

# Beauty Salon

Nume: Drăguș Andreea

Grupa: 30236

Cuprins

[Beauty Salon 1](#_Toc134692317)

[Deliverable 1 3](#_Toc134692318)

[1. Specificatia proiectului 3](#_Toc134692319)

[2. Cerinte functionale 3](#_Toc134692320)

[3. Cazuri de utilizare (usecase) 4](#_Toc134692321)

[4. Cerinte non-functionale 7](#_Toc134692322)

[5. Design Constraints 8](#_Toc134692323)

[6. Glossary 8](#_Toc134692324)

[Deliverable 2 9](#_Toc134692325)

[1. Domain Model 9](#_Toc134692326)

[1.1. Diagrama UML Domain Model 10](#_Toc134692327)

[2. Architectural Design 11](#_Toc134692328)

[2.1.Conceptual Architecture 11](#_Toc134692329)

[2.3. Component and Deployment Diagram 13](#_Toc134692330)

[2.3.1. Deployment Diagram 13](#_Toc134692331)

[2.3.2. Component Diagram 13](#_Toc134692332)

[Delivrable 3 14](#_Toc134692333)

[1. Design Model 14](#_Toc134692334)

[1.1.Dynamic Behavior 14](#_Toc134692335)

[1.1.1. Primul scenariu: Make an appointment 14](#_Toc134692336)

[1.1.1.1. Diagrama Secventiere 14](#_Toc134692337)

[1.1.1.2. Diagrama Comunicare 14](#_Toc134692338)

[1.1.2. Al doilea scenariu: Forgot Password/Modify Account 15](#_Toc134692339)

[1.1.2.1. Diagrama Secventiere 15](#_Toc134692340)

[1.1.2.2. Diagrama Comunicare 15](#_Toc134692341)

[1.2. Class Diagram 16](#_Toc134692342)

[2. Data Model 17](#_Toc134692343)

[3. System Testing 18](#_Toc134692344)

[4. Future Improvements 18](#_Toc134692345)

[5. Conclusion 19](#_Toc134692346)

[6. Bibliography 19](#_Toc134692347)

# Deliverable 1

## 1. Specificatia proiectului

Acest proiect reprezinta o poarta intermediara intre clientii si antreprenorii care au in vizor domeniul infrumusetarii. Astfel, la baza acestei lucrari sta o platforma interactiva prin intermediul careia informatiile sa fie transmise cu usurinta intre clienti si un salon de beauty.

Scopul proiectului este eficientizarea timpului si a organizarii unui salon. Astfel, toate informatiile relevante pentru un client sunt la un click distanta si se pot realiza de asemenea si programari.

## 2. Cerinte functionale

Aplicatia este conceputa astfel incat sa perceapa 3 tipuri de utilizatori: admin, client, angajat. Cele trei tipuri de actori au atat caracteristici comune precum: nume, username, parola, dar si elemente specifice: clientul are puncte de loialitate, iar angajatul are un domeniu specific in care activeaza: hair-dresser, nail-tech sau make-up artist.

Fiecare tip de utilizator dispune de functionalitati diferite:

* ***Admin:*** poate sa gestioneze baza de date si sa faca diverse interogari precum: sa adauge utilizatori de orice tip in baza de date, sa stearga utilizatori, sa adauge programari, sa stearga programari, sau sa actualizeze informatiile deja existente.
* ***Client:*** poate sa isi creeze un cont, sa isi actualizeze diferite date, sa caute angajati dupa diverse criterii precum domeniul de activitate, sa creeze o programare sau sa stearga din programarile sale. Cu fiecare programare adaugata, punctele de loialitate ale clientului se incrementeaza cu o valoare proportionala cu valoarea serviciilor din programare.
* ***Angajat:*** poate sa isi creeze cont, sa isi actualizeze detaliile contului, sa creeze sau sa stearga din propriile programari.

Fiecare actiune mentionata mai sus este insotita de validari si verificari astfel incat sa nu poata fi realizata daca nu indeplineste anumite criterii.

La adaugarea unui user, se verifica coerenta datelor: field-urile de nume, username si parola trebuie sa fie nenule si tipul de utilizator sa fie unul dintre cele 3 valide. Daca din aceasta perspectiva este totul in regula, atunci se face o verificare in baza de date pentru a ne asigura ca nu exista un alt user care sa aiba acelasi username. La finalizarea cu succes a verificarilor, userul e adaugat atat in tabelul de user cat si in tabelele aferente tipului de user.

In cazul in care o actiune nu este validata cu succes, se vor arunca exceptii specifice: InvalidUserException, InvalidClientException, InvalidEmployeeException, InvalidAppointmentException.

La update-ul unui user, se aplica din nou aceleasi cerinte mentionate mai sus, astfel incat sa ne asiguram ca nu exista update-uri inconsistente.

La stergerea unui user, daca acesta este de tipul client sau angajat, se vor sterge si toate programarile aferente lui.

Interogarile care au ca scop gasirea unui user sau a mai multora conform diferitor criterii, vor arunca exceptii specifice in cazul in care nu este niciun utilizator care sa indeplineasca cerintele: UserNotFoundException, ClientNotFoundException, EmployeeNotFoundException, AppointmentNotFoundException.

O entitate importanta a proiectului este Appointment. Aceasta este alcatuita din un client, un angajat, o data respectiv ora si o lista de servicii. Pretul total al appointment-ului este calculat ca suma tutror preturilor serviciilor incluse in lista si va fi transpus sub forma unor puncte de loialitate alocate clientului. Pentru a salva in baza de date o programare, asupra acesteia se fac multiple verificari: clientul/angajatul sa fie valizi si sa fie regasiti in tabelele aferente celor doua tipuri de entitati si in tabelul care contine toti userii, data sa nu fie in trecut sau intr-o zi nelucratoare (sambata sau duminica), timpul sa respecte urmatoarele criterii: sa fie in intervalul de lucru (08:00- 18:00) si minutele sa fie 0 sau 30. Dupa ce se realizeaza validarile mentionate, se verifica si programele clinetului si al angajatului, astfel incat sa nu existe pentru nici unul dintre ei o alta programare care sa se desfasoare la data si timpul respectiv. Daca aceste cerinte sunt satisfacute, programarea este adaugata in baza de date si in listele de programari ale celor doua entitati.

## 3. Cazuri de utilizare (usecase)

Use case 1: Register

Level: User-goal

Primary Actor: User

Main success scenario: inregistrare cu success, salvare in baza de date a noului user

Use case 2: Login

Level: Subfunction

Primary Actor: User

Main success scenario: logare cu success, utilizatorul a fost identificat in baza de date

Use case 3: Update account

Level: Subfunction

Primary Actor: User

Main success scenario: update cu success, salvare in baza de date a modificarilor

Use case 4: Delete account

Level: Subfunction

Primary Actor: Admin

Main success scenario: delete cu success, stergerea contului din baza de date si a programarilor aferente acestuia

Use case 5: Create appointment

Level: User-goal

Primary Actor: Client, Angajat

Main success scenario: programarea este valida si salvata in baza de date

Use case 6: Update Appointment

Level: Subfunction

Primary Actor: Client, Angajat

Main success scenario: update cu success, salvarea in baza de date a modificarilor

Use case 7: Delete appointment

Level: Subfunction

Primary Actor: Admin

Main success scenario: delete cu success, stergerea programarii din baza de date si din listele de programari a clientului si a angajatului

Use case 7: Find appointments by criteria

Level: User-goal

Primary Actor: User

Main success scenario: gasirea cu success a unei sau a mai multor programari care sa respecte criteriile de cautare

Use case 8: Find users by criteria

Level: Subgoal

Primary Actor: Admin

Main success scenario: gasirea cu success a unui sau a mai multor utilizatori care sa respecte criteriile de cautare

Use case 9: Find clients by criteria

Level: Subgoal

Primary Actor: Admin

Main success scenario: gasirea cu success a unui sau a mai multor clienti care sa respecte criteriile de cautare

Use case 10: Find employees by criteria

Level: Subgoal

Primary Actor: Admin

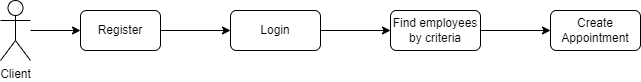
Main success scenario: gasirea cu success a unui sau a mai multor angajati care sa respecte criteriile de cautare

Use case 11: Beauty Salon

Level: High-Level

Primary Actor: Client

Main success scenario: parcurgerea cu succes a tuturor pasilor necesari realizarii cu success a unei programari in cadrul aplicatiei



## 4. Cerinte non-functionale

1. Fiabilitate

Sistemul construit este fiabil din mai multe puncte de vedere. Aparitia unei erori critice in cadrul aplicatiei are o probabilitate scazuta intrucat aceasta are la baza sa un schelet complex de validari care arunca exceptii controlate astfel incat sa blocheze orice interactiune nepermisa a utilizatorului cu sistemul. Au fost testate si cuprinse in validari un procent considerabil de situatii exceptionale care pot aparea in cadrul utilizarii aplicatiei, fapt care reduce semnificativ probabilitatea de aparitie a unei erori critice. In cazul in care se intampla acest lucru, timpul de redresare nu ar trebui sa depaseasca 10 minute, intrucat aplicatia are o structura clara si robusta, iar timpul aferent identificarii sursei problemei este redus considerabil de acest aspect.

De asemenea, a fost realizata o testare riguroasa a intregului sistem si a fiecarui caz de utilizare, atat in cazul pozitiv cat si in cazul negativ, fiind implementate un total de 65 de teste JUnit, care sa asigure functionalitatea corecta a sistemului.

2. Utilizabilitate

Din perspectiva utilizatorului, aplicatia va avea un aspect simplu, astfel incat utilizarea acesteia sa aiba un grad minimalist de dificultate. Optiunile pe care fiecare tip de utilizator le are sunt clar stabilite, iar modul de interfatare ale acestora va fi unul user-friendly.

3. Portabilitate si compatibilitate

Aplicatia este portabila, putand fii rulata pe diverse dispozitive cu diverse sisteme de operare. Cu toate acestea, exista restrictii cu privire la versiunile de java si spring-boot folosite: 19 si respectiv 3.0.4. Un alt aspect important il reprezinta dependintele care trebuie instalate inainte de rularea proiectului, fiind vitala integrarea lor.

4. Scalabilitate

Din punctul de vedere al scalabilitatii, aplicatia poate fi extinsa cu necesitate minima de a modifica restul componentelor, intrucat exista un grad redus de cuplare intre clase, pachete si respectiv layere. Scheletul proiectului este proiectat intr-o maniera puternic coeziva, astfel incat toate clasele au functionalitati targetate si foarte clar definite.

## 5. Design Constraints

Din punct de vedere arhitectural, proiectul este structurat dupa modelul MVC, astfel incat sa existe o departajare cat mai clara intre layerele aplicatiei. Astfel, exista un fir logic al traversarii datelor, pornind de la legatura cu baza de date reprezentata de layer-ul de repository, continuand cu layerul de service si ajungand la cel de controller. De asemenea, exista definite pachete aditionale aferente validarilor si a exceptiilor definite.

Din punct de vedere al limbajului de programare, proiectul este scris in Java si Spring Boot, versiunile 19 si 3.0.4.

Au fost folosite diverse librarii aditionale pentru a reduce cat mai mult din complexitatea si masa codului. Astfel, pentru generarea metodelor accesoare/mutatoare si a unor constructori au fost folosite adnotari specifice Lombok. De asemenea, pentru a genera in mod automat tabelele aferente aplicatiei s-au folosit adnotari specifice Hibernate, cu ajutorul carora s-au definit si relatiile dintre tabele. In cadrul testelor, au fost folosite diverse elemente care tin de libraria Mockito.

# 6. Glossary

***MVC*** este un model în proiectarea software-ului utilizat în mod obișnuit pentru implementarea interfețelor utilizator, a datelor și a logicii de control. Subliniază o separare între logica de business a software-ului și afișaj. Această „separare a preocupărilor” asigură o mai bună diviziune a proiectului și o întreținere îmbunătățită.

***Repository pattern*** separă logica de acces la date și o mapează la entitățile din logica de business. Realizează logica de acces la date mapata la entitatile din domeniu. În modelul Repository, entitățile de domeniu, logica de acces la date și logica de afaceri comunica folosind interfețe.

***Service pattern*** este un model arhitectural, aplicat în cadrul paradigmei de proiectare orientată către servicii, care urmărește organizarea serviciilor într-un set de layere logice. Serviciile care sunt clasificate într-un anumit nivel partajează funcționalitatea.

***Nivelul Controller*** controlează domeniul de aplicare a tranzacției și gestionează informațiile legate de sesiune pentru cerere. Controlerul trimite mai întâi la o comandă și apoi apelează logica corespunzătoare de procesare a vizualizării pentru a da răspunsul.

# Deliverable 2

# 1. Domain Model

Modelul de domeniul al aplicatiei vizeaza functionalitatile unei pagini web destinate programarilor unui salon de infrumusetare.

Clasa User are un caracter generalizat, representant toti utilizatorii aplicatiei. Variabilele instanta specifice acestei clase sunt: id, nume, username, parola. Pentru distinctia intre tipurile de utilizatori s-a folosit o enumeratie denumita “userType” si poate avea una dintre valorile: “ADMIN”, “EMPLOYEE”, “CLIENT”.

Clasele Employee si Client reprezinta subspecializari ale clasei User, completand atributele si comportamentul acesteia cu diverse capabilitati specifice fiecarui tip de utilizator. De exemplu, atat clientii cat si angajatii au asocitate cate o lista de programari.

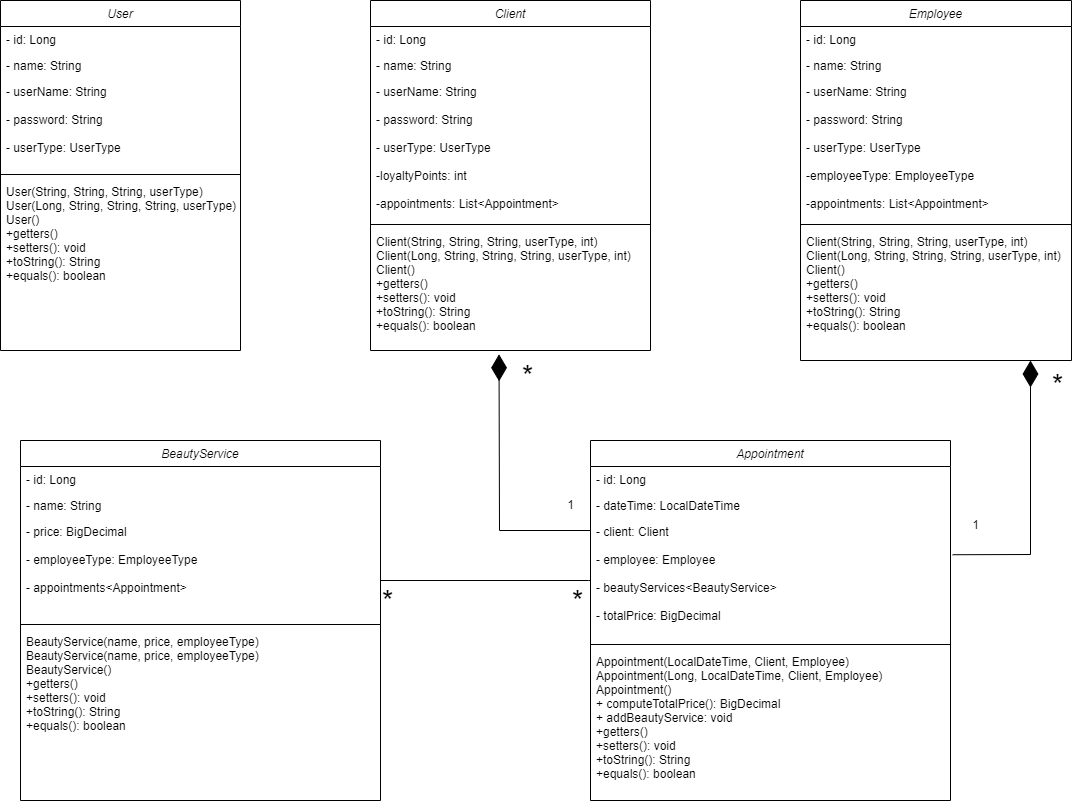
Clientii mai au si o variabila instanta denumita „loyaltyPoints”, in care se cumuleaza pretul total al programarilor la care participa si care atunci cand ajunge la o valoare stabilita se concretizeaza intr-un card cadou care poate fi folosit la urmatoarele programari in vederea obtinerii unui discount.

Angajatii au o variabila instanta de tipul enumeratie numita „employeeType” care reprezinta sectorul de activitate si poate avea una dintre urmatoarele 3 valori: „MAKEUP\_ARTIST”, „HAIR\_DRESSER” sau „NAIL\_TECH”.

O alta clasa care tine de modelul de domeniu este cea de BeautyService, aceasta reprezentand un anume serviciu prestat de angajat in cadrul unei programari. Aceasta are ca si variabile instanta denumirea, pretul, si domeniul de activitate caruia se preteaza, adica un employeeType.

Esenta aplicatiei este regasita in cadrul clasei Appointment. Aceasta are urmatoarele atribute: un id, o data si ora, un client, un angajat, o lista de servicii si un pret total. Pretul total al programarii este reprezentat de suma tuturor serviciilor de beauty regasite in lista mentionata.

# 1.1. Diagrama UML Domain Model



# 2. Architectural Design

# 2.1.Conceptual Architecture

Din punctul de vedere al arhitecturii alese, aplicatia este conceputa pe baza “Layered Architecture Design Pattern” astfel incat flowul datelor in cadrul aplicatiei sa fie unul cat de clar si structurat posibil.

Informatiile pornesc din *layerul de persistenta* care face legatura cu baza de date, in cadrul caruia sunt implementate repository-urile. Acestea extind JpaRepostiory, iar cu ajutorul frameworkului Spring interogarile si operatiile care vizeaza interactiunea cu baza de date sunt facilitate. Astfel, pentru fiecare dintre clasele de domeniu au fost create interfete de repository in cadrul carora sunt definite diverse interogari specifice fiecarei entitati. Toate interogarile returneaza tipul Optional<Entity> care va fi preluat in urmatorul layer ierarhic.

*Layerul de business* contine cate un service aferent fiecarui element din layerul de persistenta. In cadrul fiecaruia, sunt tratate toate interogarile definite in repository. Astfel, se verifica daca entitatile de tip Optional sunt prezente, adica au fost gasite datele in baza de date, iar in caz contrar se arunca exceptii specifice precum: UserNotFoundException, EmployeeNotFoundException, ClientNotFoundException, AppointmentNotFoundException si BeautyService not found exception.

De asemenea, tot in cadrul business layer este situate si pachetul de DTO, care contine cate o clasa specifica de dao fiecarei entitati din domain model. Acestea sunt folosite in cadrul controlerelor pentru a interactiona cu interfata grafica, respective pentru a primi sau transmite diverse obiecte.

Business Layer contine de altfel si pachetul de model, care contine toate clasele specifice entitatilor din domain model mentionate anterior.

*Layerul de prezentare* contine controllere definite. Acestea reprezinta calea de comunicare dintre back-end si front-end. In cadrul fiecarui controller sunt definite metode specifice de interactiune cu utilizatorul aplicatiei. Acestea preiau metodele aferente din serviceuri.

Fiecare metoda din controller are un tip clar definit: GET, POST sau PUT, respectiv un mapping unic prin care sa poata fi identificate prin intermediul unui HttpRequest. In cazul in care metoda are parametrii simpli, de tip String, Long, Enum sau DateTime, adnotarea folosita este RequestParam. In cazul in care au parametrii de tip obiect, am folosit adnotarea RequestBody.

In cazul metodelor de GET, aferente practic interogarilor, se apeleaza metodele din layerul de service iar in cazul in care acestea au comportamentul asteptat si nu arunca exceptii, se returneaza sub forma de ResponseEntity catre pagina web continutul cerut si un HttpStatus “OK”, care mentioneaza ca totul a decurs bine. In caz contrar, se arunca o exceptie de tip NotFound care va fi tratata in maniera orizontala de ControllerAdvice.

In cazul metodelor de POST, aferente salvarii in baza de date, se apeleaza metodele de add din service-uri. Daca datele care se doresc a fi salvate sunt valide, atunci metoda este finalizata cu success, returnandu-se un ResponseEntity cu continutul salvat si un HttpStatus “CREATED”. In cazul in care datele nu sunt validate pentru salvare, se va arunca o exceptie de tipul InvalidException care va fi tratata de asemenea in Controller Advice.

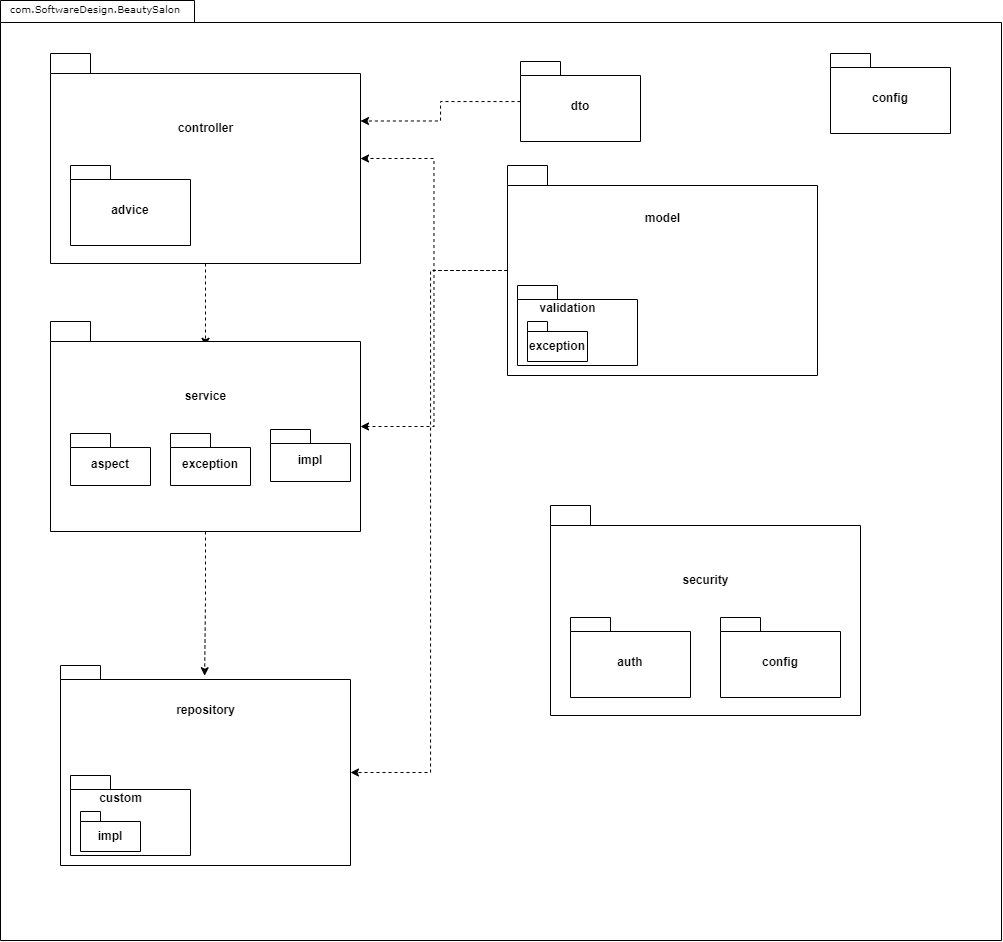
In maniera similara cu metodele de POST sunt tratate si cele de PUT, aferente actualizarii informatiilor din baza de date.

ControllerAdvice este o clasa care are rolul de ExceptionHandler. Aceasta prinde toate exceptiile aruncate in Controllere si le transmite paginii web sub forma de ResposeEntity<Map<TimeStamp, ExceptionMessage>>, astfel incat sa poata fi tratate mai usor in partea de front-end si afisate sub forma unor alerte.

De asemenea, in layerul de prezentare intra si partea de frontend care a fost implementata in Angular. Utilizatorul poate realiza diverse actiuni: poate naviga intre paginile web, se poate inregistra, se poate loga, iar in functie de tipul de user se deschid HomePage-uri specifice. Se pot realiza diverse operatii pe datele existente.

Aditional, a fost implementat un layer de Securitate, care se axeaza pe partea de autentificare si inregistrare. Astfel, pentru a accesa oricare dintre metodele din controller userul trebuie sa fie autentificat. Pe partea de inregistrare, se realizeaza tokenizare si encriptia parolei.

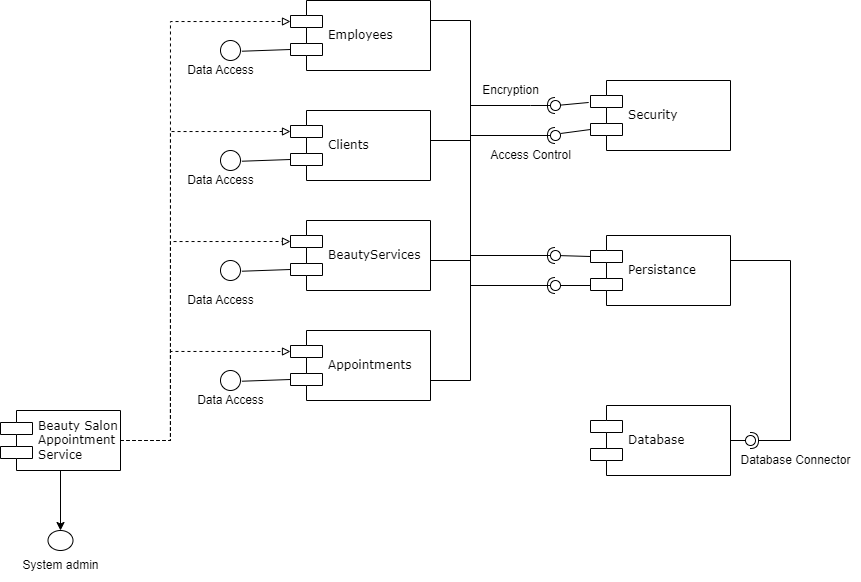
2.2. Package Design

**

# 2.3. Component and Deployment Diagram

# 2.3.1. Deployment Diagram

# 2.3.2. Component Diagram



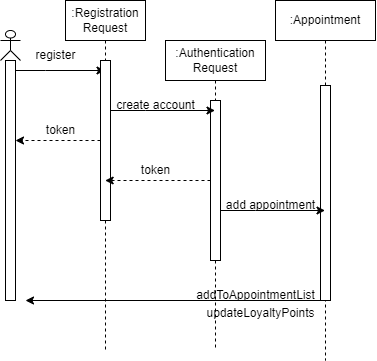
# Deliverable 3

## 1. Design Model

## 1.1.Dynamic Behavior

## 1.1.1. Primul scenariu: Make an appointment

## 1.1.1.1. Diagrama Secventiere



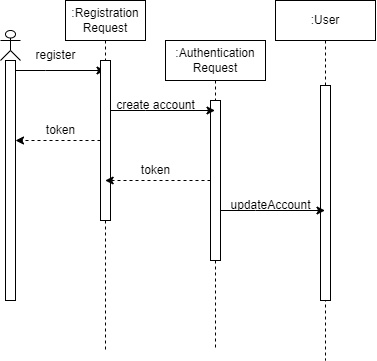
## 1.1.1.2. Diagrama Comunicare

O imagine care conține captură de ecran, text, Font, negru

Descriere generată automat

## 1.1.2. Al doilea scenariu: Forgot Password/Modify Account

## 1.1.2.1. Diagrama Secventiere

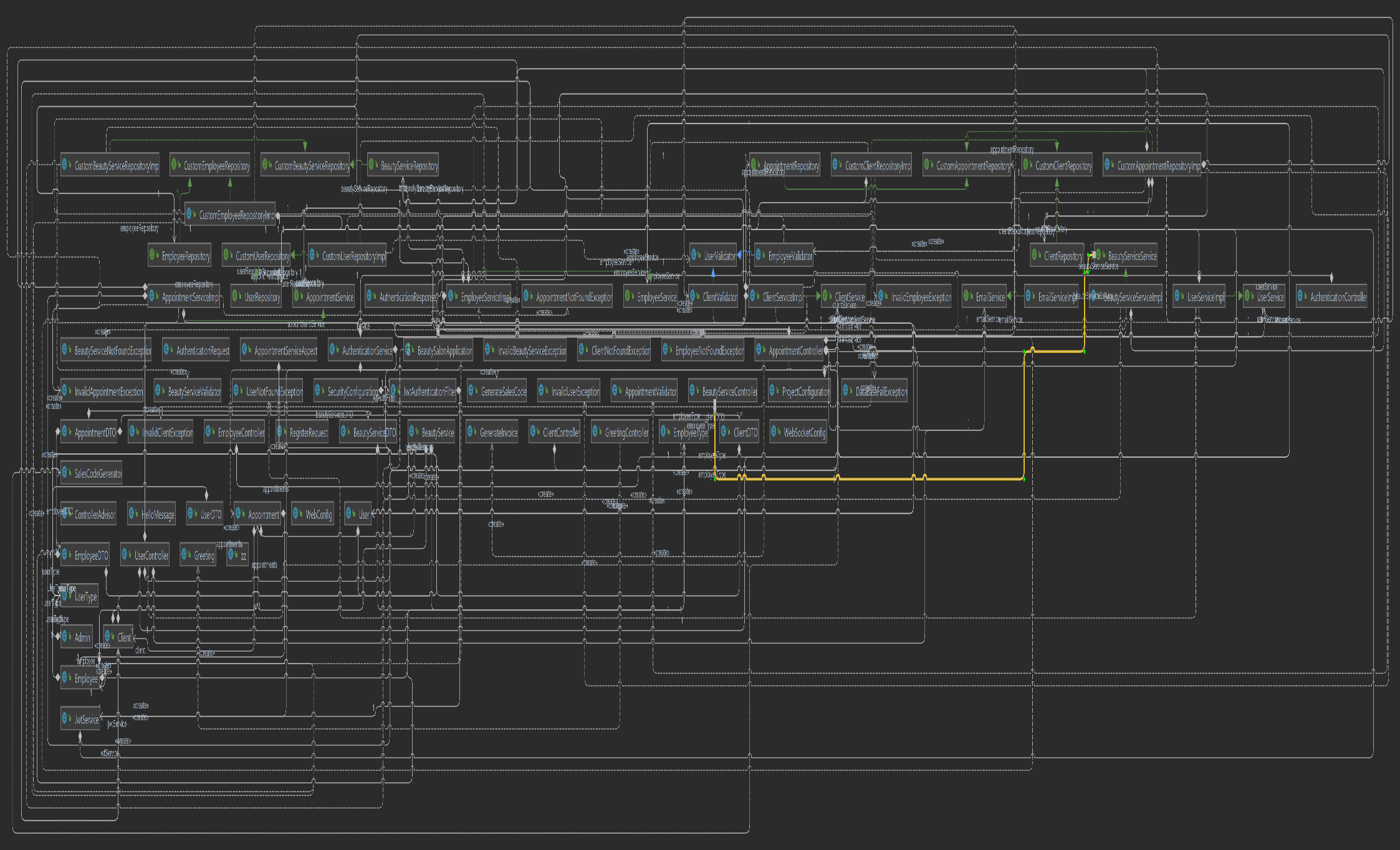


## 1.1.2.2. Diagrama Comunicare



# 

# 1.2. Class Diagram

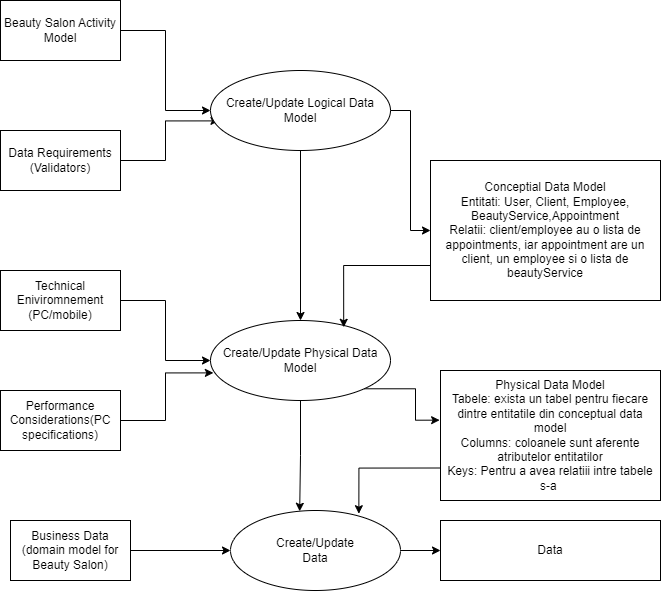
**

GoF Patterns:

1. Composite design pattern: In cadrul acestui proiect, exista mai multe cazuri in care a fost utilizat acest pattern. Cel mai relevant dintre ele se refera la clasa Appointment, care are in compozitia sa un obiect de tipul Client, un obiect ded tipul Employee si o lista de beauty services. Acest pattern este intarit prin faptul ca atunci cand se sterge un employee sau un client, se sterg si toate appointmenturile aferente acestuia.

2. Builder pattern: Acest pattern este regasit in mod special la nivelul securitatii, acolo unde obiectele complexe sunt construite pas cu pas. Implementarea este realizata prin utilizarea adnotarii “@Builder” oferite de Lombok si mai apoi constructia obiectelor prin .build().Prin utilizarea Builder Pattern, putem construi obiecte complexe într-un mod clar și flexibil. Ne permite să configurăm diferite atribute în orice ordine și să evităm constructorii suprasolicitați cu un număr mare de parametri. În plus, ne oferă posibilitatea de a extinde și personaliza procesul de construcție fără a afecta clasa de bază.

# 2. Data Model

**

# 3. System Testing

Sistemul a fost testat prin un numar total de 65 de teste junit, care cuprind atat cazuri uzuale cat si cazuri exceptionale. Fiecare serviciu a fost testat in mod individual intr-o clasa special destinata. In cadrul claselor de test aferente fiecarui serviciu, a fost testata fiecare metoda, atat pentru cazurile finalizate cu success cat si pentru cazurile finalizate cu eroare.

Astfel, cateva exemple de cazuri de test relevante sunt: adaugarea unei programari in interval orar invalid( 21:00- 07:00), adaugarea unei programari pentru un client inexistent in baza de date, stergerea unui anagajat, etc.

Pentru testele aferente serviciilor, am folosit Mock-uri aferente repositoriurilor utilizate in teste.

# 4. Future Improvements

1. Implementarea sistemului de evaluare și recenzii: permite utilizatorilor să lase evaluări și recenzii pentru serviciile și specialiștii salonului de înfrumusețare. Exista posibilitatea de a adăuga un sistem de evaluare cu rating și comentarii, astfel încât utilizatorii să poată împărtăși experiențele lor și să ofere feedback. Aceste evaluări pot fi afișate publicului și pot ajuta la construirea încrederii în serviciile salonului.

2. Implementarea unui sistem de recomandări personalizate: utilizarea algoritmilor de recomandare pentru a sugera servicii sau specialiști în funcție de preferințele și istoricul clienților. Aceasta poate contribui la creșterea satisfacției clienților și la promovarea serviciilor suplimentare.

3. Integrarea cu un sistem de contabilitate: se poate integra aplicația cu un sistem de contabilitate pentru a gestiona aspectele financiare ale salonului. Acest lucru va ajuta la urmărirea veniturilor și a cheltuielilor, la gestiunea rapoartelor financiare și va sigura că operațiile financiare sunt eficiente și precise.

# 5. Conclusion

In concluzie, acest proiect este unul complex, care are diverse flow-uri de utilizare pentru fiecare tip de user. Este un proiect care ofera diverse verificari si asigura concordanta datelor.

Etapele de proiectare si implementare au avut un grad de dificultate considerabil, intrucat a fost necesara analizarea diversilor factori care pot influenta un anumit flow, insa consider ca intreg efortul s-a concretizat intr-un rezultat satisfacator, care imbina armonios logica complexa din backend cu aspectul impunator al front-endului.

# 6. Bibliography

<https://www.baeldung.com/spring-boot>

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#legal>

<https://docs.spring.io/spring-security/reference/index.html>

<https://www.javaguides.net/2019/06/spring-boot-angular-8-websocket-example-tutorial.html>