PG6100 Enterprise 2, Continuation Exam, June 2018

This assignment is worth 100% of your final grade for PG6100. The assignment has to be submitted by email to the address from where you got these instructions from. You have 72 hours to complete this exam. Exact submission deadline will be provided when you receive these instructions by email.

The exam assignment will have to be zipped in a zip file with name *pg6100_<id>.zip*, where you should replace <id> with the unique id you received with these instructions. If for any reason you did not get such id, use your own student id, e.g. *pg6100_701234.zip*. No "rar", no "tar.gz", etc. You need to submit all source codes (eg., .kt and .xml), and no compiled code (.class) or libraries (.jar, .war). As the assignment has to be sent by email, the zip file should be small (usually email providers have a hard limit of 10MB for attachments). If your zip file is too large and cannot be sent by email, you will fail the exam and get an **F**.

The delivered project should be compilable with Maven 3.x with commands like "mvn install -DskipTests" directly from your unzipped file. Compilation failures will heavily reduce your grade. All tests should run and pass when running "mvn verify".

Note: during the exam period, I will NOT answer to any email, unless there are major issues (e.g., you got instructions for a different exam). In case of ambiguities in these instructions, do your best effort to address them (and possibly explain your decisions in the readme file).

Easy ways to get a straight **F**:

- have production code (not the test one) that is exploitable by SQL injection
- submit a solution with no test at all, even if it is working
- submit your delivery as a rar file instead of a zip (yes, I do hate rar files)

Easy ways to get your grade *strongly* reduced (but not necessarily an F):

- submit code that does not compile
- do not provide a *readme.md* file (more on this later) at all, or with missing parts
- skip/miss any of the instructions in this document (e.g., how to name the zip file)
- having bugs in your application that I find when I run and play with it

The exam must **NOT** be done in group: each student has to write the project on his/her own. During the exam period, you are not allowed to discuss any part of this exam with any other student or person. Failure to comply to these rules will result in an **F** grade and possible further disciplinary actions.

The exam consists in building an enterprise application using a microservice architecture. The main goal of the exam is to show the understanding of the different technologies learned in class. The more technologies you can use and integrate together, the better.

The enterprise application you are going to build will be tailored on a specific topic/theme, which will be discussed later in this document. However, the architecture and requirements of what to use (e.g., REST APIs behind an API Gateway) is independent from the topic/theme of the application.

The project has to be something new that you write, and not re-using existing projects (e.g., existing open-source projects on GitHub). If you plagiarize a whole project, not only you will get an **F**, but you

will be subject to further disciplinary actions. You can of course re-use some existing code snippets (e.g., from StackOverflow) to achieve some special functionalities (but recall to write a code comment about it, e.g., a link to the StackOverflow page). You are also allowed to copy&paste&adapt any code from the repository of the course at:
https://github.com/arcuri82/testing_security_development_enterprise_systems

The marking will be strongly influenced by the quality and quantity of features you can implement. For each main feature, you **MUST** have an end-to-end test to show it, and also you need to discuss it in the documentation (i.e., as a code comment */** … */* before each of the *@Test* methods). *A feature without tests and documentation is like a feature that does not exist*. For **B/A** grades, the more features you implement and test in your project, the better. Note about the evaluation: when an examiner will evaluate your project, s/he will run it and manually test it. Bugs and crashes will negatively impact your grade. Also note that integration (e.g., RestAssured when testing REST API in isolation) and end-to-end tests will play a *major* role in your grading.

In the project, the following requirement must be implemented. Note: considering 72 hours, it might well be that you will not be able to do all of them. In such case, address them in order (e.g., do not add AMQP until you have completed the REST API).

Requirements:

1. Provide a *readme.md* file in the main root folder of your project (i.e., same place as the root *pom.xml* file). In such file, you must briefly discuss your solution, providing any info you deem important. If you do not attempt to do some of the requirements, you must state so in the *readme.md* file (this will make the marking of your exam much faster), e.g., “I did not do requirement X, Y and Z”. Failure to state so will reduce your grade.
2. Write one REST API using Spring Boot and Kotlin, targeting JDK 8 (NO later version, e.g., not 9, not 10)
    a. Have at least one endpoint per main HTTP method, ie, GET, POST, PUT, PATCH and DELETE. The PATCH endpoint must use *JSON Merge Patch.*
    b. The API must use a SQL database.
    c. Provide Swagger/OpenAPI documentation for all your endpoints. You should also provide a starting class that can be used to start the application and see the UI for the Swagger documentation. Note: in this case, you can use an embedded SQL database (e.g., H2). In the *readme.md* file, specify the starting class and the URL to open the UI. In other words, an examiner should be able to start the API directly from an IDE, and point his/her browser to such Swagger documentation. Note: you might want to use a special profile to disable connections to other services, like Redis caches, which could prevent the REST API from starting when started in isolation.
    d. Write at least one test with RestAssured per each endpoint.
    e. If the service communicates with another service, you need to use WireMock in the tests to mock it.
    f. Configure Maven to build a self-executable uber/fat jar for the service.
    g. Write a Docker file for the service.

3. Your microservices should be accessible only from a single entry point, i.e., an API Gateway (e.g., Zuul).
    a. You need at least one REST API service that is started more than once (i.e., more than one instance in the Docker-Compose file), and load balanced with Eureka.
    b. In Docker-Compose, use real databases (e.g. PostgreSQL) instead of embedded ones (e.g., H2) directly in the services.
    c. Write at least one end-to-end test in which you start (using Docker-Compose) the Gateway, Eureka, 2 instances of the REST API, and a PostgreSQL database. All end-to-end tests need to be in their own folder/module (as in the examples shown in class).
4. You need to have security mechanisms (using Spring Security) in place to protect your REST APIs.
    a. You need to create the concept of "user" in your microservices. You will need to create an API with database to handle registration and login of users. Note: most of this would just be an adaptation from the examples shown in class.
    b. You must use distributed session-based authentication with Redis. Note: this latter can be disabled/changed (i.e., no Redis) when the REST API is tested in isolation.
    c. Write at least one end-to-end test (using Docker-Compose) to show that the distributed session-based authentication is working.
5. You need at least one communication relying on AMQP.
    a. Write a new service (can be as simple as a REST API with a single endpoint).
    b. Make at least one communication from the main REST API to such new service, using RabbitMQ.
    c. Write at least one end-to-end test (using Docker-Compose) to show that the AMQP communications are working.
6. Extras: In the eventuality of you finishing all of the above requirements, and only then, if you have extra time left you should add functionalities/features to your project. Those extra functionalities need to be briefly discussed/listed in the *readme.md* file (e.g., as bullet points). Each major new visible feature must have at least one test to show/verify it. Note: in the marking, the examiner might ignore extra functionalities that are not listed/discussed in the readme. What type of functionalities to add is completely up to you.

Note that there is no requirement to build a Front-End GUI. This means that it is possible to get an **A** grade without any GUI at all (well, you still get the automatically generated ones for Swagger documentation). However, a GUI would be an extra feature that would look positive for your grade. If you are going to build a web GUI, you can choose whatever technology you like, e.g., JavaScript frameworks like React or Angular. However, although a GUI is a nice addition, the main topic of Enterprise 2 is the back-end. Considering that you only have 72 hours, you might prefer to write more tests or add further REST endpoints than building a GUI.

Topic/Theme of the Application

The topic of the application is the trading of used books in a university. The application should have *at least* the following functionalities:

1. The main REST API should have the following features:
    a. Ability to CRUD (create/read/update/delete) book information
    b. For a user, specify if s/he has a copy of it and willing to sell it (and for how much)
    c. For each book, being able to retrieve the list of users willing to sell it
2. Regarding security, only an admin should be able to modify book information. A logged in user should be able to modify his/her info (e.g., price of book), but not the info of other users (e.g., for how much they sell their books)
3. The second (smaller) REST API should keep track of the last 10 events in which a user registers the fact s/he wants to sell a book. Each time this event happens in the main REST API, that must send a message on AMQP, which is going to be read by the second API. Such second API should have a GET method to retrieve such list of the most recent 10 events.

THIS MARKS THE END OF THE EXAM TEXT