Distributed computing
RabbitMQ – Assignment 3

Concurrency using processes (actors) in a distributed environment

Learning objectives

The purpose of this assignment is to demonstrate that you have mastered the principles of distributed programming using processes that work together through a message queue.

Assignment

We are going to build a booking system for ConferenceRent.com. ConferenceRent.com is a company where buildings can rent out conference rooms. A building can register with ConferenceRent. The ConferenceRent system then knows about the existence of the building and customers can rent conference rooms at these buildings.

ConferenceRent.com would like to operate internationally and therefore wants to be prepared for exponential growth. The system must therefore be scalable and robust. That is why a distributed architecture is chosen. RabbitMQ provides this and will be used as an underlying technology. There is not yet much experience in developing a distributed system through RabbitMQ and to gain some experience you are asked to use RabbitMQ to build a simulation for renting out conference rooms. So, a simplified version of RentARoom.com is being built.

The assumption is that a customer does not communicate directly with a building, but only through the rental agent (a kind of intermediary) of ConferenceRent. To be able to handle as many customers as possible at the same time, multiple processes are needed to fulfill the function of a rental agent. These processes must be able to be switched on and off depending on the load of the system.

The application must provide the following functionality:

- A customer must be able to request a list of all buildings
- A customer can book one or more conference rooms in a building.
    - When the rooms are available, the customer receives a unique reservation number.
    - If they are not available, the customer will also receive a message that this is the case.
    - A reservation must be confirmed by the customer with the reservation number before it is final
- A customer must be able to cancel an existing reservation. This can be done by means of the reservation number.
- New buildings must be able to be connected to the system on-the-fly.
- Customers do not communicate directly with a building, but with a rental agent who takes care of the booking of the conference room at the building. To be able to scale horizontally the system must support the use of multiple rental agents. The client does not know which individual rental agent he is communicating with.
  Multiple rental agents must be able to exist at the same time. Communication from clients to these rental agents is then done in a round robin way.
- The system must neatly catch incorrect situations. For example, confirming a reservation with an unknown reservation number should result in a neat error message.

Tip:
To build the system you will have to use both fanout exchanges and direct exchanges.

To be submitted to Brightspace

1. A report in which you describe your approach to the problem. You must explain your design on the basis of one or more diagrams. Also indicate how the program has been tested and what the test findings are, to make it plausible that your solution meets the requirements.

2. A zip-file containing your IntelliJ project. Name your project
   <***LastNameStudent***>**Assignment3.**
   N.B.: Use this naming and nothing else

3. Deadline: see Brightspace

Development strategy

Note that each entity in your system; a building, an agent, a client is a distinct process (a java program). That runs independent of the other entities. All the processes can run on one machine, or they can be distributed over multiple machines. The processes communicate with each other by sending and receiving messages via RabbitMQ.

To help you, we will give you some tips on how to approach developing the system.

1. Make a design first
2. Start development with only one rental agent (one process). Keep it simple, start implementing the messages they send to each other:
    - To request a list of buildings
    - To make a reservation
    - To confirm a reservation
    - To cancel a reservation
3. If all this works, you can change the implementation so that there are multiple rental agents in the system and that the application and of clients are distributed among these rental agents.
   This adds a lot of complexity to the design. For example, how do you know about the existence of all buildings, how do you get a unique reservation number?