



Universitatea POLITEHNICA București
Facultatea de Automatică și Calculatoare
Departamentul de Automatică și Informatică Industrială



LUCRARE DE LICENȚĂ

**Aplicație pentru gestiunea deplasărilor în
cadrul unei organizații**

Coordonator:

Șl.Dr.Ing Călin Munteanu

Absolvent:

Valeriu-Florin Oprea

BUCUREȘTI

Iulie 2016

Cuprins

| | |
|---|-----------|
| 1. Introducere | 4 |
| 1.1. Scopul si obiectivele lucrării de licență | 5 |
| 1.2. Descrierea domeniului din care face parte lucrarea de licență | 6 |
| 1.3. Prezentare pe scurt a capitolelor | 7 |
| 2. Descrierea problemei abordate..... | 8 |
| 2.1. Descrierea problemei abordate | 9 |
| 2.2. Studiu asupra realizărilor similare in domeniu | 10 |
| 2.3. Stabilirea funcționalităților aplicației raportat la funcționalitățile aplicațiilor similare.... | 11 |
| 3. Analiza metodelor de rezolvare a problemei și selectarea soluției tehnice..... | 12 |
| 3.1. Descrierea tehnologiilor disponibile | 13 |
| 3.1.1. Tehnologiile folosite pentru partea de client | 13 |
| 3.1.1.1. HTML | 13 |
| 3.1.1.2. AngularJS..... | 13 |
| 3.1.1.3. Twitter Bootstrap | 13 |
| 3.1.2. Tehnologiile folosite pentru partea de server..... | 13 |
| 3.1.2.1. Java | 13 |
| 3.1.2.2. Spring..... | 14 |
| 3.1.2.2.1. Spring Core | 14 |
| 3.1.2.2.2. Spring Boot | 14 |
| 3.1.2.2.3. Spring Data JPA..... | 14 |
| 3.1.2.2.4. Spring MVC..... | 14 |
| 3.1.2.3. Servicii web | 14 |
| 3.1.2.3.1. Arhitectura de tip RPC (Remote Procedure Call | 14 |
| 3.1.2.3.2. Arhitectura de tip REST (Representational State Transfer | 14 |
| 3.1.3. Tehnologiile folosite pentru partea de date..... | 14 |
| 3.1.3.1. Liquibase..... | 14 |
| 3.1.3.2. MySQL | 14 |
| 3.1.4. Mediile de dezvoltare folosite..... | 14 |
| 3.1.4.1. IntelliJ IDEA | 14 |
| 3.1.4.2. XAMPP..... | 14 |
| 3.2. Alegerea tehnologiilor utilizate..... | 14 |
| 4. Considerente legate de dezvoltarea aplicației..... | 15 |
| 4.1. Arhitectura aplicației..... | 16 |
| 4.1.1. Structura unei aplicații Java web și modul de funcționare a acesteia | 16 |
| 4.1.2. Prezentarea bazei de date | 16 |
| 4.1.3. Diagrama de process | 17 |
| 4.2. Implementarea aplicației | 17 |

| | |
|---|-----------|
| 4.2.1. Înregistrarea unui cont de utilizator | 18 |
| 4.2.2. Autentificarea și editarea contului | 18 |
| 4.2.3. Gestionarea platformei de către administrator | 18 |
| 4.2.3.1. Gestionarea utilizatorilor aplicației | 18 |
| 4.2.3.2. Gestionarea tipurilor de costuri | 18 |
| 4.2.4. Ciclul unei cereri de deplasare și permisiunile utilizatorilor asupra acesteia | 18 |
| 4.2.4.1. Crearea cererii de deplasare de către manager și trimiterea acesteia către furnizor | 18 |
| 4.2.4.2. Adăugarea costurilor de deplasare de către furnizor și retrimiteră către manager | 18 |
| 4.2.4.3. Acceptarea sau respingerea cererii de către manager | 18 |
| 4.2.4.4. Atașarea actelor necesare deplasării | 18 |
| 4.2.4.5. Permisuniile utilizatorilor asupra cererilor de deplasare | 18 |
| 4.2.5. Auditul cererilor de deplasare | 19 |
| 4.3. Testarea aplicației | 18 |
| 5. Concluzii și contribuții personale | 20 |
| 5.1. Concluzii | 21 |
| 5.2. Contribuții | 22 |
| 5.3. Dezvoltări ulterioare | 23 |
| 6. Bibliografie | 24 |

1. Introducere

Aplicațiile web au început să devină din ce în ce mai populare în rândul utilizatorilor datorită numeroaselor avantaje oferite în comparație cu aplicațiile standard care necesită instalare și rulare.

Spre deosebire de aplicațiile clasice, aplicațiile web conferă avantaje în ceea ce privește platforma de rulare, fiind independente de sistemul de operare, putând fi rulate utilizând un browser. Alt avantaj este constituit de ușurința cu care se pot face actualizări asupra aplicației, fără a impacta utilizatorii prin reinstalarea aplicației pe computerul acestora, modificările fiind făcute într-un singur loc, și anume pe server.

1.1. Scopul și obiectivele lucrării de licență

Unul dintre lucrurile esențiale într-o companie ai cărei angajați sunt nevoiți să călătorească în interes de serviciu este existența unei aplicații capabile să gestioneze aceste călătorii și tot procesul premergător realizării lor.

Lipsa unei astfel de aplicații în cadrul unei companii în care această practică este foarte des întâlnită, poate duce la îngreunarea procesului de realizare a acestor deplasări. Astfel, principalele inconveniente apărute pot fi reprezentate de durata de timp cuprinsă între etapa de solicitare respectiv cea de aprobare, imposibilitatea gestionării acestora la scară largă, cât și lipsa controlului asupra limitelor de buget și asupra costurilor de buget.

Pornind de la necesitatea unei organizații de a avea un instrument ușor de folosit pentru a ajuta angajații atunci când doresc să meargă într-o călătorie de afaceri, scopul lucrării de licență constă în dezvoltarea unei aplicații web ce va facilita gestionarea acestor deplasări. Prin intermediul aplicației se va realiza solicitarea de aprobări pentru deplasări, pornirea fluxului de lucru pentru acestea, gestionarea resurselor necesare acestor călătorii cât și extragerea de rapoate pe baza datelor introduse.

1.2. Prezentare pe scurt a capitolelor

Lucrarea este structurată în șapte capitole, fiecare capitol descriind pașii premergători realizării acestei aplicații.

Astfel, primul capitol prezintă ideea centrală a lucrării, și anume necesitatea existenței unui sistem de gestionare a călătoriilor în cadrul unei organizații ușor de utilizat.

Capitolul al doilea realizează trecerea un studiu asupra aplicațiilor de aceeași natură deja existente pe piață și prezintă funcționalitățile de bază ale aplicației.

Următorul capitol își propune să treacă în revistă tehnologiile utilizate în realizarea sistemului, motivând alegerile făcute.

Al patrulea capitol este centrat asupra aplicației în sine, prezentând arhitectura aplicației, modalitatea în care aceasta a fost dezvoltată și modul de funcționare al acesteia.

Capitolul cinci prezintă scenariul de utilizare, precum și modul de testare al acesteia. Contribuțiile personale, dezvoltările ulterioare și bibliografia sunt prezentate în ultimele două capitole.

2. Descrierea problemei abordate

2.1. Descrierea problemei abordate

Plecând de la scopul acestui proiect, și anume acela de a automatiza procesul de gestiune a deplasărilor în cadrul unei organizații, putem deduce foarte ușor care sunt utilizatorii finali ai acestui produs. Astfel, aceștia sunt reprezentați de către angajații acelei organizații care doresc să se deplaseze într-o călătorie de afaceri cât și de către furnizorii de oferte.

Utilizatorii aplicației vor fi reprezentați de către angajații instituției care va folosi aplicația, grupați în funcție de rolul îndeplinit în cadrul acesteia. Astfel, fiecare utilizator al aplicației poate avea unul sau mai multe roluri. Toți utilizatorii au rolul de angajat (ROLE_EMPLOYEE). Pe lângă acesta, unii angajați pot fi manageri (ROLE_MANAGER), furnizori (ROLE_SUPPLIER) sau administratori (ROLE_ADMIN).

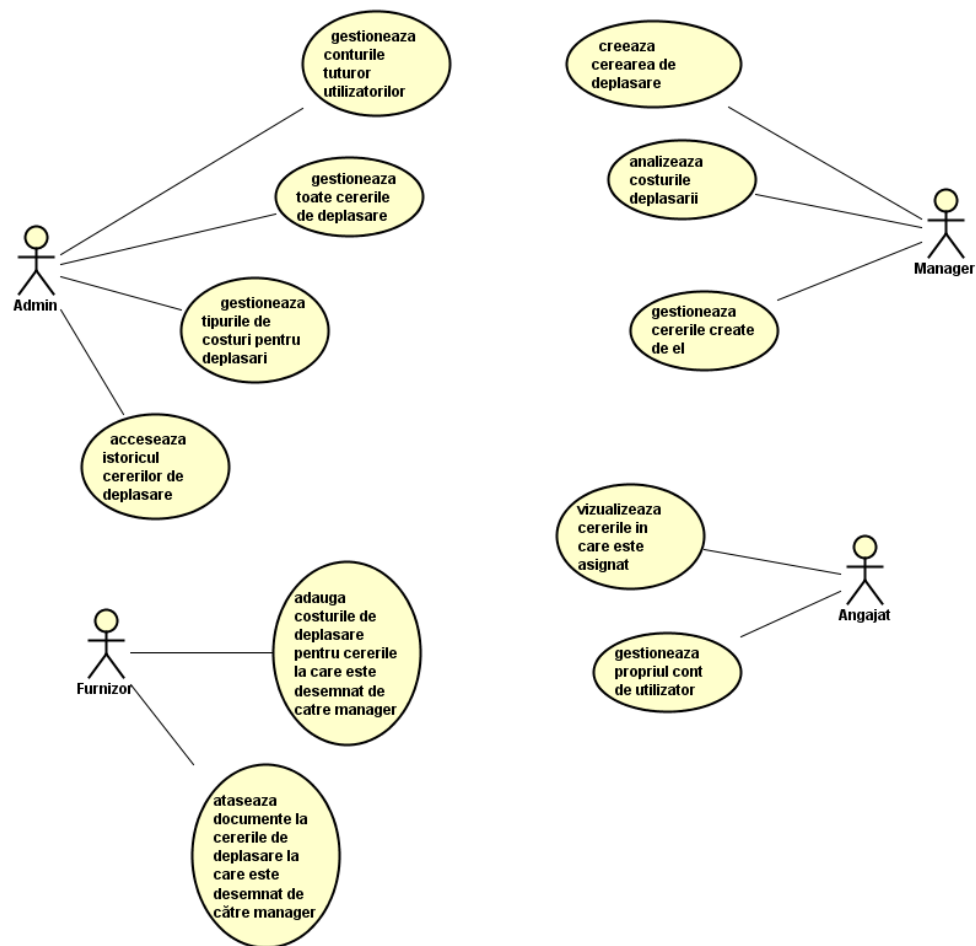


Figura 1: Diagrama cazurilor de utilizare

2.2. Descrierea domeniului din care face parte lucrarea de licență

Aplicația dezvoltată face parte din domeniul aplicațiilor de gestiune. O aplicație de gestiune este o aplicație software, prin intermediul căreia datele sunt controlate mai ușor, fiind facilitată raportarea și evidența acestora. Cel mai important rol în cadrul unei astfel de aplicații este cel al sistemului de gestiune al bazelor de date. Prin intermediul acestuia, utilizatorul creează și întreține datele.

Avantajele folosirii unui astfel de sistem sunt următoarele:

- Accesul rapid la informațiile dorite
- Posibilitatea folosirii în mod simultan de către mai mulți utilizatori
- Ușurează comunicarea între mai multe departamente

- Distanța nu va mai reprezenta un impediment pentru comunicare
- Planificarea activităților în cadrul unei instituții este mult mai simplu de realizat

2.3.Studiu asupra realizărilor similare în domeniu

Platforma KISSFLOW (Keep it Simple & Smart Workflow) a fost dezvoltată de către compania OrangeScape și proiectată să lucreze cu platforma Google Apps, lucrând cu diferite aplicații oferite de Google, cum ar fi Google Docs, Contacts sau Mail. [1]

Prin aplicațiile oferite, această platformă furnizează clienților Google Apps, diverse modalități de a-și automatiza anumite fluxuri de lucru. Una dintre aplicațiile de acest gen este Travel Request.



Figura 2: Logo-ul platformei KISSFLOW [2]

Această aplicație oferă clienților posibilitatea pornirii unor fluxuri de lucru pentru deplasări, gestionarea acestora cât și gruparea acestora în funcție de statusul în care se află procesul. [3]

O altă platformă ce oferă suport pentru gestionarea unor călătorii este Concur. Principalele funcționalități sunt:

- Găsește cele mai bune oferte de călătorii, fiind conectată cu diverse agenții de turism
- Oferă un set standard de rapoarte asupra cheltuielilor efectuate, ajutând companiile ce o folosesc să-și gestioneze cu exactitate bugetul.

Scopul principal al acestei platforme este acela de a minimiza pe cât posibil costurile necesare unei călătorii

2.3. Stabilirea funcționalităților aplicației raportat la funcționalitățile aplicațiilor similare

Pornind de la aplicațiile prezentate, aplicația pe care o voi dezvolta, va fi una stand-alone, independentă de aplicațiile conținute de platforma Google Apps, va căuta diverse oferte prin intermediul unor furnizori ce vor fi reprezentați ca niște utilizatori în aplicație. Aceștia vor primi cererile de deplasare ce conțin detaliile acestora, de la inițiatori și vor răspunde acestora cu anumite costuri de deplasare.

În funcție de costurile primite, solicitantul, va aproba oferta primită de la furnizor, caz în care aceasta va fi transmisă superiorului ierarhic dacă acesta există, sau o va respinge, caz în care se va întoarce la furnizor, care va căuta alte oferte și le va retrimite solicitantului.

Funcționalitățile similare cu cele oferite deja de aplicația Travel Request vor fi:

- Completarea unui formular pentru solicitare de aprobări pentru deplasări de către solicitant
- O cerere poate fi anulată de către cel care a inițiat-o.
- Redirecționarea cererii în funcție de statusul acesteia către următoarea persoană autorizată să o aprobe.

Pe lângă funcționalitățile oferite de aplicația Travel Request, aplicația pe care o voi dezvolta, va avea în plus următoarele funcționalități:

- Posibilitatea de înregistrare a unui utilizator nou în aplicație de către administratorul acesteia pe baza unor informații cerute cum ar fi: username, nume, prenume, rolul îndeplinit în aplicație și adresă de email.
- La fiecare înregistrare, noul utilizator va primi un email pentru a-și activa contul și totodată de a-și seta o parolă prin accesarea unui link conținut în email.
- Administratorul poate adauga noi tipuri de costuri în aplicație și poate vizualiza istoricul tuturor cererilor create.
- Prin intermediul furnizorilor, vor fi găsite cele mai bune oferte pentru deplasări
- Fiecare utilizator are anumite drepturi asupra cererilor în care este implicat, în funcție de rolul îndeplinit în cadrul aplicației.
- Fiecare utilizator își poate modifica datele de contact din aplicație sau schimba parola.

3. Analiza metodelor de rezolvare a problemei și selectarea soluției tehnice

3.1.Descrierea tehnologiilor folosite

O aplicație web este formată în principal din 3 mari părți:

- Partea de client (reprezentată de interfața cu utilizatorul)
- Partea de server (reprezentate de partea de program care prelucrează datele)
- Partea de date (reprezentată de datele stocate din baza de date)

Fiecare dintre cele 3 părți are asociate niște tehnologii ce pot fi folosite pentru realizarea unei astfel de aplicații.

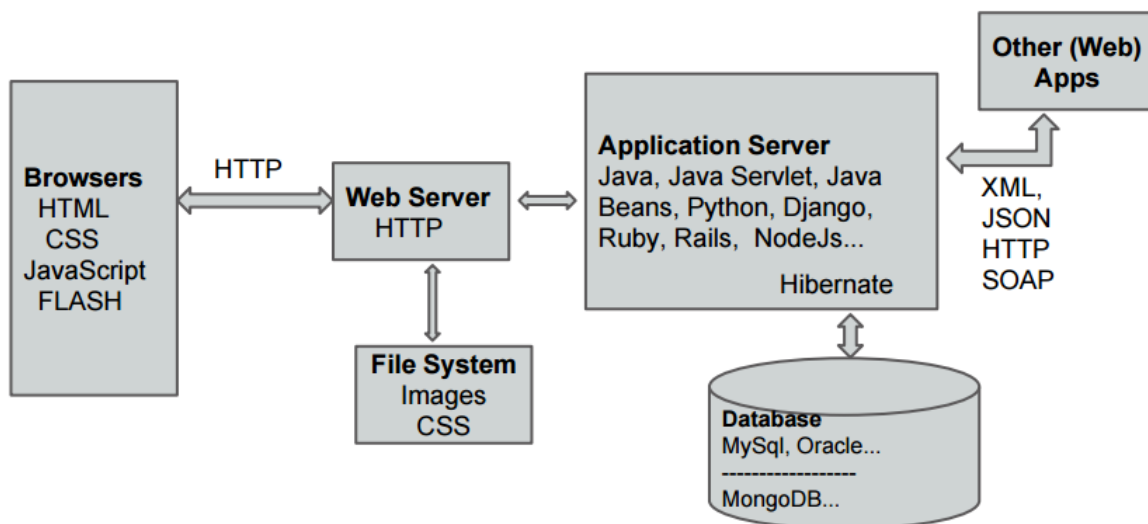


Fig 3: Arhitectura unei aplicații web [4]

3.1.1. Tehnologiile folosite pentru partea de client

Pe partea de client voi folosi următoarele tehnologii:

- HTML5
- AngularJS
- Twitter Bootstrap

3.1.1.1. Hyper Text Markup Language (HTML)

HTML este un element fundamental al sistemului World Wide Web (WWW), dezvoltat în anul 1989, la Cern, de către Tim Berners-Lee pentru a ajuta fizicienii să schimbe informație între ei, utilizând internetul.



Figura 4: Logo HTML5 [5]

WWW folosește modelul client-server, cele două componente fiind programe ce rulează pe niște calculatoare, având funcții diferite: clientul trimițând cereri către server și afișând răspunsul primit de la server.

Există un limbaj de marcare, acesta este folosit pentru a descrie pagini web cu ajutorul tagurilor disponibile. Este derivat din SGML(Standard Generalized Markup Language). [6]

3.1.1.2. AngularJS

JavaScript este un limbaj de scripting, proiectat de către Brendan Eich pentru a rula în browserele web și pentru a ajuta la dinamizarea paginilor web. Prima denumire a fost de LiveScript.[7]

Principalele caracteristici ale acestui limbaj sunt:

- Poate fi introdus în HTML
- Este interpretat de către browser în totalitate
- Nu este orientat pe obiect, e bazat pe obiecte
- Gestionează evenimentele generate de utilizator sau sistem

Una din cele mai importante caracteristici ale acestui limbaj este evoluția.

Astfel, de-a lungul timpului au apărut numeroase framework-uri și librării de JavaScript menite să ajute dezvoltatorii de aplicații web, pe partea de client a aplicațiilor.

Cele mai folosite framework-uri de JavaScript sunt următoarele:

- AngularJS
- JQuery
- ReactJS
- ExtJS



Figura 5: Logo-ul AngularJS [8]

În această aplicație voi folosi AngularJS, cel mai popular framework de tip MVC(Model View Controller) de tip client, dezvoltat și întreținut de către Google și alți dezvoltatori individuali. Este un framework folosit pentru a dezvolta aplicații de tipul Single Page Application(SPA).[9]

MVC este un design pattern ce are rolul de a împărți o aplicație în mai multe părți, fiecare dintre ele având un rol bine definit în cadrul aplicației(modelul se ocupă cu modul de reprezentare a datelor, view-ul cu afișarea acestora, iar controller-ul cu prelucrarea lor).

Principalele facilități oferite de acest framework sunt următoarele:

- Permite dezvoltarea cu ușurință de aplicații de tip SPA
- Oferă posibilitatea de a lega foarte ușor date la HTML (data-binding)
- Oferă componente reutilizabile
- Se integrează foarte ușor cu Bootstrap, permițând crearea de aplicații responsive care pot fi accesate atât de pe calculator cât și de pe dispozitivele mobile.
- Permite crearea și modelarea tagurilor proprii și a atributelor acestora

3.1.1.3. Twitter Bootstrap

Bootstrap este un framework scris în HTML,CSS si JavaScript, folosit pentru stilizarea paginilor web. A fost lansat oficial în august 2011, însă era folosit de compania Twitter încă din 2010 sub numele de Twitter Blueprint. Momentan se află la versiunea 3.3.6, urmând ca foarte curând sa apară versiunea 4.

Acest framework a dobândit o deosebit de mare popularitate în ultimii ani datorită multitudinii de facilități pe care le aduce în dezvoltarea interfeței grafice pentru aplicațiile web.



Figura 6: Logo-ul Twitter Bootstrap [10]

În primul rând, framework-ul este în totalitate open-source, adică produsul este disponibil online oricărui utilizator și nu necesită achitarea unei taxe în prealabil achiziționării. De asemenea, prin termenul de open-source se mai înțelege și faptul că oricine poate avea acces la codul sursă și poate aduce o contribuție la îmbunătățirea proiectului.

Un alt motiv pentru care Bootstrap este atât de răspândit ar fi faptul că folosirea sa accelerează foarte mult procesul de dezvoltare a unei aplicații pe partea de interfață cu utilizatorul. Totodată, website-ul oficial conține o documentație amplă, oferind atât suport pentru instalare cât și numeroase exemple de utilizare.[11]

Nu în ultimul rând, Bootstrap se pretează foarte bine pentru aplicații care trebuie să aibă design responsive, adică o aranjare în pagină diferită în funcție de rezoluția dispozitivului de pe care se accesează aplicația.

3.1.2. Tehnologiile și serviciile folosite pentru partea de server

Pentru partea de server a aplicației, voi folosi mediul de programare Java, framework-ul Spring și servii web de tip REST.

3.1.2.1. JAVA

Java este un mediu de programare dezvoltat de către compania SUN în anul 1991, capabil să ofere utilizatorilor instrumentele necesare pentru dezvoltarea de aplicații distribuite și independente de platformă.

Acest mediu este alcătuit din limbajul de programare Java, un compilator ce are rolul de a transforma fisierul sursă într-un limbaj intermediar numit *bytecode*, mașina virtuală Java Virtual Machine (JVM) ce permite transformarea codului intermediar în instrucțiuni ce sunt executate de procesor și o bibliotecă ce conține numeroase clase grupate în pachete.

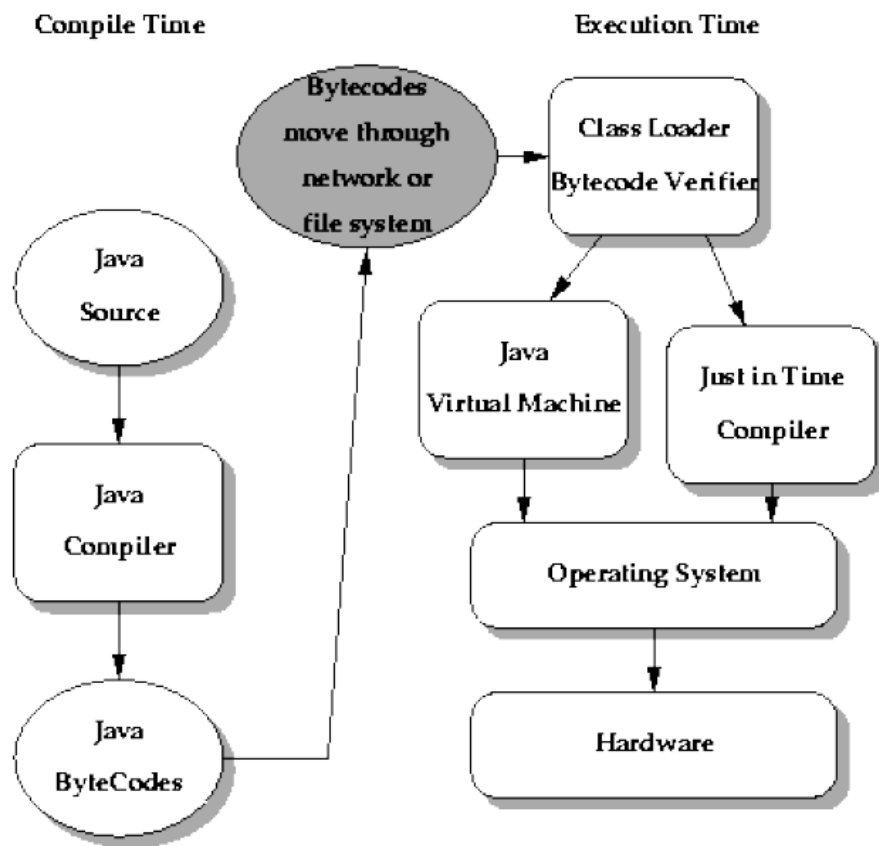


Figura 7: Fluxul de lucru al unui program java [12]

Principalele caracteristici ale limbajului Java sunt:

- Simplitatea
- Robustețea
- Securitatea
- Dinamic
- Independența de platformă
- Suportul pentru multithreading

Prima caracteristică este satisfăcută de faptul că acest limbaj este un limbaj orientat pe obiecte în întregime, spre deosebire de C++. Existența conceptului de Garbage Colector, prin care zonele de memorie nu mai sunt dezalocate de către programator precum în C++, întărește ideea că acest limbaj este caracterizat de simplitate. [13]

Robustețea este asigurată de existența celor două nivele de verificare a codului (primul la compilare iar al doilea în timpul rulării), asigurând astfel un grad mărit de siguranță a codului.

O altă problemă pe care acest limbaj reușește s-o trateze este aceea a securității.

Deoarece multe decizii privind evoluția programului sunt luate la runtime, Java este un limbaj dinamic.

Independența față de platformă este realizată prin intermediul mașinii virtuale ce trebuie instalată pe platforma respectivă.

Un alt avantaj oferit este suportul pentru multithreading la nivel de limbaj.

Java oferă câteva produse care au fost concepute să ruleze pe platforme diferite:

- Java Standard Edition (J2SE) – conține limbajul Java și mediul în care sunt executate aplicațiile
- Java Enterprise Edition (J2EE) – pune la dispoziție instrumentele necesare dezvoltării aplicațiilor organizate într-o arhitectură multi-nivel.

Comparativ cu alte limbaje, Java are avantajul de a fi în strânsă legătură cu tehnologia Internet.

De-a lungul timpului au apărut numeroase framework-uri care acoperă specificațiile Java EE.

Principalele framework-uri de Java sunt:

- Spring
- Play
- Dropwizard
- Spark

3.1.2.2.Spring

Framework-ul Spring este o aplicație open-source, apărută în 2004, ca o alternativă la modelul JavaBean Enterprise (EJB), având ca scop principal simplificarea modului de dezvoltare a aplicațiilor Java. Acesta poate fi privit ca un container, care nu necesită instalare sau configurare ci doar o colecție de arhive Java (JAR) ce trebuie incluse în proiect, dar în același timp și ca un API (Application Programming Interface) ce conține o colecție de clase, metode, interfețe și adnotări.[14]

Acest container poate fi privit ca nucleul acestui framework, având rolul de a gestiona obiectele Java, folosind callback-uri. Callback-ul este o referință către un cod executabil sau o parte a unui cod executabil, care este transmis ca parametru către alt cod.

Astfel, containerul se ocupă de crearea și inițializarea obiectelor, care se mai numesc si bean-uri(Beans).

Fiind format din 20 de module, acest framework oferă o întreagă gamă de servicii grupate în câteva părți principale:

- Core
- Data Access/Integration
- Web
- AOP (Aspect Oriented Programming)
- Instrumentation
- Test

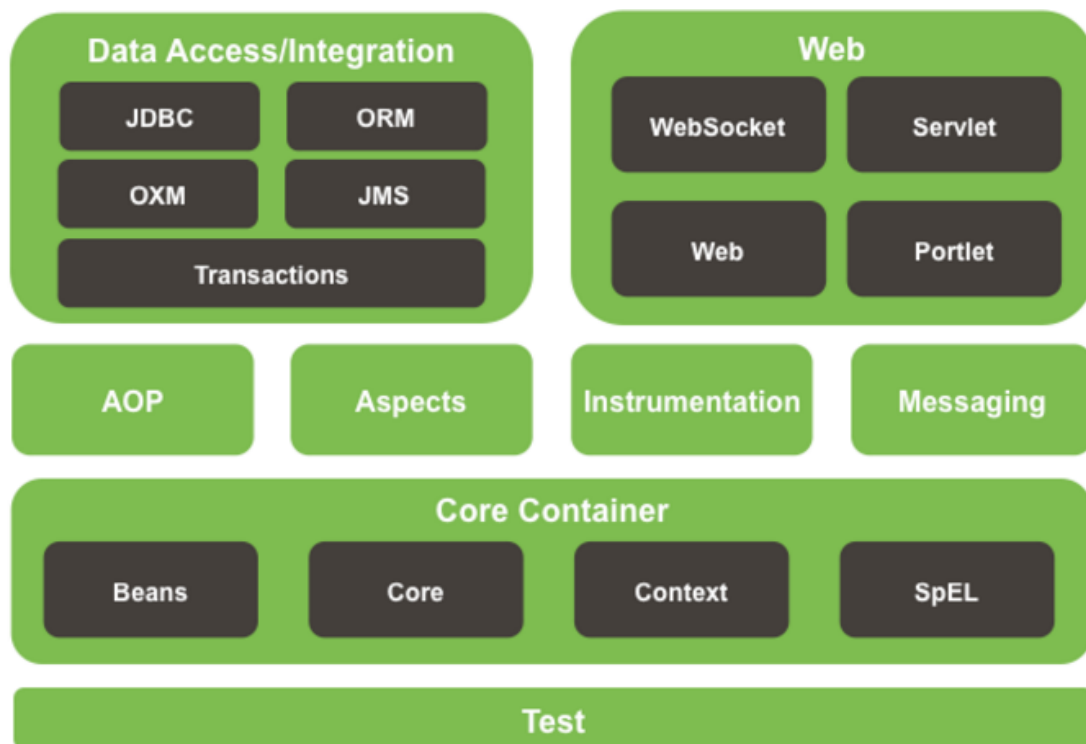


Figura 8: Componentele framework-ului Spring[15]

3.1.2.2.1. Spring Core

Spring Core este modulul fundamental al acestui framework, fiind folosit de către restul modulelor. Principalele funcționalități sunt bazate pe conceptele de Injectarea a dependențelor (Dependency Injection – DI) și Inversarea Controlului (Inversion of control – IoC).

Injectarea dependențelor este un exemplu concret de Inversare a Controlului, ce constă în injectarea de către container a unui obiect în alt obiect, păstrându-le în același timp independente unul de celălalt. [16]

3.1.2.2.2. Spring Boot

Spring Boot este o aplicație dezvoltată de către dezvoltatorii framework-ului Spring, cu scopul de a simplifica procesul de configurare și rulare a unei aplicații Spring.

3.1.2.2.3. Spring Data JPA

Spring Data este partea componentă a acestui framework, care se ocupă cu accesul la datele aplicației. Unul dintre modulele acestei părți este Spring Data JPA (Java Persistence API). Principalul avantaj al folosirii acestui modul este acela că, programatorul nu trebuie să mai descrie operațiunile de acces la baza de date de nivel redus(interogări SQL) , acestea fiind generate în mod dinamic la runtime, pe baza unor interfețe definite de programator, care prin metodele descrise, stabilesc operațiunile asupra bazei de date. [17]

3.1.2.2.4. Spring MVC

Spring Web e partea componentă a framework-ului cu ajutorul căreia, se pot dezvolta aplicații web.

Spring MVC este unul dintre modulele acestei părți, care oferă o implementare de tipul Model-View-Controller aplicațiilor web, permițând în același timp folosirea tuturor instrumentelor puse la dispoziție de către Spring. Scopul patternului MVC este acela de a separa diferitele aspecte ale unei aplicații, cu scopul de a o face ușor de modificat și întreținut.

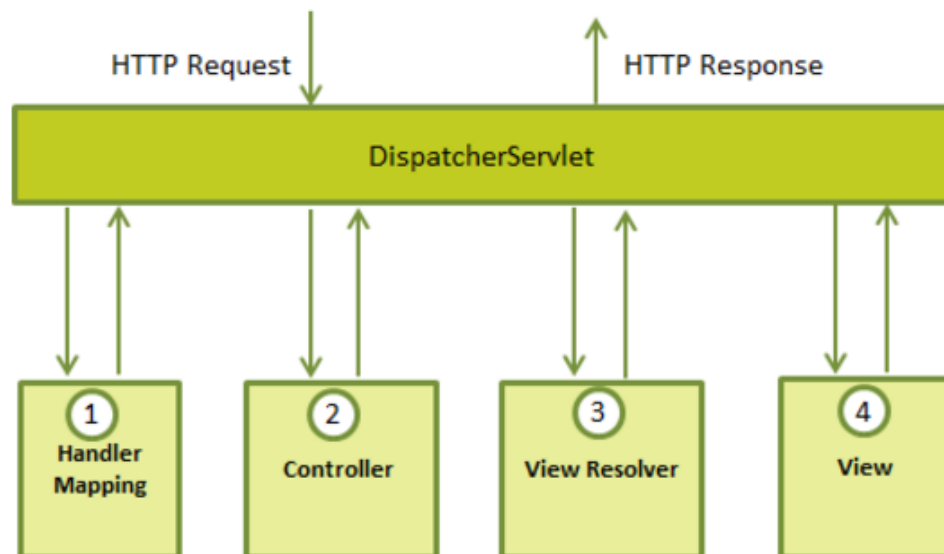


Figura 9: Diagrama fluxului de lucru în Spring MVC[18]

Astfel, modelul (M) are rolul de încapsula informația aplicației sub forma unor clase numite Plain Old Java Object (POJO).

View-ul (V) are rolul de a randa datele conținute în model sub forma de pagini HTML.

Controllerul (C) este responsabil pentru procesarea cererilor utilizatorilor, construirea modelului și transmiterea acestuia mai departe către view pentru randare.

Spring MVC este proiectat în jurul unui dispecer care are rolul de a manipula cererile și răspunsurile. Spring MVC poate fi integrat cu diverse servicii web.

3.1.2.3. Servicii web

Odată cu evoluția tehnologiei, au apărut îmbunătățiri și în sfera Internetului. Astfel au apărut numeroase standarde și protocoale care au permis comunicarea între aplicațiile distribuite pe platforme aflate la distanțe mari, prin intermediul Internetului.

Serviciile web sunt servicii disponibile pe Internet, prin intermediul cărora se accesează diferite informații. Acestea nu sunt legate la vreun sistem de operare sau la vreun calculator.

Principalele avantaje oferite de folosirea serviciilor web sunt:

- Interoperabilitatea între aplicații (este permis schimbul de date între aplicații de pe platforme diferite)
- Dezvoltarea rapidă
- Reutilizarea serviciilor existente
- Distribuirea ușoară a informației între consumatori

Acționând ca un intermediar între două aplicații aflate pe platforme diferite, un serviciu web are ca elemente componente limbajul XML (eXtensible Markup Language) și protocolul de transport HTTP (HyperText Transfer Protocol).

Numeroasele standarde și protocoale apărute, au adăugat un mare plus de complexitate anumitor servicii web, ducând la împărțirea acestora în două mari categorii:

3.1.2.3.1. Arhitectura de tip RPC (Remote Procedure Call)

Arhitectura de tip RPC este o arhitectură complexă, orientată pe servicii, prin intermediul căreia este permisă rularea unui program de pe un calculator pe un alt calculator, fără ca detaliile acestei interacțiuni de la distanță să fie stabilite explicit de cineva.

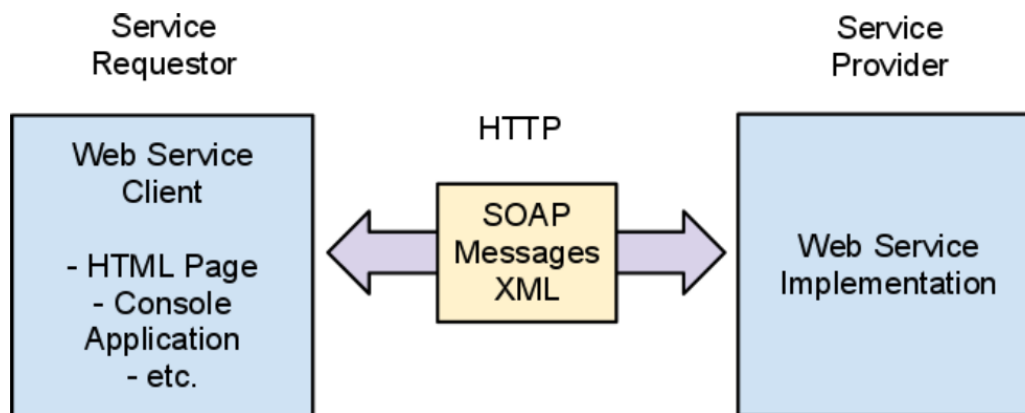


Figura 10: Arhitectura orientate pe servicii[19]

Un serviciu web de tip RPC este alcătuit dintr-un furnizor de servicii, un consumator de servicii și un registru de servicii care au rolul de a implementa trei operații:

- Publicare
- Descoperire
- Conectare

Operația de publicare este operația prin care furnizorul de servicii înregistrează serviciul în registrul de servicii.

Operația de descoperire constă în găsirea de către solicitant a serviciului în cadrul registrului de servicii pe baza unor criterii cum ar fi tipul serviciului, caracteristicile furnizorului.

Operația de conectare are rolul de a conecta furnizorul de servicii cu solicitantul printr-o legătură de tipul client-server, care poate fi dinamică sau statică. [20]

Aceasta arhitectură folosește numeroase protocoale, grupate pe patru niveluri, într-o stivă de protocoale:

- ❖ **Nivelul de transport** al serviciului: este responsabil cu transportul mesajului între consumator și furnizor prin intermediul unor protocoale de transport cum ar fi HyperText Transfer Protocol(HTTP), Simple Mail Transfer Protocol (SMTP) sau File Transfer Protocol (FTP).
- ❖ **Nivelul de mesaje XML**: este responsabil de codarea mesajelor într-un format XML prin intermediul unor protocoale precum eXtensible Markup Language-Remote Procedure Call (XML-RPC) sau Simple Object Acces Protocol/Service Oriented Architecture Protocol (SOAP).

- ❖ **Nivelul de descriere** al serviciului: este responsabil cu descrierea interfeței serviciului cu ajutorul limbajului Web Service Description Language (WSDL).
- ❖ **Nivelul de descoperire** al serviciului: se ocupă cu centralizarea serviciilor într-un registru comun și cu descoperirea serviciului prin intermediul Universal Description, Discovery and Integration (UDDI).

3.1.2.3.2. Arhitectura de tip REST (Representational State Transfer)

Arhitectura de tip REST este o arhitectura orientată pe resurse, apărută ca urmare a complexității destul de ridicate a arhitecturii de tip RPC și construită pe baza componentelor care au bazele WEB-ului. Astfel, această arhitectură aplică arhitectura web-ului asupra serviciilor web, iar componentele folosite sunt protocolul HyperText Transfer Protocol (HTTP), limbajul eXtreme Markup Language (XML) și identificatorul unei resurse (URI). [20]

Această arhitectură poate fi caracterizată prin:

- Datele asupra cărora serverul operează la cererea clientului aflate în URI
- Operația făcută de server asupra datelor, descrisă de către metoda HTTP.

REST poate fi descrisă din punct de vedere tehnic prin câteva componente de bază:

- Resursa
- URI
- Reprezentarea
- Interfața uniformă

În REST resursa poate fi reprezentată de orice informație ce poate fi stocată sub formă octeți.

Orice resursă este caracterizată de nume și de adresă, grupate sub forma unui identificator, numit Uniform Resource Identifier (URI). Uneori URI-ul unei resurse se identifică cu URL-ul (Uniform Resource Locator) acesteia, reprezentând adresa la care resursa se găsește.

Resursa poate fi caracterizată de mai multe URI-uri.

Principiul de bază al REST este reprezentat de către interfața uniformă, care este caracterizată de metodele furnizate de protocolul HTTP, pentru operațiile ce pot fi efectuate asupra unei resurse:

- Primirea unei resurse: HTTP GET
- Crearea unei noi resurse: HTTP POST
- Modificarea unei resurse existente: HTTP PUT
- Ștergerea unei resurse existente: HTTP DELETE

Reprezentarea unei resurse se face printr-un flux de octeți.

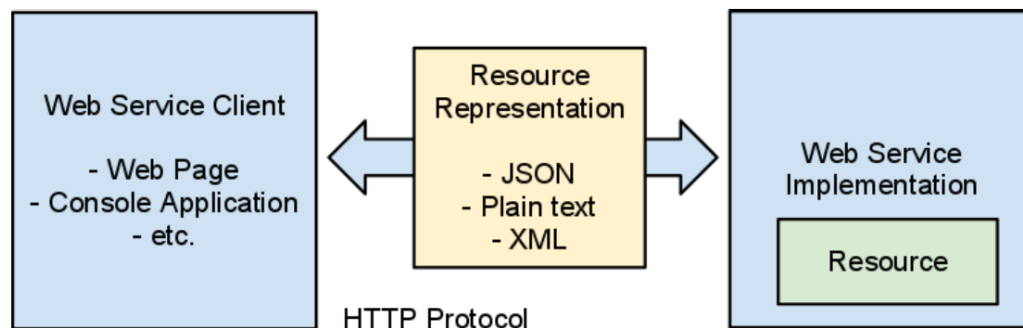


Figura 11: Arhitectura de tip REST[19]

Diferența majoră dintre cele două arhitecturi este aceea că pentru cea de tip RPC au fost create noi protocoale cum ar fi XML-RPC sau SOAP, spre deosebire de REST, unde se folosește HTTP.

3.1.3. Tehnologiile folosite pentru partea de date

Pentru această parte voi folosi Liquibase și MySQL.

3.1.3.1. Liquibase

Liquibase este o bibliotecă open-source cu ajutorul căreia sunt gestionate și aplicate mai ușor modificările într-o bază de date, prin intermediul unor fișiere XML.

Liquibase execută modificările apărute în acest fișier XML.

Pentru a gestiona aceste fișiere, la prima rulare, Liquibase crează două tabele în baza de date:

- **Databasechangelog** – păstrează modificările apărute în bază
- **Databasechangeloglock** – se asigură că nu se încearcă modificarea bazei de date de mai multe persoane în același timp.

3.1.3.2. MySQL

MySQL este unul dintre cele mai populare Sisteme de Gestionare a Bazelor de Date Relationale.

Principalele caracteristici care l-au făcut popular sunt:

- Este open-source

- Poate lucra pe mai multe sisteme de operare si cu numeroase limbaje de programare
- Suportă baze de date mari
- Este customizabil
- Este rapid

3.1.4. Mediile și instrumentele folosite pentru dezvoltare

Pentru dezvoltarea voi folosi IDE-ul IntelliJ IDEA. Pentru baza de date, voi folosi utilitarul XAMPP, care oferă un sistem de gestiune a bazelor de date MySQL.

3.1.4.1. IntelliJ IDEA

IntelliJ IDEA e un mediu de dezvoltare integrat (IDE) dezvoltat de către Jet Brains, folosit pentru dezvoltarea aplicațiilor Java.

Poate fi integrat cu diverse tool-uri sau tehnologii și suportă numeroase limbaje de programare.

Principalele limbaje de programare suportate sunt: Java, PHP, JavaScript, Haskell, Python.

Oferă suport pentru numeroase framework-uri și tehnologii printre care Android, JUnit, Maven, Gradle, JSP, JSF, EJB, Spring.

Principalele sisteme de versionare suportate sunt:

- GitHub
- Subversion
- Perforce
- CVS

3.1.4.2. XAMPP

XAMPP este un pachet de programe open-source ce oferă utilizatorilor câteva instrumente ajutătoare cum sunt următoarele:

- Apache HTTP Server
- MySQL database
- Apache Tomcat
- Interpretoare pentru limbaje de scripting precum PHP si Perl.

4. Considerente legate de dezvoltarea aplicației

4.1. Arhitectura aplicației

Prin arhitectura unei aplicații software se înțelege structura acesteia, adică modul în care sunt legate părțile componente, pentru a putea comunica astfel încât să formeze acel tot unitar, care este sistemul software.

Unul dintre importante roluri în realizarea arhitecturii unui sistem îl joacă tehnologiile folosite și modul în care acestea interacționează pentru a pune în mișcare sistemul.

4.1.1. Structura unei aplicații Java web și modul de funcționare a acesteia

O aplicație web funcționează pe principiul client-server. Clientul trimite o cerere către server reprezentată de o pagină HTML statică, prin intermediul protocolului HTTP, și primește ca răspuns tot o pagină HTML tot prin intermediul protocolului HTTP, care este randată de către browser. Problema care apare însă este aceea că conținutul paginii HTML este unul static.

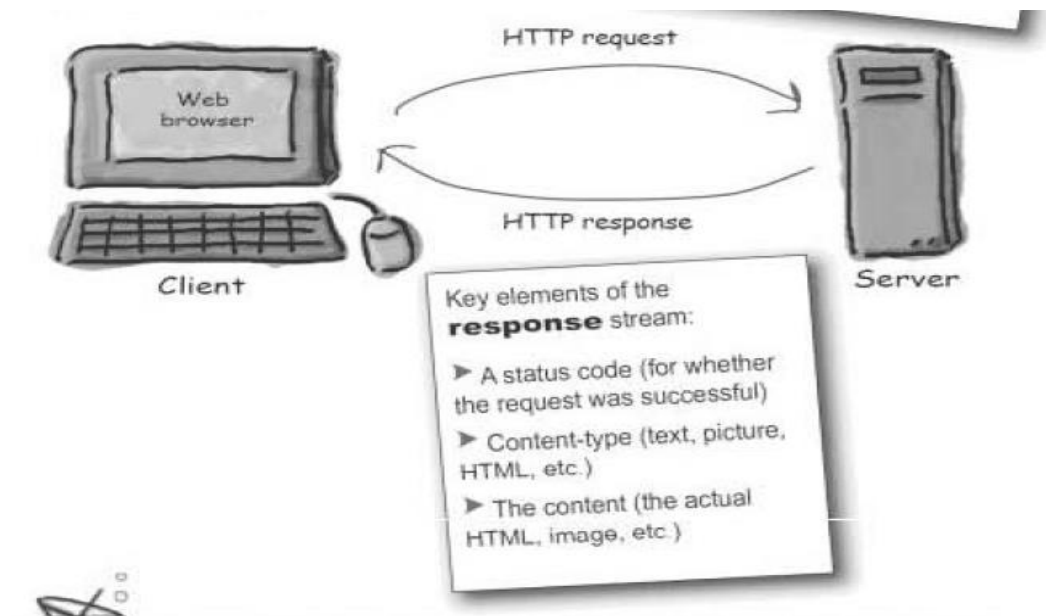


Figura 12: Modelul client-server [21]

Soluția găsită a fost aceea de a mai folosi o aplicație ajutătoare, către care clientul să trimită cererea. Astfel, clientul trimite o cerere către un URL la care se găsește aplicația ajutătoare, serverul observă că cererea este trimisă pentru acel program ajutător, pe care îl lansează în execuție, trimițându-i și parametrii primiți de la metoda de GET sau POST. După ce rezolvă cererea, programul ajutător creează pagina HTML, o trimite către server și își încheie execuția. Pagina HTML este transmisă mai departe către client, care o randează în browser.

Programul ajutător poartă numele de Common Gateway Interface (CGI). În mod normal, programele CGI sunt scrise în Perl, PHP, Python sau chiar C. Limbajul Java a venit însă cu o alternativă și anume Servlets.

Spre deosebire de CGI-uri, Servlets au câteva necesități ce trebuie îndeplinite:

- La primirea unei cereri, Servletul trebuie instanțiat
- Metodele doPost() și doGET() trebuie apelate
- Ciclul de viață al unui Servlet trebuie gestionat

Deoarece Servlets nu au o metoda main(), ei trebuie gestionați tot de o aplicație Java. Astfel a apărut noțiunea de Container. Astfel, Servletul este deployat în acest Container și este gestionat de către acesta, Containerul având rolul de a îndeplini toate necesitățile enumerate mai sus.

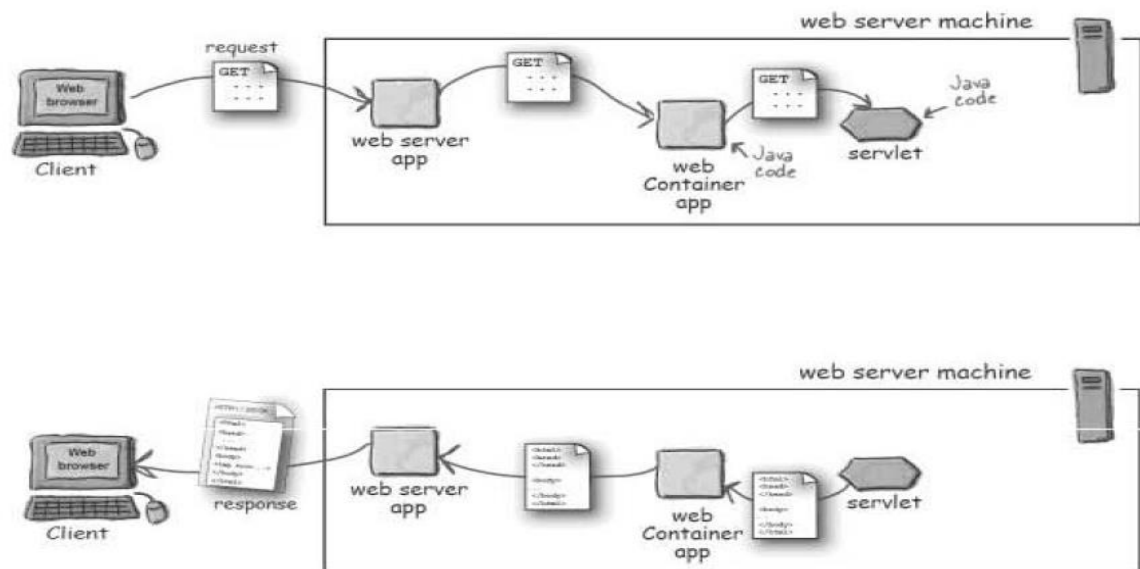


Figura 13: Modelul client-server folosind un container web[22]

Principalele facilități oferite de către Containerul web sunt:

- Gestionarea ciclului de viață al Servletilor
- Oferă suport pentru multithreading, creând câte un thread pentru fiecare cerere venită la Servlet

Etapele ce au ca efect gestionarea unei cereri de către un Servlet sunt următoarele:

- Clientul trimite cererea prin accesarea adresei la care se găsește Servletul

- Containerul vede cererea trimisă către Servlet și creează câte un obiect de tipul `HttpServletRequest` și `HttpServletResponse`.
- Containerul găsește Servletul pentru care a fost trimisă cererea, creează câte un thread pentru această cerere și transmite obiectele create către Servlet, apelează metoda `service()` din Servlet.
- În funcție de tipul cererii, metoda `service()` apelează `doPost()` sau `doGet()`
- Metoda `doGet()` generează pagina de răspuns
- După ce threadul își încheie activitatea, containerul convertește obiectul de tip răspuns într-un HTTP response, îl trimite la client și distruge obiectele de cerere și de răspuns.

Principalul Container folosit este Apache Tomcat. Acesta oferă suport pentru tehnologiile Java EE prin intermediul Apache TomEE.

Pentru a putea fi deployat pe containerul de aplicație, codul Java trebuie împachetat într-un format adecvat. Acest lucru este posibil prin intermediul unui manager de proiect, produs de organizația Apache, numit Maven. Acesta folosește o abordare declarativă. Astfel, prin intermediul unui model obiect al proiectului reprezentat de obicei de un fișier numit `POM.xml`, creăm o descriere a proiectului și a atributelor acestuia (biblioteci, framework-uri folosite). Pe baza acestei descrieri, este generat de către Maven un fișier `.war`, care este deployat de către containerul de web.

Unul dintre avantajele folosirii Spring Boot este acela că acesta vine cu o versiune de Tomcat integrată, fiind lansat în execuție simultan cu rularea aplicației.

4.1.2. Prezentarea bazei de date

Baza de date în care sunt stocate datele necesare aplicației a fost creată cu ajutorul platformei XAMPP.

Pentru etapa de proiectarea am ales următoarele entități:

- User → salvează datele despre utilizatorii acestei aplicații
- Authority → păstrează categoriile de utilizatori ai aplicației
- UserAuthority → tabela de legătură între tabela User și Authority
- Expense → reține costurile unei deplasări
- ExpenseType → reține tipurile de costuri ale unei deplasări
- BusinessTravelRequest → salvează detaliile unei deplasări
- Attachment → salvează fișierele atașate unei deplasări
- BusinessTravelRequestHistory → păstrează modificările apărute asupra unei deplasări
- DatabaseChangeLog → salvează modificările apărute asupra structurii bazei de date
- DatabaseChangeLogLock → se asigură că nu se execută operații asupra bazei de date simultan de către mai mulți utilizatori

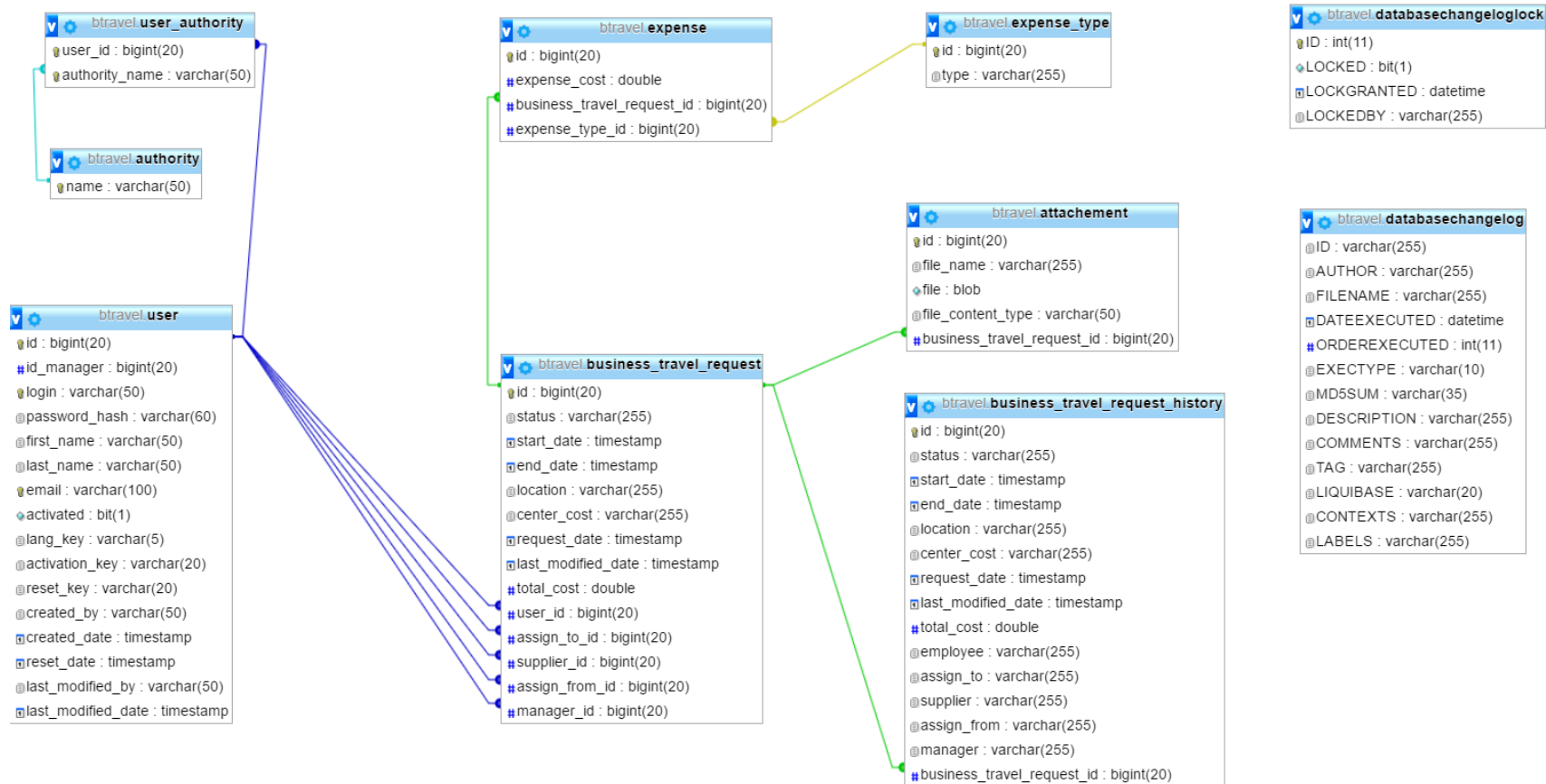


Figura 14: Structura bazei de date

Relațiile între entități:

| | User | Authority | UserAuthority | Expense | ExpenseType | BusinessTravelRequest | Attachment | BTRHistory |
|------------------------------|------|-----------|---------------|---------|-------------|-----------------------|------------|------------|
| User | | N:N | 1:N | | | 1:N | | |
| Authority | N:N | | 1:N | | | | | |
| UserAuthority | 1:N | 1:N | | | | | | |
| Expense | | | | | 1:N | 1:N | | |
| ExpenseType | | | | 1:N | | | | |
| BusinessTravelRequest | 1:N | | | 1:N | | | 1:N | 1:N |
| Attachment | | | | | | 1:N | | |
| BusinessTravelRequestHistory | | | | | | 1:N | | |

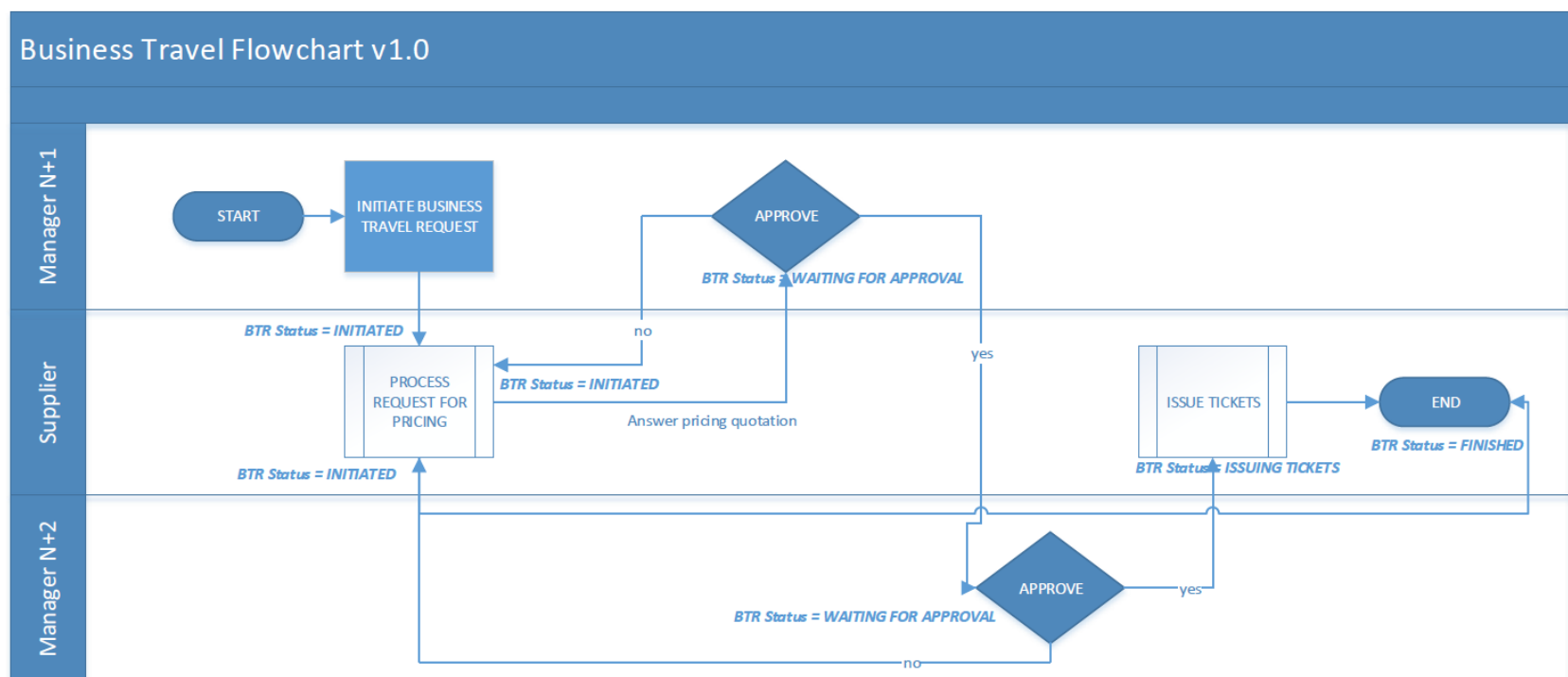


Figura 15: Diagrama de proces a aplicației

4.1.3. Diagrama de proces

Diagrama de process reprezintă modul în care poate fi vizualizat un proces pe etape sub forma unei scheme logice. Pașii din process sunt reprezentați sub forma unor forme geometrice conectate prin săgeți care indică următoarea etapă din cadrul procesului.

Unul dintre mediile cele mai folosite pentru crearea unor astfel de diagrame de procese este Visio, mediu oferit de către Microsoft.

Pe baza cerințelor funcționale am construit diagrama de process (figura 15).

4.2. Implementarea aplicației

Aplicația dezvoltată de mine este una de tipul Single Page Application(SPA), adică o aplicație alcătuită dintr-o singură pagină web, în interiorul căreia sunt interschimbate diferite cadre.

Acest lucru a fost posibil și ușor de realizat cu ajutorul framework-ului AngularJS, folosind unul dintre conceptele de bază ale acestui framework, numit rutare.

Având la bază design patternul MVC, acest framework reușește să separe funcțiile în cadrul aplicației client. Într-o aplicație client, dezvoltată folosind AngularJS, modelul este reprezentat de către datele ce sunt randate de către view, view-ul este reprezentat printr-o pagina HTML (sau o parte dintr-o pagina HTML acolo unde este folosită rutarea), iar controllerul este reprezentat de scripturi javascript care prelucrează datele, salvându-le în model.

Un view poate avea asociat unul sau mai multe controllere și modele. Legătura între view și cele două componente se realizează prin intermediul unor componente AngularJS, numite directive.

Directivele sunt niște attribute folosite în interiorul tag-urilor HTML, care au roluri diverse în cadrul paginii web.

Aplicația actuală, este alcătuită din pagina principală index.html, care este legată la modulul app.js prin intermediul directivei ng-app.

În index.html sunt declarate toate scripturile folosite în cadrul aplicației. Pe lângă acestea, această pagină mai conține și elementele fixe cum ar fi bara de navigare și footerul paginii. Acestea sunt reprezentate prin niste bucăți de cod HTML, numite template-uri.

Legătura dintre aceste template-uri și pagina principală este realizată prin intermediul directivei ui-view, prin intermediul căreia este încărcat main-ul paginii.

Fiind o aplicație modulară, în modulul principal app.js sunt declarate alte module folosite în cadrul acesteia.

Principalele funcționalități oferite de aplicație sunt:

- Înregistrarea unui cont de utilizator

- Autentificarea si editarea contului
- Gestionarea utilizatorilor aplicației de către administratorul platformei
- Gestionarea tipurilor de costuri de către administratorul platformei
- Crearea și gestionarea cererilor de deplasare
- Auditul cererilor de deplasare

Bara de navigare va avea diferite meniuri în funcție de tipul utilizatorului logat.

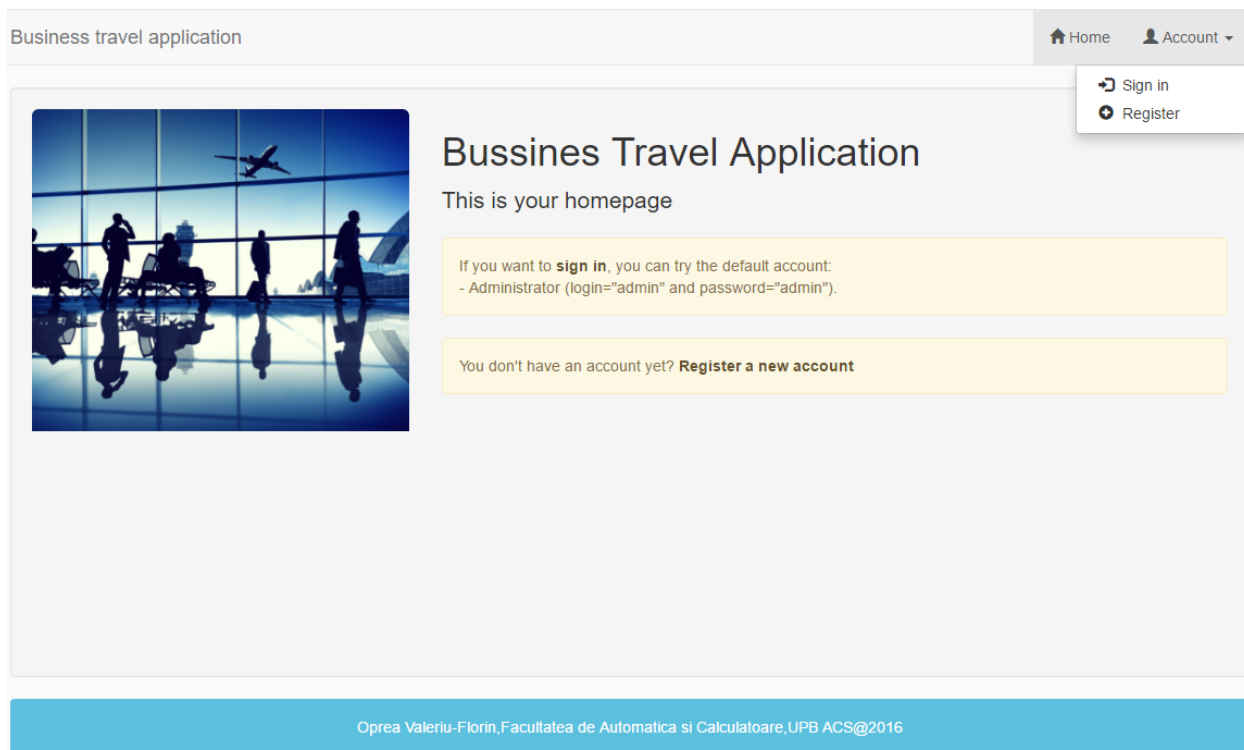


Figura 16: Ecranul principal al aplicației

În ecranul principal, sunt vizibile link-urile către primele două funcționalități ale aplicației:

- Autentificarea unui utilizator
- Crearea de cont nou

Prin accesarea uneia dintre cele două opțiuni, conținutul principal al paginii se va schimba, un alt template fiind încărcat.

Acest lucru este realizat în AngularJS prin intermediul conceptului de rutare. Rutarea constă în înlocuirea unei părți a unei pagini cu un alt template, care se găsește la o altă adresă.

La nivel de cod, rutarea se realizează prin intermediul directivei `ui-sref`, care primește o valoare ce va fi transmisă către controller-ul de configurare asociat template-ului curent. Acesta

primește template-ul ce se dorește a fi randat în pagina principală, are asociat un controller de configurare, prin intermediul căruia sunt returnați anumiți parametri ai service-ului cu ajutorul căruia se realizează rutarea.

Service-urile sunt niște obiecte prin intermediul cărora sunt apelate niște metode ce ajută la simplificarea dezvoltării și efectuarea de operații necesare.

Unul dintre cele mai importante service-uri este \$http cu ajutorul căruia sunt executate diferite cereri către server.

Pentru rutare, principalul service folosit este \$stateProvider.

Partea de server ce deservește diversele cereri primite de la client, folosește tot o arhitectură de tipul MVC. Schimbul de date între front-end și back-end se realizează prin intermediul serviciilor web de tip REST. Astfel, cererile către server definesc anumite operații asupra resurselor. Acestea sunt trimise sub forma de JSON, la o anumită adresă.

Controller-ul Java, conține metode capabile să deservească aceste cereri. Fiecare dintre metodele existente în această componentă, este adnotată cu adnotarea specifică controllerelor folosite în Spring MVC @RequestMapping, în care sunt specificate valorile anumitor atribute ale cererii realizate de către client. Astfel, principalele atribute specificate sunt:

- Calea la care se găsește resursa (Ex: value="/users")
- Tipul metodei REST (Ex: method = RequestMethod.POST)
- Formatul în care se găsește resursa la acea cale (produces = MediaType.APPLICATION_JSON_VALUE)

Cererile de la client pot fi de patru tipuri:

- a) Cerere de solicitare a unei resurse, efectuată prin intermediul metodei HTTP GET
- b) Cerere de creare a unei resurse, efectuată prin intermediul metodei HTTP POST
- c) Cerere de modificare a unei resurse existente prin intermediul metodei HTTP PUT
- d) Cerere de stergere a unei resurse existente prin intermediul metodei HTTP DELETE

Cererile de solicitare, modificare sau stergere a unei resurse, necesită uneori transmiterea de parametri pentru identificarea acestora în baza de date. Acești parametri pot fi transmiși în mai multe moduri către controllerul de pe server. De cele mai multe ori aceștia sunt transmiși în cale.

Există două modalități de a transmite acești parametri în cale:

- Prin specificarea numelui parametrului (Ex: ...?id="10").

Când sunt transmiși în acest mod, aceștia nu apar în calea de pe server și sunt preluați în mod explicit, folosind adnotarea @RequestParam("id").

- Fără a specifica numele parametrului (Ex: .../10).

Când sunt transmiși în acest mod, în calea de pe server apare numele parametrului între {}, iar aceștia sunt preluați cu ajutorul adnotării @PathVariable.

O altă adnotare folosită pentru preluarea anumitor date de la client, este @RequestBody, aceasta putând prelua tot JSON-ul, fiind folosită în general cu metoda POST.

Pe lângă controller și model, mai sunt folosite niște elemente ajutătoare, reprezentate de niste clase care ce îndeplinesc anumite funcționalități. Astfel, pe lângă controllere, mai avem service-uri și repository-uri.

Repository-urile sunt clase Java folosite pentru a comunica cu baza de date, prin intermediul lor efectuându-se operațiile asupra acesteia.

Service-urile sunt clase Java în care de obicei este descrisă partea logică a părții de server.

Astfel datele sunt preluate de către controller prin intermediul metodei POST, prelucrate de către service-uri și salvate în baza de date prin intermediul metodelor scrise în repository-uri.

Acest flux este executat în ambele sensuri. Datele sunt preluate din baza de date prin intermediul repository-urilor, prelucrate de către service-uri și transmise mai departe către controller, de unde de obicei sunt preluate prin intermediul unor cereri de tip GET de către partea de client.

Legătura dintre aceste componente este realizată prin intermediul conceptului de bază al framework-ului Spring, injectarea dependențelor(Dependency Injection).

Astfel, în controller este injectată o instanță a service-ului, cu ajutorul cărei sunt apelate diferite metode din service. În service se injectează o instanță de repository, prin intermediul căreia sunt utilizate diverse metode din repository.

4.2.1. Înregistrarea unui cont de utilizator

Una dintre funcționalitățile de bază care ar trebui să se regăsească în orice aplicație este aceea de înregistrare a unui cont de utilizator.

Astfel, utilizatorul aplicației, când dorește să înregistreze un cont, trebuie să completeze niște câmpuri obligatorii cum sunt cele de nume utilizator, email și parolă. După ce completează aceste date într-un formular, prin apăsarea butonului de register existent în interfață, le trimite mai departe către server pentru a fi procesate.

La nivel de cod, această funcționalitate necesită lucru atât pe partea de client cât și cea de server.

Partea de client este compusă din template-ul register.html care descrie formularul ce trebuie completat de către utilizator. Datele sunt preluate în controllerul de angularJs, register.controller.js. Acesta are injectat service-ul Auth, prin intermediul căruia este realizată cererea de tip POST către server și este primit răspunsul acestei cereri. Astfel, datele completate de către utilizator în formular sunt trimise către server.

Business travel application

HomeAccount

Registration

Username

E-mail

New password

Password strength:

New password confirmation

Register

If you want to **sign in**, you can try the default account:
- Administrator (login="admin" and password="admin").

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 17: Formularul de înregistrarea a unui cont în aplicație

Controller-ul din partea de server, `AccountResource.java`, este reprezentat de o clasă, adnotată cu adnotarea `@RestController`, care arată faptul că sunt folosite serviciile REST împreună cu Spring MVC. Acesta conține metode prin intermediul cărora sunt executate operații asupra informațiilor despre userul creat, reprezentat de clasa `UserDTO`.

Metoda `registerAccount`, creează pe baza informațiilor primite de la client noul utilizator, pe care îl inserează în baza de date, prin intermediul instanței de service injectată, în prealabil făcându-se verificarea ca email-ul să fie valid.

În cazul în care adresa de email este validă (nu mai există în baza de date), un email este trimis la adresa de email primită, ce conține un link, prin intermediul căruia, viitorul utilizator își va activa contul. Trimiterea emailului se realizează prin apelul metodei `sendActivationEmail()`, conținută de clasa `MailService`, de către instanța injectată în controller. Această metoda returnează statusul cererii. În cazul în care înregistrarea a avut loc, statusul returnat va fi `CREATED(201)`, altfel `BAD REQUEST(401)`. La înregistrare, va fi generată o cheie unică, care va fi salvată în baza de date, împreună cu celelalte date despre utilizator.



Figura 18: Emailul primit

Aceasta cheie va fi conținută de url-ul primit pe email, iar prin intermediul ei, clientul realizează o cerere de tip GET către server. Cererea este rezolvată de către metoda activateAccount(), care preia din URL cheia de activare, caută utilizatorul în baza de date prin intermediul acelei chei, îi activează contul prin setarea proprietății activated cu true, modifică campul cheii, făcându-l null, iar apoi salvează utilizatorul în bază cu noile proprietăți.

În cazul în care cererea s-a realizat cu success, metoda activateAccount() va returna statusul OK, reprezentat prin codul 200. În caz contrar, va fi returnat statusul INTERNAL SERVER ERROR, reprezentat prin codul 500.

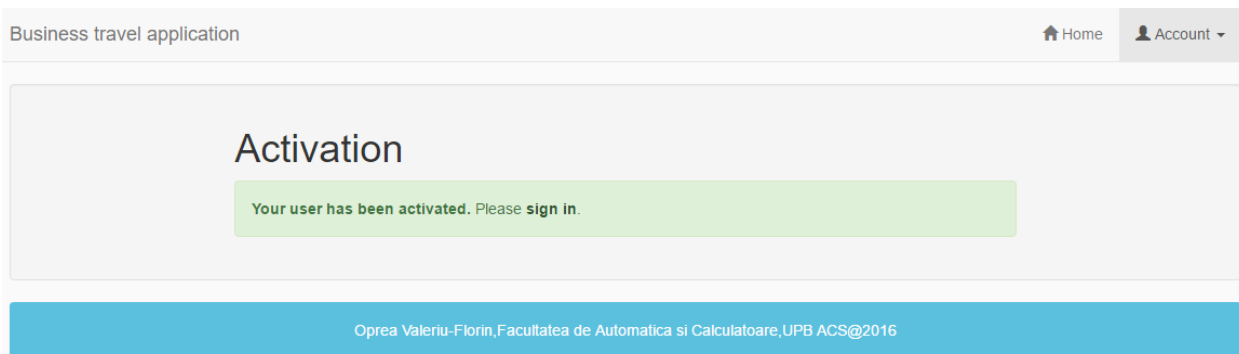


Figura 19: Ecranul dupa activarea contului

4.2.2. Autentificarea si editarea contului

Cea de-a doua funcționalitate de bază a unei aplicații este aceea de autentificare.

Pentru autentificare, utilizatorul, completează un formular ce conține câmpul username și cel de parolă, apoi le trimite către server prin apăsarea butonului Sign in.

Business travel application

Home Account

Sign in

Username

Password

☒ Remember me

Sign in

Did you forget your password?

You don't have an account yet? [Register a new account](#)

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 20: Formularul de autentificare

La nivel de cod, datele sunt preluate din formularul existent în template-ul login.html de către controller-ul login.controller.js. Prin intermediul unui \$scope, sunt trimise mai departe către service-ul Auh, care la rândului lui le trimite către service-ul AuthServerProvider, care expediază o cerere HTTP de tip POST către server prin intermediul service-ului \$http, ce conține username-ul și parola.

Daca datele trimise de client corespund cu cele existente în baza de date, atunci, se realizează autentificarea utilizatorului.



Bussines Travel Application

This is your homepage

You are logged in as user "admin".

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 21: Administratorul logat

După autentificare, utilizatorul are posibilitatea de a-și updata informatiile contului, de a-și modifica parola.

Modalitatea este aceeași ca și la crearea contului, singurele diferențe fiind reprezentate de formulare și de metodele ce deserveșc aceste cereri pe partea de server.

Pentru modificarea parolei, este folosită metoda `changePassword()` din controller.

```
@RequestMapping(value = "/account/change_password",
    method = RequestMethod.POST,
    produces = MediaType.APPLICATION_JSON_VALUE)
@Timed
public ResponseEntity<?> changePassword(@RequestBody String password) {
    if (!checkPasswordLength(password)) {
        return new ResponseEntity<>("Incorrect password", HttpStatus.BAD_REQUEST);
    }
    userService.changePassword(password);
    return new ResponseEntity<>(HttpStatus.OK);
}
```

Figura 22: Metoda de schimbare a parolei - controller

Ea primește ca parametru de la client un `String` pe care îl trimite în metoda `change` din service și returnează statusul cererii.

În service, este căutat utilizatorul logat, noua parolă este encodată, este modificat utilizatorul curent, iar apoi acesta este salvat în baza de date prin intermediul instanței de repository.

Business travel application

Home Business travel Account

User settings for [florinoprea93]

First Name
Florin

Last Name
Oprea

E-mail
florinoprea93@gmail.com

Language
English

Save

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 23: Formularul pentru editarea contului

4.2.3. Gestionarea platformei de către administratorul platformei

Fiecare utilizator are asociat cel puțin un rol în cadrul aplicației. Astfel, fiecare utilizator este un angajat (ROLE_EMPLOYEE). Pe lângă acest rol initial, angajatul mai poate avea rol de manager (ROLE_MANAGER), administrator (ROLE_ADMIN) sau furnizor (ROLE_SUPPLIER). În funcție de rolul asociat, fiecare angajat are anumite drepturi asupra meniurilor aplicației și poate opera diverse modificări în cadrul acesteia.

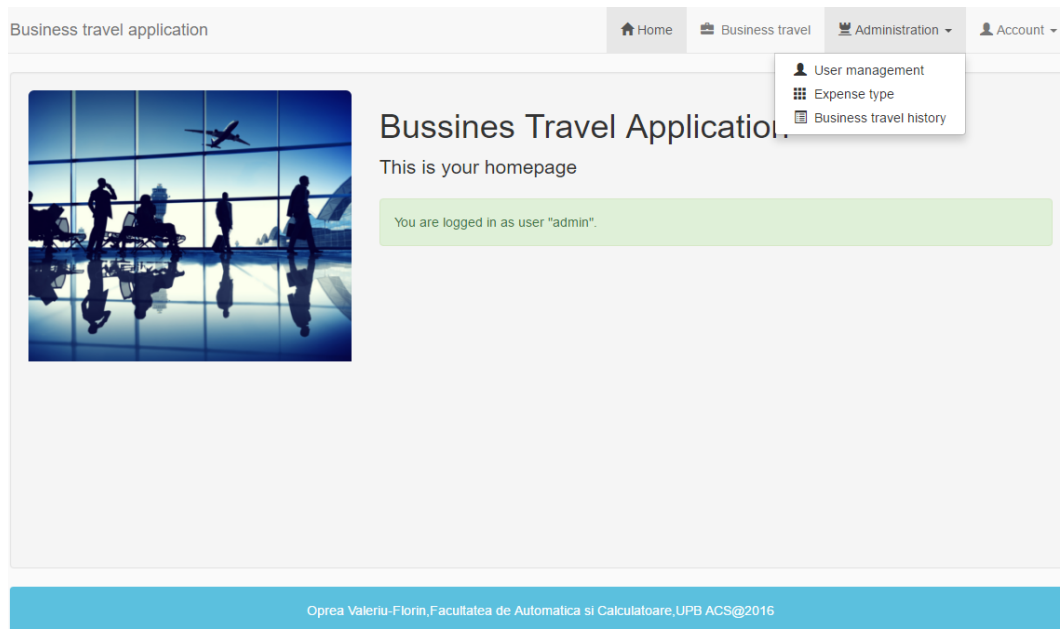


Figura 24: Atribuțiile administratorului

Astfel, administratorul poate gestiona utilizatorii aplicației, tipurile de costuri pentru deplasări, cererile de deplasare. Pe lângă acestea, acesta mai poate vedea și un istoric al tuturor cererilor de deplasare efectuate.

4.2.3.1. Gestionarea utilizatorilor aplicației

Principalul rol al administratorului este acela de a gestiona utilizatorii aplicației.

Acesta poate efectua diverse operații asupra utilizatorilor aplicației:





































- Creare
- Editare
- Stergere
- Vizualizare

Business travel application

Home Business travel Administration Account

Users

Create a new user

| ID | Manager | Login | Email | | Profiles | Created date | |
|----|----------|---------------|-------------------------|-------------|--------------------------------|----------------|---|
| 1 | | system | system@localhost | Activated | ROLE_ADMIN ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 2 | | anonymousUser | anonymous@localhost | Deactivated | | 20/06/16 18:40 |    |
| 3 | | admin | admin@localhost | Activated | ROLE_ADMIN ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 4 | manager1 | user | user@localhost | Activated | ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 5 | manager1 | manager1 | manager1@localhost | Activated | ROLE_MANAGER ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 6 | manager1 | manager2 | manager2@localhost | Activated | ROLE_MANAGER ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 7 | | supplier1 | supplier1@localhost | Activated | ROLE_SUPPLIER ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 8 | | supplier2 | supplier2@localhost | Activated | ROLE_SUPPLIER ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 9 | | user2 | user2@localhost | Deactivated | ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 10 | manager2 | user3 | user3@localhost | Activated | ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 11 | manager2 | user4 | user4@localhost | Activated | ROLE_EMPLOYEE | 20/06/16 18:40 |    |
| 17 | manager1 | florinoprea93 | florinoprea93@gmail.com | Activated | ROLE_EMPLOYEE | 25/06/16 18:25 |    |

« < 1 > »

Figura 25: Ecranul de administrare a utilizatorilor existenți în aplicație

Pentru crearea unui utilizator nou în aplicație, administratorul va apăsa pe butonul “Create user” și va completa formularul ce i se va deschide sub forma de popup, iar după completarea tuturor câmpurilor, aceasta se va închide, iar în pagina se va updata numărul utilizatorilor cu cel creat.

Noul utilizator va primi un email, la adresa completată în formular prin care va putea să își seteze o parolă nouă pentru contul creat.

The screenshot displays a web application interface for user management. A modal window titled "Create or edit a user" is open, allowing the creation of a new user. The form contains the following fields:

- Manager:** A dropdown menu.
- First name:** A text input field.
- Email:** A text input field with a required field message ("This field is required.").
- Language:** A dropdown menu.
- Login:** A text input field with a required field message ("This field is required.").
- Last name:** A text input field.
- Profiles:** A dropdown menu showing roles: ROLE_EMPLOYEE, ROLE_ADMIN, ROLE_MANAGER, and ROLE_SUPPLIER.

At the bottom of the modal are "Cancel" and "Save" buttons. The background shows a "Users" table with columns for ID, Manager, and user details. The table lists several users, including suppliers and regular users, with their roles and activation status.

| ID | Manager | First name | Last name | Email | Language | Profile | Status | Created At |
|----|----------|---------------|-----------|-------------------------|----------|------------------------------|-----------|----------------|
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | manager1 | | | | | | | |
| 5 | | | | | | | | |
| 6 | manager1 | | | | | | | |
| 7 | | supplier1 | | supplier1@localhost | | ROLE_SUPPLIER, ROLE_EMPLOYEE | Activated | 20/06/16 18:40 |
| 8 | | supplier2 | | supplier2@localhost | | ROLE_SUPPLIER, ROLE_EMPLOYEE | Activated | 20/06/16 18:40 |
| 9 | | user2 | | user2@localhost | | ROLE_EMPLOYEE | Activated | 20/06/16 18:40 |
| 10 | manager2 | user3 | | user3@localhost | | ROLE_EMPLOYEE | Activated | 20/06/16 18:40 |
| 11 | manager2 | user4 | | user4@localhost | | ROLE_EMPLOYEE | Activated | 20/06/16 18:40 |
| 17 | manager1 | florinoprea93 | | florinoprea93@gmail.com | | ROLE_EMPLOYEE | Activated | 25/06/16 18:25 |

Figura 26: Adaugarea unui nou utilizator

La nivel de cod, evenimentul de apăsare a butonului este tratat de către directiva ui-sref, prin intermediul căreia, se va trimite către controller-ul de configurare, user-management.js, calea la care se va deschide modal-ul reprezentat de template-ul user-management-dialog.html.

Acest controller, va seta de asemenea anumite detalii de configurare pentru acest nou template, cum ar fi controller-ul asociat, calea unde se găsește acest template, sau chiar valorile anumitor câmpuri din cadrul formularului.

Datele completate în formular sunt preluate de către controller-ul asociat acestui template, `user-management-dialog.controller.js`, prin cu ajutorul service-ului `$scope`, care, preia datele din modelul template-ului (fiecare câmp are asociat un model folosind directiva `ng-model`, prin intermediul căroră datele din template-ul HTML sunt vizibile în controller-ul asociat → conceptul de `data-binding`). Astfel, datele sunt trimise prin intermediul service-ului `user.service.js` către server. Trimiterea se face prin intermediul apelului funcției `save`, ce are asociată în cadrul service-ului metoda `POST`, a protocolului `HTTP`, sub forma de `json`.

La nivelul de server, controller-ul `UserResource.java` este cel care gestionează toate operațiile care se efectuează asupra datelor despre utilizatori. Metoda `createUser()` este cea care realizează înregistrarea noului utilizator al aplicației. Modul de implementare este același ca și în cazul înregistrării individuale a utilizatorului.

În formularul de adaugare, primul câmp este reprezentat de un dropdown, care conține toți utilizatorii care au rol de manager. Popularea acestui dropdown am realizat-o printr-un intermediul unui service auxiliar, `authority.service.js`, în care am descris câteva metode ce execută niște cereri către server, pentru a aduce anumite date în funcție de anumite criterii.

```
getAllManagers: function(){
    return $http.get('api/users/managers').then(function(response) {
        return response.data;
    });
},
```

Figura 27: Cerere către server pentru aducerea tuturor managerilor

La nivel de server, această cerere este deservită de către metoda `getAllManagers()`, din controller-ul `UserResource.java`, prin căutarea utilizatorilor după rolul îndeplinit. Fiecare utilizator din lista, are asociate trei butoane, prin intermediul căroră, se pot efectua operații de vizualizare completă a detaliilor, editare sau stergere.

Prin apăsarea butonului de editare, se va deschide același modal în care au fost introduse datele la crearea unui utilizator, diferența fiind aceea că de această dată, formularul vine populat cu detaliile utilizatorului ce se dorește a fi editat. Acest lucru este realizat în cadrul controller-ului de configurare, unde spre deosebire acțiunea de creare, unde câmpurile erau setate cu valoarea `null`, se realizează o cerere către server pentru a primi detaliile utilizatorului cu `username-ul` specificat în calea cererii.

```
.state('user-management.edit', {
    parent: 'user-management',
    url: '/{login}/edit',
    data: {
        authorities: ['ROLE_ADMIN'],
    },
    onEnter: ['$stateParams', '$state', '$uibModal', function($stateParams, $state, $uibModal) {
        $uibModal.open({
            templateUrl: 'scripts/app/admin/user-management/user-management-dialog.html',
            controller: 'UserManagementDialogController',
            size: 'lg',
            resolve: {
                entity: ['User', function(User) {
                    return User.get({login: $stateParams.login});
                }]
            }
        }).result.then(function(result) {
            $state.go('user-management', null, { reload: true });
        }, function() {
            $state.go('^');
        })
    })
})
```

Figura 28: Modalitatea de încărcare a dialogului de editare a unui utilizator

Prin apăsarea butonului de vizualizare, în aplicație se va încărca template-ul user-management.detail.html.

Acesta va fi populat cu date primite în urma cererii efectuate în interiorul controllerului user-management.controller.js către server. Metoda getUser() a controller-ului UserResource.java este cea care va returna detaliile despre userul cautat. Aceasta caută în baza de date după username-ul utilizatorului și returnează toate detaliile despre el.

Business travel application

Home Business travel Administration Account

User "ipopescu"

| Field | Value |
|---------------|-----------------------|
| Manager | manager1 |
| Login | ipopescu |
| First name | Ion |
| Last name | Popescu |
| Email | popescu.ion@localhost |
| Activated | true |
| Language | en |
| Created by | |
| Created date | 25/06/16 19:59 |
| Modified by | admin |
| Modified date | 25/06/16 20:00 |
| Profiles | ROLE_EMPLOYEE |

Back

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 29: Ecranul de afișare a datelor despre utilizator

Prin apăsarea butonului de ștergere, se va deschide un modal cu două butoane, unul pentru anularea ștergerii, iar altul pentru confirmarea ștergerii.

Acest modal este controlat de controller-ul user-management-delete-dialog.controller.js. Cererea de ștergere către server este realizată prin intermediul service-ului user.service.js care conține funcția delete, ce are asociată metoda HTTP DELETE.

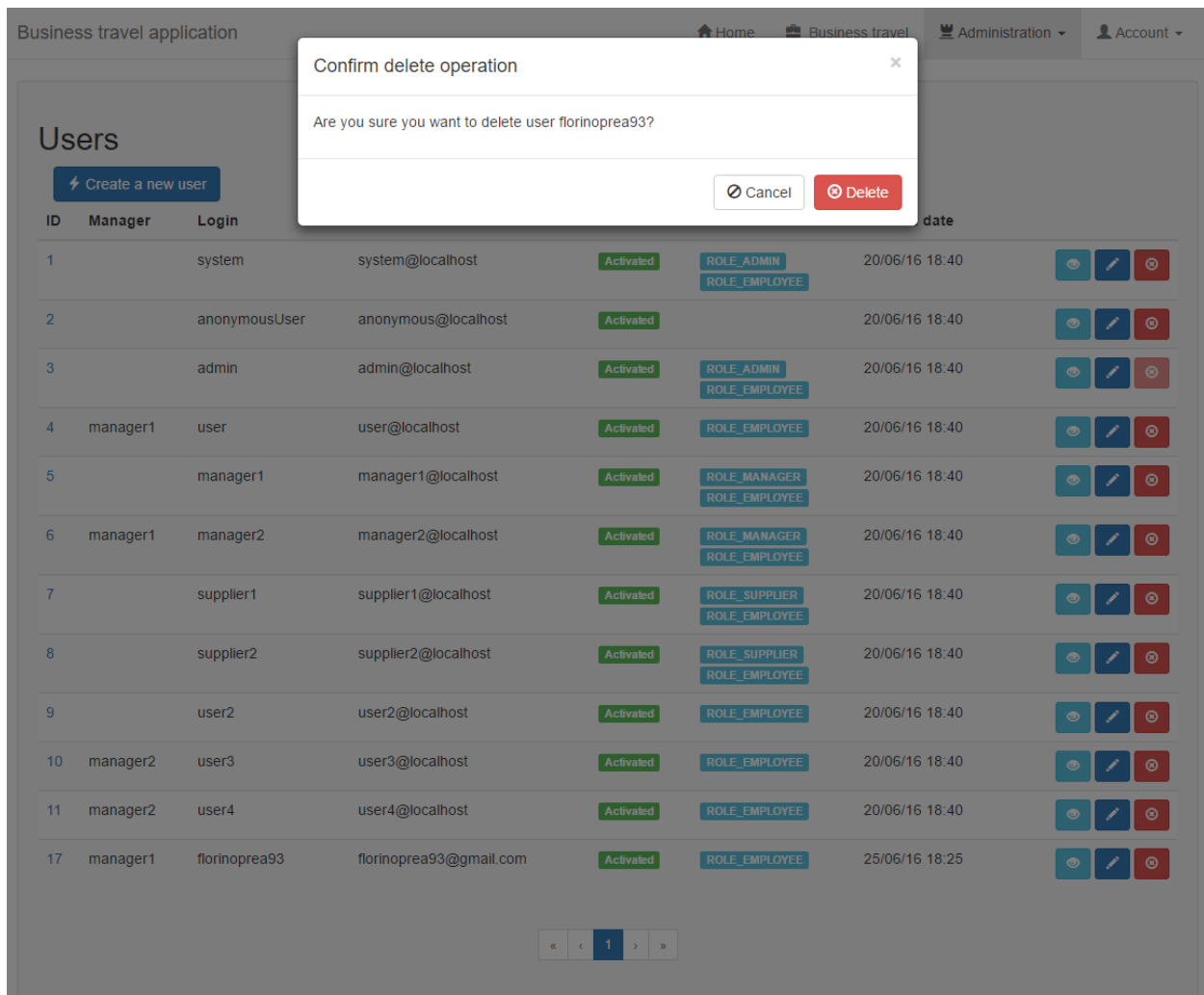


Figura 30: Modalul de confirmare sau anulare a ștergerii unui utilizator

O altă modificare pe care administratorul o poate aplica asupra conturilor celorlalți utilizatori este aceea de activare sau dezactivare a acestora, prin apăsarea etichetei aflate în dreapta adresei de email. Tratarea evenimentului de click pe acea etichetă este realizată la nivel de client prin intermediul directivei ng-click, fiind astfel apelată funcția setActive() creată în controllerul user-management.controller.js. Prin această funcție se realizează update-ul în baza de date, folosind o cerere de tip HTTP PUT.

Dezactivarea unui utilizator, conduce la imposibilitatea acestuia de se mai putea autentifica în sistem.

4.2.3.2. Gestionarea tipurilor de costuri

Un alt rol important pe care administratorul îl are în cadrul aplicației este acela de a administra tipurile de costuri ce pot apărea în cadrul unei deplasări.

El poate adăuga, edita sau șterge tipuri de costuri.

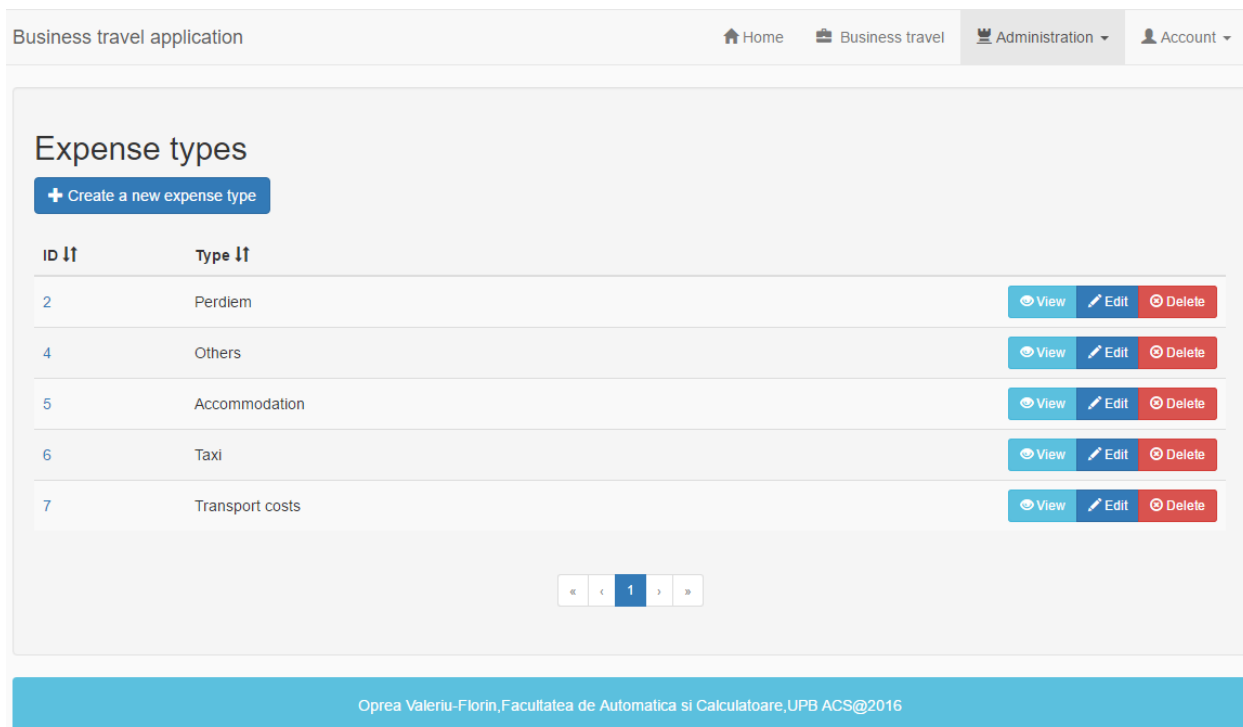


Figura 31: Ecranul tipurilor de costuri

Crearea, modificarea sau ștergerea unor tipuri de costuri se realizează ca și în cazul utilizatorilor.

Aceste tipuri vor fi vizibile în dropdown list-ul folosit de către furnizor pentru a adăuga costuri noi unei cereri de deplasare.

4.2.4. Ciclul unei cereri de deplasare și permisiunile utilizatorilor asupra acesteia

Până sa fie aprobată, o cerere de deplasarea trebuie să treacă prin diferite faze, faze la care utilizatorii aplicației sunt implicați activ.

Pornid de la premisa că un manager, dorește să-și trimită un angajat într-o călătorie , ciclul unei cereri de deplasare poate fi sumarizat în următoarele etape:

- a) Crearea cererii de deplasare de către manager și trimiterea acesteia către furnizor
- b) Adăgarea costurilor de deplasare de către furnizor și retrimiteră către manager
- c) Analizarea costurilor deplasării de către manager
- d) Dacă costurile sunt acceptate de către manager, cererea este trimisă către un manager superior în cazul în care acesta există.
- e) Dacă costurile sunt acceptate de către managerul superior, acesta trimite cererea către furnizor pentru achiziționarea biletelor, în caz contrar cererea se întoarce la furnizor pentru adăugarea altor costuri.
- f) Dacă cererea a fost acceptată și de managerul superior, atunci, furnizorul se ocupă de actele necesare deplasării, atașându-le la cerere, pentru a putea fi descărcate de către angajatul care trebuie să meargă în delegație

4.2.4.1. Crearea cererii de deplasare de către manager și trimiterea acesteia către furnizor

Pentru a crea o cerere de deplasare pentru un angajat pe care îl are în subordine, managerul se loghează în aplicație, intră în meniul “Business travel” existent în bara de start a aplicației. La acea pagină se vor găsi toate cererile create de către managerul logat.

Pentru a crea o nouă cerere, acesta apasă butonul “Create a new business travel request”, aflat în colțul din stânga sus al paginii. La apăsarea acestui buton, se va deschide un modal ce va conține formularul ce va conține detaliile cererii de deplasare.

The screenshot shows a web application interface for creating a business travel request. A modal window titled "Create or edit a business travel request" is open. It contains the following fields:

- Business travel start date:** 2016-06-30
- Business travel end date:** 2016-07-06
- Location:** France
- Center cost:** RO121BRD21212121
- Assign to:** supplier1
- Employee:** florinoprea93

At the bottom right of the modal, there are two buttons: "Cancel" and "Save".

Figura 32: Crearea unei cereri de deplasare

Principalele detalii ale cererii sunt reprezentate de data de start și de sfârșit a călătoriei, locația deplasării, angajatul pentru care este creată cererea de deplasare cât și furnizorul căruia îi va fi trimisă pentru a completa lista de costuri. Angajatul și furnizorul vor fi aleși din niște liste deja populate cu angajați. Lista cu angajați îi va conține doar pe cei aflați în subordinea managerului logat, iar cealaltă îi va conține doar pe cei cu rol de furnizor.

În ceea ce privește implementarea, partea de client este reprezentată ca și la celelalte pagini de template-uri HTML, randate în pagina principală de către controller-ul de configurare `businesstravelrequest.js`. Fiecare dintre template-uri va avea asociat un controller care asigură transportul datelor între client și server.

Pe lângă câmpurile din formular, există unele câmpuri care vor fi completate automat în funcție de acțiunile utilizatorului aplicației.

Unul dintre câmpurile esențiale pentru ciclul cererii, este statusul în care se află aceasta la un moment dat. O cerere poate exista într-unul dintre următoarele statusuri: `initiated`, `waiting for approval`, `issuing tickets`, `finished` sau `closed`. Aceste statusuri le-am reprezentat sub forma unei ENUM în java (`StatusBTR.java`). Acesta va fi setat automat la nivel de cod, nefiind necesară modificarea lui la nivel de interfață.

Deoarece la schimbarea statusului unei cereri, câmpurile `assign to` și `assign from` devin variabile, a fost nevoie de folosirea altor două câmpuri (`manager` și `supplier`), pentru a putea păstra datele fundamentale inițierii cererii (de către cine a fost realizată cererea și către cine a fost alocată).

Cele două câmpuri vor fi inițializate o singură dată în cadrul metodei care realizează salvarea cererii în baza de date.

4.2.4.2. Adăugarea costurilor de deplasare de către furnizor și retrimiteră către manager

După logarea în aplicație, fiecare furnizor poate vizualiza în meniul “Business travel” toate cererile în care a fost asignat de către manager.

Pentru a putea adăuga costuri, acesta trebuie să editeze cererile cu statusul “INITIATED”.

La apăsarea butonului de edit pe o anumită cerere se va deschide modalul folosit la crearea cererii de către manager, având câmpurile completate de către acesta. Pe lângă aceste câmpuri, în modal vor mai fi afișate un dropdown list ce va conține tipurile de costuri adăugate de administratorul aplicației, un câmp numeric unde vor fi introduse valorile pentru costuri și un tabel ce va conține costurile adăugate, împreună cu niște butoane pentru editare și stergere.

Noile elemente din interfață sunt ascunse în funcție de statusul cererii folosind directive `ng-if`. Acestea sunt controlate de un alt controller, specificat în directiva `ng-controller`. După completarea costurilor prin apăsarea butonului “Send expenses”, acestea sunt preluate de controller-ul specificat în directiva `ng-controller` și trimise către server pentru a fi prelucrate.

Business travel application

Home Business travel Account

Create or edit a business travel request

Business travel start date
2016-06-30

Business travel end date
2016-07-07

Location
France

Center cost
RO121BRD2121212121

Expenses €

Expense type

- Others
- Accommodation
- Taxi
- Transport costs

Expense cost
This field is required.

| ID | Expense type | Expense cost |
|----|--------------|--------------|
| 4 | Perdiem | 123 |

[Edit](#) [Delete](#)

[Send expenses](#) [Cancel](#) [Save](#)

Figura 33: Adăugarea costurilor pentru o cerere

La nivel de server, clasa service `BusinessTravelRequestService.java` conține metoda `calculateTotalCost()` care calculează costul total pentru o deplasare specificată prin intermediul id-ului, transmis ca parametru.

La salvarea modificărilor efectuate asupra cererii, se va salva automat și costul total calculat de aceasta metoda, iar statusul cererii va trece în “WAITING FOR APPROVAL”.

4.2.4.3. Acceptarea sau respingerea cererii de către manager

După adăugarea costurilor de către furnizor, managerul care a inițiat cererea poate vizualiza costurile, iar în funcție de acestea poate accepta sau respinge cererea. În cazul în care acceptă costurile primite atunci, cererea poate ajunge la un alt manager superior pentru aprobare în cazul în care acesta există, sau la furnizor pentru a se ocupa de actele necesare deplasării și de a le atașa la cererea respectivă. Statusul acesteia va fi modificat în “ISSUING TICKETS”. În caz

contrar, cererea se va intoarce la furnizor, pentru căutarea altor oferte, trecând în statusul “INITIATED”.

```

$scope.setAccepted = function (id,isAccepted) {
    BusinessTravelRequest.get({id:id}, function (result) {
        $scope.businessTravelRequest = result;
        if (userLoggedIn.idManager !== null) {
            if (isAccepted) {
                $scope.businessTravelRequest.status = "WAITING_FOR_APPROVAL";
                $scope.businessTravelRequest.assignTo = managerN2;
            } else {
                $scope.businessTravelRequest.status = "INITIATED";
                $scope.businessTravelRequest.assignTo = $scope.businessTravelRequest.supplier;
            }
        } else {
            if (isAccepted) {
                $scope.businessTravelRequest.status = "ISSUING_TICKETS";
                $scope.businessTravelRequest.assignTo = $scope.businessTravelRequest.supplier;
            } else {
                $scope.businessTravelRequest.status = "INITIATED";
                $scope.businessTravelRequest.assignTo = $scope.businessTravelRequest.supplier;
            }
        }
    });

    BusinessTravelRequest.update($scope.businessTravelRequest,function(){
        $scope.refresh();
    });
});

```

Figura 34: Funcția care gestionează acceptul costurilor sau refuzul lor la nivel de client

Acceptarea sau respingerea costurilor de către manager se face prin apăsarea pe eticheta corespunzătoare acțiunii dorite. Implementarea este asemănătoare cu cea de activare/dezactivare a unui utilizator de către administratorul platformei.

Business travel application
Home
Business travel
Account

Business travel requests

+ Create a new business travel request
Search for business travel request

| ID | Status | Business travel start date | Business travel end date | Location | Assign to | Total cost | |
|----|----------------------|----------------------------|--------------------------|----------|-----------|------------|---|
| 5 | Waiting for approval | Jun 30, 2016 12:00:00 AM | Jul 7, 2016 12:00:00 AM | France | manager1 | 156 | <div>Accept</div> <div>Reject</div> <div>View</div> <div>Edit</div> <div>Cancel</div> <div>Delete</div> |

«
1
»

Oprea Valeriu-Florin, Facultatea de Automatica si Calculatoare, UPB ACS@2016

Figura 35: Cererile de deplasare ale unui manager logat

4.2.4.4. Atasarea actelor necesare deplasării

Ultimul pas în acest ciclu este cel al atașării anumitor documente la cerere. Acest lucru e realizat de către furnizor. Upload-ul de fișiere este realizat în modalul de editare al cererii.

După realizarea upload-ului, statusul cererii devine “FINISHED”, documentele uploadate putând fi descarcate.

Business travel application

Create or edit a business travel request

Business travel start date: 2016-06-28

Business travel end date: 2016-07-07

Location: France

Center cost: RO121BRD2121212121

Expenses €

| Expense type | Expense cost |
|-----------------|--------------|
| Others | 233 |
| Taxi | 33 |
| Accommodation | 33 |
| Transport costs | 21 |

Attachments

Choose: INTERNER.txt

Upload: SUCCESS

Reset

| Filename | Actions |
|-------------------------|---------|
| FileUploadResource.java | |
| INTERNER.txt | |

Finish Cancel Save

Fig 36: Upload-ul documentelor

4.2.4.5. Permisunile utilizatorilor asupra cererilor de deplasare

În funcție de rolul îndeplinit în cadrul aplicației, utilizatorii au anumite drepturi asupra cererilor de deplasare.

Managerul are permisiunea de a vedea toate cererile create în cadrul aplicației, de a le edita sau chiar șterge.

Managerul poate crea o cerere nouă și poate vizualiza, edita sau șterge doar cererile create de el.

Furnizorul poate vizualiza și edita doar cererile care i-au fost asignate de către manager. Angajatul are doar dreptul de a vizualiza cererile care au fost create pentru el. La nivel de implementare am folosit, design pattern-ul Strategy.

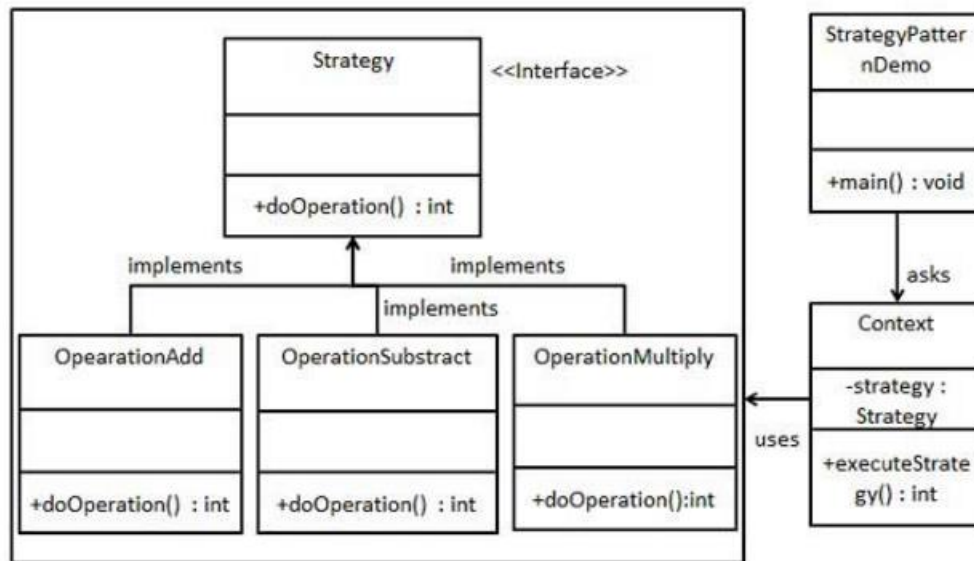


Figura 37: Strategy pattern[]

4.2.5. Auditul cererilor de deplasare

Fiecare aplicație software, trebuie să aibă implementată o funcționalitate prin care să înregistreze toate operările ce au avut loc la nivelul bazei de date.

Pentru a păstra modificările efectuate la nivelul bazei de date asupra cererilor de deplasare, am creat o noua entitate, accesibilă doar administratorului de sistem. Singura permisiune însă asupra acesteia este doar cea de vizualizare, datele fiind înregistrate în bază la nivel de server.

Astfel, pentru fiecare operație de salvare în bază, apelată în service-ul BusinessTravelRequest.java, se creează câte o instanță de tipul BusinessTravelRequestHistory.java, ce va conține câmpurile ce se doresc a fi salvate în bază. Astfel, după fiecare salvare a cererii de deplasare, va fi salvată și instanța nou creată. Metoda save() din cadrul service-ului BusinessTravelRequest.java este apelată atât de metoda de POST cât și de către metoda de PUT din cadrul controller-ului.

History of the business travel requests



| ID ↑↓ | Status ↑↓ | Assign from ↑↓ | Assign to ↑↓ | Total cost ↑↓ | Location ↑↓ | Last modified date ↑↓ | Business travel request ↑↓ | |
|-------|----------------------|----------------|--------------|---------------|-------------|-------------------------|----------------------------|----------------------|
| 7 | Initiated | manager1 | supplier1 | 0 | France | Jun 28, 2016 2:26:50 AM | 1 | View |
| 8 | Waiting for approval | supplier1 | manager1 | 320 | France | Jun 28, 2016 2:28:24 AM | 1 | View |
| 9 | Issuing tickets | manager1 | supplier1 | 320 | France | Jun 28, 2016 2:29:04 AM | 1 | View |
| 10 | Initiated | manager2 | supplier1 | | Spain | Jun 28, 2016 9:43:53 PM | 7 | View |
| 11 | Initiated | supplier1 | supplier1 | 675 | Spain | Jun 28, 2016 9:44:46 PM | 7 | View |
| 12 | Waiting for approval | supplier1 | manager2 | 485 | Spain | Jun 28, 2016 9:45:25 PM | 7 | View |
| 13 | Waiting for approval | manager2 | manager1 | 485 | Spain | Jun 28, 2016 9:45:39 PM | 7 | View |
| 14 | Issuing tickets | manager1 | supplier1 | 485 | Spain | Jun 28, 2016 9:46:00 PM | 7 | View |
| 15 | Finished | supplier1 | supplier1 | 320 | France | Jun 28, 2016 9:46:20 PM | 1 | View |

Figura 38: Ecran cu istoricul cererilor

4.3. Testarea aplicației

Pentru testare am folosit framework-ul JUnit.

5. Contribuții personale

5.1. Concluzii

Trăind într-o eră în care timpul este foarte important, iar industria IT în plină expansiune, am ales să realizez o aplicație care să ajute la gestionarea acestui factor important, folosindu-mă de cunoștințele acumulate până în acest moment și de cele mai noi tehnologii existente în domeniu la ora actuală.

În urma unui internship în cadrul unei companii multinaționale, am constatat că existența unei aplicații de gestiune a deplasărilor angajaților în interes de serviciu este o necesitate în zilele noastre. O astfel de aplicație ajută angajații să scutească timp și resurse în realizarea formalităților necesare unei astfel de deplasări.

Pentru realizarea aplicației pașii urmați au fost:

- Am început cu un studiu privind aplicațiile similare pentru a extrage funcționalitățile de bază
- Am sintetizat informațiile referitoare la necesitățile companiilor ale căror angajați au deplasări dese
- Pe baza funcționalităților identificate am ales instrumentele pe care le-am folosit în dezvoltarea aplicației
- Utilizând instrumentele am trecut la implementarea propriu-zisă a aplicației

5.2. Contribuții personale

Contribuțiile personale s-au materializat în trei direcții:

1. Sinteza de informații

1.1. Referitoare la domeniul gestiunii informațiilor legate de deplasări

În cadrul aceluși internship, am avut oportunitatea de a înțelege mai multe lucruri în legătură cu acest proces de deplasare în interes de serviciu care se aplică astăzi la numeroși angajați.

1.2. Referitoare la aplicațiile similare din domeniu

Principalele aplicații în acest domeniu sunt Travel Request și Concur.

Travel Request este o aplicație ce aparține platformei KISSFLOW, dezvoltată de către compania OrangeScape, ce permite automatizarea procesului de deplasare în scopul interesului de serviciu, având ca funcționalități de bază:

- Crearea de fluxuri de lucru pentru o deplasare

- Gestionarea acestor fluxuri
- Gruparea acestora în funcție statusul actual al cererii

Concur este o platformă de gestionare a călătoriilor, ce este conectată cu diverse agenții de turism, având ca scop principal minimizarea costurilor acestora.

1.3. Alegerea instrumentelor utilizate pentru dezvoltare

Pornind de la premisa că aplicațiile ce rulează în browser au început să devină din ce în ce mai folosite am ales ca aplicația dezvoltată să fie una web. Pentru realizarea unei aplicații web, există numeroase tehnologii ajutătoare atât pe partea de back-end cât și pe cea de front-end.

Astfel, pentru partea de back-end am ales JAVA, iar pentru partea de front-end principala tehnologie folosită este ANGULARJS, framework de Javascript, realizat și întreținut de către inginerii de la Google.

1.4. Sinteza informațiilor referitoare la instrumentele utilizate în dezvoltare

Java este un mediu de dezvoltare caracterizat de simplitatea, robustețea, securitatea, fiind independent de platformă, fiind utilizat din ce în ce mai mult în realizarea părții de back-end a aplicațiilor web, prin framework-uri apărute.

AngularJS este un framework ce folosește o arhitectură de tipu MVC, destinat realizării părții de client a unei aplicații web. Principala caracteristică este aceea că se pot dezvolta aplicații de tip SPA (Single Page Application).

Pentru stocarea datelor și gestionarea datelor, am folosit baza de date MySQL.

2. Conceperea de soluții

2.1. Elaborarea la nivel conceptual a arhitecturii sistemului

La nivel conceptual, arhitectura aplicației este formată din două module, care dependente unul de celălalt. Primul modul este reprezentat de partea de administrare a utilizatorilor și a rolurilor acestora.

Cel de-al doilea modul este reprezentat cerința propriu-zisă a aplicației, și anume, gestionarea cererilor de deplasare. Legătura între cele două module este realizată prin intermediul rolurilor îndeplinite de utilizatorii aplicației.

2.2. Stabilirea la nivel conceptual a funcționalităților fiecărui modul al aplicației

Principalele funcționalități oferite de primul modul se referă la administrarea conturilor de utilizatori ai aplicației și sunt următoarele:

- Creare de noi conturi în cadrul aplicației
- Editarea conturilor existente
- Ștergea de conturi

Cel de-al doilea modul oferă funcționalități de creare și gestionare a cererilor de deplasare.

3. Implementare

3.1. Implementarea funcționalităților fiecărui modul al aplicației

Fiind o aplicație web, implementarea presupune implementarea părții de front-end și cea a părții de back-end. Cele două comunicând prin intermediul cererilor de tip client-server.

În primul modul, sunt implementate funcționalitățile de creare, editare sau ștergerea a unui cont de utilizator.

Crearea și editarea sunt realizate prin intermediul unui formular, care este completat de către utilizatorul aplicației. Datele sunt preluate și trimise către server sub forma unor cereri HTTP către client, acesta primind răspuns de la server.

3.2. Conectarea modulelor astfel încât ele să formeze o aplicație unitară

Conectarea modulelor se realizează prin intermediul rolurilor îndeplinite de către utilizatorii sistemului.

5.3. Dezvoltări ulterioare

Dezvoltările ulterioare se pot materializa în:

- Crearea de rapoarte care să extragă anumite date statistice despre cererile create
- Îmbunătățirea structurii bazei de date
- Automatizarea procesului de căutare de oferte către furnizor.

6. Bibliografie

- [1] <https://kissflow.com/>
- [2] <http://www.mavenwave.com/services/cloud-solutions/partners/>
- [3] https://kissflow.com/butterfly_effect/travel-request-approval-workflow-forms-admin-google-apps/
- [4] M. Anghel, *Aplicații WEB cu suport JAVA, Curs 1*
- [5] <https://www.w3.org/html/logo/>
- [6] http://ciobanu.cich.md/lectii_view.php?id=1
- [7] Michael Morrison, *Head First JavaScript 2008*
- [8] <http://www.potenzaglobalsolutions.com/blogs/why-is-angularjs-the-best-choice>
- [9] Pawel Kozlowski, Peter Bacon Darwin, *Mastering Web Application Development with AngularJS*
- [10] <http://www.webgiant.co.nz/services/>
- [11] <http://getbootstrap.com/>
- [12] Bart Baesens, Aimee Backiel, Seppe vanden Broucke, *Beginning Java Programming*
- [13] Patrick Niemeyer, Daniel Leuck, *Learning Java, 4th Edition ,2013*
- [14] <http://javabeat.net/spring-4/>
- [15] http://www.tutorialspoint.com/spring/spring_architecture.htm
- [16] Craig Walls, *Spring in Action, 4th Edition*
- [17] <http://javabeat.net/spring-data-jpa/>
- [18] http://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm
- [19] M. Anghel, *Aplicații WEB cu suport JAVA, Curs 9*
- [20] <http://www.xfront.com/REST.html>
- [21] Bryan Basham, Kathy Sierra, Bert Bates, *Head First Servlets & JSP, 2th Edition*

