Creating a GraphQL API

Roland Guijt
MVP | CONSULTANT | TRAINER | AUTHOR

@rolandguijt rolandguijt.com

Overview

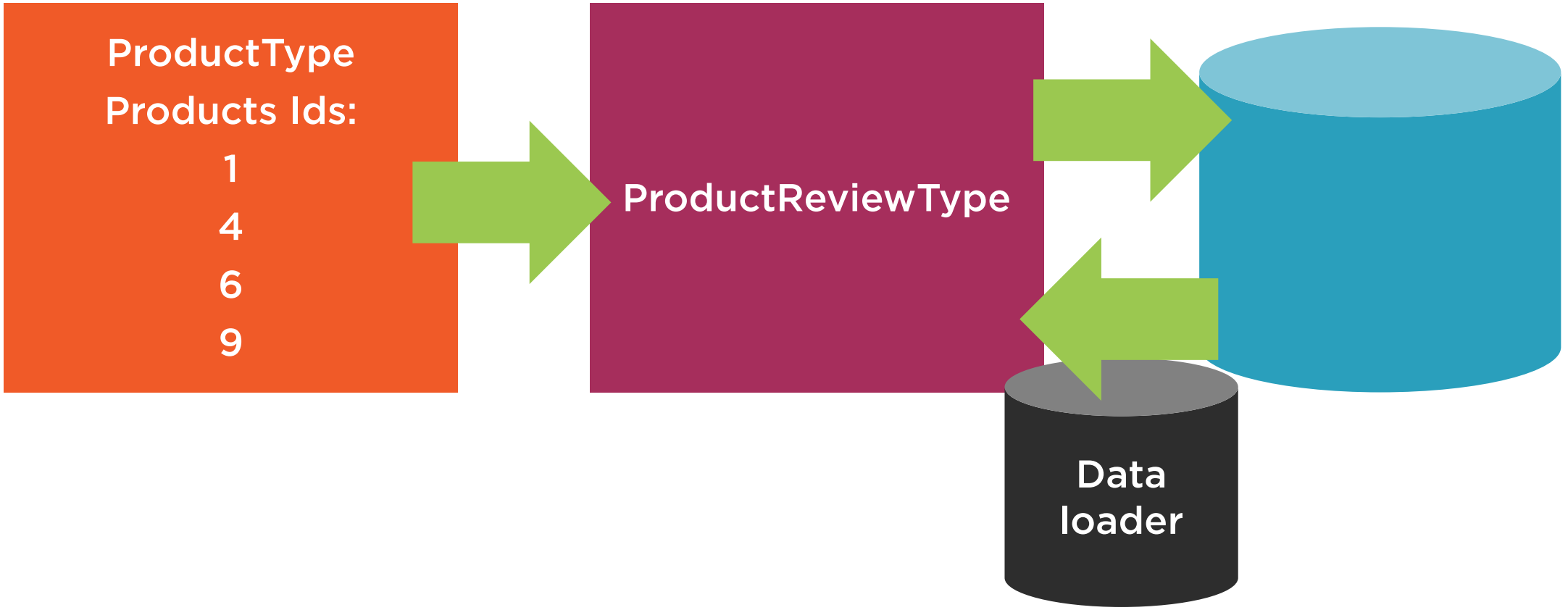
- Adding scalar and complex types
- Data loader
- Arguments
- Authorization
- Interfaces

Database Query Efficiency

Implement a cache in the repository or data layer

GraphQL solution: Data Loader

Dataloader


ProductType
Products Ids:
1
4
6
9
ProductReviewType
Data
loader

Dataloader values

1	ProductReview
1	ProductReview
1	ProductReview
4	ProductReview
4	ProductReview
8	ProductReview
8	ProductReview
8	ProductReview

GraphQL API Structure

Authorization

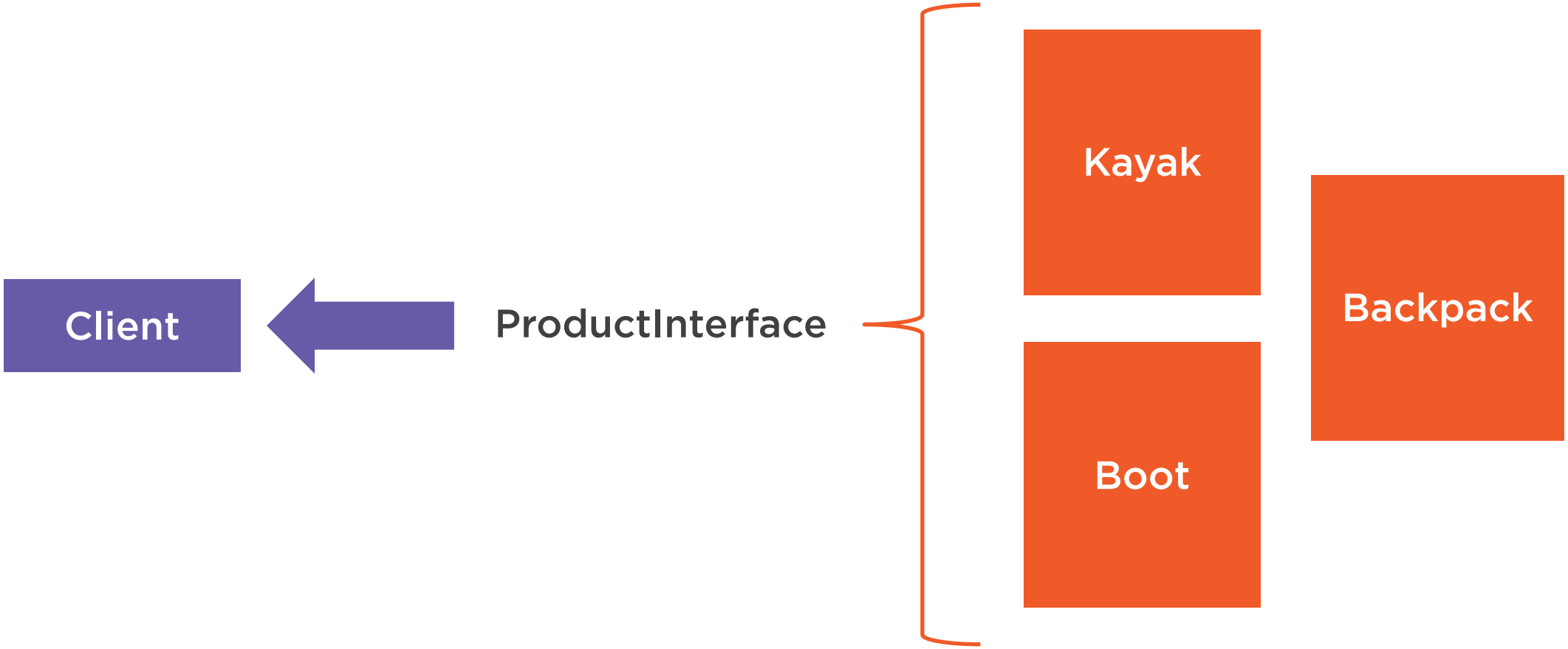
Use ASP.NET Core features

AuthorizationService

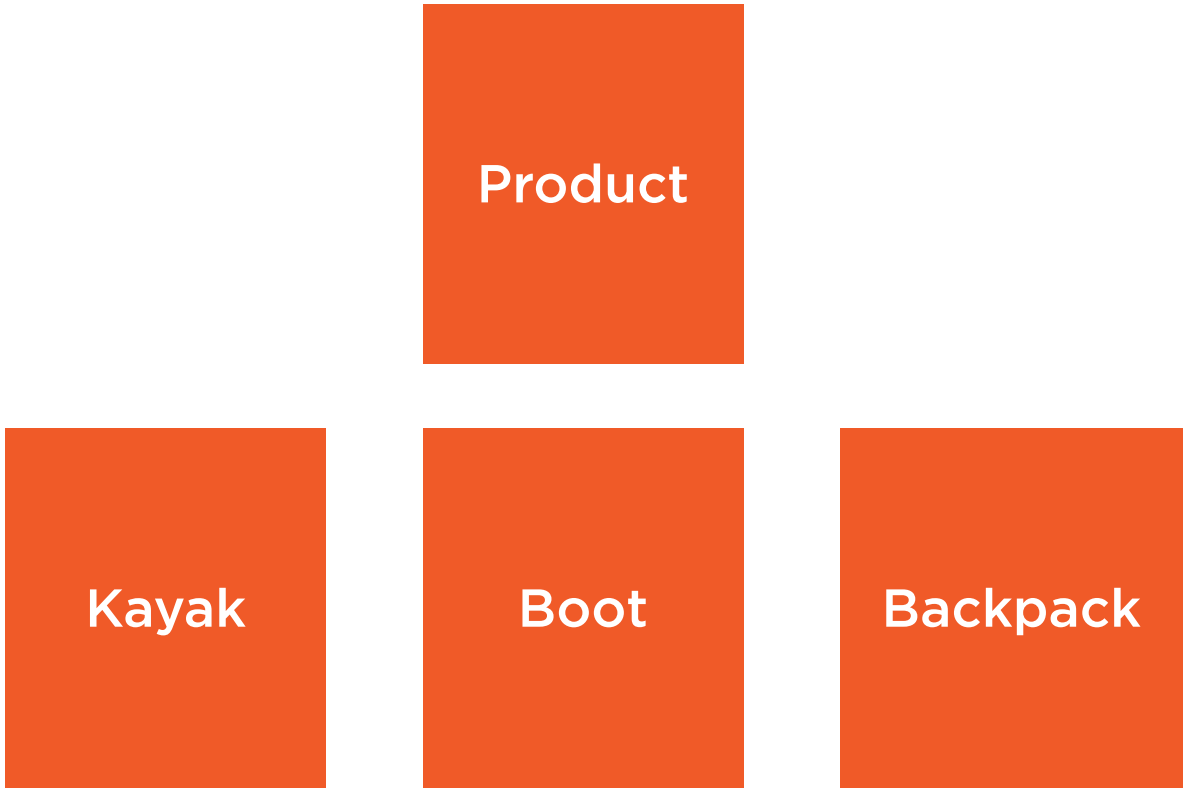
Policies

Understanding ASP.NET Core Security

Interfaces



Entities



Defining the Interface

```
public class ProductInterface : InterfaceGraphType<Product>
{
    public ProductInterface()
    {
        Name = "Product";
        Field(d => d.Id);
        Field(d => d.Name);
        ..
    }
}
```

Implementing the Interface

```
public class BootProductType : ObjectGraphType<Boot>
{
    public BootProductType()
    {
        Name = "Boot";
        Field(d => d.Id);
        Field(d => d.Name);

        Interface<ProductInterface>();
    }
}
```

Using the Interface

```
public class CarvedRockQuery: ObjectGraphType
{
    public CarvedRockQuery(ProductRepository productRepository)
    {
        Field<ListGraphType<ProductInterface>>(
          "products",
          resolve: context => productRepository.GetAll()
        );

    …
```

Summary

- Built-in and custom scalar types
- Complex types in the hierarchy
- Data loader optimizes data retrieval
- Arguments let the consumer pass in parameters
- Authorization in schema layer
- Interfaces to do polymorphism