3. MikroORM

#mikroorm #backend

```
yarn add @mikro-orm/cli @mikro-orm/core @mikro-orm/migration @mikro-orm/posgresql
pq
```

- @mikro-orm/posgresql and pq → these are for postgresql, but packages for other DBs can also be installed

---

Initial setup and config

package,json

```
"mikro-orm": {
    "useTsNode": true,
    "configPaths": [
      "./src/mikro-orm.config.ts",
      "./dist/mikro-orm.config.js"
    ]
  }
```

constants.ts

```
export const __prod__ = process.env.NODE_ENV === "production"; // is the env
variable set as "production" ?
export const COOKIE_NAME = "qid";
```

mikro-orm.config.ts

```
import { MikroORM } from "@mikro-orm/core";
import path from "path";
import { __prod__ } from "./constants";
import { Post } from "./entities/Post";
import { User } from "./entities/User";

export default {
        migrations: {
                path: path.join(__dirname, "./migrations"),
                pattern: /^[\w-]+\d+\.[tj]s$/,
        },
        entities: [Post, User],
        dbName: "lireddit",
        type: "postgresql",
        user: "postgres",
        password: "postgres",
        debug: !__prod__,
} as Parameters<typeof MikroORM.init>[0];
```

- The **entities** are the names of the database tables that mikroorm will interact with (see below)
- as Parameters<typeof MikroORM.init>[0] allows this export to be passed into MikroORM..init() in index.ts
- `entities: [Post, User],` this should be updated everytime we add a new entity
- after creating the entity and updating the above parameter we run `mikro-orm migration:create` to create a new migration that will update the database, add columns and create the new tables if necessary
- the new migrations are automaticlly run with the `await orm.getMigrator().up();` line in index.ts, below.

---

index,ts

```
import { MikroORM } from "@mikro-orm/core";
import microConfig from "./mikro-orm.config";

const main = async () => {
  const orm = await MikroORM.init(microConfig);
  await orm.getMigrator().up(); // run migration
```

- initialize MikroORM using the config and set up migrator to run at startup

---

How to interact with the DB

We will be using these in the Resolvers to interact with the DB

```
const post = orm.em.create(Post, { title: "my first post" }) // the Post class is
used to create the missing fields
await orm.em.persistAndFlush(post)
```

```
// or
await orm.em.nativeInsert(Post, { title: 'my first post 2', createdAt: new
Date(), updatedAt: new Date()}) // we must provide all fields

const posts = await orm.em.find(Post, {})
console.log(posts)
```

- **inline** method of creating posts and pushing them into db, or searching for (all) posts with mikroORM
 we do not use this much but instead use **resolvers**

How to create the migrations

```
npx mikro-orm migration:create
```

- create migrations, i.e. create the DB table according to the entity schemas that are defined
- and then when we start the server the migration is automatically run since we set it up that way in index.ts