BACKEND

Description

Nest framework TypeScript starter repository.

Folder structure

```
.
├── src
│   ├── config
│   ├── core
│   │   ├── entities
│   │   ├── exceptions
│   │   ├── http
│   │   │   ├── controllers
│   │   │   └── guards
│   │   ├── repositories
│   │   │   ├── criterias
│   │   │   └── eloquents
│   │   ├── requests
│   │   ├── services
│   │   ├── tests
│   │   └── transformers
│   ├── mails
│   │   ├── adapters
│   │   ├── constants
│   │   ├── interfaces
│   │   ├── mails
│   │   └── services
│   └── users                    -> Module user
│       ├── entities             -> Contains entities
│       ├── enums                -> Contains enums
│       ├── http
│       │   ├── controllers      -> Contains controllers
│       │   ├── guards           -> Contains guards
│       │   ├── midlewares       -> Contains midlewares
│       │   └── requests         -> Contains request
│       ├── mails                -> Contains emails
│       ├── repositories         -> Contains repositories
│       ├── resources            -> Contains information such as views,
fonts, css...
│       ├── services             -> Contains services
```

```
│       └── transformers        -> Contains transformers
└── test
```

File structure conventions

Some code examples display a file that has one or more similarly named companion files. For example, `hero.controller.ts` and `hero.service.ts`

Single responsibility

Apply the single responsibility principle (SRP) to all components, services, and other symbols. This helps make the app cleaner, easier to read and maintain, and more testable.

Code Rule

Nestjs is inspired by Angular, so you can use some rules from Angular.

Small functions

Do define small functions

Consider limiting to no more than 75 lines.

Consider limiting files to 400 lines of code.

Naming

General Naming Guidelines

Do use consistent names for all symbols.

Do follow a pattern that describes the symbol's feature then its type. The recommended pattern is `feature.type.ts`.

Separate file names with dots and dashes

Do use dashes to separate words in the descriptive name.

Do use dots to separate the descriptive name from the type.

Do use consistent type names for all components following a pattern that describes the component's feature then its type. A recommended pattern is `feature.type.ts`.

Do use conventional type names including `.service, .component, .pipe, .module`, and `.directive`. Invent additional type names if you must but take care not to create too many.

Symbols and file names

Do use consistent names for all assets named after what they represent.

Do use upper camel case for class names.

Do match the name of the symbol to the name of the file.

Do append the symbol name with the conventional suffix (such as Component, Directive, Module, Pipe, or Service) for a thing of that type.

Do give the filename the conventional suffix (such as `.component.ts`, `.directive.ts`, `.module.ts`, `.pipe.ts`, or `.service.ts`) for a file of that type.

E2E test

Do name end-to-end test specification files after the feature they test with a suffix of `.e2e-spec.ts`

Database rule

Normally, naming the database will be an underscore (like `user_plan`), but for full standardization on the mongoose system with NestJs, we will use lowerCase as a plural for the collection. Example: `user_plan` -> `userplans`

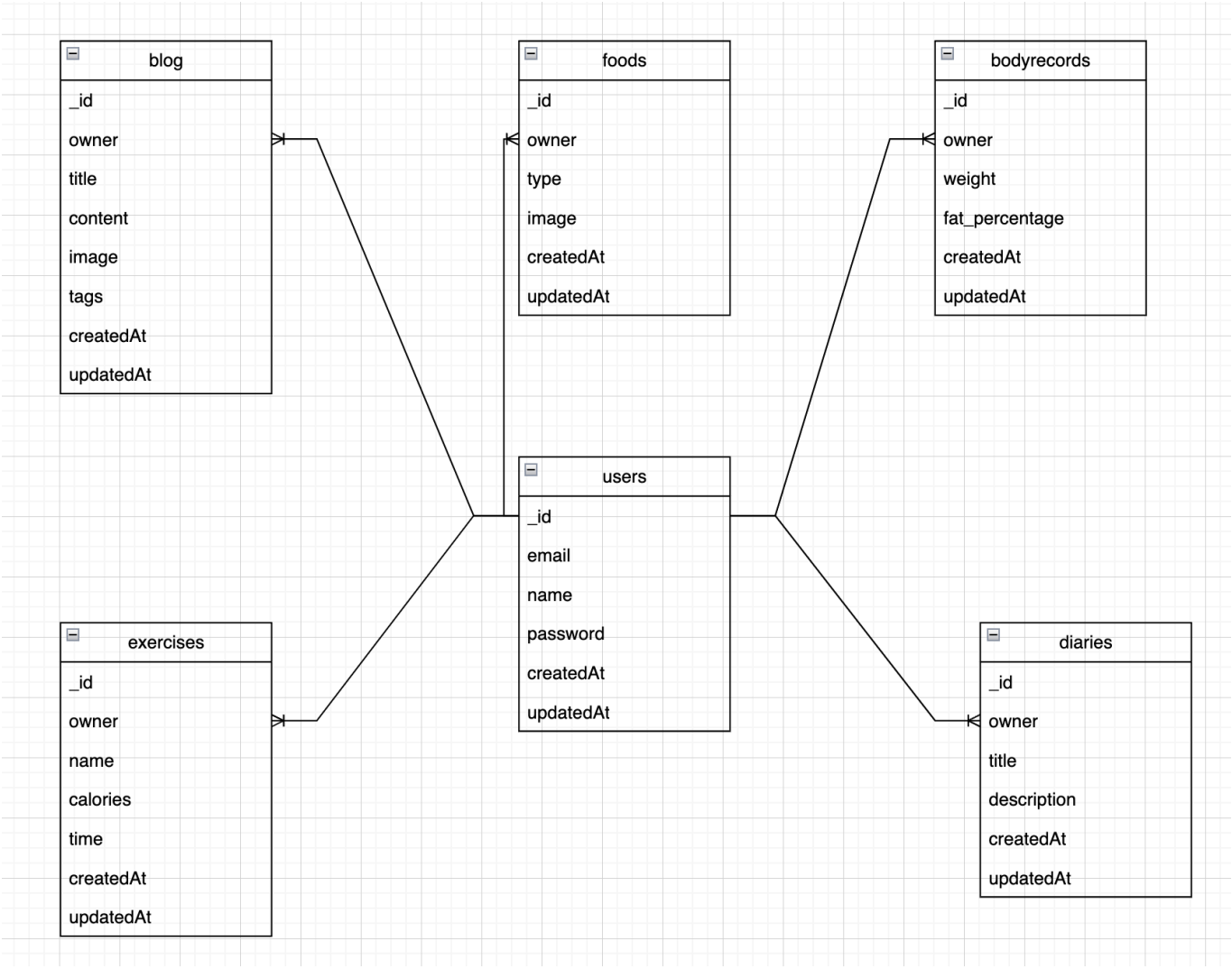
Database

> MongoDb

Version

> MongoDb: 6.0.5

Database basic diagram



Step by step to running api application

1.Go to /health_app_api directory

```
$ cd /health_app_api
```

2.Create .env file similar to file .env.example

3.Run docker

```
docker-compose up -d
```

4.Installation dependencies

```
#using npm
$ npm install
# or using yarn
$ yarn install
```

5.Running the app

```
# development
$ npm run start

# watch mode
$ npm run start:dev

# production mode
$ npm run start:prod
```

Response

Success

Return a data object

```

```

Validate error - 422

```
{
  "httpStatus": 422,
  "message": null,
  "messages": [
    "password must be longer than or equal to 15 characters",
    "password must be a string",
    "email should not be empty"
  ]
}
```

Entities not found - 404

```
{
  "httpStatus": 404,
  "message": "Enterprise not found",
  "errors": null
}
```

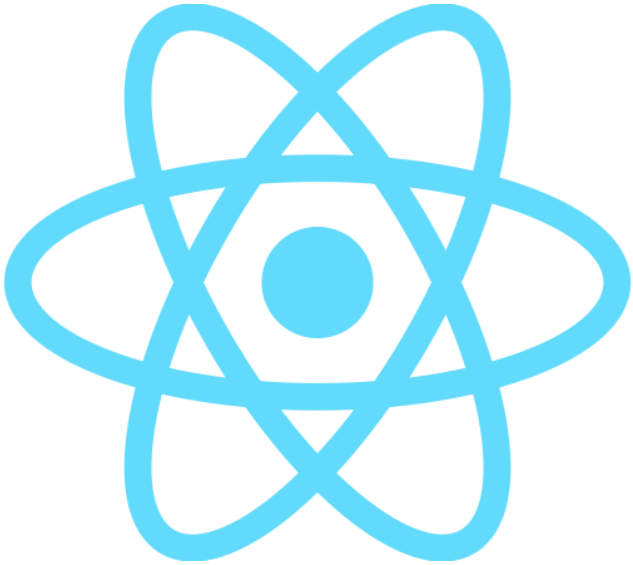
Unauthorized - 401

```

```

```
{
  "httpStatus": 401,
  "message": "Unauthorized",
  "errors": null
}
```

FRONTEND



Description

React The library for web and native user interfaces

Technical

Tailwind CssA utility-first CSS framework packed with classes Redux toolkit Batteries-included toolset for efficient Redux development

Folder structure

```
.
├── src
│   ├── assets                  -> store images and icons
│   ├── api                     -> Call api
│   ├── pages
│   │   ├── HomePage
│   │   │   ├── index
│   │   │   └── components      -> components only appears in this page
│   │   └── ...                 -> other pages
│   ├── layouts
│   │   ├── MainLayout
│   │   │   ├── index
│   │   │   └── components      -> components only appears in this layout
│   │   └── ...                 -> other layouts
```

```
│   ├── routes
│   │   ├── PrivateRoute
│   │   │   └── index           -> components only appears in this layout
│   │   ├── ...                 -> components only appears in
│   │   └── index
│   ├── redux
│   │   ├── slice               -> store all redux toolkit slice
│   │   │   └── ...
│   │   │   └── index           -> sum up the slices
│   │   └── store
│   ├── components              -> components shared for the whole app
│   │   ├── forms
│   │   │   └── LoginForm
│   │   │   └── ...
│   │   ├── buttons
│   │   └── ...
│   └──  hooks
└── App
└── index
```

Run application

Step 1: Make sure the api application running and go to /health_app_api directory

```
$ cd /health_app_frontend
```

Step 2: Copy create .env files similar to file .env.example

Default application will run on port 3000. if you want to change port , you want to change the PORT environment variable in .env file. Make sure this url is WHITELIST_DOMAINS environment variable in .env file for api app

Step 3: Install dependencies

```
$ npm install
```

Or using yarn

```
$ yarn install
```

Step 3: Run app

```
$ npm start
```

Or using yarn

```
$ yarn start
```