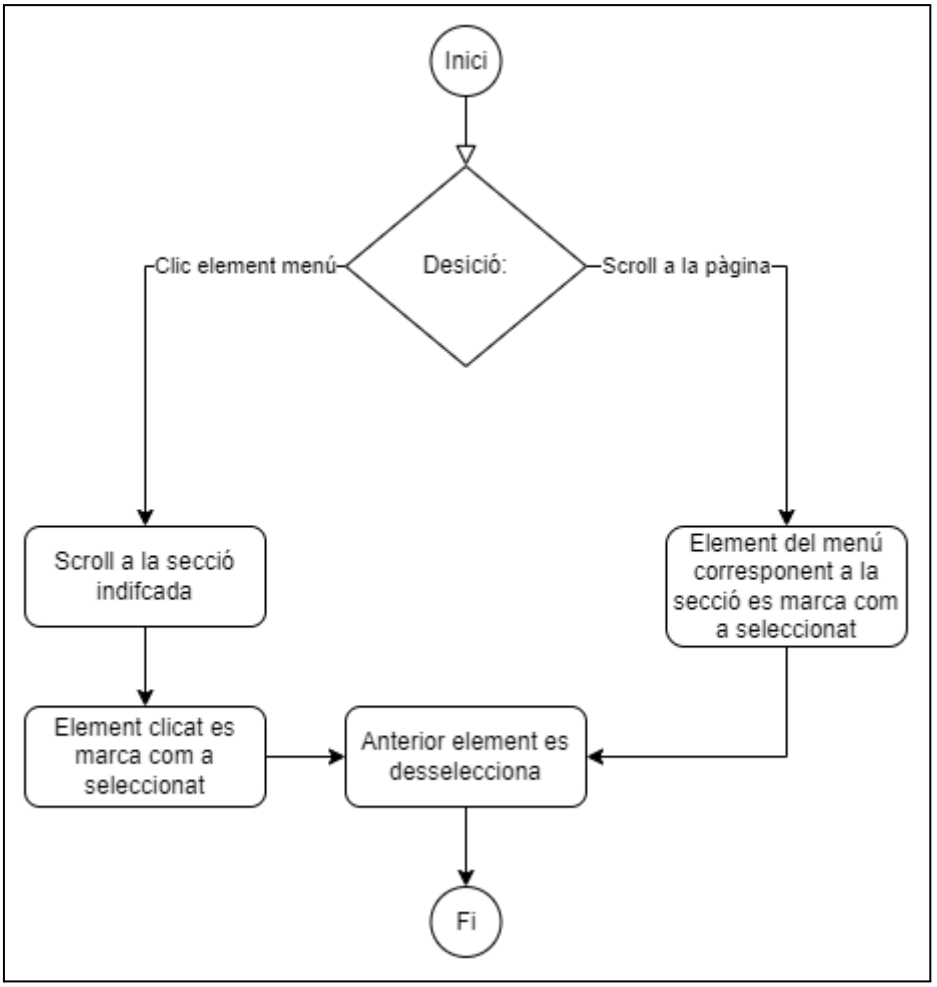


Figura 71. Diagrama de flux del component StickyMenu.

10.2.3.2 Recommended pages

El component RecommendedPages està destinat a informar a l'usuari de possibles recomanacions relacionades amb la lectura actual. Aquesta informació es mostra en forma de carrusel, l'usuari pot navegar elements endavant i enrere. Cada element està format per una imatge i el títol corresponent. Tant la imatge com el títol et redirigeixen directament al Chapter o Case Study que correspon. A més, també es mostren totes les tags de tipus stage, si es fa clic a una de les tags, et redirigeix a la pàgina de cerca amb els resultats de cerca de la tag corresponent. A continuació les *Figures 72 i 73* corresponent a la fitxa de casos d'ús i el diagrama de flux, respectivament.

RECOMMENDED PAGES	
DESCRIPCIÓ	Permet a l'usuari navegar a través del contingut recomenat.
ACTORS	Usuari

PRECONDICIÓ	Hi ha com a mínim una entrada en el carousel
FLUX PRINCIPAL	1. Si l'usuari fa clic sobre la fletxa següent, LLAVORS: 1.1. S'avança el carrusel. 2. Si l'usuari fa clic sobre la fletxa anterior, LLAVORS: 2.1. El carrusel retrocedeix. 3. Si l'usuari fa clic sobre l'imatge d'un element del carrusel, LLAVORS: 3.1. Es redirigeix l'usuari al Chapter o Case Study corresponent. 4. Si l'usuari fa clic sobre el títol d'un element del carrusel, LLAVORS: 4.1. Es redirigeix l'usuari al Chapter o Case Study corresponent. 5. Si l'usuari fa clic sobre una tag d'un element del carrusel, LLAVORS: 5.1. Es redirigeix l'usuari a la pàgina de cerca amb els resultats corresponent a la tag clicada.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L'usuari ha navegat a través de les pàgines recomanades o ha entrat a un Chapter o Case Study o ha realitzat una cerca per la tag corresponent.

Figura 72. Fitxa de casos d'ús del component RecommendedPages.

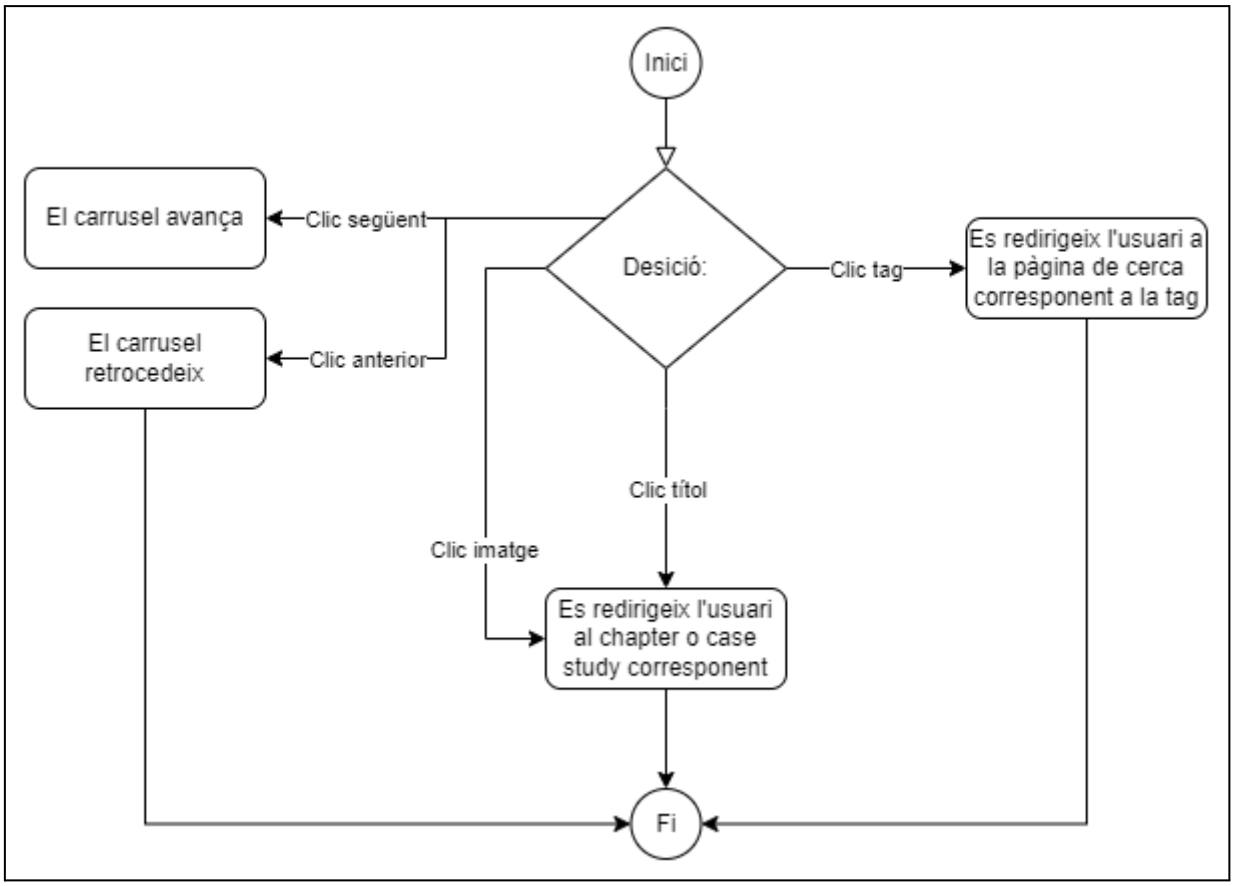


Figura 73. Diagrama de flux del component RecommendedPages.

10.3 Anàlisi i disseny Collection Pages

Les collections pages és una pàgina destinada a la cerca d'elements en el CMS. En ella es poden cercar tota classe d'elements del contingut, Cerca per context, per autor, per tags, per títol, etc. Està formada per barra de cerca on pots escriure el contingut que vols cercar i una llista amb els resultats trobats (veure *Figura 74*).

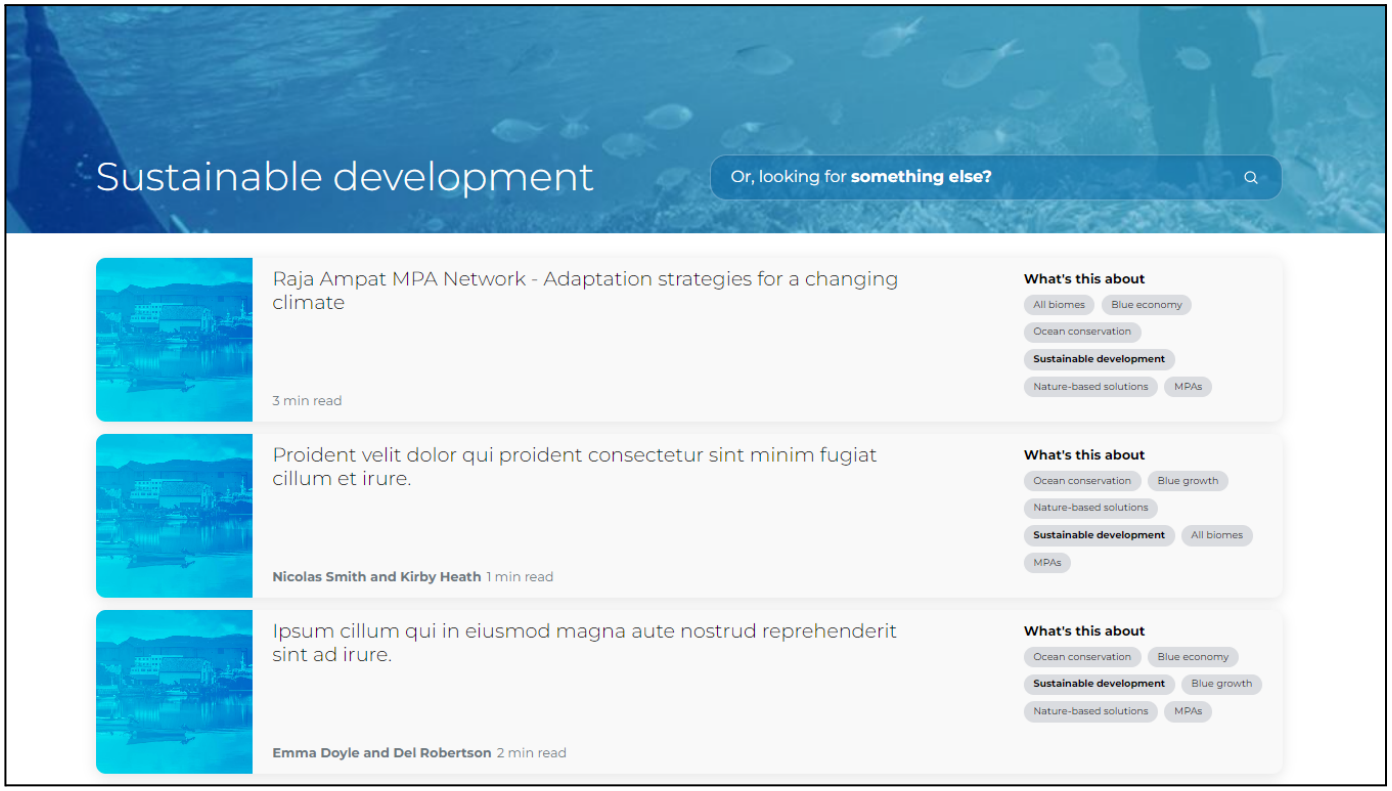


Figura 74. Collection pages.

10.4 Anàlisi i disseny Pages

Les pages recullen tots els Chapters o Case Studies del lloc web. Es pot dividir en dues pàgines diferents. Una principal on es mostren totes les pàgines del CMS i una segona que habilita l'edició de cada una d'elles.

La pàgina principal disposa d'un menú de cerca per tags i per text. A més, les pàgines es poden dividir en dos grups, un grup que pertany als esborranys, és a dir, pàgines que encara no s'han completat i un segon que pertany les pàgines en viu, és a dir, pàgines que ja són visibles per la resta d'usuaris (veure *Figura 75*).

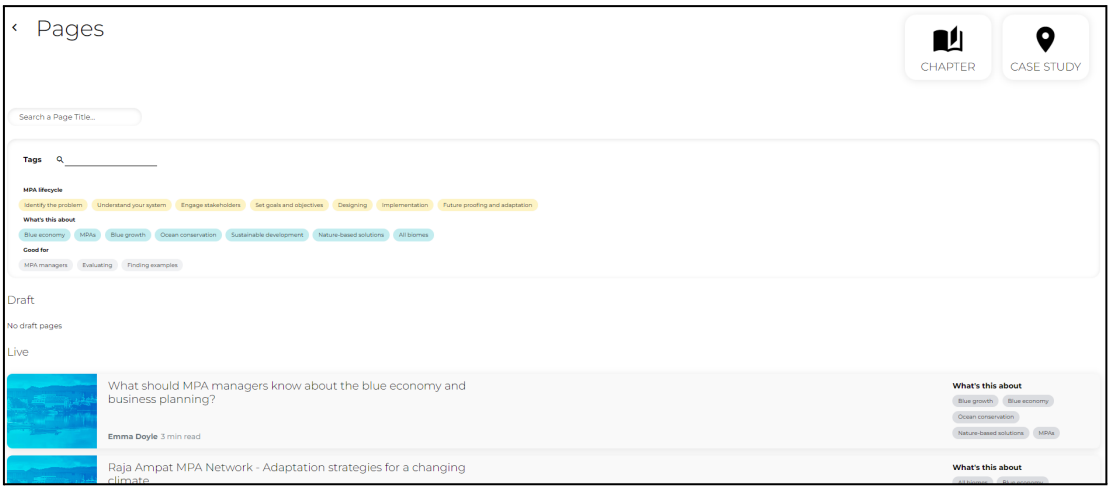


Figura 75. Pàgina principal de les pages.

La segona pàgina, que permet l'edició dels Chapters i Case Studies varia en funció del tipus que sigui. Com s'ha mencionat les capçaleres són diferents. A l'editor es pot observar una secció on es pot introduir text i cards i un altre per l'edició dels cards (veure *Figures 76, 77 i 78*).

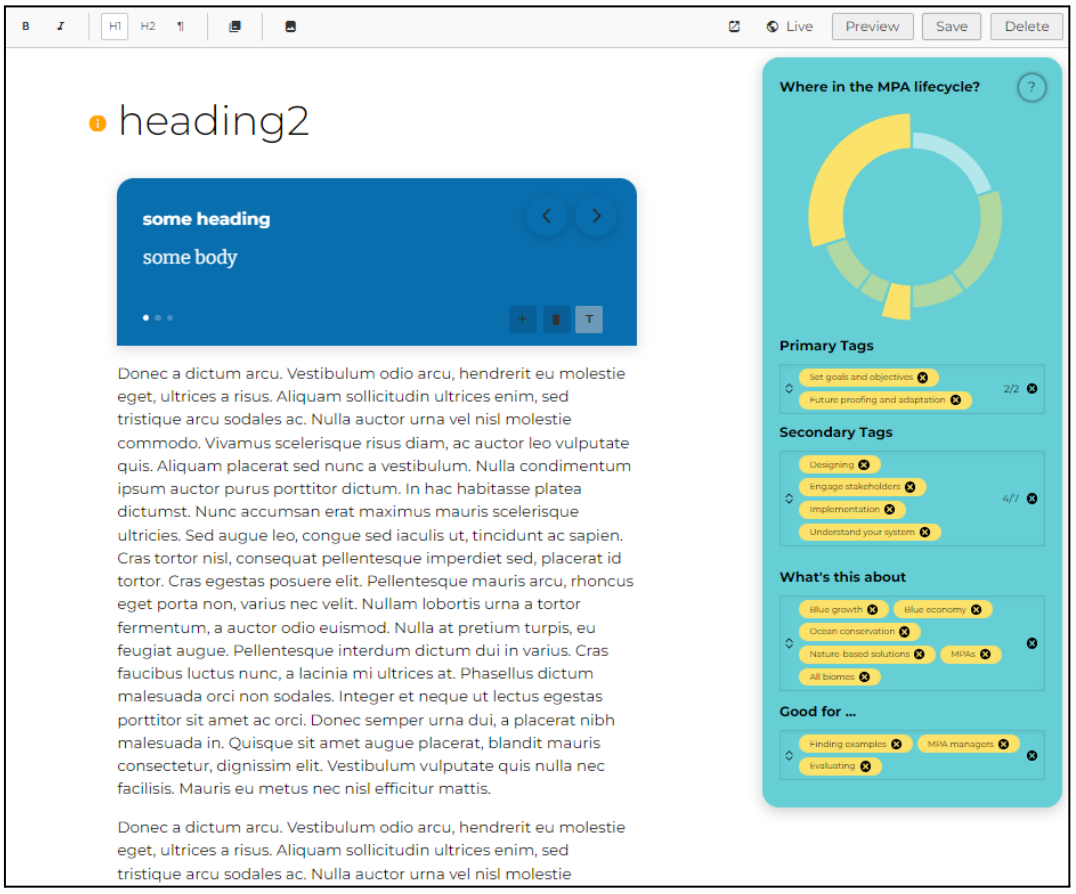


Figura 76. Editor de text i tags.

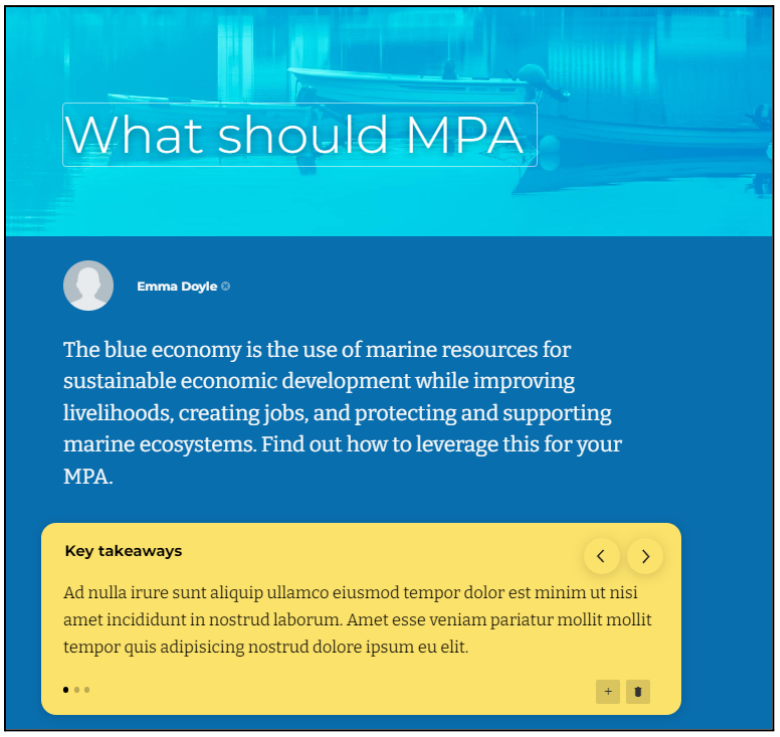


Figura 77. Editor de la capçalera del Chapters.

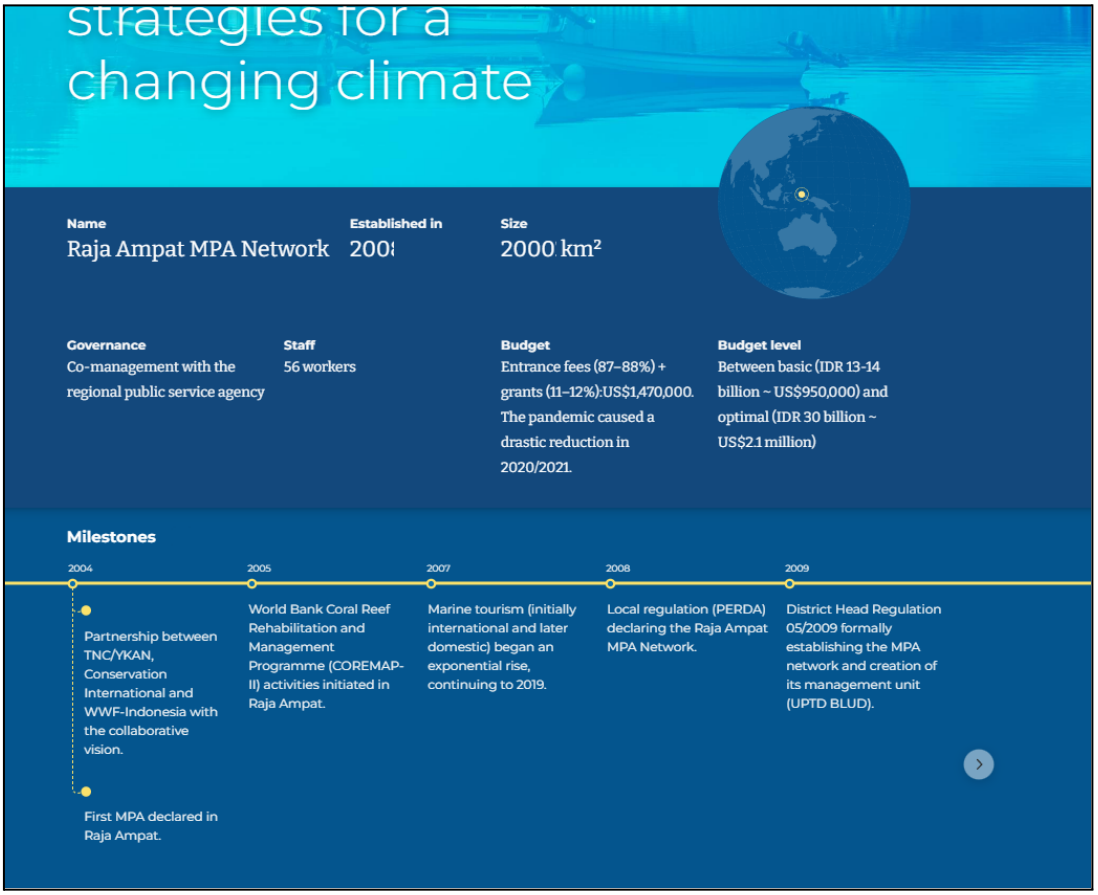


Figura 78. Editor de la capçalera dels Case Studies.

10.5 Anàlisi i disseny Tag Editor

L'editor de tags ens ha de permetre afegir, modificar i eliminar tags del tipus *topic*. Anem a veure amb la seva fitxa de casos d'ús i diagrama de flux com s'ha de comportar (veure *Figures 79 i 80*).

EDITOR DE TAGS	
DESCRIPCIÓ	Permet a l'usuari editar les tags de tipus *topic*.
ACTORS	Usuari
PRECONDICIÓ	Cap.
FLUX PRINCIPAL	1. Si l'usuari fa clic al botó afegir, LLAVORS: 1.1. Es desactiva el botó afegir. 1.2. S'afegeix una nova tag. 1.3. Es fa focus a la nova tag. 1.4. Apareix botó de cancel·lar. 1.5. Apareix botó de confirmar. 1.6. Si l'usuari modifica la tag, LLAVORS: 1.6.1. El text de la tag es modificat. 1.7. Si l'usuari prem el botó de cancel·lar, LLAVORS: 1.7.1. La tag s'esborra. 1.8. Si l'usuari prem el botó de confirmar, LLAVORS: 1.8.1. La tag es guarda. 1.8.2. La tag està unfocused. 1.9. Si l'usuari fa unfocus de la tag, LLAVORS: 1.9.1. La tag es borra. 2. Si l'usuari busca a la barra de cerca, LLAVORS: 2.1. Es filtren les tags en funció del text buscat. 3. Si l'usuari apreta el botó de borrar d'una tag, LLAVORS: 3.1. Si la tag està en més d'una pàgina, LLAVORS:

	3.1.1. S’obre modal de confirmació: 3.1.1.1. Si l’usuari confirma, LLAVORS: 3.1.1.1.1. La tag es esborrada. 3.2. Si la tag no està en cap pàgina, LLAVORS: 3.2.1. La tag es borrada. 4. Si l’usuari apreta una tag, LLAVORS: 4.1. La tag es focalitza. 4.2. Apareix botó de cancel·lar. 4.3. Apareix botó de confirmar. 4.4. Si l’usuari modifica la tag, LLAVORS: 4.4.1. El text de la tag es modificat. 4.5. Si l’usuari prem el botó de cancel·lar, LLAVORS: 4.5.1. La tag torna en el seu estat original. 4.5.2. La tag està unfocused. 4.6. Si l’usuari prem el botó de confirmar, LLAVORS: 4.6.1. La tag es guarda. 4.6.2. La tag està unfocused
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L’usuari ha pogut editar les tags del tipus topic.

Figura 79. Fitxa casos d'ús de l’editor de tags.

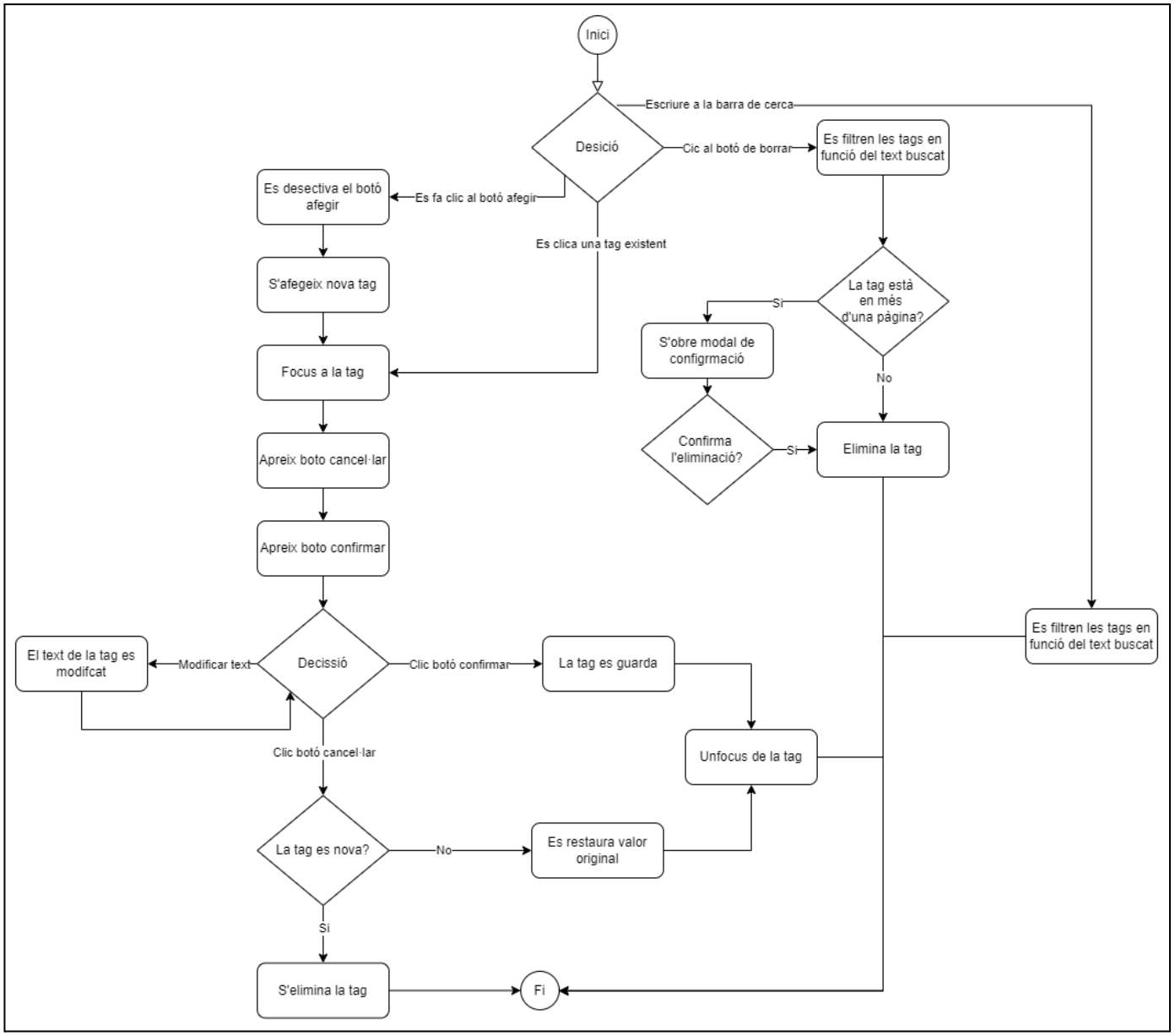


Figura 80. Diagrama de flux de l'editor de tags.

10.6 Anàlisi i disseny Page Ordering

La Page Ordering està destinada a reordenar els components de la pàgina principal. La pàgina principal està formada per 5 elements. Madlib, Chapters, Case Studies, Lifecycle i Searchbar. El funcionament serà drag-drop. A continuació si mirem les *Figures 81 i 82* podem veure les seves corresponents fitxes de casos d'ús i diagrama de flux.

PAGE ORDERING	
DESCRIPCIÓ	Permet a l'usuari re-ordenar el contingut de la landing pages.

ACTORS	Usuari
PRECONDICIÓ	Cap.
FLUX PRINCIPAL	1. Si l'usuari arrossega un element a una altre posició, LLAVORS: 1.1. Intercanvien posicions i s'actualitza l'ordre. 2. Si l'usuari clica el botó de guardar, LLAVORS: 2.1. Guarda els canvis a la base de dades.
FLUX ALTERNATIU	Cap.
POSTCONDICIÓ	L'ordre de la landing page ha estat modificat.

Figura 81. Fitxa casos d'ús del la pàgina d'ordenació

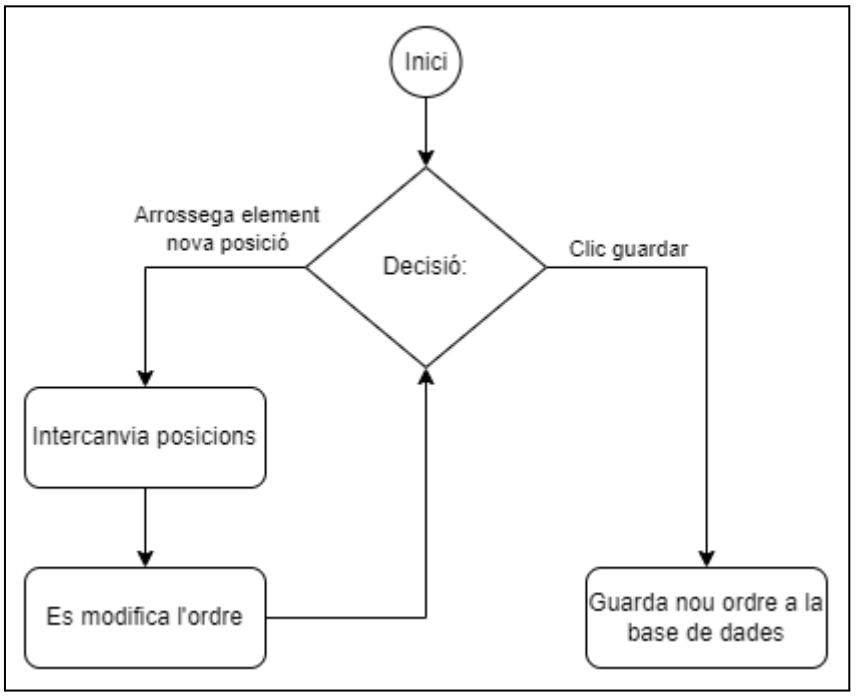


Figura 82. Diagrama de flux de la pàgina d'ordenació.

10.7 Anàlisi i disseny Authors

La pàgina d'autor ha de permetre afegir, eliminar i modificar autors, les propietats que tenen són la imatge de perfil, el nom i la biografia (veure *Figura 83*).

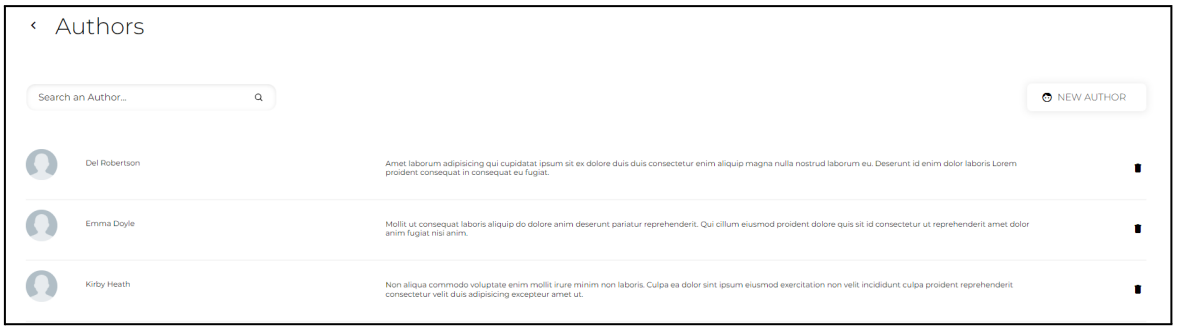


Figura 83. Pàgina d'edició d'autors.

10.8 Anàlisi i disseny Users

La pàgina d'autor ha de permetre eliminar i modificar usuaris, les propietats que es poden modificar són el nom i el seu rol corresponent (veure *Figura 84*).

Figura 84. Pàgina d'edició d'usuaris.

11. Implementació, proves i resultats

Aquest punt se centrarà a explicar com s'ha implementat cada component o pàgina de l'aplicació realitzat per mi. Tots els components realitzat per els altres membres de l'equip no s'han inclòs a la memòria.

11.1 Components

Aquest punt està enfocat a la implementació dels components Svelte que s'han realitzat al transcurs del projecte.

11.1.1 CircularSegment

Per tal de poder generar un segment és necessari que es passi com a paràmetre la variable data la qual és de tipus Segment (veure *Figura 54*). Aquest component funciona amb sincronia amb el CircleMenu. És important tenir això present, ja que requereix una

configuració que és recuperada en el component a través d'una funció de Svelte anomenada getContext (veure *Figura 85*).

```
const circleConfig = getContext<CircleConfig>('circleConfig');
```

Figura 85. Recuperació de les dades de configuració del CircleMenu.

La finalitat de la configuració del CircleMenu és saber l'amplada de cada segment, el gap que hi ha entre segments, la mida del cercle complet i el radi. Tots aquests paràmetres de configuració es fan servir a la funció describeArc que és l'algorisme més rellevant per aquest component, ja que és el que s'encarrega de generar el segment del cercle (veure *Figura 86*).

```
const describeArc = (segment: Segment, type: keyof
CircleConfig['thicknesses']) => {
    const thickness = circleConfig.thicknesses[type];
    const startAngle = segment.startAngle + circleConfig.gap / 2;
    const endAngle = segment.endAngle - circleConfig.gap / 2;
    const center = circleConfig.size / 2;

    const innerStart = polarToCartesian(center, center,
circleConfig.radius, endAngle);
    const innerEnd = polarToCartesian(center, center,
circleConfig.radius, startAngle);
    const outerStart = polarToCartesian(center, center,
circleConfig.radius + thickness, endAngle);
    const outerEnd = polarToCartesian(center, center, circleConfig.radius
+ thickness, startAngle);
    const largeArcFlag = endAngle - startAngle <= 180 ? '0' : '1';
    return [
      'M',
      outerStart.x,
      outerStart.y,
      'A',
      circleConfig.radius + thickness,
      circleConfig.radius + thickness,
      0,
      largeArcFlag,
```

```
      0,
      outerEnd.x,
      outerEnd.y,
      'L',
      innerEnd.x,
      innerEnd.y,
      'A',
      circleConfig.radius,
      circleConfig.radius,
      0,
      largeArcFlag,
      1,
      innerStart.x,
      innerStart.y,
      'L',
      outerStart.x,
      outerStart.y,
      'Z'
    ].join(' ');
  };
```

Figura 86. Algorisme de generació de segments.

A continuació explicaré el funcionament d'aquest algorisme. Per això definiré cada una de les variables que es defineixen per tenir una visió més clara del seu funcionament.

- **thickness**: Correspon a l'amplada del segment, és recuperada de la configuració del CircleMenu.
- **startAngle**: Correspon a l'angle d'inici proporcionat en la variable data. En cas que hi hagi un gap se suma la meitat d'aquest espai(l'altra meitat està a l'angle de fi).
- **endAngle**: Correspon a l'angle de fi proporcionat en la variable data. En cas que hi hagi un gap se suma la meitat d'aquest espai(l'altra meitat està a l'angle d'inici).
- **center**: Correspon al centre del cercle, obtingut en la variable de configuració circleConfig.
- **innerStart**: Correspon al punt interior inicial. Es realitza una conversió de coordenades polars a cartesianes.

- **innerEnd**: Correspon al punt interior final. Es realitza una conversió de coordenades polars a cartesianes.
- **outerStart**: Correspon al punt exterior inicial. Es realitza una conversió de coordenades polars a cartesianes.
- **outerEnd**: Correspon al punt exterior inicial. Es realitza una conversió de coordenades polars a cartesianes.
- **largeArcFlag**: És un flag que indica si el segment que es crea és llarg.

L'objecte de retorn correspon a un array que conté diverses propietats en la sintaxis de dibuix d'elements SVG. Per tal de dibuixar cada Arc segueix la següent seqüència de paràmetres de configuració: M A L A L Z. Anem a explicar quina és la funció de cada paràmetre.

- **M - Move**: Es mou en les coordenades indicades, requereix dos paràmetres, les coordenades X i Y.
- **A - Arcs**. És una secció d'un cercle o elipse, necessita un radi x i un radi y, en el nostre cas volem un cercle pel qual el radi de x serà el mateix que y. El tercer paràmetre és la rotació de l'eix x, com que nosaltres el volem recte serà 0. El quart paràmetre consisteix en una flag que indica si l'arc és major o menor a 180 graus. El quart paràmetre és una flag que indica si l'arc s'ha de moure en angles positius o negatius i finalment les coordenades x y.
- **L - Line**. Traça una línia entre els punts x i y.
- **Z.** Dibuixa una línia des de l'últim punt fins al primer.

Per tant, podem veure que el que fa l'algorisme és el següent:

1. M - Se situa a les coordenades exteriors inicials del segment.
2. A - Dibuixa un arc de les coordenades exteriors inicials a les coordenades exteriors finals.
3. L - Dibuixa una línia de les coordenades exteriors inicials a les coordenades interiors finals.
4. A - Dibuixa un arc de les coordenades interiors finals a les coordenades interiors inicials.

5. L - Dibuixa una línia de les coordenades interiors inicials a les coordenades exteriors inicials.
6. Z - Uneix final amb inici.

Per comprovar que el funcionament és el podem provar de generar un segment i que quan es fa hover o clic reacciona acord amb les propietats de configuració (veure *Figura 87 i 88*).

Figura 87. Generació segment de 90 graus amb estat unselected.

Figura 88. Interacció Hover amb el segment.

11.1.2 CircleMenu

Per tal de poder generar un menú circular és necessari que es passin els següents paràmetres:

- **data: MenuConfig[]**: Consisteix en els elements que formaran el menú, tenen dues propietats el percentatge del cercle que ocupen i el tipus de segment que són.
- **config**: Consisteix en tota la configuració del cercle (veure *Figura 50*).
- **currentPageIndex: number**: Consisteix en el índex de l'element seleccionat en el menú.
- **currentSegmentHovered: number**: Consisteix en l'índex de l'element que està "hovered" actualment.

L'algorisme més rellevant és el del càlcul dels segments, on posteriorment es farà servir per generar tots els segments del menú conjuntament amb el component CiruclarSegment (veure *Figura 89*). El seu funcionament consisteix a generar l'angle d'inici i angle de fi de cada segment en funció de la posició i el percentatge donat.

```
  const calcSegments = (menuElements: MenuElement[]) => {
    let currentAngle = 0;
    return menuElements.map<Segment>(({percentage, type}, i) => {
      const startAngle = currentAngle;
      const endAngle = currentAngle = startAngle + (360 * percentage) /
100;
      return { startAngle, endAngle, type, tagId: i};
    });
  };
```

Figura 89. Funció de generació dels segments.

Per veure el resultat d'implementació es pot observar la *Figura 47*.

11.1.3 LifeCycle

Els paràmetres necessaris per implementar aquest component són:

- **allTags: Tag[]**: Totes les tags del CMS.
- **tags: PageTag[]**: Les tags relacionades a la pàgina.
- **editable: boolean**: És un flag per habilitar l'edició de les tags.

Després també necessitem unes variables internes per poder controlar el seu funcionament:

- **currentTextHovered: number**: Indica quina de les tags tipus *stage* està sent hovered, aquesta variable estarà lligada al component CircleMenu.
- **LIFECYCLE_CONFIG: number[]**: correspon es percentatges del circle menú, la posició coincideix amb l'id de la tag.
- **LIFECYCLE_TEXT: string[]**: És el text que es mostra com ajuda quan es fa hover un element del CircleMenu, és a dir, una breu descripció de les tags *stage*.
- **MAX_PRIMARY_TAGS: number**: Són el nombre màxim de tags primàries habilitades en el mode edició, només s'aplica a les tags tipus *stage*.
- **MAX_SECONDARY_TAGS: number**: Són el nombre màxim de tags secundàries habilitades en el mode edició, només s'aplica a les tags tipus *stage*.
- **fitTextOptions: object**: Són les opcions de configuració pel text informatiu quan es fa hover sobre un element.

A continuació adjunto el codi del block de TypeScript que s'encarrega de gestionar tot el seu funcionament, controla les tags seleccionades en el mode edició i també inicialitza alguns dels components necessaris (veure *Figura 90*).

```
<script lang='ts'>
  import type { SubTypes, Tag } from '$lib/types';
  import { groupBy } from '$lib/helpers/utils';
  import MultiSelect, { Option } from 'svelte-multiselect';
  import CircleMenu, { type MenuElement } from './CircleMenu.svelte';
  import TagContainer from  './TagContainer.svelte';
  import textFit from 'textfit';
  import { afterUpdate } from 'svelte';
  import HelpPopup from './HelpPopup.svelte';


  export let allTags: Tag[] = null;
  export let tags: SubTypes.PageTag[]; // binding (updated as tags are
changed)
  export let editable = false;

  let currentTagHovered: number;
  let infoTextElement: HTMLElement;

  const helpText = '<b>What is the MPA lifecycle</b>';
  const fitTextOptions = {
    alignVert: true,
    alignHoriz: true,
    multiLine: true
  };
  const LIFECYCLE_CONFIG = [20, 20, 10, 5, 5, 10, 30]; // index matches
tag id
  const LIFECYCLE_TEXT = [
    'Although there is no single recipe, there ways to optimize the
<b>design</b> of your MPA based on your specific needs.',
    '<b>Where does this fit in the MPA management lifecycle?</b>',
    '<b>Where does this fit in the MPA management lifecycle?</b>',
    '<b>Where does this fit in the MPA management lifecycle?</b>',
    '<b>Where does this fit in the MPA management lifecycle?</b>',
```

```
    '<b>Where does this fit in the MPA management lifecycle?</b>',
    '<b>Where does this fit in the MPA management lifecycle?</b>',
  ]; // index matches tag id

  const MAX_PRIMARY_TAGS = 2;
  const MAX_SECONDARY_TAGS = 7;

  type PageTagOption = SubTypes.PageTag & Option;

  // used internally to bind to multiselect & keep track of selected tags
  const selectedTagOptions = groupBy(
    tags.map<PageTagOption>(t => ({value: t.tag.id, label: t.tag.value,
tag: t.tag, category: t.category})),
    t => t.tag.type === 'STAGE' ? t.category : t.tag.type
  );

  const allPageTagOptions = editable && allTags.map<PageTagOption>(
    tag => ({ value: tag.id, label: tag.value, tag, category: 'PRIMARY' })
  );
  const groupedOptions = editable && groupBy(allPageTagOptions, ({tag}) =>
tag.type);

  $: selectedStageTagIds = new Set(
    [...tags.filter(({tag}) => tag.type === 'STAGE').map(({tag}) =>
tag.id)]
  );

  $: availableStageOptions = editable && groupedOptions.STAGE
    .filter(({tag}) => !selectedStageTagIds.has(tag.id));

  $: if (editable) {
    tags = Object.values(selectedTagOptions).flat().map(o =>
      ({ tag: o.tag, category: o.category })
    );
  }

  function sortAndGroupTags(_tags: typeof tags) {
    // sort primary tags to the top and sort by id so it's consistent
    const tagSortVal = (t: SubTypes.PageTag) => (t.category === 'PRIMARY'
? 0 : 100) + t.tag.id;
```

```
    const sortedTags = _tags.sort((a,b) => tagSortVal(a) - tagSortVal(b));
    return groupBy(sortedTags, t => t.tag.type);
  }

  $: renderTags = !editable && sortAndGroupTags(tags);

  $: menuData = LIFECYCLE_CONFIG.map<MenuElement>((percentage, i) => {
    const category = tags.find(({tag}) => tag.id === i)?.category;
    return {
      percentage,
      type: category === 'PRIMARY' ? 'main' : category === 'SECONDARY' ?
'secondary' : 'unselected'
    };
  });


  afterUpdate(() => {
    if(infoTextElement) textFit(infoTextElement, fitTextOptions);
  });
</script>
```

Figura 90. Bloc de Typescript del component LifeCycle.

Svelte proporciona una eina que permet fer que s'executi codi quan una variable dins de l'etiqueta indicada és modificada. Si observem en el codi, hi ha diverses línies amb l'etiqueta $: indica que és un bloc reactiu i que quan qualsevol variable que estigui dins d'aquest sigui modificada s'executarà tot aquest bloc. La majoria de les funcions reactives funcionen en el mode edició d'aquest component. Per exemple, la variable menuData és actualitzada cada vegada que les opcions de les tags principals en el mode edició són modificades, fent que es generi un nou CircleMenu conforme a les tags principals i secundàries que són escollides.

Per tal de poder tenir un component que funcioni en mode visualització i edició faig servir una funcionalitat de Svelte que et permet utilitzar condicionals en el codi HTML (realment no és codi HTML, posteriorment és compilat i transformat amb HTML, JS i CSS). Aquesta és la sentència {#if} que et permet renderitzar components a partir d'una condició booleana. Si observem la *Figura 91*, correspon al codi HTML. Podem veure que

hi ha blocs {#if}/{:else} on validen el valor editable, fent que el component es renderitzi d'una forma o un altre depenent del mode escollit.

```
<div class='lifecycle'>
  <div class='top-section'>
    <h5 class='title'>Where in the MPA lifecycle?</h5>
    <HelpPopup text={helpText}>
      <h3>What is the MPA lifecycle?</h3>
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Gravida
laoreet leo scelerisque aliquet mattis malesuada turpis volutpat.
Ultrices lectus suspendisse sed pharetra. Proin elementum lacus volutpat
felis, nulla convallis aenean faucibus. Ante nisi, volutpat pretium diam
porta eget. Egestas tempor, risus tortor malesuada. Mus et cras risus
dictum. Quisque sollicitudin nisi, feugiat aenean.
      </p>
      <a href="/">More about this framework</a>
    </HelpPopup>
  </div>
  <div class="circle-menu-section">
      <div class="circle-menu">
          {#if currentTagHovered != null}
            <div class="info-text" bind:this={infoTextElement}>
              {@html LIFECYCLE_TEXT[currentTagHovered]}
            </div>
          {/if}
          <CircleMenu data={menuData}
bind:currentSegmentHovered={currentTagHovered}/>
      </div>

      <div class="tag-container">
        {#if editable}
          <div class="subtitle">Primary Tags</div>
          <MultiSelect
            bind:selected={selectedTagOptions.PRIMARY}
            options={availableStageOptions}
            maxSelect={MAX_PRIMARY_TAGS}
          />
          <div class="subtitle">Secondary Tags</div>
```

```
          <MultiSelect
            bind:selected={selectedTagOptions.SECONDARY}
            options={availableStageOptions.map(o => ({...o, category:
'SECONDARY'}))}
            maxSelect={MAX_SECONDARY_TAGS}
          />
        {:else}
          <TagContainer tags={renderTags.STAGE} bind:currentTagHovered/>
        {/if}
      </div>
  </div>
  <div class="bottom-section">
    <h5 class='title'>What&apos;s this about</h5>
    <div class="tag-container">
      {#if editable}
        <MultiSelect bind:selected={selectedTagOptions.TOPIC}
options={groupedOptions.TOPIC} />
      {:else}
        <TagContainer tags={renderTags.TOPIC} />
      {/if}
    </div>
    <h5 class='title'>Good for ...</h5>
    <div class="tag-container">
      {#if editable}
        <MultiSelect bind:selected={selectedTagOptions.USER}
options={groupedOptions.USER} />
      {:else}
        <TagContainer tags={renderTags.USER}/>
      {/if}
    </div>
  </div>
</div>
```

Figura 91. Codi HTML del component LifeCycle.

Per tal de fer el mode d'edició utilitzo un paquet de node JS anomenat svelte-multiselect el qual et permet tenir un dropdown amb la possibilitat d'afegir i treure elements (Afegir/Eliminar tags). A més també disposa del component HelpPopup que s'explica en

un altre punt, resumint, quan l'usuari fa hover en ell es mostra un text informatiu per guiar a l'usuari.

El resultat d'aquest component implementat correspon als de les *Figures 92, 93, 94, 96, 97*. Per tal de comprovar que el seu funcionament és el correcte s'han provat diverses visualitzacions amb diferents configuracions, per al mode edició, per comprovar el funcionament en el mode edició s'ha fet diverses assignacions de tags on posteriorment hi han set guardats en la base de dades i visualitzats.

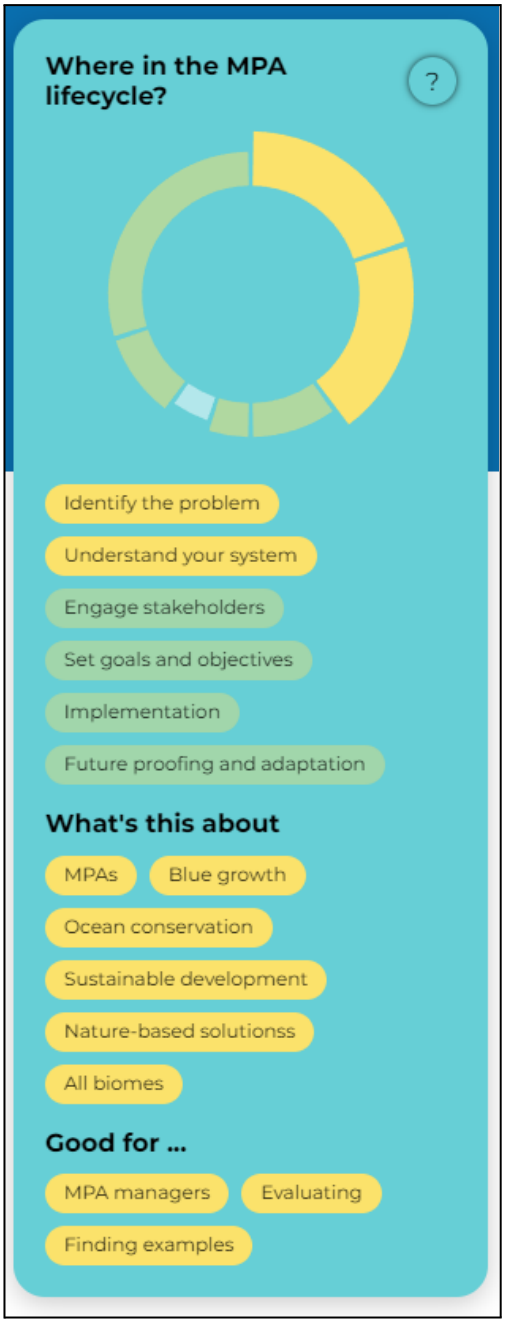


Figura 92.Visualització Desktop.

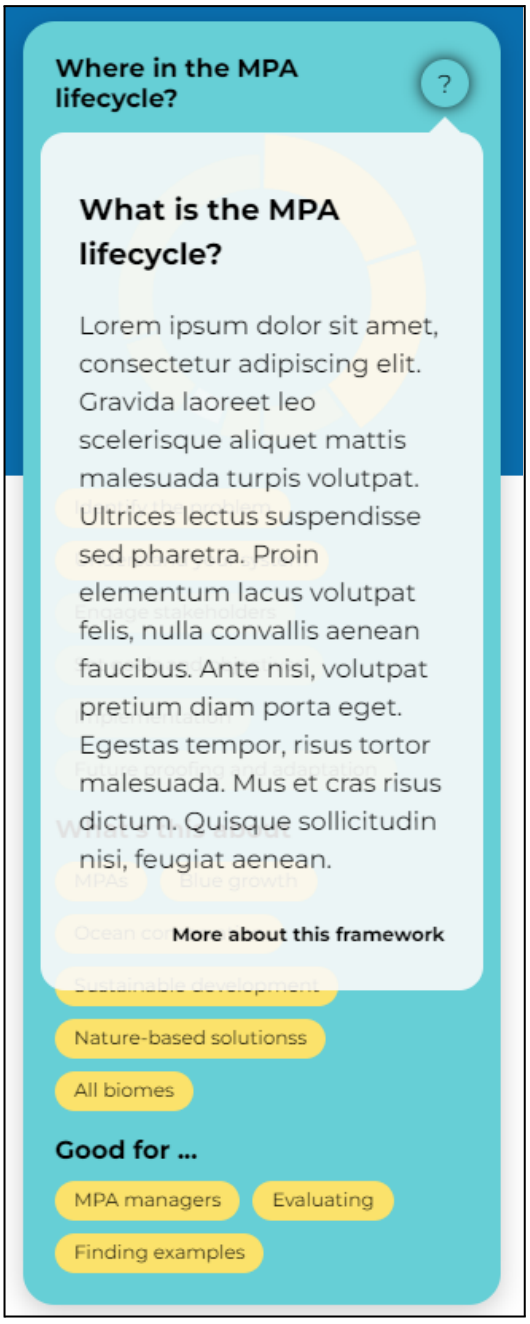


Figura 93. Visualització Desktop amb ajuda.

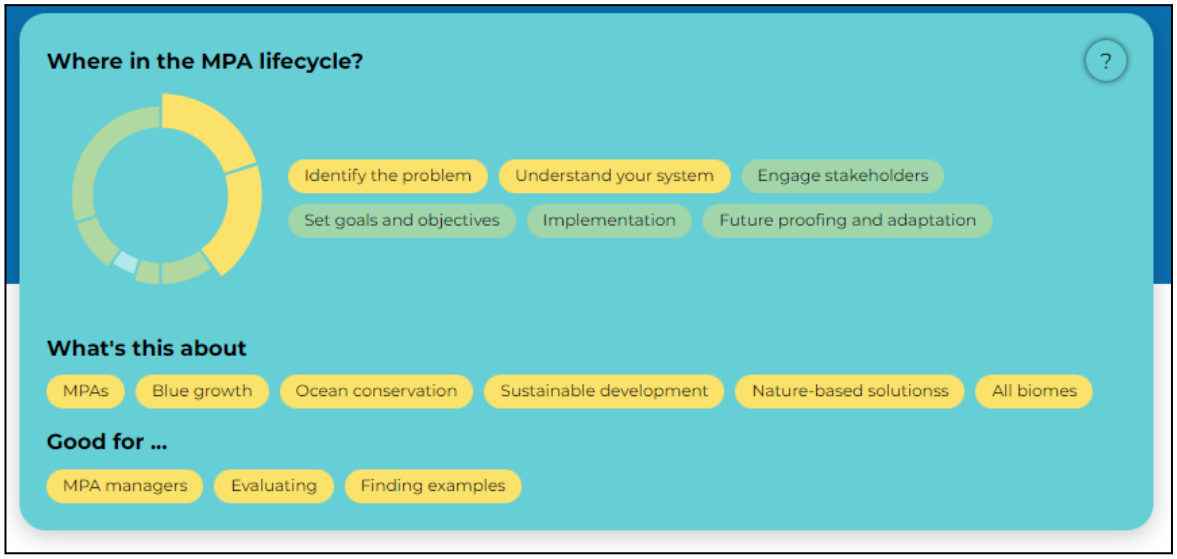


Figura 94. Visualització mòbil.

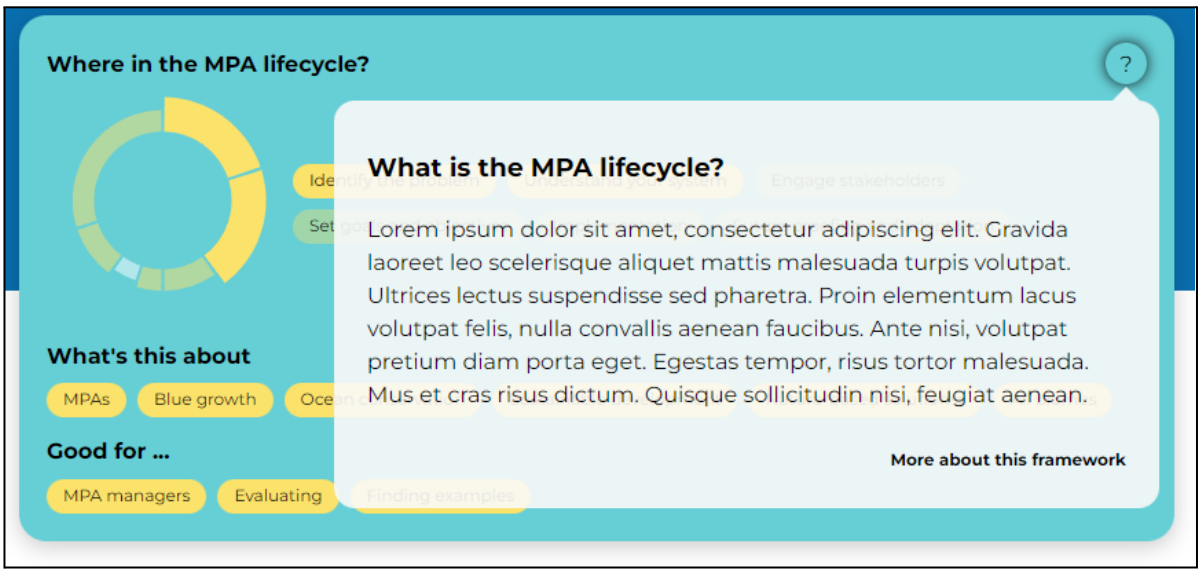


Figura 95. Visualització mòbil amb ajuda

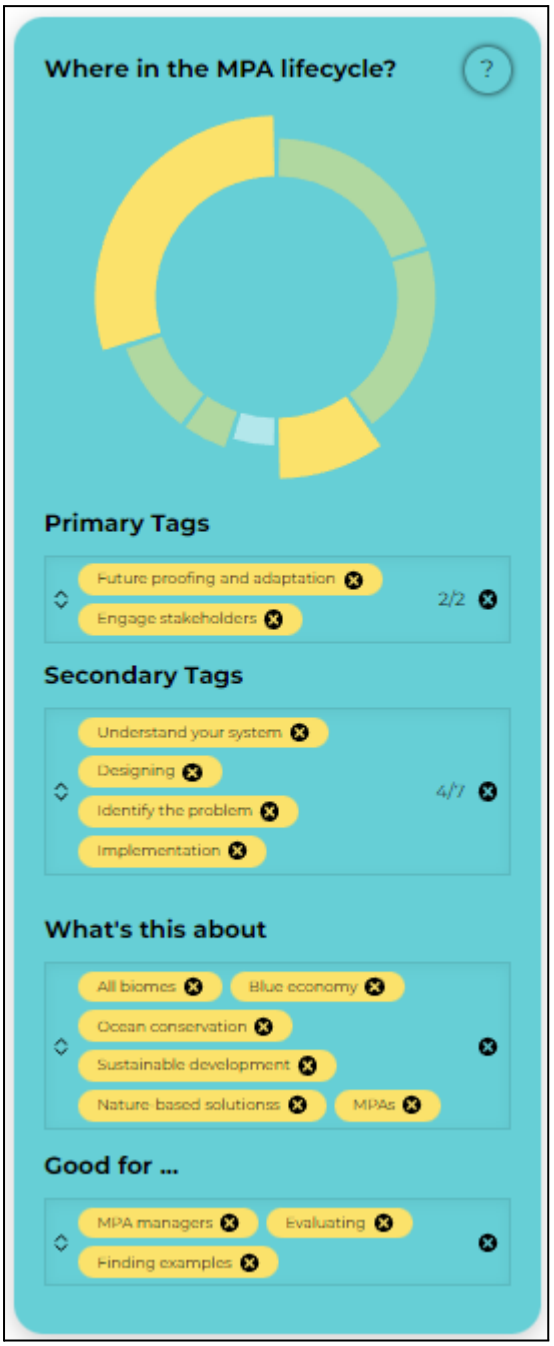


Figura 96. Visualització en mode edició

11.1.4 HelpPopup

Per la implementació d'aquest component es fa servir una funcionalitat de Svelte anomenada slots, els slots permeten passar un component o qualsevol codi HTML i posteriorment es pot tractar en el component pare. Aquesta mecànica permet fer que el component aquest sigui reutilitzable fent que el component que es passa en el slot sigui el contingut que es mostrarà en el Popup (veure *Figura 97*).

```
<div class="help-popup" data-text={false}>
  ?
  <div class="popup-body">
    <slot/>
    <slot class="bottom-link" name="bottom-link" />
  </div>
</div>
```

Figura 97. Implementació del HelpPopup.

Per comprovar el seu funcionament es procedeix a fer hover sobre la icona (?) i veure que el text d'ajuda es mostra correctament, en funció del que s'ha enviat al slot (veure *Figura 98*).

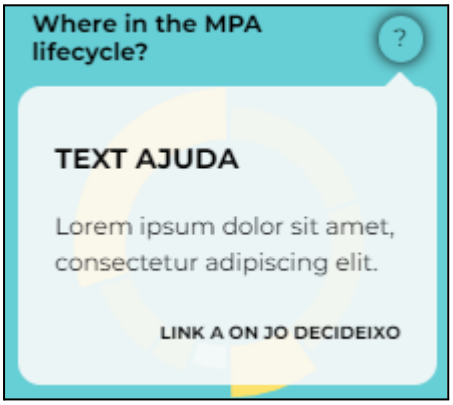


Figura 98. Resultat del HelpPopup.

11.1.5 Dot

Els paràmetres de funcionament del component Dot són els següents:

- **active: boolean**: Indica si el punt està actiu o no.
- **progress: boolean**: Indica si el punt té una animació.

Els punts més interessants en la implementació d'aquest component és la gestió d'esdeveniments per indicar que l'animació d'esdeveniments ha acabat. Abans de procedir en com gestiona els esdeveniments és important saber com funcionen les animacions en CSS (veure *Figura 99*).

```
<style lang="stylus">

  $defaults = {
    color: black,
    size: 7px,
    progress-duration: 10s,
    fade: 0.25
  };

  .dot {
    position: relative;
    border-radius: var(--dot-size, $defaults.size);
    overflow: hidden;
    transition: opacity 100ms ease, height 100ms ease, flex 100ms ease;
    cursor: pointer;
    -webkit-tap-highlight-color: transparent;
    flex: 0 0 var(--dot-size, $defaults.size);
    height: var(--dot-size, $defaults.size);

    @keyframes dot-progressbar-animation {
      from { width: 0; }
      to { width: 100%; }
    }



    &.progress.active {
      flex: 0 1 200px;
      cursor: default;
      .dot-progressbar {
        animation: var(--dot-progress-duration,
$defaults.progress-duration) dot-progressbar-animation 0s linear;
      }
    }

  }



</style>
```

Figura 99. Animacions CSS del component Dot.

Si observem el Keyframe podem veure que l'animació consisteix a fer que el punt vagi creixent de zero al màxim de la seva amplada. El temps d'aquesta animació ve definit per la variable progress-duration la qual és de 10 segons. És important saber el temps d'aquesta animació, ja que implica el temps que cada card es mostrarà en el TextSlider. Si analitzem la classe .dot.progress.active podem veure que és on es defineix l'animació del punt.

Sabent com funciona l'animació podem utilitzar un esdeveniment de Svelte anomenat animationend que s'executa quan l'animació acaba. Aquí entre el lloc de l'EventDispatcher el qual es crea a l'inici del codi i en finalitzar l'animació llancem un esdeveniment personalitzat en el component pare. Això ens permet gestionar tot el control de l'índex actual (veure *Figura 100*).

```
<script lang="ts">
  import { createEventDispatcher } from 'svelte';

  export let active = false;
  export let progress = false;

  const dispatch = createEventDispatcher<{progressAnimationFinished:
null}>();

</script>

<div class="dot" class:active class:progress on:click>
  <div class="dot-progressbar" on:animationend={() =>
dispatch('progressAnimationFinished')} />
  <div class="dot-background" />
</div>
```

Figura 100. Codi del component Dot.

11.1.6 CarouselDots

Pel funcionament d'aquest component hi ha 3 paràmetres que permeten la seva configuració:

- **progress**: boolean: Indica si el carruselestarà animat.

- **pagesCount**: number: Indica el nombre de pàgines que hi haurà.
- **currentPageIndex**: number: Indica l'índex actual del carrusel.

La Implementació d'aquest component consisteix a generar pagesCount Dots i en funció dels paràmetres configurar els punts en mode estàtic o dinàmic. El component Dot ja està pensat per tenir aquest funcionament, per la qual cosa, només és necessari tenir un control de l'índex de pàgina actual en funció de si l'usuari fa clic a un punt o bé l'animació acaba, i, per tant, capturem l'esdeveniment progressAnimationFinished i assignem un nou índex de pàgina (veure *Figura 101*).

```
<script lang="ts">
  import Dot from './Dot.svelte';

  export let progress = false;
  export let pagesCount = 1;
  export let currentPageIndex = 0;
</script>

<div class="carousel-dots">
  {#each Array(pagesCount) as _, i}
    <Dot
      {progress}
      on:click={() => currentPageIndex = i}
      active={currentPageIndex === i}
      on:progressAnimationFinished={() => currentPageIndex = i >=
pagesCount - 1 ? 0 : i + 1}
    />
  {/each}
</div>

<style lang="stylus">

  .carousel-dots {
    display: flex;
    align-items: center;
    padding: 10px 30px;
    column-gap: 7px;
  }
```

```
</style>
```

Figura 101. Implementació del CarouselDots.

Per comprovar el seu correcte funcionament ho faré quan faci el testatge del component TextSlider, ja que està pensat per funcionar amb sincronia amb ell (veure *Figura 102*).

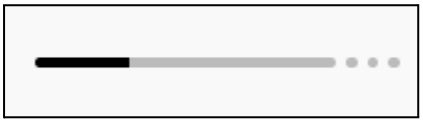


Figura 102. Resultat del CarouselDots

11.1.7 Cards

Aquest component va ser realitzat per mi inicialment, però posteriorment es van afegir un conjunt de funcionalitats pels meus companys, em centraré a explicar el que jo vaig implementar. Per tal de poder reutilitzar el component Cards hi ha disponibles un conjunt de paràmetres de configuració del component, aquests són:

- **cards: CardData[]**: Correspon a tots els cards que hi haurà en el component.
- **style: CardsStackStyle**: Correspon a l'estil dels cards, un card pot tenir o no títol, en funció d'això es definirà l'estil del card.
- **currenPageIndex**: number: Correspon al card actual, amb aquest paràmetre es pot tenir control del card actual i fer que funcioni amb sincronia amb altres components com el CircleMenu.
- **progress: boolean**: Flag per habilitar la visualització automàtica.

A continuació podem veure el tipus CardData i CardsStackStyle (veure *Figura 103*). CardData consta de dues propietats que corresponen al títol i el text del card. CardStackStyle pot tenir dos valors diferents, 'default' o 'no- heading'.

```
export type CardStackStyle = 'no-heading' | 'default';

export type CardData = {
```

```
  heading: string;
  body: string;
}
```

Figura 103. Tipus CardData i CardsStackStyle.

Per crear el component es va fer servir un paquet de nodejs auxiliar que permet crear carrusels de forma ràpida i personalitzada aquest paquet s'anomena splidejs. Splidejs ens ofereix una sèrie de configuracions, les que faig servir en aquest component són les que es mostren a la *Figura 104*. Aquests paràmetres indiquen que volem que el carrusel sigui en cicle, és a dir, un cop es passa l'última torni a la primera, que la seva altura sigui automàtica, que tingui un gap de -3 i que no inclogui la paginació, ja que per la paginació farem servir el nostre propi component (CarouselDots).

```
let options = SplideOptions({
    rewind: true,
    autoHeight: true,
    gap: -3,
    pagination: false
  });
```

Figura 104. Configuracions splidejs.

A continuació anem a veure com generar els cards. Per tal de poder tenir un control dels elements del carrusel hem de definir la propietat hasTrack a fals conjuntament amb el component SplideTrack que ens ofereix aquesta llibreria. Fent servir aquests elements, splidejs ens permet situar els elements de control com les fletxes o la paginació a qualsevol altre lloc que nosaltres vulguem del component. Dit això, si observem la *Figura 105*, podem veure l'estructura bàsica del component. Cal dir que es pot ignorar tots els condicionals relacionats amb l'edició d'aquest component, ja que, com he dit abans aquesta part ha estat feta pels meus companys.

```
<Splide {options} bind:this={splide} on:move={e => currentPageIndex =
e.detail.index} hasTrack={false}>
    {#if fixedTitle}
      <div class="fixed-title">
        <CardHeading text={fixedTitle} />
      </div>
    {/if}
    <SplideTrack>
      {#each cards as card, i}
        <SplideSlide>
          <div class="slide" class:no-heading={style==='no-heading'}>
            {#if style !== 'no-heading'}
              <CardHeading bind:text={card.heading} editable={editable &&
currentPageIndex === i} />
            {/if}
            <CardBody bind:text={card.body} editable={editable &&
currentPageIndex === i} />
          </div>
        </SplideSlide>
      {/each}
    </SplideTrack>
    {#if editable}
      <div class="editor-buttons">
        <IconButton icon="add" on:click={onClickAddCard} />
        <IconButton icon="delete" on:click={onClickRemoveCard}
disabled={!removable && cards.length === 1} />
        {#if canToggleHeading}
          <IconButton icon="title" active={style === 'default'}
on:click={onClickChangeType} />
        {/if}
      </div>
    {/if}
    {#if editable || cards.length > 1}
      <div class="carousel-dots card-bottom">
        <CarouselDots
          bind:currentPageIndex
          pagesCount={cards.length}
          progress={!editable && progress}
        />
      </div>
```

```
    {/if}
  </Splide>
```

Figura 105. Generació del carrusel de cards amb splidejs.

Podem observar que el component Splide fa servir les propietats slots que ofereix Svelte, on a dins nosaltres podem definir els elements desitjats. En cas que el card tingui un títol, podem veure que es renderitza un component Svelte anomenat CardHeading, el qual l'única funció que té és pintar el títol del card, aquest títol serà fix en tots els cards del carrusel (Implementat per un company). Seguidament, es defineix el component SplideTrack on a dins posarem el cos del card, recordem, que definim aquest component per tal de poder controlar els elements de control a criteri nostre. Per cada card passat com a paràmetre a través de la variable cards és renderitzat a través del bucle for. Per cada card és necessari utilitzar el component SplideSlide el qual correspon a una targeta del carrusel. A dins d'aquest component l'element que situem com a slot correspondrà a la visualització del card. Per tant, si analitzem el component, podem veure que es comprova l'estil d'aquests cards a través de la variable style, la qual ens indica si els cards tenen o no títol. En funció d'aquest paràmetre el títol serà renderitzat. L'altra part del card és el cos, el qual fa servir un component propi que veurem en el pròxim punt.

A l'inici d'aquest punt he dit que hi ha una variable anomenada currentPageIndex. Aquesta serveix per tenir control de quin és el card actual. Splidejs emet un esdeveniment quan un card es mou, per tant, aprofitem aquest esdeveniment per a actualitzar l'índex actual del card. Això és important, ja que com he explicat el punt anterior, el CarousselDots té un paràmetre que també actualitza l'índex cada vegada que es clica un punt o bé acaba una animació.

Podem fer que treballi amb sincronia amb el nostre component gràcies a una funcionalitat de Svelte que ens permet vincular variables entre els components. Es podria comparar amb un pas per referència en llenguatges de programació com C. La gràcia de vincular les variables és utilitzar la reactivitat de Svelte fent que en cada moment que aquesta canvia s'actualitzin tots els components respectius automàticament (veure *Figura 106*).

```
<CarouselDots
    bind:currentPageIndex
    pagesCount={cards.length}
    progress={!editable && progress}
/>
```

Figura 106. Vinculació dels cards amb el component CarouselDots.

Si observem la *Figura 107*, podem veure el resultat de la implementació i comprovar el seu funcionament fent clic a les fletxes o als punts i veure que el card s'actualitza acord amb l'índex indicat. També podem esperar a fer que l'animació del carrusel finalitzi i passi al card següent.

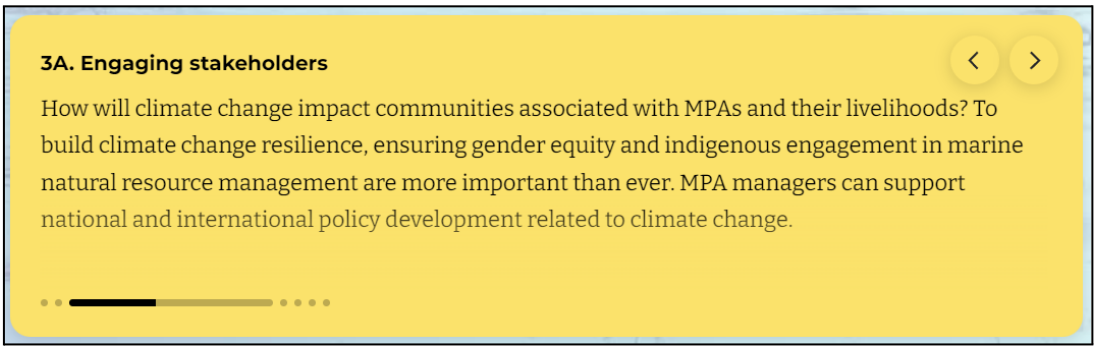


Figura 107. Resultat d'implantació de les Cards.

11.1.8 CardBody

Aquest component té dos mètodes de funcionament, igual que els cards pot estar habilitat en mode edició o mode visualització. Inicialment, la meva feina va ser la seva visualització, així que em centraré a explicar el seu funcionament bàsic. Només té un paràmetre, aquest és el text. La gràcia d'aquest component és que si hi ha més text del que pot ser vist es crea un gradient indicant que es pot fer "scroll down". Anem a veure com funciona.

Svelte ens dona una funcionalitat molt interessant Anomenada Actions. Les actions són funcions que són cridades quan s'ha creat un element. Per poder fer servir aquesta funcionalitat hem d'utilitzar la propietat use en una etiqueta html, a aquesta propietat se li

pot assignar una funció que en el nostre cas serà la que gestionarà tota la visualització del "scroll". Si observem la *Figura 108*, correspon a la funció gradientAction.

```
function gradientAction(node: HTMLDivElement) {
    const isScrollable = () => node.scrollHeight > node.clientHeight;
    const isAtBottom = () => Math.round(node.clientHeight +
node.scrollTop) >= node.scrollHeight;
    const updateGradient = () => showGradient = isScrollable() &&
!isAtBottom();
    node.addEventListener('scroll', updateGradient);
    window.setTimeout(() => updateGradient());
  }
```

Figura 108. Implementació de la funció gradientAction.

El funcionament d'aquesta funció consisteix a comprovar si l'element és "scrollable" i si s'ha arribat al final del "scroll". A més també es vincula un EventListener per tal que s'executi una funció cada vegada que s'emet l'esdeveniment scroll. Bàsicament, comprova que l'estat actual del text, és a dir, si compleix les condicions per mostrar gradient, i conseqüentment, actualitzar el flag que permet que es mostri aquest scroll.

A continuació podem observar la implementació del component, el qual gestiona dos esdeveniments que són mouseenter i scroll. Quan aquests esdeveniments es produeixen s'executa una funció anomenada showScroll que gestiona la visualització de la barra de scroll dins del component. També vinculem l'estil del gradient amb el flag que indica si s'ha de mostrar el gradient (veure *Figures 109, 110 i 111*).

```
  function showScroll() {
    hideScroll = false;
    if (scrollTimeout) window.clearTimeout(scrollTimeout);
    scrollTimeout = window.setTimeout(() => hideScroll = true, 2000);
  }
```

Figura 109. Funció que gestiona la visualització del scrollbar.

```
<div
  class="content"
  class:hide-scrollbar={hideScroll}
  class:gradient={showGradient}
  on:mouseenter={showScroll}
  use:gradientAction
  on:scroll={showScroll}
>
  <EditableText bind:value={text} {editable} placeholder='Body text...'
/>
</div>
```

Figura 110. Implementació del component CardBody.

```
.gradient {
    --gradient-color: $colors.highlight-1;
  }

  .gradient::before {
    content: '';
    position: absolute;
    width: calc((100%) - var(--content-padding) -
var(--content-right-padding) - var(--scrollbar-width));
    height: 76%;
    left: inherit;
    z-index: 10;
    background: linear-gradient(180deg, rgba(0, 0, 0, 0) 50%,
var(--gradient-color) 95%);
    transition: all 2s;
    pointer-events: none;
  }
```

Figura 111. Estil CSS que aplica el gradient a la card.

El resultat del gradient es pot observar a la *Figura 107* del punt anterior. També podem veure que s'elimina el gradient quan s'arriba al final del contingut del card (veure *Figura 112*).

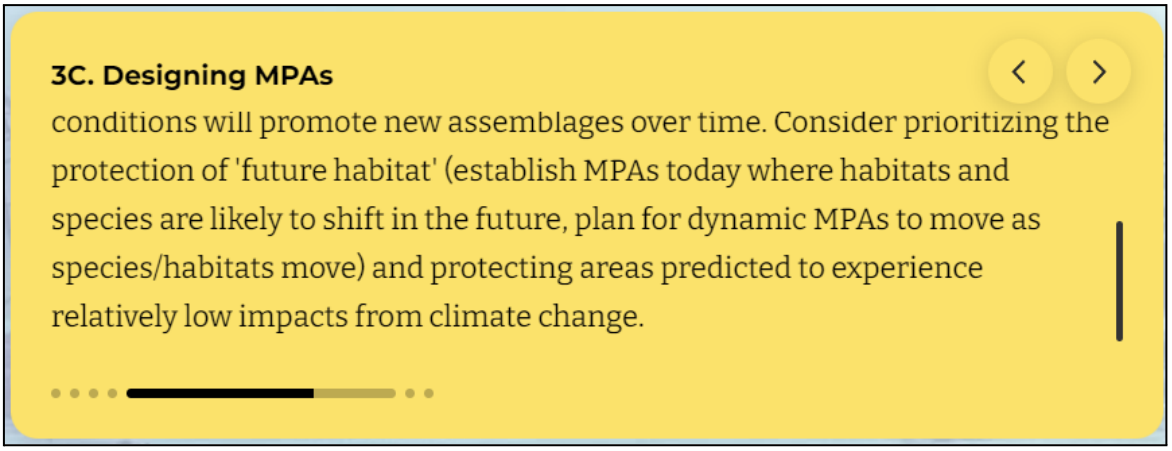


Figura 112. CardBody sense gradient al arribar al final del contingut.

11.1.9 ContentCarousel

Com s'ha comentat en el punt 11.1.7 Cards, s'ha utilitzat un paquet de nodejs per implementar la funcionalitat dels cards. El paquet també s'ha fet servir per aquest component. Per tal de fer funcionar aquest component hi ha dos paràmetres. Aquests són title el qual consisteix en un string que representa el títol del component i un segon anomenat slides el qual representen diverses pàgines de forma resumida. El punt més destacable d'aquestes pàgines és que contenen les tags. Si observem la *Figura 113*, podem veure el tipus de dada de la variable slides.

```
type Page.ContentCard = {
    id: number;
    slug: string;
    title: string;
    img: string;
    tags: {
        tag: {
            id: number;
            value: string;
            type: TagType;
        };
        category: TagCategory;
    }[];
```

Figura 113. Tipus ContentCard.

En utilitzar splidejs també s'ha hagut de definir un conjunt de paràmetres per configurar el seu component. A continuació faig una llista amb tots els paràmetres usats. Es pot complementar la informació observant la *Figura 114*, que correspon als valors dels paràmetres per aquest component.

- **type**: Defineix el tipus de carrusel.
- **rewind**: Determina si el carrusel rebobina.
- **perMove**: Determina el nombre de slides que es mouen per moviments.
- **gap**: Determina l'espai que hi ha entre slides.
- **autoWidth**: Si és cert, l'amplada de les slides està determinada per la seva amplada.
- **pagination**: Determina si es crea la paginació en el carrusel.
- **padding**: Permet definir el padding pel carousel.
- **noDrag**: El selector dels nodes no pot ser arrestrat.
- **breakpoints**: Conjunt d'opcions pel disseny responsiu.

```
const options = SplideOptions({
    type: 'slide',
    rewind: false,
    perMove: 2,
    gap: 30,
    autoWidth: true,
    pagination: false,
    padding: { right: '13%' },
    noDrag: '.no-drag, .no-drag *',
    breakpoints: {
      1024: {
        perMove: 1
      },
      768: {
        autoWidth: true,
        gap: 50,
        padding: {left: '22.5%', right: '0%'}
      }
    }
  });
```

Figura 114. Configuració del carousel en el component ContentCarousel.

La Implementació d'aquest component és molt semblant a la del component Pages. Per aquesta mateixa raó no entraré tan en detall en explicar-la. La principal diferència és que no utilitzem el component SplideTrack (veure *Figura 115*).

```
<Splide {options}>
    {#each slides as slide}
      <SplideSlide>
        <ContentCarouselCard page={slide}/>
      </SplideSlide>
    {/each}
  </Splide>
```

Figura 115. Implementació del carousel en el component ContentCarousel.

A continuació podem observar el resultat d'aquest component. Per comprovar el seu correcte funcionament s'han d'assignar un conjunt de slides i comprovar que pots navegar entre elles i que tots els paràmetres de configuració actuen com hi haurien (veure *Figura 116*).

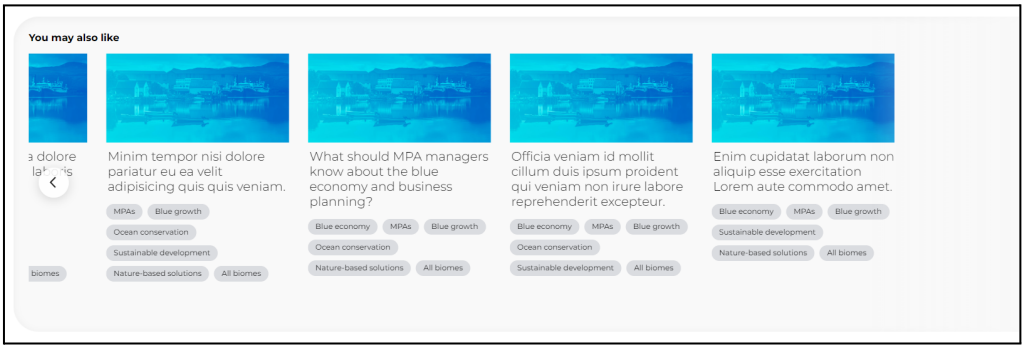


Figura 116. Resultat del component ContentCarousel.

11.1.10 StickyMenu

Aquest component utilitza una llibreria de tercers que va ser utilitzada i modificada per mi en un projecte anterior, originalment la llibreria estava escrita amb JavaScript pla, però, la vaig reescriure a TypeScript. Durant la implementació d'aquest component vaig observar un mal funcionament i vaig haver de fer unes petites correccions. El funcionament del component StickyMenu està pensat perquè sigui automàtic en cada Chapter o Case Study, és a dir, a partir de les seccions dels Chapters es generin les opcions dels menús. Disposa d'un paràmetre anomenat menuOptions el qual és un array

d'objectes formats per un id i un títol. A partir d'aquest paràmetre es generen totes les entrades en el codi HTML (veure *Figura 117*).

```
<nav class="sticky-menu" bind:this={nav}>
  {#each menuOptions as { id, title }, i}
    <div class="menuoption" class:active={i === 0} on:click={() =>
scrollToFunction(id, i)}>
      <div href="#{id}">
        {title}
      </div>
    </div>
  {/each}
</nav>
```

Figura 117. Generació del menú.

A la figura anterior hi ha algunes coses a destacar. Es pot veure que l'element HTML nav es vincula a una variable anomenada nav. Això es fa per poder utilitzar la llibreria mencionada anteriorment (MenuSpy). Primer de tot és necessari definir tots els paràmetres de configuració que estan a la variable msParams (veure *Figura 118*).

```
let msParams: MenuSpyParams = {
    menuItemSelector: '[href^="#"]',
    activeClass: 'active',
    threshold: 150,
    enableLocationHash: false,
    hashTimeout: 0,
    callback: null
};
```

Figura 118. Configuració del MenuSpy.

A continuació podem veure una descripció de cadascun dels paràmetres.

- **menuItemSelector**: Selector dels elements del menú.
- **activeClass**: Classe que s'aplica a l'element del menú relatiu a la secció visible actual.

- **threshold**: Quantitat d’espai entre el menú i la següent secció a ser activada.
- **enableLocationHash**: Habilita el canvi d’ubicació del hash del navegador.
- **hashTimeout**: Temps d’espera per aplicar la ubicació del hash del navegador.
- **callbakck**: Funció a ser cridada cada vegada que un nou element del menú és activat.

Definits els paràmetres s'ha de crear el component, això ho faig a la funció onMount que s'executa un cop els components han estat renderitzats (veure *Figura 119*).

```
onMount(() => (ms = new MenuSpy(nav, msParams)));
```

Figura 119. Inicialització del MenuSpy.

És necessari que les seccions estiguin vinculades amb el mateix id que el menú per tal que MenuSpy pugui reconèixer en quina secció està actualment, dins del component Section es defineix aquest id.

L'altra cosa a destacar és que s'assigna una funció a cada opció del menú quan es produeix l'esdeveniment de clic. Aquesta funció utilitza una llibreria anomenada svelte-scrollto que el que permet és fer una animació de scroll fins a la ubicació indicada. Podem observar la seva definició a la *Figura 120*.

```
let scrollToFunction = (id: string, index: number) => {
  animateScroll.scrollTo({
    element: `#${id}`,

    onStart: () => {
      ms.activateItem(ms.scrollItems[index]);
      ms.dissableUpdate();
    },
    onDone: () => ms.enableUpdate()
  });
};
```

Figura 120. Funció scrollToFunction.

Per tal de comprovar el seu correcte funcionament, és necessari tenir un Chapter o Case Study de prova i navegar a través de la pàgina i comprovar que el menú s'actualitza acord a la secció de la pàgina on està ubicat. També s'ha de poder navegar a qualsevol secció de la pàgina fent clic a qualsevol element del menú i observar un scroll fins a la secció seleccionada. A la *Figura 121* es poden veure els resultats de la implementació.

Figura 121. Resultat del StickyMenu

11.1.11 TagsEditor

L'editor de Tags està format per un component anomenat TagsEditor. Per al seu funcionament és necessari passar per paràmetre totes les tags. Les quals inclouen el nombre de repeticions en totes les pàgines del CMS (veure *Figura 122*).

```
type Tag.WithPageCount = {
    id: number;
    value: string;
    type: TagType;
    _count: {
        pageTags: number;
    };
}
```

Figura 122. Tipus de dades Tag WithPageCount.

La pàgina està formada per una barra superior que permet afegir una nova tag, buscar entre totes les tags disponibles i indicar a l'usuari si s'està guardant el contingut. A continuació a la *Figura 123* podem veure la seva estructura HTML.

```
div class="tool-bar">
    <div class="tool-bar-item">
      <IconButton text="Add Tag" icon="add" on:click={onClickAdd}
disabled={loading || !!newTag}/>
    </div>
    <div class="tool-bar-item">
      <Searchbar
        bind:search={tagSearch}
        type="top"
        placeholder={"Search a Tag..."}
        submit={null}
        />
    </div>
    <div class="spinner p-responsive" class:loading>
      Saving...<Spinner/>
    </div>
  </div>
```

Figura 123. Barra superior del component TagsEditor.

En la barra superior hi ha 3 coses a destacar. Primer de tot és la funcionalitat d'afegir una nova tag. Podem veure que l'esdeveniment de clic està vinculat a una funció anomenada onClickAdd (veure *Figura 124*). Aquesta el que fa és configurar una nova

tag creant un objecte sense id i amb el valor de "New Tag". Posteriorment, el visualitzador de les tags automàticament ja mostra la tag. En el pròxim punt s'explicarà en més detall com es gestiona aquesta nova tag, ja que està tot gestionat en un component a part.

```
const onClickAdd = () => newTag = { id: undefined, value: 'New Tag' };
```

Figura 124. Funció onClickAdd.

Per tal de fer la cerca de les tags es fa servir el component SearchBar. En aquest component lliguem una variable anomenada tagSearch que correspon al text de la barra de cerca. Llavors, gràcies a les propietats reactives de Svelte generem una expressió regular i filtrem les tags tal com es veu a la *Figura 125*.

```
$: searchRegex = new RegExp(tagSearch, 'i');
$: filteredTags = tags.filter(tag => searchRegex.test(tag.value));
```

Figura 125. Filtratge de les tags.

L'últim element de la barra superior és un "spinner" que s'activa quan es processen els canvis a la base de dades. Svelte permet aplicar un estil CSS de forma dinàmica vinculant-lo a una variable. Aquesta variable està enllaçada al component TagEditor que s'encarrega de gestionar el seu estat en funció de si s'està processant una petició a la base de dades.

Finalment, queda la llista de les tags. Si mirem la *Figura 126* que correspon al codi HTML que les renderitza podem veure que hi ha dues seccions, una on es renderitzen totes les tags obtingudes de la base de dades (en cas que s'hagi filtrat per text només les tags filtrades) amb un botó que permet eliminar la tag. El botó té una funció vinculada a ell que s'encarrega de gestionar l'eliminació, el seu nom és onClickDelete. Aquesta funció comprova si la tag està present a una pàgina o més i en cas afirmatiu es mostra un modal de confirmació per procedir amb l'eliminació. Podem veure aquesta funció a la *Figura 126*. La segona secció correspon a les noves tags on hi ha un condicional que

comprova si s'ha clicat el botó de crear una nova tag comprovant el contingut de la variable newTag.

```
<ul class="tags-list">
    {#each filteredTags as tag (tag.id)}
      <li class="tag-row">
        <TagEditor bind:loading bind:tag />
        <span class="page-count">{tag._count?.pageTags} pages</span>
        <IconButton icon="delete" on:click={() => onClickDelete(tag)}
disabled={loading}/>
      </li>
    {/each}

    {#if newTag}
      <li class="tag-row">
        <TagEditor bind:loading bind:tag={newTag} />
      </li>
    {/if}
  </ul>
```

Figura 126. Llista de tags.

```
  const onClickDelete = async (tag: SubTypes.Tag.WithPageCount) => {
    if (tag._count.pageTags == 0) {
      await handleDelete(tag);
    } else {
      openModal(DeleteModal, {
        title: 'Delete Tag',
        message:
          'This tag is used on some pages. Are you sure you want to
delete it?' +
          `It will be removed from ${tag._count?.pageTags} pages.`,
        confirmText: tag.value,
        onYes: () => handleDelete(tag),
      });
    }
  };
```

Figura 127. Funció onClickDelete.

A l'inici de la memòria s'ha mencionat una eina Anomenada TextRazor. Es van fer diferents testos amb el contingut final de l'aplicació, però no es va implementar, ja que es va decidir com una característica futura en una pròxima fase de desenvolupament del projecte.

11.1.12 TagEditor

El component TagEditor correspon a cada una de les tags que hi ha en el component TagsEditor. Permet dos paràmetres:

- **tag: Pick<SubTypes.Tag, 'id' | 'value'>**: Correspon a una tag.
- **loading: boolean**: Indica si s'està fent un canvi a la base de dades.

L'editor està format pel components EditableText i IconButton. L'EditableText et permet modificar el text de la tag i tenir un control sobre si la tag està "focused". Les icones estan destinades a controlar els esdeveniments de guardar i de cancel·lació (veure *Figura 128*).

```
<div class="tag-editor" class:editing={focused}>
  <EditableText
    bind:value={editTag}
    editable={true}
    placeholder="Tag name"
    bind:focused
  />
  <IconButton icon="done" on:click={onClickSaveTag} disabled={loading ||
editTag.trim().length === 0}/>
  <IconButton icon="close" on:click={onClickCancelTag} disabled={loading}/>
</div>
```

Figura 128. Estructura de l'editor de tag.

Quan es clica a sobre el botó de guardar s'executa la funció onClickSaveTag. Aquesta s'encarrega de modificar l'estat de la variable loading i en funció de si és una tag nova, crea o actualitza la tag fent una crida a l'API (veure *Figura 129*).

```
  const onClickSaveTag = async () => {
```

```
    loading = true;

    try {
      const apiCall = tag.id == null
        ? createTag({ value: editTag })
        : updateTag(tag.id, { value: editTag });
      const _tag = await apiCall;
      tag.id = _tag.id; // id from createTag
      tag.value = _tag.value;
      toaster('Tag saved', { type: 'done' });
      focused = false;
    }
    catch(e) {
      toaster('Error saving tag', { type: 'error' });
    }

    loading = false;
  };
```

Figura 129. Funció onClickSaveTag.

Quan es clica a sobre el botó de cancel·lar o es fa clic fora de la tag, és a dir, la tag passa a ser "unfocus" s'executa la funció onClickCancelTag. Aquesta funció s'encarrega de restaurar el valor de la tag i en cas que fos una nova tag assignar el valor undefined (automàticament s'elimina en el component TagsEditor) (veure *Figura 130*).

```
const onClickCancelTag = () => {
    if (tag.id == null) tag = undefined;
    else editTag = tag.value;
    focused = false;
  };

  $: if (!focused) onClickCancelTag();
```

Figura 130. Funció onClickCancelTag.

11.1.13 PageOrdering

Per tal de fer funcionar el pageOrdering es necessita com a paràmetre la variable componentsOrder que correspon a una entrada de la taula KeyValue (veure *Figura 131*).

```
type KeyValue = {
  key: string;
  value: Prisma.JsonValue;
}
```

Figura 131. Tipus KeyValue.

A partir del paràmetre componentsOrder s'obté un array format pel nom dels components, la posició on està ubicat representa l'ordre en la landing page. Posteriorment, es crea una llista amb ajuda d'un paquet de nodejs (svelte-sortable-list). Cada element de la llista té assignada una icona en un lookup (componentsIconLookup) que identifica la que ha de fer servir (veure *Figura 132*). Quan un component és arrestat a una altra posició s'intercanvia la posició i actualitza les posicions a l'array. En el moment que es clica el botó de guardar ubicat a sota de la llista es fa una crida a l'API per actualitzar l'ordre de la pàgina.

```
<script lang="ts">
  import type { KeyValue, HomepageComponentName } from "$lib/types";
  import SortableList from 'svelte-sortable-list';
  import { updateKeyValue } from "$lib/api";
  import { getToaster } from '$lib/helpers/utils';
  import Spinner from "$lib/components/generic/Spinner.svelte";


  export let componentsOrder: KeyValue;

  let components = componentsOrder.value as HomepageComponentName[];

  let list = components.map((component, index) => ({
    id: index,
    value: component,
```

```
  }));

  let saving = false;

  const toaster = getToaster();

  const sortList = ev => {list = ev.detail; console.log("event: ",
ev.detail)};

  const saveList = async () => {
    saving = true;

    components = list.map(item => item.value);

    try{
      await updateKeyValue('landingPage', {value: components});
      toaster('Page ordering updated', {type: 'done'});
    } catch(e) {
      toaster('Error saving page ordering', {type: 'error'});
    }

    saving = false;
  };
  const componentsIconLookup = {
    'chapters': 'view_carousel',
    'casestudies': 'view_carousel',
    'lifecycle': 'incomplete_circle',
    'search': 'search',
    'madlib': 'arrow_drop_down_circle',
  }

</script>
<div class="cms-pageordering">

    <h3>LANDING PAGE</h3>

    <div class="grid-links">
      <SortableList
        {list}
        key="id"
```

```
        let:item
        on:sort={sortList}
      >
        <div class="item">
          <span
class="material-icons">{componentsIconLookup[item.value]}</span>
          {item.value}
        </div>
      </SortableList>
    </div>
    <button on:click={saveList} disabled={saving}>
      <div style="{saving ? "display: none;" : ""}">
        Save
      </div>
      <div class="spinner p-responsive" class:saving>
        Saving...<Spinner/>
      </div>
    </button>
</div>
```

Figura 132. Fitxer index.svelte corresponent a l'ordenació de pàgines.

11.1.14 Altres

Durant tot el procés d'implementació a part del desenvolupament dels principals components explicats en els punts anteriors també va ser necessari modificar o afegir alguna funcionalitat en algun component fet pels meus companys, tant de codificar noves funcionalitats com modificació dels estils (modificar estils, especificar breakpoints per al disseny responsiu, etc.), etc. He decidit no detallar aquests canvis, ja que els considero canvis molt petits i que no són rellevants a l'hora de fer la memòria. Tots els canvis mencionats poden ser comprovats a través de tots els commits i pull requests fets en el repositori del projecte.

A part de l'aplicació principal, també es va realitzar una eina que permet als usuaris fer una ressenya sobre la rellevància de cada frase del contingut. Tot i que l'aplicació ja està feta encara no s'ha enviat als usuaris indicats perquè es completi.

Tot el codi estarà disponible en el repositori de la fundació, ja que aquesta era una de les clàusules del contracte. De moment es manté en privat aquest repositori durant la fase de desenvolupament, així que a l'hora d'entregar la memòria facilitaré una còpia del repositori, posteriorment es podrà recuperar quan s'obri.

11.2 Endpoints i base de dades

En aquest punt explicaré en detall la feina que he realitzat referent a tots els endpoints i a la base de dades.

11.2.1 Base de dades

La base de dades s'ha implementat a través de Prisma conjuntament amb PostgreSQL. Aquest punt faré una petita introducció de quin és el seu disseny. No tota la base de dades ha estat dissenyada per mi. Els següents punts concreto la feina i els dissenys que he fet. Si mirem la *Figura 133*, podem veure el diagrama Entitat-Relació de la base de dades.

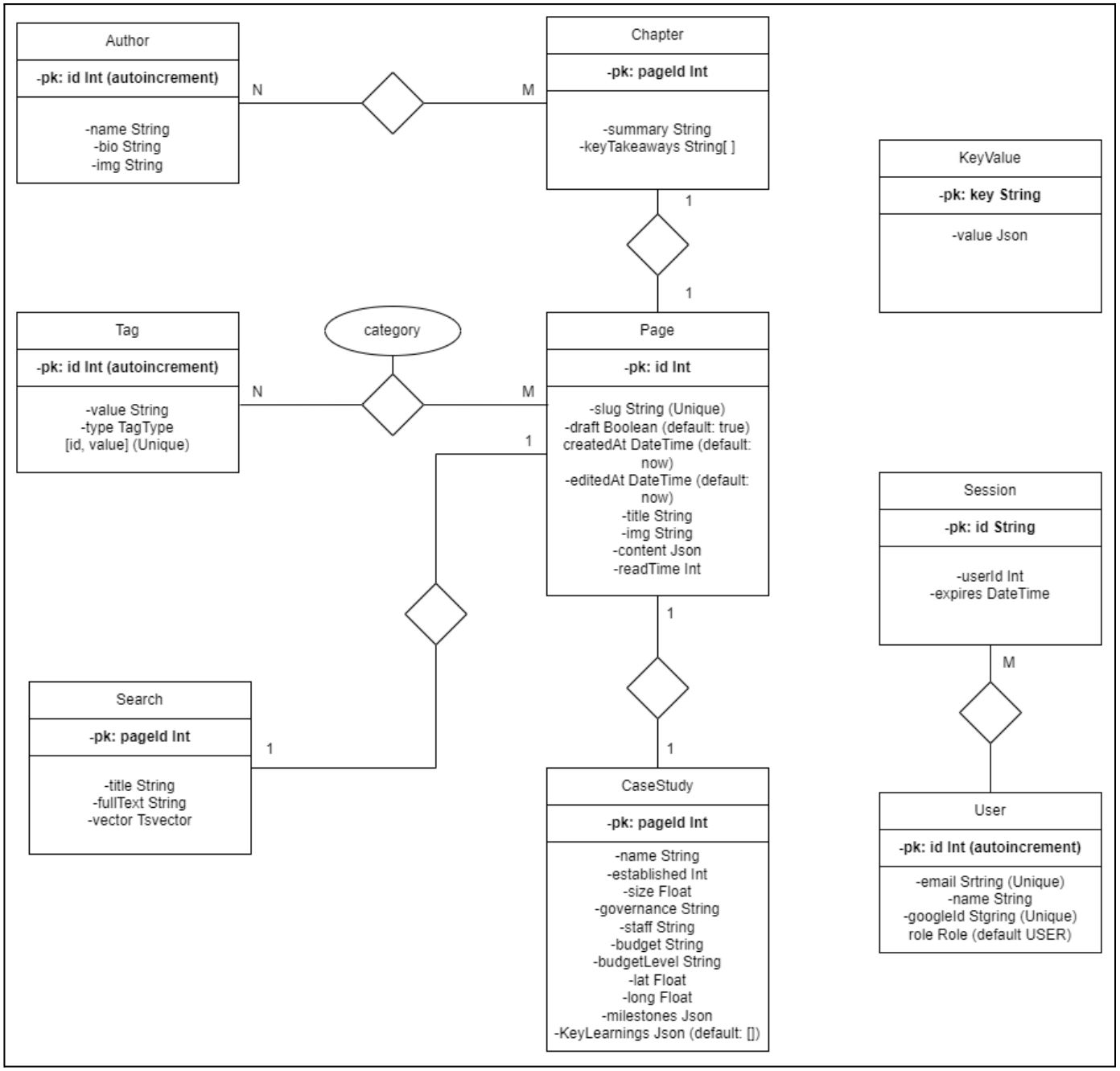


Figura 133. Diagrama Entitat-Relació de la base de dades.

Podem observar que la base de dades consta de la taula User que està relacionada amb Session. Aquí es guarden les dades referents als usuaris que tenen accés al CMS. També podem veure que una pàgina pot estar relacionada amb un Chapter o Case Study. A més, els Chapters tenen autors. La taula Search guarda la informació per a les cerques dins de la pàgina. Les taules Tags i KeyValue s'expliquen en detall en els pròxims punts.

11.2.2 Tags

Abans d'iniciar amb l'editor de tags crec que és preferible explicar com s'han guardat les tags a la base de dades, en punts anteriors s'ha explicat un component anomenat lifecycle el qual contenia una sèrie de tags i tenia un mode edició que permetia afegir o treure tags de cada pàgina. Aquestes tags en el mode edició són guardades a la base de dades. Així que entrem en detall de quines característiques tenen les tags.

Durant tota la memòria s'ha anat dient que les tags poden ser de tres tipus diferents (*stage*, *topic*, *user*). Aquest tipus és necessari guardar-lo a la base de dades. Però a més a més depenent de la relació de la tag en una pàgina en concret, pot tenir una visualització o un altre, també és necessària una categoria per identificar-ho. Aquesta categoria es guarda en la relació de Tag i Page. A continuació si observem la *Figura 134*, podem observar un esquema Entitat-Relació de la base de dades referent a les tags i a les pàgines.

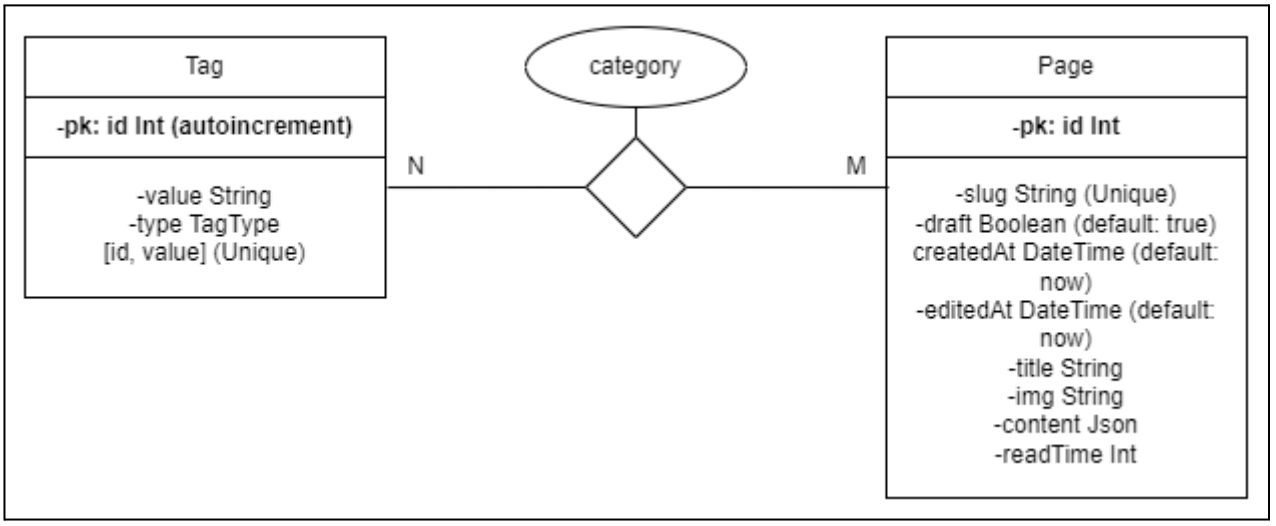


Figura 134. Entitat-Relació Tag Page.

Per tal de transformar això a Prisma és necessari definir 3 models diferents i dos tipus de dades, que farem servir els numeradors. Els models seran, un pel tipus de tag, un per la categoria de la tag i un segon per la relació de les tags amb les pàgines. És a dir, Tindrem el model Tag i Page, els quals corresponen a una tag i una pàgina i un últim un model anomenat TagsOnPages que representa la relació N:M i s'encarrega de guardar

la informació de la categoria de la tag i relacionar les tags de cada pàgina (veure *Figura 135*).

```
model Page {
  id         Int      @id @default(autoincrement())
  slug       String   @unique
  draft      Boolean  @default(true)
  createdAt  DateTime @default(now())
  editedAt   DateTime @default(now())
  title      String
  img        String
  content    Json
  readTime   Int
  tags       TagsOnPages[]
  caseStudy  CaseStudy?
  chapter    Chapter?
  search     Search?
}

model Tag {
  id         Int       @id @default(autoincrement())
  value      String
  type       TagType
  pageTags   TagsOnPages[]

  @@unique([value, type])
}

enum TagType {
  STAGE
  TOPIC
  USER
}

enum TagCategory {
  PRIMARY
  SECONDARY
}
```

```
model TagsOnPages {
  page        Page        @relation(fields: [pageId], references: [id])
  tag         Tag         @relation(fields: [tagId], references: [id])
  category    TagCategory @default(PRIMARY)

  pageId      Int
  tagId       Int

  @@id([tagId, pageId])
}
```

Figura 135. Codificació de les tags amb Prisma.

En el moment de guardar una pàgina en el CMS també es guarden les tags relacionades. Quan es vol guardar un element a la base de dades es fa una crida a la base de dades. En el component PageEditor hi ha una funció anomenada onClickSave que depenent de si és o no una pàgina nova es crea o s'actualitza la pàgina (veure *Figura 136)*.

```
async function onClickSave() {
    saving = true;

    if (isNewPage) {
      const {id} = await createPage(_page);
      window.location.href = `/cms/pages/${id}`;
    } else {
      await updatePage(pageId, _page);
    }
    saving = false;
    addToastMessage('Saved', {type: 'done'});
    savedPage = clone(_page);
  }
```

Figura 136. Funció onClickSave del component PageEditor.

Podem veure que crida a les funcions createPage i updatePage. Aquestes funcions estan ubicades a una llibreria escrita en TypeScript que recull totes les crides a l'API (api.ts). Es passa tots els elements de la pàgina com a paràmetre, un dels paràmetres

de l'objecte consisteix en un array de tags amb la categoria corresponent. Es pot observar en el tipus PageRequest en la *Figura 137*.

```
export type PageRequest = {
  title: string;
  draft: boolean;
  slug: string;
  content: ContentDocument;
  img: string;
  tags: {
    id: number,
    category: TagCategory
  }[];
  caseStudy?: SubTypes.CaseStudy.PageHead;
  chapter?: {
    summary: string;
    authors: number[];
    keyTakeaways: string[];
  }
}
```

Figura 137. Tipus PagesRequest.

Aquesta funció el que fa és un crida a l'API. En funció de si és una pàgina nova o no, es fa una petició PUT o PATCH (veure *Figura 138*).

```
async function _page(data: PageRequest, id?: number) {
  const newPage = id === undefined;
  const response = await fetch(
    newPage ? '/api/pages/create' : `/api/pages/${id}`,
    {
      method: newPage ? 'PUT' : 'PATCH',
      body: JSON.stringify(data)
    }
  );

  const page = await response.json() as Page;
  return page;
```

```
}

export async function createPage(data: PageRequest) {
  return _page(data);
}

export async function updatePage(id: number, data: PageRequest) {
  return _page(data, id);
}
```

Figura 138. Crida a l'API per actualizar o crear pàgina.

Posteriorment, aquesta petició és enviada, si observem l'URL correspon a "/api/pages/...". Això indica que serà processada pel nostre endpoint situat a "src/routes/api/pages/...". (veure *Figura 139*).

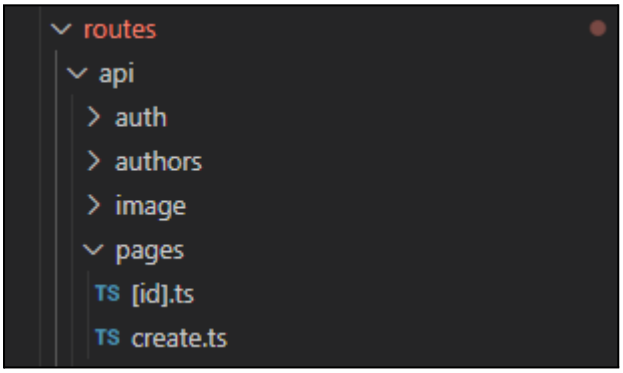


Figura 139. Ubicació endpoint en el projecte.

A dins de cada fitxer ([id].ts i create.ts) hi ha unes funcions destinades a cada un dels mètodes HTTP. Una pel PATCH i un altre pel PUT. A dins de cada una d'elles es fa una validació de les dades i les accions corresponents per actualitzar la base de dades. Més endavant amb l'editor de tags veurem en detall en què consisteixen aquestes validacions.

Ara hem vist com les tags estan guardades en el sistema. Una de les seccions del CMS és l'editor de tags. Les tags que estan permeses a ser editades únicament són les de tipus *topic*, a més, només poden ser editades per als usuaris amb privilegis de content manager o superiors. Per tant, és necessari que l'usuari estigui registrat/loggejat i que tingui suficients permisos. Per poder comprovar això fem servir els endpoints. Per a la

visualització d'una pàgina sempre s'executa el controlador del mètode HTTP GET (veure *Figura 140*).

```
export const get = authMiddleware(
  { role: 'CONTENT_MANAGER' },
  async () => {
    return {
      body: {
        tags: await prisma.tag.findMany({
          where: { type: "TOPIC" },
          ...countTags,
          orderBy: {
            value: 'asc'
          }
        })
      }
    };
  });
```

Figura 140. Controlador del mètode get per l'editor de tags.

Podem veure que utilitzem un middleware on indiquem que volem validar que l'usuari que està intentant accedir a les dades té un rol mínim de content manager. Si la validació de permisos és exitosa, s'executa la funció de callback, que en aquest cas, retorna la propietat body. Si observem els valors de body, podem veure que fa una consulta la base de dades a través de Prisma on demana totes les tags de tipus *topic* i a més un comptador de tags. El comptador de tags és fet servir per saber en quantes pàgines la tag està present.

Quan es volen guardar els canvis realitzats en l'editor el procediment a seguir és molt similar a l'explicat anteriorment. Envia les dades fent una petició a través de l'API. Un punt extra que també està aquí és que podem validar els permisos en les peticions de l'API de la mateixa forma a la qual es fa a la *Figura 140*. Aquestes dades són rebudes per l'endpoint de l'API i posteriorment processades.

Cal destacar els validadors de dades, abans de guardar qualsevol element a la base de dades ens assegurem que el seu format sigui adequat. A continuació explicaré el seu funcionament. Les validacions de les dades es fan amb schemes JSON. Quan la petició és processada per l'API s'executa una funció anomenada validate. Aquesta funció rep dos paràmetres. Les dades a validar i un identificador de quin tipus de dades és (veure *Figura 141*).

```
export const validate: Validate = (schema: string, data: unknown) => {
  const _validate = ajv.getSchema(schema);

  if (!_validate) throw new Error(`Schema not found: ${schema}`);
  const valid = _validate(data);

  if (!valid) {
    validate.errors = _validate.errors;
    log.error(_validate.errors);
    log.error(JSON.stringify(data, null, 2));
    throw new Error(_validate.errors.map(e => e.message).join(', '));
  }
};
```

Figura 141. Funció de validació de dades.

A partir de l'identificador podem veure que s'obté l'esquema JSON. Posteriorment, valida que les dades passades són correctes. El codi que ensenyaré a continuació consisteix en tota la validació que es realitza per cada tag. Si mirem la *Figura 142*, podem veure que es fan 3 validacions diferents, una per comprovar que no està buit, un altre per comprovar si la tag té un format invàlid i una última per validar si és un format vàlid. En cas que totes les comprovacions siguin correctes es procedeix a actualitzar la base de dades.

```
import { beforeAll, describe, expect, test } from "vitest";
import { validate } from "./validation";
import tagValid from "./testdata/tag.valid.json";
import tagInvalid from "./testdata/tag.invalid.json";
import { schemaExpectInvalid } from "./testutil";
```

```
describe("Tag", () => {

  beforeAll(() => {
    validate.errors = undefined;
  });

  test("empty", () => {
    expect(() => validate("tag", {})).toThrowError();
  });

  test("invalid", () => {
    tagInvalid.forEach(tag => schemaExpectInvalid("tag", tag));
  });

  test("valid", () => {
    tagValid.forEach(tag => {
      validate("tag", tag);
    });
  });

});
```

Figura 142. Fitxer tag.test.ts.

Per les comprovacions d'invalidesa i validesa fem servir esquemes JSON, els esquemes JSON ens permeten descriure el format de les dades de forma entenedora per les persones, a més, permet assegurar-nos de la qualitat de les dades que l'usuari puja a la base de dades. L'esquema JSON per a les tags es mostra a la *Figura 143*.

```
{
  "$id": "tag",

  "type": "object",
  "properties": {
    "value": {
      "type": "string",
      "minLength": 1,
      "maxLength": 30
```

```
    }
  },
  "additionalProperties": false,
  "minProperties": 1
}
```

Figura 143. Esquema JSON per les tags.

A continuació si observem les *Figures 144 i 145* podem veure alguns exemples de tags vàlides i invàlides, respectivament. Tots els elements dins d'aquests array seran validats un per un. Tal com es pot veure a la *Figura 142*.

```
[
  {
    "value": "New Tag"
  },
  {
    "foo": "additional properties should be valid & discarded",
    "value": "New tag"
  },
  {
    "value": "valid as max 30 chars - 123456"
  }
]
```

Figura 144. Esquema JSON per les tags vàlides.

```
[
  {
    "value": 1
  },
  {
    "value": ""
  },
  {
    "value": "should fail as max 30 chars 123"
```

```
  }
]
```

Figura 145. Esquema JSON per les tags invàlides.

En punts anteriors s'ha mencionat que fèiem servir una CDN anomenada Fastly. Tant els Chapters com els Case Studies disposen de tags, pel que és important tenir una bona gestió de la memòria cau. Per al que hem d'assegurar que cada vegada que una tag és modificada invalidem les pàgines que estaven guardades a la memòria cau i obligar a l'usuari a tornar-les a obtenir.

En cas que la tag hi hagi set actualitzada a la base de dades s'emet un esdeveniment per tal que es pugui executar una funció que indiqui a Fastly les surrogate keys que han estat modificades. Aquest esdeveniment pot ser per la creació, modificació o eliminació d'una tag. A continuació si observem la *Figura 146*, podem veure els tipus d'aquests esdeveniments.

```
export type TagDeletedEvent = {
  type: 'tag-deleted';
  details: { id: number };
}

export type TagUpdatedEvent = {
  type: 'tag-updated';
  details: { id: number };
}

export type TagCreatedEvent = {
  type: 'tag-created';
  details: { id: number };
}
```

Figura 146. Esdeveniments de les tags.

Quan s'emet l'esdeveniment hi ha una funció anomenada processEvent que el captura i procedeix a actualitzar l'estat de la surrogate Key (veure *Figura 147*).

```
async function processEvent(event: Event): Promise<EventOutput> {
  switch (event.type) {
    case 'page-deleted':
    case 'page-updated':
      return {
        ...await purgePage(event.details.id),
        event
      };
    case 'tag-deleted':
    case 'tag-updated':
      return {
        ...await purgeTag(event.details.id),
        event
      };
    case 'page-created':
      return { ...await purgeSurrogate('pages'), event };
    case 'tag-created':
      return { ...await purgeSurrogate('tags'), event };
    case 'key-value-updated':
      return { ...await purgeKeyValue(event.details.id), event };
    default:
      console.error('unknown event type', JSON.stringify(event));
      return {
        status: 'error',
        error: 'unknown event type',
        event,
      };
  }
}
```

Figura 147. Funció processEvent.

Podem veure que hi ha 2 funcions diferents, una anomenada purgeTag que s'encarrega d'eliminar la surrogate key de Fastly i una segona anomenada purgeKeyValue que s'encarrega de canviar l'estat de la surrogate key per tal que se sàpiga que ha estat modificada. Per comunicar-se amb Fastly es fa a través d'una crida a la seva API. Per exemple si observem la *Figura 148*, podem veure com es fa la crida a l'API de Fastly.

```
async function purgeSurrogate(key: string):
Promise<DistributiveOmit<EventOutput, 'event'>> {
  const response = await
fetch(`https://api.fastly.com/service/${process.env.FASTLY_SERVICE_ID}/pu
rge/${key}`, {
    method: "POST",
    headers: {
      "Fastly-Key": process.env.FASTLY_API_KEY,
      "fastly-soft-purge": "1",
      "Accept": "application/json",
    }
  });

  if (response.ok) {
    return { status: 'ok' };
  } else {
    return { status: 'error', error: await response.json() };
  }
}
```

Figura 148. Funció purgeSurrogate.

Amb això ja està acabada la gestió de la memòria cau per a les tags. Falta explicar com cada pàgina gestiona les surrogate keys. SvelteKit ens proporciona un fitxer opcional anomenat hooks.ts. En aquest fitxer es pot definir una funció que es diu handle. La funció handle s'executa prèviament a tots els endpoints. Pel que cada vegada que es carrega una pàgina aquesta funció s'executa. Això ens permet inicialitzar la variable events.locals.cacheKeys i recuperar totes les surrogate keys rebudes en la capçalera (veure *Figura 149*).

```
export const handle: Handle = async ({ event, resolve }) => {
  event.locals.cacheKeys = new Set();
  const response = await resolve(event);
  if (response.ok) {
    const cacheHeaders = getCacheHeaders(event);
    for (const [key, value] of Object.entries(cacheHeaders)) {
      if (value != null) response.headers.set(key, value);
    }
```

```
  }
  return response;
};

function getCacheHeaders(event: RequestEvent): CacheHeaders {
  if (event.routeId == null) return {};
  return {
    'Cache-Control': getCacheControl(event),
    'Surrogate-Key': event.locals.cacheKeys.size > 0 ?
Array.from(event.locals.cacheKeys).join(' ') : undefined,
    'Surrogate-Control': 'stale-while-revalidate=30,
stale-if-error=3600',
  };
}
```

Figura 149. Funcions handle i getCacheHeaders del fitxer hooks.ts.

Posteriorment, a dins de cada pàgina on s'utilitzen hi ha influència de les tags, cal assignar el valor de les surrogate Keys actuals de la pàgina a la variable events.locals.cacheKeys. Per exemple tal com es fa al fitxer [slug].ts (veure *Figura 150*).

```
import type { RequestHandler } from "@sveltejs/kit";
import { getPageComponentProps } from "$lib/prisma/wrappers";

export const get: RequestHandler<{slug: string}> = async ({ locals,
params: { slug } }) => {
  const pageProps = await getPageComponentProps(slug, false);

  if ('error' in pageProps.body) return pageProps;

  const pageId = pageProps.body.page.id;
  const pageTags = pageProps.body.page.tags;

  locals.cacheKeys.add(`page-${pageId}`);
  pageTags.forEach(tag => locals.cacheKeys.add(`tag-${tag.tag.id}`));

  return pageProps;
};
```

Figura 150. Fitxer [slug].ts.

A més les pàgines quan es creen es genera un vector de cerca per a futures cerques, quan una tag és modificada aquest vector ja no és vàlid pel que s'ha d'actualitzar. Això es fa dins de la funció updateTag del fitxer wrappers.ts (veure *Figura 151*).

```
const createPageSearchIndex = prisma.$queryRaw`SELECT CAST
(create_page_search_index("pageId") AS TEXT) FROM "TagsOnPages" WHERE
"tagId" = ${id};`;
```

Figura 151. Actualització del vector de cerca quan una tag és actualitzada.

Si mirem la *Figura 152*, podem veure el resultat de l'editor de tags. Per comprovar el seu funcionament s'han afegit, modificat i eliminat tags i comprovat posteriorment que s'han actualitzat a la base de dades i a totes les pàgines que estaven relacionades.

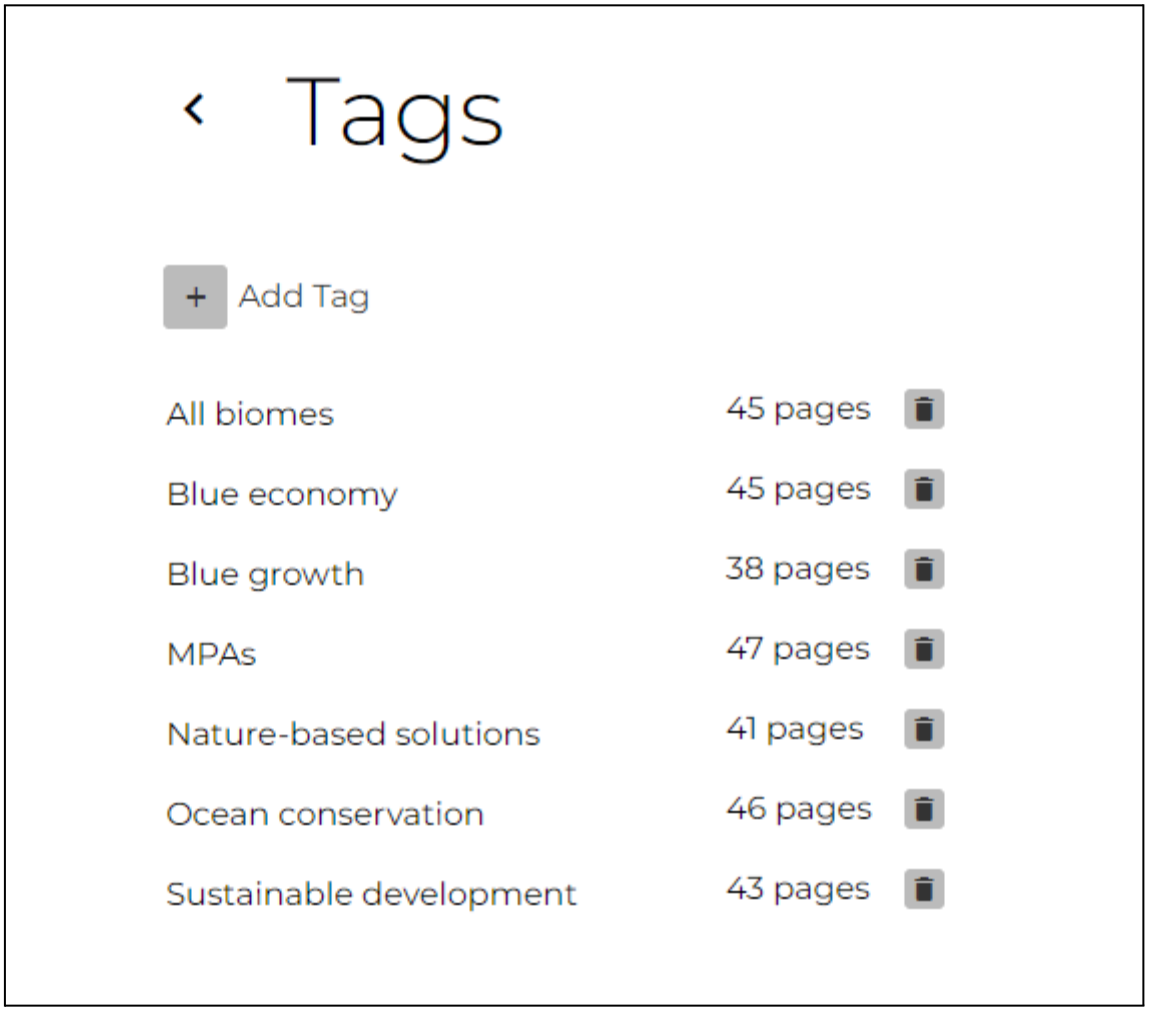


Figura 152. Editor de tags.

11.2.3 Page Ordering

El procés utilitzat per la gestió en el back-end de l'ordenació de pàgines el mateix que l'utilitzat per les tags. Així que en aquest punt faré menció de com es guarden a la base de dades i dels principals fitxer de validació de les dades.

Per guardar l'ordenació de les pàgines es va decidir utilitzar una taula Key-Value. En ella es guarden parelles de clau-valor. En el cas de l'ordenació de les pàgines es guarda la clau identificativa, en aquest cas "homepage-components". El valor assignat és un array on cada element identifica un component i la seva posició representa l'ordre dels components (veure *Figura 153*).

	key [PK] text	value jsonb
1	homepage-componen...	["lifecycle", "chapters", "search", "madlib", "casestudi...

Figura 153. Entrada a la taula KeyValue per la landing page.

Per la creació d'aquesta taula la seva codificació amb Prisma és la següent (veure *Figura 154*).

```
model KeyValue {
  key   String @id
  value Json
}
```

Figura 154. Codificació de la taula KeyValue.

Els permisos per poder accedir a l'ordenació de pàgines són d'administrador, pel que a l'endpoint del CMS, és a dir, el fitxer "src/routes/cms/pageordering/index.ts" dins del mètode de control GETs'executarà un middleware per autentificar l'accés. Posteriorment, s'executarà la funció de callback per obtenir les dades (veure *Figura 155*).

```
import { authMiddleware } from "$lib/auth";
import { prisma } from "$lib/prisma";
import { pageComponentsOrder } from "$lib/prisma/queries";

export const get = authMiddleware(
  { role: 'ADMIN' },
  async () => {

    const componentsOrder = await prisma.keyValue.findUnique({
      ...pageComponentsOrder,
      where: { key: 'landingPage' }
    });

    return { body: { componentsOrder } };
  }
);
```

Figura 155. Fitxer Index.ts de Page Ordering.

Seguidament, quan es guardin els canvis el procés a seguir és el mateix que l'explicat en el punt anterior. Primer de tot es farà una crida a l'api amb un nou array d'ordenació i es validaran les dades. L'esquema JSON es pot veure a continuació (veure *Figura 156*).

```
{
  "$id": "keyValue",
  "type": "object",

  "properties": {
    "value": { "type": ["array", "object"] },
    "key": { "type": "string" }
  }
}
```

Figura 156. Esquema JSON per les KeyValue.

A continuació, un exemple d'un element vàlid per la taula KeyValue (veure *Figura 157*). No s'ha introduït cap element més, ja que de moment a la taula KeyValue només es guarda aquest tipus d'objectes.

```
[
  {
    "value": ["string", "string", "string"]
  }
]
```

Figura 157. Exemple vàlid per l'entrada a la taula KeyValue.

Després, si les comprovacions han estat totes correctes, s'emet l'esdeveniment per actualitzar la surrogate key relacionada amb l'ordenació de pàgina. Aquestes surrogate Key només està vinculada a la landing page tal com es pot veure a la *Figura 158*.

```
import type { RequestHandler } from "@sveltejs/kit";
import { error404 } from "$lib/errors";
import { prisma } from "$lib/prisma";
import { pageForContentCard, tag, keyValue as KeyValueSelect } from
"$lib/prisma/queries";
import { groupBy } from "$lib/helpers/utils";

export const get: RequestHandler<{ id: string }> = async ({locals}) => {

  const pages = await prisma.page.findMany({
    where: { draft: false },
    ...pageForContentCard
  });

  const tags = await prisma.tag.findMany({
    where: { type: "TOPIC" },
    ...tag
  });


  const keyValue = await prisma.keyValue.findUnique({
    ...KeyValueSelect,
    where: { key: "landingPage" }
  });
```

```
  locals.cacheKeys.add(`pages`);
  locals.cacheKeys.add(`tags`);
  locals.cacheKeys.add(`key-value-${keyValue.key}`);

  const groups = groupBy(pages, p => p.chapter ? 'chapters' :
'caseStudies');

  return pages
    ? { body: { ...groups, tags, componentsOrder:  keyValue.value} }
    : error404('Page not found');
};
```

Figura 158. Fitxer index.ts corresponent a la landing page.

Finalment, cal explicar com s'aplica l'ordenació dels elements. Per fer l'ordenació faig servir una propietat CSS anomenada flex. Aquesta permet assignar una posició als elements html dins del pare amb la propietat CSS "display: flex". Pel que hem d'obtenir l'índex de posició de cada component en l'array guardat a la base de dades. (veure *Figures 159 i 160*).

```
  <div class="ordered-components">
    <div style="order: {getComponentOrder('lifecycle')}">
      <MpaManagementLifecycle/>
    </div>

    <div style="order: {getComponentOrder('chapters')}">
      <LandingCarousel pages={chapters} title="Get the <b>answers</b> to
all your questions" />
    </div>

    <div style="order: {getComponentOrder('search')}"
class="ordered-component inline-searchbar">
      <Searchbar type={'inline'}/>
      <TagContainer tags={tagsForContainer}/>
    </div>

    <div style="order: {getComponentOrder('madlib')}">
```

```
      <LandingMadlib />
    </div>

    <div style="order: {getComponentOrder('casestudies')}">
      <LandingCarousel pages={caseStudies} title="Explore what <b>others
have done</b>" />
    </div>
  </div>
```

Figura 159. Assignació de la posició als components de la landing page.

```
  const getComponentOrder = (name: HomepageComponentName): number =>
componentsOrder.indexOf(name);
```

Figura 160. Funció getComponentOrder.

Si mirem la *Figura 161*, podem veure el resultat d'implementació del component PageOrdering. Per comprovar el seu funcionament, s'ha modificat l'ordre dels components i posteriorment revistat a la base de dades, també s'ha anat a la landing page i comprovat que l'ordre ha estat modificat i correspon a l'assignat en el component.

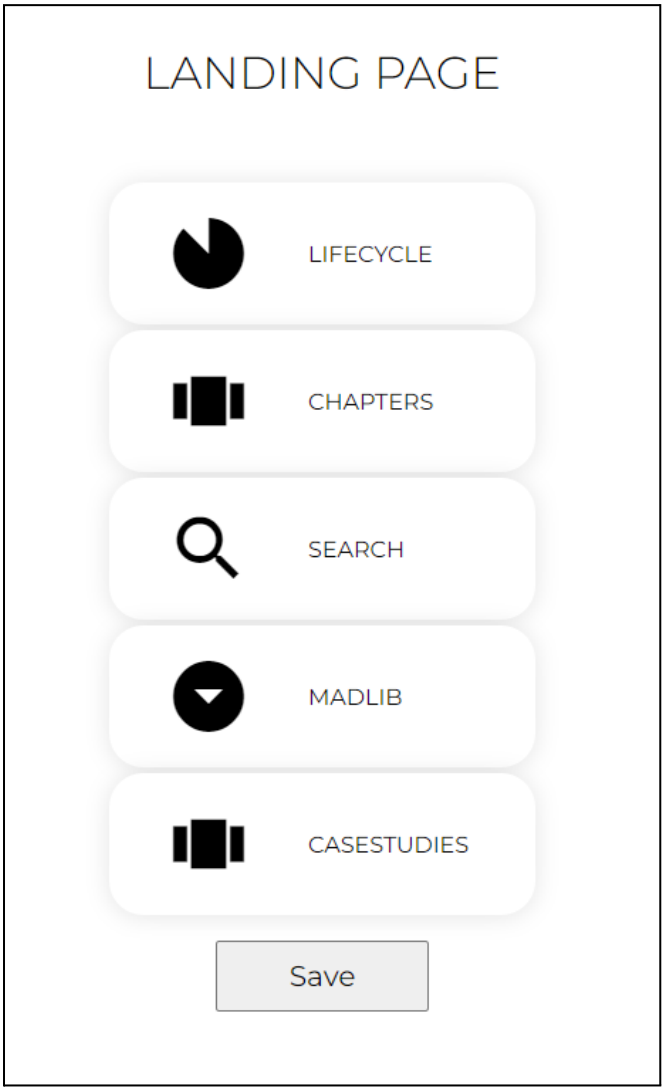


Figura 161. Component PageOrdering.

11.2.4 Recommended Pages

Les pàgines recomanades es generen en el moment que es recupera a la base de dades un Chapter o Case Study per ser visualitzat. Quan hem recuperat el contingut i amb ell totes les tags associades es demana a la base de dades un conjunt de pàgines indiferentment de si són Chapters o Case Study que tinguin en comú unes tags. A la funció getRecommendedPages es passa per paràmetre tot el conjunt de tags associades al Chapter o Case Study del que es vulguin obtenir les pàgines recomanades. Un cop s'han recuperat tots els candidats es demana que es contin el nombre de tags que té cada un i s'ordenen en ordre ascendent (veure *Figura 162*).

```
export async function getRecommendedPages(page: Pick<SubTypes.Page.Full,
'id' | 'tags'>) {
  const candidates = await prisma.page.findMany({
    where: {
      draft: false,
      tags: { some: { OR: page.tags.map(t => ({ tagId: t.tag.id })) } },
      NOT: { id: page.id }
    },
    orderBy: { tags: { _count: 'asc' } },
    ...pageForContentCard
  });

  return candidates;
};
```

Figura 162. Funció getRecommendedPages.

12. Conclusions

En l'àmbit personal estic molt satisfet amb tots els coneixements que he assolit. Gràcies a tot l'equip de treball hem aconseguit fer una primera versió molt bona.

A l'hora de realitzar el disseny gràfic, la meva aportació no hi ha set tant com la que m'esperava, però he pogut seguir de molt a prop el seu desenvolupament, també he tingut l'oportunitat de fer algunes aportacions com ara el disseny de l'editor de tags i de l'ordenació de pàgines.

Pel que fa al desenvolupament usant les tecnologies Svelte, HTML, CSS, Javascript i TypeScript, aquest ha estat sense dubte l'assoliment al qual puc dir que estic més orgullós. Gran part de la meva feina ha estat basada en aquest punt i sento que he adquirit nous coneixements i tècniques. Un punt que també està relacionat amb els dos anteriors és que el lloc web sigui accessible. Pel que fa a disseny; s'ha tingut sempre en compte; s'han utilitzat colors accessibles fent servir esquemes de colors amb contrasts per diversos tipus de discapacitats. Pel que fa a nivell de codi sempre s'intenta seguir amb els estàndards recomanats, però no s'ha tingut el temps suficient per poder validar exhaustivament l'accessibilitat del codi.

GitHub ha estat molt important en aquest projecte, gràcies a ell hem pogut tenir un control de versions molt bo, cosa que ha permès que jo pogués treballar paral·lelament amb tot l'equip.

Si parlem del backend, he fet aportacions per les tags i l'ordenació de pàgines, però m'hauria agradat fer més. He fet alguns dels dissenys de la base de dades, però no tots. Gràcies a això he pogut fer servir Prisma en profunditat i entendre com funciona, ja que totes les consultes que es feien a la base de dades eren a través d'aquesta eina, malauradament, no he tocat gaire PostgreSQL més enllà de la seva configuració inicial i posada a punt per al projecte, pel fet que en fer servir Prisma no he tingut la necessitat de fer-ho.

La gestió de la memòria cau l'he hagut de fer en les tags i en el page ordering. Em vaig interessar molt pel tema i vaig dedicar uns dies a entendre en profunditat al seu funcionament. Estic satisfet amb el que he aconseguit.

El sistema d'autenticació d'usuari va ser implementat pel líder tècnic del projecte. La meva feina va ser entendre com funciona i utilitzar-lo per gestionar l'accés a certes pàgines del CMS i de l'API.

Igual que el tema de la gestió de la memòria cau, els vectors de cerques també els vaig haver d'usar per a l'editor de tags. Vaig haver d'entendre com funciona, fer servir i modificar algunes de les funcions claus pel seu ús.

Finalment, l'últim dels propòsits era la creació d'una API. Aquest punt va relacionat amb Prisma, ja que la majoria de peticions que es feien a l'API, per obtenir les dades es feien a través de Prisma. Aquest ha estat un dels punts on he dedicat més temps i he adquirit més coneixements.

Estic molt satisfet amb la feina feta. Ha quedat un producte del qual estic content dels resultats. Encara queden implementacions i detalls a polir, com per exemple els temps

inicials de càrrega que són elevats o el disseny responsiu que no està del tot complet. Que seran fets en una següent fase de desenvolupament. Un dels meus objectius era saber les bases per crear una aplicació des de zero i adquirir els coneixements necessaris per poder-ho replicar, sens dubte penso els he aconseguit. La feina en equip ha estat molt bona i l'equip de treball sempre m'ha donat suport.

13. Treball futur

Hi ha hagut funcionalitats del projecte que s'han deixat per a futures versions. En aquest punt faré un llistat de tot el treball futur que es durà a terme.

- Implementar un diagrama personalitzable en el CMS.
- Implementar pàgines estàtiques d'errors i informació, són pàgines informatives que no estan implementades. Una destinada a mostrar informació a l'usuari respecte tot l'equip de treball, informació de contacte, etc. I unes altres per mostrar possibles errors, p.e l'error 404 pàgina no trobada.
- Implementar un nou mòdul anomenat "Do I need a MPA" que s'utilitzarà per guiar els usuaris per saber si necessiten una MPA pel problema a solucionar.
- Revisió dels estils responsius.
- Sistema de còpies de seguretat.
- Sistema de monitoratge del lloc web.
- Sistema de compartició social.
- Tests de rendiment.
- Sistema d'analítiques.
- Millorar el sistema de recomanació de pàgines.
- Generació de tags automàticament amb ajuda d'IA.

14. Bibliografia

[1] - *Promoting Effective Marine Protected Areas*. (s. d.). UNEP - UN Environment Programme. Recuperat el 7 de juliol de 2022, de https://www.unep.org/explore-topics/oceans-seas/what-we-do/promoting-effective-marine-protected-areas.

[2] - Fastly. (s. d.). *Precios de edge cloud*. Recuperat el 26 de juliol de 2022, de https://www.fastly.com/es/pricing.

[3] - *TextRazor - The Natural Language Processing API*. (s. d.). TextRazor. Recuperat el 26 de juliol de 2022, de https://www.textrazor.com/plans.

[4] - *Pricing · Plans for every developer*. (s. d.). GitHub. Recuperat el 26 de juliol de 2022, de https://github.com/pricing.

[5] - *Slack*. (s. d.). Slack. Recuperat el 26 de juliol de 2022, de https://app.slack.com/plans/T01BNRNV6KU.

[6] - *Notion*. (s. d.). Notion. Recuperat el 26 de juliol de 2022, de https://www.notion.so/pricing.

[7] - *Qué es SCRUM*. (s. d.). Proyectos Ágiles. Recuperat el 26 de juliol de 2022, de https://proyectosagiles.org/que-es-scrum/.

[8] - *Transparency*. (s. d.). Visualization for Transparency Foundation. Recuperat el 10 de juny de 2022, de https://www.fundaciovit.org/transparency.

[9] - *What is serverless?* (s. d.). Red Hat. Recuperat el 31 de juliol de 2022, de https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless.

[10] - Roberts, M. (s. d.). *Serverless Architectures*. martinfowler.com. Recuperat el 2 d'agost de 2022, de https://martinfowler.com/articles/serverless.html.

[11] - *¿Qué es FaaS?* (s. d.). Red Hat. Recuperat el 2 d'agost de 2022, de https://www.redhat.com/es/topics/cloud-native-apps/what-is-faas.

[12] - colaboradores de Wikipedia. (2020, 3 abril). *Backend as a service*. Wikipedia, la enciclopedia libre. Recuperat el 2 d'agost de 2022, de https://es.wikipedia.org/wiki/Backend_as_a_service.

[13] - *SvelteKit docs*. (s. d.). SvelteKit. Recuperat el 3 d'agost de 2022, de https://kit.svelte.dev/docs/introduction#what-is-sveltekit.

[14] - *SvelteKit docs*. (s. d.). SvelteKit. Recuperat el 3 d'agost de 2022, de https://kit.svelte.dev/docs/routing.

[15] - Malakannawar, A. (s. d.). *What is a vector search engine?* DEV Community. Recuperat el 8 d'agost de 2022, de https://dev.to/asmit2952/what-are-vector-search-engines-3lp1.

[16] - Fastly. (s. d.). *What is caching?*. Recuperat el 8 d'agost de 2022, de https://docs.fastly.com/en/fundamentals/what-is-caching.

[17] - Fastly. (s. d.). *What is a CDN?*. Recuperat el 8 d'agost de 2022, de https://docs.fastly.com/en/fundamentals/what-is-cdn.

[18] - GeeksforGeeks. (s. d.). *Explain DOM Diffing*. Recuperat el 12 d'agost de 2022, de https://www.geeksforgeeks.org/explain-dom-diffing/.

[19] - *Svelte*. (s. d.). Svelte. Recuperat el 7 de juliol de 2022, de https://svelte.dev/.

[20] - *Introduction | Vue.js*. (s. d.). Vue. Recuperat el 13 d'agost de 2022, de https://vuejs.org/guide/introduction.html.

[21] - *Getting Started –*. (s. d.). React. Recuperat el 13 d'agost de 2022, de https://reactjs.org/docs/getting-started.html.

[22] - *SvelteKit*. (s. d.). SvelteKit. Recuperat el 13 d'agost de 2022, de https://kit.svelte.dev/.

[23] - E. (s. d.). *GitHub - EmilTholin/svelte-routing: A declarative Svelte routing library with SSR support*. GitHub. Recuperat el 13 d'agost de 2022, de https://github.com/EmilTholin/svelte-routing#readme.

[24] - *Svelte Router*. (s. d.). Jikkai. Recuperat el 13 d'agost de 2022, de https://jikkai.github.io/svelte-router/#/guide/basic-usage.

[25] - Wikipedia contributors. (2022, 24 junio). *TypeScript*. Wikipedia. Recuperat el 7 de juliol de 2022, de https://en.wikipedia.org/wiki/TypeScript.

[26] - *JavaScript With Syntax For Types.* (s. d.). Typescriptlang. Recuperat el 13 d'agost de 2022, de https://www.typescriptlang.org/.

[27] - *Documentation - TypeScript for the New Programmer*. (s. d.). TypeScript. Recuperat el 13 d'agost de 2022, de https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html.

[28] - *GitHub Copilot · Your AI pair programmer*. (s. d.). GitHub. Recuperat el 13 d'agost de 2022, de https://github.com/features/copilot.

[29] - C. (s. d.). *GitHub - CodedotAI/gpt-code-clippy*. GitHub. Recuperat el 13 d'agost de 2022, de https://github.com/CodedotAI/gpt-code-clippy.

[30] - *AI Assistant for software developers | Tabnine*. (s. d.). TabNine. Recuperat el 13 d'agost de 2022, de https://www.tabnine.com/get-started?utm_source=google.com&utm_medium=cpc&utm_campaign=208800395&utm_content=108352236131&utm_term=tabnine&gclid=Cj0KCQjwl92XBhC7ARIsAHLl9am8kiU1xlJP-QLjyNiXmzVXvvJdjsRL4I1D65a1NpcA9xqN0CXa6YEaAiESEALw_wcB.

[31] - Kite. (2021, 8 abril). *Kite - Free AI Coding Assistant and Code Auto-Complete Plugin*. Code Faster with Kite. Recuperat el 13 d'agost de 2022, de https://www.kite.com/.

[32] - *PostgreSQL: About*. (s. d.). The PostgreSQL Global Development Group. Recuperat el 4 d'agost de 2022, de https://www.postgresql.org/about/.

[33] - *8.11. Text Search Types*. (2022, 11 agosto). PostgreSQL Documentation. Recuperat el 14 d'agost de 2022, de https://www.postgresql.org/docs/current/datatype-textsearch.html.

[34] - *MySQL*. (s. d.). MySQL. Recuperat el 14 d'agost de 2022, de https://www.mysql.com/.

[35] - *SQLite Home Page*. (s. d.). SQLite. Recuperat el 14 d'agost de 2022, de https://www.sqlite.org/index.html.

[37] - *Red global de contenidos CDN*. (s. d.). Cloudflare. Recuperat el 14 d'agost de 2022, de https://www.cloudflare.com/es-es/cdn/.

[38] - *AWS | Servicio de entrega de contenidos y transferencia de datos*. (s. d.). Amazon Web Services, Inc. Recuperat el 14 d'agost de 2022, de https://aws.amazon.com/es/cloudfront/.

[39] - *Azure Content Delivery Network*. (s. d.). Microsoft Azure. Recuperat el 14 d'agost de 2022, de https://azure.microsoft.com/es-es/services/cdn/.

[40] - *Visual Studio Code - Code Editing. Redefined*. (s. d.). Visual Studio Code. Recuperat el 31 d'agost de 2022, de https://code.visualstudio.com/.

[41] - *Expressive, dynamic, robust CSS — expressive, robust, feature-rich CSS preprocessor*. (s. d.). Stylus. Recuperat el 31 d'agost de 2022, de https://stylus-lang.com/.

[42] - *Svelte Splide.* (s. d.). Splide - The lightweight, flexible and accessible slider/carousel. Recuperat el 14 d'agost de 2022 de https://splidejs.com/integration/svelte-splide/.

[43] - J. (s. f.-c). *GitHub - janosh/svelte-multiselect: Keyboard-friendly, accessible and highly customizable multi-select component*. GitHub. Recuperat el 31 d'agost de 2022, de https://github.com/janosh/svelte-multiselect.

[44] - S. (s. f.-d). *GitHub - STRML/textFit: A jQuery-free component that quickly fits single and multi-line text to the width (and optionally height) of its container.* GitHub. Recuperat el 31 d'agost de 2022, de https://github.com/STRML/textFit.

[45] - L. (s. f.-d). *GitHub - langbamit/svelte-scrollto: Animating vertical and horizontal scrolling. Inspired by rigor789/vue-scrollto and uses some of its code and functionality!!* GitHub. Recuperat el 31 d'agost de 2022, de https://github.com/langbamit/svelte-scrollto.

[46] - *MenuSpy*. (s. d.). MenuSpy. Recuperat el 31 d'agost de 2022, de https://leocs.me/menuspy/.

15. Manual d’usuari i/o instal·lació

15.1 Manual d’instal·lació

Per tal de posar l'aplicació en marxa primer de tot es necessita tenir instal·lat PostgreSQL en l'equip. Es pot aconseguir l'última versió a través de la seva pàgina web de forma gratuïta. En el moment de la seva instal·lació es demana per l'usuari i contrasenya. Per defecte l'usuari estàndard s'anomena postgres. Cal recordar aquesta contrasenya, ja que serà necessari per connectar-nos a la base de dades.

Un cop instal·lat PostgreSQL hem de crear una base de dades nova. Això ho podem fer amb una aplicació que s'instal·la automàticament quan instal·lem PostgreSQL. Si obrim PgAdmin demana una contrasenya, aquesta és la que hem definit durant la instal·lació. *Figura 163*.

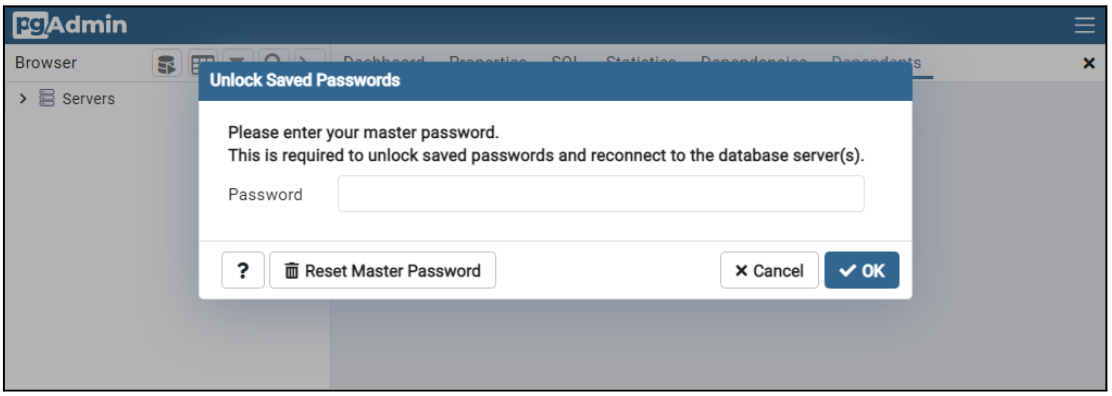


Figura 163. Inicialització PgAdmin.

Un cop hem entrat correctament haurem de seleccionar el servidor on volem crear una base de dades. Per defecte el nom és PostgreSQL, novament haurem d'introduir la contrasenya (veure *Figura 164*).

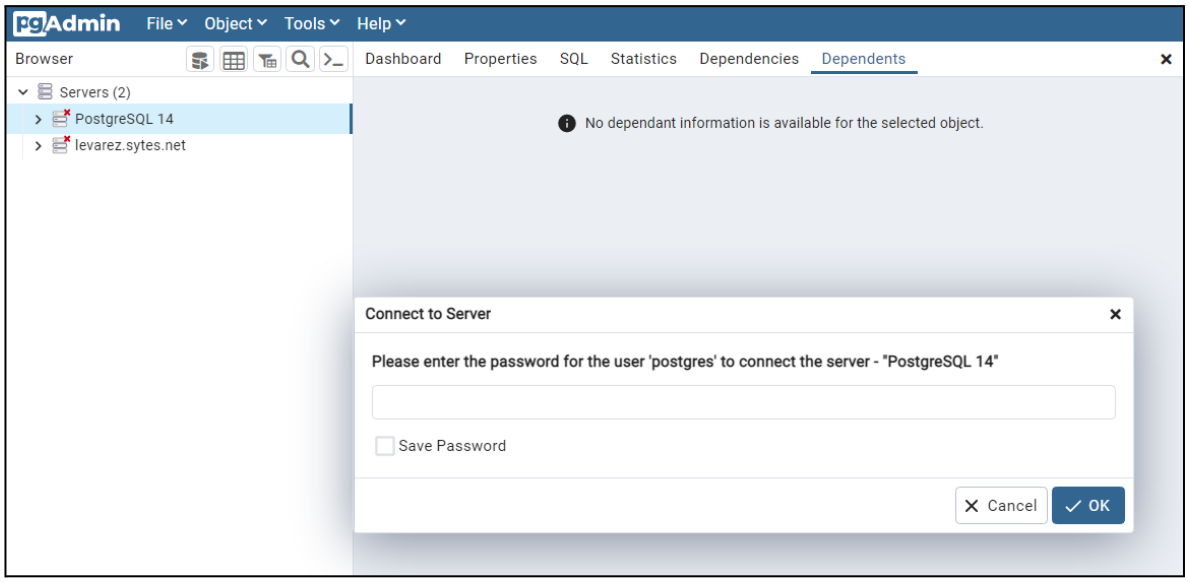


Figura 164. Selecció servidor.

Seguidament, hem de crear una nova base de dades, si seleccionem amb clic dret on posa DataBases Create→Database (*Figura 165*) s'obre una finestra on podem definir el nom de la base de dades, usuari propietari i altres propietats. Només cal definir el nom de l'usuari propietari (per defecte és el de postgres) i el nom de la base de dades que nosaltres desitgem, en aquest cas mpa. *Figura 166*.

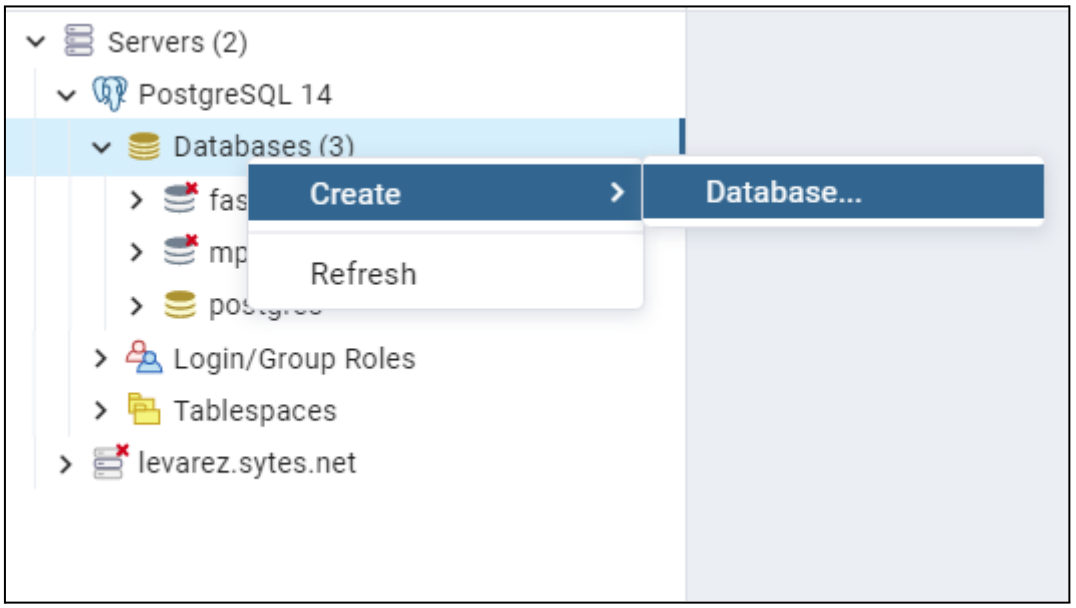


Figura 165. Creació de la base de dades

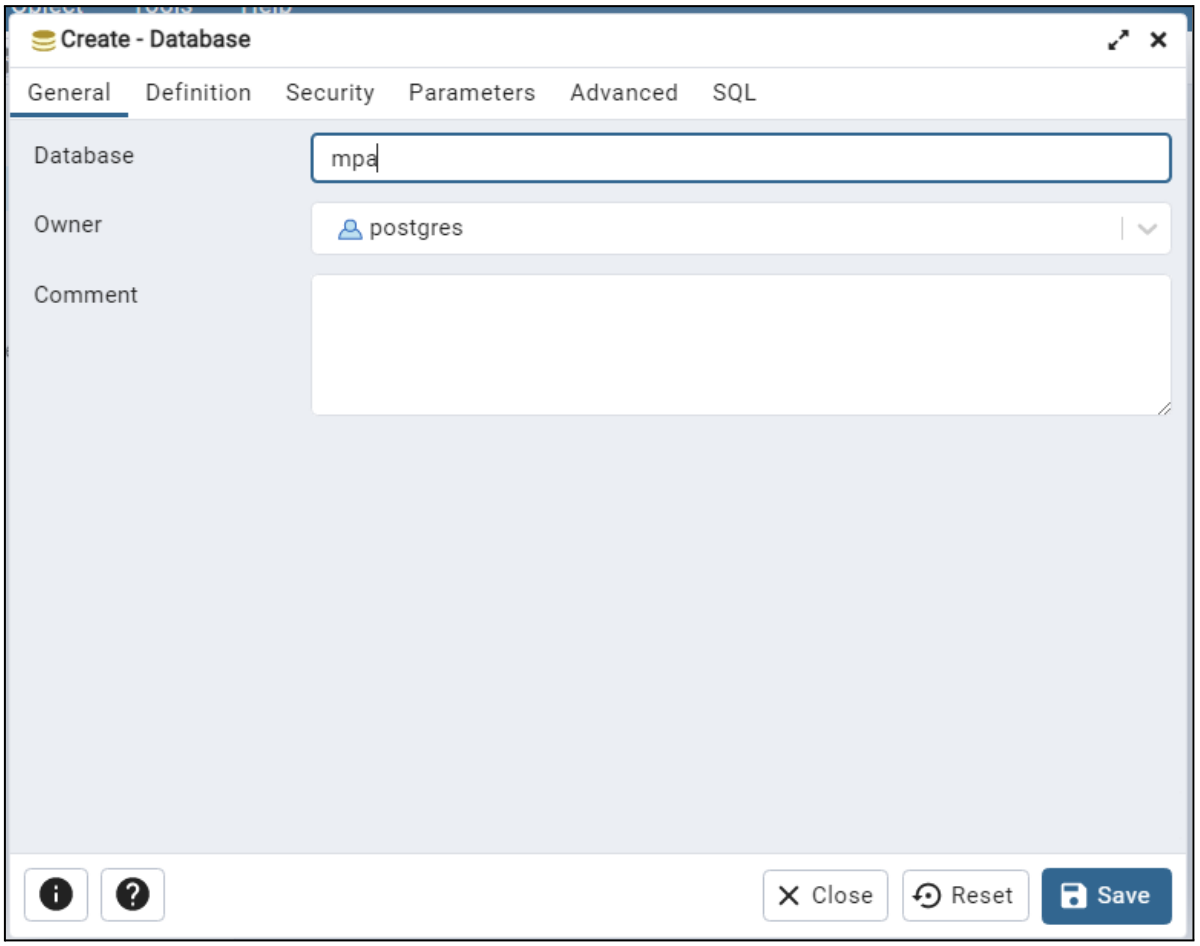


Figura 166. Configuració de la nova base de dades.

Amb aquests passos ja tenim a punt la base de dades. Obrim el projecte i creem un fitxer .env per definir les variables d'entorn. L'omplim amb la informació corresponent. *Figura 167*. Les variables de Google i AWS es poden aconseguir seguint els passos indicats en els seus llocs web.

```
DATABASE_URL="postgresql://usuari:contrasenya@localhost:5432/nom_db?schem
a=public"
GOOGLE_OAUTH_CLIENT_ID="El teu id de Google OAtuh"
GOOGLE_OAUTH_CLIENT_SECRET="El teu secret de Google OAuth"
AWS_S3_CONTENT_BUCKET=mpatoolkit-dev
AWS_ACCESS_KEY_ID=El teu id d'accés a AWS
AWS_SECRET_ACCESS_KEY=La teva clau d'accés a AWS
AWS_REGION=Regio AWS
JWT_SECRET_KEY="secret"
DEBUG=*
```

```
LOG_TRANSPORT=pretty
ORIGIN=http://localhost:3000
VITE_UPLOAD_BASE_URL=vite.base.url
```

Figura 167. Variables d'entorn.

Un cop tenim les variables configurades instal·lem les dependències amb la comanda "yarn dev". Configurem la base de dades amb la comanda "npx prisma migrate dev" i arranquem el servidor de desenvolupament. "yarn dev", per omplir la base de dades amb dades de mostra podem fer servir la comanda "yarn db:seed" o "yarn db:nuke" per netejar i tornar a omplir la base de dades.

15.2 Manual d'usuari.

Per accedir a l'aplicació es pot fer a través de l'URL "localhost:3000". Ens portarà a la landing page i podrem navegar per tota l'aplicació (veure *Figura 168*).

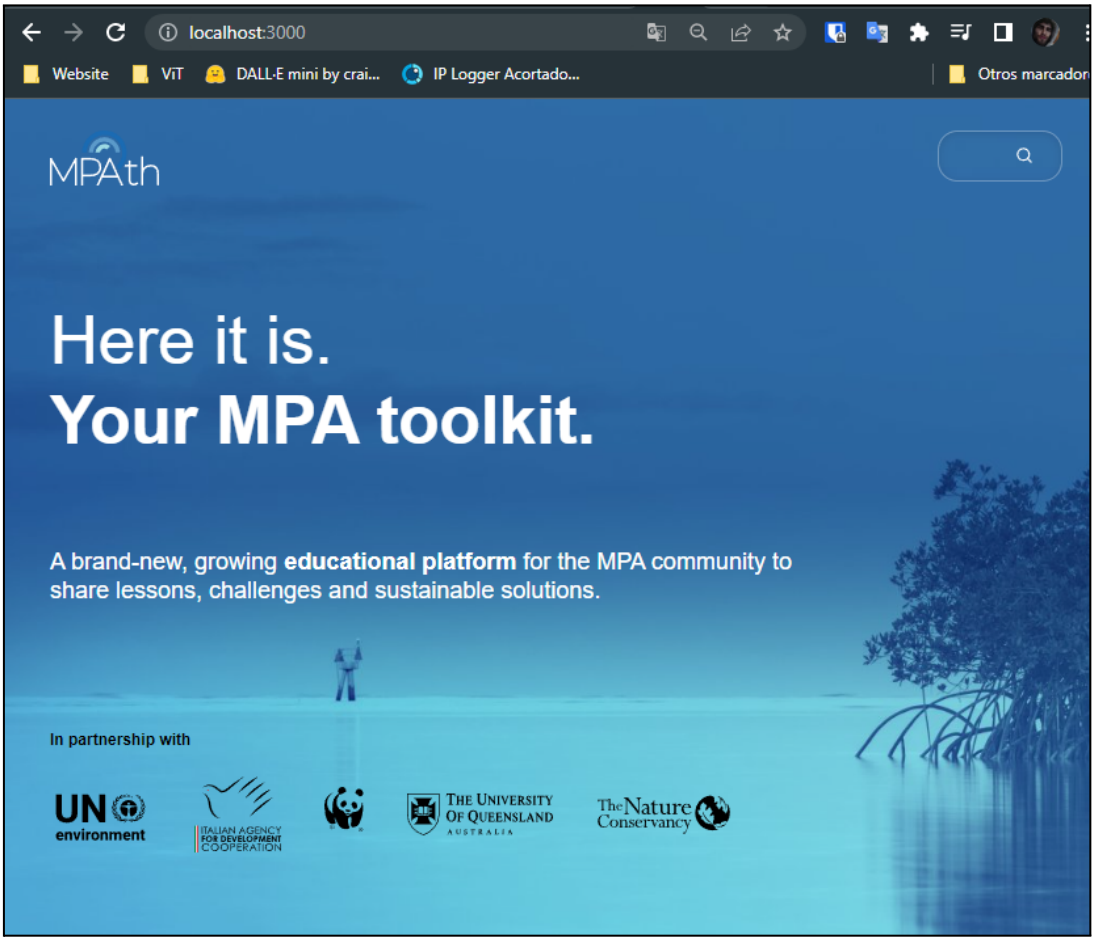


Figura 168. Accés a l'aplicació amb el navegador.

En cas que es vulgui accedir al CMS es fa a través de l'URL "localhost:3000/cms". La primera vegada que s'executa l'aplicació el primer usuari que es registri té permisos d'administrador (veure *Figura 169*).

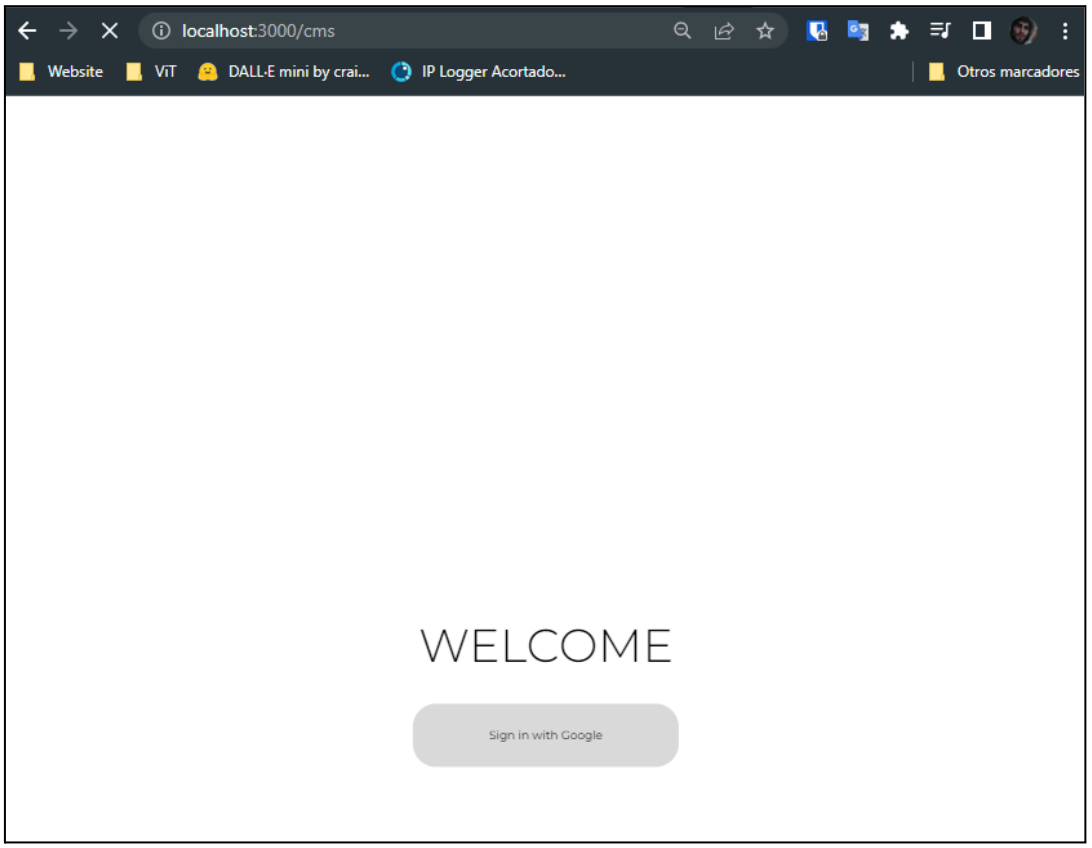


Figura 169. Accés al CMS a través del navegador.

Un cop s'ha entrat amb el compte de Google ja es mostra el menú principal del CMS. *Figura 170*.

Figura 170. Menú principal del CMS.