PG6102 Final Exam

The exam should 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, not even the lecturer (i.e., do not ask questions or clarification on the exam). In case of ambiguities in these instructions, do your best effort to address them (and possibly explain your decisions in the readme file). Failure to comply to these rules will result in an F grade and possible further disciplinary actions.

The students have 72 hours to complete the project. See the details of submission deadline from where you got this document. Note: as a general rule, usually there is no deadline extension on this type of exams. And, even if administration grants an extension (e.g., for medical reasons), it should be no more than 50% of the original amount (i.e., a total of 108 hours in a 72 hour exam). If for any reason you got granted an extension longer than that, you must contact administration to verify the course responsible had agreed on such extension (there were cases in the past in which such unauthorized extensions were given by mistake). Do NOT contact the course responsible directly, as exams must be marked anonymously. To make the exam conditions fair to all students, submissions with long extensions that were not authorized by the course responsible will be automatically evaluated as failed (i.e., an F).

The exam assignment will have to be zipped in a zip file with name pg6100_.zip, where you should replace 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 (e.g., .kt and pom.xml files), and no compiled code (.class) or libraries (.jar, .war).

The delivered project must be compilable with Maven 3.x with commands like "mvn package -DskipTests" directly from your unzipped file. The project must be compilable and runnable with JDK 11. The project must be self-contained, in the sense that all third-party libraries must be downloadable from Maven Central Repository (i.e., do not rely on SNAPSHOT dependencies of third-party libraries that were built locally on your machine). All tests MUST run and pass when running "mvn clean verify". Note: examiners will run such command on their machine when evaluating your delivery. Compilation failures will heavily reduce your grade.

You must provide a "readme.md" file (in Markdown notation, in the same folder as the root pom.xml) where you briefly discuss your solution, providing any info you deem important. You are NOT allowed to write it in other formats, like .docx or pdf.

The Assignment

Your assignment is to create a microservice environment that processes orders for an online shop. You should have a minimum of three services. These services will be created using Kotlin, Spring, Postgres, and RabbitMQ.

Order service: This is the service that manages creating and updating orders, along with communicating with other services as required.

Payment service: This is the service that is responsible for processing payment.

Shipping service: This is the service responsible for processing the order for shipment.

Once an order is created with the order service, it should communicate via http to the payment service in a synchronous manner, and once the payment is updated, it should update the order to reflect that payment has been made.

Once payment is successful, it should communicate to the shipping service to handle shipping the order. Once that is complete, the shipping service should communicate back to the order service that shipment is complete, and the order service should update the order.

All services should be using Postgres in a docker container to persist data using Spring Data JPA.

All services should have integration tests using MockMvc and Testcontainers, as well as WireMock where relevant.

For the Postgres containers, you have the choice of setting up multiple db containers, or using a single db container with multiple databases created in it with the init.sql, as covered by the gitlab repo final-demo.

You may use any resources from this course, from demo code we have done in class, from previous classes, and the internet.

What I am looking for is that you understood the material presented this semester, and are able to implement it. If there is anything that you cannot get working, then put that in your readme.md, along with what you tried, and where you tried to find answers.

Minimum for E grade:

Three services, as described above, including tests. All REST endpoints should return wrapped responses using ResponseEntity.

Provide a docker-compose.yml that starts up any required databases, along with a rabbitmq service with predefined queues.

Minimum for D grade:

The order service and shipping service should communicate using RabbitMQ message queues. The integration tests should be using RabbitMQ testcontainers for these.

Minimum for C grade:

There should be a Gateway Service, that runs on port 8080, that all requests are made against, and does the routing for the other three services.

Spring Caches should be implemented where appropriate.

Minimum for B grade:

There should be a Discovery Service, using Eureka Server/Client, and the Gateway should also use Eureka for discovery.

Queries to the databases should implement Pagination.

Minimum for A grade:

There should be custom exceptions for things like “Order not found”, and “User not found”, etc. These exceptions should be handled globally using @ControllerAdvice.

There should be CircuitBreakers implemented where appropriate, with appropriate fallback methods.

Provide Dockerfiles for all services.

Additionally, the docker-compose should be able to start up the discovery, gateway, order, shipping, and payment services using profiles.

If I run the docker compose, I should get Postgres and rabbitmq containers. If I run the docker-compose with -p flags, then it should be able to start those services as well.

Remember that containers cannot reference localhost in another container, so you will have to override the environmental settings, as we did in class.