Projecte d'Intercanvi de Llibres

Wellington Stephen Baéz Ramírez May Castells Raga
Cristina Malena Díaz Krishna Ocaña Zeballos
Ainhoa Pérez García Anna Marín Nuño

Abril 2025

Índex

Projecte d'Intercanvi de Llibres

1. Descripció del Projecte

Aquest projecte implementa una aplicació web utilitzant Django que permet als usuaris intercanviar, vendre o donar llibres entre ells mitjançant un sistema de punts.

2. Model Relacional

El model de dades consisteix en les següents entitats i relacions:

Entitats Principals

Declarades en `models.py`: 1. **User** (Usuari) - Estén el model d'usuari de Django (AbstractUser) - Incorpora gestió d'autenticació nativa de Django - Camps addicionals:

- `points`: Punts disponibles per a transaccions - `location`: Ubicació de l'usuari - `joined_date`: Data de registre

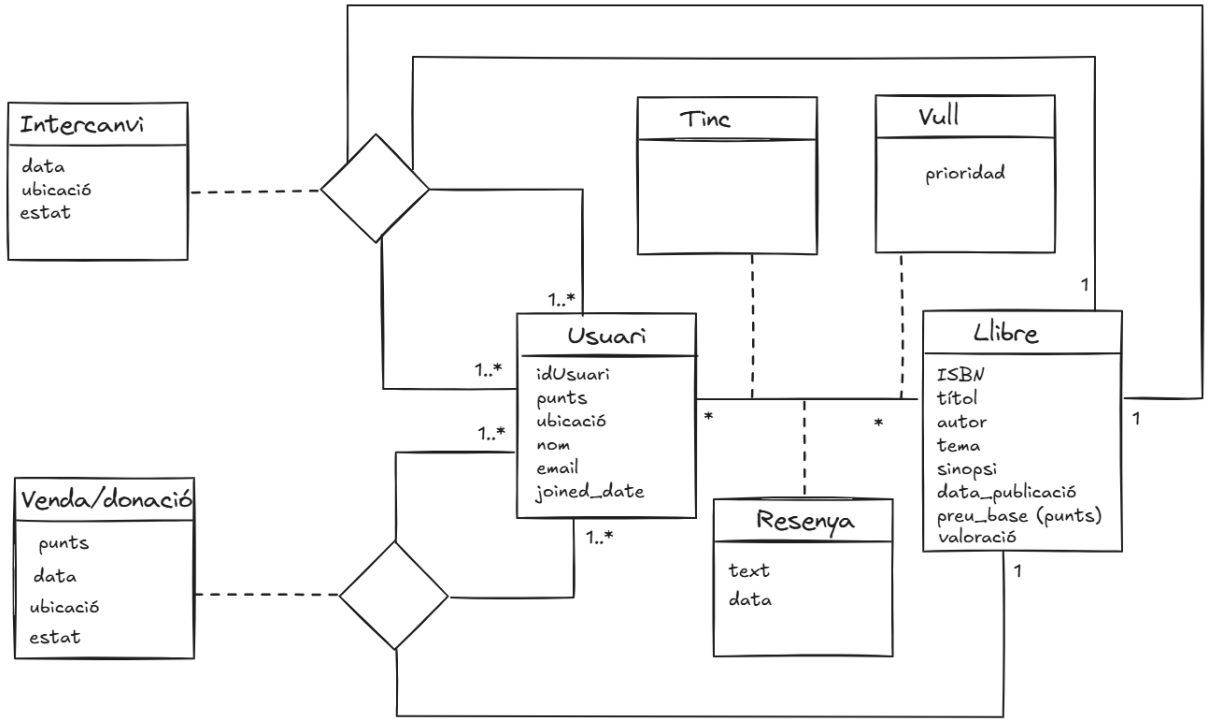
2. **Book** (Llibre)
   - Identificat per ISBN (clau primària)
   - Inclou informació bibliogràfica com títol, autor, tema
   - Preu base en punts
3. **Review** (Ressenya)
   - Permet als usuaris opinar sobre els llibres
   - Cada usuari només pot fer una ressenya per llibre
   - Té com a claus forana `user` (usuari que fa la ressenya) i `book` (llibre ressenyat)

Relacions entre Entitats

1. **Have** (Tenir)
   - Relació molts-a-molts entre User i Book
   - Indica quins llibres té cada usuari disponibles per intercanvi/venda
2. **Want** (Voler)
   - Relació molts-a-molts entre User i Book
   - Utilitza claus foranes per referenciar llibres i usuaris (`models.ForeignKey`)
   - Indica quins llibres desitja cada usuari
   - Inclou un camp de prioritat
   - Restricció: Un usuari no pot voler el mateix llibre més d'una vegada
3. **SaleDonation** (VendaDonació)
   - Registra la venda o donació d'un llibre d'un usuari
   - Inclou preu en punts, ubicació i estat de la transacció
4. **Exchange** (Intercanvi)
   - Registra intercanvis de llibres entre dos usuaris
   - Inclou llibres intercanviats, ubicació i estat de la transacció

Respecte al diagrama original s'ha respectat. ### Diagrama de Relacions

```
User 1---* Review *---1 Book
User 1---* Have *---1 Book
User 1---* Want *---1 Book
User 1---* SaleDonation *---1 Book
User(User1) 1---* Exchange *---1 User(User2)
Exchange *---1 Book(Book1)
Exchange *---1 Book(Book2)
```



Les relacions estan dissenyades per poder-se referenciar mitjançant claus foranes, i inclouen camps `related_name` per facilitar les consultes en ambdues direccions.

> Tot i que encara no s'ha implementat, s'ha decidit que només es guardarà a la base de dades pròpia de l'aplicació els llibres que els usuaris hagin intercanviat, venut, donat, volgut o valorat. La resta de llibres i la seva informació es consultaran a través d'API externes.

3. Tauler d'Administració

S'ha creat i activat un tauler d'administració per gestionar les entitats del sistema. El tauler permet als administradors gestionar usuaris, llibres, ressenyes i transaccions, a més s'han implementat filtres i funcionalitats de cerca per facilitar la gestió de les dades.

1. **Activació del tauler**:
   - S'ha activat el mòdul d'administració incloent `django.contrib.admin` a `INSTALLED_APPS` al fitxer `settings.py`
   - S'ha registrat al fitxer `urls.py` principal amb la ruta `/admin/`
2. **Personalització dels models**:
   - S'ha creat un fitxer `admin.py` a l'aplicació web on es registren i configuren tots els models
   - Cada model disposa d'una classe Admin específica que hereta de `admin.ModelAdmin`

El tauler d'administració permet:

1. **Gestió d'Usuaris**:
   - Classe personalitzada `CustomUserAdmin` que estén `UserAdmin`
   - Visualització i edició dels camps estàndards i personalitzats (punts, ubicació, data d'inscripció)
   - Filtres

2. **Gestió de Llibres**:
   - Visualització i edició de tots els detalls bibliogràfics
   - Filtres per tema i data de publicació
   - Cerca per ISBN, títol, autor i tema
3. **Gestió de Relacions**:
   - Interfícies intuïtives per a les relacions Have, Want, Exchange i SaleDonation
   - Visualització de les valoracions (Reviews) amb filtres per puntuació
4. **Funcionalitats generals**:
   - Filtres per camps rellevants a cada model
   - Camps de cerca per facilitar la localització ràpida d'entitats
   - Organització jeràrquica per dates en els models que ho requereixen

Accés al Tauler d'Administració

L'administració està disponible a la URL `/admin/` i requereix credencials de superusuari. Es pot crear un superusuari mitjançant:

```
python manage.py createsuperuser
```

O dins del contenidor Docker:

```
docker-compose exec web python manage.py createsuperuser
```

Es podrà accedir al tauler d'administració amb les credencials del superusuari creat al navegador web a la URL `http://localhost:8000/admin/`.

4. Sistema d'Autenticació i Registre

S'ha implementat un sistema d'autenticació basat estenent el sistema d'usuari de Django i el seu sistema de formularis. El sistema permet als usuaris registrar-se, iniciar sessió i gestionar el seu perfil d'usuari. D'aquesta manera s'aprofita les funcionalitats de validació que els camps ja porten per defecte.

Formularis d'Autenticació

1. **Formulari de Registre (`CustomUserCreationForm`)**:
   - Estén el `UserCreationForm` natiu de Django
   - Afegeix camps addicionals: correu electrònic (obligatori) i ubicació (opcional)
   - Utilitza transaccions atòmiques per garantir la consistència de les dades
   - Gestiona automàticament la creació del perfil d'usuari personalitzat
2. **Formulari d'Inici de Sessió (`LoginForm`)**:
   - Formulari personalitzat per a la validació de credencials
   - Camp de nom d'usuari i contrasenya amb validació adequada

Procés

1. **Procés de Registre**:
   - Formulari amb validació completa de contrasenyes i unicitat d'usuaris
   - Assignació d'ubicació si està disponible
2. **Procés d'Inici de Sessió**:
   - Validació segura de credencials
   - Redirecció personalitzada després de l'autenticació

Utilitzar com base els formularis de Django per a la creació d'usuaris i autenticació ens ha permès aprofitar les funcionalitats de validació i no haver-les de implementar manualment.

5. Configuració de Docker

El projecte s'ha configurat per funcionar en un entorn containeritzat utilitzant Docker, de manera que es pot desplegar fàcilment en qualsevol màquina amb Docker instal · lat.

Estructura de contenidors

L'entorn Docker consta principalment d'un contenidor web que executa l'aplicació Django:

1. **Contenidor web**: Basat en Python, amb totes les dependències necessàries per executar l'aplicació Django
2. **Base de dades**: Actualment utilitzem SQLite (inclosa dins del contenidor web) > *Nota: De moment no s'ha implementat una base de dades externa com PostgreSQL, depenent de les necessitats futures del projecte ja es valorarà si és necessari fer-ho.*

Arxius de configuració

El projecte inclou els següents arxius de configuració de Docker:

- **Dockerfile**: Defineix la imatge base, instal · la dependències i configura l'entorn d'execució
- **docker-compose.yml**: Orquestra els serveis, defineix els volums per persistència de dades i configura les variables d'entorn

Volums i persistència

S'han configurat volums per garantir la persistència de les dades: - El codi font del projecte es munta com un volum al contenidor - La base de dades SQLite es manté persistent entre execucions

Execució del Projecte amb Docker

Per executar l'aplicació en un entorn local:

1. **Prerequisits**:
   - Docker i Docker Compose instal · lats al sistema
   - Git per clonar el repositori
2. **Clonar el repositori**:

```
git clone https://github.com/Krisoc123/ProjecteWeb.git
cd ProjecteWeb
```

3. **Construir i iniciar els contenidors**:

```
docker-compose build
docker-compose up
```

4. **Aplicar les migracions** (només al primer inici o després de canvis al model):

```
docker-compose exec web python manage.py migrate
```

5. **Crear un superusuari** (opcional):

```
docker-compose exec web python manage.py createsuperuser
```

Accés a l'aplicació

Un cop en funcionament, l'aplicació estarà disponible a:

- **Aplicació web**: http://localhost:8000
- **Interfície d'administració**: http://localhost:8000/admin

6. Compliment dels 12 Factors

El projecte intenta cumplir la guia dels 12 factors, a continuació se'n fa un resum:

1. **Base de codi**: Mantenim un repositori Git únic per tot el codi font del projecte.
2. **Dependències**: Totes les dependències estan explícitament declarades i aïllades mitjançant Poetry, permetent un control precís de les versions.
3. **Configuració**: Tot i que actualment no utilitzem variables d'entorn per a la configuració, el projecte està preparat per implementar-les en el futur quan sigui necessari.
4. **Serveis de suport**: La base de dades és tractada com un recurs extern vinculat, encara que actualment utilitzem SQLite.
5. **Construcció, publicació, execució**: El procés de desplegament està clarament separat en aquestes fases mitjançant Docker, tot i que de moment no necessitem escalar.
6. **Processos**: L'aplicació s'executa actualment com un únic procés, seguint el model de Django.
7. **Assignació de ports**: L'aplicació s'exposa a través d'un port específic definit al docker-compose.yml.
8. **Concurrència**: L'arquitectura està preparada per créixer, tot i que en l'estat actual no requereix múltiples processos.
9. **Descartabilitat**: Els contenidors Docker es poden iniciar i aturar ràpidament sense afectar la integritat del sistema.
10. **Paritat entre entorns**: Els entorns de desenvolupament i producció són el més similars possible gràcies a Docker.
11. **Logs**: Es tracten els logs com a fluxos d'esdeveniments, que es poden consultar mitjançant `docker-compose logs`.
12. **Processos d'administració**: Les tasques administratives s'executen com a processos únics, com demostra l'ús de `python manage.py`.