

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Лабораторная работа 3

Выполнил:

Рыбалко Олег

Группа К33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

## **Задача**

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения.

## Ход работы

Отдельный микросервис будет работать с пользователями, а именно проводить их авторизацию и выполнять CRUD операции над ними. При помощи такого подхода мы сможем перенести хранение пользователей в отдельную базу данных, что облегчит дальнейшую работу с пользователями, так как при изменении другой логики мы точно не затронем пользователей.

1. Создадим директорию для нашего сервиса и перенесем файлы конфигурации из предыдущей лабораторной работы

```
> tree -L 1
.
├── db.sqlite
├── dist
├── node_modules
├── package-lock.json
├── package.json
├── src
└── tsconfig.json

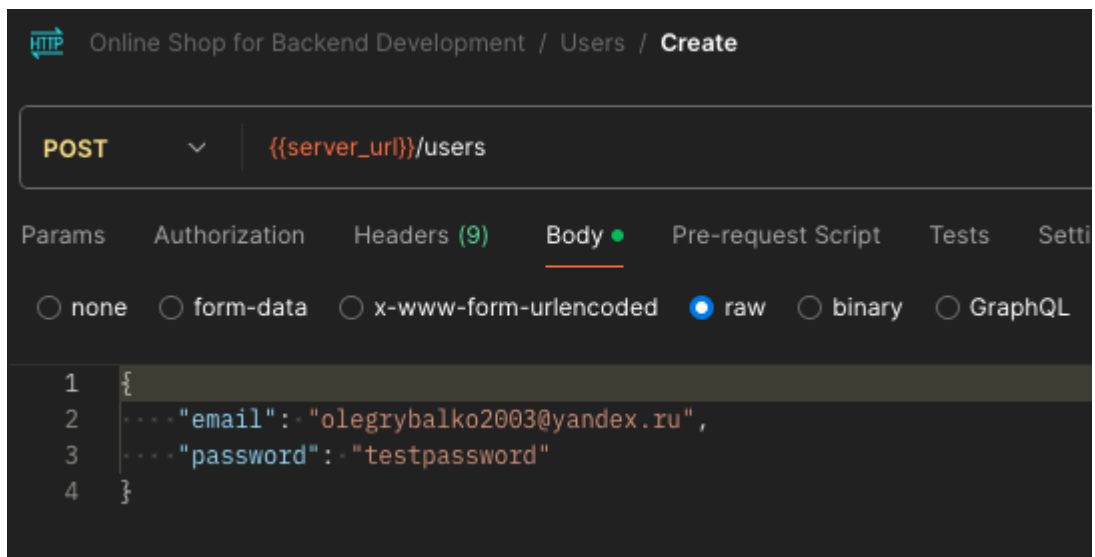
4 directories, 4 files
```

2. Перенесем все компоненты, необходимые для работы с пользователем в директорию src

```
> tree src
src
├── controllers
│   └── users
│       ├── index.ts
│       └── models.ts
├── index.ts
├── models
│   └── user.ts
├── providers
│   └── db.ts
├── routes
│   └── users
│       └── index.ts
└── services
    ├── base
    │   └── index.ts
    └── users
        └── index.ts

10 directories, 8 files
```

3. Запустим и проверим работу сервиса при помощи Postman
  - a. Создадим пользователя



```
Pretty Raw Preview Visualize JSON ↕
1 {
2   "id": 2
3 }
```

## б. Авторизуемся

```
POST {{server_url}}/users/auth
Params Authorization Headers (9) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON
1 {
2   "email": "olegrybalko2003@yandex.ru",
3   "password": "testpassword"
4 }
```

```
Body Cookies Headers (7) Test Results (1/3)
Pretty Raw Preview Visualize JSON ↕
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
3 }
```

4. Наш по работе с пользователями работает, но теперь необходимо указывать два разных порта для отправки запросов, что не совсем удобно. Для решения этой проблемы на данном этапе воспользуемся caddy <https://caddyserver.com/>. Создадим Caddyfile, в котором укажем, что запросы с путем /users должны отправляться на порт 9091, а все остальные на порт 9090.

```
> cat Caddyfile
localhost {
    reverse_proxy /users localhost:9091
    reverse_proxy localhost:9090
}
```

Запустим прокси сервер с нашей конфигурацией и убедимся, что все работает. Также caddy автоматически настраивает SSL сертификат, поэтому мы можем обращаться к серверу при помощи https протокола.

```
> caddy run --config Caddyfile
2024/05/01 11:38:04.415 INFO using provided configuration {"config_file": "Caddyfile", "config_adapter": ""}
2024/05/01 11:38:04.417 INFO admin admin endpoint started {"address": "localhost:2019", "enforce_origin": false, "origins": ["/"]}
2024/05/01 11:38:04.417 INFO http.auto_https server is listening only on the HTTPS port but has no TLS connection policies; adding one
2024/05/01 11:38:04.417 INFO http.auto_https enabling automatic HTTP->HTTPS redirects {"server_name": "srv0"}
2024/05/01 11:38:04.417 INFO tls.cache.maintenance started background certificate maintenance {"cache": "0x1400059f780"}
2024/05/01 11:38:04.423 INFO pki.ca.local root certificate is already trusted by system {"path": "storage:pki/authorities/local"}
2024/05/01 11:38:04.423 INFO http enabling HTTP/3 listener {"addr": ":443"}
2024/05/01 11:38:04.423 INFO http.log server running {"name": "srv0", "protocols": ["h1", "h2", "h3"]}
2024/05/01 11:38:04.424 INFO http.log server running {"name": "remaining_auto_https_redirects", "protocols": ["h1", "h2", "h3"]}
2024/05/01 11:38:04.424 INFO http enabling automatic TLS certificate management {"domains": ["localhost"]}
2024/05/01 11:38:04.424 WARN tls stapling OCSP {"error": "no OCSP stapling for [localhost]: no OCSP server specified in certificate"}
2024/05/01 11:38:04.425 INFO autosaved config (load with --resume flag) {"file": "/Users/rybalkooleg/Library/Application Support/Caddy/config.json"}
2024/05/01 11:38:04.425 INFO serving initial configuration
2024/05/01 11:38:04.433 WARN tls storage cleaning happened too recently; skipping for now {"storage": "FileStorage:/Users/rybalkooleg/Library/Application Support/Caddy/storage"}
2024/05/01 11:38:04.433 INFO ry_again": "2024/05/02 11:38:04.433", "try_again_in": 86399.99999975}
2024/05/01 11:38:04.433 INFO tls finished cleaning storage units
```

The screenshot shows a web browser interface with a GET request to `{{server_url}}/products`. The response body is displayed in JSON format, showing a single product object with the following details:

- `id`: 4,
- `name`: "test product",
- `quantity`: 100,
- `price`: 100,
- `imageUrl`: "https://en.wikipedia.org/wiki/File:Doom\_cover\_art.jpg",
- `createdAt`: "2024-04-25T05:46:49.255Z",
- `updatedAt`: "2024-04-25T05:46:49.255Z"

POST

▼

{{server\_url}}/users/auth

ParamsAuthorizationHeaders (9)Body ●Pre-request ScriptTests ●Settings

BodyCookiesHeaders (7)Test Results (1/3)

PrettyRawPreviewVisualizeJSON ▼

↺

1 {

2 |   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJvbGVncnliYW

3 }

## **Вывод**

В данной лабораторной работе удалось создать микросервис для работы с пользователями. Более того получилось настроить reverse проху, для удобства обращения к серверу. В итоге клиенту необходимо обращаться к одному адресу сервера, а затем caddy распределяет запросы по необходимым сервисам.