09

Refatorando User

Acabamos de ver como implementar no resolver métodos para resolvermos cada um dos campos de um tipo; vamos voltar ao que fizemos no curso anterior, quando definimos o tipo `User`:

```
type User {
  nome: String!
  ativo: Boolean!
  email: String
  role: Role!
  createdAt: DateTime
  matriculas: [Matricula]
}

type Role {
  id: ID!
  type: RolesType!
}
```

COPIAR CÓDIGO

Assim como o campo `matriculas`, o campo `role` também tem como valor um tipo objeto. Mas, vendo o código no projeto, o campo `role` não foi resolvido individualmente em `./api/user/userResolvers.js`. Para que o GraphQL recebesse os dados no formato esperado, desenvolvemos o método `getUsers()` da seguinte forma:

```
async getUsers() {
  const users = await this.get('/users')
```

```
    return users.map(async user => ({
        id: user.id,
        nome: user.nome,
        email: user.email,
        ativo: user.ativo,
        role: await this.get(`/roles/${user.role}`)
    }))
}
```

COPIAR CÓDIGO

Agora que já praticamos mais com o GraphQL, o que podemos modificar neste código?

Selecione uma alternativa

A Não há necessidade de refatoração nesse caso, pois já está sendo utilizado o `.get()` para retornar os dados como valor da propriedade `role`.

B Uma vez que o tipo-objeto `Role` utiliza os dados do endpoint `/roles`, seria possível criar mais uma entidade, `./api/role` que tenha seu próprio schema, resolver e dataSource.

C Ao invés de consultar o endpoint `/roles` para cada user na array, e montar o objeto final "na mão", é possível criar uma função no resolver de User para resolver o campo `role`.