

# ECOMENTOR

---

PROJECTE D'ENGINYERIA DEL SOFTWARE, Q2 2024-2025  
INCEPCIÓ 1 i 2  
GRUP 22

Víctor Díez Serrano, Back-end Developer

Dídac Dalmases Valcárcel, Scrum Master 1 & Front-end Developer

David Mas Escudé, Scrum Master 2 & Back-end Developer

Rubén Palà Vacas, Front-end Developer & Architectural designer

David Sanz Martínez, Back-end Developer

Neptune Christoper Lumayag Cartalla, Front-end Developer



<b>ECOMENTOR</b>	<b>1</b>
1. Requirements	3
1.1. General conception of the project	3
1.2. “NOT” list	4
1.3. Conceptual model	5
1.4. Summary of product backlog in the document	5
1.5. Non-functional requirements	9
1.5.1 Performance Requirements	9
1.5.2 Security Requirements	9
1.5.3 Usability Requirements	9
1.5.4 Reliability & Availability Requirements	9
1.5.5 Scalability Requirements	10
1.5.6 Maintainability & Extensibility Requirements	10
1.5.7 Compliance & Legal Requirements	10
1.6. Treatment of transversal aspects	11
1.7. Third-party services	12
2. Methodology	13
2.1. Project management	13
2.2. Repository management	13
2.3. Communication within the team	15
2.4. Quality management	15
2.5. Testing strategy	16
2.6. Management of configurations	16
2.7. Interaction with colleagues	16
2.8. Bug management	17
2.9. NFRs treatment	17
2.10. Coding assistants	18
3. Technical description	19
3.1. Overall conception of the architecture	19
3.1.1. Physical architecture	19
3.1.2. Architectural pattern(s) applied	21
3.2. Domain layer	21
3.2.1. Domain model diagram (optional)	21
3.2.2. Design patterns applied	21
3.3. Database diagram (UML)	22
3.4. Instrumentation and list of technologies	23
3.5. APIs	23
3.5.1 API nostre	23
3.5.2 APIs externes que utilizem	24
3.5.3 Service Consumption	24
3.6. Development tools and working environment	24

# 1. Requirements

## 1.1. General conception of the project

### 1.1.1 La aplicació

El nostre projecte té l'objectiu de desenvolupar una app per **millorar l'eficiència energètica dels edificis a Catalunya**, amb el propòsit de promocionar un estil de vida més **sostenible i ecològic**.

L'aplicació permet a tots els usuaris visualitzar de manera dinàmica en un **mapa** la informació energètica dels edificis del territori català -proporcionada pels certificats energètics-. Es podran veure diferents dades i també comparar dos edificis amb certificat energètic.

**ECOMENTOR** també permet als usuaris que no tinguin un certificat energètic la possibilitat d'obtenir un *pseudo-certificat digital* a través d'un petit **qüestionari**, on es valoraran superficialment certes característiques de l'edifici i es retornarà a l'usuari un resultat. A més a més, l'aplicació oferirà ajuda per obtenir el certificat.

Per als usuaris que ja tenen un certificat, es podran fer consultes a un **xatbot** de com millorar el seu immoble en termes de **sostenibilitat i consum**. Els usuaris també tindran a la seva disposició una **calculadora** on podran veure de forma **quantitativa** quin seria el benefici o el perjudici de canviar certs aspectes de l'immoble.

### 1.1.2 Dades rellevants

Per substantiar aquesta observació hem consultat diverses fonts oficials com l'Informe de l'Estat dels Certificats Energètics el 2023 i les dades parlen per elles mateixes. Veiem que més del 80% dels edificis nous obtenen les qualificacions A, B i C mentre que les edificacions existents només menys del 16% d'edificis registrats superen el nivell E.

També hem vist interessant cercar quants edificis no tenen un certificat energètic. Segons l'informe,

*El nombre total de certificats registrats als corresponents registres autonòmics ascendeix a 5.978.358, dels quals els edificis existents representen el 97,58%. (Pàg. 2 de l'Estat de Certificats Energètics dels Edificis)*

Com que la majoria d'edificis tenen certificat, també busquem el nombre d'habitatges amb certificat. Encara que no hi hagi números oficials, hem trobat a CoHispania, la tasadora homologada oficial del Banc d'Espanya, que el 56% dels habitatges oferts a Espanya no té publicat el certificat energètic, sigui perquè no en disposen o perquè tenen una mala qualificació.

Amb totes aquestes dades, hem vist una oportunitat per fer una aplicació que pugui resoldre aquests problemes.

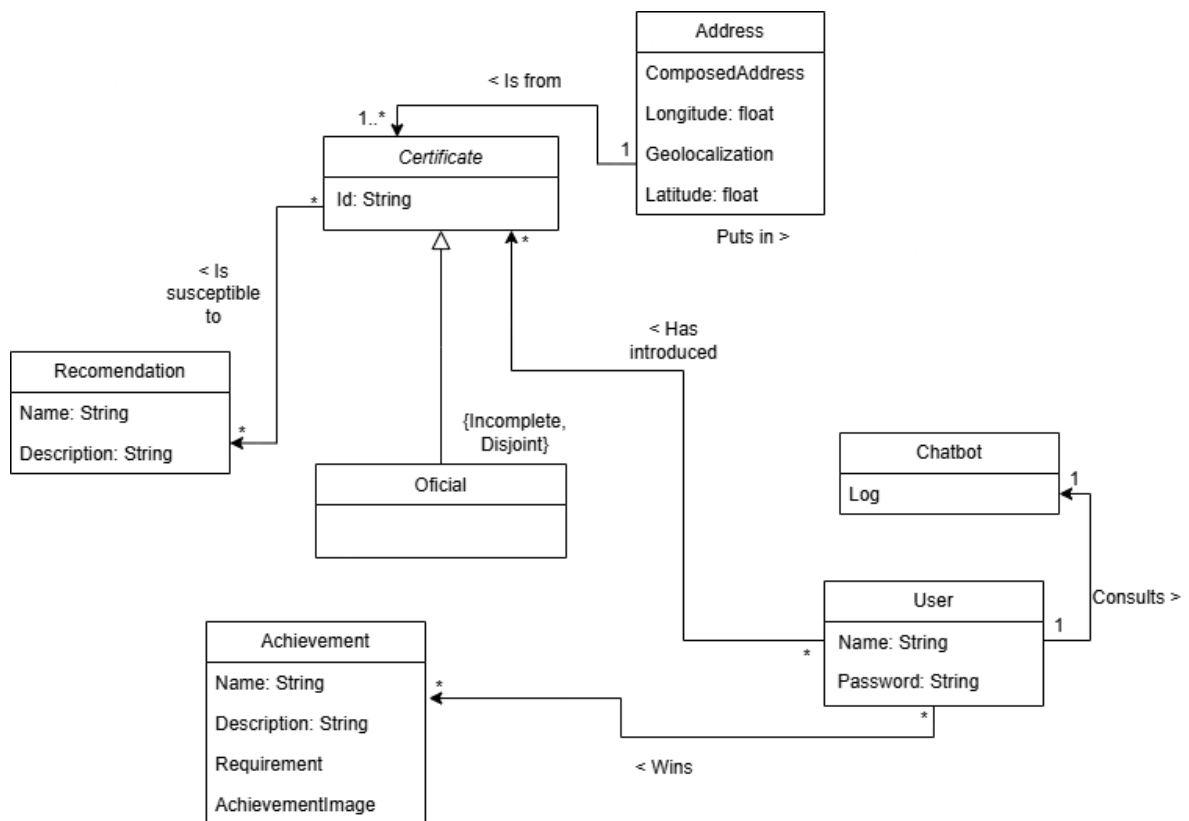
## 1.2. “NOT” list

Per acabar de clarificar les característiques del projecte, s'ha elaborat una *NOT LIST* per clarificar quina és la **identitat** de la nostra aplicació.

SÍ	NO
Mapa interactiu on veure els certificats elèctrics que tenen els edificis	Abast del mapa global (només a catalunya, Restricció Data set)
Mapa amb filtres per poder filtrar segons consum, nota del certificat, emissions	Actualització de les dades en temps real
Oferim un sistema per poder obtenir certificats electricos no oficials.	Oferim on comprar les millores esmentades en les recomanacions només diem el preu estimat que pot tenir
Oferim un sistema de recomanacions per poder millorar la nota del certificat/Consum elèctric indicant el preu estimat que suposa aplicar la millora	Oferim un sistema per obtenir certificats elèctrics oficials, que siguin reconeguts de manera oficial
Oferim un chatbot especialitzat per poder demanar recomanacions encara més personalitzades	
Comparació energetica entre edificis	

Visualització de dades històriques sobre edificis amb certificats elèctrics a catalunya	
Sistema de gamificació	
POTSER	
Visualització d'esdeveniments propers a edificis amb certificats energètics	
Calendari per avisar quan renovar un certificat	

### 1.3. Conceptual model






Disseny conceptual UML









## 1.4. Summary of product backlog in the document

Hem dividit les èpiques del nostre projecte en funcionalitats principals de l'aplicació, com és el mapa, els usuaris, els certificats oficials dels edificis, els certificats no oficials que oferim, el Xatbot, els trofeus per a la gamificació i la web admin per gestionar l'aplicació internament. Hem fet aquesta divisió per poder atomitzar les tasques per funcionalitats i d'aquesta manera poder separar millor les implementacions de les característiques de l'aplicació en els diferents sprints. Amb aquesta distribució ens assegurem que hi hagi una bona repartició de feina i cap sprint es quedi coix respecte a un altre.




Amb el nom de la tasca podem saber si es tracta d'una del front o del back, a més cada una disposa de la seva DoD en la descripció amb comentaris que reforcen el desenvolupament de cada tasca. Pel que fa a les Històries d'Usuari, també compten amb la seva frase <Com a [rol o tipus d'usuari], vull [acció o funcionalitat] per a [benefici o raó]>, junt amb el criteri d'acceptació per tal de mantenir un estàndard en la finalització d'aquestes.

NAME	PROJECT	SPRINT
⋮ ▾ ● #1 Gestionar Mapa		
#6 Visualizar mapa ●		Sprint 1
#8 Filtrar Mapa ●		Sprint 1
#16 Buscar localització ●		Sprint 1



Taiga Èpica Gestionar Mapa

⋮ ▾ ● #4 Gestió d'Usuaris		
#22 Registre d'usuari ●		Sprint 1
#65 Iniciar sesión ●		Sprint 1
#67 Actualitzar perfil ●		Sprint 1
#69 Eliminar compte ●		Sprint 1
#70 Veure historial d'activitat ●		
#234 Escollir preferència d'idioma ●		Sprint 1
#235 Navegació de l'app ●		Sprint 1
#238 Refutació ●		

### Taiga Èpica Gestió d'usuaris

⋮ ▾ ● #2 Gestionar Edificis		
#10 Conseguir y mostrar info edifici(s) ●		Sprint 1
#13 Comparar info edificis ●		
#20 Consultar dades històriques ●		

### Taiga Èpica Gestionar Edificis

⋮ ▾ ● #18 Gestionar Chatbot		
#147 Utilitzar chatbot ●		

### Taiga Èpica Gestionar Chatbot

⋮ ▾ ● #5 Gestionar Certificats			
#19 Visualitzar-certificat ●	🌿	Sprint 1	
#21 Asistir en certificat (calculadora) ●	🌿		
#112 Vincular-certificat-oficial ●	🌿	Sprint 1	
#17 Obtindre recomanacions ●	🌿		

## Taiga Èpica Gestionar Certificats

⋮ ▾ ● #189 Gestió de Trofeus			
#190 Desbloquejar Trofeu ●	🌿		
#281 Seguiment trofeu ●	🌿		

## Taiga Èpica Gestió Trofeu

⋮ ▾ ● #241 Web-Admin			
#242 Creació entitats ●	🌿	Sprint 1	
#244 Set-up-deployment ●	🌿	Sprint 1	
#275 Creació de la web-admin ●	🌿		

## Taiga Èpica Web-Admin



## 1.5. Non-functional requirements

### 1.5.1 Performance Requirements

- El sistema ha de recuperar i mostrar les dades de certificats energètics en un termini de vuit segons en condicions normals de la xarxa.
- La interfície del mapa ha de respondre a les interaccions de l'usuari (zoom, panoràmica, filtre) en (per determinar) segons.
- L'algoritme de recomanació per millorar l'eficiència energètica hauria de retornar suggeriments en un termini de (per determinar) segons.

### 1.5.2 Security Requirements

- L'autenticació d'usuari ha de ser compatible amb OAuth 2.0 i la validació de testimoni basada en JWT.
- Totes les dades delicades dels usuaris s'han de xifrar en repòs i en trànsit mitjançant el passwordencoder de spring framework.
- El sistema hauria d'implementar el control d'accés basat en rols (RBAC) per restringir les modificacions de dades.

### 1.5.3 Usability Requirements

- L'aplicació hauria de tenir una interfície d'usuari intuïtiva amb un tauler de control fàcil d'utilitzar i coherència estètica entre pantalles.
- Els usuaris haurien de poder completar les tasques bàsiques (p. ex., comparar edificis) en cinc passos o menys.

### 1.5.4 Reliability & Availability Requirements

- El sistema hauria de mantenir un temps d'activitat del 99,9%.

### **1.5.5 Scalability Requirements**

- El sistema hauria de gestionar 500 usuaris simultàniament sense una degradació significativa del rendiment.
- El sistema hauria d'emmagatzemar i recuperar de manera eficient les dades històriques dels certificats energètics durant almenys cinc anys.

### **1.5.6 Maintainability & Extensibility Requirements**

- La base de codi ha de seguir principis d'arquitectura neta mitjançant Linter i estar ben documentada amb Swagger.
- Les proves unitàries haurien de cobrir almenys el 80% de les funcionalitats crítiques.

### **1.5.7 Compliance & Legal Requirements**

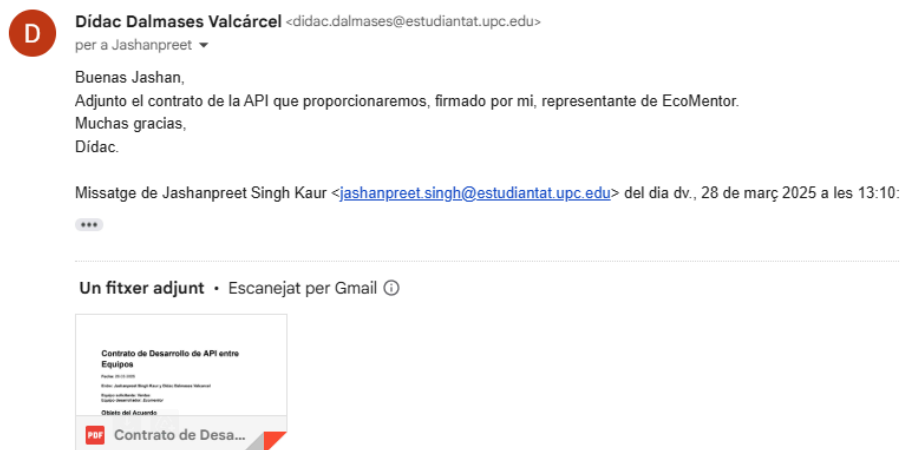
- El sistema hauria de registrar les interaccions dels usuaris per auditar-les, garantint el compliment dels estàndards de seguretat ISO 27001.

## 1.6. Treatment of transversal aspects

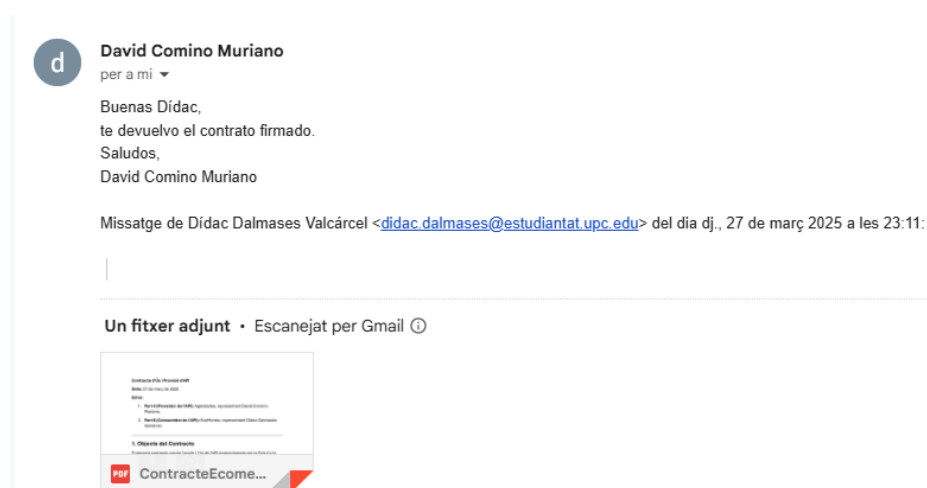
Aspect	Description	Current status
Geolocalització	Mostrem un mapa amb diferents punts que mostren els certificats energètics dels edificis.	Implementat en l'sprint 1.
Xarxes socials	Login.	Implementat en l'sprint1, falta implementar les opcions de login amb google. Es farà en l'sprint2.
Xat	La nostra aplicació tindrà un xatbot per poder fer consultes sobre certificats elèctrics.	El pla és implementar aquest aspecte en el sprint 3.
Gamificació	Trofeus per a l'usuari quan completa tasques relacionades amb els certificats.	S'implementaran en l'sprint 2.
Stakeholders reals		
Refutació	Enviar notificacions per correu quan es compleix un trofeu i permetre bloquejar comptes des de la web admin.	S'implementarà en l'sprint 2.
Calendari		
Web-app admin	Tindrem un apartat on només podrà accedir l'admin per fer CRUD d'entitats del sistema i poder bloquejar usuaris.	S'implementarà en l'sprint 3
Multiidioma	Possibilitat de canviar l'idioma de l'app entre. català, castellà i anglès	Implementat en l'sprint 1
<i>Add others as required...</i>		

## 1.7. Third-party services

S'ha realitzat un acord on EcoMentor, nosaltres, consumirà el servei ofert per Agendados i oferirà el seu servei a Ventus. Com es pot veure a les següents imatges, s'ha realitzat un contracte per arribar a un acord i mantenir-lo durant el desenvolupament, on ha estat firmat per ambdues parts



Correu firmant l'acord de la nostra API per a Ventus.



Correu de Agendados firmant l'acord de la seva API per a EcoMentor.

Els contractes es poden trobar a la carpeta ContractesAPI de l'entrega del sprint 1. En aquests, s'acorda que de la nostra part, oferirem un endpoint per a Ventus on donades unes coordenades (o una altra mesura d'àrea, no especificat al contracte), retornarem els edificis amb millor qualificació que es trobin en tal zona. Degut a que podem tornar tant com informació basada en la localització com dades ja sobre l'energia i emissions, no serà un problema adaptar-nos.

Per la part de l'API que consumirem, hem acordat amb Agendados que la seva API, donat una àrea definida per una latitud i longitud mínimes i màximes, obtenir events que passaran en aquella zona i la seva informació, d'aquesta manera poder mostrar la informació al nostre mapa o als edificis d'aquest.

## 2. Methodology

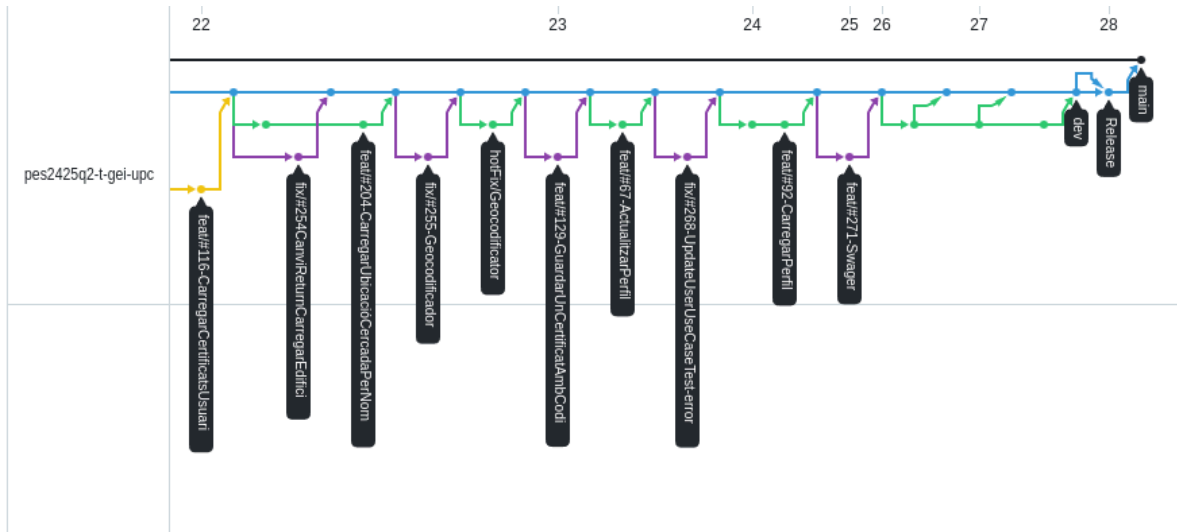
Durant el desenvolupament del projecte, hem seguit un enfocament àgil, utilitzant Scrum com a metodologia principal per organitzar el treball i gestionar els lliuraments de manera iterativa. Aquest enfocament ens ha permès adaptar-nos als canvis, validar progressivament les funcionalitats i mantenir una comunicació fluida dins de l'equip.

### 2.1. Project management

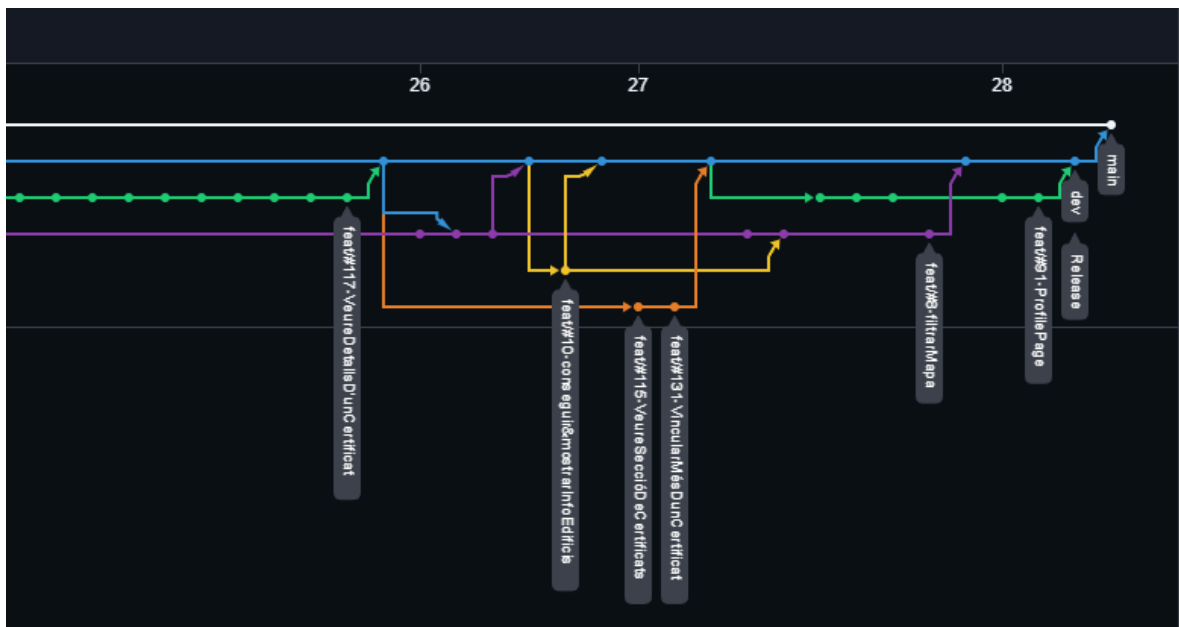
- Fem servir **Taiga** per gestionar el backlog i fer seguiment de les tasques amb **story points** per prioritzar i estimar l'esforç de cada història d'usuari.
- Les tasques al Taiga procuren tenir suficients dades per ser implementades, a més un cop completades, el temps que es va portar en realitzar-les.
- Els sprints es defineixen d'acord amb als objectius clau i les tasques es tanquen mitjançant revisió col·laborativa.
- Les reunions de revisió permeten detectar bloquejos i ajustar el roadmap.

### 2.2. Repository management

- Implementem **GitHub** com a sistema de control de versions, seguint un model **Git Flow** amb branques separades per funcionalitats, desenvolupament i producció.
- Cada canvi passa per un procés de revisió via **pull requests** abans de ser fusionat a la branca principal.
- Hem decidit cada cop que desenvolupem una funcionalitat de l'aplicació començar una nova branca des de dev amb el nom **feat/#<nombreTaiga>-<nomTasca>** per tal de mantenir una coherència durant tot el desenvolupament.
- Un cop completa la tasca hem decidit que l'estructura dels commits ha de ser **Task #<numTaiga> descripció de la tasca** per tal de saber el context del desenvolupament d'aquella funcionalitat.
- Si dona resulta que hi ha hagut una petita errada s'obre una nova branca anomenada **bugFix/<IssueDelTaiga>**.
- Es fan commits freqüents per assegurar la traçabilitat dels canvis.



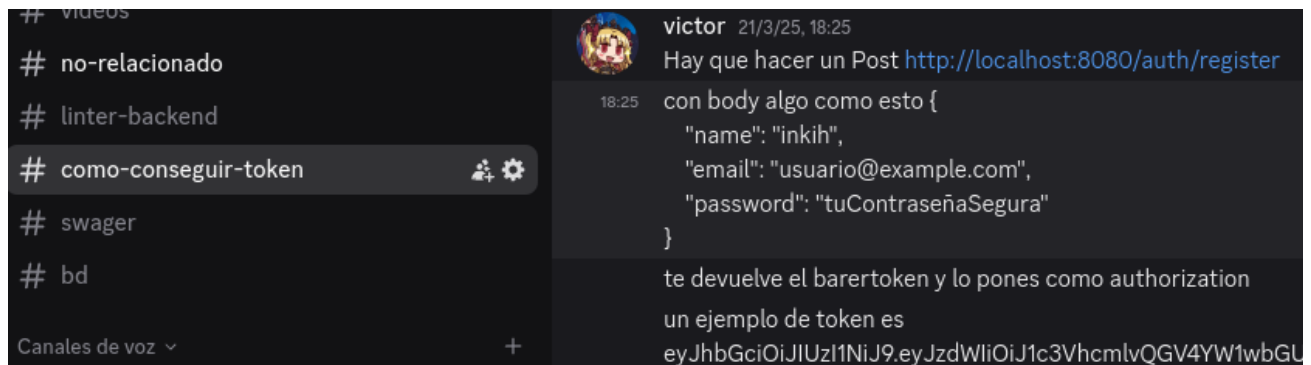
GitFlow del back-end



GitFlow Front-End

### 2.3. Communication within the team

- Per a la comunicació diària, fem servir un **grup de WhatsApp** per missatges ràpids i un **servidor de Discord** per reunions més extenses i diferents canals que es van creant cada cop que necessitem una consulta amb l'equip per a alguna determinada tasca o alguns errors que es produeixen en el desenvolupament a tall de fòrum, com a tutorials o explicant les crides a l'API això ens permet tindre sempre disponible i de manera endreçada els missatges relacionats amb un tema.



Exemple de tutorial al discord de Ecomentor

- Es realitzen **dailies informals** per sincronitzar l'equip i resoldre possibles bloquejos. Es realitzen cada cop que tenim classe presencial i ajuda a saber què és el que hem de fer d'ara en avant i valorar la feina que s'ha anat realitzant.
- Taiga facilita la documentació i assignació clara de responsabilitats, de manera que cada tasca conté la seva DoD adjunta al seu objectiu, el seu requisit i les branques del Github afectades, per tal de saber quan s'ha de donar la tasca com finalitzada.

### 2.4. Quality management

- Hem establert **estàndards de codi** per garantir coherència i mantenibilitat.
- Les revisions de codi són una part essencial del procés, assegurant bones pràctiques i identificant possibles errors abans del merge.
- Integrem un linter i proves unitàries al nostre pipeline de GitHub per evitar pujar codi que no compili ni falli en executar-se. El linter (Tant el linter del back-end com el del front-end està definit a la pàgina 24-25) verifica errors d'estil i sintaxi, mentre que els tests unitaris validen el correcte funcionament del codi.

## 2.5. Testing strategy

- Utilitzem proves unitàries i integració contínua per validar les funcionalitats abans del desplegament, hem fet servir Mockito a l'hora de fer els tests unitaris en el back-end.
- Es farà servir **Jest** per a les proves unitàries del front-end més endavant de l'sprint 1, validant la funcionalitat dels components i mòduls abans del desplegament.

## 2.6. Management of configurations

- Les configuracions es gestionen mitjançant **Docker** i variables per adaptar el projecte a diferents entorns sense comprometre la seguretat.
- Hem fet servir Docker per tal que no hi hagués problemes amb el projecte en múltiples sistemes i entorns de treball fent servir volums per emmagatzemar la base de dades amb diferent tamany de dades per tal de fer diferents proves.
- La base de dades en la qual basem el nostre projecte és un arxiu CSV amb més d'un milió de files i seixanta-nou columnes, el qual suposa una càrrega de dades molt gran, a més moltes de les columnes era informació no rellevant o que no ens aporta cap mena d'informació en el que tenim plantejat per a la nostra aplicació. Per això, hem seguit unes certes mesures per tal de filtrar la informació que ens és important i netejar les taules per tal d'optimitzar la base de dades i evitar problemes amb alguns valors nuls.
- Tenim dos volums, ja que són els que tenim pensats. Cada volum seria pràcticament igual, però a l'apartat de certificats, un en tindrà molts més (més files) per testejar temps de càrrega de certes views i l'altre en tindrà menys per poder testejar i desenvolupar altres funcionalitats de forma més fluida. Aquest procés si va bé només cal fer-ho un cop i després és anar escollint el volum desitjat.

## 2.7. Interaction with colleagues

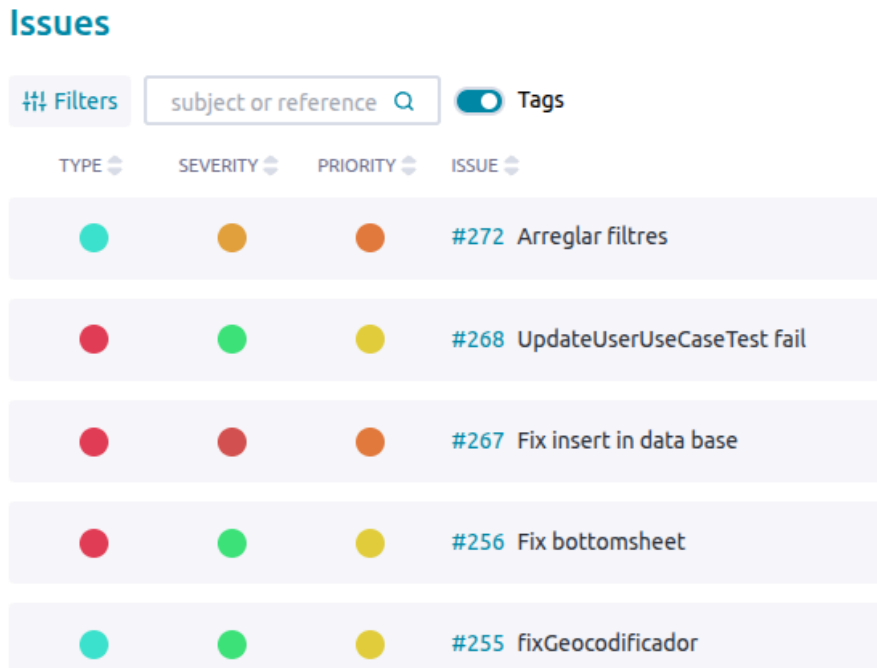
Per interactuar amb els altres grups realitzem reunions presencials durant les hores de classe també, hem creat un grup de WhatsApp per poder parlar amb ells fora de l'horari escolar.

També hem fet un contracte amb els altres dos grups per poder especificar el que ha de fer cadascun i no hi hagi problemes en el futur.



## 2.8. Bug management

- Els errors es registren a Taiga com a **incidències**, prioritzant-se segons l'impacte en el projecte.
- Obrim incidències al Taiga per tal de notificar el bug a l'equip i creem branques noves anomenades **bugFix/<IssueDelTaiga>** per tal de corregir-lo sense afectar la resta del projecte.



Alguns dels issues del taiga

## 2.9. NFRs treatment

Per cada apartat dels NFR mencionat en la secció [1.5](#) del document, aquesta és l'estratègia que estem seguint per complir amb l'establert i mantenir la integritat d'EcoMentor:

### 2.9.1 Performance requirements

- Avaluació de UI: Es definiran temps específics per a la resposta del mapa interactiu i l'algorisme de recomanació, amb proves d'usabilitat per verificar-ne el compliment.

### 2.9.2 Security Requirements

- Autenticació i autorització: S'implementarà OAuth 2.0 i JWT per validar usuaris, amb proves de penetració per identificar vulnerabilitats.
- Encriptació de dades: S'assegura que totes les credencials i dades delicades estiguin xifrades tant en trànsit com en repòs mitjançant Spring Security i PasswordEncoder.
- Control d'accés: S'aplica RBAC amb validacions automàtiques per garantir que només els usuaris autoritzats puguin modificar dades.

### 2.9.3 Usability Requirements

- Interfície intuïtiva: Es duran a terme proves d'usuari per avaluar la facilitat d'ús del dashboard i la coherència visual a les pantalles.
- Reducció de passos: Es mesuraran les tasques clau (comparar edificis, consultar recomanacions) per assegurar que es puguin completar en un màxim de cinc passos.

### 2.9.5 Scalability Requirements

- Proves de càrrega: Se simularan 500 usuaris concurrents amb eines com ara Gatling per garantir un rendiment estable.
- Optimització d'emmagatzematge: Es dissenyarà un model de dades eficient per manejar 5 anys de registres de certificats energètics sense afectar-ne el rendiment.

### 2.9.6 Maintainability & Extensibility Requirements

- Codi net i documentat: S'aplicarà Linter per verificar la qualitat del codi i es documentaran les APIs amb Swagger.
- Cobertura de proves: Es mantindrà un 80% de cobertura en funcionalitats crítiques amb proves unitàries a JUnit.

### 2.9.7 Compliance & Legal Requirements

- Registre d'activitat: S'auditaran totes les interaccions dels usuaris d'acord amb ISO 27001 mitjançant logs centralitzats.
- Compliment normatiu: Es garantirà que la gestió de dades i seguretat segueixi les regulacions aplicables a Catalunya i la UE.

## 2.10. Coding assistants

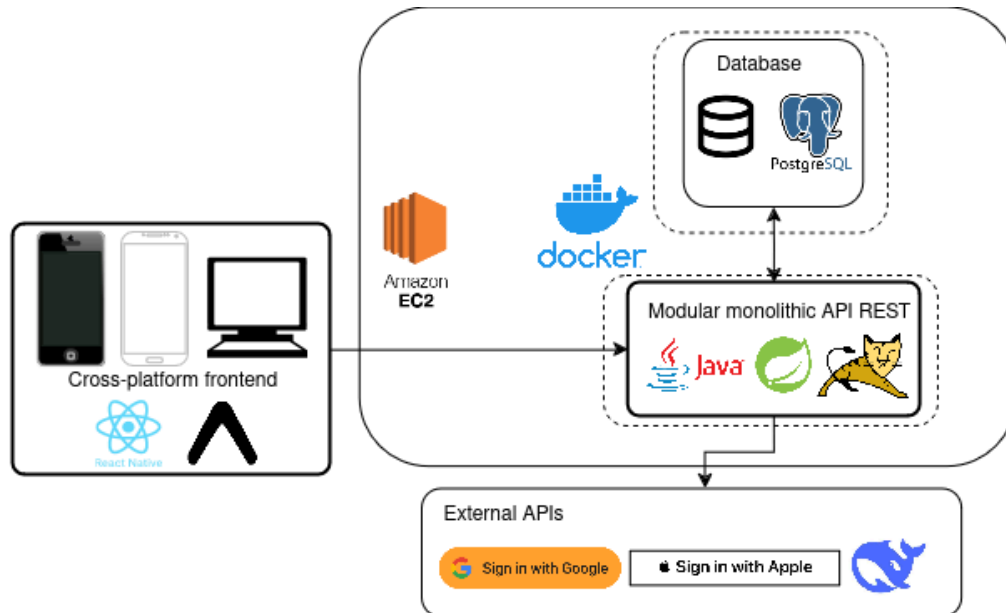
- Hem fet ús d'eines d'ajuda al desenvolupament, com **linters** per garantir la qualitat del codi i assistents d'IA per suggeriments i optimitzacions.
- Per tal de tenir una programació eficient, utilitzem GitHub Copilot amb l'IntelliJ que ens ajuda a fer de manera ràpida els tests unitaris, corregir-nos si tenim algun error de codi i resoldre dubtes.

## 3. Technical description

### 3.1. Overall conception of the architecture

#### 3.1.1. Physical architecture

L'arquitectura de l'aplicació es veu representada en el següent diagrama.



Aspectes claus i justificació:

- El front-end està desenvolupat amb **React Native** i **Expo**. L'ús d'aquesta tecnologia ens permet desenvolupar una aplicació multiplataforma amb un afegit de codi mínim. S'ha escollit aquesta opció per davant d'altres similars com **IONIC** per el coneixement previ dels membres de l'equip en **React**. Hem considerat que utilitzar una tecnologia coneguda seria un benefici important. **Flutter** es va considerar, ja que presentava avantatges en **llibreria de components** i **optimització**, però la diferència no era suficient com per justificar l'aprenentatge de **Dart**.
- Pel que fa el back-end, es desenvoluparà una aplicació en **Java Spring Boot**, seguint la arquitectura de software **modular monolithic**. L'aplicació s'executarà dins d'un contenidor **Docker**. L'arquitectura modular monolithic ens permetrà desenvolupar codi d'una manera més divisible, clara i neta. Per més detall veure [Design patterns applied](#).

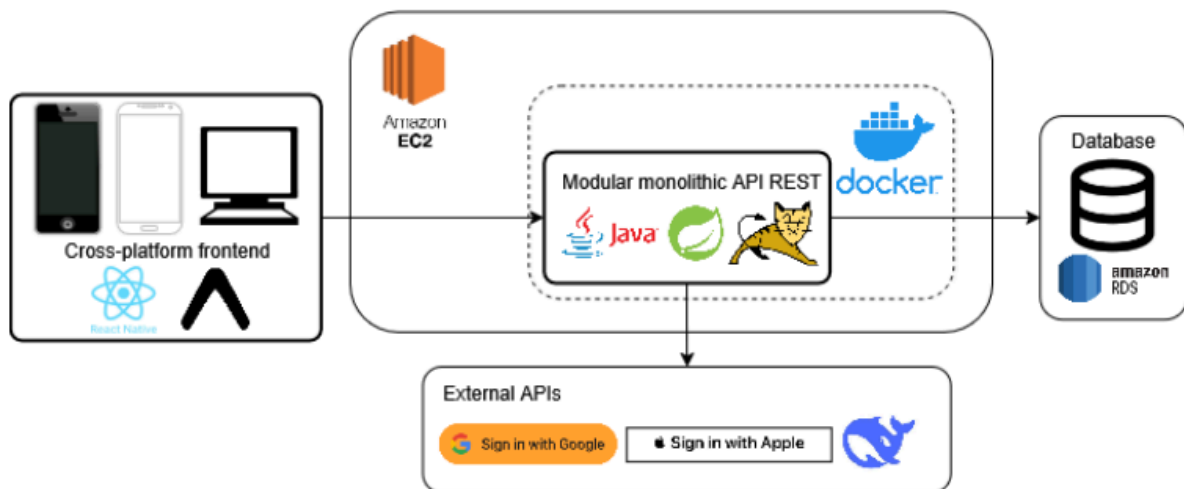
La principal justificació per escollir Java Spring Boot és: la seva gran compatibilitat amb **React Native**; la llibreria **ORM** de **Spring Boot**, **Spring Boot JPA**, que permet un disseny de base de dades àgnostic del **SGBD**; els mòduls de testing de **Java**, com **Mockito**, que proporcionen un entorn de testing molt robust. Es van contemplar altres opcions com **Django**, però donat el coneixement previ dels membres en **Java** es van sacrificar els possibles avantatges (middleware més potent, codi més net i gran integració) en favor de **una tecnologia coneguda**.

- L'SGBD escollit és **Postgres**, principalment per la seva gran compatibilitat amb **SpringBoot JPA** i per la seva utilització en projectes previs. El fet de que sigui codi obert també ha sigut un punt a favor.
- L'aplicació utilitzarà com a **APIs** externes (a nivell d'arquitectura física) les APIs d'autenticació de Google i Apple i l'API de Deepseek. Les primeres dos s'escullen per ser les dos opcions més grans i importants a nivell de login amb comptes externs. L'API de Deepseek l'hem escollit donat que és una IA bastant recent i amb resultats prometedors, a més de que ofereix una accessibilitat una mica més gran que APIs com la de OpenAI.
- Finalment, pel deployment hem escollit **Amazon Web Services**, concretament un EC2 amb la següent configuració:
  - Tipus d'instància: **t2.large**
  - Memòria: 8GB
  - Procesador: 2vCPUS

S'ha escollit **AWS** per dos principals raons:

- Àmplia documentació, tutorials i integració amb **Docker** (gestió de contenadors amb **ECS** per al CD)
- Crèdit de 300\$ per poder fer un deployment realista.

Donada la integració amb AWS, no es descarta en alguna fase del desenvolupament canviar l'arquitectura a aquesta:



On el principal canvi és que l'SGBD ja no serà intern, sino que serà el servei **RDS** d'**AWS**. No obstant, per no dificultar el desenvolupament és treballarà amb l'arquitectura amb **Dockers** i només si no es presenten problemes es plantejarà aquesta segona arquitectura.

Com que **JPA** és agnòstic de l'SGBD, qualsevol canvi que fem no afectarà a l'aplicació **Java**.

### 3.1.2. Architectural pattern(s) applied

Per facilitar el desenvolupament el patró arquitectònic és **monolític**. Es va plantejar una arquitectura basada en **microserveis**. No obstant, donada la seva complexitat i el poc coneixement de les tecnologies requerides per implementar-ho, es va escollir una arquitectura **monolítica** que, tot i ser menys elegant, és més pràctica i ràpida d'implementar per un projecte de poca duració com aquest.

## 3.2. Domain layer

### 3.2.1. Domain model diagram (optional)

### 3.2.2. Design patterns applied

#### Clean code architecture

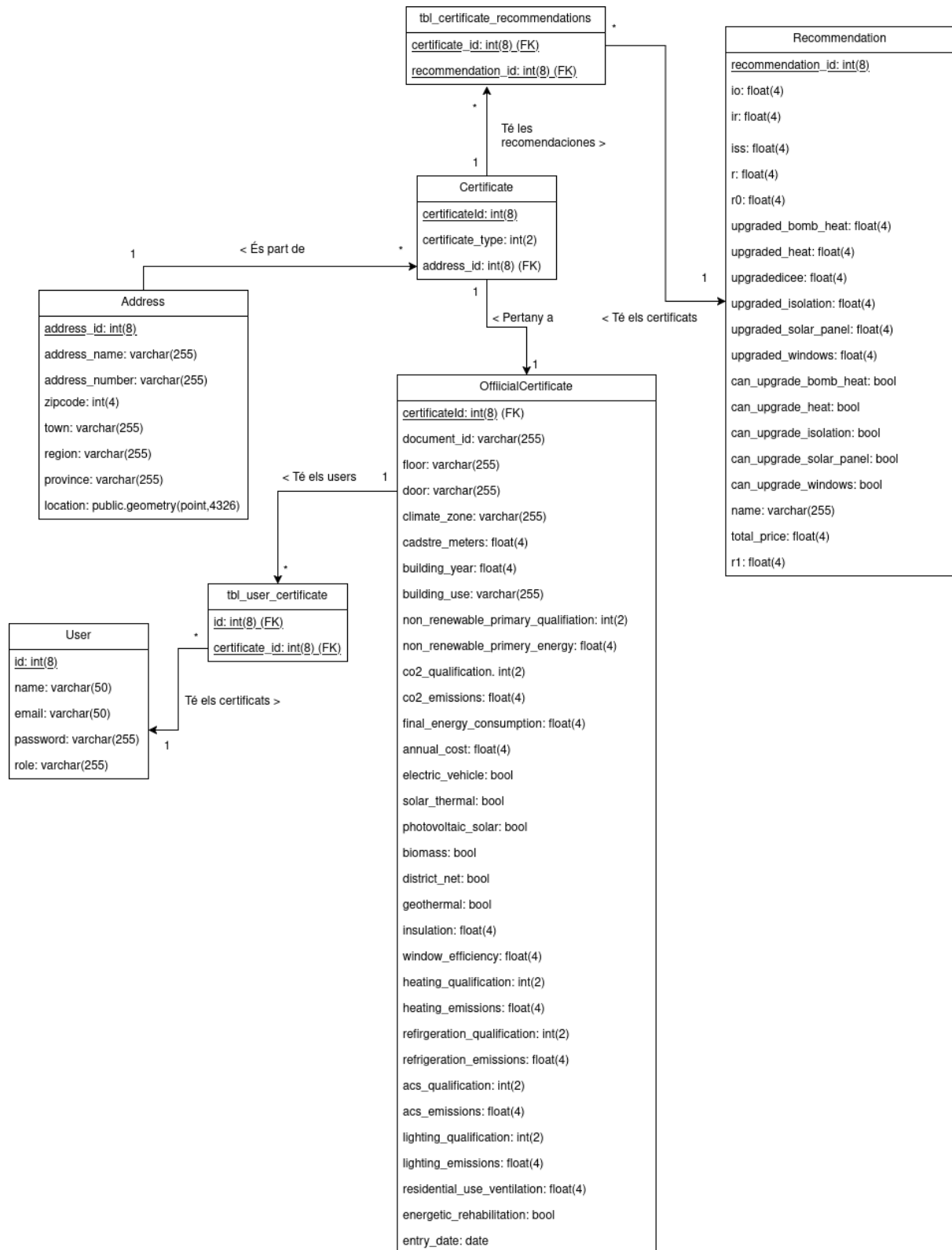
És un enfocament al disseny software que defineix diferents capes que ens permet que el codi sigui fàcil de modificar i mantenible a causa de les capes. De les diferents capes tenim primer els interface adapters, en el nostre cas utilitzem controladors (i aquí tindrem també repositoris per a mètodes) que ens permeten fer la traducció per a les entitats externes de la que usem a les entitats i use cases. Després tenim les entitats que en el nostre cas són les classes que guardarem a la base de dades, o sigui, que seran els objectes del sistema. En última instància tenim els diferents use cases que implementem. Normalment, els controladors cridaran als use cases que controlaran les entitats.

Hem optat per aquesta opció per una sèrie de motius. Un dels més importants era que buscàvem treballar en un ambient que poguéssim aprendre i entendre ràpidament per poder posar-nos a fer el codi l'abans possible i sense perdre gaire temps i aquest enfocament ens ho permet. A més a més, podem anar treballant de forma més o menys paral·lela en les nostres branques sense trepitjar-nos gaire. Aquesta estructura també ens permet fer tests de forma senzilla i en seguir-la si necessitem mirar codi que no hem fet és bastant fàcil d'entendre i agradable de llegir.

#### DTO

Utilitzem DTO (Data Transfer Object) per enviar la informació necessària al front-end o per crear files a la base de dades. Hem considerat aquest format perquè a la comunicació front-end-back-end fem servir text JSON, ja que és bastant llegible i està estandarditzat, i perquè considerem que és una forma neta d'enviar la informació.

### 3.3. Database diagram (UML)



Aquestes classes són les que tenim actualment a la base de dades. Per representar les subclasses de certificat hem optat per Table per class, així que ara tenim 2 taules i acabarem tenint tres taules de certificats. Els certificats No oficials, els que podran

crear els usuaris d'Ecomentor, s'afegiran al sprint 2 com una segona subclasse de Certificate fent que quedi Disjoint, Complete la relació de les subclasses. Les taules que fan de classe associativa només tenen les Foreign Keys perquè només representen la relació entre les taules. Les recomanacions ja tenen la taula creada a la base de dades pero de moment són la única taula que no utilitzem.

### 3.4. Instrumentation and list of technologies

#### Llenguatges de programació i frameworks:

- **React Native + Expo (JavaScript):** Utilitzat en la part **front-end** de l'aplicació.
- **SpringBoot (Java):** Extensió de l'*Spring Framework* que facilita molt la gestió i creació d'aplicacions **API REST**, a més de la llibreria Spring Data JPA que permet crear una aplicació amb back-end **ORM**.

#### Servidors:

- **Amazon EC2 + (Docker / Amazon RDS):** Utilitzarem un servidor Amazon EC2 per allotjar el nostre back-end. Inicialment, s'utilitzarà també **Docker** per crear un contenidor on s'allotjarà la base de dades. En etapes més avançades del desenvolupament no es descarta fer una migració a Amazon RDS.

#### Base de dades:

- **PostgreSQL:** Solució amb suport i documentació àmplia de bases de dades relacional.

#### Altres eines :

- **Postman:** Utilitzem Postman per poder fer crides a la API i poder testear-la

### 3.5. APIs

#### 3.5.1 API nostre

La nostra api està en el document **API\_ECOMENTOR** que es troba en la carpeta del sprint 1 és la conversió a pdf del nostre Swagger, on es mostren els diferents endpoints i les seves característiques.

### 3.5.2 APIs externes que utilitzem

De moment utilitzem l'api d'[icgc.cat](https://eines.icgc.cat/geocodificador/cerca?text={Nom de ciutat}) per poder fer cerques com Barcelona i ens retorni la latitud i longitud.

(<https://eines.icgc.cat/geocodificador/cerca?text={Nom de ciutat}>)

Utilitzem aquesta API ja que les dades dels certificats que disposem són de Catalunya, i d'aquesta manera permetem a l'usuari buscar un municipi ràpidament i centrar-se en ell.

### 3.5.3 Service Consumption

L'informació sobre el consum i servei de l'API està explicada a l'apartat 1.7. L'implementació de l'API que oferirem a Ventus resulta senzilla si es porta a terme adequadament, ja que actualment disposem de mecanismes per obtenir dades en una certa àrea, només caldrà filtrar i ordenar aquests resultats.

Per part de l'ús de l'API que consumirem, es trucarà depenent de la regió de l'usuari, obtenint així les dades en la seva zona per actualitzar aquells edificis més propers a events.

Sobre les dades obertes, un cop descarregades de la pàgina, vam optar per fer un filtratge exhaustiu per tal de netejar columnes que no ens eren d'utilitat i formatejar-la perquè encaixés en el nostre sistema, tot seguit vam inserir-les amb un runner amb trucades des de Postman, d'aquesta manera ens era més senzill realitzar proves de càrrega i tenir una millor gestió de les dades. Tot i així, en un futur volem plantejar canviar aquest sistema de manera que puguem fer actualitzacions constants a la nostra base de dades fent crides a l'API de la Generalitat.

## 3.6. Development tools and working environment

Per al back-end, hem utilitzat **Spring Boot** com a framework principal perquè proporciona una base robusta i escalable per a la nostra aplicació, juntament amb **JPA** per a la persistència de dades, la qual facilita la interacció amb la base de dades de manera eficient i estructurada. Utilitzem **Maven** per a la gestió de dependències i **Docker** per a la contenedorització, amb la intenció que el nostre projecte pugui funcionar en tots els entorns de desenvolupament, cosa que garanteix un entorn consistent en les diferents etapes de desenvolupament i desplegament. A més, utilitzem **Mockito** per a les proves unitàries, assegurant la fiabilitat del codi.

També, el nostre repositori de back-end compta amb un sistema d'**Integració Contínua (CI)** que imposa l'execució obligatòria de totes les proves automatitzades abans de poder realitzar un pull request. La nostra intenció és que, en el següent sprint, s'afegeixi al CI l'obligatorietat de passar el linter, encara que ja estem fent que abans de fer els pull requests amb dev, cada programador s'asseguri que el seu codi passa el linter que hem



establert, el qual és una versió modificada de google checks en la qual hem augmentat la mida mínima de la línia i hem suprimit l'obligatorietat de documentar amb javadoc.

Seguint amb el repositori back-end, s'implementa (a partir de l'sprint 2) un sistema de **Continuous Deployment**, on s'utilitza GitHub Actions per automatitzar el procés de creació de les imatges de Docker de l'aplicació (l'aplicació Java i el SGBD de Postgres) i la seva pujada a l'EC2 d'AWS. Gràcies a la *pipeline* del **CI** ens assegurem que la imatge és **vàlida**, i per tant el **CI+CD** ens assegura tenir **en tot moment** una versió **actualitzada i correcta**.

El deployment, per tant, té lloc a **AWS**, plataforma que ens proporciona moltes eines que permeten fer el procés de desplegament de l'aplicació més fàcil. L'ús que fem de la plataforma de gestió d'EC2 es limita a iniciar i aturar la nostra instància i poder monitoritzar el seu estat. Qualsevol altra gestió la farem accedint al propi EC2 mitjançant una terminal i **SSH**. Altres eines que ens proporciona **AWS** són per exemple l'ECS (Amazon Elastic Container Service) que en qualsevol moment podríem utilitzar per gestionar les nostres imatges de Docker.

Pel que fa el front-end, utilitzem **React Native** i **Expo** com a eines de desenvolupament. Expo ens ofereix la possibilitat de utilitzar molts components pre-carregats al projecte, a més de facilitat en aspectes de l'aplicació com **routing**. Fem ús de **i18n** per implementar la internacionalització de l'aplicació.

Al repositori front-end també hi ha integrada una *pipeline* de **CI** semblant a la de back-end. En aquest cas fem ús del linter **eslint**, el qual a més de comprobar aspectes estilístics també comproba errors en el codi.

Pel que fa el deployment, es fa ús de GitHub Actions i GitHub Artifacts (a partir de l'sprint 2) per mantenir actualitzada una imatge recent de la nostra aplicació sempre que hi hagi una nova **release** cap a main. D'aquesta manera, amb la pipeline **CI+CD** de front-end també ens assegurem de tenir la versió més recent de l'aplicació multiplataforma disponible.