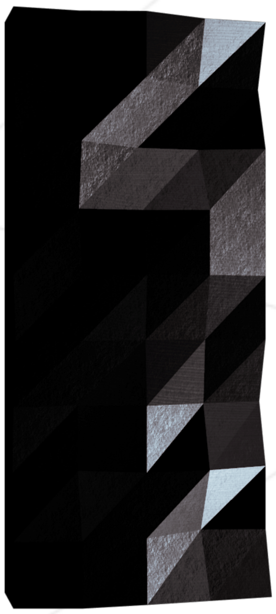


MessageQueue

RabbitMQ

Summary: we will learn how to work with a large flow of tasks using RabbitMQ message broker, a popular implementation of MQ pattern

Chapter I

Preamble

In 1876, Alexander Bell registered a patent for a telephone. Later on he established Bell Telephone Company.

At the same time, Lars Magnus Ericsson, who had been a mine worker for a long time, opened a workshop to repair telegraph equipment. After having repaired a huge number of telephones, the workshop developed its own device—a desk phone with a horn.

Bell Telephone Company and LM Ericsson & Co were main competitors in the field of telephone manufacturing.

What are these companies famous for now?

In 1885, Bell Telephone Company registered its subsidiary AT&T. UNIX operating system and C programming language were developed on the basis of this company.

Ericsson (Telefonaktiebolaget L. M. Ericsson) and Sony released a line of very popular mobile phones of the 2000s (my favorite Sony Ericsson K750i was one of them). In addition, Ericsson developed their own programming language Erlang that was used to create the RabbitMQ message broker :)

Chapter II

General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Now there is only one Java version for you, 1.8. Make sure that compiler and interpreter of this version are installed on your machine.
- You can use IDE to write and debug the source code.
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each task, make sure you follow the generally accepted Oracle standards :
- Comments are not allowed in the source code of your solution. They make it difficult to read the code.
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output.

- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter III

Exercise 00 - DocumentsGenerator

	Exercise 00
DocumentsGenerator	
Turn-in directory : *ex*00/	
Files to turn in : `DocumentsGenerator - folder`	
Allowed functions : `n/a`	

A training center needs help to implement a mechanism for generating a large number of various template documents. Each such document is created on the basis of data provided by thousands of students.

In this case, a student enters some personal data, e.g. name, phone number, course, place of residence, etc.

Information is entered on the **StudentsDataProducer** page of web application, which is a producer for several exchanges of RabbitMQ broker.

At startup, **StudentsDataProducer** shall create the required exchanges, queues, and bindings keys in the deployed local RabbitMQ cluster.

Let's define a set of tasks that should be solved using RabbitMQ:

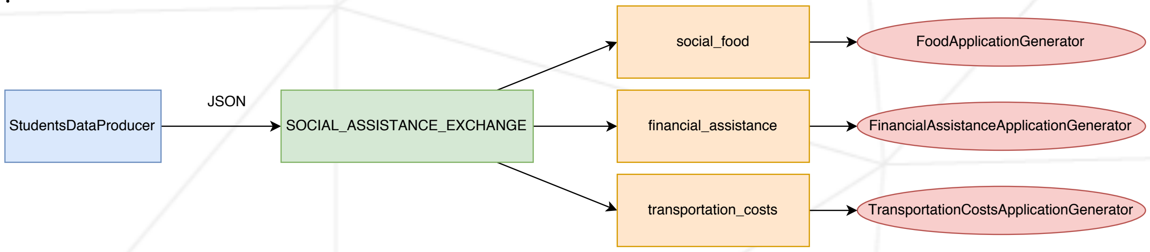
- Generation of documents for receiving social relief

After users have entered their data, all information in JSON format is transferred to **SOCIAL_ASSISTANCE_EXCHANGE** of **FANOUT** type. Three queue types are linked to **SOCIAL_ASSISTANCE_EXCHANGE: social_food, financial_assistance, transportation_costs.** Messages from each of the queues are processed by three types of consumers, respectively:

- FoodApplicationGenerator,
- FinancialAssistanceApplicationGenerator,
- TransportationCostsApplicationGenerator.

Thus, a message that falls into the **SOCIAL_ASSISTANCE_EXCHANGE** is passed to all queues. At the same time, each of the consumers generates a corresponding document.

The general workflow of **SOCIAL_ASSISTANCE_EXCHANGE** is shown below:


StudentsDataProducer
JSON
SOCIAL_ASSISTANCE_EXCHANGE
social_food
financial_assistance
transportation_costs
FoodApplicationGenerator
FinancialAssistanceApplicationGenerator
TransportationCostsApplicationGenerator

- Generation of documents for obtaining an educational grant

After user has entered their data, all information in JSON format is transferred to **GRANT_EXCHANGE** of **TOPIC** type.

There is a set of applications that use the received user data to generate a few different types of documents:

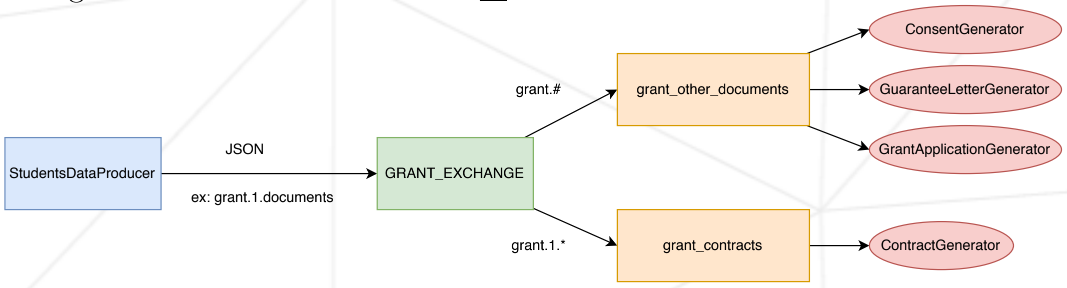
- Consent to personal data processing: **ConsentGenerator.**
- Letter of guarantee: **GuaranteeLetterGenerator.**
- Grant application: **GrantApplicationGenerator.**

All three generators are consumers of **grant_other_documents queue.** This queue is linked to **GRANT_EXCHANGE** via **binding key** = **grant.#**

For freshmen, you also need to generate grant contracts. This task shall be performed by **ContractGenerator.** This generator is a consumer of **grant_contracts** linked to **GRANT_EXCHANGE** via **binding key** = **grant.1.***

Thus, the message with student information from **StudentsDataProducer** is sent to **GRANT_EXCHANGE** with **routing key** = **grant.course number.documents**

The general workflow of **GRANT_EXCHANGE** is shown below:



Additional notes and requirements:

- When a student fills out a form in **StudentsDataProducer**, JSON information should be sent to **SOCIAL_ASSISTANCE_EXCHANGE** and **GRANT_EXCHANGE** simultaneously.
- All documents (pdf files) generated by each consumer shall contain information about the student received from the producer.
- Consumers shall save each such document in a shared folder.

- Document names shall allow to distinguish between the statements, whereas the form of a document of each type remains at a developer's discretion.
- It is recommended to implement the producer as a SpringBoot-based web application.
- Consumers are Java console applications that connect to a queue.
- You shall also attach a README file with instructions for deploying the entire system.
- Each producer and consumer is a separate project.