**What is GraphQL?**

* **Query language** - it is only query language (similar to SQL)
  + Strongly typed
  + Data trees (hierarchical structure) are fetched (queried)
* **Resolving engine** to evaluate input query

**Schema**

* **The basis is a Schema definition**, max. 3 operations in Schema (1-3)
  + Query - fetching data, asynchronous (concurrency) resolving
  + Mutation - Add, Update and Delete - modification data
    - Main processing is synchronous (in sequence)
  + Subscription - registration on events, returns 'handle'
* Schema must specify type of operation (Query, Mutation, Subscription), only 0..1 per type
* Input vs. Output types
* Basic data types - Scalar (Int, String), Enum, List
* Complex types - only data types, without any methods
* **Introspection**
  + We can inspect GraphQL API (schema) using Introspection
    - It defines data types, queries etc.
    - SDL - Schema Definition Language
    - Schema can be downloaded from GraphQL endpoint
  + Client side object can be generated from Schema
* Reserved names - for some types, fields
* **Nullable by default** - we should allow to return NULL values for any field, query (due asynchronous, concurrency processing on server side)
  + Error per each field, query can be returned

**Resolving**

* Resolving engine - specification, implementation is platform-dependent
* **Resolver**
  + It is implementation of resolving
  + It works on defined Schema
  + Returns suitable output based-on defined input
  + Defined for each Field
* **Resolver result**
  + Return type from resolving
  + Contains data and 2 other sections
    - Error, Error handling
    - Extensions

**Query** - for fetching data through GraphQL endpoint using Query language (from Client side)

* Terms, Naming
  + Field - elementary item, property (attribute)
  + Node -
  + Selection set - result of a query, fetched structured data
    - Query is selection set
* Query with name - named query
* Query with parameter - query including some input parameters
* Query with variables - query including defined variables
* We can use Aliases, Fragments, Named Queries, Variables, Directives
* Pagination - definition how to return paged data set

**Mutation**

* for data modification (Add, Edit and Delete)
* Sequence processing of resolvers
* Input types includes only fields for writting

**Subscription**

* Publish - Subscribe pattern
* WebSocket, Callback, Handle

**Web Services, REST, GraphQL**

REST - uses to define architecture, data oriented

GraphQL - query language, client first oriented

REST vs HTTP API - in principal it is the same

* REST should return some links (URLs, paths) for further navigation

More topics …

**Summary, Best Practices**

* GraphQL - Best Practices <https://graphql.org/learn/best-practices/>
* Build APIs (Schema) using **Client-first Design**
* **GraphQL** is typically **served over HTTP via a single endpoint which expresses the full set of capabilities of the service.** This is in contrast to REST APIs which expose a suite of URLs each of which expose a single resource.
* It is suitable for **Dynamic data**, **Structured data** which are frequently changed in time
* Using **JSON**, however the GraphQL spec does not require it
  + GZip - better performance
* Query vs Mutation - difference in execution (concurrent vs sequence)
* **Nullable type everytime, everywhere**
  + Not sure what result for field will be (or won't be) returned
  + Error can be thrown. In such case error will be in response - client shall (should) handle this error
* Designing APIs with **Pagination** - best practice pattern called "Connections"
* HTTP API and GraphQL shall cooperate
  + synergy, compatibility
* **Use right tools to increase (ensure) performance** - Server-side Batching & Caching
  + DataLoader, Cache …
* Don't hurry up with GraphQL, deployment shall be solved instead (infrastructure)

**Summary, recommendations, experience:**

* 1-3 operations (query, mutation, subscription)
* Use pagination for lists
* Rate limiting - cost specification => analysis of query
* Use GraphQL for dynamic data
  + Static data - HTTP Cache (lookup data), CDN (static content - images)
* Usage mainly from Client (client-side), communication Client-Server
  + gRPC for Server-Server
* async - await (highly recommended), asynchronous (non-blocking) processing
* DataLoader - pattern/component for batch processing
  + for list of Keys
  + query aggregation
* Permissios: Oauth, Token (Bearer Token)
  + Middleware for validation
  + Resolver is wrapped by other component which solves access (params e.g. userId, permissionType, operationType). It is legal use case not execute query when permission is missing.
* Cache only for raw resolvers, not for DataLoader

**Sources, Links**

GraphQL

<https://github.com/graphql>

<https://graphql.org/learn/>

GraphQL - .NET

<https://github.com/graphql-dotnet>

GraphQL ASP.NET Core Tutorial

<https://code-maze.com/graphql-asp-net-core-tutorial/>

<https://code-maze.com/graphql-aspnetcore-basics/>

<https://github.com/CodeMazeBlog/graphql-series/tree/getting-started-with-graphql>

<https://github.com/CodeMazeBlog/graphql-series/tree/consuming-graphql-with-aspnetcore>

SchemaTypes in GraphQL .NET

<https://graphql-dotnet.github.io/docs/getting-started/schema-types/>

.NET Core - Packages

GraphQL

GraphQL.Server.Transports.AspNetCore

*GraphQL.EntityFramework*

GraphQL.Client

A collection of custom GraphQL types like Email, URL and password.

<https://github.com/stekycz/graphql-extra-scalars>

Relay - A JavaScript framework for building data-driven React applications

<https://relay.dev/>

<https://github.com/facebook/relay>

The Apollo Data Graph Platform

<https://www.apollographql.com/>

GraphiQL

An in-browser IDE for exploring GraphQL.

<https://github.com/graphql/graphiql>

DataLoader

<https://github.com/graphql/dataloader>

<https://medium.com/@__xuorig__/the-graphql-dataloader-pattern-visualized-3064a00f319f>