



Universitatea POLITEHNICA București
Facultatea Automatică și Calculatoare
Departamentul de Automatică și Ingineria Sistemelor

LUCRARE DE LICENȚĂ

Sistem de control acces auto bazat pe recunoașterea automată a numerelor de înmatriculare

Coordonator
conf.dr.ing. Andreea Udrea

Absolvent
Badea Vlăduț-Costin

2016

Cuprins

1	Introducere	3
1.1	Motivarea domeniului ales	3
1.2	Comparație cu aplicație similară	5
2	Tehnologii folosite	7
2.1	Java	7
2.2	IntelliJ Idea (Community Edition)	12
2.3	XAMPP for Windows:.....	13
2.4	AngularJS	15
2.5	Bootstrap	15
3	Descrierea aplicației și detalii de implementare.....	16
3.1	Baza de date.....	18
3.2	Interfața web	19
3.2.1	Modulul de Autentificare	19
3.2.2	Pagina de start	20
3.2.3	Funcționalitățile principale	20
3.2.4	Bara de navigare	22
3.3	Detalii de implementare	24
3.3.1	Configurări inițiale.....	24
3.3.2	Structura pachetelor	24
3.3.3	Resurse	28
3.3.4	Modulul de recunoaștere a numerelor de înmatriculare	29
3.4	Pachete și Biblioteci pentru recunoașterea numerelor.....	31
3.4.1	Principiile detecției numerelor de înmatriculare	31
3.4.2	Proiecțiile verticale și orizontale ale imaginii	31
3.4.3	Identificarea și selecția zonelor candidate pentru plăcuța de înmatriculare	32
4	Rezultate experimentale.....	35
4.1	Cazul 1	35
4.2	Cazul 2	36
4.3	Cazul 3	37
4.4	Cazul 4	38
5	Concluzii și dezvoltări viitoare	39
	Bibliografie.....	40

1 Introducere

Tema aleasă pentru această lucrare este o aplicație web pentru gestiunea accesului într-o incintă pe baza recunoașterii numărului de înmatriculare și propune o soluție pentru accesul într-o incintă cu acces restricționat doar automobilelor cu numerele de înmatriculare existente într-o bază de date.

Am ales acesta temă deoarece consider că în această zonă se pot face progrese mari cu investiții financiare minime și mulțumită avansărilor din domeniul Internet of Things(IoT), senzori mai ieftini, camere capabile de conectare cu un PC folosind mai multe protocoale, etc.Sunt de părere că este un mic efort deoarece sistemul se poate folosi de actualele sisteme de acces, cele bazate pe cartelă, investiția fiind foarte mică, având în vedere că majoritatea intrărilor cu barieră sunt dotate deja cu camere de supraveghere.

1.1 Motivarea domeniului ales

Am ales să dezvolt o astfel de soluție software pentru gestionarea accesului într-o incintă restricționată pe baza recunoașterii numerelor de înmatriculare deoarece prin implementarea acestui sistem se poate face un pas înainte pentru o lume mai automatizată, mai conectată, o lume în care sistemele ne ajută să ducem o viață mai ușoară, într-o lume mai sigură, mai ușor de controlat și o lume în care tehnologia ajută la reducerea stresului tot mai prezent în viața oamenilor.

Omul a dorit întotdeauna să ducă o viață confortabilă și să-și ușureze munca, așa că a inventat unelte și tehnologii care i-au permis să ajungă în vârful lanțului de conducere, tot ceea ce are de făcut acum este să coordoneze sistemele din jurul său.

Un rol important în viața modernă îl are și Internetul. Această entitate, o rețea globală care a reușit să-i apropie pe oameni cel mai mult până acum, oferindu-le posibilitatea de a socializa cu oameni din cealaltă parte de lume. Internetul s-a dezvoltat în acest ritm alert pentru că le-a dat oamenilor acces la cunoaștere și la socializare. Acesta a deschis calea spre o lume virtuală, nemărginită, o lume creată de oameni pentru oameni. Internetul își continuă expansiunea și în zilele noastre, dovedind în continuare setea oamenilor pentru cunoaștere.

În acest moment foarte puține locuri și lucruri nu au fost atinse de internet, dovada fiind și amploarea pe care proiectul Internet of Things (IoT) o are la nivel mondial și rapiditatea cu care s-a dezvoltat. Odată ce tehnologia a permis, oamenii și-au dorit să-și automatizeze casele și obiectele cele mai utilizate din gospodăria lor.

Sistemul prezent dorește să redea oamenilor o parte din timpul pierdut, să sporească confortul acestora și să-i degreveze de acțiuni repetitive zilnice.

În momentul de față, foarte multe sisteme de acces în incinte private au la bază accesul pe cartelă magnetică. Acest sistem este o îmbunătățire față de vechiul sistem cu portar deoarece presupune instalarea unei bariere acționată electric și a unui sistem

de citire a cartelelor magnetice în apropierea barierei. Șoferul care dorește să pătrundă în această incintă va trebui să urmeze următorii pași: Va opri autovehiculul în dreptul sistemului de citire a cartei magnetice, va deschide geamul și va așeza cartea pe zona respectivă. În cazul în care șoferul are acces, bariera va fi acționată și șoferul va avea cale liberă de trecere.

Și acest sistem are câteva probleme pe care sistemul propus le rezolvă. În primul rând, cartea de acces trebuie să fie în posesia șoferului. Acest lucru reprezintă un dezavantaj major deoarece pierderea, deteriorarea sau lipsa acesteia duc la pierderea accesului. Un al doilea lucru identificat de mine este sistemul de scanare al cartei, de obicei fiind amplasat cât mai aproape de barieră pentru a micșora timpul dintre validarea accesului și accesul propriu zis. Aceste sisteme necesită un spațiu suficient de mare pentru ca o mașină să poată opri în dreptul lor, să valideze și după să-și continue deplasarea, iar în cazul clădirilor care nu dispun de suficient spațiu, va obliga șoferul să rămână în stradă, câteodată blocând traficul, pentru a valida cartea de acces. Tot această amplasare presupune și deschiderea geamului șoferului pentru a valida cartea. Simpla deschidere a geamului autovehiculului creează un disconfort șoferului.

Un alt dezavantaj îl constituie și pierderea, deteriorarea sau furtul cartei. Aceasta oferă acces celui care o are în posesie, iar în cazul furtului sau pierderii, nu i se poate restricționa accesul. În acest caz, ori este generat un nou set de cartea de acces, ori vulnerabilitatea nu poate fi înlăturată.

Sistemul dezvoltat propune următoare soluție: Accesul să se facă pe baza numărului de înmatriculare. Acest lucru simplifică foarte mult accesul atât din punct de vedere al utilizatorului sistemului cât și al administratorilor acestuia. Acest sistem propune renunțarea la sistemul de scanare al cartei și înlocuirea lui cu o cameră. Cum majoritatea sistemelor existente au deja o cameră, investiția în schimbarea sistemului actual scade, deoarece două elemente din cele necesare funcționării: o barieră și o cameră există deja. Al treilea element este un sistem de calcul care să prelucraze informațiile de la cameră și care să stabilească dacă numărul de înmatriculare are sau nu acces în incintă.

Deoarece accesul se face pe baza numărului de înmatriculare, element deja existent și fără de care nu se poate circula, modul de acces este simplificat. În cazul pierderii numărului de înmatriculare, acest lucru este reglementat prin lege și în cel mai rău caz, mașina dispune de două astfel de numere deci există și o soluție de rezervă.

Timpul de acces este redus semnificativ deoarece este eliminată purtarea în permanență a unui dispozitiv în plus iar acest lucru oferă un confort sporit șoferului.

1.2 Comparație cu aplicație similară

Am căutat soluții similare pe piață și am găsit doar o singură companie din România să ofere astfel de soluții: Trittech Group.

Caracteristici	Produsul Meu	Trittech Group
Detecție pentru camera video	✓	✓
Interfața WEB administrare și raportare	✓	✓
API pentru conectare externa	✓	✓
Recunoastere numarul masinii	✓	✓
Recunoastere tara de origine	X	✓
Stocare data, ora, locație, numarul camerei, procentul de recunoastere	✓ (procentul este 100%, altfel nu se permite accesul)	✓
Posibilitate comanda deschidere bariera	(nu am avut acces la un astfel de echipament)	✓
Salvarea imaginii în baza de date de pe care s-a facut identificarea	✓	X
Posibilitatea programării accesului	✓	X

În tabel sunt prezente toate caracteristicile care mi-au fost transmise referitor la produsul lor software, nu am avut acces la interfața de administrare să fac comparație exactă a tuturor caracteristicilor, dar prețul cerut pentru soluția lor software a fost de 1000 euro. Oferta completă mai includea camera, bariera și serverul fiecare element având prețul său.

Așa cum reiese și din tabelul precedent, această soluție software oferă următoarele funcționalități:

1. Oferă o interfeță web pentru utilizator cât mai intuitivă și ușor de folosit
2. Oferă un API (Application Programming Interface) pentru dezvoltare ulterioară și conectare externă
3. Oferă un modul pentru recunoașterea numerelor de înmatriculare
4. Oferă unelte de monitorizare a traficului dar și de generare de rapoarte
5. Oferă posibilitatea perimiterii accesului într-un interval prestabilit

Am dezvoltat aceste funcționalități plecând de la un scenariu ce presupune o parcare de dimensiuni reduse (o parcare de bloc sau parcare din curtea unei firme) în care se dorește gestionarea cât mai ușor cu puțință a accesului auto. Pentru a avea o evidență cât mai precisă, am gândit sistemul pentru zone diferite de intrare și ieșire. Deoarece dreptul de acces îl au doar mașinile locatarilor sau mașinile angajaților

firmei, numerele acestora de înmatriculare sunt cunoscute și astfel sunt introduse în baza de date.

Modul de derulare al acțiunilor este după cum urmează: o mașină sosește la zona de acces, zonă în care se află montate o cameră și o barieră. La apariția mașinii, un senzor de prezență declanșează capturarea unei imagini și trimiterea acesteia spre procesare. Ajunsă pe server, imaginea este procesată și este extras numărul de înmatriculare, iar imaginea pe baza căreia s-a făcut identificarea va fi salvată în baza de date. Rezultatul obținut este comparat cu datele din baza de date și în cazul în care acest număr se află în baza de date și are drept de acces, se permite accesul în incintă, iar sistemul va înregistra data și ora accesului autoturismului. Pentru ieșirea din această incintă, este folosit același sistem de detectarea a prezenței automobilului și aceeași procedură este derulată, de această dată datele vor fi înregistrate ca părăsire a incintei a automobilului.

În cazul programării accesului pentru un număr, se va ține cont de perioadele definite de acces, fiecare număr de înmatriculare din baza de date are asociat și două date: data de la care numărul are acces și data până când numărul are acces. În afara acestui interval, accesul nu îi este permis chiar dacă numărul există în baza de date.

În capitolele următoare voi aborda tehnologiile folosite pentru dezvoltarea acestei aplicații, descrierea mai pe larg a aplicației și a funcționalității acesteia, problemele apărute la implementare și îmbunătățirile ce se pot aduce acestei soluții.

2 Tehnologii folosite

2.1 Java

Am ales Java pentru că este un limbaj de programare orientat obiect, bazat pe clase, capabil de multi-threading, special dezvoltat pentru a fi scris odată, rulat oriunde "Write once, run anywhere" (WORA). Acest principiu presupune scrierea codului o singură dată și rularea lui pe orice platformă care suportă Java, fără recompilare. Acest lucru este posibil deoarece aplicațiile Java sunt transformate în bytecode, care poate fi rulat de orice mașină virtuală Java (JVM - Java Virtual machine) indiferent de arhitectură.

Principiile dezvoltării acestui limbaj au fost:

- Simplu, orientat obiect și familiar
- Robust și sigur
- Portabil și independent de arhitectură
- Executat cu performanță ridicată
- Interpretat, capabil de paralelizare și dinamic

Cu aceste principii în minte, a fost dezvoltat Java, limbaj de programare care a preluat foarte mult din sintaxa C și C++, dar mai limitat la operațiile low level decât cele două limbaje, a fost gândit pur orientat obiect, bazat pe clase și capabil să fie rulat pe orice mașină ce poate să ruleze Java fără a recompila codul, informații mai detaliate putând fi aflate din "Java in a Nutshell" de Benjamin J Evans și David Flanagan [8].

Adevărata putere a limbajului de programare Java sunt framework-urile. Un framework este un software care oferă funcționalități generice care pot fi modificate și extinse de utilizator. Framework-ul este o platformă de dezvoltare cu ajutorul căruia un programator poate contrui unelte software mai complexe folosindu-se de o baza deja existentă, performantă, testată, documentată. Unele framework-uri oferă și compilatoare proprii optimizate pentru anumite operații.

Pentru realizarea acestui sistem am ales să folosesc framework-ul Spring. Acesta este unul dintre cele mai folosite framework-uri pentru Java, fiind performant, stabil, configurabil și extrem de bogat în unelte.

Spring Framework

Spring Framework este o platforma java care oferă o infrastructură suport astfel încât developer-ul să se poate concentra mai mult pe dezvoltarea aplicației și a caracteristicilor acesteia decât pe amănunte. Acest framework permite construirea aplicației plecând de la POJOs(Plain Old Java Objects) și oferă suport deplin pentru aplicații Java EE (Java Enterprise Edition);

Exemple de beneficii oferite de Spring Framework:

- Construirea locală a unei metode care face modificări în baza de date fără a fi nevoie de a apela explicit metode ale API-ului (Application Programming Interface) de tranzații cu baza de date

- Apelarea locală a unei proceduri remote fără a interacționa direct cu API-ul de remote
- Construirea locală a unei metode care să se ocupe de operații fără a apela explicit API-ul JMX (Java Management Extensions)
- Construirea locală a unei metode care să trateze mesajele fără a apela explicit API-ul JMS (Java Message Service)

După cum reiese din documentația platformei de dezvoltare [2], acest framework ușurează foarte mult munca dezvoltatorului prin oferirea unor metode simplificate de control al infrastructurii. Cel mai mare avantaj al Spring-ului este că oferă Dependency Injection și Inversion of Control.

Într-o aplicație complexă care presupune interconectarea mai multor elemente (baza de date, server, alte aplicații, etc) toate aceste elemente ajung să aibă dependențe unele față de altele și astfel nevoie de un management cât mai eficient din acest punct de vedere.

Platforma Java oferă acces la toate aceste elemente dar în mod tradițional toate ar trebui configurate de arhitecți de sistem și de developeri, partea de arhitectură necesitând experiență și cunoștințe avansate de arhitectură de sistem: sistem de operare pentru server, conectarea cu baza de date, profilele de lucru ale aplicației (developer, beta, production), etc. Spring Framework vine în ajutorul dezvoltatorilor de aplicații și oferă out-of-the-box aceste configurări și un mod simplificat de configurare.

Arhitectura Spring este gândită să se folosească de “design patterns” precum: Factory, Abstract Factory, Builder, Decorator, Service Locator pentru a construi clase și instanțe de obiecte care alcătuiesc aplicații. Aceste “design patterns” sunt “best practices”, adică modalități de programare eficiente, dezvoltate de persoane cu experiență, practici care eficientizează scrierea codului, elimină erori de programare, optimizează funcționarea aplicației. Este recomandată folosirea unei astfel de tehnici pentru un dezvoltator tânăr, fără experiență sau pentru dezvoltatori care nu au mai folosit tehnologia asta pentru că astfel va învăța să dezvolte cod “sănătos”. Toate practicile de mai sus se folosesc foarte mult de principiile POO și asigură atât implementare facilă cât și “readability” codului aplicației. Acest “readability” este necesar în zilele noastre deoarece toate proiectele sunt dezvoltate în echipe relativ mici dar care nu sunt în aceeași locație geografică. Acest mod de lucru a fost foarte rapid adaptat atât de marile companii pentru proiecte globale cât și de firme mici sau de dezvoltatori singuri, care voiau să se implice în proiecte.

Mai mult de atât, foarte multe proiecte sunt formate din module și este foarte important ca toate modulele să respecte aceleași reguli, același mod de scriere, astfel încât atunci când o persoană nouă intră pe proiect să se poată adapta mult mai ușor. Toate proiectele open-source se bazează pe astfel de practici.

Spring Framework este o aplicație structurată pe module, oferind astfel eficiență atât din punct de vedere al resurselor utilizate cât și a modului de utilizare, mai mult de atât, acesta oferă și un modul de test cu ajutorul căruia pot fi implementate teste automate.

Câteva module foarte utilizate la scară mare, care fac parte din aplicația prezentă sunt: Core, JDBC, JMX, Transactions, Context, conform documentației [3];

Spuneam mai devreme că marele avantaj al acestui framework este IoC (Inversion of Control) sau mai este cunoscut ca DI (Dependency Injection). Acesta este un proces în care obiectele definesc dependențele lor, adică obiectele cu care lucrează, doar prin constructorul cu parametrii, argumente transmise către o metodă "factory" sau se setează unele proprietăți ale obiectului după ce acesta a fost instanțiat de către o altă metodă. Container-ul "injectează" apoi dependențele când creează bean-ul. Un bean este o clasă care trebuie să îndeplinească următoarele convenții: să aibă constructor fără parametrii, să fie serializabilă și să ofere metode de "set" și "get" pentru attributele sale (metode cunoscute ca "getters & setters"). Acesta încapsulează alte obiecte într-un singur obiect și astfel poate fi accesat din mai multe locuri. Acest proces este exact inversul deoarece bean-ul controlează instanțierea obiectelor apelând constructorii fără parametrii folosind un mecanism denumit "Service Locator".

Interfața BeanFactory pune la dispoziția dezvoltatorului unelte de configurare avansată și un mecanism capabil să utilizeze orice tip de obiect. O sub-interfață a acestei interfețe poartă denumirea de ApplicationContext și asigură o integrare ușoară cu modulul de AOP al Spring-ului (AOP - Aspect Oriented Programming) astfel dezvoltatorul are acces la managerul de mesaje, publicarea evenimentelor, etc.

Pe scurt, BeanFactory pune la dispoziția dezvoltatorului metode de configurare ale framework-ului și funcționalități de bază iar ApplicationContext adaugă funcționalități specifice mediului Enterprise. În Spring, obiectele care formează coloana vertebrală a aplicației și care sunt utilizate de containerul IoC se numesc beans.

ApplicationContext reprezintă container-ul Spring-ului și este responsabil de instanțierea, configurarea și asamblarea bean-urilor. Această clasă primește instrucțiunile de configurare fie dintr-un XML (spring-config.xml), fie prin intermediul anotațiilor (reprezentate în cod cu @Anotare) fie prin metode clasice Java.

Spring JDBC (Java Database Connectivity)

Aproape toate aplicații din ziua de astăzi au nevoie să stocheze informații într-o bază de date sau să extragă informații dintr-o bază de date. Acest lucru presupune interconectarea celor două elemente: aplicația și baza de date. Cum baza de date poate doar stoca și extrage informații, elementul care are nevoie de informațiile puse la dispoziție de baza de date este aplicația. Lucrul cu baza de date din interiorul unei aplicații este foarte complex, în mod normal pentru fiecare interogare la baza de date este nevoie de următorii pași: conectarea la baza de date, autentificarea cu un user și o parolă, selectarea schemei bazei de date, introducerea unei interogări (sql sau non-sql) și citirea rezultatului.

Trebuie ținut seama și de cazurile nefavorabile în care se poate afla baza de date: nu răspunde din diferite motive (este oprită, nu este la adresa respectivă, nu este de tipul așteptat), datele de autentificare nu sunt bune, interogarea nu este validă, rezultatul nu poate fi întors sau nu este întors suficient de rapid. Analizând toți acești pași și numărul de cazuri de care trebuie ținut cont, conectarea unei aplicații de orice tip la o bază de date devine foarte dificilă.

Conform documentației Spring JDBC [5], Spring vine în ajutorul dezvoltatorului și oferă o unealtă de conectare la baze de date de diverse tipuri cu mult mai puține configurări de făcut. Astfel, dezvoltatorul trebuie să definească parametrii de conectare (username, password, adresa bazei de date, tipul bazei de date) și să creeze

interogările, de restul ocupându-se Spring-ul. La fiecare apel către baza de date folosind o interogare, modulul Spring-ului se va ocupa să deschidă o conexiune, să pregătească execuția interogării, să proceseze excepțiile și erorile venite de la baza de date să managerieze tranzacțiile, să întoarcă rezultatul venit de la baza de date și apoi să închidă conexiunea către baza de date. Toate operațiunile low-level sunt gestionate automat de modulul Spring-ului, dezvoltatorul se poate preocupa de datele întoarse de baza de date mai mult decât de operațiile de nivel jos.

Spring Web

Deoarece această aplicație este una web, am avut nevoie și de modulul web din Spring Framework. Uneltele puse la dispoziție prin intermediul acestui modul oferă funcționalități pentru zona de web, cum ar fi: funcționalitate de upload al unui fisier în mai multe părți, unelte pentru configurarea protocolului HTTP, dar și implementări de tipul REST sau MVC.

REST (REpresentational State Transfer) este un tip de arhitectură care se bazează pe HTTP pentru a realiza comunicația între utilizator (de pe o pagina web) și aplicație (de pe server). Acesta oferă suport pentru toate operațiile CRUD (Create, Read, Update, Delete) dar într-un mod mai simplu decât alternativele sale: RPC(Remote Procedure Calls) sau servicii web precum SOAP(Simple Object Access Protocol). REST nu este un standard așa că nu există reguli care să-l limiteze, de aceea este și foarte popular în rândul aplicațiilor scrise în limbaje precum Java, C# sau Pearl.

MVC(Model-View-Controller) este o metodologie sau "design pattern" de relaționare într-un mod eficient a interfeței cu modelul de date care stă la baza aplicației. Această metodologie este foarte des întâlnită în dezvoltarea programelor scrise în limbajele Java, C, C++. Este considerată de foarte mulți dezvoltatori ca o modalitate eficientă de reutilizarea a codului și care le permite să reducă semnificativ timpul dezvoltării unei aplicații cu interfața pentru utilizator.

Model - reprezintă structura logică de date în aplicația software și clasa de nivel înalt asociată acestuia. Acest obiect nu conține informații despre interfața utilizatorului.

View - este o colecție de clase care reprezintă elemente în interfața utilizatorului cum ar fi: butoane, checkbox-uri, campuri de date, etc.

Controller - reprezintă clasa care conectează modelul și view-ul și este folosit să se facă comunicarea între cele 2 elemente pe care le leaga.

Maven

Maven este un software pentru management de proiect bazat pe concepul Project Object Model (POM). Acesta gestionează build-ul și documentația pe baza unui singur fisier.

Maven are ca proprietăți principale următoarele: ușurează procesul de build al unei aplicații, pune la dispoziția dezvoltatorului un sistem uniform de build pentru toate sistemele, informații de calitate despre proiect, oferă îndrumare pentru bune practici de dezvoltare și oferă posibilitatea adăugării de noi capabilități, conform documentației acestuia [6].

Cu ajutorul uneltelor puse la dispoziție de Maven, acesta poate genera loguri cu modificări, referințe încrucișate pentru surse, lista de dependențe și poate executa unit tests (teste automate).

Pe baza acestui fișier pom.xml în care sunt trecute informații precum numele aplicației și grupul de care aparține dar și configurări precum modul de “ambalare” al aplicației (în cazul Java poate fi jar, war sau arhivă de tipul tar.gz), dependențele proiectului. Acestea se specifică folosind tag-urile xml `<dependency></dependency>` și aceste dependențe se găsesc pe site-ul <https://mvnrepository.com>.

În momentul în care se inițializează build-ul, folosind comanda: `mvn install`, Maven scanează fișierul pom.xml, extrage informațiile și dependențele și verifică dependențele externe ale aplicației (Exemplu: Spring) și în cazul în care fișierele respective nu se găsesc în folderul de build sau sunt versiuni mai noi disponibile, Maven se va conecta la serverul sau și va downloada toate dependențele asigurând astfel toate elementele necesare pentru aplicație. După ce se realizează build-ul aplicației, în cazul în care există, se vor inițializa testele automate (unit tests) și fiecare clasă de test va fi rulată și se va genera un raport la finalul fiecărei clase de tipul: “10 tests found, 9 successfull, 1 error” și stack trace-ul (calea către fișierul în care a fost detectată eroarea, tipul erorii și cascada de erori rezultate în urma acelei erori).

Am folosit acest tool pentru mă asigura că toate dependențele sunt îndeplinite, toate modulele sunt disponibile și pentru realizarea build-ului aplicației.

```
C:\Users\VladutCostin\Desktop\PlatesRecognition>mvn spring-boot:run
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building plates-recognition 0.1.0
[INFO] -----
[INFO]
[INFO] >>> spring-boot-maven-plugin:1.3.5.RELEASE:run (default-cli) > test-compile @ plates-recognition >>>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ plates-recognition ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 25 resources
[INFO] skip non existing resourceDirectory C:\Users\VladutCostin\Desktop\PlatesRecognition\target\generated-resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ plates-recognition ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ plates-recognition ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\VladutCostin\Desktop\PlatesRecognition\src\test\resources
```

Figura 2.1 Exemplu de pornire a aplicației

În această imagine se observă că la rularea comenzii `mvn spring-boot:run` se pornește build-ul aplicației, apoi pe baza fiecărui modul găsit în pom.xml și a modificărilor care au apărut în program se face sau nu compilarea.

Exemplu de o dependență din pom.xml:

```
<groupId>ro.platesRecognition</groupId>
<artifactId>plates-recognition</artifactId>
<version>0.1.0</version>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.5.RELEASE</version>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
</dependencies>
```

După cum se poate observa din secvența de cod de mai sus, pom-ul este organizat pe tag-uri și numele fiecăruia este suficient de descriptiv. Primele trei tag-uri sunt necesare pentru definirea numelui aplicației, grupul din care face parte precum și versiunea. Tag-ul parinte este necesar pentru ca aplicația este una care se bazează pe framework-ul Spring și este nevoie de includerea acestuia ca parinte. Urmează un tag <dependencies> sub care sunt definite dependențele aplicației, dependențe care dacă nu se găsesc local, sunt descărcate din repository-ul Maven-ului. În secvența de cod de mai sus sunt 2 dependințe necesare: spring-web și spring-data-jpa (modulul care se ocupă cu generarea interogărilor către baza de date).

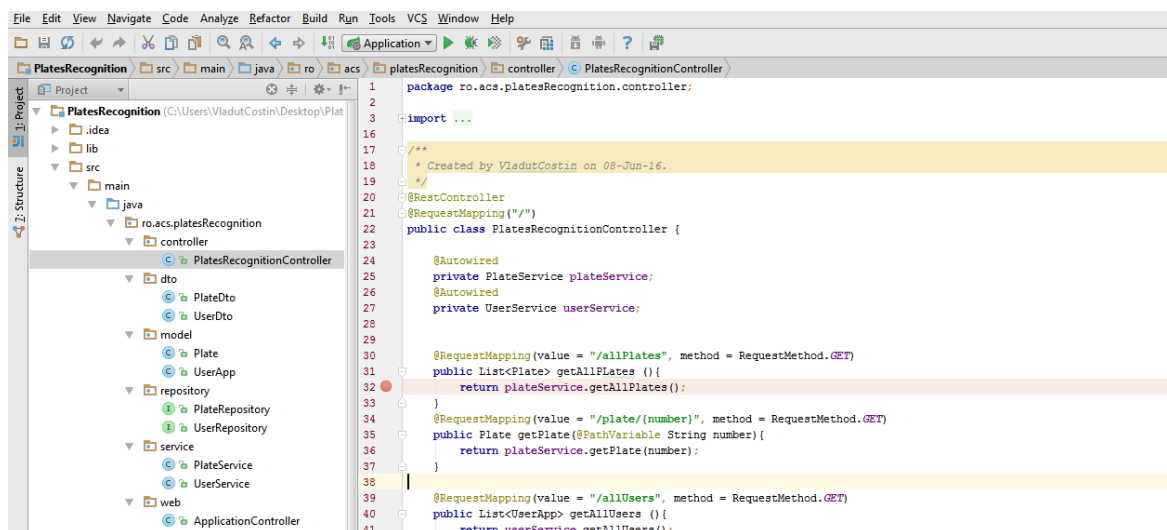
2.2 IntelliJ Idea (Community Edition)

Am ales IDE-ul (Integrated Development Environment) celor de la JetBrains deoarece este cel mai bun software de acest gen de pe piață, valoarea acestuia fiind recunoscută și de către Google de care a și fost adoptat ca și mediu de dezvoltare pentru Android. Printre avantajele pe care acesta le are, cele de care am fost cel mai mult atras au fost următoarele: Indexarea fișierelor din proiect și astfel oferă în timp real sugestii atât pentru cod ca și nume de clasă, tip de dată și în cazul apelului unei funcții dintr-o altă clasă, oferă și tipul și numele atributelor definite în interfața, conform documentației puse la dispoziție pe site-ul producătorului.

Un alt element foarte bun este integrarea tool-ului de Version Control, tool care oferă posibilitatea de a da “commit” modificărilor, a da “revert” sau de a vedea istoricul proiectului. *Commit* reprezintă comanda de aplicare a modificărilor developerului asupra proiectului, iar *revert* reprezintă comanda de revenire la o versiune mai veche a fișierului din cadrul proiectului. Acest mod de lucru pe baza unui program de version control este recomandat atât în proiecte de mari dimensiuni care presupune mai multe echipe din mai multe locuri geografice să lucreze pe același proiect, cât și pentru proiecte de mici dimensiuni pentru că oferă posibilitatea urmăririi tuturor modificărilor,

dar este și un loc de backup pentru proiect în toate stadiile sale. Această aplicație este și ea sincronizată cu un cont de Bitbucket, un serviciu gratuit de VCS (Version Control System).

Un alt element pe care îl apreciez la acest IDE este modul inteligent al completării codului și al sugestiilor bazate pe elementele existente deja în proiect. Sunt lucruri care nu par importante la prima vedere dar salvează foarte mult timp de debugging pentru erori banale cum ar fi nepotrivirea tipurilor la apelul unei funcții, lipsa de “(”, “,” , “”, etc. și ajută la obținerea unui cod curat de la început. Un alt utilitar foarte bun este și suportul pentru mai multe limbaje de programare și pentru versiunea plătită se oferă și suport pentru foarte multe framework-uri.



În secvența de cod de mai sus se observă în stânga structura proiectului, sus bara cu unelte iar în dreapta se observă clasa PlateRecognitionController și o parte din implementările și configurările făcute. Este clasa care se ocupă de relația dintre aplicație, baza de date și interfața web.

2.3 XAMPP for Windows:

XAMPP este o soluție open source și cross-platform (funcționează pe mai multe platforme) de server web dezvoltată de Apache Friends. Elementul de baza este serverul web Apache care reprezintă cel mai folosit mediu pentru un server web pentru aplicațiile de pretutindeni.

Al doilea tool prezent în acesta suită este MariaDB, baza de date Mysql foarte ușor de configurat iar al treilea element din pachetul software este suportul pentru limbajele de programare PHP și Pearl, conform documentației puse la dispoziție de dezvoltatorul soluției [7].

Este o distribuție foarte simplă de Apache recomandată pentru proiecte mici și medii dar datorită interconectării elementelor. A prins rapid la publicul larg datorită interfeței de configurare dar și datorită interfeței de control pentru baza de date în modul vizual, din browser: PhpMyAdmin. Fiind disponibil pentru toate platformele, XAMPP este o unealtă foarte bună atât pentru începători dar și pentru aplicații care nu necesită o performanță absolută și nici configurări prea pretențioase.

Pentru aplicația acesta am folosit baza de data pusă la dispoziție de acesta suită deoarece este ușor de configurat și ușor de gestionat, atât din punct de vedere al tabelelor cât și de al legăturilor dintre ele.

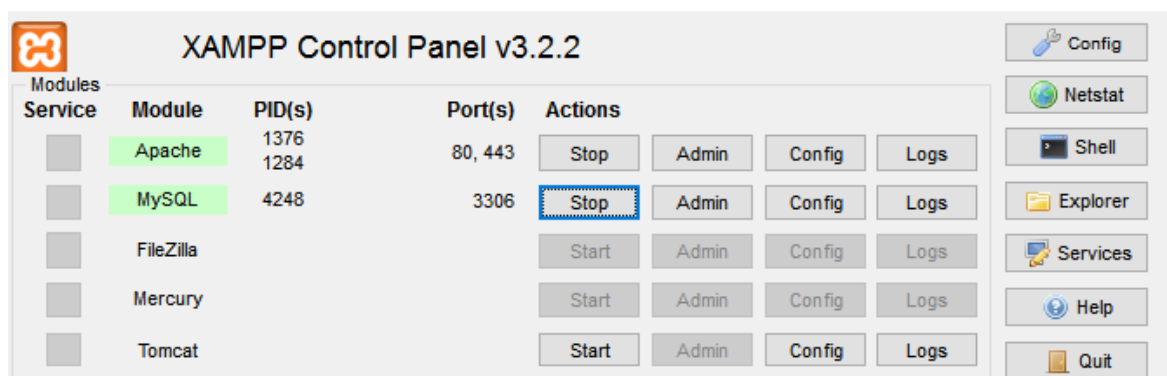


Figura. 2.3.1 – Interfața de administrare pentru XAMPP

După cum se poate observa din Figura 2.3.1 de mai sus, XAMPP oferă o interfață de control pentru toate elementele sale, un mod intuitiv și foarte potrivit pentru începători și nu numai.



Figura. 2.3.2 – Interfața de administrare a tabelelor din baza de date

În a doua imagine este interfața de administrare și configurare a bazei de date, un utilitar cunoscut sub numele de phpMyAdmin. Am folosit acest utilitar pentru crearea bazei de date și a tabelelor dar și a relațiilor dintre ele iar tot din acesta interfață se pot modifica datele din baza de date. Un utilitar la fel de util pentru cei începători dar și pentru cei mai avansați dacă nu au nevoie de configurări foarte avansate și vor să facă modificări rapide.

2.4 AngularJS

AngularJS este un framework de JavaScript, un alt limbaj de programare, de această dată client-side care facilitează interacțiunea utilizatorului cu serverul web prin intermediul paginii. Oferă acces la funcții precum cele folosite în secvența de cod de mai jos, funcții implementate cu ajutorul documentației dezvoltatorului [9].

Am ales să dezvolt aplicația web folosind acest framework pentru că îmi oferă toate uneltele necesare realizării unei aplicații web. Oferă o foarte bună integrare cu toate browser-ele moderne, oferă performanță, stabilitate și un cod curat, ușor de implementat și citit.

```
var app = angular.module('index', []);

app.controller('PlatesController', function($scope, $http) {
    $scope.plates = [];

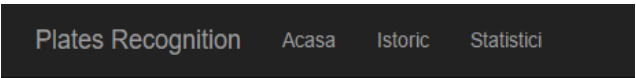
    $scope.getAllPlates = function() {
        var tablePlates = document.getElementById('tablePlates');
        if (tablePlates.getAttribute('hidden') != null) {
            tablePlates.removeAttribute("hidden");
        } else {
            tablePlates.setAttribute("hidden", 'hidden');
        }
        $http.get('http://localhost:9000/allPlates').success(function(data) {
            $scope.plates = data;
        });
    }
});
```

În secvența de cod de mai sus este un exemplu de controller, apelul către aplicația Java folosind un simplu link, cazul de succes tratat cu popularea elementului *plates* cu datele venite de la aplicația Java.

2.5 Bootstrap

Bootstrap este un framework pentru CSS cu ajutorul căruia se pot construi elemente HTML frumoase și care respectă unele standarde în ceea ce privește dimensiune, fontul, forma, spațierea, etc.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand active" href="#">Plates Recognition</a>
    </div>
    <div id="navbar" class="collapse navbar-collapse" data-toggle="collapse">
      <ul class="nav navbar-nav">
        <li class="inactive"><a href="#">Acasa</a></li>
        <li class="inactive"><a href="#">Istoric</a></li>
        <li class="inactive"><a href="#">Statistici</a></li>
      </ul>
    </div>
  </div>
</nav>
```



Folosind secvența de cod de mai sus și clasele definite de Bootstrap, am putut realiza un meniu pentru pagina aplicației, elemente găsite folosind documentația framework-ului de pe pagina dezvoltatorului [10].

3 Descrierea aplicației și detalii de implementare

Sistemul de față a fost gândit pentru a ușura accesul în incinta unei parări prin procesarea numărului de înmatriculare pe baza unei camere video conectată în zona de acces și a unui software de procesare a imaginilor care poate fi atât on-site cât și undeva într-un server de cloud. Datorită creșterii rapide a dispozitivelor care se pot conecta la Internet și datorită răspândirii foarte rapide a fenomenului de Internet of Things, această aplicație deschide calea spre un mod rapid, simplu și automatizat de a oferi acces într-un spațiu în care accesul este restricționat.

Ideea de a dezvolta acest sistem a pornit de la actualul mod de acces prezent peste tot și anume accesul pe baza unei cartele. Această metodă presupune ca soferul să aibă în permanență cartela de acces asupra lui, atunci când ajunge la intrarea în spațiu să deschidă geamul autovehiculului, să așeze cartela în zona de scanare și apoi să închidă geamul și să intre cât mai repede pentru a nu se închide bariera. Această metodă are destule dezavantaje: este necesară purtarea cartelei de acces în permanență asupra conducătorului auto, deschiderea geamului autovehiculului nu este cea mai confortabilă operațiune atunci când afară este foarte cald, foarte frig sau plouă, plasarea cartelei de acces în zona de scanare este dificilă pentru unele tipuri de autovehicule (SUV) mai ales dacă șoferul este mic de statură, în cazul accesarilor multiple, timpul pierdut pentru acces crește destul de mult.

Soluția propusă are multiple avantaje față de actuala soluție prezentă la scară foarte mare deoarece accesul se face pe baza numărului de înmatriculare, teoretic unic pe teritoriul fiecărei țări, iar diferențele dintre modelele de plăcuțe de înmatriculare pentru fiecare țară face ușoară identificarea țării de origine. Numărul este preluat cu ajutorul unei camere și transmis către un server care poate fi la o stație lângă zona de acces sau poate fi în cloud. Soluția de cloud este una mult mai bună deoarece serverele din cloud au o putere de procesare mult mai mare față de o stație, performanțele sistemului se păstrează pe toată durata de viață, în cazul unei pene de curent, cu ajutorul unei surse de curent (UPS) se poate alimenta camera în continuare pentru o perioadă mult mai mare decât dacă ar fi fost prezent și sistemul de procesare. Prin plasarea în cloud a software-ului de procesare a imaginilor dar și a interfeței de utilizare, costurile scad deoarece nu mai este nevoie de sistem de calcul de putere, nu mai este nevoie de infrastructură necesară asigurării unui timp de funcționare ridicat (UPS-uri pentru alimentarea camerei, sistemului de procesare) dar nici de construirea unui spațiu unde să fie amplasate aceste sisteme.

Interacțiunea utilizatorului cu sistemul se poate face prin accesarea adresei aplicației de pe orice dispozitiv capabil să ruleze un browser web. În acest mod, se poate elimina complet operatorul uman, deși nu este recomandat pentru a descuraja eventuale tentative de fraudă, se poate face atât depanarea cât și supravegherea sistemului de oriunde există acces la internet eficientizând astfel atât constatarea și depanarea problemei în cazul unei defecțiuni dar și operarea de la distanță de către un utilizator cu drepturi avansate.

Sistemul dispune de posibilitatea programării permisiunilor de acces. Un exemplu de un astfel de caz: în cazul în care sistemul este instalat pentru o parcare de cartier rezidențial, accesul în cadrul acestei parări este permis tuturor locatarilor

din zona, pe o perioadă nedeterminată. Problema apare atunci când proprietarii primesc vizite de la rude sau prieteni. Pe baza modului actual de acces într-o astfel de zona, cel care dorește să intre în incinta parcurii nu o poate face decât dacă are un card de acces sau cel pe care îl vizitează îi oferă cardul său.

În cazul sistemului acesta, se poate programa din cadrul interfeței un interval de acces pentru un număr de înmatriculare. După completarea formularului și finalizării procesului, sistemul va permite accesul numărului de înmatriculare doar în perioada specificată, scutind astfel efortul menționat mai sus. Această soluție este mai bună deoarece se poate introduce orice număr de înmatriculare și orice interval de acces de oriunde există conexiune la Internet, de către orice utilizator. La fel se poate modifica sau șterge orice programare viitoare.

Principiul de funcționare se bazează pe existența unui modul de recunoaștere a textului din imagine și mai exact de recunoaștere a numerelor de înmatriculare din imagini. Așa cum o camera surprinde mai multe cadre pe secunda, se selectează din acele cadre cele care au o claritate ridicată și sunt procesate. În urma procesării imaginii, se obține în format text numărul de înmatriculare din fotografie dar și o multitudine de informații referitoare la imaginea procesată. Dacă în urma identificării nu a fost obținut un număr de înmatriculare, se preia o altă imagine și se repetă procesul până la o identificare cu succes (maxim 5 încercări).

În urma unei identificări cu succes se trece la etapa următoare de validare. Numărul obținut este comparat cu cele din baza de date și în cazul în care are drept de acces se va înregistra în baza de date data și ora la care s-a făcut accesul, se va înregistra că vehiculul este prezent în incintă, în cazul în care există un operator, se va înregistra și operatorul responsabil. Aceste înregistrări sunt utile pentru generarea de rapoarte și statistici dar și pentru cazuri de furt sau alte infracțiuni, pentru a ajuta la identificarea făptașilor.

Același sistem de înregistrare în baza de date este folosit și pentru unele de acces programat, acolo făcându-se înregistrarea utilizatorului care a programat accesul, al datei și orei la care s-a făcut programarea și a numărului de înmatriculare programat.

Pentru procesarea imaginilor am folosit JavaANPR (Java Automatic Number Plates Recognition), un software de recunoaștere a numerelor de înmatriculare dezvoltat de Ondrej Martinsky de la "Brno University of Technology", sistem ce face parte din cercetarea "Security-oriented research în information technology". Am ales acest software deoarece dezvoltatorul acestuia, pe baza experimentelor sale, a declarat că sistemul a fost eficient în 85% din cazurile testate de acesta, este disponibil gratuit pe Internet sub licența de tipul "Educational Community License" și poate fi folosit pentru tipul acesta de lucrare, în cazul comercializării acest sistem nu poate fi folosit.

În foarte multe cazuri, identificarea vehiculelor după numerele de înmatriculare este foarte ușoară pentru oameni, dar nu și pentru mașinării, computere, roboți, etc. Pentru toate aceste obiecte, un număr de înmatriculare este o fotografie cu tonuri de gri sau o imagine în alb-negru, definită de o funcție de două dimensiuni x și y , unde acestea sunt coordonate spațiale și funcția este intensitatea luminoasă în acel punct. Din această cauză este necesară dezvoltarea unui sistem robust din punct de vedere matematic, acestea poartă denumirea de ANPR (Automatic Number Plates

Recognition) și este tipul de sistem care transformă datele din mediul real în cel informațional. Pentru rezolvarea acestei probleme este nevoie de mai multe ramuri tehnologice: inteligență artificială, “machine vision”, recunoașterea de modele și rețele neurale. Ondrej Martinsky s-a ocupat să îmbine aceste elemente și să realizeze un sistem capabil să recunoască numere de înmatriculare și să poată fi învățat modele noi.

3.1 Baza de date

Fiecare aplicație are nevoie de o bază de date pentru a stoca toate informațiile necesare funcționării. Așa cum a fost descris mai sus, baza de date a aplicației este cea oferită în suita de unelte ale XAMPP-ului, bază de date de tip MySQL.

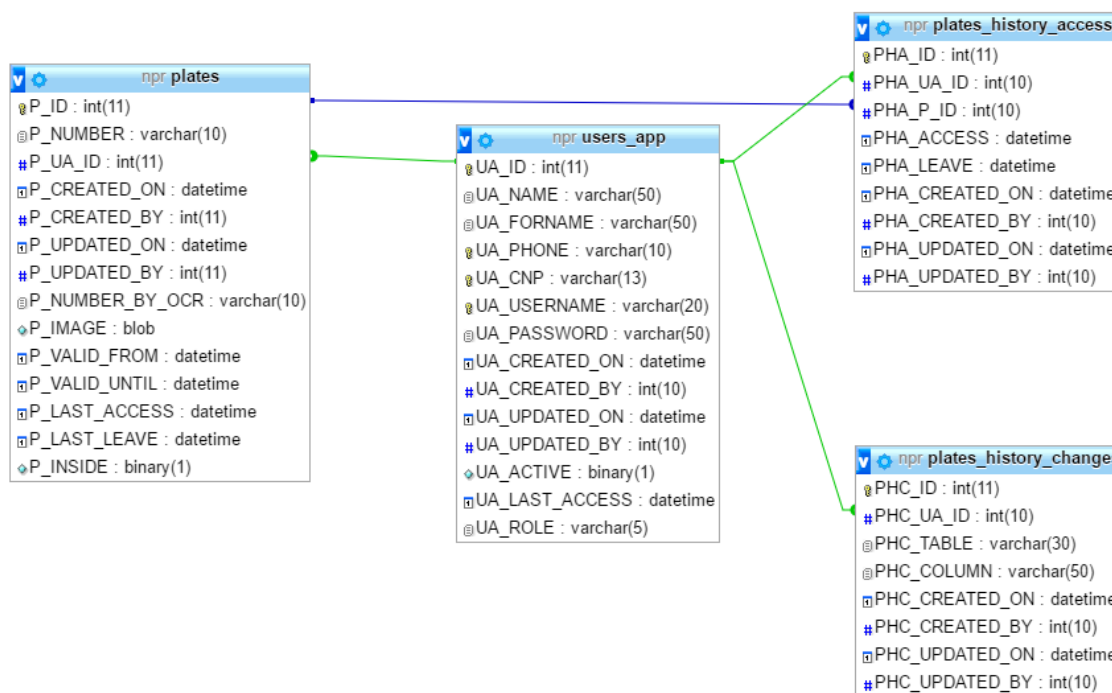


Figura 3.1.1 – Schema bazei de date cu relațiile dintre tabele

În figura 3.1.1 de mai sus este schema bazei de date cu relațiile între tabele și cu câmpurile fiecărei tabele. Există o tabelă specială pentru toate schimbările efectuate de utilizator sau de sistem.

Tabela “plates” conține toate plăcuțele de înmatriculare dar și o cheie externă către “users”. Așa cum se poate observa din imagine, un user are atribuită o plăcuță de înmatriculare și nu invers pentru a permite folosirea aplicației și fără a avea o mașină înregistrată. Am gândit astfel pentru cazurile când va fi și un operator de sistem și acesta să nu fie obligat să aibă mașina înregistrată.

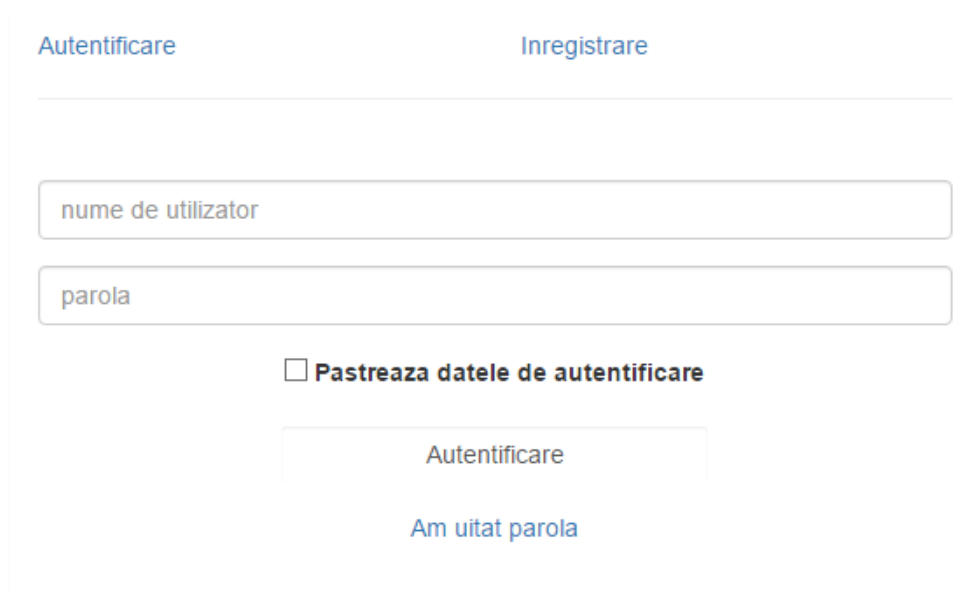
Am inclus în această structură și informații care nu sunt necesare utilizatorilor obisnuiți, dar pe baza cărora se pot efectua statistici sau se pot verifica erori, accesări nepermise, etc.

3.2 Interfața web

3.2.1 Modulul de Autentificare

Fiind o aplicație web care conține informații private ale utilizatorilor (CNP, număr de telefon, nume și prenume) este necesară protecția datelor. Următoarea imagine ilustrează pagina de autentificare cu care este întâmpinat orice utilizator la început.

Va rugam sa va autentificati!



Formular de autentificare cu două tab-uri: "Autentificare" (selecționat) și "Înregistrare".

Câmpuri de intrare:

- nume de utilizator
- parola

Opțiune:

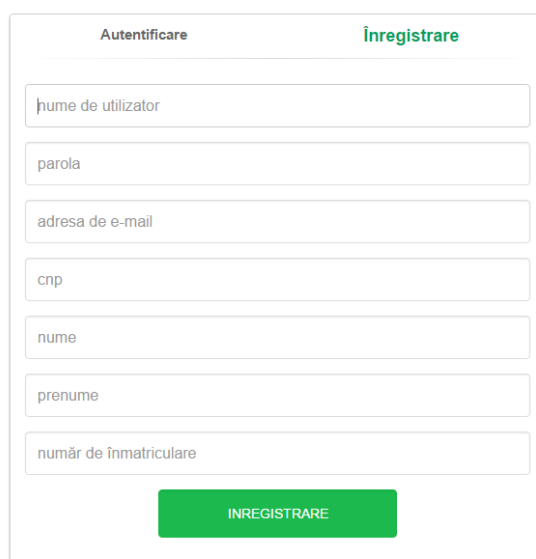
- ☐ Pastreaza datele de autentificare

Butoane:

- Autentificare
- [Am uitat parola](#)

Figura 3.2.1.1 Formularul de autentificare

Așa cum se observă în Figura 3.2.1.1 de mai sus, utilizatorul poate alege între a se autentifica sau își poate crea un cont nou.



Formular de înregistrare cu două tab-uri: "Autentificare" și "Înregistrare" (selecționat).

Câmpuri de intrare:

- nume de utilizator
- parola
- adresa de e-mail
- cnp
- nume
- prenume
- număr de înmatriculare

Buton:

- INREGISTRARE

Figura 3.2.1.2 Formular de înregistrare al unui utilizator nou

3.2.2 Pagina de start

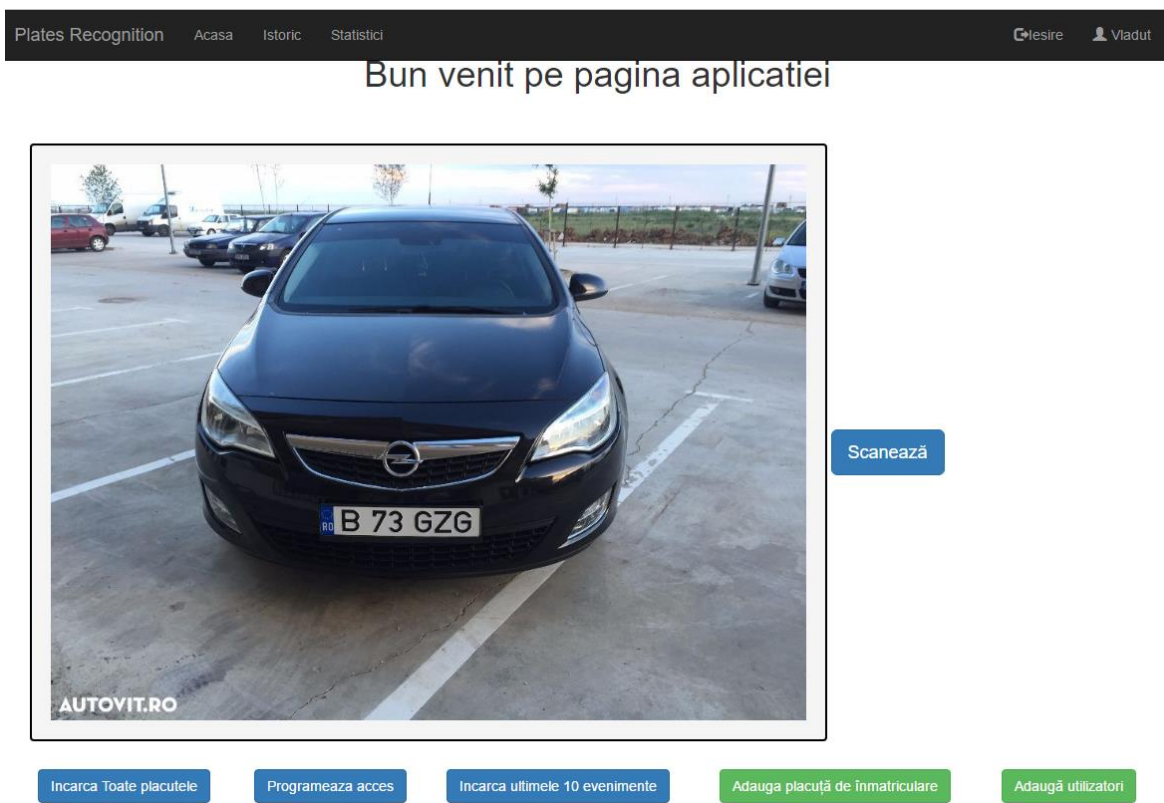


Figura 3.2.2.1 Pagina de start

În imagine este interfața aplicației După o logare cu succes cu user și parolă. Se observă zona unde este imaginea cu mașina, acolo va fi vizibil ceea ce “vede” camera video.

Interfața aplicației cuprinde un meniu principal pe o bară neagră unde se pot observa meniurile: “Acasa”, “Istoric” și “Statistici” iar “Plates Recognition” poate fi emblema firmei în cazul în care sistemul este folosit de către o firma sau orice alt text dorește utilizatorul sistemului, dar și butonul pentru “Inesire” și prenumele utilizatorului logat în acel moment.

3.2.3 Funcționalitățile principale

Prima pagină pe care este direcționat utilizatorul este formată dintr-o zonă de video unde este redat în timp real fluxul de imagini venit de la cameră sau de la mai multe camere. Butonul “Capture” de lângă această zonă are rolul de captura și trimite spre procesare a imaginii respective. Acest mod a fost gândit pentru unul dintre următoarele scenarii:

1. Se pot lua cadre din filmarea camerei periodic, perioada depinzând de puterea de procesare, deoarece procesarea fiecărei imagini consumă destul de multă putere de procesare.
2. Se poate opta pentru folosirea unui senzor de prezență care va declanșa procesarea imaginii sau a modului manual implementat.

Mai sunt prezente funcționalități precum:” Încarcă toate plăcuțele”, “Programează acces”, “Încarcă ultimele 10 evenimente”, “Adaugă plăcuță de înmatriculare” și “Adaugă utilizatori”.

Incarca Toate placutele		Programeaza acces	Incarca ultimele 10 evenimente	Adauga placuță de înmatriculare	Adaugă utilizatori
Id	Numar inmatriculare	Ultimul acces	Ultima iesire	Este prezent	Actiuni
1	CL67BVL	17-06-2016 la 9:29PM	17-06-2016 la 1:30PM	●	Editează Șterge
2	B123ABC	18-06-2016 la 1:29PM	18-06-2016 la 2:14PM	●	Editează Șterge

Figura 3.2.3.1 Lista cu numerele de înmatriculare existente în baza de date

Prima funcționalitate afișează toate plăcuțele de înmatriculare existente în baza de date cu următoarele informații: “Ultimul acces”, “Ultima iesire”, “Este prezent” și “Acțiuni”, așa cum se poate observa în Figura 3.2.3.1 de mai sus.

Deoarece sistemul poate suporta mai multe camere web, a fost gândit pentru 2 camere: 1 de acces și ce-a de-a 2-a pentru ieșirea din incintă. Acest lucru asigură o supraveghere mai eficientă, se poate ieși și intra în incintă simultan fără a bloca sistemul dar mai ales traficul. Având acest lucru în vedere, am considerat necesară afișarea celor două informații. Informația de prezență este foarte utilă atât pentru utilizatorii din parcare a unui cartier rezidențial, dar și unor administratori de firmă în cazul în care sistemul este instalat într-un astfel de loc. Pentru cele 2 situații am avut în vedere următoarele scenarii:

1. La sediul firmei urmează să sosească viitori clienți sau asociați și administratorul dorește să se informeze asupra locurilor libere de parcare și să decidă dacă întâlnirea poate avea loc la sediul firmei sau va cauta un alt loc.
2. Un utilizator dintr-un cartier rezidențial plecat în străinătate dorește să verifice dacă mașina lui mai este în parcare.

În coloana “Actiuni” sunt prezente acțiunile ce pot fi efectuate asupra numerelor de înmatriculare existente precum modificarea sau ștergerea.

Cea de-a doua funcționalitate prezentă în interfață este cea de “Programează acces” și acesta deschide un formular simplu cu informațiile minime necesare pentru programarea accesului în zona, așa cum se poate observa în Figura 3.2.3.2 de mai jos. Toate câmpurile sunt obligatoriu de completat.

Numarul

Incepend cu

Pana la

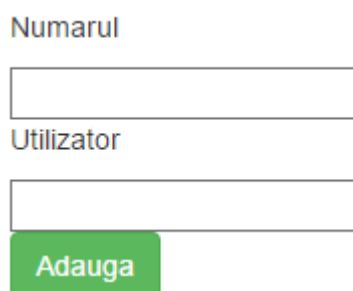
Programează

Figura 3.2.3.2 Formularul pentru programarea accesului

Numărul mașinii este foarte important deoarece el este cel pe baza căruia se permite accesul în zonă. Celelalte două informații obligatorii sunt cele 2 date: cea de început și cea de sfârșit, în acest interval automobilul cu numărul de înmatriculare specificat mai sus va avea acces de a intra și de a ieși din incintă. În cazul în care utilizatorul dorește, se pot adauga câmpuri suplimentare dar am considerat că acestea sunt minimul necesar pentru acces, mai ales că am dorit simplificarea metodei pentru ca utilizatorul care dorește să facă această programare de acces să nu piardă foarte mult timp, facilitând utilizarea acestei unelte de pe orice dispozitiv capabil să ruleze un browser și care are o conexiune la internet.

În cazul celei de-a treia funcționalități, se va afișa un tabel cu ultimele evenimente (intrări și ieșiri) care s-au petrecut. Această informație ajută atât administratorii de firmă pentru a avea o evidență cât mai clară a ceea ce se petrece în spațiul de acces în firmă, administratorii sistemului sau operatorul care poate fi de la o firmă de pază care se ocupa de protejarea spațiului dar și poliția în cazul în care s-a petrecut vreun incident, se pot accesa rapid ultimele 10 evenimente unde sunt afișate toate informațiile importante: numărul de înmatriculare, data și ora la care s-a petrecut evenimentul, în cazul în care este un operator de sistem, userul acestuia și utilizatorul căruia îi aparține numărul de înmatriculare.

Următoarea funcționalitate și una foarte importantă este cea de adăugat plăcuțe de înmatriculare iar apăsarea lui deschide un formular în care trebuiesc completate informații precum numărul de înmatriculare dorit dar și utilizatorul, așa cum se observă în Figura 3.2.3.3 de mai jos.



Numarul

Utilizator

Adauga

Figura 3.2.3.3 Formular adăugare plăcuță de înmatriculare

Este necesar ca utilizatorul să existe deja, altfel se va genera o eroare. Am luat o astfel de decizie pentru că la creerea utilizatorului sunt cerute informații confidențiale și ar fi de preferat ca utilizatorul să fie creat de fiecare utilizator în parte.

Adaugarea de utilizatori se poate face apăsând butonul “Adaugă utilizatori”. Formularul este identic cu cel din Figura 3.2.1.2, formular pentru completarea căruia va fi nevoie de informații confidențiale precum CNP, număr de telefon, adresă de e-mail.

3.2.4 Bara de navigare

În meniul “Istoric” se pot genera liste cu informații atât despre evenimentele de intrare și ieșire cât și despre evenimentele petrecute în sistem. Se pot configura parametrii: utilizator, număr de înmatriculare, data, data acces, data ieșire.

Căutarea după utilizator va returna informații referitoare la comportamentul utilizatorului respectiv în sistem de la data înregistrării în sistem până în prezent. Aceste informații vor fi oferite doar utilizatorilor cu drepturi avansate (administratori).

Selectarea de căutare după numărul de înmatriculare va avea ca rezultat istoricul intrărilor și ieșirilor din sistem pentru numărul ales, informație accesibilă doar utilizatorilor cu drepturi avansate și doar utilizatorului proprietar al acestui număr. Nu am dorit ca acest sistemul să fie o unealtă de spionaj între vecini.

În cazul selectării datei se vor întoarce toate evenimentele din acea dată, din nou, aceste informații vor fi vizibile doar pentru administratori, fiecare utilizator având acces doar la informațiile sale.

Asemenea câmpului de mai sus, selectarea datei de acces și a celei de ieșire va întoarce informații referitoare la accesul respectiv ieșirea din sistem pentru data respectivă. Informațiile pentru utilizatori de nivel superior vor include toate informațiile, iar utilizatorii simpli vor avea acces doar la evenimentele lor.

În continuare voi prezenta pe scurt modul de funcționare și metodele folosite de Ondrej Martinsky pentru identificarea plăcuțelor de înmatriculare.

În lucrarea lui Ondrej Martinsky - "Algorithmic and mathematical principles of automatic number plate recognition systems", acesta tratează principiile segmentării caracterelor, al normalizării caracterelor, al normalizării luminozității pentru a asigura invarianță față de condițiile de iluminare. Urmează aplicarea de algoritmi pentru filtrarea informațiilor utile urmând apoi proceduri de recunoaștere de caractere pe baza unor modele și a rețelelor neurale. Următorul pas este identificarea erorilor și corectarea acestora pe baza modelului numărului de înmatriculare. Pentru cazul țării noastre, este un model de tip **LLCCLLL** sau **LCCCLLL** unde L este literă și C este cifră pentru numere cu validitate extinsă, numerele provizorii au alt model: **LCCCCC**, unde L este litera și C este cifră. Pe baza acestor modele, se pot face corecturi de tipul: O recunoaștere a caracterului „B” poate fi „8” sau a caracterului „O” poate fi „0”.

3.3 Detalii de implementare

Se poate observa în Figura 3.3.1 diagrama de clase și legătura dintre pachete și clase. Am preferat să pastrez doar clasele importante în această diagramă pentru că includerea tuturor claselor ar fi generat o diagramă de neînțeles.

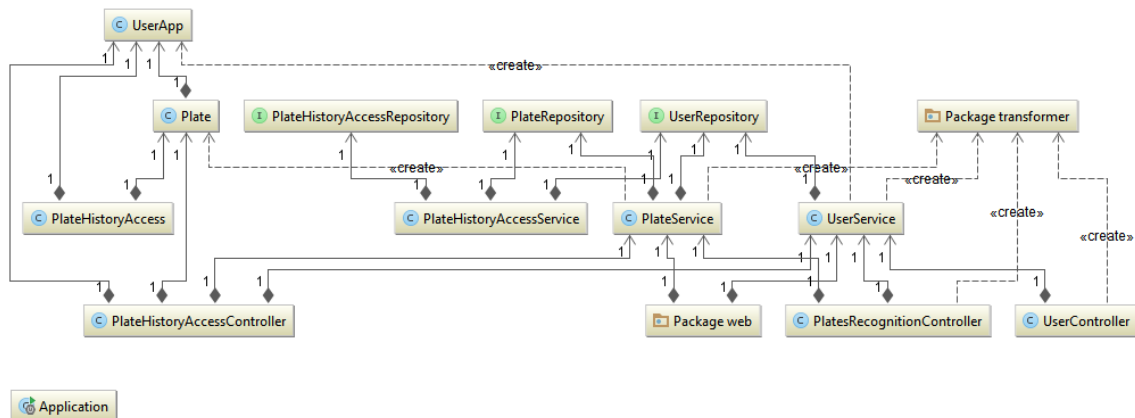


Figura 3.3.1 Diagrama de clase și legăturile dintre clase și pachete

3.3.1 Configurări inițiale

Am început dezvoltarea aplicației prin a crea un proiect nou în IDE și prin a adăuga în pom.xml numele aplicației pe care doream să o dezvolt apoi toate elementele necesare folosirii framework-ului Spring. După aceste configurări de început am avut nevoie de driver-ul pentru conectare la o bază de date MySQL. Driver-ul este bucata de software care realizează buna comunicare între cele două elemente. Acest driver este necesar pentru ca aplicația Java să știe cu ce fel de bază de date lucrează (tip, versiune, configurări speciale).

3.3.2 Structura pachetelor

Următorul pas a fost realizarea structurii de pachete pentru a putea fi detectare de framework și pentru a fi ușor de urmărit fiecare componentă a aplicației. Acest lucru se recomandă a se face la începutul proiectului deoarece odată cu începerea dezvoltării, proiectul poate căpăta foarte multe clase și fără a păstra o ordine, totul se va amesteca. Un proiect cu pachetele ordonate îi ajută atât cei care nu cunosc proiectul dar și pe dezvoltatori deoarece fiecare eroare va fi generată cu pachetul și numele clasei din care provine, făcând foarte ușoară depanarea problemei.

După realizarea pachetelor, structura aplicației arata precum în Figura 3.3.2.1 de mai jos:

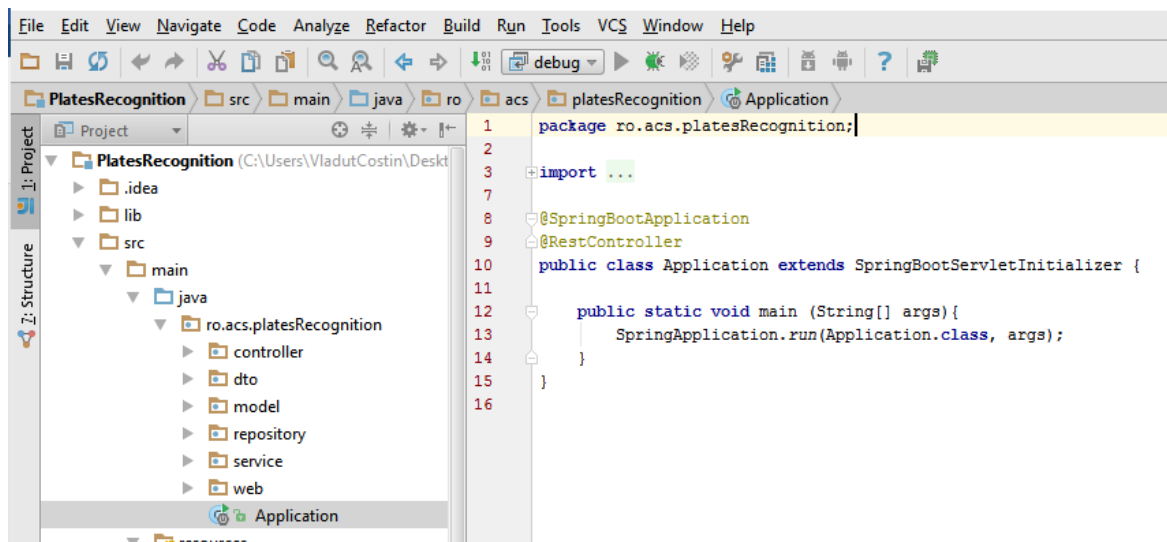


Figura 3.3.2.1 Structura pachetelor și clasa de inițializare a aplicației

Urmează explicații pentru fiecare pachet în parte și rolul său. Astfel, pachetul “controller” conține toate controller-urile aplicației adică acele clase care realizează comunicarea cu exteriorul aplicației. Fiecare clasă de controller este adnotată (conține @RestController) fapt ce-i transmite framework-ului rolul său în aplicație. Către controller vin toate request-urile (cererile din pagina web , interfața aplicației) și fiecare cerere trebuie să fie tratată aici.

Pachetul „dto” este responsabil cu tipul de clase DTO (Data Transfer Object) adică acele clase cu ajutorul cărora informația din baza de date este transferată în interfață. De obicei au aceleași attribute ca obiectele din baza de date, fără ID. Aceste obiecte au un rol foarte important deoarece un obiect din baza de date este necesar să aibă un ID, dar acest DTO nu conține câmpul ID, fiind un obiect responsabil doar cu transferul informație din pagină în felul următor: Atunci când în interfață este completat un formular de adăugare al unei plăcuțe de înmatriculare, toate informațiile introduse de utilizator ajung la aplicație care le va transforma în obiecte compatibile cu cele din baza de date. Fiind informații despre o plăcuță nouă, ea nu există încă în baza de date, deci nu i se poate atribui un ID până nu se materializează și în baza de date. Toate informațiile sosite prin intermediul DTO-ului sunt validate după criterii precum: datele să fie valabile, datele de început să nu fie mai mari decât cele de sfârșit în cazul programării unui acces, utilizatorul care a făcut o cerere către aplicație să aibă dreptul la acea cerere, se asigură integritatea bazei de date, etc. După ce toate informațiile sunt verificate și validate, se va apela procedura de inserare în baza de date.

Atunci când se dorește extragerea de informații din baza de date, obiectul din baza de date trebuie pregătit pentru afișarea în pagină dar și validat că este corespunzător cererii făcute , deci trebuie trecut iar prin această transformare.

```

@Entity
@Table(name = "PLATES")
@Data
public class Plate {
    @Id
    @GeneratedValue
    @Column(name = "P_ID")
    private Long id;
    @Column(name = "P_NUMBER")
    private String number;
    // in case of detection errors
    @Column(name = "P_NUMBER_BY_OCR")
    private String numberByOCR;
    @Column(name = "P_IMAGE")
    private Byte[] image;
    @ManyToOne
    @JoinColumn(name = "P_UA_ID")
    private UserApp userApp;
    @Column(name = "P_VALID_FROM")
    private Date validFrom;
    @Column(name = "P_VALID_UNTIL")
    private Date validUntil;
    @Column(name = "P_LAST_ACCESS")
    private Date lastAccess;
    @Column(name = "P_LAST_LEAVE")
    private Date lastLeave;
    @Column(name = "P_INSIDE")
    private Boolean inside;
}

```

Pachetul “model” este pachetul în care sunt obiectele din baza de date, adică acele clase care conțin atribute de tipul câmpurilor din baza de date. În secvența de cod de mai sus este prezentată clasa Plate și voi explica fiecare element prezent în această clasă.

@Entity este o adnotare care informează framework-ul Spring despre faptul ca această clasă este modelul unei tabeli din baza de date.

@Table și numele tabeli informează Spring cu privire la care tabelă se face referire și conține proprietățile respective, iar la pornirea aplicației Spring-ul se va conecta la baza de date și va verifica dacă în tabela “PLATES” există toate acele câmpuri și dacă tipul lor este cel specificat.

@Data este o adnotare folosită de lombok, un software Java care are ca scop reducerea numărului de linii de cod și evitarea fenomenului de “Boilingplate Code”, adică cod care se repetă în foarte multe locuri din aplicație și diferă foarte puțin. Acestă unealtă nu mai face necesară generarea de “getters” și “setters”, adică acele metode care modifică și întorc valoarea atributelor clasei. Acest lucru reduce foarte mult numărul de linii de cod și implicit și dimensiunea proiectului. La inițializarea aplicației se vor genera acele metode și astfel dezvoltatorul nu mai este nevoit să le implementeze de fiecare dată. Este o metoda de a ține codul sursă curat având aceleași funcționalități.

@Id va spune Spring-ului că acea coloană este cheie primară și astfel de aici decurg și celelalte proprietăți precum: unicitate, indexabilă, nenulă.

@GeneratedValue de această adnotare se va ține cont când se va dori introducerea de obiecte noi în baza de date și se va genera o valoare unică.

@Column este adnotarea ce specifică numele câmpului pe care îl reprezintă.

@ManyToOne și @JoinColumn sunt adnotările folosite pentru a specifica faptul că acel element este o cheie secundară și tipul de relație între cele două câmpuri.

Pentru toate adnotările de mai sus, se generează erori la pornirea aplicației dacă unul dintre elementele precizate mai sus nu se găsesc în baza de date, la fel dacă tabelul către se face referire la cheia secundară nu există sau tipul precizat este diferit de cel din baza de date.

Pachetul “repository” este pachetul în care sunt clasele ce se ocupă cu extragerea informațiilor din baza de date, obiectele care realizează interogările către baza de date și care întorc rezultatele.

Până la apariția SpringJPA, interogările se făceau destul de greoi, foarte multă muncă era depusă de dezvoltator pentru a face interogările simple precum cele de întoarcere a unui obiect după ID. Acest lucru a fost simplificat odată cu apariția SpringJPA care oferă cuvinte cheie și care simplifică foarte mult interogările, astfel: Dacă pentru a întoarce un element după ID era nevoie de scrierea următoarei interogări: “select * from numele_tabelei where id=#ID-ul” , cu ajutorul acestei unelte, generarea de interogări către baza de date este adusă sub forma unei funcții: “repository.findOne(ID)” unde „repository” este clasa ce realizează interogările și “findOne” este funcția care face interogarea pentru ID. Un alt cuvânt cheie este și “findBy” urmat de numele atributului cautat și va întoarce toate rezultatele filtrate După acel atribut. Aceste cuvinte cheie se pot asocia și astfel se pot obține interogări complexe folosind doar numele atributelor, informații despre aceste cuvinte cheie pot fi găsite în documentația Spring JPA [4].

Pachetul “service” este locul unde se găsesc toate clasele ce realizează procesările informațiilor.

```
@Service
@Transactional
public class PlateService {

    @Autowired
    private PlateRepository plateRepository;
    @Autowired
    private UserRepository userRepository;

    public List<PlateDto> getAllPlates(){
        return transformAll(plateRepository.findAll());
    }

    public PlateDto getPlate(String number){
        return transform(plateRepository.findByNumber(number));
    }
}
```

În clasa *PlateService* de mai sus este un exemplu de clasă de serviciu și se pot observa adnotările de configurare dar și modul de conectare al acestora cu restul aplicației.

`@Service` este adnotarea ce informează Spring-ul că această clasă este una de serviciu și va fi configurată astfel. O clasă de serviciu este o clasă care realizează toate operațiile logice și care returnează rezultatul procesărilor. Este clasa la care face apel Controller-ul pentru rezolvarea cererilor venite din interfață.

`@Transactional` este adnotarea care simbolizează faptul că această clasă are dreptul de a face tranzacții către baza de date și astfel Spring-ul se va asigura ca acele tranzacții se vor desfășura în ordine. Această adnotare simplifică foarte mult efortul dezvoltatorului deoarece fără acesta, fiecare tranzacție ar trebui supravegheată de dezvoltator și toți pașii necesari precum deschiderea unei conexiuni către baza de date, inițializarea unei tranzacții, verificarea că fiecare nod a terminat cu succes tranzacția și la final efectuarea modificărilor ar fi trebuit făcute de fiecare dată de dezvoltator. Această adnotare ia acest efort de pe umerii dezvoltatorului și îl lasă să se concentreze mai mult pe dezvoltarea caracteristicilor aplicației sale.

`@Autowired` face parte tot din suita de unelte ale Spring-ului și are ca rol semnalizarea faptului că este nevoie de acele clase și ele trebuie instanțiate dacă nu au fost până acum, sau se va căuta instanța dacă există deja și se va pune la dispoziția funcției de serviciu.

Așa cum am menționat și mai sus, niciodată nu se trimite către interfața web un obiect din baza de date ci un DTO, așa cum se poate observa și în exemplul de mai sus.

Ultimul pachet, "web", este folosit pentru clasele de configurare a conexiunilor dintre modulele web și cele de bază ale Spring-ului. Este vorba de configurări speciale în cazul în care rețeaua este protejată de firewall, trebuie să treacă prin proxy sau să redirecționeze traficul către un proxy. Aceste lucruri nu au fost necesare pentru aplicația curentă, toate setările implicite fiind suficiente.

Clasa rămasă în afara celorlalte este clasa responsabilă cu pornirea aplicației. Conținutul ei este vizibil în Figura 3.3.2.1 și așa cum se poate observa, cu ajutorul framework-ului Spring aplicația este pornită. Adnotările `@SpringBootApplication` și `@RestController` sunt cele ce informează Spring-ul de faptul că este o aplicație de tipul SpringBoot și astfel trebuie inițializate toate componentele necesare, iar cea de-a doua adnotare este necesară pentru ca restul controller-elor să aibă acces la cererile venite din exterior.

3.3.3 Resurse

Resursele sunt elemente necesare funcționării aplicației. Așa cum se observă în Figura 3.3.3.1 de mai jos, structura acestui pachet este și ea organizată tot sub formă arborescentă.

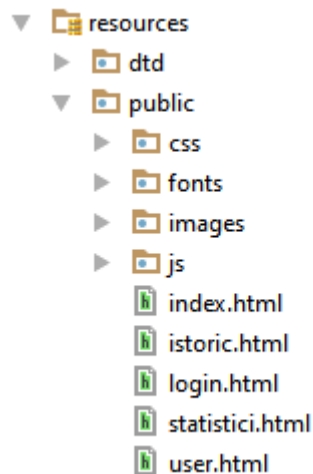


Figura 3.3.3.1 Schema pachetelor de resurse

Pachetul “public” este cel în care se găsesc toate resursele interfeței web, toate clasele CSS definite special, care nu se găsesc în framework-ul de Bootstrap se găsesc acolo, fiecare terminandu-se în extensia .css.

Pachetul “js” conține toate resursele de JavaScript cât și aplicația în AngularJS sub forma app.js

Pachetele “images” și “fonts” conțin imaginile folosite în aplicație dar și fonturi speciale dacă există, dar nu este cazul pentru aplicația de față.

Lista fișierelor ce se termină în .html sunt paginile de tip “Single Application Page” despre care am vorbit mai pe larg în capitolul dedicat AngularJS. Aici este definită forma statică a paginilor adică modul cum arată fără a fi conectate la aplicație.

În funcție de cât de dependentă de aplicația Java este fiecare pagina, este posibil ca paginile să nu poată fi vizibile fără o aplicație care să le furnizeze datele necesare.

Pachetul “dtd” este necesar pentru ca modulul de recunoaștere a plăcuțelor de înmatriculare să funcționeze. Acolo sunt definite reguli interne despre care se pot afla mai multe din documentația lui Ondrej Martinsky din lucrarea “Algorithmic and mathematical principles of automatic number plate recognition systems” [1] .

3.3.4 Modulul de recunoaștere a numerelor de înmatriculare

Acest modul este inima aplicației și este cel care este responsabil de buna funcționare a aplicației în scopul creat.

Existau două modalități de includere a acestui modul în aplicația mea: descărcarea proiectului și includerea tuturor fișierelor în proiectul meu sau folosind repository-ul Maven-ului să includ acest pachet ca dependență și la build-ul aplicației acesta să fie automat descărcat și inclus în proiect.

Am ales a doua variantă deoarece mi s-a parut un mod mai elegant de a face asta, a redus dimensiunea proiectului meu și aplicația se preocupă singură de

dependențe la build așa că m-am putut preocupa mai mult de dezvoltarea caracteristicilor aplicației mele, plus că puteau apărea incompatibilități greu de observat.

```
@RequestMapping(value = "/recognize", method = RequestMethod.POST)
public String recognize(byte[] imageBytes){
    return plateService.makeOCR(imageBytes);
}
```

În secvența de cod de mai sus este funcția din controller care face apel la funcția din serviciu de recunoaștere a imaginii trimise. Am ales folosirea unui array de bytes pentru că este un mod ușor de transfer, înțeles de ambele părți ale aplicației, atât interfața din care provine imaginea cât și aplicația care poate lua imaginea dintr-o sursă externă.

```
public String makeOCR(byte[] image){
    InputStream inputStream = new ByteArrayInputStream(image);
    BufferedImage bufferedImage = null;
    try {
        bufferedImage = ImageIO.read(inputStream);
    } catch (IOException e) {
        e.printStackTrace();
    }
    Configurator.getConfigurator().setConfigurationFileName("C:\\Users\\VladutCostin\\Desktop\\PlatesRecognition\\config.xml");
    try {
        Intelligence intelligence = new Intelligence();
        try {
            return intelligence.recognize(new CarSnapshot(bufferedImage));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
        return null;
    } catch (SAXException e) {
        e.printStackTrace();
        return null;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
```

În funcția *makeOCR* de mai sus este implementarea funcției de serviciu care face recunoașterea imaginii. A fost nevoie de transformarea șirului de octeți într-un obiect de tipul „BufferedImage” pentru a putea fi procesat de modulul de recunoaștere. S-a setat calea către fișierul de config în care se află toți parametrii necesari funcționării modulului de recunoaștere și s-a făcut instanțierea unui obiect de tipul „Intelligence”. Acesta este obiectul capabil să recunoască numere de înmatriculare, așa că i-a fost dat ca parametru obiectul obținut mai devreme, „BufferedImage”. În cazul în care recunoașterea a fost un succes, se va returna numărul de pe plăcuța de înmatriculare sub formă de String, în caz de eroare se va returna „null”.

În continuare voi descrie puțin mai amănunțit fiecare proces prin care trece fotografia până să ajungă a fi transformată în informație utilă, și anume număr de înmatriculare valid.

3.4 Pachete și Biblioteci pentru recunoașterea numerelor

3.4.1 Principiile detecției numerelor de înmatriculare

Primul pas spre identificarea numărului de înmatriculare este de identificare a zonei dreptunghiulare de fundal alb, în cazul nostru, din fotografie. Limbajul uman pentru descrierea unei plăcuțe de înmatriculare conține cuvinte precum “plăcuță”, “identificare unică”, “numere”, “cifre”, dar o mașină nu este capabilă să înțeleagă o definiție în limbaj uman, acum cum nici definiția autovehiculului nu îi este cunoscută.

Este nevoie de traducere a limbajului natural uman în limbajul mașinăriei, astfel plăcuța de înmatriculare poate avea următoarea definiție dată de Ondrej Martinsky în lucrarea “Algorithmic and mathematical principles of automatic number plate recognition systems”: “rectangular area with increased occurrence of horizontal and vertical edges”. Aplicând doar această definiție unei fotografii, este foarte posibil ca sistemul să identifice mai multe zone candidate pentru definiția de mai sus, din acest motiv pentru identificarea corectă a zonei dorite, se va face o analiză euristică.

Folosind matrice de convoluție pentru fiecare grup de pixeli, astfel pixelul y din imaginea destinație va fi afectat de cei din jurul său așa cum se poate observa în Figura 3.4.1.1 de mai jos:

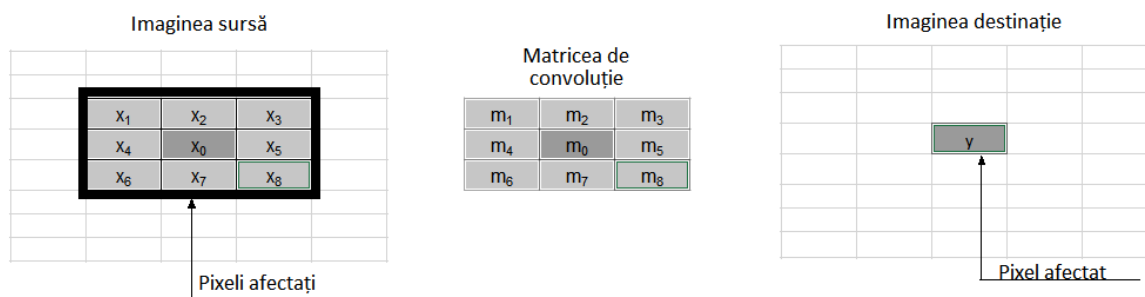


Figura 3.4.1.1 Aplicarea matricei de convoluție peste un grup de pixeli

Pentru determinarea marginilor verticale și orizontale se vor folosi tot două matrice, m_{he} și m_{ve} , Forma matricelor este cea din formula 3.4.1.1 de mai jos:

$$m_{he} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} ; m_{ve} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad (3.4.1.1)$$

3.4.2 Proiecțiile verticale și orizontale ale imaginii

După seriile de operații cu matricele de convoluție, se poate determina zona unde se află plăcuța de înmatriculare pe baza unor metode de analiză statistică. Una dintre aceste metode este proiectarea pe verticală și pe orizontală a imaginii, astfel zonele din imagine care conțin o concentrație mai mare de pixeli albi sunt zone favorabile pentru plăcuța de înmatriculare. Se face proiecția pe axele X și Y astfel încât să se obțină benzi verticale de imagine pentru proiecția pe X și benzi orizontale de imagine pentru proiecția pe Y. Proiecția pe X generează benzi înalte cât imaginea

originală și late cât zona detectată favorabil, pe când în proiecția pe Y sunt generate benzi benzi late cât imaginea originală și înalte cât zona detectată favorabilă. În aceste benzi se află informația căutată.

Folosind aceste metode se poate determina zona noastră de interes. Se extrage banda respectivă din fotografie și astfel se creează un "snapshot", o instantanee, zona de fotografie care ne interesează, restul fiind înlăturat pentru ca nu conține ceea ce ne interesează.

Cu această nouă fotografie obținută, se va trece la etapa următoare de filtrare, cea a proiecțiilor verticale, procedură asemănătoare celei de mai sus doar de aceasta dată proiecția se va face pe axa x. În urma acestei proiecții se va obține iar un grafic cu zonele de interes. Pe baza acestui grafic se va putea identifica care zone din fotografie se încadrează în cerințele formulate la început. În urma acestei filtrări se va obține o imagine mai mică, imagine care sigur conține plăcuța de înmatriculare.

În acest moment există o imagine mai mică în care se afla ceea ce ne interesează, doar că marja de eroare este încă mare și nu se poate trece la "citirea" literelor încă.

3.4.3 Identificarea și selecția zonelor candidate pentru plăcuța de înmatriculare

Din filtrările de mai sus s-a obținut o imagine în care există una sau mai multe plăcuțe de înmatriculare. Numărul maxim de candidați este limitat la nouă pentru că șansa de a fi prezente nouă plăcuțe de înmatriculare în acest cadru restrâns este foarte mică.

Sunt câteva metode euristice care sunt folosite pentru a determina costul selecției fiecărui candidat pe baza proprietăților fiecărui candidat. Acestea au fost pe obținute pe baza experimentelor de către Ondrej Martinsky în lucrarea "Algorithmic and mathematical principles of automatic number plate recognition systems" [1].

Pe baza potrivirii rezultatul căutat, lista de candidați se sortează de la cel mai probabil până la cel mai puțin probabil, apoi fiecare candidat este luat separat, începând cu cel mai probabil, și este supus unei analize euristice mai atente, mai drastice, analiza care va respinge sau va accepta candidatul. Deoarece este nevoie ca aceasta analiză să fie făcută la nivel de candidat, această procedură consumă multă putere de procesare.

Se urmărește identificarea conceptul de baza și pașii sunt următorii:

1. Detectarea numărului posibil de candidați
2. Sortarea acestora în funcție de gradul de potrivire conform cerințelor (determinat de o analiză euristică de bază)
3. Decuparea primului candidat
4. Segmentarea și analizarea acestuia cu ajutorul rețelei neurale pentru verificarea identificării unui caracter (acțiune foarte consumatoare de putere de procesare)
5. În cazul în care placuta este respinsa de algoritm, se va reveni la pasul 3.

Cele doua analize se pot descrie în felul urmator:

Prima analizează chenarul considerat candidat ca un tot unitar, bazându-se pe înălțimea imaginii, imaginile cu o înălțime mică fiind cele preferate, apoi plăcuțele cu spike-uri (vârfuri) mai numeroase în ceea ce privește proiecția verticală vor fi preferate iar apoi este calculată aria de sub fiecare vârf identificat mai devreme. Ultimul pas presupune selectarea elementelor care se află pe un singur rând, deoarece majoritatea plăcuțelor de înmatriculare din Uniunea Europeana au acest format.

Cea de-a doua analiza euristica, cea care analizeaza mai în profunzime candidații și cea care stabilește dacă bucata de imagine analizată este sau nu o placută de înmatriculare, va desface imaginea în caractere individuale din care va extrage proprietățile fiecăreia. Se va itera lista candidaților până se găsește un număr valid de înmatriculare.

Primul pas este acela de a segmenta imaginea primită în caracterele componente și i se va realiza histograma luminozității pentru a determina dacă este sau nu un număr de înmatriculare valabil prezent în imagine.

Al doilea pas este determinarea înclinației plăcuței de înmatriculare și cât de oblică este aceasta iar printr-o tehnica de “deskew”, adică corecția chenarului cu placuta de înmatriculare, se aranjează astfel încât să fie perfect dreaptă cu ajutorul unei matrici de următoarea formă:

$$A = \begin{bmatrix} 1 & S_y & 0 \\ S_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -\tan \theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.4.3.1)$$

Unde S_x și S_y sunt factori de transformare, S_x este zero și $S_y = -\tan(\theta)$ deoarece se dorește modificarea elementelor doar pe axa y .

Următoarea etapă este extragerea și normalizarea caracterelor prin normalizarea luminozității și a contrastului, apoi normalizarea folosind histograma care mărește luminozitatea zonelor întunecate fără a afecta caracteristicile globale ale imaginii.

După aceste procedee, urmează procedeul de normalizare a dimensiunilor folosind tehnica “Nearest-neighbor downsampling”, tehnică prin fiecare grup de pixeli devin un singur pixel, așa cum se poate observa în Figura 3.4.3.1, astfel se pierde din informația din imagine dar se accentuează obiectul care ne interesează cel mai mult, în cazul nostru litera pe care dorim să o recunoaștem.

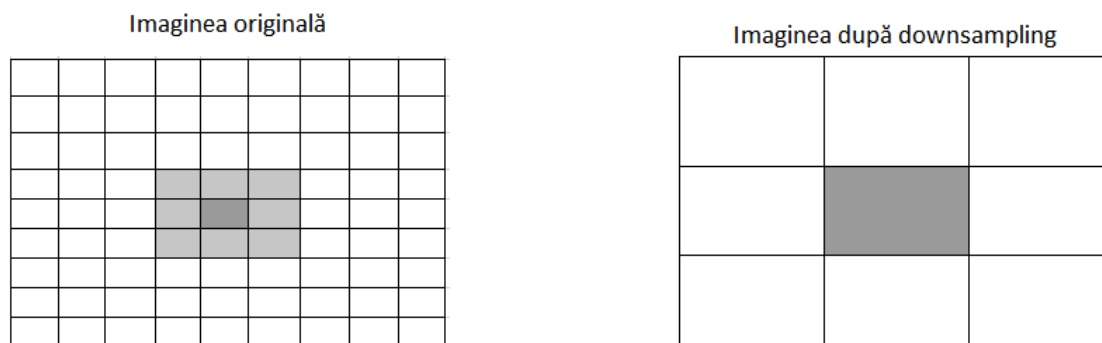


Figura 3.4.3.1 Tehnica Nearest-neighbor downsampling

Procesul de recunoastere a literelor propriu-zis foloseste retele neurale de tip feed-forward, procedeu explicat de către Ondrej Martinsky în lucrarea “Algorithmic and mathematical principles of automatic number plate recognition systems” [1].

4 Rezultate experimentale

Am testat acest sistem folosind mai multe mostre de fotografii, din mai multe unghiuri pentru a determina robustețea acestuia și gradul de acuratețe.

4.1 Cazul 1

Primul caz este într-o poziție nefavorabilă, dar condițiile de iluminare, lipsa reflecțiilor și distanța de aproximativ 2m ar trebui să asigure performanțe decente



Figura 4.1.1 Imaginea de test: numărul 1

Scenariu de test: O zi însorită, amplasarea camerei la aproximativ 2m de mașină, în lateral, o poziție nefavorabilă.

Rezultat: CL878VI

Observație: Se observă că identificarea s-a făcut eronat. Dacă aplicăm o mică corecție bazată pe modelul numărului de înmatriculare, adică cel de-al doilea "8" devine "B" se ajunge la următorul rezultat: CL87BVI. Nu este numărul corect, identificarea nu a fost făcută cu succes, în acest caz mașina nu are dreptul de acces.

4.2 Cazul 2

Am vrut să testez și comportamentul sistemului într-o poziție mai favorabilă pentru identificare, într-o zonă umbroasă cu mici reflexii. Comportamentul sistemului ar trebui să se îmbunătățească.

Pentru a avea o rată și mai bună de succes, am poziționat camera la înălțimea unei bariere dar la o distanță de aproximativ 2m de plăcuța de înmatriculare. Imaginea capturată este cea din Figura 4.2.1.



Figura 4.2.1 Imaginea de test numărul 2

Scenariu de test: O zi însorită, amplasarea camerei la aproximativ 2m de mașină, puțin deviată și înclinată.

Rezultat: CL678VL

Observație: Identificarea generată de modulul de recunoaștere a numerelor de înmatriculare a fost eronată din nou deoarece rețeaua neurală din spatele modului se așteaptă la numere de înmatriculare în format **LLCCCLL**, unde L este literă și C este cifră. Acest lucru nu este o problemă deoarece dacă aplicăm filtrarea după modelul numărului, atunci "8" va deveni "B" și astfel rezultatul după corectarea bazată pe modelul numărului va fi: CL67BVL, numărul corect de înmatriculare. Numărul este prezent în baza de date deci mașina va avea acces.

4.3 Cazul 3

Am vrut să testez și comportamentul sistemului pe timp de noapte, într-o zonă iluminată intens unde sunt și reflexii provenite de la sistemul de iluminat public. Pentru ochiul uman pare foarte ușor de recunoscut o plăcuță de înmatriculare chiar și de la distanță în astfel de condiții.

Pentru a avea o rată mai bună de succes, am micșorat distanța dintre cameră și plăcuța de înmatriculare la 1m și am poziționat camera la înălțimea unei bariere. Imaginea capturată este cea din Figura 4.3.1.



Figura 4.3.1 Imaginea de test numărul 3

Scenariu de test: Noapte, lumini stradale și camera poziționată la 1m de număr, numărul a fost capturat înclinat.

Rezultat: L67IZ

Observație: Identificarea a fost eronată grav, singurele elemente identificate bine au fost cifrele. Acest lucru se întâmplă din cauza înclinației numărului. Nu s-a efectuat o poziționare corectă și astfel foarte multă informație s-a pierdut.

4.4 Cazul 4

Am vrut să testez și comportamentul sistemului în cazul în care numărul mașinii ar fi luminat de o sursă externă pentru a determina dacă o astfel de metodă ar îmbunătăți rata de recunoaștere, camera fiind amplasată la aproximativ 3m.



Fgiura 4.4.1 Imaginea de test numărul 4

Scenariu de test: Noapte, proiectoarele unei benzinării și o sursă de lumină care să lumineze numărul mașinii.

Rezultat: null

Observație: Identificarea nu a reușit, cadrul a fost prea îndepărtat chiar și ochiului uman îi este greu să distingă numărul în aceste condiții.

5 Concluzii și dezvoltări viitoare

Această aplicație se bazează pe un modul de recunoaștere a plăcuțelor de înmatriculare distribuit sub licență educativă, deci nu poate fi vândut în această stare. Pentru a putea fi adus la o formă comercială este nevoie de găsirea unui modul open-source sau a unui care nu are astfel de restricții.

După testele efectuate cu acest modul de identificare al numerelor de înmatriculare, pare o soluție viabilă pentru următorul caz: **Poziționarea camerei într-o zonă ferită de reflexii, perpendicular pe numărul de înmatriculare, foarte bine iluminat și învățarea modelului românesc de plăcuțe de înmatriculare.**

Conform autorului acestui modul de recunoaștere a plăcuțelor de înmatriculare, Ondrej Martinsky "Algorithmic and mathematical principles of automatic number plate recognition systems", rata de succes este de 85% pentru modelul de plăcuțe de înmatriculare de tipul **LLCCLL** unde L este literă și C este cifră. În testele mele am observat că este foarte bine pusă la punct doar că necesită învățarea sistemului românesc de plăcuțe de înmatriculare altfel nu este viabilă.

Includerea unei camere cu infra-roșu ar putea ajuta la creșterea acestei rate de recunoaștere deoarece ar reduce foarte mult din problemele generate de expunere și de umbrele care pot apărea pe plăcuța de înmatriculare.

Dezvoltarea unui aplicații de mobil care să preia o parte din funcționalitățile aplicației web ar aduce mai multă popularitate în rândul acestei aplicații și ar oferi încă un avantaj în adoptarea acestui sistem la scară largă.

Dezvoltarea unui sistem de gestiune a mai multor camere pentru acest program pentru a putea fi folosit în rețele distribuite de camere, cum sunt cele din mall-uri, pentru a putea fi folosite și pe suprafețe foarte mari precum un campus universitar, un ansamblu de hale industriale sau pentru un sistem national asemenea celui pentru rovinietă.

Implementarea unui sistem de recunoaștere a mașinilor cu regim special cum sunt cele ale Serviciului de Ambulanță, Pompieri, Politie, care să permită accesul acestora în incintă fără a fi nevoie de intervenția umană.

Bibliografie

1. Ondrej Martinsky (2007). Algorithmic and mathematical principles of automatic number plate recognition systems. B.sc. Thesis
2. Documentatie Spring Framework - <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/> Accesat iunie 2016
3. Documentatie Spring Boot - <http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/> Accesat iunie 2016
4. Documentatie Spring Data JPA - <http://docs.spring.io/spring-data/jpa/docs/1.10.2.RELEASE/reference/html/> Accesat iunie 2016
5. Documentatie Spring JDBC - <http://docs.spring.io/spring-data/jdbc/docs/1.2.1.RELEASE/reference/html/> Accesat iunie 2016
6. Documentatie Maven - <https://Maven.apache.org> Accesat iunie 2016
7. Documentatie XAMPP - <https://www.apachefriends.org/ro/index.html> Accesat iunie 2016
8. Benjamin J Evans, David Flanagan (2014) - Java in a Nutshell, 6th Edition
9. Documentatie AngularJS - <https://docs.angularjs.org/api> Accesat iunie 2016
10. Documentatie Bootstrap - <http://getbootstrap.com/getting-started/#examples> Accesat iunie 2016
11. <http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages> - Accesat iunie 2016
12. Simon Kendal (2009) - Object Oriented Programming using Java