Universitatea “Politehnica” din Timișoara

Facultatea de Automatica si Calculatoare

**Calculatoare si Tehnologia Informației**

**Stop Cozi**

Lucrare de Licență

*Absolvent:*

**Alexandra PETCOV**

*Coordonator științific:*

Prof. dr. ing. **Oana Iulia Casandra HOLOTESCU**

Timișoara

2017

**CUPRINS**

**Pagina**

1. **INTRODUCERE 5**
   1. **Contextul proiectului**
   2. **Ideea şi scopul proiectului**
   3. **Structura proiectului**
      1. **Structura site-ului**
      2. **Funcționalități pentru utilizatori**
      3. **Funcționalități pentru administratori**
2. **TEHNOLOGIILE FOLOSITE 9**
   1. **Notiuni generale despre WWW (World Wide Web)**
   2. **Tehnologiile folosite**
      1. **Tehnologii folosite pe partea de Server (server-side)**
      2. **Tehnologii folosite pe partea de Client (client-side)**
   3. **Fișierele sursă folosite**
3. **SPECIFICAȚIILE APLICAȚIEI 19**
   1. **Cerințe, use case-uri**
   2. **Arhitectura**
   3. **Baza de date**
4. **DETALII IMPLEMENTARE 25**
   1. **Implementarea pattern-ului MVC**
   2. **Definirea layer-ului de persistență și configurația pentru Hibernate**
   3. **Portalul utilizatorului**
      1. **Autentificarea unui utilizator**
      2. **Înregistrarea unui utilizator**
      3. **Editare profil**
      4. **Prima pagină**
      5. **Crearea unui programări**
      6. **Încarcarea documentelor personale**
      7. **Generarea personalizată a documentelor**
   4. **Portalul funcționarului**
      1. **Autentificare**
      2. **Vizualizarea utilizatorilor**
      3. **Vizualizarea programărilor**
         * 1. **Confirmarea / Anularea unei programări**
5. **UTILIZAREA APLICAȚIEI 41**
   1. **Scenarii utilizare - screenshots**
6. **CONCLUZII 49**
   1. **Concluzii**
   2. **Viitoare îmbunătățiri**
7. **REFERINȚE 51**
8. **INTRODUCERE**
   1. **Contextul problemei**

De la apariția sa, internetul continuă să evolueze într-un ritm accelerat având un impact semnificativ asupra indivizilor, societății, guvernelor și mediului de afaceri. De la simpla interconectare a 4 universități americane la începuturile sale, astăzi internetul a devenit o parte nelipsită a vieții moderne oferindu-ne alternativa organizării și desfășurării activităților într-un mod mai simplu și mai eficient. Printre multele avantaje de care beneficiem astăzi se numără accesul rapid si facil la informație, educație, socializare și diverse servicii.

Conform „Raportului intermediar pe 2017 privind sectorul digital din UE (EDPR), profil de țară: România”, "zonele urbane sunt acoperite de conexiuni în bandă largă de mare viteză, România situându-se pe locul doi în UE în ceea ce privește proporția de abonamente din UE. …. Cu toate acestea, rata de digitalizare a economiei, inclusiv a serviciilor publice, și nivelul competentelor digitale rămân scăzute."[1]

Conform aceluiași raport, în ceea ce privește serviciile publice digitale, România se situează sub media Uniunii Europene, iar utilizarea serviciilor de e-guvernare este cea mai scăzută in UE.

În România, în interacțiunea cu instituțiile publice, cetățeanul încă este nevoit să stea la cozi interminabile, fiind adesea trimis de la un ghișeu la altul, de la o instituție la alta pentru parcurgerea pașilor din cadrul procesului administrativ. Încă se lucrează cu metode rudimentare (formulare care trebuie completate de mână, multiple fotocopii ale documentelor necesare, lipsa unui sistem electronic pentru ținerea evidenței cererilor și a documentelor depuse), fapt care îngreunează livrarea eficientă și într-un timp cât mai scurt a unor servicii de calitate.

Puținele progrese realizate față de anii precedenți în ceea ce privește furnizarea serviciilor s-a realizat prin creșterea numărului de servicii care pot fi efectuate online și prin pre-completarea automată a formularelor pentru cetățeni.

Pentru a îmbunătăți calitatea serviciului public, guvernul român a adoptat în iulie 2016 programul GovITHub. Acest program implementat la nivel național care reunește autoritățile publice și întreprinderile private a generat beneficii concrete și are impact asupra vieții de zi cu zi a cetățenilor. GovITHub a îmbunătățit conștientizarea și transparența prin implicarea oamenilor din comunitate și oferind proceduri online pentru interacțiunea cu autoritățile publice.

* 1. **Ideea şi scopul proiectului**

Proiectul „Stop Cozi” face parte din programul guvernamental „GovITHub”, care își propune creșterea calității serviciilor publice cu ajutorul tehnologiei. Ideea este preluată din acest program ( http://ithub.gov.ro/2016/10/07/stop-cozi/ ) și constă în implementarea unui portal web, principală funcționalitate fiind crearea unei programări la o instituție publică.

Serviciile publice au ca principală atribuție eliberarea, păstrarea, evidența și întocmirea diferitelor acte. Aceste acte pot fi: certificate de înmatriculare a mașinilor, acetele necesare stării civile, a cărților de identitate, permise de conducere. Serviciile publice pretează activități de lucru cu publicul, fiind necesară organizarea corespunzătoare a ghișeelor prin intermediul cărora prestează aceste servicii.

În fiecare zi lucrătoare la aceste ghișee apar, inevitabil, cozi de așteptare. Această aplicație își propune îmbunătățirea lucrului cu publicul la ghișeele de la diverse agenții publice, diminuarea cozilor în așteptare și scăderea timpilor de așteptare prin introducerea posibilității creării unei programări online, ce permite alegerea agenției publice și a serviciului dorit. Planificarea activităților de lucru cu publicul pe baza programărilor poate preveni creearea unor cozi în așteptarea foarte mari și implicit nemulțumirile cetățenilor cu privire la cozile în așteptare ce s-ar crea. Pe lângă crearea propriu zisă a programării, sistemul permite încărcarea documentelor personale, respectiv trimiterea acestora în avans pentru a ușura vizita la ghișeu.

* 1. **Structura proiectului** 
     1. **Structura site-ului**

Site-ul „Stop Cozi” este structurat în două părți. Prima parte se adresează utilizatorilor, tuturor persoanele ce doresc realizarea unei programării la o agenție publică, iar a doua parte se adresează administratorului, respectiv funcționarului ce se ocupă de gestiunea programărilor, consultarea și rezolvarea acestora. Cele două tipuri de utilizatori au privilegii diferite, funcționarul poate accesa atât pagina utilizatorului cât și pagina administratorului, pe când utilizatorul poate accesa doar pagina sa. Datele personale ale fiecărui utilizator pot fi modificate doar de el însuși.

**1.3.2 Funcţionalităţi pentru utilizatori**

Pentru utilizatori contează în primul rând ușurința cu care ei găsesc informațiile dorite și ușurința de navigare. Pentru a ajuta în această direcție site-ul este structurat în diferite secțiuni: programări online, contul utilizatorului și formulare online. Utilizatorul poate accesa cu usurintă secțiunile, acestea aflându-se în header-ul fiecărei pagini.

**Secțiunile site-ului**

Secțiunea programări online oferă utilizatorului posibilitatea creării unei programări online la o instituție publică.

Secțiunea formulare online pune la dispoziția utilizatorului o lista de documente, care sunt pre-completate automat cu datele utilizatorului și pot fi descărcate.

Contul utilizatorului conține următoarele module, printre care se numără și profilul utilizatorului, încărcarea documentelor personale, respectiv vizualizarea tuturor programărilor.

Site-ul dispune de un sistem de înregistrare și autentificare utilizatori. Detaliile tehnice se regăsesc în secțiunea 4.3. Exemple de pagini:

* Formulare online: <http://localhost:8181/generateDoc/document>
* Programari online: <http://localhost:8181/appointment/create>
* Profilul utilizatorului ( apartine de secțiunea contul utilizatorului ) :

<http://localhost:8181/user/profile>

* + 1. **Funcţionalităţi pentru administratori**

Funcționării pot intra pe următoarele paginii:

* <http://localhost:4200/userAccount> pentru a vedea toți utilizatorii înregistrați, această pagină poate fi accesată doar de administratorul site-ului, acesta având rolul de „super user”
* <http://localhost:4200/appointment> pentru a vedea toate programările, această pagină poate fi accesată de toți funcționării, fiecare dintre aceștia având dreptul de a vizualiza, confirma, respectiv rejecta programariile care au fost făcute la agenția unde acesta lucrează și care țin de serviciile care intră în atribuția lui

Mai multe detalii se găsesc în secțiunea 4.4.

1. **TEHNOLOGIILE FOLOSITE**
   1. **Noțiuni generale despre WWW (World Wide Web)**

World Wide Web a fost inventat de Sir Tim Berners-Lee în anul 1989, el își dorea să găsească o nouă modalitatea de share-uire a datelor provenite în urma experimentelor științifice. Deși internetul și hypertext-ele existau, nu se gândiseră la folosirea acestora pentru unirea documentelor între ele prin legături(link). Termenul World Wide Web (WWW sau Web) se referă la totalitatea Web site-urilor ce sunt conectate la Internet și a dispozitivelor (calculatoare, telefoane) prin care poate fi accesat conținutul acestor site-uri. WWW–ul se bazează pe următoarele tehnologii HTML, URL și HTTP. [2]

HTML-Hypertext Markup Language, este un limbaj de marcare folosit pentru crearea paginilor web.

URL- Uniform Resource Locator, folosit pentru denumirea unei adrese. În general este format din patru componente: un nume de protocol, locația site-ului, un nume(ales de propietarul web site-ului), o terminație, ce arată tipul de propietar al web site-ului. [3]

HTTP – Hypertext Transfer Protocol, este un protocol de tip text, folosit pentru transmiterea informațiilor între un program de navigare Web (browser) și un server Web. Există HTTP server (furnizează pagini web) și HTTP client (cere pagini web). „Protocolul HTTP se caracterizează prin faptul că nu memorează o succesiune a stărilor prin care trece legătura client-server. Astfel fiecare tranzacție este independentă: clientul trimite o cerere, serverul răspunde cu resursa cerută. Pentru fiecare resursă, există o tranzacție corespunzătoare.” [4].

* 1. **Tehnologiile folosite**

În următoarele două subcapitole vom explica modul de funcționare al site-ului prin enumerarea tehnologiilor folosite de acesta. Proiectul este format din două aplicații, o aplicație vizează partea utilizatorului, iar cealaltă aplicație vizează partea administratorului. Capitolul se încheie cu o secțiune în care sunt descrise toate fișierele sursă folosite de site.

Portal utilizator

Backend: Java/Spring Boot, Spring Security 4, Spring Data, Hibernate Frontend: Thymeleaf, Bootstrap, JavaScript/JQuery Database: MySQL

Portal administrator

Frontend: Angular 2, TypeScript

Portalul administratorului este conectat la backend folosind un serviciu REST. Din punctul de vedere al securității, atât portalul utilizatorului, cât și portalul administratorului folosesc Spring Security.

* + 1. **Tehnologii folosite pe partea de Server (server-side)**

Java-este un limbaj de programare orientat pe obiecte, dezvoltat de Sun Microsystems și lansat în anul 1995. Java este platform independent, acest lucru înseamnă că un cod compilat poate rula fără modificări sau cu mici modificări pe orice platformă, unde există o mașină virtuală instalată(JVM), spre deosebire de alte limbaje cum ar fi C, C++. Acest lucru este posibil deoarece sursele Java sunt compilate într-un format standard numit byte-code(cod de octeți). Acest cod este interpretat de JVM (Java Virtual Machine). Codul de octeți este un intermediar între codul mașină(dependent de tipul calculatorului) și codul sursă.[5]

Spring – este un framework „open source”, deși targetul sau nu este niciun limbaj de programare, a devenit popular în rândul comunității Java. A fost creat de RodJohnson și descris în cartea sa „Expert One-on-One: J2EE Design and Development”. Spring-ul a fost create pentru a aborda dezvoltarea aplicațiilor enterprise complexe și a face posibilă folosirea „plain-vanilla” JavaBeans pentru a obține lucruri, care anterior erau posibile doar folosind EJB (Enterprise JavaBeans). Folosirea lui nu este limitată doar la partea de server-side, ci orice aplicație Java poate beneficia de simplicitatea, testabilitatea și cuplajul slab (loose coupling) oferit de acesta.[6]

Spring Boot – oferă posibilitatea de a crea cu ușurință aplicații „stand-alone”(de sine stătătoare) bazate pe framework-ul Spring. Printre caracteristiciile oferite se număra: încorporează Tomcat (nu este necesară instalarea fișierelor de tipul WAR), configurează automat Spring oriunde este posibil, nu este necesară absolut deloc configurație XML, oferă „starter POM”(Project Object Model) pentru a simplifrica configurația Maven.

Tomcat – este un server Web „open source” (cu sursă deschisă) dezvoltat de Apache Software Foundations(ASF). Oferă un server web HTTP „Java pur”, unde codul Java poate rula. Protocolul HTTP este implementat în interiorul serverului Tomcat, acesta având grijă de „Cerere”(Request) și „Raspuns”(Response).

Spring Security 4 – este un framework de securitate ce oferă suport pentru autentificare și autorizare, pentru securizarea aplicațiilor Spring. Se integrează bine cu Spring MVC și posedă implementări ale algoritmilor populari de securitate.

Spring Data – oferă un model de programare bazat pe Spring ce îți permite accesul datelor, păstrând în același timp trăsăturile speciale care stau la baza unei „data store” (bază de date).

Hibernate – Hibernate ORM (object-relational mapping) este un tool ORM pentru limbajul Java, utilă pentru maparea modelului orientate pe obiecte peste bazele de date relaționale.

„Termenul de ORM (Object-Relațional Mapping) este folosit pentru a descrie o tehnică de programare prin care sunt transformate informațiile între formatul în care sunt stocate în bazele de date relaționale și formatul în care sunt reprezentate în limbajele de programare orientate pe obiect. Hibernate reprezintă o soluție de tip ORM pentru asigurarea persistenței și a interogării eficiente a informațiilor stocate în baze de date relaționale.

JPA (Java Persistence API) reprezintă o colecție de clase (în limbajul de programare Java), oferind functionalităti legate de interogarea eficientă a informațiilor stocate în baze de date relaționale, mai ales în situația în care volumul acestora este considerabil. Această interfată de programare implementează un nivel de abstractizare, toate informațiile fiind stocate la nivel de obiecte, astfel încât programatorul nu este obligat să cunoască detaliile specifice unui sistem de gestiune al bazelor de date.”[7]

„JDBC (Java DataBase Connectivity) este o interfață standard SQL de acces la baze de date. Acesta ne furnizează un acces uniform la baze de date relaționale. JDBC este constituit dintr-un set de clase și interfețe scrise în Java, furnizând un API standard pentru proiectanții de aplicații baze de date. JDBC-ul este o interfață "low-level", adică este folosit pentru a executa direct comenzi SQL.”[8]

Thymeleaf – este un Java template engine (un motor de tip șablon Java), folosit pe partea de server (server-side) atât pentru aplicații (medii) web cât și pentru aplicații independente.

MySQL-este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeză MySQL AB și distribuit sub Licentă Publică Generală GNU.

* + 1. **Tehnologii folosite pe partea de Client (client-side)**

Bootstrap – este cel mai popular framework HTML, CSS și JS pentru dezvoltarea proiectelor mobile, receptive pe web. [9]

Html – (Hypertext Markup Language) este un limbaj de marcare folosit pentru crearea paginilor web, standardizat de către W3C (World Wide Web Consortium). De exemplu: definirea paragrafelor, heading-urilor și a tabelelor sau chiar inserarea imaginilor și a videoclipurilor în pagina web. Navigatoarele web interpretează codul HTML pentru a afișa conținutul.

CSS – (Cascading Style Sheets) este un limbaj de formatare a elementelor dintr-o pagină web, standardizat de către W3C (World Wide Web Consortium). De exemplu: setarea culorii unui fundal, a fonturilor.

JavaScript – este o implementare a standardului ECMAScript, ce se regăsește cu precădere în navigatoarele web. Acest limbaj de programare permite aplicațiilor web să fie dinamice și interactive.

JQuery – este o bibliotecă a limbajului de programare JavaScript, folosită pentru a simplifica și îmbunătății diferite procese în HTML. De exemplu: traversarea arborelui DOM (Document Object Model) și cererile de tip AJAX (Asynchronous JavaScript and XML).

Angular2 – este unul dintre cele mai populare framework-uri al limbajului de programare JavaScript, folosit pentru crearea aplicațiilor web.

TypeScript – este un limbaj de programare „open-source” devoltat de Microsoft. Este un superset al limbajului de programare JavaScript, adăugând limbajului un mod de programarea orientat pe obiecte bazat pe clase și declarare statică opțională („static typing”).

* 1. **Fişierele sursa folosite**

Lista de directoare și fișiere este prezentată mai jos pentru a face mai clar modul de funcționare al site-ului. O prezentare a acestei structuri permite formarea unei imagini mai clare asupra locului unde se poate afla codul responsabil pentru diferitele funcții ale site-ului.

**Portalul utilizatorului:**

pom.xml – este un fișier XML care conține informații despre proiect și detaliile de configurare necesare pentru ca Maven-ul să poată realiza operația de build. Project Object Model (POM) este o unitate de lucru folosită în proiectele Maven. Aici se află definite dependențele pentru freemarker, commons-fileupload, jpa, security, jdbc, etc.

resources/

* application.properties – Spring Boot permite externalizarea configurației, pentru a putea lucra cu același cod în diferite medii. Astfel în application.properties se află definit URL pentru conexiunea la baza de date, portul folosit de serverul Tomcat, host-ul, port-ul, username-ul pentru spring.mail, etc. SpringApplication va încărca proprietățile din fișierul „application.properties”.
* import.sql – este un fișier sql ce conține date necesare populării bazei de date
* **/**templates/
* index.html

- este pagina de autentificare. Vezi secţiunea 5.1.

* userFront.html

- pagina principala a aplicației, contine informații despre utilizarea aplicației. Vezi secţiunea 5.1.

* document.html

-afișează o listă cu documentele puse la dispoziția utilizatorului pentru a putea fi completate si descarcate automat. Vezi secţiunea 5.1.

* appointment.html

-permite crearea unei noi programări. Vezi secţiunea 5.1.

* myapps.html

- permite vizualizarea tuturor programărilor ce aparțin de utilizatorul autentificat. Vezi secțiunea 5.1.

* profile.html

-permite modificarea datelor personale ale unui utilizator, pe baza acestor date se vor genera documentele pre-completate. Vezi secțiunea 5.1.

* uploadFile.html

-permite încărcarea, descărcarea si ștergerea documentelor personale. Vezi secțiunea 5.1.

* signup.html

- pagina de înregistrare, permite înregistrarea unui utilizator. Vezi secțiunea 5.1.

* header.html, iar path-ul este: /common

-este header-ul ce este prezent pe fiecare pagina.

* /static/
* css/ - Stilurile folosite in diferite secțiunii ale aplicației
* bootstrap-datepicker.css

-Bootstrap-ul ofera un widget datepicker, iar acest fișier este folosit pentru stilizarea lui.

* bootstrap-datepicker.min.css

-acelasi conținut ca si anteriorul, doar ca acest fișier a fost minimizat, adica toate spațiile albe au fost șterse pentru a reduce dimensiunea fișierului si a crește viteza.

* bootstrap.css

-ofera setări CSS globale, elemente HTML fundamentale stilate

* bootstrap.min.css

-același conținut, doar ca fișierul a fost minimizat

* dataTables.bootstrap.min.css – DataTables este un plug-in pentru biblioteca Javascript JQuery, pentru stilizarea acestuia s-au introdus cele 2 fișiere
* jquery.dataTables.min.css
* main.css

-suprascrierea stilurilor oferite de Bootstrap pentru diferite elemente HTML

* fonts/

- glyphicons este o biblioteca formata din imagini și simboluri monocromatice, folosită pentru accentuarea simplicității și orientare ușoara

* glyphicons-halflings-regular.eot

-.eot – pentru versiuni ale Internet Explore-ului mai vechi decât IE9

* glyphicons-halflings-regular.svg

-.svg – pentru sistemul de operare IOS, spre exemplu IPad, IPhone

* glyphicons-halflings-regular.ttf

-.ttf – este un font normal și vechi, dar este ușor de descărcat

* glyphicons-halflings-regular.woff

-.woff – oferă un mod care impiedica piratarea fontului

* glyphicons-halflings-regular.woff2

-.woff2 – un woff mult mai compresat, pentru noile browser-uri

* images/
* banner.png

- imaginea folosită pentru banner, a fost creată folosind Google docs

* bg.png

- imaginea folosită pentru fundal, a fost creată folosind Google docs

* js/
* bootstrap-datepicker.js

-codul folosit pentru widget-ul datepicker

* bootstrap-datepicker.min.js

-comprimarea fișierului anterior

* bootstrap.js

-codul folosit de către bootstrap

* bootstrap.min.js

-comprimarea fișierului anterior

* dataTables.bootstrap.min.js
* jquery.dataTables.min.js – integrarea DataTables se realizeaza prin includerea celor patru fisiere, cele doua prezentate anterior avand extensia .css si aceste doua
* jquery.js
* jquery.plugin.js
* main.js

-fișierul folosit pentru popularea agențiilor, serviciilor, etc

* config.json

-fișierul unde este configurat IP-ul serverului, respectiv port-ul serverului

**Portalul administratorului**

**/**app

* **/**appointment
* appointment.component.html

-afișarea tuturor programărilor, pentru un anumit oraș, agenție și serviciu

* appointment.component.ts
* /login
* login.component.html – pagina de autentificare. Vezi secțiunea 5.1.
* login.component.ts
* /navbar
* navbar.component.html – pagina se bazeaza pe Bootstrap, se definește bara de navigare, care va conține urmatoarele secțiuni: administrator, utilizatori,programări si ieșire
* navbar.component.ts
* /user-account
* user-account.component.html – afișarea tuturor utilizatorilor. Vezi secțiunea 5.1.
* user-account.component.ts
* app.module.ts - identifică componentele folosite
* app.routing.ts - sunt definite căile de rutare (routing)
* appointment.service.ts – folosit pentru a trimite date backend-ului (confirmare programare, afișarea tuturor programărilor)
* login.service.ts – folosit pentru a trimite date backend-ului (autentificare si deconectare)
* user.service.ts – folosit pentru a trimite date backend-ului (afișarea tuturor utilizatorilor, permite utilizator, dezactivează utilizator)

/assets

* /css
  + bootstrap.min.css - ofera setări CSS globale, elemente HTML fundamentale stilate, fișierul este minimizat
* /fonts
* glyphicons-halflings-regular.eot

-.eot – pentru versiuni ale Internet Explore-ului mai vechi decât IE9

* glyphicons-halflings-regular.svg

-.svg – pentru sistemul de operare IOS, spre exemplu IPad, IPhone

* glyphicons-halflings-regular.ttf

-.ttf – este un font normal și vechi, dar este ușor de descărcat

* glyphicons-halflings-regular.woff

-.woff – oferă un mod care impiedica piratarea fontului

* glyphicons-halflings-regular.woff2

-.woff2 – un woff mult mai compresat, pentru noile browser-uri

* /js
  + bootstrap.min.js
  + jquery.js

-index.html – sunt adaugate stilurile pentru bootstrap, respectiv fișierele javascript

1. **SPECIFICAȚIILE APLICAȚIEI**
   1. **Cerințe, Use-Case-uri**

Proiectul “STOP COZI” este o aplicație web duală: o aplicație ce vizează funcțiile administratorului, iar cealalata pe cele ale utilizatorului, astfel cerințele aplicației pot fi grupate în două părți, respectiv partea utilizatorului: înregistrarea utilizatorului și meniu și partea administratorului.

* Inregistarea utilizatorului – crearea unui cont nou pentru utilizatorii care nu există în baza de date, validarea se va face atât pentru adresă de email introdusă cât și pentru numele de utilizator.
* Meniu

Modificarea profilului – i se oferă utilizatorului permisiunea de a își completa, respectiv actualiza profilul.

Crearea unei programări – posibilitatea creării unei programări online selectând orașul, agenția și serviciul dorit.

Documente personale – i se oferă utilizatorului posibilitatea stocării unor documente personale, având ca scop trimiterea în avans a acesora, pentru a ușura vizită la ghișeu.

Programările mele – i se oferă utilizatorului opțiunea de a își vedea programările pe care le-a făcut, dacă sunt confirmate sau respinse.

Pre-completare documente – i se pune la dispoziția utilizatorului documente pe care și le poate descărca, fiind completate automat cu datele personale ale acestuia.

* Administrator

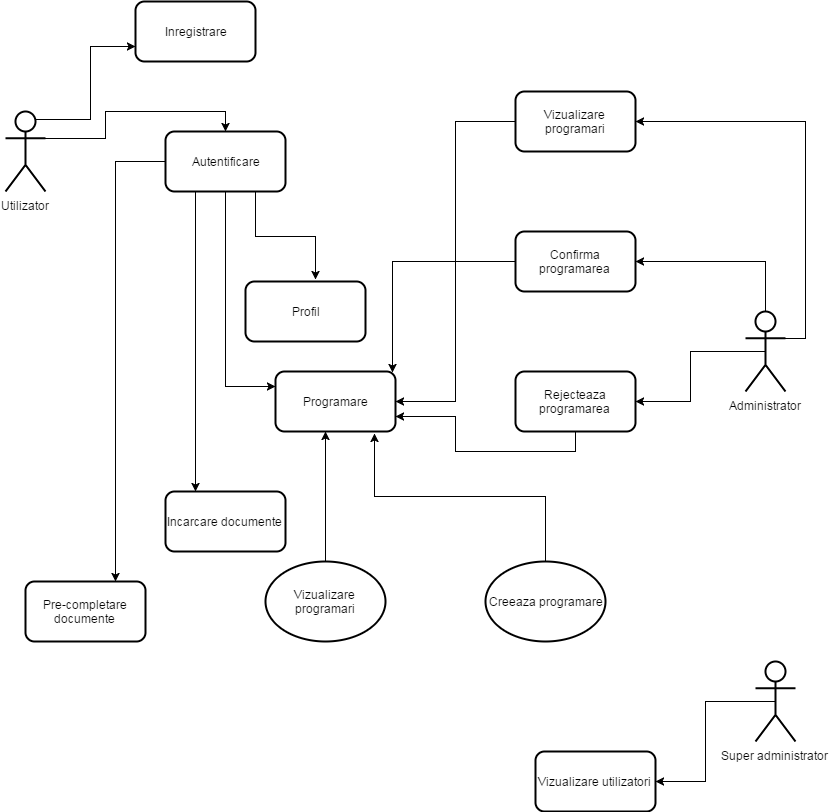
Vizualizare utilizatori – în funcție de rolul administratorului i se permite acestuia vizualizarea tuturor utilizatorilor

Confirmare programare – în funcție de agenția la care lucrează administratorul și de ghișeul de care se ocupă (serviciul pe care îl gestionează) i se permite acestuia să confirme programarea realizată de utilizatorul „X”.

Rejectare programare – în funcție de agenția la care lucrează administratorul și de ghișeul de care se ocupă (serviciul pe care îl gestionează) i se permite acestuia să rejecteze programarea realizată de utilizatorul „X”.

Vizualizare programare - în funcție de agenția la care lucrează administratorul și de ghișeul de care se ocupă (serviciul pe care îl gestionează) i se permite administratorului să vizualizeze toate programările realizate la acea agenție și pentru acel serviciu.

**Use-Case**

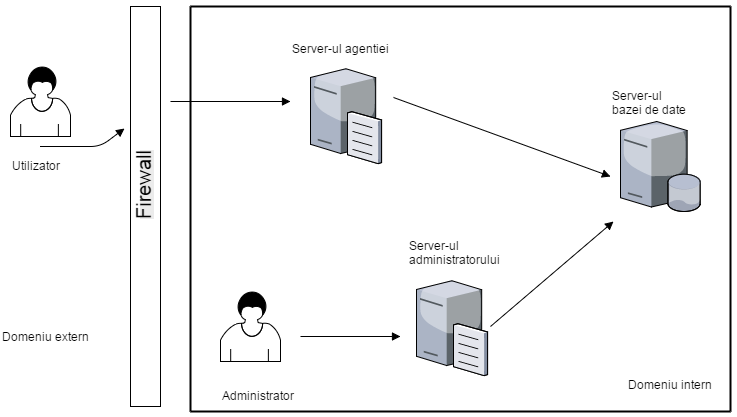


* 1. **Arhitectura**

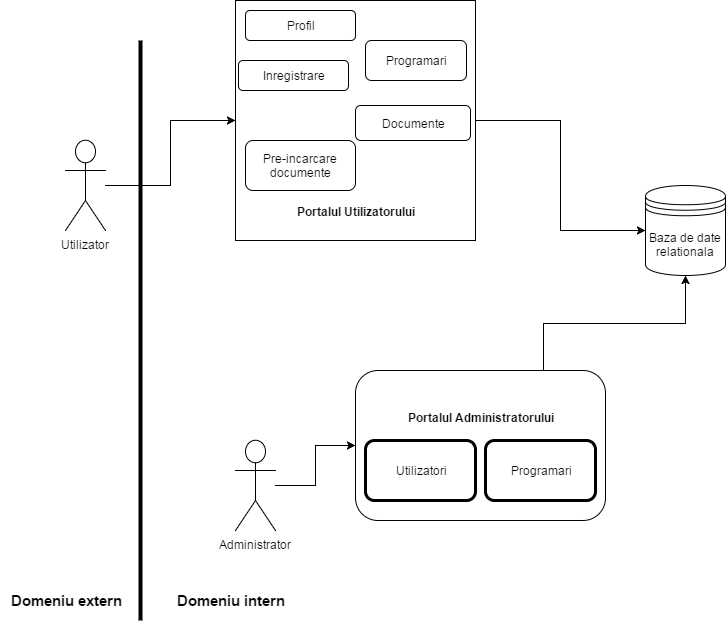
Arhitectura trebuie privită din mai multe puncte, unul dintre acestea ar fi arhitectura hardware.

**Arhitectura hardware**

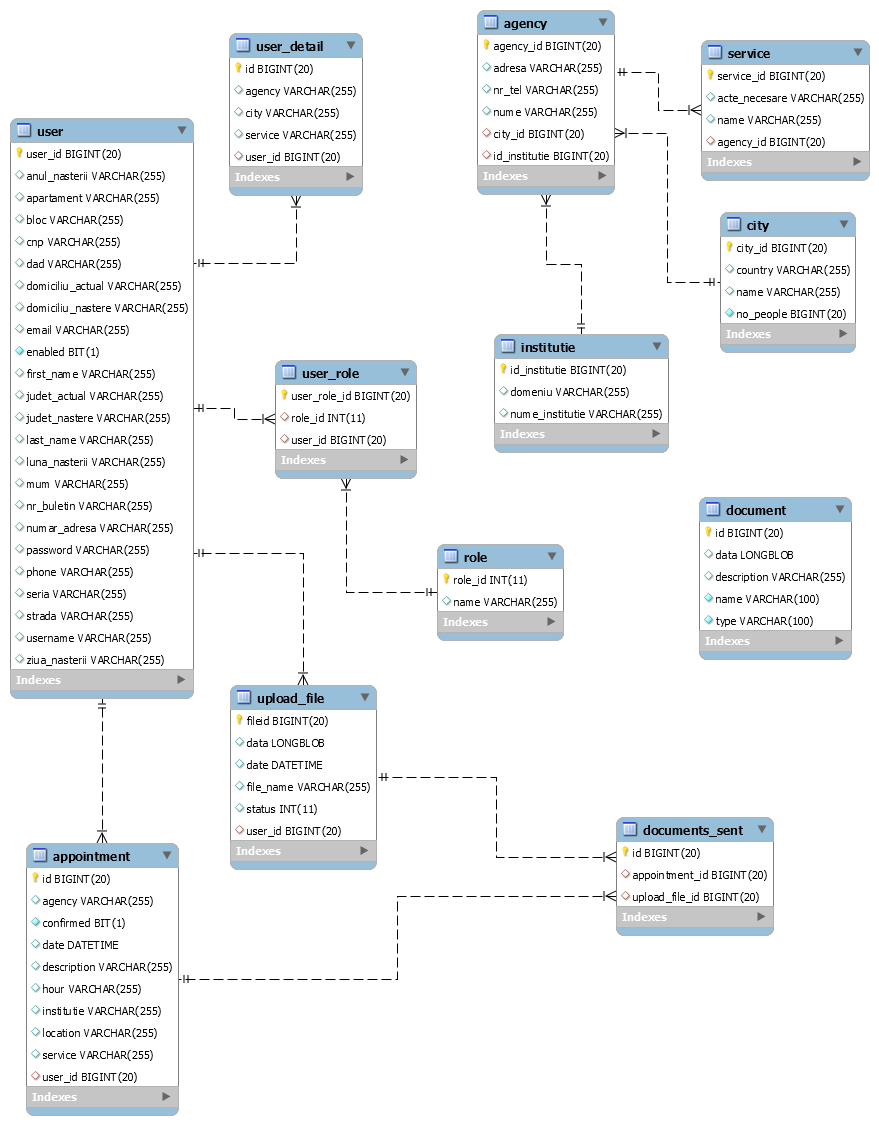
Aplicația va fi formată din două domenii, domeniul extern și domeniul intern, între acesta va exista un firewall care va bloca cererile („request-urilor”) ilegale. Se observă că domeniul intern este format din două servere, serverul agenției și serverul administratorului, amândouă servere comunică cu aceeași bază de date. Utilizatorul, care face parte din domeniul extern, accesează serverul agenției, trecând de firewall. Administratorul face parte din domeniul intern, putând accesa în mod direct serverul administratorului. Motivul pentru care s-a ales separarea portalului utilizatorului de portalul administratorului este funcționalitatea crescută pe partea utilizatorului și layout-ul mai bogat din punctul de vedere al graficii față de portalul administratorului, care este folosit doar intern. Cele două părți ale aplicației au un context diferit în dezvoltare. („different context în development”).

****

În următoarea imagine se prezintă cele două module, fiecare modul este format din mai multe blocuri. Se observă că portalul utilizatorului este format din: profil, înregistrare, programări, documente și pre-încărcare de documente, iar portalul administratorului este format din programări, respectiv utilizatori, amândouă module comunică cu bază de date relațională.

****

* 1. **Baza de date**



În imaginea anterioară este prezentată diagrama bazei de date. Se va descrie fiecare tabelă și relațiile dintre acestea. Tabela „user” conține numeroase câmpuri, datele personale ale unui utilizator sunt necesare pentru completarea automată a documentelor.

Între tabela „user” și tabela „userDetail” exist o relație de 1..n, semnificând că un utilizaor poate să aibă 1 sau n „userDetails”. Tabela „userDetail” a fost creață pentru a stoca date despre un administrator, care este un utilizator având rolul de „ADMIN”. Aceste date sunt necesare pentru a știi exact în ce oraș și la ce agenție lucrează, respectiv de care serviciu se ocupă, pentru a putea avea acces doar la programările de care este responsabil.

Între tabela „user” și „role” ar exista o relație many-to-many, semnificând că unui utilizator i se pot asocia mai multe roluri, acesta poate avea atât rolul de administrator, cât și de utilizator, iar un alt utilizator poate fi și el atât utilizator, cât și admin, din acest motiv a fost creață tabela „user\_role”, pentru a lega cele 2 tabele. Între tabela „user” și această există o relație de 1..n, respectiv între „role” și „user\_role” există o relație de 1..n.

Între tabela „user” și tabela „upload\_file” există o relație de 1..n, seminificand că un utilizator poate să își încarce 0 sau n documente.

Între tabela „user” și tabela „appointment” există o relație de 1..n, semnificând că un utilizator poate avea între 0 și n programări. Având în vedere că la crearea unei programări se pot selecta documente pentru a putea fi trimise în avans, optimizând în acest fel procesul, aceste documente fiind reprezentate prin tabela „upload\_file”, între aceste 2 tabele ar exista o relație de many-to-many (programarea „X” poate conține documentele „A” și „B”, iar porgramarea „Y” poate conține tot documentele „A” și „B”) , din acest motiv a fost creață tabela „documents\_sent”, legând astfel cele 2 tabele.

Tabela „document” caracterizează documentele ce sunt puse la dispoziția utilizatorului pentru a îi facilita vizita la ghiseu, sunt acele documente ce pot fi pre-completate cu datele utilizatorului.

Tabela „city” caracterizează orașul, între această și „agency” există o relație de 1..n, un oraș poate avea 1 sau oricâte agenții, respectiv între tabela „agency” și „service” există tot o relație de one-to-many, o agenție poate oferi unul sau mai multe serevicii.

Între tabela „institutie” și „agency” există o relație de one-to-many, o instituție poate avea mai multe agenții. (Spre exemplu: Instituția „Primarie” ce ține de domeniul executiv poate avea Primăria Timișoara, Primăria București, etc.)

1. **DETALII IMPLEMENTARE**
   1. **Implementarea pattern-ului MVC**

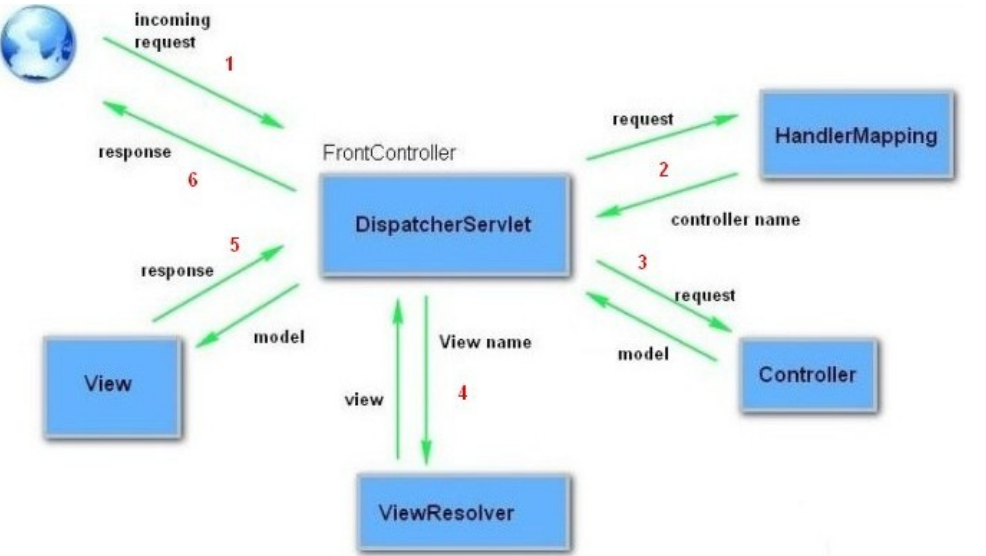
Cel mai des, aplicații Web folosesc patternul MVC (Model View Controller). Acest pattern te ajută să separi interfața utilizatorului de logica aplicației, să scrii cod mai organizat și ușor de întreținut.

Model – conține informațiile (datele), nu depinde de Controller sau de View.

View – afișează modelul date-lor (data model) și trimite acțiunile utilizatorului controller-ului. View-ul conține elementele din interfața utilizator.

Controller – într-o aplicație Web trimiterea acțiunilor utilizatorului Controller-ului este realizată prin request-uri HTTP, pe care Controller-ul le gestionează. Controller-ul primește cererile utilizatorului și le translatează în acțiuni, pe care Modelul le primește, apoi selectează View-ul care va fi afișat. Controller-ul gestionează comunicarea dintre model și view.

Spring MVC se bazează pe pattern-ul „Front controller” și pattern-ul MVC. Toate cererile sunt gestionate de un singur servlet numit DispatcherServlet, care se comportă ca un „front controller” în modulul Spring MVC. Servlet-ul „DispatcherServlet” invocă handler-ul „HandlerMapping” pentru a găsi un controller care să se ocupe de cerere(to handle the request). DispatcherServlet-ul expediază(trimite) cererea controller-ului, pentru a realiza operațiile(„buisness logic”) cu scopul îndeplinirii cererii utilizatorului. Controller-ul întoarce un obiect ce conține obiectul model și obiectul view(sau numele logic al view-ului). În Spring MVC obiectul returnat este reprezentat prin clasa ModelAndView. În cazul în care ModelAndView conține numele logic al view-ului, DispatcherServlet-ul invocă ViewResolver-ul pentru a afla obiectul view căruia îi corespunde numele. Ultima operație realizată de către DispatcherServlet, constă în trimiterea obiectului model obiectul view, ulterior fiind afișat utilizatorului.

[10]

Spring Boot nu se bazează pe configurația xml, el își configurează automat un dispatcher servlet.

* 1. **Definirea layer-ului de persistență și configurația pentru Hibernate**

Layer-ul de persistență se ocupă cu persistarea datelor (stocare și preluare) dintr-un „data store”, de exemplu o bază de date.

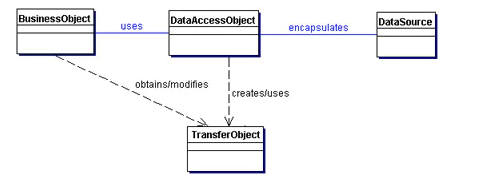
Hibernate este doar o implementare a layer-ului de persistență și se bazează pe specificațiile JPA.

DAO (Data Acces Object) este un șablon de proiectare ce separă API-urile low-level folosite pentru accesarea datelor de serviciile business high-level. DAO încapsulează toate accesele la bază de date.

Este format din 3 componente:

* + interfața DAO (definește operațiile standard ce pot fi realizate pe model)
  + clasa DAO concretă (implementarea interfeței DAO)
  + obiectele Model (POJO-uri ce conțin metode: gettere și settere, care stochează datele)

Următoare imagine prezintă pattern-ul DAO.

[11]

Fiecare tabel prezentat în capitolul 3, secțiunea 3.3, este reprezentat printr-o entitate. Entitatea este un obiect model, respectiv o componentă a pattern-ului DAO. Clasele sunt evidențiate prin folosirea adnotarii @Entity, ceea ce indică că este o entitate de tipul JPA. Nefolosindu-se adnotarrea @Table, fiecare entitate este mapata pe un tabel care va avea aceeași denumire ca și clasa. Fiecare clasă mapata are definită câte o interfața DAO, o interfață adnotată cu @Service și implementarea acestei interfețe. S-a ales aceata abordare de a separa service layer de DAO layer, pentru a se putea scrie cod modular și ușor de înlocuit (spre exemplu : înlocuirea bazei de date). DAO-ul este limitat doar la operațiile de add/update/insert/select obiecte „Entity” în/din baza de date. Layer-ul Service este cel care se ocupă de logică, manevrând datele ce se trimit(schimbă) între DAO și client.

**Definirea configurației pentru Hibernate**

În /resource, în fișierul application.properties se definește configurația pentru Hibernate. Se va seta URL pentru bază de date.

* 1. **Portalul utilizatorului**

În acest capitol se vor prezenta o parte dintre funcționalitățile aplicației și modul de implementare al acestora.

Aplicațiile ce sunt implementate folosind Spring acceptă o varietate de de view-uri, deseori denumite șabloane view(„view templates”). Aceste tehnologii sunt denumite șabloane(„templates”) deoarece expun atributele model-ului în view folosind un limbaj de marcare „during server-side rendering” [12]. (Server-side rendering este cea mai comună metodă pentru a afișa informație pe ecran. Convertește fișierele HTML pe partea de server în informație folositoare pentru browser.)

* + 1. **Autentificarea unui utilizator**

Pentru accesarea aplicație este necesară autentificarea utilizatorului. În metoda configure(HttpSecurity http), oferită de WebSecurityConfigurerAdapter s-au definit anumite pattern-uri URL pe care orice utilizator le poate accesa, nefiind nevoie de autentificare. Orice utilizator poate accesa un request dacă URL începe cu /ccs, /js, /logout, /images, etc. În metodă configure(HttpSecurity http) s-a definit posibilitatea utilizatorilor de a se autentifica folosind „form based login”, definindu-se atât pagina de autentificare, cât și pagina de logout-ul. Folosind metoda „.permitAll()” li se permite tuturor utilizatorilor să acceseze pagina de autentificare. Capabilitățile logout-ului sunt aplicate în mod automat când se folosește WebSecurityConfigurerAdapter. În mod implicit accesând URL /logout utilizator va fi deconectat prin invalidarea sesiunii HTTP, se va curăța orice configurație RememberME și holder-ul SecurityContextHolder.

Una dintre funcționalitățile de care dispune un utilizator este persistența identității sale. „Remember-me” sau persistența identității se referă la un site web care este capabil să țină minte identitatea unui „principal” între sesiuni. În aplicația „STOP COZI” s-a definit un checkbox, pentru ca utilizatorul să poată selecta opțiunea „Tine-mă minte!”. Dacă utilizatorul selectează opțiunea înainte autentificării, Spring trimite un „remember-me” cookie browser-ului, în plus față de „session cookie”. Odată ce session cookie expiră, dacă utilizatorul accesează o pagină securizată, va fi automat autentificat folosind „rememeber-me cookie”. Acest proiect folosește „Simple hash-base token approach”. Se folosește o strategie hash pentru a crea token-ul. Token-ul este creat folosin numele utilizatorului (username-ul), timpul de expirare, parolă și o cheie. După o autentificare realizată cu succes, un cookie folosind valoarea token-ului este trimis browser-ului.

****

* + 1. **Înregistrarea unui utilizator**

Pentru a se înregistra utilizatorul apasă butonul „Creeaza-ți cont”, accesând pagina <http://localhost:8181/signup> .

Următoarea imagine prezintă formularul de înregistrare, pentru întreagă pagină vezi secțiunea 5. Este implementat folosind bootstrap, codul este preluat de pe <https://bootsnipp.com/snippets/d5VXP> și modificat.



În Spring cererile (request-urile) HTTP sunt gestionate de un Controller. „HomeController” este cel care gestionează cererea GET (GET request) pentru /signup, returnând numele View-ului, în acest caz „signup”. Controller-ul este ușor de identificat fiind adnotat cu @Controller, folosind această adnotare clasa „HomeController” va fi înregistrată ca un bean în container-ul Spring. (Un bean este un obiect care este instanțiat, asamblat și administrat de container-ul Ioc Spring (Ioc – Inversion of Control)). Pentru a mapa metodele din acest controler s-a folosit adnotarea @RequestMapping care în mod implicit mapeaza toate operațiile HTTP, cum ar fi GET, POST. Două dintre metode sunt mapate: „/signup”, o metoda este folosită pentru operația de GET (numele metodei: „signup”), iar cealaltă pentru operația de POST(numele metodei: „signupPost”). Metodă „signup” folosește un obiect Model pentru a expune un nou utilizator template-ului view. După cum se poate observa în imagine, obiectul utilizator conține câmpuri: nume, prenume, telefon etc, acestea corespund câmpurilor formului în view-ul „signup” și vor fi folosite pentru a captura datele introduse în form. Numele view-ului „signup” este convertit într-un template pentru a fi redat. Thymeleaf se ocupă de parsarea template-ului „signup.html” și de a reda formularul, deasemenea Thymeleaf posedă un atribut th:field, acesta se ocupă de binding-ul valorii introduse cu o proprietate din „form-backing bean”.[12]

Expresia ”th:action=’@{/signup}’” trimite formularul („submit”) la endpoint-ul „signup” folosind POST. Metoda „signupPost” primește obiectul user a cărui atribute au fost populate la completarea formularului. Dacă numele utilizatorului sau adresa de email există, utilizatorul va fi semnalat printr-un mesaj de eroare. Datele vor fi salvate în baza de date, atribuindu-se utilizatorului automat rolul de utilizator („USER\_ROLE”). Parola va fi criptată folosind BcryptPasswordEncoder, fiind unul dintre cele mai bune mecanisme de criptare a parolei ofrite de Spring Security. Bcrypt generează intern un „salt”, însemnând ca fiecare parolă va fi criptată în mod diferit.

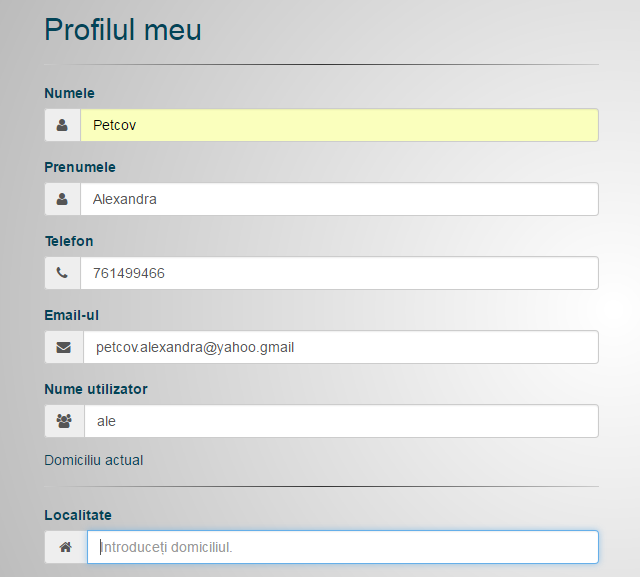
Fiecare input din formulare are setat atributul „required”, pentru a indica că este necesară completarea lui. Formularul conține un checkbox, în funcție de valoarea acestuia, type-ul input-ului ce corespunde parolei va fi de tipul text sau password. De modificare type-ul este responsabilă o funcție JavaScript implementată folosind biblioteca Jquery, care folosește metoda .bînd() pentru a atașa un „event handler” checkbox-ului.

* + 1. **Editare profil**

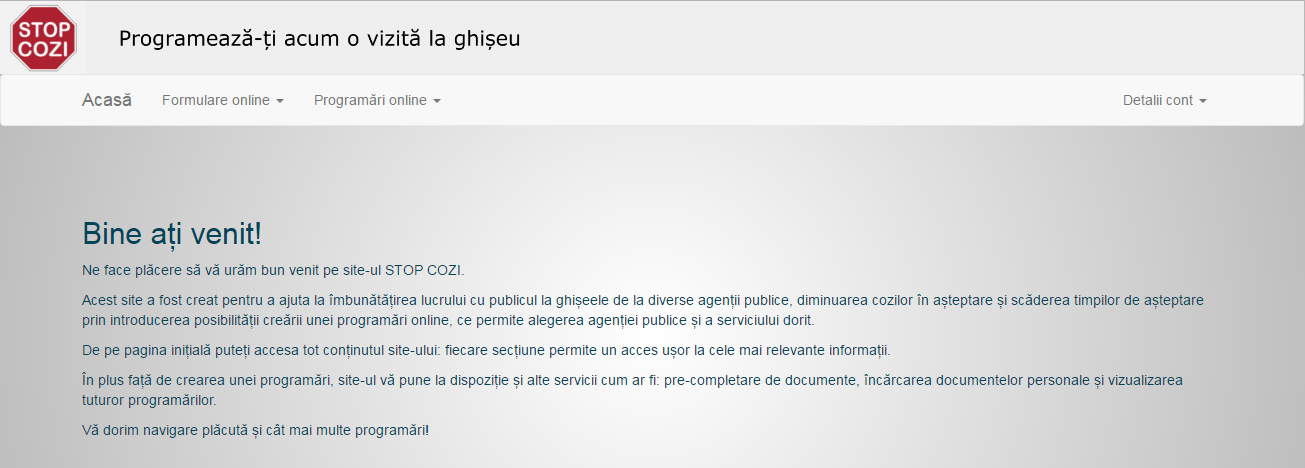
După cum se poate observa în următoarea imagine, fiecare utilizator poate să își modifice profilul. Pentru întreagă pagină vezi secțiunea 5. Pe bază informațiilor introduse, se vor pre-completa documentele puse la dispoziția utilizatorului pentru a îi facilita rezolvarea problemelor într-un timp cât mai scurt.

Pagina accesată este: <http://localhost:8181/user/profile> .

Controller-ul „UserController” este cel care se ocupă de gestionarea cererilor GET și POST, cele două metode fiind („profile” și „postProfile”). Metoda „profile” pe lângă parametrul de tip Model, are și un parametru de tip Principal (oferit de Spring Security). Principal reprezintă utilizatorul logat. Metoda „postProfile” primește obiectul user a cărui atribute au fost modificate, cu ajutorul unui obiect de tipul UserService (folosind @Autowired application context (container) a injectat o instanță UserService), modificările vor fi salvate în baza de date.

****

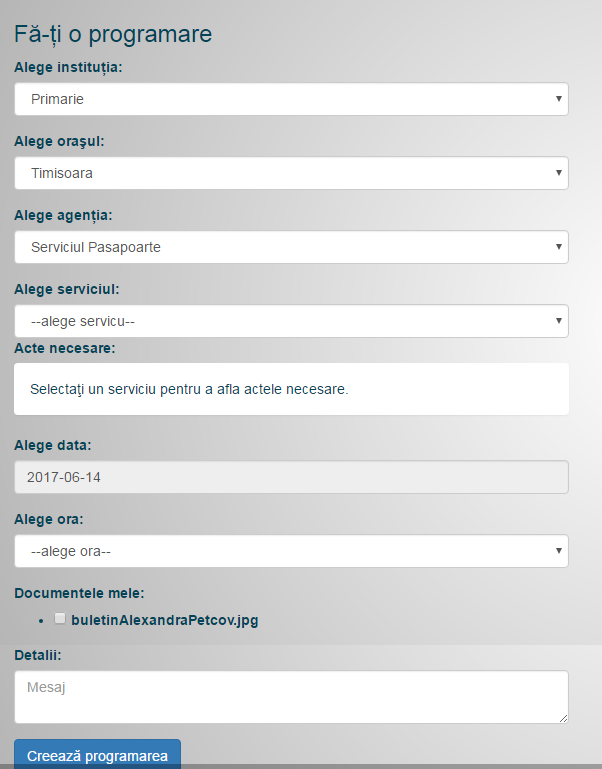
* + 1. **Prima pagina**

****

În figura de mai sus este prezentată pagina „Acasă” a aplicație.

Adresa URL: <http://localhost:8181/userFront> . Aceasta poate fi accesată apăsând „Acasă”, după ce ne-am creat o programare suntem redirecționați pe această pagină. „Acasă” este prima pagină care este afișată după autentificare. Controller-ul ce se ocupă de gestionarea cererilor este tot „HomeController-ul”.

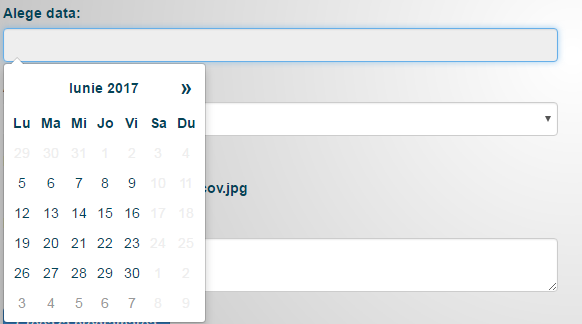
* + 1. **Crearea unui programări**



După cum se vede din captura de ecran de mai de sus, utilizatorul trebuie să selecteze instituția, orașul, agenția, serviciul și data pentru a își putea face o programare. Opțional poate alege să trimită și un mesaj, poate selecta și documentele pe care dorește să le trimită.

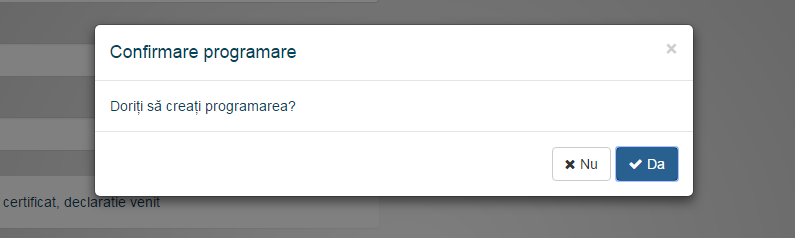
Adresa URL: <http://localhost:8181/appointment/create>

„AppointmentController-ul” este cel care se ocupă de gestionarea cererilor de tipul HTTP (GET și POST). Fiecare tag „select” are setat atributul „required”, astfel programarea nu se poate realiza până nu sunt completate toate câmpurile ce sunt obligatorii. Dropdown-urile corespunzătoare instituției și orașului sunt populate la încărcarea paginii. În funcție de instituția și orașul selectat, se va popula dropdown-ul corespunzător agenției, în funcție de agenția aleasă se va popula dropdown-ul corespunzător serviciului, iar în funcție de data, agenția și seriviciul ales se va popula dropdown-ul corespunzător orei. Se vor afișa doar orele disponibile, dacă utilizatorul logat are o altă programare pentru o anumită oră, ora accea nu va fi afișată. Pentru alegerea datei s-a folosit un bootstrap datepicker.



Popularea dropdown-urilor în funcție de valoarea aleasă anterior se face în felul următor: se folosește funcția JQuery „.change(handler)” pentru a atașa un handler unui element, când valoarea elementului se modifică, se va trimite o cerere la server, care va returna un JSON ce conține valorile pentru următorul dropdown(de exemplu: la alegerea/schimbarea orașului, se va returna o lista ce conține agențiile). În urma răspunsului, elementele de tipul „<option>” sunt create folosind JavaScript. La apăsarea butonului „creare programare”, dacă toate câmpurile sunt completate va apărea o căsuță de confirmare. Vezi următoarea imagine. Această este realizată folosind funcția confirm din bibliotecă bootbox. Bootbox.js este o bibliotecă JavaScript, ce permite crearea căsuțelor dialog folosind modale Twitter Bootstrap, fără a fi nevoit să creezi sau ștergi elementele DOM sau event handler-ele JS.

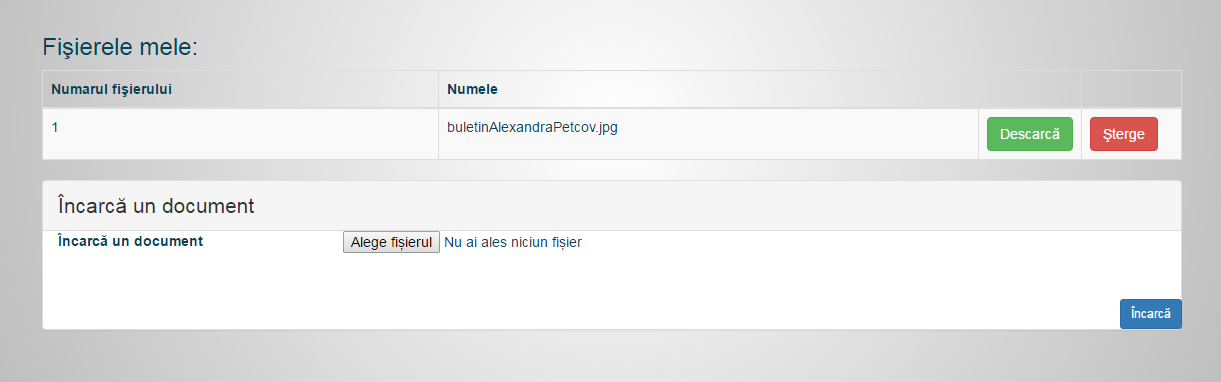
Adresa URL: <http://localhost:8181/appointment/create>



* + 1. **Încărcarea documentelor personale**

I se oferă utilizatorului posibilitatea de a-și încărca documente personale de care s-ar putea să aibă nevoie pentru rezolvarea unei programării. Documentele pot fi încărcate, descărcate și șterse. Dacă utilizatorul dorește ștergerea documentului, acestea nu vor fi șterse din bază de date, dar utilizatorul nu le mai poate vedea. Se folosește interfață MultipartResolver, oferită de Spring, pentru încărcarea fișierelor.

Controller-ul „UploadFileController” se ocupă de gestionarea cererilor HTTP. Metoda handleFileUpload() de salvarea fișierelor în bază de date, folosind un obiect de tipul UserService.

****

Descărcarea fișierelor: s-a folosit HttpServletRespose pentru a scrie un fișier direct în ServletOutputStream folosind FileCopyUtils. Pentru ca browser-ul să știe despre ce fișier este vorba, s-au setat headere-le: Content-Type, Content-Disposition și Content-Length. Header-ul Content-Disposition îi spune browser-ului care va fi numele fișierului descărcat și dacă fișierul va fi descărcat ca un ‚attachment’ sau va fi afișat în browser.

Adresa URL: <http://localhost:8181/upload/uploadFile>

* + 1. **Generarea personalizată a documentelor**

Acesta pagină afișează documentele puse la dispoziția utilizatorului, care vor fi completate automat cu informațiile luate din profilul utilizatorului. Documetele au extensia .ftl, deoarece s-a folosit FreeMarker Template Language la crearea acestora.

Apache FreeMarker este un „server-side Java template engine”: o bibliotecă Java folosită pentru generarea de texte (pagini web HTML, e-mail-uri, fișiere de configurare, cod sursă) bazată pe template-uri. Template-urile sunt scrise folosind FreeMarker Template Language (FTL), care este un limbaj simplu. [14]

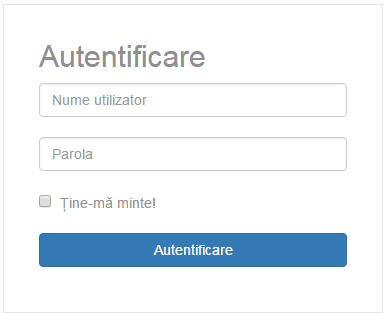
Descărcarea documentelor este la fel ca cea explicată mai sus. Vezi secțiunea 4.3.6. Pentru vizualizarea documentelor header-ul Content-Disposition îi va spune browser-ului că se dorește afișarea acestuia, folosind parametrul „inline”. Înainte de descărcarea sau vizualizarea documentului,va fi apelată metodă „updateDocument” care se va ocupa de inserarea datelor în fișier. Controller-ul ce se ocupă de gestionarea cererilor HTTP (GET și POST) este „GenerateDocumentController”, care are trei mapari, una pentru descărcarea documentelor, una pentru vizualizarea acestora, iar a treia pentru a lista toate documentele disponibile.

****

Adresa URL: <http://localhost:8181/generateDoc/document>

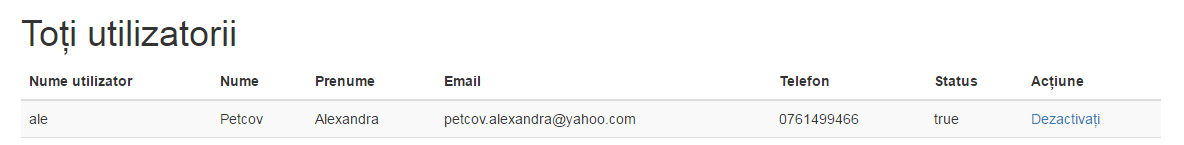
* 1. **Portalul funcționarului**
     1. **Autentificare**

Se folosește o variabila având tipul boolean, „loggedIn”, dacă aceasta are valoarea true, se va afișa prima pagină care conține următorul mesaj: „Bine ați venit!”, în caz contrar se va afișa formularul de autentificare. Dacă valoarea variabilei este true, se vor afișa serviciile: „Iesire”, „Programari”, respectiv „Utilizatori”, în caz contrar nu vor fi afișate. Pentru această se folosește „style.display = none”. Proprietatea „display” permite afișarea sau ascunderea unui element. Pentru a salva valoarea variabilei se folosește localStorage. Se folosește un obiect de tipul LoginService pentru a trimite datele backend-ului. Pentru că se cere o resursă de la un alt domeniu(alt port), se realizează o cerere de tipul CORS (cross-origin HTTP request), din această cauza este necesară activarea CORS pe server. Pentru a activa CORS, s-a implementat interfață Filter, oferită de Spring. Se vor trimite: numele utlizatorului, respectiv parola, acestea vor fi validate de către Spring Security. Dacă datele sunt valide, Spring Security va trimite că răspuns un JSessionId.



* + 1. **Vizualizarea utilizatorilor**

Vizualizarea utilizatorilor este permisă doar administratorului care are rolul de „SUPERADMIN”. Pentru a implementa aceasta restricție ne vom folosi de adnotarea @PreAuthorize si expresia hasRole(role), oferită de Spring Security.

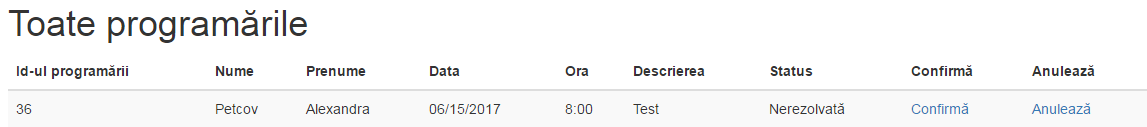


Pagina prezintă un tabel simplu unde sunt afișate datele despre fiecare utilizator, cu posibilitatea activării, respectiv dezactivării contului. Dacă contul este dezactivat, utilizatorul nu se mai poate autentifica. Se folosește un obiect de tipul UserService pentru a trimite datele backend-ului. S-au definit trei metode, o metodă pentru obținerea listei de utilizatori, iar celalate două metode pentru activarea, respectiv suspendarea contului unui utilizator. Pentru a avea acces la aceste resurse, în partea de backend acestea au fost definite că REST API. În Spring, cererile HTTP sunt gestionate de un controler, pentru a construi servicii web RESTful se folosește adnotarea @RestController. Răspunsul este convertit în format JSON, de convertirea acestuia se ocupă Spring.

Un REST API este un API (application program interface) care folosește cereri HTTP pentru a manipula date folosind GET, PUT, POST, DELETE. Un RESTful API se bazează pe tehnologia REST (Representational state transfer), un stil arhitectural.

* + 1. **Vizualizarea programărilor**

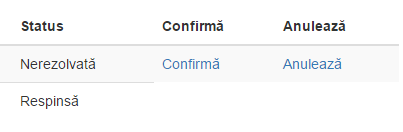
Vizualizarea programărilor este permisă doar administratorului care are rolul de „ADMIN”. Pentru a implementa aceasta restricție ne vom folosi de adnotarea @PreAuthorize si expresia hasRole(role), oferita de Spring Security.

****

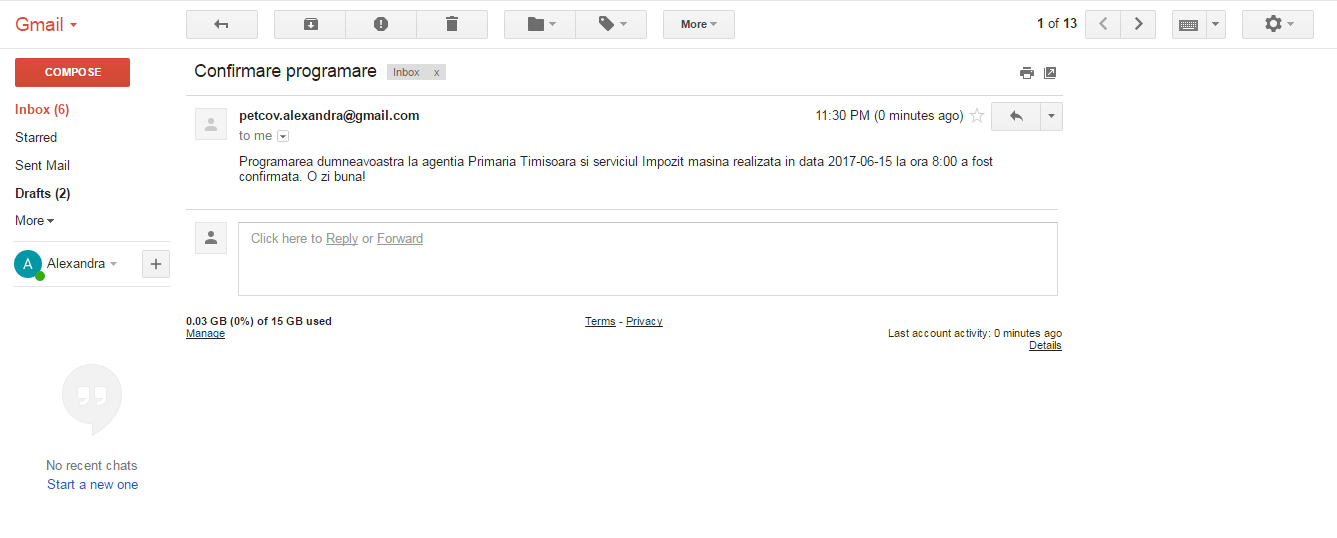
Pagina prezintă un tabel simplu unde sunt afișate datele despre fiecare programare, cu posibilitatea confirmării, respectiv anulării unei programări. Se folosește un obiect de tipul AppointmentService pentru a trimite datele backend-ului. S-au definit două metode, o metodă pentru obținerea listei programărilor, iar cealaltă metodă pentru confirmarea/anularea unei programări.

* + - 1. **Confirmarea / Anularea unei programări**

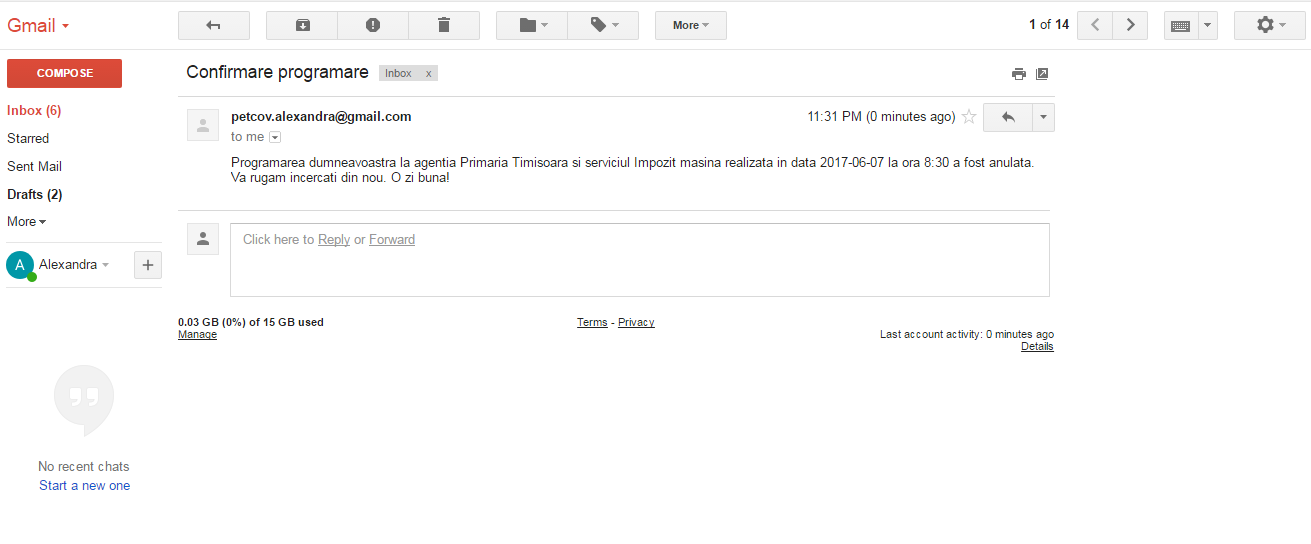
O programare poate fi anulată sau confirmată. În urma acțiunii alese de către funcționar, un mail este trimis utilizatorului pentru a îl înștiința despre decizia luată. Odată ce s-a ales o acțiune, nu este posibilă revocarea acesteia.

****

Dacă programarea este confirmată utilizatorul va primi următorul email.

****

Daca programarea este anulată, utilizatorul va primi următorul email.

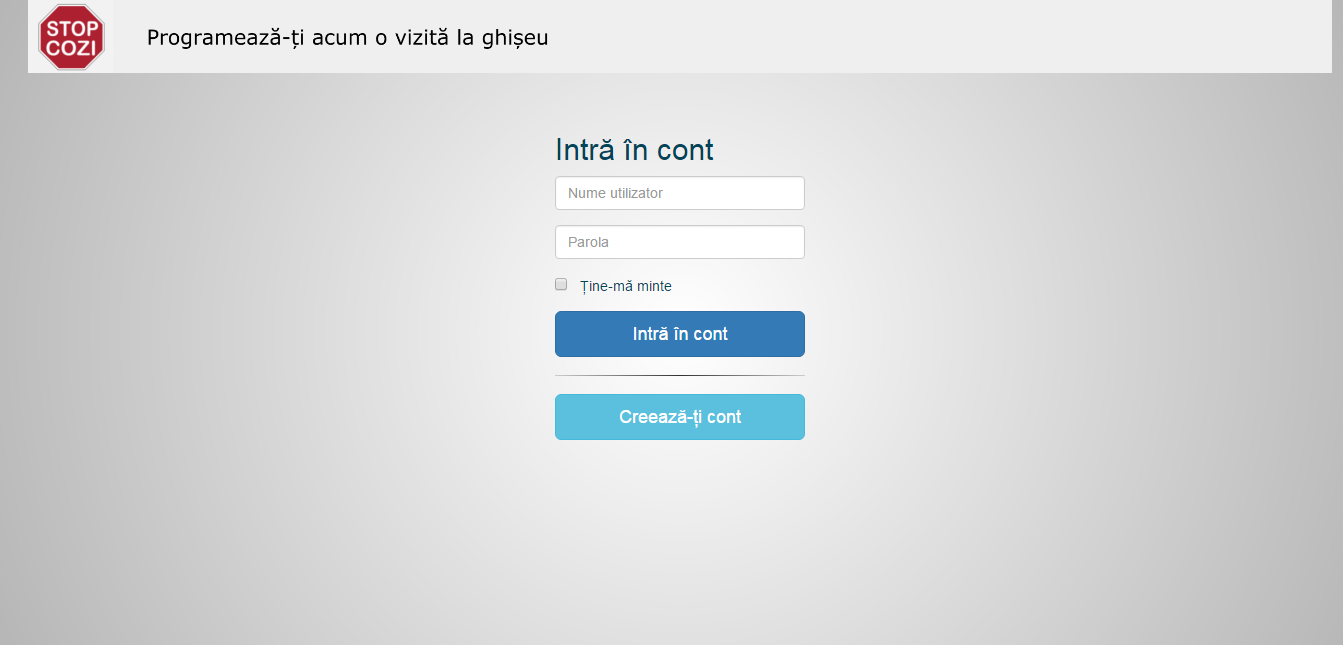
****

1. Trimiterea unui email din partea de backend a aplicației este unul dintre cele mai comune cazuri în lumea aplicațiilor enterprise. Pentru a folosi Spring în trimiterea unui email, în fișierul pom.xml trebuie adăugată dependența „spring-starter-email”. În application-properties au fost configurate proprietățile „spring-starter-email”, spre exemplu: portul, hostul, protocolul, username-ul, password-ul. S-a folosit host-ul Gmail SMTP (Simple Mail Transfer Protocol), iar în application-properties a fost configurat server-ul Gmail SMTP. Pentru a se folosi implementarea JavaMailSender, s-a injectat o instanță în clasă „SendEmail.java”, această va fi responsabilă de trimiterea mail-ului. Metoda send() este supraîncărcata și acceptă diferite tipuri de parametrii. S-a folosit SimpleMailMessage, după cum numele sugerează este un model simplu de email, doar cele mai comune proprietăți i se pot seta. Nu permite modificarea header-ului mesajului și transporta doar conținut text simplu.
2. **5. UTILIZAREA APLICAȚIEI**

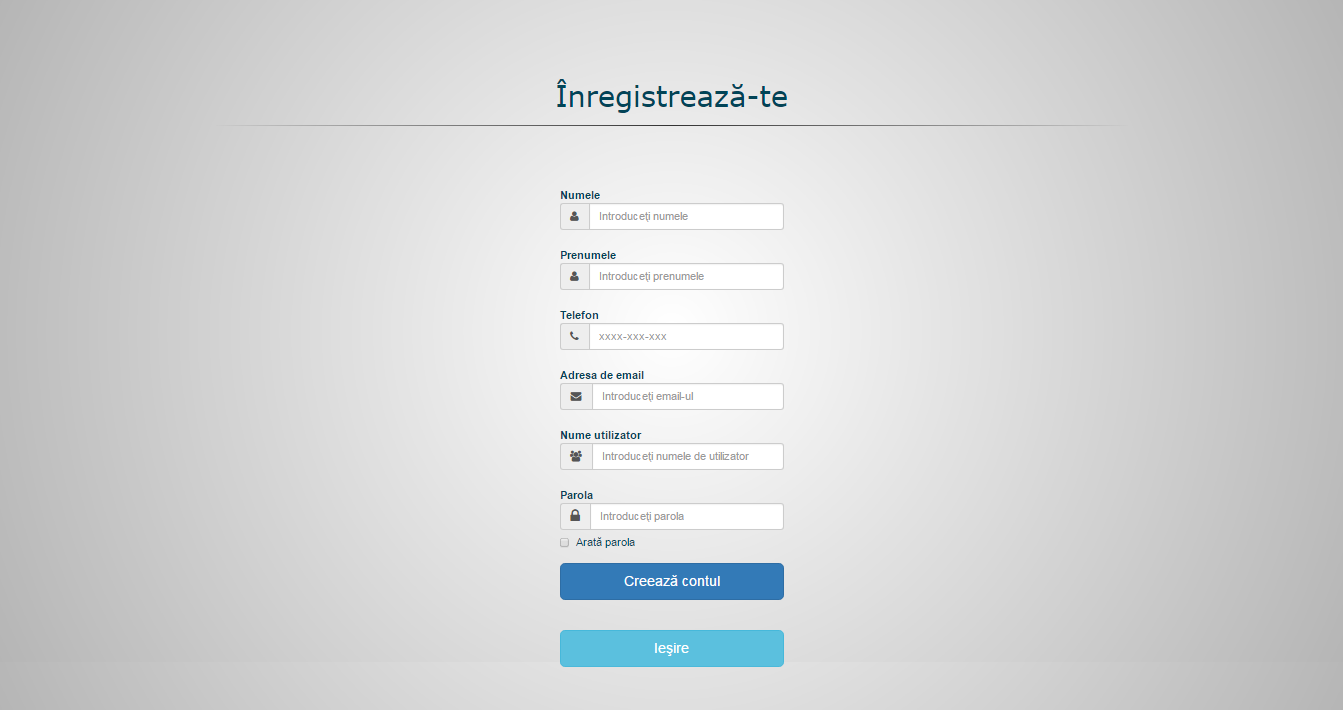
**5.1 Scenarii utilizare – screenshots**

**Portalul utilizatorului:**

1. Autentificare



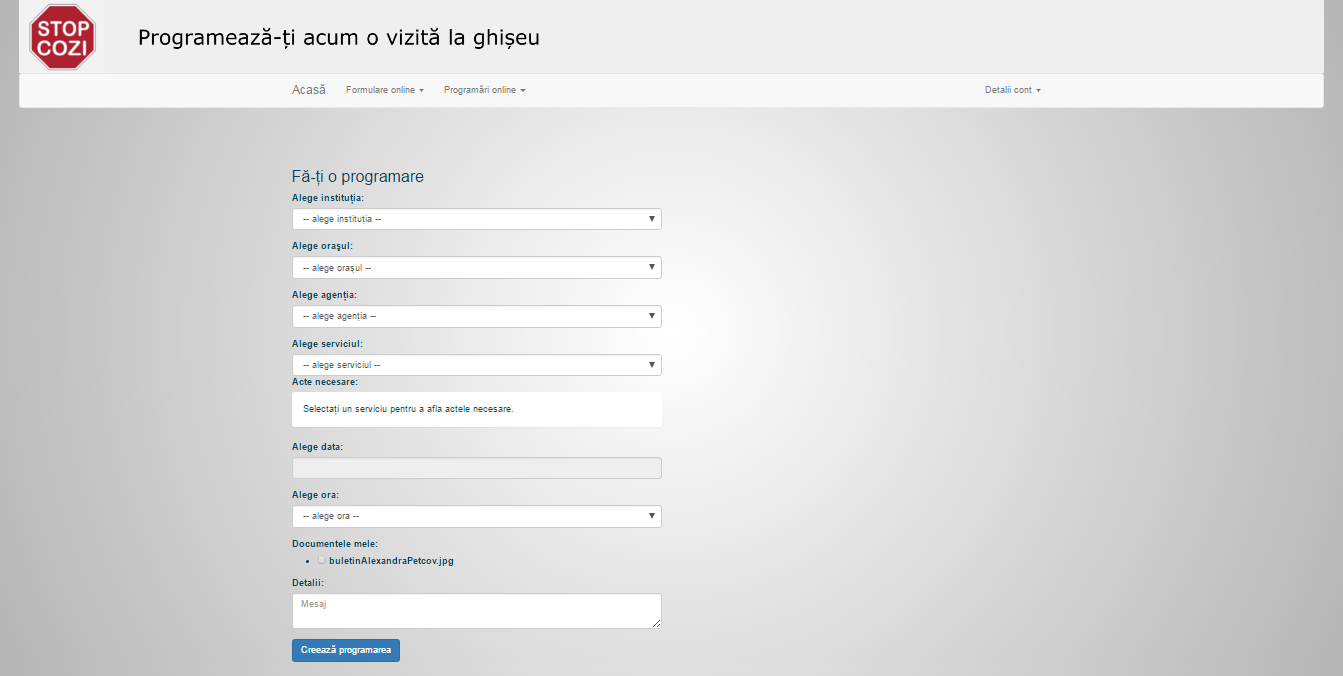
1. Înregistrare



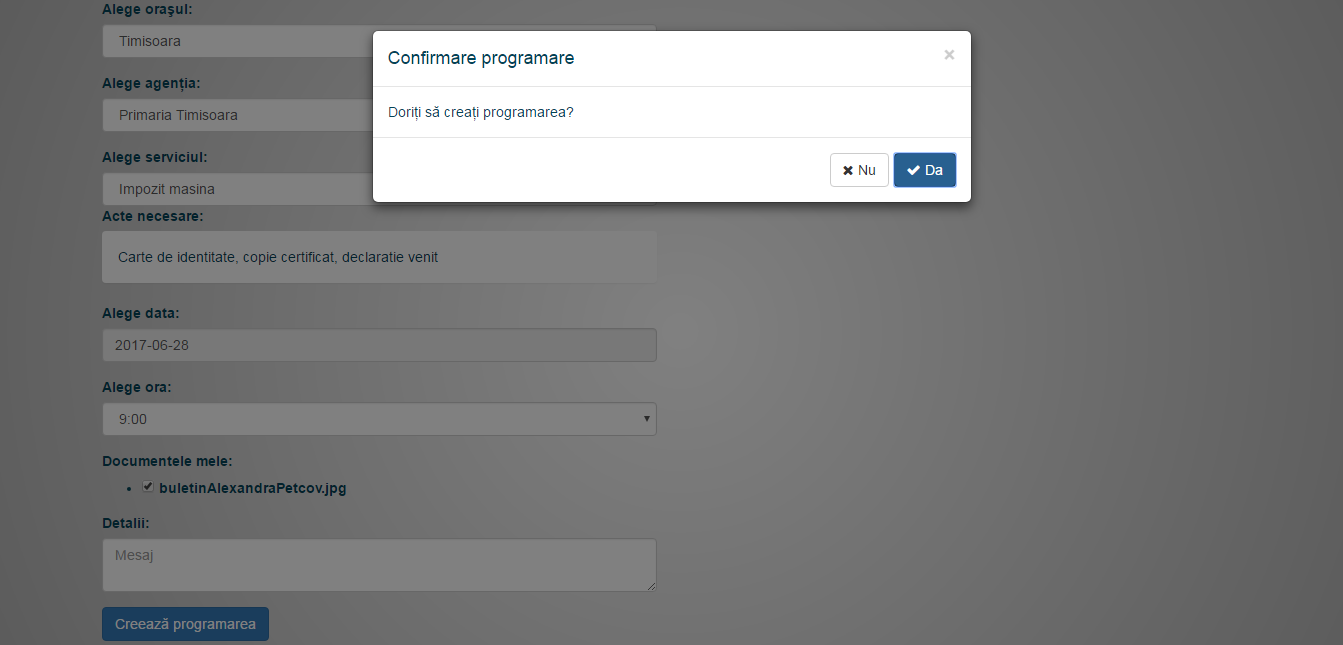
1. Acasă



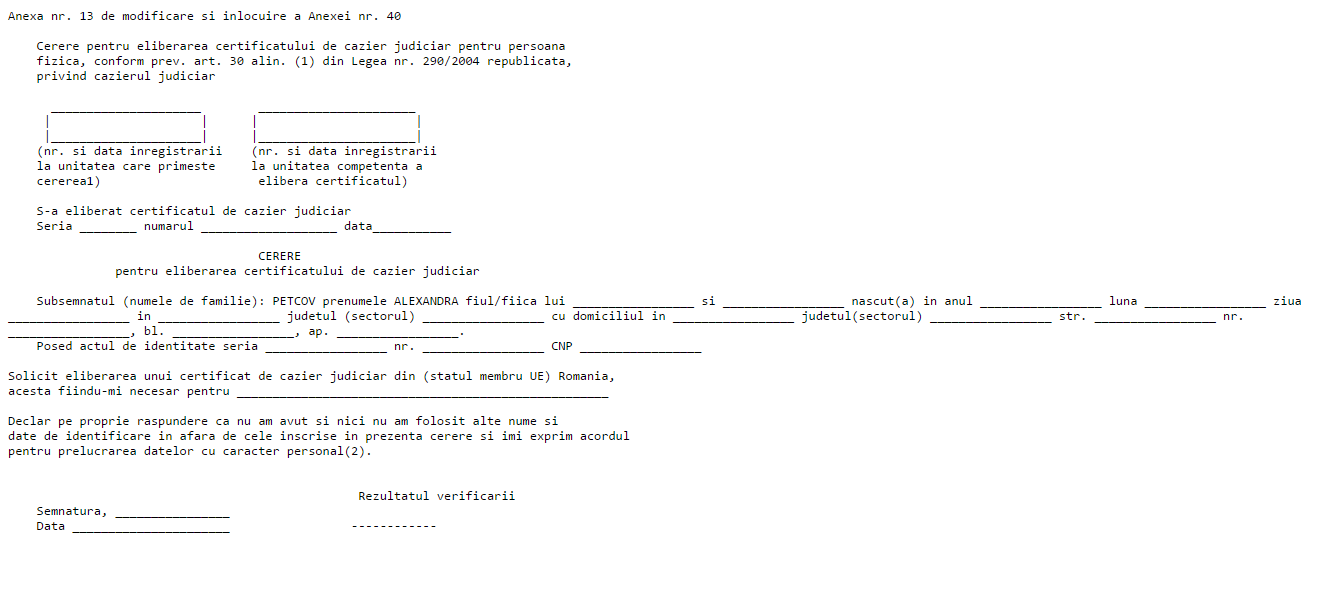
1. Creează programare



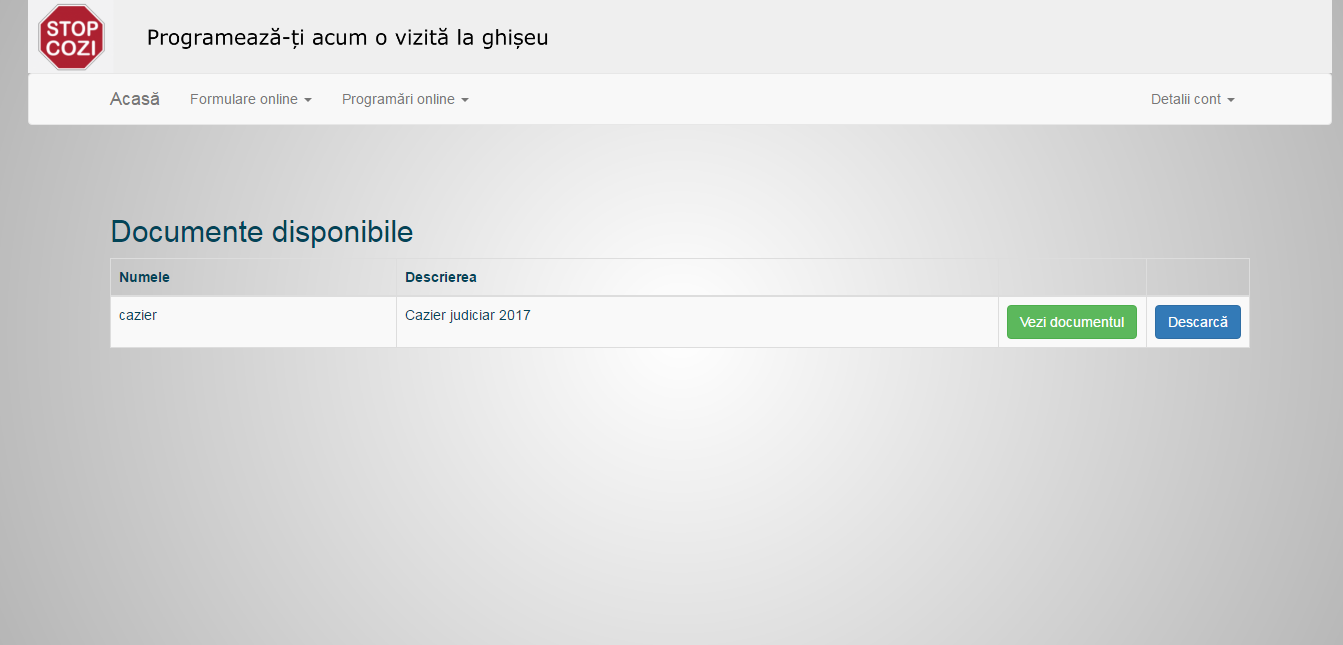
1. Confirmare programare



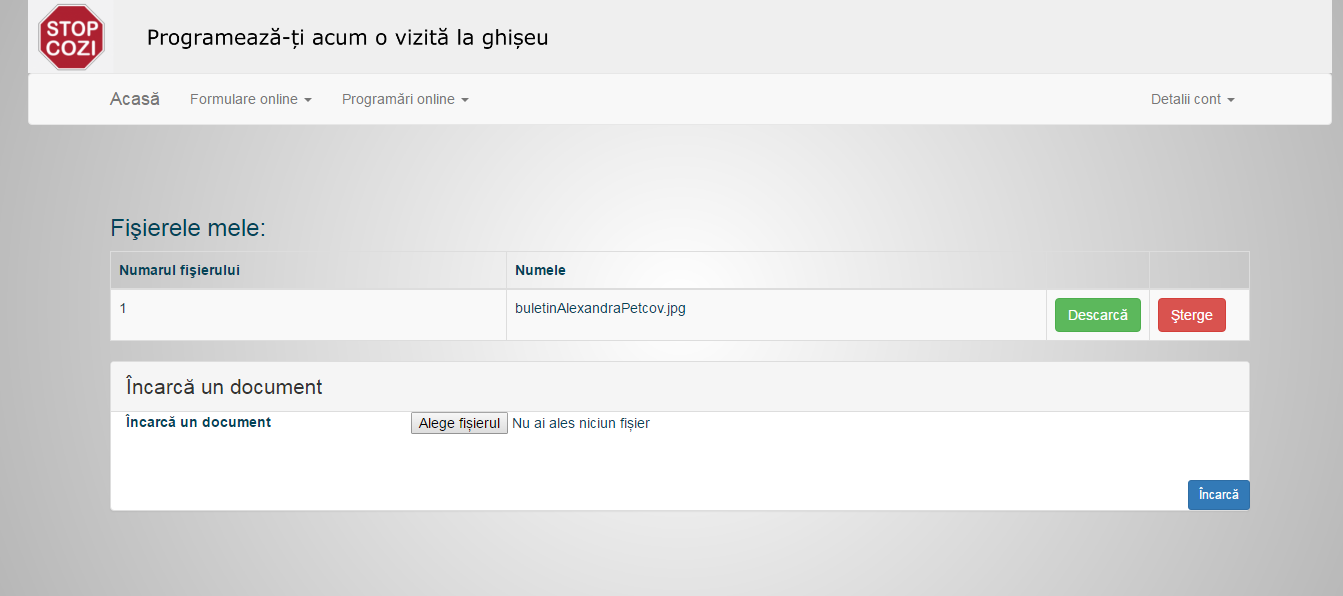
1. Exemplu: generare documente pre-completate



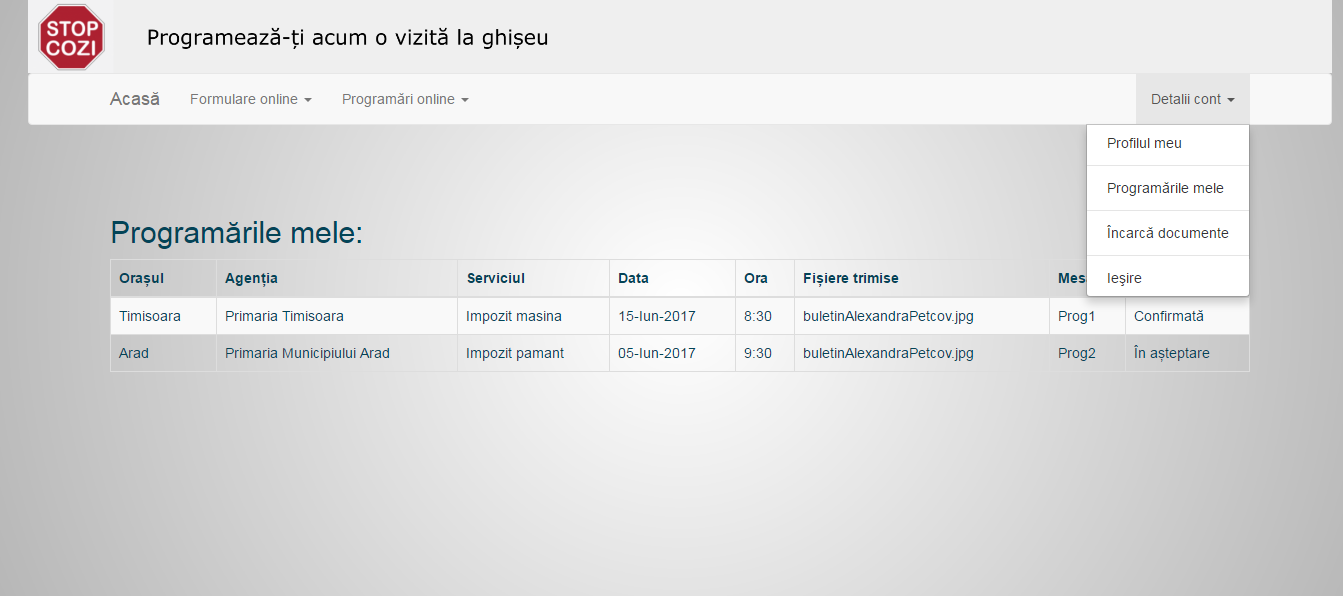
1. Vizualizarea documentelor

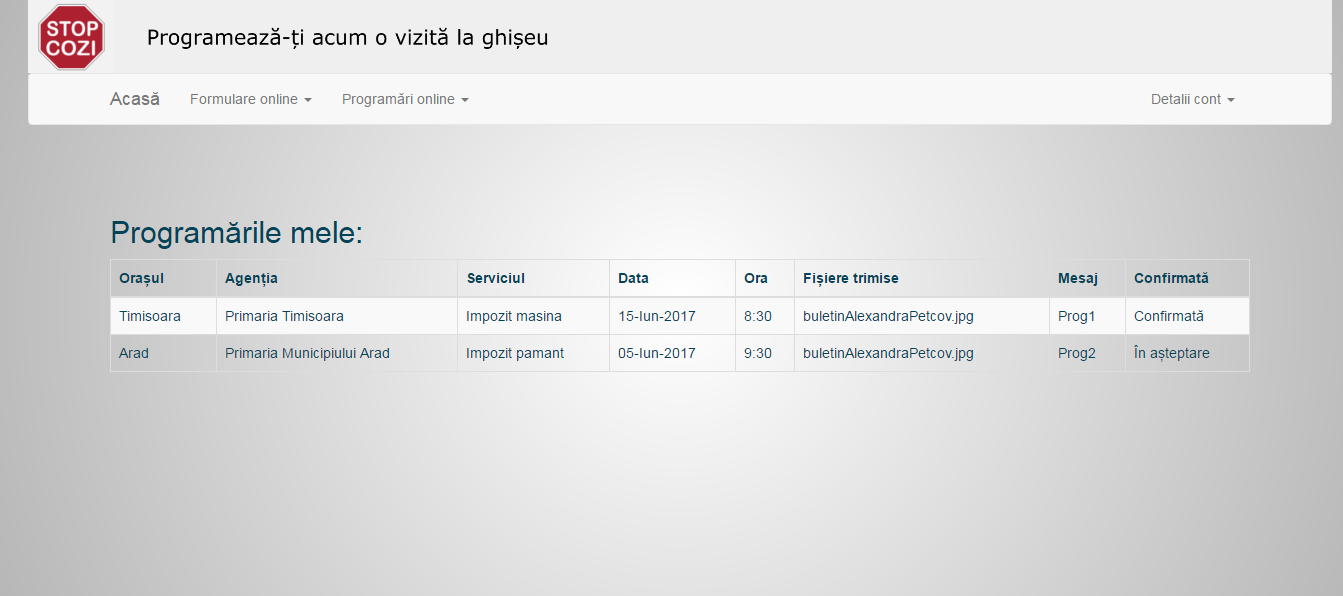


1. Încarcarea si vizualizarea documentelor personale

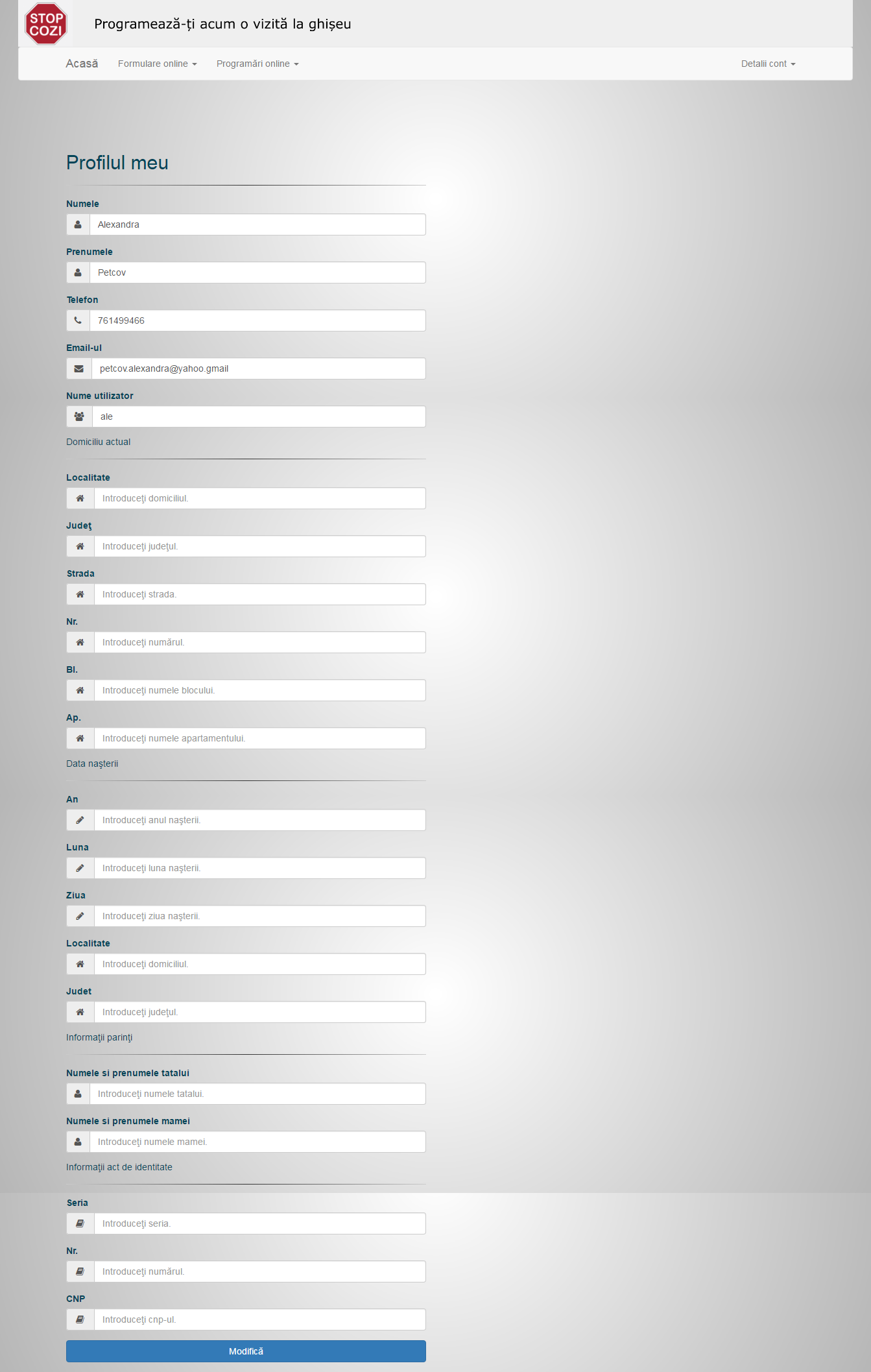


1. Vizualizarea programărilor si meniul pentru accesarea diferitelor paginilor



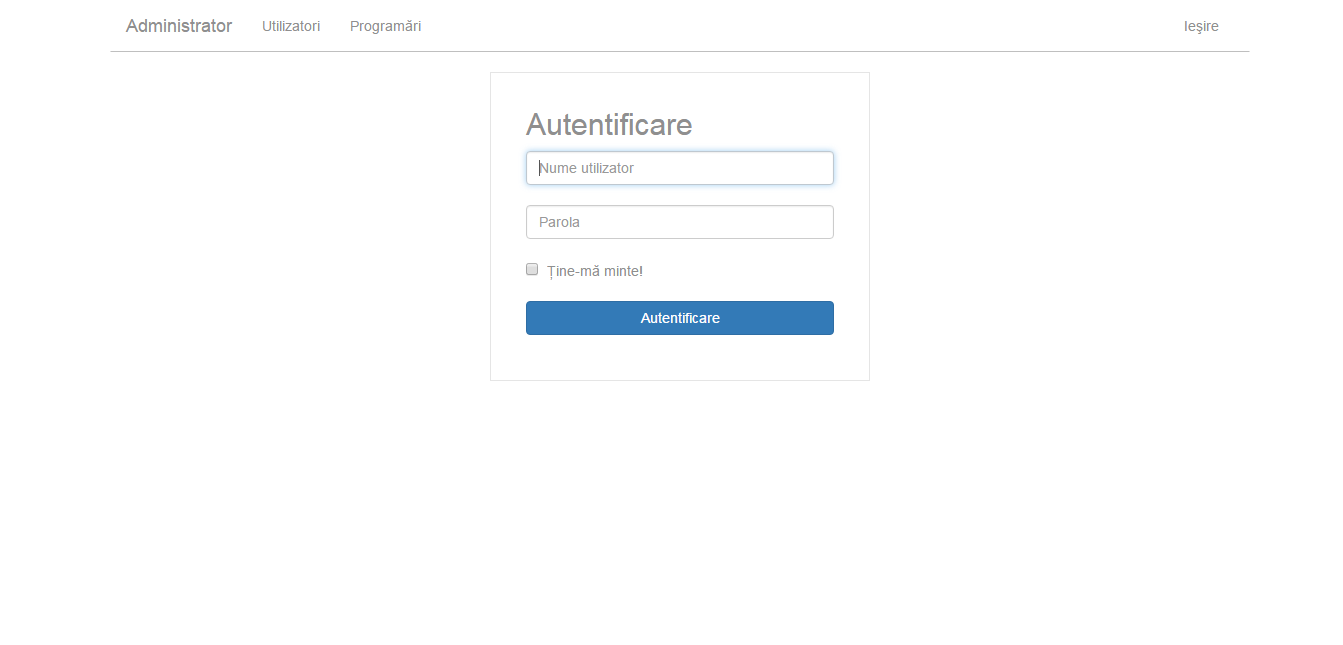


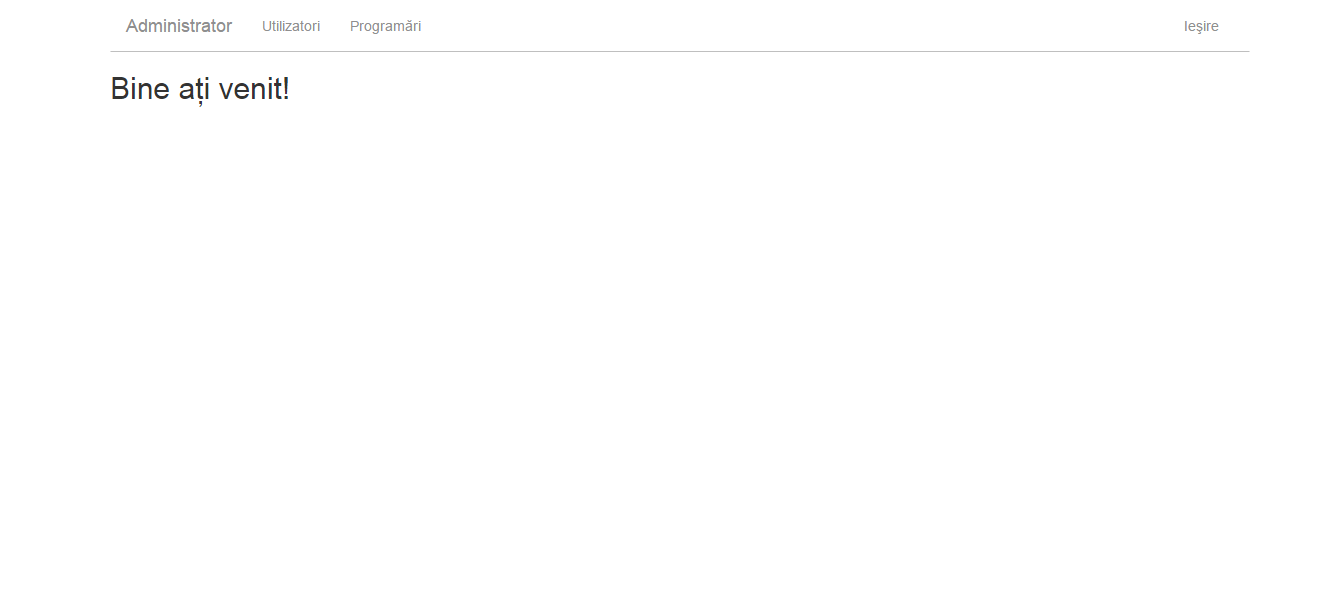
1. Accesarea profilului utilizatorului

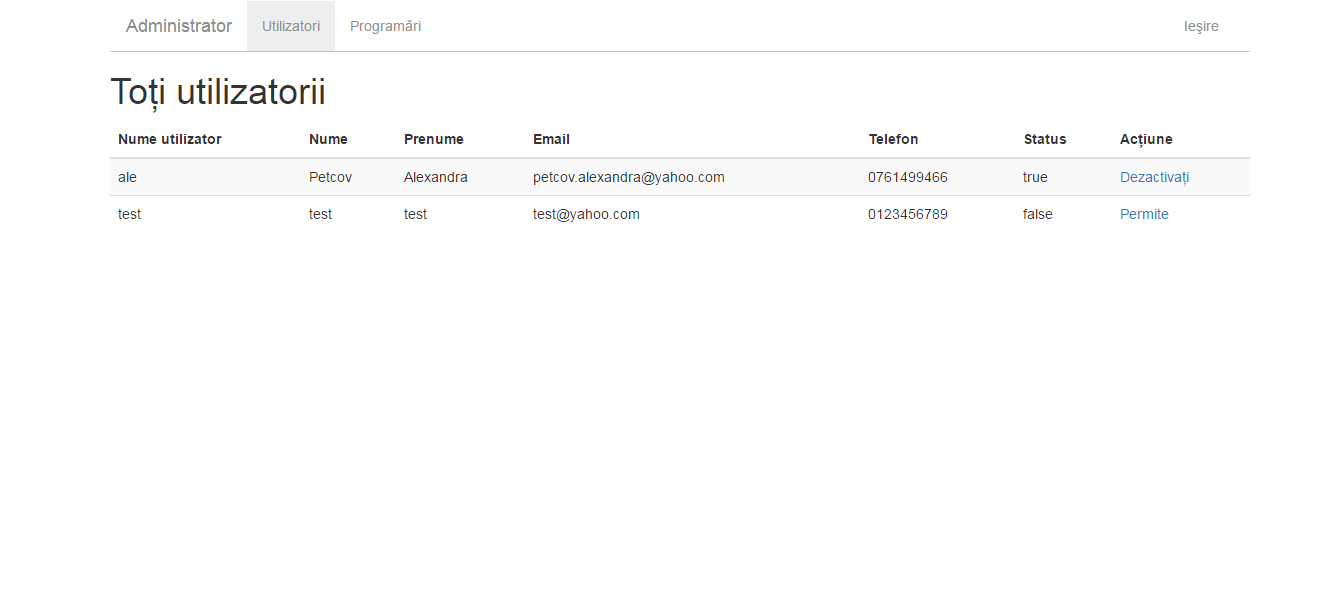


**Portalul administratorului(funcționarului):**

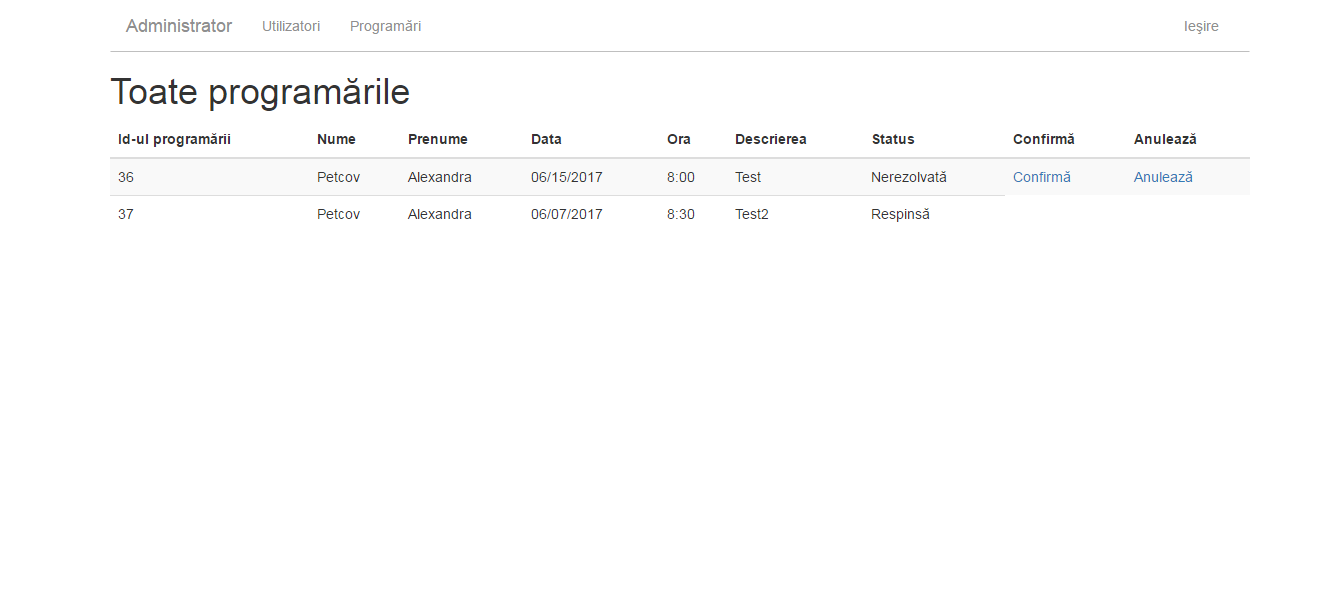
1. Autentificare



1. Pagina administratorului
2. Vizualizarea utilizatorilor



1. Vizualizarea programărilor



1. **CONCLUZII**

* 1. **Concluzii**

În fiecare zi, cetățeanul român, în interacțiunea cu instituțiile publice, este încă nevoit să stea la cozi interminabile, fiind trimis de la un ghișeu la altul, de la o instiutie la alt pentru a își rezolva problema. Inevitabil la aceste ghișee se formează cozi în așteptare interminabile, care duc la o creștere a timpului de așteptare și implicit a timpului necesar rezolvării problemei. Pentru rezolvarea unei probleme este necesară completarea unui formular de mână, lipirea unui timbru, fotocopii ale documentelor necesare, acest lucru ducând la îngreunarea livrării eficiente și într-un timp scurt al serviciului.

Aplicația „STOP COZI” este un sistem complex, ce constă într-o aplicație web duală, o aplicație destinată utilizatorilor, pentru crearea programărilor și o aplicație pentru rezolvarea (confirmarea, anularea) programărilor pentru funcționarii publici, respectiv consultarea utilizatorilor cât și a programărilor. Aplicația poate fi adaptată ușor pentru orice agenție.

Aceast proiect și-a propus îmbunătățirea lucrului cu publicul la ghișee de la diverse agenții, orașe, diminuarea cozilor în așteptare și scăderea timpilor de așteptare.

Obiectivele au fost atinse prin introducerea posibilității creării unei programări online, ce permite alegerea instituției, orașului, serviciului, datei și orei, informând utilizatorul despre actele necesare realizării serviciului, oferindu-i posibilitatea trimiterii în avans a documentelor personale, astfel se va evita pierderea timpului pentru realizarea diferitelor fotocopii, ușurând vizita la ghișeu. Aplicația pune la dispoziția utilizatorului pre-completarea documentelor. La confirmarea sau respingerea unei programării de către funcționar, utilizatorul este înștiințat printr-un email. Utilizatorul i se oferă posibilitatea vizualizării programărilor pentru a putea verifica în ce data a fost sau va fi o programare.

Aplicația web destinată funcționarilor permite consultarea și rezolvarea programărilor, respectiv consultarea utilizatorilor.

Dezvoltarea acestui proiect a necesitate utilizarea multor cunoștințe practice și teoretice dobânite în facultate, cum ar fi: programarea orientată pe obiecte, folosirea limbajului Java, baze de date, programare web. În plus au fost necesare acumularea unor cunoștințe noi și învățarea utilizării tehnologiilor noi, spre exemplul framework-ul Spring, Angular2 și Thymeleaf.

Așadar subiectul acestei lucrări prezintă o idee nouă, ce consider că poate avea un impact bun dacă ar fi implementată la nivel național. Avantajele putând fi ușor observate.

* 1. **Viitoare îmbunătățiri**

În viitor, aplicația poate fi dezvoltată, pentru a permite diverse facilităti noi. În primul rând, posibilitatea vizualizării documentelor trimise de către funcționar, selectarea acestora cu scopul de a le respinge sau accepta ca documente valide. Ținerea evidenței respectării unei programări. Posibila autentificarre folosind facebook, twitter. O altă facilitate ar fi permiterea unui utilizator să își anuleze programarea; trimiterea unui email înainte cu 24 de ore, cu scopul amintirii utilizatorului despre programarea pe care o are. Un utlizator să aibă un număr maxim de MB, până la un GB pentru incararea documentelor personale. O altă facilitate în aplicația destinată funcționarului ar fi posibilitatea suspendării contului unui utilizator pe motivul neprezentării la mai mult de n programări, momentan fiind posibilă doar activarea/suspendarea unui cont.

1. **REFERINTE**

[1] - <https://ec.europa.eu/commission/index_ro>

[2] - <https://www.lifewire.com/history-of-world-wide-web-816583>

[3] - <http://www.sepoate.com/calculatoare/internet/index_9.php>

[4] - <http://www.afahc.ro/ro/facultate/cursuri/retele_note_curs.pdf\>

[5] - <https://www.tutorialspoint.com/java/index.htm>

[6] – Spring in Action 4th Edition, Craig Walls

[7] - <http://aipi2015.andreirosucojocaru.ro/laboratoare/laborator02>

[8] - <http://www.byte.ro/byte97-07/jdb.htm>

[9] - <http://getbootstrap.com/>

[10]- <https://www.slideshare.net/EmertxeSlides/spring-mvc-44350930>

[11]- <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html>

[12]- <https://spring.io/understanding/view-templates>

[13] - <http://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html#creating-a-form>

[14] - <http://freemarker.org/>