### UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de disertație**

**Sistem de gestionare a biletelor de avion proiectat cu tehnologia Java Spring**

### Absolvent

### Birta Alexandru

**Coordonator științific**

**Lect. Dr. Ciprian Ionuț Păduraru**

**București, iunie 2023**

#### Rezumat

În această teză de disertație, este studiat un sistem de gestionare a biletelor de avion cu accent pe proiectarea și implementarea unei soluții de achiziționare a biletelor de avion eficiente și ușor de utilizat. Teza tratează problemele existente ale sistemelor de gestionare a biletelor, cum ar fi de exemplu: metode îngreunate de rezervare și transparență insuficientă. Cercetarea oferă un design complet de sistem, utilizând tehnologii backend precum Java Spring și tehnologii frontend precum Angular. Procesul de dezvoltare urmează un model agil care asigură îmbunătățiri iterative. Rezultatele arată că sistemul de management al biletelor de avion creat simplifică semnificativ procesele, îmbunătățește experiența clienților și crește eficiența generală a unei companii ce poate utiliza produsul.

#### Abstract

Această teză de disertație prezintă proiectarea, dezvoltarea și analiza aplicației Airlliant, un sistem de gestionare a biletelor de avion. Airlliant își propune să simplifice procesele de gestionare a biletelor de avion și să îmbunătățească experiența utilizatorului în industria aviației.

Teza începe cu o analiză aprofundată a provocărilor existente cu care se confruntă sistemele de gestionare a biletelor de avion. Aceste provocări includ procese complexe de rezervare, transparență limitată și comunicare ineficientă între companiile aeriene și clienți.

Faza de proiectare se concentrează pe arhitectura sistemului Airlliant. Backend-ul proiectat cu tehnologii Java Spring se ocupă de funcțiile de bază de gestionare a biletelor, inclusiv disponibilitatea zborurilor, rezervarea și emiterea de bilete. Interfața realizată cu tehnologia Angular este o interfață ușor de utilizat, care permite clienților să caute zboruri, să vadă opțiunile de preț, să facă rezervări și să își gestioneze eficient rezervările.

În general, Airlliant oferă o soluție robustă și ușor de utilizat de gestionare a biletelor de avion. Integrarea tehnologiilor Java Spring și Angular permite îmbunătățirea eficienței, satisfacția clienților și performanța generală a produsului. Airlliant servește ca un instrument valoros care simplifică procesele de gestionare a biletelor de avion și îmbunătățește experiența de călătorie atât pentru clienți, cât și pentru companiile aeriene.

1. [Introducere 5](#_Toc136454327)

[1.1 Prezentare generală 5](#_Toc136454328)

[1.2 Scopul și motivația alegerii temei 5](#_Toc136454329)

[1.3 Structura lucrării 6](#_Toc136454330)

1. [Prezentarea tehnologiilor utilizate 7](#_Toc136454331)

[2.1 Framework-ul Java Spring 7](#_Toc136454332)

[2.2 Spring Boot 8](#_Toc136454333)

[2.3 Sistemul de baze de date relaționale PostgreSQL 9](#_Toc136454334)

[2.4 Spring Data JPA și Hibernate ORM 10](#_Toc136454335)

[2.5 Biblioteca de programare de declanșatori Quartz 11](#_Toc136454336)

[2.6 Framework-ul Angular 12](#_Toc136454337)

[2.7 Keycloak 13](#_Toc136454338)

1. [Arhitectura aplicației 15](#_Toc136454339)

[3.1 Arhitectură bazată pe microservicii 15](#_Toc136454340)

[3.1.1 Modelul arhitectural al aplicației 16](#_Toc136454341)

1. [Preliminarii 19](#_Toc136454342)
2. [Concluzii 20](#_Toc136454343)
3. [Bibliografie 21](#_Toc136454344)

# Introducere

## Prezentare generală

Sistemul de gestionare a biletelor de avion prezentat în această lucrare de disertație are ca scop eficientizarea procesului de alegere a unui bilet de avion prin intermediul unei interfețe grafice pentru utilizatori ușor de utilizat. Aplicația Airlliant face acest lucru posibil prin intermediul tehnologiilor Java Spring și Angular, iar cu acestea se poate integra cu interfețe de programare a aplicațiilor externe de unde își poate aproviziona date despre zboruri accesibile din aplicație pentru a realiza un sistem eficient și automatizat.

Airlliant oferă o serie de caracteristici pentru a îmbunătăți procesul de emitere a biletelor. Acesta poate include opțiuni de căutare și filtrare a zborurilor bazate pe diverse criterii, cum ar fi data, destinația, prețul și dacă zborul este doar dus sau dus-întors. Clienții pot vedea disponibilitatea zborurilor în timp real, pot selecta locurile preferate și pot face rezervări. Sistemul poate oferi, de asemenea, și notificări pe email înainte cu 3 zile de decolarea zborului pentru a aduce aminte clienților de data când va decola avionul pentru a nu rata șansa de a se îmbarca la momentul corespunzător.

Aplicația este bazată pe modelul arhitectural de microservicii, unde aceasta are ca și componente trei module precum: AirlliantCore, AirlliantQuartz și AirlliantFrontend.

Modulul AirlliantCore se ocupă cu expunerea interfeței de programare a aplicației folosind protocolul REST (REpresentational State Transfer). Acest modul face posibilă comunicarea între microserviciile aplicației pentru a gestiona date din baza de date în scopul de a facilita funcționalitățile aplicației.

AirlliantQuartz este microserviciul ce se ocupă cu programarea notificărilor pentru zboruri pentru toți utilizatorii și pentru a crea zborurile în baza de date.

Modulul AirlliantFrontend are ca rol redarea interfeței grafice pentru utilizatori folosind tehnologii web precum HTML și CSS, care mai apoi sunt îmbogățite cu ajutorul

framework-ului Angular care ajută la un design reactiv în reprezentarea paginii web și care facilitează design-ul de tip SPA (single page application) în care aplicația folosește o singură pagină web pentru a reda conținut diferit, față de a utiliza pagini separate pentru conținut diferit.

În plus față de acestea arhitectura aplicației integrează în acest ecosistem de microservicii încă două microservicii: un microserviciu de bază de date PostgreSQL și microserviciul care se ocupă cu autentificarea în modulul de frontend pe nume Keycloak.

În general, sistemul de gestionare a biletelor de avion prezentat în disertație combină tehnologii avansate, arhitectură robustă și design centrat pe utilizator pentru a revoluționa procesele de emitere a biletelor în industria aviației. Prin abordarea provocărilor existente, îmbunătățirea eficienței operaționale și sporirea satisfacției clienților, sistemul își propune să ofere o experiență perfectă și plăcută atât clienților, cât și companiilor aeriene.

## Scopul și motivația alegerii temei

Scopul acestei disertații este de a proiecta, dezvolta și evalua un sistem de gestionare a biletelor de avion care ia în considerare limitările și provocările cu care se confruntă sistemele existente în industrie. Accentul este pus pe eficientizarea proceselor de emitere a biletelor, îmbunătățirea experienței utilizatorului și îmbunătățirea eficienței operaționale.

Motivația pentru alegerea acestei dizertații rezultă din recunoașterea deficiențelor sistemelor actuale de gestionare a biletelor din industria aviației. Procesele complexe de rezervare, transparența limitată și comunicarea ineficientă între companiile aeriene și clienți au fost identificate ca principalele provocări. Aceste probleme au ca rezultat nemulțumirea clienților, ineficiențe operaționale și oportunități de afaceri omise.

Scopul este de a dezvolta un sistem cuprinzător și ușor de utilizat, care să abordeze în mod specific aceste provocări. Folosind capacitățile backend-ului bazat pe tehnologiile Java Spring și frontend-ului realizat prin framework-ul Angular, urmează să fie creată o soluție care simplifică căutarea zborurilor, gestionarea rezervărilor și procesele de emitere a biletelor. Scopul sistemului este de a crește transparența, de a oferi informații de zbor în timp real și de a oferi clienților servicii personalizate. În plus, companiile aeriene vor primi funcții îmbunătățite de control și comunicare care le vor permite să gestioneze eficient disponibilitatea zborurilor, să actualizeze informațiile despre preț și să interacționeze cu clienții.

## Structura lucrării

# Prezentarea tehnologiilor utilizate

## Framework-ul Java Spring

Framework-ul Java Spring este o tehnologie larg utilizată și versatilă folosită în dezvoltarea de aplicații de clasă enterprise. Acesta oferă un set cuprinzător de instrumente și biblioteci care facilitează implementarea structurată și eficientă a funcționalităților dorite.

Una dintre funcționalitățile de bază ale framework-ului Java Spring este Inversion of Control (IoC) și Dependency Injection (DI). Această caracteristică permite cuplarea liberă între componentele aplicației, permițând framework-ului să gestioneze instanțiarea și configurarea obiectelor. Prin IoC și DI, dezvoltatorii pot defini și gestiona cu ușurință dependențele dintre diferite componente, rezultând cod modular și care poate fi întreținut.

O altă funcționalitate importantă oferită de framework-ul Java Spring este programarea orientată pe aspecte (AOP). Cu AOP, dezvoltatorii pot separa preocupările generale, cum ar fi jurnalul, gestionarea tranzacțiilor și securitatea, de logica de bază a aplicației. Prin încapsularea acestor preocupări în aspecte reutilizabile, framework-ul permite un cod mai curat și mai modular, promovând o mai bună organizare și mentenanță a codului.

Framework-ul Java Spring oferă, de asemenea, suport solid pentru dezvoltarea web, inclusiv implementarea serviciilor web și a API-urilor RESTful care sunt bazate pe protocolul REST. Oferă instrumente puternice pentru procesarea cererilor, rutarea cererii/răspunsului și serializare, facilitând dezvoltarea de aplicații web scalabile și eficiente. În plus, framework-ul sprijină dezvoltarea de interfețe de utilizator bazate pe web prin integrarea cu tehnologii frontend populare, precum Angular, React sau VueJS.

Persistența datelor este o altă funcționalitate esențială oferită de framework-ul Java Spring. Oferă integrare perfectă cu diverse tehnologii de baze de date, inclusiv baze de date relaționale precum MySQL, PostgreSQL și Oracle și baze de date NoSQL precum MongoDB. Framework-ul simplifică operațiunile bazei de date prin capabilitățile sale de mapare obiect-relațională (ORM), permițând dezvoltatorilor să definească modele de date și să interacționeze cu bazele de date prin intermediul API-urilor de nivel înalt.

Totodată Framework-ul Java Spring oferă suport robust pentru gestionarea tranzacțiilor. Acesta oferă mecanisme declarative de gestionare a tranzacțiilor care permit dezvoltatorilor să definească limitele tranzacțiilor pentru logica specifică de aplicație. Framework-ul se ocupă de inițierea tranzacțiilor, commit și rollback, asigurând integritatea și coerența datelor în cadrul operațiunilor în mai mulți pași.

Securitatea este un aspect critic al aplicațiilor de tip enterprise, iar framework-ul Java Spring oferă caracteristici cuprinzătoare de securitate. Oferă mecanisme de autentificare și autorizare care permit dezvoltatorilor să-și protejeze aplicațiile împotriva accesului neautorizat. Framework-ul acceptă diverse metode de autentificare, inclusiv nume de utilizator/parolă, bazate pe token sau integrare cu furnizori externi de identitate.

În plus, framework-ul Java Spring încurajează dezvoltarea bazată pe teste și oferă instrumente puternice de testare. Permite integrarea cu cadre de testare populare precum JUnit și Mockito, permițând dezvoltatorilor să scrie teste unitare și să ruleze teste automate. Framework-ul acceptă, de asemenea, testarea integrării și oferă instrumente pentru simularea și mocking-ul dependențelor.

## Spring Boot

Spring Boot este un framework Java utilizat pe scară largă care simplifică dezvoltarea de aplicații de tip enterprise robuste și scalabile. Oferă un set cuprinzător de funcționalități care permit dezvoltatorilor să creeze aplicații pregătite pentru producție, cu configurație minimă și cod standard.

Una dintre caracteristicile fundamentale ale Spring Boot este capacitatea sa de a automatiza configurarea aplicației. Urmează principiul convenției înainte de configurare, permițând dezvoltatorilor să se concentreze mai degrabă pe logica aplicației decât pe detaliile complicate de configurare. Spring Boot configurează automat diverse componente și dependențe bazate pe valori implicite sensibile și scanări ale căilor de clasă, reducând nevoia de configurare explicită.

Spring Boot oferă capabilități puternice de injectare a dependenței (DI) prin integrarea cu framework-ul Spring. DI permite cuplarea liberă între componente, promovând modularitatea, testabilitatea și mentenabilitatea. Mecanismul DI al Spring Boot gestionează instanțiarea și conectarea componentelor și permite dezvoltatorilor să definească relații între diferite părți ale aplicației.

O altă caracteristică cheie a Spring Boot este suportul pentru server încorporat. Include containere de servlet încorporate, cum ar fi Tomcat, Jetty sau Undertow, care elimină necesitatea de a implementa aplicații pe servere externe. Acest lucru simplifică implementarea și reduce costul general de gestionare a configurațiilor serverului, făcând livrarea aplicațiilor mai simplă.

Spring Boot oferă suport robust pentru construirea de API-uri RESTful. Include framework-ul Spring Web MVC care simplifică dezvoltarea punctelor terminale REST și procesarea cererilor. Adnotări precum @RestController și @RequestMapping permit dezvoltatorilor să definească rapid API-urile RESTful. Integrarea Spring Boot cu bibliotecile Jackson sau Gson permite serializarea și deserializarea automată a datelor JSON, simplificând și mai mult dezvoltarea de API.

În plus, framework-ul oferă suport extins pentru accesul la date prin integrarea cu Spring Data. Spring Boot oferă o integrare perfectă cu diverse baze de date relaționale și NoSQL, inclusiv MySQL, PostgreSQL, MongoDB și Redis. Dezvoltatorii pot folosi API-urile de acces la date simplificate ale Spring Data și generarea automată a depozitelor pentru a interacționa eficient cu bazele de date.

## Sistemul de baze de date relaționale PostgreSQL

PostgreSQL este un sistem de gestionare a bazelor de date relaționale (RDBMS) puternic și bogat în funcții, care oferă o gamă largă de metode pentru stocarea, gestionarea și manipularea datelor structurate. Este extrem de fiabil, scalabil și extensibil, ceea ce îl face o alegere populară pentru diverse aplicații de tip enterprise.

Una dintre capacitățile fundamentale ale PostgreSQL este capacitatea sa de a stoca și a prelua eficient date structurate. Acceptă standardul Structured Query Language (SQL) și oferă un set bogat de tipuri de date, inclusiv tipuri de date numerice, caractere, dată/oră și spațiale. Acest lucru le permite dezvoltatorilor să definească modele complexe de date și să gestioneze eficient diverse cerințe de date.

Integritatea datelor este un aspect critic al oricărui sistem de baze de date, iar PostgreSQL oferă mai multe mecanisme pentru a asigura consistența și validitatea datelor. Acesta acceptă aplicarea constrângerilor de integritate, inclusiv chei primare, chei unice, chei externe și constrângeri de audit. Aceste constrângeri ajută la menținerea acurateței datelor, la prevenirea dublării datelor și la stabilirea relațiilor între diferite tabele.

PostgreSQL oferă, de asemenea, capabilități puternice de interogare pentru preluarea și manipularea datelor. Acceptă o gamă largă de operațiuni SQL, inclusiv alăturări, agregări, sortări, filtre și subinterogări. Optimizatorul de interogări, parte integrată a PostgreSQL, analizează planurile de execuție a interogărilor și alege cele mai eficiente strategii de recuperare a datelor pentru a asigura performanță optimă.

## Spring Data JPA și Hibernate ORM

Spring Data JPA și Hibernate sunt tehnologii de mapare relațională obiect (ORM) și de acces la date utilizate pe scară largă în ecosistemul Java. Împreună, oferă un set cuprinzător de funcționalități care simplifică dezvoltarea straturilor de acces la date în aplicațiile de tip enterprise.

Una dintre caracteristicile fundamentale ale Spring Data JPA este suportul pentru Java Persistence API (JPA). JPA este o specificație standardizată care definește un set de interfețe și adnotări pentru maparea obiectelor Java la tabele de baze de date relaționale. Spring Data JPA se bazează pe JPA și oferă abstracții și utilități suplimentare pentru a eficientiza operațiunile de acces la date.

Spring Data JPA extrage o mare parte din codul standard asociat în mod normal cu accesul la date. Oferă interfețe de depozit care definesc un set de operațiuni CRUD (Creare, Read, Update, Delete) asupra entităților. Dezvoltatorii pot extinde aceste interfețe și pot defini interogări personalizate folosind o combinație de convenții de denumire a metodelor și adnotări de interogare JPA. Acest lucru reduce cantitatea de cod SQL manual și simplifică implementarea logicii de acces la date.

Hibernate, pe de altă parte, este un framework ORM popular care implementează specificația JPA. Acționează ca implementare de bază pentru Spring Data JPA și oferă infrastructura necesară pentru maparea obiectelor Java la tabelele de baze de date relaționale. Hibernate se ocupă de sarcini precum maparea obiect-relațională, generarea SQL și interacțiunea cu bazele de date.

Una dintre caracteristicile cheie ale Hibernate este capacitatea sa de a mapa entități Java la tabelele bazei de date. Dezvoltatorii pot adnota clase Java cu adnotări Hibernate, cum ar fi @Entity, @Table și @Column pentru a defini metadatele de mapare. Hibernate traduce automat aceste adnotări în tabele și coloane adecvate ale bazei de date, permițând o comunicare fără întreruperi între aplicație și baza de date.

## Biblioteca de programare de declanșatori Quartz

Biblioteca de programare de declanșatori Quartz încorporată în Spring Boot oferă capabilități bogate pentru a gestiona și a rula sarcini programate în aplicațiile de tip enterprise. Oferă un framework de programare flexibil și robust, care permite dezvoltatorilor să automatizeze job-uri repetitive, execuția de sarcini și declanșatoare bazate pe timp.

Una dintre caracteristicile fundamentale ale bibliotecii Quartz este capacitatea sa de a defini și programa joburi. Dezvoltatorii pot crea clase de joburi care încapsulează sarcini sau acțiuni specifice care trebuie efectuate la intervale programate. Aceste sarcini pot varia de la sarcini simple la logică de aplicație complexă, oferind flexibilitate pentru a răspunde diferitelor nevoi ale aplicațiilor.

Quartz acceptă diferiți declanșatori pentru a programa execuția joburilor. Declanșatoarele bazate pe timp, cum ar fi expresiile cronologice sau intervalele fixe, permit dezvoltatorilor să definească intervale precise pentru execuția joburilor. Expresiile cronologice permit dezvoltatorilor să specifice date, ore și intervale specifice pentru rularea joburilor recurente. Intervalele fixe permit executarea joburilor la intervale regulate. Aceste mecanisme de declanșare permit un control granular asupra când și cât de des sunt executate joburile.

Biblioteca oferă funcții pentru gestionarea persistenței și stocarea joburilor. Quartz acceptă diferite tipuri de stocare a joburilor, inclusiv stocare bazată pe memorie, stocare bazată pe JDBC și stocare în cluster. Stocarea joburilor asigură păstrarea joburilor programate și a detaliilor lor de execuție, permițând executarea și recuperarea fiabilă a jobului în cazul repornirii sau eșecului unei aplicații.

Quartz oferă mecanisme puternice de execuție a comenzilor și de gestionare a ciclului de viață. Acceptă ascultătorii de joburi care le permit dezvoltatorilor să se conecteze la diferite etape de execuție a jobului, de exemplu înainte de executarea jobului, după finalizarea jobului sau în cazul unei erori. Ascultătorii de joburi permit dezvoltatorilor să efectueze acțiuni personalizate, înregistrare în jurnal sau gestionarea erorilor în timpul procesului de execuție a jobului.

## Framework-ul Angular

Angular este un framework JavaScript utilizat pe scară largă pentru construirea de aplicații web. Oferă un set cuprinzător de funcționalități care permit dezvoltatorilor să creeze aplicații dinamice, receptive și bogate în funcționalități, cu o abordare structurată.

Una dintre funcționalitățile fundamentale ale Angular este arhitectura bazată pe componente. Aplicațiile construite cu Angular constau din componente reutilizabile și modulare. Fiecare componentă încapsulează o funcționalitate specifică și constă dintr-un șablon care definește interfața cu utilizatorul și o clasă care conține logica și datele asociate cu componentul. Această abordare modulară promovează reutilizarea codului, mentenabilitatea și testabilitatea.

Angular oferă un sistem puternic de șabloane care permite dezvoltatorilor să definească interfețe de utilizator dinamice și interactive. Sintaxa șablonului acceptă legarea de date, care creează o conexiune între datele componente și elementele UI. Acest lucru permite actualizările automate ale interfeței de utilizare atunci când datele de bază se modifică. Directivele șablonului Angular permit dezvoltatorilor să gestioneze interacțiunile utilizatorilor, să manipuleze elementele DOM și să afișeze condiționat conținut, îmbunătățind experiența utilizatorului.

O altă funcționalitate cheie a Angular este mecanismul robust de injectare a dependenței (DI). DI permite cuplarea liberă a componentelor și promovează codul modular și testabil. Containerul DI al Angular gestionează crearea și injectarea dependențelor în componente, permițând o mai bună întreținere, organizare a codului și extensibilitate.

Angular oferă un sistem de rutare puternic pentru construirea de aplicații cu o singură pagină (SPA). Routerul permite dezvoltatorilor să definească diferite rute în cadrul aplicației și să asocieze fiecare rută cu o componentă specifică. Acest lucru permite navigarea între vizualizări fără a fi nevoie să reîncărcați întreaga pagină. Routerul Angular acceptă, de asemenea, funcții precum gărzi de rute, care permit dezvoltatorilor să controleze accesul la anumite rute pe baza regulilor de autentificare sau autorizare.

Framework-ul încorporează o paradigmă de programare reactivă prin utilizarea RxJS (Reactive Extensions for JavaScript). RxJS oferă un set de operatori și modele puternice pentru procesarea fluxurilor asincrone de date. Angular folosește observabilele RxJS pentru a gestiona și a răspunde la modificările datelor, permițând gestionarea eficientă a evenimentelor, operațiuni asincrone și actualizări în timp real.

Angular permite comunicarea eficientă cu serviciile backend prin modulul său HTTP. Oferă un API la nivelul clientului pentru trimiterea cererilor HTTP către servere și procesarea răspunsurilor. Motorul HTTP acceptă funcții precum interceptarea cererilor, gestionarea erorilor și transformarea răspunsurilor, permițând integrarea perfectă cu API-urile RESTful sau cu alte servicii backend.

## Keycloak

Keycloak este un manager de autentificare și autorizare open source care oferă funcționalități complete pentru a securiza aplicațiile și serviciile web. Oferă o soluție centralizată pentru gestionarea identităților utilizatorilor, aplicarea politicilor de control al accesului și integrarea diferitelor mecanisme de autentificare.

Una dintre caracteristicile fundamentale ale Keycloak este capacitatea sa de a efectua autentificarea și autorizarea utilizatorilor. Oferă o interfață ușor de utilizat pentru gestionarea conturilor de utilizator, inclusiv funcții precum înregistrarea utilizatorilor, resetarea parolei și autentificarea cu mai mulți factori. Keycloak acceptă diverse mecanisme de autentificare, inclusiv nume de utilizator/parolă, autentificări sociale (de exemplu, Google, Facebook) și furnizori de identitate (de exemplu, LDAP, Active Directory), permițând integrarea perfectă cu diverse scheme de autentificare a utilizatorilor.

Keycloak implementează protocoale standard din industrie, cum ar fi OpenID Connect (OIDC) și OAuth 2.0, care permit autentificarea și fluxurile de autorizare sigure. Aceste protocoale permit aplicațiilor să solicite și să primească token-uri de acces care reprezintă identitățile și permisiunile utilizatorilor. Keycloak acționează ca un server de autentificare care emite și validează aceste token-e, asigurând o gestionare sigură și fiabilă a identității.

O altă caracteristică cheie a Keycloak este suportul pentru controlul accesului bazat pe rol (RBAC) și autorizarea cu granulație fină. Keycloak permite administratorilor să definească roluri și permisiuni, să le atribuie utilizatorilor sau grupurilor și să impună controlul accesului pe baza acestor roluri și permisiuni. Acest lucru permite dezvoltatorilor să implementeze reguli complexe de autorizare, să restricționeze accesul la anumite resurse și să asigure confidențialitatea și integritatea datelor.

Keycloak oferă capabilități de Single Sign-On (SSO) care permit utilizatorilor să se autentifice o dată și să acceseze fără probleme mai multe aplicații, fără a fi nevoie să-și reintroducă credențialele. Acest lucru îmbunătățește experiența utilizatorului, reduce costul general de gestionare a mai multor parole și îmbunătățește securitatea aplicațiilor prin centralizarea autentificării și a gestionării sesiunilor.

Keycloak oferă opțiuni de integrare pentru a securiza diverse tipuri de aplicații, inclusiv aplicații web, aplicații mobile și microservicii. Oferă adaptoare client și SDK-uri pentru limbaje și framework-uri de programare populare, permițând dezvoltatorilor să-și integreze cu ușurință aplicațiile cu capacitățile de autentificare și autorizare ale Keycloak.

În plus, Keycloak acceptă integrarea cu furnizori externi de identitate și protocoale de federație. Permite aplicațiilor să delege procesul de autentificare unor sisteme externe, cum ar fi platformele de social media sau furnizorii de identitate corporativă. Keycloak acționează ca un broker facilitând integrarea perfectă cu aceste sisteme externe, permițând autentificarea sigură.

# Arhitectura aplicației

## 3.1 Arhitectură bazată pe microservicii

Arhitectura bazată pe microservicii este o abordare arhitecturală care pune accent pe descompunerea aplicațiilor mari, monolitice, în servicii mai mici, autonome. Fiecare serviciu este foarte specializat și proiectat independent și are propria sa logică de aplicație, stocare a datelor și mecanisme de comunicare. Această abordare promovează scalabilitatea, agilitatea și rezistența în dezvoltarea de software.

Unul dintre aspectele fundamentale ale arhitecturii bazate pe microservicii este conceptul de autonomie a serviciilor. Fiecare microserviciu funcționează independent și poate fi dezvoltat, implementat și scalat independent de alte servicii. Acest lucru permite decuplarea și izolarea funcționalității, permițând echipelor să lucreze la diferite servicii simultan și să le implementeze în propriul ritm. Autonomia facilitează, de asemenea, izolarea defecțiunilor, deoarece defecțiunile dintr-un serviciu nu afectează neapărat întregul sistem.

Microserviciile comunică între ele folosind protocoale simple și independente de limbă, cum ar fi HTTP/REST sau cozi de mesaje. Această comunicare poate fi sincronă sau asincronă în funcție de cerințele sistemului. Prin decuplarea serviciilor prin intermediul API-urilor bine definite, acestea pot evolua independent, facilitând introducerea de noi funcții sau tehnologii fără a afecta întregul sistem.

Un alt aspect important al arhitecturii bazate pe microservicii este concentrarea pe contexte limitate. Serviciile sunt concepute pentru anumite funcții de afaceri și sunt responsabile pentru un set clar definit de sarcini. Acest lucru permite echipelor să vizeze domenii specifice de afaceri și să construiască servicii care se potrivesc exact nevoilor afacerii. Contextele limitate încurajează o mai bună înțelegere și o mai bună proprietate asupra bazei de cod, precum și întreținerea și refactorizarea mai ușoare.

Arhitectura de microservicii promovează utilizarea managementului descentralizat al datelor. Fiecare serviciu are propriul magazin de date, care este ales în funcție de nevoile specifice ale serviciului respectiv. Acest lucru promovează cuplarea slabă și evită dependențele de o bază de date centrală. În funcție de cerințele individuale, serviciile pot utiliza diferite baze de date sau tehnologii de stocare a datelor, cum ar fi baze de date relaționale, baze de date NoSQL sau cache-uri în memorie.

Microserviciile sunt de obicei implementate ca entități autonome, adesea încapsulate în containere ușoare precum Docker. Acest lucru permite implementarea, scalarea și gestionarea ușoară a serviciilor individuale. De asemenea, permite utilizarea eficientă a resurselor, deoarece serviciile pot fi scalate independent în funcție de modelele lor specifice de cerere.

Arhitectura promovează reziliența și toleranța la erori prin utilizarea unor tehnici precum redundanța și întreruptoarele de circuit. Redundanța implică rularea mai multor instanțe ale aceluiași serviciu pentru a asigura o disponibilitate ridicată și toleranță la erori. Întreruptoarele de circuit sunt utilizate pentru a gestiona defecțiunile și pentru a preveni defecțiunile în cascadă prin izolarea și eșuarea rapidă atunci când dependențele sau serviciile externe nu sunt disponibile.

Monitorizarea și observabilitatea sunt esențiale într-o arhitectură bazată pe microservicii. Fiecare serviciu creează loguri sau metrici care pot fi agregate și analizate pentru a obține informații despre starea generală de sănătate și performanța sistemului. Instrumentele centralizate de monitorizare și urmărirea distribuită pot ajuta la identificarea și depanarea problemelor din servicii.

Aplicația Airlliant are în compoziție cinci microservicii care în totalitate formează produsul final și care fiecare în parte comunică unul cu celălalt pentru a transmite date pe rețea prin intermediul API-urilor REST și a conexiunilor de baze de date facilitate de către aplicații.

### 3.1.1 Microserviciul AirlliantCore

AirlliantCore expune API-ul principal al aplicației prin intermediul framework-ului Java Spring. Cu ajutorul framework-ului integrat Spring Boot acesta utilizează serverul web dedicat pe nume Tomcat pentru a facilita traficul pe rețea prin intermediul protocolului REST și pentru a expune puncte terminale de API precum punctele terminale de zboruri, utilizatori, bilete și plăți.

AirlliantCore facilitează maparea obiectelor relaționale prin intermediul framework-ului Hibernate ORM și Spring Data JPA. Prin dependența modulului AirlliantModel, un modul specific aplicației ce nu rulează în sine, ci oferă acces la mapările specifice pentru tabelele din baza de date, microserviciul poate serializa și deserializa date primite pe rețea prin apelurile de API și le poate utiliza ca și obiecte obișnuite Java.

Prin intermediul API-ului, acest modul al aplicației gestionează preluarea, crearea, ștergerea și actualizarea datelor din baza de date pentru a efectua logica de aplicație și pentru a aproviziona modulul ce se ocupă cu interfața grafică pentru utilizatori cu datele necesare de afișare.

Totodată acest microserviciu se ocupă și cu monitorizarea metricilor expuse de către biblioteca de monitorizare Actuator inclusă în proiect. Prin intermediul Actuator se pot expune metrici importante despre utilizarea resurselor de computație sau de rețea și alte date necesare de contorizare a performanței sistemului prin intermediul metricilor de JVM (Java Virtual Machine).

Pentru un proces de dezvoltare agil și eficient, microserviciul AirlliantCore documentează în timp real date despre API prin intermediul bibliotecii Swagger OpenAPI. Această bibliotecă face posibil de a documenta date despre obiectele mapate din aplicație ca scheme asemănătoare bazelor de date prin intermediul adnotărilor aplicate pe câmpurile entităților din modulul AirlliantModel. Înafară de entități, biblioteca documentează API-ul cu ajutorul adnotărilor aplicate la nivelul punctelor terminale definite în interfața din aplicație cu documentație despre ce acceptă ca și corp de cerere și ce poate emite ca răspuns, dar și ce header-e se pot folosi și dacă sunt necesare unele câmpuri sau nu. Documentația este oferită ca și interfață grafică și este expusă ca și punct terminal în aplicație și se poate accesa printr-un URL dedicat.

# Preliminarii

# Concluzii

# Bibliografie