ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Методи і засоби автогенерації та аналізу вихідного коду програмного забезпечення

Кваліфікаційна робота

Другий (магістерський) рівень

Автор:
Носов В.С.
студ. гр. СПм-20-2

Керівник:
Федорченко В.М.
доц. каф. ЕОМ

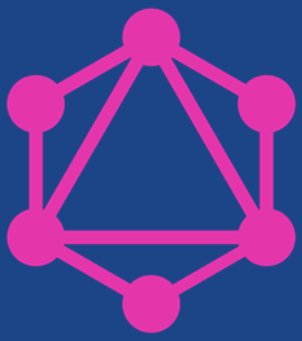
Мета і задачі роботи

Мета: аналіз та порівняння існуючих засобів для генерації вихідного коду, а також створення власного генератора вихідного коду.

Задачі:

- визначення об'єкта дослідження
- аналіз та порівняння існуючих інструментів для генерації коду
- вибір інструментів для розробки власного генератора вихідного коду
- проектування та розробка консольного додатку, який являє собою генератор вихідного коду

Визначення об'єкта дослідження



Сервіс Umbraco Heartcore GraphQL, створений для Headless CMS.

Визначення об'єкта дослідження

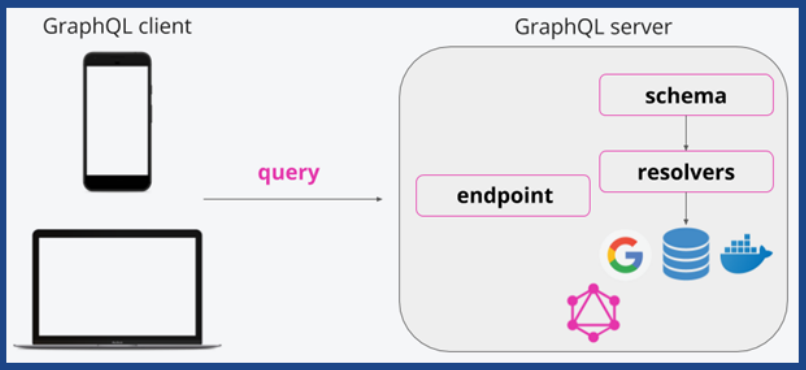
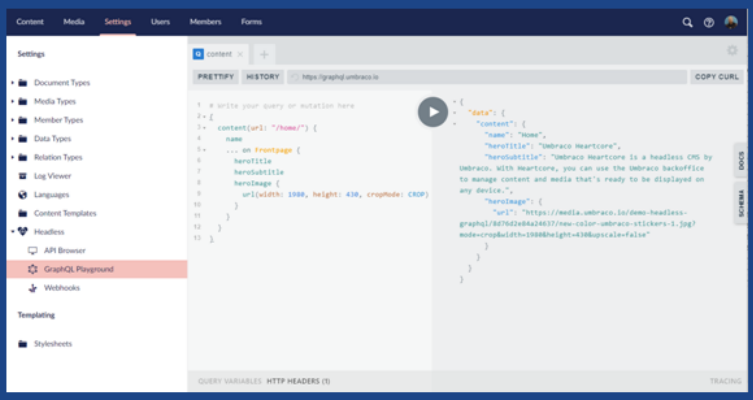


Схема взаємодії клієнтів і сервера GraphQL

Визначення об'єкта дослідження



Приклад запиту GraphQL, який отримує два текстові поля та зображення

Формування вимог до програмної системи

Під час розробки системи керування вмістом Umbraco з headless архітектурою виникає потреба в створення API для отримання контенту. Таким API може виступити, наприклад, GraphQL, який надасть можливість вибору полів на об'єктах, за якими потрібно зробити запит.

Отримання даних з системи керування вмістом, відбувається відповідно до схеми GraphQL.

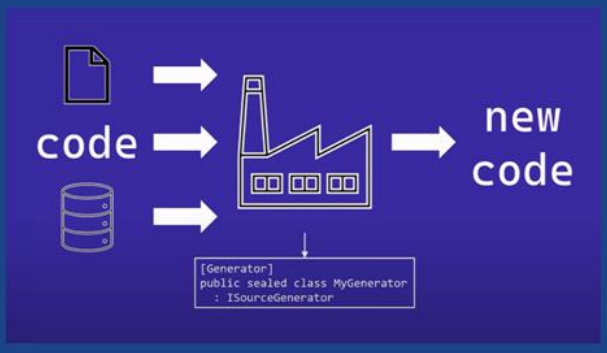
Формування вимог до програмної системи

Створення GraphQL API для Umbraco CMS залежить від структури сторінок в CMS. Джерелом такої інформації може виступити спеціальний пакет uSync. USync приймає інформацію з Umbraco CMS, яка зберігається в базі даних, і переміщує її на диск у вигляді XML файлів.

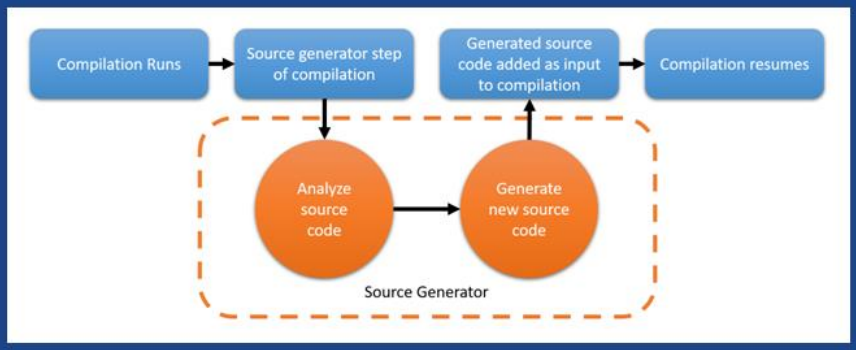
Незважаючи на те, що код схеми буде відрізнятися від класу до класу, він досить стереотипний за структурою, тому його краще генерувати автоматично, ніж писати вручну. Отже, основним завданням є створення власного генератора класів C#, який приймає XML файли у якості вхідного параметра і на основі них генерує класи мови C# – шаблонний код схеми GraphQL API.

Аналіз інструментів для кодогенерації

- CodeDOM
- Reflection
- IL weaving
- T4 templates
- Roslyn based source generators
- Інші засоби



Roslyn based source generators

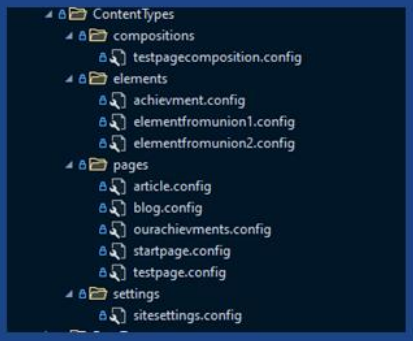


Основний інструмент розробки

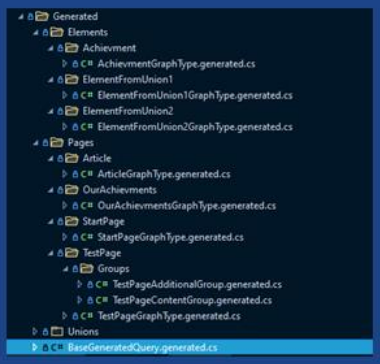
За результатами порівняння засобів генерації було обрано Roslyn, як один з найпопулярніших інструментів. Генератори коду Roslyn дозволяють легко створювати шаблонний код. Як компілятор, Roslyn канонічно знає все, що стосується синтаксису та семантики коду. Можна зібрати цілий клас, створивши такі частини, як імпорт простору імен, поля, властивості, конструктори, а потім створивши з цих частин єдиний блок компіляції. Це можна використовувати для створення шаблонних класів.

Такі генератори не мають проблем з продуктивністю, що надає можливість проведення швидкої збірки. Підтримка IntelliSense та можливість проводити налагодження з діагностикою створюють більш комфортні умови під час розробки.

Результати генерації



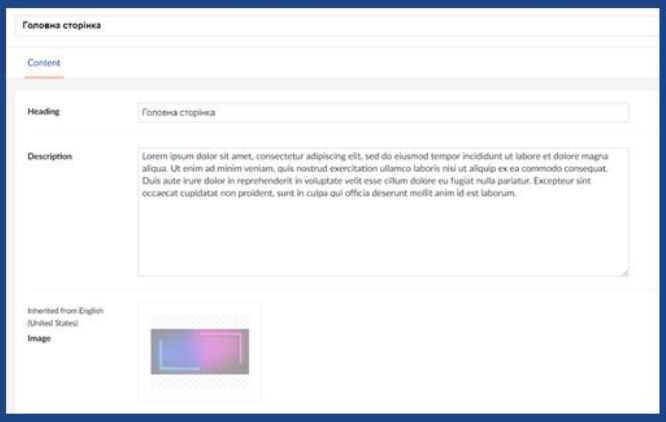
XML файли



Сгенеровані файли

Перевірка автоматично сгенерованого GraphQL API

Сторінка з контентом в CMS



Перевірка автоматично сгенерованого GraphQL API

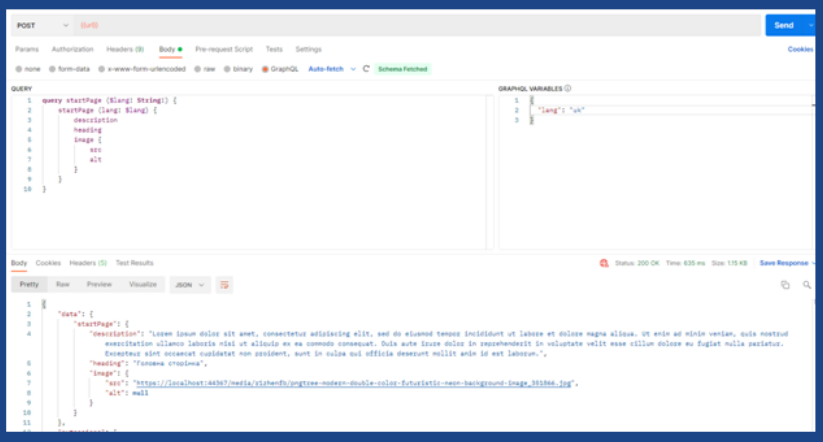
Автоматично сгенерований код

```
using GraphQL.Types;
using Umbraco.Extensions;
using CodeGenDyploma.GraphQL.Generated.Unions;
using CodeGenDyploma.GraphQL.Types.Properties;
using CodeGenDyploma.GraphQL.Types;
using Umbraco.Cms.Core.Models.PublishedContent;
using System.Collections.Generic;
using System.Linq;

namespace CodeGenDyploma.GraphQL.Generated.Pages.StartPage
{
    2 references
    public class StartPageGraphType : BaseContentGraphType
    {
        2 references
        public override string Alias => "startPage";
        0 references
        public StartPageGraphType()
        {
            Name = "startPage";
            Field<StringGraphType>(name: "description", resolve: context => context.Source?.Value(alias: "description"));
            Field<StringGraphType>(name: "heading", resolve: context => context.Source?.Value(alias: "heading"));
            Field<ImageGraphType>(name: "image", resolve: context => context.Source?.Value(alias: "image"));
        }
    }
}
```

Перевірка сгенерованого GraphQL API

Отримання контенту з CMS за допомогою Postman



Переваги та недоліки

Незважаючи на те, що код буде відрізнятися від класу до класу, він досить стереотипний за структурою, тому його краще генерувати автоматично, ніж писати вручну.

Створений генератор коду гарантує те, що майже вся наявна інформація в XML файлах матиме своє відображення в коді, тобто майже вся інформація з системи керування контентом буде відображена в програмному коді.

З іншого боку, створений генератор коду не має можливості генерувати аблюлютно всю реалізацію класів GraphQL, необхідних для створення API.

Висновки

В кваліфікаційній роботі було проведено аналіз перспектив застосування засобів генерації вихідного коду: CodeDOM, IL injecting, T4 templates, snippets, Roslyn based source generators.

За результатами проведеного аналізу та досліджень було створено спеціалізований програмний продукт, інструмент, що призначений для автоматичної генерації коду, а саме C# класів, які у спільному використанні з власноруч написаними базовими класами можуть використовуватися для дуже швидкого створення та оновлення схеми GraphQL API у веб-додатку, який являє собою систему керування вмістом Umbraco.

ДОДАТОК Б

Лістинг коду

Б.1 Проект GeneratorLogic

Б.1.1 Файл generatorSettings.json

```
{
  "pagesToIgnore": [
    "blog",
    "siteSettings"
  ],

  "propertiesToIgnore": [
    "umbracoRedirect",
    "umbracoInternalRedirectId",
    "umbracoUrlName",
    "umbracoUrlAlias"
  ],

  "typesToIgnore": [
    "Umbraco.UploadField",
    "Umbraco.Tags",
    "Umbraco.MemberPicker",
    "Umbraco.ListView",
    "Umbraco.Label",
    "Umbraco.ImageCropper",
    "Umbraco.ContentPicker",
    "Umbraco.BlockList"
  ],

  "simpleTypesDictionary": [
    {
      "key": "Umbraco.TextBox",
      "value": "StringGraphType"
    },
    {
      "key": "Umbraco.MultipleTextstring",
      "value": "ListGraphType<StringGraphType>"
    },
    {
      "key": "Umbraco.TrueFalse",
      "value": "BooleanGraphType"
    },
    {
      "key": "Umbraco.TextArea",
```

```
      "value": "StringGraphType"
    },
    {
      "key": "Umbraco.Integer",
      "value": "IntGraphType"
    },
    {
      "key": "Umbraco.DateTime",
      "value": "DateTimeGraphType"
    },
    {
      "key": "Umbraco.ColorPicker",
      "value": "StringGraphType"
    },
    {
      "key": "Umbraco.CheckBoxList",
      "value": "ListGraphType<StringGraphType>"
    },
    {
      "key": "Umbraco.RadioButtonList",
      "value": "StringGraphType"
    },
    {
      "key": "Umbraco.ColorPicker.EyeDropper",
      "value": "StringGraphType"
    }
  ],

  "reusablePages": [
    "article"
  ]
}
```

Б.1.2 Файл ConfigService.cs

```
using CodeGenDyploma.GeneratorLogic.Models;
using Newtonsoft.Json.Linq;

namespace CodeGenDyploma.GeneratorLogic.Services;

public class ConfigService
{
    private readonly  JObject _config;

    public ConfigService()
    {
        _config =
JObject.Parse(File.ReadAllText("generatorSettings.json"));
    }

    public IEnumerable<string> GetPagesToIgnore()
```

```
    {
        return _config["pagesToIgnore"].ToObject<string[]>();
    }

    public IEnumerable<string> GetReusablePages()
    {
        return _config["reusablePages"].ToObject<string[]>();
    }

    public IEnumerable<string> GetPropertiesToIgnore()
    {
        return
_config["propertiesToIgnore"].ToObject<string[]>();
    }

    public IEnumerable<string> GetTypesToIgnore()
    {
        return _config["typesToIgnore"].ToObject<string[]>();
    }

    public Dictionary<string, string>
GetSimpleTypesMappingDictionary()
    {
        IEnumerable<ConfigKeyValueModel> mappingList =

_config["simpleTypesDictionary"].ToObject<ConfigKeyValueModel[]>
();

        var dictionary = new Dictionary<string, string>();

        foreach (var pair in mappingList)
        {
            dictionary.Add(pair.Key,pair.Value);
        }

        return dictionary;
    }
}
```

Б.1.3 Файл GraphqlGenerator.cs

```
using CodeGenDyploma.GeneratorLogic.Services;

namespace CodeGenDyploma.GeneratorLogic;

public class GraphqlGenerator
{
    private readonly UnionsGeneratorService
_unionsGeneratorService;
    private readonly DocumentTypeGeneratorService
_documentTypeGeneratorService;
```

```
    private readonly BaseQueryGeneratorService
_baseQueryGeneratorService;
    private readonly FileService _fileService;

    public GraphqlGenerator(UnionsGeneratorService
unionsGeneratorService, DocumentTypeGeneratorService
documentTypeGeneratorService, FileService fileService,
BaseQueryGeneratorService baseQueryGeneratorService)
    {
        _unionsGeneratorService = unionsGeneratorService;
        _documentTypeGeneratorService =
documentTypeGeneratorService;
        _fileService = fileService;
        _baseQueryGeneratorService = baseQueryGeneratorService;
    }

    public void Generate()
    {
        _fileService.ClearGraphqlData();
        _unionsGeneratorService.Generate();
        _documentTypeGeneratorService.Generate();
        _baseQueryGeneratorService.Generate();
        Environment.Exit(0);
    }
}
```

Б.1.4 Файл DocumentTypeGeneratorService.cs

```
using CodeGenDyploma.GeneratorLogic.Extensions;
using CodeGenDyploma.GeneratorLogic.Models;
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;

namespace CodeGenDyploma.GeneratorLogic.Services
{
    public class DocumentTypeGeneratorService
    {
        private readonly FileService _fileService;
        private readonly GroupGeneratorService
_groupGeneratorService;
        private readonly ClassConstructorGeneratorService
_classConstructorGeneratorService;
        private readonly DictionaryService _dictionaryService;

        public DocumentTypeGeneratorService(FileService
fileService,
            ClassConstructorGeneratorService
classConstructorGeneratorService,
            GroupGeneratorService groupGeneratorService,
            DictionaryService dictionaryService)
```

```
        {
            _fileService = fileService;
            _classConstructorGeneratorService =
classConstructorGeneratorService;
            _groupGeneratorService = groupGeneratorService;
            _dictionaryService = dictionaryService;
        }

        public void Generate()
        {
            var elements =
_dictionaryService.GetElementsDocumentTypes().ToList();
            elements.ForEach(GenerateDocumentType);

            var pages =
_dictionaryService.GetPagesDocumentTypes().ToList();
            pages.ForEach(GenerateDocumentType);
        }

        private void
GenerateDocumentType(GeneratorDocumentTypeModel model)
        {
            string code;

            if (model.GroupsAliases.Count() == 1)
            {
                code = FormPageWithOneGroupClassCode(model,
model.IsElement);
            }
            else
            {
                code = FormPageClassCode(model);

                foreach (var group in model.GroupsAliases)
                {
                    _groupGeneratorService.Generate(group);
                }
            }

            var fileName =
$"{model.Name}{Constants.GraphqlClassNamePostfix}{Constants.Gene
ratedFileExtension}";
            _fileService.SaveGraphqlData(fileName,
$"{model.Folder}/{model.Name}", code);
        }

        private string
FormPageClassCode(GeneratorDocumentTypeModel model)
        {
            string inheritanceString = model.IsElement ?
Constants.ElementInheritance : Constants.ContentInheritance;
            var className =
$"{model.Name}{Constants.GraphqlClassNamePostfix}";
```

```
            var usings = Constants.TypesUsings;
            string namespacePath =
$"{model.Folder}.{model.Name}".Replace("/", ".");

            var @namespace =
SyntaxFactory.NamespaceDeclaration(SyntaxFactory.ParseName($"{Co
nstants.GeneratedDirecotryNamespacePart}{namespacePath}"));

            var classDeclaration =
SyntaxFactory.ClassDeclaration(className)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.AddBaseListTypes(SyntaxFactory.SimpleBaseType(SyntaxFactory.Par
seTypeName(inheritanceString)));

            var aliasDeclaration =
SyntaxFactory.VariableDeclaration(SyntaxFactory.ParseTypeName("s
tring"))

.AddVariables(SyntaxFactory.VariableDeclarator($"Alias =>
\"{model.Alias}\""));

            var aliasPropDeclaration =
SyntaxFactory.FieldDeclaration(aliasDeclaration)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.AddModifiers(SyntaxFactory.Token(SyntaxKind.OverrideKeyword));

            var ctorBodyElements =
FormCtorBodyElements(model.GroupsAliases.Select(_=>_.Alias),
model.Name);

            var classCtor =
SyntaxFactory.ConstructorDeclaration(className)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.WithBody(SyntaxFactory.Block(ctorBodyElements));

            classDeclaration =
classDeclaration.AddMembers(aliasPropDeclaration, classCtor);

            var code = usings +
@namespace.AddMembers(classDeclaration)
                                                .NormalizeWhitespace()
                                                .ToFullString();

            return code;
        }

        private string
```

```
FormPageWithOneGroupClassCode(GeneratorDocumentTypeModel model,
bool isElement)
        {
            string inheritanceString = isElement ?
Constants.ElementInheritance : Constants.ContentInheritance; ;
            var className =
$"{model.Name}{Constants.GraphqlClassNamePostfix}";
            string namespacePath =
$"{model.Folder}.{model.Name}".Replace("/", ".");

            var @namespace =
SyntaxFactory.NamespaceDeclaration(SyntaxFactory.ParseName($"{Co
nstants.GeneratedDirecotryNamespacePart}{namespacePath}"));

            var classDeclaration =
SyntaxFactory.ClassDeclaration(className)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.AddBaseListTypes(SyntaxFactory.SimpleBaseType(SyntaxFactory.Par
seTypeName(inheritanceString)));

            var aliasDeclaration =
SyntaxFactory.VariableDeclaration(SyntaxFactory.ParseTypeName("s
tring"))

.AddVariables(SyntaxFactory.VariableDeclarator($"Alias =>
\"{model.Alias}\""));

            var aliasPropDeclaration =
SyntaxFactory.FieldDeclaration(aliasDeclaration)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.AddModifiers(SyntaxFactory.Token(SyntaxKind.OverrideKeyword));

            var ctorBodyElements =
_classConstructorGeneratorService.GetConstructorBody(model.Group
sAliases.First().Properties, model.Alias);

            var classCtor =
SyntaxFactory.ConstructorDeclaration(className)

.AddModifiers(SyntaxFactory.Token(SyntaxKind.PublicKeyword))

.WithBody(SyntaxFactory.Block(ctorBodyElements));

            classDeclaration =
classDeclaration.AddMembers(aliasPropDeclaration, classCtor);

            var code = Constants.AllUsings +
model.GroupsAliases.First().Usings + "\r\n" +
@namespace.AddMembers(classDeclaration)
```

```
                                                .NormalizeWhitespace()
                                                .ToFullString();

            return code;
        }

        private IEnumerable<StatementSyntax>
FormCtorBodyElements(IEnumerable<string> aliases, string
pageName)
        {
            var ctorBodyElements = new List<StatementSyntax>();

            foreach (var alias in aliases)
            {
                string groupClassName =
$"{pageName}{alias.UpperInitial()}{Constants.GroupClassNamePostf
ix}";
                var ctorElement =
$"Field<{groupClassName}>(\"{pageName}{alias.UpperInitial()}\",
resolve: content => content.Source);";

ctorBodyElements.Add(SyntaxFactory.ParseStatement(ctorElement));
            }

            return ctorBodyElements;
        }
    }
}
```

Б.1.4 Файл FileService.cs

```
namespace CodeGenDyploma.GeneratorLogic.Services;

public class FileService
{
    public void SaveGraphqlData(string outputFileName, string
folder, string content)
    {
        string dataDirectory =
Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()
).Parent.Parent.Parent.FullName,
Constants.GraphqlProjectDirectory);
        string newDirectoryName = Path.Combine(dataDirectory,
Constants.GeneratedFolderName, folder);

        if (!Directory.Exists(newDirectoryName))
        {
            Directory.CreateDirectory(newDirectoryName);
        }

        string outputFilePath = Path.Combine(newDirectoryName,
```

```
outputFileName);
        File.WriteAllText(outputFilePath, content);
    }

    public void ClearGraphqlData()
    {
        string dataDirectory =
Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()
).Parent.Parent.Parent.FullName,
Constants.GeneratedFilesDirectory);

        if (Directory.Exists(dataDirectory))
        {
            Directory.Delete(dataDirectory, true);
        }
    }
}
```

Б.2 Проект GraphQL

Б.2.1 Файл StartPageGraphType.generated.cs

```
using GraphQL.Types;
using Umbraco.Extensions;
using CodeGenDyploma.GraphQL.Generated.Unions;
using CodeGenDyploma.GraphQL.Types.Properties;
using CodeGenDyploma.GraphQL.Types;
using Umbraco.Cms.Core.Models.PublishedContent;
using System.Collections.Generic;
using System.Linq;

namespace CodeGenDyploma.GraphQL.Generated.Pages.StartPage
{
    public class StartPageGraphType : BaseContentGraphType
    {
        public override string Alias => "startPage";
        public StartPageGraphType()
        {
            Name = "startPage";
            Field<StringGraphType>("description", resolve:
context => context.Source?.Value("description"));
            Field<StringGraphType>("heading", resolve: context
=> context.Source?.Value("heading"));
            Field<ImageGraphType>("image", resolve: context =>
context.Source?.Value("image"));
        }
    }
}
```