8. Resolver - user.ts / Mutation - register()

#graphql #resolver #authentication #mutation #mikroorm #backend

```
yarn add argon2
```

- argon2 will be used for hashing passwords in the resolver

- We create a **UsernamePasswordInput** class to simplify our code - this will have the InputType() attribute so it can be used in Arg()

/resolvers/UsernamePasswordInput.ts

```ts
import { Field, InputType } from "type-graphql";

@InputType() // InputType are used for arguments
export class UsernamePasswordInput {
  @Field()
  username: string;
  @Field()
  email: string;
  @Field(() => String) // can set type explicitly, or let typescript infer it
  password: string;
}
```

/resolvers/user.ts

```ts
import { User } from "../entities/User";
import { MyContext } from "src/types";
import {Arg, Ctx, Field, Mutation, Query, Resolver} from "type-graphql";
import argon2 from "argon2";
import { UsernamePasswordInput } from "./UsernamePasswordInput";

@Resolver()
export class UserResolver {
  @Mutation(() => UserResponse)
  async register(
    @Arg("options") options: UsernamePasswordInput, //  let typescript infer type
```

```
UsernamePasswordInput
    @Ctx() { em }: MyContext
  ): Promise<User> {
    const hashedPassword = await argon2.hash(options.password);
    const user = em.create(User, {
     username: options.username,
     password: hashedPassword,
  })
  await em.persistAndFlush(user);
  return user;
  }
}
```

This is the simplest version of this resolver with only register functionality and no error checking etc.