



Advanced Access Control in GraphQL APIs

Group 4: Wilke Klausing, Huy Viet Nguyen



Structure

- Access Control
- Access Control Models
- GraphQL
 - GraphQL vs REST
- Our Task
- Validation Use-Case



Access Control

- Control and monitor access of resources
 - Allows or permits Read/Write/Delete operations
- Subjects, Objects, and Access Rights
- Implemented through different models



Access Control Models

- Mandatory Access Control
 - Administration has authority
- Discretionary Access Control
 - Creator has authority
- Rule-Based Access Control
 - Grants access according to IF ELSE statements
- Purpose-Based Access Control
 - Defines access to purposes

HIGHLY SENSITIVE
SENSITIVE
INTERNAL
PUBLIC



GraphQL

- Query language and server-side runtime
- Queries similar to JSON
- Resolver for each requested field

```
{  
  hero {  
    name  
  }  
}
```

```
{  
  "data": {  
    "hero": {  
      "name": "R2-D2"  
    }  
  }  
}
```



GraphQL vs REST

- REST uses several endpoints
- Overfetching
- Underfetching



Our task

Create a re-usable component for at least one widely used framework/stack for developing GraphQL Web APIs allowing developers to easily add access control to their APIs

Minimum requirement: access-control on a field/parameter basis and at least one advanced access control scheme



Access Control Component

- Functionality:
 - Rule-Based Access Control (basic)
 - Purpose-Based Access Control (advanced)
 - more if we have enough time available



Our Thought Process

1. What languages
does GraphQL
support?

JavaScript	Go	PHP	C# / .NET	Python	Java / Kotlin
Ruby	Rust	Elixir	Swift / Objective-C	Scala	Flutter
Clojure	C / C++	Haskell	Elm	OCaml / Reason	Erlang
Groovy	R	Julia	Perl	D	



Our Thought Process

2. What are the most used JavaScript Frameworks?


- GraphQL.js
- Apollo Server
- Express GraphQL



Our Thought Process


- Game Changer:


APOLLO DOCS

Apollo Server  v2 


Introduction


Get started with Apollo Server


Changelog 


Apollo Federation 

 EXPAND ALL

DEFINING A SCHEMA 

FETCHING DATA 

TESTING 

PERFORMANCE 

Search Apollo Docs



Launch Apollo Studio →

Plugins

Extend Apollo Server with custom functionality

Plugins are available in Apollo Server 2.2.x and later.

Plugins enable you to extend Apollo Server's core functionality by performing custom operations in response to certain events. Currently, these events correspond to individual phases of the GraphQL request lifecycle, and to the startup of Apollo Server itself.

For example, a basic logging plugin might log the GraphQL query string associated

Plugins

Creating a plugin

- Responding to events
- Request lifecycle event flow
- End hooks
- Inspecting request and response details



Access Control Plugin

- System Integration:

```
1  ▾ const server = new ApolloServer({  
2    typeDefs,  
3    resolvers,  
4  
5    // You can import plugins or define them in-line, as shown:  
6  ▾  plugins: [  
7  
8    /* This plugin is imported in-place. */  
9    require('./ourAccessControlPlugin'),  
10  
11  ],  
12  })
```



Access Control Plugin

- Configuration:
 - No database, only two configuration files:
 1. Purpose tree as configuration file
 2. Rules as configuration file
- Goal:
 - Flexible
 - Reuseable
 - Developer Friendly




Validation Use-Case





Validation Use-Case

Healthcare Database



Username

Password

Login



Validation Use-Case:

- Reasons to share health data:
 - Avoid medication errors
 - Avoid duplicate testing
 - Genetic studies
 - Chronic disease registries
 - Substance abuse
 - Population health management
 - Larger scale analytics
 - Epidemiology/disease tracking



Thank you for your attention.