

# Современные платформы прикладной разработки

---

GRAPHQL



# Введение в GraphQL

---

## ОБЩАЯ ИНФОРМАЦИЯ

# Общая информация

GraphQL — это язык запросов для API.

Это также среда выполнения для выполнения запросов к данным.

# Общая информация

GraphQL был разработан внутри компании Facebook в 2012 году, а затем публично выпущен в 2015 году.

# Общая информация



Подход, используемый в GraphQL, радикально отличается от REST.

Вместо нескольких конечных точек, которые возвращают фиксированные структуры данных, API-интерфейсы GraphQL обычно предоставляют только одну конечную точку.

# Общая информация

Структура возвращаемых данных не является фиксированной. Вместо этого он полностью гибок и позволяет клиенту решать, какие данные ему действительно нужны.

# GraphQL vs REST

	 GraphQL	 REST
Architecture	client-driven	server-driven
Organized in terms of	schema & type system	endpoints
Operations	Query Mutation Subscription	Create, Read, Update, Delete
Data fetching	specific data with a single API call	fixed data with multiple API calls
Community	growing	large
Performance	fast	multiple network calls take up more time
Development speed	rapid	slower
Learning curve	difficult	moderate
Self-documenting	✓	—
File uploading	—	✓
Web caching	(via libraries built on top)	✓
Stability	less error prone, automatic validation and type checking	better choice for complex queries
Use cases	multiple microservices, mobile apps	simple apps, resource-driven apps

<https://www.mobilelive.ca/blog/graphql-vs-rest-what-you-didnt-know>

(06.2022)

# Выбор GraphQL

- ☐ У вас ограниченная пропускная способность и вы хотите свести к минимуму количество запросов и ответов
- ☐ У вас есть несколько источников данных и вы хотите объединить их в одном адресе
- ☐ У вас есть запросы клиентов, которые значительно различаются, и вы ожидаете совершенно разные ответы

<https://aws.amazon.com/ru/compare/the-difference-between-graphql-and-rest/>



# Выбор REST

- ☐ У вас есть небольшие приложения с менее сложными данными
- ☐ У вас есть данные и операции, которые все клиенты используют одинаково
- ☐ У вас нет требований к сложным запросам данных

<https://aws.amazon.com/ru/compare/the-difference-between-graphql-and-rest/>

# Введение в GraphQL

---

## ОСНОВНЫЕ ПОНЯТИЯ

# Основные понятия (query)

---

Для получения данных с сервера клиент отправляет на сервер информацию о том, какие данные нужны — эта информация называется запросом (**query**).

# Основные понятия (query)

```
public class Student
{
    public int Id { get; set; }
    public string FullName { get; set; }
    public DateTime BirthDay { get; set; }

    public Group Group { get; set; }
    public int GroupId { get; set; }
}
```

# Основные понятия (query)

---

```
query{  
  students{  
    fullName  
  }  
}
```

# Основные понятия (query)

```
{
  "data": {
    "students": [
      {
        "fullName": "Odio Urna"
      },
      {
        "fullName": "Ligula Congue"
      },
      . . .
    ]
  }
}
```

# Основные понятия (query)

---

```
query{  
  students{  
    fullName  
    birthday  
  }  
}
```

# Основные понятия (query)

```
{
  "data": {
    "students": [
      {
        "fullName": "Odio Urna",
        "birthDay": "2022-03-12T13:10:42.356+03:00"
      },
      {
        "fullName": "Ligula Congue",
        "birthDay": "2022-03-14T13:10:42.356+03:00"
      },
      . . .
    ]
  }
}
```



# Основные понятия (query)

```
query{  
  students(where:{groupId:{eq:1}}){  
    fullName  
    birthDay  
  }  
}
```

# Основные понятия (mutation)

Изменения данных в GraphQL производятся с помощью так называемых ***мутаций***. Обычно выделяют три вида мутаций:

- ☐ создание новых данных
- ☐ обновление существующих данных
- ☐ удаление существующих данных

# Основные понятия (mutation)

```
mutation{  
  addStudent(input:{  
    fullName:"Goga"  
  })  
  {  
    student{  
      id  
      fullName  
      birthDay  
    }  
  }  
}
```

# Основные понятия (mutation)

```
mutation{  
  editStudent(  
    input:{  
      id : 1,  
      fullName : "qwerty uiop"})  
  {student{  
    id,  
    fullName  
  }}  
}
```

# GraphQL реализация

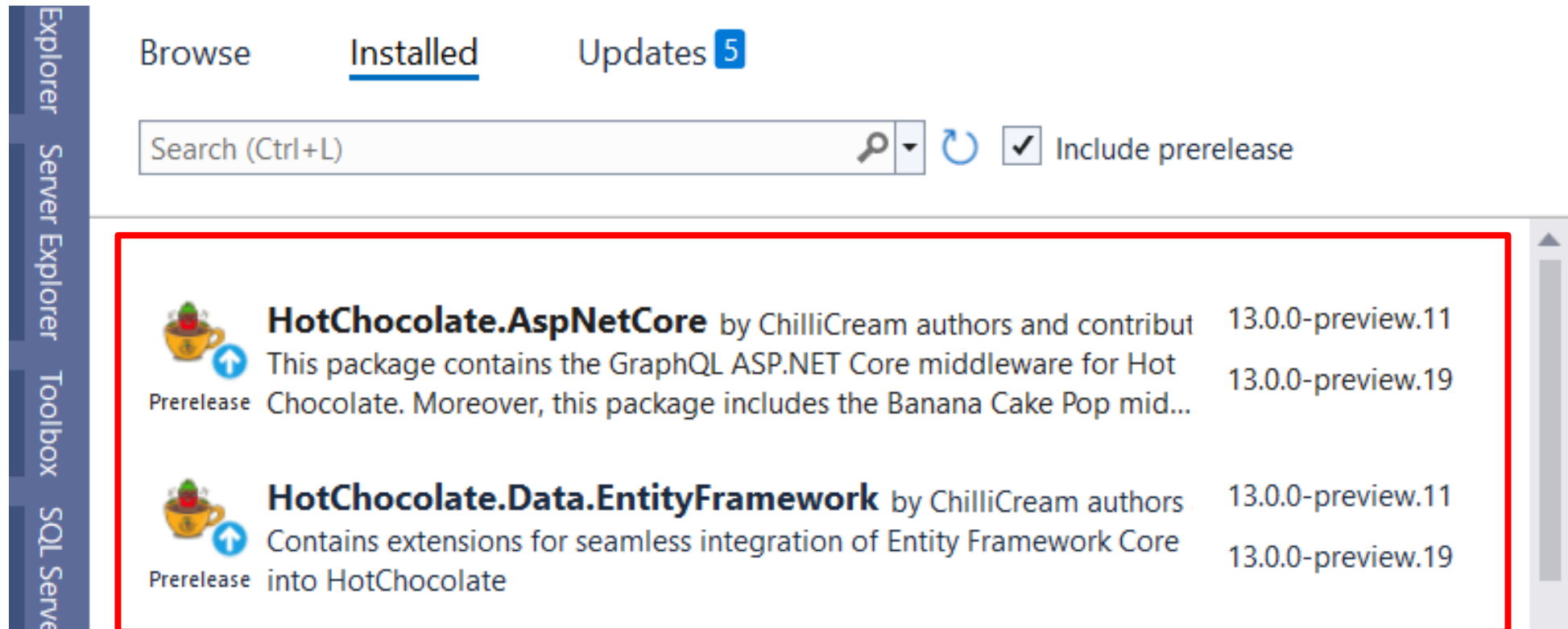
---

CHILLICREAM HOT CHOCOLATE

---

<https://chillicream.com/docs/hotchocolate/v13>

# ChilliCream Hot Chocolate



The screenshot shows the Visual Studio Package Manager interface. The 'Installed' tab is selected, and the 'Updates' count is 5. The search bar is empty. The 'Include prerelease' checkbox is checked. Two packages are listed, both marked as 'Prerelease' with a blue upward arrow icon:

- HotChocolate.AspNetCore** by ChilliCream authors and contributors. Description: This package contains the GraphQL ASP.NET Core middleware for Hot Chocolate. Moreover, this package includes the Banana Cake Pop mid...  
Versions: 13.0.0-preview.11, 13.0.0-preview.19
- HotChocolate.Data.EntityFramework** by ChilliCream authors. Description: Contains extensions for seamless integration of Entity Framework Core into HotChocolate  
Versions: 13.0.0-preview.11, 13.0.0-preview.19

# ChilliCream Hot Chocolate





# ChilliCream Hot Chocolate

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
    {
    }
    public DbSet<Dish> Dishes { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

# ChilliCream Hot Chocolate

Тип **query** в GraphQL представляет доступное только для чтения представление всех сущностей и способы их извлечения. Тип **query** требуется для каждого сервера GraphQL.

# ChilliCream Hot Chocolate (query)

```
public class Query
{
    private readonly AppDbContext _context;

    public Query(IDbContextFactory<AppDbContext> factory)
    {
        _context = factory.CreateDbContext();
    }

    public IQueryable<Dish> Dishes()
        => _context.Dishes;

    public IQueryable<Category> Categories()
        => _context.Categories;
}
```

# ChilliCream Hot Chocolate (сервис и middleware)

```
builder.Services.AddPooledDbContextFactory<AppDbContext>(opt =>
{
    opt.UseSqlite(connStr);
});

. . .

builder.Services.AddGraphQLServer()
    .AddQueryType<Query>();

. . .

app.MapGraphQL();
```

# ChilliCream Hot Chocolate

The screenshot displays the GraphQL Playground interface. The top bar shows the request method as **POST** to the URL **https://localhost**, with a status of **200 OK**, a response time of **1.21 s**, and a response size of **391 B**. The left sidebar contains a home icon and a list of requests, with **GetDishesList** selected. The main area shows the query, variables, and the JSON response.

**Query:**

```
1 query{
2   dishes{
3     name
4   }
5 }
```

**Response:**

```
1 {
2   "data": {
3     "dishes": [
4       {
5         "name": "Суп-харчо"
6       },
7       {
8         "name": "Борщ"
9       },
10      {
11        "name": "Котлета пожарская"
12      },
13    ]
14  }
15 }
```

# ChilliCream Hot Chocolate

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (112ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT "d"."Id", "d"."Calories", "d"."CategoryId", "d"."Description", "d"."Image", "d"."Name"
      FROM "Dishes" AS "d"
```

# ChilliCream Hot Chocolate

```
builder.Services.AddGraphQLServer()  
    .AddQueryType<Query>()  
    .AddProjections();
```

```
[UseProjection]  
public IQueryable<Student> GetStudents()  
    => _context.Students;
```

# ChilliCream Hot Chocolate

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]  
      Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']  
      SELECT "d"."Name"  
      FROM "Dishes" AS "d"
```



# ChilliCream Hot Chocolate

```
builder.Services.AddGraphQLServer()  
    .AddQueryType<Query>()  
    .AddProjections()  
    .AddFiltering()  
    .AddSorting();
```

```
[UseProjection]  
[UseFiltering]  
[UseSorting]
```

```
public IQueryable<Dish> Dishes()  
    => _context.Dishes;
```

# ChilliCream Hot Chocolate

The screenshot displays the GraphQL Playground interface. At the top, a POST request is sent to `tps://localhost:7022/graphql/`, resulting in a `200 OK` status, `138 ms` response time, and `159 B` of data. The request body is a GraphQL query to fetch department information for a specific ID.

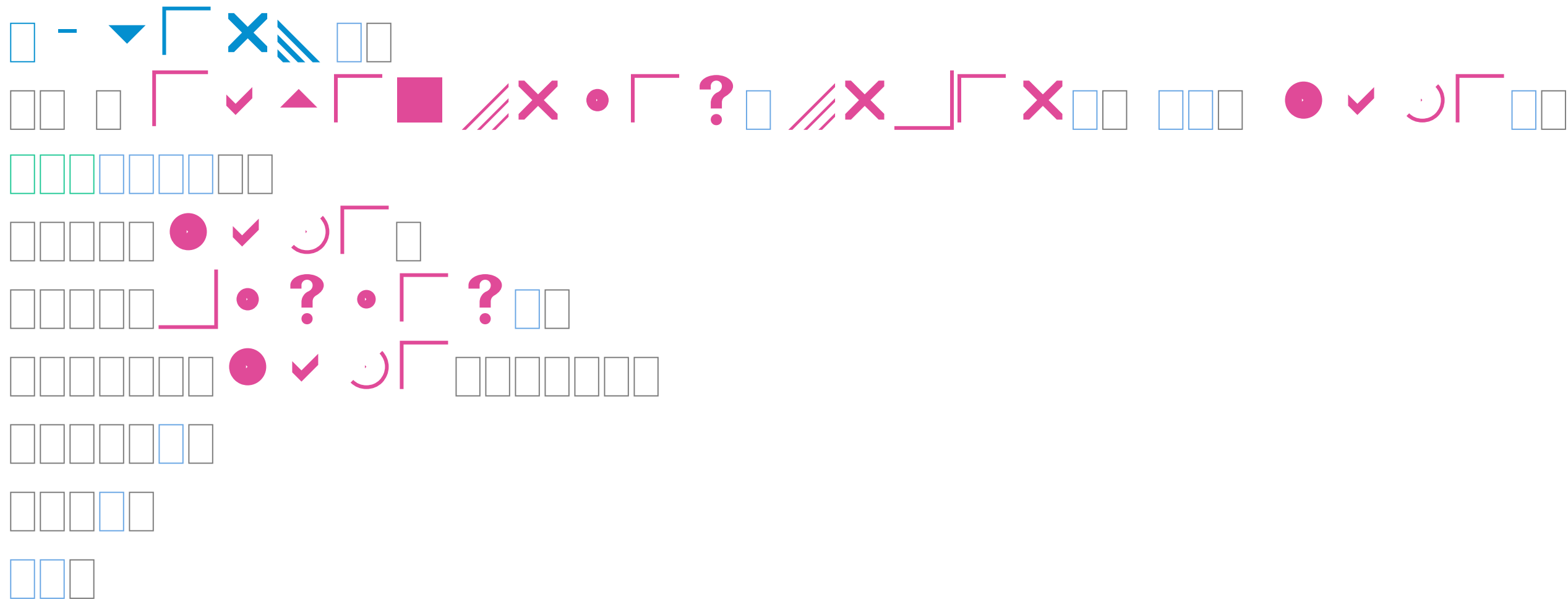
**GraphQL Query:**

```
1 query{
2   departments( where: {id:{eq:1}}){
3     name
4     groups{
5       number
6     }
7   }
8 }
```

**JSON Response:**

```
1 {
2   "data": {
3     "departments": [
4       {
5         "name": "Факультет 0",
6         "groups": [
7           {
8             "number": 691655
9           },
10          {
11            "number": 932800
12          },
13          {
14            "number": 719911
15          }
16        ]
17      }
18    ]
19  }
20 }
```

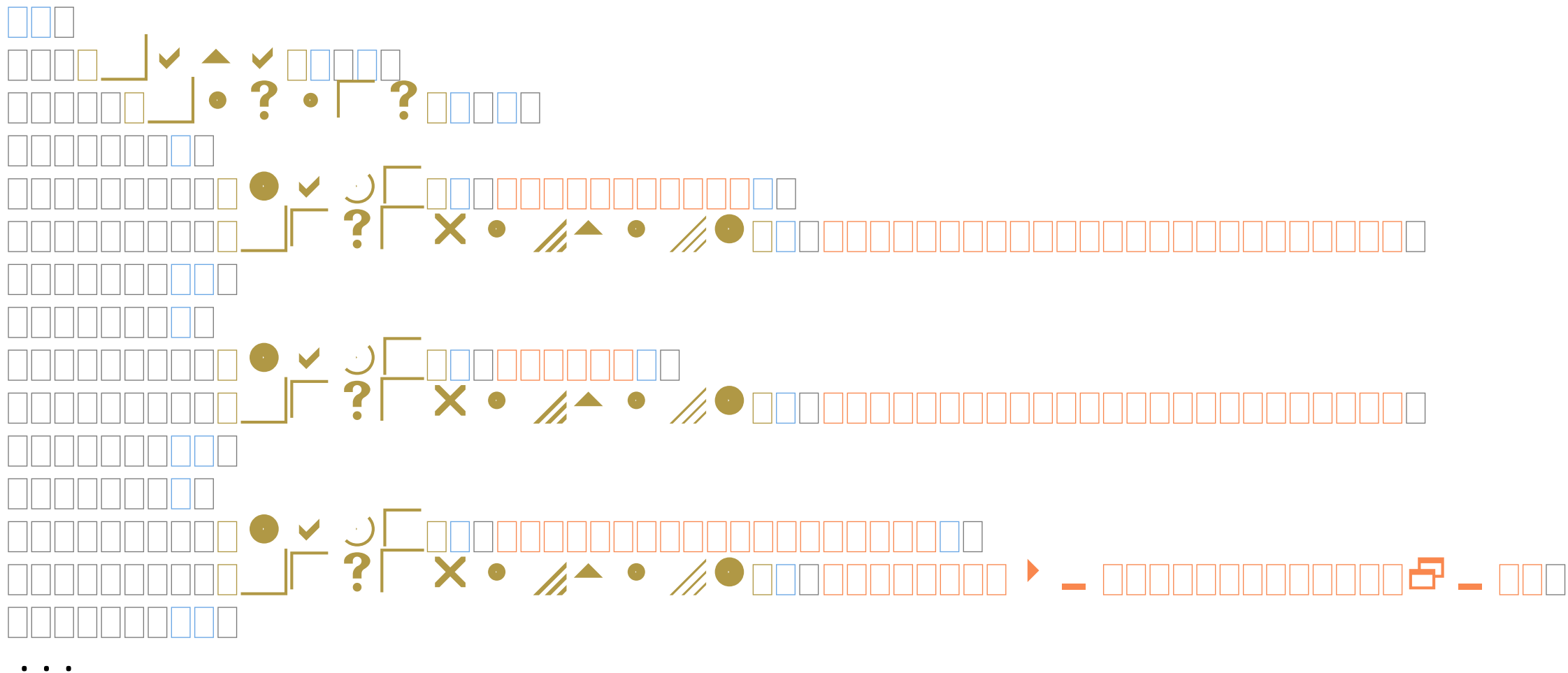
# Сортировка



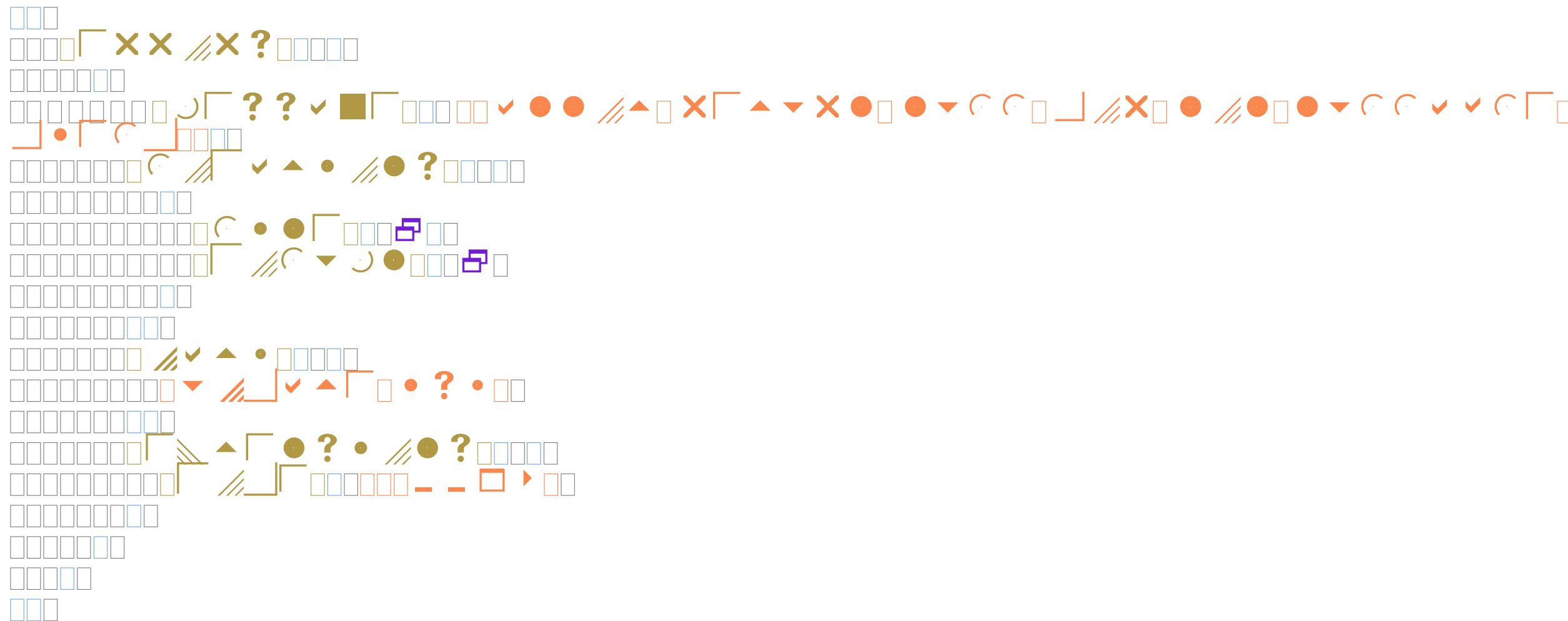
# ChilliCream Hot Chocolate

```
Microsoft.EntityFrameworkCore.Database.Command[20101]  
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']  
SELECT "c"."NormalizedName", "c"."Id", "d"."Name", "d"."Id"  
FROM "Categories" AS "c"  
LEFT JOIN "Dishes" AS "d" ON "c"."Id" = "d"."CategoryId"  
ORDER BY "c"."Id"
```

# Ответ сервера



# Ответ сервера



# ChilliCream Hot Chocolate

---

## MUTATIONS

# Mutations

Тип ***Mutation*** в GraphQL  
используется для изменения  
данных.



# Mutations

## Методы в классе мутаций

- принимают объект, содержащий исходные данные для изменения
- возвращают объект класса (payload), содержащий необходимые для клиента данные

# Mutations

```
//исходные данные для добавления блюда  
public record AddDishInput(string Name, string Description, int  
Calories, int CategoryId);
```

```
//исходные данные для редактирования блюда  
public record UpdateDishInput(int Id, string Name, int Calories,  
string Description);
```

```
// payload для изменения/добавления блюда  
public record DishPayload(Dish dish);
```

# Класс Mutations (добавление блюда)

```
public async Task<DishPayload> AddDish(AddDishInput input)
{
    var dish = new Dish
    {
        Name = input.Name,
        Description = input.Description,
        Calories = input.Calories,
        CategoryId = input.CategoryId,
    };
    _context.Dishes.Add(dish);
    await _context.SaveChangesAsync();
    return new DishPayload(dish);
}
```

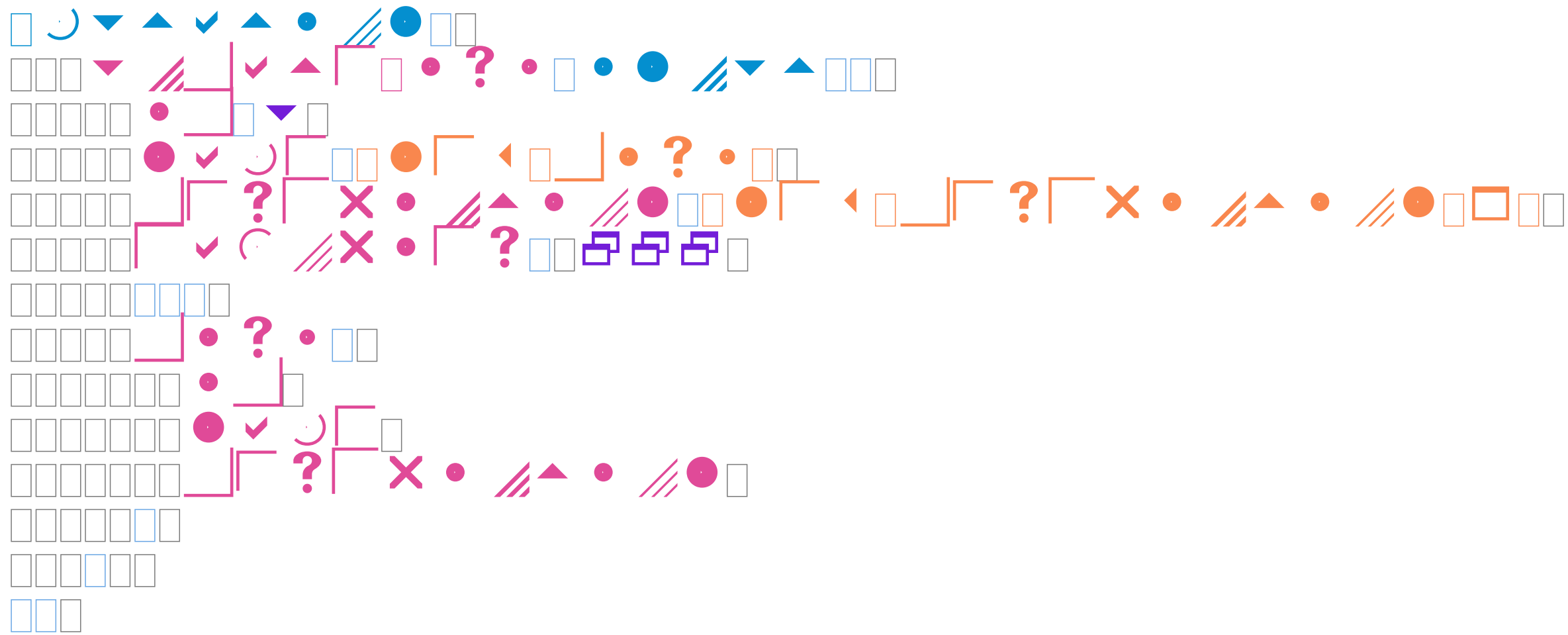
# Класс Mutations (редактирование блюда)

```
public async Task<DishPayload> UpdateDish(UpdateDishInput input)
{
    var result = await _context.Dishes
        .AsNoTracking()
        .Where(d=>d.Id==input.Id)
        .ExecuteUpdateAsync(setter=>
            setter.SetProperty(d=>d.Name, input.Name)
                .SetProperty(d=>d.Description, input.Description)
                .SetProperty(d=>d.Calories, input.Calories));
    if (result == 1)
        return new DishPayload(await _context
                                .Dishes
                                .FindAsync(input.Id));
    return null;
}
```

# Mutations

```
builder.Services.AddGraphQLServer()  
    .AddQueryType<Query>()  
    .AddMutationType<Mutation>() ←  
    .AddProjections()  
    .AddFiltering()  
    .AddSorting();
```

# Mutations



# Создание запросов

---

# Библиотеки .Net

---

ZeroQL

<https://github.com/byme8/ZeroQL/>

GraphQLinq

<https://github.com/Giorgi/GraphQLinq>



# Пример использования ZeroQL

```
var httpClient = new HttpClient();
httpClient.BaseAddress = new Uri("https://localhost:7270/graphql");

var client = new MenuClient(httpClient);
var filter = new DishFilterInput { Id=new IntOperationFilterInput()};
filter.Id.Gt = 2;
var response = await client.Query(o => o.Dishes(where: filter,
                                         null,
                                         d => new {d.Id, d.Name }));

Console.WriteLine($"GraphQL: {response.Query}");
foreach (var item in response.Data)
{
    Console.WriteLine($"{item.Id}: {item.Name}");
}
```