САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Отчет по домашней работе №2 по курсу “Бэкенд разработка”

Выполнили:
Бахарева М. А., К3342
Привалов К.А., К3342

Проверил:
Добряков Д.И.

Санкт-Петербург

2025 г.

1. Задание:

- Реализовать все модели данных, спроектированные в рамках Д31
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- Реализовать API-эндпоинт для получения пользователя по id/email

2. Ход работы

Инициализируем проект и устанавливаем зависимости

С помощью нескольких команд выполняем инициализацию и установку необходимых библиотек:

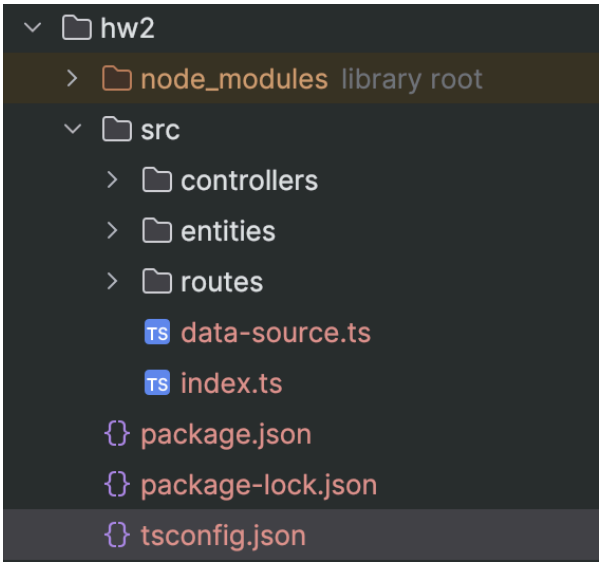
npm init -y
npm install express typeorm reflect-metadata pg
npm install --save-dev typescript ts-node-dev @types/node @types/express

и так далее.

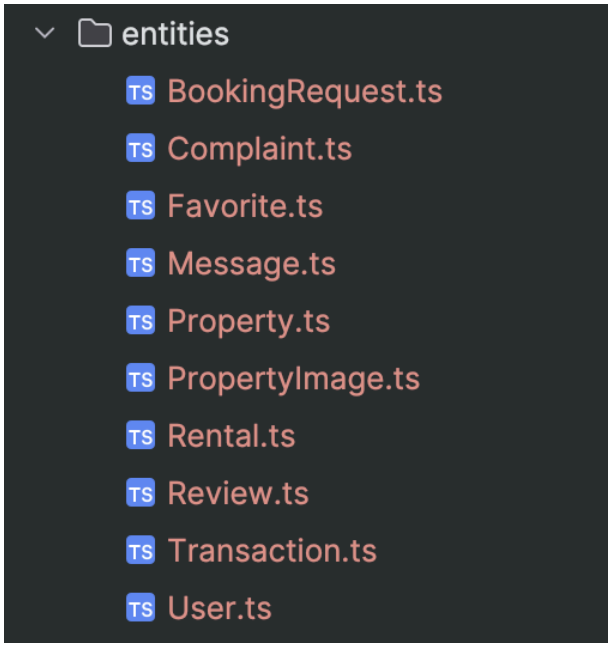
Создаем документ tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "CommonJS",
    "strict": true,
    "esModuleInterop": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "outDir": "./dist",
    "rootDir": "./src",
    "strictPropertyInitialization": false,
    "skipLibCheck": true,
    "resolveJsonModule": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

Структура проекта такая:



Создаем модели по БД, которую делали в ДЗ№1



Пример кода модели User:

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

```
import { BookingRequest } from "./BookingRequest";
import { Complaint } from "./Complaint";

export enum UserRole {
 TENANT = "tenant",
 LANDLORD = "landlord",
 ADMIN = "admin"
}

@Entity()
export class User {
 @PrimaryGeneratedColumn()
 user_id: number;

 @Column()
 first_name: string;

 @Column()
 last_name: string;

 @Column()
 birth_date: Date;

 @Column()
 phone: string;

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

 @Column("varchar")
 role: UserRole;

 @CreateDateColumn()
 created_at: Date;

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

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

 @OneToMany(() => Favorite, fav => fav.user)
 favorites: Favorite[];

 @OneToMany(() => BookingRequest, br => br.tenant)
```

```
 bookingRequests: BookingRequest[];

 @OneToMany(() => Complaint, comp => comp.user)
 complaints: Complaint[];
}
```

Далее создаем контроллеры

1. Получить все записи (GET /)
2. Получить одну запись по ID (GET /:id)
3. Создать (POST /)
4. Обновить (PUT /:id)
5. Удалить (DELETE /:id)

Чтобы избежать дублирования, используем шаблонный подход. Создаем BaseController.ts, который можно использовать для всех остальных контроллеров.

```
import { Request, Response } from "express";
import { Repository, ObjectLiteral } from "typeorm";

export class BaseController<T extends ObjectLiteral> {
 constructor(protected repository: Repository<T>) {}

 getAll = async (_req: Request, res: Response) => {
   try {
     const items = await this.repository.find();
     res.json(items);
   } catch (err) {
     res.status(500).json({ message: "Server error", error: err });
   }
 };

 getById = async (req: Request, res: Response) => {
   try {
     const item = await this.repository.findOne({
       where: {
         [this.repository.metadata.primaryColumns[0].propertyName]:
+req.params.id,
       } as any,
     });
     if (!item) return res.status(404).json({ message: "Not found"
});
     res.json(item);
   } catch (err) {
     res.status(500).json({ message: "Server error", error: err });
```

```
    }
  };

  create = async (req: Request, res: Response) => {
    try {
      const newItem = this.repository.create(req.body);
      const saved = await this.repository.save(newItem);
      res.status(201).json(saved);
    } catch (err) {
      res.status(400).json({ message: "Create failed", error: err });
    }
  };

  update = async (req: Request, res: Response) => {
    try {
      const id = +req.params.id;
      await this.repository.update(id, req.body);
      const updated = await this.repository.findOneBy({
        [this.repository.metadata.primaryColumns[0].propertyName]: id,
      } as any);
      res.json(updated);
    } catch (err) {
      res.status(400).json({ message: "Update failed", error: err });
    }
  };

  delete = async (req: Request, res: Response) => {
    try {
      await this.repository.delete(+req.params.id);
      res.status(204).send();
    } catch (err) {
      res.status(500).json({ message: "Delete failed", error: err });
    }
  };
}
```

По аналогии создаем остальные контроллеры

```
import { AppDataSource } from "../data-source";
import { User } from "../entities/User";
import { BaseController } from "./BaseController";

export const userController = new
BaseController(AppDataSource.getRepository(User));
```

После создаем роуты

Пример для сущности User

```
import { Router } from "express";
import { userController } from "../controllers/userController";

const router = Router();

router.get("/", userController.getAll);
router.post("/", userController.create);
router.put("/:id", userController.update);
router.delete("/:id", userController.delete);
router.get("/:id", userController.getById);


export default router;
```

Регистрируем роуты в index.ts

```
import express from "express";
import "reflect-metadata";
import { AppDataSource } from "./data-source";

import userRoutes from "./routes/userRoutes";
import propertyRoutes from "./routes/propertyRoutes";
import rentalRoutes from "./routes/rentalRoutes";
import messageRoutes from "./routes/messageRoutes";
import favoriteRoutes from "./routes/favoriteRoutes";
import bookingRequestRoutes from "./routes/bookingRequestRoutes";
import reviewRoutes from "./routes/reviewRoutes";
import transactionRoutes from "./routes/transactionRoutes";
import complaintRoutes from "./routes/complaintRoutes";
import propertyImageRoutes from "./routes/propertyImageRoutes";

const app = express();
app.use(express.json());

app.use("/users", userRoutes);
app.use("/properties", propertyRoutes);
app.use("/rentals", rentalRoutes);
app.use("/messages", messageRoutes);
app.use("/favorites", favoriteRoutes);
```

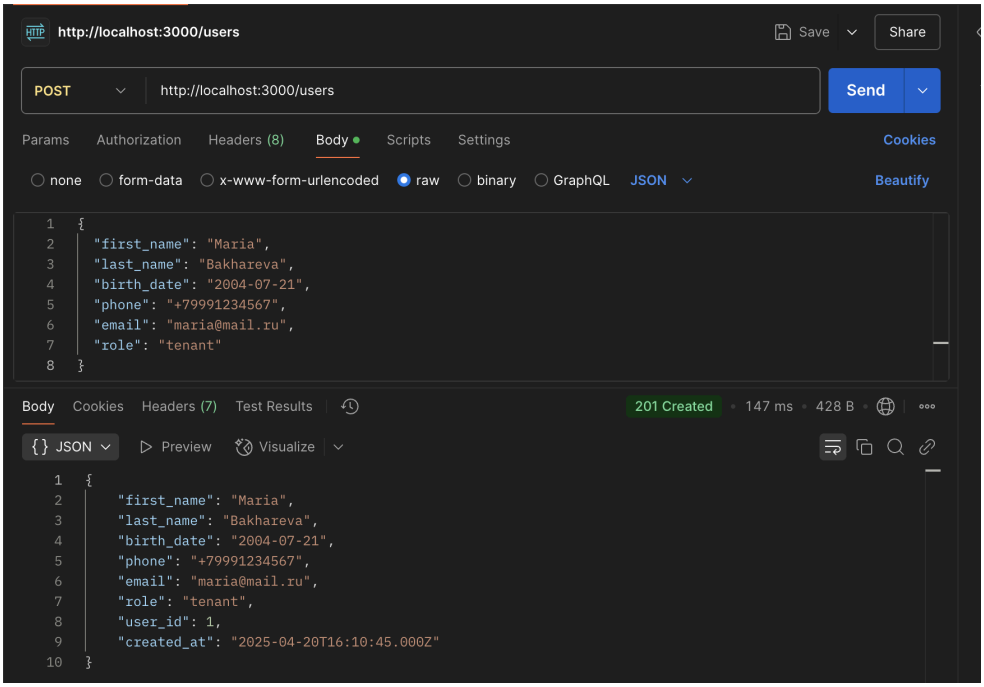
```
app.use("/booking-requests", bookingRequestRoutes);
app.use("/reviews", reviewRoutes);
app.use("/transactions", transactionRoutes);
app.use("/complaints", complaintRoutes);
app.use("/property-images", propertyImageRoutes);

AppDataSource.initialize().then(() => {
 app.listen(3000, () => {
   console.log("Server started on http://localhost:3000");
 });
});
```

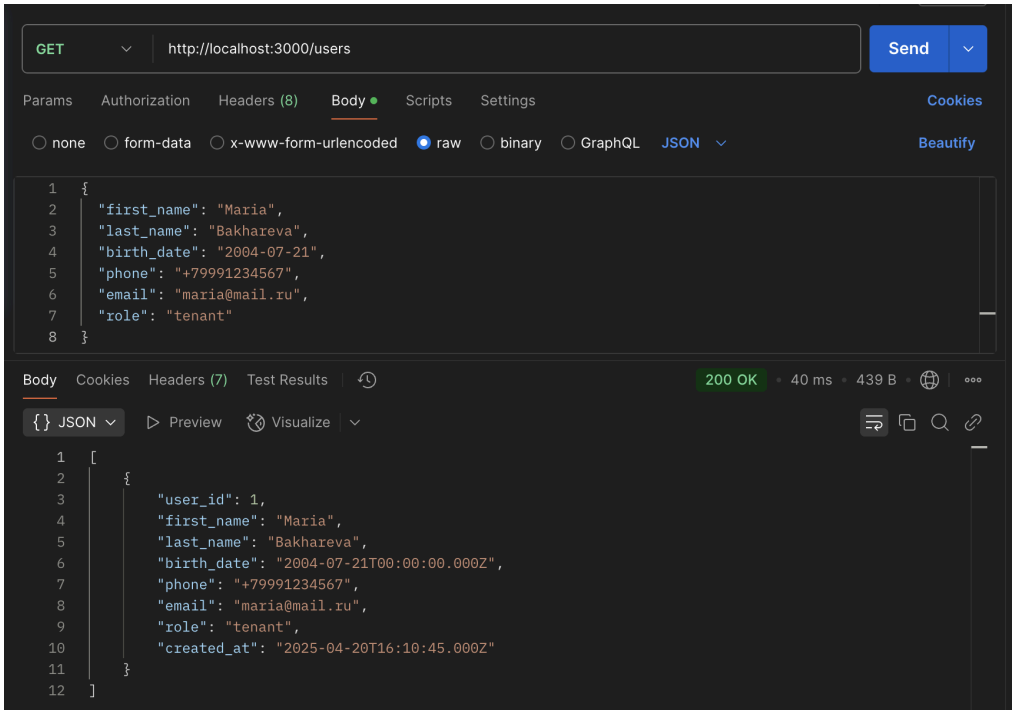
Проверим работоспособность API-эндпоинта для получения пользователя по id

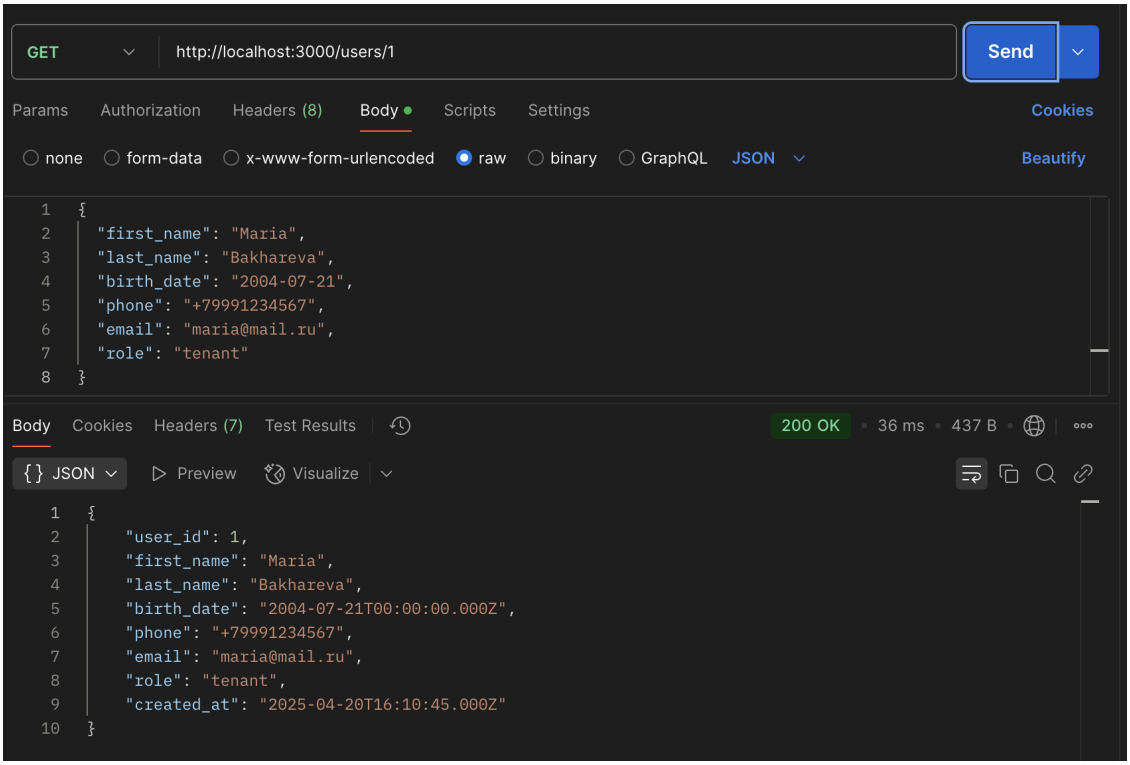
Запустим сервер командой npx ts-node src/index.ts.

```
mariabakhareva@Marias-MacBook-Air-2 hw2 % npx ts-node src/index.ts
Server started on http://localhost:3000
```

С помощью PostMan создадим запись User







Делаем вывод: эндпоинт реализован корректно, информация о созданном User доступна.

3. Вывод

В рамках лабораторной работы была реализована полноценная REST API на основе стека Express + TypeScript + TypeORM. Основные этапы включали:

1. Проектирование и реализация моделей данных в соответствии с заданной предметной областью.
2. Создание универсального контроллера BaseController, предоставляющего базовые CRUD-операции для всех сущностей.
3. Настройка маршрутов (routes) для каждой модели, с правильной обработкой запросов и корректной типизацией.
4. Подключение и конфигурация TypeORM, включая возможность дальнейшего подключения к базе данных.
5. Решение ошибок типизации TypeScript, связанных с использованием generic-контроллеров и типов Express.