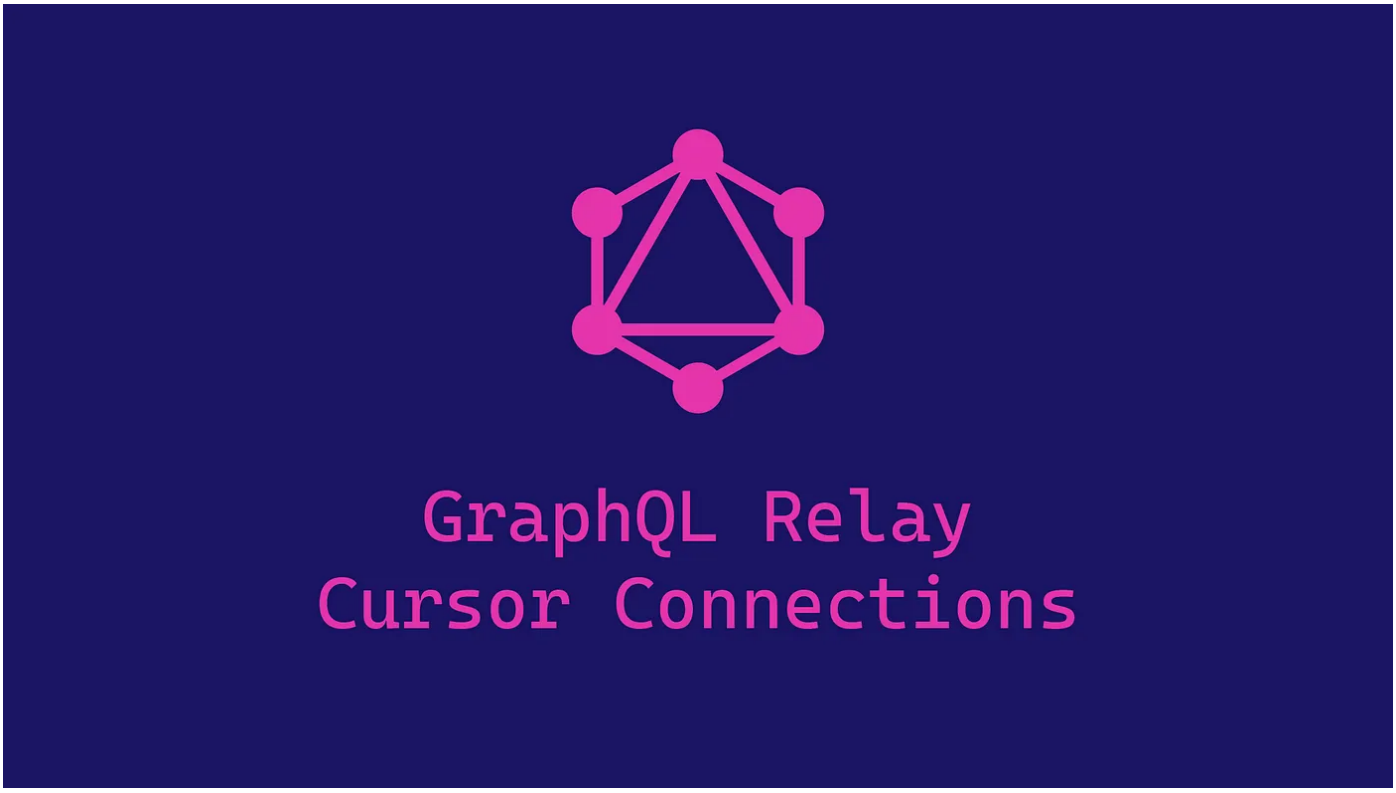
GraphQL Cursor Connections with Filtering and Ordering

Riley Conrardy · Follow

3 min read · Jul 18, 2022



The GraphQL Relay Cursor Connections specification illustrates how to implement Cursor Pagination in GraphQL. However, it does not describe how to implement more advanced query features like filtering and ordering. In this article, I will describe how to implement the GraphQL Schema for filtering and ordering with GraphQL Relay Cursor Connections. To see an example of an Apollo Server that actually implements this schema checkout the following GitHub repository!

GitHub Repository: https://github.com/rconrardy/GraphQL-Cursor-Connections-With-Filtering-and-Ordering

Reserved Types

To ensure consistency throughout your GraphQL Schema, you should reserve the following types in addition to the types already reserved in the GraphQL Relay Cursor Connections specification:

- Any type starting with FilterBy
- Any type ending with FilterBy
- Any type ending with OrderBy
- An enum named OrderByDirection

Types starting with FilterBy should describe how to filter Scalar types such as ID, Boolean, Int, and String. Here are a few examples:

```
input FilterByBoolean {
    equals: Boolean
    not: Boolean
}

input FilterByID {
    contains: ID
    endsWith: ID
    equals: ID
    gt: ID
    gte: ID
    in: [ID]
    lt: ID
    lte: ID
    mode: ID
    not: ID
    notIn: [ID]
    startsWith: ID
}

input FilterByInt {
    equals: Int
    gt: Int
    gte: Int
    in: [Int]
```

```
    lt: Int
    lte: Int
    not: Int
    notIn: [Int]
}

input FilterByString {
    contains: String
    endsWith: String
    equals: String
    gt: String
    gte: String
    in: [String]
    lt: String
    lte: String
    mode: String
    not: String
    notIn: [String]
    startsWith: String
}

input FilterByDate {
    equals: String
    gt: String
    gte: String
    in: [String]
    lt: String
    lte: String
    not: String
    notIn: [String]
}
```

Types ending with FilterBy should describe how to filter Object types such as User. Here is an example:

```
input UserFilterBy {
    and: [UserFilterBy!]
    or: [UserFilterBy!]
    id: FilterByID
    points: FilterByInt
    name: FilterByString
    isAdmin: FilterByBoolean
    createdAt: FilterByDate
}
```

Types ending with OrderBy should describe how to order Object types such as User. Here is an example:

```
input UserOrderBy {
    then: UserOrderBy
    points: OrderByDirection
    name: OrderByDirection
    isAdmin: OrderByDirection
    createdAt: OrderByDirection
}
```

The enum OrderByDirection should only consist of two values describing the direction in which to order Object types (i.e. ascending and descending). Here is an example:

```
enum OrderByDirection {
    ASCENDING
    DESCENDING
}
```

Queries

Once you have created the reserved types you need for your schema you can add them to your Cursor Connection queries like so:

```
type PageInfo {
    hasNextPage: Boolean!
    hasPreviousPage: Boolean!
    startCursor: String!
    endCursor: String!
}

type User {
    id: ID
    points: Int
    name: String
    isAdmin: Boolean
    createdAt: String
}

type UserEdge {
    cursor: String!
```

```
    node: User!
}

type UserConnection {
    pageInfo: PageInfo!
    edges: [UserEdge!]!
}

type Query {
    users(
        first: Int,
        after: String,
        filterBy: UserFilterBy
        orderBy: UserOrderBy
    ): UserConnection
}
```

Conclusion

Once you have added filtering and ordering to your GraphQL Schema you can construct complex queries that look like the following:

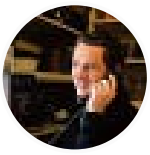
```
{
    "query": "
        query GetUsers(
            $first: Int,
            $after: String,
            $filterBy: UserFilterBy,
            $orderBy: UserOrderBy
        ) {
            users(
                first: $first,
                after: $after,
                filterBy: $filterBy,
                orderBy: $orderBy
            ) {
                pageInfo {
                    hasNextPage
                    hasPreviousPage
                    startCursor
                    endCursor
                }
                edges {
                    cursor
                    node {
                        id
                        points
```

```
                            name
                            isAdmin
                            createdAt
                        }
                    }
                }
            }
        ",
        "variables": {
            "first": 10,
            "after": "TW9jayBDdXJzb3I="
            "filterBy": {
                "and": [
                    { "points": { "gte": 100 } },
                    { "points": { "lt": 200 } }
                ]
            },
            "orderBy": {
                "points": "DESCENDING",
                "then": {
                    "name": "ASCENDING",
                }
            }
        }
    }
```

To see an example of an Apollo Server that actually implements this GraphQL Schema check out the following GitHub repository!

GitHub Repository: https://github.com/rconrardy/GraphQL-Cursor-Connections-With-Filtering-and-Ordering

GraphQL Relay Cursor Based Pagination Programming API

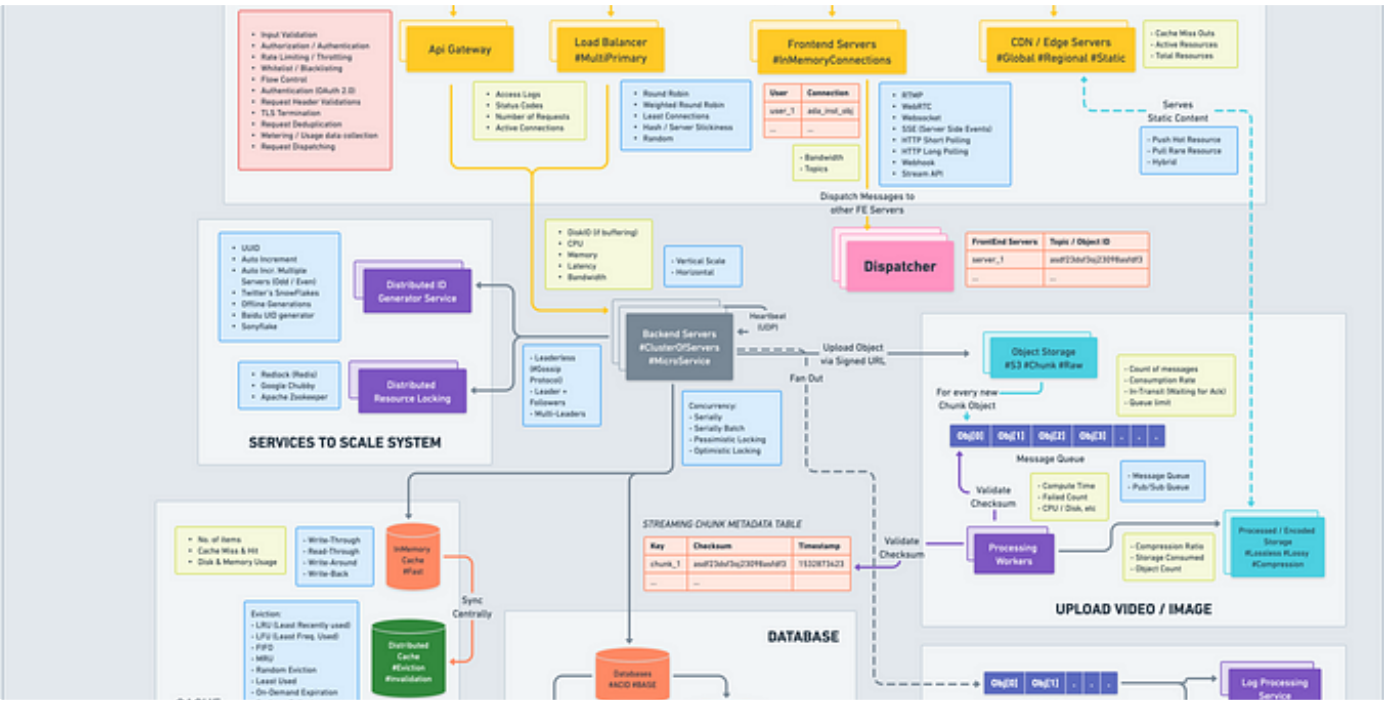


Follow

Written by Riley Conrardy

4 Followers

Recommended from Medium



Love Sharma in Dev Genius

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

· 9 min read · Apr 20

896 10

omgzui in Javarevisited

How to Design a Permission System, You Can Do this

Permission System can be simply understood as power restriction, that is, different people have different powers, and what they see and...

· 8 min read · Jan 5

78

Lists

Stories to Help You Grow as a Software Developer
19 stories · 44 saves

Leadership
30 stories · 15 saves

Company Offsite Reading List
8 stories · 4 saves

How to Run More Meaningful 1:1 Meetings
11 stories · 20 saves

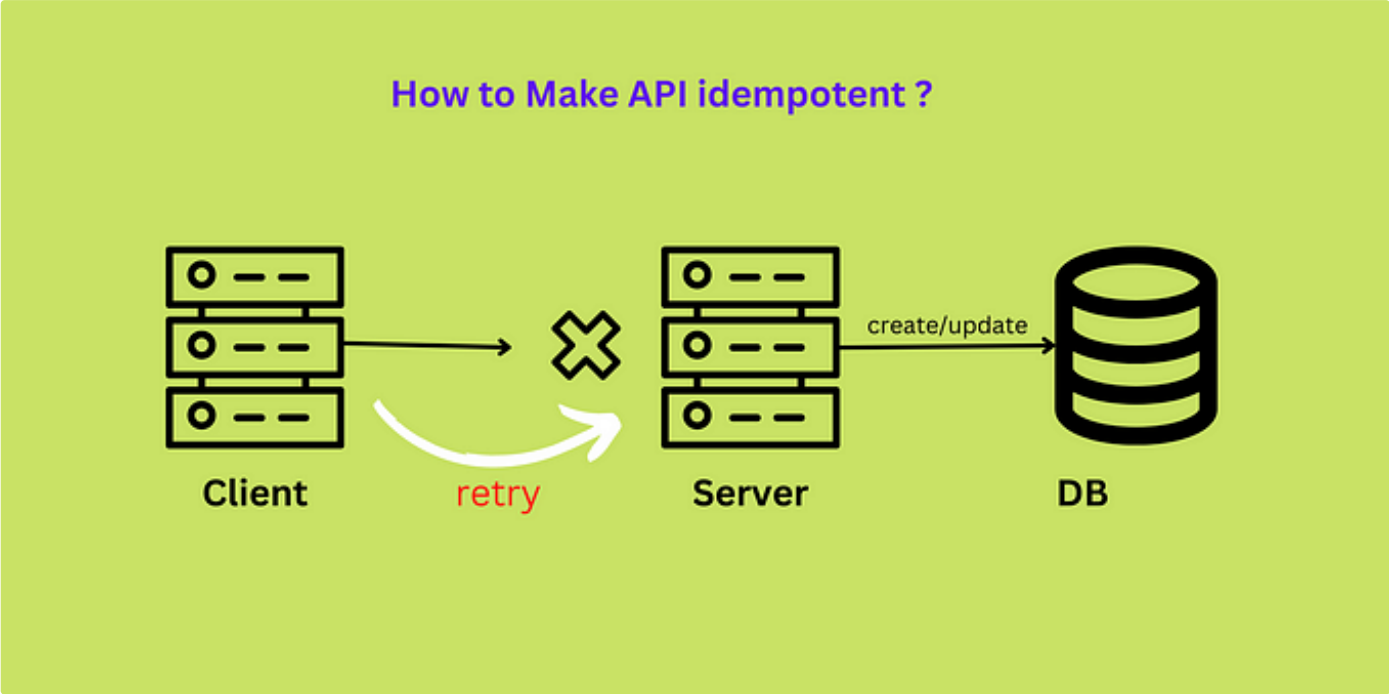
Frontend Jirachi in Bootcamp

A Guide to Implementing Refresh Tokens in JavaScript

Introduction

· 10 min read · Jan 17

96 1



Suraj Mishra in Level Up Coding

How to Implement Idempotent API (Part 1)

Discussing idempotency concept and implementation design

· 5 min read · Jan 26

84 1