

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина: Бэк-энд разработка**

**Отчет**

**Лабораторная работа 1**

**Выполнила:**

**Космач Мария**

**Группа К3339**

**Проверил:**

**Добряков Д. И.**

**Санкт-Петербург**

**2025 г.**

## **Задача**

Нужно написать свой boilerplate на express + TypeORM + typescript.

Должно быть явное разделение на:

- модели,
- контроллеры,
- роуты.

Должна быть реализована базовая пользовательская модель, jwt-авторизация и конфигурация через переменные среды.

## **Ход работы**

На рисунке 1 продемонстрирована структура шаблона, на которой видно явное разделение на модели, контроллеры и роуты.

На рисунке 2 продемонстрирован UserController, в котором есть метод для получения авторизованного юзера. Пример выполнения запроса показан на рисунке 2 и 3

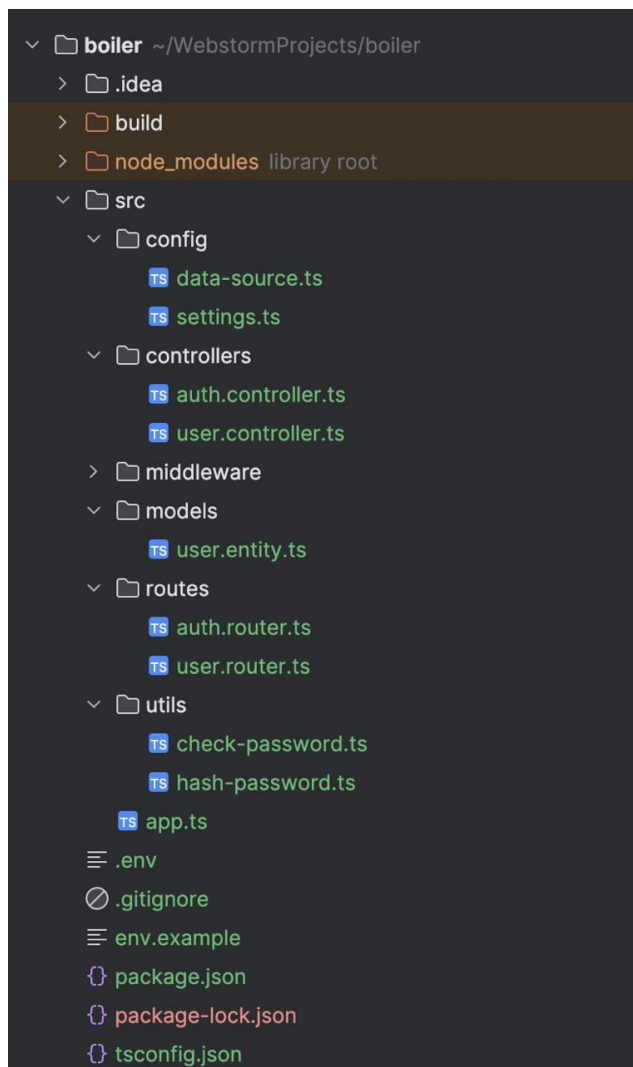


Рисунок 1 – Структура проекта

```

1  import { RequestHandler } from 'express';
2  import { UserEntity } from '../models/user.entity';
3  import dataSource from '../config/data-source';
4
5  class UserController { Show usages new *
6      private repository : Repository<UserEntity> = dataSource.getRepository(UserEntity);
7
8      me: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... ,
9          try {
10             const userId : any = (req as any).userId;
11             const user : UserEntity = await this.repository.findOneBy({ id: userId });
12             if (!user) {
13                 res.status(404).json({ message: "User not found" });
14                 return;
15             }
16             res.status(200).json(user);
17         } catch (err) {
18             next(err);
19         }
20     };
21
22     getUserById: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>,
23         try {
24             const id : number = parseInt(req.params.id);
25             const user : UserEntity = await this.repository.findOneBy({ id });
26             res.status(200).json(user);
27         } catch (err) {
28             next(err);
29         }
30     };
31
32     getUserByEmail: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>,
33         try {
34             const mail : string = req.params.mail;
35             const user : UserEntity = await this.repository.findOneBy({ mail: mail });
36             res.status(200).json(user);

```

Рисунок 2 - UserController



```

import SETTINGS from '../config/settings';

class AuthController { Show usages new *
  private repository :Repository<UserEntity> = dataSource.getRepository(UserEntity);

  register: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction )
  {
    try {
      const { mail, password, firstName, lastName } = req.body;

      const existingUser :UserEntity = await this.repository.findOne({ where: { mail } });
      if (existingUser) {
        res.status(400).json({ message: 'User already exists' });
        return;
      }

      const hashedPassword :string = hashPassword(password)
      const newUser :{mail: any; password: string; firstName:... = await this.repository.save({
        mail,
        password: hashedPassword,
        firstName,
        lastName
      });

      const token :string = jwt.sign({ id: newUser.id }, SETTINGS.JWT_SECRET_KEY, {
        expiresIn: '1h',
      });

      res.status(201).json({ user: newUser, token });
    } catch (err) {
      next(err);
    }
  }
};

```

Рисунок 5 - AuthController

The screenshot displays a REST client interface with the following details:

- URL:** `http://localhost:3000/auth/register/`
- Method:** `POST`
- Body (raw JSON):**

```

{
  "mail": "user@example.com",
  "password": "securePassword123",
  "firstName": "John",
  "lastName": "Doe"
}

```
- Response Status:** `201 Created` (65 ms, 621 B)
- Response Body (JSON):**

```

{
  "user": {
    "mail": "user@example.com",
    "password": "$2b$08$b7Bp91GR06bquHeeiqyaleLqb2LSVV0GnQA01WdU6kI/Bv6H260Ru",
    "firstName": "John",
    "lastName": "Doe",
    "id": 1,
    "createdAt": "2025-04-30T12:00:22.546Z",
    "updatedAt": "2025-04-30T12:00:22.546Z"
  },
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNzQ2MDE0NDIyLCJleHAiOjE3NDYwMTgwMjJ9.KivAolW68vN0vmUTgP00oxrnTfRAMcriw-2nQOSxgns"
}

```

Рисунок 6 – Пример запроса auth/register

На рисунке 7 продемонстрирован файл settings.ts, в котором получают переменные окружения из файла .env.

```
import dotenv from 'dotenv';

dotenv.config();

class Settings { Show usages
  // db connection settings
  DB_HOST : string = process.env.DB_HOST || 'localhost';
  DB_PORT : number = parseInt(process.env.DB_PORT || '15432');
  DB_NAME : string = process.env.DB_NAME || 'maindb';
  DB_USER : string = process.env.DB_USER || 'maindb';
  DB_PASSWORD : string = process.env.DB_PASSWORD || 'maindb';
  DB_ENTITIES : string = process.env.DB_ENTITIES || 'build/entities/*.entity.js';

  JWT_SECRET_KEY : string = process.env.JWT_SECRET_KEY || 'secret';
  JWT_TOKEN_TYPE : string = process.env.JWT_TOKEN_TYPE || 'Bearer';
  JWT_ACCESS_TOKEN_LIFETIME: number =
    parseInt(process.env.JWT_ACCESS_TOKEN_LIFETIME || '300');
}

const SETTINGS = new Settings();

export default SETTINGS;| Show usages
```

Рисунок 7 – settings.ts

## Вывод

В рамках работы был создан шаблон проекта, в основу которого закладывают express + TypeORM + typescript.