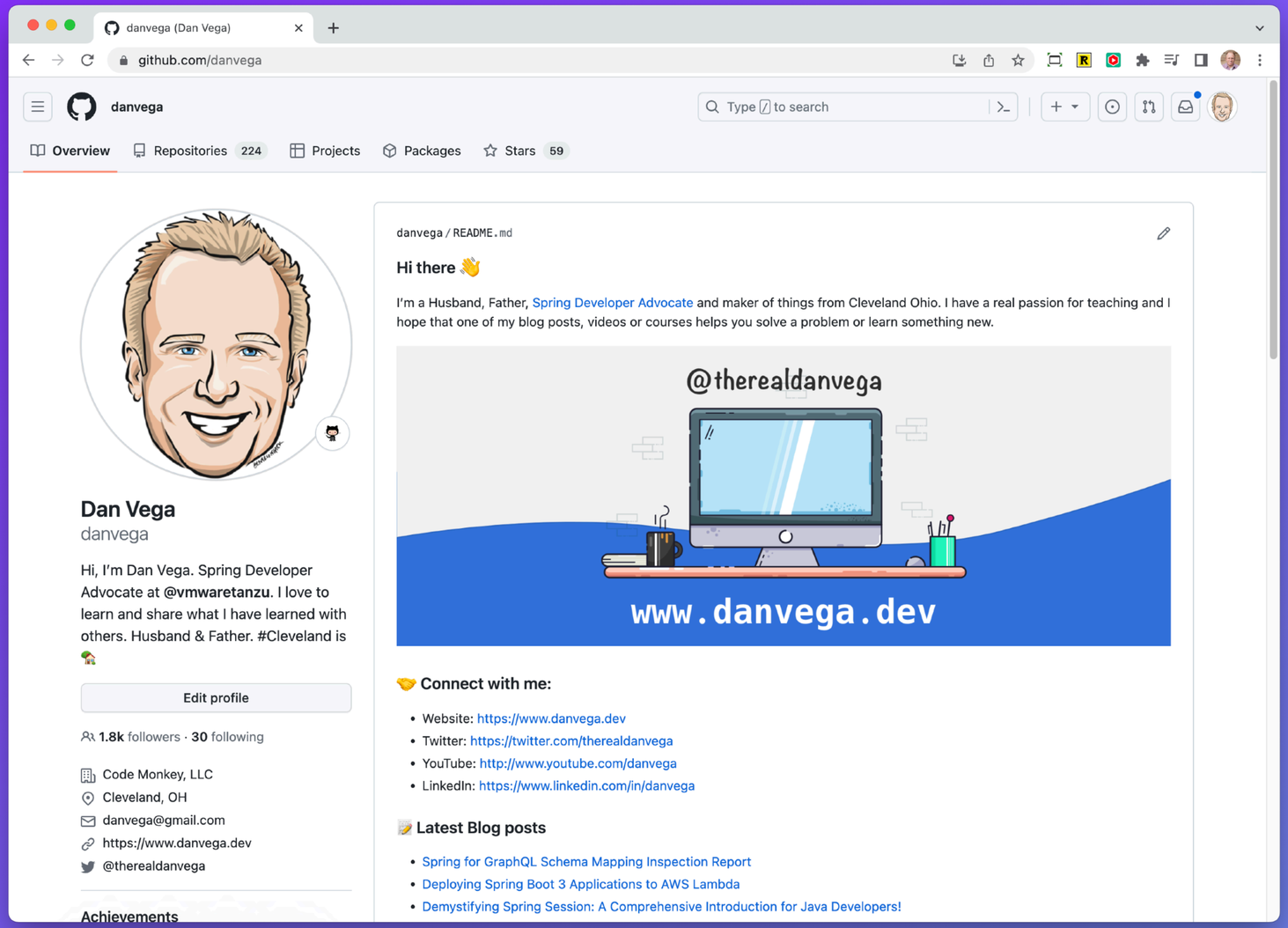
DOES YOUR API NEED A REST?

Check out GraphQL

Dan Vega - Spring Developer Advocate @ Broadcom



https://github.com/danvega/graphql-store

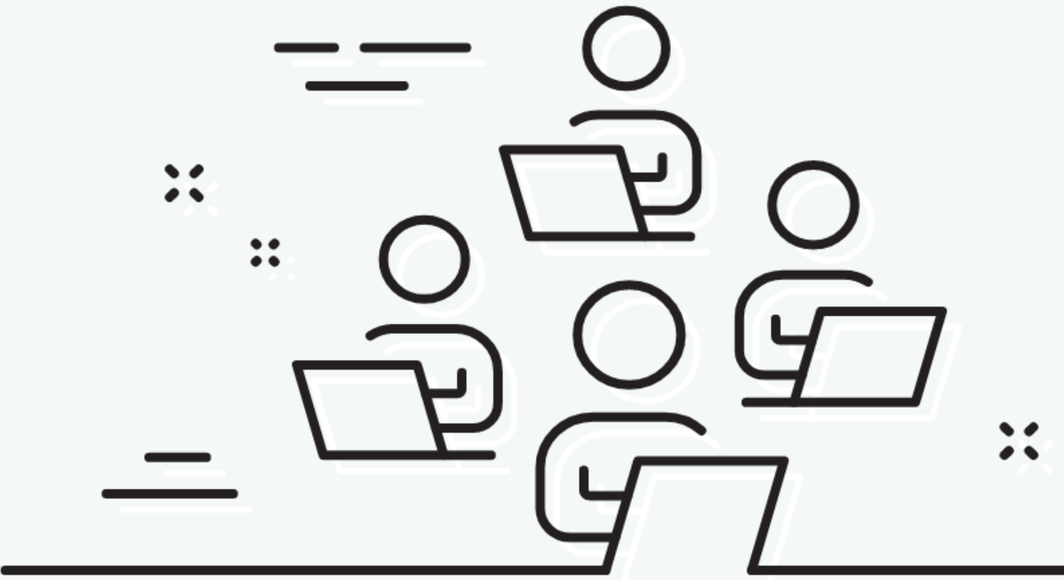
ABOUT ME

Learn more at danvega.dev

- 👨‍👩‍👧‍👦 Husband & Father
- 🏠 Cleveland
- ☕ Java Champion
- 👨‍💻 Software Development 23 Years
- 🌿 Spring Developer Advocate
- 📝 Content Creator
- 🐦 @therealdanvega
- 🏃‍♂️ Running
- 🏌️‍♂️ Golf

spring

OFFICE HOURS



https://www.springofficehours.io

AGENDA

WHY GRAPHQL

What problems are we trying to solve.

WHAT IS GRAPHQL

A basic introduction to what GraphQL is.

GETTING STARTED WITH GRAPHQL IN SPRING

How to build GraphQL APIs in Spring. (Demo)

FEATURES

Observability, Pagination, Security, Testing, Federation, DGS & More!

Q&A

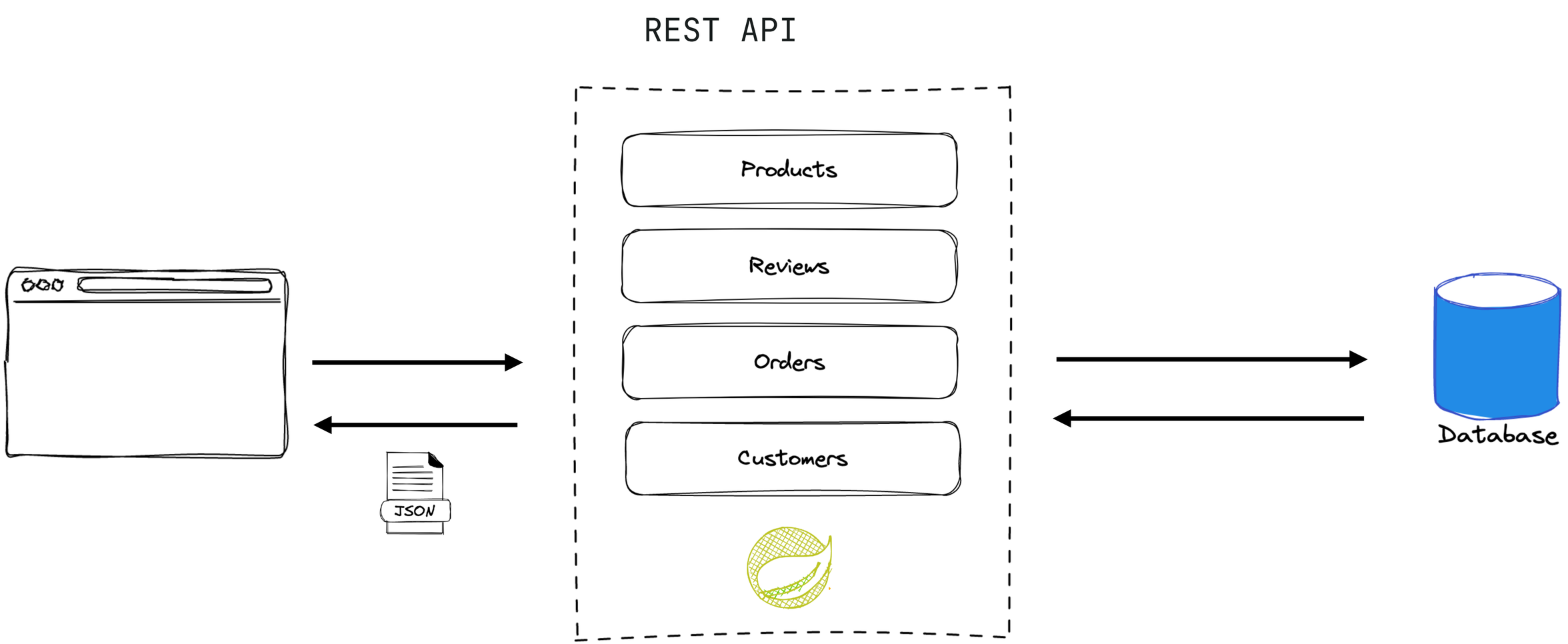
WHY GRAPHQL?

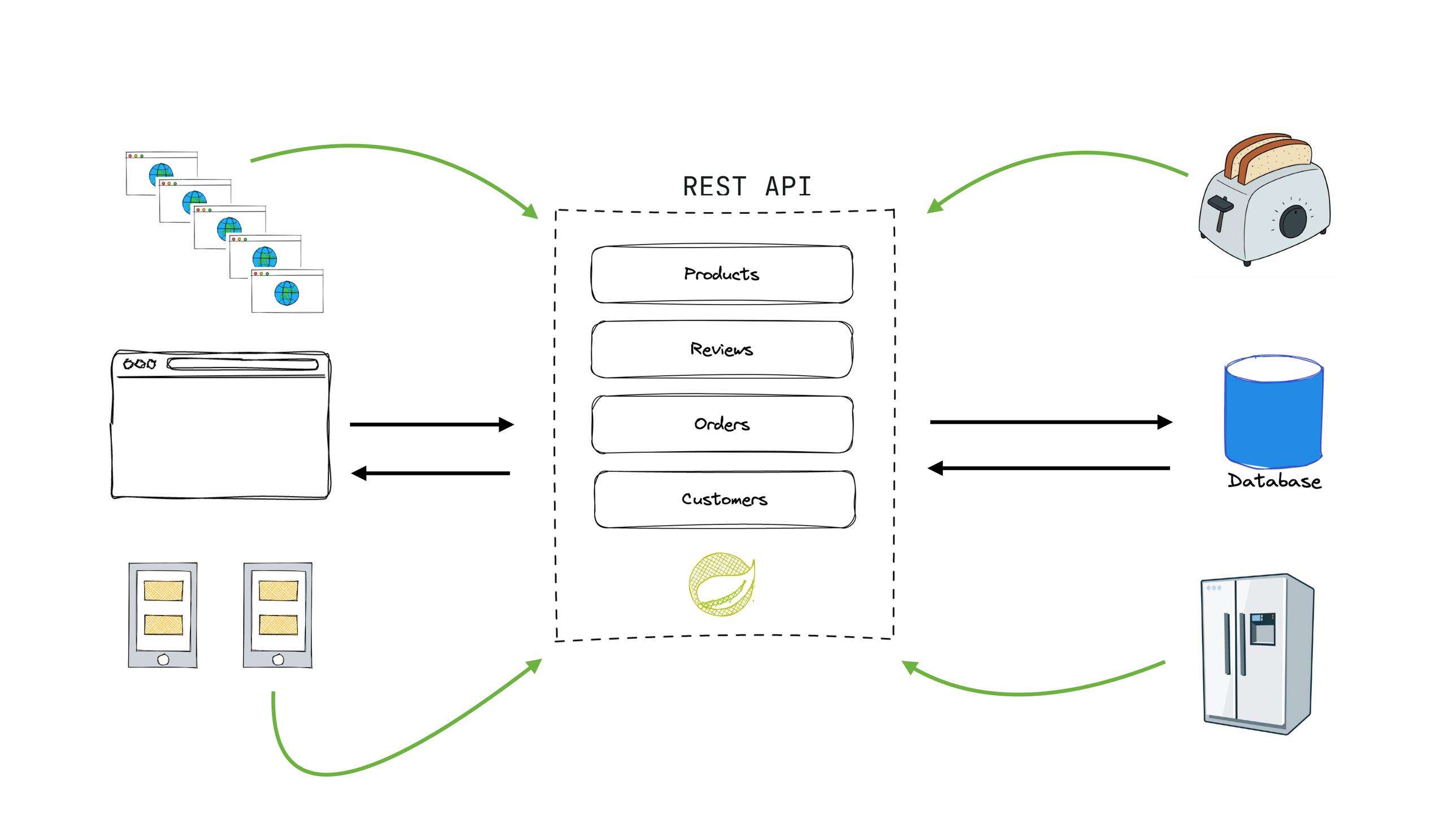
What problems are we trying to solve

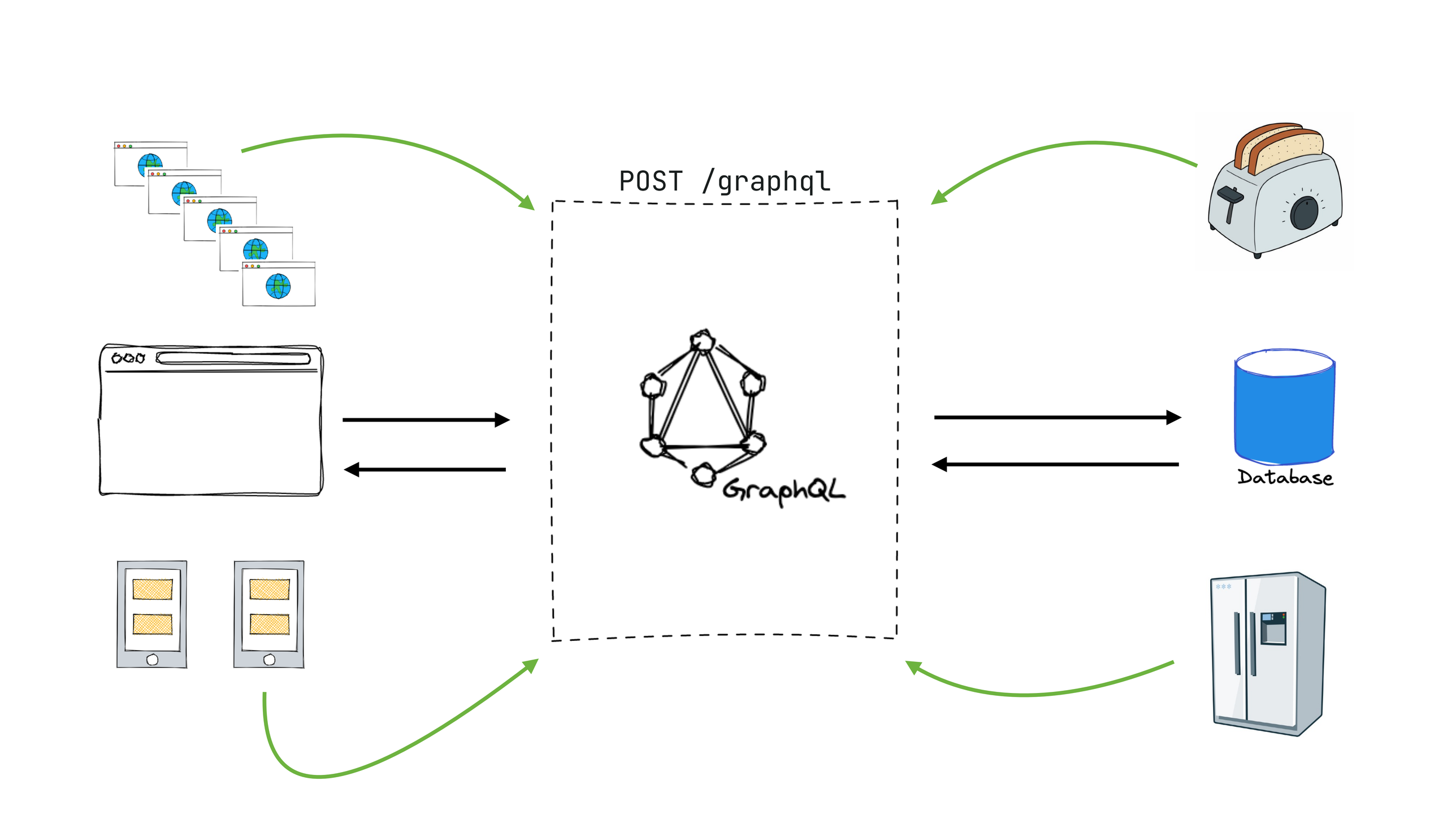
WHY GRAPHQL

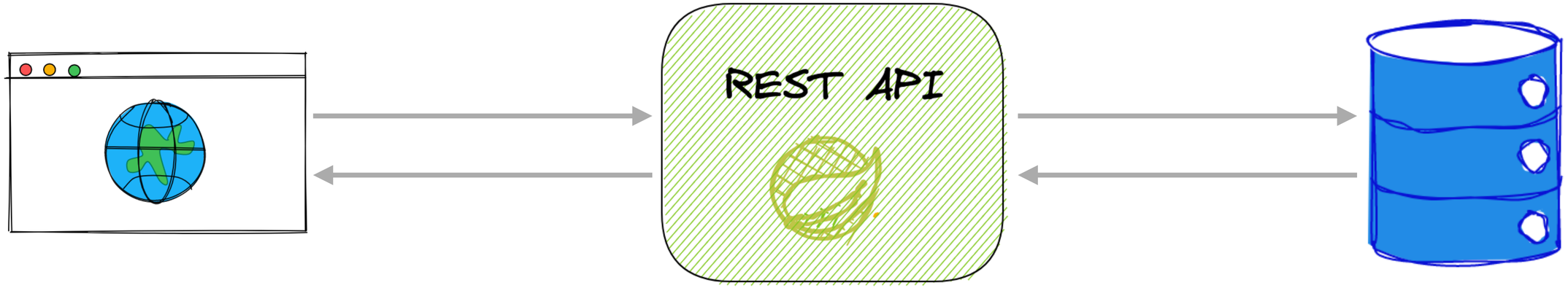
Reasons to make the move to GraphQL

- No more over-fetching
- Multiple Request for multiple resources
- Avoid REST API Explosion of endpoints
- Strongly-typed Schema
  - Self Documenting
  - Developer Tooling
- Avoids API Versioning


REST API
Products
Reviews
Orders
Customers
JSON
Database


REST API
Products
Reviews
Orders
Customers
Database

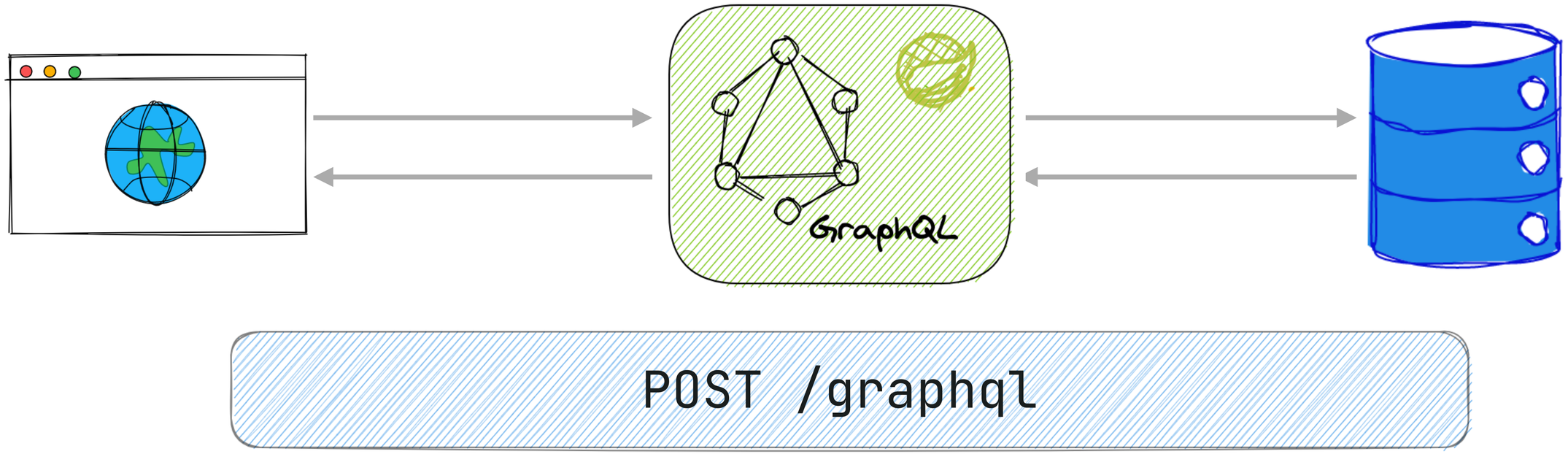

POST /graphql
GraphQL
Database




```
GET /products/123
```

```
GET /products/123/related
```

```
GET /products/123/reviews
```


GraphQL
POST /graphql

AS LONG AS WE ARE TALKING ABOUT REST VS GRAPHQL

HTTP VERBS

STATUS CODES

RESOURCES / PATHS

API VERSIONING

DOCUMENTATION

WHAT IS GRAPHQL

A basic Introduction to what GraphQL is

WHAT IS GRAPHQL

WHAT IS GRAPHQL

GraphQL Foundation

An open and neutral home for the GraphQL community, dedicated to promoting widespread adoption and accelerating development of the surrounding ecosystem

https://graphql.org/foundation/

GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data. GraphQL **isn't tied to any specific database or storage engine** and is instead backed by your existing code and data.

CLIENT + SERVER 🤝

GraphQL

October 2021 Edition

Introduction

This is the specification for GraphQL, a query language and execution engine originally created at Facebook in 2012 for describing the capabilities and requirements of data models for client-server applications. The development of this open standard started in 2015. This specification was licensed under OWFa 1.0 in 2017. The GraphQL Foundation was formed in 2019 as a neutral focal point for organizations who support the GraphQL ecosystem, and the GraphQL Specification Project was established also in 2019 as the Joint Development Foundation Projects, LLC, GraphQL Series.

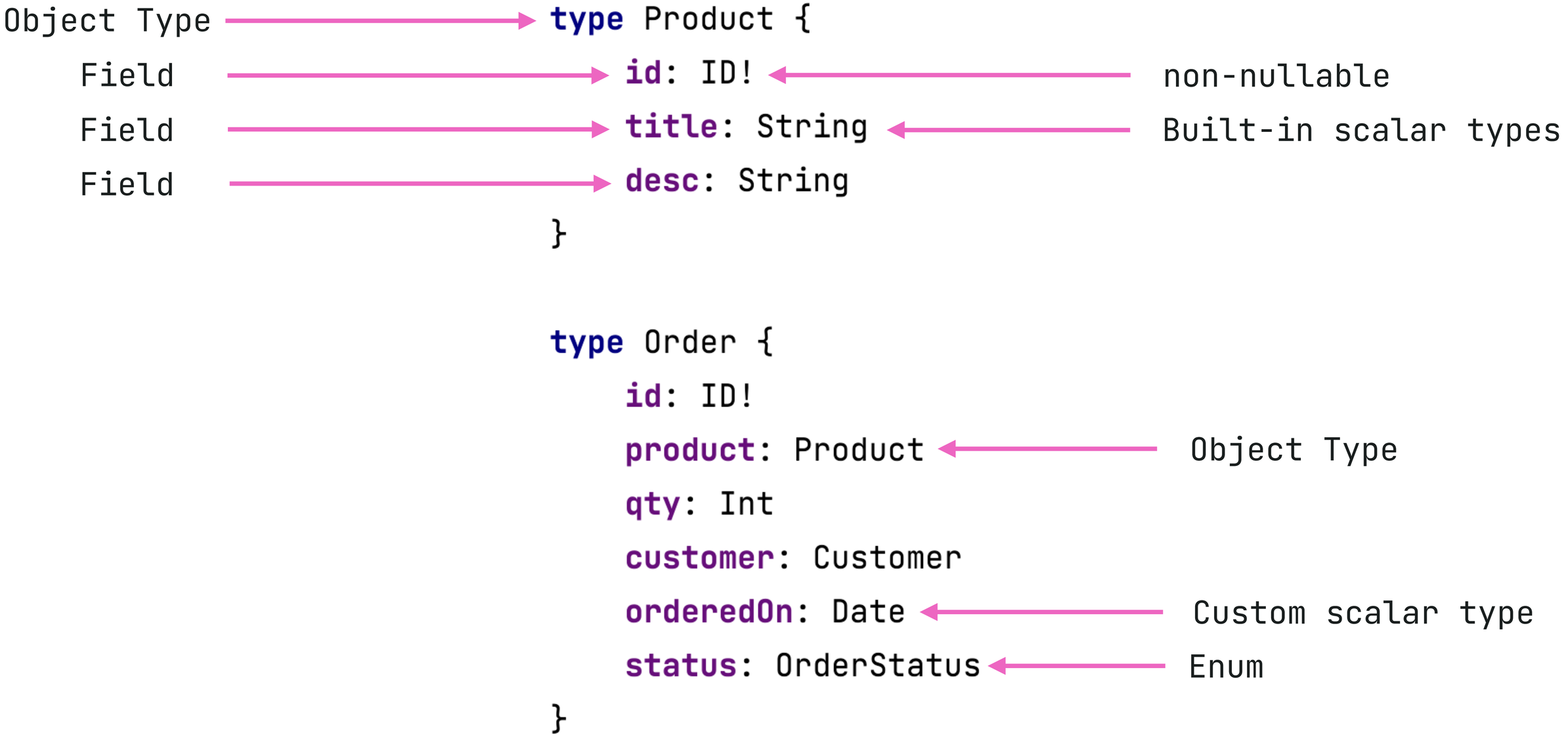
If your organization benefits from GraphQL, please consider becoming a member and helping us to sustain the activities that support the health of our neutral ecosystem.

The GraphQL Specification Project has evolved and may continue to evolve in future editions of this specification. Previous editions of the GraphQL specification can be found at permalinks that match their release tag. The latest working draft release can be found at https://spec.graphql.org/draft.

Copyright notice

TYPE SYSTEM

OBJECT TYPES

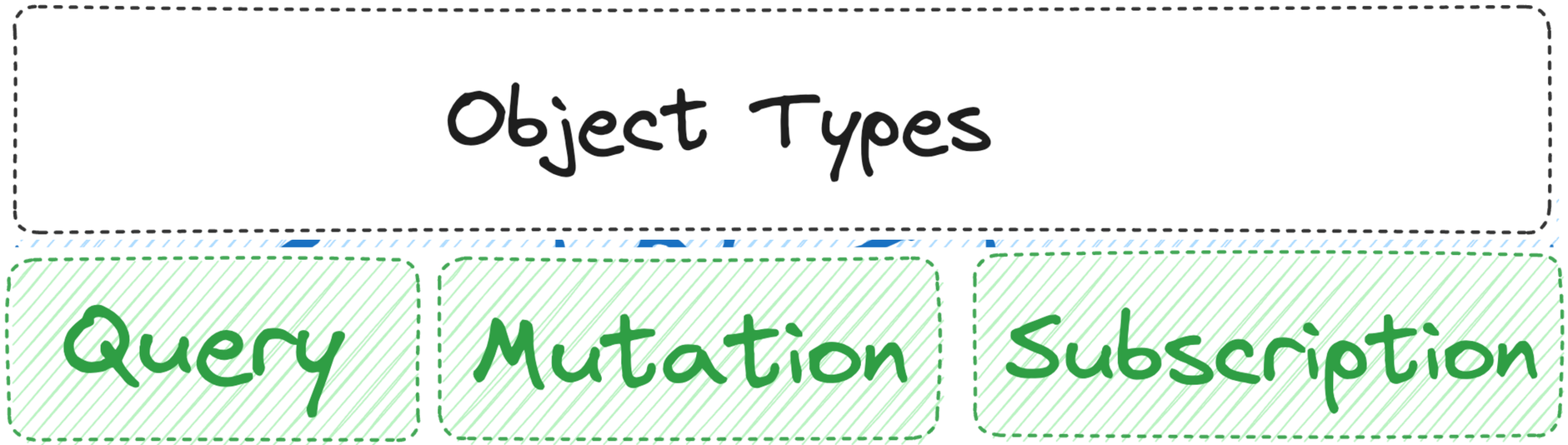


GRAPHQL SCHEMA: SCALAR TYPES

GraphQL comes with a set of default scalar types out of the box:

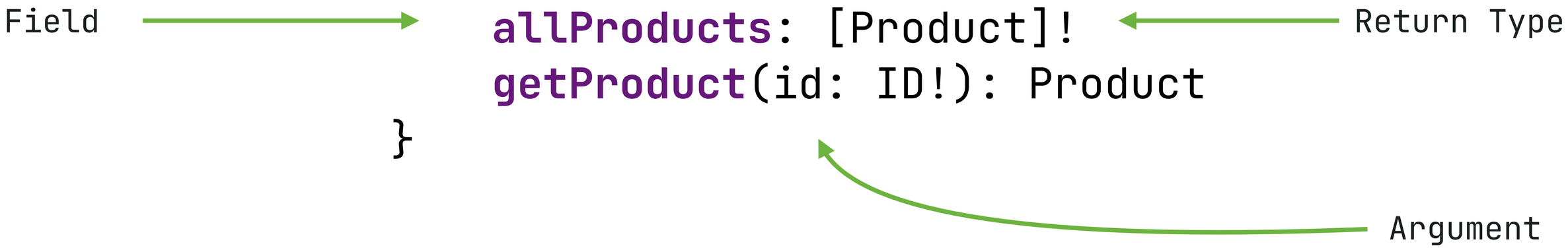
- Int: A signed 32-bit Integer.
- Float: A signed double-precision floating point value.
- String: A UTF-8 character sequence.
- Boolean: true or false.
- ID: The ID scalar type represents a unique identifier, often used to refetch an object or as the key for a cache. The ID type is serialized in the same way as a String; however, defining it as an ID signifies that it is not intended to be human-readable.

OPERATION TYPES



QUERY OPERATION

```
type Product {
    id: ID!
    title: String
    desc: String
}

type Query {
    allProducts: [Product]!
    getProduct(id: ID!): Product
}
```

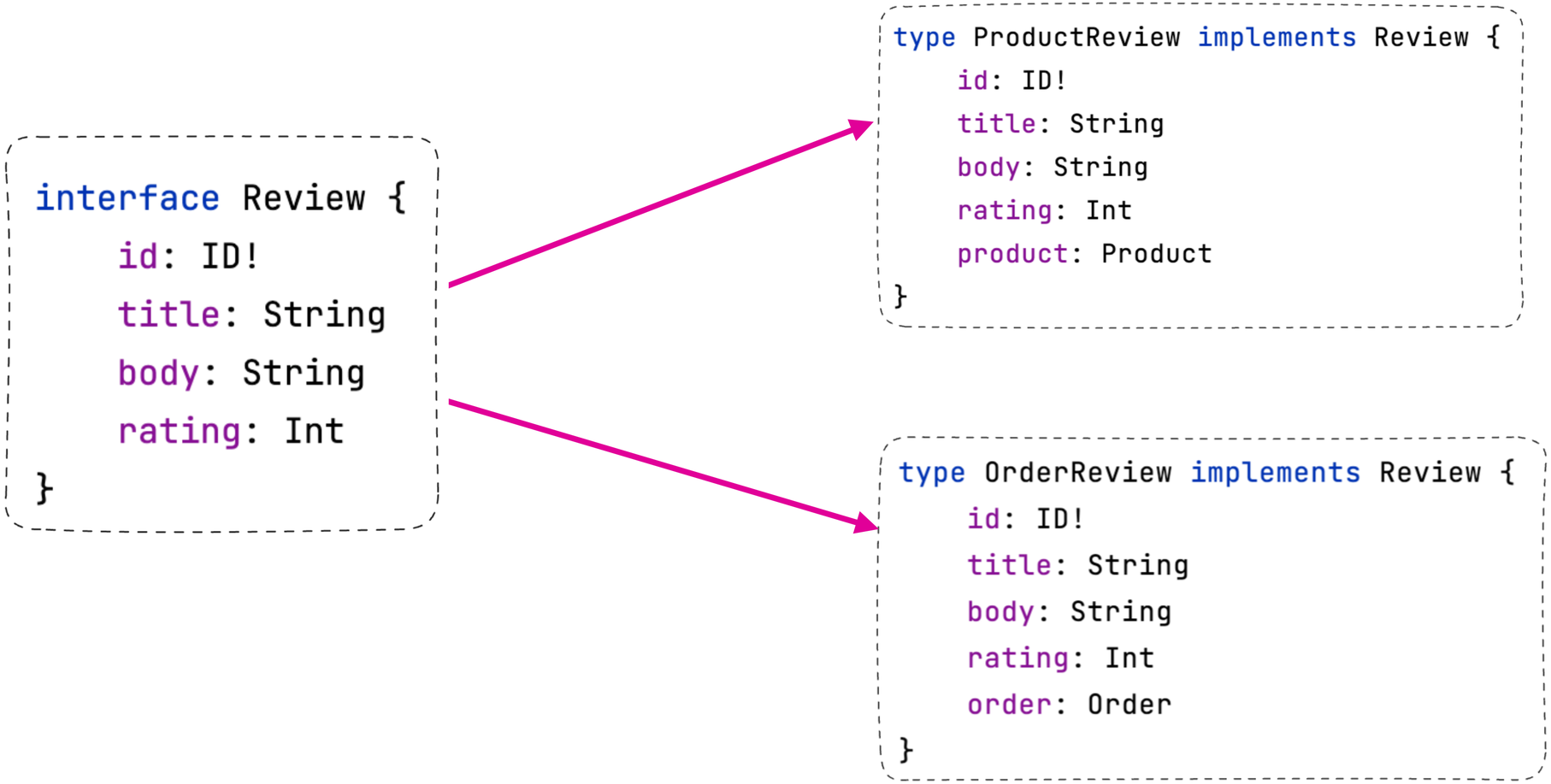


ENUMERATION TYPES

```
type Order {
    id: ID!
    product: Product
    qty: Int
    customer: Customer
    orderedOn: Date
    status: OrderStatus
}
```

```
enum OrderStatus {
    CANCELED,
    PENDING,
    ORDERED,
    SHIPPED,
    DELIVERED
}
```

INTERFACES


interface Review {
id: ID!
title: String
body: String
rating: Int
}
type ProductReview implements Review {
id: ID!
title: String
body: String
rating: Int
product: Product
}
type OrderReview implements Review {
id: ID!
title: String
body: String
rating: Int
order: Order
}

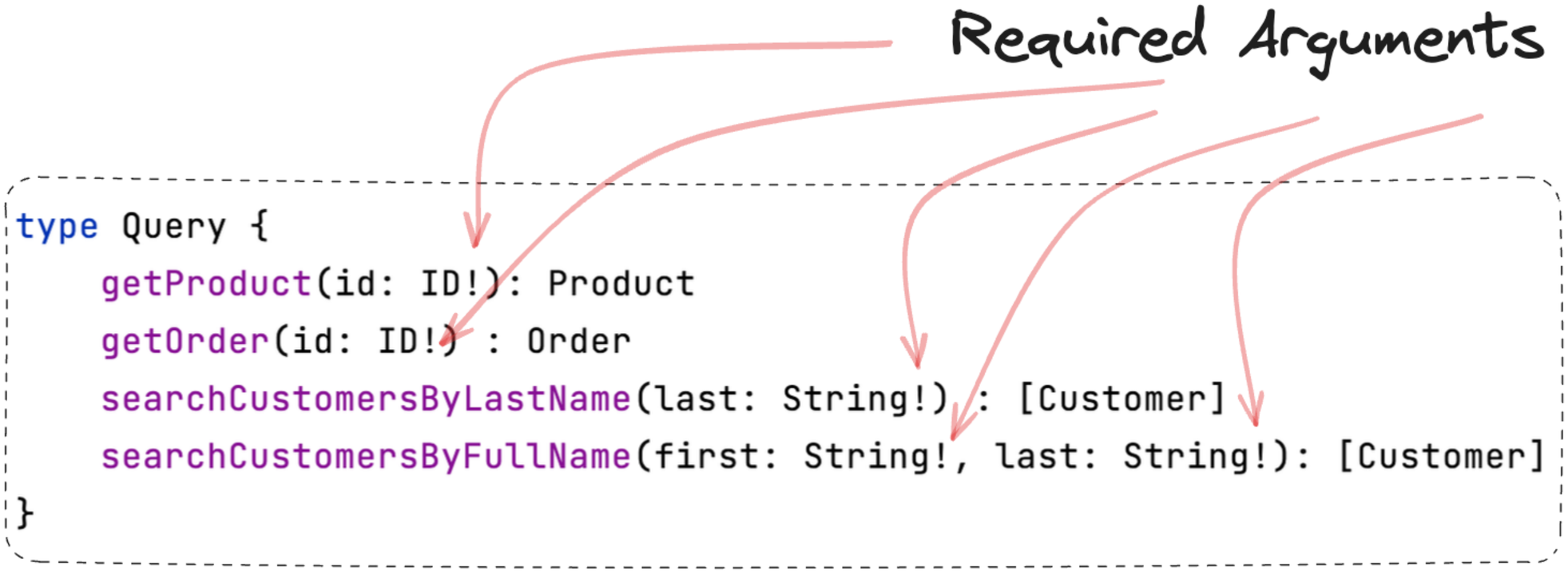
UNIONS

```
union SearchItem = Product | Customer

type Product {
    id: ID!
    title: String
    desc: String
    orders: [Order]
}

type Customer {
    id: ID!
    firstName: String
    lastName: String
    email: String
    orders: [Order]
}
```

ARGUMENTS

```
type Query {
    getProduct(id: ID): Product
    getOrder(id: ID) : Order
    searchCustomersByLastName(last: String) : [Customer]
    searchCustomersByFullName(first: String, last: String): [Customer]
}
```

ARGUMENTS


Required Arguments
type Query {
getProduct(id: ID!): Product
getOrder(id: ID!) : Order
searchCustomersByLastName(last: String!) : [Customer]
searchCustomersByFullName(first: String!, last: String!): [Customer]
}

ARGUMENTS

```
type Query {
    findCustomerById(id: ID!): Customer
}
```

```
type Customer {
    id: ID!
    firstName: String
    lastName: String
    email: String
    orders(first:Int): [Order]
}
```

```
type Order {
    id: ID!
    product: Product
    qty: Int
    customer: Customer
    orderedOn: Date
    status: OrderStatus
}
```

```
query {
  findCustomerById(id: 99) {
    firstName
    lastName
    email
    orders(first: 5) {
      id
      product {
        title
      }
      qty
      orderedOn
      status
    }
  }
}
```

INPUT TYPES

```
type Mutation {
    createProduct(title: String, desc: String) : Product
}
```

```
type Mutation {
    createProduct(
}

type Product {
    id: ID!
    title: String
    desc: String
}
```

```
type Mutation {
    createProduct(product: ProductInput) : Product
}

input ProductInput {
    title: String
    desc: String
}
```

VARIABLES

```
query {
  findCustomerById(id: 99) {
    firstName
    lastName
    email
    orders(first: 5) {
      id
      product {
        title
      }
      qty
      orderedOn
      status
    }
  }
}
```

```
query CustomerDetails($customerID:ID!) {
  findCustomerById(id: $customerID) {
    firstName
    lastName
    email
    orders(first: 5) {
      id
      product {
        title
      }
      qty
      orderedOn
      status
    }
  }
}
```

Variables Headers

```
{
  "customerID": 99
}
```

ERROR HANDLING

```
{
  "errors": [
    {
      "message": "Product with id 99 not found.",
      "locations": [
        {
          "line": 2,
          "column": 3
        }
      ],
      "path": [
        "getProduct"
      ]
    }
  ],
  "data": {
    "getProduct": null
  }
}
```


200 OK STATUS

GRAPHIQL DEMO

GETTING STARTED WITH GRAPHQL IN SPRING

How to build GraphQL APIs in Spring

GRAPHQL JAVA

SPRING FOR GRAPHQL

GRAPHQL JAVA

Implementation of the GraphQL Spec in Java

- Facebook Open Sourced GraphQL in 2015
- Started mid 2015
  - Andreas Marek, Software Architect at Atlassian
  - Bootiful Podcast with Josh Long
- Pure Execution Engine: No HTTP or IO. No high level abstractions
- Active on Github
- Adds Custom Scalar Types
- https://www.graphql-java.com/

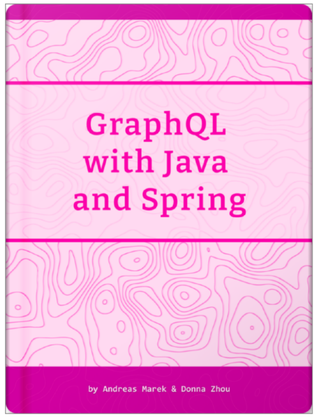
GraphQL with Java and... by Ar

leanpub.com/graphql-java/

Incognito

Books Bundles Courses Tracks Search Leanpub

GraphQL with Java and Spring



LAST UPDATED ON 2023-05-14

Andreas Marek and Donna Zhou

Learn first-hand from the founder of GraphQL Java and co-author of Spring for GraphQL how to develop GraphQL services in Java.

See the Table of Contents below for a full chapter listing.

$40.00 MINIMUM PRICE

$40.00 SUGGESTED PRICE

YOU PAY

$40.00

AUTHORS EARN

$32.00

YOU PAY IN US $

$40.00

EU customers: Price excludes VAT. VAT is added during checkout.

Add Ebook to Cart

Add to Wish List

Table Of Contents

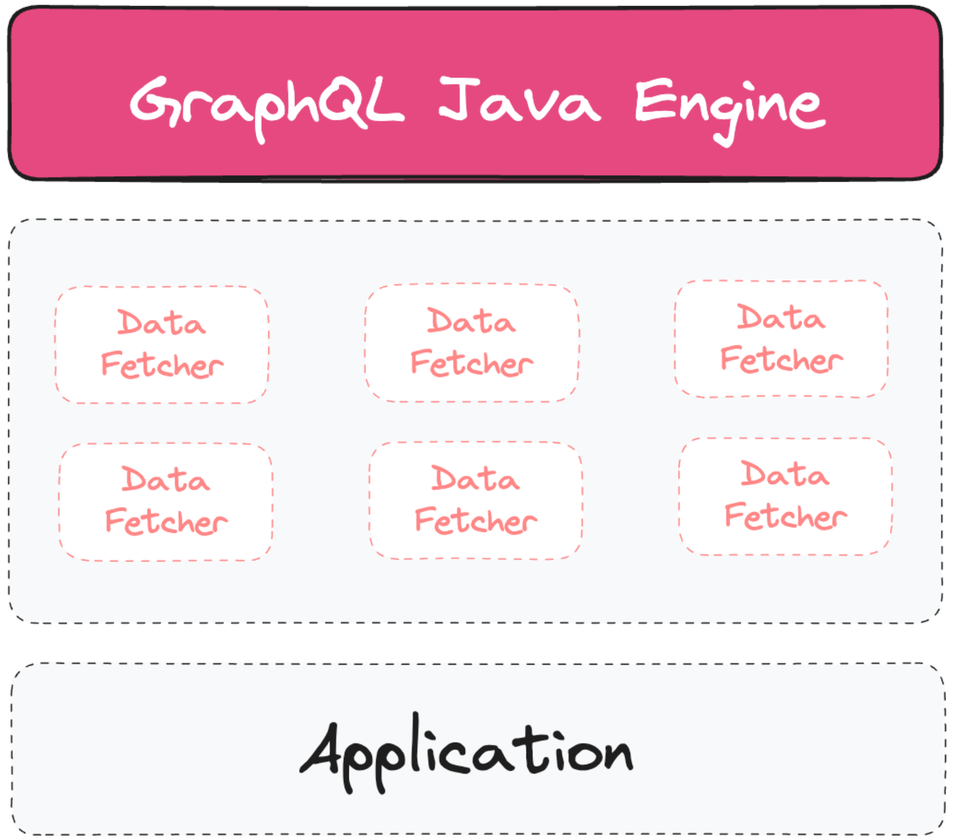
60 DAYS GUARANTEE | ENGLISH | PDF | EPUB

About the Book

Learn first-hand from the founder of GraphQL Java and co-author of Spring for GraphQL how to build GraphQL services in Java.

GRAPHQL JAVA EXECUTION ENGINE

```
query {
  findCustomerByLastName(last:"Vega") {
    id
    firstName
    lastName
    email
    orders(first: 5) {
      id
      product {
        id
        title
        desc
      }
      qty
      orderedOn
      status
    }
  }
}
```



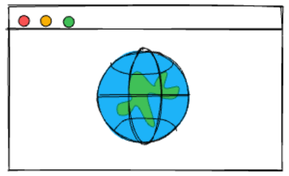
WHAT IS SPRING FOR GRAPHQL

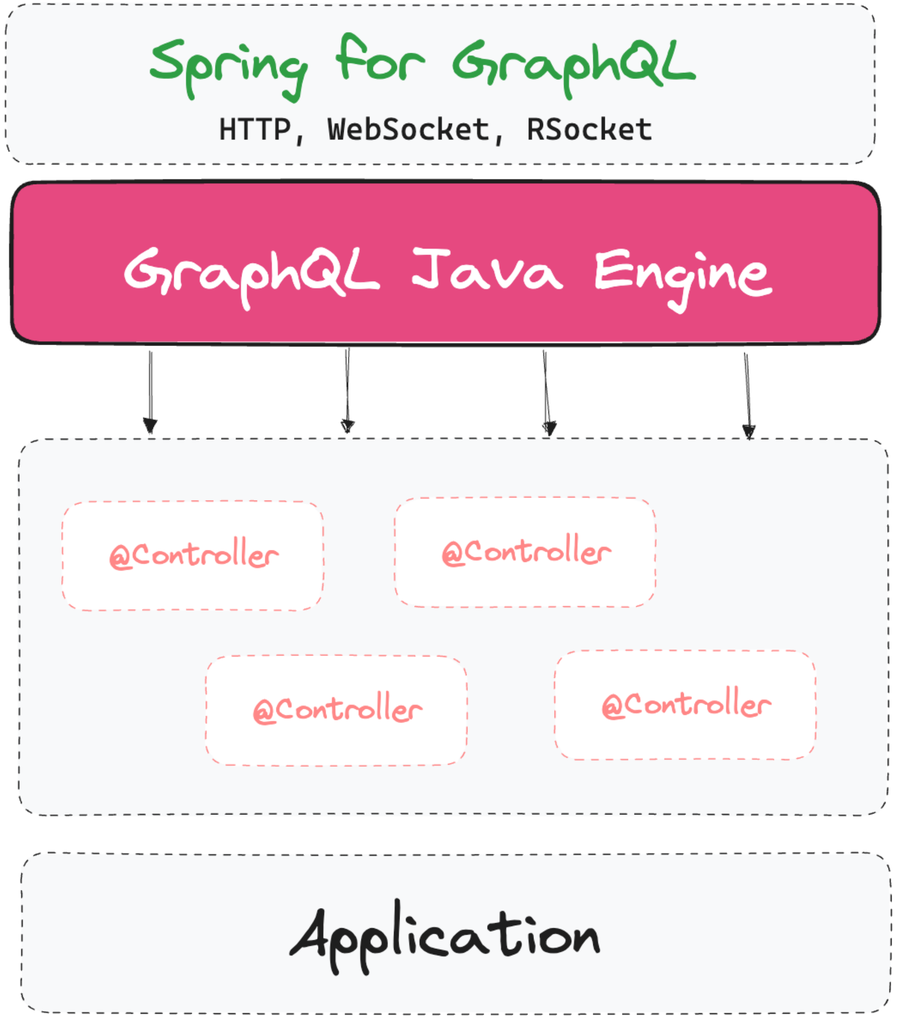
Spring for GraphQL provides support for Spring applications built on GraphQL Java

- Joint collaboration between both teams
- GraphQL Java is “limited” on purpose
- Spring Programming model
- Spring Boot Starter
- Requirements
  - 1.1.x (Nov 2022)
    - Spring Boot 3.0 / JDK 17 / GraphQL Java 19
  - 1.2.x (May 2023)
    - Spring Boot 3.1 / JDK 17 / GraphQL Java 20
  - **1.3.x (May 2024)**
    - Spring Boot 3.2 / JDK 17 / GraphQL Java 22

SPRING FOR GRAPHQL TRANSPORT LAYER

```
query {
  findCustomerByLastName(last:"Vega") {
    id
    firstName
    lastName
    email
    orders(first: 5) {
      id
      product {
        id
        title
        desc
      }
      qty
      orderedOn
      status
    }
  }
}
```





SPRING FOR GRAPHQL DEMO

https://github.com/danvega/graphql-store

start-here branch

FEATURES

Spring for GraphQL Features

OBSERVABILITY

Insights into how your APIs are performing

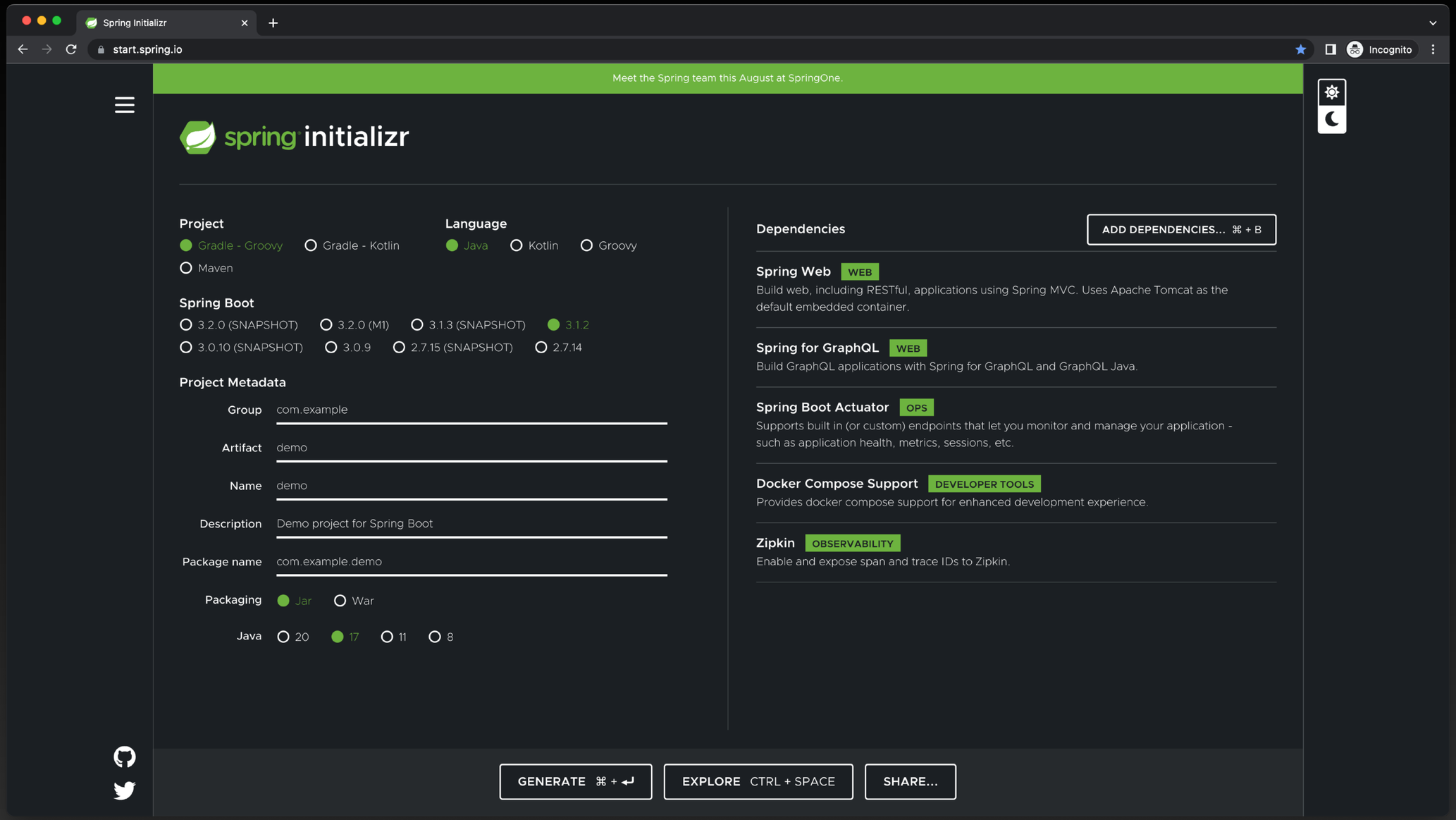
OBSERVABILITY

Spring Framework 6 & Spring Boot 3

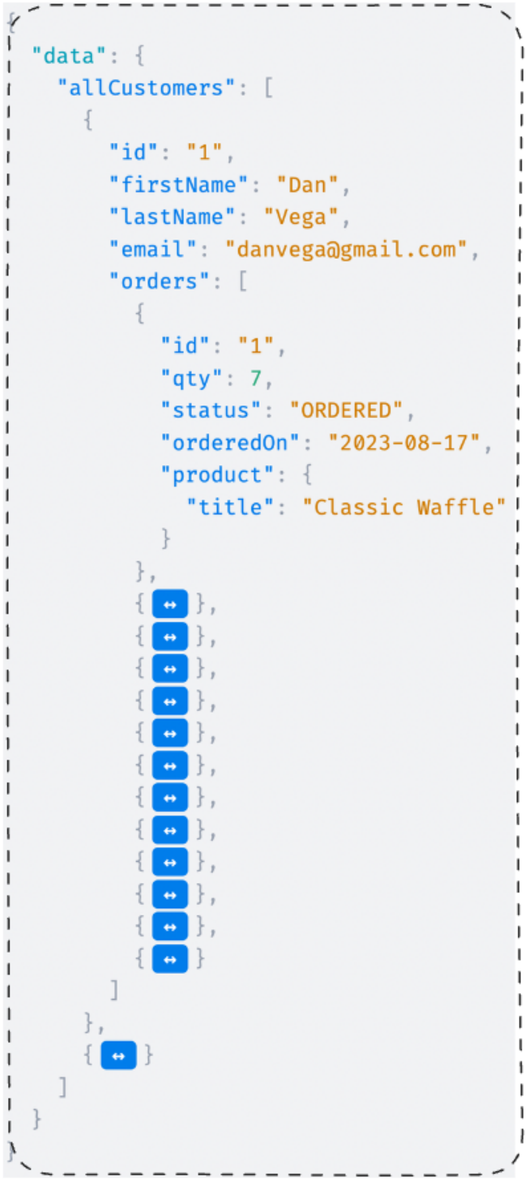
Observability is the ability to observe the internal state of a running system from the outside. It consists of the three pillars **logging**, **metrics** and **traces**.

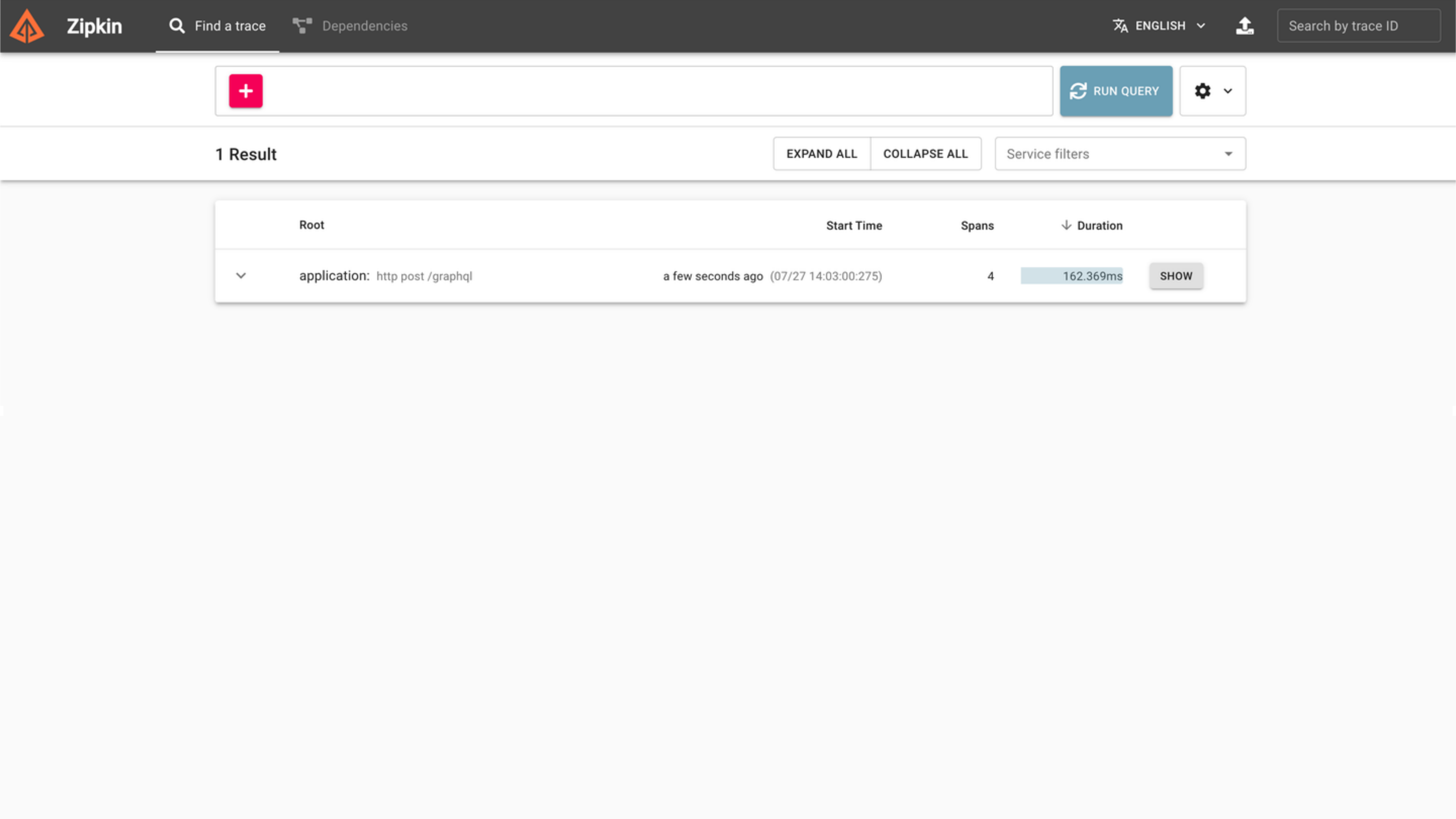
Spring projects now have their own, built-in instrumentation for metrics and traces based on the new Observation API from Micrometer

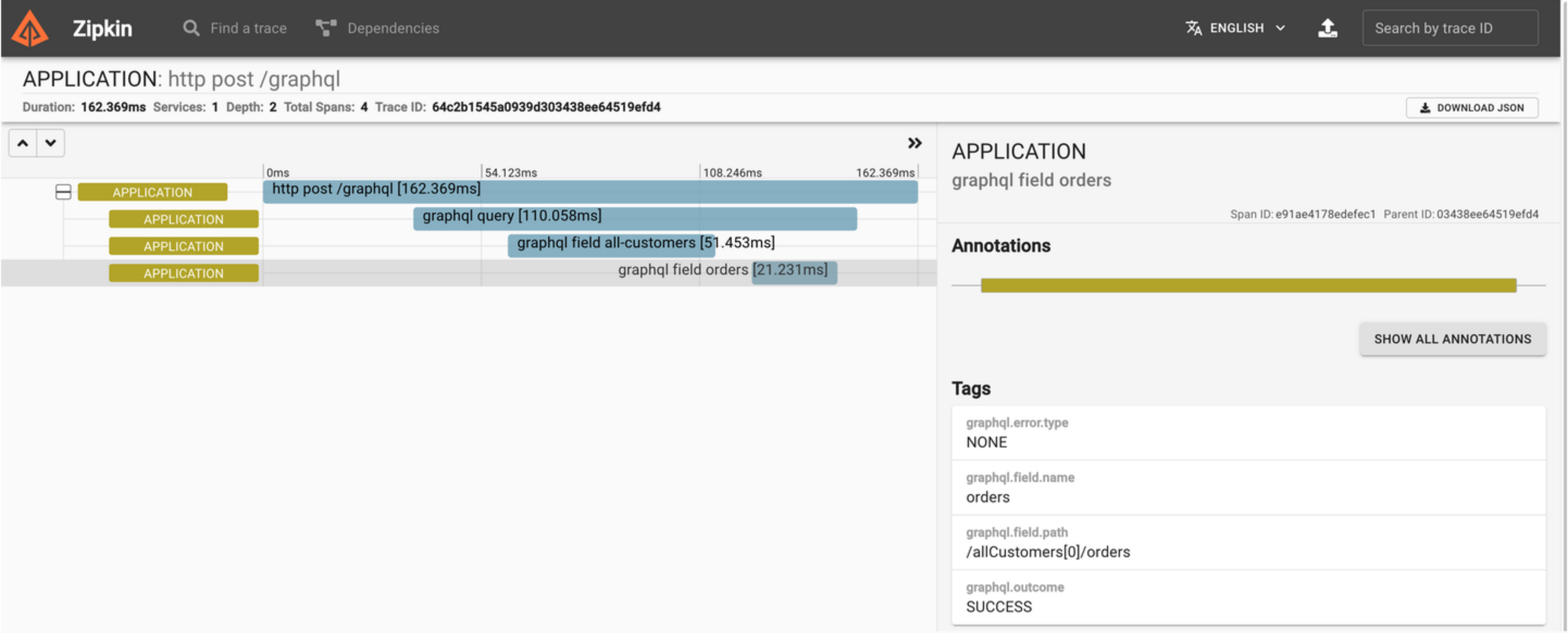
GraphQL is a good use case for Observability in general, as the GraphQL engine can fan out data fetching across REST APIs, data stores, caches, and more.


Spring Initializr
start.spring.io
Incognito
Meet the Spring team this August at SpringOne.
spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.2.0 (SNAPSHOT)
3.2.0 (M1)
3.1.3 (SNAPSHOT)
3.1.2
3.0.10 (SNAPSHOT)
3.0.9
2.7.15 (SNAPSHOT)
2.7.14
Project Metadata
Group
com.example
Artifact
demo
Name
demo
Description
Demo project for Spring Boot
Package name
com.example.demo
Packaging
Jar
War
Java
20
17
11
8
Dependencies
ADD DEPENDENCIES... ⌘ + B
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring for GraphQL
WEB
Build GraphQL applications with Spring for GraphQL and GraphQL Java.
Spring Boot Actuator
OPS
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.
Docker Compose Support
DEVELOPER TOOLS
Provides docker compose support for enhanced development experience.
Zipkin
OBSERVABILITY
Enable and expose span and trace IDs to Zipkin.
GENERATE ⌘ + ↵
EXPLORE CTRL + SPACE
SHARE...

```
query {
  allCustomers {
    id
    firstName
    lastName
    email
    orders {
      id
      qty
      status
      orderedOn
      product {
        title
      }
    }
  }
}
```




Zipkin
Find a trace
Dependencies
ENGLISH
Search by trace ID
RUN QUERY
1 Result
EXPAND ALL
COLLAPSE ALL
Service filters
Root
Start Time
Spans
Duration
application: http post /graphql
a few seconds ago (07/27 14:03:00:275)
4
162.369ms
SHOW


Zipkin
Find a trace
Dependencies
ENGLISH
Search by trace ID
APPLICATION: http post /graphql
Duration: 162.369ms Services: 1 Depth: 2 Total Spans: 4 Trace ID: 64c2b1545a0939d303438ee64519efd4
DOWNLOAD JSON
0ms
54.123ms
108.246ms
162.369ms
APPLICATION
http post /graphql [162.369ms]
APPLICATION
graphql query [110.058ms]
APPLICATION
graphql field all-customers [51.453ms]
APPLICATION
graphql field orders [21.231ms]
APPLICATION
graphql field orders
Span ID: e91ae4178edefec1 Parent ID: 03438ee64519efd4
Annotations
SHOW ALL ANNOTATIONS
Tags
graphql.error.type
NONE
graphql.field.name
orders
graphql.field.path
/allCustomers[0]/orders
graphql.outcome
SUCCESS

PAGINATION

Pagination support using the Cursor Connection Specification

PAGINATION

Spring for GraphQL 1.2 adds support for pagination

The GraphQL Cursor Connection specification defines a way to navigate large result sets by returning a subset of items at a time where each item is paired with a cursor that clients can use to request more items before or after the referenced item.

The specification calls the pattern “Connections”

A schema type with a name that ends in Connection is a *Connection Type* that represents a paginated result set.

All *~Connection* types contain an “edges” field where *~Edge* type pairs the actual item with a cursor, as well as a “pageInfo” field with boolean flags to indicate if there are more items forward and backward.

Spring for GraphQL provides *ConnectionTypeDefinitionConfigurer* to add these types on startup.

```
type Customer {
    id: ID!
    firstName: String
    lastName: String
    email: String
    orders: [Order]
    paginatedOrders(first: Int,last: Int, before: String,after: String): OrderConnection
}
```

```java
@SchemaMapping
Window<Order> paginatedOrders(Customer customer, ScrollSubrange subrange) {
    ScrollPosition scrollPosition = subrange.position().orElse(ScrollPosition.offset());
    Limit limit = Limit.of(subrange.count().orElse( other: 10));
    Sort sort = Sort.by( ...properties: "orderedOn").descending();
    return orderRepository.findByCustomer(customer, scrollPosition, limit, sort);
}
```

```
public interface OrderRepository extends CrudRepository<Order,Integer> {

    List<Order> findByCustomer(Customer customer);

    Window<Order> findByCustomer(Customer customer, ScrollPosition position, Limit limit, Sort sort);

}
```

```graphql
query {
  allCustomers {
    id
    firstName
    lastName
    email
    paginatedOrders(first:3){
      pageInfo {
        hasNextPage
        hasPreviousPage
        startCursor
        endCursor
      }
      edges {
        node {
          id
          qty
          orderedOn
          status
          product {
            title
            desc
          }
        }
      }
    }
  }
}
```


```json
{
  "data": {
    "allCustomers": [
      {
        "id": "1",
        "firstName": "Dan",
        "lastName": "Vega",
        "email": "danvega@gmail.com",
        "paginatedOrders": {
          "pageInfo": {
            "hasNextPage": true,
            "hasPreviousPage": true,
            "startCursor": "T18x",
            "endCursor": "T18z"
          },
          "edges": [
            {
              "node": {
                "id": "2",
                "qty": 6,
                "orderedOn": "2023-07-27",
                "status": "DELIVERED",
                "product": {
                  "title": "Chocolate Chip Waffle",
                  "desc": "Sweet Cream Waffle covered in Chocolate Chips"
                }
              }
            },
            {
              "node": { ↔ }
            },
            {
              "node": { ↔ }
            }
          ]
        }
      }
    ]
  }
}
```

SECURITY

Securing GraphQL APIs

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true)
public class SecurityConfig {

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .authorizeHttpRequests( auth -> {
                    auth.anyRequest().authenticated();
                })
                .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(AbstractHttpConfigurer::disable)
                .httpBasic(Customizer.withDefaults())
                .build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        var user = User.withUsername("user")
                .password("{noop}password")
                .roles("USER")
                .build();

        var admin = User.withUsername("admin")
                .password("{noop}password")
                .roles("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user,admin);
    }

}
```

```java
@QueryMapping
public List<Product> allProducts() { return repository.findAll(); }

@QueryMapping
public Optional<Product> getProduct(@Argument Integer id) {
    return repository.findById(id);
}

@Secured("ROLE_ADMIN")
@MutationMapping
public Product createProduct(@Argument ProductInput productInput) {
    return repository.save(new Product(productInput.title(), productInput.desc()));
}
```

```
mutation {
  createProduct(product:{title:"New Product",
    id
    title
    desc
  }
}
```


```
{
  "errors": [
    {
      "message": "Forbidden",
      "locations": [
        {
          "line": 31,
          "column": 3
        }
      ],
      "path": [
        "createProduct"
      ],
      "extensions": {
        "classification": "FORBIDDEN"
      }
    }
  ],
  "data": {
    "createProduct": null
  }
}
```

```graphql
query {
    allCustomers {
    id
    firstName
    lastName
    email
    paginatedOrders(first:3){
      pageInfo {
        hasNextPage
        hasPreviousPage
        startCursor
        endCursor
      }
      edges {
        node {
          id
          qty
          orderedOn
          status
          product {
            title
            desc
          }
        }
      }
    }
  }
}
```

Variables Headers

```
{
  "Authorization":"Basic dXNlcjpwYXNzd29yZA=="
}
```

GraphiQL


```
{
  "data": {
    "allCustomers": [
      {
        "id": "1",
        "firstName": "Dan",
        "lastName": "Vega",
        "email": "danvega@gmail.com",
        "paginatedOrders": {
          "pageInfo": {
            "hasNextPage": true,
            "hasPreviousPage": true,
            "startCursor": "T18x",
            "endCursor": "T18z"
          },
          "edges": [
            {
              "node": {
                "id": "19",
                "qty": 1,
                "orderedOn": "2023-08-24",
                "status": "PENDING",
                "product": {
                  "title": "Chocolate Chip Waffle",
                  "desc": "Sweet Cream Waffle covered in Chocolate Chips"
                }
              }
            },
            {
              "node": {
                "id": "7",
                "qty": 4,
                "orderedOn": "2023-08-24",
                "status": "CANCELED",
                "product": {
                  "title": "Peanut Butter Chip Waffle",
                  "desc": "Sweet Cream Waffle covered in Peanut Butter Chips"
                }
              }
            },
            {
```

CONTEXT PROPAGATION FROM TRANSPORT

```java
@Service
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

    @PreAuthorize("hasRole('ADMIN')")
    public Product create(Product product) {
        return productRepository.save(product);
    }
}
```

ACCESS TO THE AUTHENTICATED PRINCIPAL

```java
@QueryMapping
public List<Order> allOrders(Principal principal) {
    log.info("Authenticated Principal: " + principal);
    return repository.findAll();
```

```
] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
] c.w.store.controller.OrderController     : Authenticated Principal: UsernamePasswordAuthenticationToken [Principal=org.springframework.security.core.userdetails.User [Username=user, Password=[PROTECTED],
```

TESTING

How to test your GraphQL APIs

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-graphql</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.graphql</groupId>
    <artifactId>spring-graphql-test</artifactId>
    <scope>test</scope>
</dependency>
```

GRAPHQL SLICE TEST

```java
@Import({GraphQlConfig.class})
@GraphQlTest(ProductController.class)
class ProductControllerTest {

    @Autowired
    private GraphQlTester graphQlTester;

    @MockBean
    private ProductRepository productRepository;

    private final List<Product> products = new ArrayList<>();

    @Test
    public void contextLoads() {
        assertNotNull(graphQlTester);
        assertNotNull(productRepository);
    }
```

GRAPHQL TESTER FLUENT API

```java
@Test
void shouldGetAllProductsQueryReturnsAllProducts() {
    // language=GraphQL
    String document = """
        query {
          allProducts {
            id
            title
            desc
          }
        }
    """;

    when(productRepository.findAll()).thenReturn(products);

    graphQlTester.document(document)
            .execute()
            .path("allProducts")
            .entityList(Product.class)
            .hasSize(4);
}
```

src/test/resources/graphql-test/product.graphql

```java
@Test
void shouldReturnValidProductWithDocumentName() {
    when(productRepository.findById(1)).thenReturn(java.util.Optional.ofNullable(products.get(0)));

    graphQlTester.documentName("product") Request<capture of ?>
            .variable( name: "id", value: 1) capture of ?
            .execute() Response
            .path("getProduct") Path
            .entity(Product.class) Entity<Product, capture of ?>
            .satisfies(product -> {
                assertEquals( expected: "Classic Waffle",product.getTitle());
                assertEquals( expected: "Classic Sweet Cream Waffle",product.getDesc());
            });
}
```

ProductControllerTest.java | product.graphql

Default GraphQL Endpoint - http://localhost:8080/graphql

```graphql
query findProduct($id:ID) {
    getProduct(id:$id) {
        id
        title
        desc
    }
}
```

TEST OVER HTTP

```java
@SpringBootTest
@AutoConfigureHttpGraphQlTester
public class ProductControllerHttpTest {

    @Autowired
    private HttpGraphQlTester graphQlTester;
    @Autowired
    private ProductRepository productRepository;

    @Test
    @WithMockUser
    void shouldFindAllProducts() {
        graphQlTester.documentName("products") Request<capture of ?>
                .execute() Response
                .path("allProducts") Path
                .entityList(Product.class) EntityList<Product>
                .hasSize(productRepository.findAll().size());
    }
```

FULL INTEGRATION TEST

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class OrderControllerIntTest {

    private HttpGraphQlTester graphQlTester;
    @Autowired
    private OrderRepository orderRepository;

    @LocalServerPort
    int port;

    @BeforeEach
    void setUp() {
        WebTestClient client = WebTestClient.bindToServer()
                .defaultHeaders(httpHeaders -> httpHeaders.setBasicAuth( username: "user", password: "password"))
                .baseUrl(String.format("http://localhost:%s/graphql", port))
                .build();

        graphQlTester = HttpGraphQlTester.create(client);
    }

    @Test
    void contextLoads() { assertNotNull(graphQlTester); }
```

DGS FRAMEWORK

Netflix

DGS CODE GENERATION

Netflix

You can use tools such as DGS Codeine to generate Java types from the GraphQL schema.

The following can be generated:

- Client types for requests (e.q. query, mutation) input types, and response selection types
- Data types corresponding to GraphQL schema types

https://docs.spring.io/spring-graphql/reference/codegen.html

Build GraphQL Services with Spring Boot like Netflix

Netflix uses GraphQL and the DGS framework (recently fully integrated with Spring GraphQL) as our primary architecture. In this talk, you'll learn everything you need to know to do the same. This talk is a deep dive into implementing GraphQL services with Spring Boot and the DGS framework. You'll learn about GraphQL schemas and how to implement a schema. While you'll also learn about the fundamentals, the goal of the presentation is to look at more advanced scenarios you'll run into, such as batch loading, nested data fetchers, error handling, threading and parallel data fetching, and testing. GraphQL services are great stand-alone, but things really get exciting when using GraphQL Federation in a micro services ecosystem. You'll also learn about federation and how to make your service work in a federated architecture!

Paul Bakker

Staff Engineer
Netflix

GRAPHQL FEATURES

But wait there's more!

- **Batch Loading (n+1)**
- **Error Handling**
- **GraphQL Client**
- **GraalVM Native Image Support**
- **Data Integration**
- **DSG Code Generation**
- **Federation**
- **Defer**
- **And More!**

Build GraphQL Services with Spring Boot like Netflix

Netflix uses GraphQL and the DGS framework (recently fully integrated with Spring GraphQL) as our primary architecture. In this talk, you'll learn everything you need to know to do the same. This talk is a deep dive into implementing GraphQL services with Spring Boot and the DGS framework. You'll learn about GraphQL schemas and how to implement a schema. While you'll also learn about the fundamentals, the goal of the presentation is to look at more advanced scenarios you'll run into, such as batch loading, nested data fetchers, error handling, threading and parallel data fetching, and testing. GraphQL services are great stand-alone, but things really get exciting when using GraphQL Federation in a micro services ecosystem. You'll also learn about federation and how to make your service work in a federated architecture!

Paul Bakker

Staff Engineer
Netflix

STAY CONNECTED

Let’s continue the conversation

Follow me **@therealdanvega**

Email me **danvega@gmail.com**

Learn more **https://www.danvega.dev**

THANK YOU