

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Places – Aplicație de booking**

propusă de

***Vrabie Tudor-Octavian***

**Sesiunea:** *Iulie, 2018*

**Coordonator științific**

**Drd. Colab. Florin Olariu**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI  
FACULTATEA DE INFORMATICĂ

# **Places – aplicație de booking**

***Tudor-Octavian Vrabie***

**Sesiunea:** *Iulie, 2018*

Coordonator științific

***Drd. Colab. Florin Olariu***

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele \_\_\_\_\_

Data \_\_\_\_\_ Semnătura \_\_\_\_\_

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemnatul(a).....

domiciliul în .....

născut(ă) la data de ....., identificat prin CNP .....,

absolvent(ă) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de

..... specializarea ....., promoția

....., declar pe propria răspundere, cunoscând consecințele falsului în

declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.

1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul:

\_\_\_\_\_elaborată sub îndrumarea dl. / d-na

\_\_\_\_\_, pe care urmează să o susțină în fața

comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Datată azi, .....

Semnătură student .....

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezență declar că sunt de acord ca Lucrarea de licență cu titlul „*Places-aplicatie de booking*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezenței lucrări de licență.

Iași, *data*

Absolvent *Prenume Nume*

---

(semnătura în original)

## Cuprins

Introducere .....	5
Contribuții .....	8
Capitolul 1: Descrierea problemei .....	9
Capitolul 2: Aplicația server .....	13
2.1 Tehnologii folosite .....	13
2.2 Arhitectura aplicației server .....	16
2.2.1 Nivelul responsabil de tranzacțiile pe baza de date .....	17
2.2.2. Nivelul de Business .....	18
2.2.3. Nivelul de gestiune a cererilor .....	20
2.3 Prelucrarea datelor pentru răspuns .....	22
Capitolul 3: Aplicația client .....	23
3.1 Tehnologii folosite .....	23
3.2 Arhitectura aplicației client .....	23
Capitolul 4: Modul de funcționare al aplicației .....	26
Concluzie .....	32
Bibliografie .....	33

## Introducere

Pentru a motiva alegerea temei propun să facem următorul exercițiu de imaginație: câte din cele 24 de ore ale unei zile ni le petrecem folosind Internetul? Indiferent de numărul la care se gândește fiecare dintre noi, acesta e în realitate mult mai mare. Importanța utilizării mediului online în activitățile curente a devenit o necesitate, iar eficiența și limitarea temporară reprezintă factori cheie în argumentarea alegerii fiecăruia dintre utilizatori.

Fie că folosim telefonul, laptopul sau tableta, obișnuim să ne petrecem din ce în ce mai mult timp în mediul online, atât acasă, cât și la serviciu. Indiferent dacă ne petrecem timpul pe rețele de socializare, facem cumpărături sau ne plătim facturile, multe din activitățile noastre au ajuns să depindă de mediul online, în ideea de a facilita și ușura accesul nostru către orice activitate, către orice domeniu.

Unul dintre cele mai relevante exemple în acest sens, îl reprezintă planificarea unei călătorii. În momentul în care alegem o destinație de vacanță, căutările pentru hotel și cel mai eficient zbor pot fi dificile. Pentru a înțelege de unde iau naștere aceste dificultăți, este important să înțelegem care sunt problemele cu care se confruntă în acest moment aplicațiile web de căutare a zborurilor existente pe piață.

În primul rând, multe dintre aplicațiile destinate căutării zborurilor prezente în lumea online-ului se dovedesc a fi greu de utilizat, în sensul în care solicită o multitudine de informații care pot fi indisponibile în momentul începerii căutării.

În cel de-al doilea rând, filtrele care pot fi aplicate în acest moment în căutarea unui zbor sunt limitate. Deși durata totală a zborului și limita de preț pot fi alese cu ușurință în aplicațiile web existente, în cazul în care unul din parametri se dorește a fi schimbat, este nevoie de reluarea căutării totale, ceea ce reprezintă un timp adițional irosit.

Căutarea celui mai eficient zbor poate deveni complicată în cazurile în care nu există zboruri directe între orașul din care călătorim și destinația aleasă, sau în cazul în care zborurile directe sunt costisitoare.

În încercarea de a găsi un zbor care să se încadreze în bugetul destinat călătoriei, durata sau numărul escalelor sunt unele dintre cele mai importante aspecte de luat în calcul.

Având în vedere limitările motoarelor de căutare existente, menționate anterior, aplicația web **Places**, pe care am dezvoltat-o, urmărește rezolvarea fiecăreia dintre acestea.

În primul rând, informația prezentată utilizatorului este concisă, iar filtrările exacte. Fără a fi nevoie de o multitudine de informații anterioare rezervării efective, pot fi selectate orașele și datele aferente călătoriei. Dacă una din date se dorește a fi modificată, acest lucru nu implică resetarea căutării, aceasta putând fi modificată, iar noile rezultate vor fi disponibile cu ușurință. Design-ul simplist pe care această aplicație îl propune, vine în întâmpinarea utilizatorilor, reducând considerabil timpul necesar rezervării.

În cel de-al doilea rând, **Places** rezolvă una dintre cele mai deranjante limitări ale aplicațiilor folosite în prezent, în sensul în care, este primul motor de căutare ce oferă informații și sugestii de petrecere a timpului avut într-o escală. Astfel, dacă un utilizator alege pentru a zbura către destinația dorită, un zbor cu una sau mai multe escale, pe durata acestora va avea opțiuni pentru a-și petrece timpul disponibil. Într-un perimetru de 40 kilometri distanță față de aeroportul în care are loc escalea, vor fi prezentate multiple atracții turistice: muzee, catedrale, dar și alte puncte de interes: hoteluri, restaurante, cafenele și parcuri de distracții. Noutatea adusă de această opțiune, relevă din faptul că dincolo de a vizita orașul ales ca destinație pentru vacanță, timpul petrecut într-o escală poate fi folosit pentru a explora cu ușurință atracțiile turistice ale unui alt oraș. Acest beneficiu poate fi însoțit și de o reducere substanțială a costului călătoriei.

Metodologiile folosite în dezvoltarea produsului software sunt Agile și Kanban, care au permis fluidizarea întregului proces de scriere a codului sursă.

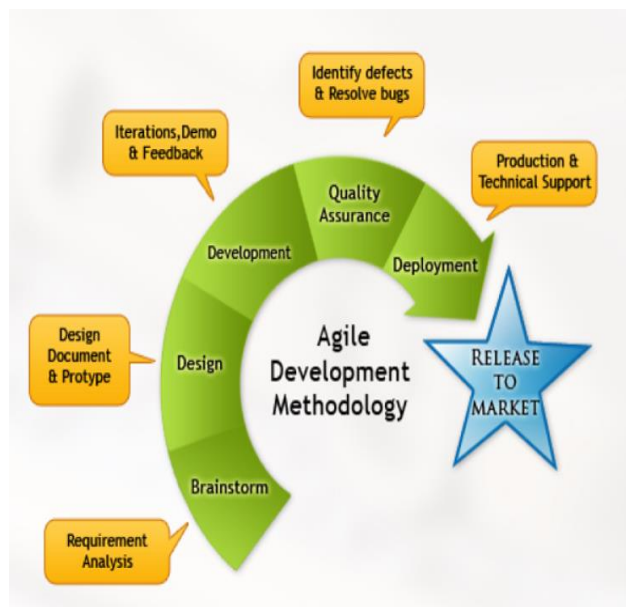


Figure 1 <http://pakar.co.id>

### Agile:

- permite adaptarea foarte ușoară a aplicației conform feedback-ului primit de la client.
- permite lansarea unui produs de tip MVP ( Most Viable Product) în fazele inițiale de dezvoltare, iar pe parcurs vor putea fi aduse îmbunătățiri/modificări.
- conduce la dificultăți de estimare a timpului investit în dezvoltare, deoarece inițial specificațiile sunt puține și

vagi, pe parcurs fiind destul de dificil de a preconiza preferințele finale ale clientului.

- este dificilă aducerea unui nou membru care să lucreze la proiect, acestuia fiindu-i foarte greu să se adapteze la cerințele clientului, nefiind inclus în sesiunile de tip SCRUM zilnice desfășurate până în momentul integrării lui.

## Kanban:

- Permite concentrarea efortului strict pe aspectele cu adevărat importante, pentru a grăbi dezvoltarea unui prototip viabil în orice stadiu al proiectului.
- Permite menținerea costurilor minime, nefiind necesare cursuri adiționale de introducere a unui programator în dezvoltarea produselor ce au la bază metodologia de dezvoltare Kanban.
- Are o tehnică de vizualizare simplistă; astfel, se poate aprecia cu ușurință stadiul în care se află proiectul.
- Uneori poate fi dificil de estimat timpul de rezolvare a unei sarcini de lucru.
- Este ideal pentru echipe cu număr redus de programatori.

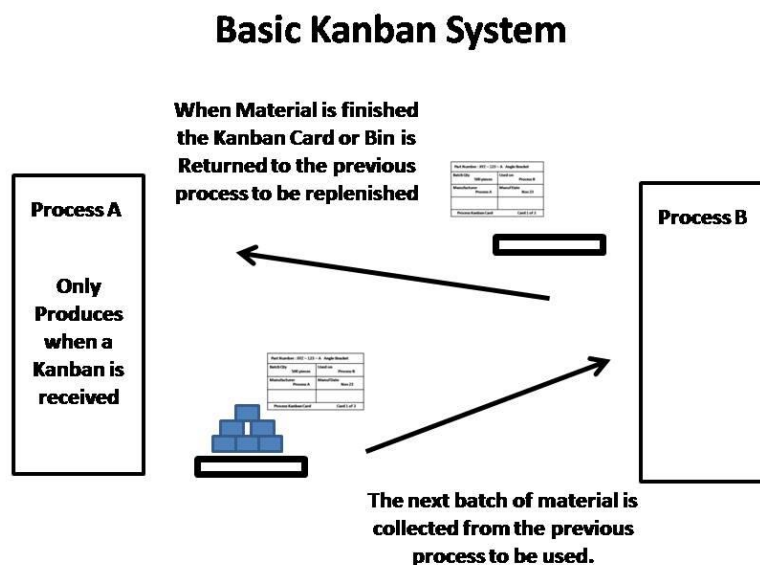


Figure 2 <http://leanmanufacturingtools.org>



## Contribuții

Din experiența personală, de foarte multe ori am preferat să renunț complet la anumite zboruri întrucat nu știam ce alternative am în apropierea aeroportului în orele de așteptare. Aplicația de față are o arhitectură care permite planificarea unui tur al obiectivelor din preajma aeroportului. Distanțele dintre acestea și locul în care se află consumatorul sunt precizate în km, pe o rază de maxim 40. Astfel, în funcție de preferințe, necesități și opțiunile personale ale fiecăruia se poate realiza o căutare avansată, după o serie de cuvinte cheie, printr-un API oferit de Google.

Tema acestei teze de licență a fost atent discutată cu profesorul coordonator, iar în urma întâlnirilor săptămânale în care am prezentat progresul și am primit feedback, am ajuns la concluzia că noutatea pe care o aduce aplicația de față poate fi benefică consumatorilor de rând.

La baza scrierii acestei lucrări au stat experiențele și cunoștințele acumulate pe parcursul celor trei ani de facultate.

Când vine vorba despre aplicații de booking marea majoritate a utilizatorilor se așteaptă să găsească o interfață cât mai simplă și zboruri cât mai actualizate. Pentru a oferi aceste standarde s-a recurs la o serie de tehnologii menite să ofere performanță pe partea de procesare de date dar și simplitate când vine vorba de interfață.

Astfel pentru alegerea tehnologiilor s-a dorit foarte mult păstrarea unui nivel foarte înalt de calitate și s-a recurs la un limbaj de backend și framework care să ofere securitate, viteză și posibilitatea de a extinde aplicația prin adaugarea de noi module.

Pe partea de frontend s-a folosit Angular. Acest framework oferă o interfață inedită. Ce a stat la baza alegerii acestui framework a fost faptul că acesta are deja foarte multe funcționalități integrate dar și faptul că aplicația poate fi ușor extinsă prin adăugarea de noi componente și funcționalități. Totodată framework-ul este oferit și întreținut de Google ceea ce constă în păstrarea unui etalon de calitate, ce constă într-o bună documentație și posibilitatea de a upgrada aplicația la următoarea versiune, ceea ce aduce noi posibilități de rezolvare, dar și rezolvă, dacă există, anumite brese, bug-uri.

## Capitolul 1: Descrierea problemei


La momentul actual, pe piață există foarte multe aplicații de booking din care consumatorii ar putea alege. Totuși, mare parte dintre acestea prezintă interfețe complexe, cu foarte multe funcționalități, de cele mai multe ori inutile, care descurajează clientul. Astfel, acesta recurge cel mai frecvent la agențiile de planificare călătorii, care fac rezervarea propriu-zisă în numele consumatorului, la prețul pieței, la care se adaugă comisionul. În cele din urmă, clientul va fi defavorizat din mai multe puncte de vedere:

- bani- prețul de cost final al călătoriei fiind mai mare cu tot cu comisioanele agențiilor, cu toate că de foarte puține ori ruta dorită va fi una directă și care să se încadreze în bugetul propus;

- timp- pe lângă numeroasele minute pierdute în faza inițială, în care clientul încearcă să facă pe cont propriu rezervarea, se adaugă timpul adițional petrecut la agenția de rezervări, fie el online, fie telefonic, fie față în față cu unul dintre agenți, la una dintre sucursale;

De cele mai multe ori, orice călătorie are la bază un buget prestabilit, fie el mic, sau mare, după posibilitățile financiare ale fiecăruia. Astfel, pentru marea majoritate a populației, cel mai dorit tip de zbor este cel la prețul cel mai mic și de cea mai scurtă durată. Cel mai frecvent, zborurile mai ieftine nu sunt directe, ci prezintă un anumit număr de escale, de unde rezultă un anumit număr de ore pierdute care, deseori sunt privite ca fiind inconveniente de către călători.

Spre exemplu, un zbor București-New York poate avea o escală de până la 6-8 ore în Istanbul. Distanța dintre aeroport și centrul orașului se poate parcurge în aproximativ 38 de minute, conform Google Maps, ceea ce rezultă în posibilitatea clientului de a petrece timp de calitate vizitând un alt oraș decât cel considerat a fi destinația finală. La prima vedere, pentru unii, transferul spre oraș și vizitarea efectivă a unor obiective ar putea fi nimic mai mult decât bani suplimentari pierduți. Totuși, punând în balanță diferența de preț dintre zborul direct, sau cu o escală foarte scurtă și zborul cu escală de mai mult de 2-3 ore, plus posibilitatea de a vedea ceva nou, pentru o anumită categorie de călători, acest lucru nu reprezintă un impediment ci o oportunitate.


 Întoarcere · vin., 13 iul.

SELECTAȚI ACEST ZBOR

6.054 RON  
 călătorie dus-întors

22:00 · Aeroportul Internațional John F. Kennedy (JFK)
 

Scaun cu spătar foarte înclinat  
 Scaune cu priză și USB  
 Videoclip la cerere

Durata călătoriei: 8 h 30 min. · **În timpul nopții**

12:30<sup>+1</sup> · Aeroportul Chopin Varșovia (WAW)
 

LOT · Economic premium · Boeing 787 · LO 27  
 Cu întârzieri frecvente de peste 30 de minute

Escală de 2 h 0 min. · Varșovia (WAW)

14:30<sup>+1</sup> · Aeroportul Chopin Varșovia (WAW)
 

Durata călătoriei: 1 h 50 min.

17:20<sup>+1</sup> · Aeroportul Internațional Henri Coandă București (OTP)
 

LOT · Economic premium · Boeing 737 · LO 645  
 Avion și echipaj de bord de la Blue Air

Imagine 1- Google Flights

Figure 3 preluata de pe Google Flights

Imagine 3:

- Zbor București-New-York cu escală în Varșovia de 2ore;
- 6.054 Ron;
- escala este de scurtă durată, iar pasagerii nu au timp să părăsească aeroportul;



Plecare · vin., 13 iul.

SELECȚAȚI ACEST ZBOR

4.315 RON  
călătorie dus-întors



05:55 · Aeroportul Internațional Henri Coandă București (OTP)

Durata călătoriei: 1 h 15 min.

07:10 · Aeroportul Internațional Atatürk (IST)

Turkish Airlines · Economic · Boeing 737 · TK 1042

Spațiu mediu pentru picioare (79 cm)

Scaune cu USB

Videoclip la cerere

Escală de 11 h 15 min. · Istanbul (IST)

18:25 · Aeroportul Internațional Atatürk (IST)

Durata călătoriei: 11 h 5 min.

22:30 · Aeroportul Internațional John F. Kennedy (JFK)

Turkish Airlines · Economic · Boeing 777 · TK 11

Spațiu mediu pentru picioare (79 cm)

Wi-Fi

Scaune cu priză și USB

Videoclip la cerere

Figure 4 preluata de pe Google Flights

Imagine 4:

-zbor București-New York cu escală în Istanbul de aproape 12 ore;

-4.315 Ron;

-escala este pe durata zilei, ceea ce înseamnă că aceste ore pot fi utilizate pentru a vizita o parte din obiectivele orașului;

-diferența de preț de aproape 2.000 Ron poate fi folosită pentru transferul de la aeroport spre oraș și pentru biletele de intrare la muzee, alte atracții turistice, cafenele, suveniruri, etc;

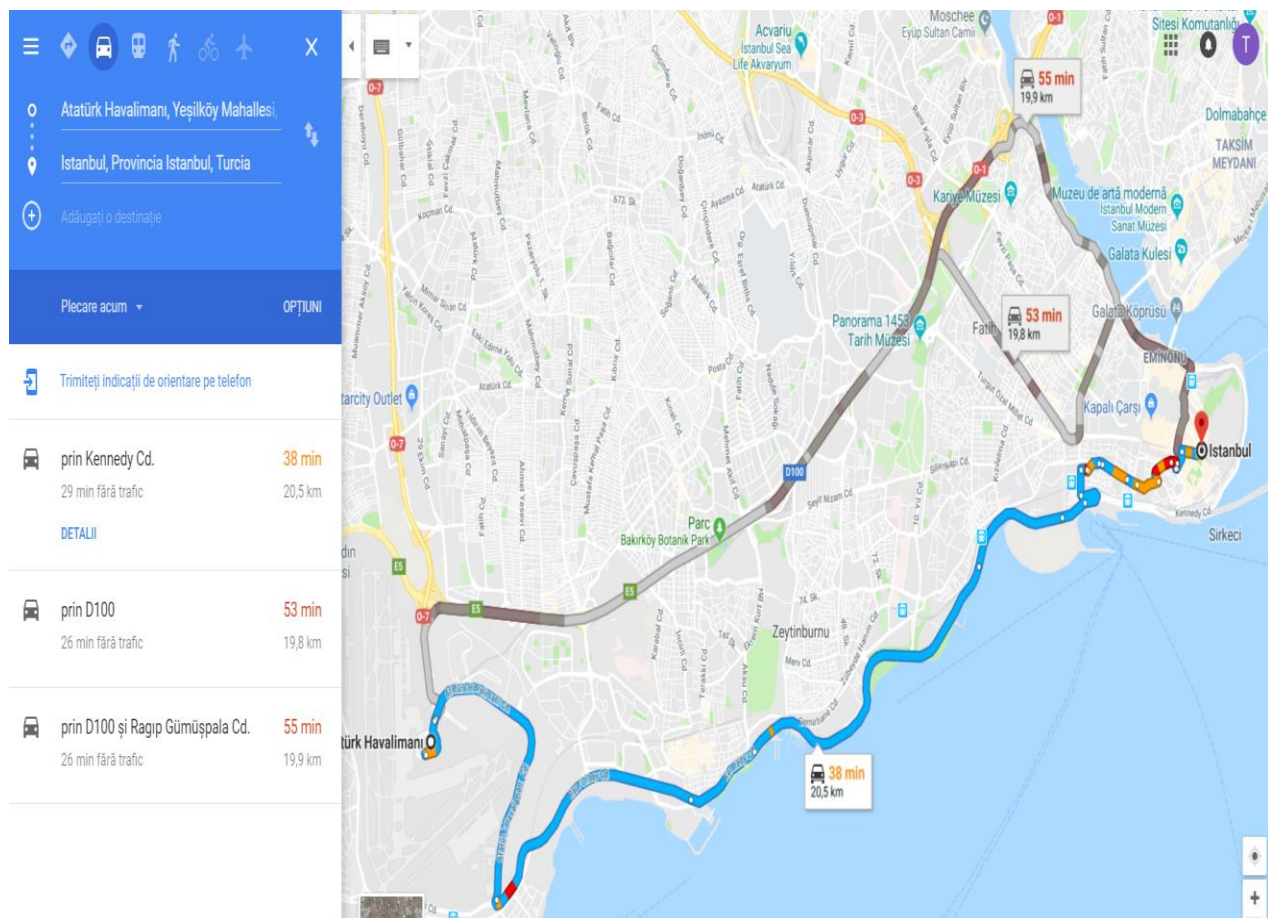


Figure 5 preluata de pe Google Maps

Imagine 5:

- se observă cum Google Maps estimează că distanța dintre aeroport și oraș poate fi parcursă într-un interval orar de aproximativ 38-55 de minute, în funcție de condițiile de trafic care nu pot fi anticipate;
- din cele aproximativ 12 ore de escală, la un calcul simplu estimativ, scăzând cele aproximativ două ore petrecute pe drum, între aeroport și oraș, rămân cca 10 ore, timp în care se pot vizita o parte dintre atracțiile pe care le oferă orașul Istanbul.

Astfel, în funcție de nevoile și preferințele fiecăruia, în funcție de categoria de vârstă și de ocupația călătorilor, se poate alege varianta mai scumpă, cu o durată a călătoriei mai mică, sau o variantă mai ieftină, cu o escală considerabil mai lungă, care permite utilizarea diferenței de bani și de ore în scop cultural

## Capitolul 2: Aplicația server

### 2.1 Tehnologii folosite

Pentru dezvoltarea aplicației am ales tehnologii de ultimă generație, care permit crearea unui produs software de calitate ce va putea fi modificat și ce va putea primi noi module astfel încât să satisfacă unul sau mai multe eșantioane de clienți.

Pe partea de baze de date am ales să folosesc Mysql. Motivul care a stat la baza alegerii mele este acela că Mysql este un sistem de management al bazelor de date relaționale cross platform (poate fi instalat pe orice OS) și faptul că în momentul în care aplicația va avea un număr mare de utilizatori, în eventualitatea unei erori de server ce conduce la oprirea acestuia din funcționare se poate deschide foarte simplu un alt access-point care să permită replicarea conexiunii la baza de date. Acest aspect este observabil nu în faza inițială a aplicației ci pe termen lung când vor fi realizate tranzacții pe baza de date de ordinul miilor/milioanelor pe secundă.

Un alt motiv pentru care am ales să folosesc Mysql este acela că este foarte ușor de integrat cu framework-ul folosit pe partea de backend ceea ce îmi permite să realizez tranzacții în mod cât mai eficient pe baza de date și într-un mod cât mai sigur evitând astfel diferite posibilități a unor breșe de securitate în aplicație cum ar fi SQL injection.

În momentul actual baza de date conține tabele cu date despre numeroase companii de aviație (cca 1000), orașe (cca 120.000), aeroporturi (cca 4000). Aceste date vor putea fi folosite în scopul introducerii de noi module în aplicație. Spre exemplu în orice moment poate fi introdusă în aplicație o pagină cu date despre companiile de aviație din lumea întreagă.

Un alt exemplu este acela că oricând poate fi adăugată o pagină care să permită îmbinarea tabelului cu informații despre orașe cu API-ul oferit de Google (Google maps) pentru a oferi utilizatorilor posibilitatea de a vedea pe hartă în timp real unde se află aeroportul și pentru a aproxima timpul în care ar putea ajunge din aeroporturi în diferite puncte cheie. Acest lucru ar fi optim deoarece Google Maps estimează distanțele în funcție de numeroase criterii cum ar fi condițiile meteo sau condițiile de trafic.

Pe partea de backend am ales să folosesc limbajul PHP împreună cu framework-ul Laravel versiunea 5.6. Avantajele utilizării acestui limbaj împreună cu acest framework le reprezintă următoarele:

#### A. Securitatea

Utilizând Laravel se poate implementa foarte repede un modul de autentificare pe partea de server care să fie sigur și conform cu cele mai înalte standarde de securitate. Acest lucru este posibil întrucât Laravel dispune de un pachet oficial numit Passport, care permite realizarea unui modul ce are la bază OAuth 2.0 care este protocolul de autentificare standardizat din punct de vedere industrial.

Acest framework previne multe din atacurile cibernetice existente, un exemplu fiind atacul de tip brute-force. Cu ușurință programatorul poate seta un număr maxim de requesturi pe minut/per endpoint.

Criptarea datelor este realizată prin intermediul standardului AES-256 folosind o cheie de 32 de biți, generată în timpul instalării inițiale ale proiectului.

În cazul optării pentru o implementare de tip MVC (Model View Controller), Laravel oferă și protecție împotriva atacurilor de tip XSS printr-o filtrare a datelor în momentul în care sunt încărcate în pagină.

Permite o filtrare foarte ușoară a request-urilor către server, fapt ce oferă spre exemplu posibilitatea întoarcerii unui status de tip 403 Forbidden în cazul în care utilizatorul curent logat nu îndeplinește anumite criterii pentru a putea realiza acel request.

Se pot seta foarte ușor Headere de securitate, cum ar fi: X-Frame-Options.

#### B. Eloquent ORM (Object relational mapping)

Laravel folosește Eloquent ORM care permite lucrul cu bazele de date într-un mod foarte eficient. Fiecare tabel din baza de date are asociat un Model care este folosit pentru a interacționa cu tabelul respectiv.

Modelul este folosit și pentru a interoga baza de date dar și pentru a insera date noi. Totodată Eloquent permite gestionarea foarte simplă a relațiilor dintre tabele prin oferirea unor funcții specifice care permit realizarea de interogări complexe într-un mod cât mai simplist.

Modul în care codul a fost scris este bazat pe Repository Pattern. Acesta este folosit pentru crearea unui nivel de abstractizare între date și logica de business a aplicației. Acest lucru are un beneficiu foarte mare întrucât este produsă o modularizare a codului, fapt care este foarte util atunci când apare o schimbare în logica care ține de baza de date.

Alte beneficii ale utilizării Repository Pattern sunt:

- minimizarea codului duplicat
- se decuplează logica aplicației

```
<?php

namespace App\Http\Controllers;

use App\Airline;
use App\Http\Transformers\FlightSearchTransformer;
use App\Services\FlightService;
use Carbon\Carbon;
use Illuminate\Http\Request;

class FlightSearchController extends ApiController {
    public function index(Request $request, FlightService $service,
        FlightSearchTransformer $transformer) {
        $params = [
            "flyFrom" => $request->input('from'),
            "to" => $request->input('to'),
            "dateFrom" => Carbon::parse($request->input('fromDate'))
                ->format('d/m/Y'),
            "dateTo" => Carbon::parse($request->input('fromDate'))
                ->format('d/m/Y'),
            "partner" => "picky",
            "adults" => $request->input('passengers', 1),
            "children" => $request->input('children', 0),
            "directFlights" => $request->input('withConnections', 0),
            "maxstopovers" => $request->input('maxstopovers', 10),
            "stopovertoto" => $request->input('duration', "48:00"),
            "typeFlight" => $request->input('typeFlight', 'round'),
            "price_to" => $request->input("price_to", 1000),
            "limit" => 10
        ];
        if($request->input('typeFlight', 'round') === 'round') {
            $params["returnTo"] = Carbon::parse($request->input('toDate'))-
            >format('d/m/Y');
            $params["returnFrom"] = Carbon::parse($request->input('toDate'))-
            >format('d/m/Y');
        }
        $data = $service->getFlights($params);
        return $this->item($data, $transformer);
    }
}
```

Figura 6 -Exemplu de Controller



După cum se poate observa în imagine, în interiorul Controller-ului nu se face altceva decât să se apeleze o funcție dintr-un repository, fără a se avea cunoștințe despre cum datele vor fi manipulate. Acest lucru ajută la păstrarea unui cod cât mai lizibil și la evitarea scrierii de logică de business în controller.

## 2.2 Arhitectura aplicației server

În acest subcapitol voi prezenta structura aplicației server, detaliile de implementare și resursele folosite.

În cadrul aplicației care se ocupă cu gestionarea logicii de pe server s-a dorit obținerea unor performanțe cât mai mari, deoarece căutarea de zboruri înseamnă procesarea unor cantități de date foarte mari într-un timp relativ scurt.

Astfel, modulul server este o aplicație de tip REST monolit. Am ales să realizez o aplicație monolit ca alternativă la o aplicație bazată pe microservicii, deoarece adeseori, o aplicație monolit este mai rapidă datorită numărului redus de requesturi realizate din interfață.

Totodată API-ul a fost realizat în așa fel încât să includă și dependency injection. Dependency injection este o tehnică folosită pentru obținerea unui cuplaj foarte mic între obiecte. Astfel, o clasă nu creează instanțe de obiecte, ci îi sunt oferite cel mai adesea ca și parametri, fie în funcții, fie în constructor. Prin dependency injection se poate respecta mai ușor “Single responsibility principle”, principiu care se referă la faptul că o clasă sau un modul ar trebui să fie responsabil pentru o singură funcționalitate oferită de produsul software.

Aplicația a fost realizată pe mai multe straturi:

- Primul nivel este responsabil de tranzacțiile pe baza de date
- Cel de al doilea nivel îl reprezintă nivelul de Business
- Al treilea nivel este nivelul de gestiune a cererilor

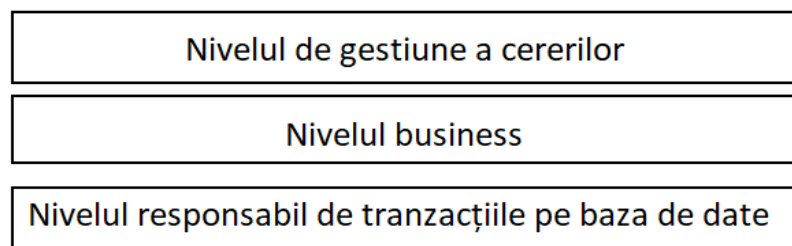


Figura 7- Arhitectura aplicației

## 2.2.1 Nivelul responsabil de tranzacțiile pe baza de date

Nivelul responsabil de tranzacțiile pe baza de date este o componentă esențială a aplicației întrucât deoarece aici se realizează crearea și administrarea bazei de date.

O provocare mare când vine vorba de bazele de date este aceea de a modifica o resursă fără ca aceasta să influențeze în vreun fel datele deja existente. Pentru a rezolva această problemă am ales să folosesc sistemul de *migrații* oferit de Laravel.

Migrațiile sunt asemenea unui sistem de versionare, acestea permițând modificarea cu ușurință a bazei de date deja existente în uz. Totodată migrațiile sunt foarte folositoare atunci când baza de date este instalată pe un sistem nou. În procesul de scriere a codului acestea permit foarte ușor trecerea de la o versiune la alta a bazei de date.

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateAirportsTable extends Migration {

    public function up() {
        Schema::create('airports', function (Blueprint $table) {
            $table->increments('id');
            $table->string('code')->nullable()->default(null);
            $table->string('latitude')->nullable()->default(null);
            $table->string('longitude')->nullable()->default(null);
            $table->string('name')->nullable()->default(null);
            $table->string('city')->nullable()->default(null);
            $table->string('state')->nullable()->default(null);
            $table->string('country')->nullable()->default(null);
            $table->string('woeid')->nullable()->default(null);
            $table->string('timezone')->nullable()->default(null);
            $table->string('phone')->nullable()->default(null);
            $table->string('type')->nullable()->default(null);
            $table->string('email')->nullable()->default(null);
            $table->string('url')->nullable()->default(null);
            $table->string('runway_length')->nullable()->default(null);
            $table->string('elev')->nullable()->default(null);
            $table->string('icao')->nullable()->default(null);
            $table->string('direct_flights')->nullable()->default(null);
            $table->string('carriers')->nullable()->default(null);
            $table->timestamps();
        });
    }

    public function down() {
        Schema::dropIfExists('airports');
    }
}
```

Figure 8 Structura migrației

În figura 8 este prezentată structura unei migrații. Aceasta are două funcții: “up” și “down”. Când migrația este rulată inițial, este apelată funcția “up”, iar în momentul în care se dorește întoarcerea la baza de date de dinainte de schimbare, se apelează funcția “down”.

### 2.2.2. Nivelul de Business

La nivelul de business se regăsesc clasele care permit intermedierea între celelalte două nivele. În principiu, nivelul acesta este nucleul aplicației, aici fiind realizate toate procesările de date.

```
<?php namespace App\Repositories;

use App\Airport;
use App\City;

class AirportRepository extends Repository {
    function assignedModel() {
        return app(Airport::class);
    }
    public function getAirports($request) {
        $cities = Airport::where('city', 'like' ,
            '%'. $request->input('search', null).'%')
            ->whereNotNull('name')
            ->select('city', 'code')
            ->get();
        return $cities;
    }
}
```

Figure 9 -Nivelul de business

În figura 9, se poate observa cum în clasa “AirportRepository”, se poate apela funcția “getAirports”, care primește ca și parametru, un set de date. Dacă în acest set de date există o proprietate cu numele “search”, această funcție va returna toate aeroporturile care conțin în numele orașului, secvența de caractere conținută de valoarea acesteia. În cazul în care această proprietate nu este specificată, vor fi returnate toate aeroporturile conținute de baza de date.

După cum se poate observa, fiecare clasă din nivelul de business, extinde o clasă generică numită “Repository”, aceasta conținând un set de funcții ajutătoare pentru generarea mai ușoară a seturilor de date.

```

<?php
namespace App\Repositories;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;

abstract class Repository {
    protected $request;

    public function getPerPageFrom(Request $request) {

        $request->has('per_page') ? $perPage = $request->input('per_page') :
        $perPage = 10;
        return $this->getModelFrom($request)->paginate($perPage);
    }
    public function getModelFrom(Request $request) {
        return $this->getModel(
            $request->has('field') ? $request->input('field') : [],
            $request->has('op') ? $request->input('op') : [],
            $request->has('term') ? $request->input('term') : [],
            $request->has('order_by_column') ? $request->
            >input('order_by_column') : [],
            $request->has('order_by_type') ? $request->input('order_by_type')
            : [],
            $request->has('with') ? $request->input('with') : []
        );
    }

    public function getModel($fields = [], $ops = [], $terms = [],
    $orderByColumns = [], $orderByTypes = [], $with = []) {
        $model = $this->assignedModel();

        $model = $this->applyOrderBy($model, $orderByColumns, $orderByTypes);
        $this->applyFilters($model, $fields, $ops, $terms);
        $this->applyWith($model, $with);

        return $model;
    }
}

```

Figure 10-Clasa Repository

În figura 10, se pot observa funcțiile din clasa abstractă “Repository”. Acestea facilitează obținerea unei colecții de date doar din parsarea cererii din interfață. Spre exemplu, din interfață pot fi cerute toate orașele, sortate în ordine alfabetică după nume.

### 2.2.3. Nivelul de gestiune a cererilor

Ultimul nivel al aplicației, este reprezentat dintr-o multitudine de clase, intitulate Controllere. Acestea au rolul de a prelua cererea de la interfață și de a apela Repository-ul aferent pentru a obține datele necesare, iar mai apoi, acestea întorc un raspuns de tip JSON interfeței.

Fiecare controller extinde la rândul lui o clasă generică intitulată ApiController. Aceasta are rolul de a oferi un set de funcții ajutătoare în formarea unui răspuns potrivit de tip JSON pentru interfață.

```
<?php namespace App\Http\Controllers;

class ApiController extends Controller {
    protected $fractalManager;

    public function __construct(Manager $fractalManager) {
        $this->fractalManager = $fractalManager;
    }

    public function collection($data, TransformerAbstract $transformer) {
        if(is_object($data) && $data instanceof LengthAwarePaginator) {
            return $this->pagedCollection($data, [
                'current' => $data->currentPage(),
                'totalItems' => $data->total(),
                'pagination' => [
                    'size' => 3
                ],
                'limit' => request('limit')
            ], $transformer);
        }

        return $this->fractalManager->respondCollection($data, $transformer,
Response::HTTP_OK);
    }

    public function item($data, TransformerAbstract $transformer) {
        if(!$data) {
            throw new NotFoundException('Entity not found');
        }

        return $this->fractalManager->respondItem($data, $transformer,
Response::HTTP_OK);
    }

    public function rawErrors($errors, $code) {
        return response()->json([
            "errors" => $errors,
            "data" => null,
            "status" => $code,
            "statusText" => Response::$statusTexts[$code]
        ], $code);
    }
}
```

```

    }
    public function rawItem($data, $code = 200) {
        return response()->json([
            "errors" => null,
            "data" => $data,
            "status" => $code,
            "statusText" => Response::$statusTexts[$code]
        ], $code);
    }

    public function error($errorMessage) {
        return $this->rawErrors([
            "general" => [
                $errorMessage
            ]
        ], Response::HTTP_UNPROCESSABLE_ENTITY);
    }

    public function unauthorized($errorMessage) {
        return $this->rawErrors([
            "general" => [
                $errorMessage
            ]
        ], Response::HTTP_UNAUTHORIZED);
    }
}

```

Figure12 -Structura ApiController

După cum se poate observa în figura 12, în interiorul clasei ApiController se află o serie de funcții predefinite pentru a ușura crearea unui răspuns standard pentru interfață. Astfel, mereu se va păstra o consistență în răspunsul oferit de aplicația server.

```

<?php
namespace App\Http\Controllers;

class CityController extends ApiController {
    public function index(Request $request, CityRepository $repository) {
        return $this->collection($repository->getCities($request), new
CityTransformer());
    }
}

```

Figure 13-Exemplu de funcție dintr-un controller propriu-zis

În figura 13 avem ca și exemplu o funcție dintr-un controller. Acestea îi sunt oferite ca și parametru, un request de unde se extrag datele de intrare și un repository, folosit pentru a genera un răspuns.

## 2.3 Prelucrarea datelor pentru răspuns

Pentru a oferi un răspuns cât mai standard la toate cererile http primite de la interfață, am folosit un pachet extern “league/fractal”. Acesta reprezintă un strat adițional în aplicație, care permite un control mult mai amănunțit asupra răspunsului oferit la o cerere http.

```
<?php namespace App\Http\Transformers;

use League\Fractal\TransformerAbstract;

class AirportTransformer extends TransformerAbstract {
    protected $availableIncludes = [];
    public function transform($data) {
        return [
            "id" => $data["id"],
            "code" => $data["code"],
            "latitude" => $data["latitude"],
            "longitude" => $data["longitude"],
            "name" => $data["name"],
            "city" => $data["city"],
            "state" => $data["state"],
            "country" => $data["country"],
            "woeid" => $data["woeid"],
            "timezone" => $data["timezone"],
            "phone" => $data["phone"],
            "type" => $data["type"],
            "email" => $data["email"],
            "url" => $data["url"],
            "runway_length" => $data["runway_length"],
            "elev" => $data["elev"],
            "icao" => $data["icao"],
            "direct_flights" => $data["direct_flights"],
            "carriers" => $data["carriers"],
        ];
    }
}
```

Figure 14 -Exemplu de prelucrare a datelor înainte de a fi trimise

## Capitolul 3: Aplicația client

### 3.1 Tehnologii folosite

Pentru realizarea interfeței am ales AngularJs. Acesta este un framework structural pentru aplicațiile dinamice, acesta având la bază HTML pentru templetizare. Printre avantajele framework-ului, este acela că datele pot fi conectate între ele, suporta dependency injection și permite structurarea codului în servicii și directive.

Aplicația, este o “aplicație pe o singura pagină” (eng. Single Page Application), asta însemnând că prima dată când este accesată, se încarcă codul Javascript, localizat în antetul paginii o singură dată, iar pe parcurs ce paginile se schimbă, nu se face altceva decât să se încarce noi module javascript, însă fără a reîncărca complet aplicația. Acest lucru oferă un plus de performanță, întrucât, încărcarea completă a unei aplicații are un timp de încărcare de ordinul secundelor.

Totodată s-a dorit ca partea de client să fie compatibilă cu orice tip de rezoluție, deoarece în marea majoritate a timpului, consumatorii pot accesa aplicația de pe dispozitivul mobil, acesta având o rezoluție mult mai mică decât monitorul unui calculator.

### 3.2 Arhitectura aplicației client

Arhitectura aplicației este împărțită pe mai multe directoare. Fiecare pagină este formată din 4 fișiere:

- Fișier cu extensia pug, acesta este un template pentru Node.js ce permite să se scrie date, iar mai apoi le convertește în HTML. Unul din cele mai mari avantaje oferite de fișierele Pug, este faptul că se poate scrie un cod cât mai lizibil
- Un fișier index.js, unde este definit modulul de Angular

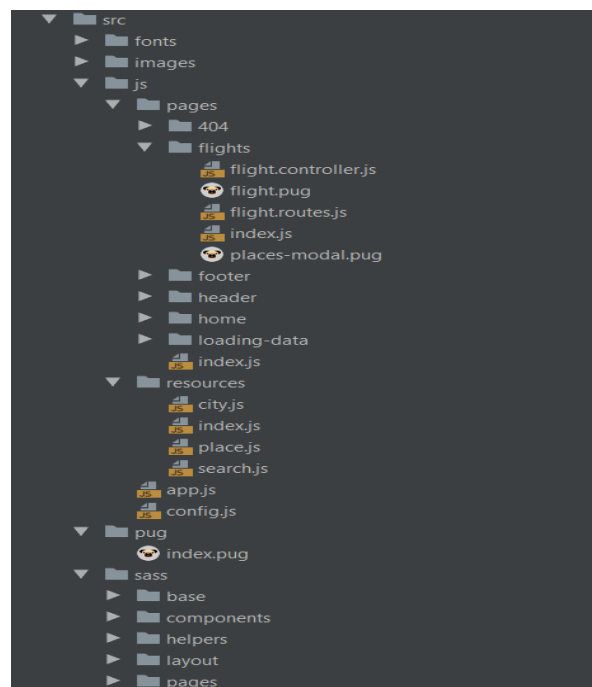


Figure 15 -Arhitectura aplicatiei client



- Un fișier\*.controller.js, care încapsulează logica din componentă. Acesta are rolul de a obține date, de a le procesa și de a le oferi mai departe fișierului Pug;
- Un fișier\*.routes.js, care definește ruta la care poate fi accesată pagina și face legătura propriu zisă dintre controller și fișierul Pug;

```
div.loading-div(ng-if="!$flightCtrl.pageLoaded")
  loading-data.center(message='Loading')
div(ng-if="$flightCtrl.pageLoaded")
  section#section-city-guides.probootstrap_section
    .container.ticket-container
      .flex-menu...
      #collapseFilter...
      .row(ng-repeat="flight in $flightCtrl.response.flights")
        .flight-container
          .flight-header...
          .flight-route...
          .flight-container...
#exampleModal...
```

Figure 16 Fisier Pug

```
import Config from '../..//config';

class FlightCtrl {
  constructor(Search, Place, City, $timeout, $state, moment, $scope, $stateParams, $uibModal) {...}
  submitSearch(model) {...}

  buy(link) {...}
  searchModal() {...}
}

/* @ngInject */
export default FlightCtrl;
```

Figure 17-Controller

```
class FlightRoutes {
  constructor($stateProvider) {
    $stateProvider.state('flight', {
      url: '/flights/{fromDate}/{toDate}/{from}/{to}/{adults}/{children}/{typeFlight}',
      //params: {model : null},
      template: require('../flight.pug'),
      controller: 'FlightCtrl',
      controllerAs: '$flightCtrl',
    });
  }

  /* @ngInject */
  static routesFactory($stateProvider) {
    FlightRoutes.instance = new FlightRoutes($stateProvider);
    return FlightRoutes.instance;
  }
}

export default FlightRoutes;
```

Figure 18-Fișier de legătură între controller și pug

```
import FlightCtrl from './flight.controller';
import FlightRoutes from './flight.routes';

let flightModule = angular.module('app.flight', []);

flightModule.config(FlightRoutes.routesFactory);
flightModule.controller('FlightCtrl', FlightCtrl);

export default flightModule;
```

Figure 19-Fișier în care este definit modulul de angular

Totodată, aplicația are un fișier de tip nucleu, în care sunt încărcate toate librăriile și modulele.

```
import 'bootstrap';

import 'angular';
import "angular-animate";
import '@uirouter/angularjs';
import '...../bower_components/ng-scrollable/src/ng-scrollable';
import '...../dist/js/rxslider';
import '...../dist/js/rxslider.min.js';
import 'ui-select';
import 'angular-sanitize';
import 'angular-ui-bootstrap';
import "angular-moment";
import "moment";
import './resources';
import './pages';
import 'angularjs-slider';
import Config from './config';

angular.module('app', [
  'ui.router',
  'ui.bootstrap',
  'angularMoment',
  'app.resources',
  'app.pages',
  'ngAnimate',
  'ui.select',
  'ngScrollable',
  'rxModule',
  'ngSanitize'
]);

angular.module('app').config(/* @ngInject */ ($locationProvider, $urlRouterProvider, $qProvider) => {
  // silence unhandled rejections
  $qProvider.errorOnUnhandledRejections(false);

  // remove ! from url
  $locationProvider.hashPrefix('');

  // default route
  $urlRouterProvider.otherwise(function() {
    return '/';
  });
});

angular.module('app').value('AppConfig', Config);
```

Figure 20 - fișier de tip nucleu

## Capitolul 4: Modul de funcționare al aplicației

Utilizatorul intră pe pagina aplicației client, unde se încarcă aplicația realizată în AngularJs. Acesta vede o pagină de întâmpinare, în care poate să își aleagă cu un număr redus de filtre, un zbor.

După ce se completează formularul și se apasă butonul “Submit”, este apela o funcție din controller, care are ca rol, trimiterea datelor către o altă pagină de angularJs.

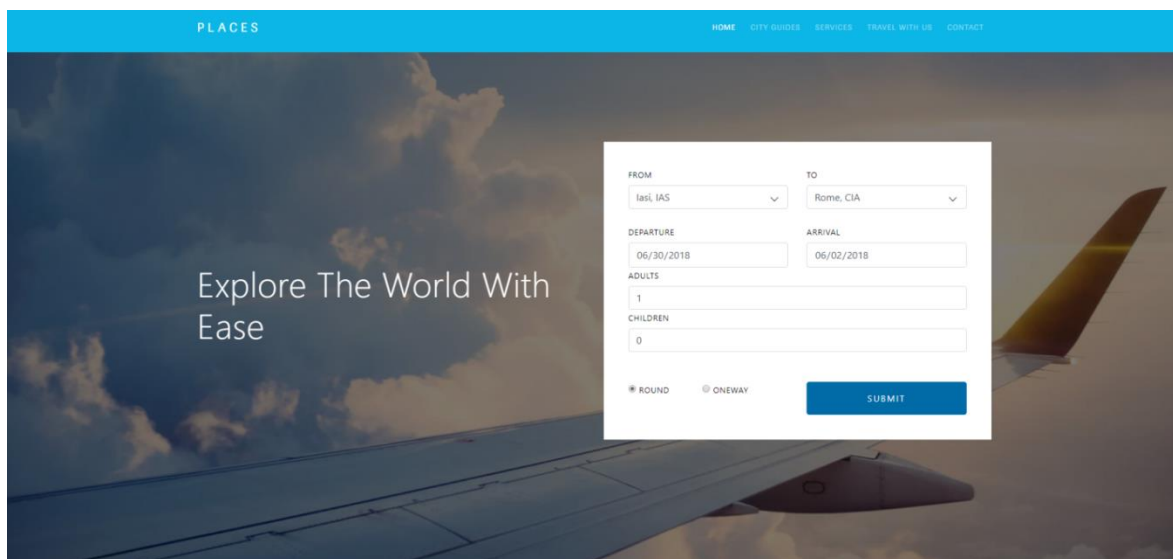


Figure 21-pagina încărcată inițial

```
class HomeCtrl {
  constructor(Search, City, $timeout, $state, moment, $scope) {...}
  submitSearch() {
    if(this.model.from && this.model.to && this.model.fromDate) {
      let sendObj = {
        from: this.model.from.split(", ")[1],
        to: this.model.to.split(", ")[1],
        fromDate: this.moment(this.model.fromDate).format('L'),
        toDate: this.model.typeFlight === 'round' ? this.moment(this.model.toDate).format('L') : null,
        adults: this.model.adults,
        children: this.model.children,
        typeFlight: this.model.typeFlight
      };
      this.$state.go('flight', sendObj);
    }
  }
  getData(searchVal) {
    return this.City.list({search: searchVal});
  }
}
```

Figure 21-funcția din controller care redirectează spre o altă pagină din aplicație

După ce are loc redirectionarea, utilizatorul este dus pe o pagină în care în mod asincron îi sunt căutate zborurile în conformitate cu cerințele sale. În timp ce acesta așteaptă încărcarea paginii, pentru a nu lăsa aplicația să se încarce parțial, am ales să folosesc un preloader.



Figure 22 - animație ce este randată în momentul în care pagina se încarcă în spate, în mod asincron

În controller, aplicația face un request la server pentru a obține zborurile conforme cu dorințele consumatorului, iar după ce primește un răspuns, în cazul în care există date, le randează în pagină, iar în cazul în care nu există date, este încărcată o pagină în care clientului îi este semnalat faptul că nu s-a găsit nici un zbor pentru preferințele acestuia.

Pentru a obține date legate de zboruri, aplicația folosește un API extern, oferit de Kiwi. Astfel, serverul face un request `Https` la url-ul `“https://api.skypicker.com/flights”`.

```
class FlightCtrl {
  constructor(Search, Place, City, $timeout, $state, moment, $scope, $stateParams, $uibModal) {...}
  submitSearch(model) {
    this.Search.search(model).$promise.then(res => {
      this.response = res.data;
      if(this.response.flights.length === 0) {
        this.$state.go('404');
      }
      this.height = this.response.maxRoutes * 262;
      this.pageLoaded = true;
    });
  }

  buy(link) {...}
  searchModal() {...}
}

// @ngInject */
export default FlightCtrl;
```

Figure 23- Locul unde are loc cererea http către server

```

class FlightRepository {
    private $appUrl = 'https://api.skypicker.com/flights';
    private $client;
    private $request;
    public function __construct() {
        $this->client = new \GuzzleHttp\Client();
    }
    public function getFlights($params) {
        $request = new \GuzzleHttp\Psr7\Request( method: 'GET', $this->appUrl);
        $promise = $this->client->sendAsync($request, [
            'query' => $params
        ]->then(function ($response) use (&$data) {
            $data = json_decode($response->getBody()->getContents());
        });
        $promise->wait();
        return $data;
    }
}

```

Figura 24 – Request-ul propriu zis la API

În figura 24, se poate observa cum în repository are loc cererea de date de la API-ul oferit de Kiwi. Requestul este realizat asincron și astfel, utilizand “\$promise->wait()”, serverul așteaptă să vină rezultatul, iar mai apoi își continuă execuția. Acest lucru a fost necesar deoarece căutarea de zboruri este realizată cu foarte multe filtrări, fapt ce poate îngreuna sosirea unui răspuns.

Filtrele puse la dispoziție sunt:

- data de plecare;
- data de sosire;
- orașul de plecare;
- orașul de sosire;
- numărul de pasageri;
- zbor direct sau cu escale;
- numărul maxim de escale;
- durata maximă a călătoriei în ore;
- tipul de zbor: dus sau dus-întors;
- prețul maxim

După ce datele au fost aduse, acestea sunt într-o formă brută și trebuie prelucrate, pentru a putea fi trimise mai departe interfeței.

```

public function formatFlights($data) {
    $flightsArray = [];
    $maxRoutes = 0;
    foreach ($data as $item) {
        $routeArray = [];
        $goingRoutes = [];
        $returnRoutes = [];
        $goingFinished = false;
        $maxRoutes = count($item->route) > $maxRoutes ? count($item->route) : $maxRoutes;
        foreach ($item->route as $index => $route) {
            $generatedArray = [
                //...
                'airline' => Airline::where('iata', $route->airline)->first(),
                'from_city' => City::where('name', $route->cityFrom)->first(),
                'to_city' => City::where('name', $route->cityTo)->first(),
                'from_airport' => Airport::where('code', $route->flyFrom)->first(),
                'to_airport' => Airport::where('code', $route->flyTo)->first(),
                'fly_duration' => date('format: "H:i"', ($route->aTimeUTC- $route->dTimeUTC)),
                'departure_date' => date('format: "H:i"', ($route->aTimeUTC)),
                'arrival_date' => date('format: "H:i"', ($route->aTimeUTC)),
                'flight_date' => date('format: "Y-m-d"', ($route->dTimeUTC))
                //...
            ];
            if(!$goingFinished) {
                $goingRoutes[] = $generatedArray;
            } else {
                $returnRoutes[] = $generatedArray;
            }
            if($route->flyTo === $item->flyTo) {
                $goingFinished = true;
            }
        }
        $flight = [...];
        $flightsArray[] = $flight;
    }
    $this->maxRoutes = $maxRoutes;
    return $flightsArray;
}

```

Figure 25- Procesarea datelor primite

În figura ce se referă la procesarea datelor primite, se poate observa cum pentru fiecare oraș, aeroport și companie aeriană, sunt aduse din baza de date informații adiționale. Aceste lucruri sunt necesare pentru a putea obține pozele companiilor aeriene și pentru a putea obține interese pentru aeroporturi, pe bază de cuvinte cheie.

O altă parte importantă a aplicației, este oferirea de obiective în jurul unei

coordonate geografice, pe bază de cuvinte cheie. Pentru a realiza această căutare, am folosit două API-uri oferite de Google și anume Google Places și Google Places Photos.

Google Places este folosit pentru a obține diferite obiective pe baza unor cuvinte cheie, în jurul unui punct geografic (latitudine și longitudine).

```

namespace App\Repositories;

class PlaceRepository {
    private $appUrl = 'https://maps.googleapis.com/maps/api/place/nearbysearch/json';
    private $client;
    private $request;

    public function __construct() {
        $this->client = new \GuzzleHttp\Client();
    }

    public function search($params) {
        $request = new \GuzzleHttp\Psr7\Request('GET', $this->appUrl);
        $promise = $this->client->sendAsync($request, [
            'query' => $params
        ])->then(function ($response) use (&$data) {
            $data = json_decode($response->getBody()->getContents());
        });
        $promise->wait();
        $data->initialLat = $params['lat'];
        $data->initialLng = $params['lng'];
        return $data;
    }
}

```

Figure 26 – requestul la Google Places

După ce datele au fost obținute, acestea sunt procesate pentru a putea fi trimise spre interfață.

```
use League\Fractal\TransformerAbstract;
use RFlaversini\Distance;

class PlaceTransformer extends TransformerAbstract {
    protected $availableIncludes = [];

    public function transform($data) {
        return $this->transformData($data);
    }

    public function transformData($data) {
        $returnData = [];
        foreach ($data->results as $result) {
            $temporaryArray = [
                'name' => $result->name,
                'photo' => array_key_exists('key: "photos"', $result) !== false ? $this->getPhotoUrl($result->photos[0]->photo_reference) : null,
                'reference' => $result->reference,
                'types' => $result->types,
                'distance' => Distance::toKilometers($data->initialLat, $data->initialLat, $result->geometry->location->lat, $result->geometry->location->lat)
            ];
            $returnData[] = $temporaryArray;
        }
        return $returnData;
    }

    public function getPhotoUrl($reference) {
        $key = env('key: "GOOGLE_KEY"');
        return "https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=$reference&key=$key";
    }
}
```

Figure 27– Procesarea datelor primite de la Google Places

În figura de mai sus, se poate observa cum este generat url-ul pentru Google Places Photos, pentru a obține o imagine cu obiectivul respectiv. Totodată este folosită formula lui Haversine pentru a calcula distanța dintre 2 puncte geografice.

$$d = 2r \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Figure 28- Haversine formulă - <https://github.com/DaniilSydorenko/haversine-geolocation>

Pentru a folosi formula lui Haversine în aplicație am folosit un pachet extern open source “rafaelfragoso/haversini-formula”. Acesta oferă un set de funcții ce primesc ca parametri coordonate geografice și întorc distanța în diferite unități de măsură.

După ce datele sunt returnate la interfață, clientului îi sunt afișate în pagină o listă cu toate zborurile găsite. Acesta poate să își aleagă zborul care se pliază cel mai bine pe nevoile lui, însă dacă între două zboruri este o escală de durată mai lungă, acesta poate să caute obiective de interes din jurul aeroportului, pentru a alege diferite acțiuni pentru a petrece timpul.






IAS - CIA			Iași - Rome
IAS - OTP	Departure 04:00 Duration 00:50	Arrival 04:50 Date 2018-06-30	
OTP - CIA	Departure 03:05 Duration 02:10	Arrival 05:15 Date 2018-07-01	
CIA - IAS			Rome - Iași
CIA - OTP	Departure 20:35 Duration 02:00	Arrival 22:35 Date 2018-07-02	
OTP - IAS	Departure 05:35 Duration 00:50	Arrival 06:25 Date 2018-07-03	
Duration 25h 15m		Price 258 €	Action 

Figure 29-Modul în care un bilet Iași-Roma, Roma-Iași este afișat în pagină







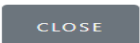
Places around IAS			×
<input type="text" value="coffe shop"/>			
Photo	Name	Distance	
	Jassyro Coffee Shop	2 km	
	Coffee Spot	2 km	
	7am Coffee To Go	1 km	
	Teo's Cafe UMF	2 km	
	Teo's Café Lulius Mall	4 km	
	Status Coffee	3 km	
			

Figure 30-Cautare de interese după cuvinte cheie

Dacă utilizatorul s-a hotărât asupra unui bilet care îi satisface nevoile, acesta poate să apese butonul de search pentru a rezerva biletul. În momentul în care butonul este apasat, are loc o redirectionare către site-ul [www.kiwi.com](http://www.kiwi.com), de unde poate fi făcută achiziția propriu-zisă.



## Concluzie

În concluzie, putem spune că **Places** este o aplicație de booking viabilă care este axată pe nevoile consumatorilor și pe oferirea unei experiențe cât mai plăcute în interfață.

Aplicația are ca scop rezolvarea problemelor regăsite cel mai adesea pe platformele de booking, acestea fiind:

- neoferirea de alternative de petrece a timpului pentru zborurile cu escale de durată mare;
- interfața și meniurile complexe, care cel mai adesea induc în eroare consumatorii;

Acest lucru se realizează prin faptul că interfața aplicației are un design simplu, orientat spre a fi foarte ușor de folosit tuturor categoriilor de oameni. Totodată, aplicația își propune să aducă în cadrul său, performanță pe partea de procesare și încărcare de date prin tehnologiile folosite în dezvoltare. Sistemul de filtrare a zborurilor este unul foarte simplu, ce îi oferă utilizatorului posibilitatea de a alege zborul care se pliază cel mai bine pe nevoile acestuia.

Atât aplicația de server cât și cea destinată clientului, a fost scrisă în așa fel încât să se poată aduce cu ușurință module noi fără a le afecta deja pe cele existente. Acest lucru este posibil datorită principiilor de programare aplicate în dezvoltarea produsului, cum ar fi aplicarea șablonului Repository pe backend pentru a avea un cuplaj mic.

Îmbunătățirile ce ar putea fi aduse aplicației sunt:

- Funcționalitatea de “navigare infinită în pagină” (eng. Infinite Scroll), în pagina cu biletele de zbor;
- Adăugarea de noi API-uri externe care să ofere zboruri;
- Integrarea Google maps, pentru a oferi o perspectivă mai bună asupra traseului, ce va fi urmat de o anumită cursă;
- Implementarea unei funcționalități de tip bulletin informative, în care utilizatorii primesc pe mail periodic oferte cu diferite zboruri;
- Implementarea unui sistem de login și memorare a zborurilor deja cumpărate, pentru a putea oferi o perspectivă asupra achizițiilor realizate;
- Posibilitatea de a realiza achiziționarea zborurilor, direct din aplicație;
- Realizarea unui sistem care să ofere clientului posibilitatea de a selecta anumite preferințe de zbor și un anumit buget, iar mai apoi prin intermediul buletinului informativ, să anunțe clientul dacă au fost găsite zboruri care să se încadreze în bugetul lui;

## Bibliografie

1. Php - <http://php.net/manual/ro/index.php>
2. Laravel - <https://laravel.com/>
3. Repository Pattern - <https://deviq.com/repository-pattern/>
4. Repository Pattern Laravel - <https://medium.com/@jsdecena/refactor-the-simple-tdd-in-laravel-a92dd48f2cdd>
5. Angularjs - <https://angularjs.org/>
6. Şablon interfață – <https://probootstrap.com/license>
7. API zboruri - <https://skypickerpublicapi.docs.apiary.io/#>
8. API Google Places - <https://cloud.google.com/maps-platform/places/>
9. API Google Places Photos - <https://developers.google.com/places/web-service/photos>
10. API Amadeus - <https://sandbox.amadeus.com/travel-innovation-sandbox/apis/get/points-of-interest/yapq-search-text>
11. Airline Logos - <https://support.travelpayouts.com/hc/en-us/articles/203956073-Airline-logos>
12. World Cities data - <https://github.com/tapvt/php-world-cities-array>
13. Coduri firme aviație - [https://en.wikipedia.org/wiki/List\\_of\\_airline\\_codes](https://en.wikipedia.org/wiki/List_of_airline_codes)
14. Lista aeroporturi - <https://gist.github.com/tdreyno/4278655>
15. Bootstrap - <https://getbootstrap.com/>
16. Agile - <https://www.versionone.com/agile-101/>
17. Kanban - <https://ro.atlassian.com/agile/kanban>
18. Imagine Agile - <http://pakar.co.id>
19. Imagine Kanban - <http://leanmanufacturingtools.org>
20. Google Maps - <https://www.google.com/maps>
21. Design bilet avion - <https://codepen.io/victorjanin/pen/oGYvjK>