🏠Техническое задание

Проект:

HomeRent

— система аренды жилья

---

1. Общая идея

Нужно разработать backend-сервис HomeRent для компании, которая сдаёт квартиры/дома в долгосрочную и краткосрочную аренду.

Система должна позволять:

Пользователям (арендаторам):

- регистрироваться и входить в систему;
- просматривать доступные объекты недвижимости;
- бронировать объект на выбранные даты;
- подписывать договор (загрузка скана PDF);
- оплачивать аренду (эмуляция, без реального платежного сервиса);
- получать подтверждения и квитанции на email;
- оставлять заявки на ремонт/поломки.

Владельцам объектов (ROLE_OWNER):

- управлять списком собственных объектов;
- видеть запросы арендаторов;
- прикреплять документы к объекту (фото, схемы, PDF).

Операторам компании (ROLE_OPERATOR):

- отслеживать активные аренды;
- проверять документы;
- видеть все заявки на поломки.

Администраторам (ROLE_ADMIN):

- управлять пользователями и ролями;
- управлять всеми объектами независимо от владельца.

Система должна использовать все темы курса, аналогично проекту ShareRide.

---

2. Технологический стек (обязательно)

Java 17+

Spring Boot 3+

Maven

Зависимости:

- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-security
- spring-boot-starter-mail
- spring-boot-starter-validation
- springdoc-openapi-starter-webmvc-ui
- Lombok
- Liquibase
- spring-boot-starter-test
- spring-security-test

БД:

- H2 для dev и test
- PostgreSQL/MySQL — опционально для prod

Минимум 1 шаблон Thymeleaf для email.

---

3. Модель данных (JPA)

3.1. User

- id
- username (уникальный)
- email
- password (BCrypt)
- enabled
- roles : Set

3.2. Role

- id
- name (ROLE_TENANT, ROLE_OWNER, ROLE_OPERATOR, ROLE_ADMIN)

3.3. Property (объект недвижимости)

- id
- owner : User (ROLE_OWNER)

- title
- address
- description
- pricePerDay
- status (AVAILABLE, BOOKED, RENTED, UNAVAILABLE)
- photos (опционально список)
- createdAt

3.4. Booking (бронирование)

- id
- property : Property
- tenant : User
- startDate
- endDate
- status (REQUESTED, APPROVED, REJECTED, ACTIVE, FINISHED)
- totalPrice

3.5. RentalContract (договор аренды)

- id

- booking : Booking
- filePath (PDF)
- uploadedAt

3.6. IssueReport (заявка о поломке)

- id
- booking : Booking
- reportedBy : User
- description
- photoPath
- createdAt
- status (OPEN, IN_PROGRESS, DONE)

---

4. База данных и Liquibase

4.1. Структура changelog’ов

1. 1.0-create-tables
    - users, roles, user_roles
    - properties
    - bookings
    - rental_contracts
    - issue_reports
2. 1.1-insert-reference-data
    - роли: TENANT, OWNER, OPERATOR, ADMIN
    - тестовый админ
    - несколько тестовых объектов недвижимости
3. 1.2-test-objects
    - тестовые бронирования и заявки о поломках

4.2. Профили

- application-dev.properties → H2, show-sql
- application-test.properties → H2 isolated
- application-prod.properties → PostgreSQL/H2 file

5. REST API

Все эндпоинты должны быть описаны в Swagger UI.

5.1. Публичные эндпоинты

```
GET /api/public/info
POST /api/auth/register
POST /api/auth/login
```

5.2. API Tenant (ROLE_TENANT)

Просмотр объектов

```
GET /api/tenant/properties/available
GET /api/tenant/properties/{id}
```

Бронирование

```
POST /api/tenant/bookings
(startDate, endDate, propertyId)
```

Управление бронированием

```
GET /api/tenant/bookings/my
POST /api/tenant/bookings/{id}/upload-contract (MultipartFile)
GET /api/tenant/bookings/{id}
```

Заявка на поломку

```
POST /api/tenant/issues
(description, bookingId, photo)
```

5.3. API Owner (ROLE_OWNER)

```
GET /api/owner/properties
POST /api/owner/properties
DELETE /api/owner/properties/{id}

GET /api/owner/bookings/pending  (ожидающие подтверждения)
POST /api/owner/bookings/{id}/approve
POST /api/owner/bookings/{id}/reject
```

5.4. API Operator (ROLE_OPERATOR)

```
GET /api/operator/bookings/active
GET /api/operator/issues
POST /api/operator/issues/{id}/status
```

5.5. API Admin (ROLE_ADMIN)

```
GET /api/admin/users
POST /api/admin/users
POST /api/admin/users/{id}/roles
GET /api/admin/properties
POST /api/admin/properties
DELETE /api/admin/properties/{id}
```

---

6. Spring Security

6.1. Аутентификация

- собственный UserDetailsService
- BCrypt
- @EnableMethodSecurity

6.2. Авторизация

Правила аналогичны ShareRide:

- public, auth — permitAll
- tenant endpoints → ROLE_TENANT
- owner endpoints → ROLE_OWNER
- operator → ROLE_OPERATOR
- admin → ROLE_ADMIN
- остальные → authenticated

6.3. @PreAuthorize – минимум 2 метода

Примеры:

- Tenant может видеть только свои бронирования.
- Owner может изменять только свои объекты.

---

7. Загрузка файлов

Необходимо:

- загрузка PDF договора
- загрузка фото поломок (MultipartFile)

Пример:

```
POST /api/tenant/bookings/{id}/upload-contract
POST /api/tenant/issues (MultipartFile photo)
```

Файлы можно хранить:

- либо в файловой системе /uploads/
- либо в БД (byte[])

---

8. Email уведомления (Spring Mail + Thymeleaf)

Сценарии:

1. Подтверждение бронирования

Содержит:

- адрес объекта
- даты аренды
- стоимость

2. Подтверждение загрузки договора

“Ваш договор успешно загружен”

3. Уведомление об окончании аренды (квитанция)

- даты
- итоговая цена
- объект

---

9. Тестирование

9.1. Unit/Controller Tests (MockMvc)

- публичные эндпоинты → 200
- защищённые без авторизации → 401
- проверка ролей → 403 / 200

9.2. Интеграционные тесты (H2 + Liquibase)

Сценарии:

1. Полный цикл аренды:

- арендатор создаёт бронирование
- владелец подтверждает
- арендатор загружает договор
- оператор видит активную аренду

2. Заявка поломки:

- создание → просмотр оператором → смена статуса

---

10. Swagger / Postman

- Swagger UI должен показывать все группы эндпоинтов
- Postman-коллекция:
    - регистрация
    - логин
    - просмотр объектов
    - бронирование
    - загрузка договора
    - заявки на ремонт
    - операции owner/operator/admin

---

11. Deployment

- сборка jar
- запуск через профиль prod
- README с инструкцией

Дополнительно:

- Dockerfile

---

12. Что вы должны сдать

1. Git-репозиторий
2. Liquibase changelogs
3. Email-шаблоны
4. Swagger скриншоты
5. Postman скриншоты
6. Unit + integration tests
7. README с инструкциями
