**UNIVERSITATEA DIN BUCUREȘTI**



**FACULTATEA**

**DE**

**MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

**APLICAȚIE WEB PENTRU PROMOVAREA ȘI ÎNCHIRIEREA HOTELURILOR DIN ROMÂNIA**

**Absolvent**

**CIOROBA ROBERT-MIHAI**

**Coordonator ştiinţific**

**Lect.dr. Alexandru Amărioarei**

**București, iulie 2020**

Rezumatul

Domeniul medical a reprezentat dintotdeauna o ramură de interes pentru întreaga umanitate. Fie că este vorba despre o simplă răceală sau de tratarea celor mai grave afecțiuni, fiecare dintre noi o sa se loveasca mai devreme sau mai târziu de această industrie.

Pentru că trăim în secolul vitezei iar fiecare secundă este importantă, cele mai multe domenii de activitate încearcă să migreze către așa-numita digitalizare. Acesta este și scopul proiectului Regele Marius, care încearcă să combine medicina și domeniul IT. Astfel, fiecare pacient își poate face o programare la medicul preferat din comfortul propriei locuințe, își poate urmări evoluția anumitor factori de risc in mod gratuit pentru ca in final sa poată primi buletinele de analize cu o simplă atingere a unui ecran.

În același timp fiecare medic are sintetizate mult mai lizibil toate programările, informațiile adiționale despre fiecare pacient plus detalii despre ceilalți colegi de breaslă.

Regele Marius este o aplicație web realizată in ASP .NET, care își propune să fluidizeze conexiunea medic-pacient, reducând semnificativ timpul de așteptare la interminabilele cozi, dar totodată ajutând și secretarele cabinetului să gestioneze totul mult mai eficient.

Summary

The medical domain has always represented a branch of interest for the whole humanity. Whether it's about a simple cold or the treatment of the most serious diseases, each one of us will sooner or later face this industry.

Because we live in the speed century and every second is important, most areas of activity try to migrate to the so-called digitization. This is also the aim of the project Regele Marius, which tries to combine medicine and IT. Each patient can make an appointment with their favorite doctor form the comfort of their own home, they can monitor the evolution of certain risk factors for free so that in the end they can receive the analysis bulletins with a simple touch of a screen.

At the same time, each doctor has a much more readable summary of all appointments, the additional information about each patient, plus details about the other colleagues.

Regel Marius is a web application made in ASP .NET, which aims to streamline the doctor-patient connection, significantly reducing the waiting time at the endless queues, but also helping cabinet secretaries manage everything more efficiently.

Cuprins

[Capitolul 1. INTRODUCERE 6](#_Toc44434276)

[1.1 Contextul general al proiectului 6](#_Toc44434277)

[1.2 Motivația lucrării 6](#_Toc44434278)

[1.3 Structura lucrării 7](#_Toc44434279)

[Capitolul 2. SCOPUL ȘI OBIECTIVELE PROIECTULUI 9](#_Toc44434281)

[2.1 Scopul proiectului 9](#_Toc44434282)

[2.2 Obiectivele proiectului 9](#_Toc44434283)

[Capitolul 3. ANALIZĂ DE PIAȚĂ ȘI STUDIU BIBLIOGRAFIC 11](#_Toc44434284)

[3.1 Domeniul hotelier în România 11](#_Toc44434285)

[3.2 Aplicații pentru închirieri in domeniul hotelier 11](#_Toc44434286)

[3.2.1 Booking 11](#_Toc44434287)

[3.2.2 Vola 12](#_Toc44434288)

[3.2.3 Airbnb 13](#_Toc44434289)

[Capitolul 4: DOCUMENTARE ȘI FUNDAMENTARE TEORETICĂ 14](#_Toc44434290)

[4.1 Cerințele sistemului 14](#_Toc44434291)

[4.1.1 Cerințele funcționale ale aplicației Travellor 14](#_Toc44434292)

[4.1.2 Cerințele non-funcționale 15](#_Toc44434293)

[4.2 Ruby 16](#_Toc44434294)

[4.3 Rails 16](#_Toc44434295)

[4.4 Active Storage 18](#_Toc44434296)

[4.5 Bcrypt 21](#_Toc44434297)

[4.6 Gemfile și Gemfile.lock 22](#_Toc44434298)

[4.7 Jquery și Ajax 23](#_Toc44434299)

[4.8 Baza de date 24](#_Toc44434300)

[4.8.1 Baze de date relaționale 24](#_Toc44434301)

[4.8.2 Baze de date NoSQL 25](#_Toc44434302)

[4.8.3 PostgreSQL 25](#_Toc44434303)

[Capitolul 5. PROIECTAREA APLICAȚIEI ȘI DETALII DE IMPLEMENTARE 27](#_Toc44434304)

[5.1 Stilul arhitectural al sistemului 27](#_Toc44434305)

[5.2 Nivele aplicației 29](#_Toc44434306)

[5.3 REST 30](#_Toc44434309)

[5.4 Entitățile aplicației 31](#_Toc44434310)

[5.5 Controllerele aplicației 33](#_Toc44434311)

[Capitolul 6. TESTARE ȘI VALIDARE 37](#_Toc44434312)

[6.1 Testarea unitară a modelelor 37](#_Toc44434313)

[6.2 Testarea unitară a controllerelor 39](#_Toc44434314)

[Capitolul 7. CONCLUZII 42](#_Toc44434315)

[8.1 Realizarea obiectivelor 42](#_Toc44434316)

[8.2 Dezvoltări ulterioare 42](#_Toc44434317)

# 

# Capitolul 1. INTRODUCERE

## 1.1 Contextul general al proiectului

Bolile au existat pe pământ chiar înaintea apariției oamenilor, animalele fiind primele care s-au lovit de acest obstacol. Pentru a putea supraviețui, ele au fost nevoite să găsească soluții pentru combaterea sau măcar pentru ameliorarea problemelor. Fie că este vorba despre saliva mamiferelor, care are rol dezinfectant sau despre consumatul ierbii în cazul carnivorelor pentru a preveni indigestiile, ființele au încercat mereu să se adapteze condițiilor apărute. Aceste obiceiuri sunt unele instictive, fiziologice. Cu obiceiuri din această categorie au început și primii oameni să se trateze. Medicina umană modernă este de departe una mult evoluată, ajungând să trateze anual milioane de persoane care cu mai puțin de un secol în urmă aveau speranța de viață aproape de zero.

Cu toate acestea, conform spuselor pacienților români, putem trage concluzia că sistemul este unul greoi in ceea ce privește programările și promovarea deși s-au făcut progrese imense atât din punct de vedere al tehnologiei si social media. Există puține spitale care au un site implementat, dar chiar și aceastea au o interfață greoaie, cu informații multe și inutile iar functionalitățile sunt inexistente.

De la această premisă a plecat și conceptul proiectului Regele Marius. O platformă prin care orice persoană ce a avut minime tangențe cu internetul își poate face o programare in mod automat pentru unul din seturile de analize puse la dispozitie de clinică, totul in mai puțin de un minut.

## 1.2 Motivația lucrării

Motivația lucrării este reprezentată de acea zicală veche, din batrâni, care spune că ,,dacă ai sănătate ai de toate”, așadar pe primul loc în această viață ar trebui să punem starea noastră de sănătate. Chiar dacă majoritatea persoanelor sunt conștiente de acest lucru, un studiu arată că 40% dintre români nu au mers în ultimul an niciodată la medic, mai jos de atât fiind situați doar grecii si bulgarii. Argumentele principale aduse pentru acest rezultat îngrijorator sunt: lipsa medicilor (în special la sate), lipsa încrederii în sistemul medical, frica și dezinformarea.

La prima vedere, niciunul dintre acești factori nu poate fi remediat prin intermediul unei simple aplicații, singura soluție viabilă fiind educația pe această tema încă din sânul familiei. Totuși, ultimele 3 din aceste 4 probleme pot fi soluționate cu ajutorul aplicației Regele Marius. Lipsa încrederii este într-o foarte puternică legătură cu dezinformarea. Tot mai multe surse (tv, social media) ne prezintă cazuri scoase din context, chiar șocante, toate acestea doar pentru un rating crescut fără a lua însă în calcul faptul că așa pot ,,otrăvii” gândirea si percepția corectă a consumatorului. O soluție sănătoasă, care sa aibă urmări pe termen lung ar fi aceea a transparenței, pentru ca fiecare utilizator să știe exact ce înseamnă o anumita analiză, de exemplu.

Un alt factor este frica. Frica apare de cele mai multe ori din teama de necunoscut. La fel ca și primele doua puncte, această problema se poate remedia prin transparență, prin informarea corectă a utilizatorului, astfel acesta știind la ce să se aștepte atunci când calcă într-un cabinet.

Ultimul beneficiu, dar și cel mai important, este acela al câștigării timpului, care in zilele noastre a ajuns cea mai prețioasă resursă. Astfel, pacientul nu este nevoit să-și creeze un cont pentru a putea utiliza platforma, proces care este vazut drept un chin de cei mai mulți utilizatori. Singurele persoane care au nevoie de un cont de utilizator sunt medicii si secretara, dar chiar și pentru ei aplicația aduce un plus de fluiditate.

## 1.3 Structura lucrării

# Urmeaza lista capitolelor proiectului, adăugând pentru fiecare dintre acestea și o scurtă descriere.

Capitolul 1, **Introducere**, oferă informații cu referire la punctul de plecare al proiectului, modul cum este privit sistemul medical din România de compatrioții noștrii plus o scurtă descriere a motivației alegerii temei.

Capitolul 2, **Scopul și obiectivele proiectului**, oferă o imagine de ansamblu asupra nevoilor pe care încearcă să le satisfacă aplicația propusă și descrie principalele funcționalități puse la dispoziția clienților și a managerilor de hotel.

Capitolul 3, **Analiză de piată și studiu bibliografic**, prezintă o analiză asupra ramurii hoteliere a turismului românesc și o analiză de piață a celor mai mari trei aplicații similare.

Capitolul 4, **Documentare și fundamentare teoretică**, descrie cerințele sistemului, tehnologiile și serviciile necesare pentru crearea și funcționarea aplicației și o analiză comparativă a tipurilor de baze de date existente.

Capitolul 5, **Proiectarea aplicației și detalii de implementare**, aduce informații despre arhitectura folosită, relațiile dintre componentele aplicației și modul de transmitere a datelor, entitățile aplicației și câteva dintre paginile web.

Capitolul 6, **Testare și validare**, prezintă modul în care este realizată testarea automată, prin intermediul testelor unitare ale modelelor și ale controllerelor aplicației.

Capitolul 7, **Concluzii**, reflectă analiza finală asupra aplicației și modului în care au fost îndeplinite obiectivele acesteia, cât și unele idei de dezvoltare ulterioară.

# Capitolul 2. SCOPUL ȘI OBIECTIVELE PROIECTULUI

## 2.1 Scopul proiectului

Scopul de bază este reprezentat tocmai de subiectul din motivația lucrării, anume acela de a permite oamenilor să aibă acces la informații reale și transparente, cat mai rapid și ușor posibil, reducând reticența oamenilor în ceea ce privește sistemul medical de la noi din țară. Pe cealaltă parte, sistemul vine și în ajutorul persoanelor care gestioneaza toți acești pacienți.

Aplicația pune la dispoziție pentru cele trei tipuri de utilizatori (pacient – care nu este nevoit să-și creeze un cont, secretară si medic) o interfață bazată pe simplitate si funcționalitate, reușind să combine toate nevoile utilizatorilor într-un mod cât mai ușor de folosit.

## 2.2 Obiectivele proiectului

Pentru a atinge scopul precizat mai sus, platforma a fost proiectată urmărind niște puncte binestabilite, fiecare funcționalitate fiind gândită pentru a se mula cât mai bine pe interesele si nevoile fiecaruia dintre cele trei tipuri de utilizatori: pacient, medic, secretară.

Funcționalitățile oferite pacienților (toate funcționalitățile sunt disponibile fara creearea unui cont in prealabil):

* Vizionarea paginii de prezentare a clinicii Regele Marius
* Accesul la o pagină de articole legate de sănătate gestionate de secretara clinicii
* Accesul la un calculator care determină în funcție de anumiți parametrii indicele de masă corporală și procentul de grăsime corporală oferindu-i o mică viziune asupra stării lui generale de sănătate
* Posibilitatea de a face o programare la o anumită analiză, sistemul atribuindu-i automat un interval orar, o dată și un medic ce poate realiza această procedură

Funcționalități oferite medicilor (conturile de medici pot fi create doar de către secretara clinicii):

* Vizionarea programărilor atribuite de sistem
* Posibilitatea de a creea un raport de analize pentru programările proprii
* Acces la datele personale ale pacienților

Funcționalități oferite secretarei:

* Acces la adăugare/stergere medici noi, analize, specializări, slider, articole
* Posibilitate vizionare a tuturor programărilor
* Pentru programările finalizate posibilitate vizionare raport medical
* Acces la statistici (grafice) generate in funcție de rezultatele obținute de utilizatori in calculatorul de masa musculară (IBM) care oferă informații despre starea de sănătate a acestora.

# Capitolul 3. ANALIZĂ DE PIAȚĂ ȘI STUDIU BIBLIOGRAFIC

## 3.1 Domeniul medical în România

,,Circa o treime din doctorii care au studiat in România, au emigrat și își defășoară activitatea profesională în străinătate” este concluzia tristă la care a ajuns un studiu realizat recent la nivel mondial[2]. Astfel, țara noastră se clasează pe primul loc în ceea ce privește procentul de medici pierduți după absolvire. Avem suficienți medici buni, pe care nu știm însă să-i exploatăm și să-i păstram ,,acasă”, aceștia preferând să plece peste hotare considerând că acolo le sunt mai apreciate calitățile și eforturile.

În ultimii ani se pot observa mici îmbunătățiri în domeniul medical de stat, dar și schimbări majore în cel privat, unde începem să putem să luptăm cu clinicile de top din Occident. Așadar sunt semne clare că se dorește investirea în acest segment.

## 3.2 Aplicații pentru închirieri in domeniul hotelier

Mai jos am făcut un studiu de piată pe câteva dintre cele mai mari clinici din România, din punct de vedere al platformei online care gestioneaza legătura cu publicul larg. De menționat este faptul că Centrul Medical Unirea, care se află pe locul 1 în ceea ce privește cifra de afaceri nu are disponibilă o astfel de platformă.

### 3.2.1 Medlife

Medlife este a doua clinică privată ca și cifră de afaceri in anul 2019 și a 2-a în ceea ce privește profitul generat depășind următorul contracandidat cu doar 0,2 milione RON. Aceste locuri în clasamente au fost posibile datorită faptului că platforma nu gestioneaza o clinică, ci un lanț întreg de clinici medicale specializate pe foarte multe domenii.

Pagina de start încearcă să prezinte toate informațiile necesare într-un mod cât mai aerisit și ușor de înțeles. Astfel, primul lucru care iese în evidență este carousel-ul din centru, care are rol informativ cu privire la cele ma noi știri interne legate de medicină.

În partea de sus, sunt disponibile accesul către contul personal, dropdown-uri separate pentru: specialități, medici, sediile disponibile, câteva pachete pentru firmele care doresc să ofere asigurări de sănătate angajaților și un mic blog.



Fig 3.1 Pagina principală a site-ului Medlife.com

### 3.2.2 Sanador

Când vine vorba despre partea estetică, deși aceasta este una strict subiectivă, Sanador reușește să atragă atenția prin mai mult minimalism.

Ca și structură putem găsi foarte multe similarități cu pagina de la Medlife și aici fiind prezent carousel-ul cu noutați, si meniurile cu specializări, clinicile și medicii disponibili, acestea din urmă fiind mutate pe partea din stânga.

Logarea în contul de utilizator nu se poate face aici in mod direct, butonul de logare sau de înregistrare fiind ascuns în submeniul de programare.



Fig. 3.2 Pagina principală a site-ului Sanador.ro

### 3.2.3 ICFundeni

Unul dintre cele mai mari spitale din România este reprezentat de Institutul Clinic Fundeni. Comparativ cu primele două studii ce caz, acesta nu se mai află în mediul privat.

Deși cu o interfață extrem de încărcată și greoaie, pagina reușește să aducă pacientului informația de care are nevoie: conducere, centrele medicale, câteva informații despre vaccinare, datele de contact etc. Dacă este să vorbim despre funcționalități, la acest capitol site-ul reușește să dezamăgească total. Singura funcționalitate cu adevărat utilă, și anume aceea de programare online, este prezentă dar nefuncțională.

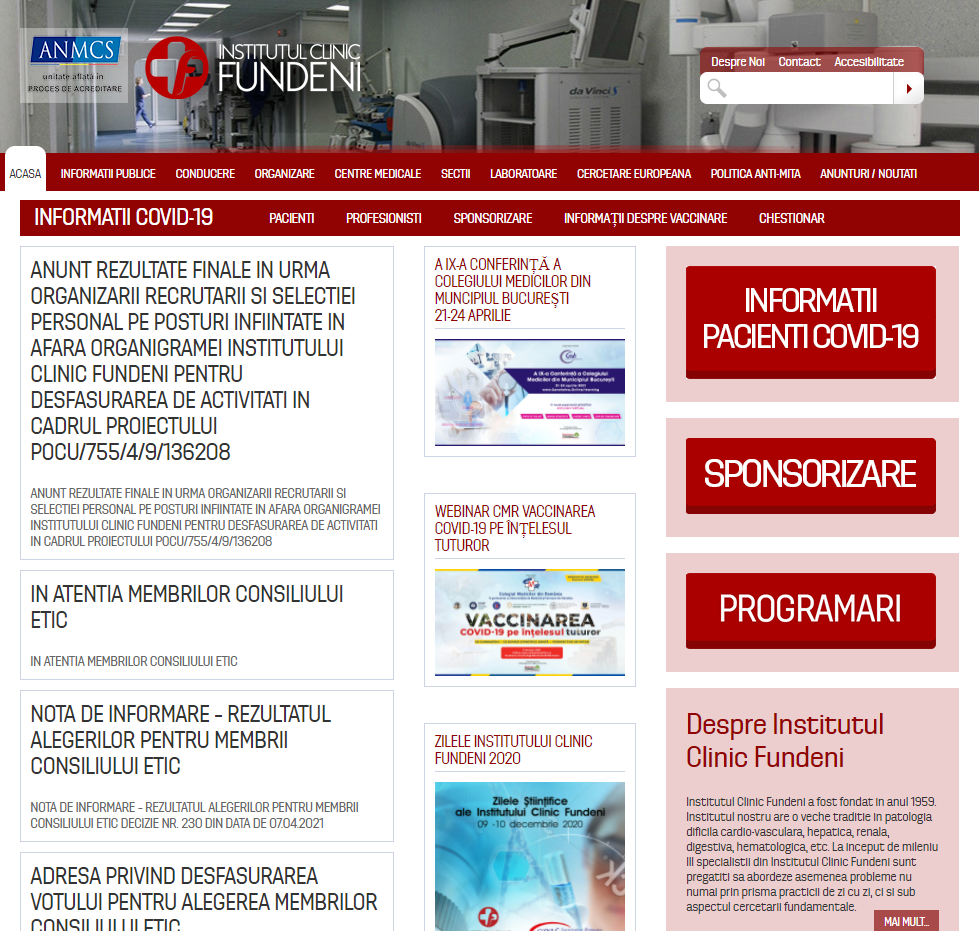


Fig. 3.3 Pagina principală a site-ului ICFundeni

# Capitolul 4: DOCUMENTARE ȘI FUNDAMENTARE TEORETICĂ

## 4.1 Cerințele sistemului

Unul dintre cei mai importanți factori pentru ca un proiect să se bucure de succes este reprezentat de claritatea cu care sunt definite cerințele și obiectivele acestuia. Acestea oferă detalii și limitări în ceea ce privește serviciile sistemului și sunt împărțite în două categorii:

* Cerințele funcționale: definesc în mod concret comportamentul sistemului în situații specifice de utilizare, descriu și determină ceea ce sistemul trebuie să fie capabil să ofere.

Exemplu: Ca utilizator Travellor, vreau să pot citi părerile altor clienți despre hoteluri, pentru a-mi forma o opinie despre serviciile oferite de acestea prin prisma celor care l-au vizitat deja.

* Cerințele non-funcționale, numite și atributele sistemului, descriu modul de funcționare al sistemului, fiabilitatea acestuia, cerințe de securitate, așteptări în ceea ce privește viteza medie de răspuns a aplicației, etc.

### 4.1.1 Cerințele funcționale ale aplicației Travellor

|  |  |
| --- | --- |
| **Descrierea cerinței funcționale** | **Beneficiar** |
| Vizualizarea paginii cu toate hotelurile | Toți utilizatorii |
| Vizualizare locație și review-uri hotel | Toți utilizatorii |
| Sortare, căutare și filtrarea hotelurilor | Toți utilizatorii |
| Creare de rezervări | Utilizator înregistrat |
| Editare de rezervări | Manager de hotel |
| Adăugare și editare de oferte | Manager de hotel |
| Adăugare și editare de hoteluri | Manager de hotel |
| Adăugare opinie despre hotel | Utilizator înregistrat |
| Vizualizare rezervări | Utilizator înregistrat, Manager de hotel |
| Înregistrare | Toți utilizatorii |
| Autentificare | Utilizator înregistrat |

Tabel 4.1 Cerințele funcționale ale aplicației

### 4.1.2 Cerințele non-funcționale

Cerințele non-funcționale vizează caracteristici precum: accesibilitatea datelor, persistența acestora, modalități de stocare a unor copii de rezervă ale datelor, metode de lansare și găzduire a aplicației, eficiență, timp de răspuns, etc.

Cele mai importante și populare dintre acestea se regăsesc, alături de o scurtă descriere, în lista de mai jos:

1. **Utilizabilitatea**, care caracterizează ușurința cu care un produs poate îndeplini și satisface nevoile utilizatorului. Aspectul și structura aplicației trebuie să asigure, indiferent de nivelul cunoștințelor tehnice ale clienților, modalități simple și eficiente de acces ale funcționalităților aplicației.

Exemplu: Utilizatorii trebuie să poată naviga în formularele aplicației folosind tasta Tab.

1. **Securitatea,** care asigură aplicația și datele acesteia împotriva atacurilor și nu le expune decât utilizatorilor autorizați în acest sens. În practică, cele mai multe aplicații folosesc diverse roluri și nivele de autorizare care asigură condifențialitatea, integritatea și disponibilitatea datelor.
2. **Scalabilitatea,** măsoară capacitatea de funcționare în parametri optimi odată cu creșterea semnificativă a numărului de utilizatori, cereri către server sau baza de date, resurse sau componente ale aplicației. De aceea, se recomandă ca modul de implementare să ia în considerare eventuale expansiuni ale aplicației și, deci, să fie eficient din acest punct de vedere.
3. **Performanța,** unul dintre cel mai importanți factori, măsoară calitatea aplicației, vizând în special timpul de răspuns al acesteia la cereri și numărul de utilizatori sau componente. Experiența în aplicație a persoanelor trebuie să nu fie afectată de performanță și trebuie să întrețină cu ușurință interacțiunile acestora cu aplicația. Se pot impune constrângeri în ceea ce privește timpul de răspuns al anumitor pagini sau secțiuni ale site-ului, precum: este necesară încărcarea tuturor hotelurilor în pagina principală în mai puțin de jumătate de secundă.

## 4.2 Ruby

Acest limbaj de programare a fost creat în anii ’90 de către Yukihiro Matsumoto, de origine japoneză și, deși este un limbaj de programare pur orientat pe obiecte (adică fiecare tip de date este reprezentat ca obiect), oferă suport și poate fi folosit pentru alte paradigme de programare precum programarea funcțională sau programarea imperativă. Yukihiro a preluat și combinat părțile sale favorite din limbaje precum Eiffel, Lisp sau Perl, pentru a crea un mod echilibrat de a programa, îmbinând programarea funcțională cu cea imperativă. Așa cum a expus public [2], creatorul a încercat să facă limbajul cât mai natural, nu doar simplu, și adaugă că, deși la prima vedere ar putea fi catalogat ca fiind simplu, Ruby este foarte complex la bază.

Printre principalele avantaje ale limbajului Ruby se numără stabilitatea, ușurința cu care poate fi înteles și învațat, faptul că suportă tehnica *mettaprogramming [3]* și comunitatea numeroasă.

## 4.3 Rails

Platforma de dezvoltare Ruby on Rails [4], apărută în anul 2004, a fost creată pentru a pune la dispoziția programatorilor de aplicații web un mod rapid de dezvoltare. Prin structura MVC [5] este întărită ideea că este necesară o delimitare clară între date, pagini și logica aplicației, de aceea proiectele rails nou create vin cu o împărțire a acestora pe foldere si fișiere, după cum este ilustrat în figura de mai jos (Fig. 4.1). Framework-ul dispune de o mulțime de pachete si librării, precum ActiveJobs, ActionMailer, ActionView, ActionPack etc.

Printre avantajele care vin la pachet cu Ruby on Rails, regăsim: prezența ActiveRecord [6], care facilitează comunicarea cu baza de date, existența a trei medii de lucru (dezvoltare, testare și producție), ușurința de a acoperi codul scris cu teste unitare sau de integrare, pornind de la cele generate automat de limbaj.

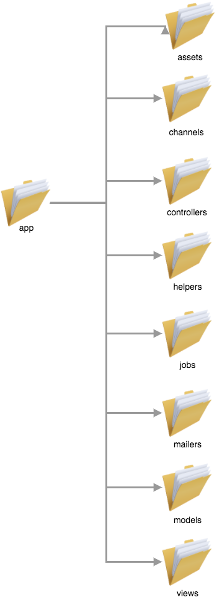


Fig. 4.1 Structura fișierelor într-un proiect Ruby on Rails

Printre principiile după care a fost creat Rails, se numără:

1. DRY (Do not repeat yourself – Nu te repeta), șablon software care susține ideea că fiecare lucru trebuie să aibă o singură sursă de adevăr. Cu alte cuvinte, fiecare stare trebuie să fie reprezentată o singură dată, într-un mod clar și autoritar. Respectând acest principiu și nescriind aceeași informație de mai multe ori, codul devine ușor de utilizat și de menținut odată cu trecerea timpului, eliminându-se redundanța. De asemenea, o mulțime de probleme și erori pot fi evitate folosind această metodă (eventual pot fi și corectate rapid, dacă acestea se află într-un singur loc în proiect).
2. Convenție înaintea configurației – prin această metodă se prezintă unul dintre cele mai bune moduri de dezvoltare a aplicațiilor web, unul prin care fișierele de configurații sunt eliminate dacă sunt respectate anumite principii și standarde în alegerea denumirilor folosite în codul sursă.

Un proiect Rails se creează cu comanda *rails new numeProiect*, la finalul căreia este creat un folder cu denumirea proiectului. Aplicația vine cu o serie de dependințe, care se află în fișierul Gemfile. Ele pot fi instalate cu ajutorul gem-ului Bundler, folosind comanda *bundle install*.

Rutele aplicației se găsesc în folderul *config*, fișierul *routes.rb.* Tot aici se setează ruta principală, corespunzătoare paginii „Acasă” a aplicației web și sunt mapate adresele web cu acțiunile potrivite din *controller-ul* țintă.

## 4.4 Active Storage

Imaginile hotelurilor sunt stocate prin intermediul tehnologiei ActiveStorage [7], ea permite simultan încărcarea de fișiere în servicii externe precum *Amazon S3, Microsoft Azure Storage* sau *Google Cloud Storage* și atribuirea acestora obiectelor Rails, reprezentate prin ActiveRecord. Mai mult, pentru mediile de dezvoltare și testare, ActiveStorage pune la dispoziția programatorilor un serviciu local, salvat pe disk-ul de memorie, pentru stocarea fișierelor sau recuperări de date.

Ca reprezentare internă [8], ActiveStorage este stocat în două dintre tablele bazei de date: *active\_storage\_blobs* și *active\_storage\_attachments.* Pentru instalare, se rulează comanda *bin/rails db:migrate*, și astfel se aplică modificările asupra bazei de date, dar în prealabil trebuie creată sau actualizată această migrare, cu ajutorul comenzii *bin/rails active\_storage:install*.

În *config/storage.yml*, fișier creat automat, pot fi configurate modalitățile de stocare a fișierelor ce urmează să fie încărcate în aplicația web, pentru fiecare dintre cele trei medii (dezvoltare, testare și producție). În figura de mai jos este prezentată, pe scurt, structura unei astfel de configurări.



Fig. 4.2 Configurarea ActiveStorage pentru cele trei medii

Mai mult, tehnologia ActiveStorage permite stocarea simultană pe mai multe servere, prin funcția *Mirror Service*, utilă în situații în care dorim să avem copii de rezervă ale datelor sau intenționăm să facem modificări asupra datelor din producție. De menționat este faptul că *Mirror Service* nu realizează operații atomice, deci nu sunt excluse situațiile în care, datorită unor erori de sistem, datele să fie încărcate cu succes într-unul dintre serviciile de stocare, dar să nu fie replicate în mod corect și în serverele copie. De aceea, se recomandă verificarea integrității tuturor datelor înainte de a se trece la operațiile următoare.

Figura următoare indică un exemplu prin care este creată o copie pe serverul *s3\_west\_coast*, a datelor originale, aflate pe serverul *s3\_east\_coast*.

Am ales această tehnologie pentru stocarea imaginilor deoarece, comparativ cu alte soluții din mediul Rails, utilizând ActiveStorage, chiar și modelele existente pot avea fișiere asociate, fără să fie nevoie de a suferi modificări explicite precum adăugarea de noi coloane în tabela din baza de date. ActiveStorage folosește asocieri polimorfice între modelele *Blob* și *Attachment*, orchestrate de către ActiveRecord.

Una dintre cele mai des întâlnite probleme care afectează performanța aplicațiilor Rails, dar și a celorlalte aplicații web, este problema celor N+1 interogări [9]. ActiveStorage vine cu o soluție și pentru aceasta, prin intermediul scope-ului predefinit *with\_attached\_image*.

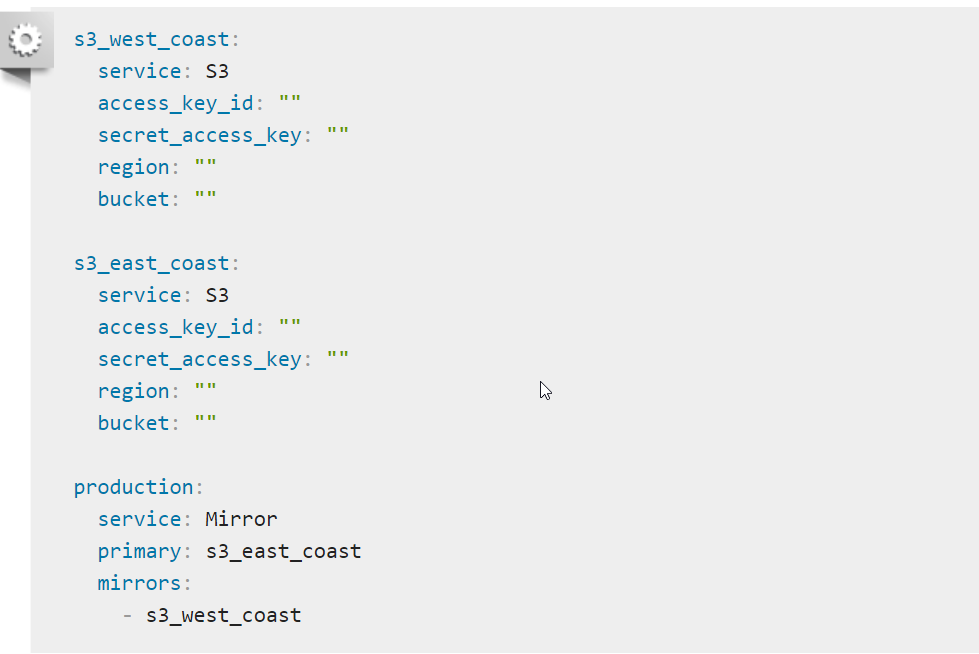


Fig. 4.3 Duplicarea informațiilor pe un alt server

## 4.5 Bcrypt

Deoarece cea mai frecvent utilizată metodă de autentificare în aplicații este cea pe bază de nume de utilizator și de parolă, una dintre cele mai mari greșeli care pot conduce la grave probleme de securitate este stocarea în clar a parolelor. Una dintre soluții este stocarea parolelor utilizând funcții hash, astfel că la fiecare autentificare sunt verificate numele de utilizator și o valoare asociată parolei alese, calculată prin funcția hash. Acestea trebuie să fie unidirecționale, deci prin cunoașterea valorii nu ar trebui să se poată determina parola asociată.

O eroare de securitate des întalnită este aceea că foarte multe persoane folosesc aceeași parolă la mai multe conturi, inclusiv cele bancare, deci este foarte important ca aplicația Travellor să evite cu orice preț aflarea parolei de către atacatori. Prin urmare, am ales să folosesc pentru criptarea parolelor utilizatorilor algoritmul bcrypt [10], dezvoltat de către Niels Provos și David Mazières în cadrul proiectului Open-BSD și disponibil în Rails prin intermediul gem-ului bcrypt.

Funcția hash a acestui algoritm ia o parte din date și creează o amprentă digitală, și pentru că acest proces este ireversibil, nu se poate ajunge înapoi la parola de la care s-a pornit. Comparativ cu MD5 [11], această funcție hash este intenționat creată să fie cu mult mai lentă, îngreunând atacurile de tip brut-force. În figura de mai jos se pot observa diferențele de performanță între cei doi algoritmi:

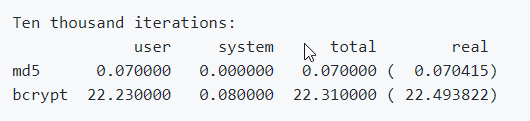


Fig. 4.4 Comparație între algoritmul md5 și bcrypt

Spre exemplu, folosind puterea de calcul a limbajului Ruby, un atacator ar putea încerca într-o secundă autentificarea cu aproximativ 450 de parole stocate cu bcrypt, comparativ cu 140.000 de parole stocate cu MD5.

OpenBSD, privit ca fiind cel mai sigur sistem de operare disponibil în prezent, folosește bcrypt ca metodă implicită de stocare a parolelor.

## 4.6 Gemfile și Gemfile.lock

De multe ori se întâmplă să ne confruntăm cu probleme comune de-a lungul dezvoltării unei aplicații și soluțiile să fie deja dezbatute îndelung și uniform acceptate. În astfel de situații este recomandat să adoptăm una dintre soluțiile oficiale, realizată de programatori experimentați, în detrimentul creării unei soluții similare, dar cu cost de performanță și investind timp și efort în această direcție. Exemple de astfel de situații pot fi: sistemul de autentificare al aplicației, paginarea sau chiar sistemul de trimitere a e-mailurilor.

Există o mulțime de pachete software implementate pentru dezvoltatorii de aplicații Rails, numite gem-uri Ruby. Cele mai multe se găsesc pe **Github** sub formă de repository public, iar comunitatea pune la dispoziție <https://rubygems.org/> pentru căutare și descărcare de astfel de pachete.

Fișierul generat automat odată cu noul proiect Rails, Gemfile, conține o listă de colecții de cod ruby - gems - și versiunile acestora.

Gemfile.lock este un fișier creat și actualizat de fiecare dată când este adăugat sau șters un nou gem. El conține versiunile exacte care au fost instalate, lucru ce conferă proiectului Rails portabilitate. În acest fel, dacă proiectul este mutat pe o altă instanță, la prima rulare se vor instala versiunile descrise de Gemfile.lock, păstrându-se un mediu de dezvoltare consistent. Din acest motiv nu se recomandă editarea directă a acestui fișier.

## 4.7 Jquery și Ajax

Jquery [12] este o bibliotecă a limbajului javascript, creată și lansată în anul 2006 de către John Resig.

Concepută ca o variantă simplă și rapidă pentru manipularea obiectelor din arborele DOM în HTML, biblioteca se dovedește utilă și în ceea ce privește crearea de animații, cereri de tip AJAX sau chiar gestionarea evenimentelor în browser. Unul dintre marile avantaje [13] ale Jquery este faptul că permite manipularea plugin-urilor, adică programatorii pot dezvolta subaplicații pornind de la funcțiile de bază ale bibliotecii, iar în acest mod resursele necesare vor fi încărcate doar la nevoie, lucru ce face ca Jquery să ocupe foarte puțin spațiu și să fie compatibil cu toate versiunile de browsere web existente.

Aflat sub licența MIT, versatil și ușor de extins, Jquery are o variantă simplă de a comunica cu API-uri externe, și anume AJAX.

AJAX, acronim pentru Asynchronous Javascript And XML, este folosit ca o modalitate de comunicare cu serverul, fără ca pagina să fie reîncărcată la fiecare interacțiune cu utilizatorul. Astfel, se pot face actualizări într-o pagină web fără reîncărcarea paginii, se pot cere și primi date de la server și se pot chiar trimite informații fără ca utilizatorul să vadă, în fundalul aplicației.

## 4.8 Baza de date

O parte foarte importantă a unei aplicații web este baza de date, așa că alegerea acesteia trebuie făcută cu atenție și răbdare, după analiza scopurilor și obiectivelor aplicației, estimarea traficului pe site și alți factori decizionali. Deși există o multitudine de articole și review-uri pe această temă, procesul de selecție al bazei de date nu este deloc simplificat, pentru că fiecare are puncte forte și puncte slabe.

### 4.8.1 Baze de date relaționale

Create în anii 70 după teorii și fundamente solide, funcționează și stochează date sub formă de relații (tabele cu rânduri și coloane, în care fiecare celulă reprezintă o informație despre obiectul respectiv). Într-un astfel de sistem, manipularea se face cu ajutorul SQL (Structured Query Language), un limbaj de interogare structurat, avându-și originile bazate pe algebra relațională.

Avantaje și puncte forte:

La capitolul de manipulare al datelor structurate, bazele de date relaționale sunt o alegere excelentă, deoarece conferă suport pentru atomicitate, consistență, izolare și durabilitate (proprietăți cunoscute și sub acronimul ACID [15]). Mai mult, datele sunt stocate și interogate ușor și rapid, iar structura bazei de date poate crește pentru că adăugarea de noi atribute, fără a le modifica pe cele existente, este simplă.

Limitarea a ceea ce utilizatorii pot accesa sau modifica, dar și crearea de roluri sunt asigurate de către structura sistemului de gestiune al bazei de date. Așadar, bazele de date relaționale se dovedesc a fi o alegere bună atunci când în aplicație vor exista mai multe tipuri de utilizatori.

### 4.8.2 Baze de date NoSQL

O bază de date NoSQL folosește un sistem de stocare și interogare diferit de cel folosit în modelul relațional, și anume unul sub formă de document. Acestea nu sunt create sau organizate după o schemă strictă, deci pot fi folosite cu date structurate, semi-structurate și nestructurate. Astfel, programatorii pot introduce modificări mult mai ușor și rapid decât cu bazele de date relaționale.

În general, acestea operează cu documente de diferite forme: BSON, XML sau JSON, dar și de tip PDF sau chiar XLS. Documentele sunt stocate și interogate folosind o cheie, iar aceste baze de date excelează la capitolul consistență și rămân funcționale chiar și atunci când noduri din rețea sunt afectate.

Avantaje:

1. Flexibilitate
2. Abilitatea de a gestiona date semistructurate și nestructurate
3. Potrivită când nu se cunoaște de la început ce tip de date vor fi stocate
4. Se pot defini date structurate într-un document fără a le afecta pe celelalte
5. Ușurință în ceea ce privește scalarea orizontală

Toate avantajele de mai sus sunt disponibile, însă cu anumite costuri: sacrificarea atomicității, consistenței, izolării și durabilității.

### 4.8.3 PostgreSQL

PostgreSQL, cunoscut și ca Postgres [16], este un sistem de gestiune de baze de date relaționale liber și gratuit, susținut de o comunitate numeroasă. Folosit pentru baza de date implicită pe serverul macOS, dar și disponibil pentru alte sisteme de operare, precum Linux, FreeBSD sau Windows, Postgres a evoluat din proiectul Ingres, al Universității din California, Berkeley, și permite adăugarea de funcții customizate, construite cu diferite limbaje de programare, precum C, C++ sau Java.

Companii renumite, precum Apple, Fujitsu, Cisco sau Juniper Network folosesc PostgreSQL pentru avantajele sale, printre care se numără:

* Sistemul de gestiune este *open-source*
* Comunitatea numeroasă și variată, bogată în programatori care dezvoltă module și le pun la dispoziția celorlalți
* Costurile reduse
* Stabilitatea de-a lungul vremii
* Disponibilitatea indiferent de platformă
* Scalabil și responsive pentru volume mari de date
* Numeroase aplicații de gestiune, operare și design a bazelor de date
* Suportă tranzacțille conform cu principiul ACID
* Metode de indexare diverse (B+ tree, GIN – Generalized Inverted Index, GiST – Generalized Search Tree)
* Permite căutarea de tip full-text

Aplicația Travellor a fost concepută și au fost scrise specificații funcționale și non-funcționale în prealabil, așa că structura bazei de date era una clară, cunoscându-se modelele și actorii de la bun început.

# Capitolul 5. PROIECTAREA APLICAȚIEI ȘI DETALII DE IMPLEMENTARE

Acest capitol conține informații detaliate în ceea ce privește proiectare și implementarea aplicației Travellor, precum și exemple, diagrame sau imortalizări ale unor secvențe.

## 5.1 Stilul arhitectural al sistemului

Aplicația este creată respectând modelul MVC (Fig. 5.1), acronim pentru Model (M), View (V) și Controller (C). Rails pune la dispoziție acest stil arhitectural (Fig. 5.2), în care fiecare model poate reprezenta o tabelă în baza de date, căpătând astfel modalități prin care să interogheze, salveze, editeze sau să șteargă date din aceasta. Prin urmare, modelele sunt folosite ca interfață între aplicație și baza de date, și, pe lângă acest rol primar, se pot crea validări și relații între acestea.

Fișierele care formează componenta View (V) sunt specifice Rails, având extensia .html.erb și sunt numite fișiere *ruby embedded*. Ele înglobează scheletul aplicației, realizat în HTML 5, și sunt populate cu informații preluate din modele, sub formă de variabile ale limbajului Ruby. Asupra acestora este aplicată stilizarea, cu ajutorul CSS și Boostrap 4, iar partea dinamică a datelor este asigurată prin intermediul limbajului Javascript, utilizând JQuery și Ajax.

Componenta care orchestrează toate demersurile și cererile este reprezentată de Controller (C). Acesta răspunde interogărilor, solicită bazei de date informațiile de care are nevoie pentru a soluționa cererea utilizatorului, parsează răspunsuri și le organizează pentru a fi afișate într-un mod inteligibil.

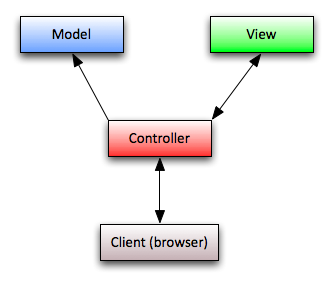


Fig. 5.1 Arhitectura MVC

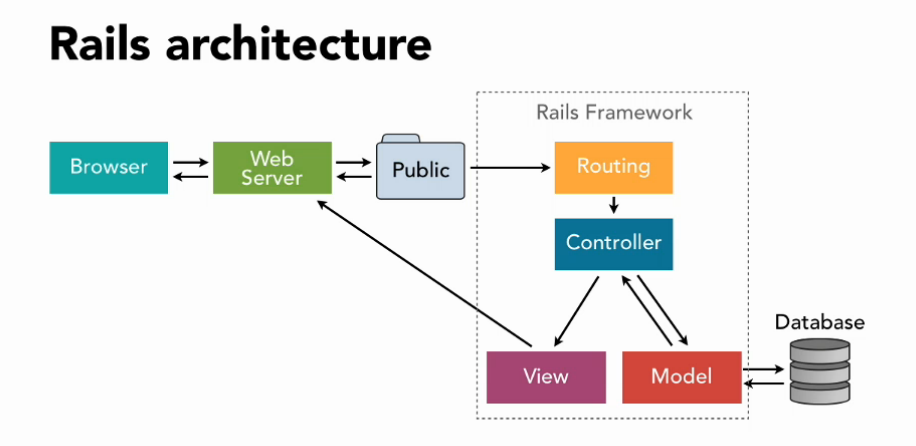


Fig. 5.2 Schema arhitecturii unei aplicații Rails

## 5.2 Nivele aplicației

## Aplicația este creată folosind Ruby 2.5, Rails 5, Postgresql, HTML5, Boostrap 4 și Jquery și se mulează pe arhitectura „Three-Tiered” [18] (împărțită în trei straturi). Fiecare dintre cele trei module este de sine stătător și conferă scalabilitate proiectului.

**Nivelul de prezentare**

## Modulul de prezentare, cel mai înalt nivel arhitectural, este cel care se face responsabil de afișarea informațiilor solicitate de către utilizatori și trebuie proiectat cât mai concis și intuitiv de folosit. Implementat cu ajutorul HTML5, CSS, Boostrap 4 și Jquery, oferă posibilitatea de a vizualiza obiectele aplicației și trimite conținutul formularelor completate de clienți către următorul nivel.

**Nivelul de logică al aplicației**

Nivelul logic este puntea de legătură între ceea ce este vizibil în platforma web și accesul la datele stocate în Postgres. Acesta se ocupă de validarea, integritatea și procesarea datelor, preia și soluționează cererile utilizatorilor, trimise prin intermediul rutelor aplicației și mapate către controllerul potrivit.

Tot sarcină a acestui nivel este și transmiterea în sens invers a datelor, dinspre baza de date către nivelul de prezentare.

**Nivelul de date**

Nivelul de date este reprezentat de către baza de date Postgresql. Informațiile despre hoteluri, rezervări, oferte, recomandări, păreri ale clienților, etc vor fi stocate și interogate prin intermediul unui API pus la dispoziție de către serverul Postgres. În cadrul proiectului, Rails dispune de ActiveRecord, care facilitează toate operațiile și interacțiunile cu baza de date.

ActiveRecord este capabil să:

* reprezinte modelul (M) și datele acestuia
* reprezinte asocierile și relațiile între entități
* reprezinte structurile ierarhice stabilite între entități
* valideze datele modelelor înainte de a fi stocate în baza de date

## 5.3 REST

*Representational state transfer (REST) [19]* definește un stil architectural al aplicațiilor web care urmărește să ofere sistemului performanță, scalabilitate, vizibilitate, portabilitate, simplitate și încredere, la baza căruia se regăsește un set de resurse. Acestea pot fi reprezentate sub diferite forme, unele dintre cele mai folosite fiind JSON (Javascript Object Notation) și XML (eXtensible Markup Language).

Resursele se pot accesa prin URL-uri, folosind un set de operații HTTP (GET, POST, DELETE, etc.). Spre exemplu, o cerere de tipul DELETE /hotels/1 va fi interpretată și va obține dintre resurse hotelul cu id-ul 1, după care se va iniția ștergerea acestuia.

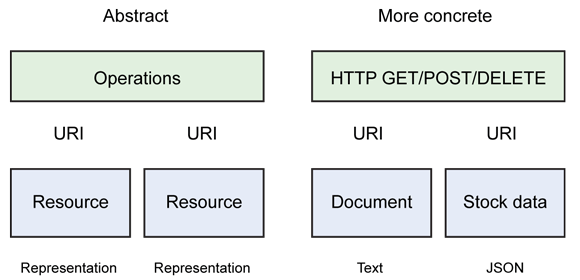


Fig. 5.3 Exemplu de arhitectură care urmează principiile REST

## 5.4 Entitățile aplicației

User Model

Utilizatorii aplicației au asociat câte un astfel de model, care, în mod intern, la nivelul bazei de date, reunește informații precum:

* 1. email (validat cu expresie regulată)
  2. parola (securizată și criptată)
  3. nume de utilizator
  4. rol (client sau manager de hotel)

Un manager poate administra mai multe hoteluri și rezervările acestuia, în timp ce un client poate face rezervări și lăsa review-uri.

 Hotel Model

Modelul care este asociat fiecărui hotel reține informații precum:

1. locație
2. oferte
3. beneficii
4. nume
5. număr de camere din fiecare tip
6. număr de stele

Camerele hotelurilor sunt stocare sub formă de Array cu 3 elemente (numărul de camere single, duble și triple), beneficiile sunt stocate sub formă de Array serializat în baza de date. Fiecare hotel are asociate relații cu locația sa, imaginile, ofertele, rezervările, managerul și review-urile.

Location Model

Acest model aparține unui singur hotel și reține adresa sa, reunind următoarele informații:

1. țară
2. localitate
3. oraș
4. stradă
5. număr

Rezervation Model

Reprezintă modul de stocare în aplicație a unei rezervări, prin informații despre:

1. hotel
2. client
3. perioadă
4. numărul de camere rezervate din fiecare tip disponibil
5. costul total

Asupra acestui model s-au aplicat o serie de validări înainte de a fi salvat în baza de date. Ca o rezervare să fie considerată validă, trebuie să fie selectată cel puțin o cameră, hotelul să aibă camere disponibile în perioada respectivă și perioada aleasă de client trebuie să fie validă (data de final a acesteia să fie cel puțin egală cu cea de început).

Offer Model

O ofertă, în cadrul aplicației Travellor, înseamnă stabilirea de către administratorului de hotel a unui preț pentru un anumit tip de cameră și o anumită perioadă. Este important ca perioada să fie validă și oferta sa nu se suprapună cu alte oferte deja existente pentru același tip de cameră și același interval de timp. De asememea, prețul trebuie să fie o valoare întreagă și pozitivă.

Review Model

Clienții își pot lăsa impresiile în aplicația propusă, sub formă de comentariu liber și de notă între 1 și 5. Review-urile nu pot fi moderate în mod direct de către manager, ceea ce le oferă veridicitate și le face reprezentative pentru formarea unei opinii în ceea ce privește beneficiile și atractivitatea proprietății listate.

Un review aparține unui singur hotel și unui singur client.

## 5.5 Controllerele aplicației

Ca aplicație de tip MVC, Rails permite administrarea și controlul logicii aplicației prin structuri numite Controller (C). Ele sunt responsabile de întreg procesul prin care o cerere ajunsă de la client, prin router, este îndeplinită (restrânge toate resursele necesare, face modificările cerute și restructurează totul sub forma unui răspuns care poate fi interpretat de către client).

În aplicațiile Ruby on Rails, un controller este o clasă, moștenită din clasa de bază ApplicationController și a cărei denumire trebuie să înceapă cu pluralul modelului pe care îl va orchestra și să fie urmat apoi de cuvântul Controller.

Spre exemplu, controller-ul care va îndeplini cererile primite pentru resursa User va purta numele de UsersController și se va găsi în *app/controllers/users\_controller.rb*. Aplicația urmează principiile arhitecturii REST și, prin urmare, metodele din controllere poartă denumiri standard. Astfel, metoda prin care se va putea randa formularul de înscriere în aplicație, găsit la adresa URL: */users/new*, se va găsi în acest controller sub denumirea de *new.*

HotelsController

Toate resursele legate de hoteluri sunt organizate și strânse de către acest controller. Este urmată arhitectura REST, astfel că acesta conține metodele:

* **index** (de tip GET, responsabilă de a întoarce utilizatorului toate hotelurile care respectă criteriile introduse). În funcție de tipul de utilizator autentificat, sunt afișate fie toate hotelurile (dacă utilizatorul este client sau neînregistrat în aplicație), fie propriile hotelurile (dacă utilizatorul este un manager). Tot în această metodă sunt îndeplinite cererile mai specifice, se pot sorta hotelurile în funcție de preț (ascendent sau descendent), în funcție de numărul de stele (crescător sau descrescător) sau în funcție de feedback-ul pe care îl are (numărul de recenzii sau scorul mediu acordat de către alți vizitatori).
* **new** (metodă de tip GET, disponibilă doar managerilor, construiește formularul prin se poate înregistra un nou hotel).
* **create** (metodă de tip POST, conduce datele introduse de manager către backend-ul aplicației, unde urmează să fie stocate în baza de date, dacă validările au trecut cu succes).
* **show** (metodă de tip GET, strânge informațiile necesare pentru ca un hotel și detaliile acestuia să fie afișat în pagina */hotels/id/show*). În această pagină vor fi regăsite imagini cu hotelul, capacitatea și prețurile unei nopți de cazare pentru fiecare dintre tipurile de camere disponibile, se pot consulta părerile unor foști clienți și se pot explora împrejurimile hotelului.
* **edit** (metodă de tip GET, face posibilă afișare formularului prin care se pot actualiza date despre hotel).
* **update** (metodă de tip POST, canalizează datele despre hotel care au fost actualizate și le transmite pentru validare și, după caz, către permanentizarea acestora în baza de date).

ReservationsController

Rezervările sunt create, pot fi vizualizate și editate cu ajutorul acestei clase și a următoarelor metode:

* **index** (metodă de tip GET), deservește atât cererile clienților, cât și pe cele ale managerilor și este inaccesibilă utilizatorilor neînregistrați. Clienții își vor vedea în pagina corespunzătoare acestei metode toate rezervările acestora, în ordine invers contrologică, începând cu cea mai recentă. Managerii de hotel pot vedea toate rezervările tuturor hotelurilor administrate de către ei sau pot opta pentru informații mai precise, prin aplicarea filtrelor puse la dispoziție de aplicație (se pot vedea doar rezervările unui anumit hotel pe care îl selectează dintr-un meniu sau doar rezervările dintr-o anumită perioadă de timp, etc).
* **new** (metodă de tip GET), solicită view-ului corespunzător informațiile și permite afișarea, pentru clienți, a formularului prin care pot înregistra o nouă rezervare.
* **create** (metodă de tip POST), primește în parametrii specifici atributele introduse de către clienți, validează rezervarea și o salvează în baza de date dacă aceasta nu conținea erori. Totodată, în funcție de succesul salvării, este trimis un mesaj de informare corespunzător.
* **edit** (metodă de tip GET), disponibilă pentru managerii de hotel, permite acestora introducerea unor noi valori și actualizarea rezervărilor.
* **update** (metodă de tip POST), transmite datele modificate către logica aplicației, loc unde acestea sunt resupuse validărilor. În cazul în care datele sunt introduse în mod corect, modicările sunt salvate și în baza de date și managerul este notificat că actualizarea s-a efectuat cu succes.

# Capitolul 6. TESTARE ȘI VALIDARE

Acest capitol conține detalii despre cum este asigurată calitatea sofware a produsului. Aplicația Travellor a fost începută în sistemul TDD (Test Driven Development) [20], care reprezintă scrierea testelor unitare înaintea scrierii codului. Principiul Red-Green-Refactor funcționează după următorul algoritm:

* + - 1. Se scrie codul testului corespunzător noii funcționalități
      2. Testul va pica fiindcă momentan nu există nicio implementare (Red)
      3. Se scrie cod suficient cât să facă testul să treacă
      4. Se rulează restul testelor și se asigură că nu s-au introdus greșeli (Green)
      5. Se refactorizează codul conform standardelor de calitate
      6. Repetă procesul cu noua funcționalitate

## 6.1 Testarea unitară a modelelor

Scopul acestei abordări este testarea fiecărei părți introduse în aplicație și este des utilizată în practică deoarece reprezintă cea mai eficientă metodă prin care se pot descoperi eventualele probleme tehnice încă de la început. De asemenea, sunt numeroase situațiile în care specificațiile tehnice ale produsului nu sunt suficiente la momentul implementării și cu ajutorul testării unitare se pot descoperi părți lipsă și scenarii de utilizare noi.

Scrierea testelor înaintea codului este o metodă benefică adesea, deoarece cel care le construiește este pus într-o perspectivă mai amplă și este nevoit să gândească și să găsească cazurile limită, eventuale erori, date de intrare și ieșire comune și relevante și în acest mod numărul de bug-uri se reduce. Deși scrierea de teste necesită timp, dezvoltarea aplicației este mult mai rapidă pentru că toată aceasă perioadă vine în ajutorul programatorului, acesta nefiind nevoit să testeze în interfața grafică toate scenariile aplicației la introducerea unei noi componente. Este suficient ca testele să treacă și poți fi sigur că nu ai introdus greșeli în celelalte părți ale aplicației. În secvența de cod alăturată se află trei teste unitare care verifică integritatea ofertelor adăugate de către managerii de hotel în cadrul aplicației propuse.



Cu ajutorul metodei *offer\_params* se va simula crearea unei oferte hoteliere, care conține detalii despre: hotelul căreia îi aparține oferta, perioada (zi de început – zi de final), tipul camerei căreia i se adresează oferta (single, dublă sau triplă) și prețul unei nopți de cazare.

Primul test asigură faptul că o ofertă cu parametri valizi va fi salvată în baza de date. S-a creat o ofertă pentru perioada 1 iulie 2020 – 31 iulie 2020, la prețul de 150 de lei, pentru fiecare noapte petrecută într-o cameră cu un pat de o persoană. Odată apelată metoda *create,* numărul de oferte din baza de date, reprezentat de *Offer.count*, a crescut cu 1, semn că noua ofertă tocmai a fost salvată.

Cel de-al doilea test validează corectitudinea intervalului de timp în care se aplică oferta. Se încearcă crearea unei oferte pentru un interval invalid: 1 iulie 2020 – 30 iunie 2020. Testul verifică inițial faptul că oferta este invalidă, prin: *assert\_not offer.valid?*, după care se arată faptul că oferta nu va fi salvată dacă utilizatorii vor introduce intervale eronate, lucru care reiese văzând că numărul de oferte din baza de date rămâne același.

Ultimul test pleacă de la premisa că există o ofertă pentru o noapte petrecută în camerele single din hotel, în perioada 1 iulie 2020 – 31 iulie 2020. Aplicația nu va permite adăugarea unei oferte care să se suprapună cu celelalte deja salvate deoarece vor exista ambiguități la calcularea prețului rezervării pentru intervalul de timp inclus în două sau mai multe oferte. Așadar, se încearcă salvarea unei noi oferte pentru același tip de cameră, în intervalul 15 iulie 2020 – 30 iulie 2020, dar oferta este detectată ca fiind invalidă și, prin urmare, la tentativa de salvare în baza de date, acest proces nu va reuși, lucru ce reiese din faptul că numărul de oferte rămâne același ca înainte de încercare.

6.2 Testarea unitară a controllerelor

Pentru ilustrarea manierei în care sunt testate acțiunile controllerelor, și implicit, API-ul (Application Programming Interface), vom lua ca referință următoarea secvență de cod.



Aceste teste verifică endpoint-ul de creare a rezervărilor și testează comportamentul acestuia la metoda post pentru trei situații:

1. Primul test încearcă crearea unei rezervări la Hotelul cu id 1, pentru perioada 26 iunie – 27 iunie, având costul total de 260 de lei pentru două nopți de cazare într-o cameră cu un singur loc. Salvarea în baza de date are loc cu succes, lucru care reiese din creșterea cu unu a numărului total de rezervări create în aplicația Travellor.
2. Cel de-al doilea test arată că nivelul de logică a aplicației refuză crearea unei rezervări care are perioada de vacanță aleasă în mod eronat ( 26 iunie – 25 iunie). Testul verifică faptul că numărul de rezervări din baza de date rămâne nemodificat la încercarea de creare a unei rezervări invalide.
3. Ultimul test indică faptul că serviciul de back-end se opune creării unei rezervări care nu are selectată nicio cameră de cazare. Similar cu cel de-al doilea, este testat numărul de rezervări din baza de date, care trebuie să rămână același ca înainte de tentativa de salvare a noii rezervări.

# Capitolul 7. CONCLUZII

Ultimul capitol al lucrării cuprinde o analiză asupra scopurilor aplicației, a modului în care acestea au fost atinse, dar și o serie de precizări care ar putea constitui îmbunătățiri ulterioare ale platformei.

## 8.1 Realizarea obiectivelor

Obiectivul principal al aplicației, acela de a crea un mediu plăcut și propice atât pentru persoanele în căutare de cazare pentru destinațiile turistice pe care le plănuiesc, cât pentru administratorii de hoteluri din România, a fost îndeplinit cu succes. Clienții pot vizualiza proprietățile listate în platformă chiar și fără a fi înregistrați, pot consulta ofertele acestora, pot vedea capacitatea, beneficiile hotelurilor și disponibilitățile camerelor pentru perioada dorită. Odată cu înregistrarea în aplicație, un utilizator primește și dreptul de a rezerva una sau mai multe camere, iar toate acestea vor rămâne în istoricul aplicației. Pentru un manager de hotel, listarea proprietății se face simplu și rapid, completând un formular concis cu informațiile cele mai relevante pentru cineva aflat în căutare de cazare, precum: locație, beneficii, capacitate, preț etc.

Aplicația este una performantă, în același timp oferind securitate parolelor utilizatorilor, acestea fiind stocate în formă criptată, cu ajutorul Bcrypt. Totodată, aplicația este dezvoltată sub un stil arhitectural ușor de extins, partea de back-end poate fi utilizată pentru orchestrarea datelor folosite în realizarea unei varianta de mobil a platformei.

## 8.2 Dezvoltări ulterioare

Una dintre îmbunătățirile ce ar putea fi aduse aplicației ar fi extinderea acesteia pe dispozitivele mobile iOS și Android. De asemenea, în planul de dezvoltare se regăsesc și o serie de funcționalități care ar putea fi implementate pentru o experiență completă și mai plăcută a utilizatorilor, precum:

* Crearea unei modalități de comunicare între clienți și manageri, în prezent aceste interacțiuni realizându-se prin e-mail, după realizarea unei rezervări în aplicație;
* Crearea unui mod prin care un client să poată solicita o ofertă personalizată;
* Crearea posibilității pentru clienți de a obține un tur virtual în camera de hotel și în locurile cu acces gratuit ale proprietății;
* Notificarea și prin SMS la crearea de rezervare, modificarea acesteia și o serie de mesaje care să reamintească perioada în care rezervarea se poate anula în mod gratuit;
* Întocmirea unui sistem care să contorizeze numărul de rezervări ale clienților și să ofere stimulente și bonusuri celor fideli sau celor care călătoresc frecvent.

**Referințe**

[1] Cojocea Luminiţa. (2014). *Promovarea online a hotelurilor din România: Compromis între deficienţe şi avantaje?*

[2] About Ruby, <https://www.ruby-lang.org/en/about/>

[3] Metaprogramming, <https://en.wikipedia.org/wiki/Metaprogramming>

[4] Rails, <https://rubyonrails.org/>

[5] Model-view-controller, <https://ro.wikipedia.org/wiki/Model-view-controller>

[6] Active Record Basics, <https://guides.rubyonrails.org/active_record_basics.html>

[7] Active Storage Overview, <https://edgeguides.rubyonrails.org/active_storage_overview.html>

[8] Active Storage Codebase, <https://github.com/rails/rails/tree/master/activestorage>

[9] Bret Doucette, *N+1 Queries and How to Avoid Them!,* <https://medium.com/@bretdoucette/n-1-queries-and-how-to-avoid-them-a12f02345be5>

[10] Bcrypt Codebase, <https://github.com/codahale/bcrypt-ruby>

[11] MD5, <https://ro.wikipedia.org/wiki/MD5>

[12] Jquery homepage, <https://jquery.com>

[13] Jquery API, <https://api.jquery.com/>

[14] Codd’s 12 rules, <https://en.wikipedia.org/wiki/Codd%27s_12_rules>

[15] ACID, <https://en.wikipedia.org/wiki/ACID>

[16] PostgreSQL: The World’s Most Advanced Open Source Relational Database, <https://www.postgresql.org/>

[17] Appendix L. PostgreqSQL Limits, <https://www.postgresql.org/docs/12/limits.html>

[18] Three-tier architectures, <https://www.ibm.com/support/knowledgecenter/zh-tw/SSEQTP_8.5.5/com.ibm.websphere.base.iseries.doc/ae/covr_3-tier.html>

[19] M. Tim Jones, *Representational State Transfer (REST) in Ruby*, <https://www.ibm.com/developerworks/library/os-understand-rest-ruby/>

[20] Introduction to Test Driven Development (TDD), <http://agiledata.org/essays/tdd.html>