C# Call GraphQL API

What is GraphQL API?

- https://graphql.org
- A query language for your API

Schema Query Result

Why GraphQL?

- ❏ Provides a complete and understandable description of the data in your API
- ❏ Clients can ask for exactly what they need and nothing more
- ❏ Easy to evolve APIs over time
- ❏ Enables powerful developer tools

GraphQL over REST

- Single endpoint vs multiple endpoints
- Flexible structure in return data vs fixed
- Strongly typed vs weakly typed
- Invalid requests are automatically checked vs manually checked

How to consume a graphql API?

- Using Strawberry.Shake
  https://chillicream.com/docs/strawberryshake/v13/get-started/console

- Public GraphQL APIs for testing
  https://github.com/graphql-kit/graphql-apis

Step 01 - Install Strawberry Shake locally

- ❑ Create a project, let's say **WpfApp1**
- ❑ Open terminal with **Ctlr + `**
- ❑ **Go up to the parent directory of the solution**

- ❑ Install **manifest-tool**

  $ dotnet new tool-manifest
- ❑ Install the **Strawberry Shake** tools

  $ dotnet tool install StrawberryShake.Tools --local

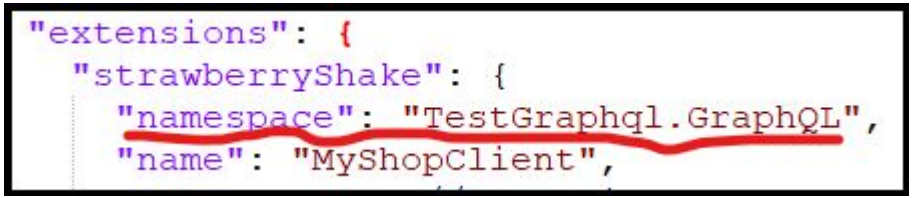
Step 02 - Install server

(Still in the parent directory of the solution)

- Add Strawberry package to our code generation
  - **dotnet add WpfApp1 package StrawberryShake.Server**

Step 03

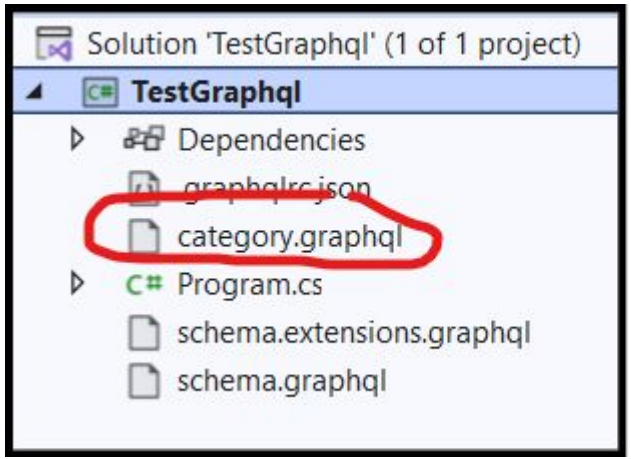
- Add a GraphQL client to your project using the CLI tools
- Let's say your url is: https://graphql.anilist.co
  - **dotnet graphql init https://graphql.anilist.co -n AnimeClient -p ./WpfApp1**
  - **dotnet graphql init localhost:4000/graphql -n MyShopClient -p ./TestGraphql**
- Customize the namespace of the generated client to be **WpfApp1.GraphQL**
  - Head over to the **.graphqlrc.json**
  - Insert a namespace property to the **StrawberryShake** section.



Step 04 - Write your queries (text file then rename)

Create a file **queries.graphql** to store your queries

Or: **category.graphql**



queries.graphql | MainWindow.xaml.cs

```
1  query GetCharacterById
2    Character(id: 1) {
3      name {
4        first
5        middle
6        last
7        full
8        native
9        userPreferred
10     }
11     image {
12       large
13       medium
14     }
15   }
16 }
```

Step 05 - Build the project (Ctrl + Shift + B)

- ❏ Inspect folder: **obj/Debug/net9.0/berry**
- ❏ You should found an auto-generated class named: **MyShopClient.Client.cs**

Step 06 - Fetch the data

```
async void SampleQuery()
{
    var serviceCollection = new ServiceCollection();
    var host = "https://graphql.anilist.co";
    serviceCollection
        .AddAnimeClient()
        .ConfigureHttpClient(client =>
            client.BaseAddress = new Uri(host));

    IServiceProvider services = serviceCollection.BuildServiceProvider();

    var client = services.GetRequiredService<IAnimeClient>();
    var result = await client.GetCharacterById.ExecuteAsync();

}
```

Ví dụ lấy AllCategories

```
// GetAll
var result = await client.AllCategories.ExecuteAsync();
foreach (var item in result.Data!.AllCategories!.Nodes) {
    Console.WriteLine($"{item!.Id} - {item.Name}");
}
```

Tham số hóa query

Bước 1: Cập nhật category.graphql, bổ sung

```
query CategoryById($id: Int!){
  categoryById(id: $id) {
    id
    name
  }
}
```

Bước 2: Biên dịch

- Chạy lại lệnh init
  - **dotnet graphql init localhost:4000/graphql -n MyShopClient -p ./TestGraphql**
- Biên dịch lại với Ctrl + Shift + B

Bước 3: Gọi API

```
// GetById
var result = await client.CategoryById.ExecuteAsync(id: 4);
var item = result.Data!.CategoryById;
Console.WriteLine($"{item!.Id} - {item.Name}");
```

Chú ý chỗ truyền đối số là tham số của hàm ExecuteAsync.

DeleteById

Bước 1: Cập nhật query

Bước 2: Biên dịch lại

- Chạy lại lệnh init
- Biên dịch lại với Ctrl + Shift + B

Bước 3: Mã nguồn và xử lí lỗi

```
// deleteByid
int id = 3;
var result = await client.DeleteCategoryById.ExecuteAsync(id: id);
if (result.Errors.Count > 0) {
    Console.WriteLine(
        $"Cannot delete category with id: {id}. Reason: {result.Errors[0].Message}");
} else {
    Console.WriteLine($"Delete successfully category with id={id}");
}
```

UpdateById

Bước 1: Bổ sung câu query

```
mutation UpdateCategoryById(
  $id: Int!,
  $patch: CategoryPatch!
){
  updateCategoryById(input:{
    id: $id
    categoryPatch: $patch
  }) {
    category {
      id
      name
    }
  }
}
```

QUERY VARIABLES
REQUEST HEADERS

```
{
  "id": 6,
  "patch": {
    "name": "Ultra New Name"
  }
}
```

```
{
  "data": {
    "updateCategoryById": {
      "category": {
        "id": 6,
        "name": "Ultra New Name"
      }
    }
  }
}
```

Bước 2: Mã nguồn

```
int id = 7;
var patch = new CategoryPatch() {
    Name = "New Category Name"
};

var result = await client.UpdateCategoryById.ExecuteAsync(id, patch);
if (result.Errors.Count > 0) {
    Console.WriteLine($"{result.Errors[0].Message}");
} else {
    Console.WriteLine($"Successfully update category id={id}");
}
```

Create

Bước 1: Cập nhật query

Bước 2: Mã nguồn

```
var input = new CreateCategoryInput() {
    Category = new CategoryInput() {
        Name = "Xiaomi"
    }
};

var result = await client.CreateCategory.ExecuteAsync(input);
Console.WriteLine(
    $"""
    Successfully create a new category.
    Newly created id: {result.Data.CreateCategory.Category.Id}
"""");
```