Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

О Т Ч Е Т

по лабораторной работе №1
«Реализация boilerplate»
курса «Бэкенд разработка»

Выполнили:
Бахарева М.А., K3342
Привалов К.А., K3342
Проверил:
Добряков Д.И.

Санкт-Петербург,
2025

Содержание

Ход работы

Задание

Нужно написать свой boilerplate на Express + TypeORM + Typescript. Должно быть явное разделение на:

- модели
- контроллеры
- роуты

Основная часть

Инициализация проекта

Для начала инициализируем Node.JS проект командой npm init -y. Устанавливаем TypeScript, настраиваем в проекте (npm install –save-dev typescript) и создаем файл tsconfig.json со следующим содержанием:

```
{
 "compilerOptions": {
   "target": "ES2020",
   "module": "CommonJS",
   "outDir": "dist",
   "rootDir": "src",
   "strict": true,
   "moduleResolution": "node",
   "esModuleInterop": true,
   "experimentalDecorators": true,
   "emitDecoratorMetadata": true,
   "skipLibCheck": true,
   "resolveJsonModule": true
 }
}
```

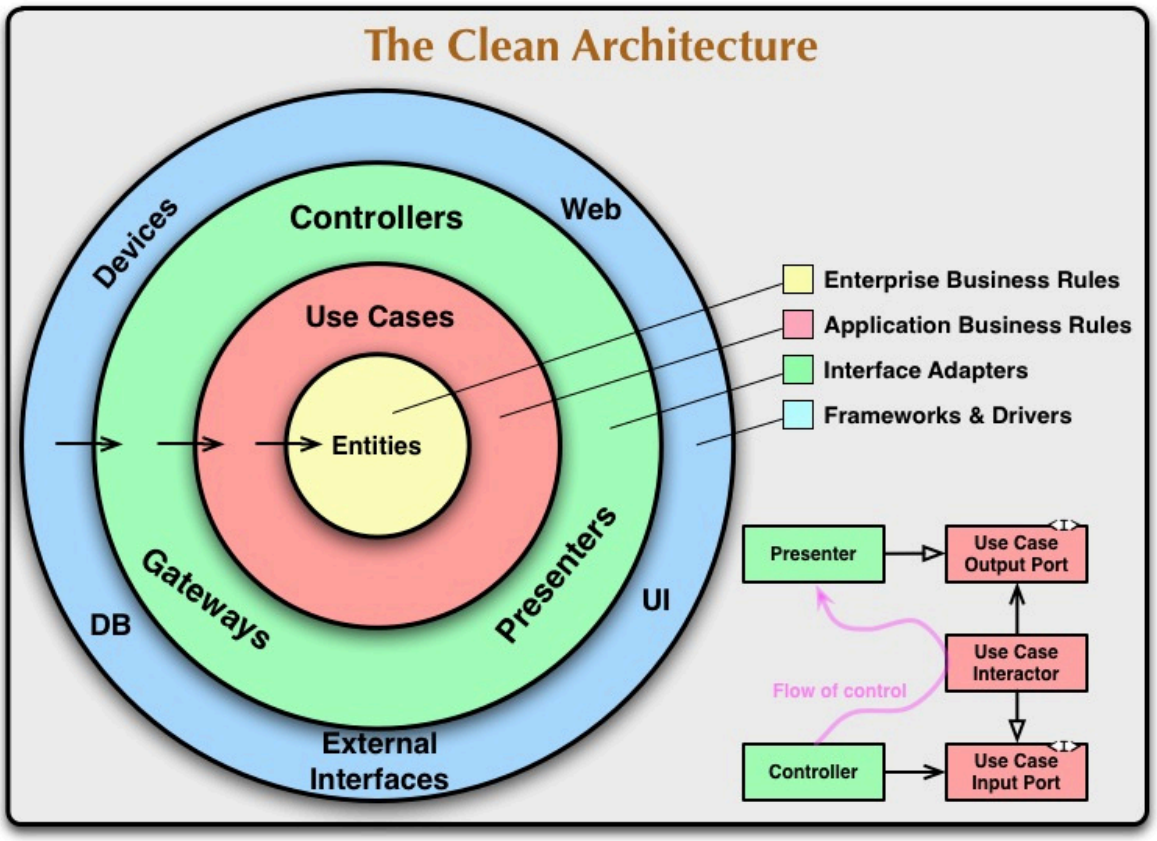
Далее устанавливаем необходимые зависимости:

- express (HTTP сервер)
- bcrypt (шифрование данных)
- dotenv (чтение env переменных)
- jsonwebtoken (создание JWT)
- typeorm (ORM)
- swagger-jsdoc, swagger-ui-express (OpenAPI генерация)

Реализация компонентов

Начнем реализацию boilerplate с разделения по директория.

Необходимо четкое разделение подобно идее чистой архитектуры:



Таким образом получилась следующая структура:

```
src - корневая директория
├── config - инициализация БД, переменных окружения
├── controllers - контроллеры
├── entities - сущности и типы
├── middleware - промежуточное ПО для обработки запросов
│   └── validator - валидация данных в API
├── migrations - скрипты миграций
├── routes - эндпоинты
└── utils - директория с утилитами и вспомогательными функциями

9 directories
```

Авторизация

Авторизация реализована с использованием библиотек `jsonwebtoken` и `bcrypt`.

Во время регистрации и логина пользователю выдается JWT, которые он далее использует при взаимодействии с API. Пароль пользователей хешируется, может быть применено несколько раундов хеш-функции в зависимости от конфигурации проекта.

База данных

Выбранной СУБД стала PostgreSQL, так как является самой популярной и доступной на текущий момент. Для подключения к БД используется TypeORM:

```
import 'reflect-metadata';
import { DataSource } from 'typeorm';
import dotenv from 'dotenv';

dotenv.config();

export const AppDataSource = new DataSource({
 type: 'postgres',
 host: process.env.POSTGRES_HOST,
 port: parseInt(process.env.POSTGRES_PORT || '5432', 10),
 username: process.env.POSTGRES_USER,
 password: process.env.POSTGRES_PASSWORD,
 database: process.env.POSTGRES_DB,
 synchronize: false,
 migrationsRun: true,
 logging: false,
 ssl: true,
 entities: [__dirname + '/../entities/*.{ts,js}'],
 migrations: [__dirname + '/../migrations/*.{ts,js}'],
});
```

Сущности

Сущностью, реализованной в boilerplate, стала сущность пользователя, который имеет имя, фамилию, почту, пароль, роль, дату регистрации и дату обновления информации.

Промежуточное ПО

Было реализовано несколько middleware:

1) Проверка JWT
2) Проверка роли: user или admin
3) Валидация данных при логине и регистрации

Контроллеры и эндпоинты

Реализованы контроллеры и добавлены эндпоинты для получения пользователя/пользователей, редактирования и удаления, а также логина и регистрации.

Вывод

Мы создали удобный boilerplate для переиспользования при создании новых проектов на Node.JS. Научились реализовывать обвязку контроллеров Swagger’ом на Node.JS для генерации документации. Попробовали работу с TypeORM с различными конфигурациями (работа с миграциями и синхронизация созданием таблиц из сущностей).