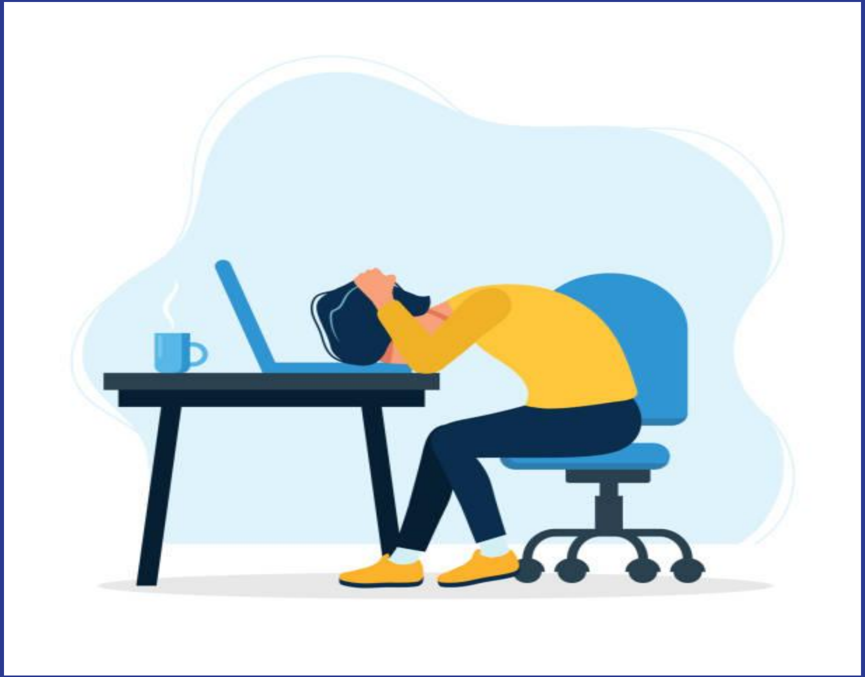
https://www.linkedin.com/in/mehreentahir93/

martha & MARLEY SPOON
COOKING REDESIGNED
martha & MARLEY SPOON
martha & MARLEY SPOON



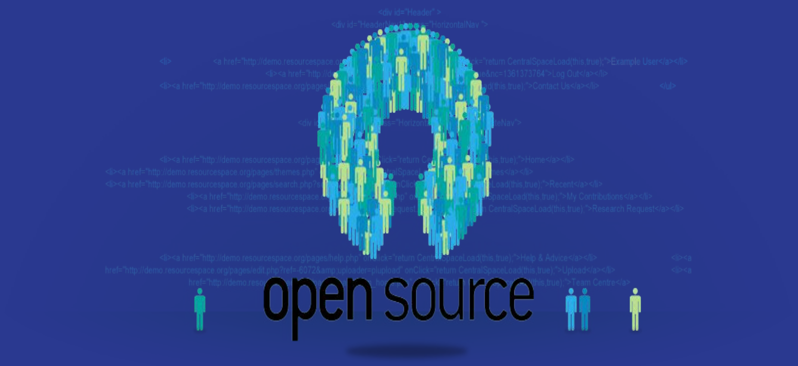
INTEGRATING GRAPHQL WITH RUBY ON RAILS

WEBINAR ROADMAP

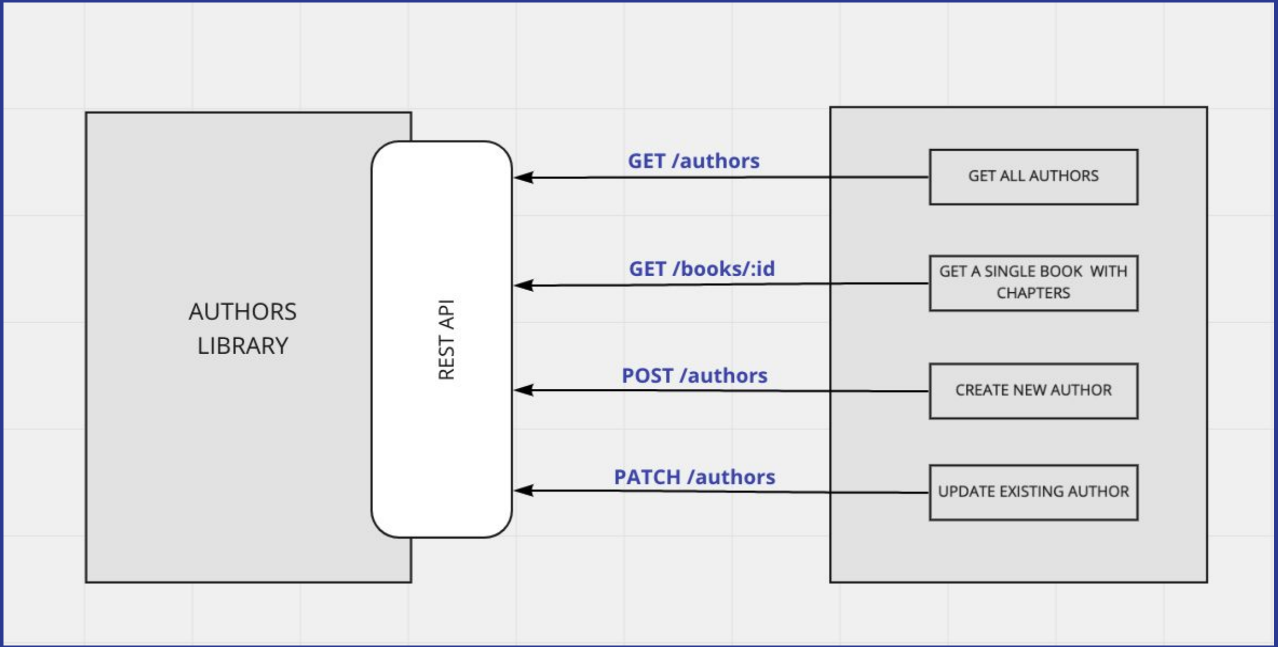
- ❏ What Is GraphQL
- ❏ Typical REST Application
- ❏ REST Potential Drawback
- ❏ Graphql Application
- ❏ Graphql Operations
- ❏ Proof Of Concept Application

WHAT IS HYPE ABOUT

WHY & HOW IT CAME INTO EXISTENCE



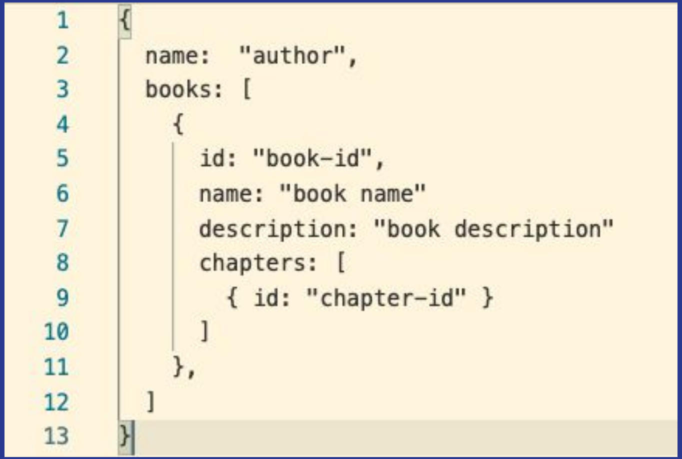
ARCHITECTURE



SITUATIONS

UNDER FETCHING

Need To Display Single Author With All Details

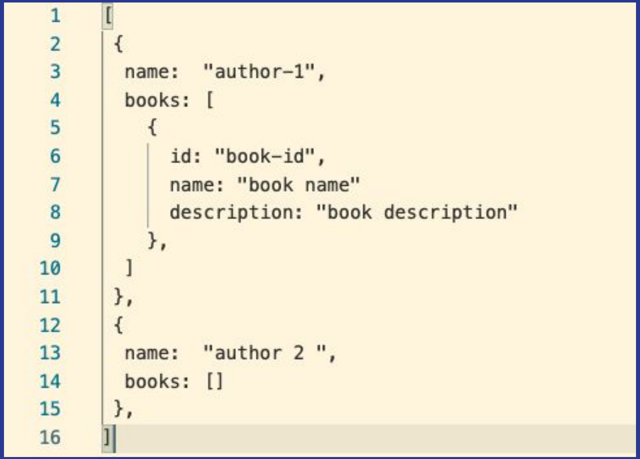


```
{
  name:  "author",
  books: [
    {
      id: "book-id",
      name: "book name"
      description: "book description"
      chapters: [
        { id: "chapter-id" }
      ]
    },
  ]
}
```

NOT ENOUGH DATA PER REQUEST
(Doesn’t fetch chapter details)

OVER FETCHING

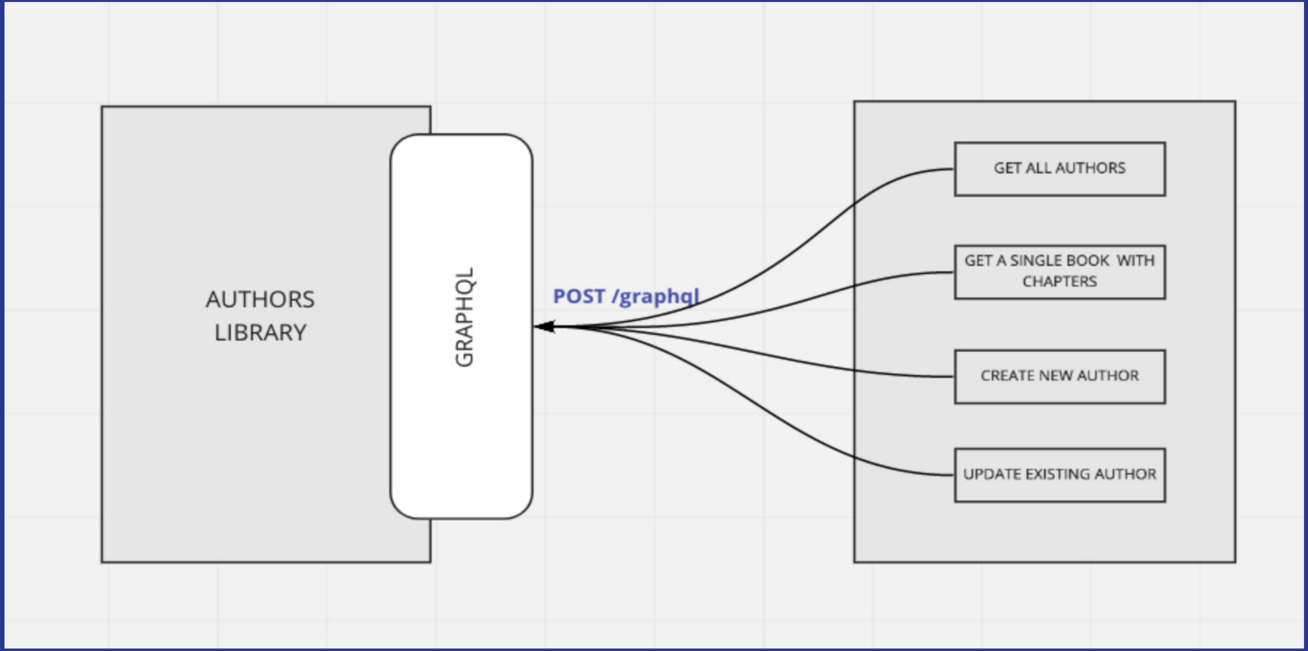
Need To List All Author Names



```
[
 {
  name:  "author-1",
  books: [
    {
      id: "book-id",
      name: "book name"
      description: "book description"
    },
  ]
 },
 {
  name:  "author 2 ",
  books: []
 },
]
```

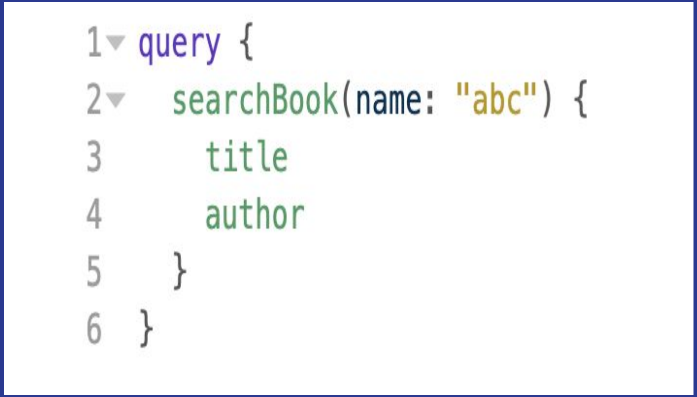
EXTRA DATA PER REQUEST
(Fetches un necessary book details)

ARCHITECTURE



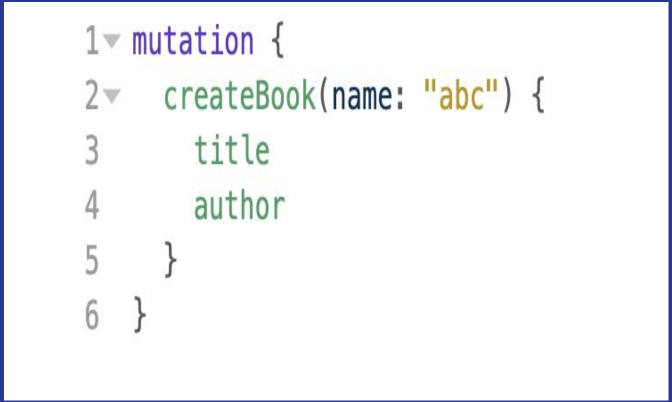
QUERIES & MUTATIONS

QUERIES
mechanism for reading data



```
query {
  searchBook(name: "abc") {
    title
    author
  }
}
```

MUTATIONS
mechanism for creating/updating data



```
mutation {
  createBook(name: "abc") {
    title
    author
  }
}
```

PRACTICAL
THEORY

Today's proof of concept will be on a basic example, in which author can have many books and each book can have multiple chapters.

CREATING PROJECT SKELETON

- ❑ Create rails project
  - ❑ rails new graphql_rubyconf --skip-test
- ❑ Generate models
  - ❑ rails g model Author email:string name:string
  - ❑ rails g model Book name:string description:text author:belongs_to
  - ❑ rails g model Chapter name:string short_description:text book:belongs_to
- ❑ Migrate DB
  - ❑ rake db:migrate
- ❑ Add missing associations in author and books models
  - ❑ author#`has_many :books`
  - ❑ book#`has_many :chapters`
- ❑ Datalayer is now all set!

SEEDING DATA FOR QUERYING

- ❑ Add faker gem
  - ❑ gem “faker” (https://github.com/faker-ruby/faker)
  - ❑ bundle install

- ❑ Populate seed file

```
2.times do
  author = Author.create(name: Faker::Name.name, email: Faker::Internet.email)

  3.times do
   book = author.books.create(name: Faker::Lorem.sentence(word_count:2), description: Faker::Lorem.paragraph(sentence_count:2))

   2.times do
     book.chapters.create(name: Faker::Lorem.sentence(word_count: 2), short_description: Faker::Lorem.paragraph(sentence_count: 2))
   end

  end
end
```

- ❑ Verify data in console

GRAPHQL INSTALLATION & OBJECT GENERATIONS

- Install graphql
  - gem ‘graphql’ (https://github.com/rmosolgo/graphql-ruby)
  - bundle install
- Run graphql install
  - rails generate graphql:install
- Check the graphql generator section added into rails generators
  - rails generate
- Schema file analysis
- Everything is ready for building some basic queries and we can boot our servers **BUT** visualization is not good yet.

ADDING BETTER VISUALIZATION

- ❏ Install graphiql-rails
  - ❏ gem "graphiql-rails" (https://github.com/rmosolgo/graphiql-rails)
  - ❏ bundle install

- ❏ Following the guide, update the route file

```
if Rails.env.development?
  mount GraphiQL::Rails::Engine, at: "/graphql", graphql_path: "graphql#execute"
end
```

- ❏ Boot the server at http://localhost:3000/graphql (this is the single point of interaction)

- ❏ Try the default query

GRAPHQL QUERIES

Queries are used to read data

Terminology:

field
object

(1) QUERY LIST OF AUTHORS WITH BASIC INFORMATION

- ❑ Query to fetch all authors, which return all authors with name,id and book_count.
- ❑ Create GraphQL objects
  - ❑ rails g graphql:object author
- ❑ Add fields into author type
  - ❑ name
  - ❑ id
  - ❑ book_count
- ❑ Include these queries into base QueryType
  - ❑ add authors field which returns array of author type
  - ❑ define the corresponding rails method
- ❑ Execute the basic query

(2) QUERY LIST OF AUTHORS WITH BOOKS

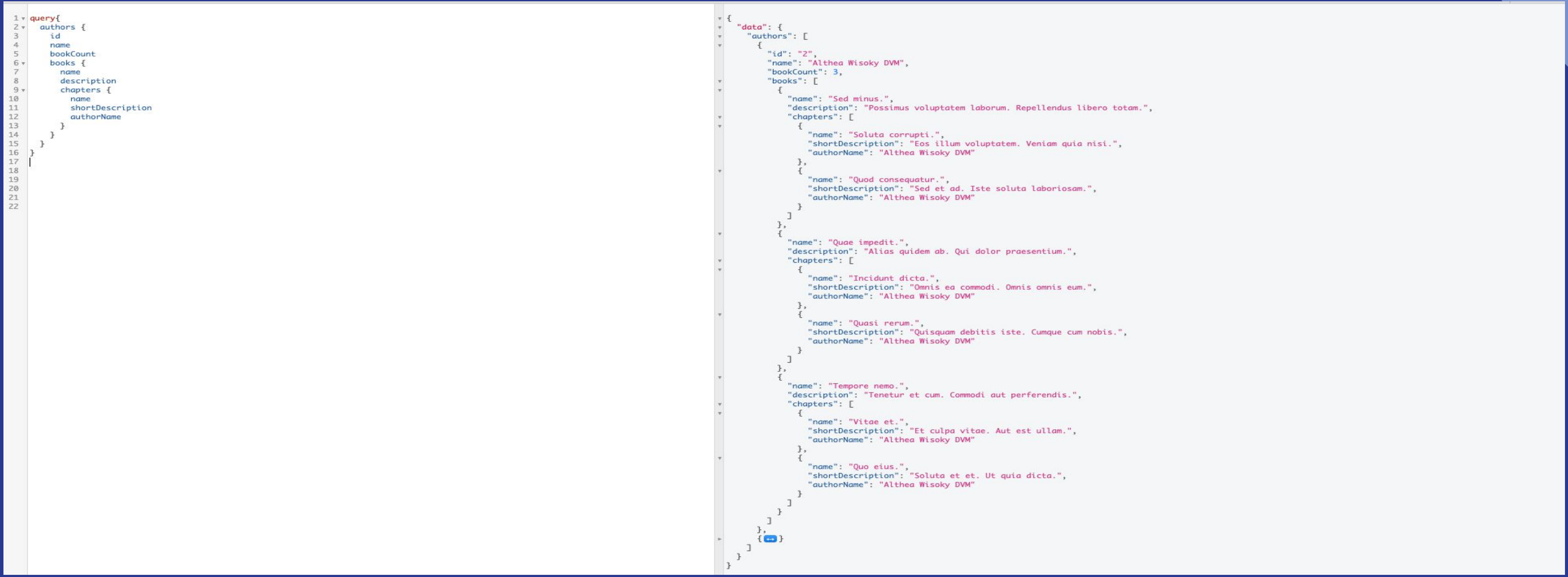
- ❏ Extend the query to fetch relational data for instance books list for the authors.
- ❏ Create GraphQL objects
  - ❏ rails g graphql:object book
- ❏ Add fields into book type
  - ❏ name
  - ❏ id
  - ❏ description
- ❏ Add the association field for books into author type.
- ❏ Execute the basic relational query

(3) QUERY LIST OF AUTHORS WITH BOOKS AND ITS CHAPTERS

- ❑ Extend book to fetch chapters and author details
- ❑ Create GraphQL objects
  - ❑ rails g graphql:object chapter
- ❑ Add fields into chapter type
  - ❑ name
  - ❑ short_description
  - ❑ id
  - ❑ author_name
- ❑ Add the association of chapters into book type.
- ❑ Execute the query

QUERY
FETCH LIST OF AUTHORS

❑ Execute the query



```
query{
  authors {
    id
    name
    bookCount
    books {
      name
      description
      chapters {
        name
        shortDescription
        authorName
      }
    }
  }
}
```

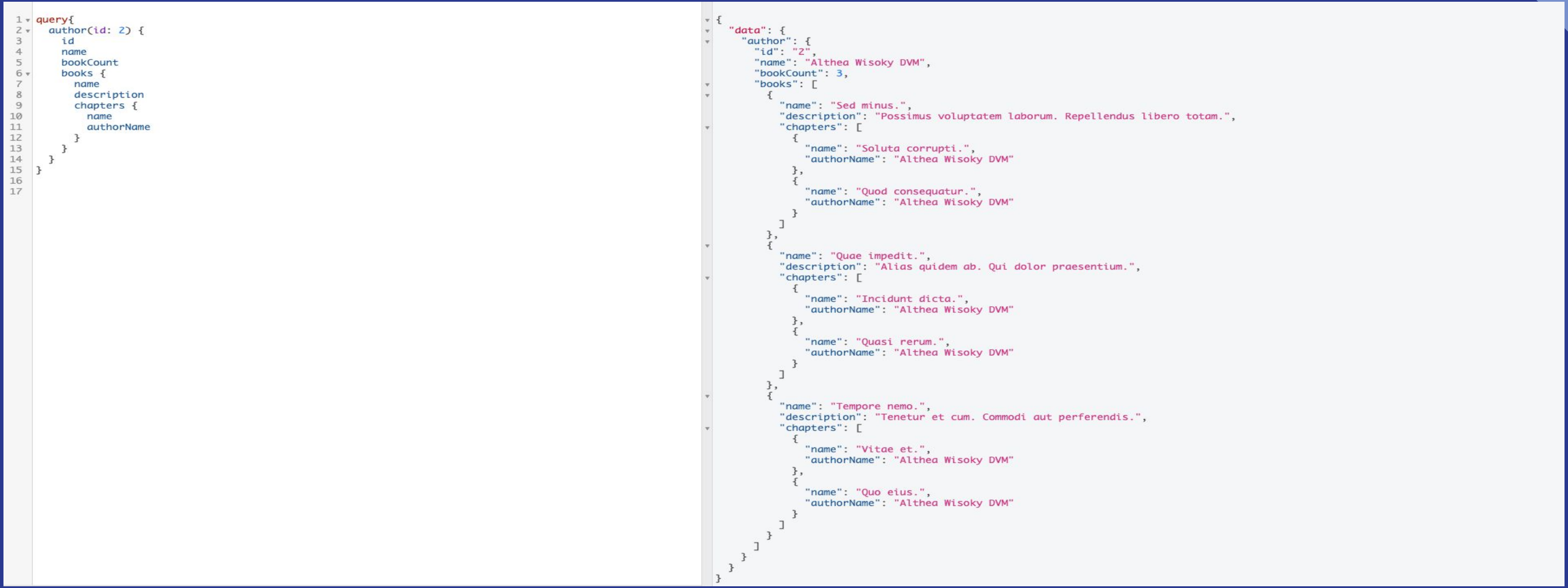
```
{
  "data": {
    "authors": [
      {
        "id": "2",
        "name": "Althea Wisoky DVM",
        "bookCount": 3,
        "books": [
          {
            "name": "Sed minus.",
            "description": "Possimus voluptatem laborum. Repellendus libero totam.",
            "chapters": [
              {
                "name": "Soluta corrupti.",
                "shortDescription": "Eos illum voluptatem. Veniam quia nisi.",
                "authorName": "Althea Wisoky DVM"
              },
              {
                "name": "Quod consequatur.",
                "shortDescription": "Sed et ad. Iste soluta laboriosam.",
                "authorName": "Althea Wisoky DVM"
              }
            ]
          },
          {
            "name": "Quae impedit.",
            "description": "Alias quidem ab. Qui dolor praesentium.",
            "chapters": [
              {
                "name": "Incidunt dicta.",
                "shortDescription": "Omnis ea commodi. Omnis omnis eum.",
                "authorName": "Althea Wisoky DVM"
              },
              {
                "name": "Quasi rerum.",
                "shortDescription": "Quisquam debitis iste. Cumque cum nobis.",
                "authorName": "Althea Wisoky DVM"
              }
            ]
          },
          {
            "name": "Tempore nemo.",
            "description": "Tenetur et cum. Commodi aut perferendis.",
            "chapters": [
              {
                "name": "Vitae et.",
                "shortDescription": "Et culpa vitae. Aut est ullam.",
                "authorName": "Althea Wisoky DVM"
              },
              {
                "name": "Quo eius.",
                "shortDescription": "Soluta et et. Ut quia dicta.",
                "authorName": "Althea Wisoky DVM"
              }
            ]
          }
        ]
      },
      {↔}
    ]
  }
}
```

QUERY FOR SINGLE AUTHOR

- ❏ Define single author query in base query for graphql
  - ❏ add author field which returns author type
  - ❏ block which takes the required ‘id’ argument
  - ❏ define the corresponding rails method

QUERY
FETCH SINGLE AUTHOR

❑ Execute the query



```
query{
  author(id: 2) {
    id
    name
    bookCount
    books {
      name
      description
      chapters {
        name
        authorName
      }
    }
  }
}
```

```
{
  "data": {
    "author": {
      "id": "2",
      "name": "Althea Wisoky DVM",
      "bookCount": 3,
      "books": [
        {
          "name": "Sed minus.",
          "description": "Possimus voluptatem laborum. Repellendus libero totam.",
          "chapters": [
            {
              "name": "Soluta corrupti.",
              "authorName": "Althea Wisoky DVM"
            },
            {
              "name": "Quod consequatur.",
              "authorName": "Althea Wisoky DVM"
            }
          ]
        },
        {
          "name": "Quae impedit.",
          "description": "Alias quidem ab. Qui dolor praesentium.",
          "chapters": [
            {
              "name": "Incidunt dicta.",
              "authorName": "Althea Wisoky DVM"
            },
            {
              "name": "Quasi rerum.",
              "authorName": "Althea Wisoky DVM"
            }
          ]
        },
        {
          "name": "Tempore nemo.",
          "description": "Tenetur et cum. Commodi aut perferendis.",
          "chapters": [
            {
              "name": "Vitae et.",
              "authorName": "Althea Wisoky DVM"
            },
            {
              "name": "Quo eius.",
              "authorName": "Althea Wisoky DVM"
            }
          ]
        }
      ]
    }
  }
}
```

GRAPHQL MUTATIONS

Mutations are used for Creating and updating data

Terminology:

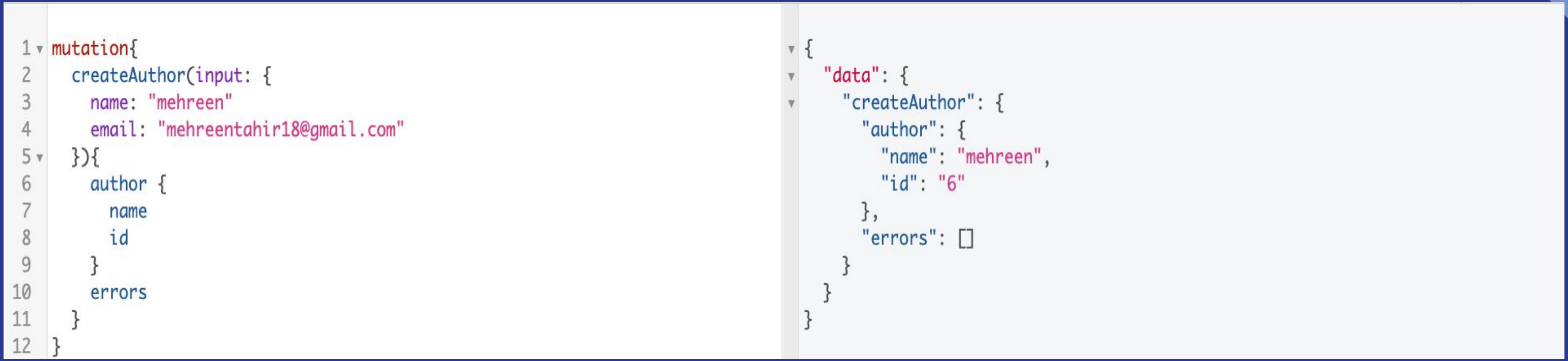
arguments
fields
resolve

CREATE AUTHOR MUTATION

- ❑ Create new mutation
  - ❑ Inside mutations folder, create a new file named as create_author.rb
  - ❑ Create a class and inherit from base mutation
    Mutations::CreateAuthor < Mutations::BaseMutation
  - ❑ Add description to this mutation
  - ❑ Add arguments which mutation needs for its creation
    - ❑ name
    - ❑ email
  - ❑ Add fields which mutation will return upon creation, in our case it will be
    - ❑ author
    - ❑ errors
  - ❑ Add the resolve method
    - ❑ Perform the active record part in it and create it
    - ❑ Return hash with author and error keys
- ❑ Add the mutation type as field in base mutation type, with referring the mutation we created
  - ❑ field :create_author, mutation: Mutations::CreateAuthor

CREATE AUTHOR MUTATION

- Execute the mutation



```
mutation{
  createAuthor(input: {
    name: "mehreen"
    email: "mehreentahir18@gmail.com"
  }){
    author {
      name
      id
    }
    errors
  }
}
```

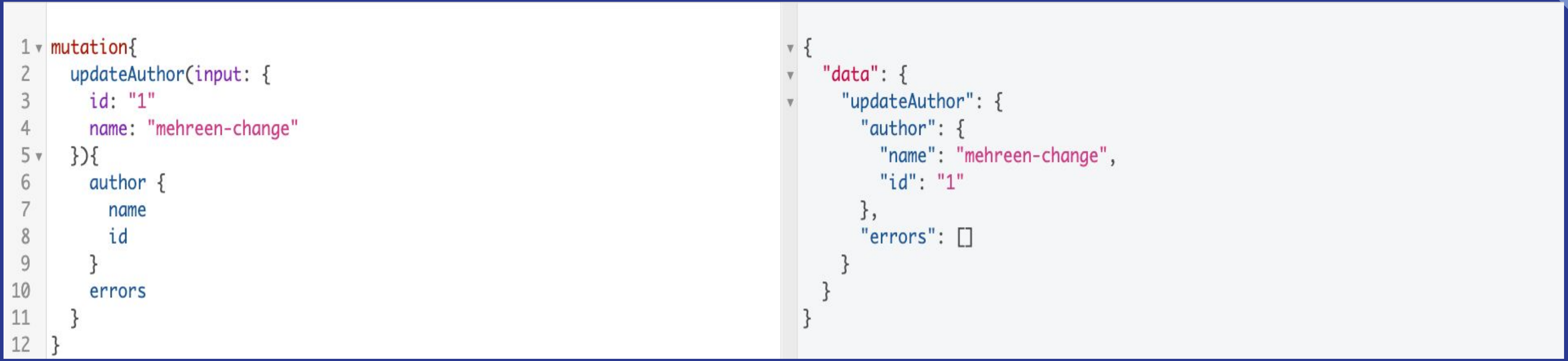
```
{
  "data": {
    "createAuthor": {
      "author": {
        "name": "mehreen",
        "id": "6"
      },
      "errors": []
    }
  }
}
```

UPDATE AUTHOR MUTATION

- ❏ Create new mutation
  - ❏ Inside mutations folder, create a new file named as update_author.rb
  - ❏ Create a class and inherit from base mutation
    Mutations::UpdateAuthor < Mutations::BaseMutation
  - ❏ Add description to this mutation
  - ❏ Add arguments which mutation needs for its creation
    - ❏ name
    - ❏ ID
  - ❏ Add fields which mutation will return upon creation, in our case it will be
    - ❏ author
    - ❏ errors
  - ❏ Add the resolve method
    - ❏ Perform the active record part in it and update it
    - ❏ Return hash with author and error keys
- ❏ Add the mutation type as field in base mutation type, with referring the mutation we created
  - ❏ field :update_author, mutation: Mutations::UpdateAuthor

UPDATE AUTHOR MUTATION

❑ Execute the mutation



```
mutation{
  updateAuthor(input: {
    id: "1"
    name: "mehreen-change"
  }){
    author {
      name
      id
    }
    errors
  }
}
```

```
{
  "data": {
    "updateAuthor": {
      "author": {
        "name": "mehreen-change",
        "id": "1"
      },
      "errors": []
    }
  }
}
```

RESOURCES

- Hosted Demo Application

  https://github.com/mehreen-tahir/graphql-with-rails
- GRAPHQL
  - https://graphql.org/
  - https://www.youtube.com/watch?v=eIQh02xuVw4
  - https://github.com/rmosolgo/graphql-ruby
  - https://github.com/rmosolgo/graphiql-rails
  - https://github.com/faker-ruby/faker

THANK YOU