Introduction

EDIT

We all love GraphQL! It's really great and solves many problems that we have with REST APIs, such as overfetching and underfetching. But developing a GraphQL API in Node.js with TypeScript is sometimes a bit of a pain.

What?

TypeGraphQL is a library that makes this process enjoyable by defining the schema using only classes and a bit of decorator magic. Example object type:

```
@ObjectType()
class Recipe {
  @Field()
  title: string;

  @Field(type => [Rate])
  ratings: Rate[];

  @Field({ nullable: true })
  averageRating?: number;
}
```

It also has a set of useful features, like validation, authorization and dependency injection, which helps develop GraphQL APIs quickly & easily!

Why?

As mentioned, developing a GraphQL API in Node.js with TypeScript is sometimes a bit of a pain. Why? Let's take a look at the steps we usually have to take.

First, we create all the schema types in SDL. We also create our data models using ORM classes, which represent our database entities. Then we start to write resolvers for our queries, mutations and fields. This forces us, however, to begin with creating TypeScript interfaces for all arguments and inputs and/or object types. After that, we can actually implement the resolvers, using weird generic signatures, e.g.:

```
) => {
  // common tasks repeatable for almost every resolver
  const auth = Container.get(AuthService);
  if (!auth.check(ctx.user)) {
    throw new NotAuthorizedError();
  }
  await joi.validate(getRecipesSchema, args);
  const repository = TypeORM.getRepository(Recipe);

  // our business logic, e.g.:
  return repository.find({ skip: args.offset, take: args.limit });
};
```

The biggest problem is code redundancy which makes it difficult to keep things in sync. To add a new field to our entity, we have to jump through all the files: modify the entity class, then modify the schema, and finally update the interface. The same goes with inputs or arguments: it's easy to forget to update one of them or make a mistake with a type. Also, what if we've made a typo in a field name? The rename feature (F2) won't work correctly.

TypeGraphQL comes to address these issues, based on experience from a few years of developing GraphQL APIs in TypeScript. The main idea is to have only one source of truth by defining the schema using classes and a bit of decorator help. Additional features like dependency injection, validation and auth guards help with common tasks that would normally have to be handled by ourselves.

INSTALLATION →