GraphQL with Python frameworks

Create next-generation API with ease

Bartosz Kazuła

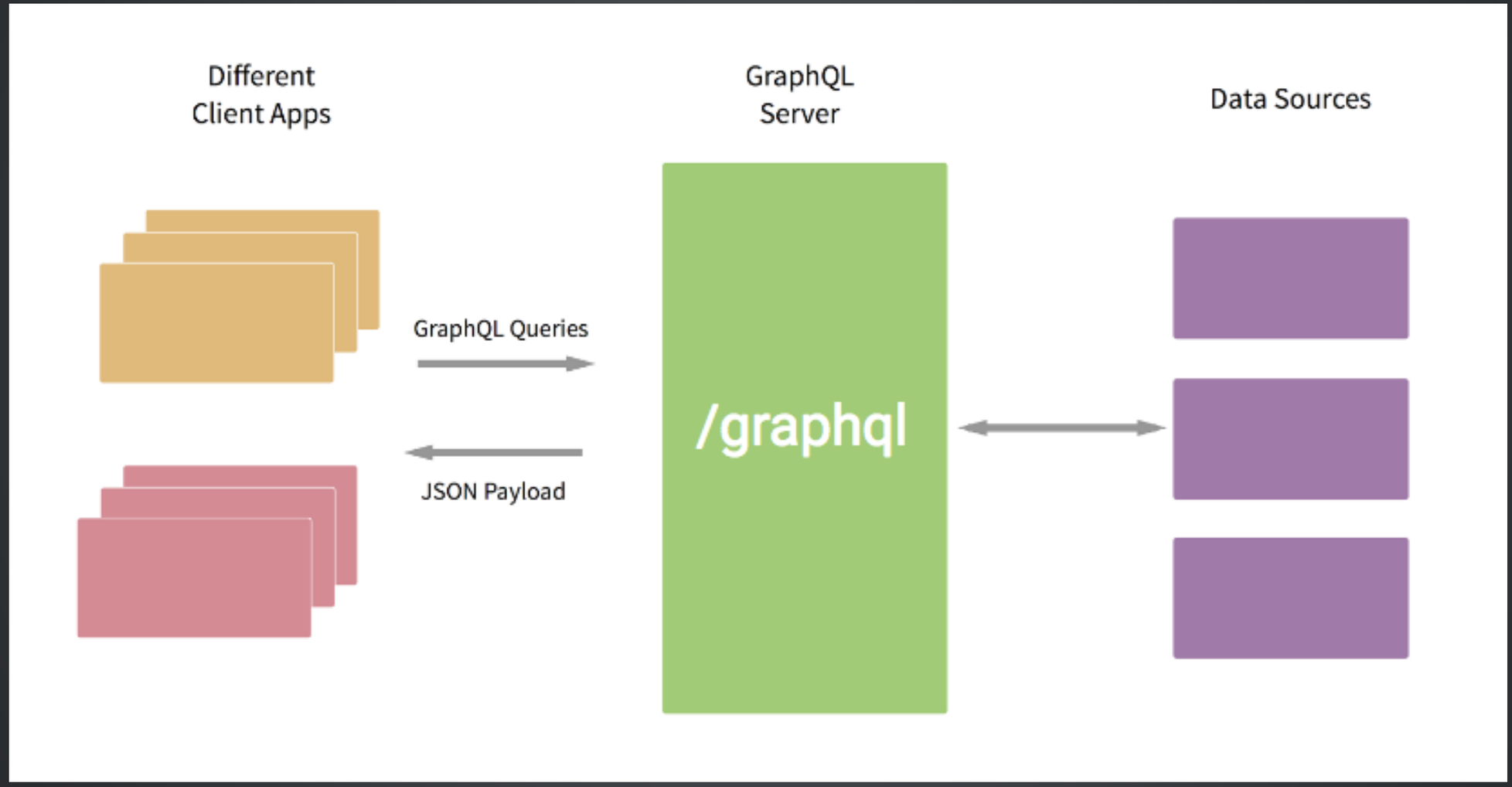
About mua

- Frontend/Backend/Mobile developer
- Software architect
- Highly focused on new technologies

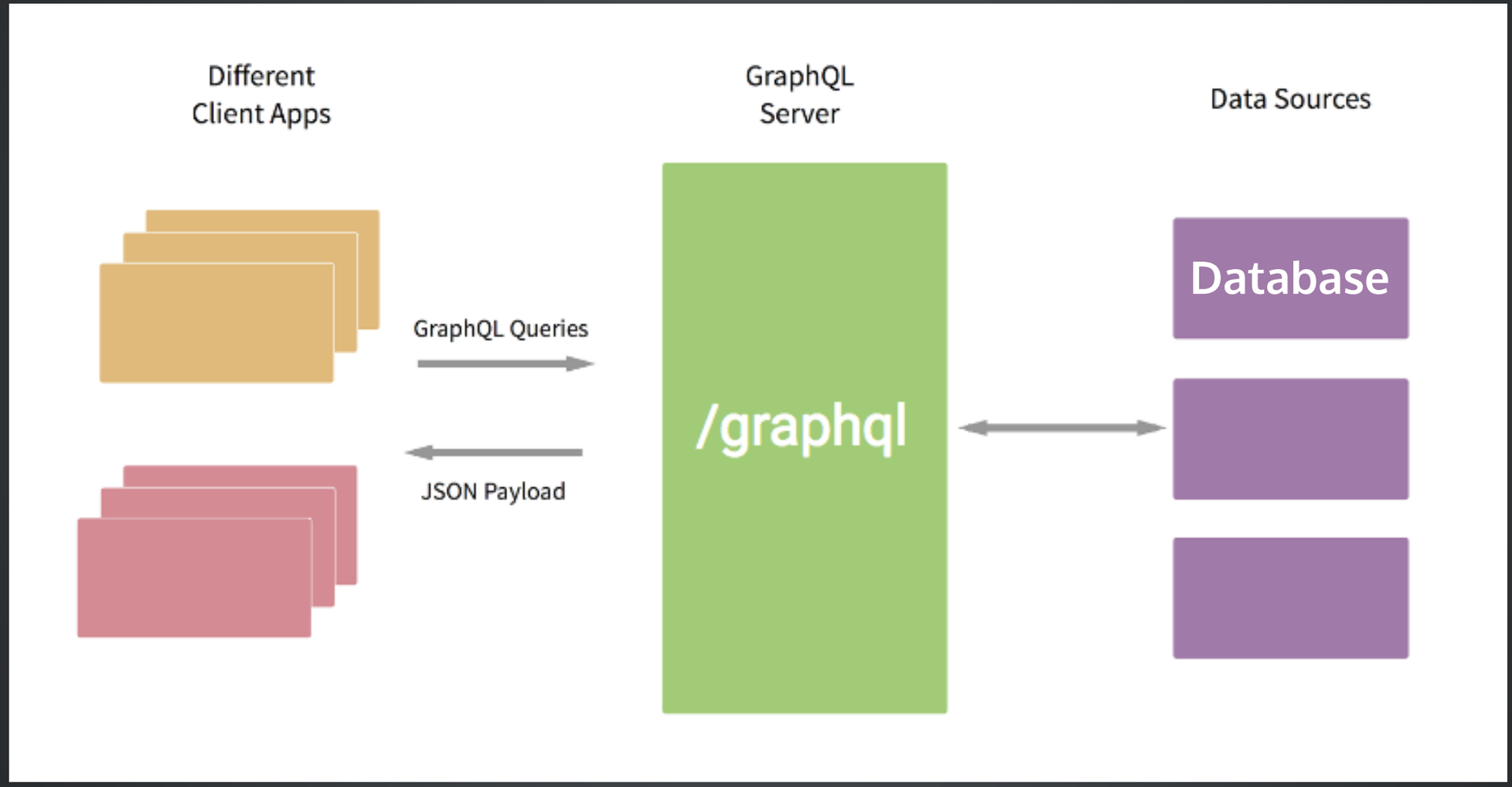
What is GraphQL?

A query language for your API

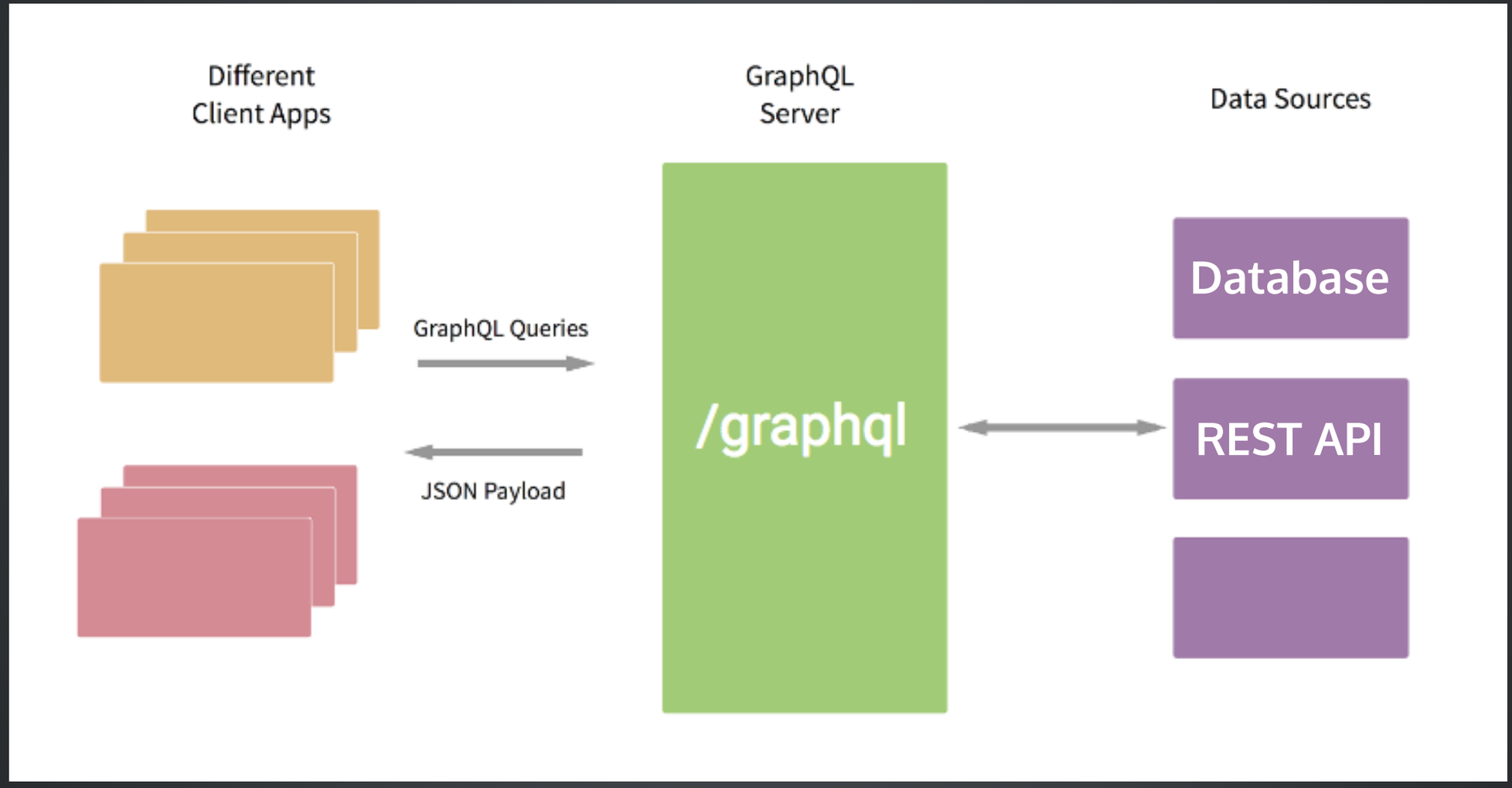
How it works


Different
Client Apps
GraphQL
Server
Data Sources
GraphQL Queries
JSON Payload
/graphql

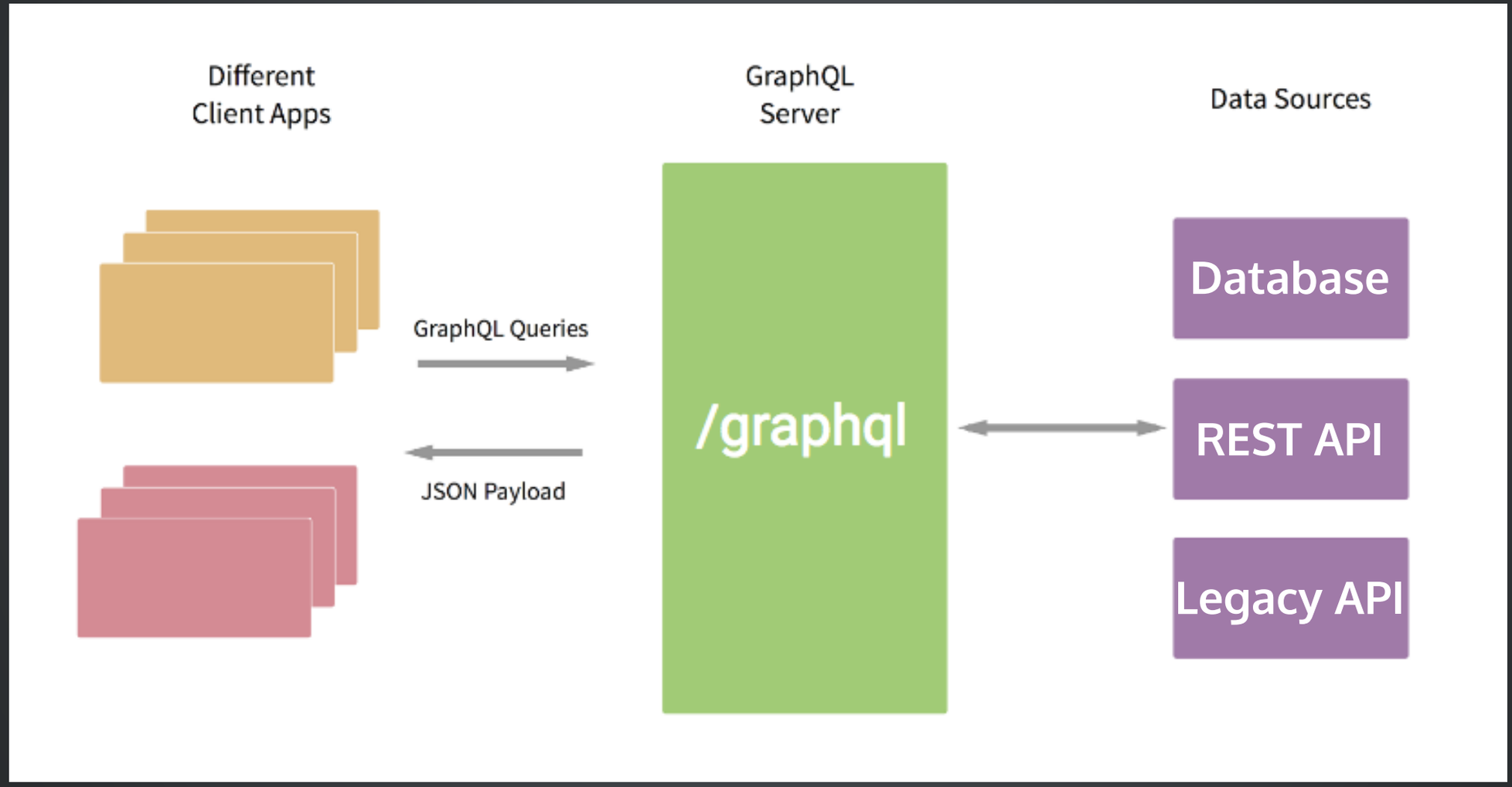
How it works


Different
Client Apps
GraphQL
Server
Data Sources
GraphQL Queries
JSON Payload
/graphql
Database

How it works


Different
Client Apps
GraphQL
Server
Data Sources
GraphQL Queries
JSON Payload
/graphql
Database
REST API

How it works


Different
Client Apps
GraphQL
Server
Data Sources
GraphQL Queries
JSON Payload
/graphql
Database
REST API
Legacy API

Is it production ready?

Data structures

- Schema
- Type
- Query
- Mutation

Schema

Entrypoint to API, defines queries and mutations that can be accessed via GraphQL

```
type Schema {
    query: Query
    mutation: Mutation
}
```

Type

The most basic components of a GraphQL schema, which just represent a kind of object you can fetch from your service, and what fields it has

```
type Character {
    name: String!
    appearsIn: [Episode]!
}
```

Query

Asking for fields and objects based on provided parameters. Here you can define the shape of response.

```
{
    hero {
        name
    }
}
```

Mutation

The way to modify data. If the mutation field returns an object type, you can ask for nested fields.

```
mutation CreateReviewForEpisode($ep: Episode!, $review: ReviewInput!) {
        createReview(episode: $ep, review: $review) {
            stars
            commentary
        }
}
```

```
{
    "ep": "JEDI",
    "review": {
        "stars": 5,
        "commentary": "This is a great movie!"
    }
}
```

GraphiQL

A graphical interactive in-browser GraphQL IDE for testing and writing queries.

Graphene

GraphQL in Python made easy

https://graphene-python.org/

```
pip install graphene
```

Let's build some API

- Add user / get user / get all users
- Add blog post / get post / get all posts

Django

```
pip install graphene graphene-django
```

```
pip install graphene graphene-django
```

```
INSTALLED_APPS = (
    # ...
    'graphene_django',
)

GRAPHENE = {
    'SCHEMA': 'app.schema.schema' # Path to main schema
}
```

```
pip install graphene graphene-django
```

```
INSTALLED_APPS = (
    # ...
    'graphene_django',
)

GRAPHENE = {
    'SCHEMA': 'app.schema.schema' # Path to main schema
}
```

```
from django.urls import path
from graphene_django.views import GraphQLView

urlpatterns = [
    # ...
    path('graphql/', GraphQLView.as_view(graphiql=True)),
]
```

User model

```
class User(models.Model):
    name = models.CharField(max_length=20)
    surname = models.CharField(max_length=30)
    age = models.IntegerField()
```

User model

```
class User(models.Model):
    name = models.CharField(max_length=20)
    surname = models.CharField(max_length=30)
    age = models.IntegerField()
```

UserType

```
from graphene_django import DjangoObjectType

class UserType(DjangoObjectType):
    class Meta:
        model = User
```

User Query

```
class UserQuery(graphene.ObjectType):
    users = graphene.List(UserType)

    def resolve_users(self, info):
        return User.objects.all()
```

Schema

app/schema.py

```
schema = graphene.Schema(query=UserQuery)
```

```
{
  users {
    name
    surname
    age
  }
}
```


```
{
  "data": {
    "users": [
      {
        "age": 70,
        "name": "Luke",
        "surname": "Skywalker"
      },
      {
        "age": 99,
        "name": "Obi",
        "surname": "Wan-Kenobi"
      },
      {
        "age": 80,
        "name": "Anakin",
        "surname": "Skywalker"
      }
    ]
  }
```

Getting one user

```
class UserQuery(graphene.ObjectType):
    users = graphene.List(UserType)
    user = graphene.Field(UserType, name=graphene.String())

    def resolve_users(self, info):
        return User.objects.all()

    def resolve_user(self, info, name):
        return User.objects.filter(name=name).first()
```

Getting one user

```
class UserQuery(graphene.ObjectType):
    users = graphene.List(UserType)
    user = graphene.Field(UserType, name=graphene.String())

    def resolve_users(self, info):
        return User.objects.all()

    def resolve_user(self, info, name):
        return User.objects.filter(name=name).first()
```

Getting one user

```
class UserQuery(graphene.ObjectType):
    users = graphene.List(UserType)
    user = graphene.Field(UserType, name=graphene.String())

    def resolve_users(self, info):
        return User.objects.all()

    def resolve_user(self, info, name):
        return User.objects.filter(name=name).first()
```

```
{
  user(name: "Luke") {
    name
    surname
    age
  }
}
```


```
{
  "data": {
    "user": {
      "name": "Luke",
      "surname": "Skywalker",
      "age": 70
    }
  }
}
```

User mutation

Plain GraphQL mutation

```
class UserMutation(graphene.Mutation):
    user = graphene.Field(UserType)

    class Arguments:
        name = graphene.String()
        surname = graphene.String()
        age = graphene.Int()

    def mutate(self, info, **kwargs):
        user = User.objects.create(**kwargs)
        return UserMutation(user=user)
```

Add mutation to the schema

```
class Mutations(graphene.ObjectType):
    create_user = UserMutation.Field()

schema = graphene.Schema(query=UserQuery, mutation=Mutations)
```

Add mutation to the schema

```
class Mutations(graphene.ObjectType):
    create_user = UserMutation.Field()

schema = graphene.Schema(query=UserQuery, mutation=Mutations)
```

```
mutation {
  createUser(
    age: 50,
    name: "Han",
    surname: "Solo"
  ) {
    user {
      name
      surname
      age
    }
  }
}
```

```
{
  "data": {
    "createUser": {
      "user": {
        "name": "Han",
        "surname": "Solo",
        "age": 50
      }
    }
  }
}
```

Django Forms-based mutation

```
class UserForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ('name', 'surname', 'age')

class UserMutation(DjangoModelFormMutation):
    class Meta:
        form_class = UserForm
```

```
mutation {
  createUser(input:{
    name: "Leya",
    surname: "Skywalker",
    age: 80
  }) {
    user {
      name
      surname
      age
    }
  }
}
```

```
{
  "data": {
    "createUser": {
      "user": {
        "name": "Leya",
        "surname": "Skywalker",
        "age": 80
      }
    }
  }
}
```

Django FAQ

- I use REST Framework - how to live?
  Take a look at graphene_django.rest_framework package.
- How to validate data?
  When using forms - use built-in validation.
  When going plain - add your own validator.
- How to receive user data?
  Use info.context variable passed to every function.

GraphQL with Django
Q&A

Flask

Installation

```
pip install graphene graphene_sqlalchemy Flask-GraphQL
```

```
app.add_url_rule(
    '/graphql',
    view_func=GraphQLView.as_view(
        'graphql',
        schema=schema,
        graphiql=True
    )
)
```

Post model

```
class Post(Model):
    __tablename__ = 'posts'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    intro = Column(Text)
    content = Column(Text)
    author = Column(String)
```

Type and query

```
class PostType(SQLAlchemyObjectType):
    class Meta:
        model = Post
```

Type and query

```
class PostType(SQLAlchemyObjectType):
    class Meta:
        model = Post
```

```
class Query(graphene.ObjectType):
    posts = graphene.List(PostType)

    def resolve_posts(self, info):
        return Post.query.all()
```

```
{
  posts {
    name
    intro
    content
    author
  }
}
```


```
{
  "data": {
    "posts": [
      {
        "name": "Hello, world",
        "intro": "Introduction to
brand new blog",
        "content": "I created new
blog and I like it!",
        "author": "Admin"
      },
      {
        "name": "My father!!",
        "intro": "Check out who my
father is!",
        "content": "Today I found out
that my father is Darth Vader!",
        "author": "Luke Skywalker"
      }
    ]
  }
}
```

Get one post

```
class Query(graphene.ObjectType):
    posts = graphene.List(PostType)
    post = graphene.Field(PostType, id=graphene.Int())

    def resolve_posts(self, info):
        return Post.query.all()

    def resolve_post(self, info, id):
        return Post.query.get(id)
```

Get one post

```
class Query(graphene.ObjectType):
    posts = graphene.List(PostType)
    post = graphene.Field(PostType, id=graphene.Int())

    def resolve_posts(self, info):
        return Post.query.all()

    def resolve_post(self, info, id):
        return Post.query.get(id)
```

```
{
  post(id:2) {
    id
    name
    intro
    content
    author
  }
}
```


```
{
  "data": {
    "post": {
      "id": "2",
      "name": "My father!!",
      "intro": "Check out who my father is!",
      "content": "Today I found out that my father is Darth Vader!",
      "author": "Luke Skywalker"
    }
  }
}
```

A mutation has no helpers ☹️

Create post mutation

```
class CreatePost(graphene.Mutation):
    post = graphene.Field(PostType)

    class Arguments:
        name = graphene.String()
        intro = graphene.String()
        content = graphene.String()
        author = graphene.String()

    def mutate(self, info, **kwargs):
        post = Post(**kwargs)
        db_session.add(post)
        db_session.commit()
```

Add mutation to schema

```
class Mutations(graphene.ObjectType):
    create_post = CreatePost.Field()

schema = graphene.Schema(query=Query, mutation=Mutations)
```

```
mutation {
  createPost(
    author: "Obi Wan-K
    intro: "Anakin zdr
    content: "Ten młod
    name: "Niedowiary!
  ) {
    post {
      name
      intro
      content
      author
    }
  }
}
```

```
{
  "data": {
    "createPost": {
      "post": {
        "name": "Niedowiary!",
        "intro": "Anakin zdradził!",
        "content": "Ten młody dureń
przeszedł na ciemną stronę mocy!",
        "author": "Obi Wan-Kenobi"
      }
    }
  }
}
```

FAQ

How to handle errors?

Check out the "errors" list next to data. If it has some errors then handle them.

You can customize your GraphQLView to handle errors differently.

Best authorization approach?

There are two:

1. Private & public GraphQL endpoints (**BAD!**)
2. Login required decorator wrapping classes/methods

How to limit database calls?

Read about batching queries.

Thanks for watching

Bartosz Kazuła

bartosz@kazula.eu

bkazula