UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAŞI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DOMENIUL: Calculatoare şi tehnologia informaţiei
SPECIALIZAREA: Tehnologia informaţiei

Sisteme Distribuite -

Laborator 5

Servicii cu Spring Boot și Kotlin

Iaşi, 2022

Laborator 5 - Servicii cu Spring Boot și Kotlin

Introducere

Atenție! Cei care lucrează de pe stațiile din laborator, săriți direct la Secțiunea "Crearea proiectului" (pagina 3).

GUI cu Tkinter

Pentru implementarea unei interfețe grafice utilizând Tkinter, vezi cursul 5 de la disciplina Paradigme de Programare. De asemenea, vezi:

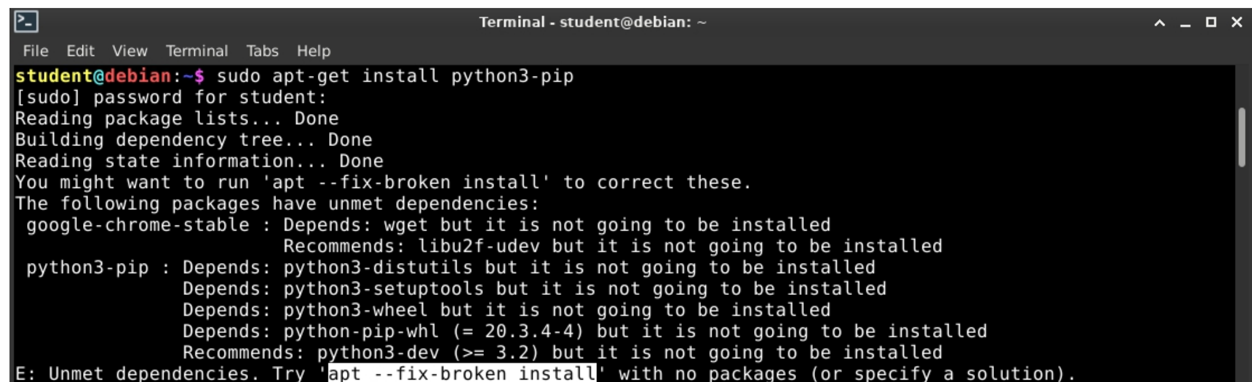
- https://tkdocs.com/tutorial/index.html
- https://docs.python.org/3/library/tk.html
- https://pythonbasics.org/tkinter/.

Pentru o soluție de tip Drag-and-Drop (Tkinter), se va deschide un terminal și se vor executa următoarele comenzi:

```
sudo apt install python3-pip # pip package manager for python packages
```

În cazul în care nu aveți anumite dependențe instalate la rularea comenzilor anterioare, rulați și comanda:

```
sudo apt --fix-broken install
```



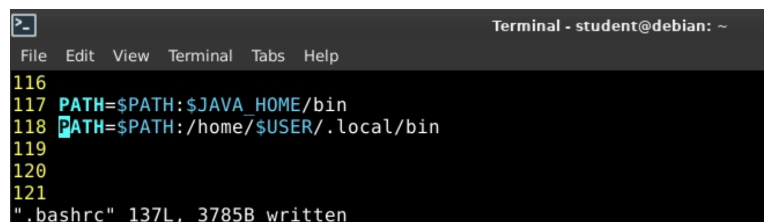

```
sudo pip3 install pygubu # GUI designer for Tkinter
pygubu-designer # run the designer
```

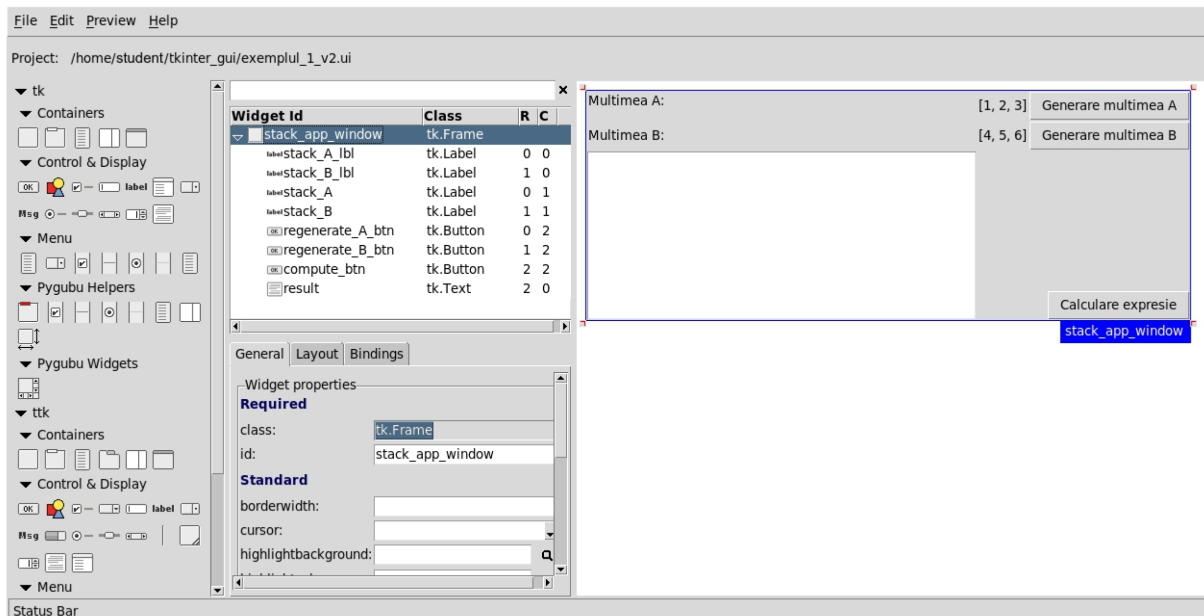
În cazul în care obțineți eroarea "command not found", atunci cel mai probabil executabilul acestui program a fost pus în directorul "~/.local/bin". Pentru a include directorul respectiv în variabila PATH, deschideți *fișierul ~/.bashrc* și adăugați la final:

```
PATH=$PATH:/home/$USER/.local/bin
```

Pentru actualizarea fișierului:

```
. ~/.bashrc
```





Designer-ul grafic PyGubu (Tkinter)

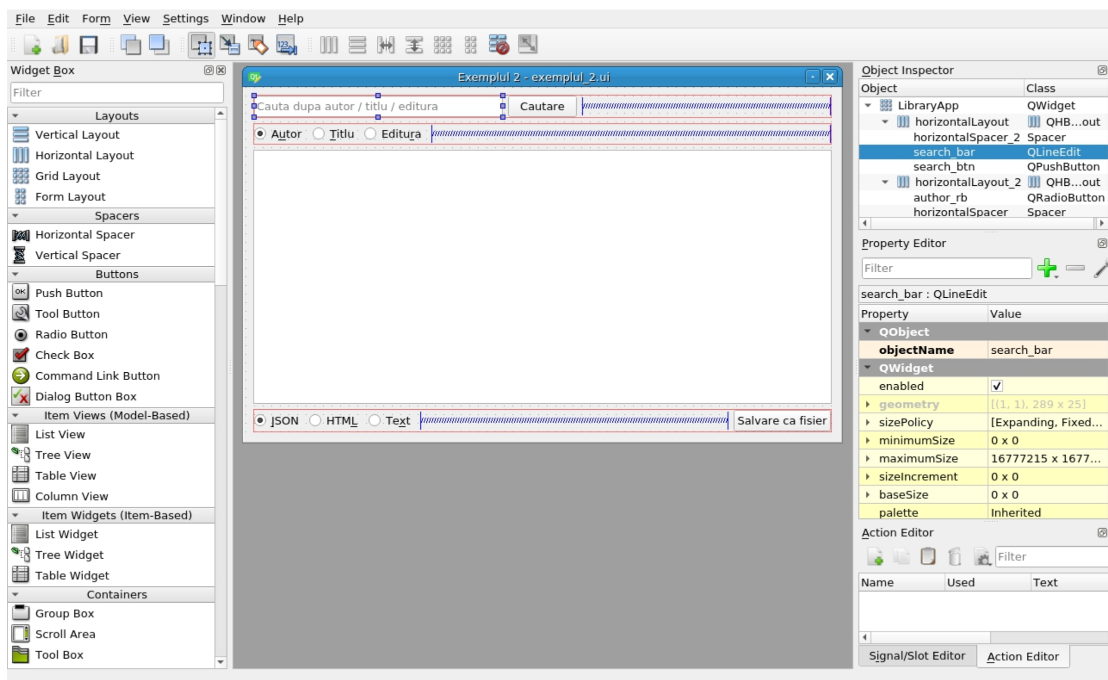
GUI cu PyQt5

Pentru implementarea unei interfețe grafice utilizând PyQt5, vezi:

- https://www.riverbankcomputing.com/static/Docs/PyQt5/
- https://pythonspot.com/pyqt5/
- https://likegeeks.com/pyqt5-tutorial/

Pentru un designer grafic (PyQt5), se vor executa în terminal comenzile:

```
wget http://download.qt.io/official_releases/online_installers/qt-unified-linux-x64-online.run
chmod a+x qt-unified-linux-x64-online.run
./qt-unified-linux-x64-online.run
```



Designer-ul grafic Qt5 Designer (PyQt5)

Crearea proiectului

Pentru crearea unui proiect **Spring Boot** folosind Maven / Gradle, vezi laboratorul 3 de la disciplina Sisteme Distribuite (Capitolul 1. Crearea unui proiect Spring Boot, subpunctele 1-10) .

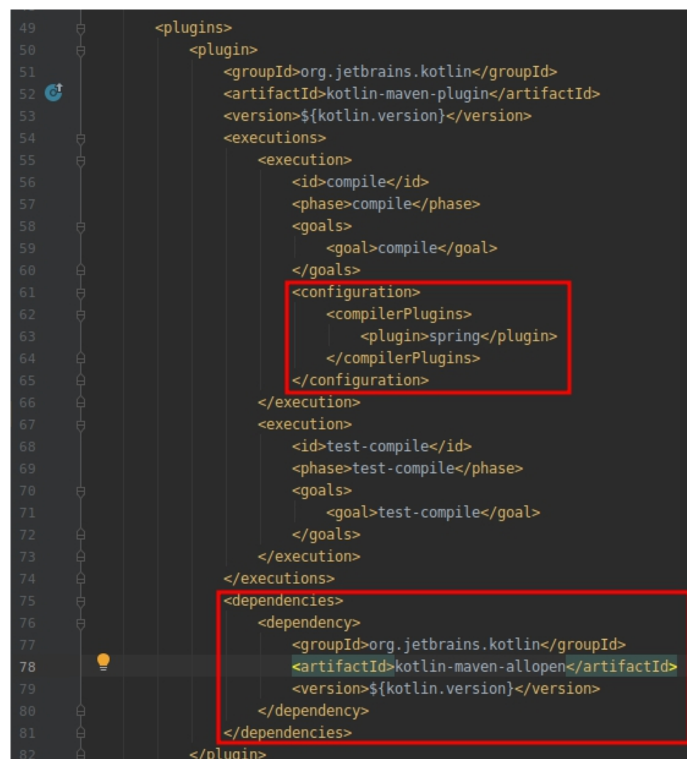
După ce s-a creat proiectul şi s-au adăugat plugin-urile specificate în laboratorul 3, adăugaţi următoarea dependenţă suplimentară (element subordonat al tag-ului **<dependencies>**):

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Clasele în Kotlin sunt, în mod implicit, marcate ca şi **final**, deci nu se pot moşteni decât dacă dezvoltatorul le marchează explicit ca **open** (de exemplu, **open** class MyClass ...). Spring necesită ca acele clase ce vor primi anumite tipuri de adnotări (cum ar fi **@Component**) să fie moştenibile, adică marcate cu **open**. Acest lucru este făcut automat de plugin-ul **kotlin-maven-allopen**, aşadar îl veţi adăuga ca dependenţă la compilare, astfel:

Adăugaţi următorul element de configurare în interiorul tag-ului **<plugin>**, corespunzător plugin-ului **kotlin-maven-plugin** (consultaţi figura de mai jos pentru locaţia exactă):

```
<dependencies>
    <dependency>
        <groupId>org.jetbrains.kotlin</groupId>
        <artifactId>kotlin-maven-allopen</artifactId>
        <version>${kotlin.version}</version>
    </dependency>
</dependencies>
```



```
49  <plugins>
50      <plugin>
51          <groupId>org.jetbrains.kotlin</groupId>
52          <artifactId>kotlin-maven-plugin</artifactId>
53          <version>${kotlin.version}</version>
54          <executions>
55              <execution>
56                  <id>compile</id>
57                  <phase>compile</phase>
58                  <goals>
59                      <goal>compile</goal>
60                  </goals>
61                  <configuration>
62                      <compilerPlugins>
63                          <plugin>spring</plugin>
64                      </compilerPlugins>
65                  </configuration>
66              </execution>
67              <execution>
68                  <id>test-compile</id>
69                  <phase>test-compile</phase>
70                  <goals>
71                      <goal>test-compile</goal>
72                  </goals>
73              </execution>
74          </executions>
75          <dependencies>
76              <dependency>
77                  <groupId>org.jetbrains.kotlin</groupId>
78                  <artifactId>kotlin-maven-allopen</artifactId>
79                  <version>${kotlin.version}</version>
80              </dependency>
81          </dependencies>
82      </plugin>
```

Apoi, se adaugă plugin-ul **spring** ca şi dependenţă la faza de compilare, cu următorul copil al tag-ului **<execution>**, pus în locația indicată în figura anterioară.

```
<configuration>
```

```
    <compilerPlugins>
        <plugin>spring</plugin>
    </compilerPlugins>
</configuration>
```

Principii SOLID

S.O.L.I.D. este un acronim pentru cinci principii de proiectare orientate pe obiecte.

- **Single-Responsibility Principle (SRP)** - o clasă ar trebui să aibă un singur motiv de schimbare, ceea ce înseamnă că o clasă ar trebui să aibă o singură sarcină (job)
- **Open-Closed Principle (OCP)** - entitățile software (clase, module, funcții, etc) ar trebui să fie deschise pentru extindere, dar închise pentru modificări
- **Liskov Substitution Principle (LSP)** - Subtipurile (clasele derivate) trebuie să fie substituibile tipului de bază (clasei de bază)
- **Interface Segregation Principle (ISP)** - clienții nu ar trebui obligați să depindă de metode pe care nu le utilizează
- **Dependency Inversion Principle (DIP):**
  - modulele high-level nu ar trebui să depindă de modulele low-level. Ambele ar trebui să depindă de abstractizări;
  - abstractizările nu ar trebui să depindă de detalii. Detaliile ar trebui să depindă de abstractizări

Pentru mai multe detalii, vezi cartea ***„Agile software development: principles, patterns, and practices”*** (2003) scrisă de ***Robert C. Martin*** (cunoscut ca „Uncle Bob”).

Cozi de mesaje

O coadă de mesaje este utilizată pentru comunicarea între procese, sau între firele de execuție (thread-urile) aceluiași proces. Acestea oferă un protocol de comunicare asincron în care emițătorul și receptorul nu au nevoie să interacționeze în același timp (mesajele sunt reținute în coadă până când destinatarul le citește)

Avantajele utilizării cozilor de mesaje:

- redundanță - procesele trebuie să confirme citirea mesajului și faptul că acesta poate fi eliminat din coadă
- vârfuri de trafic (traffic spikes) - adăugarea în coadă previne aceste spike-uri, asigurând stocarea datelor în coadă și procesarea lor (chiar dacă va dura mai mult)
- mesaje asincrone
- îmbunătățirea scalabilității
- garantarea faptului că tranzacția se execută o dată
- monitorizarea elementelor din coadă

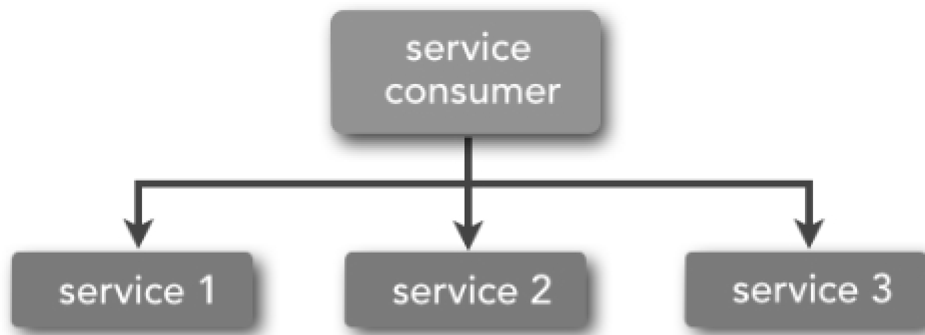
Servicii

Toate arhitecturile bazate pe servicii sunt în general arhitecturi distribuite, componentele fiind accesate *remote* printr-un anumit protocol (REST, SOAP, AMQP, JMS, RMI, etc).

Arhitecturile bazate pe servicii oferă îmbunătățiri față de aplicațiile monolitice, dar introduc de asemenea un nivel mai mare de complexitate (contractele serviciilor, disponibilitatea, securitatea, tranzacțiile).

Orchestrarea serviciilor (Service orchestration)

Orchestrarea serviciilor se referă la coordonarea mai multor servicii printr-un mediator centralizat, precum un consumator de servicii.



Orchestrarea serviciilor

Pentru a înțelege mai bine, vă puteți gândi la o orchestră. Un număr de muzicieni cântă la diferite instrumente la timpi diferiți, dar sunt cu toții coordonați de o persoană centrală - dirijorul. Similar, consumatorul de servicii coordonează toate serviciile necesare pentru completarea tranzacției de afaceri (business transaction)

Coregrafia serviciilor

Coregrafia serviciilor se referă la coordonarea mai multor servicii fără un mediator central. Un serviciu apelează un alt serviciu care poate apela mai departe un alt serviciu și tot așa, rezultând **înlănțuirea serviciilor** (service chaining).

Coregrafia serviciilor

Pentru o ilustrare descriptivă, vă puteți gândi la o companie de dansuri care interpretează pe scenă. Fiecare dansator se mișcă sincronizat cu ceilalți dansatori, dar nimeni nu le coordonează mișcările.

Pentru mai multe detalii, vezi cursurile de sisteme distribuite și cartea ***„Fundamentals of Software Architecture: A Comprehensive Guide to Patterns, Characteristics, and Best Practices“*** (2020), scrisă de **Neal Ford** și **Mark Richards**.

Configurări

Atenţie! Cei care lucrează de pe stațiile din laborator, săriți direct la Secțiunea “Exemple”.

Instalare server RabbitMQ:

```
sudo apt install -y rabbitmq-server
```

Verificare status RabbitMQ:

```
sudo systemctl status rabbitmq-server.service
```

Dacă serviciul nu este activ, se execută comanda:

```
sudo systemctl start rabbitmq-server.service
```

Activarea serviciului RabbitMQ la pornirea sistemului:

```
sudo systemctl enable rabbitmq-server
```

Activarea plugin-ului de gestionare:

```
sudo rabbitmq-plugins enable rabbitmq_management
```

Crearea și configurarea utilizatorului:

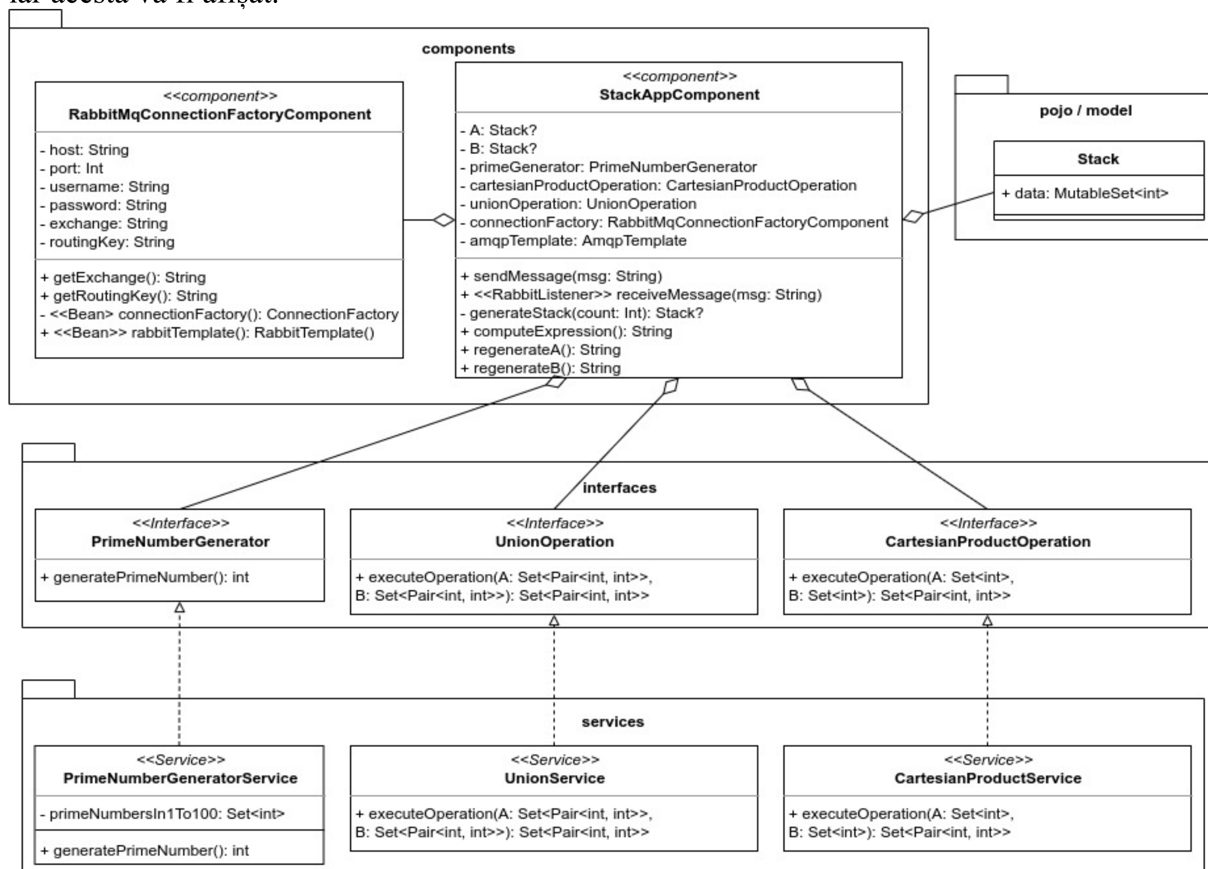
```
sudo rabbitmqctl add_user student student
sudo rabbitmqctl set_user_tags student administrator
sudo systemctl restart rabbitmq-server.service
sudo rabbitmqctl set_permissions -p / student ".*" ".*" ".*"
```

Pentru accesarea consolei de administrare, se deschide un browser web și se accesează http://localhost:15672/. Numele de utilizator: **student**, parola: **student**.

Exemple

Exemplul 1: StackApp

Cerință: Pornind de la două mulțimi A și B care conțin 20 de elemente prime aleator depuse în două colecții separate și ținând cont de $A \times B = \{(a,b) | a \in A \land b \in B\}$, să se scrie un program Kotlin care va calcula prin intermediul unor servicii expresia $(A \times B) \cup (B \times B)$ utilizând funcții specifice colecțiilor și principiile SOLID. Rezultatul este depus într-un dicționar, iar acesta va fi afișat.



Arhitectura aplicației StackApp

Comunicarea prin cozi de mesaje

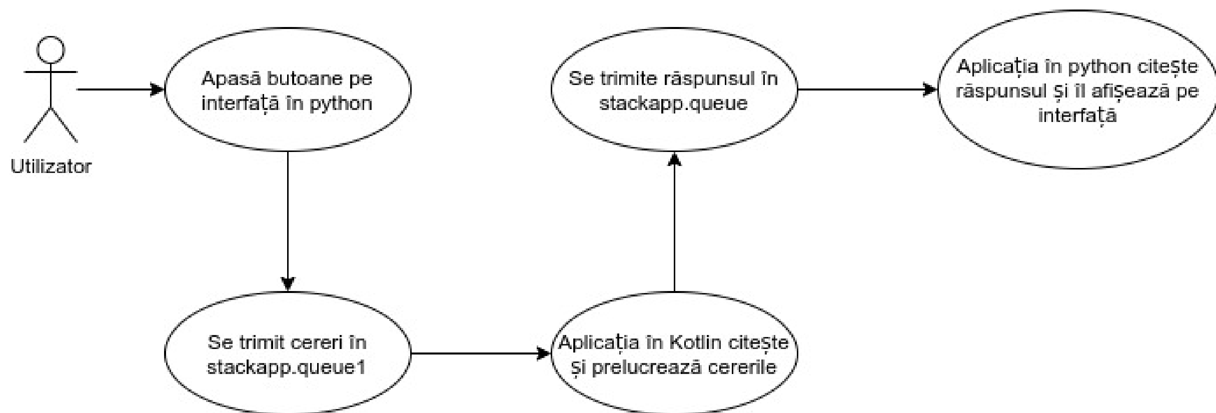


Diagrama de use-case pentru StackApp

În diagrama de mai sus, se observă existența a două cozi de mesaje (stackapp.queue și stackapp.queue1). Aceasta se datorează faptului că atât aplicația kotlin cât și aplicația python citesc/scriu într-o coadă de mesaje. Dacă ar fi fost utilizată doar o coadă, logica aplicației s-ar fi complicat (verificarea că mesajele au ajuns la „cine“ trebuie). Așadar, aplicația python (interfața) va scrie în *stackapp.queue1* fiecare apăsare de buton (ex.: regenerare mulțimea A, calculare expresie). Aplicația kotlin va citi din *stackapp.queue1*, va efectua acele operații și va scrie rezultatul acestora în *stackapp.queue*, de unde va fi preluat spre a fi afișat pe interfață.

Model

Pachetul **model** (sau **pojo**) conține o singură clasă, **Stack** ce reprezintă un șir de elemente unice (o mulțime). Aceasta va stoca elementele mulțimilor A și B.

Observație: existența pachetului **model** și a clasei Stack nu este necesară în acest caz particular, având în vedere faptul că acea clasă conține o singură variabilă de tip MutableSet. Variabilele **A** și **B** din StackAppComponent puteau fi de tip MutableSet<Int>. Exemplul este pur academic.

Services

Au fost create trei servicii:

- *PrimeNumberGeneratorService* – acesta conține o variabilă cu toate numerele prime din intervalul [1, 100] și o funcție care alege aleator un număr din acest set.
- *UnionService* – realizează prin intermediul unei funcții specifice seturilor reuniunea dintre două mulțimi, cu precizarea că elementele mulțimii sunt de fapt tuple de două numere întregi (rezultatul produsului cartezian).
- *CartesianProductService* – realizează produsul cartezian dintre două mulțimi prin intermediul unor funcții lambda imbricate (câte una pentru fiecare mulțime)

Components

Se remarcă faptul că StackAppComponent se folosește de abstractizări (interfețe), nu de implementările propriu-zise (**D**ependency inversion principle). Având în vedere simplitatea exemplului, soluția propusă respectă și celelalte principii SOLID, dar acest aspect este vizibil în acest caz particular doar pentru ***S****ingle responsibility principle* și ***O****pen-closed principle.*

StackAppComponent folosește cele trei servicii „injectate” (a se vedea laboratorul 3 – dependency injection - @Autowired), expunând funcții de comunicare prin cozi de mesaje și funcții ce generează o mulțime de numere prime și calculează expresia din cerință.

RabbitMqConnectionFactoryComponent citește fișierul de configurări *application.properties*, încărcând valorile respective în proprietățile sale (atributele sale). Această componentă conține toate setările necesare conectării la coada de mesaje, expunând o metoda ***rabbitTemplate()*** ce returneaza un obiect capabil de trimiterea de mesaje.

View

Interfața grafică a fost realizată atât cu PyQt5 cât și cu Tkinter, fiind ușor de folosit.

```
Multimea A:  [47, 71, 97, 41, 23, 79, 59, 31, 83, 7, 5, 61, 67, 3, 13, 19, 73, 2, 43, 17]   Generare multimea A
Multimea B:  [5, 79, 73, 71, 67, 19, 53, 41, 89, 2, 23, 61, 83, 29, 31, 7, 13, 59, 97, 3]   Generare multimea B

A: [47, 71, 97, 41, 23, 79, 59, 31, 83, 7, 5, 61, 67, 3, 13, 19, 73, 2, 43, 17]
B: [5, 79, 73, 71, 67, 19, 53, 41, 89, 2, 23, 61, 83, 29, 31, 7, 13, 59, 97, 3]
result: [(47, 5), (47, 79), (47, 73), (47, 71), (47, 67), (47, 19), (47, 53), (47, 41), (47, 89), (47, 2), (47, 23), (47,
61), (47, 83), (47, 29), (47, 31), (47, 7), (47, 13), (47, 59), (47, 97), (47, 3), (71, 5), (71, 79), (71, 73), (71, 71),
(71, 67), (71, 19), (71, 53), (71, 41), (71, 89), (71, 2), (71, 23), (71, 61), (71, 83), (71, 29), (71, 31), (71, 7), (71,
13), (71, 59), (71, 97), (71, 3), (97, 5), (97, 79), (97, 73), (97, 71), (97, 67), (97, 19), (97, 53), (97, 41), (97, 89),
(97, 2), (97, 23), (97, 61), (97, 83), (97, 29), (97, 31), (97, 7), (97, 13), (97, 59), (97, 97), (97, 3), (41, 5), (41, 79),
(41, 73), (41, 71), (41, 67), (41, 19), (41, 53), (41, 41), (41, 89), (41, 2), (41, 23), (41, 61), (41, 83), (41, 29), (41,
31), (41, 7), (41, 13), (41, 59), (41, 97), (41, 3), (23, 5), (23, 79), (23, 73), (23, 71), (23, 67), (23, 19), (23, 53),
(23, 41), (23, 89), (23, 2), (23, 23), (23, 61), (23, 83), (23, 29), (23, 31), (23, 7), (23, 13), (23, 59), (23, 97), (23,
3), (79, 5), (79, 79), (79, 73), (79, 71), (79, 67), (79, 19), (79, 53), (79, 41), (79, 89), (79, 2), (79, 23), (79, 61),
(79, 83), (79, 29), (79, 31), (79, 7), (79, 13), (79, 59), (79, 97), (79, 3), (59, 5), (59, 79), (59, 73), (59, 71), (59,
67), (59, 19), (59, 53), (59, 41), (59, 89), (59, 2), (59, 23), (59, 61), (59, 83), (59, 29), (59, 31), (59, 7), (59, 13),
(59, 59), (59, 97), (59, 3), (31, 5), (31, 79), (31, 73), (31, 71), (31, 67), (31, 19), (31, 53), (31, 41), (31, 89), (31,

Calculare expresie
```

Interfață grafică realizată cu Qt Creator și PyQt5

Pentru a porni interfața, se compilează întâi proiectul în kotlin: se execută din fereastra Maven --> Plugins --> spring-boot --> spring-boot:run. Apoi, se deschide un terminal în folder-ul interfeței (*qt_gui* sau *tkinter_gui*) și se execută comanda:

```
python3 exemplul_1_v1.py # pentru interfata cu PyQt5
# sau
python3 exemplul_1_v2.py # pentru interfata cu Tkinter
```

Observație: dacă nu sunt instalate dependențele, se execută următoarele comenzi:

```
sudo apt install python3-venv # pentru medii de lucru virtuale
sudo apt install python3-pip # package manager pentru python3
# cd path/to/StackApp
python3 -m venv env # creare mediu de lucru virtual
source env/bin/activate # activare mediu de lucru virtual
pip3 install -r requirements.txt # install pyqt5 tkinter, pygubu, pika,
retry
```

```
Multimea A:     [71, 23, 29, 61, 19, 17, 47, 2, 89, 41, 13, 67, 5, 31, 3, 43, 53, 83, 7, 79]   Generare multimea A
Multimea B:     [5, 79, 73, 71, 67, 19, 53, 41, 89, 2, 23, 61, 83, 29, 31, 7, 13, 59, 97, 3]   Generare multimea B
A: [71, 23, 29, 61, 19, 17, 47, 2, 89, 41, 13, 67,
 5, 31, 3, 43, 53, 83, 7, 79]
B: [5, 79, 73, 71, 67, 19, 53, 41, 89, 2, 23, 61,
83, 29, 31, 7, 13, 59, 97, 3]
result: [(71, 5), (71, 79), (71, 73), (71, 71), (7
1, 67), (71, 19), (71, 53), (71, 41), (71, 89), (7
1, 2), (71, 23), (71, 61), (71, 83), (71, 29), (71
, 31), (71, 7), (71, 13), (71, 59), (71, 97), (71,
 3), (23, 5), (23, 79), (23, 73), (23, 71), (23, 6
7), (23, 19), (23, 53), (23, 41), (23, 89), (23, 2
                                                                                                Calculare expresie
```

Interfață grafică realizată cu PyGubu și Tkinter

Exemplul 1: Configurări

Se accesează ***localhost:15672***, se introduc credențialele, apoi se navighează pe tab-ul de „Exchanges". Se va configura următorul exchange:

Add a new exchange

Name: stackapp.direct *
Type: direct
Durability: Durable
Auto delete: ? No
Internal: ? No
Arguments: = String
Add **Alternate exchange** ?

Add exchange

Se navighează apoi pe tab-ul „Queues" și se creează două cozi de mesaje:

Add a new queue

Type: Classic
Name: stackapp.queue *
Durability: Durable
Auto delete: ? No
Arguments: = String
Add **Message TTL** ? | **Auto expire** ? | **Max length** ? | **Max length bytes** ? | **Overflow behaviour** ? **Dead letter exchange** ? | **Dead letter routing key** ? | **Single active consumer** ? | **Maximum priority** ? **Lazy mode** ? **Master locator** ?

Add queue

Prima coadă de mesaje pentru StackApp

Similar, se creează și a doua coadă de mesaje, denumită **stackapp.queue1**
Tot pe tab-ul de „Queues", în tabelul de mai jos, se da click întâi pe stackapp.queue:

Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
stackapp.queue	classic	D Args	idle	0	0	0			
stackapp.queue1	classic	D Args	idle	0	0	0			

+/-

Tabel cozi de mesaje create

Prima coadă de mesaje se configurează ca mai jos:

Bindings

From	Routing key	Arguments	
(Default exchange binding)			

⇓

This queue

Add binding to this queue

From exchange: stackapp.direct *

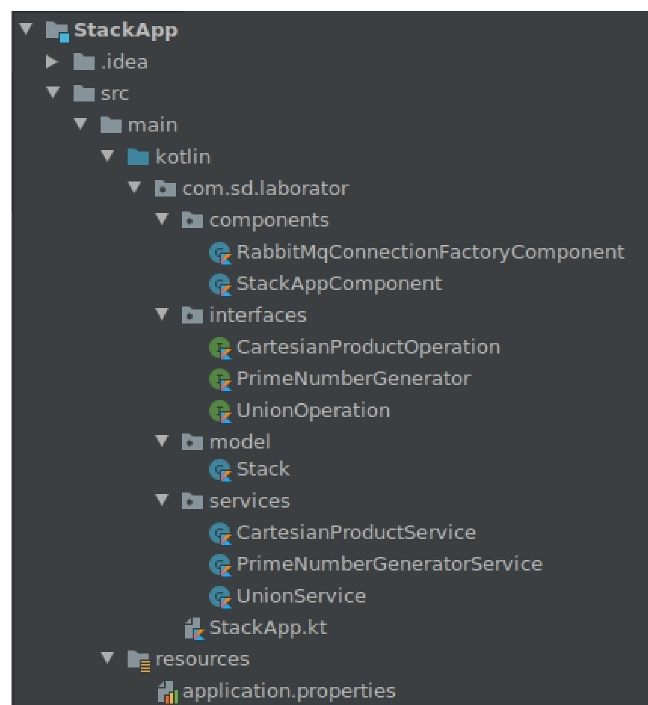
Routing key: stackapp.routingkey

Arguments: = String

Bind

Similar, se dă click pe a doua coadă de mesaje creată, iar la **Routing key:** stackapp.routingkey1

Structurarea proiectului



Ierarhia proiectului StackApp

Configurarea parametrilor pentru conexiunea cu RabbitMQ

Pentru a modifica cu ușurință setările ulterior, acestea vor fi încărcate dintr-un fișier de configurare ce va fi creat în pachetul resources (din folder-ul main). Se creează așadar fișierul *application.properties* cu următorul conținut:

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=student
spring.rabbitmq.password=student
```

```
stackapp.rabbitmq.queue=stackapp.queue1
stackapp.rabbitmq.exchange=stackapp.direct
stackapp.rabbitmq.routingkey=stackapp.routingkey
```

Exemplul 1: Codul sursă

Se recomandă ca întâi să accesați în browser consola de administrare a RabbitMQ (localhost:15672) și să stergeți cozile de mesaje create de colegii voștri, apoi să le refaceți (cu binding-uri ca în configurările de mai sus).

Aplicația python (interfața grafică)

În folder-ul cu interfața în python, se creează un fișier numit *requirements.txt*, cu următorul conținut:

```
tk==0.1.0
pika==1.1.0
retry==0.9.2
```

Se creează un mediu de lucru virtual și se instalează dependențele de mai sus, executând într-un terminal deschis în folder-ul interfeței comenzile:

```
python3 -m venv env
venvact # alias for: source env/bin/activate
pip3 install -r requirements.txt
```

Terminalul va rămâne deschis pentru a porni din mediul virtual interfața grafică.

```
# exemplul_1_v3.py
import os
import json
import tkinter as tk
from functools import partial
from mq_communication import RabbitMq


class StackApp:
    ROOT_DIR = os.path.dirname(os.path.abspath(__file__))
    A = None
    B = None

    def __init__(self, gui):
        self.gui = gui
        self.gui.title('Exemplul 1 cu Tkinter')

        self.gui.geometry("1050x300")

        self.stack_A_lbl = tk.Label(master=self.gui,
                                    text="Multimea A:")
        self.stack_B_lbl = tk.Label(master=self.gui,
                                    text="Multimea B:")
        self.stack_A = tk.Label(master=self.gui, text="[1, 2, 3]")
        self.stack_B = tk.Label(master=self.gui, text="[4, 5, 6]")

        self.regenerate_A_btn = tk.Button(master=self.gui,
                                          text="Generare multimea A",
                                    command=partial(self.send_request,
```

```
                                                request='regenerate_A'))
        self.regenerate_B_btn = tk.Button(master=self.gui,
                                              text="Generare multimea B",
                                       command=partial(self.send_request,
                                                request='regenerate_B'))
        self.compute_btn = tk.Button(master=self.gui,
                                         text="Calculare expresie",
                                        command=partial(self.send_request,
                                                request='compute'))

        self.result = tk.Text(self.gui, width=50, height=10)

        # alignment on the grid
        self.stack_A_lbl.grid(row=0, column=0)
        self.stack_B_lbl.grid(row=1, column=0)
        self.stack_A.grid(row=0, column=1)
        self.stack_B.grid(row=1, column=1)
        self.regenerate_A_btn.grid(row=0, column=2)
        self.regenerate_B_btn.grid(row=1, column=2)
        self.compute_btn.grid(row=2, column=2)
        self.result.grid(row=2, column=0)

        self.rabbit_mq = RabbitMq(self)
        self.gui.mainloop()

    def set_response(self, variable, response):
        if variable == 'A':
            self.regenerate_A(response)
        elif variable == 'B':
            self.regenerate_B(response)
        elif variable == 'compute':
            self.compute(response)

    def send_request(self, request):
        self.rabbit_mq.send_message(message=request)
        self.rabbit_mq.receive_message()

    def regenerate_A(self, response):
        self.A = response
        current_result = self.result.get("1.0", tk.END).split('\n')
        current_result[0] = 'A: ' + self.A
        self.stack_A['text'] = self.A
        self.result.delete("1.0", tk.END)
        self.result.insert(tk.END, '\n'.join(current_result))

    def regenerate_B(self, response):
        self.B = response
        current_result = self.result.get("1.0", tk.END).split('\n')
        if len(current_result) == 1:
            current_result.append('B: ' + self.B)
        else:
            current_result[1] = 'B: ' + self.B
        self.stack_B['text'] = self.B
        self.result.delete("1.0", tk.END)
        self.result.insert(tk.END, '\n'.join(current_result))

    def compute(self, response):
```

```
        dict_response = json.loads(response)
        result = ''
        for key in dict_response:
            result += '{}: {}\n'.format(key, dict_response[key])
        self.stack_A['text'] = dict_response['A']
        self.stack_B['text'] = dict_response['B']
        self.result.delete("1.0", tk.END)
        self.result.insert(tk.END, result)

if __name__ == '__main__':
    root = tk.Tk()
    app = StackApp(root)
    root.mainloop()
```

Se remarcă utilizarea funcției ***partial*** din modulul functools. În mod normal, un buton nu poate trimite un parametru la apelul functiei asignate evenimentului click. Totuși, utilizând funcția **partial**, poate fi trimis un parametru suplimentar cu o valoare prestabilită. Pentru mai multe detalii, vezi https://docs.python.org/3/library/functools.html

Se observă de asemenea partea de comunicare prin cozi de mesaje (funcțiile **send_request** și **set_response** - apelată din modulul *mq_communication*).

Modulul *mq_communication* realizează conexiunea propriu-zisă cu RabbitMQ.

```
# mq_communication.py
import pika
from retry import retry


class RabbitMq:
    config = {
        'host': '0.0.0.0',
        'port': 5678,
        'username': 'student',
        'password': 'student',
        'exchange': 'stackapp.direct',
        'routing_key': 'stackapp.routingkey1',
        'queue': 'stackapp.queue'
    }
    credentials = pika.PlainCredentials(config['username'],
                                        config['password'])
    parameters = (pika.ConnectionParameters(host=config['host']),
                  pika.ConnectionParameters(port=config['port']),
                  pika.ConnectionParameters(credentials=credentials))

    def __init__(self, ui):
        self.ui = ui

    def on_received_message(self, blocking_channel,
                            deliver, properties, message):
        result = message.decode('utf-8')
        blocking_channel.confirm_delivery()
        try:
            variable, response = result.split('~')
            self.ui.set_response(variable, response)
        except Exception as e:
            print(e)
            print("wrong data format")
```

```
        finally:
            blocking_channel.stop_consuming()


    @retry(pika.exceptions.AMQPConnectionError, delay=5, jitter=(1, 3))
    def receive_message(self):
        # automatically close the connection
        with pika.BlockingConnection(self.parameters) as connection:
            # automatically close the channel
            with connection.channel() as channel:
                channel.basic_consume(self.config['queue'],
                                      self.on_received_message)
                try:
                    channel.start_consuming()
                # Don't recover connections closed by server
                except pika.exceptions.ConnectionClosedByBroker:
                    print("Connection closed by broker.")
                # Don't recover on channel errors
                except pika.exceptions.AMQPChannelError:
                    print("AMQP Channel Error")
                # Don't recover from KeyboardInterrupt
                except KeyboardInterrupt:
                    print("Application closed.")

    def send_message(self, message):
        # automatically close the connection
        with pika.BlockingConnection(self.parameters) as connection:
            # automatically close the channel
            with connection.channel() as channel:
                self.clear_queue(channel)
                channel.basic_publish(
                            exchange=self.config['exchange'],
                            routing_key=self.config['routing_key'],
                            body=message)

    def clear_queue(self, channel):
        channel.queue_purge(self.config['queue'])
```

Se remarcă funcțiile **receive_message** și **send_messsage** ce citesc/scriu într-o coadă de mesaje.

De reținut: utilizarea unui bloc **with** automatizează închiderea variabilei deschise (spre exemplu: închiderea fișierului, a conexiunii, a canalului, etc), apelând la ieșirea din blocul **with** identat, funcția **close**.

Se remarcă utilizarea unui design pattern: **decoratorul.** Acesta este folosit prin adnotarea **@retry** ce reîncearcă apelarea funcției receive_message la apariția unei erori de tipul *AMQPConnectionError*. Pentru mai multe detalii, vezi https://pypi.org/project/retry/

Se observă de asemenea utilizarea unei funcții de callback - **on_received_message**, ce va fi apelată în momentul în care se citește un mesaj din coadă.

Pentru a porni interfața, se revine la terminalul cu mediul virtual pornit și se execută:

```
python3 exemplul_1_v3.py # modificati denumirea corespunzator
```

Aplicația kotlin (procesarea request-urilor)

Se creează întâi fișierul *src/main/kotlin/com.sd.laborator/StackApp.kt*

```
package com.sd.laborator

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class StackApp

fun main(args: Array<String>) {
    runApplication<StackApp>(*args)
}
```

Pachetul model

Se creează fișierul Stack.kt:

```
package com.sd.laborator.model

data class Stack(var data: MutableSet<Int>)
```

Pachetul interfaces

- CartesianProductOperation.kt

```
package com.sd.laborator.interfaces

interface CartesianProductOperation {
    fun executeOperation(A: Set<Int>, B: Set<Int>): Set<Pair<Int,
Int>>
}
```

- PrimeNumberGenerator.kt

```
package com.sd.laborator.interfaces

interface PrimeNumberGenerator {
    fun generatePrimeNumber(): Int
}
```

- UnionOperation.kt

```
package com.sd.laborator.interfaces

interface UnionOperation {
    fun executeOperation(A: Set<Pair<Int, Int>>, B: Set<Pair<Int,
Int>>): Set<Pair<Int, Int>>
}
```

Pachetul services

- CartesianProductService.kt

```
package com.sd.laborator.services

import com.sd.laborator.interfaces.CartesianProductOperation
import org.springframework.stereotype.Service

@Service
```

```
class CartesianProductService: CartesianProductOperation {
    override fun executeOperation(A: Set<Int>, B: Set<Int>):
Set<Pair<Int, Int>> {
        var result: MutableSet<Pair<Int, Int>> = mutableSetOf()
        A.forEach { a -> B.forEach { b -> result.add(Pair(a, b)) } }
        return result.toSet()
    }
}
```

- PrimeNumberGeneratorService.kt

```
package com.sd.laborator.services

import com.sd.laborator.interfaces.PrimeNumberGenerator
import org.springframework.stereotype.Service

@Service
class PrimeNumberGeneratorService: PrimeNumberGenerator {
    private val primeNumbersIn1To100: Set<Int> = setOf(2, 3, 5, 7, 11,
13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97)

    override fun generatePrimeNumber(): Int {
        return primeNumbersIn1To100.elementAt((0 until
primeNumbersIn1To100.count()).random())
    }

}
```

- UnionService.kt

```
package com.sd.laborator.services

import com.sd.laborator.interfaces.CartesianProductOperation
import com.sd.laborator.interfaces.UnionOperation
import org.springframework.stereotype.Service

@Service
class UnionService: UnionOperation {
    override fun executeOperation(A: Set<Pair<Int, Int>>, B:
Set<Pair<Int, Int>>): Set<Pair<Int, Int>> {
        return A union B
    }

}
```

Pachetul components

- StackAppComponent.kt

```
package com.sd.laborator.components

import com.sd.laborator.interfaces.CartesianProductOperation
import com.sd.laborator.interfaces.PrimeNumberGenerator
import com.sd.laborator.interfaces.UnionOperation
import com.sd.laborator.model.Stack
import org.springframework.amqp.core.AmqpTemplate
```

```
import org.springframework.amqp.rabbit.annotation.RabbitListener
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Component

@Component
class StackAppComponent {
    private var A: Stack? = null
    private var B: Stack? = null

    @Autowired
    private lateinit var primeGenerator: PrimeNumberGenerator
    @Autowired
    private lateinit var cartesianProductOperation:
CartesianProductOperation
    @Autowired
    private lateinit var unionOperation: UnionOperation
    @Autowired
    private lateinit var connectionFactory:
RabbitMqConnectionFactoryComponent

    private lateinit var amqpTemplate: AmqpTemplate

    @Autowired
    fun initTemplate() {
        this.amqpTemplate = connectionFactory.rabbitTemplate()
    }

    @RabbitListener(queues = ["\${stackapp.rabbitmq.queue}"])
    fun recieveMessage(msg: String) {
        // the result: 114,101,103,101,110,101,114,97,116,101,95,65 --
> needs processing
        val processed_msg = (msg.split(",").map
{ it.toInt().toChar() }).joinToString(separator="")
        var result: String? = when(processed_msg) {
            "compute" -> computeExpression()
            "regenerate_A" -> regenerateA()
            "regenerate_B" -> regenerateB()
            else -> null
        }
        println("result: ")
        println(result)
        if (result != null) sendMessage(result)
    }

    fun sendMessage(msg: String) {
        println("message: ")
        println(msg)
        this.amqpTemplate.convertAndSend(connectionFactory.getExchange
(),
                                         connectionFactory.getRoutingK
ey(),
                                         msg)
    }

    private fun generateStack(count: Int): Stack? {
        if (count < 1)
            return null
```

```
        var X: MutableSet<Int> = mutableSetOf()
        while (X.count() < count)
            X.add(primeGenerator.generatePrimeNumber())
        return Stack(X)
    }

    private fun computeExpression(): String {
        if (A == null)
            A = generateStack(20)
        if (B == null)
            B = generateStack(20)
        if (A!!.data.count() == B!!.data.count()) {
            // (A x B) U (B x B)
            val partialResult1 =
cartesianProductOperation.executeOperation(A!!.data, B!!.data)
            val partialResult2 =
cartesianProductOperation.executeOperation(B!!.data, B!!.data)
            val result =
unionOperation.executeOperation(partialResult1, partialResult2)
            return "compute~" + "{\"A\": \"" + A?.data.toString() +"\",
\"B\": \"" + B?.data.toString() + "\", \"result\": \"" +
result.toString() + "\"}"
        }
        return "compute~" + "Error: A.count() != B.count()"
    }

    private fun regenerateA(): String {
        A = generateStack(20)
        return "A~" + A?.data.toString()
    }

    private fun regenerateB(): String {
        B = generateStack(20)
        return "B~" + B?.data.toString()
    }
}
```

Se remarcă adnotarea clasei cu **@Component** pentru a putea fi descoperită la pornirea aplicației cu spring. De asemenea, se observă că variabilele private sunt declarate ca **lateinit**, deoarece vor fi injectate de spring (sunt adnotate cu **@Autowired**). Pentru variabila **amqpTemplate**, a fost creată o metodă care să injecteze valoarea (un RabbitTemplate), aceasta fiind adnotată cu @Autowired în locul variabilei propriu-zise.

Se poate vedea faptul că ***listener-ul*** *(*funcția care citește din coadă) este adnotată cu *@RabbitListener(queues = ["\${stackapp.rabbitmq.queue}"])*, primind ca parametru coada de mesaje din care citește. Totodată, se observă că funcția primește direct parametrul **msg** de tip String, ce reprezintă mesajul citit din coadă.

Pentru metoda de trimitere a unui mesaj, este nevoie de numele **exchange-ului** și de **routing key**. **Atenție la cheia de rutare! aceasta selectează practic coada destinație (exchange-ul fiind acelasi pentru ambele cozi).**

StackAppComponent este mediatorul (vezi orchestrarea serviciilor). Aici se utilizează toate serviciile cu scopul de a realiza funcționalitatea dorită (calcularea expresiei în cazul de față).

- RabbitMqConnectionFactoryComponent.kt

```
package com.sd.laborator.components
```

```
import
org.springframework.amqp.rabbit.connection.CachingConnectionFactory
import org.springframework.amqp.rabbit.connection.ConnectionFactory
import org.springframework.amqp.rabbit.core.RabbitTemplate
import org.springframework.beans.factory.annotation.Value
import org.springframework.context.annotation.Bean
import org.springframework.stereotype.Component

@Component
class RabbitMqConnectionFactoryComponent {
    @Value("\${spring.rabbitmq.host}")
    private lateinit var host: String
    @Value("\${spring.rabbitmq.port}")
    private val port: Int = 0
    @Value("\${spring.rabbitmq.username}")
    private lateinit var username: String
    @Value("\${spring.rabbitmq.password}")
    private lateinit var password: String
    @Value("\${stackapp.rabbitmq.exchange}")
    private lateinit var exchange: String
    @Value("\${stackapp.rabbitmq.routingkey}")
    private lateinit var routingKey: String

    fun getExchange(): String = this.exchange

    fun getRoutingKey(): String = this.routingKey

    @Bean
    private fun connectionFactory(): ConnectionFactory {
        val connectionFactory = CachingConnectionFactory()
        connectionFactory.host = this.host
        connectionFactory.username = this.username
        connectionFactory.setPassword(this.password)
        connectionFactory.port = this.port
        return connectionFactory
    }

    @Bean
    fun rabbitTemplate(): RabbitTemplate =
RabbitTemplate(connectionFactory())
}
```

Se remarcă adnotarea clasei cu **@Component** pentru a putea fi descoperită la pornirea aplicației cu spring. De asemenea, se observă adnotările **@Value** care inițializează variabilele adnotate cu valorile din fișierul *application.properties*. Pentru ca spring-ul să poată gestiona obiectele **ConnectionFactory** și respectiv **RabbitTemplate** create, funcțiile care creează aceste obiecte trebuie adnotate cu @Bean. Pentru mai multe detalii despre bean-uri, vezi;

- https://www.baeldung.com/spring-bean
- https://docs.spring.io/spring-javaconfig/docs/1.0.0.m3/reference/html/creating-bean-definitions.html

Pentru pornirea aplicației, din meniul **Maven**, se apasă clean --> compile --> spring-boot:run. Apoi dintr-un terminal cu mediul virtual pornit (și dependențele din requirements.txt instalate), se execută comanda **python3 exemplul_1_v1.py**

Exemplul 2: LibraryApp

Cerință: Să se scrie un program Kotlin care să realizeze gestiunea unei biblioteci prin intermediul unor servicii, utilizând principiile SOLID. Aplicația va conține trei moduri de afișare a datelor (HTML, JSON și Raw) și va expune utilizatorului prin interfață funcționalități de tip CRUD (Create, Retrieve, Update, Delete).

Arhitectura aplicației este reprezentată în diagrama de mai jos. Spre deosebire de exemplul anterior, se observă și ***I****nterface segregation principle* în cadrul LibraryPrinter. Deși LibraryPrinter înglobează toate cele trei tipuri de afișare, această funcționalitate poate fi ușor modificată implementând doar interfețele necesare unui client.

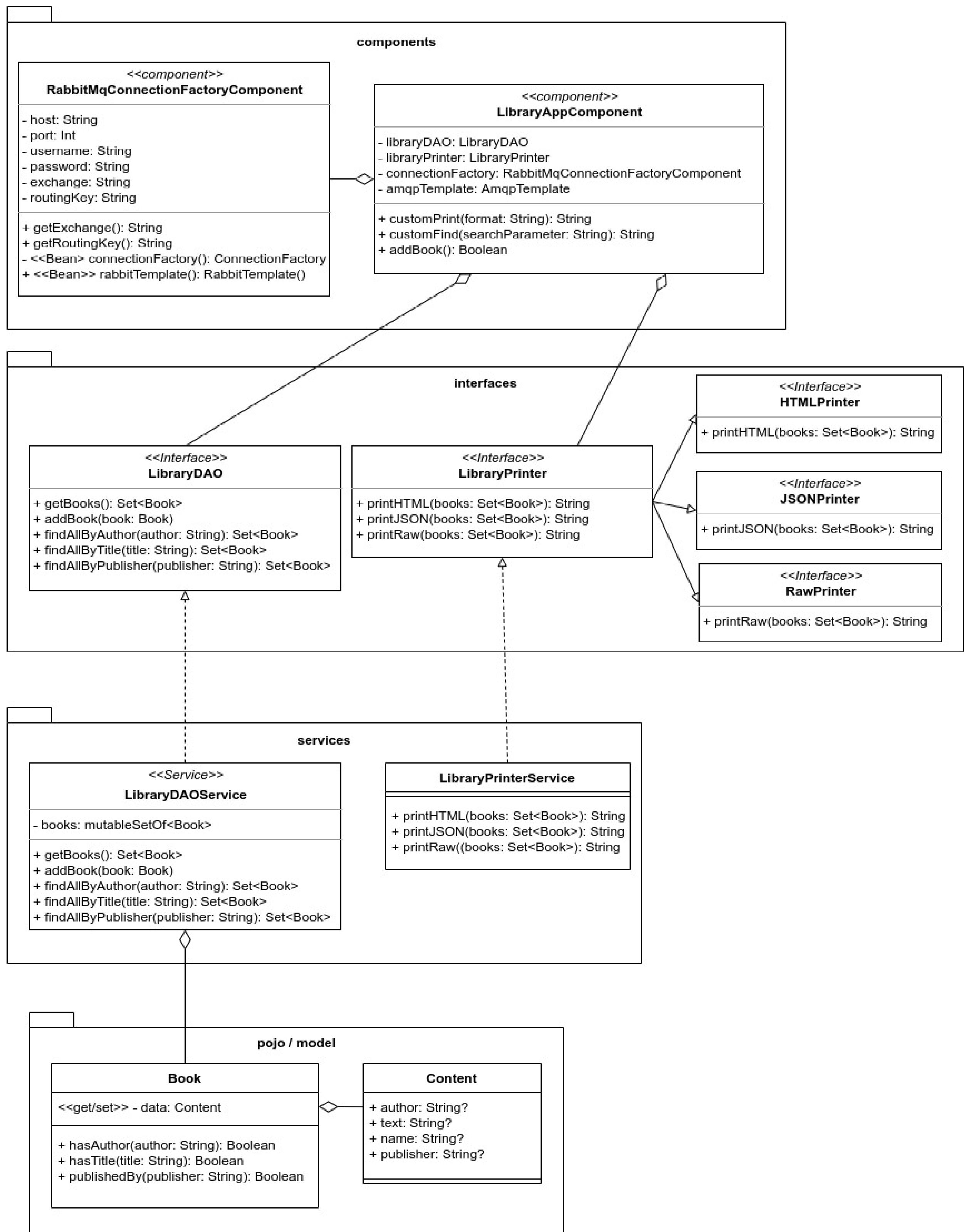
Similar cu exemplu anterior, compilați întâi proiectul în kotlin, executați din fereastra Maven --> Plugins --> spring-boot --> spring-boot:run. Apoi, deschideți un terminal în folder-ul interfeței (*qt_gui*) și executați comanda:

```
python3 exemplul_2.py # interfata cu PyQt5
```

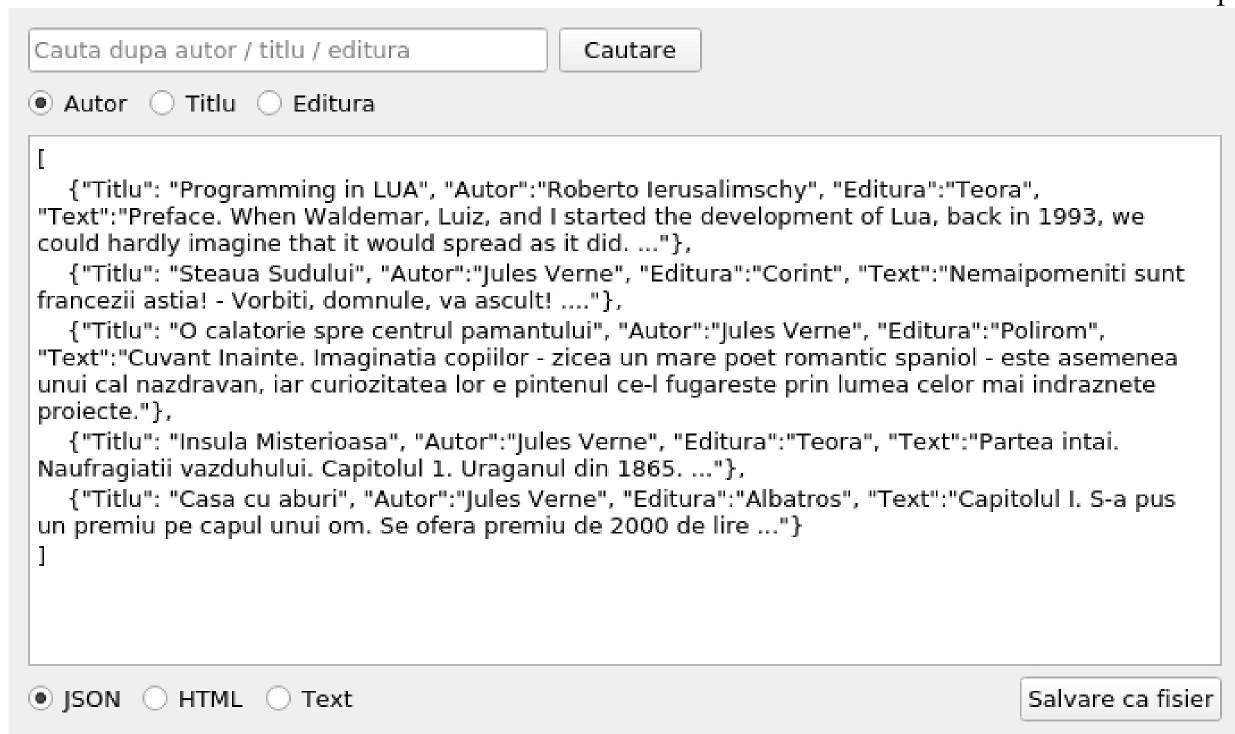
Soluția propusă are următoarele flow-uri:

- căutare fără introducere de cuvinte cheie -> indiferent de selecția căutării (autor / titlu / editura), programul va afișa toate cărțile în formatul specificat (JSON / HTML / Text)
- căutare cu introducere de cuvinte cheie -> programul va filtra lista de cărți în funcție de câmpul dorit (autor / titlu / editură), afișând rezultatul în formatul selectat
- căutare urmată de salvare fișier -> va salva conținutul găsit într-un fișier cu extensia .html, .json sau .txt (în funcție de selecție)

Observație: la realizarea unei căutări cu filtrare, rezultatul va fi afișat în format JSON indiferent de selecția curentă. Aceasta se datorează faptului că interfața din python nu transmite (momentan) tipul de fișier. Puteți modifica exemplul astfel încât să trimiteți încă un parametru (modul de printare) la căutarea cu filtrare.



Arhitectura aplicației LibraryApp



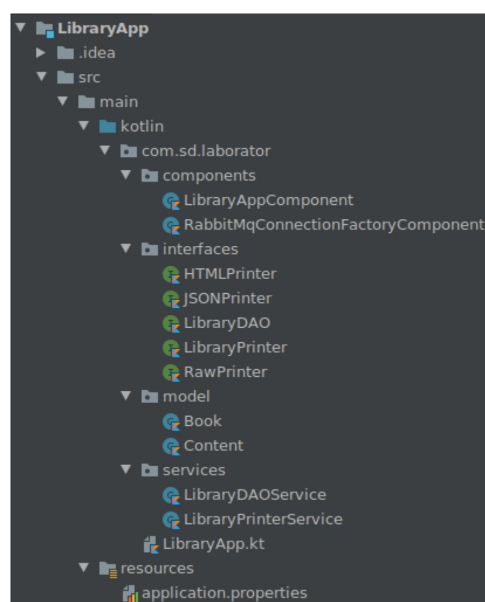
Interfața grafică pentru LibraryApp realizată cu PyQt5

Exemplul 2: Configurări

Analog cu exemplul 1, se creează:

- un exchange: **libraryapp.direct**
- două cozi
  - **libraryapp.queue**
  - **libraryapp.queue1**
- două binding-uri:
  - libraryapp.queue -> libraryapp.direct, **libraryapp.routingkey**
  - libraryapp.queue1 -> libraryapp.direct, **libraryapp.routingkey1**

Structura proiectului



Ierarhia proiectului LibraryApp

Configurarea parametrilor pentru conexiunea cu RabbitMQ

Se creează fișierul *src/main/resources/application.properties* cu următorul conținut:

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=student
spring.rabbitmq.password=student
libraryapp.rabbitmq.queue=libraryapp.queue1
libraryapp.rabbitmq.exchange=libraryapp.direct
libraryapp.rabbitmq.routingkey=libraryapp.routingkey
```

Configurarea proiectului

Se reiau pașii de la exemplul 1 (sunt aceleași dependențe și plugin-uri).

Exemplul 2: codul sursă

Se creează întâi în pachetul *com.sd.laborator* fișierul *LibraryApp.kt:*

```
package com.sd.laborator

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication


@SpringBootApplication
class LibraryApp

fun main(args: Array<String>) {
    runApplication<LibraryApp>(*args)
}
```

Pachetul model

- Book.kt

Se remarcă *getters* și *setters* care accesează atribute ale variabilei **data** de tip **Content**.

```
package com.sd.laborator.model

class Book(private var data: Content) {

    var name: String?
        get() {
            return data.name
        }
        set(value) {
            data.name = value
        }

    var author: String?
        get() {
            return data.author
        }
        set(value) {
            data.author = value
        }

    var publisher: String?
```

```
        get() {
            return data.publisher
        }
        set(value) {
            data.publisher = value
        }

    var content: String?
        get() {
            return data.text
        }
        set(value) {
            data.text = value
        }

    fun hasAuthor(author: String): Boolean {
        return data.author.equals(author)
    }

    fun hasTitle(title: String): Boolean {
        return data.name.equals(title)
    }

    fun publishedBy(publisher: String): Boolean {
        return data.publisher.equals(publisher)
    }

}
```

- Content.kt

```
package com.sd.laborator.model

data class Content(var author: String?, var text: String?, var name:
String?, var publisher: String?)
```

Pachetul interfaces

- HTMLPrinter.kt

```
package com.sd.laborator.interfaces

import com.sd.laborator.model.Book

interface HTMLPrinter {
    fun printHTML(books: Set<Book>): String
}
```

- JSONPrinter.kt

```
package com.sd.laborator.interfaces

import com.sd.laborator.model.Book

interface JSONPrinter {
    fun printJSON(books: Set<Book>): String
```

```
}
```

- LibraryDAO.kt

```
package com.sd.laborator.interfaces

import com.sd.laborator.model.Book

interface LibraryDAO {
    fun getBooks(): Set<Book>
    fun addBook(book: Book)
    fun findAllByAuthor(author: String): Set<Book>
    fun findAllByTitle(title: String): Set<Book>
    fun findAllByPublisher(publisher: String): Set<Book>
}
```

- LibraryPrinter.kt

```
package com.sd.laborator.interfaces

interface LibraryPrinter: HTMLPrinter, JSONPrinter, RawPrinter
```

- RawPrinter.kt

```
package com.sd.laborator.interfaces

import com.sd.laborator.model.Book

interface RawPrinter {
    fun printRaw(books: Set<Book>): String
}
```

Pachetul services

- LibraryDAOService.kt

Observație: Abrevierea DAO înseamnă Data Access Object.

```
package com.sd.laborator.services

import com.sd.laborator.interfaces.LibraryDAO
import com.sd.laborator.model.Book
import com.sd.laborator.model.Content
import org.springframework.stereotype.Service

@Service
class LibraryDAOService: LibraryDAO {
    private var books: MutableSet<Book> = mutableSetOf(
        Book(Content("Roberto Ierusalimschy","Preface. When Waldemar,
Luiz, and I started the development of Lua, back in 1993, we could
hardly imagine that it would spread as it did. ...","Programming in
LUA","Teora")),
        Book(Content("Jules Verne","Nemaipomeniti sunt francezii astia!
- Vorbiti, domnule, va ascult! ....","Steaua Sudului","Corint")),
```

```
        Book(Content("Jules Verne","Cuvant Inainte. Imaginatia
copiilor - zicea un mare poet romantic spaniol - este asemenea unui
cal nazdravan, iar curiozitatea lor e pintenul ce-l fugareste prin
lumea celor mai indraznete proiecte.","O calatorie spre centrul
pamantului","Polirom")),
        Book(Content("Jules Verne","Partea intai. Naufragiatii
vazduhului. Capitolul 1. Uraganul din 1865. ...","Insula
Misterioasa","Teora")),
        Book(Content("Jules Verne","Capitolul I. S-a pus un premiu pe
capul unui om. Se ofera premiu de 2000 de lire ...","Casa cu
aburi","Albatros"))
    )
    override fun getBooks(): Set<Book> {
        return this.books
    }

    override fun addBook(book: Book) {
        this.books.add(book)
    }

    override fun findAllByAuthor(author: String): Set<Book> {
        return (this.books.filter { it.hasAuthor(author) }).toSet()
    }

    override fun findAllByTitle(title: String): Set<Book> {
        return (this.books.filter { it.hasTitle(title) }).toSet()
    }

    override fun findAllByPublisher(publisher: String): Set<Book> {
        return (this.books.filter
{ it.publishedBy(publisher) }).toSet()
    }
}
```

- LibraryPrinterService.kt

```
package com.sd.laborator.services

import com.sd.laborator.interfaces.LibraryPrinter
import com.sd.laborator.model.Book
import org.springframework.stereotype.Service

@Service
class LibraryPrinterService: LibraryPrinter {
    override fun printHTML(books: Set<Book>): String {
        var content: String = "<html><head><title>Libraria mea
HTML</title></head><body>"
        books.forEach {
            content +=
"<p><h3>${it.name}</h3><h4>${it.author}</h4><h5>${it.publisher}</h5>${
it.content}</p><br/>"
        }
        content += "</body></html>"
        return content
    }

    override fun printJSON(books: Set<Book>): String {
```

```
        var content: String = "[\n"
        books.forEach {
            if (it != books.last())
                content += "    {\"Titlu\": \"${it.name}\", 
\"Autor\":\"${it.author}\", \"Editura\":\"${it.publisher}\", 
\"Text\":\"${it.content}\"},\n"
            else
                content += "    {\"Titlu\": \"${it.name}\", 
\"Autor\":\"${it.author}\", \"Editura\":\"${it.publisher}\", 
\"Text\":\"${it.content}\"}\n"
        }
        content += "]\n"
        return content
    }
    override fun printRaw(books: Set<Book>): String {
        var content: String = ""
        books.forEach {
            content += 
"${it.name}\n${it.author}\n${it.publisher}\n${it.content}\n\n"
        }
        return content
    }
}
```

Pachetul components

- RabbitMqConnectionFactoryComponent.kt

```
package com.sd.laborator.components

import 
org.springframework.amqp.rabbit.connection.CachingConnectionFactory
import org.springframework.amqp.rabbit.connection.ConnectionFactory
import org.springframework.amqp.rabbit.core.RabbitTemplate
import org.springframework.beans.factory.annotation.Value
import org.springframework.context.annotation.Bean
import org.springframework.stereotype.Component

@Component
class RabbitMqConnectionFactoryComponent {
    @Value("\${spring.rabbitmq.host}")
    private lateinit var host: String
    @Value("\${spring.rabbitmq.port}")
    private val port: Int = 0
    @Value("\${spring.rabbitmq.username}")
    private lateinit var username: String
    @Value("\${spring.rabbitmq.password}")
    private lateinit var password: String
    @Value("\${libraryapp.rabbitmq.exchange}")
    private lateinit var exchange: String
    @Value("\${libraryapp.rabbitmq.routingkey}")
    private lateinit var routingKey: String

    fun getExchange(): String = this.exchange

    fun getRoutingKey(): String = this.routingKey
```

```
    @Bean
    private fun connectionFactory(): ConnectionFactory {
        val connectionFactory = CachingConnectionFactory()
        connectionFactory.host = host
        connectionFactory.username = username
        connectionFactory.setPassword(password)
        connectionFactory.port = port
        return connectionFactory
    }

    @Bean
    fun rabbitTemplate(): RabbitTemplate =
RabbitTemplate(this.connectionFactory())

}
```

- LibraryAppComponent.kt

```
package com.sd.laborator.components

import com.sd.laborator.interfaces.LibraryDAO
import com.sd.laborator.interfaces.LibraryPrinter
import com.sd.laborator.model.Book
import org.springframework.amqp.core.AmqpTemplate
import org.springframework.amqp.rabbit.annotation.RabbitListener
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Component
import java.lang.Exception

@Component
class LibraryAppComponent {
    @Autowired
    private lateinit var libraryDAO: LibraryDAO

    @Autowired
    private lateinit var libraryPrinter: LibraryPrinter

    @Autowired
    private lateinit var connectionFactory:
RabbitMqConnectionFactoryComponent
    private lateinit var amqpTemplate: AmqpTemplate

    @Autowired
    fun initTemplate() {
        this.amqpTemplate = connectionFactory.rabbitTemplate()
    }

    fun sendMessage(msg: String) {
        this.amqpTemplate.convertAndSend(connectionFactory.getExchange
(),
                                         connectionFactory.getRoutingK
ey(),
                                         msg)
    }

    @RabbitListener(queues = ["\${libraryapp.rabbitmq.queue}"])
    fun recieveMessage(msg: String) {
```

```
        // the result needs processing
        val processedMsg = (msg.split(",").map
{ it.toInt().toChar() }).joinToString(separator="")
        try {
            val (function, parameter) = processedMsg.split(":")
            val result: String? = when(function) {
                "print" -> customPrint(parameter)
                "find" -> customFind(parameter)
                else -> null
            }
            if (result != null) sendMessage(result)
        } catch (e: Exception) {
            println(e)
        }
    }

    fun customPrint(format: String): String {
        return when(format) {
            "html" -> libraryPrinter.printHTML(libraryDAO.getBooks())
            "json" -> libraryPrinter.printJSON(libraryDAO.getBooks())
            "raw" -> libraryPrinter.printRaw(libraryDAO.getBooks())
            else -> "Not implemented"
        }
    }

    fun customFind(searchParameter: String): String {
        val (field, value) = searchParameter.split("=")
        return when(field) {
            "author" ->
this.libraryPrinter.printJSON(this.libraryDAO.findAllByAuthor(value))
            "title" ->
this.libraryPrinter.printJSON(this.libraryDAO.findAllByTitle(value))
            "publisher" ->
this.libraryPrinter.printJSON(this.libraryDAO.findAllByPublisher(value)
)
            else -> "Not a valid field"
        }
    }

    fun addBook(book: Book): Boolean {
        return try {
            this.libraryDAO.addBook(book)
            true
        } catch (e: Exception) {
            false
        }
    }
}
```

Interfața în python este realizată similar cu cea de la exemplul 1. Aceasta va fi preluată din codul sursă atașat laboratorului.

Aplicaţii şi teme

Aplicații de laborator:

- Respectând principiul lui **Liskov**, să se implementeze o alternativă a *LibraryPrinterService* din exemplul 2.
- Să se modifice exemplul 2 astfel încât fișierul salvat în urma unei căutări cu filtrare (după autor/titlu/editură) să fie salvat și în format HTML / text.
- La salvarea fişierului, să se modifice formatul în funcție de opțiunea aleasă (json / html / text)
- Să se îmbunătățească opțiunea de căutare prin detectarea unor potriviri parțiale (ex.: „Insula“ -> „Insula Misterioasă”), iar căutarea să fie case-insensitive.
- Respectând principiile **SOLID**, să se adauge opțiunea de a printa și în format XML.
- Să se modifice interfața ultimului exemplu: se va adăuga un buton care să deschidă o nouă fereastră cu un formular (autor, text, denumire, editură) pentru introducerea unei cărți în bibliotecă. După preluarea datelor de pe interfață, se va apela metoda *addBook* din *LibraryAppComponent*.

Teme pe acasă:

- Să se reimplementeze primul exemplu folosind înlănțuirea serviciilor (chaining) în loc de orchestrarea lor.
- Să se reimplementeze comunicarea prin cozi de mesaje din aplicația python (exemplul 2) utilizând un adaptor asincron (vezi https://pypi.org/project/pika/ pentru adaptoarele oferite de modulul *pika* și documentația acestuia: https://pika.readthedocs.io/en/stable/)
- Se considera ca avem cativa chelneri si cativa bucatari care fiecare dintre ei pot primi si deservi 5 comenzi diferite (numerotate de la 1 la 5 - continutul comenzii nu este relevant) care sunt modelati prin obiecte (ulterior vor deveni servicii de diverse tipuri si implementari - de ex pe biletele de examen). Fiecare are un identificator unic de tipul clasa si identificator (de ex Bucatar37x56AF sau Chelner72XX45E). Pornindu-se de la experienta din laborator sa se proiecteze și să se implementeze un sistem care gestioneaza interactiunea dintre cele doua echipe (de chelneri si de bucatori). Evident ca vor avea cozi asociate si timpi diferiti pentru livrarea fiecarui meniu.

bonus - Să se reimplementeze ultima problema, utilizând un alt framework (la alegere) pentru comunicarea prin cozi de mesaje.

-