

Aplikacje internetowe

FilmCrawler

Jan Kowalski,
143936

Opis projektu

- ▶ Stworzenie serwisu do crawlowania informacji o filmach. Serwis będzie pobierał informacje z serwisu IMDB. Serwis składał się będzie z endpointu, za którego pomocą będzie można wpisać tytuł filmu, o którym informację chce się uzyskać. Informacjami o filmach pobieranymi z serwisu będą tytuł filmu, aktorzy, reżyser, opis itp. Wykonana aplikacja przeszukująca poszczególne serwisy będzie starała się wykorzystać gotową infrastrukturę przekazywania informacji o filmach i innych danych jakie będą udostępniały strony IMDB.

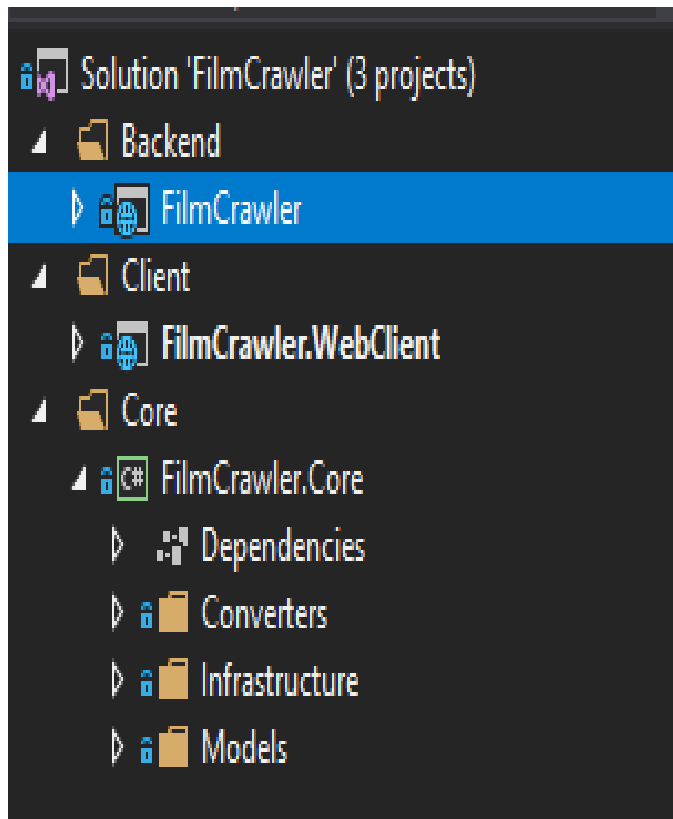
Technologie



IMDB a semantyczne sieci Web



Realizacja projektu



```
1 import requests
2 import time
3 import random
4
5 basePath="http://localhost:50309/api/ParseImdbMovie?fileName=tt0"
6
7 for i in range(410000,411000):
8     strIndex=str(i)
9     print("Current movie index "+strIndex)
10    r = requests.post(basePath+strIndex)
11    print(r.status_code, r.reason)
12    # sleepValue=random.randint(1, 3)
13    # print("SleepValue "+str(sleepValue))
14    # time.sleep(sleepValue)
15    # print("After SleepValue " + str(sleepValue))
16    print("Koniec")
```

Realizacja funkcji crawlowania konkretnego tytułu przez serwis FilmCrawler

```
using System.Threading.Tasks;
using FilmCrawler.Core.Infrastructure.CQRS.CommandBase.Interfaces;
using FilmCrawler.Core.Infrastructure.CQRS.QueryBase.Interfaces;
using FilmCrawler.Features.ParseImdbMovieCommand;
using Microsoft.AspNetCore.Mvc;

namespace FilmCrawler.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ParseImdbMovieController : BaseApiController
    {
        public ParseImdbMovieController(ICommandHandlerFactory commandHandlerFactory,
            IQueryHandlerFactory queryHandlerFactory) : base(commandHandlerFactory, queryHandlerFactory)
        {
        }

        // POST api/values
        [HttpPost]
        public async Task<IActionResult> Post(string fileName)
        {
            await _commandHandlerFactory.ResolveAndExecuteAsync(new ParseImdbMovieCommand(fileName));
            return Ok();
        }
    }
}
```

 namespace FilmCrawler.Features.ParseImdbMovieCommand

{

5 references | DESKTOP-M94H1SH\Jan, 38 days ago | 2 authors, 2 changes

public class ParseImdbMovieCommand:ICommand

{

3 references | DESKTOP-M94H1SH\Jan, 38 days ago | 1 author, 1 change | 0 exceptions

public string FilmId { get; set; }



1 reference | DESKTOP-M94H1SH\Jan, 38 days ago | 2 authors, 2 changes | 0 exceptions

public ParseImdbMovieCommand(string filename)

{

FilmId = filename;

}

}

}

```

using FluentValidation;

namespace FilmCrawler.Features.ParseImdbMovieCommand
{
    1 reference | DESKTOP-M94H1SH\Jan, 38 days ago | 2 authors, 2 changes
    public class ParseImdbMovieValidator: AbstractValidator<ParseImdbMovieCommand>
    {
        0 references | DESKTOP-M94H1SH\Jan, 38 days ago | 2 authors, 2 changes | 0 exceptions
        public ParseImdbMovieValidator()
        {
            RuleFor(c => c.FilmId).NotEmpty();
        }
    }
}

```


1 reference | DESKTOP-M94H1SH\Jan, 23 days ago | 2 authors, 7 changes

```
public class ParseImdbMovieAsyncCommandHandler:BaseCommandAsyncHandler<ParseImdbMovieCommand>
```

```
{
```

```
    private readonly FilmCrawlerDatabaseContext _dbContext;
```

0 references | DESKTOP-M94H1SH\Jan, 34 days ago | 1 author, 1 change | 0 exceptions

```
    public ParseImdbMovieAsyncCommandHandler(IValidatorFactory validatorFactory,  
        FilmCrawlerDatabaseContext dbContext) : base(validatorFactory)
```

```
    {
```

```
        _dbContext = dbContext;
```

```
    }
```

2 references | DESKTOP-M94H1SH\Jan, 23 days ago | 2 authors, 7 changes | 0 exceptions

```
protected override async Task RunCommandAsync(ParseImdbMovieCommand command)
```

```
{
    //INJECT HTTP CLIENT
    var basePath = "https://www.imdb.com/title/" + command.FilmId;
    using (var httpClient = new HttpClient())
    {
        var pageAsString = await httpClient.GetStringAsync(basePath);

        var doc = new HtmlDocument();
        doc.LoadHtml(pageAsString);

        var schemaScript = doc.DocumentNode.Descendants("script")
            .Where(n => n.Attributes.FirstOrDefault(a => a.Value == "application/ld+json") != null)
            ?.FirstOrDefault();

        if (schemaScript != null)
        {
            var movieModel = JsonConvert.DeserializeObject<ImdbMovieDto>(schemaScript.InnerText);

            var existedMovie = _dbContext.ImdbMovie.FirstOrDefault(p => p.Url == movieModel.Url);

            if (existedMovie != null)
                return;
        }
    }
}
```

```
var actorsDb = new List<Actor>();

if (movieModel.Actor!=null) ...

var genresDb = new List<Genre>();

if(movieModel.Genre!=null) ...

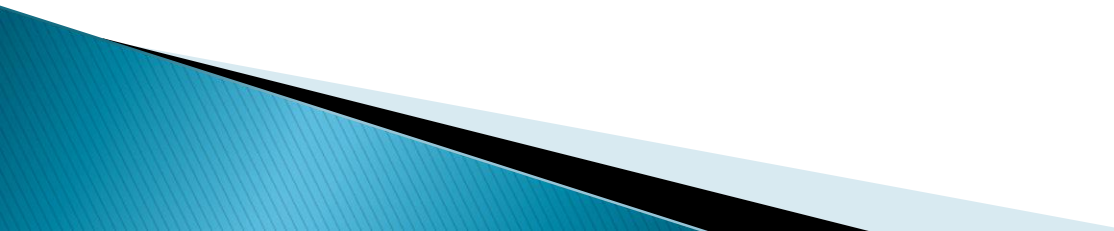
var directoresDb = new List<Director>();

if(movieModel.Director!=null)
{
    foreach (var director in movieModel.Director)
    {
        var existedDirector = _dbContext.Director.
            FirstOrDefault(a => a.Url == director.Url);
        if (existedDirector == null)
        {
            var currentDirector = new Director(director.Url, director.Name);
            _dbContext.Director.Add(currentDirector);
            directoresDb.Add(currentDirector);
        }
    }
    _dbContext.SaveChanges();
}
```

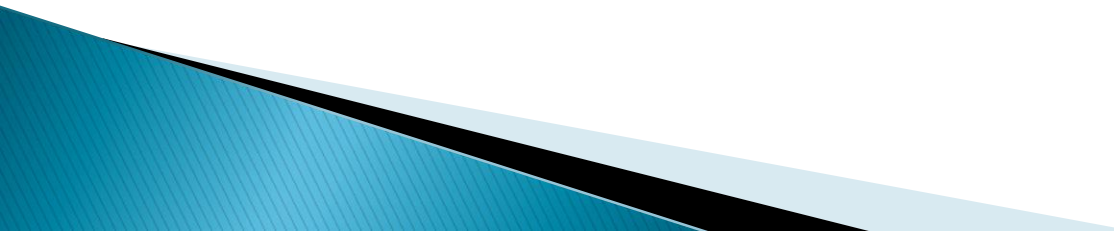
```
var creatorsDb = new List<Creator>();
if(movieModel.Creator!=null)
{
    foreach (var creator in movieModel.Creator)
    {
        var existedCreator = _dbContext.Creator.FirstOrDefault(a => a.Url == creator.Url);
        if (existedCreator == null)
        {
            var currentCreator = new Creator(creator.Url);
            _dbContext.Creator.Add(currentCreator);
            creatorsDb.Add(currentCreator);
        }
    }
    _dbContext.SaveChanges();
}

_dbContext.ImdbMovie.Add(ImdbMovie.CreateImdbMovie(movieModel,actorsDb,genresDb,
    directoroesDb,creatorsDb));
await _dbContext.SaveChangesAsync();
}
```

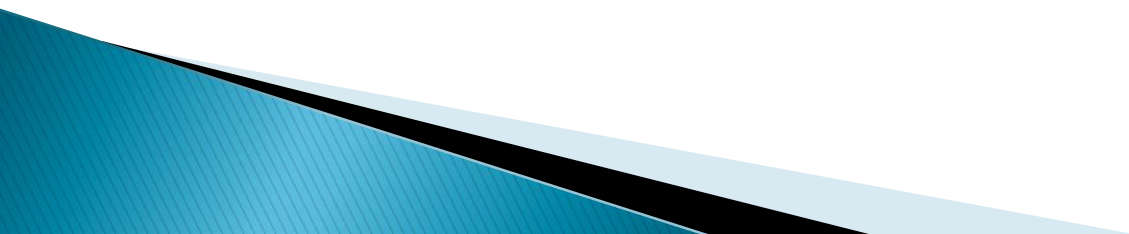
Zestawienie uzyskanych informacji

- ▶ Ponad 115 tys. filmów w ~24h
 - ▶ Ponad 112 tys. aktorów
 - ▶ Ponad 94 tys. twórców filmów
 - ▶ 28 kategorii filmów
- 

Problemy i błędy

- ▶ Słaba wydajność serwisu w przypadku większej ilości danych. Problem z bazą danych
 - ▶ Niezapisanie adresu zdjęć filmów. Błąd został wykryty po skończeniu projektu
 - ▶ Brak biblioteki w platformie .NET do parsowania formatu JSON-LD
- 

Pytania



Dziękuję za uwagę