Proiect
Baze de Date II

Companie Aeriană

Catrangiu Radu-Ovidiu
343C1

Cuprins

Descrierea Temei

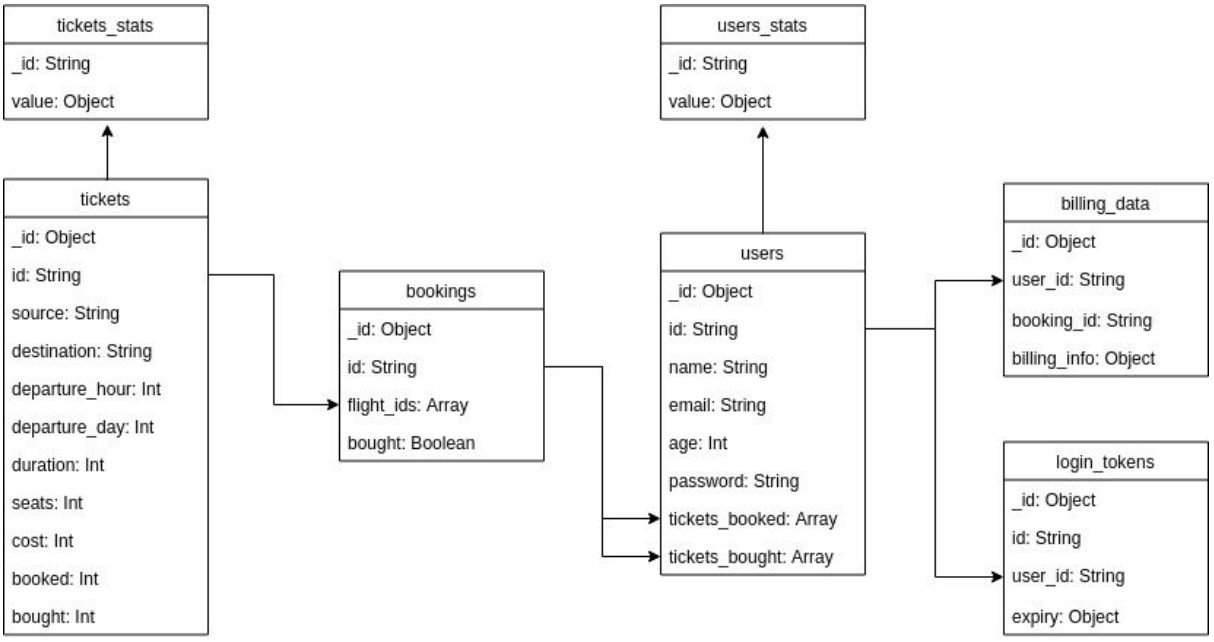
Tema reprezintă o companie de transport aerian. Este implementată în JavaScript folosind NodeJS şi MongoDB pentru backend, respectiv Vue.JS şi Bootstrap pentru frontend. Serverul de NodeJS expune un API pentru accesarea bazei de date şi răspunde la cereri HTTP de tip GET pentru afişarea interfeţei grafice.

Pagina de întâmpinare este o pagină de login sau de creare de cont. După autentificare utilizatorul este redirecţionat către */client*, unde poate să introducă anumite criterii de căutare a unui zbor, să rezerve zboruri şi să vizualizeze toate zborurile rezervate şi cumpărate. După rezervarea unui zbor utilizatorul primeşte un link către o pagină unde poate să vizualizeze detaliile rezervării şi să cumpere ulterior biletul sau biletele de avion.

Pagina de administrare afişează toate zborurile, oferă posibiltatea de ştergere sau de adăugare a unor noi zboruri şi oferă date statistice legate de Utilizatori şi Zboruri.

Descrierea Bazei de Date

Diagrama Bazei de Date



Structura Tabelelor

tickets:

- _id: Index generat de MongoDB
- id: String unic generat programatic (uuidv4)
- source: Sursa zborului.
- destination: Destinaţia zborului.
- departure_hour: Ora din zi în care are loc zborul.
- departure_day: Ziua din an în care are loc zborul.
- duration: Durata în ore a zborului
- seats: Numărul de locuri total disponibil în cadrul zborului.
- booked: Numărul de bilete rezervate pentru zborul curent.
- bought: Numărul total de bilete cumpărate pentru zborul curent.
- cost: Preţul unui bilet.

bookings:

- _id: Index generat de MongoDB
- id: String unic generat programatic (uuidv4)
- flight_ids: Array de id-uri ale biletelor ce trebuiesc cumpărate pentru a zbura de la sursă la destinaţie. Zboruri cu escală.
- bought: Boolean necesar pentru a genera corect pagina de booking astfel încât să nu poată fi cumpărat de două ori.

users:

- _id: Index generat de MongoDB
- id: String unic generat programatic (uuidv4)
- name: Numele utilizatorului
- email: Email-ul utilizatorului
- age: Vârsta utilizatorului
- password: Parola utilizatorului. Stocată în clar în această implementare rudimentară. Altfel ar trebui adăugat `salt` şi criptată, apoi stocat hash-ul.
- tickets_booked: Array de obiecte unde un obiect conţine date despre un zbor rezervat.
- tickets_bought: Array de obiecte unde un obiect conţine date despre un zbor cumpărat.

billing_data:

- _id: Index generat de MongoDB
- user_id: ID-ul userului ale cărui date de cumpărare sunt stocate în billing_info.
- booking_id: ID-ul rezervării pentru care sunt folosite datele din billing_info pentru a efectua o plată.
- billing_info: Datele cardului utilizat pentru a efectua plata de către utilizator.

login_tokens:

- _id: Index generat de MongoDB
- id: String unic generat programatic (uuidv4). Reprezintă cookie-ul ce facilitează autentificarea unui utilizator.
- user_id: ID-ul userului ce este autentificat.
- expiry: Obiect de tip Date utilizat de MongoDB pentru a şterge documentul din colecţie după 3600 de secunde de la creare.

Descrierea Constrângerilor de Integritate

Pentru implementarea constrângerilor de integritate am folosit un validator în instanţa de MongoDB în NodeJS. Acest validator primeşte o schemă JSON pe care am configurat-o folosind documentaţia oficială. Această configuraţie poate fi găsită în fişierul *config.js*.

tickets:

- id: String unic
- source: String de lungime minimă 2
- destination: String de lungime minimă 2
- departure_hour: Integer în intervalul [0, 23]
- departure_day: Integer în intervalul [1, 365]
- duration: Integer mai mare sau egal cu 1
- seats: Integer
- booked: Integer
- bought: Integer
- cost: Integer mai mare sau egal cu 1

bookings:

- id: String unic
- flight_ids: Array ce conţine numai obiecte de tip String
- bought: Boolean

users:

- id: String unic
- name: String de lungime minimă 3
- email: String unic de lungime minimă 3
- age: Integer
- password: String de lungime minimă 3
- tickets_booked: Array ce conţine numai obiecte. Un obiect conţine următoarele proprietăţi: ‘ticket’ : String, ‘flights’: Integer, ‘cost’: Integer
- tickets_bought: Array ce conţine numai obiecte. Un obiect conţine următoarele proprietăţi: ‘ticket’ : String, ‘flights’: Integer, ‘cost’: Integer

billing_data:

- user_id: String
- booking_id: String
- billing_info: Obiect de forma: ‘card_number’: String de lungime minima 2, ‘name’: String de lungime minima 2, ‘expiry_month’: Integer in intervalul [1, 12], ‘expiry_year’: Integer in intervalul [18, 40], ‘card_cvv’: String de fix 3 caractere.

login_tokens:

- id: String unic
- user_id: String
- expiry: Obiect

Descrierea procedurilor și funcţiilor

/admin - insert():

Inserează un nou zbor în colecţia *tickets*.

/admin - delete():

Șterge un zbor din colecţia *tickets*.

/admin - list():

Întoarce toate zborurile din colecţia *tickets*.

/stats - update():

Apelează două funcţii **mapReduce** în funcţie de parametrul *collection*. Dacă are valoarea *tickets* funcţia rulează mapReduce pe colecţia *tickets*, altfel rulează mapReduce pe colecţia *users*. Prima variantă generează date în colecţia *tickets_stats*, iar a doua în colecţia *users_stats*.

/stats - list():

În funcţie de parametrul *collection* întoarce datele din colecţiile *tickets_stats* sau *users_stats*.

/account - login():

Caută userul în colecţia *users* după email, verifică corectitudinea parolei, generează un token şi îl adaugă în colecţia *login_tokens*.

/account - create():

Inserează un user nou în colecţia *users* apoi apelează funcţia login().

/account - check_token():

Caută un token primit ca parametru în colecţia *login_tokens*. Dacă îl găseşte întoarce răspuns “Valid token”, altfel eroare “Invalid token”.

/client - get_user_info():

Caută şi întoarce datele stocate în colecţia *users* despre utilizatorul care face apelul. Sunt omise câmpurile “password”, “_id” şi “id”.

/client - get_optimal_route():

Caută în colecţia *tickets* şi întoarce zborurile care satisfac condiţiile utilizatorului.

/client - book_ticket():

Caută zborurile din lista primită ca parametru în colecţia *tickets*, le incrementează proprietatea *booked*, apoi inserează în colecţia *bookings* noua rezervare şi inserează în proprietatea de tip array *tickets_booked* a utilizatorului datele despre rezervare (id, număr de zboruri, cost total).

/client - buy_ticket():

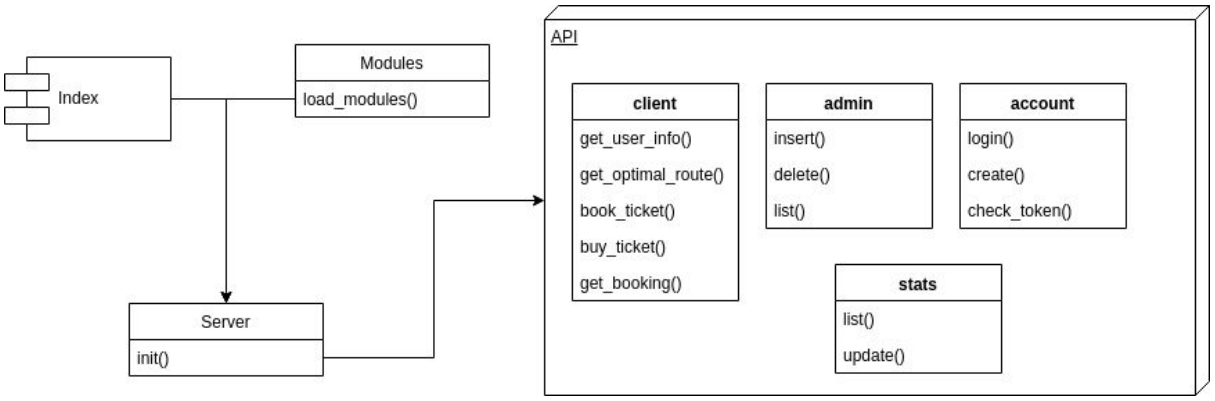
Inserează în colecţia *billing_data* datele cardului utilizatorului. Caută în colecţia *bookings* rezervarea. Pentru toate zborurile din rezervare actualizează proprietăţile *booked* şi *bought*, decrementând-o pe prima şi incrementând-o pe a doua. Actualizează rezervarea din colecţia *bookings*, setând proprietatea *bought* ca *true.* Caută userul cumpărător în colecţia *users*. Caută datele rezervării în datele utilizatorului. Se actualizează documentul utilizatorului din colecţia *users* eliminând datele rezervării din proprietatea array *tickets_booked* şi adăugându-le în proprietatea array *tickets_bought*.

/client - get_booking():

Folosind codul rezervării primit ca parametru caută în colecţia *bookings* rezervarea, apoi caută în colecţia *tickets* datele despre zborul/zborurile din rezervare şi le întoarce.

Descrierea Aplicaţiei

Diagrama de Clase



Structura Claselor

În index se apelează clasa Modules prin metoda load_modules() pentru a iniţializa modulele de mediu (environment), mai precis colecţiile specificate în config şi funcţia care identifică utilizatorul după token la fiecare apel de API.

Modules

Citeşte fişierul de configurare şi iniţializează conexiunea la baza de date. Creează colecţiile adăugând opţiuni pentru constrângeri şi pentru fiecare colecţie apelează createIndex() dacă este specificat în fişierul de configurare. În cazul colecţiei *login_tokens* este setat indexul *expiry* astfel încât fiecare document să fie şters automat în 3600 de secunde.

De asemenea, clasa Modules iniţializează şi funcţia de autentificare care primeşte un token, mai precis cookie-ul cu care este apelat API-ul, şi verifică în colecţia *login_tokens* dacă există acest token şi cui îi aparţine, setând în environment parametrul *user_id*, astfel fiecare funcţie din API să aibă acces la user_id-ul apelantului.

Server

Serverul expune o funcţie de iniţializare care primeşte următorii parametri: port, rpc_config, modules şi callback. *Port*-ul este cel din configuraţie, *port*-ul pe care va porni serverul, *rpc_config* este un obiect în care sunt definite fişierele în care se află funcţiile (handler), dacă serviciile sunt accesate prin autentificare (useAuth) şi calea fiecărui serviciu din API. *Modules* este obiectul cu modulele iniţializate. *Callback* este funcţia apelată la finalul iniţializării. De asemenea, în server sunt iniţializate şi toate handler-urile pentru cererile de tip GET.

Client

Clasa *client* expune 5 funcţii publice şi 2 funcţii private:

- ***get_user_info*** : nu primeşte niciun parametru şi caută în baza de date informaţii despre userul apelant şi le întoarce.
- ***get_optimal_route*** : primeşte ca parametri datele de căutare introduse de user, caută în baza de date toate zborurile şi calculează zborul optim folosind funcţia privată *find_optimal_route*.
- ***book_ticket*** : primeşte ca parametru lista de zboruri ce trebuiesc rezervate. Actualizează toate zborurile din listă incrementându-le proprietatea *booked*. Inserează în colecţia *bookings* noua rezervare şi adaugă în lista de rezervări ale userului noua rezervare.
- ***buy_ticket***: primeşte ca parametru id-ul rezervării şi informaţiile necesare pentru efectuarea plăţii. Actualizează toate zborurile din listă incrementându-le proprietatea *bought* şi decrementându-le proprietatea *booked*. Marchează rezervarea în baza de date ca fiind cumpărată şi o elimină din lista de rezervări ale userului, adăugând-o în lista de bilete cumpărate de către user.
- ***get_booking***: primeşte ca parametru id-ul rezervării. Caută în baza de date rezervarea, apoi toate zborurile menţionate în rezervare şi întoarce datele despre aceste zboruri.
- *find_optimal_route*: este funcţia care calculează zborurile necesare pentru a ajunge de la sursă la destinaţie.
- *compute_distance*: este o funcţie ce calculează metrica pentru sortarea zborurilor în *find_optimal_route*.

Admin

Clasa *admin* expune 3 funcţii publice:

- ***insert***: funcţie care primeşte ca parametru datele unui nou zbor ce este introdus în baza de date.
- ***delete***: funcţie care pimeşte ca parametru id-ul unui zbor ce este şters din baza de date.
- ***list***: functie care nu primeşte niciun parametru şi care întoarce toate zborurile.

Account

Clasa account expune 3 funcţii publice:

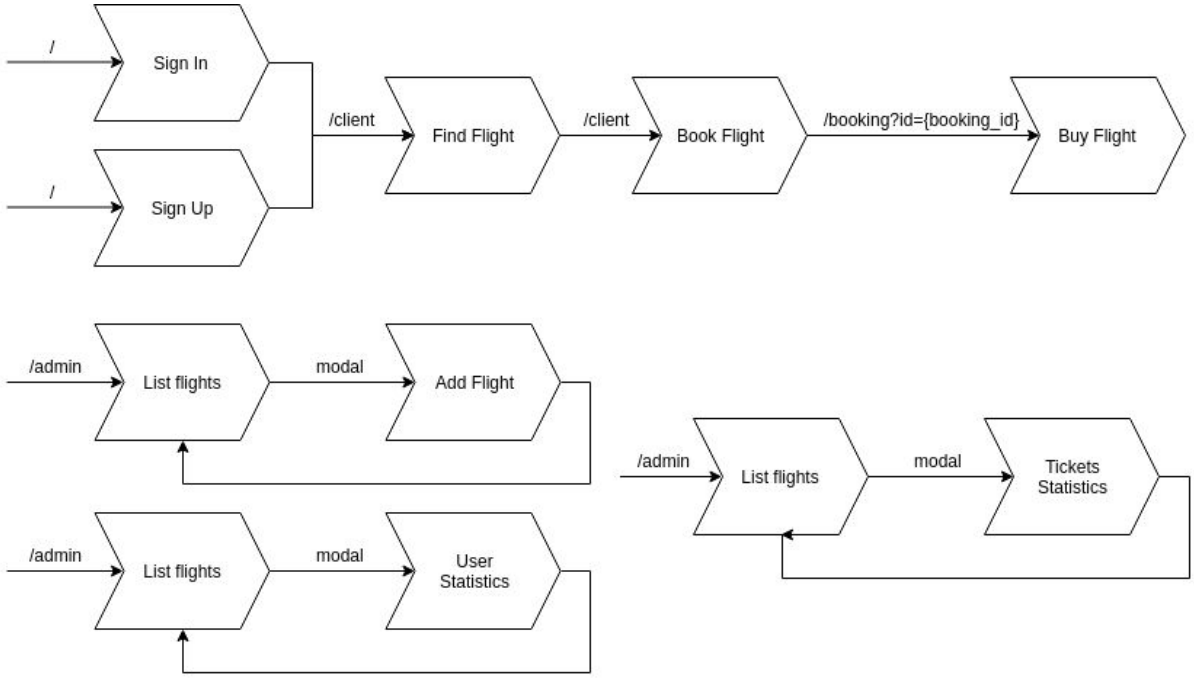
- ***login***: funcţie care primeşte ca parametru un email şi o parolă, caută în baza de date după email, dacă găseşte user-ul verifică să coincidă parola, apoi creează un token de login pe care îl introduce în colecţia *login_tokens*.
- ***create***: funcţie care primeşte ca parametru numele, email-ul, vârsta şi parola cu care creează un utilizator nou în colecţia *users*. După introducerea noului utilizator în baza de date este apelată funcţia *login*.
- ***check_token***: funcţie care primeşte ca parametru un token pe care îl caută în baza de date pentru a-i determina validitatea.

Stats

Clasa *stats* expune 2 funcţii publice şi 2 funcţii private:

- ***update***: funcţie care primeşte un parametru în funcţie de care apelează una dintre funcţiile private *update_users_stats* sau *update_tickets_stats*.
- ***list***: funcţie care primeşte un parametru în funcţie de care listează conţinutul uneia dintre colecţiile *tickets_stats* sau *users_stats*.
- *update_users_stats*: funcţie care apelează **mapReduce** pentru a genera un raport în colecţia *users_stats*.
- *update_tickets_stats*: funcţie care apelează **mapReduce** pentru a genera un raport în colecţia *tickets_stats*.

Workflow

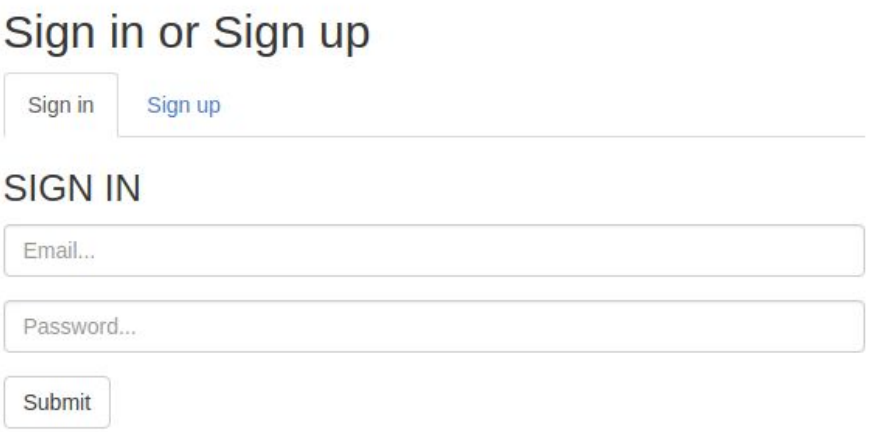


Modul de Conectare la Baza de Date

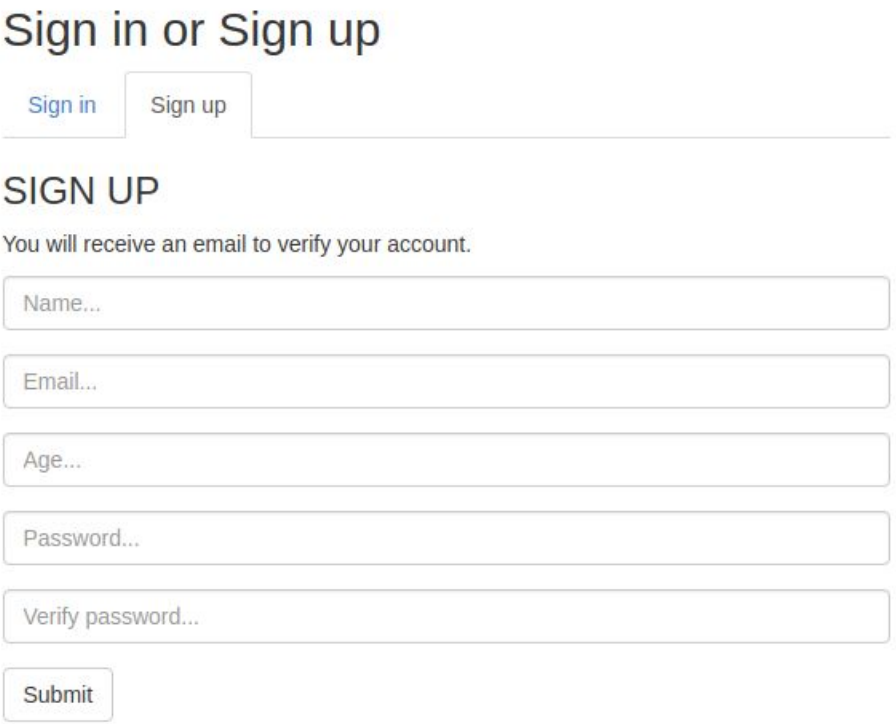
La iniţializare serverul se conectează la baza de date folosind un URL de forma `mongodb://${user}:${password}@${login_url}`. De asemenea se specifică baza de date la care se doreşte conectarea şi colecţiile din baza de date respectivă, pentru a fi iniţializate, cu constrângerile specificate în fişierul de configurare, în cazul în care nu există.

Capturi de ecran

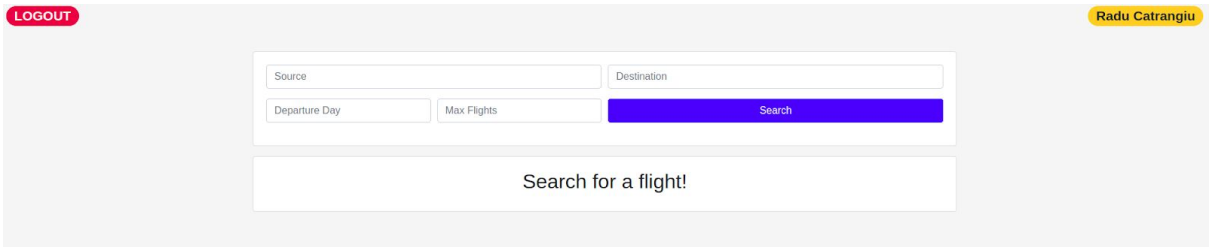
Pagina de **Sign in**

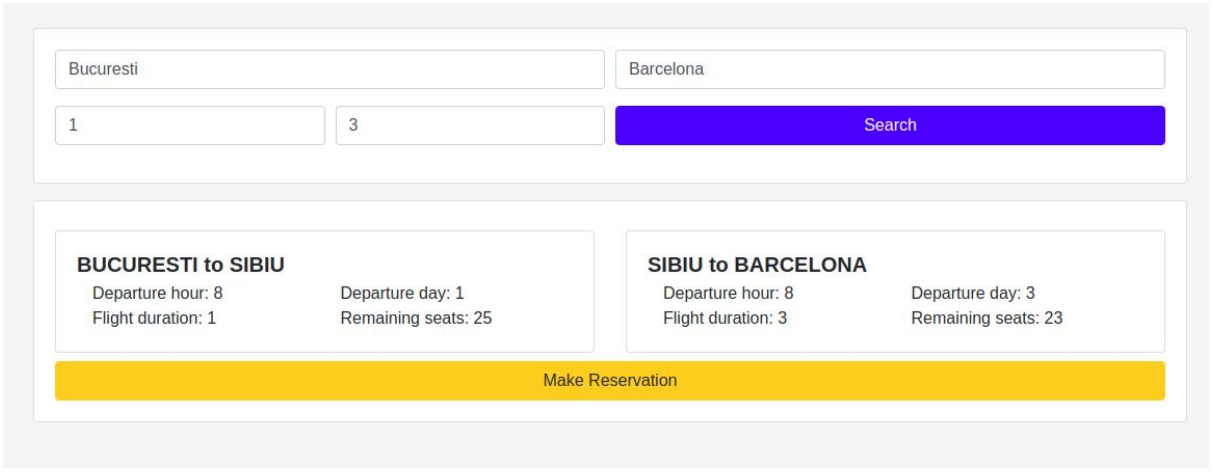


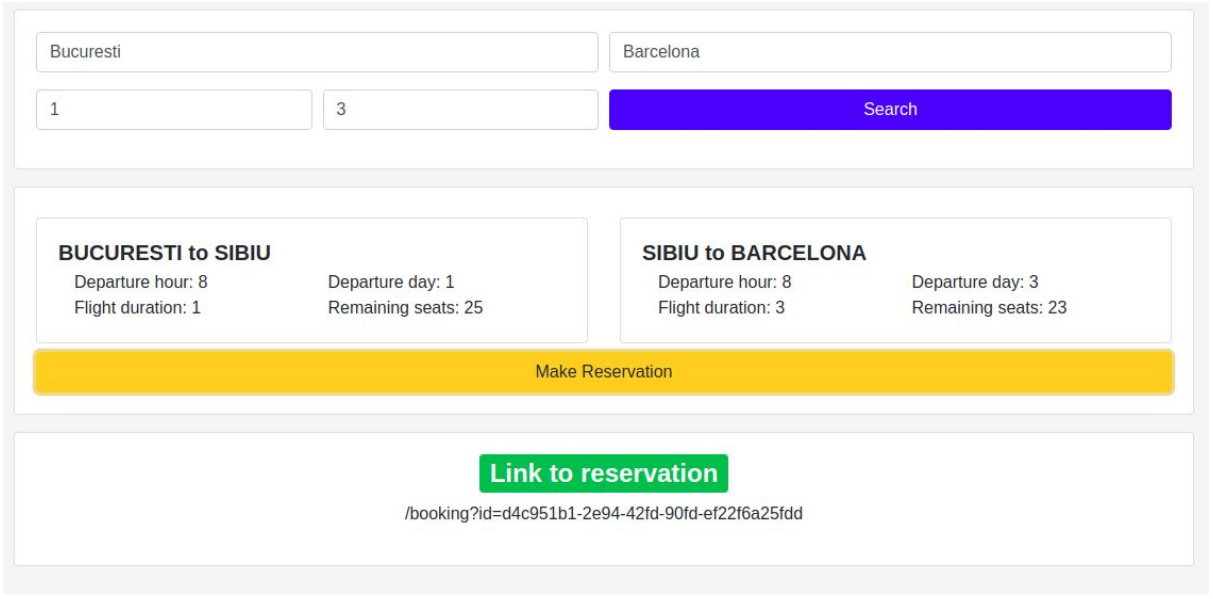
Pagina de Sign up

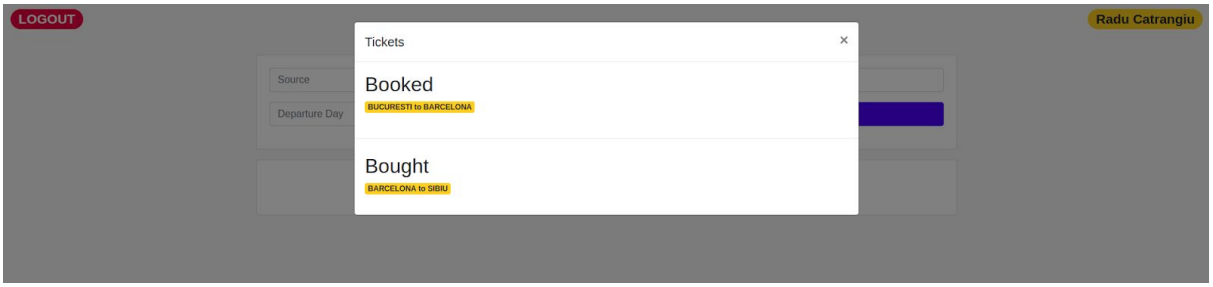


Pagina de căutare a zborului

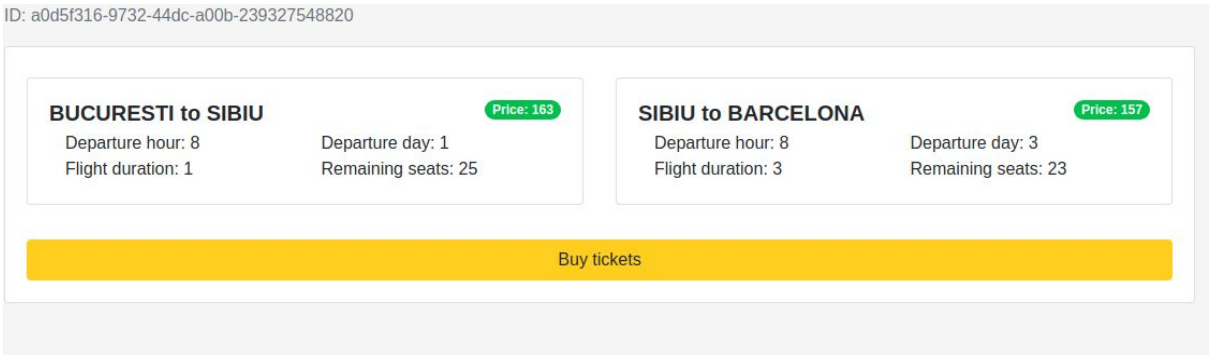


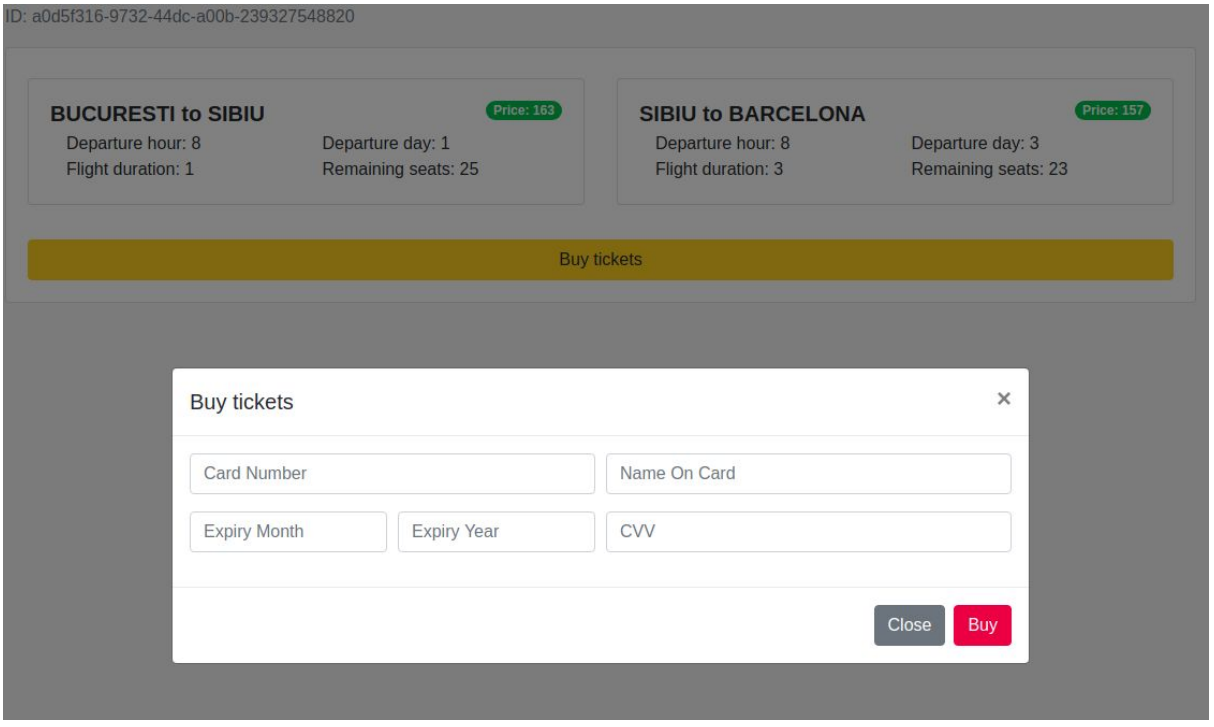


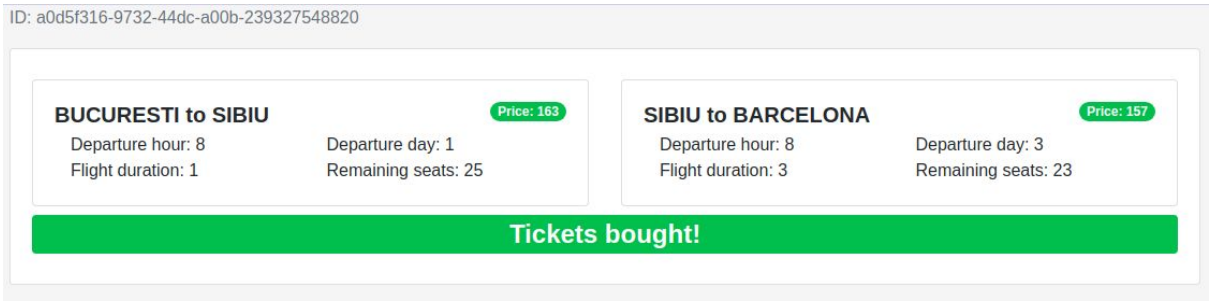




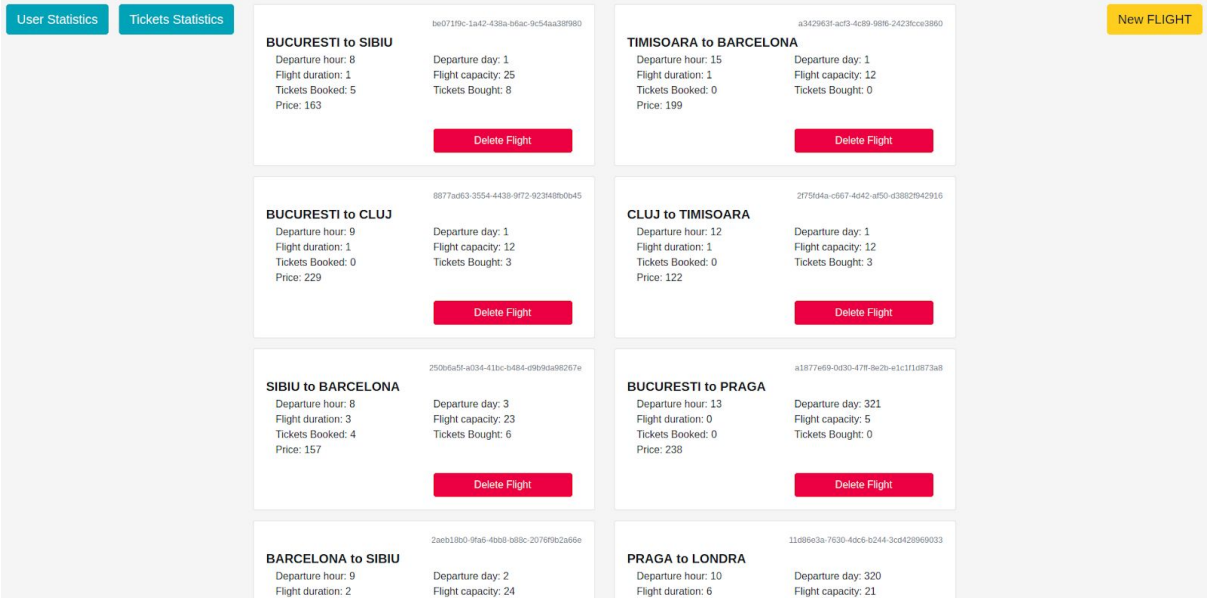
Pagina rezervării

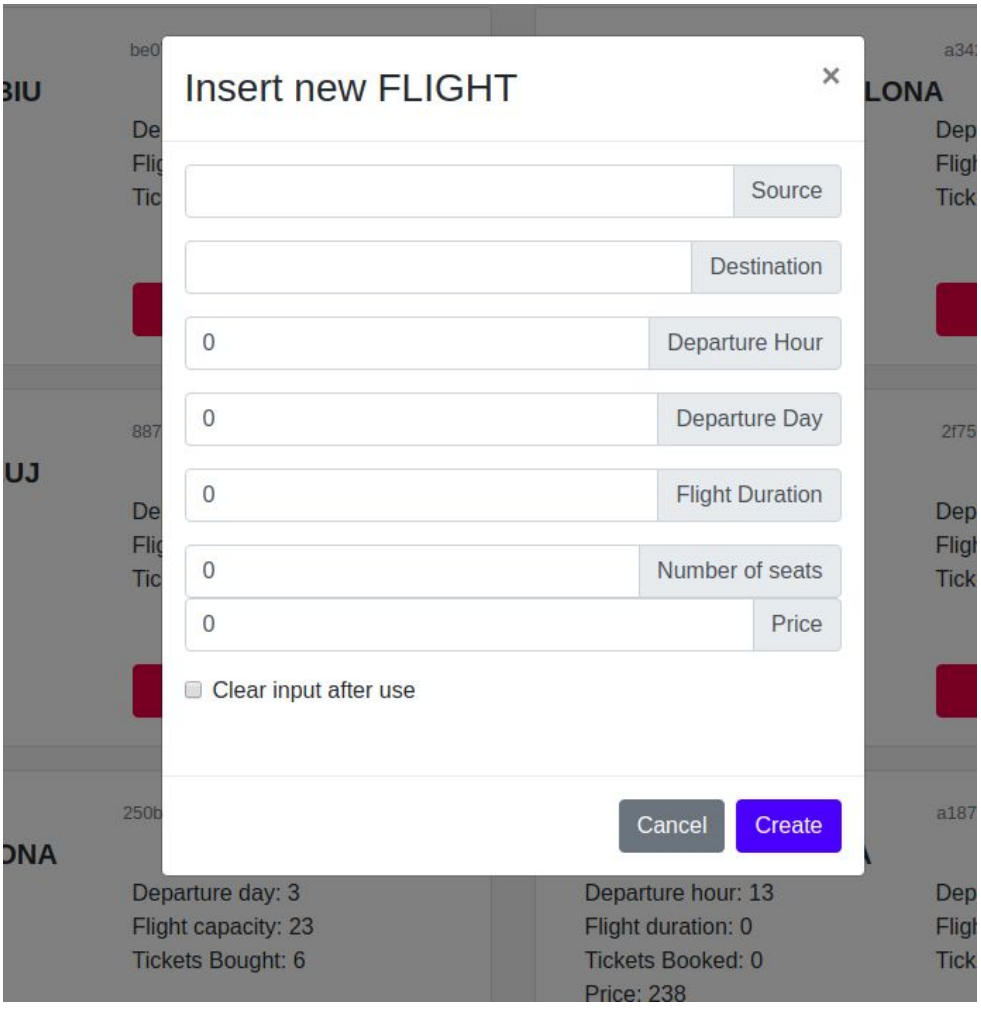




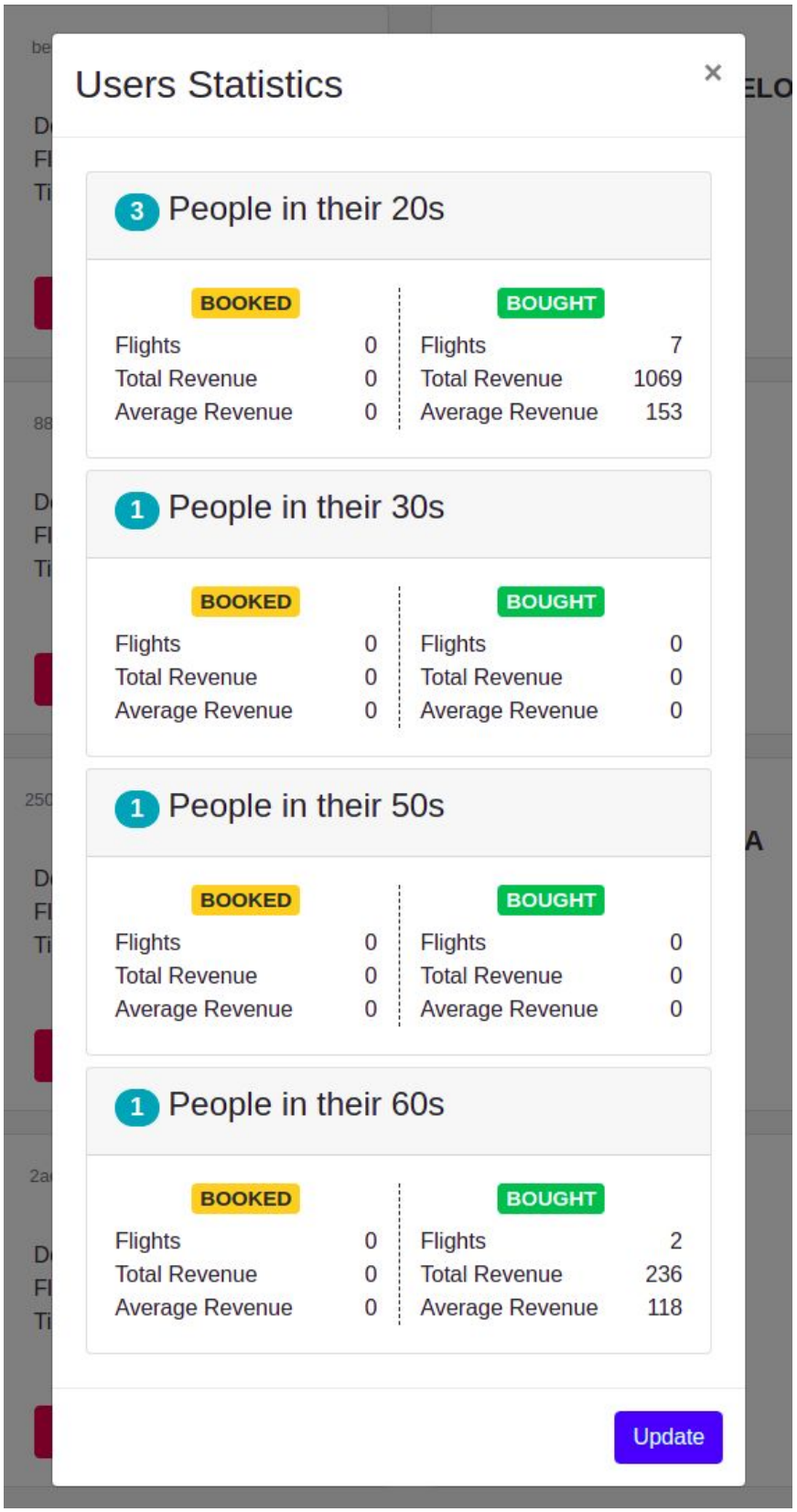


Pagina de administrare şi statistici

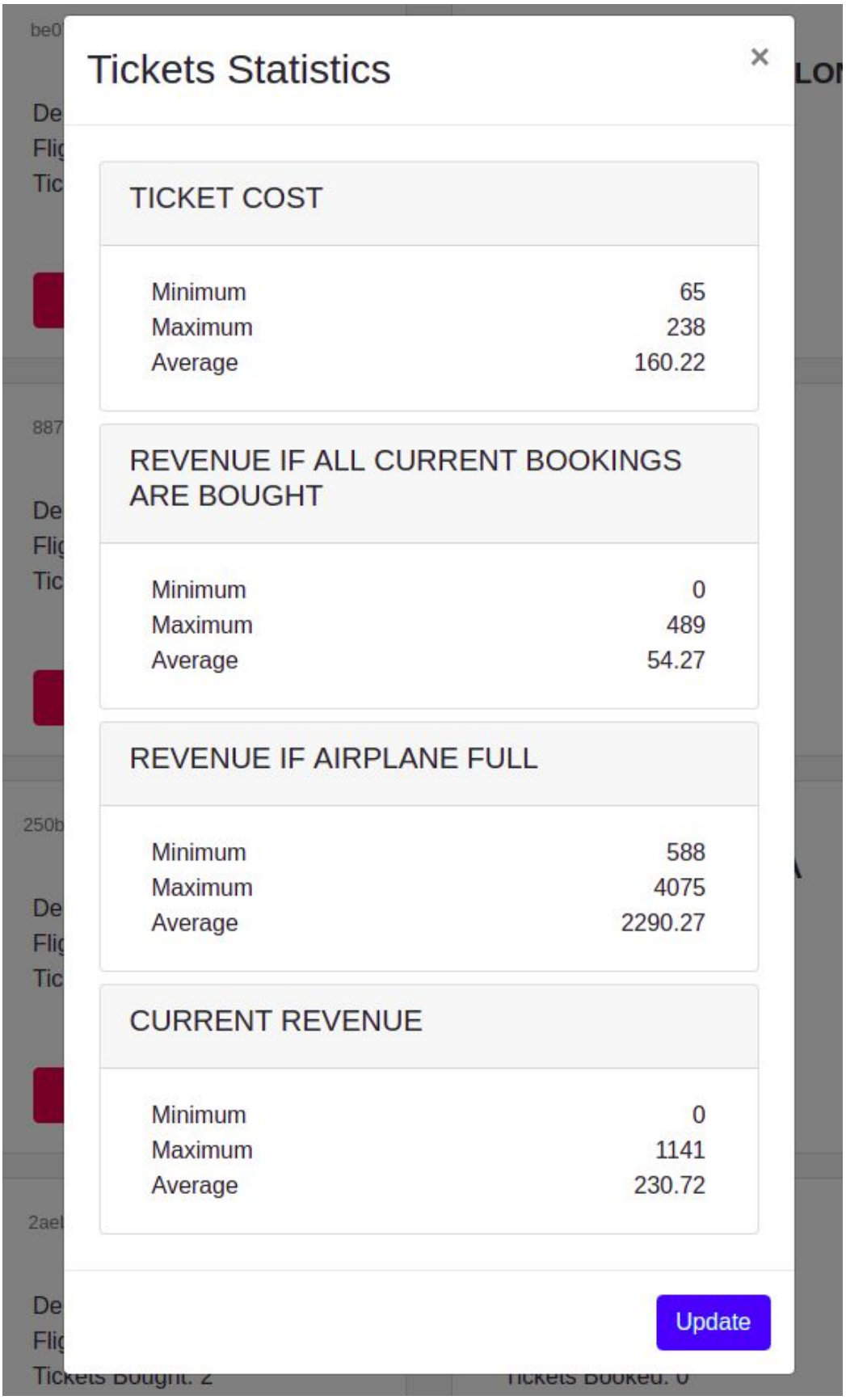




Raport Useri



Raport Bilete



Concluzii

Proiectul a fost unul util deoarece nu există aplicaţie din domeniu care să nu folosească cel puţin o bază de date, iar ca programator trebuie să interacţionezi cu acestea măcar o dată pentru a te familiariza cu modul de lucru.

Bibliografie

https://docs.mongodb.com/
https://getbootstrap.com/docs/4.2/
https://developer.mozilla.org/ro/docs/Web/JavaScript
https://vuejs.org/v2/guide/