Insecure GraphQL

/b1twis3

GraphQL

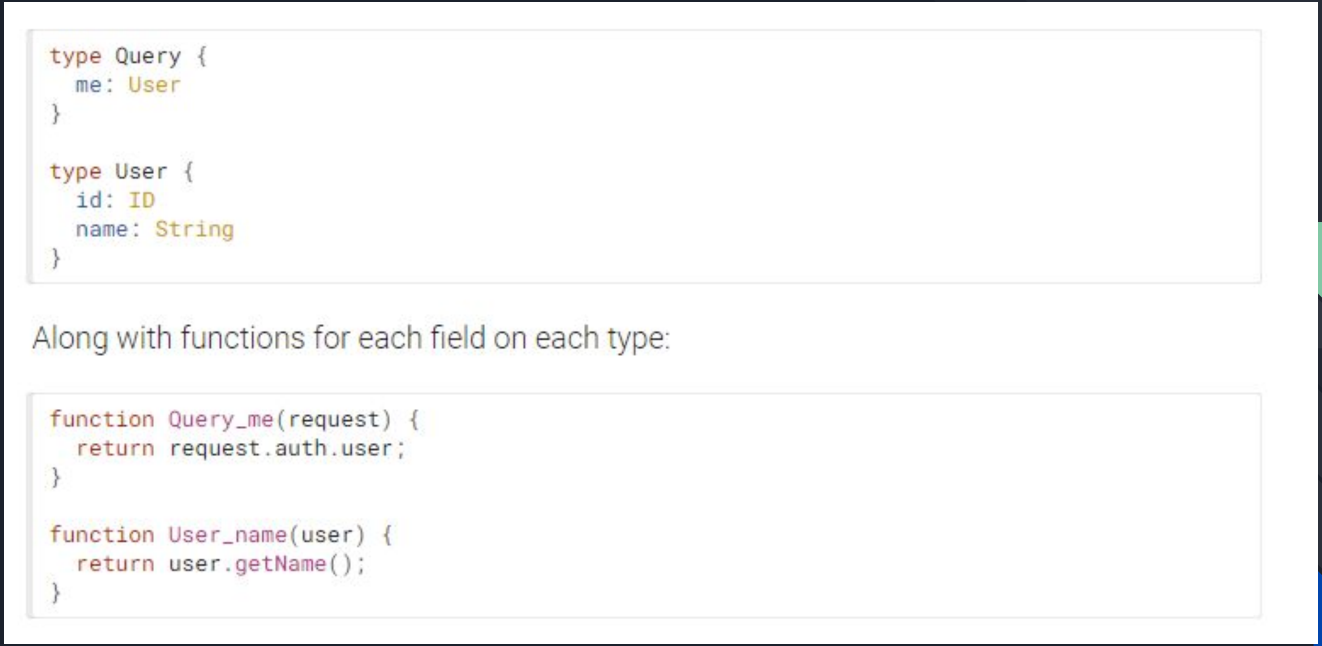
1. “GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data”

2. “GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data”

3. “A GraphQL service is created by defining types and fields on those types, then providing functions for each field on each type”

https://graphql.org/learn/

GraphQL



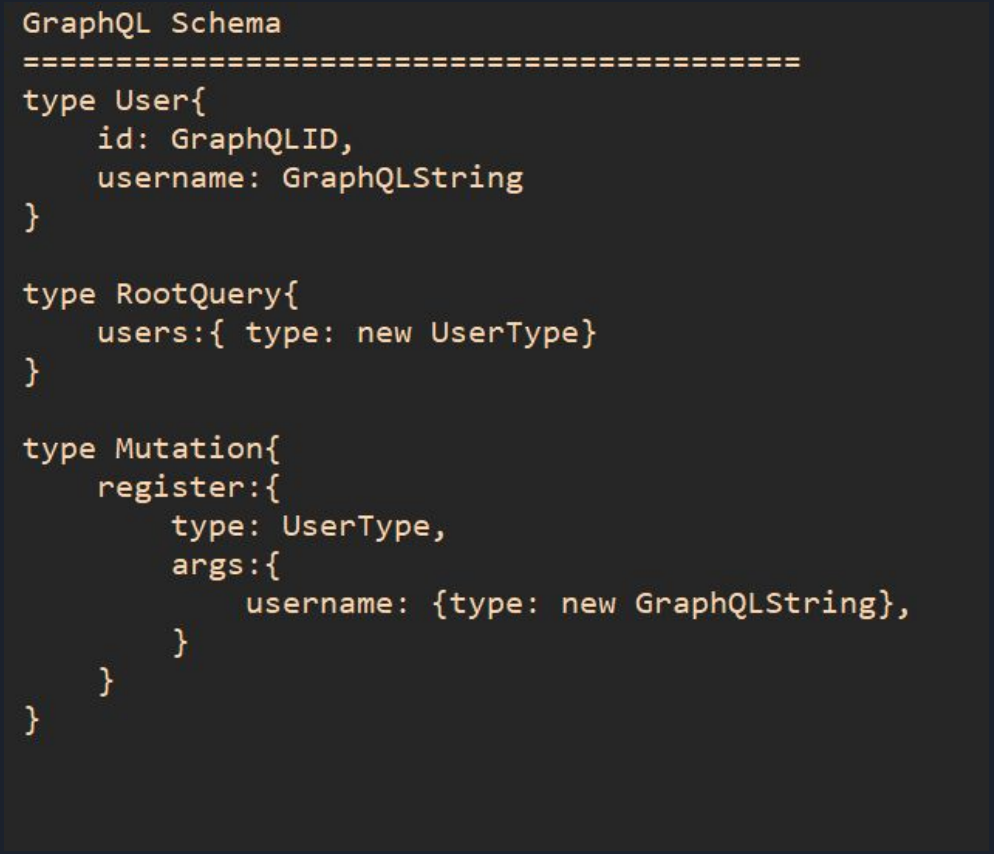
```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Along with functions for each field on each type:

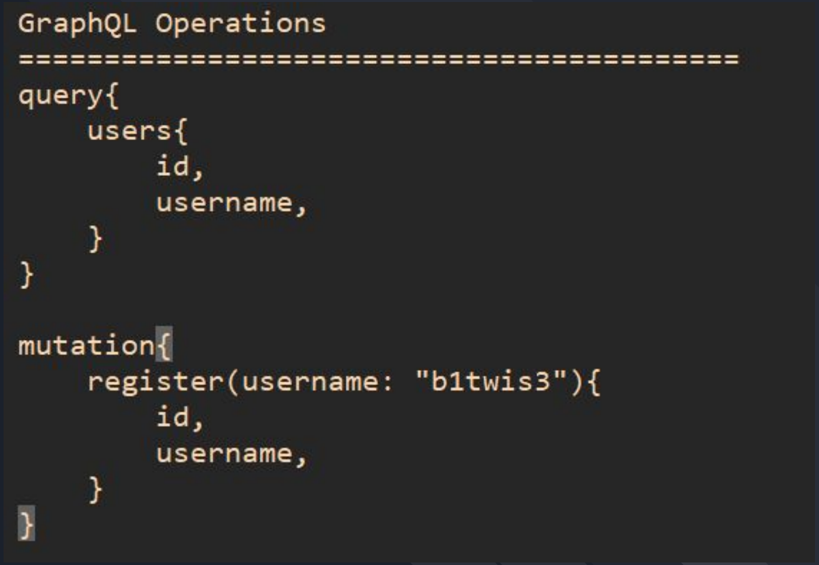
```
function Query_me(request) {
  return request.auth.user;
}

function User_name(user) {
  return user.getName();
}
```

https://graphql.org/learn/

GraphQL

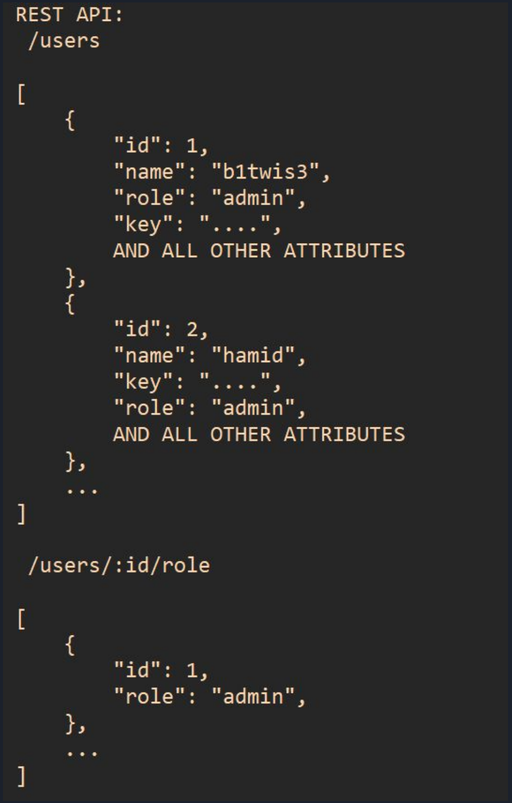


```
GraphQL Schema
==========================================
type User{
    id: GraphQLID,
    username: GraphQLString
}

type RootQuery{
    users:{ type: new UserType}
}

type Mutation{
    register:{
        type: UserType,
        args:{
            username: {type: new GraphQLString},
        }
    }
}
```



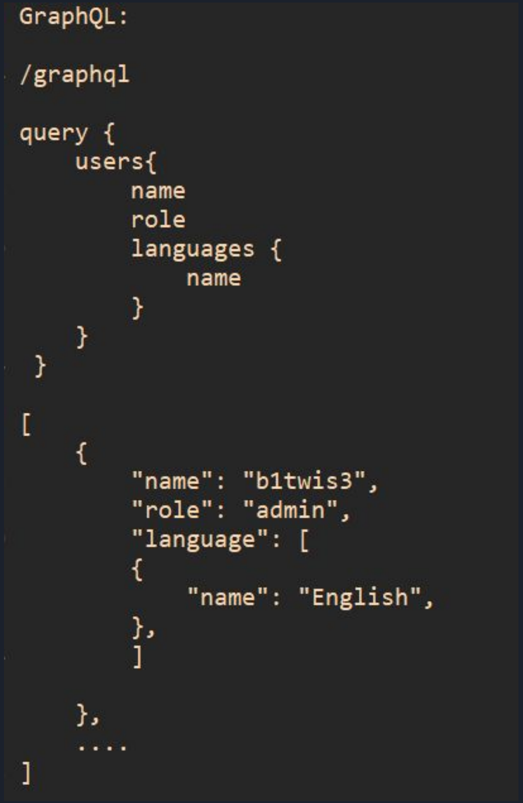
```
GraphQL Operations
==========================================
query{
    users{
        id,
        username,
    }
}

mutation{
    register(username: "b1twis3"){
        id,
        username,
    }
}
```

GraphQL [REST API vs GraphQL]



```
REST API:
 /users

[
    {
        "id": 1,
        "name": "b1twis3",
        "role": "admin",
        "key": "....",
        AND ALL OTHER ATTRIBUTES
    },
    {
        "id": 2,
        "name": "hamid",
        "key": "....",
        "role": "admin",
        AND ALL OTHER ATTRIBUTES
    },
    ...
]

 /users/:id/role

[
    {
        "id": 1,
        "role": "admin",
    },
    ...
]
```

VS

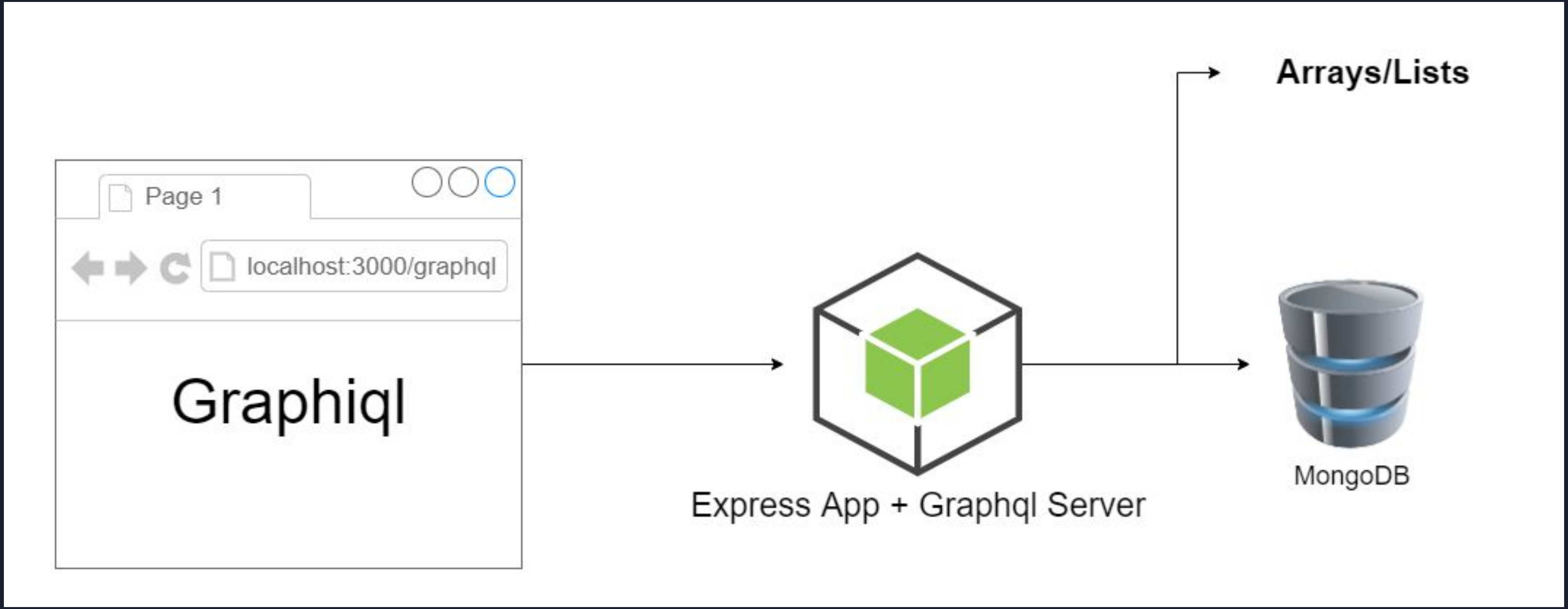


```
GraphQL:

/graphql

query {
    users{
        name
        role
        languages {
            name
        }
    }
 }

[
    {
        "name": "b1twis3",
        "role": "admin",
        "language": [
        {
            "name": "English",
        },
        ]

    },
    ....
]
```

GraphQL [Insecure]

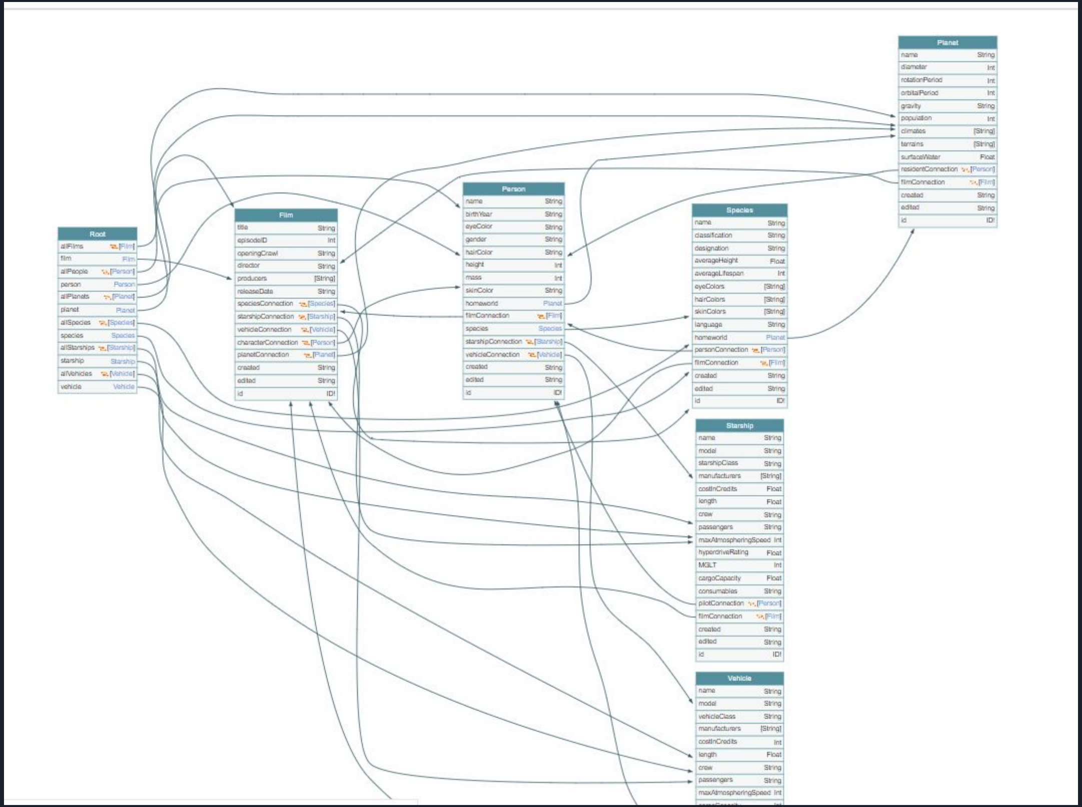
- Introspection
- Input Validation Issues
- DoS
- Information Disclosure
- IDOR
- Broken Auth
- NoSQL/SQL Injection
- Logic Flaws

GraphQL [Overview]



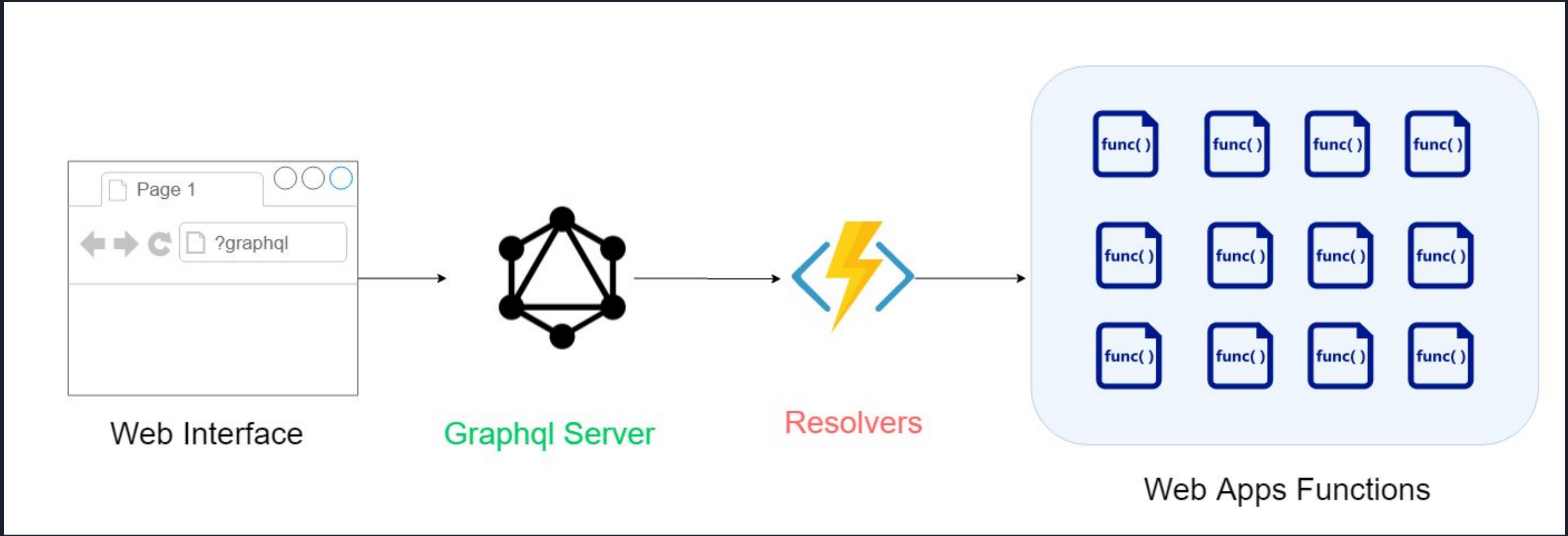
Demo 0
Demo 1 (Mutation)

GraphQL [Introspection]



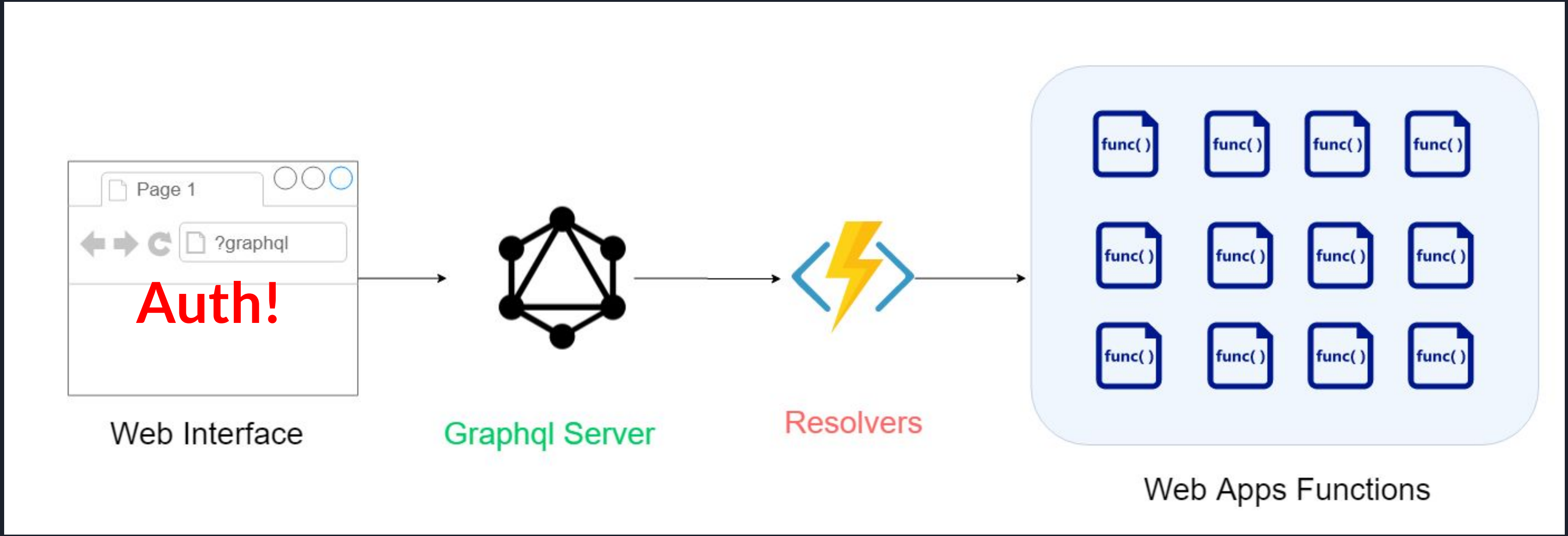
https://apis.guru/graphql-voyager/
https://graphql.org/learn/introspection/

GraphQL [AuthZ]



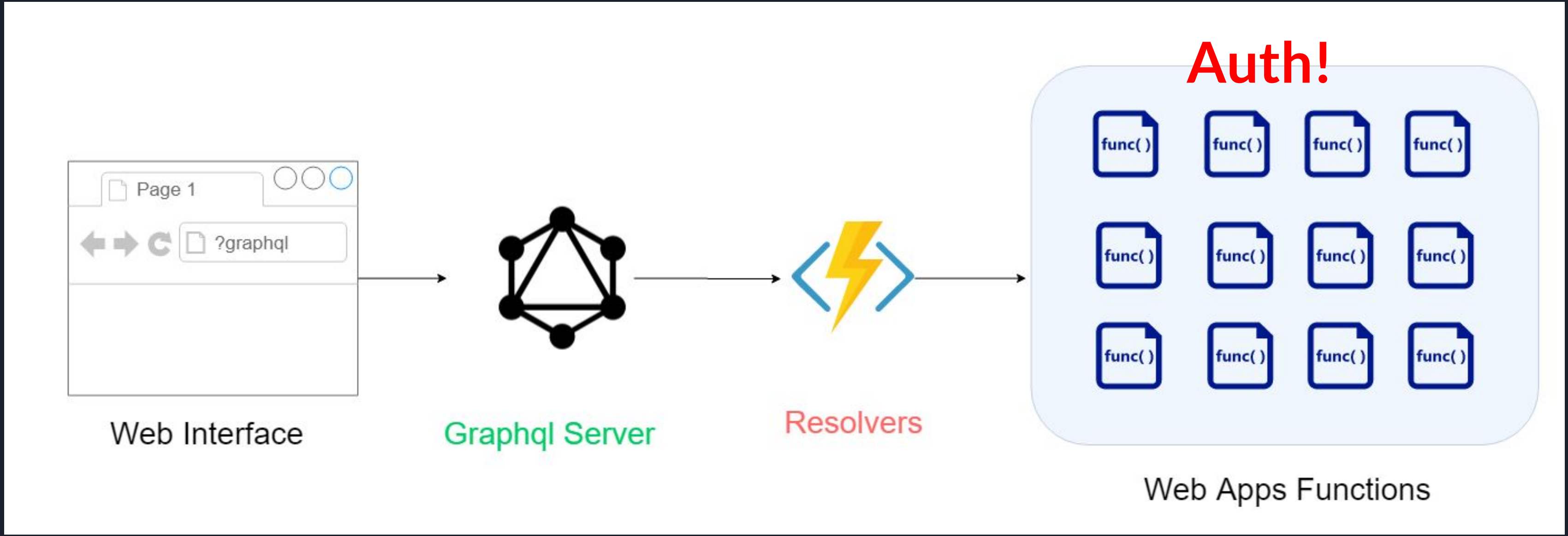
Demo 2

GraphQL [AuthZ]



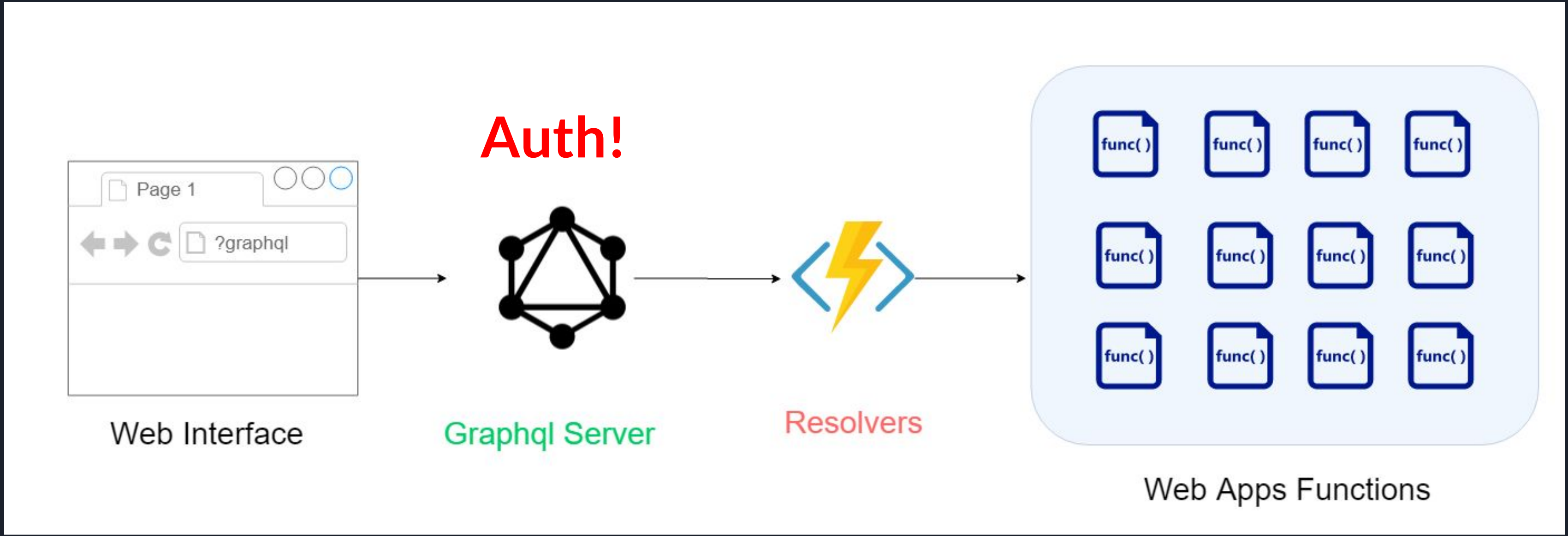
Demo 2

GraphQL [AuthZ]



Demo 2

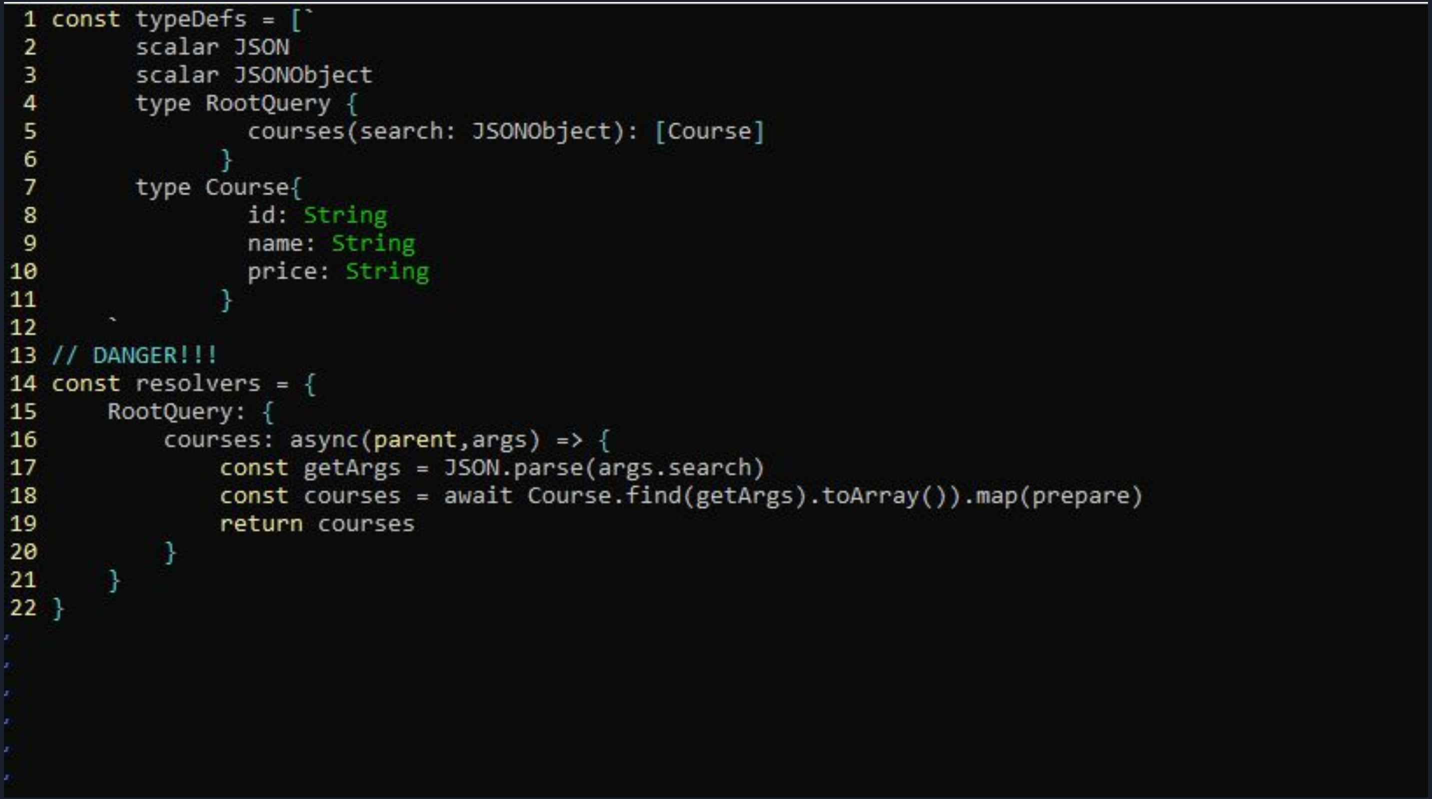
GraphQL [AuthZ]



Demo 2

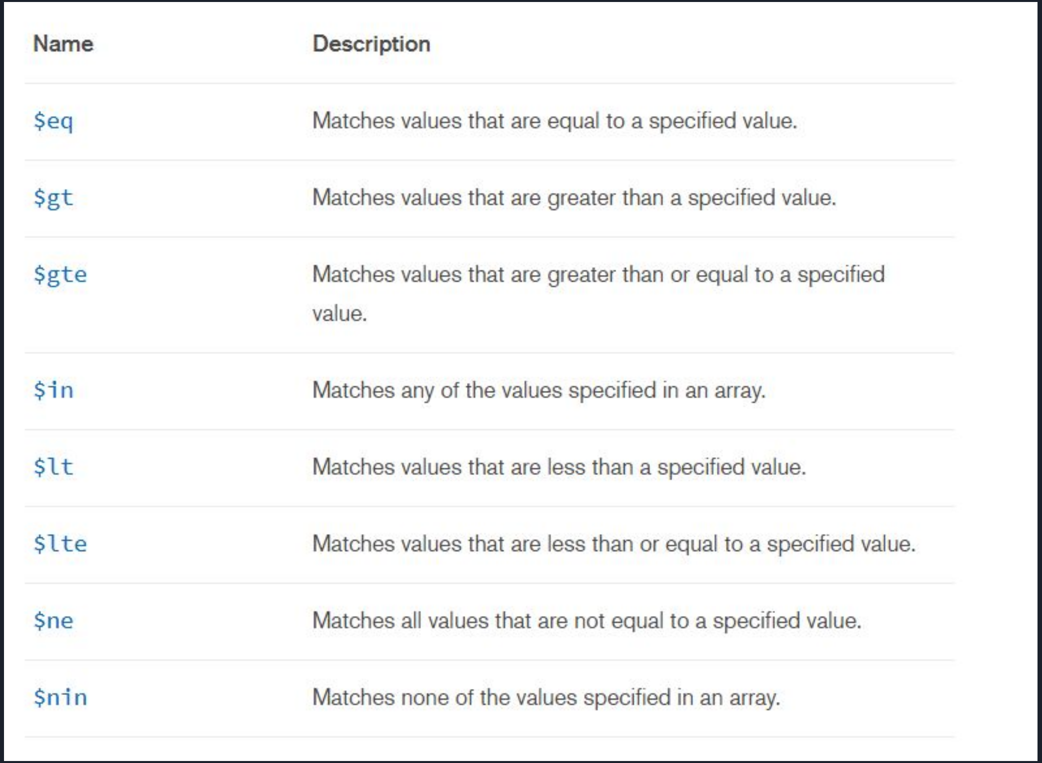
Directives https://blog.logrocket.com/graphql-directives-are-underrated/

GraphQL [JSON]



```
const typeDefs = [`
      scalar JSON
      scalar JSONObject
      type RootQuery {
              courses(search: JSONObject): [Course]
            }
      type Course{
              id: String
              name: String
              price: String
            }
    `
// DANGER!!!
const resolvers = {
    RootQuery: {
        courses: async(parent,args) => {
            const getArgs = JSON.parse(args.search)
            const courses = await Course.find(getArgs).toArray()).map(prepare)
            return courses
        }
    }
}
```

GraphQL [Input Validation]

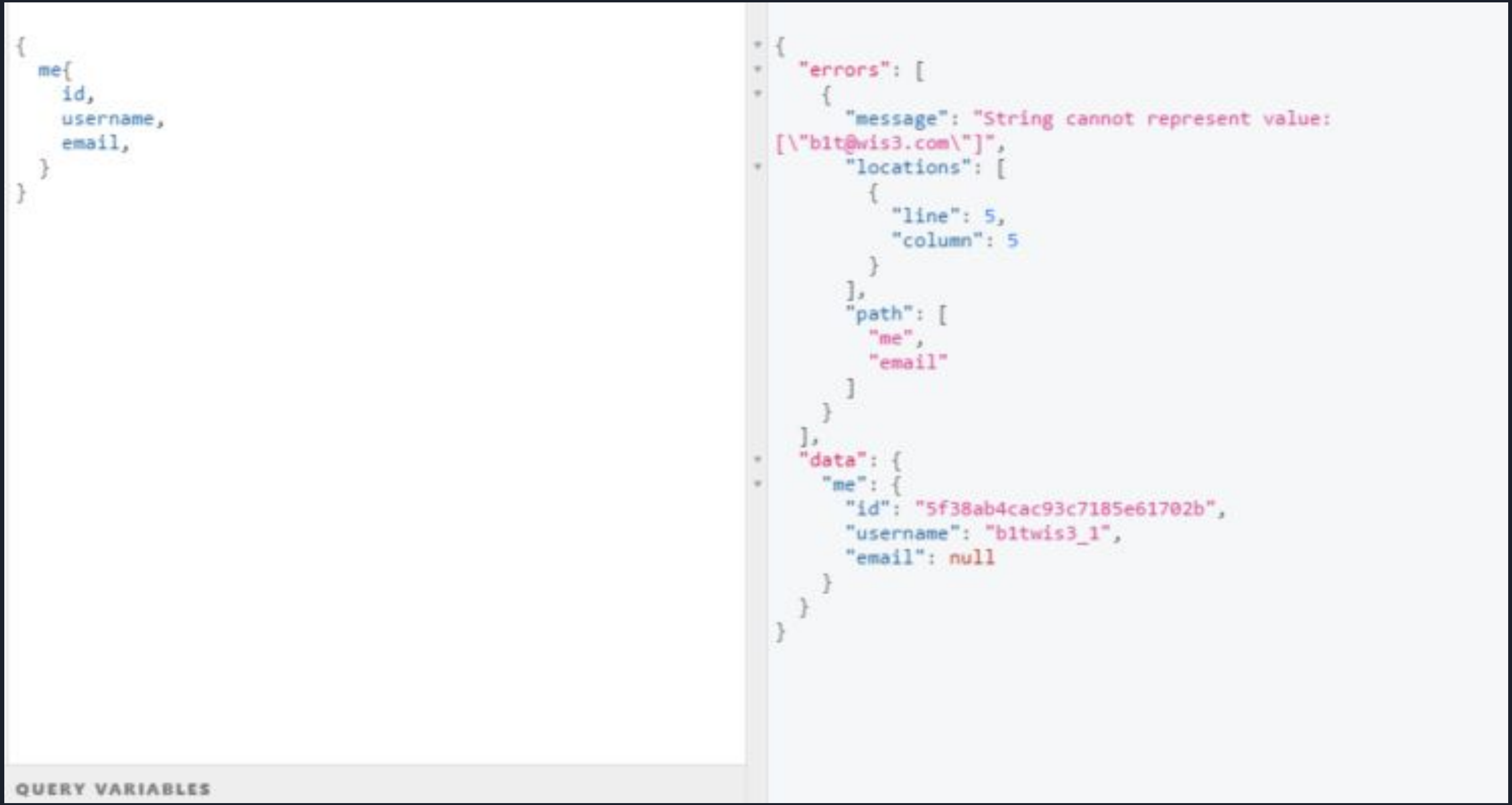


Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

MongoDB Query Operators

https://docs.mongodb.com/manual/reference/operator/query/#query-selectors

GraphQL [Type Mismatching]



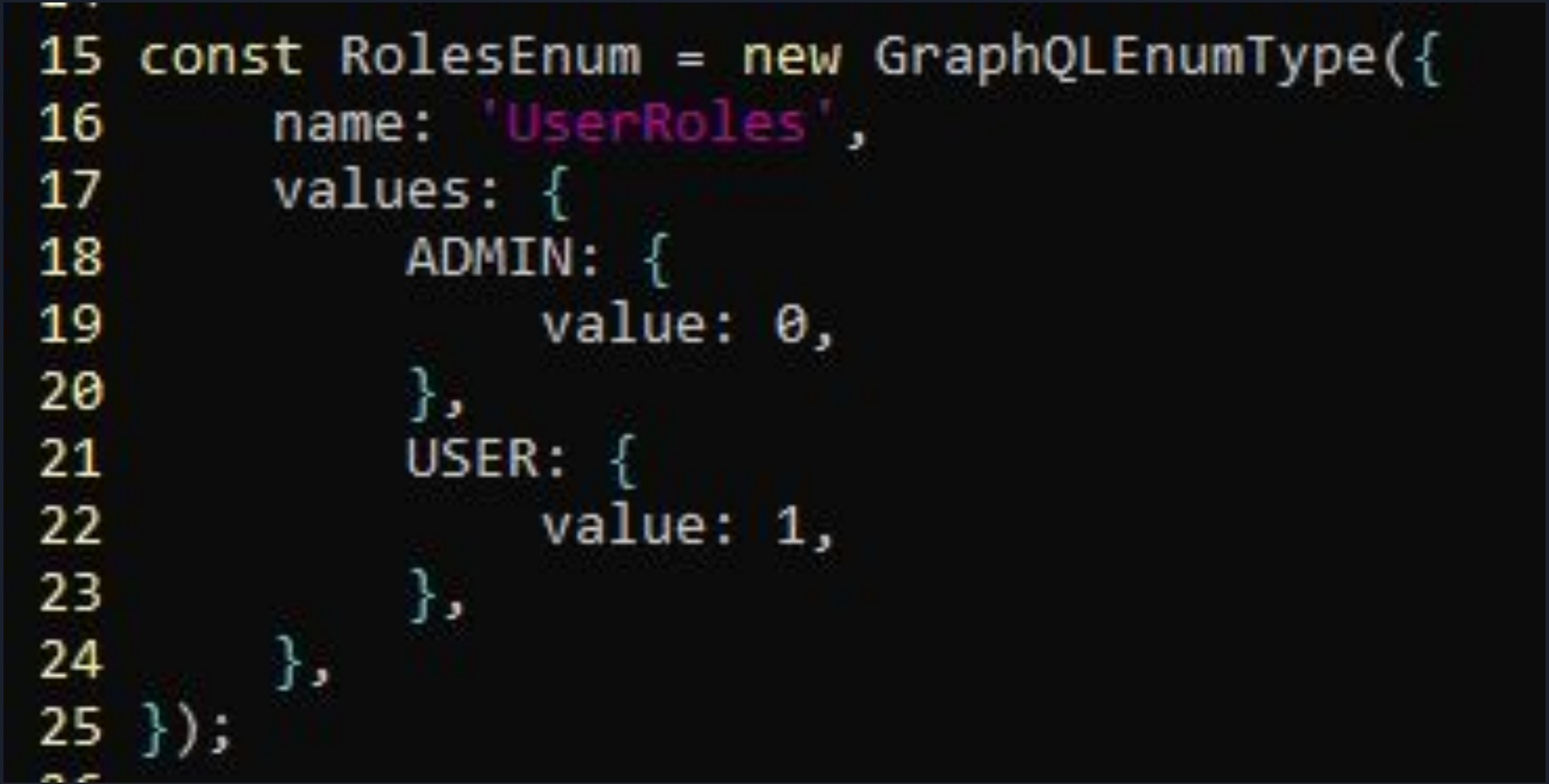
```
{
  me{
    id,
    username,
    email,
  }
}
```

QUERY VARIABLES

```
{
  "errors": [
    {
      "message": "String cannot represent value:
[\"b1t@wis3.com\"]",
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ],
      "path": [
        "me",
        "email"
      ]
    }
  ],
  "data": {
    "me": {
      "id": "5f38ab4cac93c7185e61702b",
      "username": "b1twis3_1",
      "email": null
    }
  }
}
```

Demo 3

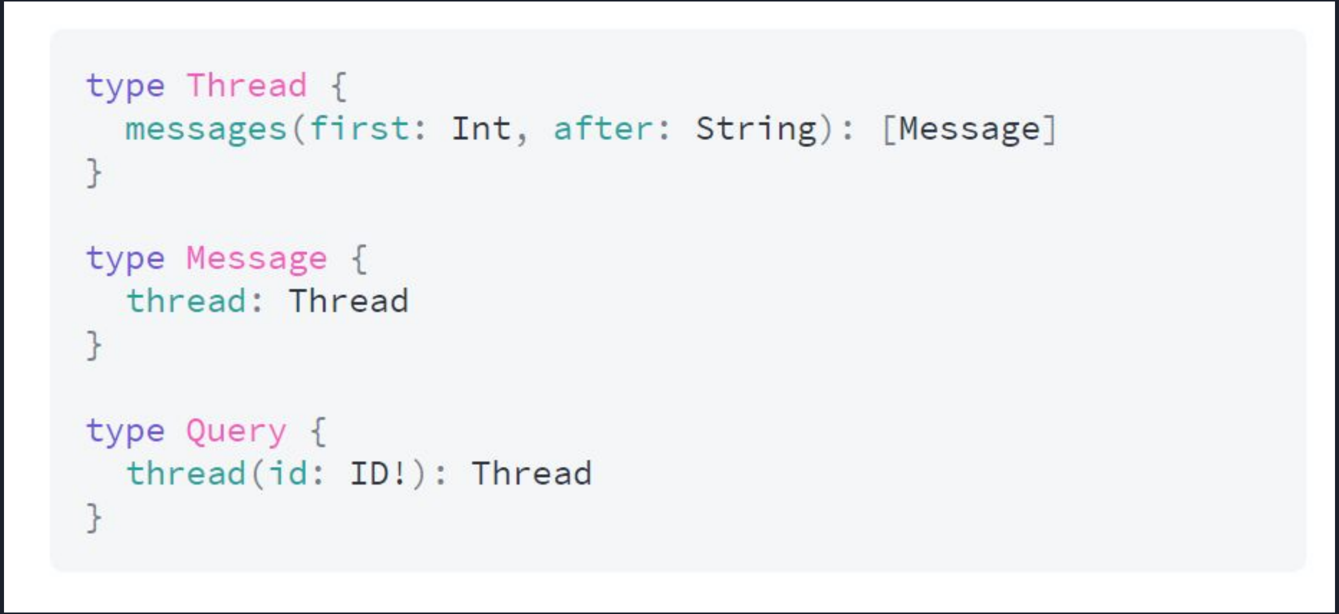
GraphQL [Enum Types]



```
const RolesEnum = new GraphQLEnumType({
    name: 'UserRoles',
    values: {
        ADMIN: {
            value: 0,
        },
        USER: {
            value: 1,
        },
    },
});
```

Demo 3

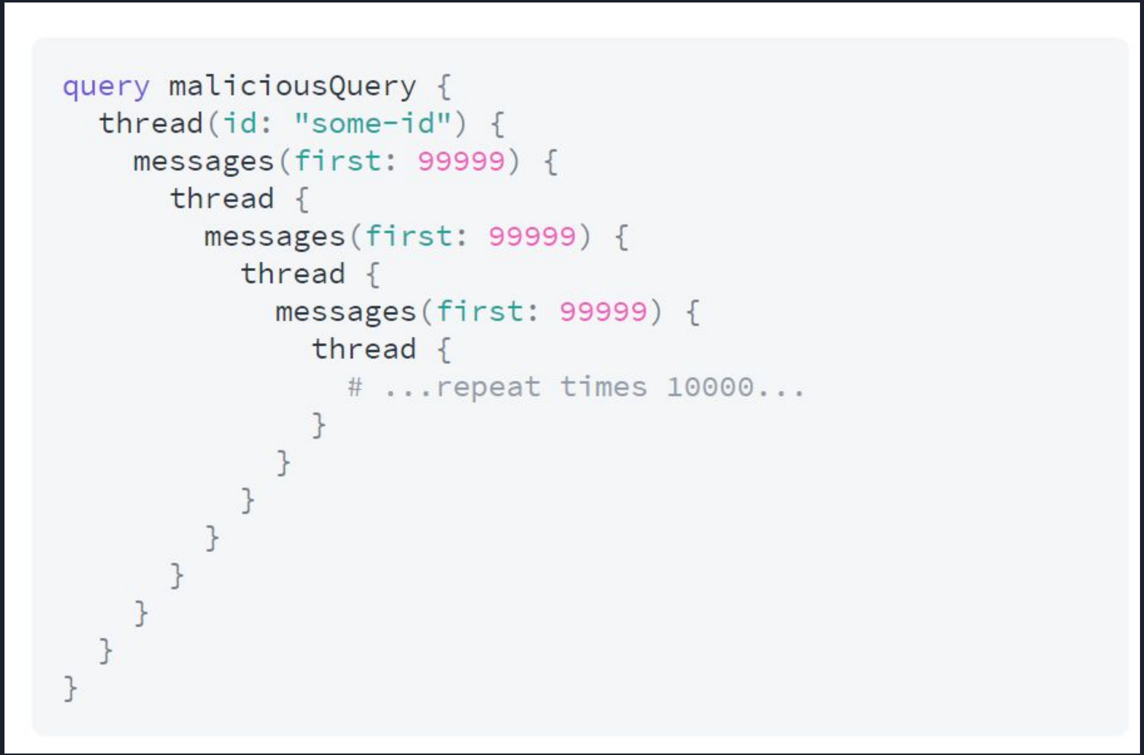
GraphQL [Max Query Depth]



```
type Thread {
  messages(first: Int, after: String): [Message]
}

type Message {
  thread: Thread
}

type Query {
  thread(id: ID!): Thread
}
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/

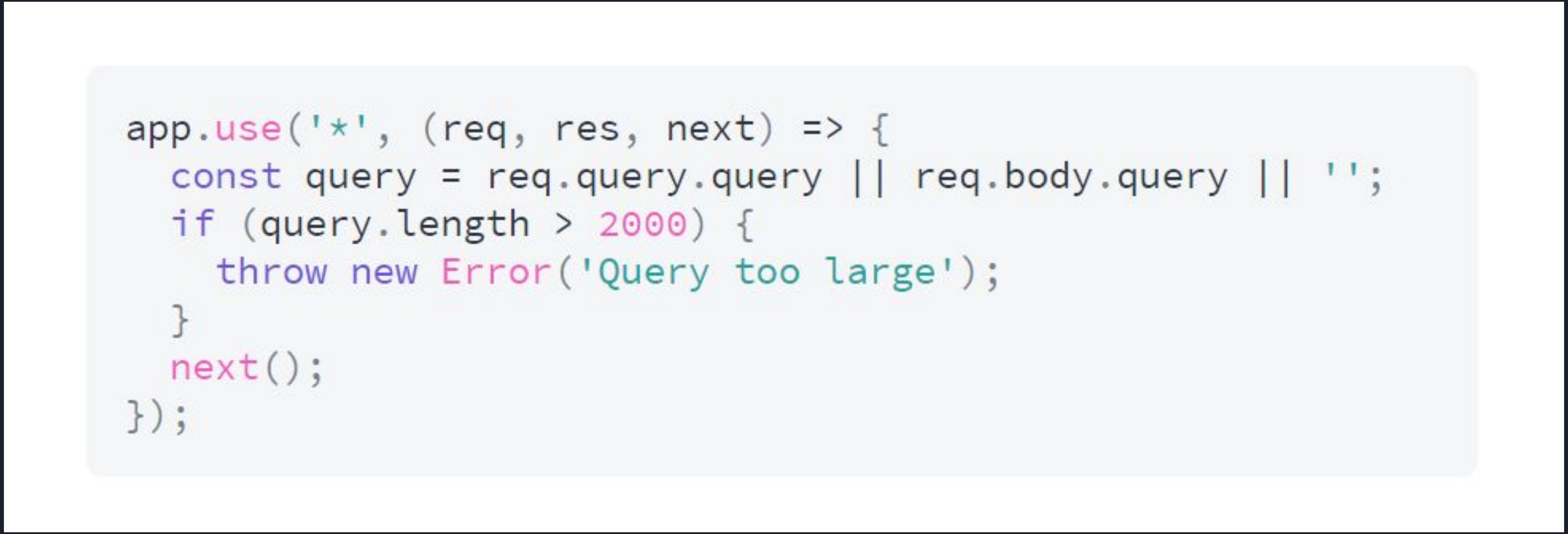
GraphQL [Max Query Depth]



```
query maliciousQuery {
  thread(id: "some-id") {
    messages(first: 99999) {
      thread {
        messages(first: 99999) {
          thread {
            messages(first: 99999) {
              thread {
                # ...repeat times 10000...
              }
            }
          }
        }
      }
    }
  }
}
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/

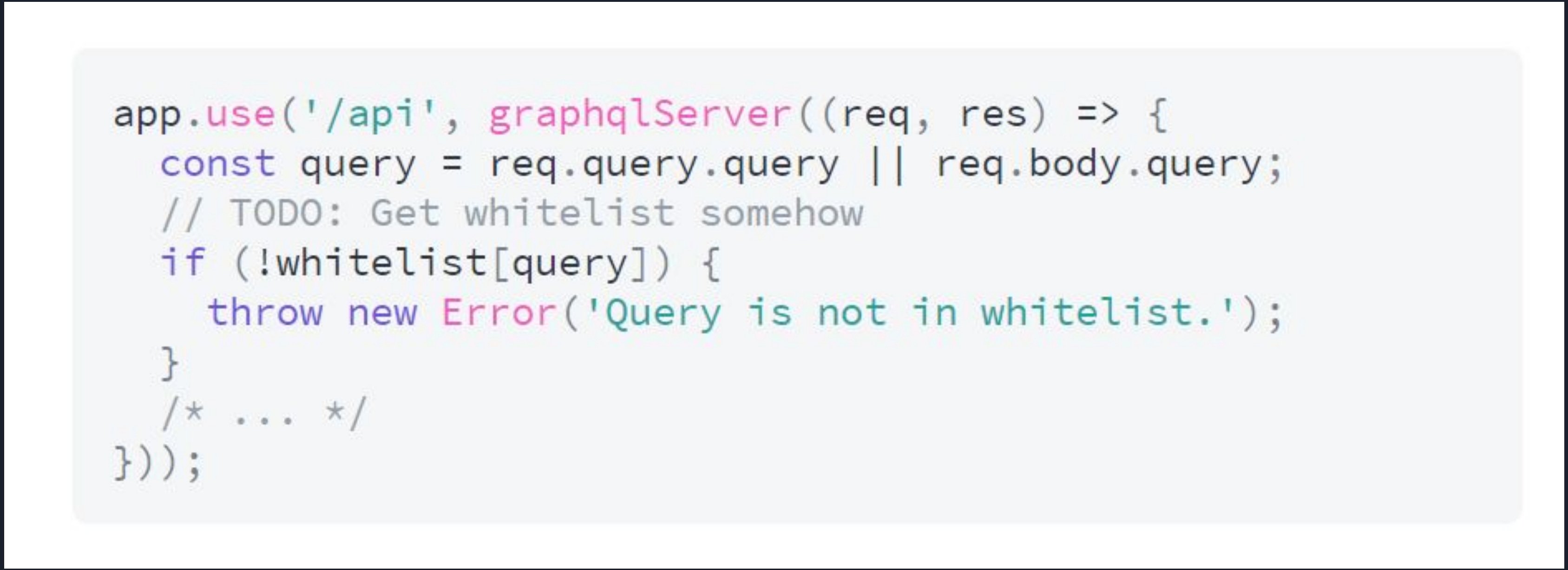
GraphQL [Max Query Depth]



```
app.use('*', (req, res, next) => {
  const query = req.query.query || req.body.query || '';
  if (query.length > 2000) {
    throw new Error('Query too large');
  }
  next();
});
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/

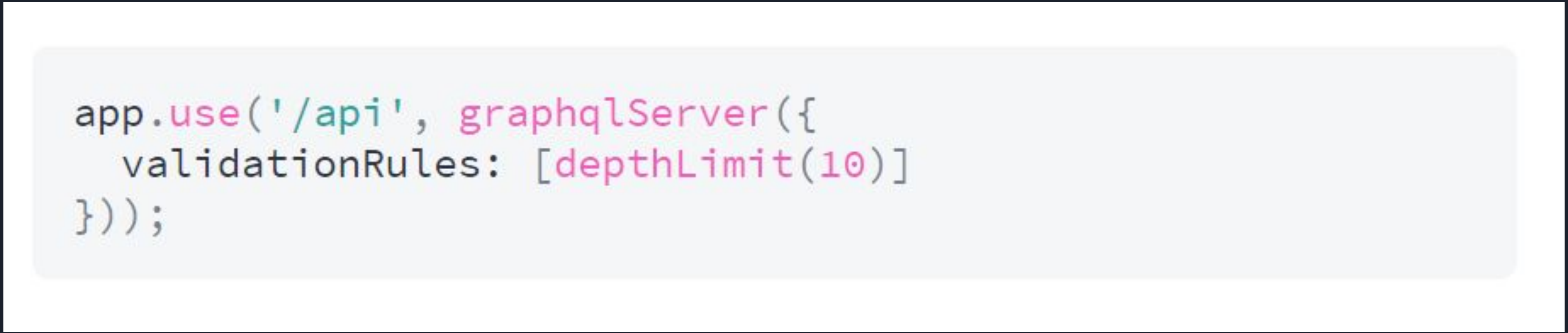
GraphQL [Max Query Depth]



```
app.use('/api', graphqlServer((req, res) => {
  const query = req.query.query || req.body.query;
  // TODO: Get whitelist somehow
  if (!whitelist[query]) {
    throw new Error('Query is not in whitelist.');
  }
  /* ... */
}));
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/

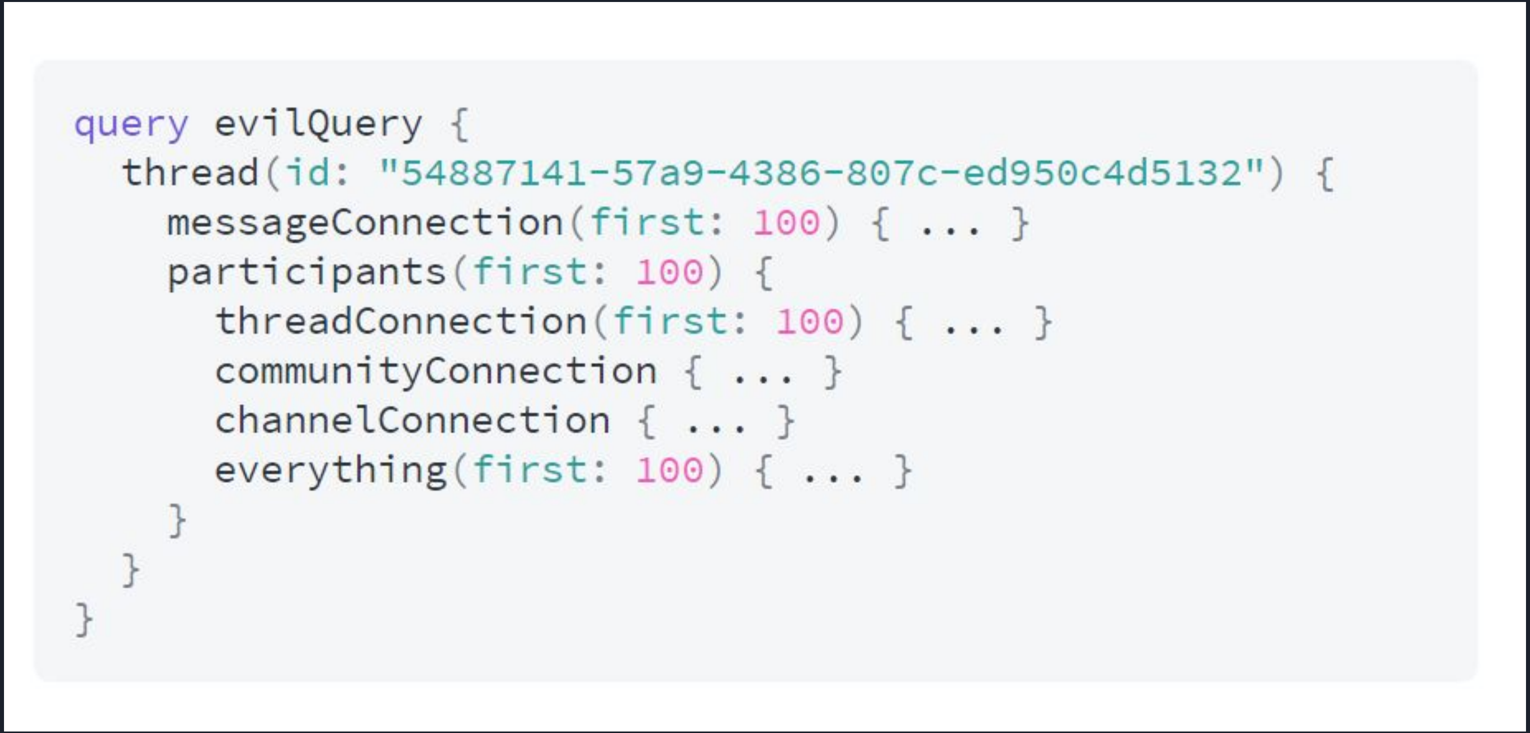
GraphQL [Max Query Depth]



```
app.use('/api', graphqlServer({
  validationRules: [depthLimit(10)]
}));
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/
npm: https://github.com/stems/graphql-depth-limit

GraphQL [Query Complexity]



```
query evilQuery {
  thread(id: "54887141-57a9-4386-807c-ed950c4d5132") {
    messageConnection(first: 100) { ... }
    participants(first: 100) {
      threadConnection(first: 100) { ... }
      communityConnection { ... }
      channelConnection { ... }
      everything(first: 100) { ... }
    }
  }
}
```

Source: https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b/
https://developer.github.com/v4/guides/resource-limitations/