Universitatea Transilvania Brașov
Facultatea de Matematică și Informatică

DOCUMENTAȚIA PROIECTULUI

Record Store

STUDENȚI
Shaikh Muhammad Saud - 10LF333
Ureche Tudor - 10LF333

Cuprins

Link Github: https://github.com/saudz12/Record-Store-API-

Prezentarea Proiectului

Proiectul își propune să ofere o bază bună a unei aplicații backend de gestionare a unui magazin de articole muzicale. End goal-ul este să facem administrarea magazinului cât mai facilă pentru un posibil client. Am încercat să oferim cât mai multe endpoint-uri utile pentru a acoperi toate nevoile aplicației: interacțiune cu catalogul (stock-ul magazinului), gestionare a Artiștilor și Albumelor, o metodă pentru a permite unei persoane să se autentifice și să poată plasa comenzi.

Tehnologiile folosite

.NET 9.0 - A fost framework-ul principal utilizat pentru dezvoltarea aplicației, oferind suport nativ pentru crearea de API-uri moderne și performante.

SQL SERVER - Am ales SQL Server pentru gestionarea bazei de date, întrucât se integrează foarte bine cu ecosistemul .NET și oferă performanță și fiabilitate.

ASP.NET Core - Framework-ul web utilizat pentru dezvoltarea efectivă a API-ului. Ne-a permis definirea ușoară a endpoint-urilor și gestionarea metodelor HTTP (GET, POST, PUT, DELETE) necesare aplicației.

EF Core - pentru ORM - Am folosit acest ORM pentru a simplifica interacțiunea cu baza de date. Ne permite să lucrăm cu datele direct din cod, fără a scrie SQL manual, folosind concepte orientate pe obiecte.

JWT (JSON Web Token) - pentru autentificarea utilizatorilor. Token-ul are semnătură criptografică pentru securitate.

Swashbuckle.AspNetCore: Am integrat Swagger în aplicație pentru a documenta și testa ușor API-ul. Acesta oferă o interfață grafică intuitivă și interactivă pentru toate endpoint-urile, fiind foarte util în dezvoltare și testare.

Baza de Date

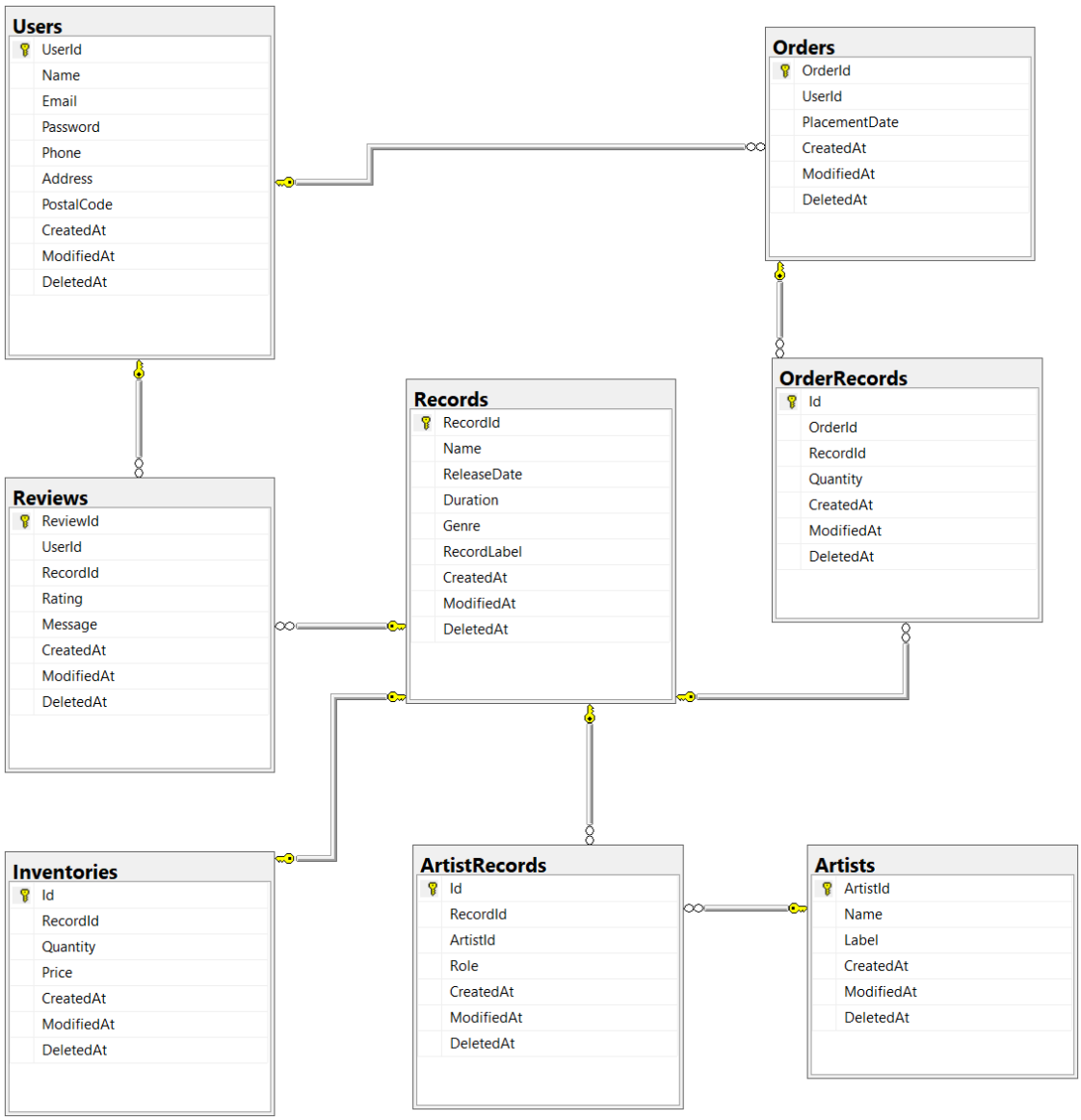
Pe parcursul proiectării bazei de date, ne-am asigurat că aceasta respectă forma normală a treia (3NF), pentru a elimina redundanțele și a facilita întreținerea datelor pe termen lung.

Având în vedere că scopul aplicației a fost construirea unui API pentru un magazin online de discuri, a fost necesar să definim atât structura inventarului, cât și entitățile legate de utilizatori și artiști.

Inventarul este reprezentat prin tabela Records, care conține informații despre fiecare disc (EP, LP, Single etc.), și tabela Inventories, care asociază fiecărui Record un preț și o cantitate disponibilă în stoc.

Tabela Artists este legată de Records printr-un context intermediar (ArtistRecords), întrucât un disc poate avea mai mulți artiști contribuitori, iar un artist poate contribui la mai multe discuri — relație de tip many-to-many.

Relația dintre artiști și discuri poate fi folosită drept criteriu de filtrare sau sortare în aplicație, pentru a permite utilizatorilor să navigheze ușor în catalogul muzical.



Pentru gestionarea utilizatorilor, am definit tabela Users, în care stocăm informațiile de autentificare și identificare ale fiecărui utilizator.

Fiind vorba de un magazin online, era esențial să includem posibilitatea ca un utilizator să poată plasa comenzi, astfel că am creat tabela Orders, care înregistrează comenzile efectuate.

Între tabelele Users și Orders există o relație de tip unu-la-mulți (1:N) — un utilizator poate plasa mai multe comenzi, dar fiecare comandă aparține unui singur utilizator. Din acest motiv, nu este necesară o tabelă de joncțiune, spre deosebire de relația dintre Artists și Records.

În schimb, între Orders și Records este necesară o tabelă de joncțiune, deoarece:

- O comandă poate conține mai multe discuri diferite (EP, LP, etc.).
- Un disc poate apărea în mai multe comenzi diferite.

Prin urmare, am definit o tabelă intermediară (OrderRecords, de exemplu) care gestionează această relație de tip many-to-many, împreună cu o coloană pentru cantitate, în cazul în care un utilizator dorește să comande mai multe exemplare din același disc.

API

Endpointuri

OrderRecords

- GET /api/order-records
- POST /api/order-records
- GET /api/order-records/{id}
- PATCH /api/order-records/{id}
- DELETE /api/order-records/{id}
- GET /api/order-records/order/{orderId}/record/{recordId}
- DELETE /api/order-records/order/{orderId}/record/{recordId}
- GET /api/order-records/order/{orderId}
- GET /api/order-records/record/{recordId}

Orders

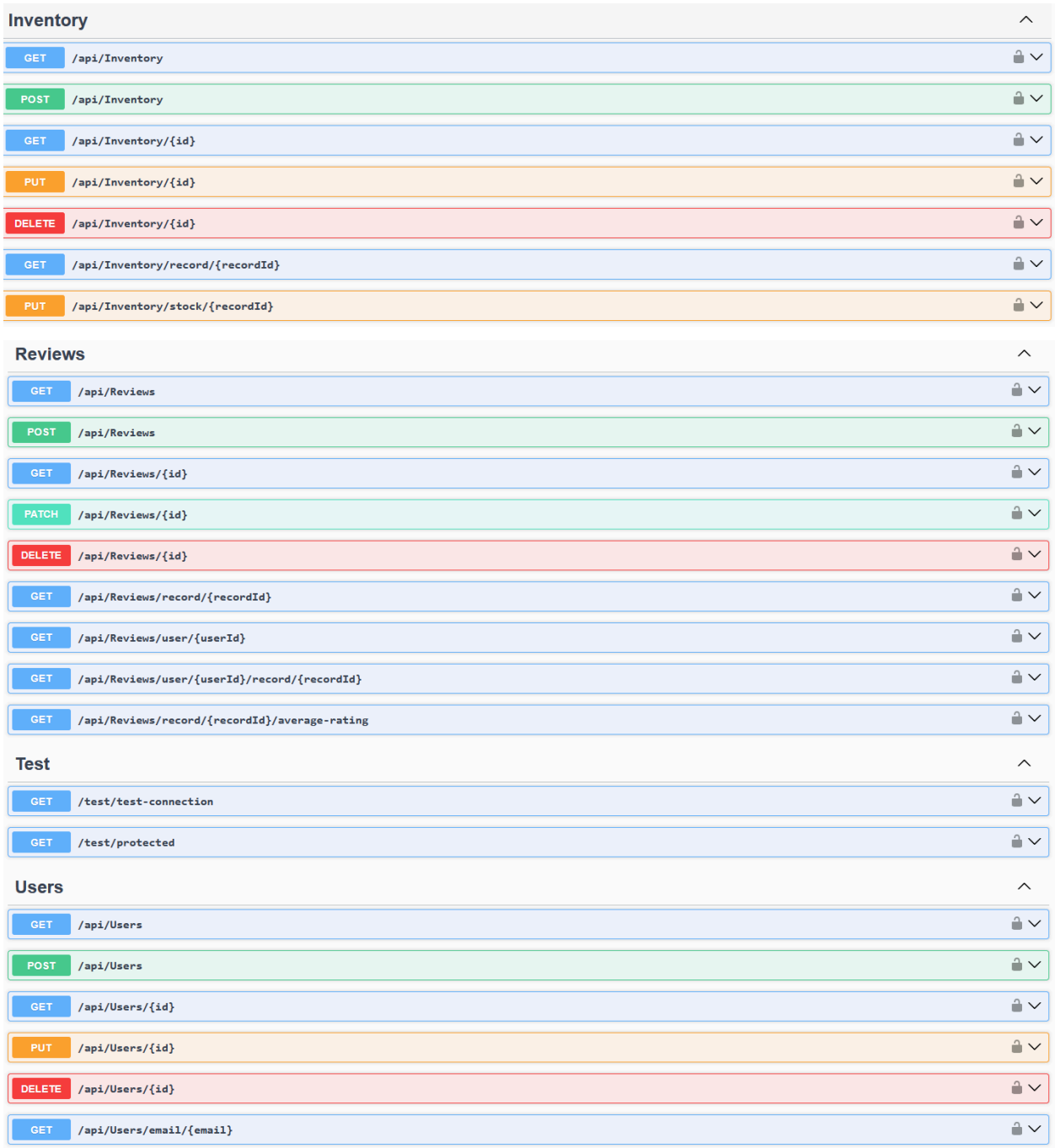
- GET /api/Orders
- POST /api/Orders
- GET /api/Orders/{id}
- DELETE /api/Orders/{id}
- GET /api/Orders/user/{userId}

Records

- GET /api/Records/query
- GET /api/Records
- POST /api/Records
- GET /api/Records/{id}
- PUT /api/Records/{id}
- DELETE /api/Records/{id}
- GET /api/Records/genre/{genre}
- GET /api/Records/search

Reviews

- GET /api/Reviews
- POST /api/Reviews
- GET /api/Reviews/{id}
- PATCH /api/Reviews/{id}
- DELETE /api/Reviews/{id}
- GET /api/Reviews/record/{recordId}
- GET /api/Reviews/user/{userId}
- GET /api/Reviews/user/{userId}/record/{recordId}
- GET /api/Reviews/record/{recordId}/average-rating



CRUD

În privința conceperii serviciilor pentru business layer-ul API-ului - servicii pentru fiecare entitate definită se folosesc de repositories pentru a crea un grad de separare între BL și DAL. API endpoints de Get, Post, Put, Patch și DELETE (**CRUD**) se folosesc de aceste servicii. Fiecărei tabelă îi sunt definite astfel de endpoint uri:

- Get: Dăm fetch la datele din tabelă după un criteriu - toate, după id, după nume etc. Pentru Records avem și posibilitatea de a filtra, sorta și pagina răspunsul.
- Post: adăugăm o intrare nouă în baza de date
- Put: editățm COMPLET o intrare în baza de date - atributele nemenționate vor fi resetate
- Patch: editățm PARȚIAL o intrare în baza de date - atributele nemenționate vor rămâne neschimbare
- Delete : ștergem o intrare din baza de date pe baza unui criteriu de selecție

! Pentru toate tipurile care alterează conținutul bazei de date am impus restricții.

Pentru **Artist**:

- Metode de extragere, adăugare, editare și ștergere în funcție de ID.

Pentru **Record**:

- Metode de extragere, adăugare, editare și ștergere în funcție de ID + câteva endpoint-uri speciale pentru extragere: unul pentru căutare în funcție de un cuvânt, gen muzical și unul cu filtrare, sortare și paginare.

Pentru **ArtistRecord**

- CRUD pe baza id-ului Artistului și/sau Albumului

Pentru Inventory

- CRUD pe baza id-ului din Inventar / id Record

Pentru **User**

- CRUD pe baza id-ului / email

Pentru **Order**

- Metode de extragere, adăugare și sterger pe id-ului Order-ului sau User-ului. Editare nu am considerat necesar întrucât lista de produse se definește în OrderRecords

Pentru **OrderRecords**

- CRUD pe baza id-ului din Orders și/sau Record / direct OrderRecords id

Pentru **Reviews**

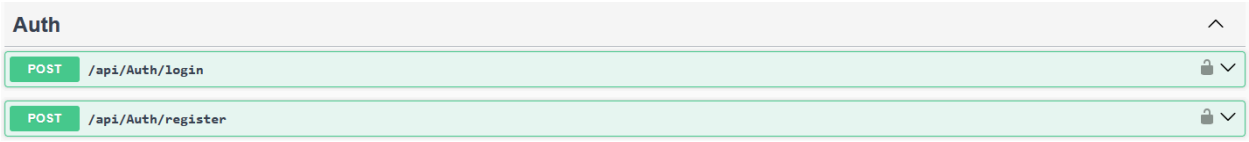
- CRUD pe baza id-ului din User și/sau Record. Patch (Update-ul) și Delete bazat pe id-ul tabelei Reviews pentru a evita editare multiplă.

*Endpoint uri speciale

- Pentru testare vizibilitate Utilizator - get protejat si neprotejat
- Pentru autentificare / logare

Autentificare

Autentificarea se folosește de un JWT cu semnătură pentru a crea o conexiune securizată. În cadrul aplicației este nevoie de autentificare pentru a face orice fel de modificare de intrări din baza de date.



Utilizări

Aplicația ideal este folosită pe post de backend pentru gestionarea unui magazin de discuri. Pentru rulare sunt necesare următoarele:

- Server pentru baze de date la care aplicația să se poată conecta
- Un environment pentru aplicația în sine (capabil să ruleze aplicații .NET)
- Opțional o aplicație front-end pentru interacțiunea cu userii

Pentru a se folosi endpoint-urile de Update, Delete și Create, un utilizator trebuie să fie autentificat în aplicație din motive de securitate (4xx la acces) . Toate endpoint-urile de get pot fi accesate și fără logare / autentificare.

Contribuții

Proiectul a fost împărțit în 3 arii mari de dezvoltare:

1) Proiectarea Bazei de Date și interacțiunea cu aceasta în cod:
   → concepere diagrama E-R (ambii)
   → trecerea în ORM (.NET EF Core) a tabelelor și definirea relațiilor aflate între acestea în DbContext (ambii)
   → Data Access Layer - definirea la crearea unor mecanisme de mapare, DTOs și repositories pentru interacționa cu baza de date direct din cod (Saud)
2) Design-ul aplicației și scrierea operațiilor CRUD.(Saud)
3) Autentificare (Tudor)
   → Conceperea întregii logici de autentificare.

Pentru fiecare sub-arie s-au făcut branch-uri pe github pentru a evita merge conflicts și pentru a stabili o anumită separare în dezvoltarea software-ului.