**SD**

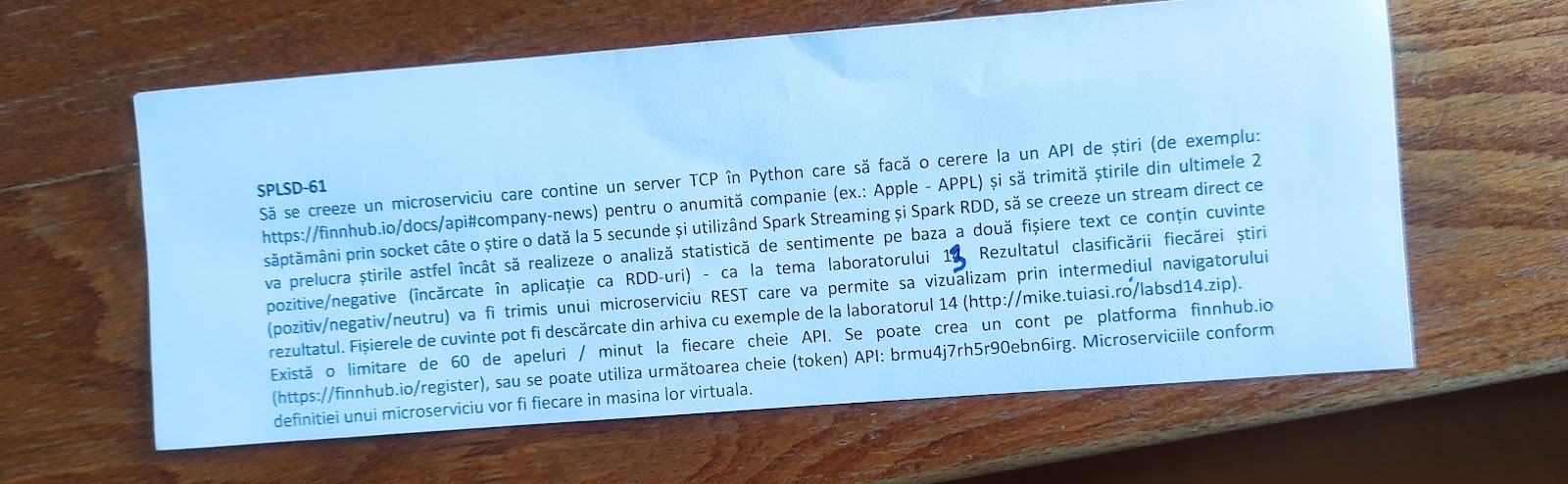
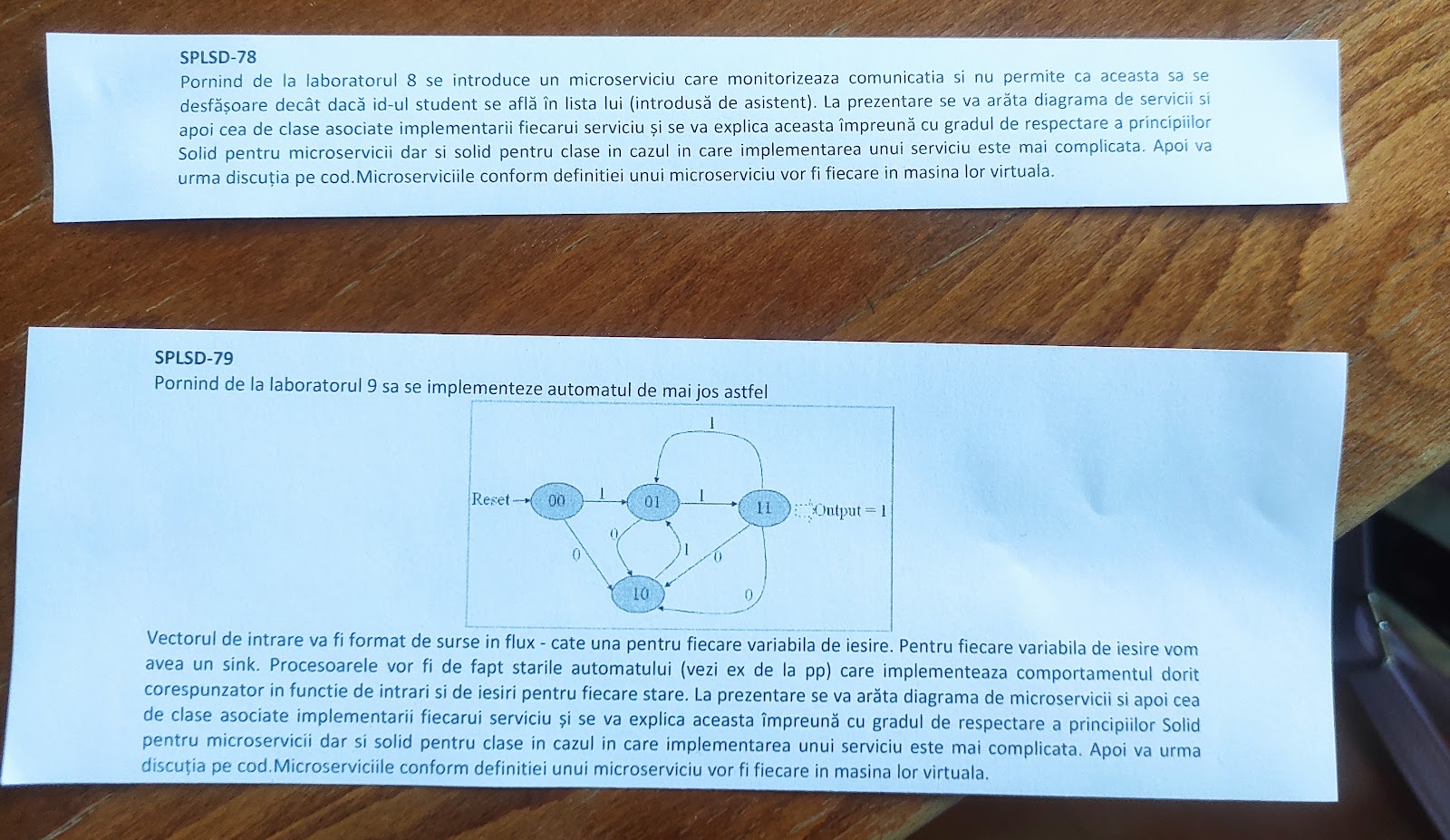
Deci am avut de făcut un microserviciu in kotlin care să să aibă un server tcp și care folosind serializare comunica cu un site de stock exchange și după folosind framework-u Apache spark sa fac un direct stream cu un program in python (comunica cu microserviciul) care trebuie să facă niște calcule pe datele primite (ex: calculare profit deschidere închidere și filtrare cele care au mai mic de 40%)și le afișa/scria într-un fișier.

Problema 62

Sa se creeze un microserviciu ce contine un server TCP in Kotlin care sa faca o cerere la un API de stiri. Pentru o anumita companie si sa trimita știrile din ultimele 2 saptamani prin socket si cate o stire o dată la 5 secunde si utilizând SPARK Streaming si SPARK RDD sa se creeze un stream direct ce va prelucra știrile astfel încât să realizeze o analiză statistica de sentimente pe baza a 2 fisiere text ce contin cuvinte pozitive/ nevative(incarcate in aplicatii ca RDD uri) ca la tema 13. Rezultatul clasificarii stirilor va fi trimis unui microserviciu WEB care va permite sa vizualizam prin intermediul navigatorului rezultatul. Fisierele de cuvinte pot fi desc. De la lab 14.

Exista o limita de 60 de apeluri pe minut la fiecare cheie API .

Se poate face un cont pe platforma finhub sau se poate utliza API : \* litere random aici\*

f

Sa modific l6 ex1 asa incat sa am cate un ms pentru fiecare operatie crud. Ele sa transmita mesajele la un alt ms (repository care are o baza de date) prin Kafka. El sa trimita raspunsuri, daca e cazul, tot prin kafka la fiecare ms crud si de la ms-urile crud inapoi la BeerAppCLI.

Pornind de la Okazii din laboratorul 7, am avut de implementat un microserviciu care monitoriza procesele din aplicația de licitatie. Se implementa cu Kafka, cozi de mesaje etc..si la final trebuiau puse mesajele intr-o baza de date SQL. Si diagramele alea de facut,  de explicat cum am gândit programul si dupa se trecea la discutia pe cod

Un microserviciu ce contine un server TCP in Kotlin care sa faca o cerere la un API si trebuia folosit Spark Streaming, Spark RDD si sockets

Problema nr 08.

Realizati o aplicatie de monitorizare a vremei (laboratorul 3), unde fiecare serviciu este incapsulat individual. Creati un serviciu care sa respecte modelul de replicare al unui serviciu, si sa replice serviciul principal(WeatheAppController-ul) de un anumit numar de ori. De asemenea sa se creeze un serviciu de vizualizare a elementelor citite care sa comunice cu o interfata grafica realizata in PyQt. Toate serviciile comunica prin tcp intre ele si fiecare serviciu este pus intr-o masina virtuala.

Diagrame+SOLID+explicat cod.

Problema 27

Pornind de la lab 8(cel cu docker), exemplul cu Student-Teacher, să se introducă un microserviciu de procesare de flux al datelor de intrare și sa se faca filtrarea (de ex. Sa filtrez după numărul portului). Sa se mai adauge și un microserviciu de adaugare în baza de date a informațiilor filtrate. La final, discutia pe cod + diagramele micros. si clase + SOLID

Deci. Ai un state machine ( nu te intereseaza atat de mult valoarea din nod, nici tranzitia de pe ramuri)

Fiecare nod este o functie serverless care citeste DINTR-O COADA.

(Iti trebuie dependenta de rabbitmq pt micronaut, iar functia trebuie configurata sa asculte update-uri in coada, spre deosebire de cea din lab care asculta la o ruta ce primea un Post)

Userul face un post la un rest, iar acesta pune  in coada de mai sus.

Logica de graf era descrisa in erastoteneService(paralela la lab), iar acolo am descris un nod generic ( un hash map <string,List<string>>, adica

<nod\_curent, List<tranzitie, destinatie>>).

Pentru a respecta logica din lab, cand functia serverless primeste ceva in coada, destructureaza mesajul (pt ca tot tu il concatenezi cu un separator definit de tine atunci cand ai pus in coda), iar datele obtinute din destructurare(entry\_node, tranzitie) le impachetezi intr-un NodeRequest (paralela la EratosteneRequest).

Dupa ce faci handler.apply(EratosteneRequest), raspunsul (adica nodul in care ai ajuns dupa iterarea din hashMap) este impachetat intr-un NodeResponse (tot paralela la EratosteneResponse). Rezultatul este doar apoi luat cu un getter din NodeResponse si afisat.

Logica din graph:

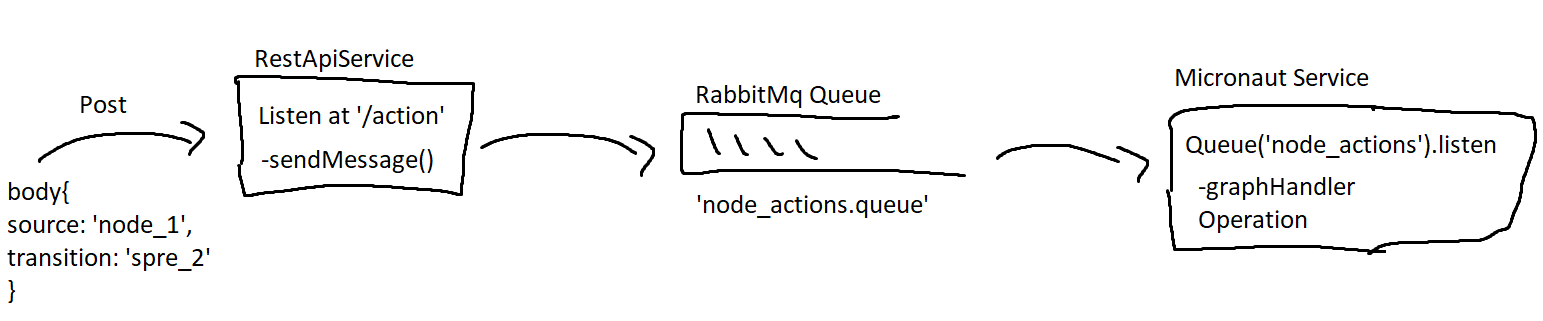
Daca entry\_node se gaseste ca si cheie in hashMap, iar la acea cheie ai disponisibla operatia pe care o doresti sa o calculezi din coada, atunci inseama ca se doreste a calcula o operatie valida (ceva ce graphul suporta), deci returnezi nodul destinatie.

Altfel inseamna ca se doreste a calcula o operatie invalida (in loc de nod\_destinatie ca rasp, pui si tu “no se puede” :)) )

\*pe bilet nu scrie despre diagrama de microserv sau clase, dar tot o faci pt microserv sau ceva asemanator cu cea de mai jos in diagrams.io, altfel rip.

\*cum asculta micronaut la RabbitMq:

<https://micronaut-projects.github.io/micronaut-rabbitmq/latest/guide/#consumer>



—--

21)Laborator 7- Okazii

-introducere procesor de flux sub forma unui microserviciu separat care sa monotorizeze operatiile efectuate de celelalte procesoare de flux

-modificarea microserviciului ca sa trimita unui sistem de cozi mesajele care trec prin el

-sistemul de cozi va trebui organizat utilizanti ontologi intr-o baza de date a lui de ditpul SQLite

-mictoserviciile vor fi fiecare in masina lor virtuala

-se prezinta mai intai diagramele de servicii si cele de clase apoi discutie pe cod

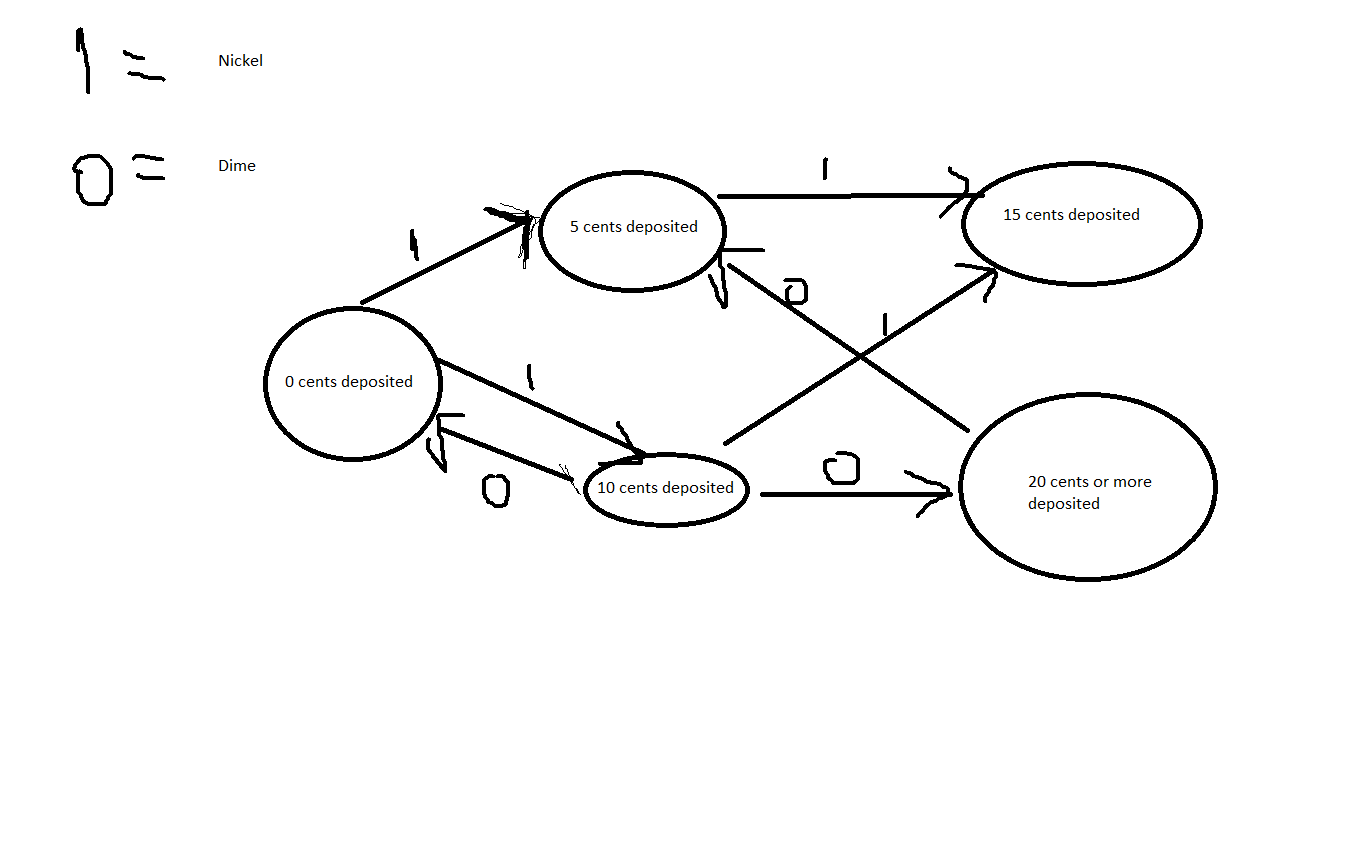
Simularea unui state machine ca cel din imagine prin coregrafia de microservicii (chanining). Pentru comunicarea dintre acele microservicii trebuia folosit:

Subiect1: Cozi RabbitMQ

Subiect2: TCP

Subiect3: Kafka (asta nu am văzut ca e dar ar trebui sa existe avand in vedere ca exista cu Rabbit si TCP )

Bonus: Era un subiect în care în loc de microservicii ca noduri trebuia sa folosesti funcții serverless (deci micronaut) care comunica prin RabbitMQ.



Pornind de la aplicația exemplu de chat din laboratorul 8, să se

modifice aceasta astfel încât să se introducă un procesor de flux care

să filtreze toată comunicația după niște reguli (de exemplu: se

acceptă toate care vin din gama de porturi XX-YY), iar rezultatul se

salvează într-un fișier local.

Dacă se consideră necesar se mai pot crea oricâte servicii se dorește

conform gândirii studentului. Se vor respecta principiile SOLID pentru

microservicii. La prezentare se va arăta diagrama de clase și se va

explica aceasta împreună cu gradul de respectare a principiilor. Apoi

va urma discuția pe cod.

Pornind de la aplicatia exemplu de chat din lab 8, sa se modifice aceasta prin utilizarea corutinelor si sa se introduca un procesor flux (un microserviciu separat care sa realizeze statistici cu privire la erorile de comunicare (specifice TCP) si tipul acestora care au aparut (daca au aparut) si apoi aceste informatii sunt trimise unui serviciu REST care va afisa cererea unui navigator pagina completa cu erorile aparute (deci va trebui ca acesta sa tina local un fisier XML in care sa faca adaugare). Daca se considera necesar, se mai pot crea oricate servicii se doreste conform gandirii studentului. Microseriviciile conform definitiei unui microserviciu vor fi fiecare in masina lor virtuala. Se va utiliza coregrafia si respectarea principiilor SOLID pentru microservicii. La prezentare se va arata diagrama de servicii si apoi cea de clase asociate implementarii fiecarui sserviciu si se va explica aceasta impreuna cu gradul de respecctare a principiilor SOLID pentru microservicii dar si SOLID pentru clase in cazul in care implementarea unui erviciu este mai complicata. Apoi va urma discutia pe cod.