

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа 2

Выполнила:

Космач Мария

Группа К3339

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## **Задача**

- Реализовать все модели данных, спроектированные в рамках ДЗ1;
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript;
- Реализовать API-эндпоинт для получения пользователя по id/email.

## **Ход работы**

Вариант: Сервис для аренды недвижимости

### **1. Реализация моделей**

Были выделены следующие модели: User, Message, Property, Comforts, RentalAgreement, Review, Photo

Также были созданные необходимые enum-файлы: RentalAdvertisementStatus, RentType, RentalStatus.

Пример реализованной модели RentalAgreementEntity предоставлен на рисунке 1.

Все оставшиеся модели были созданы по аналогии с показанной сущностью.

```

@Entity({name: 'rentals'}) Show usages
export class RentalAgreementEntity extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @ManyToOne(() : typeof PropertyEntity => PropertyEntity)
  @JoinColumn({name: "property_id"})
  property: PropertyEntity;

  @ManyToOne(() : typeof UserEntity => UserEntity)
  @JoinColumn({name: "renter_id"})
  renter: UserEntity;

  @Column({type: "decimal", name: "total_price"})
  totalPrice: number;

  @Column({type: "timestampz", name: "start_date"})
  startDate: Date;

  @Column({type: "timestampz", name: "end_date"})
  endDate: Date;

  @Column({
    type: "enum",
    enum: RentalStatus,
    name: "status"
  })
  status: RentalStatus;

  @OneToMany(() : typeof ReviewEntity => ReviewEntity, review : ReviewEntity => review.rentalAgreement)
  reviews: ReviewEntity[];
}

```

Рисунок 1 - RentalAgreementEntity

## 2. Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript;

Для каждой модели был реализован контроллер, содержащий набор CRUD-методов для работы с сущностями. Также были реализованы роуты.

Пример кода контроллера для сущности RentalAgreement показан на рисунке 2, остальные контроллеры реализованы по аналогии

```

class RentalAgreementController { Show usages
  private repository = dataSource.getRepository(RentalAgreementEntity);

  getById: RequestHandler = async (req, res, next) => { Show usages
    try {
      const id = parseInt(req.params.id);
      const rental = await this.repository.findOne({
        where: {id},
        relations: ['property', 'renter', 'reviews'],
      });
      if (!rental) {
        res.status(404).json({message: 'Rental Agreement not found'});
        return;
      }
      res.status(200).json(rental);
    } catch (err) {
      next(err);
    }
  };

  getForRenter: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record
    try {
      const renterId = parseInt(req.params.renterId);
      const rentals : RentalAgreementEntity[] = await this.repository.find({
        where: {renter: {id: renterId}},
        relations: ['property', 'reviews'],
      });
      res.status(200).json(rentals);
    } catch (err) {
      next(err);
    }
  };
};

```

Рисунок 2 – RentalAgreementController

3. Реализовать API-эндпоинт для получения пользователя по id/email. Для сущности User был сделан контроллер и роутер.

Реализация контроллера продемонстрирована на рисунке 3.

```

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

  getUserById: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<a
    try {
      const id : number = parseInt(req.params.id);
      const user : UserEntity = await this.repository.findOneBy({ id });
      res.status(200).json(user);
    } catch (err) {
      next(err);
    }
  };

  getUserByMail: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response
    try {
      const mail : string = req.params.mail;
      const user : UserEntity = await this.repository.findOneBy({ email: mail });
      res.status(200).json(user);
    } catch (err) {
      next(err);
    }
  };
}

export default new UserController(); Show usages

```

Рисунок 3 - UserController

## Вывод

В рамках работы были созданы модели, спроектированные в рамках ДЗ1, для моделей были созданы контроллеры, в которых содержатся CRUD-методы, работа которых реализована при помощи TypeORM; и роутеры.