Universitatea Transilvania Brașov
Facultatea de Matematică și Informatică

DOCUMENTAȚIA PROIECTULUI

Record Store

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

Cuprins

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.

Tehnologiile folosite

.NET 9.0 - A fost framework-ul principal pentru dezvoltarea aplicației noastre, cu suport built-in pentru dezvoltare de API-uri.

SQL SERVER - Pentru gestionarea bazei de date am ales SQL Server întrucât platforma .NET oferă suport bun pentru acesta

ASP.NET Core - Am avut nevoie și de acest framework web pentru crearea API-ului. Cu ajutorul acestuia am creat endpoint-uri și metode http specifice aplicației noastre.

JWT - JSON Web Token pentru autentificarea userilor. Tokenul are semnătură pentru securitate.

Swashbuckle.AspNet.Core: Acest pachet ne-a permis să utilizăm Swaggerul în aplicația noastră. Swaggerul este un tool de mare folos, oferă o interfață intuitivă pentru testarea endpoint-urilor atunci cand aplicația rulează.

Baza de Date

Pe parcursul proiectării bazei de date ne-am asigurat ca aceasta să se afle în forma normală 3. Întrucât ne-am propus să construim API-ul unui magazin online de discuri a trebuit să ne definim inventarul și metode pentru utilizatori.

Inventarul constă din tabela Records care ține mine datele despre un anumit EP/LP/Single și tabela Inventories care îi asociază unui record dat preț și cantitatea în care se află acesta (available stock).

Tot legată de tabela prin context Records este și Artists, care reprezintă un anumit artist (contribuitor la un album dacă cream în tabela de joncțiune ArtistRecords o intrare cu ambele) - poate fi folosit drept criteriu de filtrare/sortare.

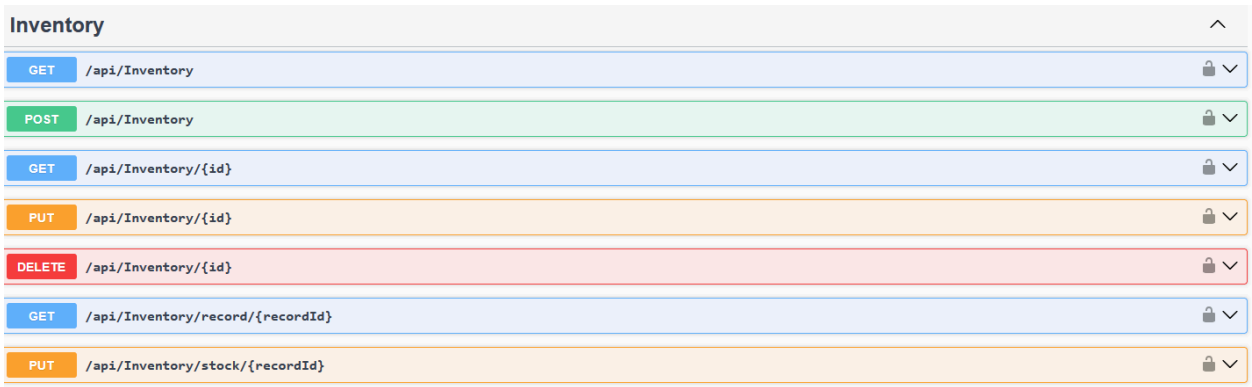
Pentru utilizatori ne definim tabela de Users în care îi stocăm și Orders în care înregistrăm toate comenzile acestora (fiind un magazin online trebuie să avem posibilitatea de a adăuga într o comandă itemi din catalog - inventar). Între acestea se stabilește o relație 1-N (un utilizator poate plasa mai multe comenzi, dar fiecare comandă este unic plasată de un singur utilizator), așadar nu suntem nevoiți de a crea

o tabela de joncțiune ca la Artiști și Records (un artist poate apărea pe mai multe albume cu roluri diferite, iar un album poate avea mai mulți artiști pe el).

Un alt loc în care întâlnim tabele de joncțiune este între Orders și Records: pe o comandă, putem avea mai multe albume diferite(reținem cantitatea în tabelă pentru cazul în care dorim să adăugăm mai multe copii) iar un album poate apărea pe mai multe comenzi.

API

Endpointuri



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

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

Test

- GET /test/test-connection
- GET /test/protected

Users

- GET /api/Users
- POST /api/Users
- GET /api/Users/{id}
- PUT /api/Users/{id}
- DELETE /api/Users/{id}
- GET /api/Users/email/{email}

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.

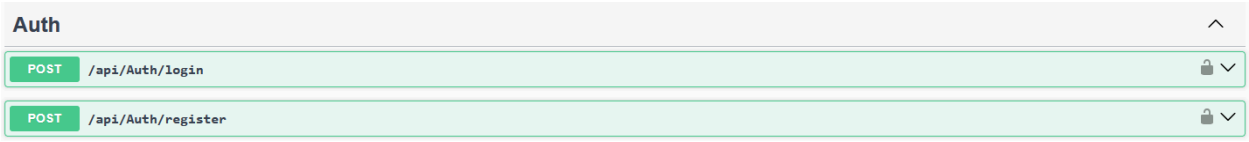
Fiecărei tabelă îi sunt definite astfel de endpoint uri:

- 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.

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

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.

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