**UNIVERSITATEA DIN BUCUREȘTI**

**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA**

**BAZE DE DATE SI TEHNOLOGII SOFTWARE**

**Lucrare de disertatie**

**FlyBird – Sistem informatic si de gestiune al zborurilor**

**Coordonator științific Absolvent**

**Banu Iulia Teodora Vătafu Ștefan**

**București, Iulie 2023**

Cuprins

Contents

[Cuprins 2](#_Toc137631504)

[Abstract 4](#_Toc137631505)

[1.Introducere 5](#_Toc137631506)

[*1.1 Motivarea alegerii temei* 5](#_Toc137631507)

[*1.2 Prezentarea generala a proiectului* 6](#_Toc137631508)

[2. Arhitectura proiectului 8](#_Toc137631509)

[3.1 Frontend 10](#_Toc137631510)

[3.1.1 Angular 10](#_Toc137631511)

[3.1.2 Bootstrap 12](#_Toc137631512)

[3.2 Backend 13](#_Toc137631513)

[3.2.1 Java 13](#_Toc137631514)

[3.2.2 Spring 14](#_Toc137631515)

[3.2.3 Hibernate 14](#_Toc137631516)

[3.3.3 Spring Security 14](#_Toc137631517)

[3.3.4 Spring Rest 15](#_Toc137631518)

[3.3 Baza de date MySQL 16](#_Toc137631519)

[4. Implementare 17](#_Toc137631520)

[4.1 Implementare JWT (Java Web Token) 17](#_Toc137631521)

[4.2 Inregistrarea unui nou user 19](#_Toc137631522)

[*4.3 Logarea unui utilizator in aplicatie* 23](#_Toc137631523)

[4.4 Blog dinamic 26](#_Toc137631524)

[4.5 Cautarea unui zbor 28](#_Toc137631525)

[4.6 Adaugarea unui zbor nou 30](#_Toc137631526)

[4.7 Update-ul unui zbor nou 32](#_Toc137631527)

[4.8 Eliminarea/Stergerea unui zbor 36](#_Toc137631528)

[4.9 Rezervarea unui zbor 37](#_Toc137631529)

[4.10 Adaugarea boarding pass-ului in rezervare. 38](#_Toc137631530)

[5. Concluzii și implementări viitoare 39](#_Toc137631531)

[Bibliografie 40](#_Toc137631532)

Abstract

Scopul acestui proiect este realizarea unui sistem informatic si de gestiune al zborurilor. Obiectivul este acela de a implemneta o aplicatie web completa, complexa si usor de manevrat pentru toate tipurile de utilizatori, dar si pentru administratorii acesteia.

Această aplicație web va fi împărțită în doua mari categorii: Frontend și Backend utilizându-se pentru fiecare diferite tehnologii care vor fi detaliate mai pe larg în capitolele ce vor urma.

Prima categorie, partea de la Frontend, are scopul de a realiza o interfata dinamica si usor de manevrat pentru utilizatori. Pentru realizarea ei se va folosi un framework de Java Script, Angular, ce se bazeaza pe Typescript si Bootstrap pentru adaugarea componentelor de interfata, dar cuprinde si o serie larga de librarii utile in implementarea de noi functionalitati.

A doua categorie a proiectului, Backend-ul, a fost realizata folosind limbajul de programare Java, folosind pattern-ul arhitectural Spirng MVC (Model-View-Controller) pentru a realiza RESTful Web Services. Împreună cu Spring MVC vom mai folosii tehnologii precum Hybernate care ne va ajuta la maparea domeniilor la o bază de date relațională și framework-ul Spring Security pentru autentificare și autorizare.

Ca și sistem de gestiune a bazelor de date relaționale vom utiliza MySQL care este un SGBD open-source.

1.Introducere

## *1.1 Motivarea alegerii temei*

Motivul alegerii unei aplicatii Web in domeniul aviatic ca si tema de disertatie a aparut datorita domeniului in care am studiat si datorita dorintei de a realiza o astfel de aplicatie utilizand diferite tehnologii care combinate sa produca un produs finit.

Mai mult decat atat aceste aplicatii sunt foarte des intalnite in zilele noastre si acest fapt se datoreaza avantajelor acestora:

* Compatibilitatea dintre sisteme – majoritatea aplicatiilor web trebuie sa indeplineasca o cerinta minima si anume existenta unui browser precum (Micrososft Edge, Google Chrome, Safari), astfel indiferent de sistemul de operare folosit aplicatiile web pot rula cu usurinta.
* Reducerea Costurilor – aplicatiile web pot reduce costurile atat pentru furnizor datorita costurilor mici pentru suport dar si pentru partea de intretinere cat si pentru utilizator deoarece sistemul acestuia are nevoie de specificatii minime pentru a putea utiliza aplicatia.
* Mult mai gestionabil – dupa cum am spus mai sus aplicatiile web necesita cerinte minimme din partea statiilor de lucru ale utilizatorului si impreuna cu avantajul ca acestea sunt instalate doar pe server, actualizarea sistemului este se poate face foarte rapid si foarte simplu.

## *1.2 Prezentarea generala a proiectului*

În această lucrare de disertație am ales să dezvolt un sistem informatic de gestionare al zborurilo, mai exact o aplicație web care se va focusa atât pe utilizatori cât și pe partea de administrare.

Numele aplicației este FlyBird și va avea o pagină principală în care se vor găsi informații generale despre Companie. Navigarea pe pagină se va face din bara de navigare din partea de sus a site-ului. Din această zonă, se poate naviga foarte usor către diferite secțiuni ale site-ului:

Din perspectiva unui utilizator vom avea urmatoarele zone care vor contine diferite functionalitati:

* Info unde:
* userii vor putea descoperii informatii despre avioanele puse la dispozitie de catre companie, cu specificatiile fiecaruia dintre ele;
* detalii despre clasele de travel ( 3 la numar ) : Economy, Premium Economy si Bussines Class;
* Următoarea secțiune este Fly unde:
* se vor afla informatii despre Check-in, statusul zborului si mobile check-in;
* A treia secțiune prezentă este Plan unde:
* se vor afla informatii legate de bagaje si cerinte speciale pentru: animale, persoane cu probleme medicale;
* A patra secțiune este secțiunea de Book unde:
* se poate face booking unui zbor;
* fiecare utilizator isi poate vizualiza zborurile rezervate;
* A cincea sectiune va face parte din pagina de profil a utilizatorului, unde se vor afla date personale despre acesta ca de exemplu: Nume, Prenume, Data nasterii, Oras, CNP, Numar de telefon, Email.
* Ultima zonă este reprezentată de pagina de autentificare și de înregistrare, unde utilizatorii își pot crea contul și ulterior se pot autentifica beneficiind de secțiunile menționate mai sus. To aici, utilizatorii isi vor schimba parola in cazul in care acesta a uitat-o.

Din perspectiva unui administrator vor exista toate paginile pe care le poate vizualiza și utilizatorii cu toate funcționalitățile acestora, însă va avea și o secțiune specială numită „Administration Dashboard” care va reprezenta interfața de administrare a aplicației, specială doar utilizatorilor cu rol de admin, unde se vor putea efectua și vizualiza următoarele date:

* La deschiderea sectiunii „Dashboard” din bara de navigare, acesta va avea subcategorii precum „Dashboard Flights”, „Create a new Flight”;
* In cadrul „Dashboard Flights” acestuia i se va deschide o lista cu toate zborurile unde se vor putea updata si sterge zborurile;
* In cadrul „Create a new Flights”, adminul poate adauga un zbor nou care se va vedea imediat in lista de zboruri;
* Aceesta va avea posibilitatea si de a modifica rezervarile clientilor in cazul in care unul dintre acestia comite o eroare, prin intermediul sectiunii „Edit Bookings”;

2. Arhitectura proiectului



Figură 1. Arhitectura Apliației Web. Sursă Proprie.

În continuare vom analiza diagrama de mai sus:

Frontend-ul reprezintă componenta aplicației web, pe care utilizatorul o poate vedea și cu care poate interacționa. Această componentă a apliacției este împărțită la rândul ei în două sub-categorii: partea estetică (design-ul) și dezvoltarea implementării interfeței. Partea estetică a aplicației se referă la: meniuri, navigatia platformei, designul butoanelor, fonturile, caruselele cu imagini. În timp ce dezvoltarea aplicației se focusează pe modul în care de exemplu, funcționează butoanele și dacă acestea îndeplinesc corect funcționalitățile pentru care au fost create.

Frontendul nu face logică de business adică nu prelucrează datele. Scopul lui este doar de a colecta datele și de a le afișa utilizatorului sub o formă cât mai prietenoasă. Acesta va face un Request catre Backend acolo unde se face prelucrarea de date.

Pentru a se procesa datele avem nevoie de un server de Backend. Aceasta este componenta care stochează datele și care comunică cu baza de date, comunică cu diferite sisteme de fișiere și poate comunica chiar cu alte sisteme de backend și face toate prelucrările necesare. După aceasta etapa a prelucrărilor, serverul va trimite un Response către aplicația de Frontend, iar aceasta la rândul ei va afișa rezultatele către user.

3. Tehnologii folosite

3.1 Frontend

Pentru implementarea Frontend-ului ne vom ajuta de Visual Studio Code care reprezinta un editor de cod sursa foarte puternic reusind sa ruleze pe toate sistemele de operare. Acest tool este foarte utilizat datorita suportului cu care vine incorporat pentru JavaScript, TypeScript si Node.js. Cu toate acestea a devenit foarte populat si pentru limbaje de programare precum Java, C# sau Python datorita ecosistemului foarte bogdat de extensii pentru aceste limbaje. [1]

3.1.1 Angular

Angular este o platforma de dezvolatare construita de catre TypeScript. Acesta este proiectat sa trimita update-urile facute cat mai rapid, avand astfel posibilitatea sa profiti de acestea aproape instant.

Fiind o platforma Angular include:

* Un framework pentru construirea de aplicatii web scalabile;
* O colectie variata de librarii integreate care ajuta la implementarea diferitor caracteristici;
* O multitudine de instumente care te pot ajuta sa iti dezvolti, constuiesti si testezi codul.

Pentru o mai buna intelegere a modului cum functioneaza Angular vom caracteriza mai departe componentele de baza ale acestuia:

* Modulele – Aplicatiile Angular sunt modulare si astfel Angular are propriul sistem de modularitate numit NgModules. NgModules reprezinta containare pentru un anumit bloc de cod dedicat unui domeniu al aplicatiei, sau a unui workflow. Fiecare aplicatie Angular are cel putin un modul, care reprezinta radacina si care este denumita AppModule, reprezentand mecanismul de bootstrap care lanseaza aplicatia.
* Componentele – reprezinta blocurile din care este compusa aplicatia. Componenta include o clasa TypeScript reprezentata de o adnotare @Component(), un template HTML si stilurile. Clasa @Component() specifica urmatoarele informatii:

1. Un selector CSS care defineste modul in care componenta este folosita in template.
2. Un template HTML care indica cum sa fie randata componenta;
3. Un set de stiluri CSS care ajuta la design-ul elementelor HTML.

* Servicii si Dependency Injection – pentru date sau logica care nu sunt impartite intre componente, se va crea o clasa service. O astfel de clasa este imediat precedata de adnotarea @Injectable(). Aceasta adnotare ofera metadatele care permit altor furnizori sa fie injectati ca si dependinte in clasa ta. Cu ajutorul Dependency Injection componentele vor fi mult mai eficiente deoarece nu vor prelua datele de pe server si nu se vor conecta direct la consola aceste sarcini se vor delaga catre servicii. [2]



Figură 2. Diagrama Arhitecturala a unei Aplicati Angular [2]

3.1.2 Bootstrap

Bootstrap este un framework open source de frontend, care contine elemenete de baza pentru dezvoltarea aplicatiilor web receptive. Acest framework este format pe baza HTML, CSS si JavaScript.

Cu ajutorul Bootstrap se poate detecta dimensiunea ecranului pe care o foloseste utilizatorul, adaptand dimensiunile componentelor aplicatiei web automat. Pe langa aceasta functionalitate de baza, printre componentele incluse in interfetele Bootstrep se regasesc barile de navigatie, carusele de imagini si butoane.

Printre avantajele utilizarii acestui framework se regasesc:

* Fiind responsive, ajuta la performantele aplicatiei web deoarece tot procesul de optimizare este cu mult ameliorat.
* Compatibilitatea cu diferite browsere web;
* Fiind un framework foarte popular documentatia este bogata, iar datorita acestui lucru se pot gasi foarte usor exemple de cod care trebuie doar adaptate la necesitatile fiecaruia.

Exista si mici dezavantaje in folosirea Bootstrap cum ar fi:

* Necesitatea ca browserele sa fie updatate, deoarece daca se foloseste o versiune noua a frameworkului, iar versiunea browserului este mai veche vor exista discrepante.
* Daca se va utiliza un numar mare de functii va rezulta la dimensiuni ale fisierelor foarte mari ceea ce poate duce la ingreunarea proiectului. [3]

3.2 Backend

Pentru implementarea Backend-ului ne vom ajuta de IntelliJ IDEA care reprezinta un mediu de dezvoltare integrat (IDE). Este unul dintre cele mai utilizate IDE-uri pentru dezolvatarea aplicatiilor software in limbajul Java.

Printre caracteristicile sale cele mai importante se numara:

* Integrarea cu instrumente de control al versiunii precum Git, SVN;
* Suport pentru dezvoltarea de aplicatii web prin framework-uri popilare precum Spring avand capacitatea de a rula si testa aplicatiile local sau pe servere externe.

3.2.1 Java

Java este un limbaj de programare orientat pe obiecte. A fost lansat pentru prima data in anul 1995 si a devenit foarte popular in lumea dezvoltarii software, fiind utilizat intr-o foarte mare varietate de aplicatii si sisteme.

Este un limbaj de programare compilat si interpretat, ceea ce inseamna ca codul sursa Java este transformat intr-un format intermediar (bytecode) care apoi este interpretat de masina virtuala Java (JVM) pentru a rula programul. Astfel aplicatiile Java pot rula pe aproape orice platforma, fiind foarte usor sa fie portabile.



Figură 3. Arhitectura Java [4]

3.2.2 Spring

Spring Framework este este un framework open-source pentru dezvoltarea aplicatiilor in Java. Acesta ofera o gama laga de functionalitati care fac ca dezvoltarea aplicatiilor web se fie mai usoara si mult mai eficienta. Printre acestea se numera:

* Inversion of control (Inversiunea de control) si dependency injection (injectia de dependinte). Spring usureaza crearea obiectelor si gestiunea dependentelor dintre acestea prin intermediul unui container care se ocupa cu crearea, configurarea si gestionarea ciclului de viata al acestora.
* Securitatea: Spring pune la dispozitie diferite instrumente pentru securizarea aplicatiiei atat din punct de vedere al autentificarii cat si din punct de vedere al autorizarii.
* Accesul bazei de date: Spring ofera un set de instrumente pentru accesarea bazelor de date, prin intermediul framework-ului Hibernate.

3.2.3 Hibernate

Hibernate ofera o solutie de mapping obiect-relational (ORM) care faciliteaza interactiunea dintre bazele de date relationale si obiectele create in Java. Astfel programatorul lucreaza intr-un mod mai natural, folosind obiecte si metode, in loc sa scrie interogari SQL.

Hibernate este proiectat astfel incat sa se poata integra cu alte framework-uri si librarii precum Srping si Struts.

3.3.3 Spring Security

Spring Security ofera un set de instrumente pentru ajutorul implementarii fucntionalitatilor de autentificare, autorizare si gestionare a sesiunilor in aplicatiile web.

Acesta este folosit pentru a prevenii atacurile de tipul Cross-Site Scripting, SQL injection, Cross-Site Request Forgery.

3.3.4 Spring Rest

Spring MVC este un framework de dezolvare a aplicatiilor web in Java.

Abrevierea MVC vine de la Model-View-Controler ce reprezinta un pattern arhitectural prin care se separa logica in:

* Model – reprezentand obiectele si logica acestora;
* View – reprezentand aspectul vizual al aplicatiei si defineste modul in care datele sunt afisate utilizatorului.
* Controller – gestioneaza interactiunea dintre model si vizualizare. Aici se primesc cereri HTTP de la utilizatori si de face procesarea datelor.

Spring REST ( Representational State Transfer) este o extensie a framework-ului Srping MVC axandu-se pe dezolvatarea serviciilor RESTful in Java. Aceste servicii web reprezinta o modatlitate foarte populara de la construii servicii web care se bazeaza pe protocale HTTP pentru a permite mai usor comunicarea dintre client si server. Prin aceasta extensie serviciile web RESTful pot comunica cu alte aplicatii si sisteme prin intermediul formatelor XML, JSON.



Figură 4. Diagrama Spring MVC [5]

3.3 Baza de date MySQL

MySQL este un sistem de management al bazelor de date relational si open source, folosit pentru stocarea si gestionarea datelor intr-un mod eficient. Este una dintre cele mai populare si utilizate baze de date in aplicatiile web datorita faptului ca ofera o performanta ridicata si securitate. Este compatibil cu o multitudine de sisteme de operare printre care si Linux si Windows, iar integrarea sa se poate realiza cu limbaje de programare precum: Java, Python.

4. Implementare

4.1 Implementare JWT (Java Web Token)

JWT ( JSON Web Token) este un standard open source pentru transmiterea securizata a informatiilor intre parti sub forma de obiecte JSON;

Un JWT este format din 3 parti: Header, Payload, Signature.

Headerul: cuprinde lista de operatiuni criptografice care sunt aplicate tokenului.

Exemplu: { "alg": "HS256", "typ": "JWT" }

In exemplul de mai sus alg reprezinta algoritmul folosit pentru a semna sau cripta tokenul si typ reprezinta tipul de informatie semnat sau criptat de obicei este JWT.

Payloadul: reprezinta chiar informatiile/datele ce trebuie transferate folosint tokenul. Aceasta parte se mai numeste si "claims". Claims-ul poate sa fie de 3 tipuri: registered, public si privat.

Signature: aceasta ultima parte din token este folosita pentru a verifica daca mesajul ce trebuia transmis nu a suferit modificari pe parcurs.

Primul pas pentru implementarea JWT a fost de a genera clasa WebSecurityConfig in cadrul careie am definit metoda SecurityFilterChain. SecurityFilterChain reprezinta un lant de filtre care urmareza sa fie aplicati pentru asigurarea securitatii unei aplicatii.

Printre acestea in cadrul proiectului am ales sa folosesc metroda authorizeHttpRequest() impreuna cu metoda requestMatchers() pentru a permite toate requesturile venite de la URL-urile: „/user/login” si „/user/register”. Iar pentru restul requesturilor venite este necesar ca utilizatorul sa fie autentificat.

Tot in cadrul acestei clase a fost definita si metoda PasswordEncoder care are rolul de a cripta parola, pentru o mai mare protectia asupra atacurilor cibernetice.

Al doilea pas al implementarii a fost de a genera clasa JwtUtil in care am definit metode precum:

* createToken();
* generateToken() aceasta metoda are ca parametru un nume de utilizator (userName) si cu ajutorul metodei createToken are ca rol generarea si returnarea unui token semant care va contine toate informatiile despre user;
* validateToken aceasta metoda verifica daca tokenul este valid. Tokenu este considerat valid daca subiectul corespunde cu numele utilizatorului fiind folosita metoda extractUsername;
* extractUsername;
* extractClaim;

A treia si ultima etapa din implementarea JWTRequestFilter. Clasa JWTRequestFilter este responsabila de verificarea tokenului in fiecare cerere HTTP. Aceasta extinde interfata „OncePerRequestFilter” cara asigura ca filtrul este aplicat o singura data pe fiecare cerere.

Metoda doFilterInternel() este suprascrisa este suprascrisa pentru a efectua logica filtrului in interiorul acestei clase. Aceasta primeste cererea HTTP(HttpServletRequest) raspunsul HTTP(HttpServletResponse) si lantul de filtre( FilterChain). In interiorul acestei metode se extrage header-ul „Authorization” din cerere si se verifica daca acesta incepe cu prefixul „Bearer”. Daca aceasta conditiei este indeplinita se extrage numele utilizatorului folosind „jwtUtil.extractUsername(token)”.

Daca numele utilizatorului este disponibil si autentificarea actuala nu exista, se incarca detalii despre utilizatorul respectiv utilizand metoda loadUserByUsername din cadrul userDetailsService. Urmatorul pas este de validare a tokenului. Daca tokenul este valid se creaza un obiect UsernamePasswordAuthenticationToken care contine informatii despre utilizator precum username, role, parola, iar aceste informatii sunt stocate in „SecurityContextHolder”.

A diagram of a server

Description automatically generated with low confidence

Figură 5. Diagrama Functionala JSON Web Token (JWT) [6]

4.2 Inregistrarea unui nou user

Pentru aceasta etapa am implementat prima data baza de date a userilor, care va contine urmaotarele coloane:

* Id;
* Email;
* Password;
* User\_role
* Cpassword;

A screenshot of a computer

Description automatically generated

Figură 6. Tabela Users din cadul bazei de date. Sursa Proprie.

Urmatorul pas a fost de a realiza clasa User care o sa serveasca ca si domeniu in aplicatie pe partea de backend. Pentru atributul Id, am folosit „annotarea” @Id care indica fa ptul ca membrul asupra caruia este pusa annotarea reprezinta cheia primara a entiatii user. Tot pentru aceasta coloana, s-a folosit si annotarea @GeneratedValue care ne ajuta la modul in care se incrementeaza o anumita coloana. Restul atributelor au fost annotate cu operatorul @Column care ajuta la maparea dintre atribute si coloanele bazei de date.

Dupa cum am mentionat mai sus in cadrul tabelei se regaseste coloana user\_role. Acest atribut este deifnit ca avand tipul UserRole insemnand ca acesta poate fi setat utilizand una dintre valorile definite in enumeratia UserRole in cadrul careia se regasesc 2 constante: „ADMIN” si „USER”.

In continuare am definit servis-ul, ce reprezinta o clasa care ofera functionalitati specifice pentru aplicatie si care este utilizata pentru a gestiona logica de business si interactiunea cu alte componente ale aplicatiei, cum ar fi repository-ul. Serviciile sunt implementate ca si calse in Java, fiind utilizate pentru a separa logica de business de infrastructura.

Am definit metoda createUser() care are ca rol de a salva un nou utilizator in baza de date folosind metoda save() din cadrul repository-ulu setand rolul automat de „USER” si criptand parola.

In ultimul pas al implementarii in cadrul backed-ului am definit controlerul, unde am realizat metoda signupUser() annotata cu @PostMapping. Aceasta annotare este folosita pentru a mapa o cere HTTP POST la o metoda specifica intr-un contoller.

In Spring Framework, un controller este o clasa Java care gestioneaza cererile HTTP primite de la serverul web si produce raspunsurile corespunzatoare. Controllerul este utilizat pentru a crea interfata partea vizuala a apicatiei si partea de business.

Metoda signupUser() din cadrul controllerul, va primi obiectul RegisterRequest ca un parametru trimis prin corpul cererii HTTP si va crea un nou utilizator folosind informatiile furnizate de client. RegisterRequest a fost gandit precum un DTO (Data Transfer Object), reprezentand o clasa simpla continand doar 3 atribute: email, password si cpassword si este folosita pentru a evita expunerea directa a obiectelor.

Mergand mai departe pe partea de frontend, aici primul pas a fost de a genera component register in Angular. In cadrul acesteia am modelat fisierul html si css pentru a reda o interfata placuta utilizatorului.

Graphical user interface, application

Description automatically generated

Figură 7. Interfata de inregistrare. Sursa Proprie.

A doua etapa a fost de a genera clasa aferenta inregistrarii utilizatorului. Aceasta va contine cele 3 atribute pe care le vedeti si in imagina de mai sus: email, password si cpassword. Aceasta clasa este utilizata in servicul de inregistrare a utilizatorului din Angular.

In Angular, un serivce reprezinta o clasa TypeScript care care furnizeaza anumite functionalitati, putand fi injectata in alte compontente sau chiar servicii pentru a impartasii functionalitatile cu acestea. In majoritatea timpului serviciile sunt utilizate pentru a efectua operatii asincrone pentur gestionarea datelor intr-un mod consistent si pentru a efectua interactiuni cu serverul prin apelurile de tip HTTP.

In interiorul serviciului de inregistrare am creat metoda registerUser care va primit obiectul/clasa User care a fost mentionata in a doua etapa si care va avea ca scop de a face apelarea catre backend pentru inregistrarea unui nou utilizator.

Penultima etapa a acestei implementari a fost ca in cadrul fisierul TypeScript din interiorul componentei register sa generam o metoda numita userRegister() care va avea ca scop apelarea metodei din service si de a trimite un mesaj in momentul in care userul a fost inregistrat cu succes.

Ultima etapa a constat in adaugarea functionalitatilor in cadrul fisierului html, folosind directiva ngSubmit care este de utilizata in Angular pentru a detecta si gestiona evenimentul de submit a unui formular HTML. Aceasta a apelat metoda realizata in cadrul fisierul TypeScript de inregistrare a unui utilizator.

Graphical user interface, application

Description automatically generated

Figură 8. Exemplu inregistrare user. Sursa proprie.

A screenshot of a computer

Description automatically generated

Figură 9. Inregistrarea cu succes a unui user. Sursa proprie.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 10. Inserarea automata dupa inregistrarea cu succes a unui user in tabela users. Sursa proprie.

## *4.3 Logarea unui utilizator in aplicatie*

Pentru logarea utilizatorilor ne-am ajutat de implementarea de la sectiunea 4.1. In crearea tokenului din interiorul clasei JwtUtil, am adaugat si rolul utilizatorului preluat din baza de data. Am adaugat in interiorul UserController-ului metoda createAuthenticationToken() asupra careie am folosit adnotarea @PostMapping(„/login”) pentru a mapa metoda pentru a gestiona solicitarile HTTP POST catre URL-ul /login.

Metoda primeste un obiect LoginDTO, obiect care contine datele de logare a unui utilizator, precum si un obiect ‚HttpServletResponse” care este utilizat pentru a trimite raspunsul inapoi catre client.

Am construit in bloc try unde se incearca autentificarea utilizatorului folosind un obiect UsernamePasswordAuthentication creat cu numele de utilizator in cazul nostru emailul si parola. Daca autentificarea esueaza din cauza credentialelor gresite, este aruncata o exceptie „BedCredentialsException” cu un mesaj de eroare.

Daca autentificarea are loc cu succes, se incarca detaliile utilizatorului folosind numele utilizatorului ca si parametru pentru identificarea acestuia in baza de date.

In final se genereaza un nou obiect JSON in corpul raspunsului. Acesta va contine ID-ul utilizatorului si rolul acestuia. De asemenea, adauga un header cu tokenul JWT generat.

Pentru partea de frontend, am generat serviciul StorageService care este responsabil pentru gestionarea datelor din LocalStorage. Am marcat acest serviciu ca injectabil, putand astfel sa il injectam in alte componente ca dependinta.

In acest serviciu am definit mai multe metode pentru a gestiona autentificarea user-ului si a datelor acestuia.

* saveUserId() – salveaza un id al unui utilizator in LocalStorage;
* saveUserRole() – salveaza rolul utilizatorului in LocalStorage;
* saveToken() – salveaza tokenul de autentificare in LocalStorage;
* hasToken() – verifica daca tokenul este prezent in LocalStorage;
* getToken() – returneaza tokenul din LocalStorage;
* signOut() – sterge tokenul si id-ul din LocalStorage, deconectand userul;
* isUserLoggedIn() – verifica daca utilizatorul este logat si daca are rolul de “USER”;
* getUserRole() – returneaza rolul din LocalStorage;
* getUser() – returneaza id-ul din LocalStorage;
* isAdminLoggedIn() – verifica daca utilizatorul este autentificat si daca are rolul de “ADMIN”;

Urmatorul pas a fost de a genera un pachet nou pentru rutele folosite de admin, si user dupa logare pentru a redirectiona cei doi utilizatori catre paginile lor de interes.

Pentru a putea realiza autorizarea pentru fiecare endpoint am generat in serviciile create, o metoda createdAuthorizationHeader() care va returna un obiect HttpHeader care va contine un header de autorizare setat. Tokenul pentru header-ul de autorizare fiind obtinut din serviciul de stocare mentionat mai sus.

Ulterior am generat si serviciul de logare care va contine o metoda login() cu doi parametrii username si parola. Aceasta foloseste serviciul HttpClient pentru a face o solicitare HTTP POST la un URL specificat, trimitand cei doi parametrii ca payload al cererii. Dupa ce raspunsul este primit vom folosii un operator RxJS, ‚map’ pentru a salva id-ul si rolul utilizatorului in LocalStorage folosind metodele din cadrul StorageService si vom extrage tokenul de autentificare din header-ul ‚Authorization’ al raspunsului indepartand stringul ‚Bearer ’ si salvand tokenul.

Ultimul pas pentru a finaliza logarea unui utilizator a fost de a gestiona autentificare acestuia prin intermediul unei forme de autentificare. Astfel am generat componenta de logare cu fisierele aferente. Am construit un formulat in fisierul html, urmand ca in fisierul TypeScript in metoda ngOnInit(), sa construim formularul folosind FormBuilder. Acest formular va contine doua campuri, username si parola, ambele campuri fiind obligatorii.

Am generat si metoda userLogin() care este apelata in momentul in care utilizatorul doreste sa se autentifice. Acesta va folosii serviciul de logare pentru a trimite datele formularului catre server si de a incerca autentificarea. Daca autentificarea are loc cu succes, aceasta verifica rolul utilizatorului si il redirectioneaza catre pagina dorita, fie catre pagina de ‚/welcome’ in cazul utilizatorului, fie catre pagina de ‚/admin/dashboard’ in cazul administratorului.

A screenshot of a computer

Description automatically generated

Figură 11. Autentificarea cu rolul de admin. Sursa proprie.

4.4 Blog dinamic

Aceasta implementare a fost gandita cu scopul de a informa utilizatorii despre noile noutati in care este implicata compania.

Pentru aceasta mica implementare am folosit plaforma Contentful. Contentful este o platformă de gestionare a conținutului (CMS) care permite companiilor și dezvoltatorilor să creeze și să gestioneze conținutul digital. Este utilizată în special pentru construirea și gestionarea conținutului pentru site-uri web, aplicații mobile și alte aplicații digitale.

Contentful oferă o interfață de utilizator intuitivă și un API puternic, ceea ce permite dezvoltatorilor să construiască experiențe personalizate și să integreze ușor conținutul în aplicațiile lor. Platforma oferă, de asemenea, capacități de gestionare a fluxului de lucru, ceea ce face mai ușor pentru echipe să colaboreze în crearea și publicarea de conținut.

Contentful este o soluție flexibilă și scalabilă, fiind utilizată de companii de diferite dimensiuni și în diferite domenii, de la start-up-uri și agenții de design la întreprinderi mari și organizații guvernamentale.

Intr-o prima etapa am generat un Content Model in platform mentionand campurile pe care fiecare Content il va avea.

Graphical user interface, text, application, email

Description automatically generated

Figură 11. Generare Content Model platforma Contenful [7].

Dupa ce au fost generate Content-urile, am generat in cadrul proiectului un serviu in Angular pentru a realiza comunicarea intre aplicatie si platforma Contentful.

In cadrul serviciului, am creat o variabila client care va stoca SpaceID-ul si Access Tocken-ul pentru utilizatorul platformei, astfel reusind sa facem conexiunea la spatiul de lucru generat, urmand sa generam o noua metoda numita getAllEntries() pe care o vom apela in cadrul componentei news care ne va ajuta sa returnam toate „Contenturile” de pe platforma pentru spatiul de lucru indicat insa sub forma de json.

In a doua etapa am folosit directiva ngFor pentru a itera prin elementele jsonului, putand astfel sa afisam imagina,titlul, summary alocate fiecarui „Content”.

Graphical user interface, application

Description automatically generated

Figură 12. Dynamic Blog. Sursa proprie.

4.5 Cautarea unui zbor

Pentru aceasta functionalitate am generat in prima instanta compontenta flight-search in Angular. In fisierul html am construit formular-ul de cautare care va contine 3 campuri:

* Leaving From de tipul text;
* Arriving At de tipul text;
* Departure Date de tipul date;

Am generat un nou serviciu search-flight in cadrul caruia am construit metoda GetFlight() care va apela endpointul din backend.

In fisierul typescript din componenta flight-search, am in cadrul metodei ngOnInit am setat variabila „now” care va reprezenta data curenta in care ne aflam. Aceasta este folosita in cadrul fisierului html pentru „Departure Date” pentru a seta in calendar data curenta astfel nu vom putea alege o data mai veche pentru care sa cautam zboruri. Metoda onSubmit() va apela la randul ei metoda GetFlight() din cadrul serviciului search-flight si va returna rezultatul sub forma de tabel.

Implementarea back-endului pentru aceasta parte a constat in generarea unei clase domeniu Flight care va avea ca rol generarea tabelei in cadrul careia se afla 5 variabile: id, arrivingat, leavingfrom, leavingat si nrseats. Dupa aceasta prima etapa a fost construit DTO-ul acestei clase care va contine aceleasi variabile mai putin id, repositoriul in care am generat metoda „findByLeavingfromAndArrivingatAndLevingdate” care va genera selectul pentru returnarea tuturor zborurilor in functie de data de plecare, locul de plecare si locul de aterizare urmand a fi folosit in service-ul „FlightService” unde se va intoarce sub forma de lista toate zborurile gasite. Iar in ultimul pas s-a generat controller-ul ”FlightController” in care vom folosi @GetMapping pentru metoda getFlights. Acest endpoint este folosit in service-ul din frontend.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Figură 13. Cautarea unui zbor. Sursa proprie.

A screenshot of a computer

Description automatically generated

Figură 14. Tabela de zboruri. Sursa Proprie.

4.6 Adaugarea unui zbor nou

Adaugarea unui unui zbor nou se va putea face doar de catre userul cu drep de admin.

Iar sectiunea de adaugarea unui zbor se va afla in bara de navigare specifica adminului (dashboardul adminului).

Pentru adaugarea unui nou zbor am adaugat in controlerul dedicat adminului o metoda numita createFlight() pe care am adnotat-o cu @PostMapping pentru a mapa o cerere HTTP Post la aceasta metoda. Ruta catre aceasta metoda fiind „/create/flight”.

In cadrul acestei metode am folosit @RequestBody pentru a marca faptul ca paramentrul flights de tipul Flights este legat de cerearea HTTP. Logica acestei metode este destul de simpla se va folosii metoda save() din cadrul repositoriului flightRepo care va adauga un zbor nou in baza de date.

Ajungand la partea de frontend, am generat prima data componenta de createFlight, in cadrul careia am adaugat un formular nou care va contine campurile unui zbor. Am realizat cu ajutorul directivei [(ngModel)] legatura bidirectionala intre valorile inputului si variabilele clasei FlightSearch generand un obiect de tipul FlightSearch in interiorul fisierului TypeScript al componentei de creare.

Al doilea pas a fost de a adauga metoda de creare a zborului in fisierul search-flight.service.ts unde vom returna raspunul cererii Post din cardul backendului. Ulterior vom utiliza aceasta metoda in cardul metodei onSubmit() din fisierul TypeScript al componentei createFlight unde vom afisa si un mesaj „Flight Created with Success” in momentul in care zborul se va inregista. Metoda onSubmit() este o metoda asociata de obicei cu evenimentul de trimitere a unui formular.

A screenshot of a computer

Description automatically generated

Figură 15. Adaugarea unui nou zbor. Sursa proprie.

A screenshot of a computer

Description automatically generated

Figură 16. Tabela flights, unde putem vedea adaugarea zborului. Sursa proprie.

4.7 Update-ul unui zbor nou

Update-ul unui zbor se va putea face doar de catre admin, acesta avand o sectiune dedicata in cadrul barii de navigare.

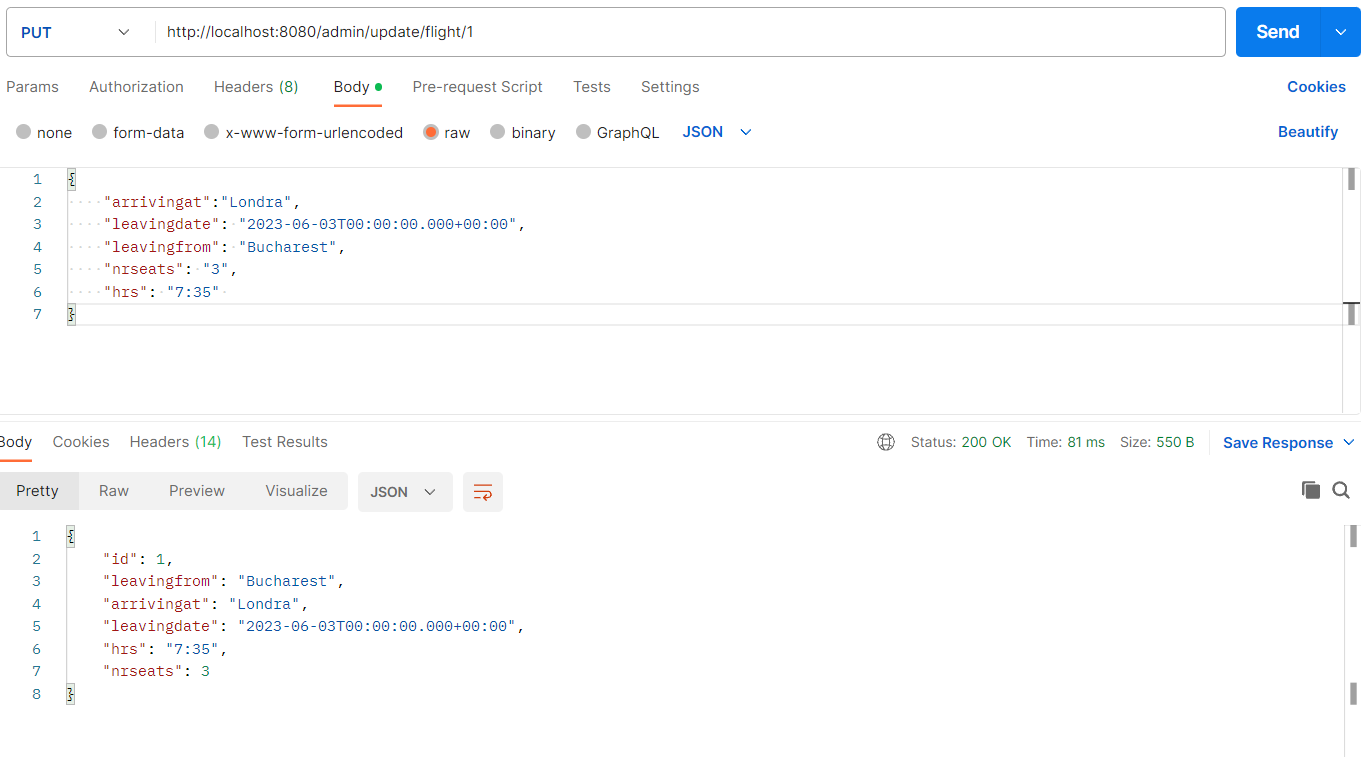
Pentru aceasta actiune am generat in cadrul controler-ului adminului, o metoda „updateFlight()” pe care am adnotat-o cu @PutMapping pentru a mapa o cere HTTP de tip PUT. Aceast update se va face pentru cate un zbor, variabila de identificare al acestuia fiind id-ul, iar in cazul in care nu vom gasi zborul cu id-ul trimis, se va intoarce o eroare „Flight does not exist with id”. In cazul in care vom gasi id-ul, va face un update pe atributele modificate ale zborului.

Initial am testat aceasta functionalitate in Postman, unde am updatat campul de hrs pentru prima valoare din tabela flights, care initial era setat pe null a fost setat la ora 07:35.

A screenshot of a computer

Description automatically generated

Figură 17. Tabela flights, inainte de a efectua update-ul. Sursa proprie.



Figură 18. Update-ul generat din Postma. Sursa proprie.

A screenshot of a computer

Description automatically generated

Figură 19. Tabela flights dupa efectuarea update-ului. Sursa proprie.

Am continuat prin adaugarea unei a doua metode insa de aceasta data in cadrul „FlightController” pentru a afisa un zbor in functie de id. Aceasta metoda a fost adnotata cu @GetMapping si in functie de id-ul pe care il trimitem in cadrul URL-ului se va intoarce zborul respectiv. In caz contrar se va intoarce o eroare.

Aceasta metoda o vom folosii in cadrul frontend-ului in momentul in care vom apasa butonul „Update” vom naviga automat catre un formular, unde campurile zborului respectiv se vor completa automat, uramand ca adminul sa efectueze modificarile necesare.

Dupa cum am mentionat mai sus pentru sectiunea de frontend a acestei implementari am generat o noua componenta „updateflight” unde am schitat prima data un formulat in cadrul fisierului html. Dupa finalizarea aspectului formularului, am generat in „service-flight” cele doua metode cea de preluare a datelor „getFlightId()” si cea de update „updateFlight()” pe care le vom apela ulterior in fisierul TypeScript al componentei de update prin intermediul metodelor ngOnInit() unde am extras valoarea parametrului „id” al rutei curente folosind sintaxa this.route.snapshot.params[‘id’] pentru a afisa informatiile in formular despre zborul cu id-ul respective, si a metodei onSubmit() pentru a salva modifcarile realizate.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 30. Lista de zboruri din cadrul Admin Dashboard. Sursa proprie.

A screenshot of a computer

Description automatically generated with low confidence

Figură 21. Formularul de update autocompletat cu informatiile din tabel. Sursa prorpie.

A screenshot of a phone

Description automatically generated with low confidence

Figură 22. Update efectuat asupra zborului cu id 1. Sursa proprie.

A screenshot of a computer

Description automatically generated with low confidence

Figură 24. Lista zborurilor dupa efectuarea update-ului. Sursa proprie.

4.8 Eliminarea/Stergerea unui zbor

Eliminarea unui zbor se va face din tab-ul dashbordului de catre admin apasand pe butonul „Delete”.

Astfel in interiroul „AdminController” s-a dezvoltat functia „deleteFlight()” pe care am adnotat-o cu @DeleteMapping pentru a marca o cerere HTTP Delete. Aceasta metoda are ca scop stergerea unui zbor in functie de parametrul de cautare variabil id.

In Angular, in serviciul „search-flight” am adaugat o metoda care va avea ca scop apelarea endpoitului din backend. Aceasta metoda fiind ulterior folosita in fisierul TypeScript al dashboard-ului intr-o alta metoda deleteFlight() la finalul careia vom returna noua lista dupa stergerea zborului.

Aceasta ultima metoda o vom asocia butonului de Delete din dashboard.

A screenshot of a computer

Description automatically generated

Figură 25. Stergerea zborului cu id 13. Sursa proprie.

4.9 Rezervarea unui zbor

Pentru rezervarea unui zbor am inceput prin adaugarea in backend a clasei Booking, care va contine variabile precum „id” care a fost si adnotata cu @Id pentru a marca faptul ca acest camp este cheia primara in baza de date. Campurile userid si flightsid reprezinta relatia many to one intre rezervare si utilizator, respectiv zbor. Pe acestea le-am adnotat cu „@ManyToOne” si „@JoinColumn” pentru a specifica coloanele de legatura in tabelul booking, iar restul campurilor: leavinfrom, arrivingat si leavingdate reprezinta informatii suplimentare pe care le vom folosii pe partea de frontend pentru a prelua datele zborului si pe partea de rezervare.

Mergand mai departe la serviciul dedicat rezervarilor, am realizat o metoda bookFlight() care primeste doi parametrii „userId” si „flightId” reprezentand identificatorii userului si a zborului pentru care se va face rezervarea. Cei doi identificatori sunt gasiti cu ajutorul repositoriului „userRepo” respectiv „flightRepo” in cazul in care acestia nu sunt gasiti se va arunca cate o exceptie „RuntimeException” pentru fiecare. Tot in cadrul acestei metode vom obtine si numarul curent de locuri disponibile din obiectul „flight” pentru a anticipa dezvoltarea privind updata-rea locurilor unui zbor in functie de numarul de boarding passuri pe care le vom inregistra pe o rezervare. Insa vom trata mai pe larg aceasta dezvoltare in urmatorul capitol. La finalul acestei metode vom crea o noua instana a clasei „Booking” si vom seta proprietatile mentionate mai sus, urmand sa salvam rezervarea folosind metoda save din cadrul „bookingRepo” si returnand rezervarea salvata.

Pentru controler, am adaugat in controllerul dedicat rezervarilor, o metoda bookFlight() pe care am marcat-o cu @PostMapping. Aceasta metoda primeste ca parametrii „userId” si „flightId”, parametrii de cale, urmand sa apelam metoda bookFlight() din cadrul serviciului pasand parametrii si returnand rezultatul.

In angular am inceput prin generarea serviciului pentru rezervare in cadrul caruia am construit metoda bookFlight() tot cu cei doi parametrii ca si mai sus. Aceasta metoda construieste URL-ul pentru a efectua cererea POST catre serviciul de rezervari folosind sablonul bookingURL adaugat ca si variabila separata si parametrii „userId” si „flightId”. Rezultat intors trebuie sa fie sub forma de obiect Observable de tipul Booking, insemnand ca asteapta un obiect de tip Booking ca raspuns de la server.

In fisierul TypeScript al componentei booking am adaugat proprietatile:

* userId;
* flightId;
* arrivingat;
* leavingfrom;
* flight;
* boardingPass;

pe care le vom folosii in template-ul HTML si in logica componentei. Constructorul primeste instante ale serviciilor necesare prin injectia de dependeta.

Prima metoda descirsa este ngOnInit() care este apelata automat dupa ce componenta este initalizata si preia flightId-ul din ruta curenta si userId-ul din serviciul de stocare. Apelul „searchflightservice.getFlightId” va genera o cerere catre server pentru a obtine detaliile zborului specificat de flightId si actualizeaza proprietatea flight cu datele obtinute.

A doua metoda onSubmit() este apelata cand utilizatorul trimite formularul de rezervare. Aceasta foloseste serviciul bookginServie pentru a efectua cererea de rezervare a zborului.

4.10 Adaugarea boarding pass-ului in rezervare.

Deoarece o rezervare se poate face pentru mai multe persoane, am adaugat si conceptul de boarding pass. Astfel pentru fiecare rezervare putem avea pana la maxim 3 boarding passuri adaugate.

Am generat prima data in pachetul domain din backend, entitatea BoardingPass care va contine atributele: id, name, identifycard si booking. Cea din urma reprezinta rezervarea ascitata boarding pass-ului si a fost mapata printr-o relatie @ManyToOne cu clasa Booking. Relatia este definita prin intermediul coloanei „booking\_id” in baza de date. Astfel in cadrul clasei Booking, am adaugat si o metoda addBoardingPass() care perminte adaugarea unui boarding pass la rezervare. Metoda verifica daca numarul de boarding pass-uri existente depaseste 10 si arunca o exceptie in caz afirmativ. Apoi va adauga boarding pass-ul la lista boardingPasses si seteaza rezervarea curenta ca referinta.

Am definit si repositoriul „BoardingPassRepo” , iar in cadrul repositoriului Booking am adaugat o metoda countBoardingPassesByFlightsId(), care va utiliza o interogare personalizata pentru a numara boarding pass-urile asociate unui anumit zbor. Metoda va returna un intre care va reprezenta numarul de boarding pass-uri asociate zborului specificat.

Urmatorul pas a fost ca in cadrul serviciului rezervarilor sa adaug si metoda addBoardingPassToBooking(), care va cauta intial in baza de date rezervarea pe baza id-ului furnizat „bookingId” utilizand metoda findById din cadrul repositoriului de rezervari. Daca rezervare nu este gasita se va arunca o exceptie si vom face ulterior si verificarea daca numele si cardul de itentitate sunt nenule, aruncand o exceptie si in acest caz daca unul dintre eleste este null. Vom salva boarding pass-ul in baza de date utilizand metoda save(), adaugand-ul ulterior si in lista de boarding pass-uri a rezervarii prin metoda addBoardingPass definita in entitatea Booking. De asemenea vom obine zborul asociat rezervarii si vom actualiza numarul de locuri disponibile in functie de numarul total de boarding pass-uri adaugate rezervarii. In final vom salva rezervarea in baza de date utilizand metoda save().

Revenind la controllerul pentru rezervari a fost creat un endpoint de tip POST care permite aduagarea unui boarding pass la o rezervare existenta in sistem. Endpoint-ul primeste id-ul rezervarii si obiectul „boardingPass” care contine infromatii despre pasagerul pentru care se adauga boarding passul. Se va apela metoda addBoardingPassToBooking() din serviciul „bookingService” pentru a adauga boarding passul la rezervare.

Pentru partea de frontend am adaugat in cadrul serviciului de rezervari metoda addBoardingPassToBooking() care este responsabila cu adaugarea unui boarding pass la o rezervare existenta. Aceasta metoda primeste 2 parametrii: id-ul rezervarii si un obiect boardinPass. Cu aceasta ocazie vom genera si clasa BoardingPass cu cele 2 variabile name si identifycard. Revenind la metoda aceasta va construii URL-ul corespunzator cererii catre server.

Ultimul pas a fost acela de a adauga in componenta TypeScript a rezervarii metodele:

* addBoardingPass() – care este apelata atunci cand utilizatorul doreste sa adauge un nou boarding pass. Iar in cadrul ei se va verifica daca numarul de boarding pass-uri adaugate nu depaseste limita de 3. In caz contrar vom afisa un mesaj de avertizare.
* removeBoardingPass() – care este responsabila cu eliminarea unui boarding pass din lista. Eliminarea se va face pe baza index-ului acestuia.
* addBoardingPassToBooking() – este utilizata pentru adaugarea boarding pass-ului la rezervare, parcurgand lista si verificand ca sunt completate numele si cardul de identitate. Daca aceste informatii sunt disponibile, se construieste un obiect, „Bordingpass” si se utilizeaza serviciul de rezervare pentru a adaugat un boarding pass la rezervare prin intermediul metodei addBoardingPassToBooking().

5. Concluzii și implementări viitoare

# Bibliografie

|  |  |
| --- | --- |
| [1] | Visual Studio Code, 08 2022. [Interactiv]. Available: https://code.visualstudio.com/docs. |
| [2] | Angular, 08 2022. [Interactiv]. Available: https://angular.io/guide/architecture. |
| [3] | J. A., 07 2022. [Interactiv]. Available: https://www.hostinger.com/tutorials/what-is-bootstrap/. |
| [4] | [Interactiv]. Available: https://www.javatpoint.com/java-architecture. |
| [5] | „terasolunaorg.github.io,” [Interactiv]. Available: https://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.html. |
| [6] | R. GARCIA, „JWT tokens and security – working principles and use cases,” [Interactiv]. Available: https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/. |
| [7] | [Interactiv]. Available: https://app.contentful.com/. |