

APLICATIE ONLINE DE CUMPARAT BILETE DE AVION

BAZE DE DATE 2

TAZLAUANU BIANCA
343C3



CUPRINS

I. DESCRIEREA TEMEI

II. DESCRIEREA BAZEI DE DATE:

A. DIAGRAMA BAZEI DE DATE

B. STRUCTURA TABELELOR

C. DESCRIEREA CONSTRÂNGERILOR DE INTEGRITATE

D. DESCRIEREA PROCEDURILOR ȘI FUNCȚIILOR

III. DESCRIEREA APLICAȚIEI:

A. DIAGRAMA DE CLASE

B. STRUCTURA CLASELOR SI MODUL DE FUNCTIONARE

C. DIAGRAMA DE STĂRI ȘI FLUXUL DE LUCRU (WORKFLOW) PENTRU APLICAȚIE

D. PREZENTAREA MODULUI ÎN CARE SE FACE CONEXIUNEA CU BAZA DE DATE

IV. CONCLUZII

V. BIBLIOGRAFIE

DESCRIEREA TEMEI

Proiectul are ca tema implementarea unei aplicatii web pentru cumpararea biletelor de avion. Aplicatia este intuitiva si usor de folosit si permite utilizatorilor sa isi selecteze zborul dorit si sa isi cumpere bilet. De asemea, utilizatorii beneficiaza si de optiunea de a isi vizualiza zborurile achizitionate. O data ce un bilet este achizitionat, pe ecran va fi afisat si biletul de imbarcare impreuna cu toate detaliile zborului. Clientii vor putea sa isi aleaga zborul in functie de punctul de plecare, destinatie, precum si ziua zborului.

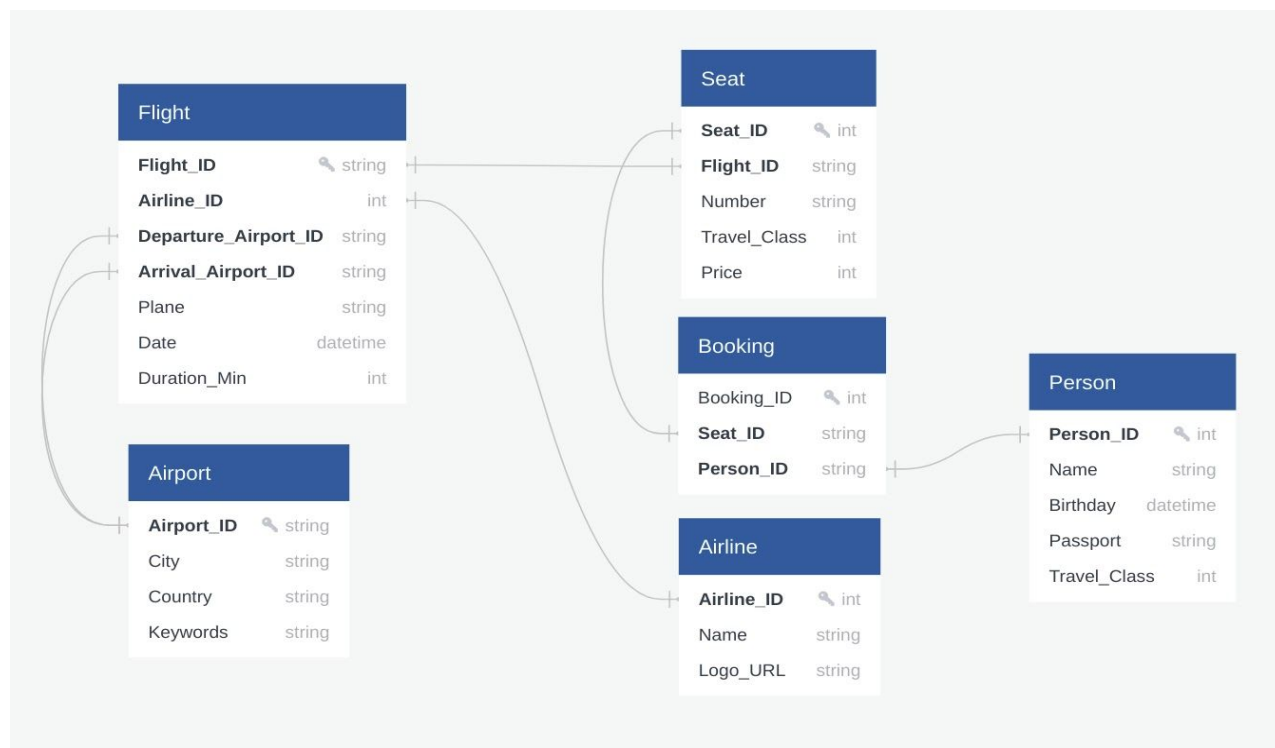
Aplicatia este realizata folosind tehnologii multiple incluzand python, flask, MongoDB pentru partea de backend si javascript, HTML, Bootstrap si Jinja2 pentru realizarea frontend-ului.

DESCRIEREA BAZEI DE DATE

Baza de date a fost implementata folosind MongoDB si dispune de 6 colectii: Person, Airport, Flight, Seat, Booking, Airline.

DIAGRAMA BAZEI DE DATE

Aceasta este diagrama reprezentativa pentru tabelele folosite in cadrul proiectului.



Aplicatia se foloseste sase colectii care contin informatii despre zboruri, aeroporturile de pe care decoleaza si de pe care aterizeaza avioanele, detalii despre rezervarile facute si persoanele care au facut rezervarile. Deoarece toate informatii prezente in tabele sunt necesare si esentiale pentru buna functionare a proiectului, toate campurile sunt obligatorii. De asemenea, sunt prezente elemente unice pentru fiecare colectie pentru a fi identificate datele individual.

STRUCTURA TABELELOR

Descrierea colectiilor si a campurilor din cadrul lor:

Person:

Descriere: Contine informatii in legatura cu persoanele care au achizitionat un bilet de avion

Campuri:

- Person_ID int: ID-ul unic al fiecarei persoane
- Name string: Numele persoanei
- Birthday datetime: Data nasterii
- Passport string: Numarul pasaportului
- Travel_Class int: Simbolizeaza clasa zborului (Econim/Business/First) sub forma unui numar

Flight

Descriere: Contine informatii in legatura cu un zbor al unui avion

Campuri:

- Flight_ID string: ID-ul unic al fiecarui zbor
- Airline_ID int: ID-ul companiei aeriene care are zborul respectiv
- Departure_Airport_ID string: ID-ul aeroportului de pe care decoleaza avionul
- Arrival_Airport_ID string: ID-ul aeroportului pe care aterizeaza avionul
- Plane string: Numele avionului reprezentand tipul avionului
- Date datetime: Reprezinta data la care incepe zborul
- Duration_Min int: Durata zborului in minute de la timpul de plecare

Airline

Descriere: Contine informatii despre companiile aeriene (Ex: Swiss, KLM)

Campuri:

- Airline_ID int: ID-ul companiei aeriene prin care aceasta poate fi identificata
- Name string: Numele companiei aeriene
- Logo_URL string: Un url catre logo-ul companiei care va fi folosit pentru partea de frontend a aplicatiei

Airport

Descriere: Contine informatii despre aeroporturile de pe care pleaca si aterizeaza zborurile

Campuri:

- Airport_ID string: ID-ul aeroportului, de exemplu OTP - Aeroportul Bucuresti
- City string: Numele orasului in care se afla aeroportul
- Country string: Numele tarii in care se afla aeroportul
- Keywords string: Cuvinte cheie prin care se poate identifica aeroportul

Seat

Descriere: Contine informatii despre un loc dintr-un avion, pentru un zbor anume

Campuri:

- Seat_ID int: ID-ul de identificare unic
- Flight_ID string: ID-ul zborului din care face parte
- Number int: Numarul locului, folosit pentru a instiinta un calator de locul pe care il va ocupa in timpul zborului in avion
- Travel_Class int: Acesta reprezinta clasa de comfort din care face parte: Economic, Business sau First, fiecare codificat cu o anumita valoare numerica
- Price int: Reprezinta pretul pe care trebuie un calator sa plateasca pentru a sta pe locul respectiv in cadrul unui zbor, este pretul zborului

Booking

Descriere: Reprezinta toate rezervarile facute, face maparea intre biletele cumparate si calatori

Campuri:

- Booking_ID int: ID-ul de identificare unic
- Person_ID int: ID-ul persoanei care a cumparat biletul respectiv
- Seat_ID int: ID-ul biletului cumparat, adica a locului pentru care persoana a platit

DESCRIEREA CONSTRÂNGERILOR DE INTEGRITATE

Pentru identificatori, a fost aplicata constrangerea de unicitate, astfel incat nu pot exista date duplicate. De asemenea, sunt verificate ca ID-urile clientilor sunt valide atunci cand rezerva un zbor.

Constrangeri:

- Chei ID unice: Pentru fiecare colectie imparte este creat un index care trebuie sa fie unic.

Partea de cod care se ocupa de ID-urile unice

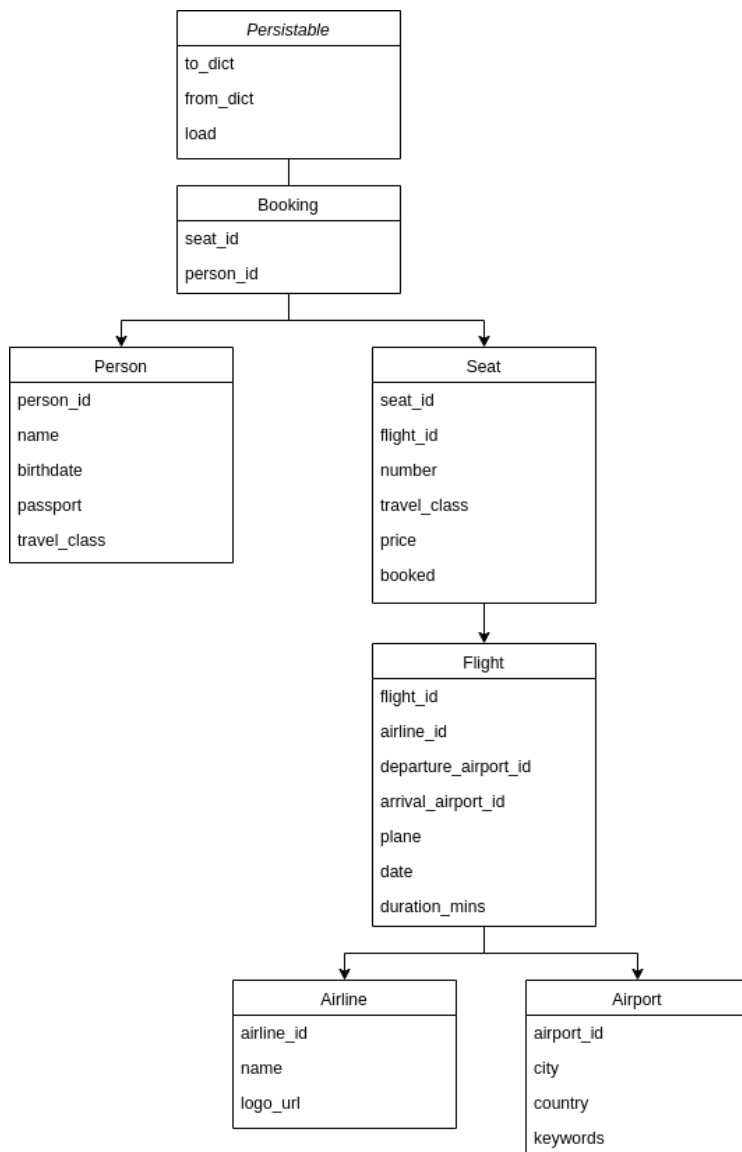
```
mongo.db.airports.create_index([("airport_id", pymongo.ASCENDING)], unique=True)
mongo.db.airlines.create_index([("airline_id", pymongo.ASCENDING)],
unique=True)
mongo.db.flights.create_index([("flight_id", pymongo.ASCENDING)], unique=True)
mongo.db.seats.create_index([("seat_id", pymongo.ASCENDING)], unique=True)
mongo.db.bookings.create_index([("seat_id", pymongo.ASCENDING), ("person_id",
pymongo.ASCENDING)], unique=True)
mongo.db.persons.create_index([("person_id", pymongo.ASCENDING)],
unique=True)
```

- Verificare ID-uri corecte: Atunci cand clientul doreste sa rezerve un zbor, datele lui sunt verificate, se verifica ca ID-ul persoanei este valid, adica ca exista in baza de date, iar atunci cand se face rezervarea, se verifica integritatea operatiei

Partea de cod care verifica corectitudinea datelor din colectii

```
person_query = mongo.db.persons.find({"person_id": person_id})
num_persons = person_query.count()
if num_persons != 1:
    raise ValueError(f"Found {num_persons} people with {person_id}.")

update_result = mongo.db.seats.update_one({"seat_id": seat_id, "booked": False}, {
"$set": {"booked": True}})
if update_result.modified_count != 1:
    return render_template("seat_booking_failed.html", seat_id=seat_id)
mongo.db.bookings.insert_one(booking.to_dict())
```



DESCRIEREA APLICATIEI

Aplicatia are mai multe componente, dintre care cele mai semnificative sunt serverul care constituie backend-ul aplicatiei, partea de generare de date aleatorii ce vor fi inserate pentru inceput in tabele (precum zboruri, aeroporturi, etc...) precum si structura de clase folosite pentru a usura manevrarea datelor. In continuare este prezentata ierarhia claselor folosite in cadrul aplicatiei.

DIAGRAMA DE CLASE

Clasele folosite sunt cele din diagrama alaturata. In cazul clasei Flight, aceasta contine referinte catre obiecte de tip Airline si Airport, clasa Seat are la randul ei referinta catre clasa Flight, reprezentand zborul din care face parte. Clasa Booking va tine referinta catre persoana care a facut rezervarea si locul rezervat.

STRUCTURA CLASELOR SI MODUL DE FUNCTIONARE

Toate clasele folosite mostenesc o clasa comunca, Persistable, deoarece toate implementeaza aceleasi functii (to_dict, from_dict, load) folosite pentru obtinerea si manevrarea mai usoara a datelor atunci cand sunt trimise catre template-ul de Jinja2 care face legatura dintre backend si frontend.

DIAGRAMA DE STĂRI ȘI FLUXUL DE LUCRU (WORKFLOW) PENTRU

APLICAȚIE

Pentru început sunt generate în mod aleator zboruri. Se vor alege aeroporturi dintr-o listă dată și se vor crea zboruri pe baza distanțelor dintre locațiile respective. Pentru a avea o listă cât mai realistă de zboruri, am folosit o matrice de distanțe, pentru a calcula zborul în funcție de distanțele dintre locațiile respective.

Parte din codul de generare de zboruri

```
airplanes = list({
    "Airbus A220": {"rows": 36, "cols": 6},
    "Boeing 777": {"rows": 39, "cols": 8},
}).items()

airlines = {
    "Swiss": (Airline(0, "Swiss",
"https://seeklogo.net/wp-content/uploads/2017/02/swiss-international-
air-lines-logo.png"), 1.09),
    "KLM": (Airline(1, "KLM",
"https://upload.wikimedia.org/wikipedia/commons/thumb/c/c7/KLM_logo.s
vg/500px-KLM_logo.svg.png"), 0.89),
    "Austrian": (Airline(2, "Austrian",
"https://upload.wikimedia.org/wikipedia/commons/thumb/c/c2/Austrian_A
irlines%27_logo_%282018%29.png/800px-Austrian_Airlines%27_logo_%28201
8%29.png"), 1.01),
    "Delta": (Airline(3, "Delta",
"https://1000logos.net/wp-content/uploads/2017/09/Delta-Air-Lines-Log
o.png"), 0.95),
}

airports = {
    "ZRH": {"obj": Airport("ZRH", "Zurich", "Switzerland", []),
"price_modifier": 1.5, "airlines": ["Swiss", "Austrian", "Delta"]},
    "VIE": {"obj": Airport("VIE", "Vienna", "Austria", []),
"price_modifier": 1.1, "airlines": ["Austrian", "KLM"]},
    "SYD": {"obj": Airport("SYD", "Sydney", "Australia", []),
"price_modifier": 1.3, "airlines": ["Swiss", "Delta"]},
    "LHR": {"obj": Airport("LHR", "London", "England", []),
"price_modifier": 1.2, "airlines": ["Austrian", "Swiss", "KLM"]},
    "OTP": {"obj": Airport("OTP", "Bucharest", "Romania", []),
"price_modifier": 0.8, "airlines": ["Swiss"]},
    "JFK": {"obj": Airport("JFK", "New York City", "United States
```



```

of America", []), "price_modifier": 1.4, "airlines": ["Swiss", "KLM",
"Delta"]},
    }

    distance_matrix = {
        "ZRH": [0.0, 1.0, 15.0, 1.0, 2.3, 9.0],
        "VIE": [1.0, 0.0, 14.0, 2.0, 1.3, 10.0],
        "SYD": [15.0, 14.0, 0.0, 16.0, 13.0, 9.0],
        "LHR": [1.0, 2.0, 16.0, 0.0, 3.0, 8.0],
        "OTP": [2.3, 1.3, 13.0, 3.0, 0.0, 11.5],
        "JFK": [9.0, 10.0, 9.0, 8.0, 11.5, 0.0],
    }

    prices = {
        1: {"rows": set(range(0, 9)), "price_modifier": 1.9},
        2: {"rows": set(range(9, 100000)), "price_modifier": 0.9},
    }

```

La inceput sunt generate zboruri pentru primele trei luni ale anului 2020. Dupa ce sunt generate acestea sunt inserate in baza de date.

Parte din codul pentru insertia datelor in colectii

```

    mongo.db.airports.create_index([("airport_id", pymongo.ASCENDING)],
unique=True)
    mongo.db.airlines.create_index([("airline_id", pymongo.ASCENDING)],
unique=True)
    mongo.db.flights.create_index([("flight_id", pymongo.ASCENDING)],
unique=True)
    mongo.db.seats.create_index([("seat_id", pymongo.ASCENDING)],
unique=True)
    mongo.db.bookings.create_index([("seat_id", pymongo.ASCENDING),
("person_id", pymongo.ASCENDING)], unique=True)
    mongo.db.persons.create_index([("person_id", pymongo.ASCENDING)],
unique=True)

    check_insert_many(mongo.db.airports, db_dict["airports"])
    check_insert_many(mongo.db.airlines, db_dict["airlines"])
    check_insert_many(mongo.db.flights, db_dict["flights"])
    check_insert_many(mongo.db.seats, [b for a in
db_dict["seats"].values() for b in a.values()])
    check_insert_many(mongo.db.bookings, db_dict["bookings"])
    check_insert_many(mongo.db.persons, db_dict["persons"])

def check_insert_many(collection, inserted_vals):

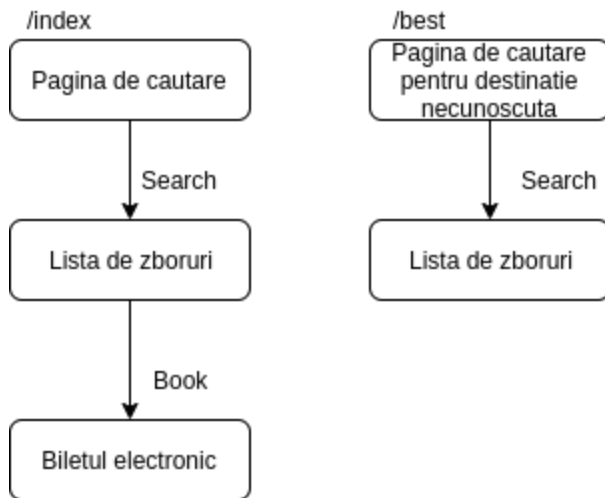
```

```

result = collection.insert_many(a.to_dict() for a in inserted_vals)
if len(result.inserted_ids) != len(inserted_vals):
    raise ValueError(f"Tried to insert: {len(inserted_vals)},
inserted: {len(result.inserted_ids)}")

```

Modul de functionare al aplicatiei web



Utilizatorul va fi intampinat de pagina principala de cautare de zboruri. In interfata grafica, clientul isi va introduce datele personale: Nume, Prenume, Zi de nastere, Numarul pasaportului si Clasa la care doreste biletul.

Flight search

Flight details	
From	Date
LHR	01/15/2020
To	Departure at
JFK	07:00 AM



Passenger	
Name	Birth Date
Bianca Tazlauanu	08/30/1997
Passport number	Travel class
55553497	<input checked="" type="radio"/> 1st <input type="radio"/> 2nd

[Search](#)

Aceste date vor fi folosite pentru cumpararea biletului, precum si pentru verificarile facute la aeroport inainte de zbor. Pe langa acestea, clientul va introduce datele pentru zborul pe care il doreste. Clientul trebuie sa includa destinatia, originea, ziua si ora plecarii. Aplicatia va cauta un zbor intre cele doua orase pentru ziua ceruta, care sa fie la cel mult 48 de ore fata de ora introdusa.

Dupa ce datele sunt furnizate, se va apasa pe butonul Search si vor fi afisate intr-o noua pagina o lista cu zborurile gasite care indeplinesc criteriile cerute.

Raportul pentru zborurile care indeplinesc criteriile de selectie

Flight search			
	Departure	Arrival	Book
	LHR 2020-1-15 12:15 Price: 1391 EUR	JFK 2020-1-15 20:15 Available seats: 35.29%	
	Departure	Arrival	Book
	LHR 2020-1-15 19:09 Price: 1183 EUR	JFK 2020-1-16 03:28 Available seats: 85.37%	
Departure		Arrival	

Dupa ce datele sunt furnizate, se va apasa pe butonul Search si vor fi afisate intr-o noua pagina o lista cu zborurile gasite care indeplinesc criteriile cerute. Dupa cum se vede in exemplul prezentat, au fost gasite 4 zboruri, 2 dintre ele fiind vizibile in imaginea de mai sus. Pagina ofera informatii despre zborurile existente: compania aeriana de care apartine zborul, pretul, ora de decolare si cea de sosire, precum si procentajul locurilor disponibile pentru zborul respectiv. Mai departe, clientul va putea sa rezerve zborul apasand pe butonul Book.

Raportul pentru biletul de avion cumparat

Boarding pass		
Name	Passport no.	Airline
Bianca Tazlauanu	55553497	KLM
Date	Time	Seat
2020-01-15	19:09	1A



Se va deschide o noua interfata cu un raport al biletul clientului pentru zborul respectiv in care se va afisa si locul pe care l-a primit. Clientul poate sa imprime biletul care contine si un QRCode pe care compania aeriana il poate folosi pentru verificarea validitatii biletului.

O alta posibilitate, daca clientul alege Best Flights, acesta poate introduce un aeroport de plecare si o zi si va primi un raport cu cele mai bune zboruri ordonate in functie de pret si punctul de origine.

127.0.0.1:5000/best

Flight search

Flight details

From

LHR

Date

01/16/2020

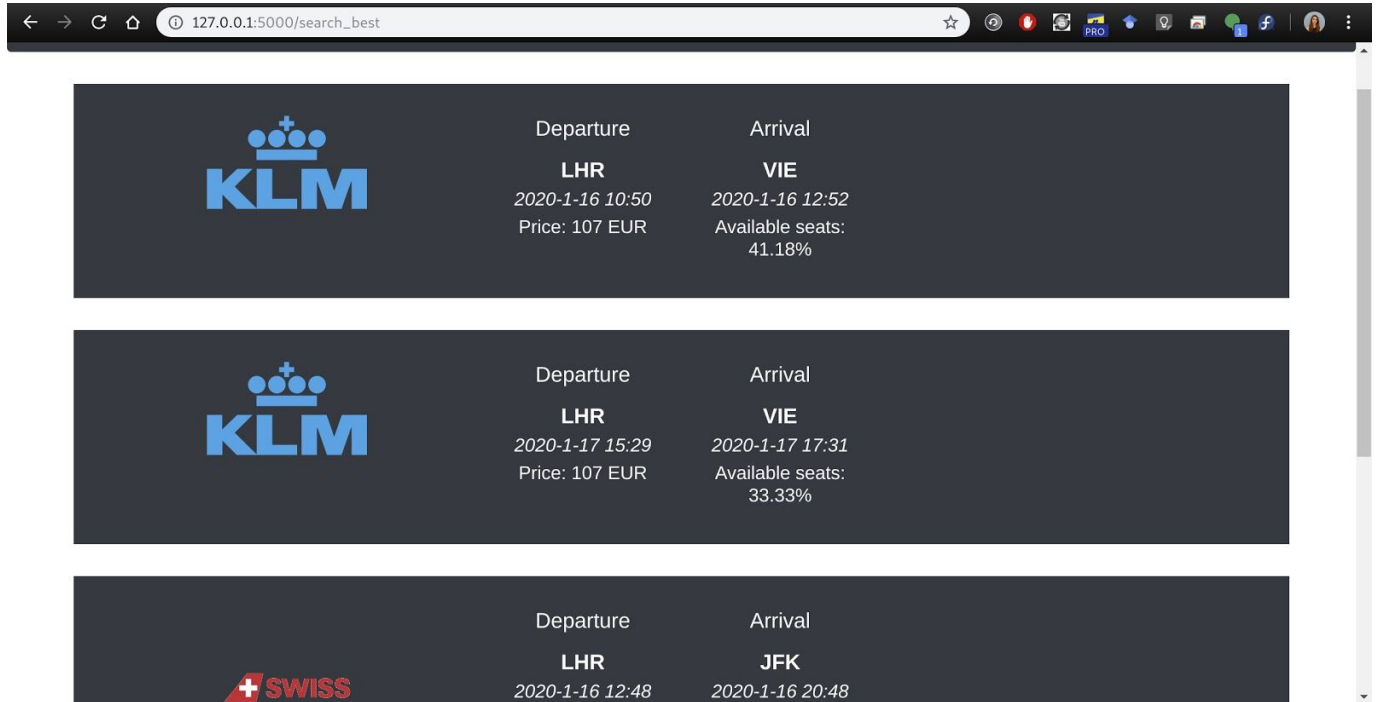
Travel class




1st

2nd

Search

Aceasta este raportul cu rezultatele primite pentru query-ul anterior:



	Departure LHR 2020-1-16 10:50 Price: 107 EUR	Arrival VIE 2020-1-16 12:52 Available seats: 41.18%
	Departure LHR 2020-1-17 15:29 Price: 107 EUR	Arrival VIE 2020-1-17 17:31 Available seats: 33.33%
	Departure LHR 2020-1-16 12:48	Arrival JFK 2020-1-16 20:48

PREZENTAREA MODULUI ÎN CARE SE FACE CONEXIUNEA CU BAZA DE DATE

Aplicatia foloseste pentru partea de server flask, un web application framework, care se conecteaza in faza initiala la serverul de MongoDB, deschis in prealabil.

```
app.config["MONGO_URI"] = "mongodb://localhost:27017/myDatabase"  
mongo = PyMongo(app)
```

Aplicatia comunica cu serverul de mongodb si se foloseste de variabila mongo.db pentru a realiza operatiile pe baza de date. Serverul de mongodb este pornit din terminal cu ajutorul comenzii *sudo systemctl start mongod* sau *mongod --dbpath /home/\$USER/db*. Imediat ce serverul de flask s-a conectat la serverul de mongodb, sunt populate colectiile cu date initiale si se porneste aplicatia web.

Implementare Backend si Frontend

Partea de backend este realizata cu flask, care face maparea intre caile din URL si functiile apelate in server. Pentru partea de frontend, fiecare pagina se foloseste de un template in HTML si Bootstrap care este afisat utilizatorului impreuna cu datele provenite de la baza de date. Pentru a se adauga datele in template-uri, am folosit Jinja2, un web template engine pentru python.

Dupa ce clientul introduce datele personale si datele legate de zborul pe care il doreste, din HTML va pleca cererea catre server. In server se va face cautarea pentru zborul dorit.

Parte din codul pentru functia de cautare

```
@app.route('/search', methods = ["POST"])
def search():
    name = request.values["pass_name"]
    birthdate =
datetime.datetime.strptime(request.values[f"pass_birthdate"],
"%Y-%m-%d")
    travel_class = int(request.values["pass_class"])
    passport = request.values["pass_passport"]

    global person_id
    person = Person(person_id, name, birthdate, passport, travel_class)
    person_id += 1
    mongo.db.persons.insert_one(person.to_dict())

    src_airport = request.values["from"]
    dst_airport = request.values["to"]
    dep_date = request.values["dep_date"]
    dep_time = request.values["dep_time"]

    dep_datetime = datetime.datetime.strptime(f"{dep_date} {dep_time}",
"%Y-%m-%d %H:%S")
    next_day = dep_datetime + datetime.timedelta(hours=48)
    print("Flights between", dep_datetime, next_day)
```

Datele provenite de la user sunt parsate in server si se foloseste MapReduce pentru obtinerea procentului de ocupabilitatea a unui zbor:

```
occupancy = mongo.db.seats.map_reduce(
    Code("""function() {
        emit(this.flight_id, { booked: this.booked });
    }"""),
    Code("""function(key, values) {
        var results = {"flight_id": key, "occupancy": 0.0};
        occupied = 0;
        total = 0;
        values.forEach(function (value) {
            total++;
            if (value["booked"] === true)
                occupied++;
        });
    });
```

```

        results["occupancy"] = occupied / total;
        return results;
    }"""),
    "occupancy", query={"flight_id": {"$in": flight_ids}})

occupancy = {v["value"]["flight_id"]:v["value"]["occupancy"] for v in
occupancy.find()}

```

FUNCTII FOLOSITE

Functie de map reduce pentru calcularea gradului de ocupare al fiecarui zbor in parte

```

occupancy = mongo.db.seats.map_reduce(
    Code("""function () {
        emit(this.flight_id, { booked: this.booked });
    }"""), Code("""function (key, values) {
        var results = {"flight_id": key, "occupancy": 0.0};
        occupied = 0;
        total = 0;
        values.forEach(function (value) {
            total++;
            if (value["booked"] === true) occupied++;
        });
        results["occupancy"] = occupied / total;
        return results;
    }"""), "occupancy", query={
    "flight_id": {"$in": flight_ids}
})

occupancy = {v["value"]["flight_id"]: v["value"]["occupancy"] for v
in occupancy.find()}

```

Functie de map reduce pentru calcularea preturilor fiecarui zbor. Pe baza datelor obtinute se pot sorta preturile zborurilor

```

price = mongo.db.flight_seats.map_reduce(
    Code("""function () {
        var price_sum = 0;
        var seats = 0;
        this.seats.forEach(function (value) {
            price_sum += value["price"];
            seats++;
        });
    }"""),
    "price", query={
    "flight_id": {"$in": flight_ids}
})

```

```

        emit(this.airline.airline_id, { price: price_sum, seats: seats
    });
    }"""), Code("""function (key, values) {
        var price_sum = values.reduce((a, b) => a["price"] + b["price"],
    0);
        var seat_sum = values.reduce((a, b) => a["seats"] + b["seats"],
    0);
        var results = {"airline_id": key, "price": price_sum / seat_sum};
        return results;
    }"""), "price")
    price = {v["value"]["airline_id"]: v["value"]["price"] for v in
price.find()}

```

Functie de aggregate care face join intre colectia seats si flights, i se asociaza fiecarui seat, zborul corespunzator

```

found_flights = db.flights.aggregate([
    matcher,
    { "$lookup": {
        "from": "seats",
        "localField": "flight_id",
        "foreignField": "flight_id",
        "as": "seat"
    }
    },
    { "$unwind": "$seat" },
    { "$group": {
        "_id": "$flight_id",
        "flight": { "$first": "$CURRENT" },
        "seats": { "$push": "$seat" },
    }
    }
])

```


CONCLUZII

Aplicatia este una complexa care poate fi folosita pentru cautarea unor zboruri la un pret optim sau pentru cautarea unei rute specifice pentru a cumpara un bilet de avion. Interfata grafica este usor de folosit si intuitiva, iar rezultatele sunt oferite in mod corect utilizatorilor.

BIBLIOGRAFIE

<https://docs.mongodb.com/manual/indexes/#Indexes-unique%3Atrue>
<https://docs.mongodb.com/manual/installation/>
<https://medium.com/@LondonAppBrewery/how-to-download-install-mongodb-on-windows-4ee4b3493514>
https://www.tutorialspoint.com/mongodb/mongodb_map_reduce.htm
<https://tutorials.technology/tutorials/introduction-to-mapreduce-using-mongodb.html>
<https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/>
https://www.w3schools.com/bootstrap/bootstrap_get_started.asp
<https://www.fullstackpython.com/flask.html>