SPRIBE

Test task

for Senior Java Developer

ver. 2

Please create an application like a booking system. The Java application should be a service with a REST API (monolith), built using Spring Boot, Spring Web and Spring Data.

Application description:

in the application Users can add the new booking subject - **Unit** - with a set of properties:

- parameter 1: number of rooms;
- parameter 2: type of accommodation (HOME, FLAT, APARTMENTS);
- parameter 3: **Unit**'s floor.

The **Unit** has an availability property - the ability to select. **Unit** availability is determined by a booking for a range of dates. Each **Unit** has its own cost, specified by the User on creating, plus 15% of the booking system markup. The **Unit** also has a general description specified by the User. In addition to adding a new **Unit**, it is necessary to develop a query (endpoint) for selecting **Units** by all specified criteria, with the date range and cost. The query should return a sortable list with custom pagination (page, number of objects per page). The User must be able to book a **Unit**, after which it becomes unavailable for other users to choose from, as well as the ability to cancel the booking for the selected **Unit**. The final case is emulation of payment for a booking, without which it is automatically cancelled after 15 minutes.

Additional requirement: the main statistics - the number of **Units** available for booking - must be stored in the cache. The caching system is chosen by the Developer, it can be a thread-safe Map, or an external caching system, for example, Redis, Memcached or Hazelcast, or any other custom cacher. The cache must be updated with each change in the status of **Units**, and also have the ability to recover from a potential system crash. An additional endpoint is required to obtain the number of **Units** available for booking.

Requirements to the Test task:

- Java any versions since 21;
- preferably use Gradle for building the project;
- don't use Spring Security Framework (or it's not necessary);
- for a database use PostgreSQL, manage DB schema with Liquibase (any format - xml, sql, yaml, json); data management should be provided with Spring Data JPA or Spring Data JDBC;
- an application can be started in the native environment (without containers); PostgreSQL and any other possible services should be started in Docker containers; Docker containers should be described with docker-compose file;
- mostly models, controllers and services should be covered by unit and functional tests, preferably JUnit 5 and Spring Test Framework;

- the project should have API documentation (preferably Swagger/OpenAPI Specification - dynamic or static);
- a DB structure must contain at least tables for **Units**, their properties, Users, payment, events;
- you can skip an input and output models validation;
- add 10 Units with properties and creation events in the Liquibase ChangeLog, and 90 Units with random parameters and costs on an application start.

Please archive the project and send it to the HR manager.