GraphQL - Service & Client

Java 21 and Spring Boot

Agenda

- ➔ Transition from SOAP to REST
- ➔ History of GraphQL
- ➔ Market share / Popularity
- ➔ Similarities between REST and GraphQL
- ➔ Difference between REST and GraphQL
- ➔ Choosing REST or GraphQL
- ➔ Resources / Documentation
- ➔ Schema / Contract Driven
- ➔ Operations: Query + Mutation
- ➔ Data types: Inbuilt & Custom
- ➔ Field Selection by Clients
- ➔ Data loading on the Server
- ➔ CQRS (Command Query Responsibility Separation)
- ➔ Spring Boot GraphQL
- ➔ Tools: GraphiQL & Others

Agenda

- Demo Repo
- Execution of Query
  - Single Query
  - Multiple query
  - Parallel
  - Batching
  - N+1 without ORM
- Execution of Mutation
  - Single Mutation
  - Validation
  - Multiple mutation
  - Sequential
  - Transactions and Boundaries
- Error / Exception handling
- REST & GraphQL coexistence
- Async execution of operations and loaders
- Spring GraphQL Client
  - Code Generation
- MDC and Context Propagation
- Security and CORS

SOAP to REST

- SOAP Services were dominant till 2010
- REST Services gained popularity since Google Search Engine became popular due to its highly responsive AJAX search
- REST Services overtook SOAP by 2012
- REST gained traction:
  - JSON
  - CRUD operations by HTTP semantic methods (GET/POST/PUT/…)
  - REST services can be consumed directly in Frontend Javascript within HTML
  - Simplicity
  - Loosely coupling between Client and Server

History of GraphQL

- GraphQL started as an internal Facebook project to overcome REST API limitations, then became open-source and is now a widely used data query language for APIs.
- Officially open-sourced by Facebook in 2015
- Gained traction among developers due to its advantages like flexibility and data control
- Moved to the newly formed GraphQL Foundation under the Linux Foundation in 2018
- Increased adoption by major corporations like Netflix, Amazon, GitHub, and Google
- Became the default API access method for many public APIs

Market share / Popularity

- https://blog.postman.com/graphql-vs-rest/
- https://www.postman.com/state-of-api/api-technologies/#api-technologies

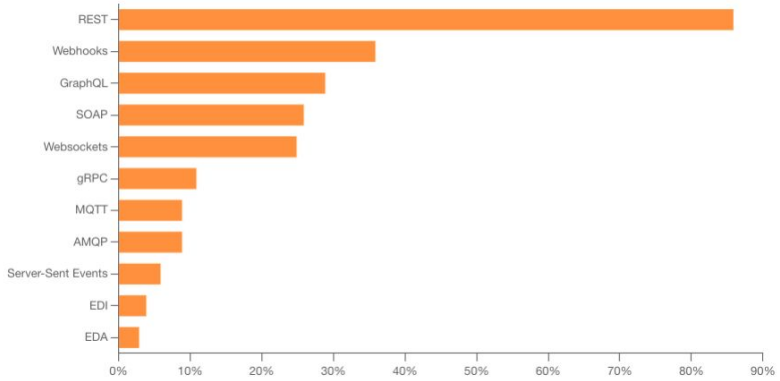
Market share / Popularity

- **Knowledge of GraphQL will be expected for backend engineers in next 3 years***

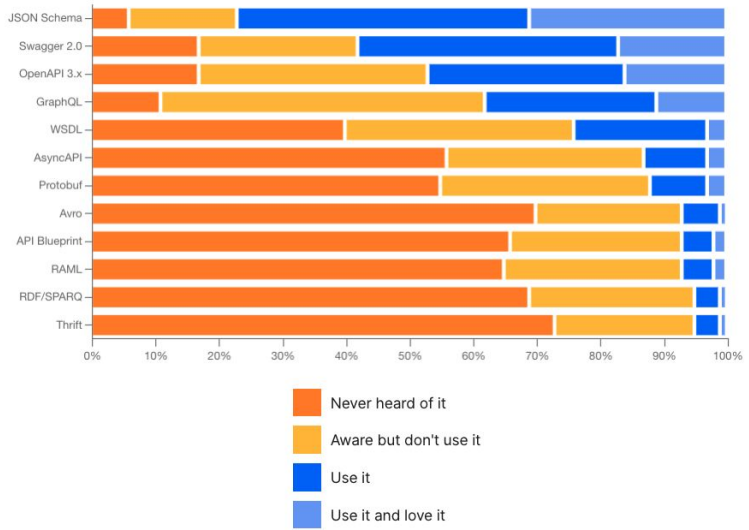
While REST remains the most-used API architecture by far, it has lost a bit of ground to newcomers. This year, 86% of respondents said they used REST, down from 89% last year and 92% the year prior.

SOAP registered a notable drop. It was used by just 26% of all respondents this year versus 34% last year. The decline makes SOAP the fourth most-used architecture in our survey, down from the third spot last year.

GraphQL took SOAP's spot and was used by 29% of survey-takers.



We also asked folks which API specifications they use and love. JSON Schema remains the top pick, named by almost twice as many respondents as any other. Swagger/Open API 2.0 and Open API 3.x were the next most popular choices, almost evenly tied.



Similarities between REST and GraphQL

- **Client-Server Model**: Both utilize a client-server model. The client application initiates requests, and the server fulfills those requests by sending back data.
- **Stateless Communication**: Neither REST nor GraphQL APIs store information about past client requests on the server side. Each request is treated independently.
- **HTTP Protocol**: Both primarily rely on HTTP as the underlying communication protocol for sending and receiving data. They leverage standard HTTP verbs like GET, POST, PUT, and DELETE.
- **Data Formats**: JSON is the most common data format used for communication in both REST and GraphQL. Other formats like XML are also supported in some cases.
- **Middleware and Extensions**: Both architectures can be extended using middleware to add functionalities like authentication, logging, and caching. This allows for customization and improved performance.
- **API Interoperability**: Both REST and GraphQL APIs can potentially be used with any client-side or server-side programming language, making them interoperable with various development environments.

Difference between REST and GraphQL

- Uses a single endpoint with a flexible query language. Clients specify exactly the data they need in a single request, reducing over- and under-fetching
- Clients define the desired data structure in their queries, ensuring they receive only the relevant data
- Requires defining a schema that outlines the data structure and relationships, which can add complexity
- Returns a single 200 OK status code for all requests, with detailed error information included in the response body
- Ideal for complex data models with interrelated data and dynamic client requirements
-

Choosing REST or GraphQL

A general rule of thumb:

- **Choose REST**: For simpler APIs, getting started quickly, or well-defined data access patterns.
- **Choose GraphQL**: For complex data models, reducing over-fetching, or providing clients with high flexibility in data access.

Resources

https://graphql.org/
https://graphql.org/learn/
https://github.com/graphql/graphql-spec

https://www.graphql-java.com/
https://github.com/graphql-java/graphql-java
https://github.com/graphql-java/graphql-java-extended-scalars

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

Schema / Contract Driven

https://docs.spring.io/spring-boot/reference/web/spring-graphql.html#web.graphql.schema

https://github.com/harishkannarao/graphql-web-mvc/blob/main/src/main/resources/graphql/schema.graphqls

Operations: Query + Mutation

- ★ Queries
  - Description / Comment
- ★ Mutations
  - Description / Comment

Data types: Inbuilt & Custom

- ★ Built-in Scalar Types:
  - 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 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.
  - enum
- ★ Custom Scalars
  - scalar Date
- ★ Input Type : Object / Collection
- ★ Output Type : Object / Collection
- ★ Interface Type

Field Selection by Clients

- ➔ Example
- ➔ Prevent Under Fetching
- ➔ Prevent Over Fetching

Data loading on the Server

- ➔ Example
- ➔ Prevent Under Fetching
- ➔ Prevent Over Fetching

CQRS (Command Query Responsibility Separation)

- ➔ CQRS stands for Command Query Responsibility Segregation.expand_more It's a software design pattern that separates how your application handles reads and writes.expand_more Traditionally, a single model handles both updating and reading data.expand_more CQRS splits this into two distinct models
- ➔ input for requests params in Query and Mutations
- ➔ type for response params in Query and Mutations

Spring Boot GraphQL

- v1.0.0 released on 18-May-2022
- https://docs.spring.io/spring-graphql/reference/index.html
- https://github.com/spring-projects/spring-graphql
- https://docs.spring.io/spring-graphql/reference/observability.html
- Other frameworks:
  - https://www.graphql-java.com/
    - https://github.com/graphql-java/graphql-java
  - https://netflix.github.io/dgs/

Tools: GraphiQL & Others

- GraphiQL (similar to Swagger)
  - https://docs.spring.io/spring-graphql/reference/graphiql.html
- https://graphql.org/community/tools-and-libraries/
- https://github.com/Kong/insomnia (meaning RESTLess)
- https://www.postman.com/product/graphql-client/ (from Postman)
- https://support.smartbear.com/readyapi/docs/testing/graphql.html (from ReadyApi)

Demo

https://github.com/harishkannarao/graphql-web-mvc

Execution of Query

- Single Query
  - Scalar
  - Custom Scalars
  - Custom Types
- Multiple query
- Parallel
- Batching
- N+1 without ORM

Execution of Mutation

- ❏ Single Mutation
- ❏ Validation
- ❏ Multiple Mutation
- ❏ Sequential
- ❏ Transactions and Boundaries

Error / Exception handling

- Unexpected errors (Internal Server Errors)
- Client errors / validation errors (Bad Request)
- Business errors

REST & GraphQL coexistence

- Supplementary to each other
- Share business logic and domain

Async execution of operations and loaders

- Query mapping
- Batch mapping

Spring GraphQL Client

- Spring's GraphQL Client
  - https://docs.spring.io/spring-graphql/reference/client.html
    - GraphQlClient (Similar to RestTemplate or RestClient)
    - GraphQlTester (Similar to TestRestTemplate)
  - Document Template with variable substitution
    - Similar to PreparedStatement in JDBC
- Code Generation
  - https://docs.spring.io/spring-graphql/reference/client.html#client.dgsgraphqlclient
  - https://github.com/Netflix/dgs-codegen
- Choose based on Project's use case

MDC and Context Propagation

- Request Id / Correlation Id example

Security and CORS

- Spring Method Security
- Role Based Access Control (RBAC)
- Http Request Header (Bearer Token) or Http Cookie based
- Integration with various auth protocols and Identity Provider (IDP)
  - OAuth 2.0
  - OpenID Connect
  - JWT
  - SAML
  - LDAP
- CORS

Questions

Thank You