**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 și de gestiune al zborurilor. Obiectivul este acela de a implemneta o aplicație web completă, complexa și usor de manevrat pentru toate tipurile de utilizatori, dar și 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 interfață dinamică și 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 adăugarea componentelor de interfată, dar cuprinde si o serie larga de librarii utile în implementarea de noi funcționalități.

A doua categorie a proiectului, Backend-ul, a fost realizată 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

## *Motivarea alegerii temei*

Motivul alegerii unei aplicatii Web în domeniul aviatic ca si tema de disertație a aparut datorită domeniului în care am studiat și datorită dorinței de a realiza o astfel de aplicație utilizând diferite tehnologii care combinate sa produca un produs finit.

Mai mult decat atât aceste aplicații sunt foarte des intâlnite în zilele noastre și acest fapt se datoreaza avantajelor acestora:

* Compatibilitatea dintre sisteme – majoritatea aplicațiilor web trebuie să indeplinească o cerința minimă și anume existenta unui browser precum (Micrososft Edge, Google Chrome, Safari), astfel indiferent de sistemul de operare folosit aplicațiile web pot rula cu usurintă.
* Reducerea Costurilor – aplicațiile web pot reduce costurile atat pentru furnizor datorită costurilor mici pentru suport dar și pentru partea de intreținere cât și pentru utilizator deoarece sistemul acestuia are nevoie de specificații minime pentru a putea utiliza aplicația.
* Mult mai gestionabil – dupa cum am spus mai sus aplicațiile web necesită cerințe minime din partea stațiilor de lucru ale utilizatorului și împreună cu avantajul că acestea sunt instalate doar pe server, actualizarea sistemului se poate face foarte rapid și foarte simplu.

## *1.2 Prezentarea generală 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 următoarele zone care vor conține diferite funcționalitați:

* Info unde:
* userii vor putea descoperii informații despre avioanele puse la dispoziție de către companie, cu specificațiile fiecaruia dintre ele;
* detalii despre clasele de travel ( 3 la numar ) : Economy, Premium Economy și Bussines Class;
* Următoarea secțiune este Fly unde:
* se vor afla informații despre Check-in, statusul zborului și mobile check-in;
* A treia secțiune prezentă este Plan unde:
* se vor afla informații legate de bagaje și cerinte speciale pentru: animale, persoane cu probleme medicale;
* A patra secțiune este secțiunea de Book unde:
* se poate face booking unui zbor;
* fiecare utilizator iși poate vizualiza zborurile rezervate;
* A cincea sectiune va face tot din zona de booking unde se vor putea adauga bording pass-uri pentru maxim 3 persoane.
* 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”;
* În cadrul „Dashboard Flights” acestuia i se va deschide o listă cu toate zborurile unde se vor putea updata și șterge zborurile;
* In cadrul „Create a new Flights”, adminul poate adaugă un zbor nou care se va vedea imediat în lista de zboruri;

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 reprezintă un editor de cod sursă foarte puternic reușind să ruleze pe toate sistemele de operare. Acest tool este foarte utilizat datorită suportului cu care vine încorporat pentru JavaScript, TypeScript si Node.js. Cu toate acestea a devenit foarte populat și pentru limbaje de programare precum Java, C# sau Python datoritș ecosistemului foarte bogat de extensii pentru aceste limbaje. [1]

3.1.1 Angular

Angular este o platformă de dezvolatare construita de către TypeScript. Acesta este proiectat să trimita update-urile făcute cat mai rapid, având astfel posibilitatea să profiti de acestea aproape instant.

Fiind o platformă, Angular include:

* Un framework pentru construirea de aplicații web scalabile;
* O colecție variată de librarii integreate care ajută la implementarea diferitor caracteristici;
* O multitudine de instumente care te pot ajuta sa iți dezvolți, constuiești și testezi codul.

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

* Modulele – Aplicatiile Angular sunt modulare și astfel Angular are propriul sistem de modularitate numit NgModules. NgModules reprezintă containare pentru un anumit bloc de cod dedicat unui domeniu al aplicației, sau a unui workflow. Fiecare aplicație Angular are cel putin un modul, care reprezintă rădacina și care este denumită AppModule, reprezentând mecanismul de bootstrap care lansează aplicația.
* Componentele – reprezintă blocurile din care este compusă aplicația. Componenta include o clasa TypeScript reprezentată de o adnotare @Component(), un template HTML și stilurile. Clasa @Component() specifică următoarele informații:

1. Un selector CSS care definește modul în care componenta este folosită în template.
2. Un template HTML care indică cum sa fie randată componenta;
3. Un set de stiluri CSS care ajută la design-ul elementelor HTML.

* Servicii și Dependency Injection – pentru date sau logică care nu sunt imparțite între componente, se va crea o clasă service. O astfel de clasă este imediat precedată de adnotarea @Injectable(). Această adnotare oferă metadatele care permit altor furnizori să fie injectați ca și dependinte în clasa ta. Cu ajutorul Dependency Injection componentele vor fi mult mai eficiente deoarece nu vor prelua datele de pe server și nu se vor conecta direct la consola aceste sarcini se vor delaga către servicii. [2]



Figură 2. Diagrama Arhitecturală a unei Aplicați Angular [2]

3.1.2 Bootstrap

Bootstrap este un framework open source de frontend, care conține elemenete de baza pentru dezvoltarea aplicațiilor web receptive. Acest framework este format pe baza HTML, CSS și JavaScript.

Cu ajutorul Bootstrap se poate detecta dimensiunea ecranului pe care o folosește utilizatorul, adaptând dimensiunile componentelor aplicației web automat. Pe lânga aceasta funcționalitate de bază, printre componentele incluse în interfețele Bootstrep se regăsesc bările de navigație, carusele de imagini și butoane.

Printre avantajele utilizării acestui framework se regăsesc:

* Fiind responsive, ajută la performanțele aplicației web deoarece tot procesul de optimizare este cu mult ameliorat.
* Compatibilitatea cu diferite browsere web;
* Fiind un framework foarte popular documentația este bogată, iar datorită acestui lucru se pot gasi foarte usor exemple de cod care trebuie doar adaptate la necesitatile fiecăruia.

Există și mici dezavantaje în folosirea Bootstrap cum ar fi:

* Necesitatea ca browserele să fie updatate, deoarece dacă se folosește o versiune nouă a frameworkului, iar versiunea browserului este mai veche vor exista discrepanțe.
* Dacă se va utiliza un număr mare de funcții va rezulta la dimensiuni ale fisierelor foarte mari ceea ce poate duce la îngreunarea proiectului. [3]

3.2 Backend

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

Printre caracterișticile sale cele mai importante se numără:

* Integrarea cu instrumente de control al versiunii precum Git, SVN;
* Suport pentru dezvoltarea de aplicații web prin framework-uri populare precum Spring având capacitatea de a rula și testa aplicațiile local sau pe servere externe.

3.2.1 Java

Java este un limbaj de programare orientat pe obiecte. A fost lansat pentru prima dată in anul 1995 și a devenit foarte popular în lumea dezvoltării software, fiind utilizat într-o foarte mare varietate de aplicații și sisteme.

Este un limbaj de programare compilat și interpretat, ceea ce înseamnă că codul sursă Java este transformat într-un format intermediar (bytecode) care apoi este interpretat de masina virtuală Java (JVM) pentru a rula programul. Astfel aplicațiile Java pot rula pe aproape orice platformă, fiind foarte usor să fie portabile.



Figură 3. Arhitectura Java [4]

3.2.2 Spring

Spring Framework este este un framework open-source pentru dezvoltarea aplicațiilor în Java. Acesta oferă o gamă lagă de funcționalități care fac ca dezvoltarea aplicațiilor web să fie mai usoară și mult mai eficientă. Printre acestea se numeră:

* Inversion of control (Inversiunea de control) și dependency injection (injecția de dependințe). Spring usurează crearea obiectelor și gestiunea dependențelor dintre acestea prin intermediul unui container care se ocupă cu crearea, configurarea și gestionarea ciclului de viața al acestora.
* Securitatea: Spring pune la dispoziție diferite instrumente pentru securizarea aplicațiiei atât din punct de vedere al autentificării cât și din punct de vedere al autorizării.
* Accesul bazei de date: Spring oferă un set de instrumente pentru accesarea bazelor de date, prin intermediul framework-ului Hibernate.

3.2.3 Hibernate

Hibernate oferă o soluție de mapping obiect-relational (ORM) care facilitează interacțiunea dintre bazele de date relaționale și obiectele create în Java. Astfel programatorul lucreaza într-un mod mai natural, folosind obiecte și metode, în loc să scrie interogări SQL.

Hibernate este proiectat astfel încât să se poată integra cu alte framework-uri și librării precum Srping și Struts.

3.3.3 Spring Security

Spring Security oferă un set de instrumente pentru ajutorul implementării funcționalităților de autentificare, autorizare și gestionare a sesiunilor în aplicațiile 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 aplicațiilor web în Java.

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

* Model – reprezentând obiectele și logica acestora;
* View – reprezentând aspectul vizual al aplicației și defineste modul în care datele sunt afișate utilizatorului.
* Controller – gestionează interacțiunea dintre model și vizualizare. Aici se primesc cereri HTTP de la utilizatori și se face procesarea datelor.

Spring REST ( Representational State Transfer) este o extensie a framework-ului Srping MVC axându-se pe dezolvatarea serviciilor RESTful în Java. Aceste servicii web reprezintă o modatlitate foarte populară de a construii servicii web care se bazează pe protocale HTTP pentru a permite mai ușor comunicarea dintre client și server. Prin această extensie serviciile web RESTful pot comunica cu alte aplicații și 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 relațional și open source, folosit pentru stocarea și gestionarea datelor într-un mod eficient. Este una dintre cele mai populare și utilizate baze de date în aplicațiile web datorită faptului că oferă o performanță ridicată și securitate. Este compatibil cu o multitudine de sisteme de operare printre care Linux și 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 securizată a informațiilor între parți sub forma de obiecte JSON;

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

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

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

În exemplul de mai sus alg reprezintă algoritmul folosit pentru a semna sau cripta tokenul și typ reprezintă tipul de informație semnat sau criptat de obicei este JWT.

Payloadul: reprezintă chiar informațiile/datele ce trebuie transferate folosint tokenul. Această parte se mai numește și "claims". Claims-ul poate să fie de 3 tipuri: registered, public și privat.

Signature: aceasta ultimă parte din token este folosită pentru a verifica dacă mesajul ce trebuia transmis nu a suferit modificări pe parcurs.

Primul pas pentru implementarea JWT a fost de a genera clasa WebSecurityConfig în cadrul căreie am definit metoda SecurityFilterChain. SecurityFilterChain reprezintă un lanț de filtre care urmareză să fie aplicate pentru asigurarea securitații unei aplicații.

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

Tot în cadrul acestei clase a fost definită și metoda PasswordEncoder care are rolul de a cripta parola, pentru o mai mare protecția asupra atacurilor cibernetice.

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

* createToken();
* generateToken() aceasta metodă are ca parametru un nume de utilizator (userName) și cu ajutorul metodei createToken are ca rol generarea și returnarea unui token semant care va conține toate informațiile despre user;
* validateToken aceasta metodă verifică dacă tokenul este valid. Tokenu este considerat valid dacă subiectul corespunde cu numele utilizatorului fiind folosită metoda extractUsername;
* extractUsername;
* extractClaim;

A treia și ultima etapă din implementarea JWTRequestFilter. Clasa JWTRequestFilter este responsabilă de verificarea tokenului în fiecare cerere HTTP. Aceasta extinde interfața „OncePerRequestFilter” cara asigură că filtrul este aplicat o singură dată pe fiecare cerere.

Metoda doFilterInternel() este suprascrisă pentru a efectua logica filtrului în interiorul acestei clase. Aceasta primește cererea HTTP(HttpServletRequest) raspunsul HTTP(HttpServletResponse) și lanțul de filtre( FilterChain). În interiorul acestei metode se extrage header-ul „Authorization” din cerere și se verifică dacă acesta incepe cu prefixul „Bearer”. Daca această condție este îndeplinită se extrage numele utilizatorului folosind „jwtUtil.extractUsername(token)”.

Dacă numele utilizatorului este disponibil și autentificarea actuală nu există, se incarcă detalii despre utilizatorul respectiv utilizând metoda loadUserByUsername din cadrul userDetailsService. Următorul pas este de validare a tokenului. Dacă tokenul este valid se crează un obiect UsernamePasswordAuthenticationToken care conține informații despre utilizator precum username, role, parola, iar aceste informații sunt stocate în „SecurityContextHolder”.

A diagram of a server

Description automatically generated with low confidence

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

4.2 Inregistrarea unui nou user

Pentru această etapa am implementat prima data baza de date a userilor, care va conține urmăotarele 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 servească ca și domeniu în aplicație pe partea de backend. Pentru atributul Id, am folosit „annotarea” @Id care indică fa ptul că membrul asupra caruia este pusă annotarea reprezintă cheia primară a entiații user. Tot pentru această coloana, s-a folosit și annotarea @GeneratedValue care ne ajută la modul în care se incrementează o anumită coloană. Restul atributelor au fost annotate cu operatorul @Column care ajută la maparea dintre atribute și coloanele bazei de date.

După cum am menționat mai sus în cadrul tabelei se regăseste coloana user\_role. Acest atribut este deifnit ca având tipul UserRole însemnând că acesta poate fi setat utilizând una dintre valorile definite în enumerația UserRole în cadrul căreia se regăsesc 2 constante: „ADMIN” și „USER”.

În continuare am definit servici-ul, ce reprezintă o clasă care oferă funcționalitați specifice pentru aplicație și care este utilizată pentru a gestiona logica de business și interacțiunea cu alte componente ale aplicației, cum ar fi repository-ul. Serviciile sunt implementate ca și calse în Java, fiind utilizate pentru a separa logica de business de infrastructură.

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

În ultimul pas al implementării în cadrul backed-ului am definit controlerul, unde am realizat metoda signupUser() annotata cu @PostMapping. Această annotare este folosită pentru a mapa o cere HTTP POST la o metodă specifică intr-un contoller.

În Spring Framework, un controller este o clasă Java care gestionează cererile HTTP primite de la serverul web și produce răspunsurile corespunzătoare. Controllerul este utilizat pentru a crea interfața partea vizuală a apicației și partea de business.

Metoda signupUser() din cadrul controllerul, va primi obiectul RegisterRequest ca un parametru trimis prin corpul cererii HTTP și va crea un nou utilizator folosind informațiile furnizate de client. RegisterRequest a fost gândit precum un DTO (Data Transfer Object), reprezentând o clasă simplă conținând doar 3 atribute: email, password și cpassword și este folosită pentru a evita expunerea directă a obiectelor.

Mergând mai departe pe partea de frontend, aici primul pas a fost de a genera component register în Angular. În cadrul acesteia am modelat fișierul html și css pentru a reda o interfață placută utilizatorului.

Graphical user interface, application

Description automatically generated

Figură 7. Interfața de înregistrare. Sursă proprie.

A doua etapă a fost de a genera clasa aferentă inregistrării utilizatorului. Aceasta va conține cele 3 atribute pe care le vedeți și în imagina de mai sus: email, password și cpassword. Această clasă este utilizată în servicul de inregistrare a utilizatorului din Angular.

în Angular, un seriviciu reprezinta o clasa TypeScript care furnizează anumite funcționalități, putând fi injectată în alte compontente sau chiar servicii pentru a impartășii funcționalitățile cu acestea. În majoritatea timpului serviciile sunt utilizate pentru a efectua operații asincrone pentru gestionarea datelor într-un mod consistent și pentru a efectua interacțiuni cu serverul prin apelurile de tip HTTP.

În interiorul serviciului de înregistrare am creat metoda registerUser care va primi obiectul/clasa User care a fost menționată în a doua etapă și care va avea ca scop de a face apelarea cștre backend pentru înregistrarea unui nou utilizator.

Penultima etapă a acestei implementări a fost ca în cadrul fisierul TypeScript din interiorul componentei register să generăm o metodă numită userRegister() care va avea ca scop apelarea metodei din service și de a trimite un mesaj în momentul în care userul a fost înregistrat cu succes.

Ultima etapă a constat în adăugarea funcționalităților în cadrul fișierului html, folosind directiva ngSubmit care este des utilizată în Angular pentru a detecta și gestiona evenimentul de submit a unui formular HTML. Aceasta a apelat metoda realizată în cadrul fișierul TypeScript de înregistrare a unui utilizator.

Graphical user interface, application

Description automatically generated

Figură 8. Exemplu înregistrare user. Sursă proprie.

A screenshot of a computer

Description automatically generated

Figură 9. Înregistrarea cu succes a unui user. Sursă proprie.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 10. Inserarea automată după înregistrarea cu succes a unui user în tabela users. Sursă proprie.

## *4.3 Logarea unui utilizator în aplicație*

Pentru logarea utilizatorilor m-am ajutat de implementarea de la secțiunea 4.1. În crearea tokenului din interiorul clasei JwtUtil, am adăugat și rolul utilizatorului preluat din baza de data. Am adaugăt în interiorul UserController-ului metoda createAuthenticationToken() asupra căreie am folosit adnotarea @PostMapping(„/login”) pentru a mapa metoda pentru a gestiona solicitările HTTP POST către URL-ul /login.

Metoda primește un obiect LoginDTO, obiect care conține datele de logare a unui utilizator, precum și un obiect ‚HttpServletResponse” care este utilizat pentru a trimite răspunsul înapoi către client.

Am construit un bloc try unde se incearcă autentificarea utilizatorului folosind un obiect UsernamePasswordAuthentication creat cu numele de utilizator în cazul nostru emailul și parola. Dacă autentificarea esuează din cauza credențialelor greșite, este aruncată o excepție „BedCredentialsException” cu un mesaj de eroare.

Dacă autentificarea are loc cu succes, se încarcă detaliile utilizatorului folosind numele utilizatorului ca și parametru pentru identificarea acestuia în baza de date.

În final se generează un nou obiect JSON în corpul răspunsului. Acesta va conține ID-ul utilizatorului și rolul acestuia. De asemenea, adaugă 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, putând astfel să il injectez în alte componente ca dependința.

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

* saveUserId() – salvează un id al unui utilizator în LocalStorage;
* saveUserRole() – salvează rolul utilizatorului în LocalStorage;
* saveToken() – salvează tokenul de autentificare în LocalStorage;
* hasToken() – verifică dacă tokenul este prezent în LocalStorage;
* getToken() – returnează tokenul din LocalStorage;
* signOut() – sterge tokenul și id-ul din LocalStorage, deconectând userul;
* isUserLoggedIn() – verifică dacă utilizatorul este logat și dacă are rolul de “USER”;
* getUserRole() – returnează rolul din LocalStorage;
* getUser() – returnează id-ul din LocalStorage;
* isAdminLoggedIn() – verifică dacă utilizatorul este autentificat și dacă are rolul de “ADMIN”;

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

Pentru a putea realiza autorizarea pentru fiecare endpoint am generat în serviciile create, o metodă createdAuthorizationHeader() care va returna un obiect HttpHeader care va conține un header de autorizare setat. Tokenul pentru header-ul de autorizare fiind obținut din serviciul de stocare menționat mai sus.

Ulterior am generat și serviciul de logare care va conține o metodă login() cu doi parametrii username și parola. Aceasta folosește serviciul HttpClient pentru a face o solicitare HTTP POST la un URL specificat, trimițând cei doi parametrii ca payload al cererii. După ce raspunsul este primit vom folosii un operator RxJS, ‚map’ pentru a salva id-ul și rolul utilizatorului în LocalStorage folosind metodele din cadrul StorageService și vom extrage tokenul de autentificare din header-ul ‚Authorization’ al raspunsului indepartând stringul ‚Bearer ’ și salvând 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 fișierele aferente. Am construit un formulat în fișierul html, urmând ca în fișierul TypeScript în metoda ngOnInit(), să construiesc formularul folosind FormBuilder. Acest formular va conține două câmpuri, username și parola, ambele campuri fiind obligatorii.

Am generat și metoda userLogin() care este apelată în momentul în care utilizatorul doreste să se autentifice. Acesta va folosii serviciul de logare pentru a trimite datele formularului către server și de a încerca autentificarea. Dacă autentificarea are loc cu succes, aceasta verifica rolul utilizatorului și îl redirecționează către pagina dorită, fie către pagina de ‚/welcome’ în cazul utilizatorului, fie către pagina de ‚/admin/dashboard’ în cazul administratorului.

A screenshot of a computer

Description automatically generated

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

4.4 Blog dinamic

Această implementare a fost gândită cu scopul de a informa utilizatorii despre noile noutati în care este implicată compania.

Pentru aceasta mică 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.

Într-o primă etapă am generat un Content Model în platformă menționând campurile pe care fiecare Content îl va avea.

Graphical user interface, text, application, email

Description automatically generated

Figură 12. Generare Content Model platformă Contenful [7].

După ce au fost generate Content-urile, am generat în cadrul proiectului un serviu în Angular pentru a realiza comunicarea între aplicație și platforma Contentful.

În cadrul serviciului, am creat o variabilă client care va stoca SpaceID-ul și Access Tocken-ul pentru utilizatorul platformei, astfel reușind să facem conexiunea la spațiul de lucru generat, urmând să generăm o noua metoda numita getAllEntries() pe care o vom apela în cadrul componentei news care ne va ajuta sa returnăm toate „Contenturile” de pe platformă pentru spațiul de lucru indicat însa sub formă de json.

În a doua etapă am folosit directiva ngFor pentru a itera prin elementele jsonului, putând astfel să afișam imagina,titlul, summary alocate fiecărui „Content”.

Graphical user interface, application

Description automatically generated

Figură 13. Dynamic Blog. Sursă proprie.

4.5 Căutarea unui zbor

Pentru această funcționalitate am generat în primă instanță compontenta flight-search în Angular. În fișierul html am construit formular-ul de căutare care va conține 3 câmpuri:

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

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

În fișierul typescript din componenta flight-search, în cadrul metodei ngOnInit am setat variabila „now” care va reprezenta data curentă în care ne aflăm. Aceasta este folosită în cadrul fisierului html pentru „Departure Date” pentru a seta în calendar data curentă astfel nu vom putea alege o dată mai veche pentru care să cautam zboruri. Metoda onSubmit() va apela la randul ei metoda GetFlight() din cadrul serviciului search-flight și va returna rezultatul sub formă de tabel.

Implementarea back-endului pentru această parte a constat în generarea unei clase domeniu Flight care va avea ca rol generarea tabelei în cadrul careia se afla 5 variabile: id, arrivingat, leavingfrom, leavingat si nrseats. După această prima etapa a fost construit DTO-ul acestei clase care va conține aceleași variabile mai putin id, repositoriul în care am generat metoda „findByLeavingfromAndArrivingatAndLevingdate” care va genera selectul pentru returnarea tuturor zborurilor în funcție de data de plecare, locul de plecare și locul de aterizare urmând a fi folosit în service-ul „FlightService” unde se va întoarce sub formă de listă toate zborurile găsite. Iar în ultimul pas s-a generat controller-ul ”FlightController” în care vom folosi @GetMapping pentru metoda getFlights. Acest endpoint este folosit în service-ul din frontend.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Figură 14. Căutarea unui zbor. Sursă proprie.

A screenshot of a computer

Description automatically generated

Figură 15. Tabela de zboruri. Sursă Proprie.

4.6 Adăugarea unui zbor nou

Adăugarea unui unui zbor nou se va putea face doar de către userul cu drep de admin.

Iar secțiunea de adăugarea unui zbor se va afla în bara de navigare specifică adminului (dashboardul adminului).

Pentru adăugarea unui nou zbor am adăugat în controlerul dedicat adminului o metodă numită createFlight() pe care am adnotat-o cu @PostMapping pentru a mapa o cerere HTTP Post la această metodă. Ruta către această metoda fiind „/create/flight”.

În cadrul acestei metode am folosit @RequestBody pentru a marca faptul că paramentrul flights de tipul Flights este legat de cerearea HTTP. Logica acestei metode este destul de simplă se va folosii metoda save() din cadrul repositoriului flightRepo care va adaugă un zbor nou în baza de date.

Ajungând la partea de frontend, am generat prima data componenta de createFlight, în cadrul căreia am adăugat un formular nou care va conține câmpurile unui zbor. Am realizat cu ajutorul directivei [(ngModel)] legătura bidirectională între valorile inputului și variabilele clasei FlightSearch generând un obiect de tipul FlightSearch în interiorul fișierului TypeScript al componentei de creare.

Al doilea pas a fost de a adauga metoda de creare a zborului în fișierul search-flight.service.ts unde vom returna răspunul cererii Post din cadrul backendului. Ulterior vom utiliza această metodă în cadrul metodei onSubmit() din fișierul TypeScript al componentei createFlight unde vom afișa și un mesaj „Flight Created with Success” în momentul în care zborul se va înregistra. Metoda onSubmit() este o metodă asociată de obicei cu evenimentul de trimitere a unui formular.

A screenshot of a computer

Description automatically generated

Figură 16. Adăugarea unui nou zbor. Sursă proprie.

A screenshot of a computer

Description automatically generated

Figură 17. Tabela flights, unde putem vedea adăugarea zborului. Sursă proprie.

4.7 Update-ul unui zbor nou

Update-ul unui zbor se va putea face doar de către admin, acesta având o secțiune dedicată în cadrul barii de navigare.

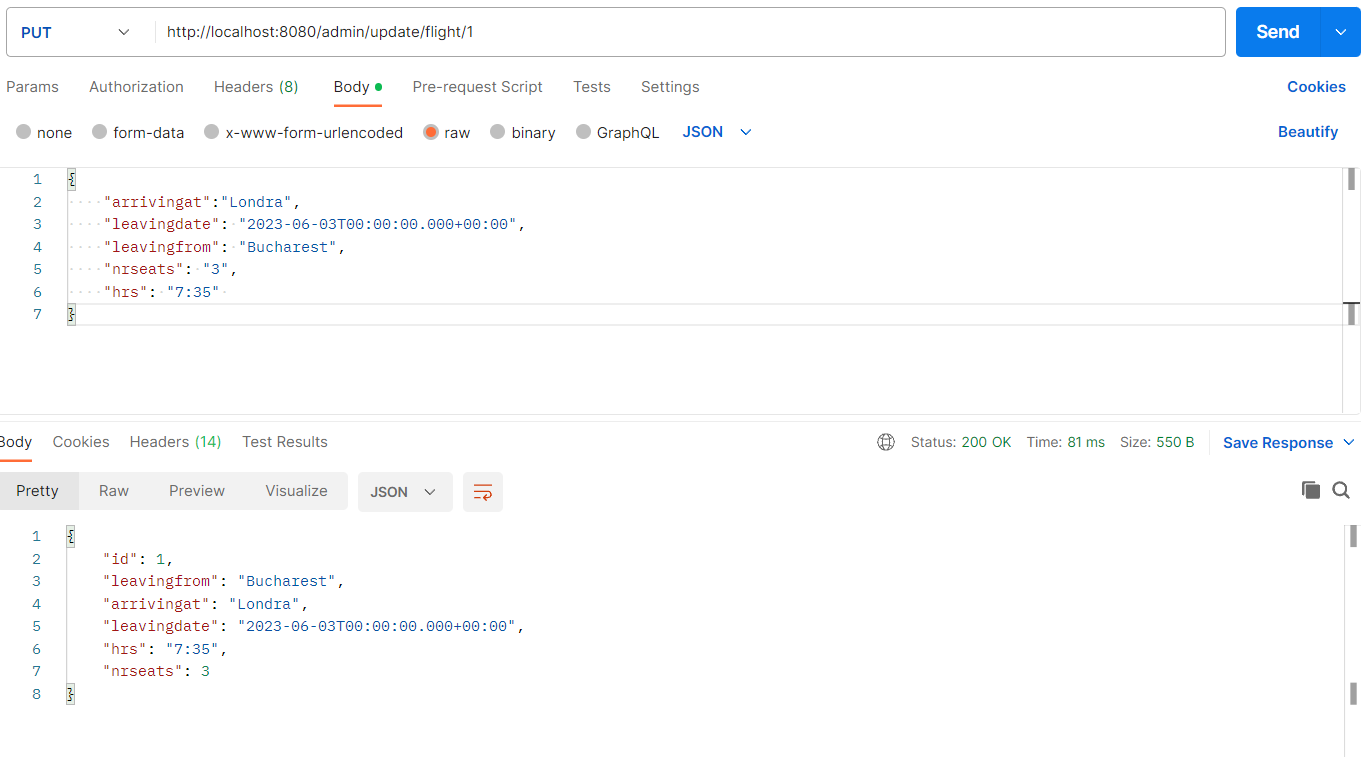
Pentru această acțiune am generat în cadrul controler-ului adminului, o metodă „updateFlight()” pe care am adnotat-o cu @PutMapping pentru a mapa o cere HTTP de tip PUT. Aceast update se va face pentru câte un zbor, variabila de identificare al acestuia fiind id-ul, iar în cazul în care nu vom găsi zborul cu id-ul trimis, se va întoarce o eroare „Flight does not exist with id”. În cazul în care vom găsi id-ul, va face un update pe atributele modificate ale zborului.

Initial am testat această funcționalitate în Postman, unde am updatat câmpul de hrs pentru prima valoare din tabela flights, care inițial era setat pe null a fost setat la ora 07:35.

A screenshot of a computer

Description automatically generated

Figură 18. Tabela flights, înainte de a efectua update-ul. Sursă proprie.



Figură 19. Update-ul generat din Postma. Sursă proprie.

A screenshot of a computer

Description automatically generated

Figură 20. Tabela flights după efectuarea update-ului. Sursă proprie.

Am continuat prin adăugarea unei a doua metode însa de aceasta data în cadrul „FlightController” pentru a afișa un zbor în funcție de id. Această metodă a fost adnotată cu @GetMapping și în funcție de id-ul pe care îl trimitem în cadrul URL-ului se va întoarce zborul respectiv. În caz contrar se va întoarce o eroare.

Această metodă o vom folosii în cadrul frontend-ului în momentul în care vom apăsa butonul „Update” vom naviga automat către un formular, unde campurile zborului respectiv se vor completa automat, urmând ca adminul să efectueze modificările necesare.

După cum am menționat mai sus pentru secțiunea de frontend a acestei implementări am generat o nouă componentă „updateflight” unde am schițat prima dată un formulat în cadrul fișierului html. După finalizarea aspectului formularului, am generat în „service-flight” cele două metode cea de preluare a datelor „getFlightId()” și cea de update „updateFlight()” pe care le vom apela ulterior în 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 afișa informațiile în formular despre zborul cu id-ul respectiv și a metodei onSubmit() pentru a salva modifcările realizate.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 21. Lista de zboruri din cadrul Admin Dashboard. Sursă proprie.

A screenshot of a computer

Description automatically generated with low confidence

Figură 22. Formularul de update autocompletat cu informațiile din tabel. Sursă prorpie.

A screenshot of a phone

Description automatically generated with low confidence

Figură 23. Update efectuat asupra zborului cu id 1. Sursă proprie.

A screenshot of a computer

Description automatically generated with low confidence

Figură 24. Lista zborurilor după efectuarea update-ului. Sursă proprie.

4.8 Eliminarea/Ștergerea unui zbor

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

Astfel în interiroul „AdminController” s-a dezvoltat funcția „deleteFlight()” pe care am adnotat-o cu @DeleteMapping pentru a marca o cerere HTTP Delete. Această metodă are ca scop ștergerea unui zbor în funcție de parametrul de căutare variabil id.

În Angular, în serviciul „search-flight” am adăugat o metodă care va avea ca scop apelarea endpoitului din backend. Această metodă fiind ulterior folosită în fisierul TypeScript al dashboard-ului într-o altă metodă deleteFlight() la finalul căreia vom returna noua listă dupa ștergerea zborului.

Această ultimă metodă o vom asocia butonului de Delete din dashboard.

A screenshot of a computer

Description automatically generated

Figură 25. Ștergerea zborului cu id 13. Sursă proprie.

4.9 Rezervarea unui zbor

Pentru rezervarea unui zbor am început prin adăugarea în backend a clasei Booking, care va conține variabile precum „id” care a fost și adnotata cu @Id pentru a marca faptul că acest camp este cheia primară în baza de date. Câmpurile userid și flightsid reprezintă relația many to one între rezervare și utilizator, respectiv zbor. Pe acestea le-am adnotat cu „@ManyToOne” și „@JoinColumn” pentru a specifica coloanele de legătura în tabelul booking, iar restul campurilor: leavinfrom, arrivingat și leavingdate reprezintă informații suplimentare pe care le vom folosii pe partea de frontend pentru a prelua datele zborului și pe partea de rezervare.

Mergând mai departe la serviciul dedicat rezervarilor, am realizat o metodă bookFlight() care primeste doi parametrii „userId” și „flightId” reprezentând identificatorii userului și a zborului pentru care se va face rezervarea. Cei doi identificatori sunt gasiți cu ajutorul repositoriului „userRepo” respectiv „flightRepo” în cazul în care acestia nu sunt gasiți se va arunca cate o excepție „RuntimeException” pentru fiecare. Tot în cadrul acestei metode vom obtine și numărul curent de locuri disponibile din obiectul „flight” pentru a anticipa dezvoltarea privind updata-rea locurilor unui zbor în funcție de numărul de boarding passuri pe care le vom înregistra pe o rezervare. Însă vom trata mai pe larg această dezvoltare în următorul capitol. La finalul acestei metode vom crea o nouă instană a clasei „Booking” și vom seta proprietațile menționate mai sus, urmând să salvăm rezervarea folosind metoda save din cadrul „bookingRepo” și returnând rezervarea salvată.

Pentru controler, am adaugăt în controllerul dedicat rezervărilor, o metodă bookFlight() pe care am marcat-o cu @PostMapping. Această metodă primeste ca parametrii „userId” și „flightId”, parametrii de cale, urmând să apelam metoda bookFlight() din cadrul serviciului pasând parametrii și returnând rezultatul.

În angular am inceput prin generarea serviciului pentru rezervare în cadrul căruia am construit metoda bookFlight() tot cu cei doi parametrii ca și mai sus. Această metodă construieste URL-ul pentru a efectua cererea POST catre serviciul de rezervări folosind sablonul bookingURL adaugat ca și variabilă separată și parametrii „userId” și „flightId”. Rezultatul întors trebuie să fie sub formă de obiect Observable de tipul Booking, insemnând ca asteaptă un obiect de tip Booking ca răspuns de la server.

În fișierul TypeScript al componentei booking am adăugat proprietățile:

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

pe care le vom folosii în template-ul HTML și în logica componentei. Constructorul primeste instante ale serviciilor necesare prin injecția de dependetă.

Prima metodă descirsă este ngOnInit() care este apelata automat dupa ce componenta este inițalizată și preia flightId-ul din ruta curentă și userId-ul din serviciul de stocare. Apelul „searchflightservice.getFlightId” va genera o cerere către server pentru a obține detaliile zborului specificat de flightId și actualizează proprietatea flight cu datele obținute.

A doua metodă onSubmit() este apelată când utilizatorul trimite formularul de rezervare. Aceasta foloseste serviciul bookginServie pentru a efectua cererea de rezervare a zborului.

4.10 Adăugarea boarding pass-ului în rezervare.

Deoarece o rezervare se poate face pentru mai multe persoane, am adăugat și conceptul de boarding pass. Astfel pentru fiecare rezervare putem avea până la maxim 3 boarding passuri adăugate.

Am generat prima data în pachetul domain din backend, entitatea BoardingPass care va conține atributele: id, name, identifycard și booking. Cea din urmă reprezintă rezervarea ascitată boarding pass-ului și a fost mapata printr-o relatie @ManyToOne cu clasa Booking. Relația este definită prin intermediul coloanei „booking\_id” în baza de date. Astfel în cadrul clasei Booking, am adăugat și o metodă addBoardingPass() care perminte adăugarea unui boarding pass la rezervare. Metoda verifică dacă numarul de boarding pass-uri existente depaseste 3 și aruncă o excepție în caz afirmativ. Apoi va adăuga boarding pass-ul la lista boardingPasses și setează rezervarea curentă ca referintă.

Am definit și repositoriul „BoardingPassRepo” , iar în cadrul repositoriului Booking am adăugat o metoda countBoardingPassesByFlightsId(), care va utiliza o interogare personalizată pentru a număra boarding pass-urile asociate unui anumit zbor. Metoda va returna un intreg care va reprezenta numărul de boarding pass-uri asociate zborului specificat.

Următorul pas a fost ca în cadrul serviciului rezervărilor să adaug și metoda addBoardingPassToBooking(), care va cauta ințial în baza de date rezervarea pe baza id-ului furnizat „bookingId” utilizând metoda findById din cadrul repositoriului de rezervari. Dacă rezervare nu este gasită se va arunca o excepție și vom face ulterior și verificarea dacă numele și cardul de itentitate sunt nenule, aruncând o excepție și în acest caz dacă unul dintre ele este null. Vom salva boarding pass-ul în baza de date utilizând metoda save(), adăugând-ul ulterior și în lista de boarding pass-uri a rezervării prin metoda addBoardingPass definită în entitatea Booking. De asemenea vom obține zborul asociat rezervării și vom actualiza numărul de locuri disponibile în funcție de numărul total de boarding pass-uri adăugate rezervării. În final vom salva rezervarea în baza de date utilizând metoda save().

Revenind la controllerul pentru rezervari a fost creat un endpoint de tip POST care permite adăugarea unui boarding pass la o rezervare existentă în sistem. Endpoint-ul primește id-ul rezervării și obiectul „boardingPass” care conține infromații despre pasagerul pentru care se adaugă boarding passul. Se va apela metoda addBoardingPassToBooking() din serviciul „bookingService” pentru a adăuga boarding passul la rezervare.

Pentru partea de frontend am adăugat în cadrul serviciului de rezervări metoda addBoardingPassToBooking() care este responsabilă cu adăugarea unui boarding pass la o rezervare existentă. Această metodă primește 2 parametrii: id-ul rezervării și un obiect boardinPass. Cu această ocazie vom genera și clasa BoardingPass cu cele 2 variabile name și identifycard. Revenind la metoda, aceasta va construii URL-ul corespunzător cererii către server.

Ultimul pas a fost acela de a adauga în componența TypeScript a rezervării metodele:

* addBoardingPass() – care este apelată atunci când utilizatorul dorește să adauge un nou boarding pass. Iar în cadrul ei se va verifica dacă numărul de boarding pass-uri adăugate nu depasește limita de 3. În caz contrar vom afișa un mesaj de avertizare.
* removeBoardingPass() – care este responsabilă cu eliminarea unui boarding pass din lista. Eliminarea se va face pe baza index-ului acestuia.
* addBoardingPassToBooking() – este utilizată pentru adăugarea boarding pass-ului la rezervare, parcurgând lista și verificând că sunt completate numele și cardul de identitate. Dacă aceste informații sunt disponibile, se construiește un obiect, „Bordingpass” și se utilizează serviciul de rezervare pentru a adaugăt un boarding pass la rezervare prin intermediul metodei addBoardingPassToBooking().

A screenshot of a computer

Description automatically generated

Figură 26. Rezervarea unui zbor cu adăugarea unui bording pass. Sursă proprie.

5. Concluzii și implementări viitoare

Ca și implementari viitoare aș dori sa aduc in cadrul aplicatiei urmatoarele functionalitati:

* Check-in, astfel persoana care a facut rezervarea să realizeze și checkin-ul.
* Resetarea parolei unui utilizator prin apăsarea butonului Forgot Password și primirea parolei noi pe email.
* Posibilitatea user-ului sa își planifice o vacanță oferită de către compania noastră, prin adăugarea în secțiunea PLAN a ofertelor pe care compania le oferă și corelarea acestora cu zborurile.
* Posibilitatea de alegere a locurilor în avion, unde îmi doresc să afișez o hartă a avionului cu locurile disponibile și cele ocupate.
* Funcționalitatea adminului de a modifica locurile rezervate și anumite rezervari.

# 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/. |