Mutations

In GraphQL, in order to modify the server-side data, we use mutations (read more **here**). The official **Apollo** documentation shares an `upvotePost()` mutation example. This mutation allows to increase a post `votes` property value. In order to create an equivalent mutation in Nest, we'll make use of the `@Mutation()` decorator.

Schema first

Let's extend our `AuthorResolver` used in the previous section (see **resolvers**).

```
@Resolver('Author')
export class AuthorResolver {
  constructor(
    private readonly authorsService: AuthorsService,
    private readonly postsService: PostsService,
  ) {}

  @Query('author')
  async getAuthor(@Args('id') id: number) {
    return await this.authorsService.findOneById(id);
  }

  @Mutation()
  async upvotePost(@Args('postId') postId: number) {
    return await this.postsService.upvoteById({ id: postId });
  }

  @ResolveProperty('posts')
  async getPosts(@Parent() { id }) {
    return await this.postsService.findAll({ authorId: id });
  }
}
```

Notice that we assumed that the business logic has been moved to the `PostsService` (respectively querying post and incrementing `votes` property).

Type definitions

The last step is to add our mutation to the existing types definition.

```
type Author {
  id: Int!
  firstName: String
  lastName: String
  posts: [Post]
}

type Post {
  id: Int!
  title: String
  votes: Int
}

type Query {
  author(id: Int!): Author
}

type Mutation {
  upvotePost(postId: Int!): Post
}
```

The `upvotePost(postId: Int!): Post` mutation should be available now.

Code first

Let's add another method to the `AuthorResolver` used in the previous section (see **resolvers**).

```
@Resolver(of => Author)
export class AuthorResolver {
  constructor(
    private readonly authorsService: AuthorsService,
    private readonly postsService: PostsService,
  ) {}

  @Query(returns => Author, { name: 'author' })
  async getAuthor(@Args({ name: 'id', type: () => Int }) id: number) {
    return await this.authorsService.findOneById(id);
  }

  @Mutation(returns => Post)
  async upvotePost(@Args({ name: 'postId', type: () => Int }) postId: number) {
    return await this.postsService.upvoteById({ id: postId });
  }

  @ResolveProperty('posts')
  async getPosts(@Parent() author) {
```

```
    const { id } = author;
    return await this.postsService.findAll({ authorId: id });
  }
}
```

The `upvotePost()` takes the `postId` (`Int`) as an input parameter and returns an updated `Post` entity. For the same reasons as in the **resolvers** section, we have to explicitly set the expected type.

If the mutation has to take an object as a parameter, we can create an input type.

```
@InputType()
export class UpvotePostInput {
  @Field() postId: number;
}
```

> **HINT**
> Both `@InputType()` and `@Field()` are imported from the `type-graphql` package.

And then use it in the resolver class:

```
@Mutation(returns => Post)
async upvotePost(
  @Args('upvotePostData') upvotePostData: UpvotePostInput,
) {}
```

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more **here**.

Principal Sponsor

Sponsors / Partners

blueanchor.io

NOVOLOGIC

Designed by Jakub Staroń, hosted by Netlify