САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Соловьева П.А.

Группа К3344

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Создать boilerplate на Express.js + TypeORM + TypeScript с явным разделением на:

1) Модели (entities)
2) Контроллеры (controllers)
3) Роуты (routes)

Ход работы

1. Структура проекта

Проект организован по принципу разделения ответственности (Separation of Concerns) со следующей структурой:

```
rental-service/
├── package.json
├── tsconfig.json
├── .env                    # Настройки окружения
├── db.sqlite
├── src/
│   ├── config/
│   │   └── data-source.ts        # Настройка TypeORM
│   │   └── dotenv.ts
│   ├── entities/                 # Модели данных
│   │   ├── User.ts
│   │   ├── Property.ts
│   │   ├── Amenity.ts
│   │   ├── PropertyAmenity.ts
│   │   ├── Rental.ts
│   │   ├── Message.ts
│   │   └── Review.ts
│   ├── repositories/             # Работа с БД
│   │   ├── user.repository.ts
│   │   ├── property.repository.ts
│   │   ├── rental.repository.ts
│   │   ├── message.repository.ts
│   │   └── review.repository.ts
│   ├── services/                 # Бизнес-логика
│   │   ├── user.service.ts
│   │   ├── property.service.ts
│   │   ├── rental.service.ts
│   │   ├── message.service.ts
│   │   └── review.service.ts
│   ├── controllers/              # Контроллеры
│   │   ├── user.controller.ts
│   │   ├── property.controller.ts
│   │   ├── rental.controller.ts
│   │   ├── message.controller.ts
│   │   └── review.controller.ts
│   ├── routes/                   # Роуты
│   │   ├── user.routes.ts
│   │   ├── property.routes.ts
│   │   ├── rental.routes.ts
│   │   ├── message.routes.ts
```

```
│   │   └── review.routes.ts
│   ├── middleware/               # Middleware
│   │   ├── auth.ts
│   │   └── errorHandler.ts
│   └── index.ts                  # Точка входа приложения
└── README.md
```

2. Настройка зависимостей

В `package.json` определены основные зависимости:

```
 "dependencies": {
 "bcryptjs": "^2.4.3",
 "class-transformer": "^0.5.1",
 "class-validator": "^0.14.0",
 "cors": "^2.8.5",
 "dotenv": "^16.3.1",
 "express": "^4.18.2",
 "helmet": "^7.1.0",
 "jsonwebtoken": "^9.0.2",
 "morgan": "^1.10.0",
 "multer": "2.0.2",
 "reflect-metadata": "^0.1.13",
 "sqlite3": "^5.1.6",
 "swagger-jsdoc": "^6.2.8",
 "swagger-ui-express": "^5.0.1",
 "typeorm": "^0.3.17"
},
 "devDependencies": {
   "@types/cors": "^2.8.17",
   "@types/express": "^4.17.21",
   "@types/jsonwebtoken": "^9.0.5",
   "@types/morgan": "^1.9.10",
   "@types/multer": "^1.4.11",
   "@types/node": "^20.10.0",
   "@types/swagger-jsdoc": "^6.0.4",
   "@types/swagger-ui-express": "^4.1.8",
   "nodemon": "^3.0.2",
   "ts-node": "^10.9.1",
   "typescript": "^5.3.2"
 }
```

4. Модели (Entities)

С помощью TypeORM реализованы все необходимые сущности. Ниже пример модели пользователя:

```
import { Entity, PrimaryGeneratedColumn, Column, OneToMany, CreateDateColumn,
UpdateDateColumn } from "typeorm";
import { Property } from "./Property";
import { Rental } from "./Rental";
import { Message } from "./Message";
import { Review } from "./Review";


export type UserRole = 'owner' | 'tenant' | 'admin';
```

```
@Entity()
export class User {
@PrimaryGeneratedColumn("uuid")
id!: string;


@Column()
first_name!: string;


@Column()
last_name!: string;


@Column({ unique: true })
email!: string;


@Column({ nullable: true })
phone_number?: string;


@Column()
password_hash!: string;


@Column({ type: "text", default: "tenant" })
role!: UserRole;


@CreateDateColumn()
created_at!: Date;


@UpdateDateColumn()
updated_at!: Date;


@OneToMany(() => Property, (p) => p.owner)
properties!: Property[];


@OneToMany(() => Rental, (r) => r.tenant)
rentals!: Rental[];


@OneToMany(() => Message, (m) => m.sender)
sent_messages!: Message[];


@OneToMany(() => Message, (m) => m.receiver)
received_messages!: Message[];


@OneToMany(() => Review, (rev) => rev.reviewer)
reviews!: Review[];
}
```

5. Контроллеры (Controllers)

Пример CRUD-контроллера для пользователей:

```
import { Request, Response } from "express";
import { UserService } from "../services/user.service";

export class UserController {
 static async create(req: Request, res: Response) {
   try {
     const saved = await UserService.create(req.body);
     res.status(201).json(saved);
   } catch (err: any) {
     res.status(500).json({ error: err.message });
   }
 }

 static async findAll(req: Request, res: Response) {
   try {
     const { skip = 0, take = 20, email } = req.query as any;

     if (email) {
       // API-эндпоинт для поиска по email
       const user = await UserService.findByEmail(email);
       return res.json(user);
     }

     const users = await UserService.findAll(Number(skip), Number(take));
     res.json(users);
   } catch (err: any) {
     res.status(500).json({ error: err.message });
   }
 }

 static async findById(req: Request, res: Response) {
   try {
     const user = await UserService.findById(req.params.id);
     if (!user) return res.status(404).json({ message: "User not found" });
     res.json(user);
   } catch (err: any) {
     res.status(500).json({ error: err.message });
   }
 }

 static async update(req: Request, res: Response) {
   try {
     const updated = await UserService.update(req.params.id, req.body);
     if (!updated) return res.status(404).json({ message: "User not found"
});
     res.json(updated);
   } catch (err: any) {
     res.status(500).json({ error: err.message });
   }
 }

 static async delete(req: Request, res: Response) {
   try {
     const result = await UserService.delete(req.params.id);
     res.json(result);
   } catch (err: any) {
```

```
    res.status(500).json({ error: err.message });
   }
 }
}
```

6. Роуты (Routes)

Пример маршрутов пользователей с использованием middleware для аутентификации:

```
import { Router } from "express";
import { UserController } from "../controllers/user.controller";

const router = Router();

router.post("/", UserController.create);
router.get("/", UserController.findAll);
router.get("/:id", UserController.findById);
router.put("/:id", UserController.update);
router.delete("/:id", UserController.delete);

export default router;
```

7. Middleware аутентификации

JWT-based middleware для защиты маршрутов:

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";
import { AppDataSource } from "../config/data-source";
import { User } from "../entities/User";

export interface AuthRequest extends Request {
 user?: User;
}

export const authMiddleware = async (req: AuthRequest, res: Response, next:
NextFunction) => {
 try {
   const token = req.header("Authorization")?.replace("Bearer ", "");
   if (!token) return res.status(401).json({ message: "Access denied. No
token provided." });

   const decoded = jwt.verify(token, process.env.JWT_SECRET ||
"fallback-secret") as any;
   const userRepository = AppDataSource.getRepository(User);
   const user = await userRepository.findOne({ where: { id: decoded.userId }
});

   if (!user) return res.status(401).json({ message: "Invalid token." });

   req.user = user;
   next();
 } catch (error) {
   res.status(401).json({ message: "Invalid token." });
 }
```

```
};
```

Вывод

В ходе выполнения лабораторной работы был успешно создан boilerplate на Express.js + TypeORM + TypeScript с четким разделением на модели, контроллеры и роуты.

Достигнутые результаты:

1. Архитектурное разделение: четкое разделение ответственности между слоями (entities, controllers, routes)
2. TypeORM интеграция: работа с базой данных SQLite с автоматической синхронизацией схемы
3. Аутентификация: JWT-based аутентификация с middleware для защиты маршрутов
4. REST API: полный набор CRUD операций для сущностей
5. Безопасность: хеширование паролей через bcryptjs
6. Обработка ошибок: централизованная обработка ошибок
7. Валидация: использование TypeORM декораторов для валидации данных

Технологический стек: Node.js, Express.js, TypeScript, TypeORM, SQLite, JWT

Проект готов к использованию как основа для разработки веб-приложений с REST API и легко расширяем дополнительной функциональностью.