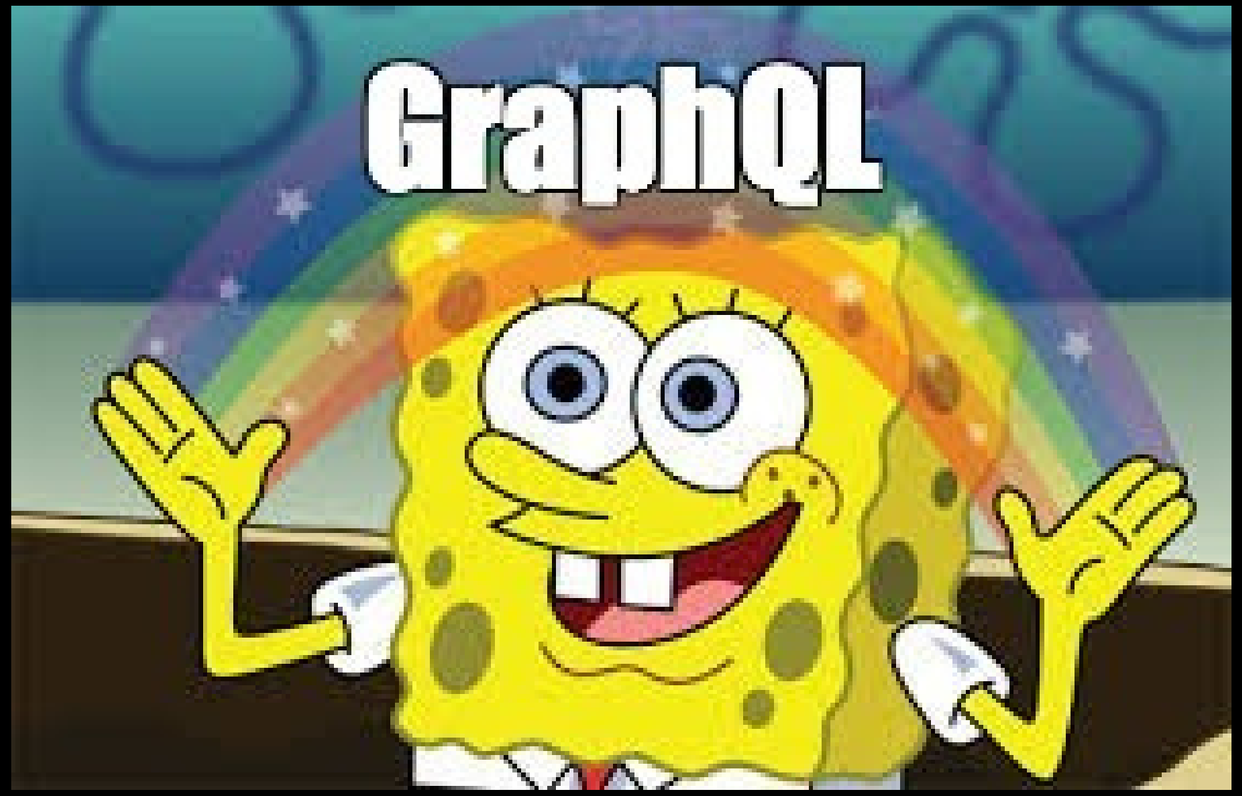
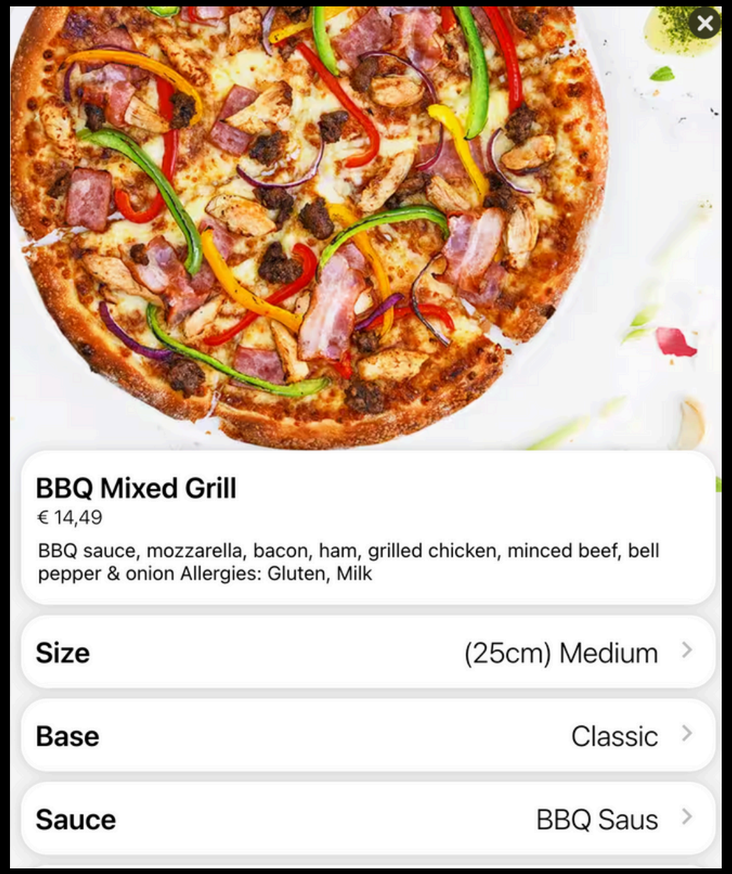
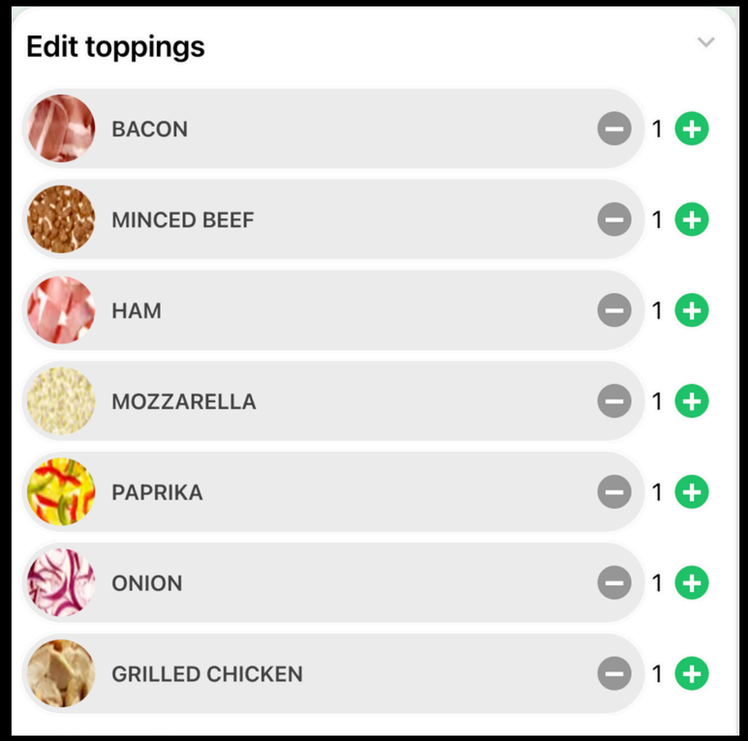
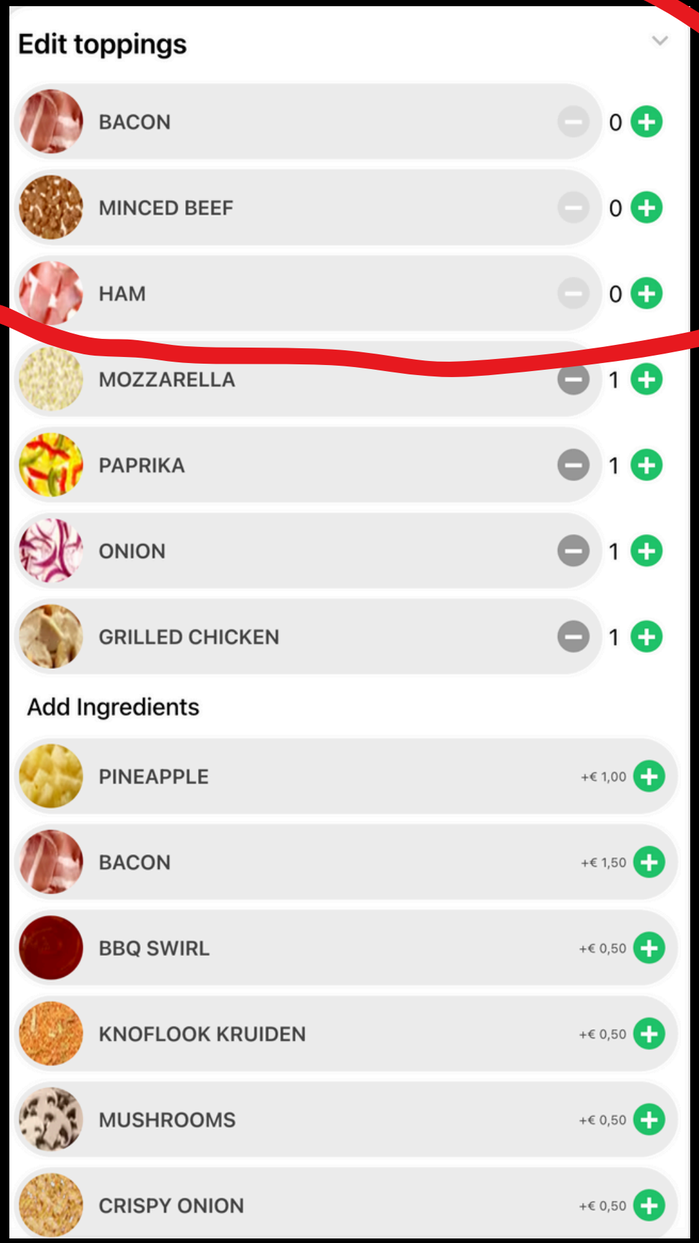


in Interprise Applications

Introduction to GraphQL







What if...

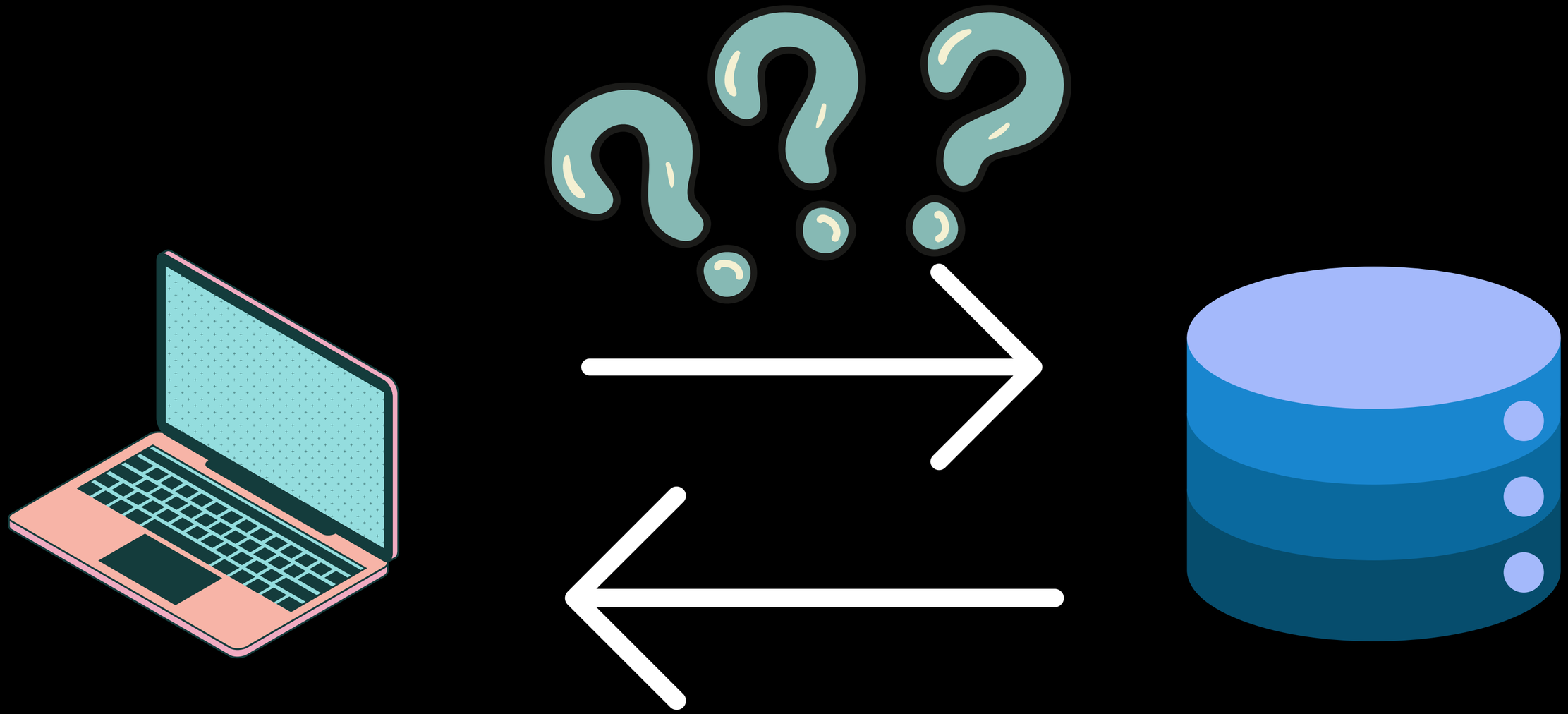
receipt.md

```
1   ------------------------------
2   |     |     Receipt
3   ------------------------------
4   Client Name: John Doe
5   Address: 123 Main Street, Springfield
6
7   Order Details:
8   - Pizza: BBQ Mixed Grill
9   - Size: Medium
10  ------------------------------
```

postgres.sql

Run on active connection | Select block

```sql
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(15),
    address VARCHAR(255)
);

CREATE TABLE pizzas (
    pizza_id SERIAL PRIMARY KEY,
    order_id INT NOT NULL,
    name VARCHAR(100),
    size VARCHAR(20),
    crust_type VARCHAR(50),
    sauce VARCHAR(50),
    cheese_type VARCHAR(50),
    is_vegetarian BOOLEAN,
    calories INT,
    price DECIMAL(5, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

```



receipt.md

```
1   ------------------------------------
2   |     |     |     Receipt
3   ------------------------------------
4   Client Name: John Doe
5   Address: 123 Main Street, Springfield
6
7   Order Details:
8   - Pizza: BBQ Mixed Grill
9   - Size: Medium
10  ------------------------------------
```

```
3     }
4   }
```

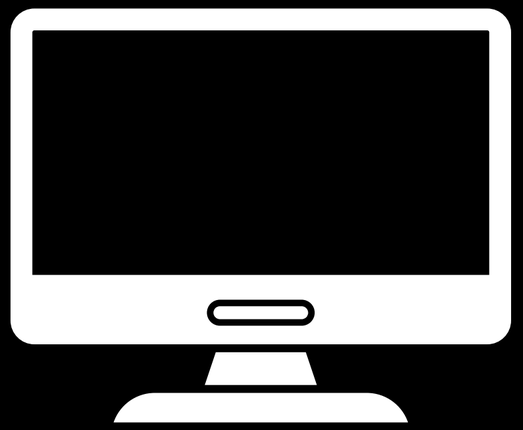
schema.graphql

```graphql
query getOrder($orderId: ID!) {
  order(id: $orderId) {
    customer {
      name
      address
    }
    pizza {
      name
      size
    }
  }
}
```

graphqlResponse.json > ...

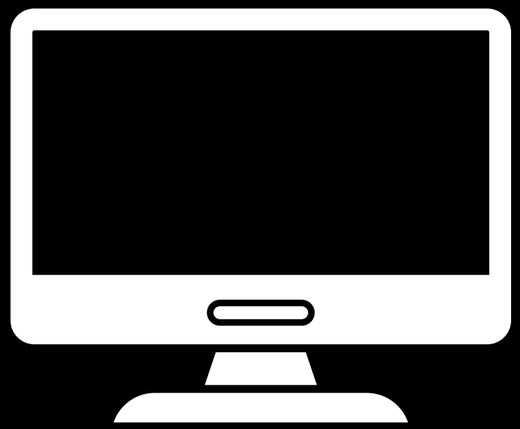
```json
{
    "order": {
      "customer": {
        "name": "John Doe",
        "address": "123 Main Street, Springfield"
      },
      "pizza": {
        "name": "BBQ Mixed Grill",
        "size": "Medium"
      }
    }
}
```

REST API

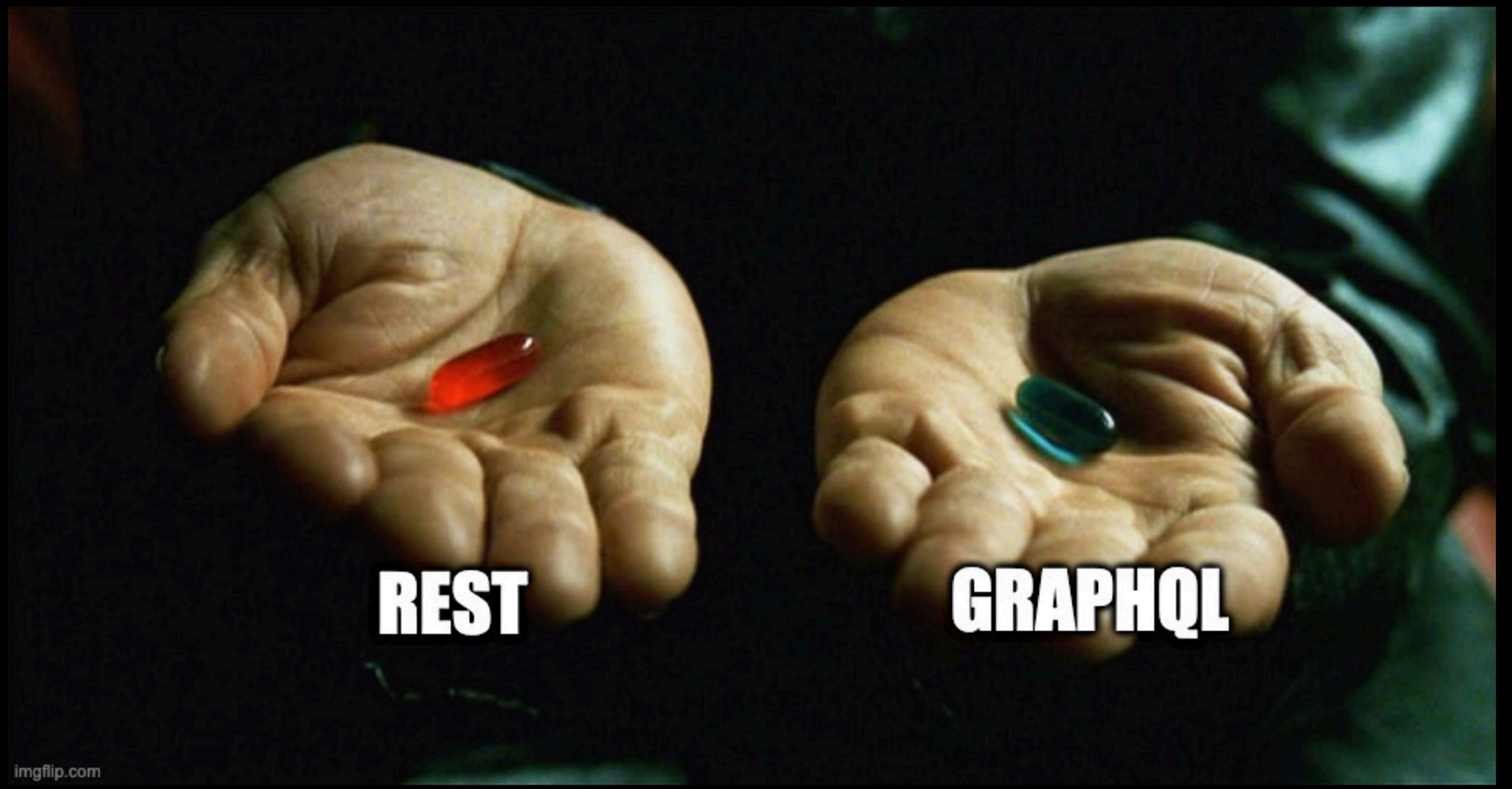


/GET/pizza

/GET/customer

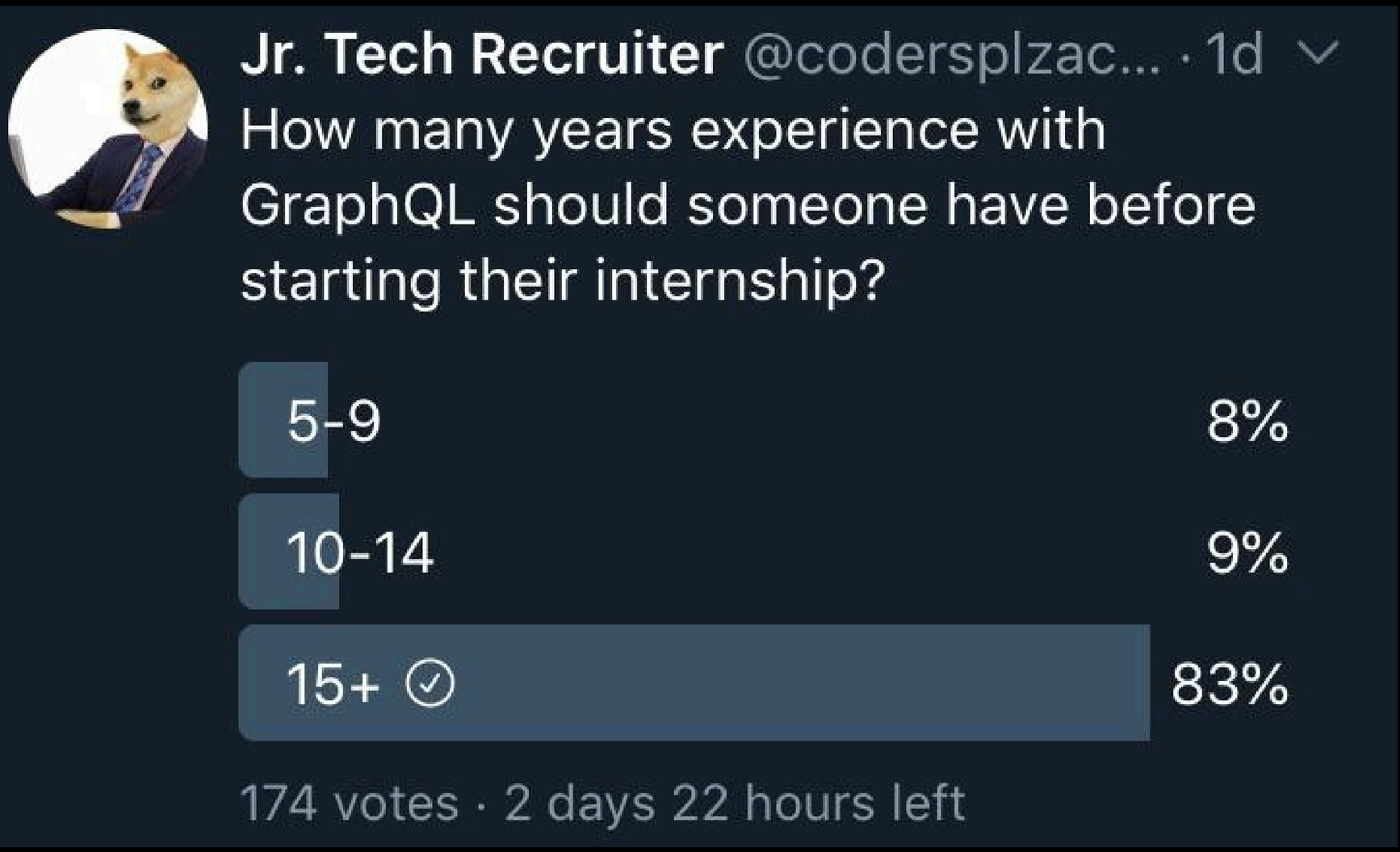


/graphql

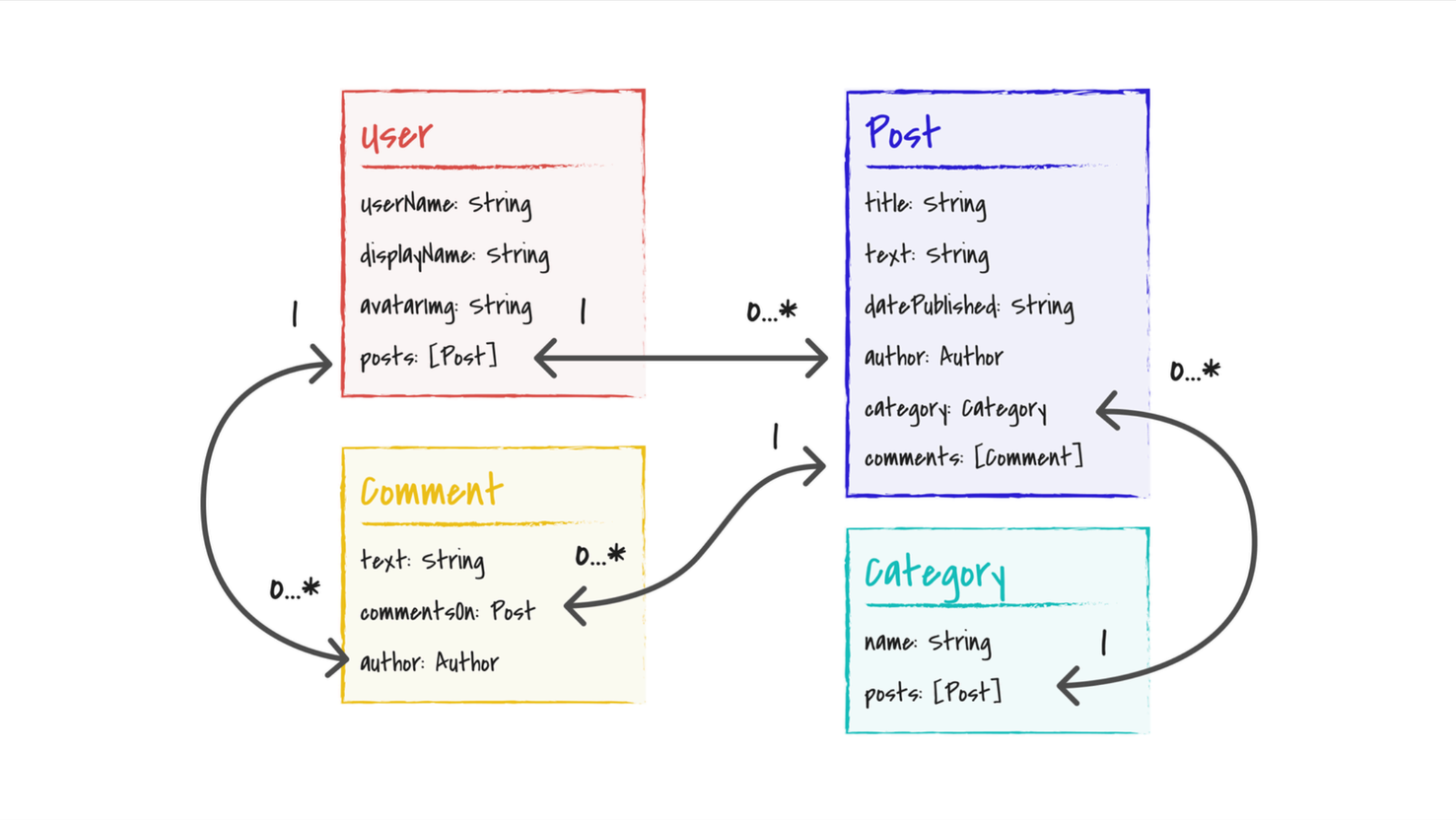

REST
GRAPHQL
imgflip.com

2. GraphQL Basics

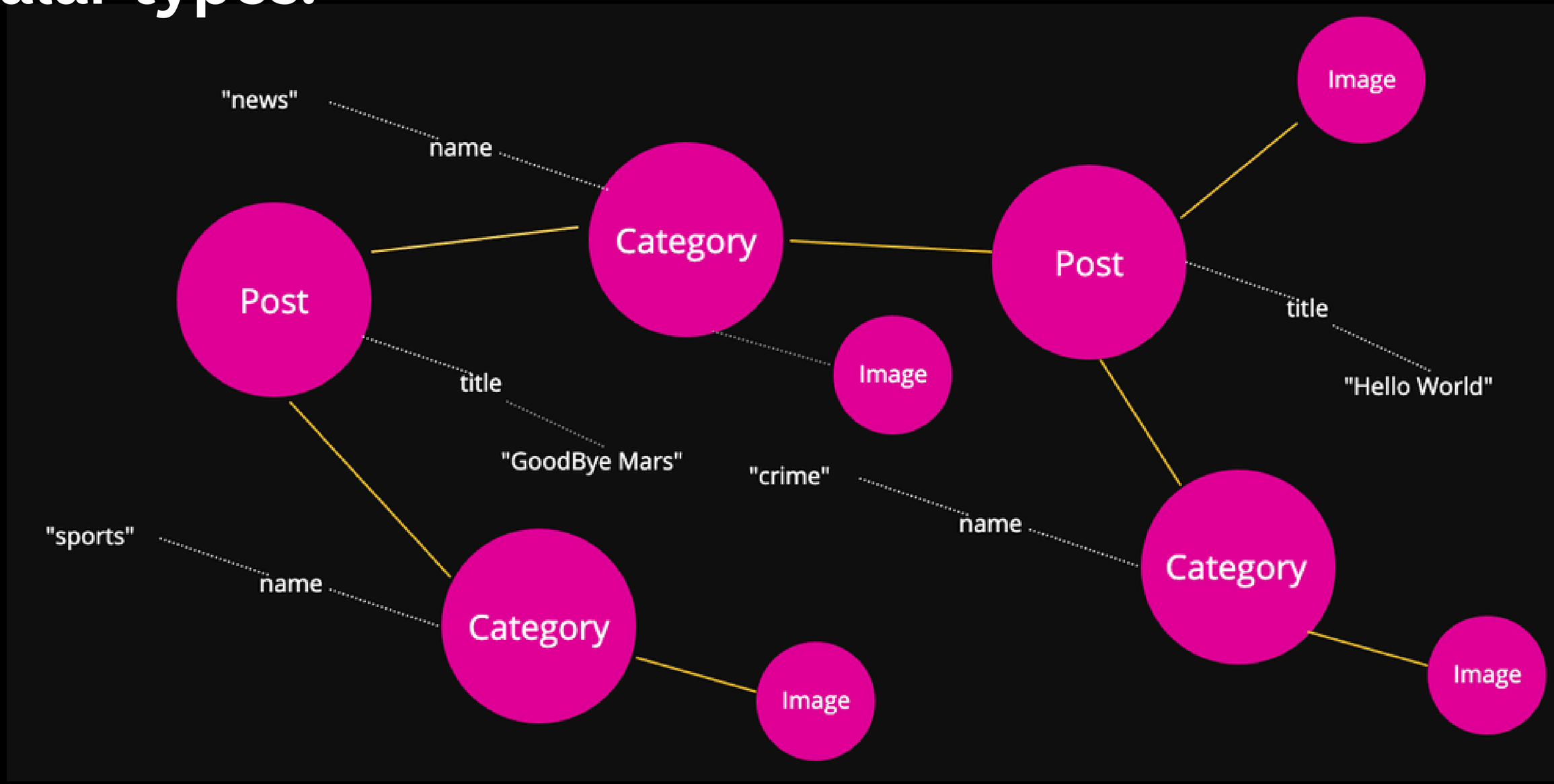
what is....



Schema:

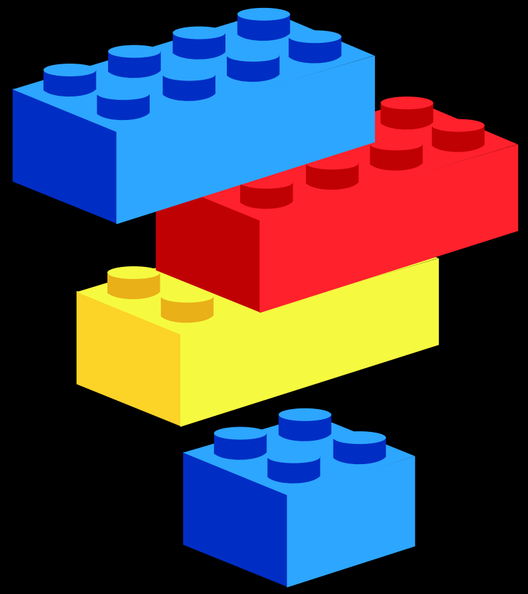

User
userName: String
displayName: String
avatarImg: String
posts: [Post]
Post
title: String
text: String
datePublished: String
author: Author
category: Category
comments: [Comment]
Comment
text: String
commentsOn: Post
author: Author
Category
name: String
posts: [Post]
1
1
0...*
0...*
1
0...*
0...*
1

Scalar types:



```
223     query {
224       post(id: "1") {
225         title  # Scalar
226         category {
227           name  # Scala
228         }
229         image {
230           url   # Scala
231         }
232       }
233     }
```

Object Types:



```
15    type Pizza {
16      name: String
17      size: String
18      toppings: [Topping]
19    }
20
21    type Topping {
22      name: String
23    }
24
```

Why type Query is Essential?

```
130  type Query {
131    pizza(id: ID!): Pizza
132    allPizzas: [Pizza]
133    allToppings: [Topping]
134    customer(id: ID!): Customer
135    allCustomers: [Customer]
136    order(id: ID!): Order
137    allOrders: [Order]
138  }
```

```
type Pizza {
  name: String
  size: String
  toppings: [Topping]
}

type Topping {
  name: String
}

type Customer {
  id: ID!
  name: String
  address: String  # Field for customer address
}

type Order {
  id: ID!
  customer: Customer   # Reference to the customer who placed the order
  pizzas: [Pizza]      # List of pizzas in the order
  orderDate: String    # Date when the order was placed
  totalAmount: Float   # Total amount for the order
}
```

```
query {
  order(id: "789") {
    customer {
      name
      address
    }
    pizzas {
      name
      size
      toppings {
        name
      }
    }
    orderDate
    totalAmount
  }
}
```

```
query {
  order(id: "789") {
    customer {
      address
    }
    pizzas {
      toppings {
        name
      }
    }
  }
}
```

```
GET /pizzas/{id}
GET /pizzas
GET /toppings
GET /customers/{id}
GET /customers
GET /orders/{id}
GET /orders
```

/graphql

When Query type is not enough

In REST API

```
GET /pizzas/{id}
GET /pizzas
GET /toppings
GET /customers/{id}
GET /customers
GET /orders/{id}
GET /orders
```

```
### 9 Endpoints for Mutations (POST, PUT, DELETE) Operations:
8. **POST /pizzas**: To create a new pizza.
9. **PUT /pizzas/{id}**: To update a specific pizza.
10. **DELETE /pizzas/{id}**: To delete a specific pizza.
11. **POST /customers**: To create a new customer.
12. **PUT /customers/{id}**: To update a specific customer.
13. **DELETE /customers/{id}**: To delete a specific customer.
14. **POST /orders**: To create a new order.
15. **PUT /orders/{id}**: To update a specific order.
16. **DELETE /orders/{id}**: To delete a specific order.
```

Mutations:

/graphql

real-time, modifying data

```
164  type Mutation {
165    addPizza(name: String!, size: String!): Pizza  # Mutation for creating a new pizza
166    updatePizza(id: ID!, name: String, size: String): Pizza  # Mutation for updating a pizza
167    deletePizza(id: ID!): String  # Mutation for deleting a pizza
168  }
169
170  type Subscription {
171    pizzaAdded: Pizza  # Subscription that triggers when a new pizza is added
172  }
```

CRUD

```
64  mutation {
65    addPizza(name: "Pepperoni Feast", size: "Large") {
66      id
67      name
68      size
69    }
70  }
```

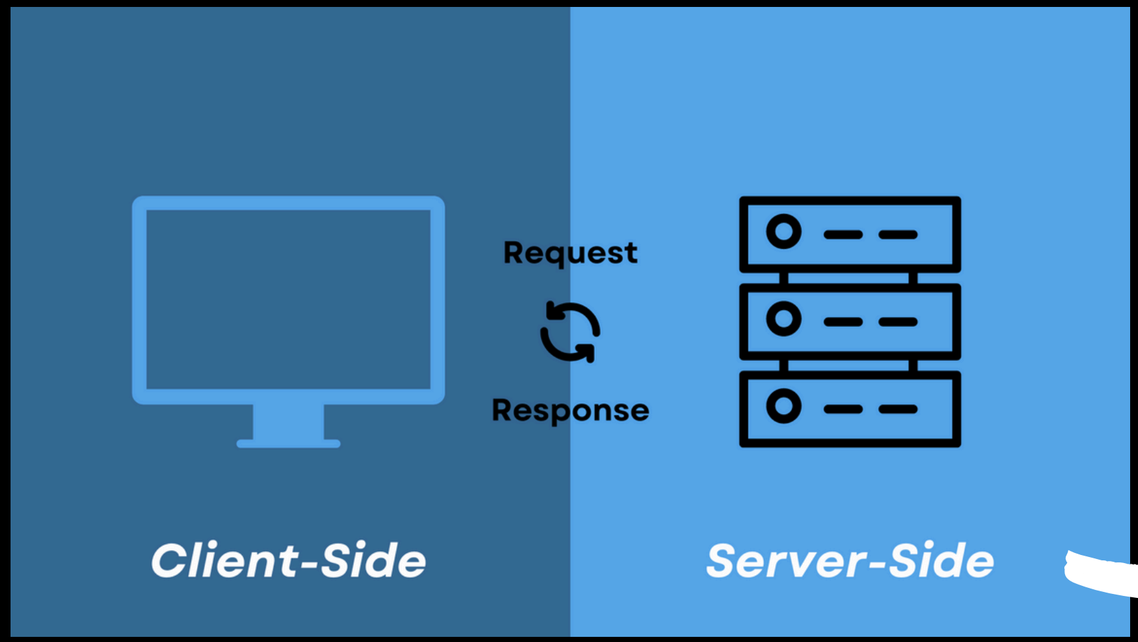
```
72  mutation {
73    updatePizza(id: 1, na                    xe", ize: "M      ium"
74      id
75      name
76      size
77    }
78  }
```

```
80    mutation {
81      deletePizza(id: 1)
82    }
83
```

```
5     qu      ry {
6       izza(id: 1) {
87        id
88        name
89        size
90      }
91    }
```

At last...Enterprises

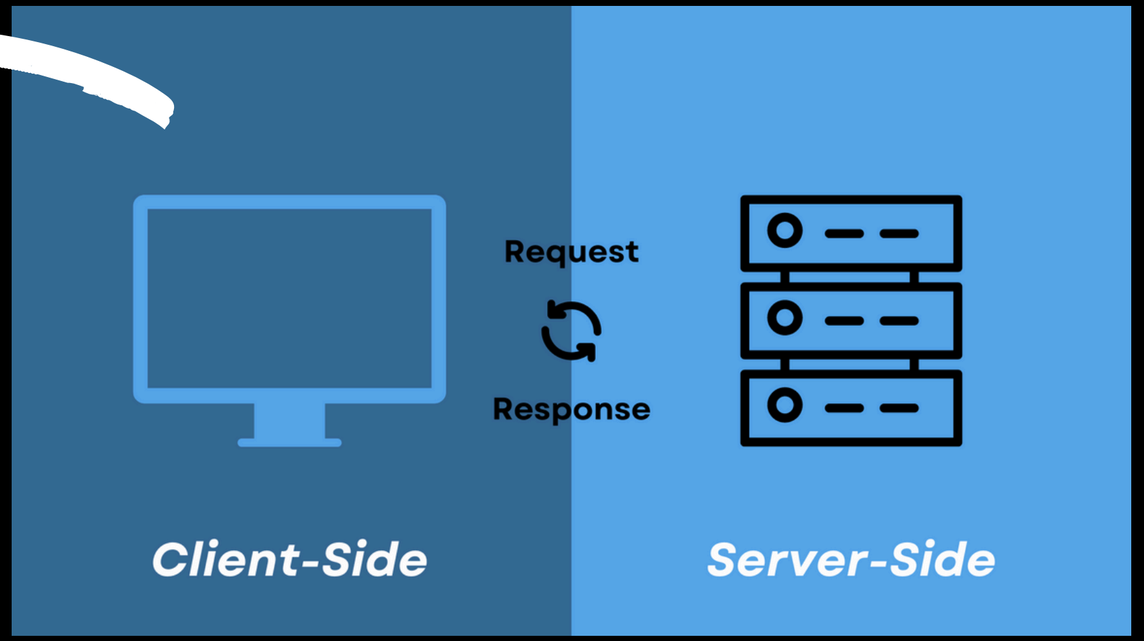
GraphQL Rendering



loading...


HTML


fast


Request
Response
Client-Side
Server-Side

Scalability

N+1 problem

Optimizing with DataLoader for Batch Loading

N+1 problem

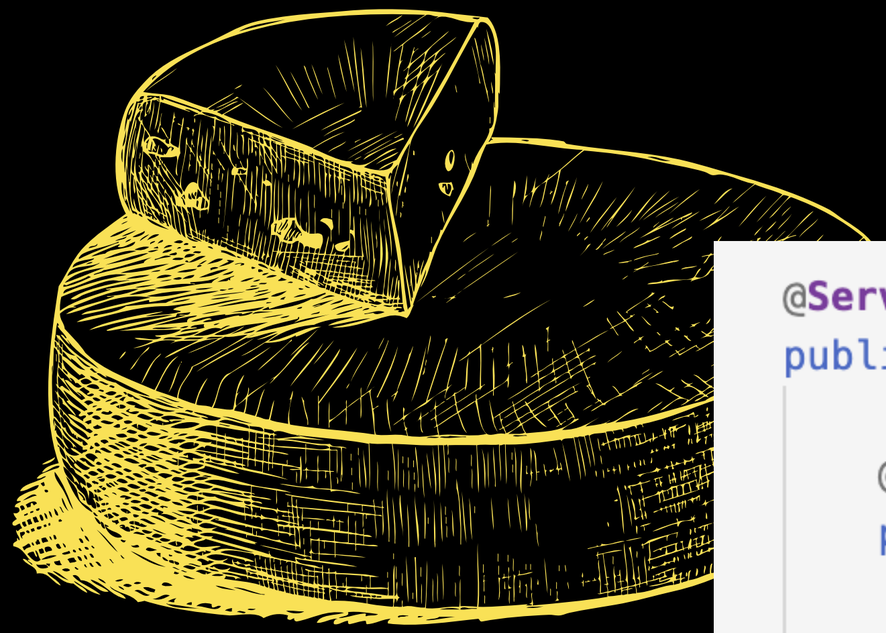
```
type Author {
  id: ID!
  name: String!
  bio: String
}

type Post {
  id: ID!
  title: String!
  content: String
  author: Author  # Reference to the Author type, creating a potential N+1 problem
}
```

<1 query to fetch the 10 posts>
<10 additional queries (1 per post) to fetch each associated author.>

```java
public CompletableFuture<List<Author>> loadAuthors(List<Long> authorIds) {
    return CompletableFuture.supplyAsync(() -> {
        // Fetch all authors in a single batch based on authorIds
        List<Author> authors = authorRepository.findAllById(authorIds);

        // Map authors to their IDs to make it easier to match
        Map<Long, Author> authorMap = authors.stream()
                                             .collect(Collectors.toMap(Author::getId, author -> author));

        // Return the authors in the order of the requested authorIds
        return authorIds.stream()
                        .map(authorMap::get)
                        .collect(Collectors.toList());
    });
}
```

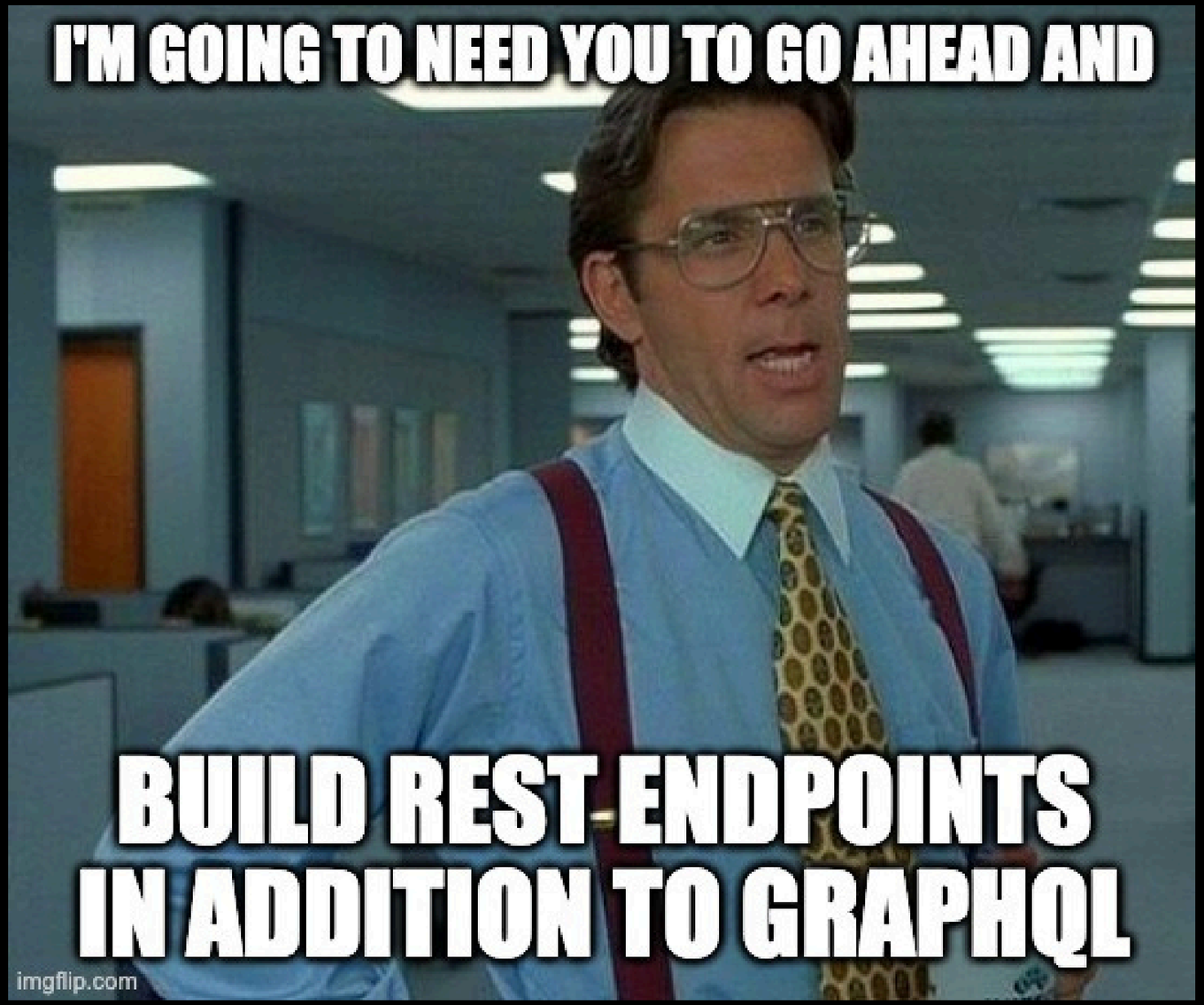
```java
private AuthorBatchLoader authorBatchLoader;

public CompletableFuture<Author> getAuthor(Post post) {
    // Use the batch loader to fetch the author of the post in a batched request
    return authorBatchLoader.loadAuthors(Collections.singletonList(post.getAuthorId()))
                            .thenApply(authors -> authors.get(0)); // Get the single author from the list
}
```

Scaling with Pagination



```java
@Service
public class ItemService {

    @Autowired
    private ItemRepository itemRepository;

    public List<Item> getItems(int offset, int limit) {
        Pageable pageable = PageRequest.of(offset, limit);
```

Docker and Containerisation

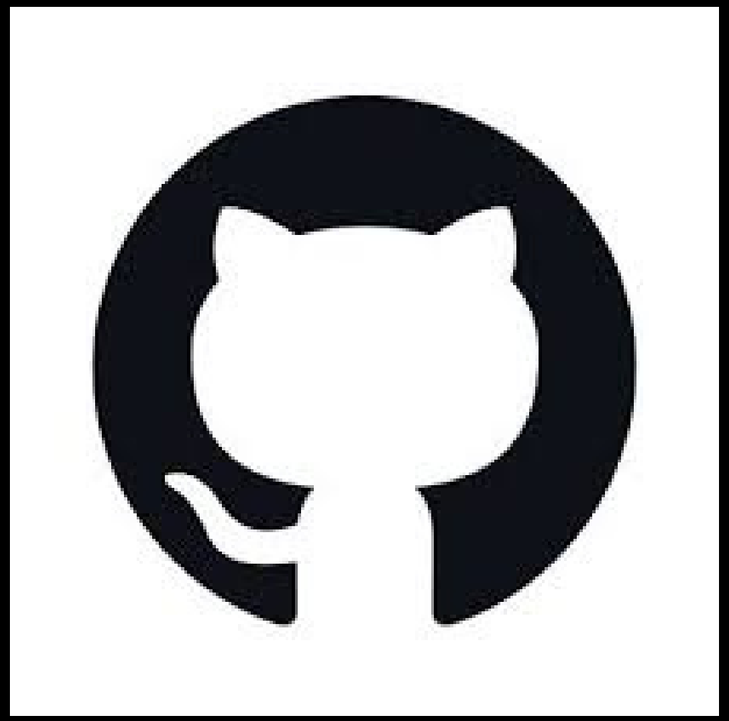

I'M GOING TO NEED YOU TO GO AHEAD AND
BUILD REST ENDPOINTS
IN ADDITION TO GRAPHQL
imgflip.com

Bonus

UserController.java > ...

```java
@RestController
@RequestMapping("/api")
public class UserController {

    @Autowired
    private GraphQLService graphQLService;

    @GetMapping("/users")
    public ResponseEntity<Object> getUsers() {
        Object users = graphQLService.fetchUsersWithGraphQL();
        return ResponseEntity.ok(users); // Return data as JSON response
    }
}
```

Another Bonus



one endpoint

improved performance

schemas


GRAPHQL IS A DECENT ALTERNATIVE
BUT IT WON'T REPLACE REST!... AT LEAST FOR NOW

Dev container set-up

github.com/sebivenlo/esd-2024-graphql-in-enterprise-applications/tree/start