W poniższym dokumencie chciałbym opisać przebieg pracy i podjęte decyzje na każdym etapie tworzenia aplikacji kina.

1. Kompromisy czasowe:
2. Nie realizuję obsługi sieci kin – skalowanie sprzedaży biletów dla każdego kina ma niewielką wartość dydaktyczną w porównaniu do kosztu czasowego zaimplementowania dodatkowych tabel baz danych dla kin oraz tabelki transferowej dla
3. Package:
4. Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Tools

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

<https://github.com/dotnet/aspnetcore/issues/35834>

<https://github.com/Microsoft/DockerTools/issues/99>

1. Functional requirements
2. Ticket reservations allowing the user to choose the movie, the time and the seats. Powinien być łatwy w obsłudze ~~i umożliwiać użytkownikom płatność online.~~
3. Repertoire schedule with movie info and screening time
4. Movie info: for each movie there will be TMDb API description, casting, director, score, reviews(?) and a trailer.
5. News and events: the Cinema should offer some miscellaneous events to be more attractive to wider audiences.
6. ~~Movie scores system i recenzji (also the ones they didnt buy tickets for – TMDb also offers movie scores and if this decision leads to poor movie scored people might still be interested to check out movies with the score discrepancy from TMDb’ score. Also the scores could be used later on for machine learning recommendations), that will help other users pick the movie.~~

Movie liking system – logged users should be able to give a like to a movie to show interest in watching it and they could browse their liked movies.

1. Newsletter system for logged users (not to overwhelm the new to the site) that lets the user keep track of what’s new in the Cinema and in the repertoire
2. Time constraint compromises:
3. Reviews dont bring much educational value to this project so they get skipped
4. Online payment system with Stripe could be very difficult to do on my setup considering the difficulties i had with security certificates dotnet dev-certs on my pc
5. Recommendation system that relies on users’ movie gradings and heavily on their ticket purchases would be fun with machine learning but i will skip it for now
6. Storing images would be cool to do on the cloud but learning AWS feels like a bit much rn.
7. App users:
8. People curious about what the Cinema offers
9. People interested in buying the tickets
10. An admin updating the schedule and the repertoire
11. UML usecase diagram:

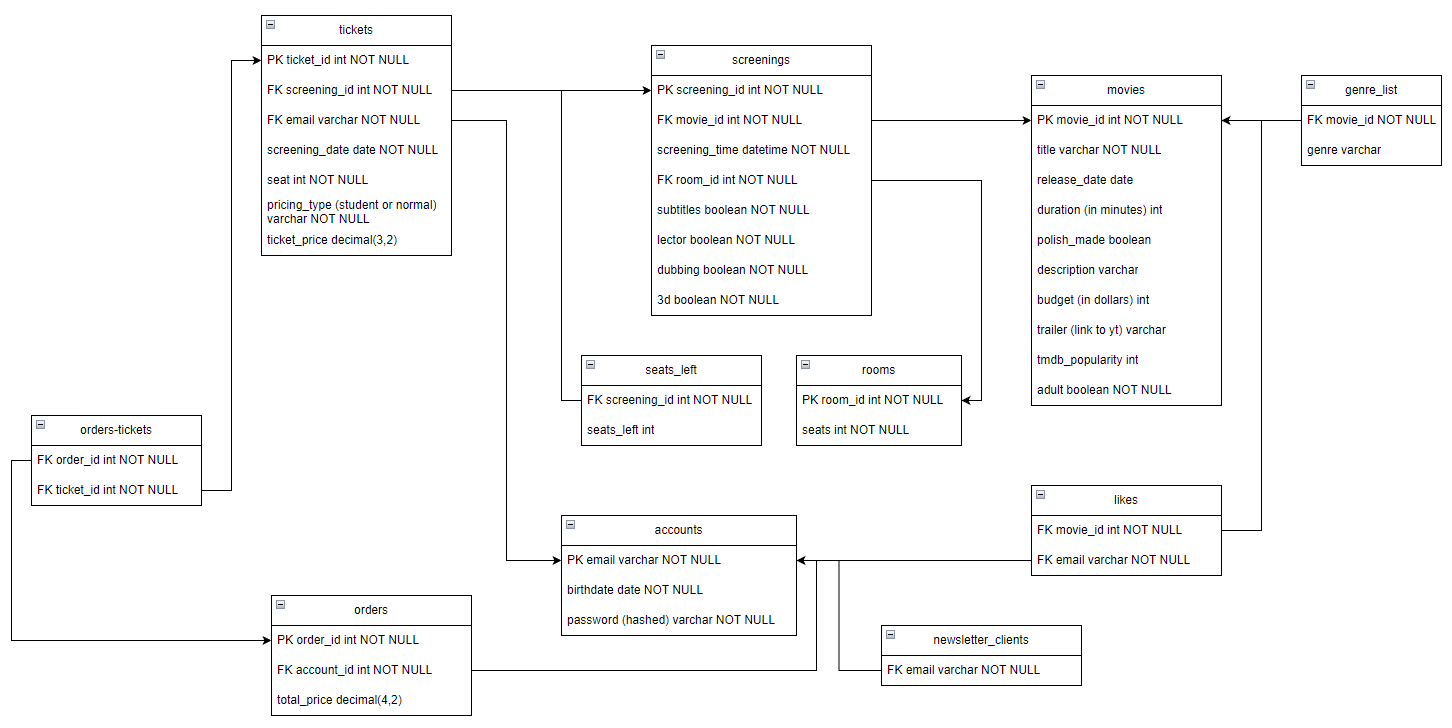
UML usecase diagram for the Cinema

Obraz zawierający diagram

Opis wygenerowany automatycznie

1. Database
2. Require date of birth to verify if the account can buy tickets for mature-audiences-only movies
3. The TMDb\_popularity of a movie could be used as one of the factors of displaying movies on the main page
4. Maximum price of one ticket is 999 (in case of inflation) and maximum order price is 9999 (maybe too small…)
5. Two ticket prices – student and normal

Early ERD diagram of the cinema



1. TMDb API

Data that could be used by my application:

https://api.themoviedb.org/3/movie/10331?api\_key=x

{

    "adult": **false**,

    "backdrop\_path": "/d0NwvSRJQQzkubWKsidX4caQ6Yi.jpg",

    "budget": 114000,

    "genres": [

        {

            "id": 27,

            "name": "Horror"

        }

    ],

    "id": 10331,

    "original\_language": "en",

    "original\_title": "Night of the Living Dead",

    "overview": "A disparate group of individuals takes refuge in an abandoned house when corpses begin to leave the graveyard in search of fresh human bodies to devour. The pragmatic Ben does his best to control the situation, but when the murderous zombies surround the house, the other survivors begin to panic.",

    "popularity": 22.087,

    "poster\_path": "/b6yJXwYAXgqJKNdOrEQxxbQ8oG4.jpg",

    "release\_date": "1968-10-04",

    "runtime": 96,

    "title": "Night of the Living Dead"

}

Trailer ids: https://api.themoviedb.org/3/movie/10331?api\_key=x

{

    "id": 10331,

    "results": [

        {

            "iso\_639\_1": "en",

            "iso\_3166\_1": "US",

            "name": "NIGHT OF THE LIVING DEAD Theatrical Trailer [1968]",

            "key": "DIuI6T48Sj0",

            "site": "YouTube",

            "size": 2160,

            "type": "Trailer",

            "official": **false**,

            "published\_at": "2022-05-10T19:46:20.000Z",

            "id": "62a4fbb68efe7300657134b7"

        },

1. WordPress

Obraz zawierający tekst

Opis wygenerowany automatycznie

Enabling CORS (because WordPress server and ASP .NET server are on two different ports):

[EnableCors("AllowWordPressApi")]

public class HomeController : Controller

var builder = WebApplication.CreateBuilder(args);

// Add CORS policy

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowWordPressApi",

policyBuilder =>

{

policyBuilder.WithOrigins("http://127.0.0.1:8080")

.AllowAnyHeader()

.AllowAnyMethod();

});

});

app.UseRouting();

// Enable CORS for the specified policy

app.UseCors("AllowWordPressApi");

app.UseAuthorization();

Retrieving posts from WordPress API in a controller:

public async Task<IActionResult> Index()

{

string apiEndpoint = "http://127.0.0.1:8080/wordpress/wp-json/wp/v2/posts";

HttpWebRequest request = (HttpWebRequest)WebRequest.Create(apiEndpoint);

request.Method = "GET";

request.ContentType = "application/json";

WebResponse response = await request.GetResponseAsync();

StreamReader reader = new StreamReader(response.GetResponseStream());

string jsonResponse = reader.ReadToEnd();

List<Post> posts = new List<Post>();

posts = JsonConvert.DeserializeObject<List<Post>>(jsonResponse);

1. Designing home page of the cinema