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

Sisteme Distribuite - Laborator 6

Microservicii Web cu Spring Boot în Kotlin

Iaşi, 2022

Laborator 6 SD: Microservicii Web cu Spring Boot în Kotlin

Introducere

Aplicații monolitice

Primele aplicații erau de dimensiuni mici, iar logica de prezentare era la un loc cu logica de business - vezi modelul cu două sau trei niveluri despre care am discutat. Ele sunt legate de vremea când aplicația era executată pe o singură mașină. Nu uitați că acest termen a fost intodus de abia în epoca microserviciilor și a norului și de aceea el se referă la o mare varietate de implementări ale aplicațiilor. Deși gestiunea lor și mai ales depanarea era mai ușoară datorită centralizării este clar că astfel de abordări nu sunt potrivite pentru nor. Unii ingineri consideră că sunt ușor de replicat - iar acestă percepție este datorată neînțelegerii ideii din spatele multiplicării (adică de reducere a zonelor de supraîncărcare specifice unei astfel de aplicații) de fapt le este ușor să le gestioneze la zona de ops și de aici confuzia.

Aplicații cu microservicii

Reamintim de la curs geneza termenului de microserviciu. Termenul de microaplicații a fost introdus în 2011 de către James Lewis de la ThoughtWorks care a început să studieze micro aplicațiile. Acesta era un model de proiectare care începuse deja să fie utilizat de unele companii care utilizau SOC. Ulterior, în 2012, în urma unor discuții la un congres de specialitate, s-a modificat în microservicii pentru a evita confuziile.

În cazul aplicațiilor web bazate pe microservicii se utilizează o abordare simplă, discutabilă (vezi problemele API) unde dacă sunt corect proiectate se poate ajunge ca să fie formate dintr-o suită de servicii mici, executate independent, care comunică prin mecanisme simple (lightweight) bazate pe HTTP.

Pentru mai multe detalii, vezi:

- Building Microservices (2015) - Sam Newman
- Hands-on microservices with Kotlin (2018) - Juan Antonio Medina Iglesias

Principii generale pentru proiectarea cu microservicii

În primul rând vă reamintesc că pentru cei care vor să facă proiectare corectă trebuie să pornească de la principiile discutate la curs a căror grad de generalitate este mai mare și astfel acoperă majoritatea problemelor care pot apărea în proiectare. Pentru simpli implementatori multe firme disperate și nu numai au încercat să creeze un fel de rezumat minimal astfel încât implementatorul să priceapă vag proiectul venit din zona ierarhică de proiectare. Tocmai pentru a putea face diferența între aceste două abordări vom prezenta aceste simplificări dar nu uitați că ele reprezintă abilități incorecte și autolimitative pentru un proiectant de arhitecturi serioase (dimensiuni medii și mari) cu microservicii. Vă rog să nu confundați acest rezumat pentru indieni cu principiile SOLID pentru microservicii care sunt diferite de principiile și modelele de proiectare primare. Să începem cu rezumatul:

- **Modelare ținând cont de graful(urile) parțial sau total asociat afacerii**. După cum am discutat la curs de la SOA acesta a fost un prim deziderat al comunității de afaceri - potrivirea proiectării cu lumea reală. Totuși de abia abordarea bazată pe microservicii are flexibilitatea necesară îndeplinirii unui astfel de deziderat. Este clar că și pentru aplicațiile mici trebuie măcar o echipă dacă nu măcar un software architect cu experiență de minim cinci ani.
- **Responsabilitate unică (single responsibility)** - Fiecare microserviciu are ca responsabilitate o singură parte din funcționalitatea aplicației, iar acea responsabilitate este încapsulată în interiorul lui. Nu uitați cataloagele de servicii și compunerea de servicii.

Această simplificare conduce de multe ori în faza de reproiectare pentru nor la spargerea ”cu ciocanul” în niște monoliți mai mici ai aplicației și implementarea lor ca microservicii. Aceste comportamente sunt des întâlnite în proiectarea SOA (care are multe exemple) de unde și confuzia majoră pe subiect. Deci nu scăpați de curs și de citit în plus pe lângă el.

• **Ascunderea implementării** - Microserviciile au un contract (o interfață) care reprezintă exact funcționalitatea primară sau mai complexă de afaceri pe care o implementează. Din confuzia cu proiectarea orientată pe obiect combinată cu neînțelegerea principiilor pentru microservicii în acest caz fie se face o spargere prea mare (nu se urmărește funcționalitatea de afaceri și reutilizare) fie dăm iar peste micromonoliți (vezi observația de la cazul anterior). Detaliile interne nu ar trebui dar pot fi la rândul lor implementate de alte servicii de afaceri sau ajutătoare.

• **Izolare (Isolation)** - Un microserviciu prin însăți definiția lui este izolat de restul entităților din nor datorită execuției virtualizate. Nu discutăm aici problemele de securitate ale mașinilor virtuale sau maniera în care trebuie proiectată sigur o aplicație bazată pe MSA. În cazul acestei virtualizări mașina virtuală care îl încapsulează poate conține în afara suportului necesar bunei lui execuții chiar și o formă avansată de virtualizare inclusiv un SGBD **DACĂ acest lucru a rezultat din faza de analiză și proiectare.**

• **Instalare independentă (Independently deployable)** - este caracteristică emergentă din faptul că codul și ce mai trebuie este încapsulat într-o mașină virtuală.

• **Creat pentru gestiunea posibilelor erori (Build for failure)** - După cum am discutat la curs această simplificare se referă simultan la mai multe aspecte specifice rezilienței. Reamintim pe scurt că aceste aspecte sunt luate diferit în considerare în funcție de nivelul de analiză arhitecturală pe care ne aflăm. La nivelul global prin acest principiu se înțelege abilitatea de restaurare automată fără pierdere de date a zonei afectate din diverse motive (de la simple erori care vin dintr-o analiză incompletă în faza primară de proiectare, trecând prin implementare defectuoasă și ajungând la atacuri cibernetice). Pe măsură ce gradul de detalii crește se observă că pentru a obține comportamentul global așteptat avem nevoie atât de abordări moderne în proiectarea MSA (vezi stratul de instrumentației discutat la curs) combinate cu abordările clasice pe care le-ați întâlnit și la mine dar și la alte materii. Ne referim la proiectarea cu tratarea erorilor, mecanisme de tratare a erorilor inclusiv cele avansate care permit restaurarea automată. Deoarece din nefericire în majoritatea cazurilor în cazul dezvoltării de MSA se lucrează sub un Agile prost înțeles și implementat vom prezenta mai jos o simplificare rezultată din aceste erori conceptuale pe care o veți găsi des referită și utilizată (o serie de puncte cheie minimale care trebuie urmărite dar din nefericire ele nu țin cont decât de specificul CI/CD din DevOps neglijând după cum am spus analiza și proiectarea strict necesare):

  • *Upstream* = înțelegerea felului în care dezvoltatorii o să trimită sau nu notificări de eroare clienților, ținând totodată cont de evitarea cuplării

  • *Downstream* = cum vor gestiona dezvoltatorii defectarea unui microserviciu sau a unui subsistem al acestuia (de ex. un sistem de gestiune a bazelor de date).

  • *Logging* = scrierea tuturor erorilor într-un jurnal de execuție care poate fi local sau global, ținând cont de cât de des se realizează salvarea acestor informații, de cantitatea de date și cum pot fi acestea accesate. De asemenea, trebuie luate în considerare și cazuri speciale, cum ar fi informații sensibile și implicații de performanță.

  • *Monitoring* = Monitorizarea trebuie să fie proiectată cu mare atenție. Este foarte dificil de gestionat o eroare fără informațiile potrivite în sistemele de

monitorizare. Dezvoltatorii trebuie să determine care elemente ale aplicației au informații semnificative.

- *Alerting* = se referă la abordarea veche a utilizării de declanșatoare (de fapt generare de evenimente) cu privire la ce se întâmplă în sistem. Reamintim că la ora actuală se constată că se renunță la aceste abordări și se trece în MSA în utilizarea arhitecturilor flux (vezi cursul) chiar dacă evenimentele rămân inima modelului declanșarea va fi intrinsecă dominant (conform automatul de stare asociat) și nu invers ca până acum.

- *Recovery* = Proiectarea trebuie să conțină și metodele specifice revenirii în cazul unei erori (vezi obs de mai sus). Revenirea automată (automatic recovery) este ideală, dar având în vedere că aceasta poate eșua, nu trebuie evitată revenirea manuală (manual recovery). În noile abordări de flux aceasta începe să scadă ca importanță - vezi teoria automatelor finite. Însă veți întâlni abordări vechi din DevOps și cu influențe masive în proiectare a confuziei dintre SOA și MSA care mai au nevoie de utilizarea pe scară largă a acestor cârpăceli. În cazul unei proiectări de la zero așa ceva se poate obține (cu destulă greutate) doar dacă se merge pe abordarea completă multinivel de echipe de arhitecți software prezentată la curs (unde vi s-a spus să citiți în plus pe subiect - măcar TOGAF complet)

- *Fallbacks* = Un mecanism bun de tratare a erorilor permite ca aplicația să funcționeze în continuare după apariția unei erori în sistem, în timp de dezvoltatorii lucrează să rezolve problema respectivă.

- **Scalabilitate/Multiplicitate/Multiplicabilitate (Scalability)** - Microserviciile trebuie să poată fi replicate independent. Acest lucru ține de maniera de proiectare utilizată și aici modelele de proiectare specifice sunt critice deoarece prin utilizarea lor se observă imediat în proiect dacă încapsularea într-o mașină virtuală a fost făcută ținând cont de aceste principii sau nu (dacă nu am o cuplare de ex între serviciu și unele din dependențele lui(care în mod normal ar trebui să fie tot microservicii de afaceri sau ajutătoare).

- **Automatizarea (Automation)** - Microserviciile trebuie proiectate ținând cont de lanțul specific CI/CD, de la construire și testare până la instalare și monitorizare. Modelul pentru dezvoltare/integrare continuă CI/CD trebuie proiectat de la începutul arhitecturii.

Domain-Driven Design (DDD)

Proiectarea bazată pe analiza domeniului reprezintă o manieră pentru dezvoltarea aplicațiilor complexe prin conectarea continuă a implementării la un model (care evoluează continuu) a conceptelor business de bază. De fapt ea reprezintă iar o simplificare masivă a abordărilor specifice sistemelor mari (TOGAF etc). Evident că cine a înțeles proiectarea mare poate urma aceste reguli simple cine nu le va interpreta complet aiurea cu rezultate care deja se văd mai ales în conjuncție cu un Agile prost implementat. La curs v-am explicat cum pe baza analizei workflow-urilor complexe ale unei organizații în primul rând se stabilește dacă nu există deja o ontologie complexă specifice domeniului(lor) de afaceri gestionate de aceasta. Apoi dacă nu s-a realizat reorganizarea grafurilor complexe de workflow se trece la faza de reorganizare a lor ținând cont de un număr variabil de dimensiuni în funcție de specificul și constrângerile unei afaceri. În momentul în care s-a încercat simplificare acestui proces complex (probabil de către dezvoltatori fără experiență în proiectare de sisteme mari) ei au luat-o invers (cel mai probabil datorită influenței de gândire de la programarea orientată obiect și cea modulară dar și confuziei grave care conduce la ideea fixă că principiile OOP pot fi direct utilizate în programarea MSA ceea ce este complet greșit! Deci vă rog înțelegeți și învățați să aplicați corect MĂCAR SOLID pentru microservicii.

Premisele DDD:

• accentul principal al proiectului cade pe domeniul de bază (core domain) și pe logica domeniului (domain logic) - de fapt aici se face reorganizarea în mai multe grafuri aflate în planuri paralele și interoperante a workflow-ului afacerii)

• Proiectele mai complexe trebuie bazate pe un model (orice proiect are nevoie de un model (poate fi de-a gata sau trebuie să rezulte din analiză și proiectare)!)

• inițierea unei colaborări creative între experții tehnici și experții din domeniu pentru a ajunge din ce în ce mai aproape de modelul conceptual al problemei

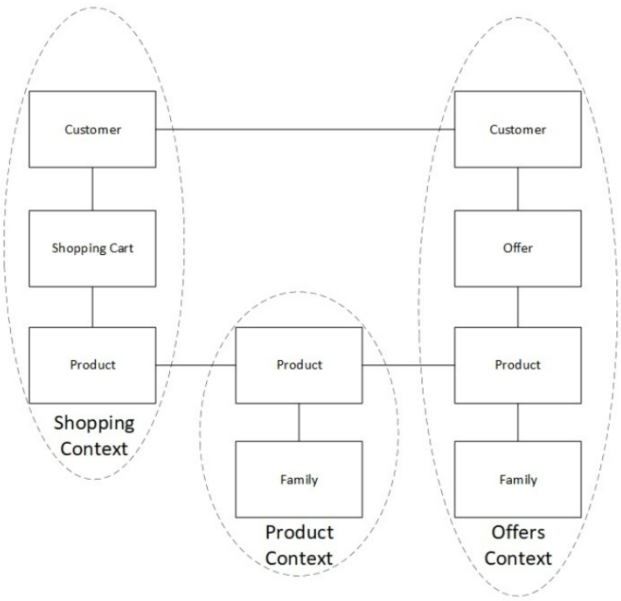
Problema: când complexitatea scapă de sub control, software-ul nu mai poate fi înțeles suficient de bine pentru a putea fi schimbat sau extins cu ușurință. Dacă complexitatea domeniului nu este tratată în proiectare, nu contează că tehnologia infrastructurii suport este bine concepută. Exact ce nu au înțeles dezvoltatorii din proiectarea mare!

Principiile de proiectare:

• **Context mărginit (Bounded context):** Când se abordează un sistem complex, de obicei se abstractizează într-un model care descrie aspectele diferite ale sistemului și cum poate fi folosit pentru a rezolva probleme. Când există mai multe modele, iar codul de bază al diferitelor modele este combinat, software-ul devine plin de erori (buggy), nesigur și greu de înțeles. Se observă iar că nu pomenește nimeni de analiză și proiectare dar maimuța care scrie cod a început să simtă ca ar avea nevoie de așa ceva. În DDD, se definește contextul în care se aplică un model, se stabilesc explicit granițele în ceea ce privește organizarea echipei și utilizarea în anumite părți ale aplicației, păstrând modelul **consecvent** cu aceste limite - vezi observația cu grafurile.

• **Limbaj generic specific (Ubiquitous language):** În DDD trebuie alcătuit un limbaj comun și riguros între dezvoltatori și utilizatori. Acest limbaj trebuie să fie bazat pe modelul de domeniu, ajutând în a avea o conversație generală între toți experții din domeniu, acest lucru fiind esențial la abordarea testării (deh dacă nu aplicăm principiile corecte de analiză primară nu obținem ontologia și atunci o luăm după ureche și ca atare a rezultat acest așa zis limbaj).

• **Capturarea/maparea contextului (Context mapping):** Într-o aplicație de dimensiuni mari, proiectată pentru mai multe contexte mărginite (bounded contexts), se poate pierde vederea de ansamblu. Inevitabil, contextele mărginite vor fi nevoite să comunice date între ele. O mapare de context este o vedere de ansamblu (global view) asupra sistemului ca un întreg, care ilustrează maniera în care contextele mărginite ar trebui să comunice între ele. (după cum spuneam o iau de sus în jos (bottom up) și la sfârșit se crucesc că nu seamănă cu ce avea nevoie clientul). Poate așa (cu aceste analize comparative) am să vă conving de importanța respectării fazelor de analiză și proiectare în maniera în care vă zic începând din anul II.



Exemplu de mapare de context

Pentru mai multe detalii, vezi cartea **„Domain-Driven Design”** scrisă de **Eric Evans**, precum și comunitatea DDD: https://dddcommunity.org/.

Folosirea DDD în microservicii

Aici vom încerca să simplificăm (vezi observațiile anterioare despre implementator) anumite aspecte complexe care țin de proiectarea corectă a sistemelor mari cu microservicii. Evident se pierde atât de mult că nu se mai înțelege mai nimic.

- **Bounded Context** - Nu trebuie creat un microserviciu care include mai mult de un context mărginit (aici apar erorile cele mai mari pentru că fiecare ia un subflux de afaceri și apoi încearcă să le potrivească “cu ciocanu” - problema abordării bottom up)

- **Ubiquitous Language** - Dezvoltatorii trebuie să se asigure că maniera de comunicare utilizată este suficient de general valabil, astfel încât operațiile și interfețele care sunt expuse să fie exprimate utilizând limbajul domeniului context (vezi observațiile anterioare)

- **Context Model** - Modelul utilizat de microserviciu trebuie definit într-un context mărginit și să folosească un limbaj generic (ubiquitous language), chiar și pentru entități care nu sunt expuse în nici o interfață pe care o oferă microserviciul

- **Context Mapping** - Trebuie examinat contextul mărginit al întregului sistem pentru a înțelege dependențele și cuplarea microserviciilor. Adică încearcă să se potrivească cu ce era. Evident că fără analiza globală se scapă foarte multe aspecte din vedere ba mai rău în loc să îi ajute îi încurcă pe termen lung. Aceasta se datorează rezistenței la schimbare în organizații care nu înțeleg necesitatea reanalizării și reproiectării workflow-urilor și cer o implementare a situației existente. Mai ales în cazul UE de Romania nici nu mai spun aceasta abordare este nesănătoasă). Din această cauză abordarea DDD & Agile pur este recomandată numai pentru proiectele de dimensiuni mici sau medii spre mici. Evident că până și la Web rămâne valabilă observația.

Exemple

Pentru un exemplu de microservicii în Kotlin cu framework-ul **Ktor**, se recomandă parcurgerea modelului de la adresa: https://dzone.com/articles/kotlin-microservices-with-ktor

Exemplul 1: BeerApp

Cerință: Să se creeze un microserviciu în Kotlin care să gestioneze o bază de date cu tipurile de bere. Aplicația Kotlin va comunica cu o interfață de tip CLI din python prin intermediul framework-ului RabbitMQ. Baza de date utilizată de microserviciu este SQLite3.

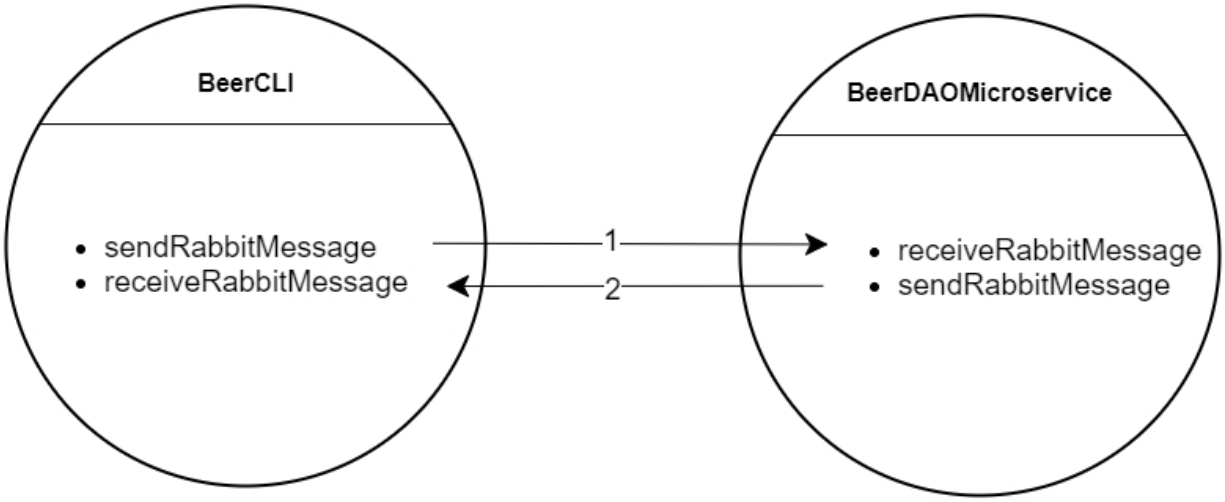


Diagrama de microservicii

În continuare, se prezintă diagrama UML pentru microserviciul BeerDAOMicroservice.

Pentru structurarea microserviciului BeerDAOMicroservice, s-au folosit 3 nivele, fiecare având rol și responsabilitate diferită în cadrul aplicației:

- prezentare (responsabil de interacțiunea cu utilizatorul)
- business (responsabil de partea de logică, adică procesarea/agregarea datelor)
- persistență (responsabil pentru păstrarea pe termen lung a informațiilor și care de obicei se ocupă cu transformări bidirecționale între reprezentările datelor care sunt specifice sistemelor de gestiune a bazelor de date (SGBD) și cele utilizate în aplicație (de obicei în nivelul de afaceri și mai rar (de obicei dacă aplicația nu are strat explicit pentru afaceri sau este proiectată cu cele din partea de prezentare).

Astfel, o cerere din nivelul de prezentare va trece prin nivelul de business pentru a accesa nivelul de persistență. Acest lucru produce o decuplare între modulele aplicației si previne ca aplicația să devină dificil de întreținut.

Reamintim că există și modelul cu două niveluri care este specific unor aplicații foarte simple unde nivelul de business și nivelul de persistență sunt combinate într-un singur nivel (de ex. atunci când componentele de pe nivelul de persistență pot fi integrate direct în componente de pe nivelul de busines). Totuși în aplicații serioase se merge pe abordarea multilayer-multitier. Reamintim că această abordare este o generațizarea a abordării cu trei niveluri deoarece fiecare nivel este separat în substraturi fiecare având fiecare rolul lui specific (de ex. strat de validare a datelor, straturi de securitate sau chiar straturi de afaceri).

Atenție: Nu confundați serviciile ca componente ale nivelului de business în cadrul unei aplicații, cu serviciile din cadrul SOA (eng. Service Oriented Architecture) deoarece întâi se face

proiectarea și apoi implementarea (lucru nu prea respectat din nefericire) și atunci pot exista suprapuneri parțiale în cazul implementării sau nu.

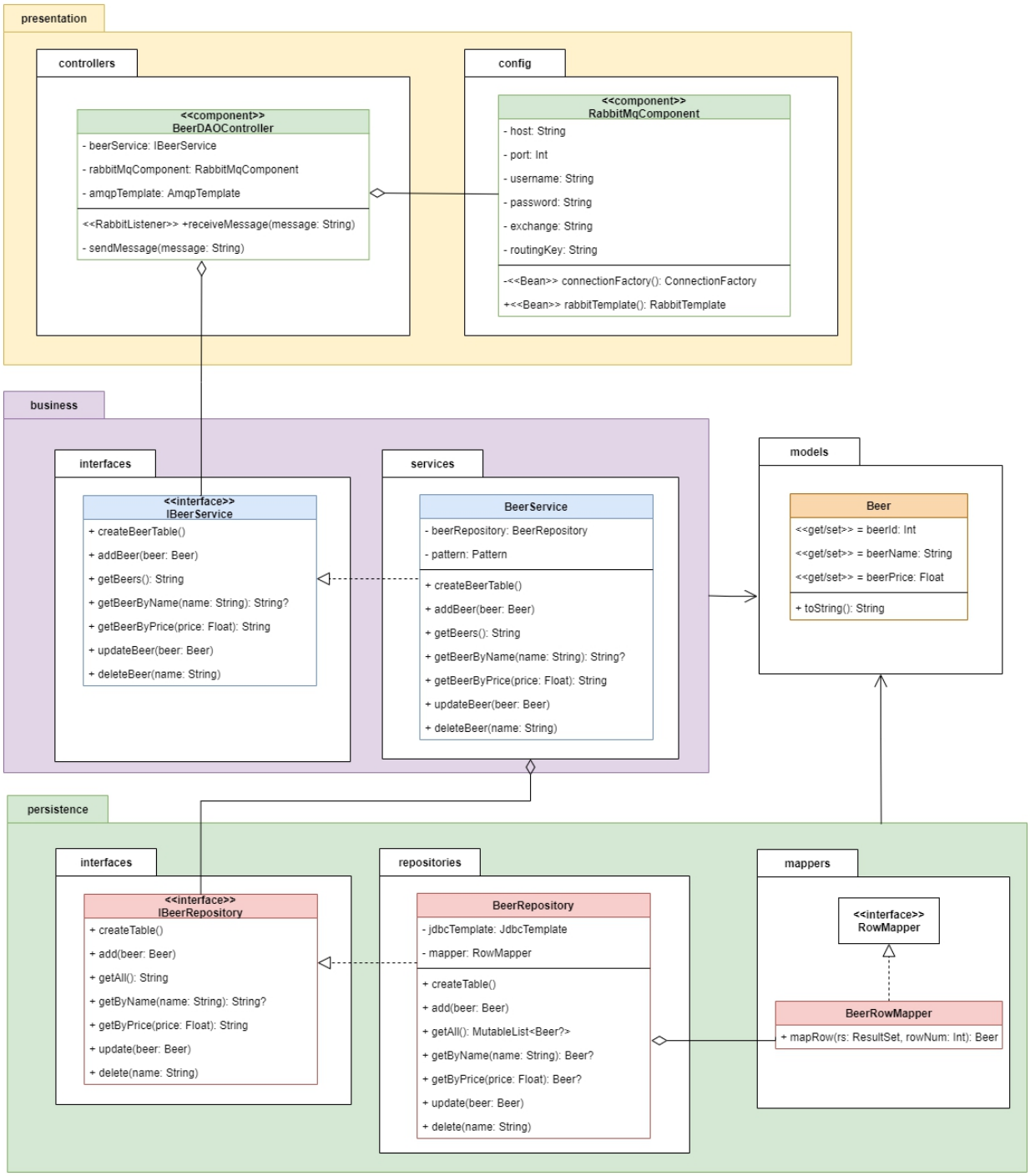


Diagrama UML pentru microserviciul BeerDAOMicroservice

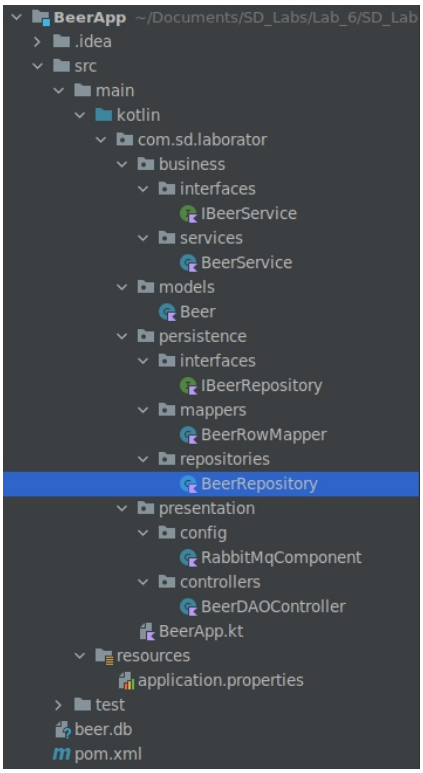
Exemplul 1: Configurări

Similar cu laboratorul 5 de la disciplina Sisteme Distribuite, se configurează următoarele cozi de mesaje, shimburi (exchange) și chei de rutare în interfața RabbitMQ (localhost:15672):

- un exchange: **beerapp.direct**
- două cozi
  - **beerapp.queue**
  - **beerapp.queue1**

- două legături (binding):
  - **beerapp**.queue -> **beerapp**.direct, **beerapp.routingkey**
  - **beerapp**.queue1 -> **beerapp**.direct, **beerapp.routingkey1**

Structura proiectului



Ierarhia aplicației BeerApp

Configurarea parametrilor din fișierul appplication.properties

```
spring.datasource.url=jdbc:sqlite:beer.db
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=student
spring.rabbitmq.password=student
beerapp.rabbitmq.queue=beerapp.queue1
beerapp.rabbitmq.exchange=beerapp.direct
beerapp.rabbitmq.routingkey=beerapp.routingkey
```

Se remarcă parametrul **spring.datasource.url** care precizează faptul că jdbc-ul (java database connector) va utiliza o bază de date de tip sqlite denumită beer.db.

Crearea proiectului

Proiectul se creează conform instrucțiunilor din laboratorul 5 de la disciplina Sisteme Distribuite.

Se adaugă dependențele pentru **sqlite** în fișierul pom.xml**:**

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

```
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.36.0.2</version>
</dependency>
```

Exemplul 1: Codul sursă

Microserviciul BeerDAOMicroservice

- **BeerApp.kt**

```
package com.sd.laborator

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

@SpringBootApplication
class BeerApp

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

- **Beer.kt**

```
package com.sd.laborator.models

class Beer(private var id: Int, private var name: String, private var
price: Float) {

    var beerID: Int
        get() {
            return id
        }
        set(value) {
            id = value
        }
    var beerName: String
        get() {
            return name
        }
        set(value) {
            name = value
        }
    var beerPrice: Float
        get() {
            return price
        }
        set(value) {
            price = value
        }

    override fun toString(): String {
        return "Beer [id=$beerID, name=$beerName, price=$beerPrice]"
    }
}
```

- **BeerRowMapper.kt**

```
package com.sd.laborator.persistence.mappers

import com.sd.laborator.models.Beer
import org.springframework.jdbc.core.RowMapper
import java.sql.ResultSet
import java.sql.SQLException

class BeerRowMapper : RowMapper<Beer?> {
    @Throws(SQLException::class)
    override fun mapRow(rs: ResultSet, rowNum: Int): Beer {
        return Beer(rs.getInt("id"), rs.getString("name"),
rs.getFloat("price"))
    }
}
```

- **IBeerRepository.kt**

```
package com.sd.laborator.persistence.interfaces

import com.sd.laborator.models.Beer

interface IBeerRepository {
    // Create
    fun createTable()
    fun add(beer: Beer)

    // Retrieve
    fun getAll(): MutableList<Beer?>
    fun getByName(name: String): Beer?
    fun getByPrice(price: Float): MutableList<Beer?>

    // Update
    fun update(beer: Beer)

    // Delete
    fun delete(name: String)

}
```

- **BeerRepository.kt**

```
package com.sd.laborator.persistence.repositories

import com.sd.laborator.models.Beer
import com.sd.laborator.persistence.interfaces.IBeerRepository
import com.sd.laborator.persistence.mappers.BeerRowMapper
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.dao.EmptyResultDataAccessException
import org.springframework.jdbc.UncategorizedSQLException
import org.springframework.jdbc.core.JdbcTemplate
import org.springframework.jdbc.core.RowMapper
import org.springframework.stereotype.Repository

@Repository
class BeerRepository: IBeerRepository {
    @Autowired
    private lateinit var _jdbcTemplate: JdbcTemplate
```

```
    private var _rowMapper: RowMapper<Beer?> = BeerRowMapper()

    override fun createTable() {
        _jdbcTemplate.execute("""CREATE TABLE IF NOT EXISTS beers(
                                        id INTEGER PRIMARY KEY
AUTOINCREMENT,
                                        name VARCHAR(100) UNIQUE,
                                        price FLOAT)""")
    }

    override fun add(beer: Beer) {
        try {
            _jdbcTemplate.update("INSERT INTO beers(name, price)
VALUES (?, ?)", beer.beerName, beer.beerPrice)
        } catch (e: UncategorizedSQLException){
            println("An error has occurred in
${this.javaClass.name}.add")
        }
    }

    override fun getAll(): MutableList<Beer?> {
        return _jdbcTemplate.query("SELECT * FROM beers", _rowMapper)
    }

    override fun getByName(name: String): Beer? {
        return try {
            _jdbcTemplate.queryForObject("SELECT * FROM beers WHERE
name = '$name'", _rowMapper)
        } catch (e: EmptyResultDataAccessException) {
            null
        }
    }

    override fun getByPrice(price: Float): MutableList<Beer?> {
        return _jdbcTemplate.query("SELECT * FROM beers WHERE price <=
$price", _rowMapper)
    }

    override fun update(beer: Beer) {
        try {
            _jdbcTemplate.update("UPDATE beers SET name = ?, price = ?
WHERE id = ?", beer.beerName, beer.beerPrice, beer.beerID)
        } catch (e: UncategorizedSQLException){
            println("An error has occurred in
${this.javaClass.name}.update")
        }
    }

    override fun delete(name: String) {
        try {
            _jdbcTemplate.update("DELETE FROM beers WHERE name = ?",
name)
        } catch (e: UncategorizedSQLException){
            println("An error has occurred in
${this.javaClass.name}.delete")
        }
    }
```

```
}
```

- **IBeerService.kt**

```
package com.sd.laborator.business.interfaces

import com.sd.laborator.models.Beer

interface IBeerService {
    fun createBeerTable()
    fun addBeer(beer: Beer)

    fun getBeers(): String
    fun getBeerByName(name: String): String?
    fun getBeerByPrice(price: Float): String

    fun updateBeer(beer: Beer)

    fun deleteBeer(name: String)
}
```

- **BeerService**

```
package com.sd.laborator.business.services

import com.sd.laborator.business.interfaces.IBeerService
import com.sd.laborator.models.Beer
import com.sd.laborator.persistence.interfaces.IBeerRepository
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Service
import java.util.regex.Pattern

@Service
class BeerService: IBeerService {
    @Autowired
    private lateinit var _beerRepository: IBeerRepository
    private var _pattern: Pattern = Pattern.compile("\\W")

    override fun createBeerTable() {
        _beerRepository.createTable()
    }

    override fun addBeer(beer: Beer) {
        if(_pattern.matcher(beer.beerName).find()) {
            println("SQL Injection for beer name")
            return
        }
        _beerRepository.add(beer)
    }

    override fun getBeers(): String {
        val result: MutableList<Beer?> = _beerRepository.getAll()
        var stringResult: String = ""
        for (item in result) {
            stringResult += item
        }
        return stringResult
```

```
    }

    override fun getBeerByName(name: String): String? {
        if(_pattern.matcher(name).find()) {
            println("SQL Injection for beer name")
            return null
        }
        val result = _beerRepository.getByName(name)
        return result.toString()
    }

    override fun getBeerByPrice(price: Float): String {
        val result = _beerRepository.getByPrice(price)
        var stringResult: String = ""
        for (item in result) {
            stringResult += item
        }
        return stringResult
    }

    override fun updateBeer(beer: Beer) {
        if(_pattern.matcher(beer.beerName).find()) {
            println("SQL Injection for beer name")
            return
        }
        _beerRepository.update(beer)
    }

    override fun deleteBeer(name: String) {
        if(_pattern.matcher(name).find()) {
            println("SQL Injection for beer name")
            return
        }
        _beerRepository.delete(name)
    }
}
```

- **BeerDAOController.kt**

```
package com.sd.laborator.presentation.controllers

import com.sd.laborator.business.interfaces.IBeerService
import com.sd.laborator.models.Beer
import com.sd.laborator.presentation.config.RabbitMqComponent
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 BeerDAOController {
    @Autowired
    private lateinit var _beerService: IBeerService

    @Autowired
    private lateinit var _rabbitMqComponent: RabbitMqComponent
```

```
    private lateinit var _amqpTemplate: AmqpTemplate

    @Autowired
    fun initTemplate() {
        this._amqpTemplate = _rabbitMqComponent.rabbitTemplate()
    }

    // citesc din queue1
    // scriu in queue
    @RabbitListener(queues = ["\${beerapp.rabbitmq.queue}"])
    fun receiveMessage(msg: String) {
        val (operation, parameters) = msg.split('~')
        var beer: Beer? = null
        var price: Float? = null
        var name: String? = null

        // id=1;name=Corona;price=3.6
        if("id=" in parameters) {
            println(parameters)
            val params: List<String> = parameters.split(';')
            try {
                beer = Beer(
                    params[0].split('=')[1].toInt(),
                    params[1].split('=')[1],
                    params[2].split('=')[1].toFloat()
                )
            } catch (e: Exception) {
                print("Error parsing the parameters: ")
                println(params)
                return
            }
        } else if ("price=" in parameters) {
            price = parameters.split('=')[1].toFloat()
        } else if ("name=" in parameters) {
            name = parameters.split("=")[1]
        }
        println("Parameters: $parameters")
        println("Name: $name")
        println("Price: $price")
        println("Beer: $beer")
        val result: Any? = when(operation) {
            "createBeerTable" -> _beerService.createBeerTable()
            "addBeer" -> _beerService.addBeer(beer!!)
            "getBeers" -> _beerService.getBeers()
            "getBeerByName" -> _beerService.getBeerByName(name!!)
            "getBeerByPrice" -> _beerService.getBeerByPrice(price!!)
            "updateBeer" -> _beerService.updateBeer(beer!!)
            "deleteBeer" -> _beerService.deleteBeer(name!!)
            else -> null
        }
        println("Result: $result")
        if (result != null) sendMessage(result.toString())
    }

    private fun sendMessage(msg: String) {
        println("Message to send: $msg")
```

```
this._amqpTemplate.convertAndSend(_rabbitMqComponent.getExchange(),
_rabbitMqComponent.getRoutingKey(), msg)
    }
}
```

- **RabbitMqComponent**

```
package com.sd.laborator.presentation.config

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 RabbitMqComponent {
    @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("\${beerapp.rabbitmq.exchange}")
    private lateinit var exchange: String
    @Value("\${beerapp.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())
}
```

Microserviciul BeerCLI

```
import pika
from retry import retry
```

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

    def on_received_message(self, blocking_channel, deliver,
properties,
                            message):
        result = message.decode('utf-8')
        blocking_channel.confirm_delivery()
        try:
            print(result)
        except Exception:
            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'])


def print_menu():
    print('0 --> Exit program')
    print('1 --> addBeer')
    print('2 --> getBeers')
    print('3 --> getBeerByName')
    print('4 --> getBeerByPrice')
    print('5 --> updateBeer')
    print('6 --> deleteBeer')
    return input("Option=")


if __name__ == '__main__':
    rabbit_mq = RabbitMq()
    rabbit_mq.send_message("createBeerTable~")
    while True:
        option = print_menu()
        if option == '0':
            break
        elif option == '1':
            name = input("Beer name: ")
            price = float(input("Beer price: "))
            rabbit_mq.send_message("addBeer~id=-
1;name={};price={}".format(name, price))
        elif option == '2':
            rabbit_mq.send_message("getBeers~")
            rabbit_mq.receive_message()
        elif option == '3':
            name = input("Beer name: ")

rabbit_mq.send_message("getBeerByName~name={}".format(name))
            rabbit_mq.receive_message()
        elif option == '4':
            price = float(input("Beer price: "))

rabbit_mq.send_message("getBeerByPrice~price={}".format(price))
            rabbit_mq.receive_message()
        elif option == '5':
            id = int(input("Beer ID: "))
            name = input("Beer name: ")
            price = float(input("Beer price: "))

rabbit_mq.send_message("updateBeer~id={};name={};price={}".format(id,
name, price))
            rabbit_mq.receive_message()
        elif option == '6':
            name = input("Beer name: ")
            rabbit_mq.send_message("deleteBeer~name={}".format(name))
        else:
            print("Invalid option")
```

```
```

Pentru testarea exemplului, din IntelliJ se execută **Maven -> Lifecycle -> clean + compile + package**. La final, se deschide directorul **target** creat și se execută în terminal comanda:

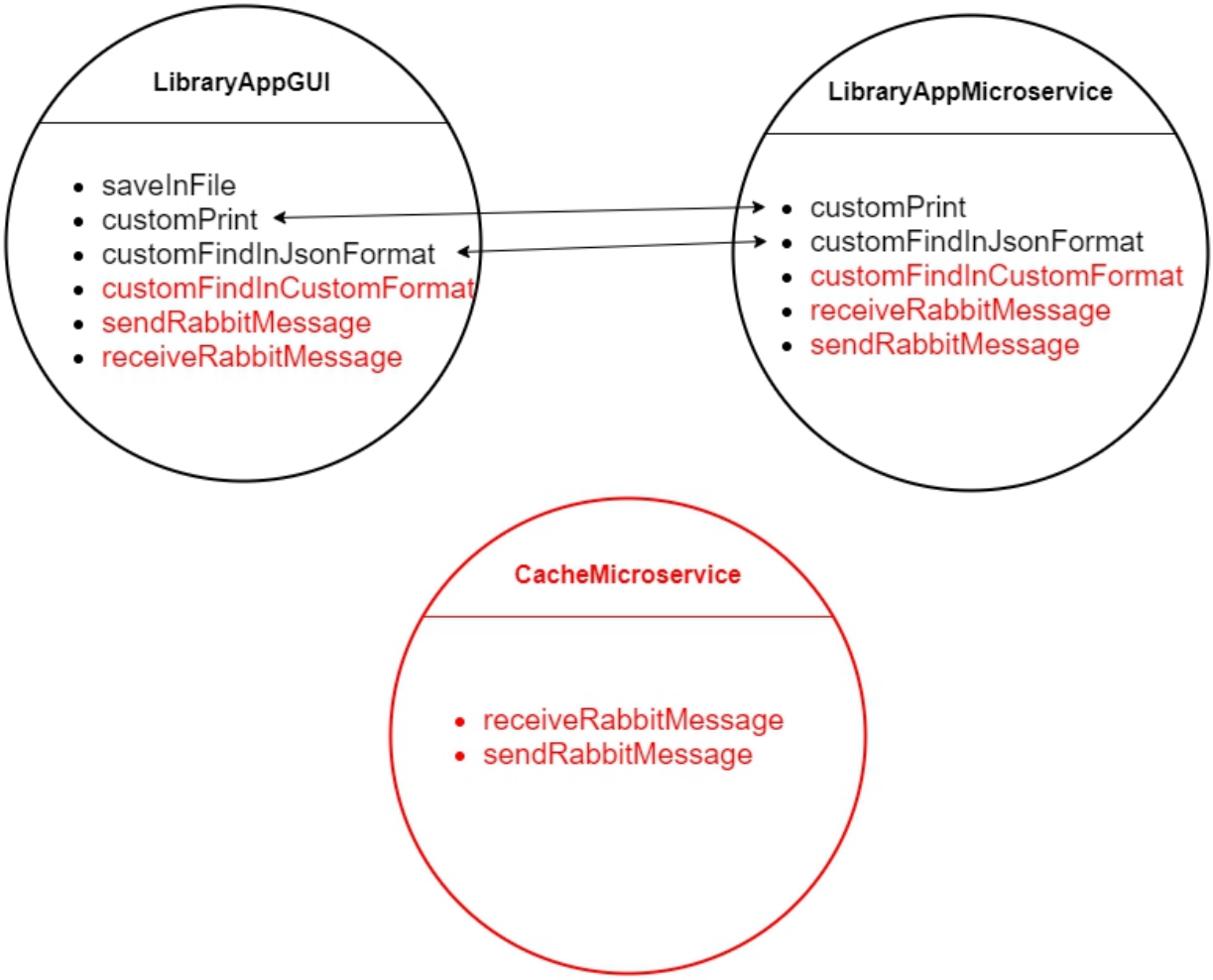
```
java -jar BeerApp-1.0.0.jar
```

Pentru a porni interfața CLI, se execută într-un terminal deschis în directorul cu aplicația python comenzile:

```
python3 -m venv env
source env/bin/activate
pip3 install pika==1.1.0 retry==0.9.2
python3 beer_app_cli.py
```

Exemplul 2: LibraryApp

Cerință: Să se scrie un program Kotlin care să realizeze prin intermediul unui microserviciu WEB gestiunea unei biblioteci 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).



Spre deosebire de exemplul anterior, microserviciul LibraryAppMicroservice va fi de tip Web, expunând către microserviciul de GUI metodele de afișare ale bibliotecii. Arhitecturile

pentru microserviciul LibraryAppMicroservice, respectiv CacheMicroservice sunt reprezentate în diagramele de clase de mai jos.

Observație: funcționalitățile marcate cu roșu din diagrama de mai sus nu se regăsesc în exemplul curent. Acestea trebuie implementate în tema pe acasă.

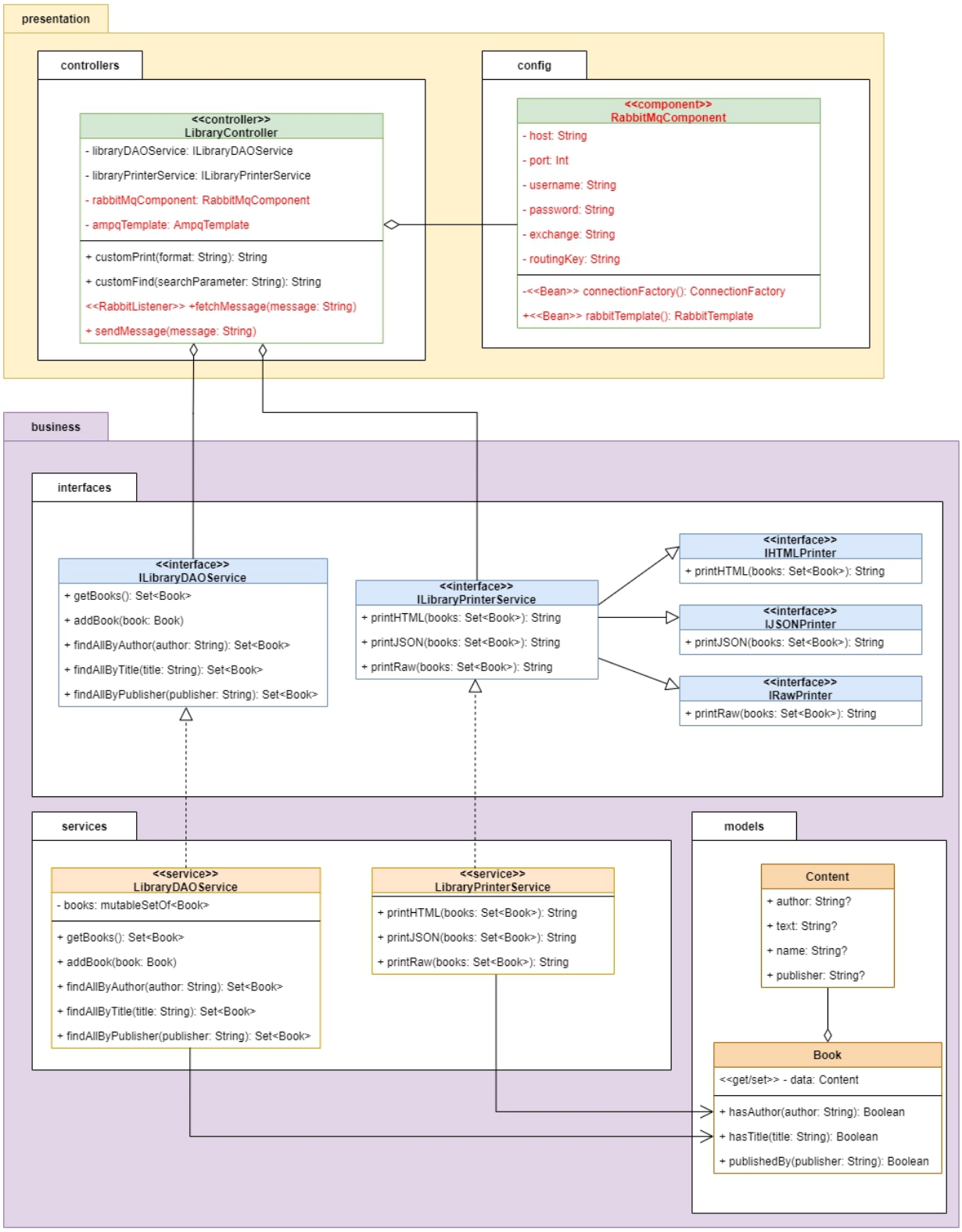


Diagrama UML al microserviciului LibraryAppMicroservice

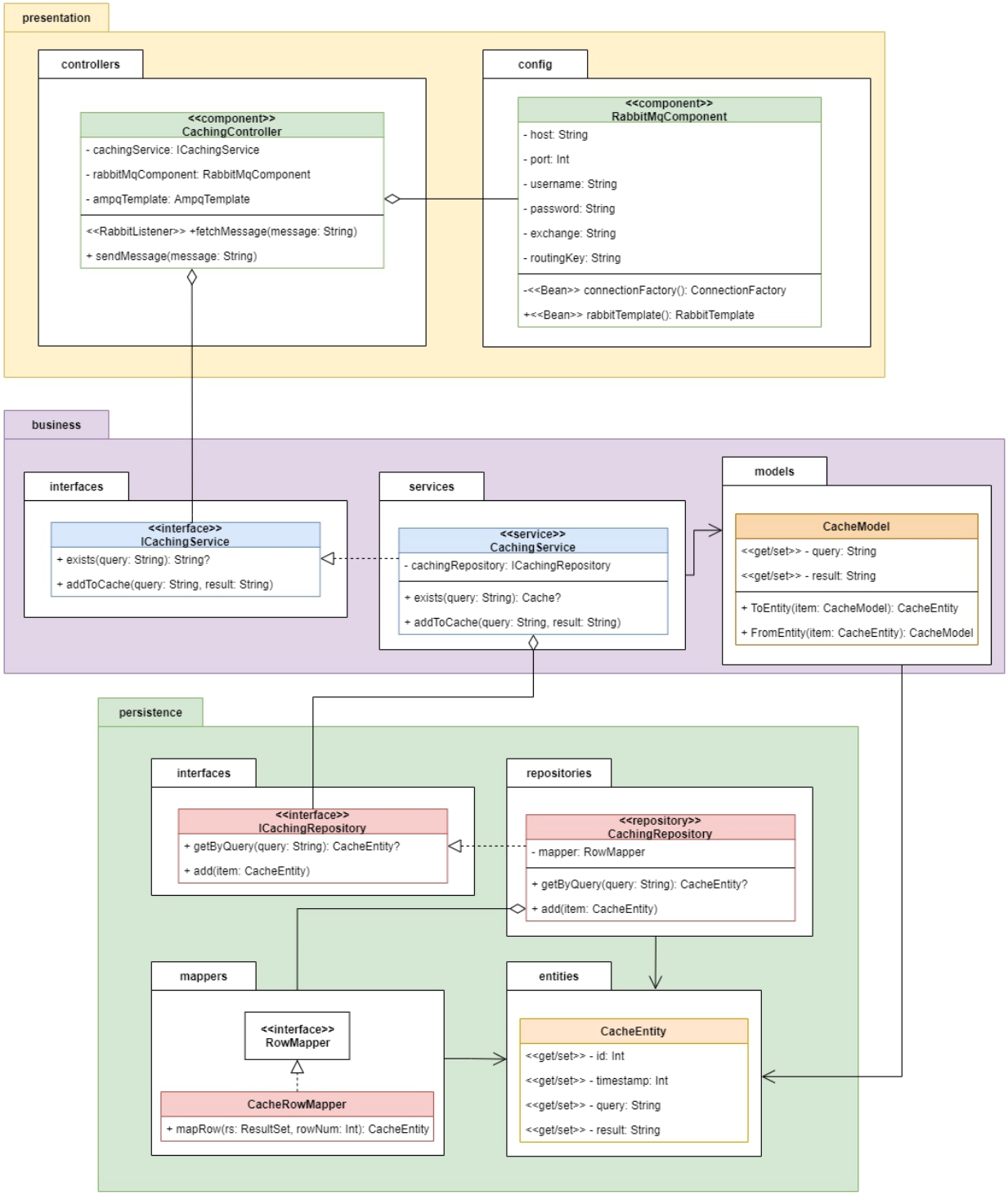
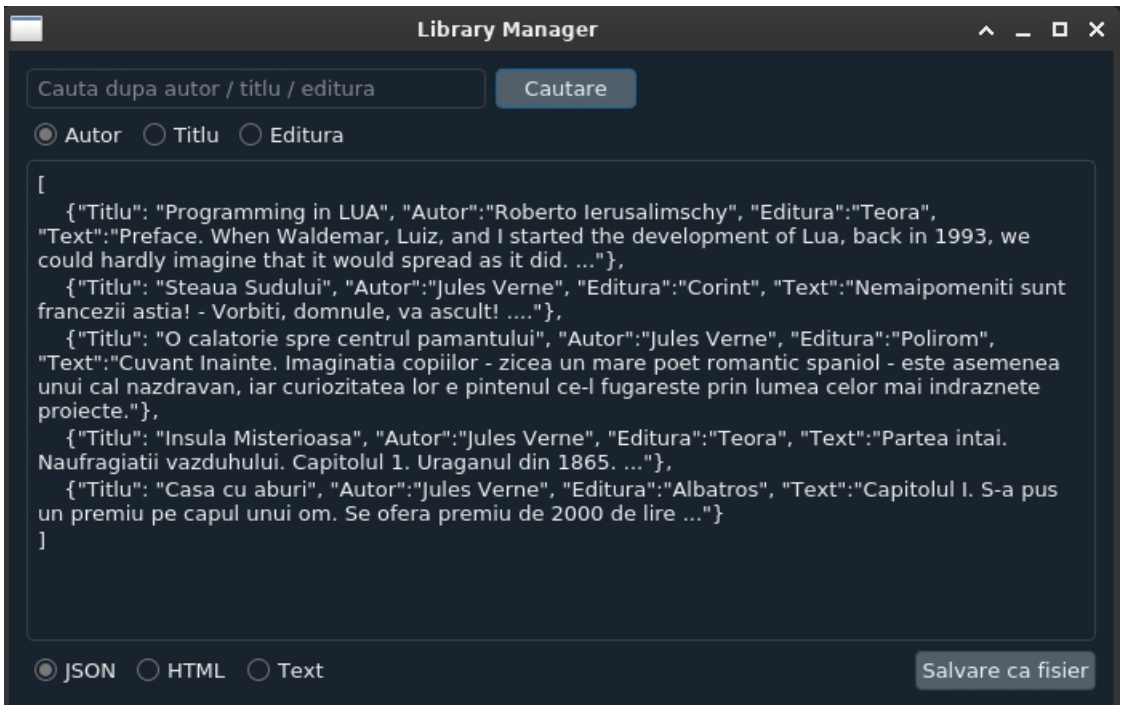


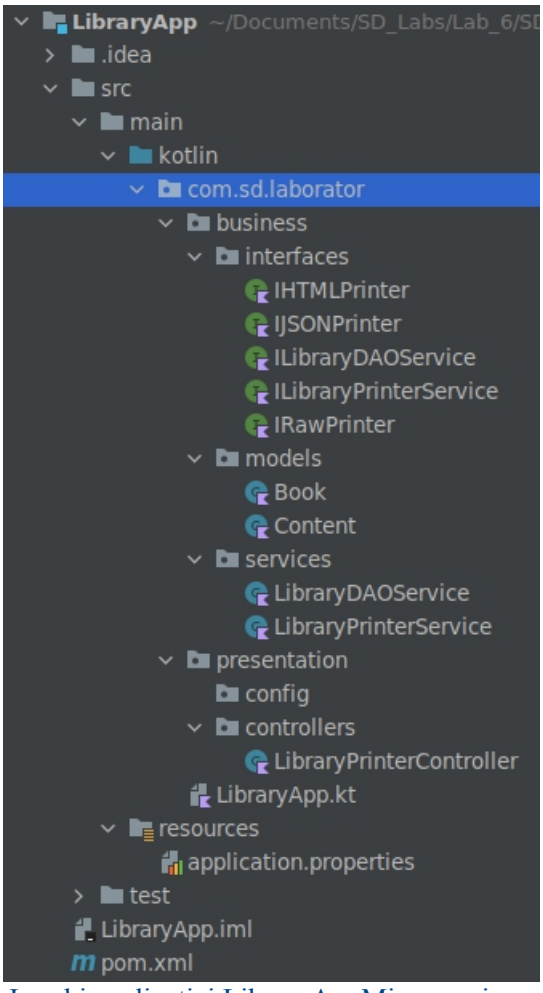
Diagrama UML al microserviciului CacheMicroservice

În acest exemplu s-a proiectat în așa fel pentru ca nivelul de business și nivelul de persistență să folosească *DTO*-uri (eng. Data Transfer Object) separate (business: **models**, persistență: **entities**), cu scopul de a încapsula anumite atribute ale modelului de Cache (precum *timestamp)*.



Interfața grafică LibraryAppGUI realizată cu PyQt5

Structura proiectului



Ierarhia aplicației LibraryAppMicroservice

Crearea proiectului

Proiectul se creează similar cu exemplul anterior. La final, se mai adaugă dependența de **spring-boot-starter-web** în fișierul pom.xml:

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

Exemplul 2: Codul sursă

Microserviciul LibraryAppMicroservice

- **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)
}
```

- **Content.kt**

```
package com.sd.laborator.business.models

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

- **Book.kt**

```
package com.sd.laborator.business.models

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)
    }

}
```

- **LibraryPrinterService.kt**

```
package com.sd.laborator.business.services

import com.sd.laborator.business.interfaces.ILibraryPrinterService
import com.sd.laborator.business.models.Book
import org.springframework.stereotype.Service

@Service
class LibraryPrinterService : ILibraryPrinterService {
    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
    }
}
```

- **LibraryDAOService.kt**

```
package com.sd.laborator.business.services

import com.sd.laborator.business.interfaces.ILibraryDAOService
import com.sd.laborator.business.models.Book
import com.sd.laborator.business.models.Content
import org.springframework.stereotype.Service

@Service
class LibraryDAOService : ILibraryDAOService {
    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()
    }
}
```

- **IHTMLPrinter**

```
package com.sd.laborator.business.interfaces

import com.sd.laborator.business.models.Book

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

- **IJSONPrinter**

```
package com.sd.laborator.business.interfaces

import com.sd.laborator.business.models.Book

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

- **ILibraryDAOService**

```
package com.sd.laborator.business.interfaces

import com.sd.laborator.business.models.Book

interface ILibraryDAOService {
    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>
}
```

- **ILibraryPrinterService**

```
package com.sd.laborator.business.interfaces

interface ILibraryPrinterService : IHTMLPrinter, IJSONPrinter,
IRawPrinter
```

- **IRawPrinter**

```
package com.sd.laborator.business.interfaces

import com.sd.laborator.business.models.Book

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

- **LibraryPrinterController**

Se observă faptul că microserviciul are acces prin intermediul unui ILibraryDAOService la baza de date (pe moment, la datele hardcodate), lipsind nivelul de persistență și totodată la funcționalitățile de printare ale LibraryPrinter. Funcționalitățile microserviciului WEB sunt expuse la următoarele URL-uri:

» http://localhost:8080/print?format=html

» http://localhost:8080/find?author=Jules%20Verne

» http://localhost:8080/find?title=Steaua%20Sudului

» http://localhost:8080/find?publisher=Corint

```
package com.sd.laborator.presentation.controllers

import com.sd.laborator.business.interfaces.ILibraryDAOService
import com.sd.laborator.business.interfaces.ILibraryPrinterService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.stereotype.Controller
import org.springframework.web.bind.annotation.RequestMapping
```

```
import org.springframework.web.bind.annotation.RequestMethod
import org.springframework.web.bind.annotation.RequestParam
import org.springframework.web.bind.annotation.ResponseBody

@Controller
class LibraryPrinterController {
    @Autowired
    private lateinit var _libraryDAOService: ILibraryDAOService

    @Autowired
    private lateinit var _libraryPrinterService:
ILibraryPrinterService


    @RequestMapping("/print", method = [RequestMethod.GET])
    @ResponseBody
    fun customPrint(@RequestParam(required = true, name = "format",
defaultValue = "") format: String): String {
        return when (format) {
            "html" ->
_libraryPrinterService.printHTML(_libraryDAOService.getBooks())
            "json" ->
_libraryPrinterService.printJSON(_libraryDAOService.getBooks())
            "raw" ->
_libraryPrinterService.printRaw(_libraryDAOService.getBooks())
            else -> "Not implemented"
        }
    }

    @RequestMapping("/find", method = [RequestMethod.GET])
    @ResponseBody
    fun customFind(
        @RequestParam(required = false, name = "author", defaultValue
= "") author: String,
        @RequestParam(required = false, name = "title", defaultValue =
"") title: String,
        @RequestParam(required = false, name = "publisher",
defaultValue = "") publisher: String
    ): String {
        if (author != "")
            return
this._libraryPrinterService.printJSON(this._libraryDAOService.findAllB
yAuthor(author))
        if (title != "")
            return
this._libraryPrinterService.printJSON(this._libraryDAOService.findAllB
yTitle(title))
        if (publisher != "")
            return
this._libraryPrinterService.printJSON(this._libraryDAOService.findAllB
yPublisher(publisher))
        return "Not a valid field"
    }

}
```

Microserviciul LibraryAppGUI

În codul de mai jos, se remarcă că, spre deosebire de aplicația din laboratorul precedent în care comunicația era realizată prin cozi de mesaje, exemplul curent utilizează modulul *requests* trimițând cereri HTTP de tip GET către microserviciul Web din Kotlin.

```
import os
import sys
import requests
import qdarkstyle
from requests.exceptions import HTTPError
from PyQt5.QtWidgets import (QWidget, QApplication, QFileDialog,
                             QMessageBox)
from PyQt5 import QtCore
from PyQt5.uic import loadUi

def debug_trace(ui=None):
    from pdb import set_trace
    QtCore.pyqtRemoveInputHook()
    set_trace()
    # QtCore.pyqtRestoreInputHook()

class LibraryApp(QWidget):
    ROOT_DIR = os.path.dirname(os.path.abspath(__file__))

    def __init__(self):
        super(LibraryApp, self).__init__()
        ui_path = os.path.join(self.ROOT_DIR, 'library_manager.ui')
        loadUi(ui_path, self)
        self.search_btn.clicked.connect(self.search)
        self.save_as_file_btn.clicked.connect(self.save_as_file)

    def search(self):
        search_string = self.search_bar.text()
        url = None
        if not search_string:
            if self.json_rb.isChecked():
                url = '/print?format=json'
            elif self.html_rb.isChecked():
                url = '/print?format=html'
            else:
                url = '/print?format=raw'
        else:
            if self.author_rb.isChecked():
                url = '/find?author={}'.format(
                            search_string.replace(' ', '%20'))
            elif self.title_rb.isChecked():
                url = '/find?title={}'.format(
                            search_string.replace(' ', '%20'))
            else:
                url = '/find?publisher={}'.format(
                            search_string.replace(' ', '%20'))
        full_url = "http://localhost:8080" + url
        try:
            response = requests.get(full_url)
            self.result.setText(response.content.decode('utf-8'))
        except HTTPError as http_err:
```

```
            print('HTTP error occurred: {}'.format(http_err))
        except Exception as err:
            print('Other error occurred: {}'.format(err))

    def save_as_file(self):
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        file_path = str(
            QFileDialog.getSaveFileName(self,
                                        'Salvare fisier',
                                        options=options))
        if file_path:
            file_path = file_path.split("'")[1]
            if not file_path.endswith('.json') and not
file_path.endswith(
                    '.html') and not file_path.endswith('.txt'):
                if self.json_rb.isChecked():
                    file_path += '.json'
                elif self.html_rb.isChecked():
                    file_path += '.html'
                else:
                    file_path += '.txt'
            try:
                with open(file_path, 'w') as fp:
                    if file_path.endswith(".html"):
                        fp.write(self.result.toHtml())
                    else:
                        fp.write(self.result.toPlainText())
            except Exception as e:
                print(e)
                QMessageBox.warning(self, 'Library Manager',
                                    'Nu s-a putut salva fisierul')


if __name__ == '__main__':
    app = QApplication(sys.argv)

    stylesheet = qdarkstyle.load_stylesheet_pyqt5()
    app.setStyleSheet(stylesheet)

    window = LibraryApp()
    window.show()
    sys.exit(app.exec_())
```

Pentru testarea exemplului, din IntelliJ se execută **Maven -> Lifecycle -> clean + compile**, apoi **Maven -> Plugins -> spring-boot -> spring-boot:run.**

Pentru a porni interfața grafică, se execută într-un terminal deschis în directorul cu aplicația python comenzile:

```
python3 -m venv env
source env/bin/activate
pip3 install -r requirements.txt
python3 library_manager.py
```

Aplicaţii şi teme

Aplicații de laborator:

- Să se reimplementeze persistarea datelor în exemplul LibraryApp utilizând o bază de date SQLite (ca în exemplul 1), având tabela **Book** în diagrama E-R de mai jos. Cu această ocazie se va adăuga un nivel de persistență ca în cazul microserviciul BeerAppMicroservice.

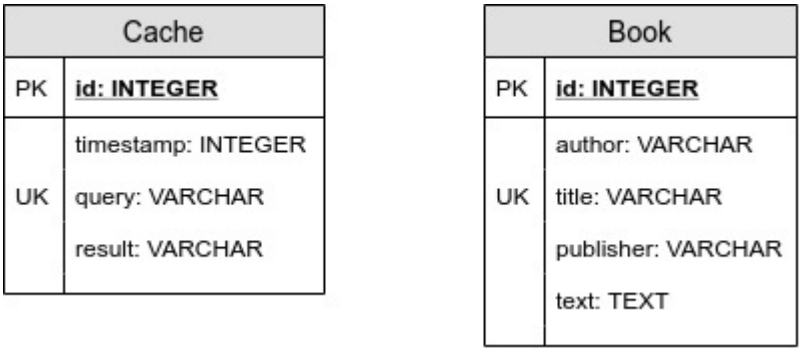


Diagrama Entitate-Relație

- Să se combine funcționalitatea de căutare cu cea de afișare într-un anumit format (HTML, JSON, raw) sub forma unui nou conector (end-point) accesibil printr-o cerere (request) HTTP de tip GET către un URL:
  http://localhost:8080/find-and-print?author=<author-name>&format=json

Teme pe acasă:

- Să se implementeze un mecanism de caching care salvează în baza de date interogarea utilizatorului (dacă aceasta nu există deja în baza de date), rezultatul căutării pe baza interogării și data apariției (timestamp) căutării. Se va face diagrama de servicii (vezi exemple la curs) și apoi cea de clase. Vezi diagramele UML de clase, precum și diagrama E-R de mai sus pentru mai multe detalii despre implementare. Microserviciul **CachingMicroservice** trebuie să comunice prin intermediul a trei cozi de mesaje una pentru trimis fisier pentru imprimanta, una pentru trimis comenzile pentru imprimanta si una pentru receptia starii imprimantei. Se poate modifica **LibraryPrinterMicroservice si crea oricate servicii se mai considera necesar dar cu justificare**. Astfel, la fiecare interogare introdusă de utilizator, se verifică întâi cache-ul. În cazul unui HIT (interogarea a mai fost introdusă anterior), dacă timestamp-ul nu este mai vechi de o oră, se ia rezultatul din cache. Dacă timestamp-ul depășește intervalul de o oră sau în cazul unui MISS, se realizează căutarea propriu-zisă, iar rezultatul este actualizat/scris în cache. Modalitatea de interacțiune (chain, orchestration sau grid)dintre microservicii se lasă la alegerea studentului.
  Observaţie: Pentru a respecta principiile de proiectare ale microserviciilor, trebuie creat un proiect nou pentru microserviciul de CacheMicroservice si orice alte mai sunt considerate necesare.

- Pornind de la exemplul rezolvat anterior, pe baza analizei datelor din zona de cache să se implementeze un mecanism (de fapt un alt microserviciu) care primește o zonă de date ca fișier sau ca obiect creează un arbore merkle și poate fi utilizat pentru căutări rapide în respectiva zonă. Acest nou microserviciu va fi utilizat pentru căutarea în cache.