Business requirements

This task is an extension of the RESTful web-service from Home Task #1.

Application should support *user-based authentication*. This means a user is stored in a database with some basic information and a password.

The system should be *extended* to expose the following REST APIs:

1. Change *single field* of **main entity**.
2. Make an *order (or any relevant action)* on **main entity** for a User.
3. Get information about user's *orders*.
4. Get information about user's *order*: cost and timestamp of a purchase.
    5. Get the most *widely used* secondary entity of a user with the highest cost of all *orders*.
        a. Demonstrate SQL execution plan for this query.
6. Search **main entity** by several **secondary entities** ("and" condition).
7. Sign up.
8. Log in.
9. All *read* operations should support pagination. Please create a flexible and non-erroneous solution. Handle all exceptional cases.

User Permissions

Guest:

1. Read operations for **main entity**.
2. Sign up.
3. Log in.

User:

1. Make an order on **main entity**.
2. All read operations.

Administrator (can be added only via database call):

1. All operations, including addition and modification of entities.

Technical requirements

General requirements

1. Code should be clean and should not contain any "developer-purpose" constructions.
2. Application should be designed and written with respect to OOD and SOLID principles.
3. Code should contain valuable comments where appropriate.
4. Public APIs should be documented using Javadoc.
5. A clear layered structure should be used: responsibilities of each application layer should be defined.
6. JSON should be used as a format of client-server communication messages.
7. Abstraction should be used to avoid code duplication.
8. Convenient error/exception handling mechanism should be implemented: all errors should be meaningful.

App requirements

1. JDK version: 8. Use Streams, java.time.*, an etc. where it is appropriate.
2. Application packages root: com.epam.esm.
3. Spring Boot.
4. JPA / Hibernate should be used for data access.
5. Audit data should be populated using JPA features.
6. Java Code Convention is mandatory. Exception: margin size – 120 characters.
7. Build tool: Gradle 4.+. Multi-module project.
8. Spring Framework 5.+.
9. Application container: Spring IoC.

10. Database: PostgreSQL 9.+ or 10.+.
11. Testing: JUnit 5.+. Mockito.
12. Service layer should be covered with unit tests not less than 80%.
13. Spring Transaction testing should be performed in scope of this task.
14. Spring Security should be used as a security framework.
15. Server should support only stateless user authentication and verify integrity of JWT token.
16. Use OpenID as an authentication protocol. At least two different providers should be used.
17. Implement CSRF protection.
18. For demo, generate at least:
    a. 1000 users,
    b. 1000 secondary entities,
    c. 10 000 main entities.
    d. All entities should be linked. All values should look meaningful: random words, but not random letters.
19. Pagination should be implemented for all GET-all endpoints.
20. APIs should be demonstrated using Postman tool.
21. For demo, prepare Postman collections with APIs.
22. No major sonar issues should exist. Use default configuration for sonarqube.

Please note that only GA versions of tools, frameworks, and libraries are allowed.

App restrictions

It is forbidden to use:

1. Powermock.
2. Any Hibernate specific features.

Materials

This section contains links to materials recommended for self-study.

a. Spring in Action, 4th edition
b. Foundations of RESTful architecture & REST practices
c. Best Practices for Designing a Pragmatic RESTful API
d. Data Transfer Object
e. Understanding Spring Web Application Architecture: The Classic Way
f. PowerMock examples and why better not to use them
g. Аутентификация при помощи JWT и Spring Security
h. JWT
i. OAuth 2.0 простым и понятным языком
j. Spring Boot and OAuth2
k. OAuth2 Grants
l. Spring Transaction Testing
m. Random Words Generator
n. Java/JPA Developer's Guide
o. CSRF

Optional tasks

This section contains additional tasks which will help you learn more and get acquainted with more technologies and engineering approaches.

1. Use Oauth2 as an authorization protocol.
    a. OAuth2 scopes should be used to restrict data.
    b. Implicit grant and Resource owner credentials grant should be implemented.
2. Migrate to use Spring Data repositories (additional branch in repository is required to maintain changes).

3. Support HATEOAS on REST endpoints

Please note that optional tasks should be started only if all main requirements are met and after discussion with mentor. Having optional tasks completed without completing main part of this task will lead to necessity to postpone demo.