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

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

---

Validating the data

- First we have a utility function for validating the registration username and password :

utils/validateRegister.ts

```
import { UsernamePasswordInput } from "src/resolvers/UsernamePasswordInput";

export const validateRegister = (options: UsernamePasswordInput) => {
  if (!options.email.includes("@")) {
    return [
      {
        field: "email",
        message: "invalid email",
      },
    ];
  }

  if (options.username.length <= 2) {
    return [
      {
        field: "username",
        message: "Length must be greater than 2",
      },
    ];
  }

  if (options.username.includes("@")) {
    return [
      {
        field: "username",
        message: "Username cannot include an '@' symbol",
      },
    ];
  }
```

```
  if (options.password.length <= 3) {
    return [
      {
        field: "password",
        message: "Length must be greater than 3",
      },
    ];
  }

  return null;
};
```

A better register resolver

- Here we first validate the data entered
- And then use createQueryBuilder() to insert the data into DB because persistAndFlush() causes an error (haha)
- We also catch the err.code === "23505" error (duplicate username)

resolvers/user.ts

```
import { validateRegister } from "../utils/validateRegister";
import { EntityManager } from "@mikro-orm/postgresql";

@Mutation(() => UserResponse)
  async register(
    @Arg("options") options: UsernamePasswordInput, //  let typescript infer type
UsernamePasswordInput
    @Ctx() { em, req }: MyContext
  ): Promise<UserResponse> {
    const errors = validateRegister(options);
    if (errors) {
      return { errors };
    }

    const hashedPassword = await argon2.hash(options.password);

    let user;
    try {
```

```
      const result = await (em as EntityManager)
        .createQueryBuilder(User)
        .getKnexQuery()
        .insert({
          username: options.username,
          password: hashedPassword,
          email: options.email,
          created_at: new Date(), // mikroORM adds the underscores in DB so we
must write it like this with Knex
          updated_at: new Date(),
        })
        .returning(["*", "created_at as createdAt", "updated_at as updatedAt"]);
      user = result[0];
    } catch (err) {
      // duplicate username error
      if (err.code === "23505") {
        return {
          errors: [
            {
              field: "username",
              message: "That username is already taken",
            },
          ],
        };
      }
    }

    req.session.userId = user.id; // logs in the user (by sending cookie to
browser)
    return { user };
  }
```