

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-98025

**EXPERIMENTÁLNA WEBOVÁ APLIKÁCIA PRE  
BIOMETRICKÝ AUTENTIFIKAČNÝ SERVER**

**BAKALÁRSKA PRÁCA**

**2021**

**Ján Vladár**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-98025

**EXPERIMENTÁLNA WEBOVÁ APLIKÁCIA PRE  
BIOMETRICKÝ AUTENTIFIKAČNÝ SERVER  
BAKALÁRSKA PRÁCA**

Študijný program: Aplikovaná informatika  
Názov študijného odboru: Informatika  
Školiace pracovisko: Ústav informatiky a matematiky  
Vedúci záverečnej práce: Ing. Pavol Marák, PhD.

**Bratislava 2021**

**Ján Vladár**



## ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Ján Vladár**

ID študenta: **98025**

Študijný program: **aplikovaná informatika**

Študijný odbor: **informatika**

Vedúci práce: **Ing. Pavol Marák, PhD.**

Miesto vypracovania: **Ústav informatiky a matematiky**

Názov práce: **Experimentálna webová aplikácia pre biometrický autentifikačný server**

Jazyk, v ktorom sa práca vypracuje: **slovenský jazyk**

Špecifikácia zadania:

Cieľom práce je navrhnúť, implementovať a otestovať web aplikáciu, ktorá vzdialene komunikuje so školským biometrickým serverom a svojim klientom ponúka biometrickú autentifikáciu a experimentálne funkcie ako vzdialená konfigurácia biometrického servera a vizualizácia výsledkov spracovania obrazu.

Úlohy:

1. Preskúmajte cieľovú oblasť biometrie, možnosti tvorby web aplikácií a zabezpečenia sietovej komunikácie.
2. Navrhnite architektúru web aplikácie, jej zabezpečenie protokolom SSL/TLS a zvoľte konkrétné vývojové nástroje.
3. Navrhnite a implementujte komunikáciu USB snímača odtlačkov prstov a web aplikácie.
4. Navrhnite a implementujte spôsob výmeny údajov medzi web aplikáciou a biometrickým serverom pomocou knižnice Google Protocol Buffers.
5. Implementujte navrhnutú web aplikáciu, ktorá umožní klientovi biometrickú autentifikáciu a experimentálne funkcie.
6. Otestujte web aplikáciu a vyhodnotiť výsledky.
7. Vytvorte písomnú dokumentáciu.

Zoznam odbornej literatúry:

1. Jain, A. – Maio, D. – Maltoni, D. *Handbook of Fingerprint Recognition, Second Edition*. London, UK: Springer, 2009. 496 s. ISBN 978-1-84882-253-5.
2. COSMINA, I. et al. *Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools*, 5th Edition. Apress, 2017.
3. MAAREK, S. *Complete Introduction to Protocol Buffers 3*. Packt Publishing, May 2018.

# SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:

Aplikovaná informatika

Autor:

Ján Vladár

Bakalárská práca:

Experimentálna webová aplikácia pre biometrický autentifikačný server

Vedúci záverečnej práce:

Ing. Pavol Marák, PhD.

Miesto a rok predloženia práce:

Bratislava 2021

Biometrická autentifikácia prostredníctvom odtlačkov prstov je rezonujúcou térou v oblasti informačnej bezpečnosti. Táto práca prináša návrh, implementáciu a testovanie webovej aplikácie, ktorá zabezpečuje komplettnú komunikáciu s biometrickým systémom na rozpoznávanie odtlačkov prstov s názvom OpenFinger, na ktorý posielajú všetky potrebné dátá súvisiace s identifikáciou osôb. Navrhli a implementovali sme taktiež dátovú komunikáciu medzi webovou aplikáciou a snímačom odtlačkov Futronic FS-80H. V prvej časti práce sme sa zamerali na vysvetlenie pojmov v oblasti biometrie a analýzu dostupných riešení v problematike tvorby webovej aplikácie, ktorá musí neustále komunikovať so vzdialeným serverom. V ďalšej kapitole sme najprv špecifikovali funkcionálne a nefunkcionálne požiadavky. Následne sme sa venovali celkovému návrhu riešenia webovej aplikácie na klientskej a aj serverovej strane, aké všetky požiadavky musí vedieť naša webová aplikácia obsluhovať a čo všetko na to potrebujeme. V tretej kapitole sa už konkrétnie venujeme implementácii webovej aplikácie a jej prepojeniu a komunikácií so systémom OpenFinger s následným zobrazovaním výsledkov a odpovedí od tohto biometrického systému. V tejto časti je taktiež popísaný celý proces od spracovania odtlačku a všetkých druhov požiadaviek, ktoré si môže používateľ zvoliť: predspracovanie, registrácia odtlačku, verifikácia, identifikácia a zobrazenie odtlačku. Tento proces pokračuje následnou serializáciou pomocou Google Protocol Buffers až po spracovanie odpovede od servera a následnou prácou s týmito odpovedami.

Klúčové slová: webová aplikácia, biometria, odtlačky prstov, OpenFinger, Google Protocol Buffers

# ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA  
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Ján Vladár
Bachelor's thesis:	An experimental web application for a biometric authentication server
Supervisor:	Ing. Pavol Marák, PhD.
Place and year of submission:	Bratislava 2021

Biometric autentification via fingerprints is a resonant topic in in the field of information security. This work provides design, implementation and test of a web application which provides complete communication with biometric system for a fingerprint recognition OpenFinger where all relevant data, about identification of users, is sent. We also designed and implemented data communication between web application and fingerprint sensor Futronic FS-80H. In the first part, we focused on explanation of terms in field of biometrics and analysis of available solutions of issues regarding creation of the web application, which has to constantly communicate with remote server. In the following chapter, we specified functional and non-functional requirements. Then, we focused on complete design of solution for the web application on user's and server's side, regarding all the requests our web application needs to provide and what is needed to do so. In the third chapter, we finally concentrate on implementation of the web application and its connection and communication with the system OpenFinger with following display of results and answers from mentioned biometric system. In this part, we also described the whole process from processing of the fingerprints and all kinds of requests which can be selected: preprocessing, fingerprint registration, verification, fingerprint identification and display of the fingerprint. This process continues with following serialization by Google Protocol Buffers to processing answers from server and subsequent work with these answers.

Keywords: web application, biometrics, fingerprints, OpenFinger, Google Protocol Buffers

## **Vyhľásenie autora**

Podpísaný Ján Vladár čestne vyhlasujem, že som bakalársku prácu Experimentálna webová aplikácia pre biometrický autentifikačný server vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.  
Uvedenú prácu som vypracoval pod vedením Ing. Pavla Maráka PhD.

Bratislava, dňa 4.6.2021

.....

podpis autora

## **Podakovanie**

Na tomto by som sa chcel podakovať vedúcemu tejto bakalárskej práce Ing. Pavlovi Marákovi PhD. za odborné rady a pripomienky. Taktiež za pomoc pri vyhľadávaní potrebných podkladov a správne nasmerovanie v začiatkoch práce. Vďaka jeho konzultáciám a odbornosti v oblasti biometrie, sa častokrát pre mňa ľahko riešiteľné problémy podarilo lepšie porozumieť a vyriešiť.

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Analýza problematiky a dostupných riešení</b>	<b>3</b>
1.1 Základné pojmy v oblasti biometrie . . . . .	3
1.2 Analýza dostupných riešení . . . . .	5
1.2.1 OpenFinger . . . . .	5
1.2.2 Komunikácia so serverom . . . . .	9
1.2.3 Šifrovaný prenos pomocou protokolu SSL/TLS . . . . .	11
1.2.4 Výber programovacích jazykov a prostredí . . . . .	12
<b>2 Návrh riešenia</b>	<b>16</b>
2.1 Funkcionálne vlastnosti aplikácie . . . . .	16
2.2 Nefunkcionálne vlastnosti aplikácie . . . . .	17
2.3 Webová aplikácia . . . . .	17
2.3.1 Frontend aplikácie . . . . .	18
2.3.2 Backend aplikácie . . . . .	19
2.3.3 Diagram udalostí . . . . .	20
2.3.4 Databáza . . . . .	21
2.3.5 Hibernate a JPA . . . . .	21
<b>3 Implementácia a dosiahnuté výsledky</b>	<b>23</b>
3.1 Spracovanie a odoslanie obrazu na server . . . . .	23
3.1.1 Komunikácia so senzorom Futronic FS-80H . . . . .	24
3.1.2 Spracovanie a získavanie dát zo senzora . . . . .	25
3.1.3 Získavanie dát z nahraného obrázka . . . . .	25
3.2 Serverová časť . . . . .	26
3.2.1 Typy požiadaviek . . . . .	27
3.2.2 Serializácia a deserializácia pomocou Google Protocol Buffers . . . . .	35
3.2.3 Komunikácia so systémom OpenFinger . . . . .	37
3.3 Sumarizácia dosiahnutých výsledkov . . . . .	37
<b>Záver</b>	<b>40</b>
<b>Zoznam použitej literatúry</b>	<b>42</b>
<b>Prílohy</b>	<b>I</b>



# Zoznam obrázkov a tabuliek

Obrázok 1	Detail papilárnych líníí . . . . .	4
Obrázok 2	OpenFinger v režime rozpoznávania pomocou odtlačkov prstov	7
Obrázok 3	Štruktúra odtlačku prsta a taxonómia Level-2 znakov . . . . .	8
Obrázok 4	Snímač odtlačkov prstov Futronic FS-80H. . . . .	8
Obrázok 5	Porovnanie rýchlosťi serializácie a deserializácie Protobuf a JSON	10
Obrázok 6	Ukážka procesu vytvorenia SSL kanála . . . . .	12
Obrázok 7	Moduly frameworku Spring vo verzii 4.0 . . . . .	14
Obrázok 8	Návrh klientskej časti aplikácie vo forme diagramu prípadov použitia. . . . .	18
Obrázok 9	Návrh backendovej časti aplikácie vo forme diagramu prípadov použitia . . . . .	19
Obrázok 10	Návrh aplikácie vo forme diagramu aktivít . . . . .	20
Obrázok 11	Návrh aplikácie vo forme diagramu aktivít . . . . .	21
Obrázok 12	Diagram, ktorý nám ukazuje mapovanie medzi Java modelom Student a databázovou tabuľkou student . . . . .	22
Obrázok 13	Používateľské rozhranie našej web aplikácie, ktoré zachytáva tvorbu požiadavky a zobrazenie výsledku predspracovania odtlačku prsta	23
Obrázok 14	Diagram, ktorý znázorňuje prepojenie odtlačku s aplikáciou . . .	24
Obrázok 15	Obsah .proto súborov pre registračnú požiadavku a odpoveď. .	27
Obrázok 16	Diagram regisitračnej požiadavky a odpovede od servera s následnými akciami . . . . .	28
Obrázok 17	Ukážka úspešnej registrácie používateľa . . . . .	28
Obrázok 18	Obsah .proto súborov pre identifikačnú požiadavku a odpoveď.	29
Obrázok 19	Diagram identifikačnej požiadavky a odpovede od servera s následnými akciami . . . . .	30
Obrázok 20	Obsah .proto súborov pre verifikačnú požiadavku a odpoveď. .	30
Obrázok 21	Diagram verifikačnej požiadavky a odpovede od servera s následnými akciami . . . . .	31
Obrázok 22	Ukážka úspešnej verifikácie používateľa . . . . .	31
Obrázok 23	Obsah .proto súborov pre požiadavku a odpoveď predspracovania. . . . .	32

Obrázok 24	Diagram požiadavky predspracovania a odpovede od servera s následnými akciami . . . . .	33
Obrázok 25	Všetky druhy výsledkov predspracovania, ktoré si môže používateľ nechať zobraziť na strane klienta . . . . .	33
Obrázok 26	Ukážka kódu, ktorý ukazuje ako zobrazíme obrázok z poľa bajtov v jazyku Java na normálny obrázok v jazyku HTML . . . . .	34
Obrázok 27	Ukážka požiadavky na zobrazenie odtlačku prsta . . . . .	35
Obrázok 28	Diagram Protobuf triedy <code>Wrapper</code> aj s jej ďalšími zapúzdrenými údajmi pri verifikačnej požiadavke . . . . .	36
Obrázok 29	Model soketovej komunikácie medzi aplikáciou a serverom aj s čítaním a mapovaním odpovede do triedy <code>Wrapper</code> . . . . .	37

# Úvod

V súčasnej spoločnosti sa čím ďalej tým viac overuje naša identita, či sú to online stránky, emaily, bankovníctvo alebo sociálne siete. Strata hesla alebo jeho krádež a následne zneužitie našich citlivých informácií je problém nejedného z nás. Existujú rôzne metódy zabezpečenia, ale v dnešnej dobe je dôležité, aby vedeli predchádzať hrozbe zneužitia a boli spoľahlivé a účinné. Jednou z možností je biometrické rozpoznanie na základe odtlačku prsta. Na to aby sme vedeli takýto systém použiť, musíme mať používateľskú aplikáciu a samotný systém na rozpoznávanie odtlačkov.

V tejto práci sa budeme venovať riešeniu ako navrhnúť takéto komunikačné rozhranie, ktoré by otvorilo možnosť ako rýchlo a spoľahlivo overovať identitu daného človeka bez potreby pamätať si heslo alebo iný identifikačný údaj, ktorý sa môže zneužiť alebo stratíť. Biometrický systém na rozpoznávanie odtlačkov prstov už navrhnutý je a vie spoľahlivo vykonávať potrebné úkony, jedná sa o biometrický systém OpenFinger vyvinutý na pracovisku ÚIM FEI STU. Riešením ako vyriešiť nasadenie takejto identifikácie pomocou rozpoznávania odtlačkov prstov je webová aplikácia, ktorá bude zabezpečovať komunikáciu s biometrickým serverom. Naša aplikácia bude vedieť takýto biometrický systém podľa potreby nastavovať a bude mu vedieť povedať, čo konkrétnie od neho vyžaduje vykonáť. Bude vedieť takúto komunikáciu vykonávať rýchlo a bezpečne, aby čo najrýchlejšie vedel používateľ či bol alebo neboli identifikovaný. Na toto všetko ale potrebuje vedieť aj uchovávať dátá o odtlačkoch v systéme a musí ich vedieť zoskenovať pomocou senzora odtlačkov prstov. Musí vedieť komu tieto odtlačky a údaje patria a vedieť ich správne použiť a poslat serveru s biometrickým systémom. Aplikácia slúži ako inteligentný spoločník, ktorý podľa požiadavky vie dať serveru všetky potrebné dátá a informácie na vykonanie určitej požiadavky ako je registrácia nového používateľa do systému a následne pridanie odtlačku. Keď už máme uložené dátá o viacerých používateľoch, tak môžeme systém využívať na ich verifikáciu alebo identifikáciu, ktorá nám poslúži spoľahlivo a jednoznačne na zistenie, či sa naozaj jedná o používateľa, ktorého overenie totožnosti vyžadujeme. Na takúto požiadavku musíme pohotovo reagovať a všetky potrebné dátá slúžiace na overenie totožnosti zabezpečene poslať systému OpenFinger. Musíme tiež zabezpečiť, aby bola táto komunikácia rýchla a mala čo najmenšiu pamäťovú náročnosť v prípadoch kedy môžeme overovať odtlačok aj voči niekoľko stovkám iných. Neskôr sa nám otvárajú aj ďalšie možnosti ako využiť aplikáciu, napríklad na zobrazenie výsledkov zo systému OpenFinger a tými sú vedieť zobraziť predspracovaný odtlačok vo viacerých formách a fázach extrakcie alebo zobraziť si samotný odtlačok zoskenovaný cez skener odtlačkov prstov.

Táto práca obsahuje kapitolu č.1, v ktorej sme sa zamerali na vysvetlenie pojmov z oblasti biometrie a analýzu dostupných riešení. V kapitole č.2 sme sa najskôr zamerali na vysvetlenie funkcionálnych a nefunkcionálnych požiadaviek a následne na celkový návrh riešenia webovej aplikácie na klientskej, ako aj na serverovej strane. Kapitola č.3 obsahuje vysvetlenie prepojenia a komunikácie so systémom OpenFinger ako aj už konkrétnu implementáciu webovej aplikácie aj s vysvetlením konkrétnych požiadaviek, ktoré si môže používateľ zvoliť: predspracovanie, registrácia odtlačku, verifikácia, identifikácia, zobrazenie odtlačku. Následne pokračujeme implementáciou serializácie konkrétnych požiadaviek pomocou Google Protocol Buffers až po spracovanie odpovede od servera.

# 1 Analýza problematiky a dostupných riešení

Predmetom tejto práce je biometria a jej prepojenie pomocou biometrického servera a webovej aplikácie, konkrétnejšie špecifická oblast biometrie a tou sú odtlačky prstov a možnosti ich využitia pri rozpoznávaní identity osôb. Pracovať budeme s biometrickým serverom OpenFinger, ktorý je plne funkčný, avšak nemá navrhnuté používateľské komunikačné rozhranie vo forme webovej alebo inej aplikácie. Pri návrhu webovej aplikácie ide o komplexné riešenie od zosnímania odtlačku až po neustále šifrované vymieňanie údajov medzi biometrickým serverom a používateľskou webovou aplikáciou. Do štruktúry základných poznatkov patrí jednoznačne zosnímanie a práca s odtlačkami prstov, následne spracovanie týchto údajov a ďalšia práca s údajmi za pomoci biometrického servera. Biometrický server musí dostať požiadavku so všetkými potrebnými údajmi, ktoré potrebuje, aby vedel čo konkrétnie má s týmito dátami vykonať a čo od neho požadujeme ako výstup. Tieto dáta musíme vedieť uchovávať pre prípadné ďalšie použitie a s následnými výsledkami a odpovedami zo servera musíme vedieť ďalej pracovať a správne ich vyhodnotiť a použiť. Dáta musíme vedieť rýchlo prenášať medzi serverom a aplikáciou a následne z nich musí aplikácia vedieť používateľovi poskytnúť žiadanú odpoveď alebo vyhodnotenie požiadavky.

## 1.1 Základné pojmy v oblasti biometrie

Na úvod si vysvetlíme aj niektoré základné pojmy z oblasti biometrie, ktoré sú východiskovým bodom pre pochopenie širších súvislostí a našich výsledkov.

**Odtlačok prsta.** Je vizuálna reprezentácia štruktúry obrazca resp. obrazcov ktoré sú tvorené papilárnymi líniemi na bruškách prstov rúk a nôh. Sú považované za jedinečné a nemenia sa v čase. Nemeniteľnosť odtlačkov prstov žačína od 4. mesiaca tehotenstva po celý život človeka. Pri porovnaní tej istej osoby v odstupe napr. 40 rokov je možné zistiť zmenu veľkosti plochy, narušenie vráskami, avšak individuálne znaky zostávajú nezmenené [1].

**Papilárne línie.** Sú vyvýšené časti pokožky na dlaňovej strane rúk a chodidlovej strane nôh, ich tvar pripomína striedavú postupnosť kožných priehlbín a vyvýšení. V roku 1668 taliansky lekár Marcello Malpighi upozornil na rozličnosť papilárnych línií na prstoch rúk

a až doteraz sa nenašli na svete dvaja ľudia s rovnakými papilárnymi líniami vo všetkých podobnostiach (ani v prípadoch jednovaječných dvojčiat). Táto individualita odtlačkov sa prejavuje hlavne v nespojitosti papilárnych línií, t.j. v ich lokálnych tvarových anomaliách: rozdvojenie a ukončenie papilárnej línie. Neodstrániteľnosť spočíva v zachovaní papilárnych línií v pôvodnom stave i pri poškodení vrchnej časti kože. Zmena nastane až pri poškodení zárodočnej, teda spodnej časti kože. Obrazce papilárnych línií nie je možné odstrániť zodratím alebo spálením kože. Takéto poškodenia sú len dočasné a po zahojení sa obnovia pôvodne obrazce [1].



Obr. 1: Detail papilárnych línií [1].

**Biometria.** Znamená využitie jedinečných a opakovateľne identifikovateľných znakov človeka na automatizované rozpoznanie identity. Biometria sa stala súčasťou našich životov a dnes je všade okolo nás. Biometria nám pomáha o niečo lepšie ochrániť veci okolo nás a väčšina biometrie slúži k bezpečnostným účelom [2].

**Biometrický systém.** Je to systém, ktorý pomocou hardvérových a softvérových prostriedkov zabezpečuje identifikáciu alebo verifikáciu osôb prostredníctvom zvoleného biometrického znaku. Ide teda o určitú formu elektronického kľúča, kde kľúčom je samotný človek. Na biometrický systém sa môžeme pozerať ako na pracovnú stanicu, ktorá pozostáva z nejakého zariadenia, na ktorom sú vykonávané všetky úlohy súvisiace s analýzou biometrických údajov a ich porovnávaním. Na vstup prídu údaje z biometrického senzora, ktorý môže byť na odlišnom mieste ako samotný server ako je tomu aj v našom prípade a tieto údaje sa môžu prenášať aj sietou [3].

**Daktyloskopia.** Je náuka o obrazcoch papilárnych línií na vnútornej strane článkov prstov. Pochádza z gréckych slov daktylos (prst) a skopeo (pozerať). Je to metóda ktorá využíva odobrané odtlačky na rozpoznanie osôb, využíva sa najmä v kriminalistike alebo v autentifikácii osôb a povolení alebo zamietnutí ich prístupu do systému [1].

**Level-2 (druhostupňové znaky).** Sú to vlastnosti odtlačkov, ktoré sa prejavujú na lokálnej úrovni. Ak sa pozrieme hlbšie do štruktúry obrazcov, objavíme prítomnosť malých nepravidelností resp. anomálií v papilárnych líniach, ktorých distribúcia a poloha sú jedinečnou charakteristikou pre každého jedinca. Tieto Level-2 znaky nesú aj názvy ako daktyloskopické markanty alebo charakteristické znaky. Sú základom pre takmer všetky biometrické systémy. Sú považované za ústredný prvok jedinečnosti odtlačkov. Najfrekventovanejšie Level-2 vzory sú rozdvojenia a ukončenia papilárnej línie. Každý odtlačok obsahuje iný počet markantov. Christophe Champod so spoluautormi vo svojej publikácii Fingerprints and Other Ridge Skin Impressions [4] uvádza, že priemerná hustota markantov v okolí singulárnych bodov je 0,49 markantu na  $1mm^2$  a v oblastiach mimo singulárnych bodov je to 0,18 markantu na  $1mm^2$ . Techniky porovnávajúce Level-2 znaky hľadajú podobnosť v polohách a orientácii daktyloskopických markantov alebo ich zoskupení, aby sa zabezpečila odolnosť voči tomu aby neboli problém ak je rovnaký prst vždy zosnímaný s inou deformáciou [3].

## 1.2 Analýza dostupných riešení

Cieľom analýzy je porovnať a analyzovať momentálne dostupné riešenia zamerané na tvorbu webových aplikácií, komunikáciu s biometrickým serverom a samotnú biometriu. V práci spomenieme tie, ktoré považujeme za najprepracovanejšie, resp. najzaujímavejšie z nich.

### 1.2.1 OpenFinger

Táto časť obsahuje opis systému OpenFinger [3], s ktorým komunikuje naša aplikácia. Tento autentifikačný systém je nainštalovaný na školskom výkonného serveri s operačným systémom Debian 10. OpenFinger je zložený zo 4 hlavných modulov.

- **Biometrický modul spracovania odtlačkov prstov** predstavuje kompletný samostatne fungujúci unimodálny systém.
- **Biometrický modul spracovania obrazcov žíl na prste** predstavuje kompletný samostatne fungujúci unimodálny systém.
- **Modul fúzie**, ktorý spája skóre porovnania z obidvoch unimodálnych systémov a umožňuje tak multimodálny režim činnosti.
- **Modul automatizovaného vyhodnotenia úspešnosti** jednotlivých unimodálnych systémov ako aj multimodálneho systému (zloženého z oboch unimodálnych systémov, pri ktorých je využitá fúzia na úrovni porovnávania). Vypočítané ukazovatele sú vizualizované vo forme interaktívnych grafov.

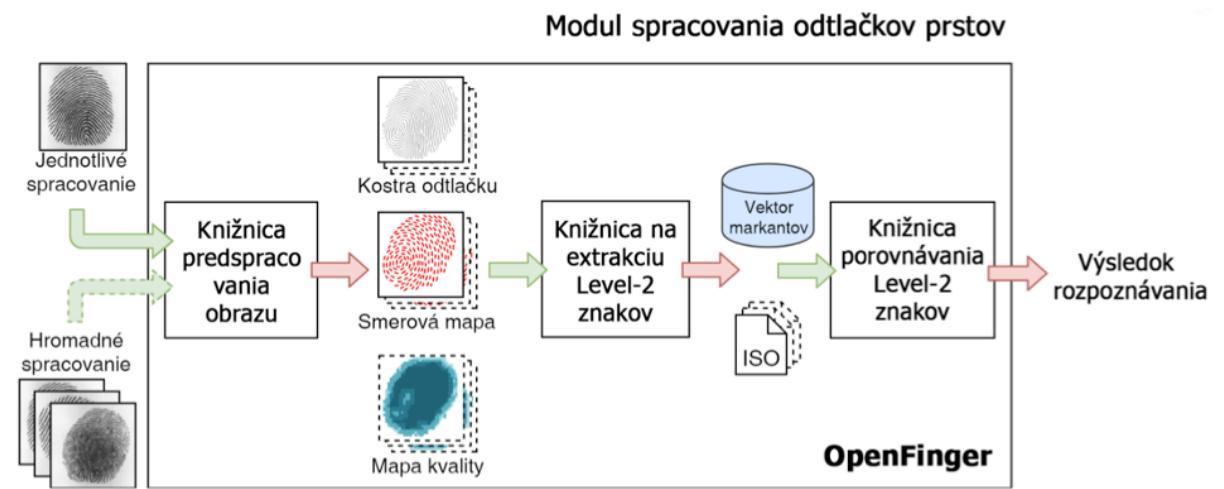
Obidva biometrické moduly pracujú v troch hlavných fázach: predspracovanie, extrakcia charakteristických vlastností a ich porovnávanie. Celú architektúru systému OpenFinger zachytáva bloková schéma na obrázku 1.2.

Školský biometrický server, na ktorom je nainštalovaný OpenFinger bol rozšírený aj o rozhranie, ktoré umožňuje vykonávať verifikáciu a identifikáciu v počítačovej sieti pomocou šifrovaného TCP sietového spojenia zabezpečeného protokolom SSL/TLS. Klientom je naša aplikácia, ktorá sa pripojí na port a IP adresu servera. Klient pomocou našej webovej aplikácie zoskenuje a odošle zašifrovaný obrázok odtlačku, ktorý server spracuje. Takáto možnosť a nadstavba vo forme našej aplikácie demonštruje jeho praktické využitie. My v našej aplikácii využívame na vstup odtlačok prsta a preto využívame prvý spomenutý, biometrický modul spracovania odtlačkov prstov.

**Modul spracovania odtlačkov prstov.** Vstupom je obrázok odtlačku získaný priamo z jedného z externých USB senzorov alebo z databázy odtlačkov, v našom prípade sa jedná o senzor Futronic FS-80H, ktorý je jedným z troch modelov snímačov odtlačkov prstov, pre ktoré je v knižnici zabudovaná podpora, to nám umožňuje prenášať obraz aj vo formáte pola bajtov. Obrázok prejde transformáciou v knižnici predspracovania, kde je klúčovou operáciou aplikovanie adaptívneho Gaborovho filtra. Takto kvalitatívne upravený obrázok smeruje do knižnice na extrakciu, ktorá využíva konvolučnú sieť z architektúry ResNet-18, ktorá bola natrénovaná pomocou vzoriek Level-2 znakov zozbieraných softvérovým nástrojom OF Sample Collector na interaktívny zber a úpravu rôznych druhov trénovacích vzoriek zo zvolenej databázy obrazov. Konvolučná sieť odhalí v obraze Level-2 znaky a tie sú následne zaslané do našej aplikácie na uloženie do databázy re-

gistrovaných používateľov (v režime registrácie). Alebo sú poslané do knižnice určenej na porovnávanie. Podľa toho, či systém pracuje v režime verifikácie alebo identifikácie, sa odhalené Level-2 znaky porovnávajú s jednou alebo viacerými identitami, ktoré aplikácia vyberie z databázy a pošle opäť na vyhodnotenie a porovnanie. Porovnanie funguje pomocou odhalených Level-2 znakov a je realizované jedným z dvoch algoritmov, ktoré sú na výber: BOZORTH3 alebo Suprema Biomini SDK [3].

Na obrázku vidíme blokovú schému zachytávajúcu hlavné kroky spracovania odtlačku. Modul dokáže fungovať v móde spracovania jedného odtlačku prsta ako aj v móde hromadného paralelného spracovania viacerých obrazov.



Obr. 2: OpenFinger v režime rozpoznávania pomocou odtlačkov prstov [3].

Na obrázku nižšie môžeme vidieť taxonómiu Level-2 znakov, ktoré ukladáme do našej databázy a podľa ktorých na základe požiadavky z webovej aplikácie systém OpenFinger vykonáva porovnanie s aktuálne nasnímaným odtlačkom. Na obrázku napravo vidíme daktyloskopické markanty.

Minutiae	Example	Minutiae	Example
ridge ending		bridge	
bifurcation		double bifurcation	
dot		trifurcation	
island (short ridge)		opposed bifurcations	
lake (enclosure)		ridge crossing	
hook (spur)		opposed bifurcation/ridge ending	



Obr. 3: Štruktúra odtlačku prsta a taxonómia Level-2 znakov [5].

**Futronic FS-80H** je snímač od spoločnosti , s ktorým komunikuje naša aplikácia a zároveň je jedným z troch snímačov, pre ktorý je zabudovaná podpora v systéme OpenFinger. Futronic FS-80H je snímač rozlíšením 500 PPI, s veľkosťou obrazu 320 x 480px a váhou 120g, ktorý umožňuje detekciu živého prsta na povrchu senzora. Snímacia plocha má rozmer 16 x 24 mm, obsahuje infračervené LED osvetlenie a optický CMOS senzor [6].



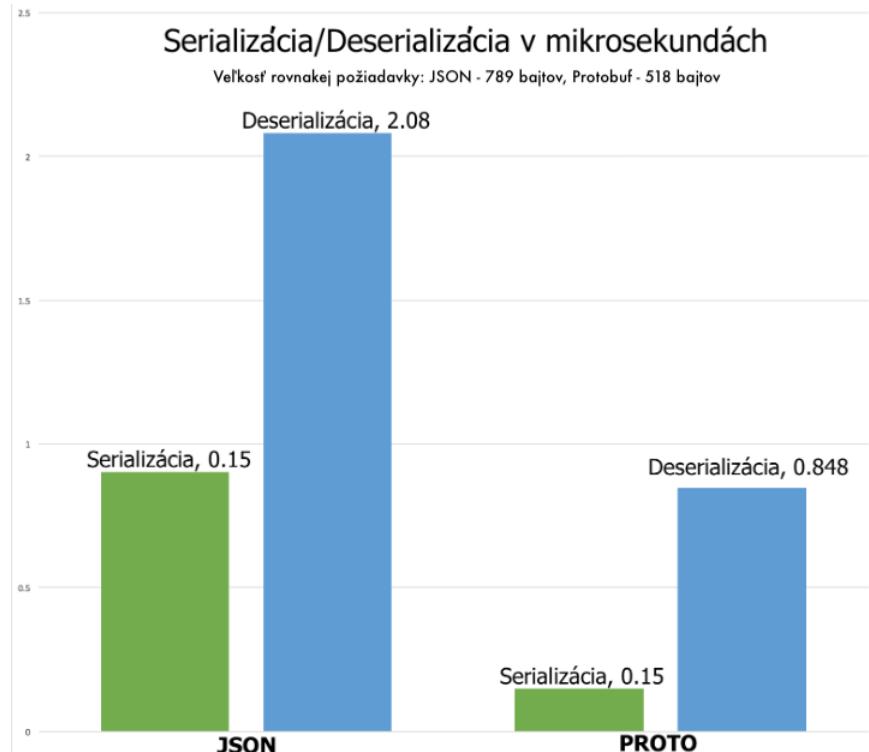
Obr. 4: Snímač odtlačkov prstov Futronic FS-80H.

### 1.2.2 Komunikácia so serverom

Hlavná funkcia aplikácie je správne a rýchlo komunikovať zo vzdialeným systémom OpenFinger na školskom serveri. Naša aplikácia je napísaná v jazyku Java a systém OpenFinger v jazyku C++. Z dôvodu, že naša komunikácia musí byť rýchla a zároveň prenášame pomerne veľké dátu, ktoré musia byť navzájom bez problémov prečítateľné tak nižšie si porovnáme dva vybrané a podľa nás najlepšie formáty na takúto komunikáciu a to sú formáty Google Protocol Buffers a JSON.

- **Google Protocol Buffers** tiež známy ako Protobuf je protokol, ktorý spoločnosť Google vyvinula s cieľom umožniť serializáciu a deserializáciu štruktúrovaných údajov medzi rôznymi službami a programovacími jazykmi. Cieľom Google bolo vytvoriť open-source metódu ako XML, avšak menšiu, rýchlejšiu a jednoduchšiu na vzájomnú komunikáciu. Vývojári kládli dôraz na jednoduchosť a výkon. Samotná spoločnosť Google využíva Protobuf na ukladanie a výmenu štruktúrovaných informácií všetkého druhu. Dátové štruktúry (nazývané „messages“) a služby sú opísané v súbore `.proto` a sú kompliované s `protoc` príkazom. Vo vygenerovanom kóde sú všetky potrebné metódy ako napríklad `get`/`set` metódy a taktiež metóda na serializáciu a deserializáciu z pola bajtov. Správy sú serializované do binárneho formátu podľa dopredu zadefinovanej štruktúry napísanej v jazyku Protobuf, takúto binárnu správu vieme potom posieláť alebo prijímať v našej komunikácii so systémom OpenFinger napísanom v jazyku C++ [7]
- **JSON** alebo JavaScript Object Notation je formát, ktorý sa tiež používa na štruktúrovanie údajov. JSON je tiež alternatíva k XML a používa sa hlavne ako spôsob prenosu údajov medzi serverom a webovou aplikáciou. JSON používa spôsob lepsie čitateľný človekom ako kombináciu zloženú z hodnôt, z ktorých každý ma jedinečný kľúč. Formát JSON umožňuje ľubovoľné vnáranie slovníkov a zoznamov. Môžeme tak napríklad vnoriť slovník do iného slovníka alebo zoznam do slovníka. Týmto spôsobom možno modelovať takmer akúkoľvek dátovú štruktúru, či už jednoduchú alebo zložitú, čo z formátu JSON robí výkonný a zároveň jednoduchý nástroj na usporiadanie hodnôt ako dátových objektov. Praktické využitie je napríklad ak posielame JSON dátu s použitím Ajaxu, čím si vieme vymeniť dátu medzi web aplikáciou, webovým prehliadačom a serverom bez potreby znova načítania stránky [8].

Následne by sme chceli porovnať tieto dva formáty na základe faktov, podľa ktorých sme sa rozhodli pretože tieto formáty je možné používať zameniteľne, obidve boli však navrhnuté s ohľadom na rôzne ciele. JSON vznikol z podmnožiny programovacieho jazyka JavaScript a jeho správy sa vymieňajú v textovom formáte, sú úplne nezávislé a podporované takmer všetkými programovacími jazykmi. Protobuf nie je len formát správy, ide súčasne o súbor pravidiel a nástrojov, ktoré si navzájom vymieňajú správy, okrem toho má viac dátových typov ako JSON, ako sú enumerátory a metódy. Protobuf sa ľahšie viaže na objekty, porovnanie reťazcov v JSON môže byť pomalé. Pretože JSON je textový formát, tak kódovanie a dekódovanie celých čísel a desatinnych čísel môže byť pomalé. JSON nie je určený pre čísla. Formát JSON je užitočný, ak chcete, aby boli údaje lepšie čitateľné človekom, pretože ku každej hodnote má aj klúč. Ak je naša aplikácia napísaná v JavaScripte, väčší zmysel má určite JSON. Analýza reťazcov, polí a objektov JSON vyžaduje postupné skenovanie, čo znamená, že dopredu v hlavičke nie je známa veľkosť ani počet prvkov. Práca s číslami by mala byť v Protobufe výrazne rýchlejšia a zabezpečuje nám to predovšetkým spätnú kompatibilitu. Protobuf býva pri kódovaní celého čísla rýchlejší ako JSON. Pre dvojité dekódovanie sa Protobuf ukázal ako podstatne rýchlejší než JSON [9]. Porovnanie rýchlosťi týchto dvoch formátov na rôznych grafoch:



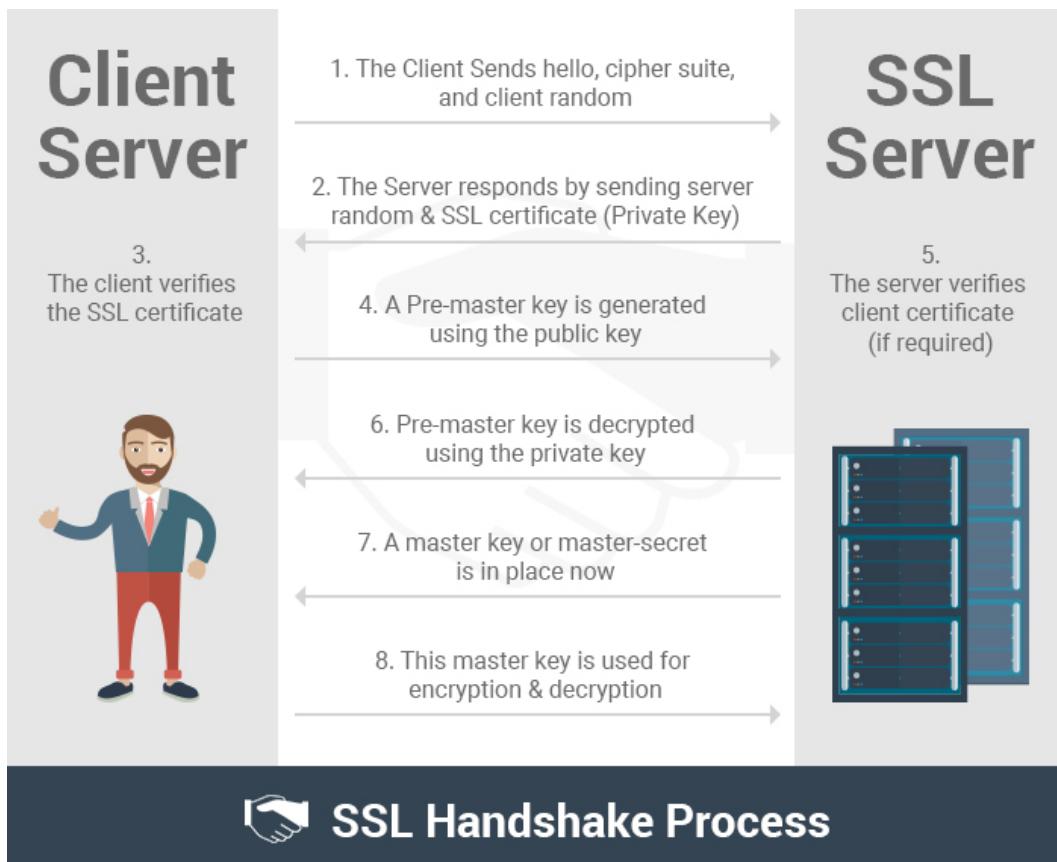
Obr. 5: Porovnanie rýchlosťi serializácie a deserializácie Protobuf a JSON [10].

### 1.2.3 Šifrovaný prenos pomocou protokolu SSL/TLS

V rámci komunikácie medzi webovou aplikáciou a systémom OpenFinger budeme posielat citlivé údaje, ktoré slúžia na účely overenia identity. Sú to súkromné dátá, ktoré z dôvodu možného zneužitia, nemôžu byť prenášané nezašifrované. Na šifrovanú komunikáciu sa používajú SSL/TLS certifikáty, ktoré patria medzi nevyhnutnú súčasť zabezpečenia internetu. SSL/TLS certifikáty sa dnes používajú k akémukoľvek zabezpečeniu na internete. Či sa už jedná o prihlásovacie stránky a administráciu, nakupovanie, transakcie kreditnými kartami alebo ako v našom prípade, prenos biometrických dát.

SSL certifikáty, správnejšie nazývané TLS certifikáty zaistujú ochranu prenášaných dát pomocou šifrovania a umožňujú identifikáciu prevádzkovateľa webovej stránky. Zásadný zlom pre nasadzovanie SSL certifikátov nastal v roku 2016, kedy Google oznámil, že stránky s adresou HTTPS budú mať vo vyhľadávania oproti HTTP prioritu. Pod heslom „HTTPS everywhere“ prebiehajú kampane na rozšírenie zabezpečenej komunikácie, podporené webovými prehliadačmi, ktoré označujú nezabezpečené webstránky negatívnym označením "NOT SECURE".

**Secure Socket Layer (SSL)** bol vytvorený spoločnosťou Netscape pre zaistenie bezpečnej komunikácie a prenosu dát medzi webovými servermi a prehliadači. Protokol používa dôveryhodnú tretiu stranu, certifikačnú autoritu (CA), ktorá identifikuje jednu alebo obe strany komunikácie. Z technického pohľadu sa jedná o vrstvu vloženú medzi transportnú (napr. TCP / IP) a aplikačnú (napr. HTTP) vrstvu, ktorá poskytuje zabezpečenie komunikácie šifrovaním a autentizáciu komunikujúcich strán. Nasledovníkom SSL je štandardizovaný protokol Transport Layer Security (TLS). SSL komunikácia funguje na princípe asymetrickej šifry, kde každá z komunikujúcich strán má dvojicu šifrovacích kľúčov - verejný a súkromný. Verejný kľúč je možné zverejniť a ak týmto kľúčom ktokoľvek zašifruje nejakú správu, je zabezpečené, že ju bude môcť rozšifrovať len a len majiteľ použitého verejného kľúča svojím súkromným kľúčom [11].



Obr. 6: Ukážka procesu vytvorenia SSL kanála[12]

#### 1.2.4 Výber programovacích jazykov a prostredí

V tejto časti práce sa budeme bližšie venovať výberu vývojových prostredí a programovacích jazykov, resp. porovnaniu frameworkov. V našej aplikácii sme riešili komplet riešenie od backendu a prác s databázou a serverom až po frontend. Museli sme si preto vybrať vývojové prostredie, ktoré bude mať plnú podporu pre dané komplexné riešenie. Na základe porovnania dostupných riešení sme sa rozhodli v práci použiť nástroj, resp. framework Spring a programovací jazyk Java, ktorý nám po porovnaní všetkých dostupných nástrojov vyšiel ako najlepšia voľba. Na prácu s databázou sme používali jazyk SQL. Frontend aplikácie sme písali v jazyku HTML, Javascript a CSS.

**IntelliJ IDEA.** Vývojové prostredie IntelliJ IDEA je produkтом českej softvérovej firmy JetBrains. Prvá verzia vyšla už v roku 2001 a neustále sa vyvíja. Každý aspekt produktu IntelliJ IDEA bol navrhnutý tak, aby maximalizoval produktivitu vývojárov. Vďaka inteligentnej asistencii pri programovaní a peknému dizajnu je vývoj nielen produktívny, ale aj príjemný. Toto vývojové prostredie je určené pre jazyk Java, no rozumie a vie pracovať aj s inými jazykmi, ktoré sú potrebné pre vývoj webových aplikácií [13]. Sú to jazyky

JavaScript, HTML, JPQL a aj SQL, ktoré využívame na prácu s databázou. Z týchto dôvodov až 72% všetkých Java vývojárov používa práve IntelliJ IDEA. Jedna z ďalších dôležitých vecí je fakt, že IntelliJ IDEA poskytuje bohatú sadu vstavaných nástrojov a vynikajúcemu podporu frameworku Spring. Už hned pri vytváraní projektu si môžeme zvoliť typ projektu Spring Initializr, ktorý sa integruje s API Spring Initializr na generovanie a importovanie nového projektu. Sprievodca projektom nás prevedie úvodnou konfiguráciou, kde môžeme určiť názov, verziu Java, nástroj na zostavenie, ďalšie rozšírenia a ďalšie možnosti pre náš Spring projekt [14].

**Spring.** Jedná sa o najpoužívanejší Java vývojársky framework súčasnosti pre vývoj Java EE aplikácií. Prvá verzia bola napísaná Rodom Johnsonom, ktorý ju vydal v októbri 2002. Rod Johnson sa vo svojej knihe zaobrá vývojom Java EE aplikácií a venuje pozornosť problémom, s ktorými sa programátori stretávajú. V knihe je priebežne prezentovaný kód frameworku, ktorý sa nazýva Interface21, a mal by vývoj Java EE aplikácií uľahčiť. Za pomocí Juergen Hoellera je neskôr framework rozšírený a pod názvom Spring Framework uvoľnený ako open-source. Spring Framework môže byť použitý ľubovoľnou Java aplikáciou. Spring sa stal populárny v Java komunite ako alternatíva k Enterprise Java Beans (EJB), alebo ako jeho nadstavba.

## Dôvody vzniku.

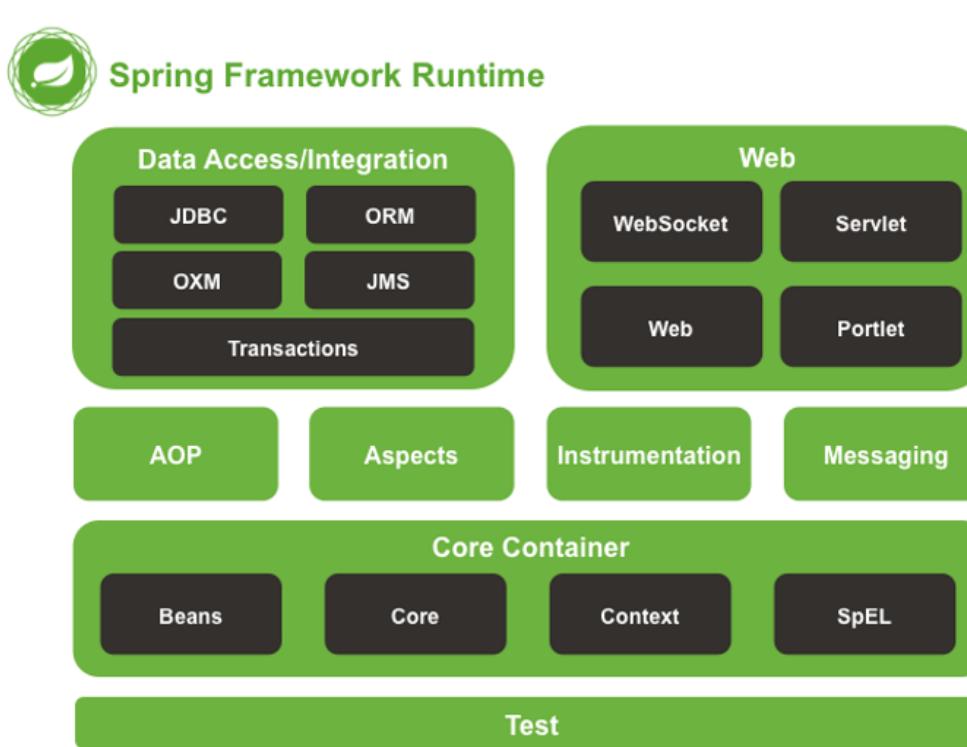
- Odstránenie tesných programových väzieb jednotlivých POJO objektov a vrstiev pomocou návrhového vzoru Inversion of Control.
- Podpora implementácie komponentov pre prístup k dátam, či už formou priameho JDBC či ORM (object-relation mapping) technológií a nástrojov ako je Hibernate, TopLink, iBatis alebo JDO.
- Lahká rozšíritelnosť pomocou modulov
- Abstrakcia vedúca k zjednodušenému používaniu ďalších častí Java EE, ako napríklad JMS, JMX, JavaMail, JDBC, JCA alebo JNDI.
- Podpora zabezpečenia pomocou Spring Security
- Podpora AOP(aspektovo-orientované programovanie)

Jadro Springu je postavené na využití návrhového vzoru Inversion of Control a je označovaný ako IOC kontajner. Tento návrhový vzor funguje na princípe presunutia zodpo-

vednosti za vytvorenie a previazanie objektov z aplikácie na framework. Objekty možno získať prostredníctvom vsádzania závislostí, čo je špeciálny prípad Inversion of Control. Dependency Injection rieši vlastný spôsob vloženia objektov. Základné tri spôsoby vloženia objektov sú Setter Injection, Constructor Injection a Interface Injection. Objekty vytvorené kontajnerom sú nazývané JavaBeans. Objekty sú frameworkom vytvorené typicky na základe načítania konfiguračného súboru vo formáte XML, ktorý obsahuje definície týchto Beans. Spring Framework sa nezaoberá riešením už vyriešených problémov. Namiesto toho využíva preverené a dobre fungujúce open-source nástroje, ktoré v sebe integruje. Tým sa stáva ich použitie často jednoduchšie.

Spring je modulárny framework: Umožňuje využiť napríklad len časť, ktorá sa hodí k riešeniu daného problému. Účelom Spring je zjednodušenie návrhu J2EE aplikácií so zamieraním na architektúru aplikácie (miesto na technológiu), na jednoduchú testovateľnosť, na neinvazívnosť a modulárnosť.

Spring sám o sebe len spája dohromady časti aplikácie, zodpovedá za správnu inicializáciu a správu objektov, avšak nerieši žiadnu funkčnosť aplikácie ako takej [15]. Takúto podporu ako prípadne rozšírenie funkčnosti rieši cez prídavné moduly ktoré sa neustále jemne upravujú, vo verzii Spring 4.0 vyzerali nasledovne :



Obr. 7: Moduly frameworku Spring vo verzii 4.0 [16].

**Spring MVC.** Modul Servlet obsahuje Spring implementáciu model-view-controller (MVC) pre webové aplikácie, ktorá je veľmi populárna. Výhoda MVC vzoru je v tom, že každá časť vzoru sa sústredí na niečo iné a tak vieme pekne rozložiť celú aplikáciu. Napríklad vieme dátá z Controllera posunúť na iný View. Alebo Controller požiada o vrátenia dát inú business službu, ktorá sa špecializuje na niečo iné. Controller teda vie kto sa špecializuje na určité podmienky, napríklad ak chceme v controlleri zoznam odtlačkov prstov, tak sa pošle požiadavku z controllera do business služby a tá vráti zoznam odtlačkov. Controller získa zoznam odtlačkov a posunie ho do View – teda napríklad na zobrazenie. View vie ako zobraziť dátá, ktoré prijal. Model sú dátá, ktoré nám prúdia aplikáciou [17]

## 2 Návrh riešenia

Cieľom tejto bakalárskej práce je navrhnúť webovú aplikáciu, ktorá bude komunikovať so vzdialeným serverom, na ktorom je systém OpenFinger. Komunikácia bude prebiehať na základe rôznych požiadaviek, ktoré si používateľ zadá a odošle spolu so zoskenovaným alebo nahraným odtlačkom prsta. Používateľ zadá požiadavku na naskenovanie jeho odtlačku cez externý snímač Futronic FS-80H. Takto zoskenovaný odtlačok musíme aj s ďalšími požiadavkami rôzneho typu spracovať a vo vhodnom formáte poslať systému OpenFinger a vzápäť spracovať jeho odpoveď, ktorú požadujeme. Odpoved musíme podľa jej typu spracovať a vložiť do databázy pre prípadné ďalšie použitie alebo čo najrýchlejšie zobrazit na strane klienta.

### 2.1 Funkcionálne vlastnosti aplikácie

- Prepojenie senzora Futronic FS-80H s webovou aplikáciou a jeho ovládanie zo strany klienta. Klient odošle požiadavku na zoskenovanie odtlačku prsta, ktorú musí aplikácia spracovať a poslať požiadavku na skener pripojený cez USB port.
- Ďalšia možnosť ako získať odtlačok prsta je nahrať zoskenovaný odtlačok ako obrázok z disku zariadenia, na ktorom beží aplikácia a vybrať, na aký účel má byť tento odtlačok použitý.
- Odtlačok, ktorý príde zo skenera alebo z obrázku sa musí vhodne spracovať a poslať v správnej forme na server, ktorý musí podľa danej požiadavky vedieť, čo s ním má spraviť a akú má poslať spätnú odpoveď ako výsledok. Celý objekt, ktorý posielame na server preto nemôže vyzerat stále rovnako z dôvodu viacerých požiadaviek, ktoré môže používateľ zadať a vyžadovať od našej aplikácie.
- Používateľ sa musí vedieť svojim odtlačkom zaregistrovať. Musí zadať požiadavku na registráciu aj s jeho osobnými údajmi, aby sme vedeli komu tento odtlačok patrí. Tento odtlačok alebo jeho špecifikácie, ktoré získame si musíme uložiť do databázy.
- Používateľa musíme vedieť verifikovať na základe jeho odtlačku a zadaného mena, pod ktorým by mal byť nahraný v databáze používateľov. Takto zoskenovaný odtlačok musíme vedieť porovnať s jeho ďalšími už uloženými odtlačkami.
- Používateľa môžeme chcieť aj identifikovať podľa jeho odtlačku prstu a tým zistíť, či sa už nachádza takýto používateľ v databáze. Zoskenovaný odtlačok musíme

porovnať so všetkými dostupnými odtlačkami v systéme.

- Používateľ si môže zvoliť aj možnosť predspracovania odtlačku prsta aj s parametrami pre nastavenie OpenFingeru, s ktorými si vie nastaviť bližšie špecifikácie požiadavky. Zoskenovaný odtlačok aj s príslušnými údajmi pošleme systému OpenFinger, ktorý nám vráti kostru daného odtlačku.
- Používateľ si môže zvoliť aj možnosť zobrazenia odtlačku, ktorý sme mu zoskenovali pomocou senzora alebo nahrali z úložiska.
- Používateľ si musí vedieť pridať do jeho databázy odtlačkov aj ďalší prst alebo ďalší odtlačok tak aby mu bolo umožnené prípadne overenie identity aj s iným prstom alebo odtlačkom. Takto zoskenovaný odtlačok sa musí správne priradiť do databázy k už existujúcemu používateľovi.
- Aplikácia musí vedieť v reálnom čase vyhodnocovať, ukladať do databázy a zobrazať odpovede zo servera na strane klienta, ktoré môžu byť rôzneho typu závisle od druhu požiadavky ktorú pošleme systému OpenFinger

## 2.2 Nefunkcionálne vlastnosti aplikácie

- Minimálna spotreba dát a čo najrýchlejšia komunikácia so serverom na základe zoserializovaných dát v dopredu určenom formáte, ktoré sa musia rýchlo poslať systému OpenFinger a čo najrýchlejšie musíme vedieť jeho odpoved aj spracovať.
- Minimálna veľkosť posielaných dát z ktorých budeme chcieť zobraziť výsledok na strane klienta. Pri verifikácii a identifikácii, kde posielame viac odtlačkov musíme dbať na veľkosť týchto všetkých údajov a preto si ich vždy pri registrácii pripraviť a uložiť len ich jedinečné a potrebné údaje.

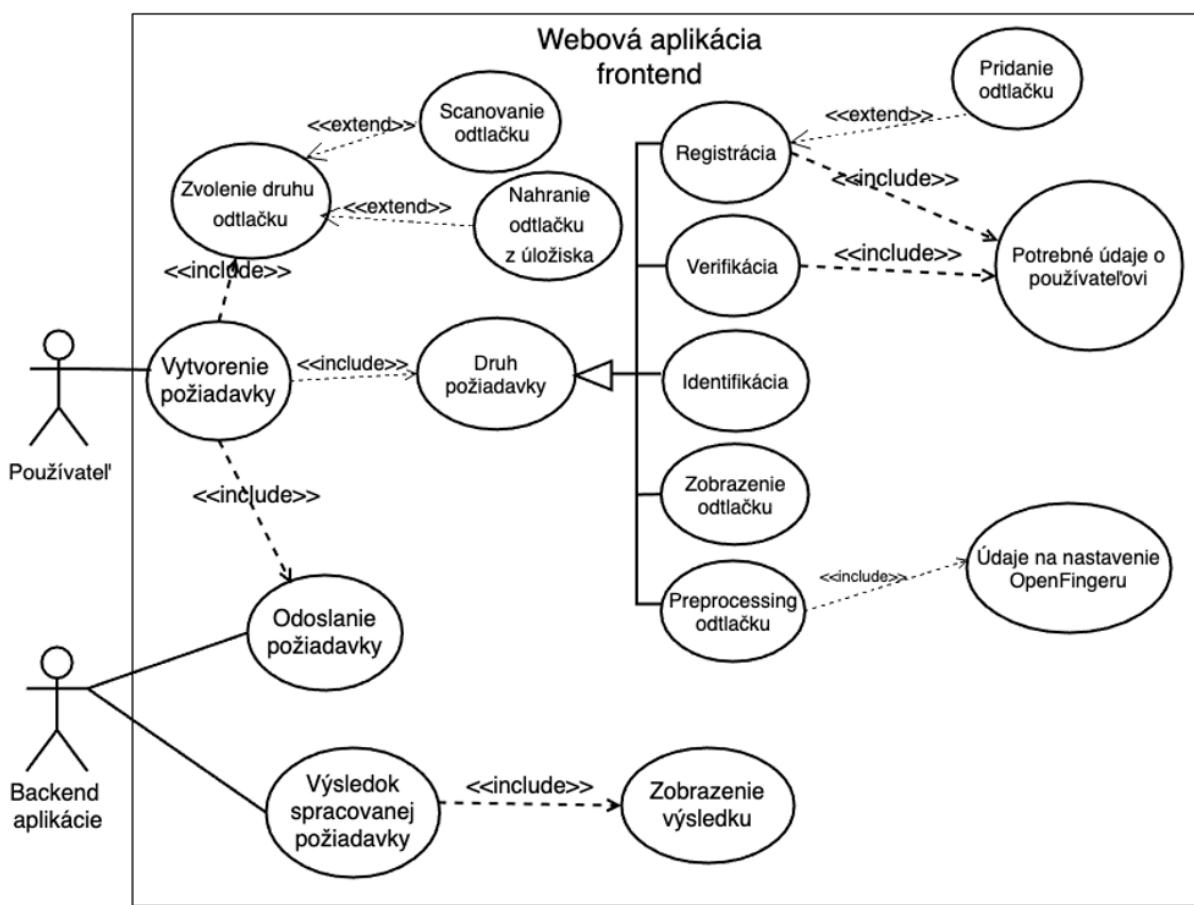
## 2.3 Webová aplikácia

Na základe analýzy vývojových prostredí a osobných skúseností sme si zvolili jazyk Java a framework Spring. Vďaka Springu sa nám bude lepšie robiť prepájanie jednotlivých častí aplikácie a samotná implementácia. Táto aplikácia bude tvoriť klientskú časť už spomínaného biometrického systému OpenFinger, ktorá bude slúžiť na využívanie tohto systému aj z diaľky bez potreby byť priamo v blízkosti servera, na ktorom OpenFinger funguje. Pri spustení aplikácie sa používateľovi zobrazí menu, v ktorom si vyberie požiadavku čo vyžaduje od aplikácie a zadá svoje prístupové osobné údaje. Zároveň si zvolí druh akým chce nahrať svoj odtlačok a odošle požiadavku. Požiadavka sa v aplikácii podľa jej druhu

spracuje a odošle systému OpenFinger, ktorý nám taktiež podľa druhu požiadavky pošle odpoveď. Túto odpoveď spracujeme a podľa jej typu si ju uložíme do databázy a zobrazíme na strane klienta.

### 2.3.1 Frontend aplikácie

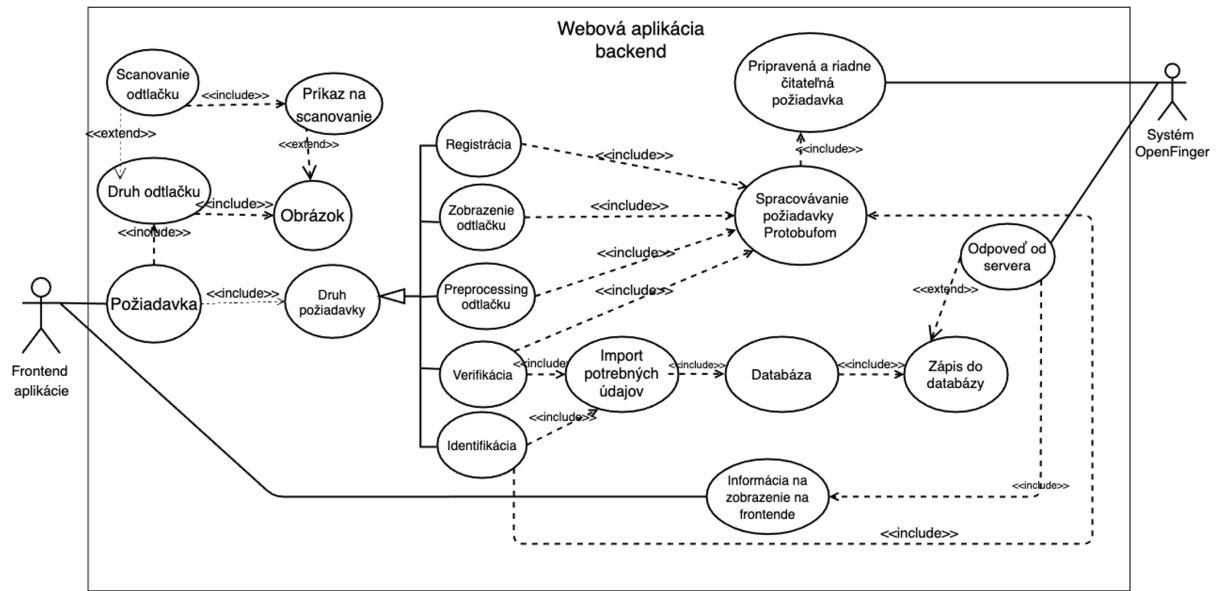
Klientská časť aplikácie najprv obslúži používateľa, ktorý nám odošle požiadavku s jeho osobnými údajmi, ktoré sú pri každej požiadavke iné. Zvolí si taktiež ako chce svoj odtlačok poslať na spracovanie. To, ako môžu vyzerať tieto požiadavky a čo všetko si používateľ môže zvoliť si ukážeme na diagrame prípadov použitia nižšie. Následne sa nám na konci ako výsledok zobrazí požadovaný výsledok požiadavky



Obr. 8: Návrh klientskej časti aplikácie vo forme diagramu prípadov použitia.

### 2.3.2 Backend aplikácie

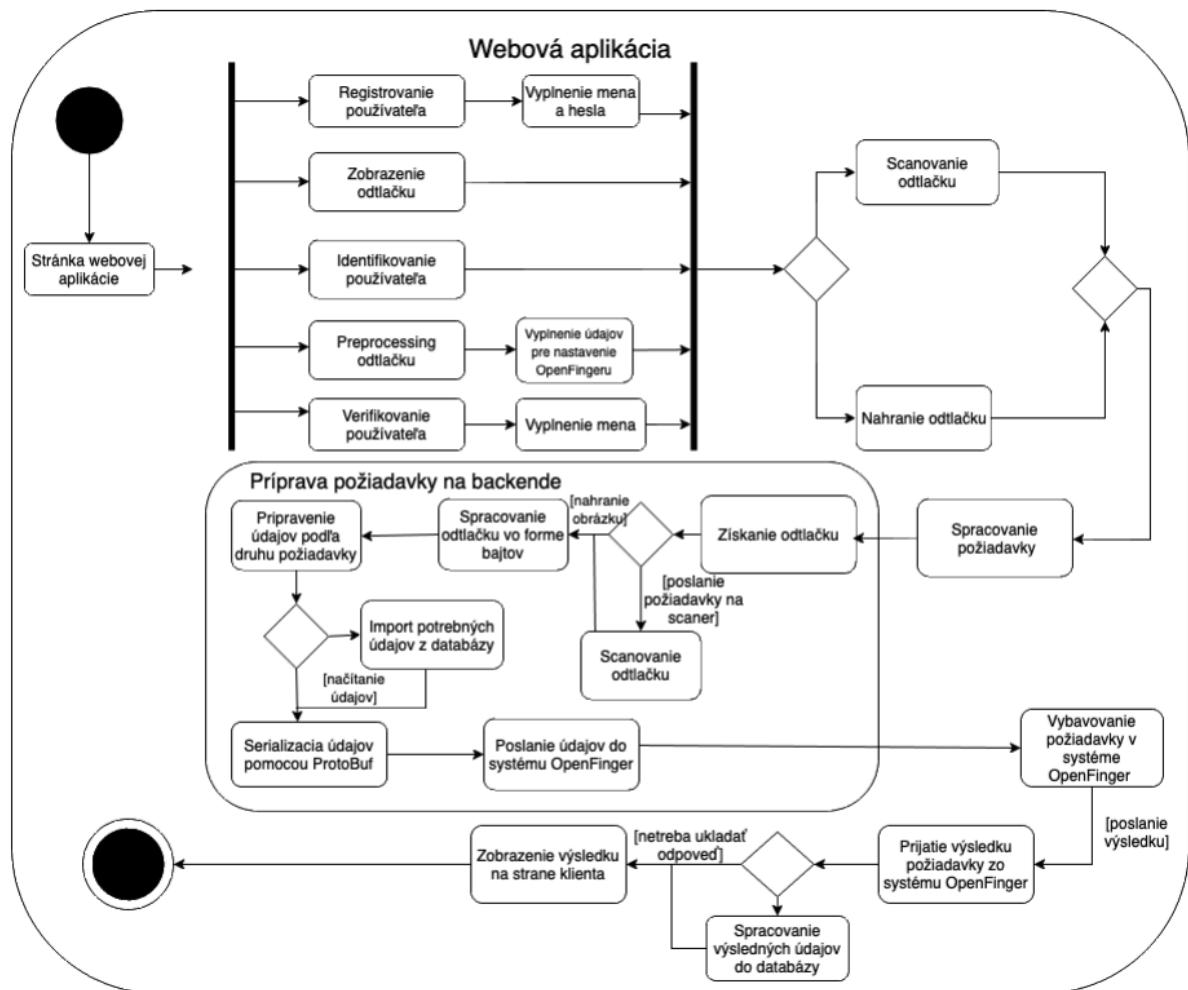
Klient zadá požiadavku, ktorú spracujeme a špecificky pre každý typ požiadavky vykonáme príslušné operácie. Rozlišujeme, či nám už prišiel odtlačok alebo nám prišla požiadavka na jeho zoskenovanie cez nás senzor Futronic FS-80H. Ďalej nasleduje rozlísiť typ požiadavky, s ktorým nám príde vždy iné množstvo a typ používateľských údajov. Na základe tejto požiadavky zadáme požiadavku na import dodatočných údajov z databázy alebo rovno pošleme odtlačok s priatými údajmi systému OpenFinger. Hned po odoslaní požiadavky musíme čakať na výsledok, ktorý nám od systému OpenFinger príde. Je vždy rôzneho typu podľa druhu požiadavky. Rovnako ako pri spracovaní požiadavky, tak aj teraz máme na výber náš spracovaný výsledok uložiť do databázy a poslať na frontend alebo len priamo poslať výsledok na frontend. Nižšie vidíme graf prípadov použitia pre backendovú časť aplikácie.



Obr. 9: Návrh backendovej časti aplikácie vo forme diagramu prípadov použitia

### 2.3.3 Diagram udalostí

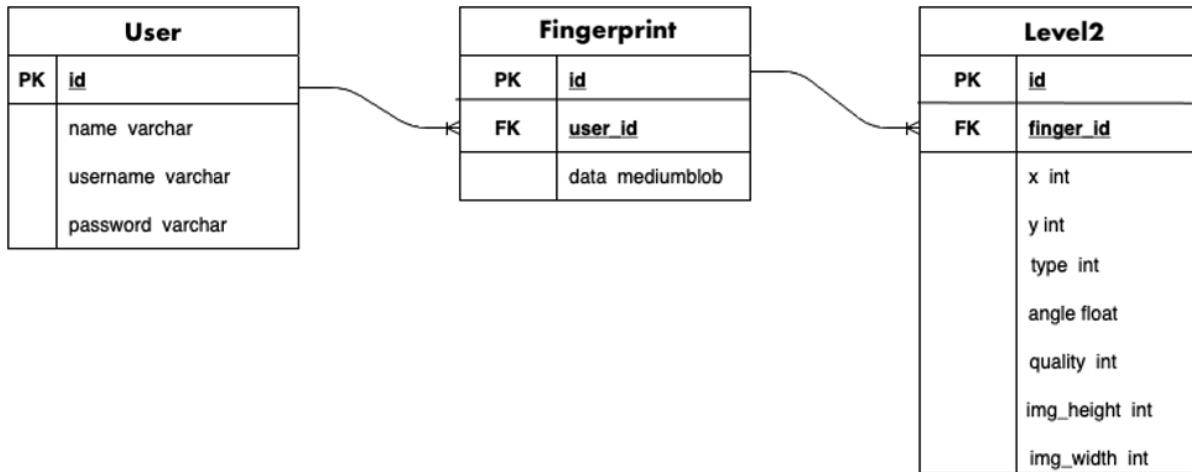
Používateľ a aj naša aplikácia musí byť pripravená a vedieť si poradiť s rôznymi typmi požiadaviek alebo odpovedí od systému OpenFinger. Ku každému typu požiadavky musíme pristupovať individuálne a rozlišovať aké údaje zoserializujeme a pošleme ako požiadavku a čo nám naopak príde ako odpoved, ktorú musíme deserializovať a ďalej s ňou vedieť narábať a zúžitkovat. Udalosti alebo aktivity ktoré aplikácia vykonáva si ukážeme na diagrame aktivít nižšie.



Obr. 10: Návrh aplikácie vo forme diagramu aktivít

### 2.3.4 Databáza

V aplikácii pracujeme s množstvom údajov, ktoré nám prídu od používateľa alebo od systému OpenFinger. Tieto údaje sú dôležité pretože ich s veľkou pravdepodobnosťou budeme ešte potrebovať, napríklad na verifikáciu alebo identifikáciu používateľa. Používateľ si taktiež môže zaregistrovať viac odtlačkov prstov, ktoré pri jeho následnom pokuse o verifikáciu alebo identifikáciu musíme vedieť použiť a správne poslať na vyhodnotenie. Kvôli veľkým dátam a našej snahe o rýchlu komunikáciu so serverom musíme každý odtlačok pri jeho registrácii poslať na spracovanie systému OpenFinger, ktorý nám vráti Level-2 znaky. Tieto špeciálne znaky si uložíme ku každému jednému odtlačku do databázy. Pri identifikácii porovnávame daný odtlačok so všetkými odtlačkami v databáze. Tento postup nám výrazne zmenší objem toku dát na server do systému OpenFinger. Každý používateľ môže mať v databáze N odtlačkov prstov a každý odtlačok môže mať N Level-2 znakov. Jeden Level-2 znak sa skladá z kombinácie štyroch číslic ktoré opisujú jeden špeciálny identifikačný znak daného odtlačku. Diagram nižšie nám popisuje databázu ktorú používame.

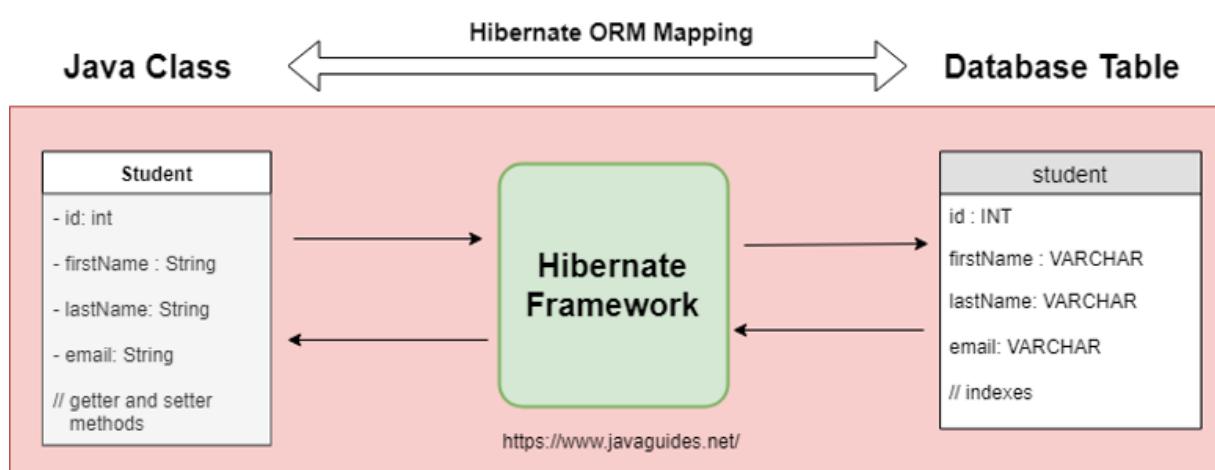


Obr. 11: Návrh aplikácie vo forme diagramu aktivít

### 2.3.5 Hibernate a JPA

**Hibernate** je framework, ktorý slúži na objektovo-relačné mapovanie Java objektov na tabuľky relačných databáz. Hibernate sám na pozadí vytvára SQL príkazy nad databázou a preto nemusíme písat SQL príkazy my. Ak chceme uložiť mapu objektov, napríklad používateľa, ktorý má odtlačok prsta, alebo aj viac objektov typu odtlačok prsta, tak nemusíme písat všetky SQL príkazy. Stačí, ak zavoláme jednoduchú metódu na uloženie objektu do databázy a Hibernate sa postará o zvyšok. Hibernate je jedna z najpopulárnejších implementácií JPA štandardov a je voľne dostupná (open-source).

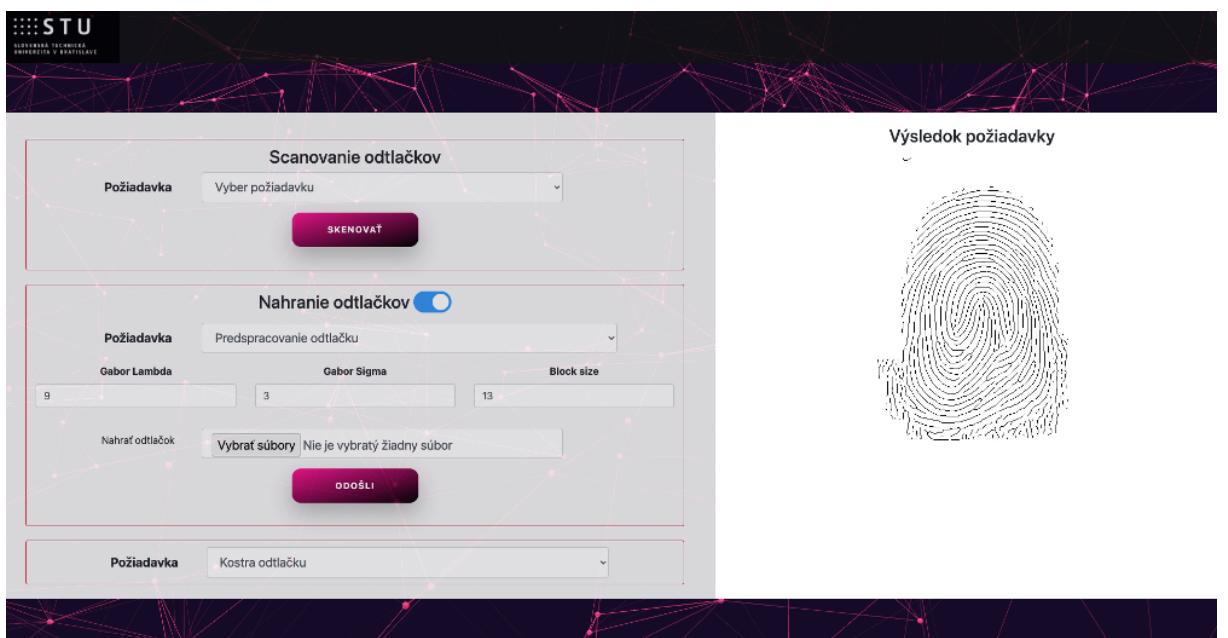
**JPA (Java Persistence API)** definuje len špecifikáciu, ktorá nám umožňuje mapovanie Java objektov na tabuľky v databáze a naopak. Je to rámc pre správu relačných údajov v Java aplikáciach, zatiaľ čo Hibernate je špecifická implementácia JPA. Na to, aby sme mohli používať JPA v skutočnej aplikácii, potrebujeme implementáciu JPA. V tejto aplikácii použijeme implementáciu, ktorú nám poskytuje framework Hibernate. Ak používame JPA štandardy, tak je v budúcnosti úplne jedno, akú implementáciu JPA budeme používať. Pri programovaní budeme používať JPA anotácie, ktoré pochádzajú z balíka `javax.persistence` [18].



Obr. 12: Diagram, ktorý nám ukazuje mapovanie medzi Java modelom `Student` a databázovou tabuľkou `student` [19].

### 3 Implementácia a dosiahnuté výsledky

V tejto kapitole je opísaná implementácia navrhnutého riešenia webovej aplikácie ako aj jej serverová časť, na ktorej prebieha komunikácia so systémom OpenFinger. Vo webovej aplikácii sme sa rozhodli implementovať všetko obsiahnuté v návrhu riešenia. Na vývoj webovej aplikácie bol použitý framework Spring a jazyk Java spolu s jazykmi na tvorbu klientskej časti ako Javascript, CSS a HTML. Na serializáciu a deserializáciu prenášaných údajov sme použili Protocol Buffers (Protobuf) knižnicu, ktorá je rýchla a umožňuje nám rýchlo a ľahko prenášať údaje v reálnom čase. Ako databázové riešenie sme využili H2 database [20].



Obr. 13: Používateľské rozhranie našej web aplikácie, ktoré zachytáva tvorbu požiadavky a zobrazenie výsledku predspracovania odtlačku prsta

#### 3.1 Spracovanie a odoslanie obrazu na server

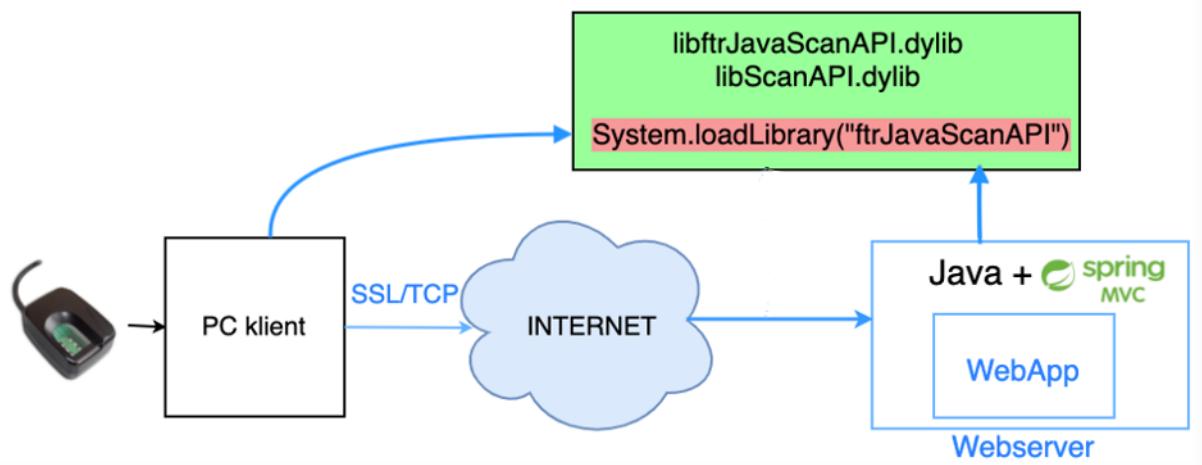
V tejto časti sa budeme bližšie venovať opisu ako vieme spracovať obraz odtlačku prsta a ako ho vieme pripraviť a spracovať tak, aby ho systém OpenFinger, ktorý sa nachádza na školskom serveri vedel rýchlo a jasne prečítať a vedel s ním podla našej požiadavky robiť potrebné operácie. Obraz vieme načítať zo snímača odtlačkov prstov Futronic FS-80H,

ktorý sme si museli prepojiť s našou aplikáciou tak, aby sme s ním vedeli komunikovať. Ďalšia možnosť ako nahrať odtlačok prsta je nahrať ho z disku.

### 3.1.1 Komunikácia so senzorom Futronic FS-80H

Pre úspešnú komunikáciu a prepojenie skenera Futronic FS-80H sme potrebovali knižnice, ktoré sme si museli importovať do nášho projektu a spustiť. Tieto knižnice a metódy na komunikáciu so senzorom boli však navrhnuté v jazyku C++. My však musíme vedieť k takýmto metódam pristúpiť a na to nám slúži rozhranie JNI (Java Native Interface). JNI umožňuje prístup k natívnym metódam vyvinutým v natívnych jazykoch, ktoré vedia vytvoriť zdieľané knižnice ako napr. jazyk C++ alebo C. Tieto natívne knižnice potom v Java načítame a metódy (funkcie) v nich obsiahnuté, môžeme štandardne volať. Na platforme Windows sú takéto knižnice ako .dll, na platforme Linux ako .so a na platforme MacOS sú známe ako .dylib a práve posledné spomínané musíme mať v projekte importované aj my [21]. K tejto knižnici potom vieme pristupovať cez, v našom prípade dopredu zadefinovaný, názov triedy, ktorá sa musí volať `Scanner` a musí sa nachádzať v zložke `ScanApiHelper`.

V tejto triede máme dopredu zadefinované niektoré metódy, cez ktoré vieme so senzorom vykonávať a nastavovať rôzne operácie od vydania príkazu na skenovanie až po napríklad nastavenie intenzity svetla pri skenovaní odtlačku. V tejto triede si samozrejme predtým ako chceme komunikovať so senzorom musíme túto knižnicu najprv načítať. Túto knižnicu načítame volaním nižšie tak, ako môžeme vidieť na grafe, predtým si však vo vývojovom prostredí nastavíme priečinok, kde má túto knižnicu hľadať. V našom prípade sa jedná o priečinok `dist`.



Obr. 14: Diagram, ktorý znázorňuje prepojenie odtlačku s aplikáciou

### 3.1.2 Spracovanie a získavanie dát zo senzora

Používateľ si najprv musí na klientskej strane zvoliť požiadavku na zoskenovanie odtlačku a vybrať si čo chce s týmto odtlačkom vykonať, o tom ale neskôr. Po stlačení tlačidla Skenovať spracujeme celý formulár cez `@PostMapping` metódu v tele ktorej máme uložené všetky potrebné údaje. Hned na začiatku metódy `scan` si vytvoríme novú inštanciu triedy `Scanner` po ktorej vzápäť pošleme požiadavku na zapnutie/otvorenie zariadenia. Hned ako sa nám podarí skener pripraviť a zapnúť, vyšleme požiadavku na zoskenovanie odtlačku aj s intenzitou svetla od 1 do 7, ktorá nám určuje jas zosnímaného odtlačku spolu s druhým parametrom, ktorý znázorňuje vopred pripravené bajtové pole o rozmere 320\*420 bajtov, ktorý nám je vopred známy a nemenný. Je to rozmer, v ktorom senzor sníma odtlačky. Celý proces poslania požiadavky a sken odtlačku môžeme vidieť nižšie:

```
Scanner scanner = new Scanner();
if(scanner.OpenDevice()) {
    byte[] data = new byte[320*480];
    scanner.GetImage2(3,data);
    ...
}
```

Ked už máme odtlačok nahraný do premennej `data` tak ďalej si musíme obrázok vždy invertovať aby bol pre systém OpenFinger lepšie čitateľný a boli nájdené naozaj len správne Level-2 znaky. Na invertovanie obrazu sme si vytvorili vlastnú funkciu, kde každý pixel invertujeme cez vzorec  $novýpixel = 255 - pôvodnýpixel$ . Po invertovaní môžeme tento obraz z premennej typu `BufferedImage` premeniť naspäť na bajty a pomocou `.copyFrom(data)` prenesieme celý obsah zoskenovaného odtlačku do premmenej typu `ByteString` čo je špeciálny Protobuf formát na prenos bajtov, čiže prenesieme si tento obraz vo formáte ktorý budeme ďalej používať na serializáciu a deserializáciu Protobuf požiadaviek.

### 3.1.3 Získavanie dát z nahraného obrázka

Rovnako ako pri možnosti zoskenovania odtlačku, aj v tomto prípade si musí používateľ vybrať možnosť ako a odkiaľ sa má nahrať alebo zoskenovať jeho odtlačok prsta, s ktorým chce vykonať požiadavky. Tentokrát sa budeme venovať druhému možnému variantu a to je nahranie odtlačku z úložiska vo forme obrázku. Po stlačení tlačidla `Odošli` spracujeme celý formulár cez `@PostMapping` metódu, v tele ktorej máme uložené všetky potrebné údaje ako to bolo pri požiadavke na zoskenovanie prsta skenerom, ale v tomto prípade nám

ešte príde nahraný súbor. Súbor označovaný pod menom `file` nemusí byť len vo formáte grayscale ako tomu bolo pri obrázku zo skeneru, ktorý nám po zosnímaní odtlačku vrátil obrázok typu `.bmp` ktorý bol ľahšie čitateľný a konvertovateľný do poľa bajtov. Na začiatku si cez metódu `file.getInputStream()` nahráme celý obsah do premennej `img` ktorá je typu `BufferedImage` s ktorou vieme dalej pracovať. Tento obrázok si avšak rovnako ako zo skenera, tak aj teraz musíme invertovať nech je lepšie čitateľný pre systém OpenFinger. Invertovaný obrázok uložený v premmenej `img` využijeme na vytvorenie ďalšej práznej premennej `image`, taktiež typu `BufferedImage` ktorej veľkosť si určíme podľa premennej `img`. Po tom ako máme prvú premennú typu `BufferedImage` naplnenú surovými bajtami a druhú premennú pripravenú, môžeme si premennú `img` prekopírovať do `image` pomocou metódy `.createGraphics().drawImage(...)`, v ktorej sa nachádzajú dôležité parametre ktoré nám zaručujú správny prenos dát medzi dvomi farebnými modelmi. Teraz už máme `BufferedImage` správneho typu, ktorého obsah musíme uložiť do Poľa bajtov `array` z ktorého si ho už rovnako ako pri skenovaní, prenesieme do premennej typu `ByteString`

```
BufferedImage imag = ImageIO.read(file.getInputStream());
BufferedImage img = convertingService.invertimage(imag);
BufferedImage image = new
BufferedImage(img.getWidth(),img.getHeight(),BufferedImage.TYPE_BYTE_GRAY);
imag.createGraphics().drawImage(image, 0, 0, Color.WHITE, null);
byte[] img_array=((DataBufferByte) imag.getRaster().getDataBuffer()).getData();
ByteString bat = ByteString.copyFrom(img_array);
```

## 3.2 Serverová časť

V tejto kapitole sa budeme venovať implementácií toho, čo nám beží na pozadí aplikácie bezprostredne po odoslaní formuláru a spracovaní odtlačku prsta, či už vo forme zoskenovaného alebo nahratého odtlačku. Pretože máme viacero požiadaviek, ktoré si môže používateľ zvoliť, tak ku každej musíme pristupovať jedinečne a vykonávať špecifické volania. Pri každej požiadavke máme iné volania na databázu a posielame vždy inú zserializovanú správu, ktorá má v tele rôzne údaje. To s akými údajmi pracujeme a kde ich ukladáme, poprípade ako ich zobrazujeme na strane klienta sa budeme venovať nižšie.

### 3.2.1 Typy požiadaviek

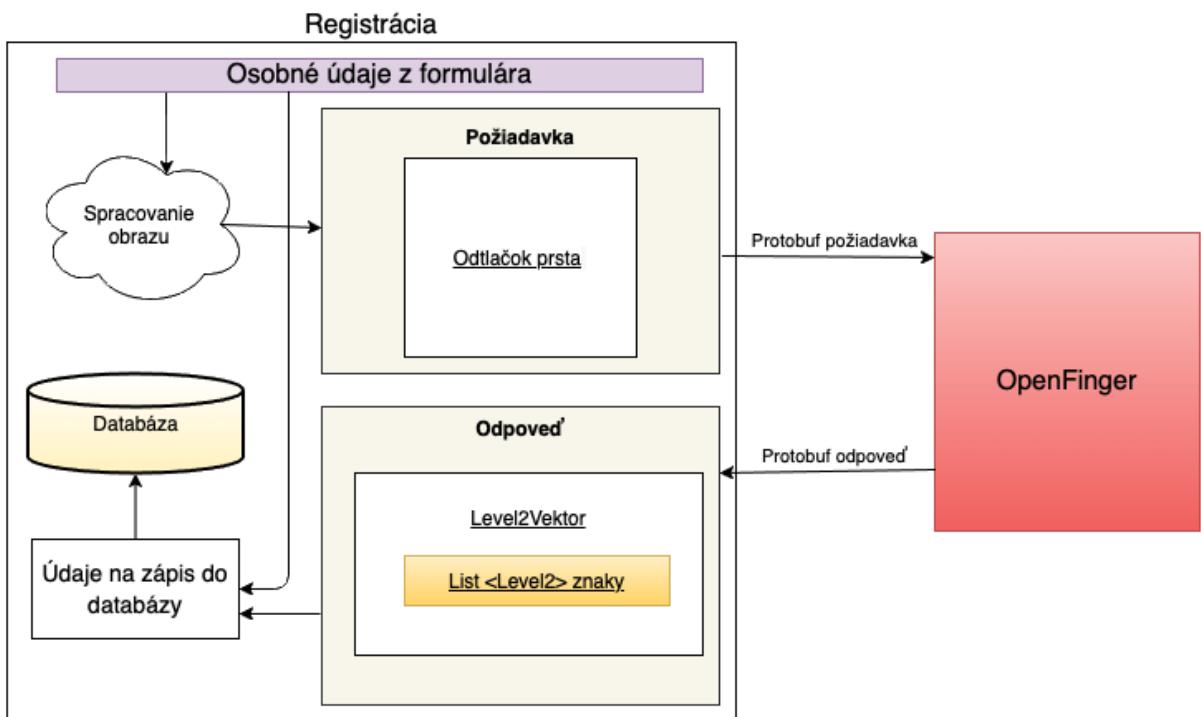
V aplikácii si môže používateľ zvoliť jednu z 5 požiadaviek. Celú schému môžeme vidieť na obrázku č. 2.3 v návrhu riešenia. Niektoré z požiadaviek komunikujú s databázou, kedy potrebujú poslať zo spracovaným odtlačkom aj odtlačky, ktoré požadujeme porovnať a vyhodnotiť prípadnú zhodu. Ďalšie požiadavky nepotrebuju na skompletizovanie požiadavky údaje z databázy a stačí im len samotný obrázok a údaje ktoré povedia serveru aký proces má vykonať s odtlačkom. Napríklad ho môžeme chcieť predspracovať alebo v nám nájst všetky špecifické Level-2 znaky, ktoré nám server vráti.

#### Registrácia

Pri procese registrácie potrebujeme od používateľa jeho meno a používateľské meno, pod ktorým bude chcieť vystupovať a pri prípadnej budúcej verifikácii sa na základe tohto mena môže verifikovať. Tieto parametre spolu si odložíme na zápis do databázy. Najskôr ale musíme poslať registračnú požiadavku systému OpenFinger. Táto regisračná požiadavka obsahuje len odtlačok prsta, ktorý chceme spracovať a typ, že sa jedná práve o regisračnú požiadavku. Server ju spracuje a vráti nám požiadavku typu **Level2Vector**, v ktorej je N množstvo Level-2 znakov. Počet znakov závisí od množstva nájdených špecifických znakov na danom odtlačku prsta. Po spracovaní požiadavky si tieto údaje spracujeme spolu s menom a priezviskom a uložíme do databázy. Používateľovi priradíme odtlačok ktorý bude mať N Level-2 znakov. Daný používateľ môže mať v databáze samozrejme aj viac svojich odtlačkov.

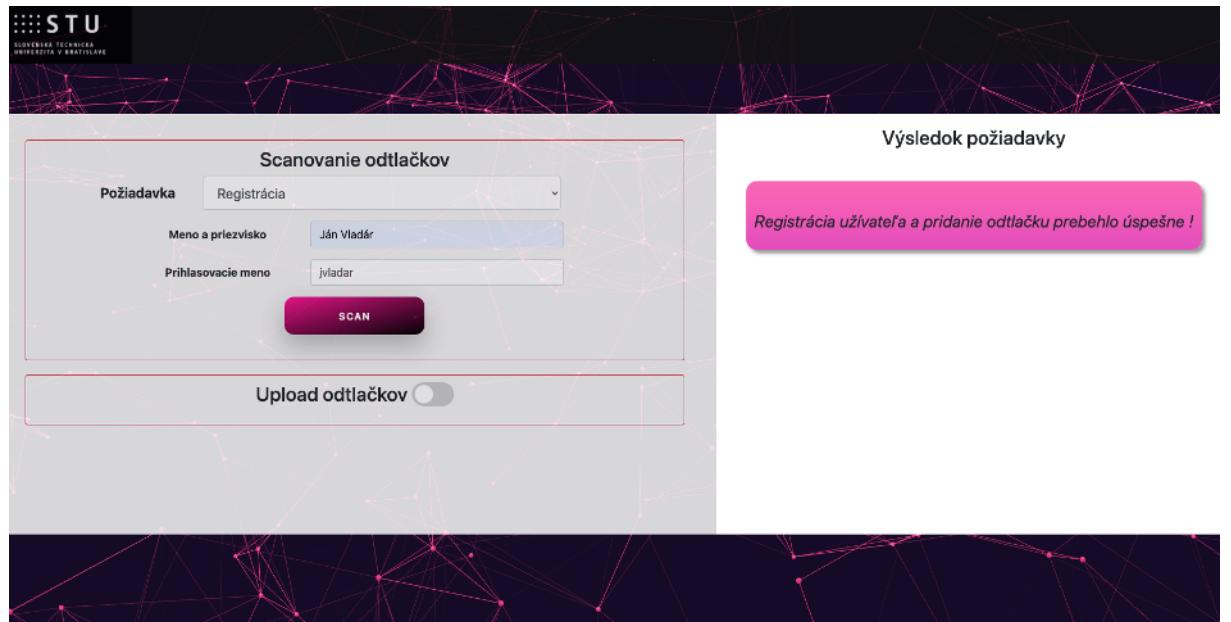
<b>Požiadavka</b>	<b>Odpoveď</b>
<pre>syntax = "proto3";  import "Fingerprint.proto"; message ExtractionRequest{     Fingerprint fingerprint = 1; } }</pre>	<pre>syntax = "proto3";  import "Level2Vector.proto"; message ExtractionResponse {     Level2Vector level2vector = 1; } }</pre>

Obr. 15: Obsah .proto súborov pre regisračnú požiadavku a odpoved.



Obr. 16: Diagram regisračnej požiadavky a odpovede od servera s následnými akciami

Na obrázku nižšie môžeme vidieť ako vyzerá úspešná registrácia na strane klienta.



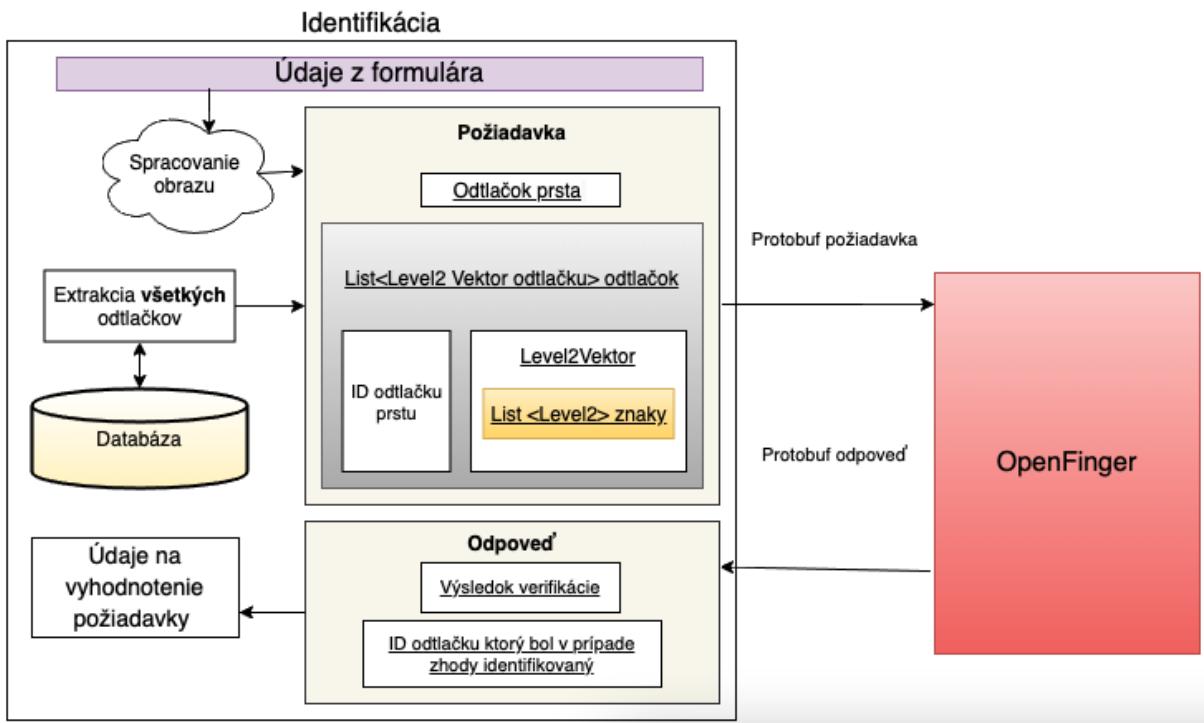
Obr. 17: Ukážka úspešnej registrácie používateľa

## Identifikácia

Pri procese identifikácie nepotrebujeme od používateľa žiadne dátu pretože jeho identitu chceme porovnať voči všetkým používateľom v systéme. Jediné, čo potrebujeme je jeho odtlačok prsta, ktorý spracujeme. Teraz vidíme prečo si ukladáme pri registrácii len údaje typu `Level2Vector` a nie celé odtlačky, je to kvôli veľkosti súborov. Keby chceme pri procese identifikácie posielat niekoľko desiatok odtlačkov, bolo by to veľmi neefektívne. Dôležité je správne vyextrahovať z databázy všetky údaje typu `Level2Vector` tak, aby sme späťne vedeli ktorému používateľovi tento odtlačok patril. Z toho dôvodu si musíme ku každému údaju typu `Level2Vector` teraz pridať aj ID odtlačku pretože takýchto údajov typu `Level2Vector` bude viac. Takúto požiadavku pošleme na server, ktorý kontroluje zhodu s každým jedným odtlačkom a ak nájde zhodu tak automaticky nám odošle odpoveď, v ktorej sa nachádza výsledok vo forme True/False a ID odtlačku prsta, ktoré si mi potom priradíme k danému používateľovi, ktorému tento odtlačok patrí.

Požiadavka	Odpoved'
<pre>syntax = "proto3"; import "Fingerprint.proto"; import "Level2Vector.proto";  message Level2VectorOffFingerprint{     int32 fingerprint_id = 1;     Level2Vector level2vector = 2; }  message IdentificationRequest{     Fingerprint fingerprint = 1;     repeated Level2VectorOffFingerprint vectors = 2; }</pre>	<pre>syntax = "proto3";  message IdentificationResponse{     bool success = 1;     int32 fingerprint_id = 2; }</pre>

Obr. 18: Obsah .proto súborov pre identifikačnú požiadavku a odpoved'.



Obr. 19: Diagram identifikačnej požiadavky a odpovede od servera s následnými akciami

## Verifikácia

Pri procese verifikácie potrebujeme od používateľa jeho používateľské meno, ktoré si zadal pri registrácii. Na základe tohto údaju si z databázy vyextrahujeme všetky údaje typu **Level2Vector**, ktoré má používateľ uložené v databáze. Každý údaj typu **Level2Vector** predstavuje jeden odtlačok prsta, ktorý má N Level-2 znakov. Verifikačná požiadavka obsahuje spracovaný odtlačok prsta a vyššie spomínané údaje typu **Level2Vector**. Server nám odpovedá výsledkom verifikácie, či používateľ bol alebo nebol verifikovaný voči odtlačkom z databázy ktoré sa nachádzali pod jeho používateľským menom a ako ďalší parameter je skôr úspešnosti verifikácie, ktoré nám určuje aká veľká zhoda bola zaznamenaná.

```

Požiadavka
syntax = "proto3";
import "Fingerprint.proto";
import "Level2Vector.proto";

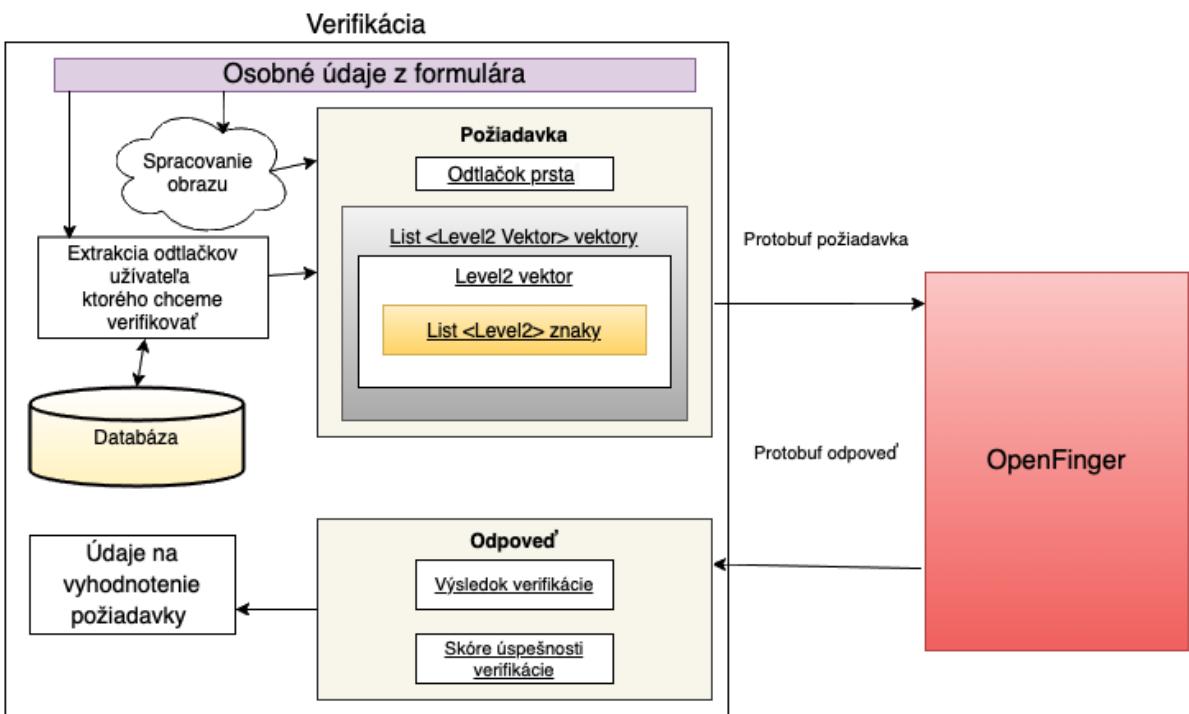
message VerificationRequest{
    Fingerprint fingerprint = 1;
    repeated Level2Vector level2vectors = 2;
}

Odpoveď
syntax = "proto3";

message VerificationResponse{
    bool result = 1;
    float score = 2;
}

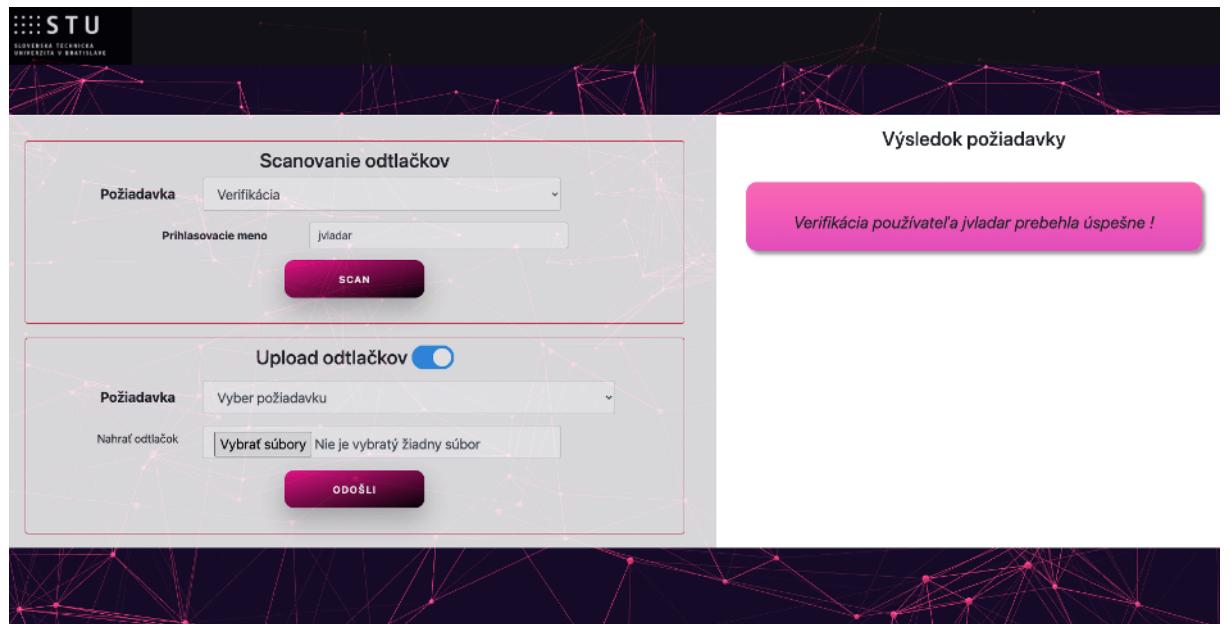
```

Obr. 20: Obsah .proto súborov pre verifikačnú požiadavku a odpoved.



Obr. 21: Diagram verifikačnej požiadavky a odpovede od servera s následnými akciami

Na obrázku nižšie môžeme vidieť úspešný výsledok verifikácie používateľa.



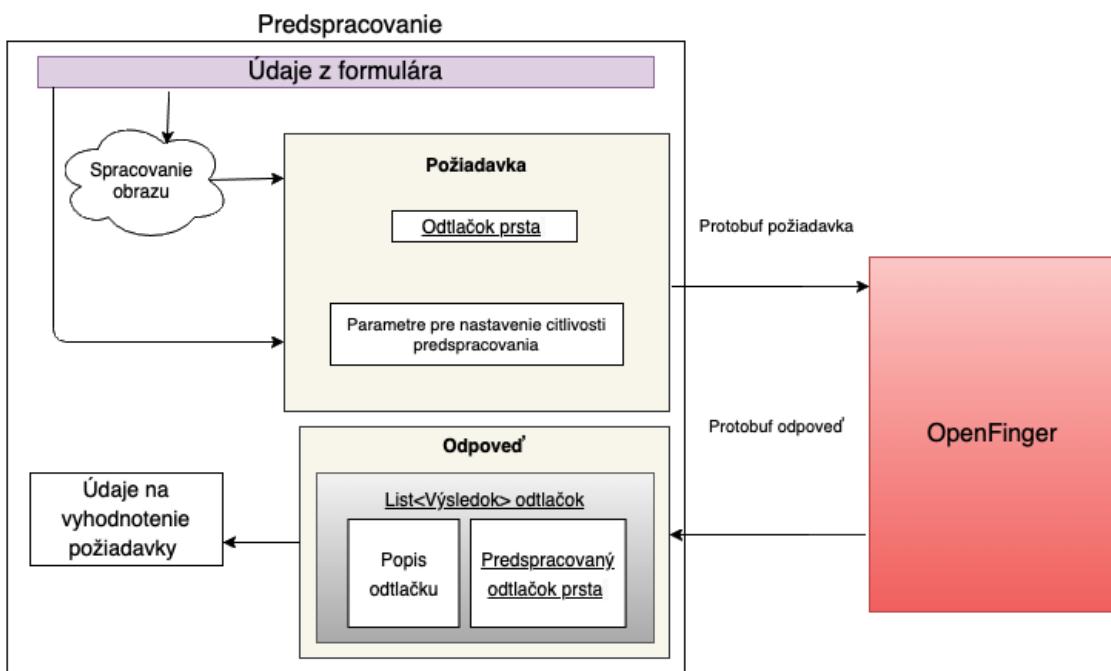
Obr. 22: Ukážka úspešnej verifikácie používateľa

## Predspracovanie

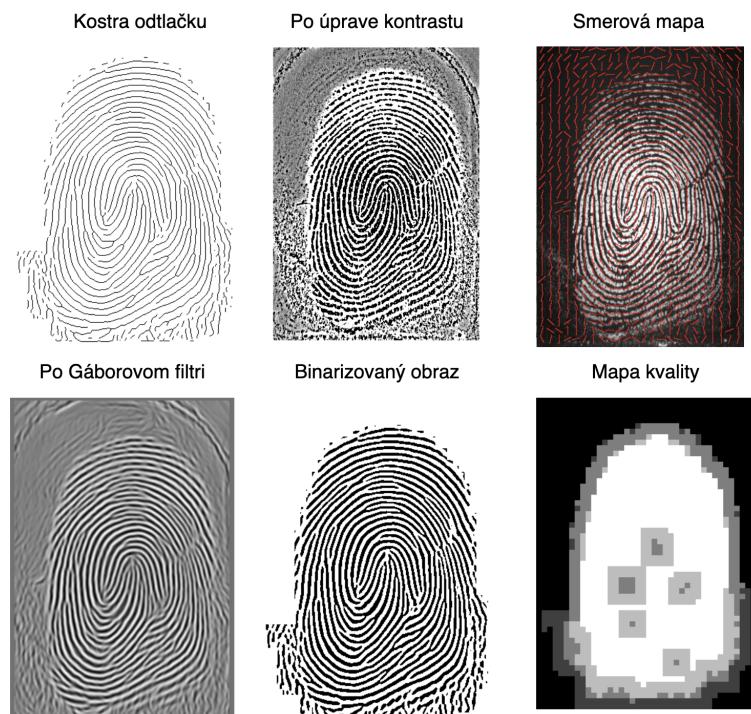
Pri procese predspracovania odtlačku prsta potrebujeme od používateľa len vyplniť údaje, ktoré určujú presnosť a citlivosť s akou má systém OpenFinger vykonať predspracovanie odtlačku. Tieto údaje sú už automaticky prednastavené, ale je možné ich meniť podľa potreby. Požiadavka predspracovania obsahuje odtlačok prsta a vyššie spomínané údaje. Ako výsledok nám príde 6 rôznych odtlačkov prstov a každý bude vyzeráť inak – na indexe 0 nám príde mapa kvality obrazu, na indexe 1 – obraz po úprave kontrastu, na indexe 2 – smerová mapa, na indexe 3 – obraz po prefiltrovaní Gáborovým filtrom, na indexe 4 – binarizovaný obraz, na indexe 5 – kostra odtlačku. Na indexe 5 je hlavný výsledok a je to odtlačok, kde budú zvýraznené obrazce papilárnych línii a budú na ňom lepšie vidieť rozpoznávacie črty odtlačku. Takéto odtlačky si následne musíme správne z Protobuf triedy vyextrahovať podobným procesom, ako sme opísali v kapitole 3.1.3 a následne zobrazit na strane klienta aj s menu, kde si môže klient vybrať, ktorý z odtlačkov chce zobraziť. Nižšie môžeme vidieť ako vyzerá odtlačok prsta po predspracovaní a aké všetky výsledky vieme zobraziť používateľovi na klientskej strane.

Požiadavka	Odpoveď
<pre>syntax = "proto3"; import "Fingerprint.proto";  message PreprocessingParams{     // block size for orientation map     // and Gabor filter     int32 block_size = 1;     // lambda parameter for Gabor filter     // (defines average ridge width)     double gabor_lambda = 2;     // sigma parameter for Gabor filter     // (defines filter strength)     double gabor_sigma = 3; }  message PreprocessingRequest {     Fingerprint fingerprint = 1;     PreprocessingParams params = 2; }</pre>	<pre>syntax = "proto3"; import "Fingerprint.proto";  message PreprocessingResult{     string info = 1;     Fingerprint fingerprint = 2; }  message PreprocessingResponse {     repeated PreprocessingResult results = 1; }</pre>

Obr. 23: Obsah .proto súborov pre požiadavku a odpoved predspracovania.



Obr. 24: Diagram požiadavky predspracovania a odpovede od servera s následnými akciami



Obr. 25: Všetky druhy výsledkov predspracovania, ktoré si môže používateľ nechat zobraziť na strane klienta

Predspracovaný odtlačok, ktorý máme vo forme bajtov si musíme vedieť nejako zobraziť na strane klienta. Na toto použijeme premennú binárneho dátového formátu typu Base64, ktorú pošleme na klientskú stranu aplikácie. Takúto premennú si potom vieme v jazyku HTML premeniť na obrázok.

The diagram illustrates the process of displaying a processed printout (Obrázok) as an image in HTML. On the left, a box labeled "Obrázok" contains a Java code snippet. An arrow points from this box to a second box labeled "HTML", which contains the resulting HTML code.

**Java**

```
BufferedImage image1 = //namapovaný obrázok;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ImageIO.write(image1, formatName: "bmp", baos);
byte[] bytes = baos.toByteArray();

byte[] encodeBase64 = Base64.getEncoder().encode(bytes);
String s = new String(encodeBase64, charsetName: "UTF-8");
model.addAttribute( s: "fotka",s);
```

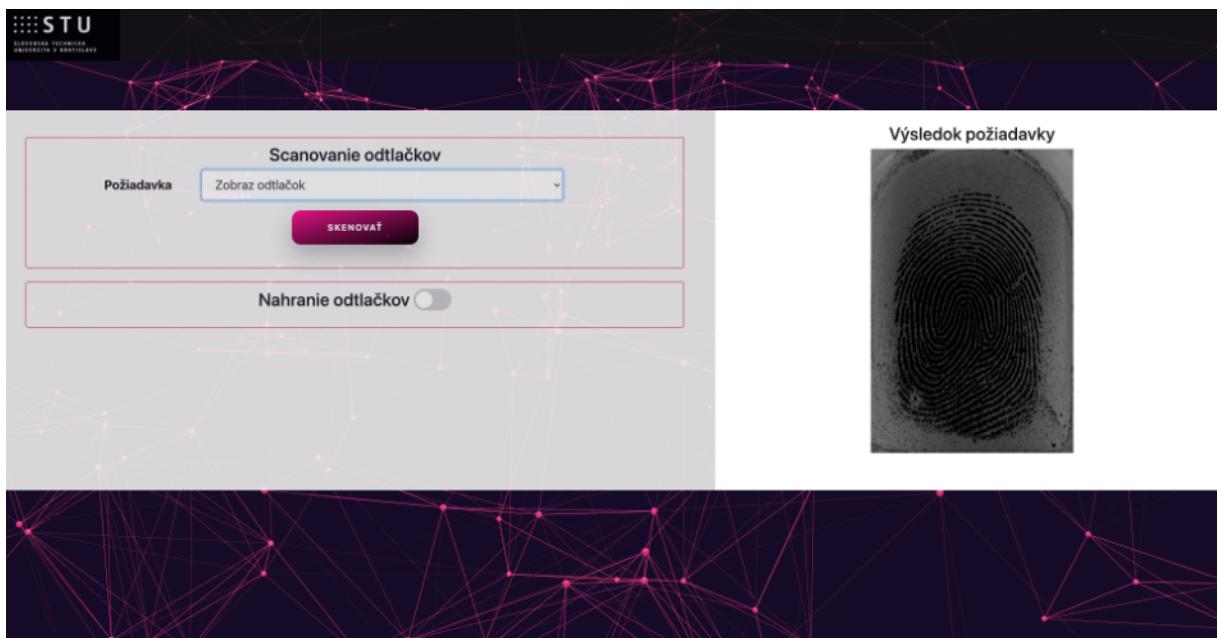
**HTML**

```
<h3 >Výsledok požiadavky</h3>
<td></td>
```

Obr. 26: Ukážka kódu, ktorý ukazuje ako zobrazíme obrázok z poľa bajtov v jazyku Java na normálny obrázok v jazyku HTML

## Zobrazenie odtlačku

Posledný druh požiadavky je zobrazenie odtlačku prstu alebo inak povedané zobrazenie spracovaného obrazu. V tejto požiadavke nekomunikujeme s databázou a ani so serverom, našou úlohou je premeniť dátu z obrázku alebo skenera podobným spôsobom, ako to bolo vyššie pri predspracovaní. Spracovaný obraz si uložíme a následne zobrazíme na strane klienta rovnakým štýlom ako pri predspracovaní.



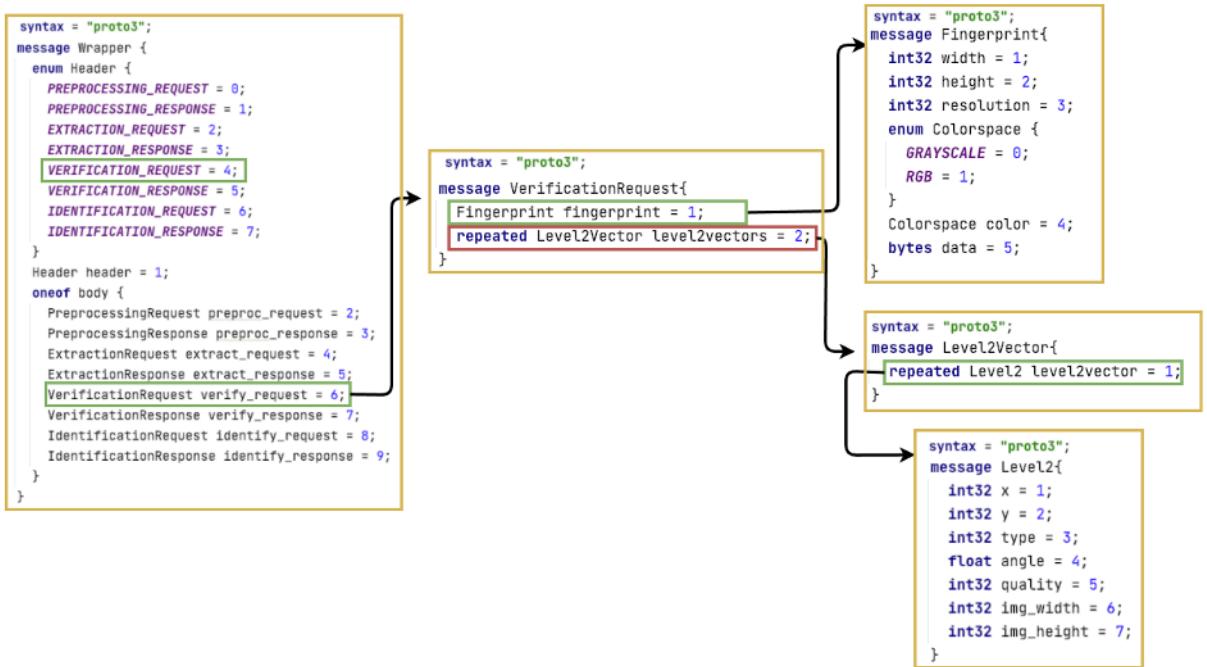
Obr. 27: Ukážka požiadavky na zobrazenie odtlačku prsta

### 3.2.2 Serializácia a deserializácia pomocou Google Protocol Buffers

Z dôvodu neustálej komunikácie medzi našou Java aplikáciou a systémom OpenFinger, ktorý je v jazyku C++, musíme dbať na rýchlosť prenášaných dát, ktoré zároveň budú ľahko čitateľné pre obidve strany. Je to aj z dôvodu, keď prenášame pomerne veľa údajov, ktoré majú v sebe ďalšie a ďalšie údaje ako je to napríklad v prípade Identifikácie, o ktorej sme písali vyššie. Z každej Protobuf triedy si vieme automaticky vygenerovať Java triedu, ktorá obsahuje všetky potrebné metódy na ukladanie alebo volanie jednotlivých druhov premenných.

**Serializácia údajov** Ako sme už spomenuli, každá požiadavka je špecifická v dátach, ktoré potrebuje spracovať a tým pádom aj odoslať na server. Z toho dôvodu potrebuje 4 rozličné Protobuf triedy, do ktorých vieme uložiť celú našu požiadavku. Avšak potrebujeme aj triedy na uloženie údajov, ktoré vyextrahujeme z databázy pretože systém OpenFinger, ktorý bude chcieť prečítať túto správu, musí vedieť prečítať každú časť tejto správy a vedieť s ňou pracovať. Musíme si ale uvedomiť, že nám nestačí posielat len jednu zo 4 rôznych požiadaviek, pretože server, ktorý počúva na nejakej ip adrese a porte nevie, ktorá požiadavka mu kedy príde, server len vie, že mu nejaká požiadavka prišla. Preto sme si vytvorili Protobuf triedu s názvom **Wrapper**, ktorej prvý parameter je enumerátor, ktorý znázorňuje aká požiadavka sa prenáša. Takto vieme docieliť, že server bude vedieť

vždy požiadavku posielanú cez soketovú komunikáciu prečítať, pretože vždy bude očakávať požiadavku typu `Wrapper`. Na nasledujúcom grafe môžeme vidieť syntax Protobuf podľa štandardu `proto3`. Na ukážku sme vybrali ako vyzerá štruktúra údajov prenášaných vo verifikačnej požiadavke a ako sa všetky údaje na seba nabaľujú. Naľavo vidíme všeobecnú triedu `Wrapper`, ktorá je ďalej poprepájaná.

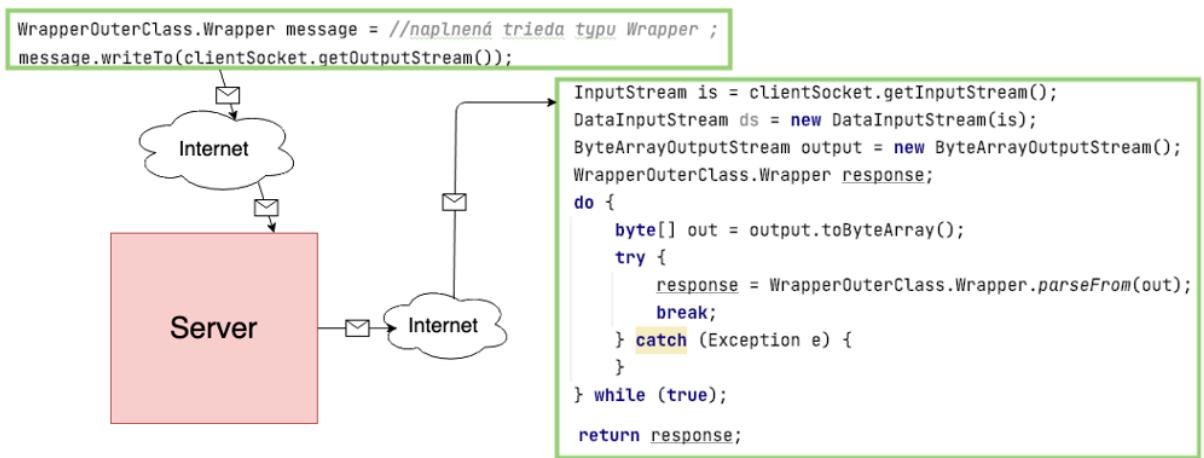


Obr. 28: Diagram Protobuf triedy `Wrapper` aj s jej ďalšími zapúzdrenými údajmi pri verifikačnej požiadavke

**Deserializácia údajov** Postup pri deserializácii je podobný ako sme opísali vyšie. Po odoslaní triedy `Wrapper` môžeme vzápäť očakávať odpoveď od servera s výsledkom požiadavky. Hneď po odoslaní naša aplikácia počúva na sokete a postupne zbiera všetky potrebné dátá, ktoré namapuje do triedy `Wrapper`, z ktorej vieme potom jednotlivé údaje spracovať v našej Java aplikácii. Napríklad pri predspracovaní si vieme z poľa bajtov ktoré je uložené v triede `Fingerprint` namapovať všetky potrebné údaje a zobraziť výsledný obraz odtlačku. Keď už máme úspešne prečítané údaje vložené v triede `Wrapper`, tak postupujeme rovnako ako v kapitole 3.1.3. Ak nám prídu výsledky v inej forme, tieto dátá si vieme podľa typu požiadavky okamžite uložiť do databázy alebo zobraziť na strane klienta.

### 3.2.3 Komunikácia so systémom OpenFinger

Celá vyššie spomínaná komunikácia funguje cez soketovú komunikáciu a cez server, ktorý počúva na IP adresu a portu. Ako sme už spomínali v sekcií vyššie, komunikujeme na základe triedy `Wrapper`, ktorá obsahuje údaj o tom, ktorú požiadavku zo sebou nesie a ďalší, hlavný údaj tvorí samotná požiadavka. Dôvod je taký, že server, ktorý počúva na vyššie spomínanom portu nevie aká požiadavka mu príde. Takáto komunikácia musí začať zadefinovaním premennej typu `Socket` ktorá v sebe nesie informácie o IP adrese a portu na ktorý má smerovať. Ďalšia úloha je spraviť komplet celú serializáciu triedy `Wrapper`, ktorú na záver zapíšeme do premennej typu `OutputStream`, ktorú tvorí premenná typu `Socket` so zavolením metódy na odoslanie tejto správy na miesto, ktoré sme jej zadefinovali. Naopak pri čítaní dát zo servera si zavoláme premennú typu `Socket` s metódou na čítanie z daného miesta (IP adresa a port). Tieto dáta si ukladáme do premennej typu `InputStream` z ktorej čítame až dovtedy, pokým sa nám úspešne nenamapuje premenná typu `Wrapper`, s ktorou už vieme ďalej pracovať. Na grafe nižšie môžeme vidieť zjednodušený model, akým už vopred zoserializovanú triedu typu `Wrapper` odosielame a naopak aj prijíname a mapujeme.



Obr. 29: Model soketovej komunikácie medzi aplikáciou a serverom aj s čítaním a mapovaním odpovede do triedy `Wrapper`

## 3.3 Sumarizácia dosiahnutých výsledkov

Všetky požiadavky a ciele, ktoré sme si predom určili sa nám podarilo aj splniť. Výsledkom tejto bakalárskej prace je webová aplikácia, ktorá je úspešne prepojená so senzorom Futronic FS-80H a systémom OpenFinger, s ktorým vie komunikovať a vymieňať si požiadavky typu Google Protocol Buffers. Aplikáciu sme testovali na rôznych odtlačkoch

prstov, ktoré sme si nasnímali alebo nahrali z úložiska. Následne sme testovali všetky typy požiadaviek a kontrolovali správnosť výsledkov. Aplikácia vie taktiež reagovať a poslať používateľovi upozornenie keby zadal prázdne alebo nesprávne prihlásovacie meno alebo registračné údaje. Ďalej si zrekapitujeme všetky dôležité funkcionality.

- Implementácia úspešnej komunikácie senzora Futronic FS-80H s našou aplikáciou. Senzor pripojený cez USB port zoskenuje priložený prst, hned ako klient odošle požiadavku na zoskenovanie odtlačku prsta.
- Možnosť nahrania odtlačku prsta z disku zariadenia, na ktorom beží aplikácia s následným výberom a vyplnením požiadavky.
- Úspešne vieme spracovať rôzne typy obrazov. Vieme spracovať a serializovať grayscale obrazy a rovnako aj rôzne druhy obrazov typu RGB.
- Obrazy rôznych typov si vieme z poľa bajtov následne deserializovať a upraviť do podoby, aby sme ich vedeli zobraziť na strane klienta
- Používateľ sa vie svojim odtlačkom zaregistrovať. Po zadaní požiadavky aj s jeho osobnými údajmi pošleme zoserializovanú požiadavku s odtlačkom systému OpenFinger, ktorý nám vráti N Level-2 znakov, ktoré si uložíme do databázy a priradíme ich k danému používateľovi.
- Používateľ si môže do systému pridať aj ďalšie odtlačky prstov, na základe ktorých by sa mohol chcieť v budúcnosti verifikovať alebo identifikovať.
- Používateľa vieme na základe jeho odtlačku a zadaného mena verifikovať. Na základe používateľského mena si vyberieme z databázy jeho všetky odtlačky a aj s aktuálne spracovaným odtlačkom tieto dáta zoserializujeme do verifikačnej požiadavky, ktorú pošleme systému OpenFinger. Ako odpoveď nám príde výsledok o úspešnosti verifikácie, ktorý vyhodnotíme.
- Používateľa vieme na základe jeho odtlačku prsta aj identifikovať a zistíť, či sa nachádza alebo nenachádza v systéme. Najskôr si musíme vybrať z databázy všetky odtlačky a s aktuálne spracovaným odtlačkom ich zoserializovať do požiadavky, ktorú posielame systému OpenFinger. Ako odpoveď nám príde výsledok identifikácie vo forme identifikačného čísla odtlačku, s ktorým bola nájdená zhoda, ak sme zhodu nenašli, vráti sa nám -1.

- Používateľ si môže zvoliť aj možnosť predspracovania obrazu. V tomto prípade si zoserializujeme len jeho odtlačok ktorý pošleme systému OpenFinger. Ako výsledok nám príde 6 rôznych obrazov, ktoré znázorňujú rôzne fázy predspracovania. Tieto obrazy zobrazíme na strane klienta, ktorý si ich môže podľa potreby pozerať.
- Posledná požiadavka, ktorú vieme obslúžiť je zobrazenie odlačku prstu, kedy si aktuálne spracovaný odtlačok invertujeme a zobrazíme na strane klienta.
- Aplikácia vie v reálnom čase vyhodnocovať, ukladať do databázy a zobrazať odpovede na požiadavky na strane klienta, ktoré sú vždy rôzneho typu na základe požiadavky.
- Naša aplikácia vie komunikovať s minimálnou spotrebou dát, ktorú sme zabezpečili posielaním zserializovaných tried typu Google Protocol Buffers. Minimalizáciu veľkosti posielaných dát a následné zvýšenie rýchlosťi komunikácie sme zabezpečili aj vďaka regisitračnému procesu každého odtlačku, ktorý nám vracia len Level-2 znaky, ktoré reprezentujú v databáze jeden odtlačok. Tento spôsob nám pri verifikácii alebo identifikácii výrazne zmenší veľkosť potrebných dát, ktoré musíme posielat.

### Verejný Github repozitár so zdrojovými kódmi

Všetky relevantné zdrojové kódy webovej aplikácie sú dostupné v nasledujúcom GitHub repozitári

- Experimetálna webová aplikácia pre biometrický autentifikačný server  
<https://github.com/jvladar/BiometricSpringApp>

# Záver

V tejto bakalárskej práci sme sa zamerali na oblasť automatizovanej biometrie. Táto práca prináša návrh, implementáciu a testovanie webovej aplikácie, ktorá komunikuje s biometrickým systémom na rozpoznávanie odtlačkov prstov s názvom OpenFinger. Webová aplikácia je prepojená so senzorom Futronic FS-80H, z ktorého vie spracovať odtlačky prstov a na základe používateľských požiadaviek šifrovane komunikovať so školským biometrickým serverom. Komunikáciu senzora s aplikáciou zabezpečujú špeciálne knižnice na USB komunikáciu medzi senzorom Futronic a Java webovou aplikáciou. Webová aplikácia je vyvíjaná v jazyku Java a frameworku Spring, v jazyku HTML, CSS a Javascript. Odtlačky prstov a požiadavky sa serializujú pomocou Google Protocol Buffers, cez ktorý aplikácia šifrovane cez SSL/TCP protokol komunikuje so serverom na základe posielania zserializovaných požiadaviek a odpovedí. Používateľ si môže v aplikácii zvoliť, či sa chce pomocou aplikácie zaregistrovať, verifikovať, identifikovať, nechať si svoj odtlačok predspracovať alebo len čisto zobraziť, taktiež si môže aj pridať ďalší odtlačok prsta do databázy. Predtým si musí používateľ zvolať odkiaľ chce svoj odtlačok nahrať do aplikácie. Na výber má z možnosti zoskenovania skenerom alebo nahrania z úložiska vo forme obrázku. Po zvolení požiadavky s potrebnými vyplnenými údajmi a úspešným spracovaním odtlačku si aplikácia z databázy vyextrahuje ďalšie potrebné údaje a celú požiadavku odošle na server. Pri registrácii používateľa posielame na server vždy len jeho odtlačok prsta a ako odpoveď od servera nám príde niekoľko desiatok špecifických Level-2 znakov, ktoré v aplikácii spracujeme a uložíme k danému používateľovi do databázy. Pri požiadavke predspracovania posielame taktiež len aktuálne zosnímaný odtlačok, avšak tentokrát nám odpovedí 6 špecifických obrázkov, sú to výsledky rôznych druhov predspracovania, na ktorých môžeme vidieť viaceré charakteristické črty a vlastnosti odtlačku. Pri ďalších požiadavkách posielame na server kombinácie už spracovaných odtlačkov vo forme Level-2 znakov s údajmi o konkrétnej požiadavke a jej špecifikácii, ktoré server spracuje a vyhodnotí zhodu. Pri verifikácii porovnáva spracované odtlačky daného používateľa s jeho aktuálne nasnímaným a pri identifikácii porovnáva všetky odtlačky z databázy s odtlačkom aktuálne nasnímaným. OpenFinger má knižnicu na automatizovanú extrakciu charakteristických znakov odtlačku. Zamerali sme sa na Level-2 znaky prestavujúce tvarové zvláštnosti papilárnych línii. OpenFinger systém využíva spôsob ich extrakcie založený na kombinácii konvečných metód a konvolučnej neurónovej siete. Výsledky extrakcie sú exportované v štandardizovanom elektronickom formáte ISO/IEC 19794-2. Odhalené Level-2 znaky sú v samostatnej knižnici porovnávané algoritmami BOZORTH3 a Suprema BioMini SDK.

Odpoveď od servera je následne zhoda vo forme identifikačného čísla odtlačku, ktorý bol identifikovaný ako zhodný alebo vo forme potvrdenia, či bola alebo nebola úspešná verifikácia. Tento výsledok sa následne spracuje a zobrazí na strane klienta v aplikácii. Možnosti používania webovej aplikácie sú demonštrované na už spomínaných príkladoch a na jej možnom nasadení na školskom serveri so systémom OpenFinger.

# Zoznam použitej literatúry

1. *Daktyloskopia*. Dostupné tiež z: <https://sk.wikipedia.org/wiki/Daktyloskopia>.
2. *Biometrický systém*. Dostupné tiež z: [https://www.fei.stuba.sk/buxus/docs/2020/pavol\\_marak\\_autoreferat.pdf](https://www.fei.stuba.sk/buxus/docs/2020/pavol_marak_autoreferat.pdf).
3. MARÁK, Pavol. *OpenFinger: systém na daktyloskopickú autentifikáciu s GPU akceleráciou*. 2020. Dostupné tiež z: [https://www.fei.stuba.sk/buxus/docs/2020/pavol\\_marak\\_autoreferat.pdf](https://www.fei.stuba.sk/buxus/docs/2020/pavol_marak_autoreferat.pdf).
4. CHAMPOD, C. *Fingerprints and Other Ridge Skin Impressions*. 2004. Dostupné tiež z: [http://www.esalq.usp.br/lepe/imgs/conteudo\\_thumb/Fingerprints-and-Other-Ridge-Skin-Impressions.pdf](http://www.esalq.usp.br/lepe/imgs/conteudo_thumb/Fingerprints-and-Other-Ridge-Skin-Impressions.pdf).
5. *Daktyloskopické markanty*. Dostupné tiež z: <https://www.mdpi.com/1424-8220/13/3/3142/htm>.
6. *Futronic FS-80H*. Dostupné tiež z: [https://www.futronic-tech.com/pro-detail.php?pro\\_id=1543](https://www.futronic-tech.com/pro-detail.php?pro_id=1543).
7. *Protocol Buffers*. Dostupné tiež z: <https://developers.google.com/protocol-buffers>.
8. *JSON*. Dostupné tiež z: <https://support.apple.com/sk-sk/guide/shortcuts/apd0f2e057df/ios>.
9. *Protocol Buffers vs. JSON*. Dostupné tiež z: <https://www.bizety.com/2018/11/12/protocol-buffers-vs-json/>.
10. *Protocol Buffers – porovnanie rýchlosťi*. Dostupné tiež z: <https://dzone.com/articles/protobuf-alternative-to-rest-for-microservices>.
11. *SSL certifikát*. Dostupné tiež z: <https://www.sslmentor.sk/ssl/ssl-certifikaty>.
12. *SSL kanál*. Dostupné tiež z: <https://comodosslstore.com/blog/wp-content/uploads/2017/07/ssl-handshake.jpg>.
13. *Verzie ItelliJ Idea*. Dostupné tiež z: <https://www.itnetwork.sk/java/intellij-idea-netbeans-eclipse-pokročilu-pracu/intellij-idea-uvod-do-vyvojoveho-prostredia>.
14. *IntelliJ Idea*. Dostupné tiež z: <https://www.jetbrains.com/idea/>.

15. JOHNSON, Rod. *Spring framework*. 2004-2010. Dostupné tiež z: <https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/>.
16. *Spring framework – moduly a zlepšenia*. Dostupné tiež z: <http://springtutorials.com/introduction-to-spring-modules/>.
17. *Spring MVC*. Dostupné tiež z: <https://robime.it/spring-mvc-zaklady-jaroslav-beno/>.
18. *Java Persistance a Hibernate*. Dostupné tiež z: <https://www.learn2code.sk/blog/java-persistence-jpa-hibernate-orm>.
19. *Java Persistance a Hibernate, prepojenie a rozdiely*. Dostupné tiež z: <https://www.javaguides.net/2018/12/what-is-difference-between-jpa-and-hibernate.html>.
20. *H2 databáza*. Dostupné tiež z: <https://www.h2database.com/>.
21. *Java native interface*. Dostupné tiež z: <https://www.itnetwork.sk/java/jni/nastavenie-prostredia-pre-jni-eclipse-mingw>.

# Prílohy

A Štruktúra elektronického nosiča . . . . .	II
---	----

# A Štruktúra elektronického nosiča

Štruktúra prílohy:

## /ScannerApplication

- zdrojové súbory webovej aplikácie

## /ftrJavaScanAPI

- zdrojové súbory a knižnice potrebné na komunikáciu so senzorom Futronic FS-80H

## /BPJanVladar.pdf

- pdf hlavná časť záverečnej práce

Názov odovzdaného archívu: BPJanVladar.zip