Spring GraphQL

Andreas Marek
Rossen Stoyanchev

Copyright © 2021 VMware, Inc. or its affiliates.

Who are we

Andreas Marek

- GraphQL Java creator and maintainer
- GraphQL contributor and Technical Steering Committee member
- Spring GraphQL co-creator
- Working at Atlassian, Sydney
- @andimarek on twitter and github

Rossen Stoyanchev

- Spring Framework committer - Spring MVC, WebFlux, web messaging, RSocket
- RSocket Java committer
- Spring GraphQL co-creator
- Working at VMware, Cambridge, UK
- @rstoya05 on twitter

Agenda

- What is GraphQL?
- GraphQL Java
- Spring GraphQL

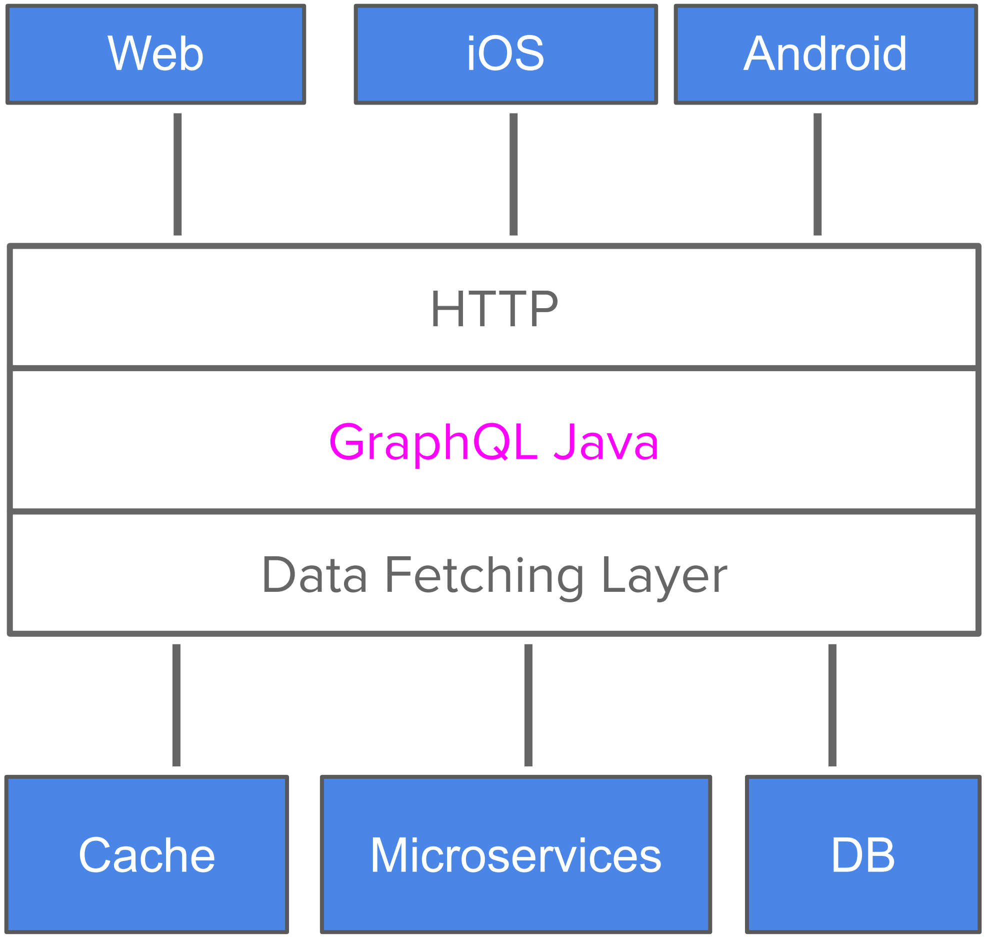
What is GraphQL?

What is GraphQL?

GraphQL is a technology for client server data exchange

- A clients wants to access data on a server (across a network)
- Originally developed by Facebook for their iOS app in 2012
- Open sourced in 2015, governed by a non profit foundation today
- Two pillars: statically typed API + query language
- Sweet spots are Single Page Apps and native clients
- An alternative to REST(ish) APIs
- My favourite argument for GraphQL: the developer experience

GraphiQL demo


Web
iOS
Android
HTTP
GraphQL Java
Data Fetching Layer
Cache
Microservices
DB

The two pillars of GraphQL

GraphQL Schema

- Describes your API
- Defined on the server
- Based on a simple static types system
- Schema Definition Language (SDL) is used to describe a Schema

GraphQL Query Language

- Custom query language
- Clients define the query based on their needs
- Every field needs to be requested explicitly

GraphQL Schema Example

```
type Query {
    allEmployees: [Employee]
}

type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}

type Department {
    id: ID!
    name: String
    employees: [Employee]
}
```

```graphql
type Mutation {
    updateSalary(input: UpdateSalaryInput!): UpdateSalaryPayload
}

input UpdateSalaryInput {
    employeeId: ID!
    salary: String!
}

type UpdateSalaryPayload {
    success: Boolean!
    employee: Employee
}
```

GraphQL Query Example

```
{
    allEmployees {
        id
        name
        department {
            name
            employees {
                id
            }
        }
    }
}
```

The GraphQL ecosystem

GraphQL ecosystem is based on a specification

- The GraphQL **spec**ification defines how GraphQL queries should be executed
- It defines the GraphQL schema + query language
- First there was the spec + reference implementation in JS (GraphQL.JS)
- Next it was implemented in every major language

GraphQL Java

GraphQL Java

GraphQL Java is the GraphQL implementation for Java

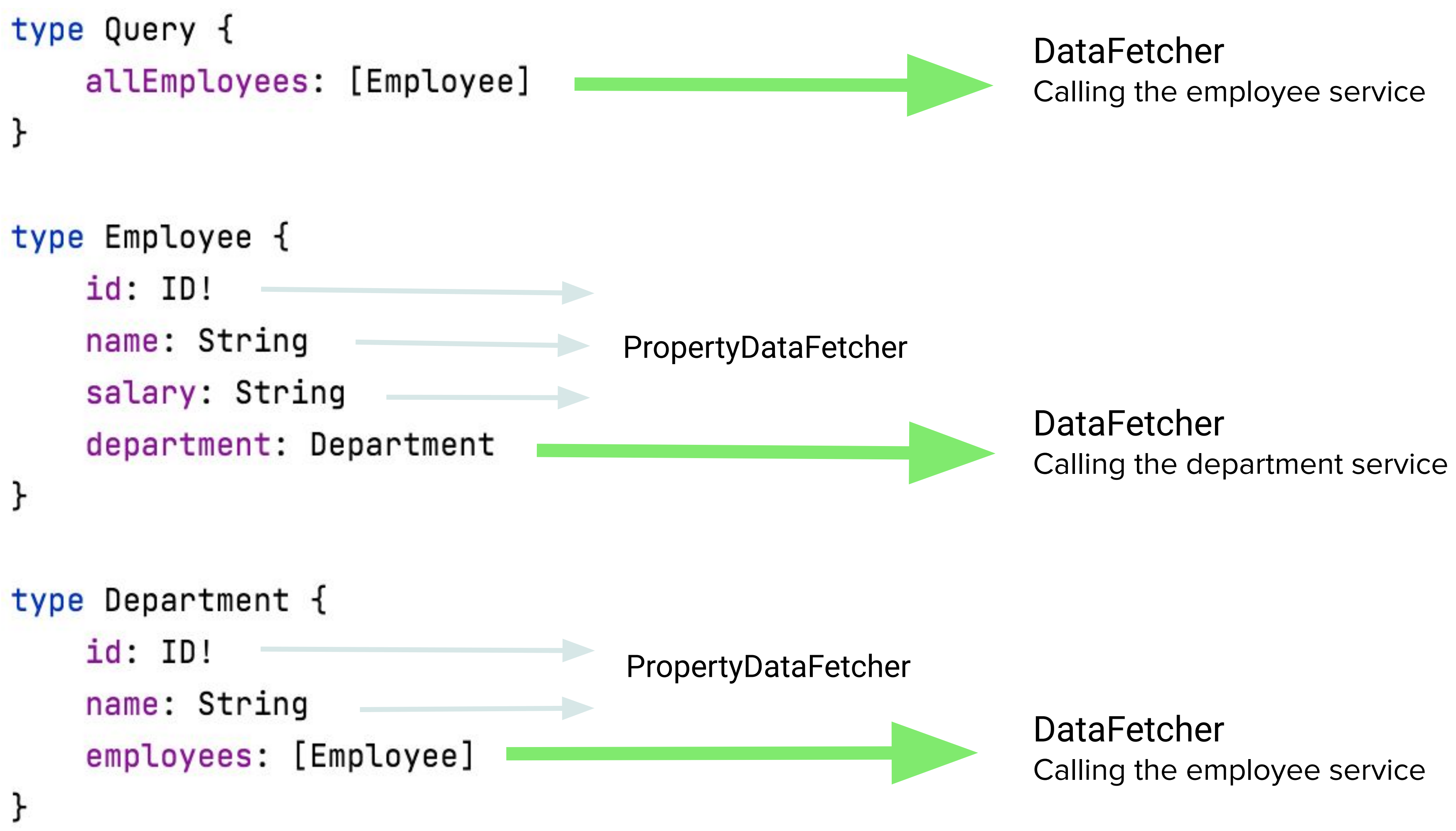
- It is an implementation for the server side GraphQL execution (also called execution engine)
- Started mid 2015
- Pure engine: no HTTP or IO. No high level abstracts.
- Used word wide and empowers a whole ecosystem of libraries build on top
- https://graphql-java.com/

How to think in GraphQL Java

Schema first and DataFetchers

- Start designing by putting the Schema first
- Use case and client oriented
- Define the schema in SDL (textual format, preferred) or programmatically
- Schema is made out of types with fields
- Fundamental rule: **every field has a DataFetcher associated with**
- DataFetcher fetches the data **for one field**
- If you don’t specify a DataFetcher a default DataFetcher is provided

```java
public interface DataFetcher<T> {

    T get(DataFetchingEnvironment environment) throws Exception;

}
```



```
type Query {
    allEmployees: [Employee]
}

type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}

type Department {
    id: ID!
    name: String
    employees: [Employee]
}
```

DataFetcher
Calling the employee service
PropertyDataFetcher
DataFetcher
Calling the department service
PropertyDataFetcher
DataFetcher
Calling the employee service

Request Execution: DataFetchers sequence

```
{
    allEmployees {
        id
        name
        department {
            name
            employees {
                id
            }
        }
    }
}
```

1 **Query.allEmployees**

2a Employee.id

2b Employee.name

2c **Employee.department**

3a Department.name

3b **Department.employees**

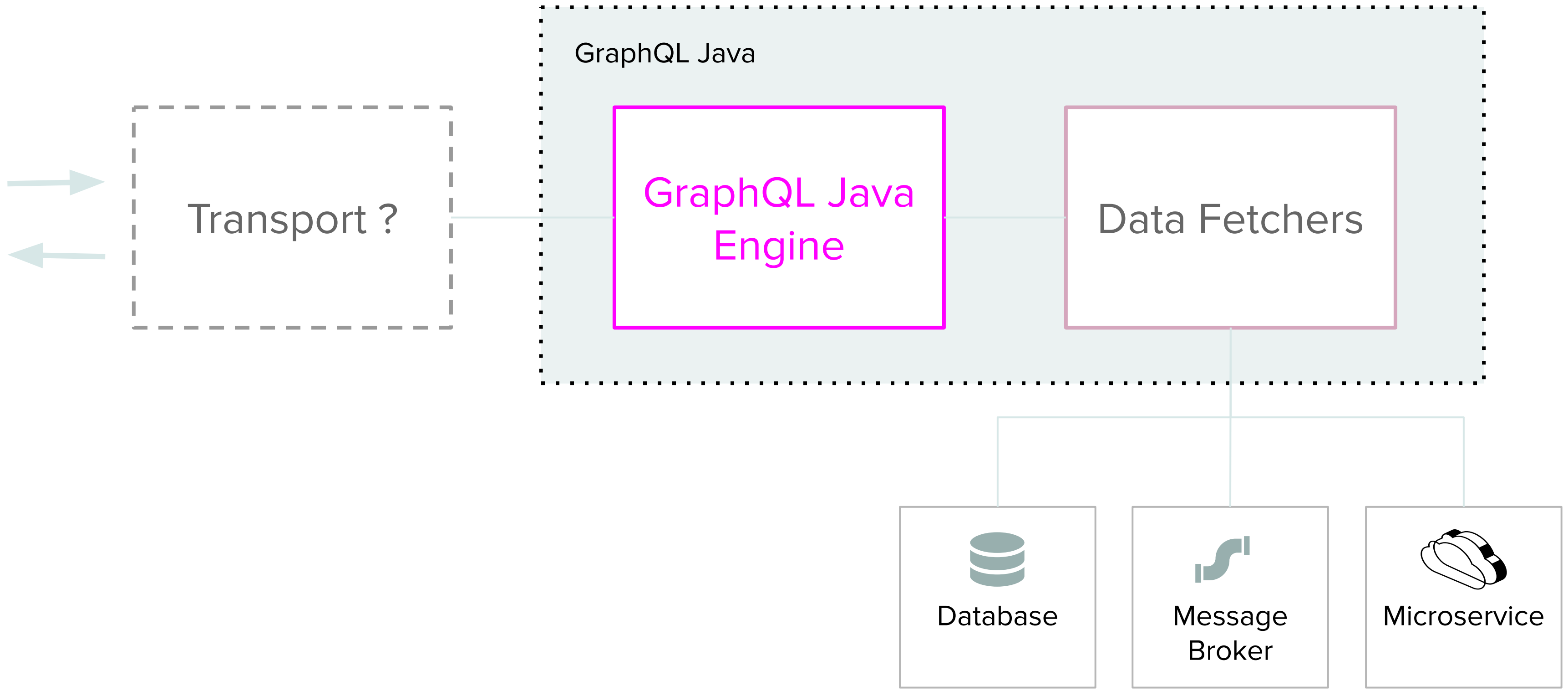
4 Employee.id

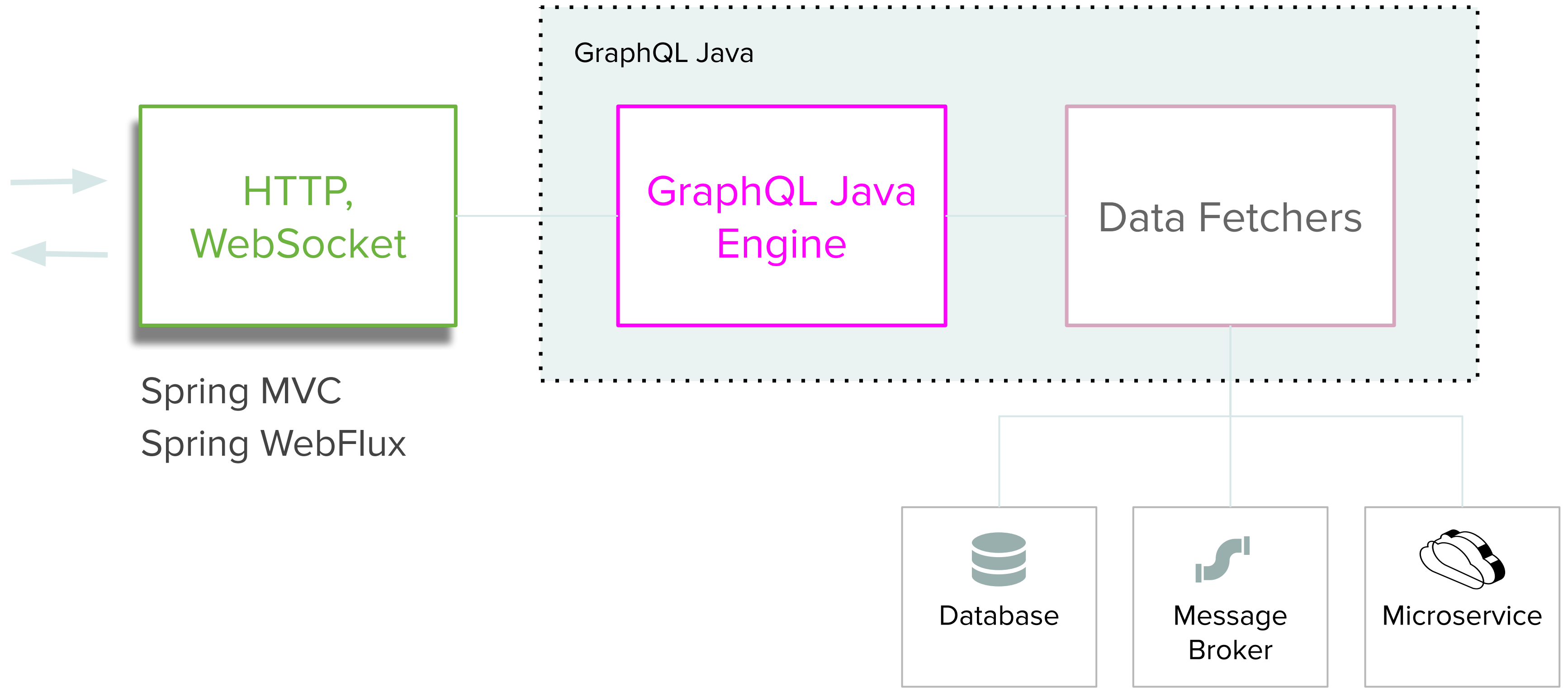
From GraphQL Java to Spring GraphQL

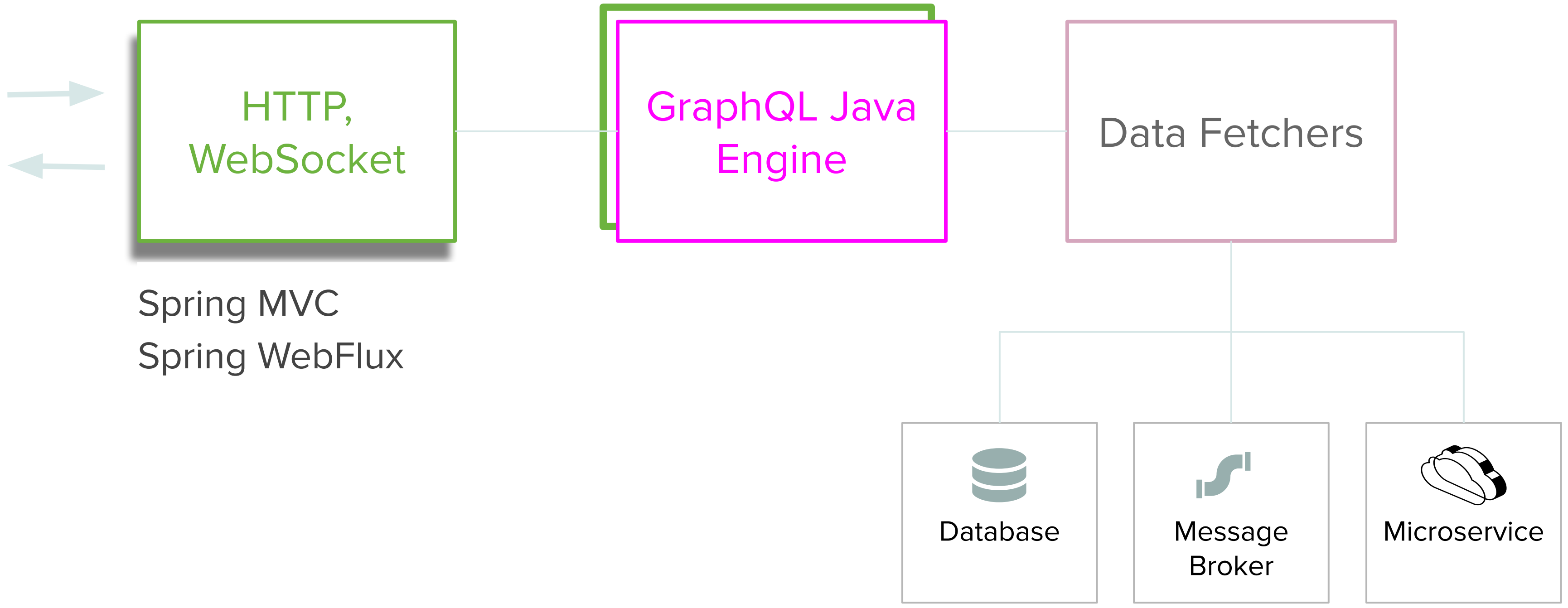
Spring GraphQL is the missing gap in the developer story

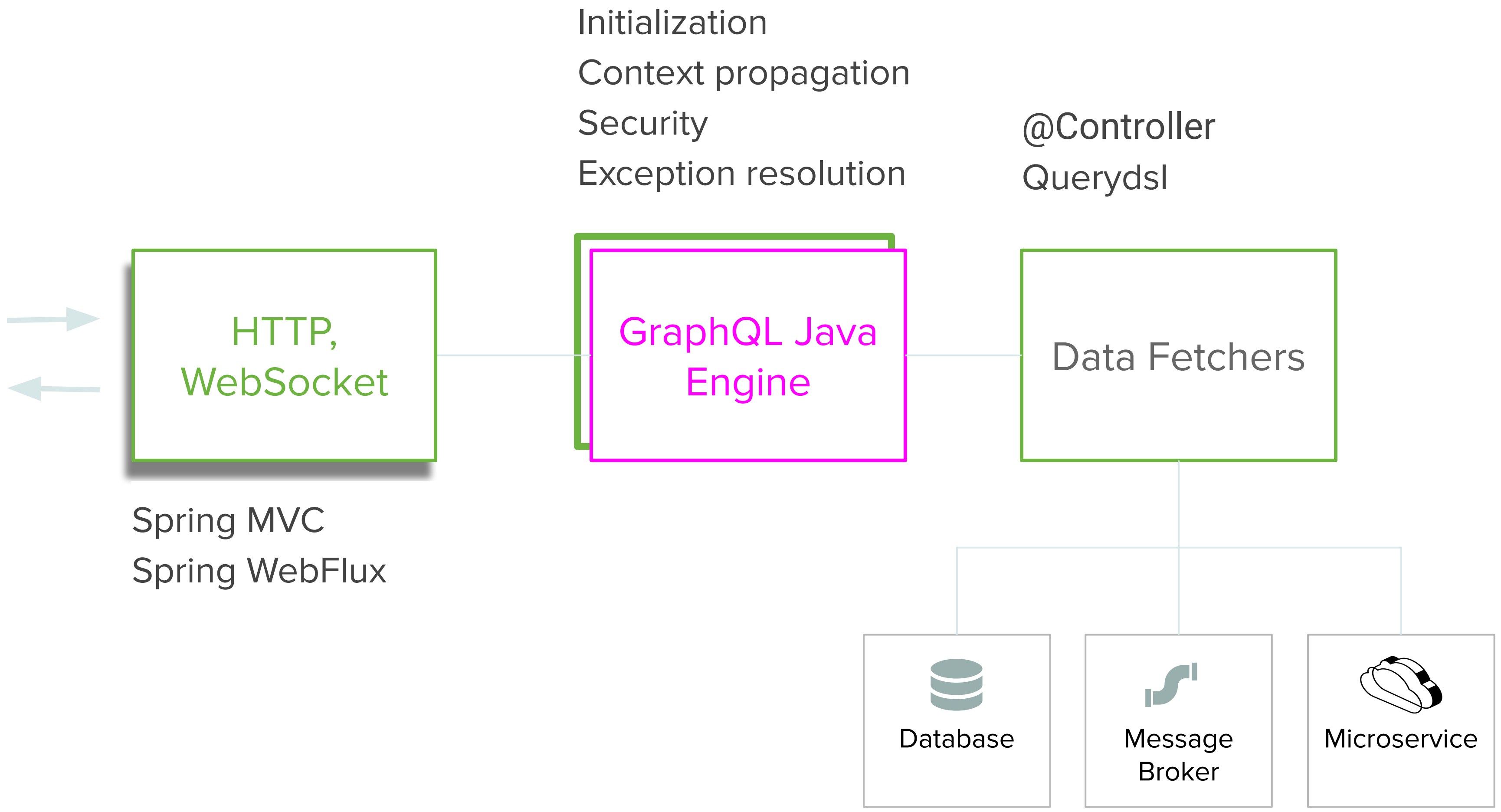
- GraphQL Java is “limited” on purpose
- GraphQL Java lets you do everything, but not everything is as simple and convenient as possible
- The Spring and GraphQL Java teams came together to fix that
- Spring GraphQL is focused on comprehensive and first level support
- It aims to be a fundamental building block build directly on GraphQL Java

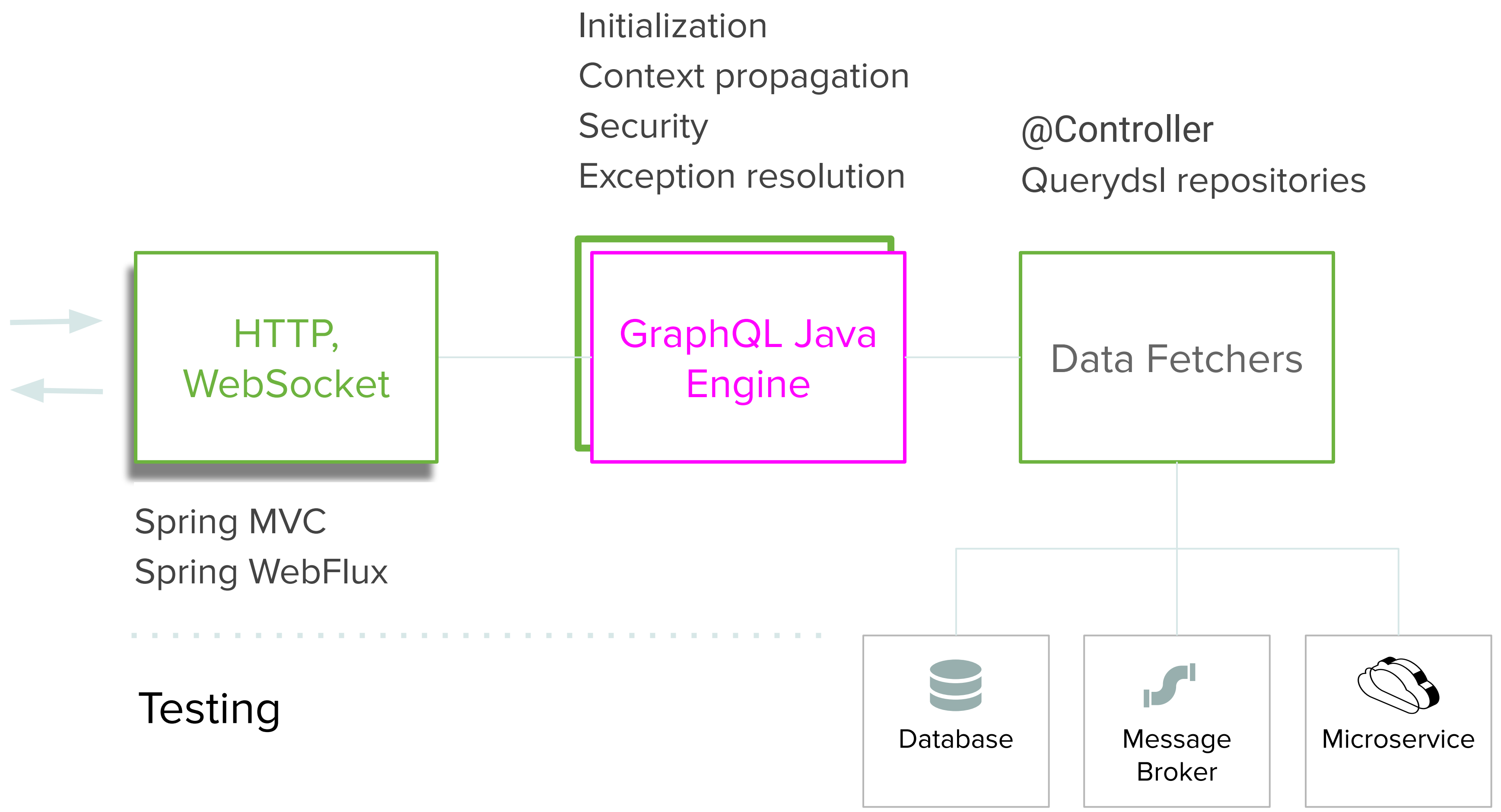
Spring GraphQL


Transport ?
GraphQL Java
GraphQL Java Engine
Data Fetchers
Database
Message Broker
Microservice

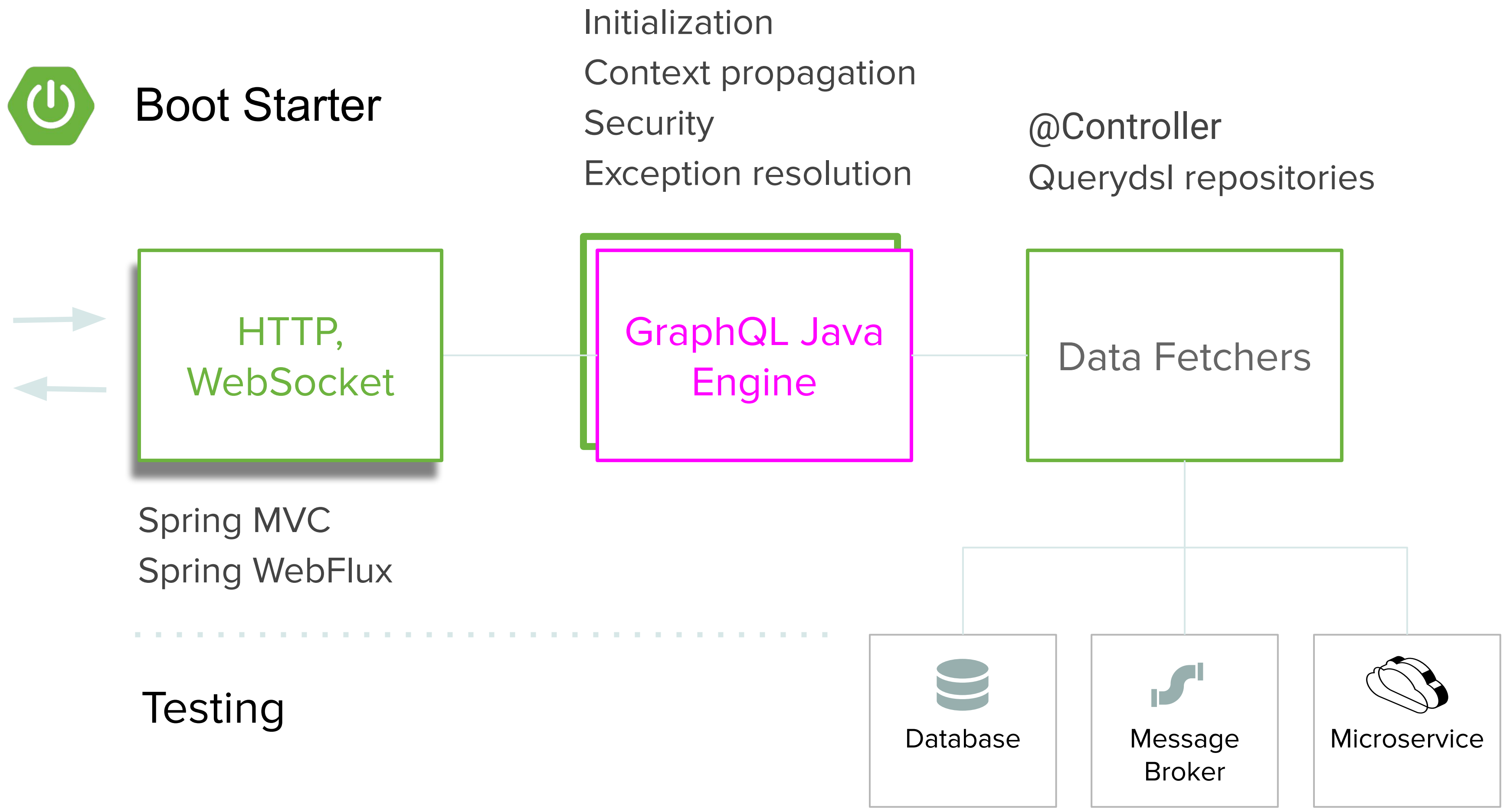

GraphQL Java
HTTP, WebSocket
GraphQL Java Engine
Data Fetchers
Spring MVC
Spring WebFlux
Database
Message Broker
Microservice


Initialization
Context propagation
Security
Exception resolution
HTTP, WebSocket
Spring MVC
Spring WebFlux
GraphQL Java Engine
Data Fetchers
Database
Message Broker
Microservice

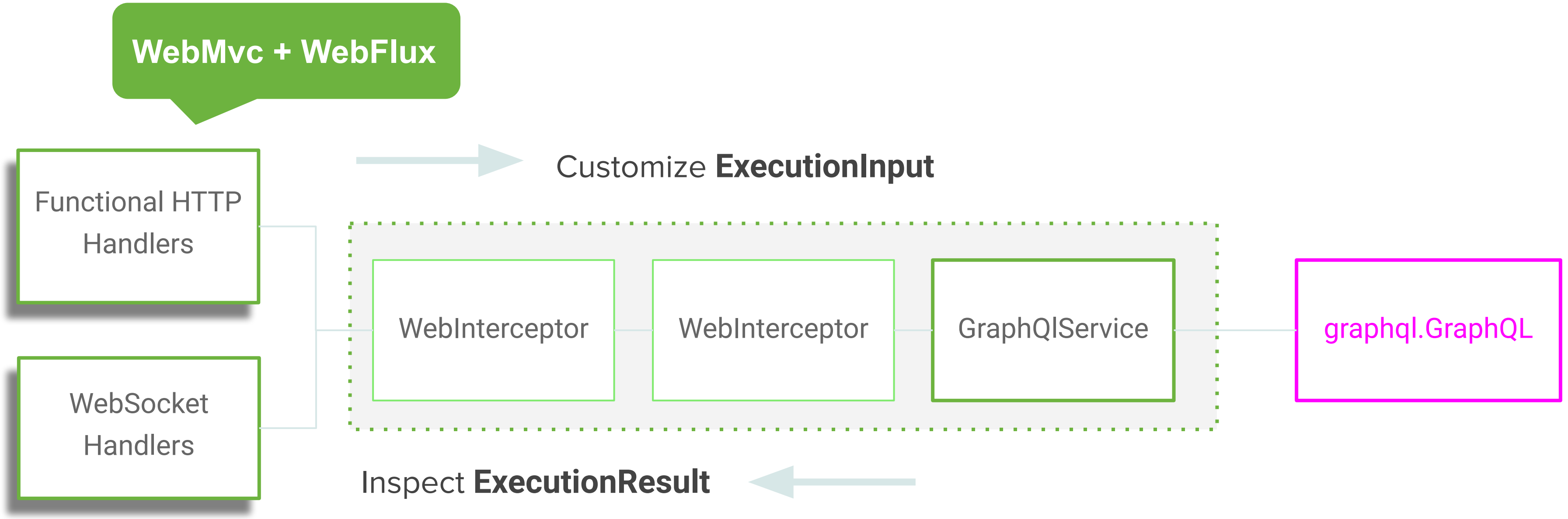

Initialization
Context propagation
Security
Exception resolution
@Controller
Querydsl
HTTP, WebSocket
GraphQL Java Engine
Data Fetchers
Spring MVC
Spring WebFlux
Database
Message Broker
Microservice


Initialization
Context propagation
Security
Exception resolution
@Controller
Querydsl repositories
HTTP, WebSocket
GraphQL Java Engine
Data Fetchers
Spring MVC
Spring WebFlux
Database
Message Broker
Microservice

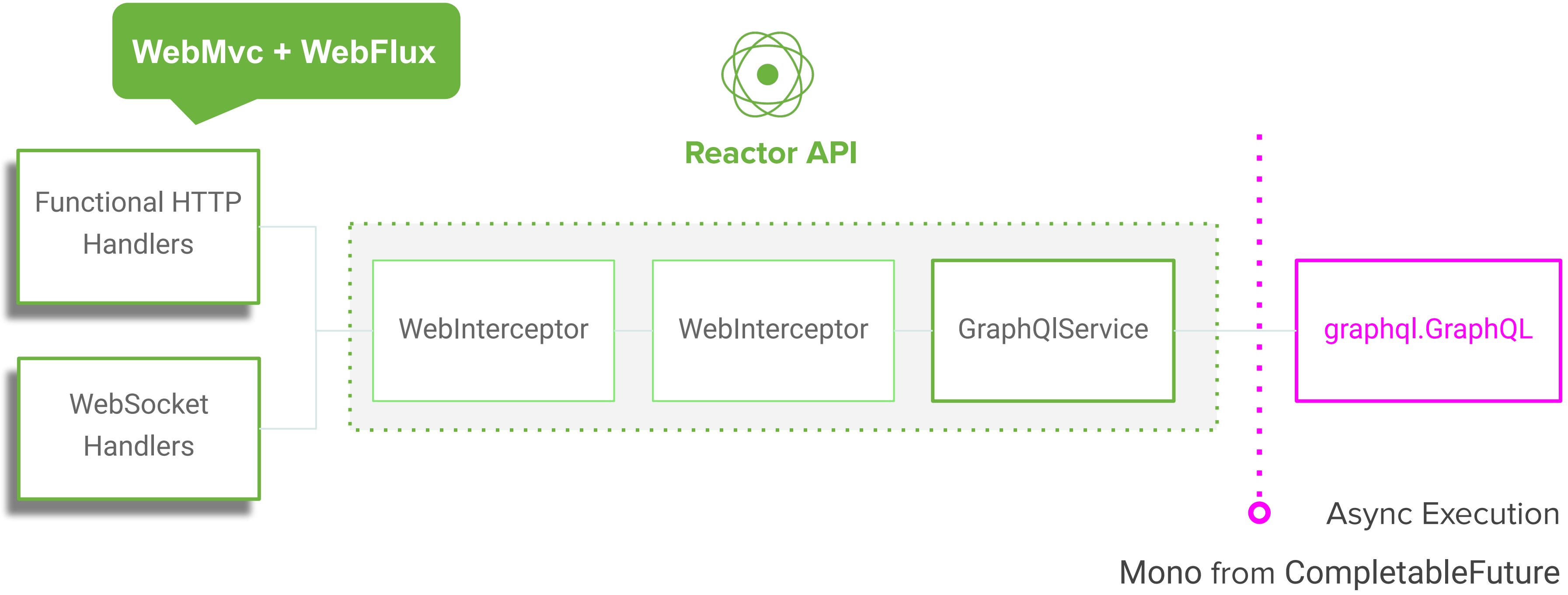
Testing

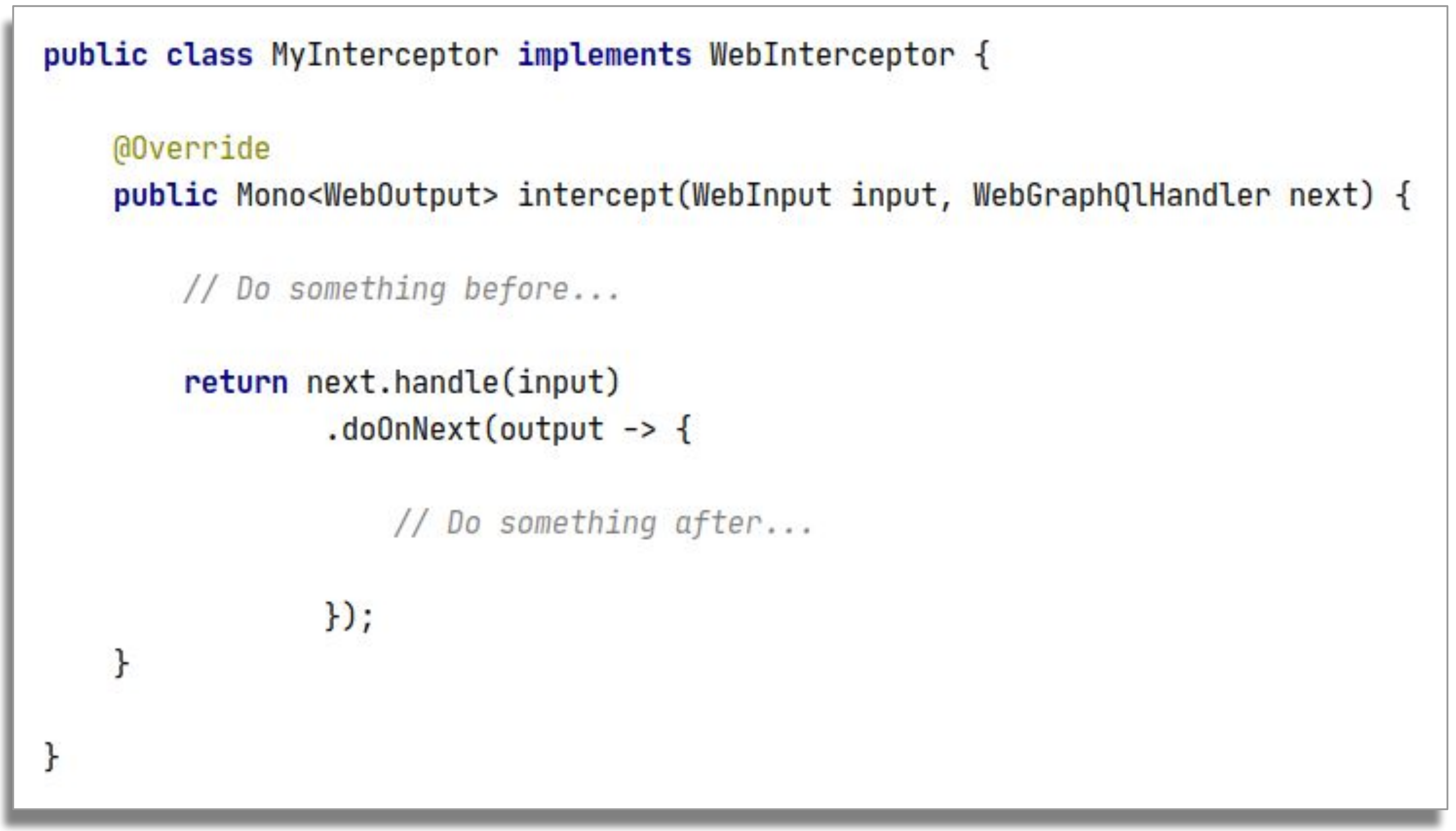

Boot Starter
Initialization
Context propagation
Security
Exception resolution
@Controller
Querydsl repositories
HTTP, WebSocket
GraphQL Java Engine
Data Fetchers
Spring MVC
Spring WebFlux
Testing
Database
Message Broker
Microservice

Web Transports



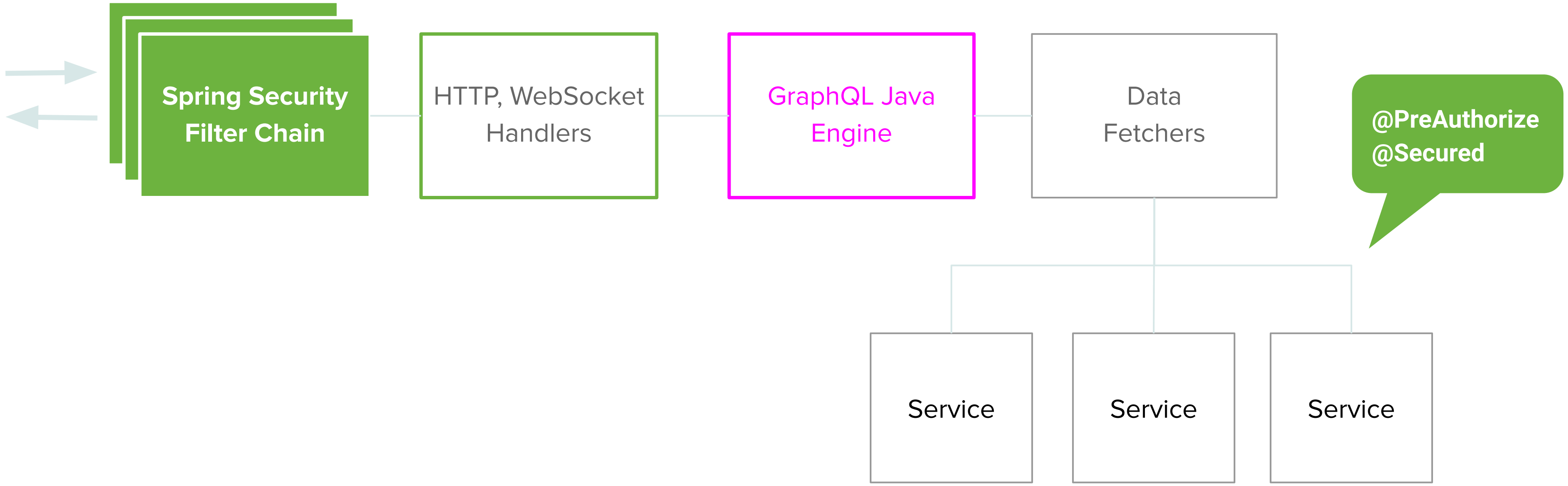
Web Transports





```java
public class MyInterceptor implements WebInterceptor {

    @Override
    public Mono<WebOutput> intercept(WebInput input, WebGraphQlHandler next) {

        // Do something before...

        return next.handle(input)
                .doOnNext(output -> {

                    // Do something after...

                });
    }

}
```

Security


Spring Security Filter Chain
HTTP, WebSocket Handlers
GraphQL Java Engine
Data Fetchers
@PreAuthorize
@Secured
Service
Service
Service

Security and Context Propagation

Spring MVC

ThreadLocal context propagation from Servlet container thread

Need to register ThreadLocalAccessor

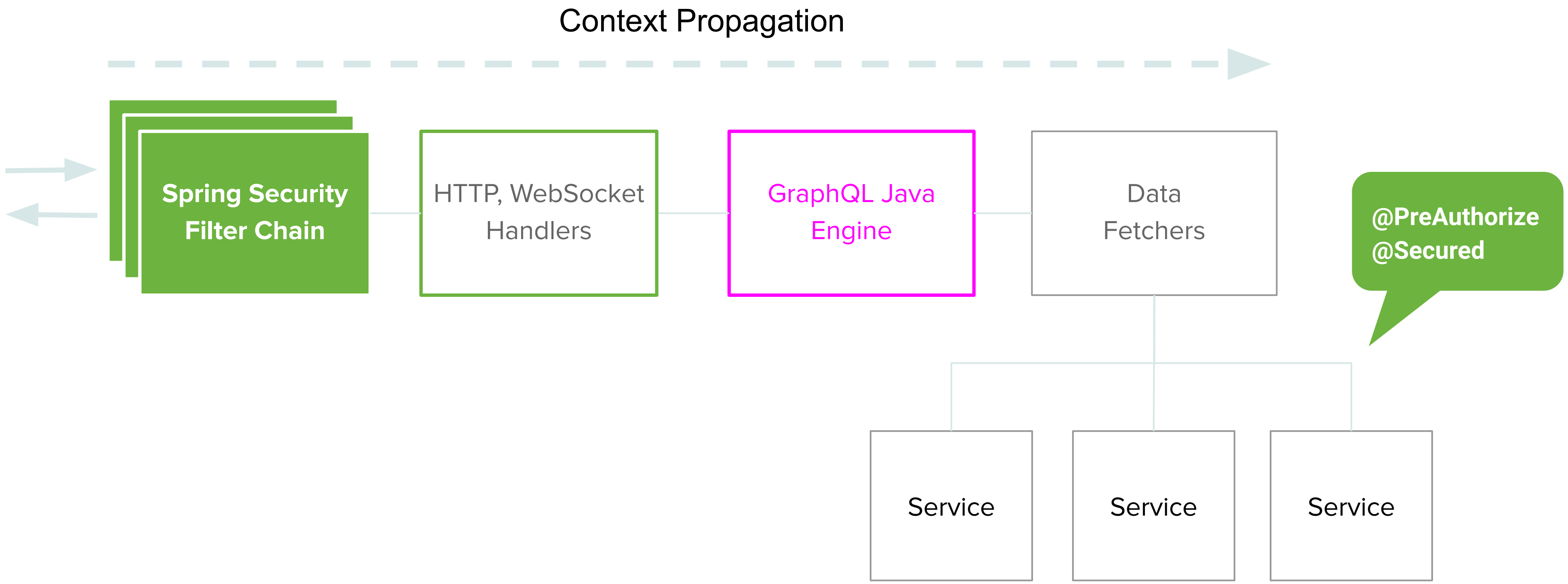
Built-in accessor for Spring Security context

Spring WebFlux

Reactor context propagation from web layer

Spring Security context propagated

Security

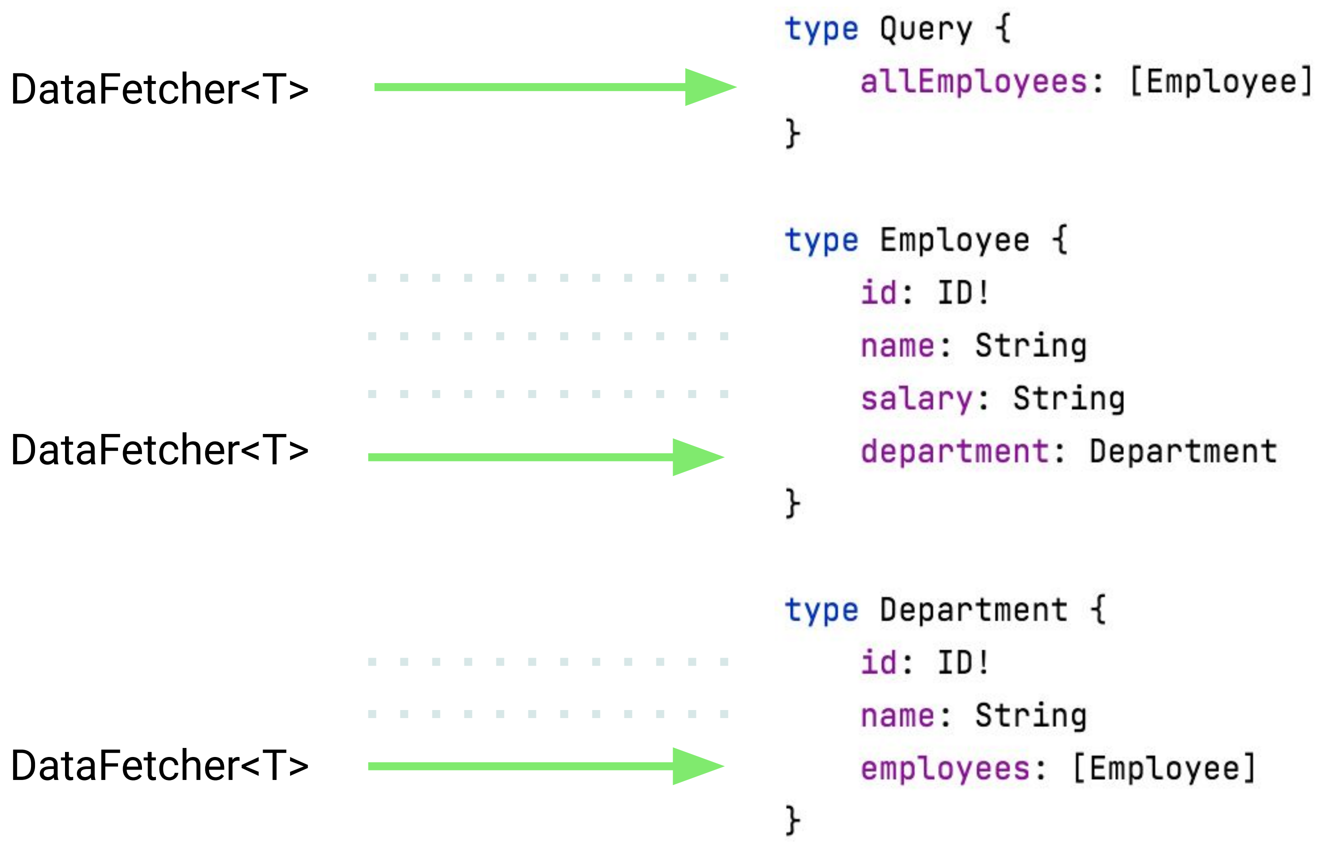

Context Propagation
Spring Security Filter Chain
HTTP, WebSocket Handlers
GraphQL Java Engine
Data Fetchers
@PreAuthorize
@Secured
Service
Service
Service

Data Layer

The DataFetcher Contract

```java
public interface DataFetcher<T> {

    T get(DataFetchingEnvironment environment) throws Exception;

}
```

DataFetcher Wiring to Schema Fields



```
type Query {
    allEmployees: [Employee]
}

type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}

type Department {
    id: ID!
    name: String
    employees: [Employee]
}
```

```
public void configure(RuntimeWiring.Builder wiringBuilder) {

    wiringBuilder.type("Query", builder -> builder.dataFetcher(
            "allEmployees", environment -> this.employeeService.getAllEmployees()));







}
```

```java
public void configure(RuntimeWiring.Builder wiringBuilder) {

    wiringBuilder.type("Query", builder -> builder.dataFetcher(
            "allEmployees", environment -> this.employeeService.getAllEmployees()));

    wiringBuilder.type("Department", builder -> builder.dataFetcher(
            "employees", environment -> {
                Department department = environment.getSource();
                return this.employeeService.getEmployeesForDepartment(department.getId());
            }));




}
```

```java
public void configure(RuntimeWiring.Builder wiringBuilder) {

    wiringBuilder.type("Query", builder -> builder.dataFetcher(
            "allEmployees", environment -> this.employeeService.getAllEmployees()));

    wiringBuilder.type("Department", builder -> builder.dataFetcher(
            "employees", environment -> {
                Department department = environment.getSource();
                return this.employeeService.getEmployeesForDepartment(department.getId());
            }));

    wiringBuilder.type("Mutation", builder -> builder.dataFetcher(
            "updateSalary", environment -> {
                Map<String, Object> inputMap = environment.getArgument("input");
                String employeeId = (String) inputMap.get("employeeId");
                BigDecimal salary = new BigDecimal((String) inputMap.get("newSalary"));
                this.employeeService.updateSalary(employeeId, salary);
                return null;
            }));
}
```

```java
@Controller
public class EmployeeController {

    @QueryMapping
    public List<Employee> allEmployees() {
        return this.employeeService.getAllEmployees();
    }
```

type: **Query**, field: allEmployees

```graphql
type Query {
    allEmployees: [Employee]
}

type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}
```

```java
@Controller
public class EmployeeController {

    @QueryMapping
    public List<Employee> allEmployees() {
        return this.employeeService.getAllEmployees();
    }

    @SchemaMapping
    public List<Employee> employees(Department department) {
        return employeeService.getEmployeesForDepartment(department.getId());
    }
```

type = **Department**, field = employees

```graphql
type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}

type Department {
    id: ID!
    name: String
    employees: [Employee]
}
```

```
type Mutation {
  updateSalary(input: UpdateSalaryInput!): UpdateSalaryPayload
}
```

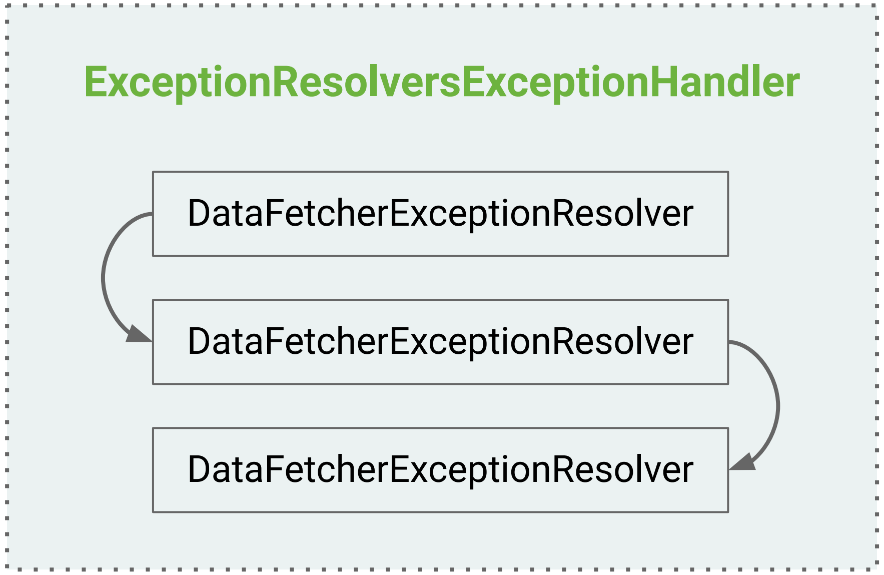
```java
@Controller
public class EmployeeController

    @QueryMapping
    public List<Employee> allEmployees() {
        return this.employeeService.getAllEmployees();
    }

    @SchemaMapping
    public List<Employee> employees(Department department) {
        return employeeService.getEmployeesForDepartment(department.getId());
    }

    @MutationMapping
    public void updateSalary(@Argument SalaryInput input) {
        String employeeId = input.getEmployeeId();
        BigDecimal salary = input.getNewSalary();
        this.employeeService.updateSalary(employeeId, salary);
    }
```

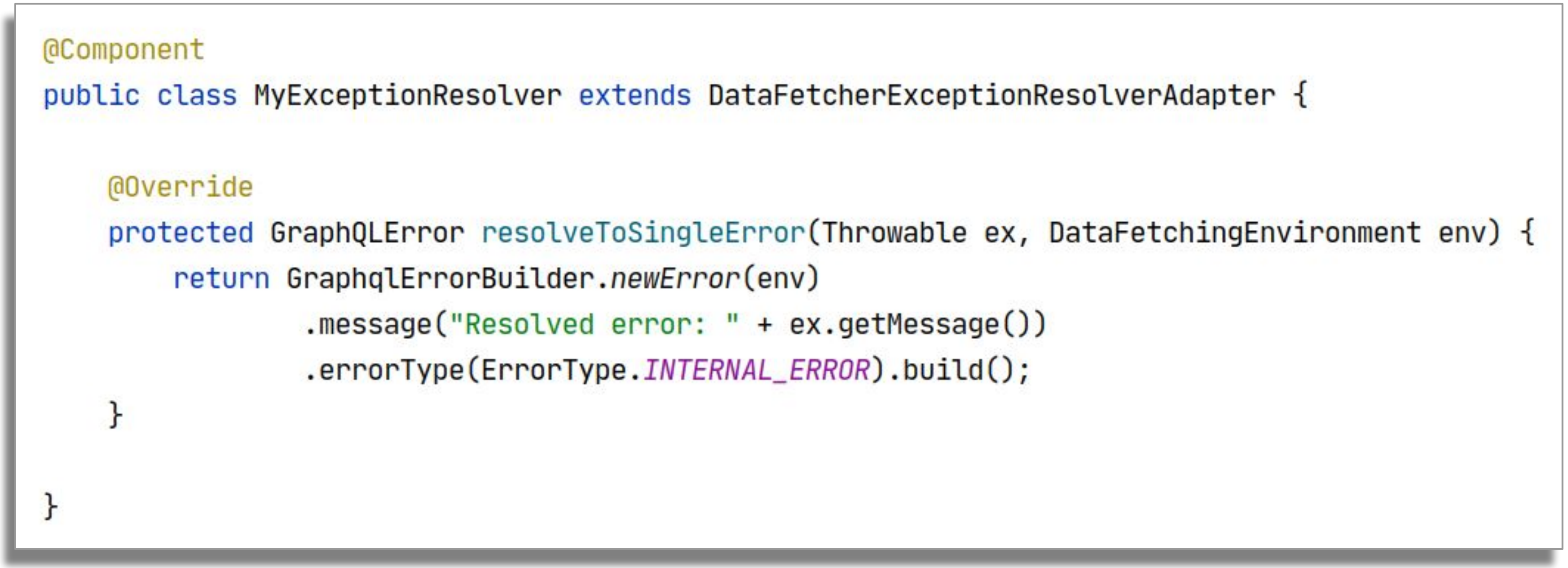
type = **Mutation**, field = updateSalary

Exception Handling

GraphQL Java allows registering a single DataFetcherExceptionHandler

Spring GraphQL enables a DataFetcherExceptionResolver chain


ExceptionResolversExceptionHandler
DataFetcherExceptionResolver
DataFetcherExceptionResolver
DataFetcherExceptionResolver



```java
@Component
public class MyExceptionResolver extends DataFetcherExceptionResolverAdapter {

    @Override
    protected GraphQLError resolveToSingleError(Throwable ex, DataFetchingEnvironment env) {
        return GraphqlErrorBuilder.newError(env)
                .message("Resolved error: " + ex.getMessage())
                .errorType(ErrorType.INTERNAL_ERROR).build();
    }

}
```

Querydsl

Typesafe way to express queries in Java that works across multiple data stores

Spring Data has support for Querydsl

Spring GraphQL and Querydsl

QuerydslDataFetcher

Adapts a Spring Data repository to DataFetcher

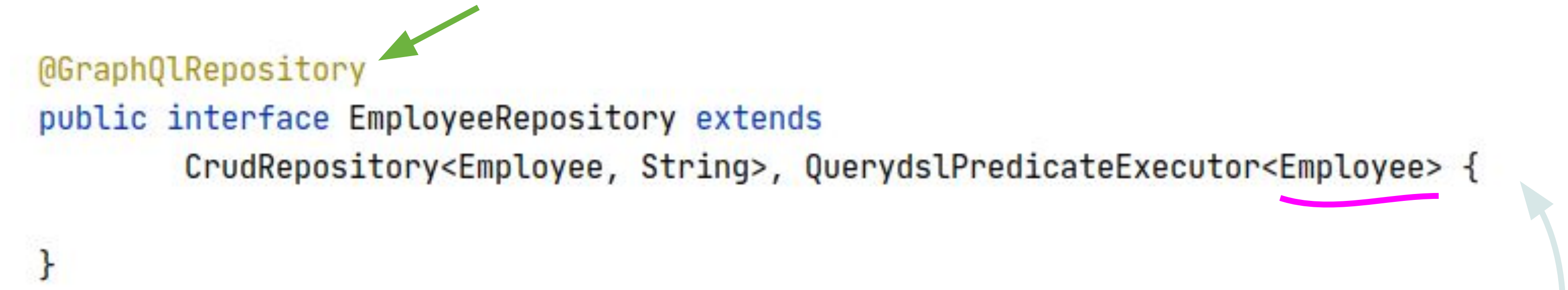
Translates GraphQL query parameters to Querydsl Predicate

```java
public interface EmployeeRepository extends
        CrudRepository<Employee, String>, QuerydslPredicateExecutor<Employee> {

}
```

Adapt the Repository to a DataFetcher

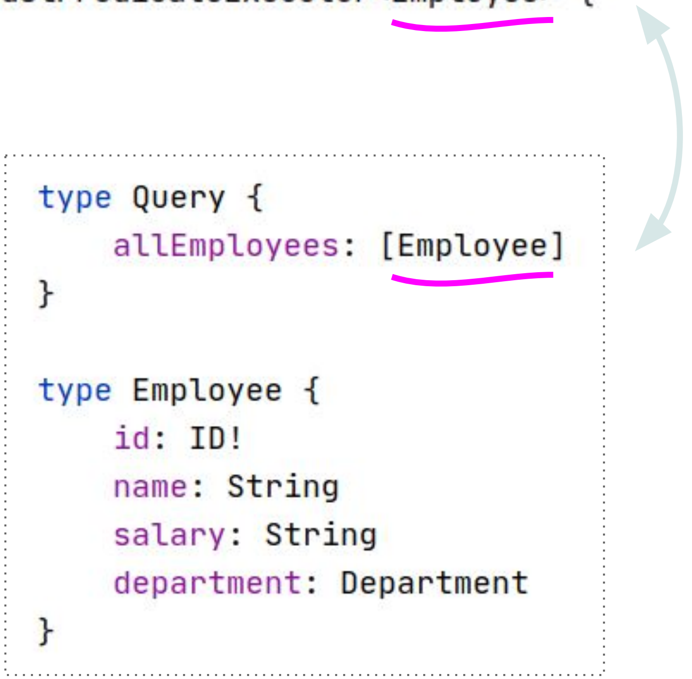
```java
// For single result queries
DataFetcher<Employee> dataFetcher =
        QuerydslDataFetcher.builder(repository).single();


// For multi-result queries
DataFetcher<Iterable<Employee>> dataFetcher =
        QuerydslDataFetcher.builder(repository).many();
```

Automatic Registration



```java
@GraphQlRepository
public interface EmployeeRepository extends
        CrudRepository<Employee, String>, QuerydslPredicateExecutor<Employee> {

}
```

Match based on query return type for top-level queries



```graphql
type Query {
    allEmployees: [Employee]
}

type Employee {
    id: ID!
    name: String
    salary: String
    department: Department
}
```

More on Querydsl

Customize how GraphQL request parameters are mapped to Querydsl Predicate

Transform the resulting Objects via interface and DTO projections

GraphQlTester

Workflow for testing GraphQL

Automatic checks to verify no errors in response

Use JsonPath to specify data to inspect

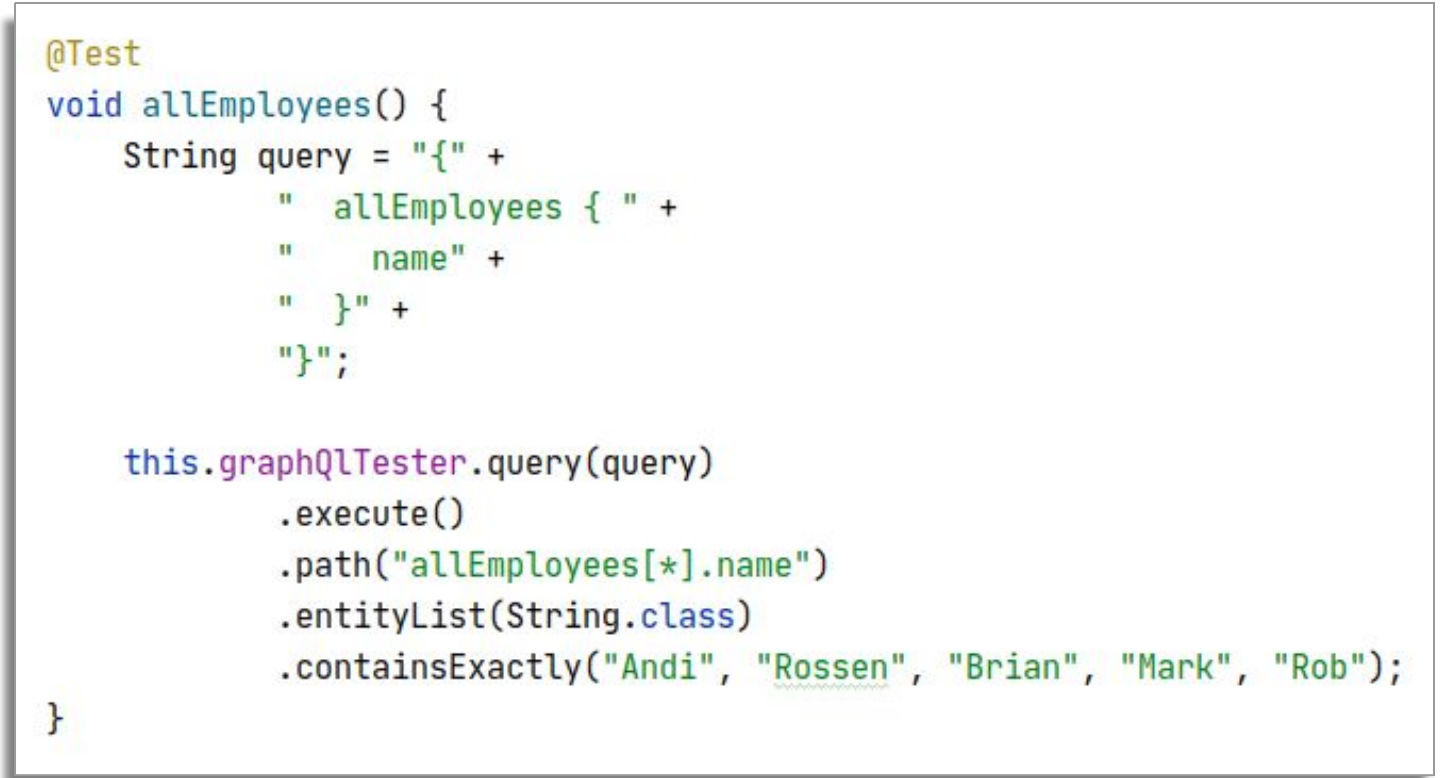
Data decoding

WebGraphQlTester

Extension for GraphQL tests over web transports

Specify HTTP specific inputs

Uses WebTestClient



```java
@Test
void allEmployees() {
    String query = "{" +
            "  allEmployees { " +
            "    name" +
            "  }" +
            "}";

    this.graphQlTester.query(query)
            .execute()
            .path("allEmployees[*].name")
            .entityList(String.class)
            .containsExactly("Andi", "Rossen", "Brian", "Mark", "Rob");
}
```

Spring Boot Starter

Dependencies

Autoconfig

Properties

Metrics

GraphiQL UI and Schema pages

Spring Boot Starter

Currently In Spring GraphQL repository, group id ‘org.springframework.experimental’

Due to move to Spring Boot, after version 2.6 is released

Collaboration with Netflix DGS

Optional, alternative starter to run DGS on the spring-graphql core

DGS programming model + spring-graphql WebMvc / WebFlux foundation

spring-graphql starter to become the main starter eventually

Netflix DGS Features

Annotation based registration of data fetchers, data loaders, scalars, etc.

Code generation for GraphQL Schema -> Java/Kotlin

GraphQL client for Java/Kotlin

Federation

Roadmap Timeline

M3: mid-October

Starter integrated into Spring Boot, after 2.6 release

RC phase: early 2022

GA: May 2022 with Spring Boot 2.7

Roadmap Features

Evolve controller programming model, #63 (batch loaders), #110 (bean validation)

Automated registration of Spring Data repositories, #99

Query by Example (QBE) support as an alternative to Querydsl, #115

GraphQL client, #10

More…

Roadmap Features

Your feedback

Thank you

Andreas Marek	@andimarek
Rossen Stoyanchev	@rstoya05

© 2021 Spring. A VMware-backed project.