Rozšíření systému výdejny zdravotnických pomůcek pro diabetiky

Bc. Anna Hana Pilková

Diplomová práce
2025

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Anna Hana Pilková**
Osobní číslo:	**A23440**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Rozšíření systému výdejny zdravotnických pomůcek pro diabetiky**
Téma práce anglicky:	**Extension of a Medical Supply Dispensing System for Diabetic Patients**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Analyzujte současná řešení mobilních aplikací zaměřených na zdravotnické pomůcky a specifické potřeby diabetiků.
3. Navrhněte mobilní aplikaci pro výdejnu zdravotnických pomůcek pro diabetiky podle požadavků společnosti DiaTop.
4. Implementujte navrženou mobilní aplikaci a v případě potřeby rozšiřte funkcionalitu stávající webové aplikace.
5. Integrujte mobilní aplikaci s webovou aplikací společnosti DiaTop.
6. Věnujte pozornost zabezpečení aplikace.
7. Proveďte testování aplikace a popište její nasazení.

Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. RAVULAVARU, Arvind. *Ionic Framework: Building Cross-Platform Mobile Apps Using Ionic and Angular*. Packt Publishing, 2016. ISBN 9781786468251.
2. SPILCA, L. *Spring Security in Action*. 2. Manning Publications, 2020. ISBN 9781617297731.
3. MATTHIAS, Karl a KANE, Sean P. *Docker: Up & Running*. O'Reilly Media, 2015. ISBN 978-1-491-91757-2.
4. DORMANN, Andreas. *Ionic 7*. D&d Verlag Bonn, 2023. ISBN 9783945102602.
5. CALLAGHAN, Michael. *Deploying a Mobile Application to the App Store with Ionic Capacitor*. Amazon Digital Services LLC – Kdp, 2020. ISBN 9798686640993.

Vedoucí diplomové práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **27. října 2024**
Termín odevzdání diplomové práce: **2. června 2025**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne ....

podpis autora

ABSTRAKT

Diplomová práca sa zaoberá návrhom a tvorbou rozšírenia systému výdajne zdravotníckych pomôcok pre diabetikov. Zameriava sa na návrh rozšírenia, implementáciu a analýzu podobných riešení. Teoretická časť je orientovaná na technológie použité pri implementácii. Porovnáva kandidátov technológií a spomedzi nich vyberá najvhodnejších pre daný systém. V praktickej časti sa venuje analýze podobných riešení mobilných aplikácií. Rozoberá ich výzor a funkcie. Výsledky analýzy zohľadňuje v samotnom návrhu a implementácii. Rovnako nasleduje požiadavky zadávateľa, ktoré do systému dopĺňa.

Klíčová slova: Ionic, Capacitor, Kafka, Angular, Docker, mobilná aplikácia, webová aplikácia

ABSTRACT

The diploma thesis focuses on the design and development of an extension of the medical supplies dispensing system for diabetic patients. It concentrates on the proposal of the extension, its implementation, and the analysis of similar solutions. The theoretical part is oriented towards the technologies used in the implementation. It compares potential technology candidates and selects the most suitable ones for the given system. The practical part analyzes similar mobile application solutions, examining their appearance and functionality. The results of this analysis are reflected in the design and implementation. Additionally, the thesis incorporates the client's requirements into the system.

Keywords: Ionic, Capacitor, Kafka, Angular, Docker, mobile application, web application

Chcela by som sa poďakovať spoločnosti Diatop za možnosť naďalej pracovať na tomto projekte. Rovnako patrí poďakovanie môjmu vedúcemu diplomovej práce Ing. Tomášovi Vogeltanzovi, Ph.D. za jeho podporu a venovaný čas. Veľká vďaka patrí môjmu priateľovi a rodine, ktorí mi boli veľkou oporou. Taktiež ďakujem Júliusovi Rebeťákovi a spoločnosti Alfeus za sprístupnenie, nastavenie a pomoc s napojením na ich systém. Rovnako sa chcem poďakovať Marekovi Pflieglerovi, Petrovi Mervartovi a Petre Kastlerovej za ich čas, ktorý venovali kontrole tejto práce.

OBSAH

ÚVOD **13**

I TEORETICKÁ ČÁST **14**

1 HYBRIDNÁ MOBILNÁ APLIKÁCIA **15**

1.1 IONIC 16

1.1.1 Ionic CLI 17

1.1.2 Ionic UI Komponenty 17

1.2 CAPACITOR 18

1.2.1 Natívny most 19

1.2.2 Natívny runtime 21

1.2.3 Pluginy 22

1.2.4 Capacitor vs Apache Cordova 25

1.3 IONIC VS FLUTTER 27

2 BEZPEČNOSŤ **30**

2.1 BEZPEČNOSTNÉ HROZBY 30

2.1.1 Bezpečnostné hrozby špecifické pre mobilné aplikácie 35

2.1.2 Bezpečnostné hrozby špecifické pre webové aplikácie 38

2.2 AUTENTIFIKÁCIA POMOCOU BIOMETRIKY V MOBILNOM ZARIADENÍ 40

2.2.1 Výhody a potenciálne riziko 41

2.2.2 Apple 42

2.2.3 Android 45

2.3 ZABEZPEČENIE APLIKÁCIE SSL/TLS CERTIFIKÁTOM 50

2.3.1 TLS a SSL protokol 51

3 ZÁSADY GDPR PRI TVORBE APLIKÁCIÍ **53**

3.1 POJMY V GDPR 53

3.2 POŽIADAVKY GDPR 54

3.3 TECHNICKÉ ASPEKTY IMPLEMENTÁCIE GDPR 55

3.3.1 Šifrovanie a zabezpečenie dát 55

3.3.2 Pseudonymizácia 58

3.4 Cookies a sledovacie technológie 59
3.5 Kroky pri tvorbe aplikácie vyhovujúcej GDPR 60
4 KOMUNIKAČNÉ MECHANIZMY MEDZI MIKROSERVISAMI A NOTIFIKAČNÝ SYSTÉM 65
4.1 Komunikačné mechanizmy medzi mikroservisami 65
4.2 Notifikačný systém v mobilných aplikáciách 68
4.2.1 Architektúra notifikačného systému 69
4.2.2 Poskytovatelia služieb pre push notifikácie a ich porovnanie 71
4.2.3 Firebase Cloud Messaging (FCM) 71
5 NASADENIE APLIKÁCIE 74
5.1 Spôsoby nasadenia backend časti aplikácie 75
5.2 Spôsoby nasadenia mobilnej aplikácie 77
5.2.1 Google Play 78
5.2.2 Apple App Store 78
II PRAKTICKÁ ČÁST 80
6 ANALÝZA SÚČASNÝCH RIEŠENÍ PODOBNÝCH MOBILNÝCH APLIKÁCIÍ 81
6.1 Mobilná aplikácia Dôvera 81
6.2 Mobilná aplikácia Medevio 83
7 NÁVRH MOBILNEJ APLIKÁCIE 85
7.1 Funkčné požiadavky 85
7.2 Aktéri 86
7.3 Model prípadov užitia 86
7.4 Architektonický model aplikácie 87
7.5 Architektúra systému 89
7.5.1 Dátový model 89
7.5.2 Model tried 90
8 POPIS IMPLEMENTÁCIE 98
8.1 Verifikácia emailovej adresy 98
8.2 Obnova zabudnutého hesla 100

8.3 Šifrovanie citlivých údajov 102
8.4 Mobilná aplikácia 104
8.4.1 Prihlásenie 104
8.4.2 Registrácia 107
8.4.3 Používateľská aplikácia 108
8.4.4 Administrátorská aplikácia 121
8.5 Push notifikácie 127
9 POPIS NASADENIA 133
9.1 Nasadenie webovej aplikácie 133
9.1.1 Príprava Gateway mikroservisy 133
9.1.2 Príprava Diatop mikroservisy 134
9.1.3 Príprava Admin mikroservisy 134
9.1.4 Príprava Model mikroservisy 135
9.1.5 Príprava Resource mikroservisy 135
9.1.6 Príprava PushNotifications mikroservisy 135
9.1.7 Kúpa domény 136
9.1.8 Kúpa a nastavenie VPS 136
9.1.9 Reverzný proxy server NGINX 137
9.2 Nasadenie mobilnej aplikácie 139
9.2.1 Nasadenie aplikácie na Android 139
9.2.2 Nasadenie aplikácie na iOS 139
ZÁVĚR 141
SEZNAM POUŽITÉ LITERATURY 143
SEZNAM OBRÁZKŮ 153
SEZNAM TABULEK 155
SEZNAM VÝPISŮ 156
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 158
SEZNAM PŘÍLOH 159

ÚVOD

Moderná doba sa posúva rýchlo vpred a to platí aj v súvislosti s výdajom zdravotníckych pomôcok. V dnešnej dobe ľudia nepoužívajú len webové aplikácie, ale čoraz viac siahajú práve po mobilných aplikáciách. Tento trend ovplyvňuje aj zdravotníctvo a výdaj liekov, či pomôcok. Je to ďalší spôsob, ako jednoduchšie komunikovať so zákazníkmi. Je to pridaná hodnota nielen pre spoločnosť, ale aj jej klientov. Umožňuje odľahčiť výdajne s objednávaním dodávok pomôcok a zákazníkom ponúka možnosť objednať objednávku, dostať upozornenie o jej odoslaní a sledovať dôležité informácie spojené s ich pomôckami.

Cieľom diplomovej práce je rozšíriť systém výdajne zdravotníckych pomôcok pre diabetikov. Rozšírenie sa týka tvorby mobilnej aplikácie pre objednávanie zdravotníckych pomôcok pre diabetikov na základe receptu od ošetrujúceho lekára. Okrem toho aplikácia ponúka notifikácie o stave ich objednávky a informácie o čerpaní finančných prostriedkov, ktoré im boli na tieto pomôcky pridelené. Pre administrátorov je taktiež vytvorená mobilná aplikácia, ktorá ponúka prehľad štatistík a objednávok, čakajúcich na vybavenie. Cieľom diplomovej práce je tiež doplniť existujúcu webovú aplikáciu o nové funkcie pre používateľa, ako aj administrátora. V administrátorskej časti je webová aplikácia rozšírená o štatistiky, či možnosť označiť objednávky za odoslané. Webová aplikácia pre používateľa je doplnená informáciami o čerpaní rozpočtu na pomôcky a funkciami spojenými s používateľským účtom.

Diplomová práca je rozdelená na dve časti. Prvá časť je teoretickou rovinou práce. Zaoberá sa technológiami použitými pri vývoji rozšírenia systému výdajne. Rozoberá hybridné mobilné technológie, bezpečnosť, požiadavky GDPR na aplikácie, komunikačné mechanizmy, či samotné nasadenie webovej a mobilnej aplikácie.

Druhá časť je zameraná na praktickú časť diplomovej práce. Analyzuje podobné riešenia mobilných aplikácií a výsledky analýzy implementuje do návrhu mobilnej aplikácie. Návrh sa riadi požiadavkami výdajne Diatop, ktoré sú zohľadnené v diagramoch a modeloch. Pozornosť je rovnako venovaná samotnej implementácii rozšírenia systému. Popis implementácie je doplnený o ukážky z mobilnej, či webovej aplikácie. V závere práce je popísaný postup nasadenia celého systému a teda webovej aj mobilnej aplikácie.

I TEORETICKÁ ČÁST

1 HYBRIDNÁ MOBILNÁ APLIKÁCIA

Vývoj mobilných aplikácií má v dnešnej dobe mnoho podôb aj vďaka rôznym operačným systémom, ktoré sú v dnešných mobilných zariadeniach použité. Každý operačný systém ponúka svoj vlastný jazyk - API na vývoj mobilnej aplikácie pre danú platformu.[1] V prípade Android platformy sa používa Kotlin alebo Java a aplikácia je kompilovaná pomocou Android Studio. Operačný systém iOS od spoločnosti Apple vytvoril programovací jazyk Objective C a neskôr Swift, kedy je aplikácia kompilovaná pomocou Xcode.[2] Tomuto vývoju sa často vraví natívny. Vývojári majú vďaka použitiu riešenia danej platformy na vývoj jednoduchý prístup k možnostiam, ktoré zariadenie ponúka. Jednou z nevýhod vývoja mobilnej aplikácie pre danú platformu je dvojitá práca - pre každú platformu musí byť separátny kód a teda dvojitá implementácia.[3]

Hybridná mobilná aplikácia tento problém rieši jedným kódom - implementáciou pre obe platformy.[2] Hybridná preto, pretože kombinuje webové a natívne technológie. Webové technológie sú HTML, CSS a JavaScript, ktoré sú použité v rámci implementácie a teda kód je písaný v týchto jazykoch. Avšak, používateľ aplikácie ju nepoužíva vo webovom prehliadači, ten je skrytý a pre užívateľa neviditeľný. Na spustenie aplikácie sa používa vstavaný webový prehliadač danej platformy. V prípade iOS je to WKWebView a pre Android platformu je to WebView. Kód písaný webovými technológiami je vložený do natívneho obalu pomocou technológií ako Apache Cordova či Capacitor od Ionic. Je to teda mobilná aplikácia, ktorá vo vnútri spúšťa webovú stránku – ako keby používateľ otvoril web v prehliadači, ale celý tento "prehliadač" je ukrytý v aplikácii. Vďaka tejto technológii je možné vytvoriť mobilnú aplikáciu pre viacero platforiem, ale s jedným kódom. Pomocou pluginov je možné využívať takmer úplne všetky možnosti mobilného zariadenia ako snímač odtlačku prsta, fotoaparát či súborový systém. Na to, aby užívateľské rozhranie pôsobilo rovnako, ako ostatné komponenty (tlačítka, kalendáre a iné) na mobilnom zariadení, boli vytvorené knižnice komponent ako Ionic, Xamarin a ďalšie.[3]

Výber medzi natívnym a hybridným prístupom nie je priamočiary a závisí na viacerých faktoroch. Je nutné zvážiť finančné prostriedky a čas, ktorý je možné vývoju venovať. Ak je nutný vysoký výkon, ako napríklad pri hrách, odporúča sa natívny prístup. Ak je potrebné robiť časté aktualizácie, hybridný prístup býva lepší, pretože je nutné meniť len jeden kód. Je potrebné zvážiť aj znalosti vývojárov a čas potrebný na vývoj.[4] Základné rozdiely medzi

natívnym a hybridným prístupom, je možné vidieť v tabuľke 1.1.

Tabulka 1.1 Porovnanie hybridného a natívneho prístupu

Vlastnosť	**Natívna**	**Hybridná**
Prístup k zariadeniu	Plný	Plný (pomocou plugin)
Podpora viacerých platforiem	Nie	Áno
Programovací jazyk	Špecifický pre platformu	HTML, CSS, JavaScript
Užívateľský dojem	Silný	Stredný - silný
Výkon	Vysoký	Stredný - vysoký
Zdieľanie kódu	Nie	Áno

(zdroj: 3)

1.1 Ionic

Ionic vznikol v roku 2013. Je to viacplatformový vývojový rámec (angl. cross-platform framework), ktorý zásadne mení spôsob vývoja mobilných aplikácií. Umožňuje vývojárom vytvárať aplikácie, ktoré fungujú na viacerých platformách s použitím jednotnej kódovej základne (angl. single codebase). Tým výrazne skracuje čas uvedenia na trh, znižuje náklady na vývoj a zjednodušuje následnú údržbu. Znovupoužiteľnosť kódu zaručuje jednotné správanie na všetkých platformách, kde mobilná aplikácia beží. Ionic rovnako ponúka tvorbu Progresívnych webových aplikácií (PWAs) či desktopové aplikácie použitím Electron.

Základným stavebným kameňom sú webové technológie - HTML5, CSS3 a JavaScript. Boli vybrané práve pre ich vysokú používanosť medzi programátormi. To dáva možnosť vývojárom z webového prostredia tvoriť mobilné aplikácie za použitia už im dobre známych technológií.

Ionic ponúka bohatú knižnicu UI komponent, ktoré sú optimalizované pre danú platformu. Bez nutnosti dodatočného programovania či nastavenia zo strany vývojárov sú komponenty v mobilnej aplikácii zobrazené podľa platformy, na ktorej aplikácia beží. Vďaka tomu je vývoj rýchlejší a jednoduchší a zároveň ponúka koncovým užívateľom rovnaký komfort, používateľskú skúsenosť (UX) a používateľské rozhranie (UI).

Natívne aplikácie majú vo výkone miernu výhodu. Avšak, Ionic sa im čím ďalej tým viac približuje vďaka moderným webovým technológiám a optimalizačným technikám. Je navrhnutý tak, aby bol nezávislý od výkonu konkrétnej platformy. Už samotný návrh nepredstavuje prekážku pre efektívne fungovanie aplikácie. Aplikácie vytvorené pomocou Ionic môžu

dosahovať rovnakú alebo približujúcu sa úroveň výkonu natívnych aplikácií. Musia byť však dodržané správne programátorské postupy ako efektívne využívanie zdrojov a aplikovanie rôznych optimalizačných stratégií.[5]

1.1.1 Ionic CLI

Nástroj Ionic CLI predstavuje neoddeliteľnú súčasť vývojového prostredia. Poskytuje širokú škálu funkcií na vytváranie, zostavovanie a nasadzovanie aplikácií vytvorených pomocou Ionic. Tento nástroj zefektívňuje množstvo bežných vývojárskych úloh, vďaka čomu sa vývojári môžu viac sústrediť na samotnú logiku aplikácie namiesto zdĺhavého nastavovania základnej štruktúry a konfigurácie projektu. Poskytuje taktiež príkazy na generovanie Angular komponent, direktív, stránok a iných. Rovnako ponúka okamžité aktualizovanie aplikácie pri vývoji, takže zmeny v kóde sú okamžite viditeľné a je možné ich testovať.[5]

1.1.2 Ionic UI Komponenty

V oblasti softvérových a mobilných aplikácií predstavujú komponenty používateľského rozhrania (UI) kľúčové prvky, pomocou ktorých používatelia interagujú s aplikáciou. Medzi tieto prvky patria tlačidlá, karty, zoznamy, modálne okná a mnohé ďalšie. Interakcia zásadným spôsobom ovplyvňuje celkový dojem používateľa z aplikácie a často rozhoduje o jej úspechu či neúspechu.

Ionic nepodceňuje význam používateľskej skúsenosti (UX) a venuje jej zvýšenú pozornosť. Vývojárom poskytuje knižnicu vopred pripravených UI komponent. Tieto komponenty nasledujú osvedčené princípy moderného UI/UX dizajnu. Je možné ich jednoducho integrovať do ľubovoľnej aplikácie a zabezpečujú konzistentný a plynulý používateľský zážitok naprieč rôznymi platformami – iOS, Android, ale aj webovým prostredím.

Komponenty sú vytvorené tak, aby sa automaticky prispôsobovali konkrétnej platforme, na ktorej bežia. Tým sa zachovávajú základné princípy dizajnu špecifické pre danú platformu. Avšak, komponenty je možné jednoducho rozširovať, upravovať ich vzhľad, alebo vytvárať vlastné komponenty na mieru.

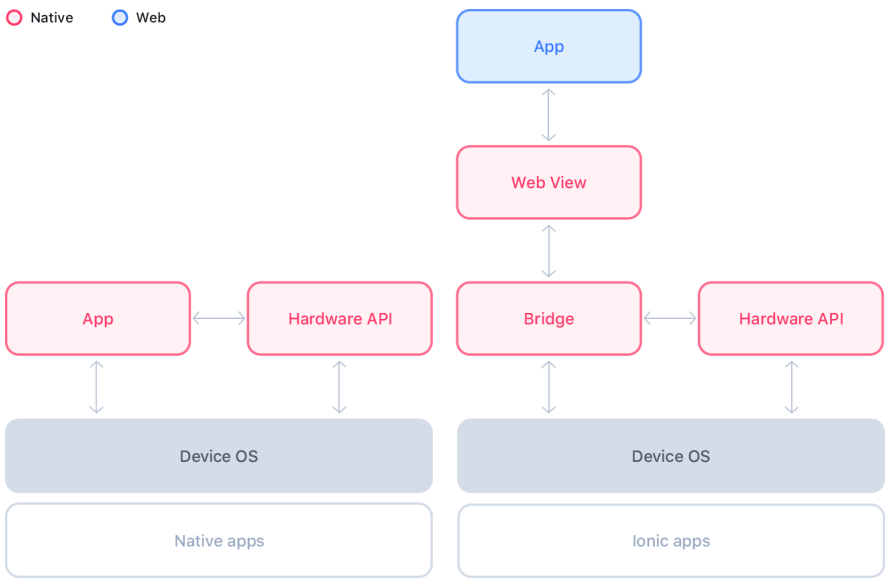
Každá komponenta poskytuje sadu CSS premenných, ktoré je možné prepísať v danej komponente, alebo využiť možnosť vytvorenia témy. Témy sú možnosť, ako jednoducho zmeniť všetky farby, ktoré Ionic komponenty používajú. To ponúka neobmedzené možnosti

na prispôsobenie aplikácie vzhľadu, ktorý korešponduje so špecifikami produktu.[5]

1.2 Capacitor

Capacitor predstavuje natívne runtime prostredie určené pre multiplatformový vývoj, ktoré umožňuje integráciu webových technológií (HTML, CSS, JavaScript) s natívnymi API mobilných operačných systémov ako Android a iOS. Jeho tvorcovia ho nazývajú novou generáciou hybridných aplikácií. Ponúka natívny kontajner pre vývojárov, ktorí sú zbehlí vo webových technológiách a chcú tvoriť moderné mobilné aplikácie bez straty prístupu k natívnym funkciám mobilných zariadení.[6]

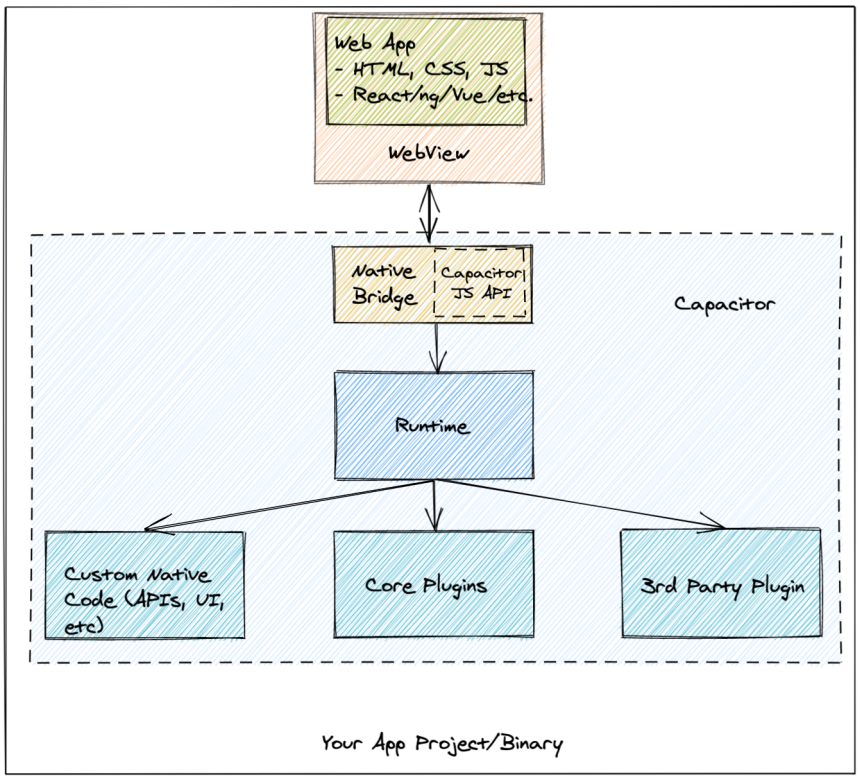
Capacitor umožňuje nasadzovať aplikácie do iOS, Android či ako webové aplikácie. Na obrázku 1.1 je znázornený architektonický rozdiel medzi natívnou mobilnou aplikáciou a hybridnou aplikáciou, ktorá používa Capacitor.



Obrázek 1.1 Natívna mobilná aplikácia vs aplikácia s použitím Capacitor

(zdroj: 7)

Capacitor je tvorený viacerými komponentami ako natívny most (native bridge), runtime, vlastný natívny kód a pluginy. Pluginy sú rozdelené do troch skupín podľa ich pôvodu. Týmito skupinami sú hlavné (core) a pluginy sprostredkované tretími stranami. Všetky tieto komponenty sa potom nachádzajú v projekte - binárnej reprezentácii danej aplikácie, kedy natívny most komunikuje s Web View (webové technológie). Schému týchto komponent zobrazuje obrázok 1.4.



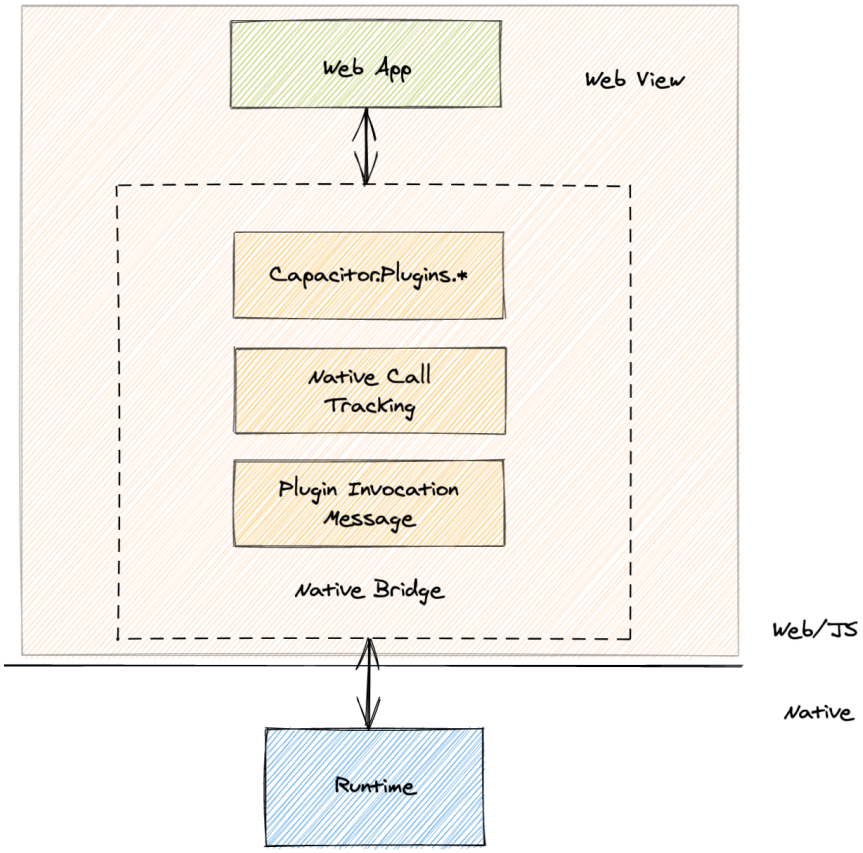
Obrázek 1.2 Komponenty Capacitor

(zdroj: 8)

1.2.1 Natívny most

Natívny most poskytuje webovému zobrazeniu (Web View) v rámci Capacitoru mechanizmus, ktorým môže pristupovať k natívnej funkcionalite zariadenia, komunikovať s ovládacími prvkami na úrovni operačného systému a používať vlastný natívny kód alebo moduly tretích strán. V tomto moste (bridge) sú sprístupnené všetky natívne pluginy a ich metódy cez tzv. **runtime JavaScript API**. Toto API je automaticky exportované do webovej vrstvy. **Runtime API** je súčasťou prostredia Capacitoru a cez `window.Capacitor.Plugins` je automaticky dostupné. Na druhej strane je **@capacitor/core API**, ktoré vývojár importuje do svojej webovej aplikácie (napr. do Angular, React alebo Vue projektu). Obe tieto API spolupracujú tak, aby umožnili jednotný prístup k natívnym funkciám naprieč rôznymi platformami (Android, iOS, web). Počas spustenia aplikácie Capacitor načíta všetky pluginy, ktoré boli nainštalované (napr. plugin `@capacitor/geolocation`, alebo priamo naprogramované vo vlastnom natívnom kóde. Tieto pluginy následne exportuje do objektu `window.Capacitor.Plugins`, ktorý obsahuje zoznam všetkých dostupných pluginov a ich

verejne prístupných metód.



Obrázek 1.3 Schéma natívneho mostu

(zdroj: 8)

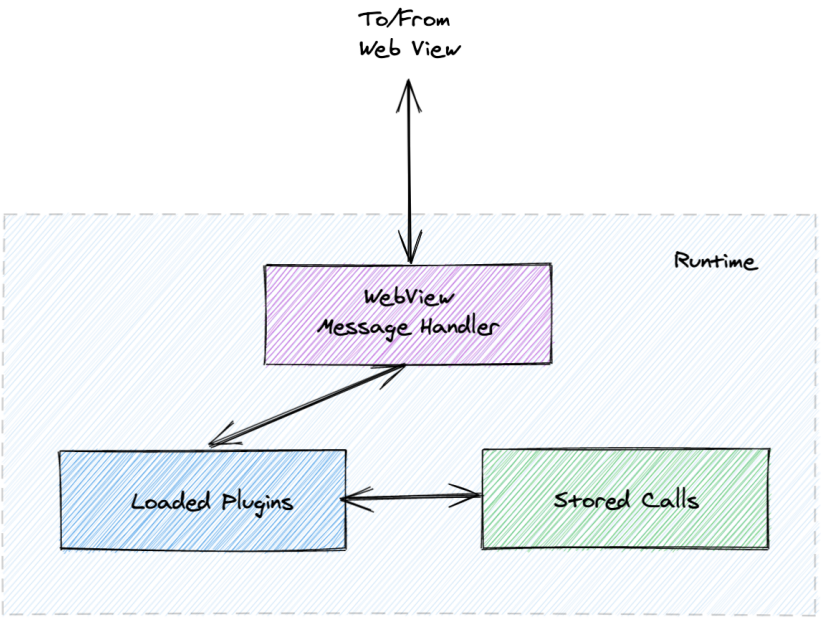
Bridge (most) zároveň zabezpečuje samotnú komunikáciu medzi webovým a natívnym svetom – tzn. odosielanie správ, spracovanie odpovedí a sledovanie požiadaviek (napr. volania metód).[8]

V praxi prebieha volanie natívnej funkcionality nasledovne: Webová aplikácia napísaná v JavaScripte (napr. pomocou `@capacitor/core`) vykoná volanie na konkrétny plugin, napríklad `Camera.getPhoto()`. Toto volanie je následne serializované do formy JSON správy, ktorá je cez **Native Bridge** odoslaná do natívnej vrstvy zariadenia. Na platforme Android sa tento prenos realizuje pomocou `WebViewInterface`, zatiaľ čo na iOS je využívané API `postMessage`. Natívna vrstva po prijatí požiadavky vykoná príslušnú operáciu (napríklad otvorenie fotoaparátu) a výsledok – či už úspech alebo chyba – odošle späť cez bridge do webovej vrstvy. Na strane webovej aplikácie sa odpoveď spracuje ako `Promise`, čo umožňuje asynchrónne spracovanie výsledku v štýle moderného JavaScriptu.[9]

1.2.2 Natívny runtime

V Natívnom runtime sú volania z Web View smerované do pluginov implementovaných ako vlastný natívny kód v projekte a natívnych pluginov nainštalovaných ako balíčky. Poskytuje spôsob, ako zabaliť modernú webovú aplikáciu do natívnej shell aplikácie pre platformy Android a iOS.[8]

Pri spustení aplikácie Capacitor načíta z natívnej vrstvy zdrojový súbor `index.html`, ktorý inicializuje webovú časť aplikácie. Tento súbor býva súčasťou zložky www, ktorú generuje buildovací nástroj frontendového frameworku.[9] Rovnako načíta všetky nainštalované pluginy a vloží do Web View JavaScript symboly pre všetky pluginy.[8] Keď sa aplikácia spustí, hlavné slovo preberá JavaScriptová časť, ktorá komunikuje s natívnou vrstvou cez tzv. most (bridge). Vďaka tomu môže webový kód volať funkcie, ako napríklad fotoaparát alebo lokalizáciu.[9]



Obrázek 1.4 Schéma Capacitor runtime

(zdroj: 8)

Keď webová aplikácia vyvolá volanie na niektorý z pluginov, Capacitor runtime spracuje toto volanie ako správu. Na základe obsahu správy vytvorí volanie príslušnej metódy natívneho pluginu a vykoná ho. Všetky volania pluginov v rámci Capacitoru sú asynchrónne, preto runtime vedie interný zoznam aktívnych volaní. Volaní, ktoré ešte neboli dokončené.

Tieto volania môžu byť jednoduché (napr. získanie údajov z natívneho API), alebo komplexnejšie (napr. otvorenie natívneho systému a čakanie na výsledok z inej aplikácie – ako je napríklad natívna kamera na platforme Android). Po vykonaní daného natívneho volania je zostavená odpoveď vo forme správy, ktorá sa odošle späť do webovej vrstvy aplikácie (Web View). Táto odpoveď má za následok, že pôvodné volanie pluginu v JavaScripte sa vyrieši, a teda Promise sa buď naplní úspešným výsledkom, alebo zlyhá s chybou.[8]

1.2.3 Pluginy

Capacitor ponúka množstvo pluginov, ktoré umožňujú JavaScriptu pripojiť sa priamo na Natívne API. Obaľujú štandardné natívne operácie, ktoré používajú rôzne API platformy a tým sprístupňujú ich funkcionalitu JavaScriptu.

Capacitor generuje JavaScript životné cyklické metódy (tzv. hooks) na strane klienta. Preto vytvorené pluginy pre Capacitor potrebujú iba používanie natívnych jazykov (Swift/Objective C a/alebo Java/Kotlin). Je rovnako možné naprogramovať vlastný plugin.

Pluginy, ktoré Capacitor ponúka, sa delia na oficiálne a takzvané *Community* - vytvorené externými programátormi a nie tímom vyvíjajúcim Capacitor.[10] Medzi oficiálne Capacitor pluginy patria:

CapacitorHttp je oficiálny plugin od Capacitor, ktorý umožňuje vykonávať HTTP požiadavky (napr. GET, POST, PUT, DELETE) priamo z natívnej vrstvy aplikácie. Na rozdiel od klasického webového rozhrania fetch, ktoré beží v rámci Web View, CapacitorHttp používa natívne siete (napr. NSURLSession na iOS a HttpURLConnection alebo OkHttp na Android).[11]

Vďaka tomu je možné obísť obmedzenia ako CORS.[12]

Volanie sa vykonáva pomocou metódy CapacitorHttp.request(...), ktorá očakáva konfiguráciu požiadavky (napr. URL, metódu, hlavičky, telo) a vracia Promise. Výsledok požiadavky sa potom spracováva v JavaScripte podobne ako pri použití fetch.[11]

Push-notifications poskytuje rozhranie pre integráciu natívnych push notifikácií v mobilných aplikáciách vytvorených pomocou Capacitoru. Umožňuje registráciu zariadenia na prijímanie notifikácií, spracovanie prijatých správ a reakciu na interakcie používateľa.[13]

Preferences poskytuje jednoduché API na ukladanie a načítavanie údajov v natívnom úložisku. Ukladanie je vo forme kľúč – hodnota (key – value store). Je určené pre potreby mobilných aplikácií vytvorených pomocou Capacitoru.

Ukladanie údajov pomocou tohto pluginu je perzistentné čo znamená, že údaje sú dostupné aj po reštarte aplikácie. Toto úložisko sa hodí najmä na uchovávanie nastavení používateľa, ako je predvolený jazyk či svetlý/tmavý mód, alebo jednoduchých konfiguračných údajov. Na Android používa `SharedPreferences` a na iOS `UserDefaults`.

Zabezpečuje jednoduché a bezpečné uloženie malého množstva údajov v aplikácii bez potreby použitia databázy.

Tento plugin nie je ideálny na ukladanie hesiel alebo citlivých údajov – na to je lepšie použiť bezpečnejší spôsob, ako napríklad Secure Storage. [14]

Medzi *Community* pluginy patria:

@aparajita/capacitor-biometric-auth s MIT lincenciou. Poskytuje aplikáciám postaveným na Capacitore prístup k natívnym biometrickým autentifikačným mechanizmom a overovaniu zariadenia na platformách iOS a Android. Podporuje všetky typy biometrie a konfiguračné možnosti dostupné na oboch platformách. Okrem toho umožňuje simuláciu biometrie a overovania zariadenia na webe, čo vývojárom umožňuje testovať logiku autentifikácie bez potreby úprav kódu.

- Plugin podporuje rôzne typy biometrie, ako sú odtlačky prstov, rozpoznávanie tváre a dúhovky, v závislosti od schopností zariadenia.
- V prípade, že biometria nie je dostupná alebo povolená, plugin umožňuje overenie pomocou zabezpečenia zariadenia, ako je PIN, vzor alebo heslo.
- Na webových platformách plugin simuluje biometrickú autentifikáciu, čo umožňuje vývojárom testovať autentifikačné toky bez potreby fyzického zariadenia s biometrickými senzormi.
- Funkcia `checkBiometry()` umožňuje zistiť, či je biometria dostupná a aký typ biometrie je podporovaný na danom zariadení. Táto funkcia tiež poskytuje informácie o dôvodoch, prečo biometria nemusí byť dostupná, ako napríklad neakti-

vovaná biometria pre aktuálneho používateľa alebo zakázaná biometria pre danú aplikáciu.

- Metóda `authenticate()` spúšťa proces biometrickej autentifikácie. V prípade úspechu sa vráti potvrdenie, v prípade zlyhania sa vyhodí chyba typu BiometryError s príslušným kódom chyby.
- Funkcia `addResumeListener()` umožňuje aplikácii reagovať na zmeny v dostupnosti biometrie, napríklad keď sa aplikácia vráti z pozadia a dostupnosť biometrie sa mohla zmeniť.[15]

@aparajita/capacitor-secure-storage rovnako s MIT licenciou. Aplikáciám postaveným na Capacitore ponúka bezpečné úložisko vo forme kľúč – hodnota pre platformy iOS a Android. Umožňuje bezpečné ukladanie údajov, ako sú prihlasovacie údaje alebo iné citlivé informácie, s využitím natívnych mechanizmov šifrovania a úložiska na každej platforme. Plugin je navrhnutý na bezpečné ukladanie citlivých údajov a využíva natívne bezpečnostné mechanizmy každej platformy. Na iOS sú údaje uložené v Keychain, ktorý je šifrovaný a prístupný len aplikáciám s rovnakým App ID. Na Android sú údaje šifrované pomocou AES v režime GCM a uložené v SharedPreferences, ktoré sú špecifické pre aplikáciu.

- Na iOS sú údaje uložené v šifrovanej systémovej kľúčenke (Keychain), ktorá je špecifická pre aplikáciu. Na Android sú údaje uložené v SharedPreferences a šifrované pomocou AES v režime GCM s tajným kľúčom generovaným Android KeyStore.[16]
- Plugin umožňuje synchronizáciu údajov s iCloud Keychain, čo umožňuje zdieľanie bezpečných údajov (napríklad prihlasovacích údajov) medzi viacerými zariadeniami používateľa.[17] Synchronizáciu je možné ovládať globálne alebo pre jednotlivé operácie.
- Plugin umožňuje ukladanie rôznych typov údajov vrátane reťazcov, čísel, booleanov, objektov, polí a dátumov vo formáte JSON.[18]
- Plugin poskytuje jednoduché a známe API, čo uľahčuje jeho integráciu do existujúcich aplikácií.[16]

1.2.4 Capacitor vs Apache Cordova

Cordova je oproti Capacitoru starším *open-source* projektom, ktorý bol vytvorený v roku 2009, kdežto Capacitor bol vydaný v roku 2018. Cordova sa zameriava na spúšťanie webových aplikácií na rôznych platformách, ale nepodporuje PWA a Electron. Na druhej strane, Capacitor podporuje aj PWA a Electron.

Cordova je síce starším projektom, avšak oproti Capacitoru má určité výhody ako:

- Cordova má vybudovaný široký ekosystém podpory v rámci vývojárskej komunity. Keďže je na trhu dlhšie, pri hľadaní určitých konkrétnych problémov je jednoduchšie nájsť riešenie a celkovú podporu komunity ako dokumentácia a riešené problémy.
- Migrácia z Cordova na Capacitor môže viesť k problémom primárne pri využívaní pluginov špecifických pre Cordova.
- Jednoduchšia pre začiatočníkov a malé projekty hlavne v prípade nastavenia projektu.
- Vyššia podpora medzi nástrojmi tretích strán, knižníc a služieb práve vďaka tomu, že je dlhšie na trhu.
- Stabilita a spoľahlivosť hrá veľkú rolu pri výbere, pretože Cordova má za sebou dlhé roky vývoja a zdokonaľovania a môže sa pýšiť práve týmito vlastnosťami.

Capacitor vyniká svojou modernosťou a novšími technológiami, medzi jeho výhody patrí:

- Využíva moderné technológie, architektúru a nástroje v porovnaní s Cordova. Vďaka pokrokovým postupom je ľahko integrovateľný s modernými front-end rámcami, ako Angular či React.
- Vďaka optimalizovaným pluginom a štíhlejšej architektúre má vyšší výkon. Ten sa prejavuje v rýchlosti načítania aplikácie, či celkovom užívateľskom zážitku.
- Podporuje PWA .
- Jednoduchšia štruktúra projektu, ktorá je intuitívna a jednoduchšia na správu kódu. Má oddelené priečinky pre jednotlivé platformy a webové zdroje vo forme "www" priečinku. Priečinky pre platformy obsahujú natívny kód, takže web a natívna časť sa nemiešajú dokopy.

- Veľké množstvo pluginov s podporou najmodernejších natívnych API. Tento systém pluginov sleduje najmodernejšie trendy a je flexibilnejší v porovnaní s Cordova. Ponúka programátorom možnosť rozširovať ich aplikácie o natívnu funkcionalitu a tvorbu nových pluginov.
- Capacitor je neustále udržovaný a vyvíjaný Ionic tímom, čo z neho robí udržateľnú a škáľovateľnú voľbu.[19]

Kľúčové rozdiely medzi Capacitor a Apache Cordova:

Natívny most: Capacitor aj Cordova používajú WebView, avšak s malými rozdielmi.

Cordova používa WebView ako natívny most. Stará sa o vykresľovanie aplikácie a komunikáciu medzi JavaScript kódom a natívnymi funkciami zariadenia.

Capacitor využíva WebView a natívny most, ktorý je modernejší, rýchlejší (má lepší výkon) a flexibilnejší v pristupovaní k natívnym API.

Ekosystém pluginov: Capacitor aj Cordova majú nemalé množstvo pluginov, ktoré je možné použiť.

Cordova ponúka množstvo možností od tretích strán s dobrou dostupnosťou. Od zdrojového kódu doplnku pre fotoaparát až po geolokáciu.

Capacitor môže využiť všetky pluginy pre Cordova a k tomu pridáva svoje vlastné. Capacitor pluginy však poskytujú bezproblémovú integráciu s jeho natívnym mostom. To predstavuje moderný prístup k vývoju mobilných aplikácií.

Nástroje a JavaScript: Capacitor aj Cordova sú vytvorené pre tvorbu mobilných aplikácií s použitím webových technológií.

Cordova nie vždy drží krok s vývojom JavaScriptu, pričom niektoré jeho moderné vlastnosti nemusia byť plnohodnotne podporované.

Capacitor na druhej strane napreduje s dobou a podporuje najnovšie trendy, ako živé obnovenie pri zmene kódu (hot reloading). Zároveň je možné ťažiť z výhod, ktoré prináša samotný TypeScript, ako je typová bezpečnosť, či lepšia podpora v editore kódu, keďže podporuje aj moderný Typescript.[20]

V dnešnom svete rastú nároky na vývoj multiplatformových aplikácií. Mení sa aj samotný prístup k hybridnému vývoju. Capacitor sa čoraz viac ukazuje ako perspektívna platforma, najmä v spojení s frameworkom Ionic. Vďaka zameraniu na moderné vývojové praktiky, úzkej integrácii s webovými technológiami a podpore pre PWA. Capacitor umožňuje naplno využiť potenciál súčasných webových technológií a zároveň jednoducho integrovať natívne funkcionality do mobilných aplikácií, čím výrazne prispieva k zlepšeniu používateľskej skúsenosti. Capacitor je atraktívne riešenie pre vývojárov, ktorí chcú vytvárať výkonné a funkčne bohaté aplikácie pre viaceré platformy súčasne.

Aj keď bola Cordova v minulosti osvedčeným nástrojom pre hybridný vývoj, tím okolo Ionic frameworku v súčasnosti odporúča vývojárom prejsť na Capacitor, ktorý predstavuje modernejšiu a flexibilnejšiu alternatívu.

Napriek mnohým výhodám Capacitoru však Cordova naďalej nachádza svoje uplatnenie v určitých špecifických prípadoch – napríklad v existujúcich projektoch s rozsiahlymi závislosťami na Cordova pluginoch alebo v tímoch, ktoré sú už zabehnuté vo vývoji s týmto pluginom. V takýchto prípadoch môže byť Cordova naďalej vhodnou voľbou.

Výber medzi Cordova a Capacitor by preto nemal byť vedený len preferenciou, popularitou alebo aktuálnym trendom, ale mal by vychádzať z požiadaviek projektu. Medzi rozhodujúce faktory je možné zaradiť: **potreba špecifických pluginov, požadovaná úroveň podpory, znalosť technológií tímu, ale aj očakávania na výkonnosť a udržateľnosť riešenia**.[19]

1.3 Ionic vs Flutter

Ionic a Flutter sú dva frameworky, ktoré zdieľajú určitú víziu ako tvoriť aplikácie, primárne hybridné aplikácie. Ich cieľom je ponúknuť framework, ktorý funguje na rôznych mobilných platformách, webe či desktope. Zároveň svoje úsilie venujú aj tomu, aby aplikácia nielen fungovala, ale aj vyzerala dobre na všetkých platformách. Samozrejme, všetko je pokryté jedným kódom (single codebase).[21]

Flutter je projektom spoločnosti Google. Je to *open-source* framework s UI knižnicou na budovanie mobilných aplikácií, ktoré pôsobia ako natívne aplikácie. Rovnako ponúka tvorbu web a desktop aplikácií, samozrejme, všetko s jedným kódom. Ako programovací jazyk sa používa Dart.[22] V podstate všetko v súvislosti s Flutter má vlastný štandard ako ich vlastný

vykresľovací engine. [21]

Flutter požíva prekladač jazyka Dart, ktorý prekladá Dart kód do natívneho strojového kódu. Tento kód beží na cieľovej platforme mobilného zariadenia. Dart má vlastný vykresľovací engine. Tento engine zobrazuje UI priamo v mobilnej aplikácii. UI komponenty, ktoré Flutter používa, sú z jeho vlastnej knižnice UI widgetov. K natívnym funkciám mobilného zariadenia pristupuje cez knižnicu pluginov (podobne ako Ionic). Je možné využiť ich pluginy alebo si vytvoriť vlastný pomocou asynchrónnej komunikačnej vrstvy špecifickej pre Flutter. Táto vrstva sa používa na komunikáciu medzi klientom (používateľským rozhraním) a hostiteľom (operačným systémom zariadenia). Zobrazenie aplikácie vo webe (ako technický náhľad) je zložitejšie. Rovnaký prekladač ako pre mobilnú aplikáciu používa aj na generovanie JavaScriptu, ktorý sa vykresľuje do web prehliadača. Následne sa vykresľovač textu a grafický engine na základe Skia nahrádzajú vlastným vykresľovacím enginom. Po tejto zmene sa od základov znovu implementujú kľúčové prvky webového prehliadača, ako je výber textu, kopírovanie, vkladanie a prístupnosť. Každá aplikácia, ktorá je vytvorená s použitím Flutter, potom obsahuje všetky tieto nanovo vytvorené (znovu implementované) základné prvky.

Flutter tím sa rozhodol dosiahnuť svoj cieľ vytvoriť „prenosný UI framework, ktorý funguje všade“ tak, že pôjde svojou vlastnou cestou. Zvolenie programovacieho jazyka Dart, použitie vlastného renderovacieho enginu, špecifická natívna implementácia a vlastný webový framework na zobrazenie v prehliadači len podčiarkujú toto rozhodnutie vybudovať samostatnú architektúru, ktorá ich do určitej miery stavia do protikladu so zvyškom vývojárskeho sveta.

Ionic tím sa vydal opačnou cestou a to použitím stávajúcich technológií - webových technológií. Na nich je postavená celá logika frameworku ako aj UI komponenty. Používa teda štandardné jazyky webu ako HTML, CSS a JavaScript.

Rovnako ako Flutter ponúka možnosť s jedným kódom nasadiť natívnu mobilnú, webovú a natívnu desktopovú aplikáciu. Aplikácia spustená na mobilnom zariadení beží v natívnom kontajneri, ktorý používa Cordova alebo Capacitor. Tie predkladajú plný prístup k natívnym funkciám. Používateľské rozhranie (UI) beží vo WebView. Pokiaľ je aplikácia PWA, potom beží priamo vo web prehliadači zariadenia. UI komponenty, ktoré sú použité v aplikácii vytvorenej pomocou Ionicu, sú založené na štandarde Webových komponent. Fungujú

v ľubovoľnom webovom prehliadači a sú kompatibilné s akýmkoľvek JavaScriptovým frameworkom, ako React, Vue či Angular (ale aj bez frameworku). Ionic poskytuje knižnicu s viac než 100 UI komponentmi. Je možné ich prispôsobiť pomocou CSS podľa vizuálnej identity značky. Je taktiež možné vytvoriť vlastnú knižnicu s využitím Stencil – open-source kompilátor web komponent od tímu Ionic. Je možné v Ionic aplikácii spustiť akúkoľvek webovú UI komponentu alebo webovú knižnicu.[21]

Tabulka 1.2 Porovnanie Flutter a Ionic

Vlastnosť	**Flutter**	**Ionic**
Aplikácie	Hamilton, Google	IBM, T-Mobile
Dokumentácia	Veľmi dobrá	Veľmi dobrá
Možnosti nasadenia	Mobil, desktop, web	Mobil, desktop, Web, PWA
Natívne API	Knižnica natívnych pluginov používajúca Flutter natívne balíky	Knižnica natívnych pluginov používajúca Cordova/Capacitor
Programovací jazyk	Dart	HTML, CSS, JavaScript
Runtime	Vlastný grafický engine	Webový prehliadač
UI Elementy	Proprietárne widgety	Webové komponenty založené na štandardoch
Výkon na mobilnom zariadení	Excelentný	Veľmi dobrý
Výkon na mobilnom webe	Slabý	Excelentný

(zdroj: 21, 22)

Ako je možné vidieť v tabuľke 1.2 oba frameworky majú určité výhody a nevýhody. V konečnom dôsledku by výber riešenia mal vychádzať z toho, s akou filozofiou sa vývojársky tím zhoduje, kde a ako plánuje nasadiť svoju aplikáciu a aké zručnosti dnes vie alebo by sa chcel naučiť v budúcnosti. Rovnako treba zohľadniť doterajšie skúsenosti a časové obmedzenie a tomu prispôsobiť technológiu.[21]

2 BEZPEČNOSŤ

Zabezpečenie aplikácie, či už webovej alebo mobilnej, je jedným z kľúčových aspektov vývoja aplikácie. Je nutné brať na vedomie ochranu údajov a ich zabezpečenie v súlade s predpismi a reguláciami. Rovnako by mal byť kladený dôraz na dodržiavanie osvedčených postupov, ktoré sú v odvetví rokmi vyskúšané. Zabezpečenie je nutné aj z pohľadu biznisu a jeho vzťahu smerom k ich zákazníkom, ktorí doň vkladajú dôveru. Predchádza sa ním potenciálnym finančným škodám a škodám na reputácii.[23]

Webová a mobilná aplikácia majú určité zraniteľnosti spoločné, no je veľa takých, ktoré sú rozdielne, a preto treba ku každej aplikácii pristupovať samostatne. Hlavné rozdiely v **architektúre a prístupe k zabezpečeniu** je možné definovať nasledovne:[24]

Mobilná aplikácia zväčša beží v takzvanom *sandboxe*. Je to prostredie operačného systému, čo obmedzuje ich prístup k systémovým zdrojom a zvyšuje bezpečnosť. Aplikácia bežiaca v takomto prostredí má limitovaný prístup k dátam v mobilnom zariadení a tým znižuje riziko úniku dát. Ich distribúcia prebieha v overenom prostredí platforiem ako Google Play a Apple App Store, kde pred ich zverejnením dochádza k ich kontrole zo strany platformy. [25] Mobilné aplikácie môžu využívať bezpečnostné mechanizmy ako biometrická autentifikácia a šifrovanie pomocou hardvérových modulov (napríklad Android Keystore, iOS Keychain).[26]

Webová aplikácia je prístupná cez webové prehliadače a často jej zabezpečenie závisí od zabezpečenia samotného prehliadača a webového servera. Aktualizácie sú okamžité a nevyžadujú zásah používateľa, čo umožňuje rýchle opravy zraniteľností. Sú však náchylnejšie na útoky ako phishing, cross-site scripting (XSS) a SQL injection, najmä ak nie sú správne zabezpečené.[24]

2.1 Bezpečnostné hrozby

Vzhľadom na podobnosti v architektúre a spôsoboch komunikácie čelia webové aj mobilné aplikácie viacerým rovnakým bezpečnostným hrozbám.[27] Útoky, ktoré cielia na chyby v autentifikácii, správe relácií alebo na nesprávne zabezpečené API rozhrania, môžu mať vážne dôsledky bez ohľadu na platformu.[28]

Session hijacking / Krádež tokenu

Tento typ útoku je spojený so zneužitím Session ID (SID). SID reprezentuje jedinečné identifikačné číslo spojenia používateľa so serverom. SID posiela server prehliadaču klienta po jeho úspešnej autentifikácii. Je to reťazec znakov, ktorý je možné použiť v URL, v hlavičke HTTP ako cookie alebo ako inú časť hlavičky. Rovnako sa môže vyskytovať v tele HTTP požiadavky. [29]

Medzi typy útokov na SID patrí:

Útok hrubou silou je založený na odhade SID. Útočník sa snaží odhadnúť SID dokým sa mu to nepodarí. Tento spôsob útoku spolieha na jednoduchosť a odvoditeľnosť SID. Spôsob ochrany proti tomuto útoku závisí na pravidlách, ktoré by sa mali dodržiavať ako prevencia. Týmito pravidlami sú:

- **Unikátnosť SID** - každý používateľ a jeho session po prihlásení má mať unikátny SID (token).
- **Náhodnosť SID** - kedy by malo byť generované náhodne a teda proces generácie by nemalo byť možné napodobniť.
- **Nezávislosť SID** - a teda nemala by existovať nijaká previazanosť s nejakým údajom, ktorý má určitý význam.
- **Neodvoditeľnosť SID** - útočník by nemal byť schopný určiť spôsob generácie SID z niekoľkých už existijúcich (vygenerovaných).
- **Dĺžka SID** - hrá kľúčovú rolu, čím dlhšie SID je, tým dlhšie trvá ho uhádnuť.
- **Expirácia SID** - po určitom čase, odporúča sa 5 až 30 minút, by SID malo prestať byť validné a teda používateľ je nútený opätovne sa prihlásiť.[30]

Cross-site scripting (XSS) kedy útočník zneužije možnosť spustenia škodlivého JavaScriptu v prehliadači obete, čím získa session token (napr. z cookie) a zneužije ho na prevzatie prihlásenej relácie. Typickým príkladom je zaslanie odkazu s vloženým skriptom, ktorý odosiela token na server útočníka. Tento script sa spustí po kliknutí samotného užívateľa.[29]

Session side jacking kedy je ukradnuté SID z paketov prúdiacich v sieti. Tieto packety si útočník zobrazuje pomocou softvéru. Avšak, útočník potrebuje prístup do siete uží-

vateľa, takže tento typ útoku je charakteristický pre verejné Wi-Fi siete. Odporúčanou obranou je použitie SSL/TLS kódovanie na všetku komunikáciu. **HTTPS protokol** zaisťuje šifrovanú komunikáciu SSL/TLS certifikátom po celý čas používateľovej session. Toto zabezpečenie chráni SID tak, že ho útočník nedokáže ukradnúť počas komunikácie medzi serverom a klientom. HTTPS protokol chráni pred Session side jacking útokom.

Ukradnutie SID pomocou malware kedy si používateľ nevedomky, nechcene infikuje svoje zariadenie. Tento malware odchytáva cookies alebo k nim môže až priamo pristupovať.[30]

Slabá autentifikácia/autorizácia

Zraniteľnosti v autorizácii znamenajú, že používateľ získa prístup k dátam alebo funkciám, na ktoré nemá oprávnenie. Medzi najčastejšie zraniteľnosti tohto typu patrí:

Broken Access Control (narušená kontrola prístupu) označuje situáciu, kedy používateľ koná mimo svojich právomocí. To typicky vedie k odhaleniu informácií, ku ktorým nemá prístup a môže dokonca spôsobiť ich modifikáciu či zničenie. Rovnako môže využívať funkcie, ku ktorým nemá právomoci a nemajú mu byť prístupné. Medzi bežné zraniteľnosti tohto typu radíme:

- **Princíp najmenšieho privilégia** je porušený. Prístup je dostupný komukoľvek, i keď by mal byť udelený len konkrétnym používateľom, či roliam.
- **Úprava URL alebo nástrojov na zmenu API požiadaviek** pre prístup k častiam aplikácie, na ktoré nemá právo (napríklad do administrátorskej časti z časti pre bežného používateľa).
- **Prístup k dátam iného používateľa** v zmysle prezerania či upravovania tak, že použije ID tohto užívateľa napríklad zmenou API požiadavky.
- **Pristupovanie k API bez potrebných prístupových kontrol** pre požiadavky ako POST, PUT či DELETE.
- **Získanie vyšších oprávnení** správaním sa ako používateľ bez skutočného prihlásenia či ako administrátor bez reálneho oprávnenia (skutočným oprávnením je bežný používateľ).

- **Nesprávne nastavenie CORS** kedy dochádza k prístupu z neautorizovaných, či nedôveryhodných zdrojových domén.
- **Vynútené prehliadanie** častí aplikácie ako neoverený používateľ alebo častí s určitými privilégiami, ktoré používateľ nemá.

Je niekoľko overených spôsobov, ako týmto zraniteľnostiam predísť. *Riadenie prístupu je účinné len v dôveryhodnom serverovom kóde alebo v bezserverovom API, kde útočník nemôže upraviť kontrolu prístupu ani metadáta*. Používajú sa tieto metódy:

- **S výnimkou verejných zdrojov platí zásada:** odmietnuť prístup.
- **Jednotná implementácia kontroly prístupu**, ktorá je následne použitá v aplikácii.
- **Minimalizácia CORS**.
- **Presadzovanie vlastníctva záznamov** namiesto toho, aby automaticky predpokladali, že používateľ môže vytvárať, čítať, upravovať alebo mazať akýkoľvek záznam.
- **Obmedzenia prístupu alebo validačné podmienky** by mali byť súčasťou doménových modelov.
- **Ochrana dát** tak, že webový server by nemal umožňovať prehliadanie obsahu priečinkov a v koreňovom adresári aplikácie by sa nemali nachádzať citlivé súbory ako .git, zálohy či dočasné konfigurácie.[31]

Zlyhania identifikácie a autentifikácie vedú k obrovským problémom spôsobených útokom cieleným na autentifikáciu. Je kľúčové identifikovať a autentifikovať používateľa. Rovnako kľúčová je správa *session*. Zraniteľné miesta je možné identifikovať ak aplikácia povoľuje alebo nie je chránená voči:

- **Credential stuffing** – útok, pri ktorom útočník systematicky skúša veľké množstvo skutočných kombinácií používateľských mien a hesiel, často uniknutých z iných služieb.
- **Slabé, neúčinné procesy obnovy zabudnutých hesiel** ako odpovede založené na znalostiach (nemožno ich zabezpečiť).

- **Slabá ochrana hesla** kedy je heslo uložené ako čistý text alebo je nedostatočne hashované či šifrované.
- **SID v URL**.
- **Znovupoužitie SID** po úspešnom prihlásení.
- **Chýbajúce zneplatnenie SID** po odhlásení či po určitom čase, kedy používateľ už nie je aktívny.

Aj týmto zraniteľnostiam sa dá predísť nasledovaním postupov:

- Nenasadzovať aplikáciu s predvolenými povereniami, najmä pre administrátorov.
- Kontrola slabých hesiel a nastavenie štandardov na vytvorenie silného hesla.
- Správca *session* má byť bezpečný, vstavaný a na strane servera. Po prihlásení generuje nové náhodné SID, ktoré sa nenachádza v URL adrese a je bezpečne uložené. Po odhlásení, nečinnosti, či uplynutí doby životnosti *session* je automaticky zneplatnený.[32]

API zraniteľnosti

Použitie a teda volanie API je častým prístupom ako vo webových, tak mobilných aplikáciách. Práve pre jeho časté použitie je rovnako vystavené útokom. Medzi najčastejšie zraniteľnosti patrí:

Nesprávne overenie oprávnenia k jednotlivým objektom nastáva vtedy, keď používateľ pristupuje aj k objektom, na ktoré nemá oprávnenie. API endpoint, ktorý dostáva ID (napríklad používateľa), by mal rovnako kontrolovať, že ID prihláseného používateľa sa zhoduje s ID, s ktorým je daný endpoint volaný.[33]

Nedostatočná kontrola oprávnení pri jednotlivých atribútoch objektu kedy používateľ môže meniť atribúty objektu, na ktoré nemá právo (napríklad z roly USER na rolu ADMIN), alebo tieto atribúty môže vidieť.[34]

Injekčné útoky

Injekčné útoky fungujú na princípe vloženia (injekcie) škodlivého kódu či príkazu do vstupných polí alebo dát aplikácie. V prípade, že aplikácia tieto vstupy nesprávne spracuje, dôjde

k ich vykonaniu, čo môže viesť k získaniu citlivých údajov, obídeniu prístupu alebo ovládnutiu systému. Medzi najbežnejšie injekčné útoky patrí:

SQL Injection (SQLi) je zraniteľnosť, kedy útočník vloží škodlivý SQL dotaz do databázy cez vstupné pole, či parameter aplikácie. Tento SQL dotaz sa najčastejšie snaží získať citlivé informácie uložené v databáze. Rovnako tento SQL dotaz môže meniť dáta v databáze, či ju rovno zmazať.

Command Injection je relevantný pre aplikácie, ktoré interagujú s operačným systémom, na ktorom bežia. Vložením škodlivého príkazu do vstupného poľa aplikácie, ktorý je následne vykonaný príkazovým riadkom. To potom môže viesť k neautorizovaným prístupom, úniku dát či dokonca k úplnému prevzatiu kontroly nad systémom.

Injekcia kódu kedy vložený škodlivý kód je vykonaný serverom.

Aj voči týmto typom útoku sa dá brániť nasledovaním určitých overených postupov. Do týchto postupov radíme:

- **validácia vstupov od používateľa**,
- **limitovanie povolenia prístupu k dátam**,
- **ochrana citlivých údajov**,
- **kontrola vstupov do aplikácie**.[35]

2.1.1 Bezpečnostné hrozby špecifické pre mobilné aplikácie

Mobilné aplikácie sú populárne a táto ich popularita čím ďalej, tým viac rastie. S tým sú spojené útoky na ich časté zraniteľnosti.

Nesprávne zaobchádzanie s prihlasovacími údajmi

Táto zraniteľnosť nastáva vtedy, keď sú prihlasovacie údaje pevne zapísané v kóde alebo nesprávne používané. Toto vystavuje mobilnú aplikáciu automatizovaným útokom pomocou verejne dostupných alebo vlastnoručne vytvorených nástrojov. Autentifikačné údaje tak môžu byť zneužité. Akonáhle útočník nájde túto zraniteľnosť, môže vďaka nim skúsiť

pristupovať k citlivým údajom. To môže viesť k úniku dát, strate používateľského súkromia, podvodným aktivitám alebo prístupu k funkcionalitám dostupným len administrátorským používateľom. Všetky tieto potenciálne hrozby majú dosah na biznis, ako poškodenie reputácie, odcudzenie informácií, prístup k dátam či podvodu.

Existuje niekoľko znakov, podľa ktorých je možné identifikovať túto zraniteľnosť v mobilnej aplikácii. Sú nimi:

Autentifikačné údaje pevne zapísané v kóde: mobilná aplikácia má tieto údaje pevne zapísané v kóde aplikácie alebo je možné ich nájsť v nejakom konfiguračnom súbore.

Nezabezpečený prenos autentifikačných údajov: mobilná aplikácia prenáša nezašifrované autentifikačné dáta alebo ich prenáša nezabezpečenými kanálmi.

Nezabezpečené uloženie autentifikačných údajov: mobilná aplikácia ukladá tieto údaje do nezabezpečeného úložiska zariadenia.

Slabá autentifikácia: závisí na slabých protokoloch alebo umožňuje jej jednoduché obídenie.

Obrana a podchytenie týchto možných zraniteľností by mala nasledovať tieto body:

- Šifrovanie autentifikačných údajov počas prenosu.
- Neukladať autentifikačné údaje v zariadení bez zabezpečenia, ale použiť bezpečné úložiská, alebo odvolateľné prístupové tokeny.
- Implementácia silných autentifikačných protokolov.
- Pravidelne aktualizovať použité API kľúče a tokeny.[36]

Nezabezpečené ukladanie dát

Nezabezpečené ukladanie dát v mobilnej aplikácii predstavuje zraniteľnosť voči viacerým typom útokov, ktoré môžu byť zneužité. Medzi útoky patrí napríklad neoprávnený prístup k súborovému systému zariadenia – či už fyzicky alebo na diaľku, zneužitie slabej alebo chýbajúcej šifrovacej ochrany, zachytávanie prenosu dát, ako aj použitie škodlivého softvéru

alebo aplikácií nainštalovaných na zariadení. Ďalším rizikovým faktorom sú rootnuté (zariadenie so získanými administrátorskými právami - Android) alebo jailbreaknuté (iOS zariadenie s odstránenými obmedzeniami systému) zariadenia. Takéto zariadenia umožňujú obísť bezpečnostné opatrenia operačného systému. Riziko navyše zvyšujú aj techniky sociálneho inžinierstva, ktoré môžu viesť k tomu, že používateľ nevedomky poskytne prístup k svojim dátam alebo umožní manipuláciu s aplikáciou.

Z technického hľadiska má táto zraniteľnosť dopady, ktoré podlamujú celkovú bezpečnosť a funkcionalitu mobilnej aplikácie. Týmito dopadmi sú:

- únik dát,
- zverejnené a ohrozené používateľské účty,
- falšovanie dát,
- nepovolený prístup k dátam aplikácie,
- nesúlad s nariadeniami.

Tabuľka 2.1 sumarizuje bežné prejavy zraniteľnosti pri ukladaní dát v mobilnej aplikácii spolu s návrhmi na ich zmiernenie/riešenie.[37]

Tabulka 2.1 Zraniteľnosti pri ukladaní dát v mobilnej aplikácii s návrhmi na ich zmiernenie/riešenie

Príznak zraniteľnosti	**Opatrenie**
Nedostatok prístupových práv	Zaviesť prístupové kontroly
Nevalidované vstupy	Vstupy validovať a sanitizovať
Neúmyselné odhalenie údajov	Zakázať logovanie citlivých údajov, odstrániť debug režimy v produkcii
Slabá správa session	Používať náhodné SID, dlžku platnosti session, zneplatnenie po odhlásení
Slabé/chýbajúce šifrovanie	Použiť silné šifrovanie (napr. AES-256), správne spravovať kľúče

(zdroj: 37)

Nezabezpečené ukladanie dát vytvára priestor pre útoky — od priamej extrakcie uložených údajov až po zachytávanie citlivých informácií počas ich prenosu. Tento fakt zdôrazňuje dôležitosť nasadenia silného šifrovania, bezpečných prenosových protokolov a komplexných bezpečnostných opatrení pri vývoji mobilných aplikácií.[37]

Reverzné inžinierstvo

V skutočnosti je každá mobilná aplikácia náchylná na reverzné inžinierstvo. Avšak, niektoré aplikácie sú náchylné viac ako iné. *Kód napísaný v jazykoch alebo frameworkoch, ktoré umožňujú dynamickú introspekciu počas behu aplikácie (napr. Java, .NET, Objective-C, Swift), je obzvlášť náchylný na reverzné inžinierstvo.* Aplikácia je považovaná za zraniteľnú, ak je pomocou týchto nástrojov pomerne jednoduché pochopiť logiku jej riadenia (control flow), reťazcové tabuľky (string table) a generovaný pseudokód alebo priamo zdrojový kód.

Reverzným inžinierstvom sa útočník snaží získať informácie o backend serveroch a kryptografických šifrách/konštantách, ukradnúť intelektuálne vlastníctvo (kód aplikácie), napadnúť backend servery alebo získať toľko informácií, že je možné vykonať modifikáciu kódu.

Voči reverznému inžinierstvu je možné sa brániť niekoľkými spôsobmi:

- určením, na ktoré časti kódu aplikovať obfuskáciu,
- násjť kompromis medzi mierou obfuskácie a výkonom aplikácie,
- odolnosť voči vráteniu pôvodného kódu po obfuskácí,
- aplikovať obfuskáciu na reťazové tabuľky, metódy.[38]

2.1.2 Bezpečnostné hrozby špecifické pre webové aplikácie

Webové aplikácie, rovnako ako mobilné, podliehajú určitým zraniteľnostiam.

Cross-site scripting (XSS)

XSS je bezpečnostné zneužitie, ktorým útočník vloží škodlivý kód do klientskej časti aplikácie a následne je tento kód spustený. Zvyčajne sa vyskytuje, keď dáta vchádzajú do aplikácie cez nedôveryhodný zdroj alebo je dynamický obsah poslaný webovému používateľovi bez validácie a kontroly na škodlivý obsah.[39] Na spustenie tohto škodlivého kódu je potrebné spustiť skript alebo spustením celého bloku kódu, o ktorom prehliadač nevie, že sa nejedná o originálny kód s pôvodným zámerom.[40]

XSS útoky vieme rozdeliť do niekoľkých kategórií, ako:

Uložené XSS útoky, kedy je vložený skript natrvalo uložený na cielenom serveri. Tento script je potom odoslaný používateľovi zo servera po tom, čo prehliadač žiada o dáta.[39]

Sú nebezpečné, pretože sa vykonávajú na úrovni servera a teda robia zmeny priamo v aplikácii.[40]

Neuložené (reflektované) XSS útoky, kedy script nie je permanentne uložený, ale sám používateľ je oklamaný a klikne na škodlivý link alebo prehliada škodlivú stránku. Script potom prichádza až do zraniteľnej webovej aplikácie. Webový server následne odráža tento script do prehliadača používateľa. Môže sa to prejaviť vo forme chybovej správy, výsledku vyhľadávania a podobne. Prehliadač potom spustí kód v domnienke, že je z dôveryhodného servera (dôveryhodný, pretože prehliadač predpokladá, že už s ním používateľ prišiel do styku).[39] Tento typ XSS útoku ovplyvňuje odozvu servera.[40]

XSS útoky založené na DOM, kedy je upravené prostredie DOM v používateľovom prehliadači. Stránka sama o sebe nie je zmenená, ale kód stránky na strane klienta beží neočakávaným spôsobom.

Škody a dopady XSS kódu sú rôzne. Od menej závažných ako zobrazovanie reklám na stránkach, kde nie sú zapnuté, až po zaslanie prihlasovacích údajov zo stránok priamo útočníkovi. Ďalším dopadom je zneužitie na ťaženie kryptomien.

Základom ochrany proti XSS útokom je identifikácia rizík. Príkladom môže byť správne nastavenie domén a adries, z ktorých môže byť sťahovaný externý obsah. Ďalšou ochranou je napríklad vyvarovanie sa použitiu `innerHtml`, ktorý nerieši problémové symboly alebo ošetrenie vstupov používateľov.[40]

Falšovanie žiadostí medzi stránkami (CSRF)

Tento útok je založený na tom, že posiela požiadavky na backend webovej aplikácie bez vedomia používateľa. Napríklad požiadavka na banku používateľa na výber peňazí. Na spustenie CSRF útoku sa často používa škodlivý XSS kód.

Voči tomuto typu útoku je možné sa brániť nasledovaním určitých zásad, ako:

GET endpointy nemenia dáta aplikácie - všetky žiadosti na API, ktoré menia dáta by mali byť POST žiadosti (alebo iné HTTP metódy).

Použitie CSRF tokenu - ktorý je v rámci session unikátny a predáva ho server prehliadaču. Tento token sa používa pri zmene dát ako napríklad POST žiadosť. Server porovná

poslaný token s tým, ktorý má uložený pre danú session. Ak sa tokeny nezhodujú, žiadosť je odmietnutá.

CSRF token nesmie byť odvoditeľný/predvídateľný - na tomto stojí tento typ ochrany a práve preto by mal byť CSRF token pregenerovaný pri každom prihlásení používateľa. [39]

2.2 Autentifikácia pomocou biometriky v mobilnom zariadení

Biometrická autentifikácia je v dnešnej dobe jedna z najpopulárnejších metód. Je to bezpečnostná metóda autentifikácie pomocou biometrických údajov, ako odtlačok prsta, sken tváre alebo iných unikátnych biologických charakteristík človeka.[15]

Biometrika je fyzická charakteristika, ktorú je možné použiť na identifikáciu jednotlivca, a teda je unikátna pre každého jednotlivca.[41]

Autentifikácia pomocou biometriky môže mať viacero spôsobov, ako:

Rozpoznanie tváre používa unikátne znaky tváre jednotlivca na jeho identifikáciou. Je to jeden z bežných spôsobov, ktorý sa používa v mobilnom zariadení.

Rozpoznanie odtlačku prsta identifikuje používateľa na základe jeho jedinečného odtlačku prsta. Tento spôsob je taktiež bežnou metódou autentifikácie používateľa.[42] Odtlačok prsta sa sníma skenerom telefónu. Tento skener používa optický obraz a kapacitné snímanie, ktoré zistí, či sa jedná o skutočný prst. Na používanie autentifikácie odtlačkom prsta musí prebehnúť prvotné nastavenie, kedy skener zachytí odtlačok. Vzor odtlačku je vytvorený určením tmavých a svetlých oblastí, tie sú tvorené ryhami odtlačku. Pri autentifikácii sa odtlačok práve skenovaného prsta porovná s tým, ktorý bol urobený pri nastavovaní.[41]

Rozpoznanie oka je spôsob identifikácie na základe jedinečného vzoru dúhovky alebo sietnice. Nie je bežne používaný, pretože je náročnejší na implementáciu. Sken dúhovky vyžaduje zdroj infračerveného svetla, kameru (vidí infračervené svetlo), a minimálne svetelné znečistenie na zaistenie presnosti. Je to jeden z najpresnejších spôsobov autentifikácie biometrikou a používa sa primárne v miestach, ktoré vyžadujú vysokú bezpečnosť, ako jadrové výskumné zariadenia.

Rozpoznanie hlasu používa k overeniu totožnosti tón, výšku a frekvenciu, ktoré sú pre jednotlivca jedinečné. Používa sa najčastejšie v call centrách bánk.

Pre mobilné zariadenia sú najbežnejším spôsobom autentifikácie jednotlivca **rozpoznanie odtlačku prsta** a **tváre**.[42]

2.2.1 Výhody a potenciálne riziko

Biometrická autentifikácia zvyšuje súkromie a bezpečnosť mobilného zariadenia a samotných aplikácií v ňom. Medzi jej výhody zaraďujeme:

Robustné overovanie identity, kedy biometrika poskytuje spôsob overenia toho, že osoba je tým, za koho sa vydáva. Vďaka tomu, že biometrické znaky sú jedinečné, existuje výrazne vyššia záruka presného overenia.[41]

Vylepšená autentifikácia práve preto, lebo biometrické identifikátory sú také jedinečné, že výrazne znižujú možnosť neoprávneného prístupu. To prináša ubezpečenie, že zločinci už nemajú prístup k cudziemu zariadeniu alebo informáciám s ukradnutými heslami či PIN kódmi.[41] Zaisťuje koncovým používateľom zvýšenú úroveň istoty práve vďaka jej jedinečnosti. Kybernetický útočník môže poznať heslo používateľa alebo odpoveď na jeho bezpečnostnú otázku. Avšak, neexistuje spôsob, ako by mohol duplikovať odtlačok prsta alebo sken dúhovky.[42]

Viacfaktorová alebo multimodálna biometrická autentifikácia je silou biometrie pri použití viacerých autentifikačných faktorov. Je to metóda využívajúca rôzne typy biometrických údajov (rozpoznávanie tváre, skenovanie odtlačkov prstov, spájanie biometrických údajov s tradičnými overovacími faktormi, ako sú heslá). Je to vlastne vrstvená obrana, ktorú je omnoho ťažšie prelomiť.[41]

Jednoduchá na používanie pre používateľov vďaka jej rýchlosti. Je jednoduchšie naskenovať tvár či priložiť prst k snímaču, ako písať dlhé heslo so špeciálnymi znakmi každý raz pri prihlasovaní.[42]

Detekcia podvodu pri autentifikácii. V dnešnej dobe je takmer nemožné replikovať či ukradnúť biometriku. Existuje asi 1 ku 64 miliárd šancí, že niekto má rovnaký odtlačok prsta.[42]

Avšak, ani biometrika nie je stopercentná a sú s ňou spojené bezpečnostné riziká a zraniteľnosti. Medzi tieto zraniteľnosti sa radia:

Riziká súkromia, pretože autentifikácia biometrikou ukladá citlivé údaje o jednotlivcovi, ako odtlačok prsta. To môže viesť k ukradnutiu týchto citlivých dát a následnému odcudzeniu identity.[41] Firmy, inštitúcie či vlády ukladajú citlivé biometrické údaje, ktoré môžu byť z týchto databáz odcudzené. K týmto údajom by sa malo pristupovať so zvýšenou bezpečnosťou.[42]

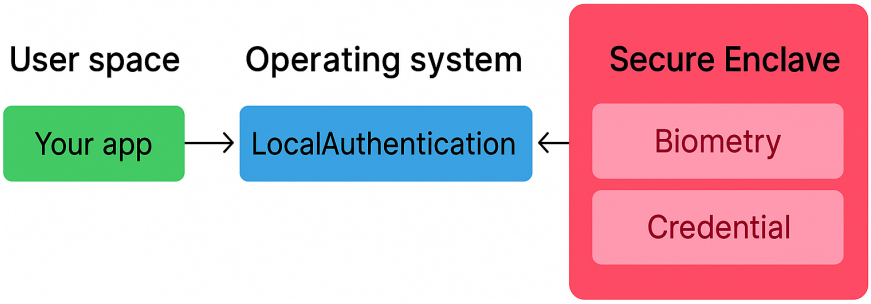
Nesprávna identifikácia pomocou biometriky, kedy používateľ nie je identifikovaný, pretože má napríklad zašpinený prst. Táto záležitosť vie byť neprívetivá voči používateľom, ktorí musia očistiť svoje prsty, alebo skúšať prihlásenie viackrát.[41] Rovnako pri autentifikácii pomocou rozpoznania tváre dochádza k neúspešným pokusom. Počas nastavenia tvárového rozpoznania používateľ zosníma svoju tvár, napríklad z určitého uhla. Avšak, pri autentifikácii môže používateľ mať na sebe okuliare, make-up či sa usmievať, čo sa nemusí zhodovať s prvotným skenom tváre, s ktorým sa autentifikácia porovnáva.[42]

Zaujatosť voči farbe pleti alebo pohlaviu môže byť problémom pri autentifikácii biometrikou. Veľa biometrických systémov je trénovaných na bielych mužoch a môžu mať problém s inou farbou pleti alebo pohlavým.[42]

2.2.2 Apple

Firma Apple predstavila autentifikáciu pomocou odtlačku prsta - **Touch ID** v roku 2013. O pár rokov neskôr, v roku 2017, predložili **Face ID** - autentifikáciu rozpoznaním tváre.[43]

Biometrickú autentifikáciu v dnešnej dobe používa mnoho používateľov zariadení od spoločnosti Apple. Pre maximalizovanie úrovne bezpečnosti aplikácia nikdy nemá prístup k samotným údajom použitým pri autentifikácii, napríklad odtlačky prstov nikdy nie sú prístupné pre aplikáciu. **Secure Enclave** spravuje tieto citlivé dáta. Je to hardvérový bezpečnostný procesor, ktorý je izolovaný od zvyšku systému (vrátane operačného systému). Proces autentifikácie koordinuje framework v spolupráci so Secure Enclave, kedy po vykonaní operácie vývojár dostane len odpoveď, či bola autentifikácia úspešná alebo nie. Samotný proces je možné vidieť na obrázku 2.1.



Obrázek 2.1 Proces autentifikácie v Apple zariadeniach

(zdroj: 44)

Face ID

Face ID umožňuje používateľom bezpečné odomknutie podporovaných zariadení Apple. Pomocou technológie mapovania presnej geometrie tváre používateľa, ktorú využíva systém kamery TrueDepth je možné vykonať bezpečnú a intuitívnu autentifikáciu. Face ID využíva neurónové siete na rozpoznávanie pozornosti, porovnávanie tvárí a ochranu proti podvodom (antispoofing). Táto technológia dokáže zariadenie odomknúť aj používateľovi nosiacemu rúško, pokiaľ to zariadenie podporuje. Systém Face ID sa automaticky prispôsobuje zmenám vo vzhľade používateľa a zároveň starostlivo chráni súkromie a bezpečnosť biometrických údajov.

Pri použití Face ID kamera TrueDepth automaticky vyhľadáva tvár používateľa pri prebudení zariadenia (zdvihnutím alebo ťuknutím na obrazovku). Rovnako automaticky vyhľadáva tvár pri autentifikácii na zobrazenie prichádzajúceho oznámenia alebo pri požiadavke na autentifikáciu zo strany podporovanej aplikácie. Akonáhle je tvár rozpoznaná, Face ID overuje pozornosť a zámer odomknutia tým, že zisťuje, či má používateľ otvorené oči a či smeruje svoj pohľad na zariadenie.

Po tom, ako kamera TrueDepth overí, že je prítomná pozorná tvár používateľa, vykoná projekciu tisícok infračervených bodiek, aby vytvorila hĺbkovú mapu tváre a zachytí 2D infračervený obraz. *Tieto údaje sa použijú na vytvorenie sekvencie 2D obrazov a hĺbkových máp, ktoré sú digitálne podpísané a odoslané do Secure Enclave. Na ochranu pred digitálnymi a fyzickými podvodmi systém náhodne mení poradie snímaných obrazov a projekcií infračerveného vzoru, pričom používa náhodný vzor špecifický pre každé zariadenie. Časť Secure Neural Engine, chránená v rámci Secure Enclave, spracuje tieto dáta na matematickú reprezentáciu, ktorú následne porovnáva s uloženými biometrickými údajmi tváre. Ti-*

eto uložené údaje tvoria matematickú reprezentáciu tváre používateľa, zachytenú v rôznych pózach.[45]

Touch ID

Touch ID technológia číta údaje o odtlačkoch prstov. Snímač číta údaje z viacerých uhlov a v priebehu používania tejto technológie dopĺňa informácie o odtlačku prsta používateľa. Rozširuje mapu odtlačku pri každom jeho použití tak, že sú identifikované ďalšie prekrývajúce sa uzly.

Po naskenovaní odtlačku prsta je tento sken odoslaný do Secure Enclave. Toto odoslanie prebieha v kanály, ktorý je zabezpečený. Na analýzu sa používa vektorová reprezentácia odtlačku, rastrová sa dočasne ukladá do šifrovanej pamäte v Secure Enclave a neskôr je zahodená. Pri registrácii odtlačku prsta na zariadení Apple sa používa technika nazývaná subdermálne mapovanie uhlov toku línií (angl. *subdermal ridge flow angle mapping*). Táto metóda sníma charakteristiky pod povrchom kože a prebieha ako stratový proces, pri ktorom sa zámerne vynecháva detailná informácia o jednotlivých bodoch odtlačku (tzv. finger minutiae data), ktoré by inak umožnili spätnú rekonštrukciu celého odtlačku. Výsledkom spracovania je mapa uzlov, ktorá je uložená v šifrovanej forme a slúži ako predloha pre budúce porovnanie pri autentifikácii. Tento šifrovaný záznam môže byť dešifrovaný a použitý výhradne v rámci bezpečnostného modulu Secure Enclave, ktorý je izolovaný od operačného systému a ostatných častí zariadenia. Uchovávané biometrické údaje nie sú spojené s identitou používateľa. Maximálna ochrana sa zaisťuje tým, že biometrické dáta nikdy neopustia zariadenie, spoločnosť Apple nimi nedisponuje a neukladajú sa do záloh. Takýto prístup zabezpečuje, že biometrické údaje zostávajú plne pod kontrolou používateľa a minimalizuje sa riziko ich zneužitia.[45]

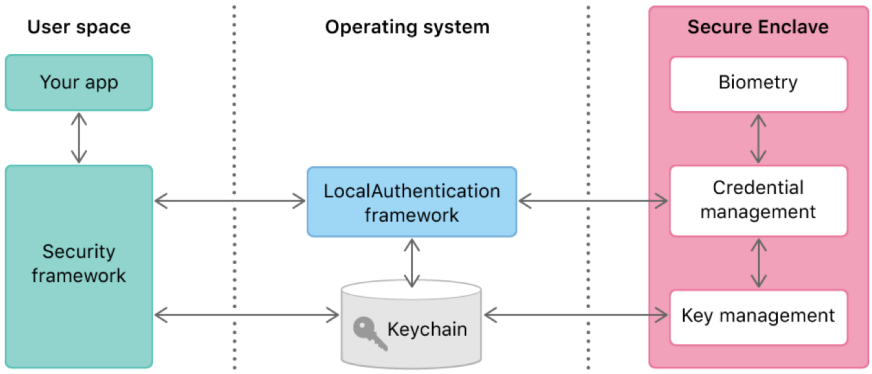
Keychain

Keychain je zabezpečené systémové úložisko pre citlivé dáta ako meno a heslo, token na prihlásenie, čísla kreditných kariet a podobne. Zabezpečené je prístupovým kódom do zariadenia, takže má doň prístup len používateľ, ktorý sa úspešne autentifikuje.[46]

V kontexte mobilných aplikácií sa keychain používa na ukladanie mena a hesla alebo tokenov určených k autentifikácii používateľa do aplikácie. Keychain položka je šifrovaná dvoma rôznymi AES-256-GCM kľúčmi. Samotný keychain je implementovaný ako SQLite

databáza uložená v súborovom systéme.[47]

Pri používaní keychain v mobilnej aplikácii na autentifikáciu používateľa pomocou biometriky ukladáme do keychain údaje, ktorými sa používateľ autentifikuje do aplikácie. Pristupovanie k keychain položkám sa používajú keychain služby v rámci Security frameworku. V prípadoch, keď je potrebné vykonať autentifikáciu používateľa, využíva Keychain Services framework LocalAuthentication, ktorý zobrazí príslušné rozhranie pre overenie totožnosti. Samotnú autentifikáciu vykonáva Secure Enclave, napríklad porovnaním aktuálneho odtlačku prsta s uloženými biometrickými vzormi. Secure Enclave následne vráti výsledok autentifikácie vo forme úspechu alebo neúspechu, pričom tento výsledok rozhoduje o povolení alebo zamietnutí prístupu k požadovaným údajom v Keychaine. Žiadna softvérová vrstva v používateľskom priestore ani samotný operačný systém nemá prístup k biometrickým údajom, ako sú napríklad uložené odtlačky prstov. Týmto spôsobom sa zabezpečuje maximálna ochrana súkromia a bezpečnosti používateľa. Na obrázku 2.2 je zobrazený nákres autentifikácie používateľa do aplikácie s využitím keychain a biometrie.[48]



Obrázek 2.2 Proces autentifikácie v Apple zariadeniach s keychain a biometrikou

(zdroj: 48)

2.2.3 Android

Android používa koncept kryptografických kľúčov chránených autentifikáciou používateľa, ktorý vyžaduje komponenty **úložisko kryptografických kľúčov a poskytovateľ kryptografických služieb** a **používateľské autentifikátory**.

Úložisko kryptografických kľúčov a poskytovateľ kryptografických služieb ukladá kryptografické kľúče a poskytuje štandardné kryptografické operácie nad týmito kľúčmi. Har-

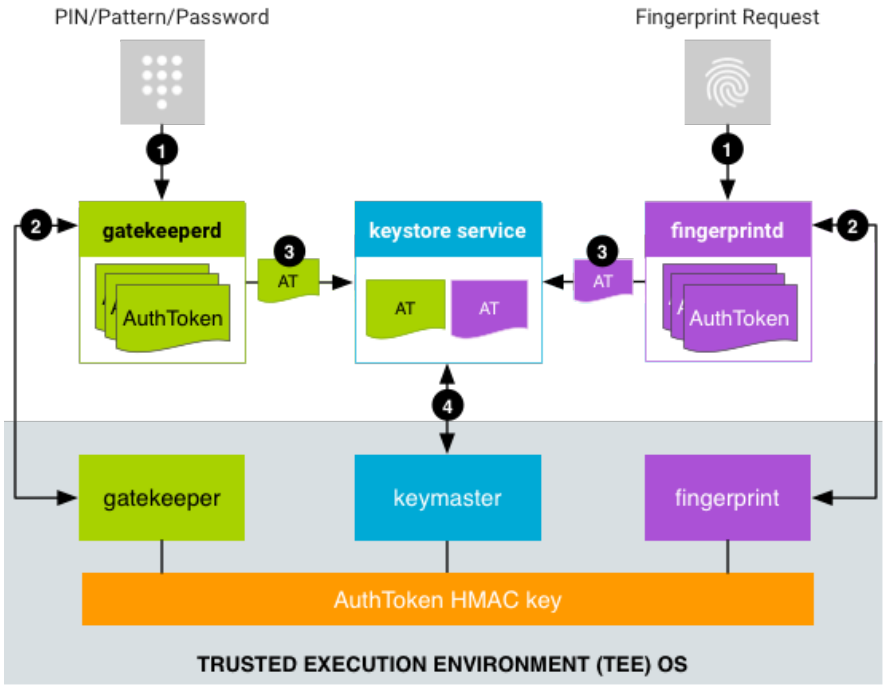
dvérom podporovaný Keystore a Keymaster, ktoré poskytujú kryptografické služby ako napríklad bezpečné ukladanie kľúčov. Toto riešenie môže zahŕňať použitie Trusted Execution Environment (TEE) alebo Secure Element (SE).

Používateľské identifikátory overujú prítomnosť používateľa a/alebo úspešnej autentifikácie. Android podporuje Fingerprint pre autentifikáciu cez odtlačok prsta a zariadenia s Android verzia 9 a vyššie môžu používať BiometricPromp - jednotné rozhranie na integráciu odtlačkov prstov a ďalších biometrických metód. Komunikujú so službou Keystore cez autentifikovaný kanál.

Komponenty Gatekeeper, Fingerprint a Biometric spolupracujú so službou Keystore a ďalšími súčasťami systému, aby podporili používanie hardvérom podporovaných autentifikačných tokenov (AuthTokens).

Autentifikácia

Používateľ sa môže prihlásiť PINom, vzorom či biometrikou, kedy zdieľajú spoločný tajný kľúč, ktorý používajú na vzájomné overovanie správ, ktoré si medzi sebou posielajú. Proces autentifikácie je zobrazený na obrázku 2.3.



Obrázek 2.3 Proces autentifikácie v Android zariadeniach

(zdroj: 26)

Podľa obrázka 2.3 je proces autentifikácie nasledovný:

1. Používateľ zadá heslo/PIN/vzor a táto požiadavka ide do Gatekeeper. Pri biometrike požiadavka ide na fingerprintd alebo iný daemon.

2. Príslušný daemon (napr. gatekeeperd alebo fingerprintd) odošle údaje do Trusted Execution Environment (**TEE**) kedy Gatekeeper dostane hash PIN-u/vzoru/hesla a Fingerprint komponenta dostane dáta o odtlačku prsta.

3. Ak je autentifikácia v TEE úspešná TEE komponenta (Gatekeeper alebo Fingerprint) vytvorí a podpíše **AuthToken** (pomocou HMAC kľúča)

4. Daemon odovzdá AuthToken do **keystore service** a tá odovzdá AuthToken do **Keymastera**, ktorý overí podpis a rozhodne o uvoľnení kryptografických kľúčov na základe časovej platnosti autentifikácie.[26]

V prípade mobilnej aplikácie, ktorá používa biometriku ako autentifikačný nástroj, sa vyvolá rovnaký proces ako na obrázku 2.3, avšak mobilná aplikácia nemá priamy prístup k AuthToken ani biometrickým dátam, dostane len informáciu o úspešnej alebo neúspešnej autentifikácii.[49]

Odtlačok prsta

Používatelia Androidu môžu zaregistrovať niekoľko odtlačkov, ktorými sú schopní vykonať autentifikáciu. Na tento účel slúži Fingerprint HIDL (*Hardware Interface Definition Language*). Prepojuje knižnicu špecifickú pre výrobcu a hardvér odtlačku prsta (napríklad s čítačkou odtlačkov prstov). Senzor odtlačku prsta čaká na dotyk - používateľ položí prst na tento senzor. Knižnica špecifická pre výrobcu rozhodne, či sa odtlačok zhoduje so sadou zaregistrovaných odtlačkov prstov. Ak sa nájde zhoda, tento odtlačok je poslaný do služby `FingerprintService`.

Na obrázku 2.4 je znázornený proces, ako prebieha autentifikácia pomocou odtlačku prsta. Tento proces má nasledujúce kroky:

1. Mobilná aplikácia komunikuje prostredníctvom `BiometricManager` so systémovými službami.

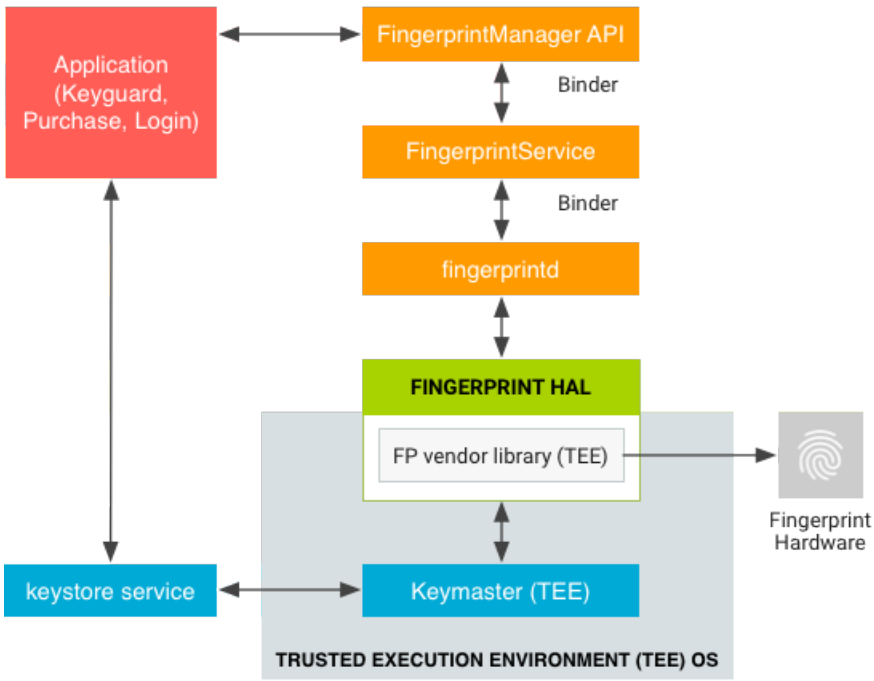
2. `FingerprintService` a fingerprintd (systémový démon) zabezpečujú komunikáciu s Fingerprint HAL vrstvou.

3. FingerprintService a fingerprintd vykonávajú volania cez Fingerprint HAL do knižnice špecifickej pre výrobcu, kde prebieha registrácia odtlačkov prstov a ďalšie operácie spojené s autentifikáciou.

4. Fingerprint HAL je implementovaný v jazykoch C/C++ a predstavuje rozhranie `IBiometricsFingerprint` HIDL. Táto vrstva sprostredkúva komunikáciu medzi systémom Android a hardvérom snímača odtlačkov prstov. [50]

5. Po úspešnej autentifikácii komponenty v TEE vytvoria AuthToken, ktorý je podpísaný pomocou bezpečného HMAC kľúča. [26]

6. AuthToken je následne odovzdaný službe Keystore, ktorá komunikuje s Keymasterom. Keymaster overí AuthToken a na jeho základe rozhodne o sprístupnení kryptografických kľúčov uložených v Keystore.[51]



Obrázek 2.4 Proces autentifikácie v Android zariadeniach pomocou odtlačku prsta

(zdroj: 50)

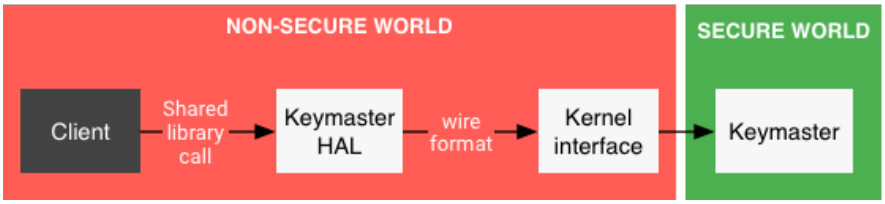
Rozpoznanie tváre

Android predstavil od verzie 10 novú technológiu autentifikácie pomocou rozpoznania tváre. Táto nová technológia na podporovanom hardvéri dokáže bezpečne spracovať snímky kamier, pričom zachováva bezpečnosť a súkromie počas overovania tváre.

Autentifikácia rozpoznaním tváre je podobná ako pri odtlačku prsta. Mobilné aplikácie nemajú priamy prístup k `FaceManager`, ale používajú `BiometricPrompt`. `FaceManager` priamo komunikuje s `FaceService`, ktorá sa stará o prístup k hardvéru na autentifikáciu tváre.

Keystore

Android Keystore umožňuje mobilným aplikáciám bezpečne ukladať kryptografické kľúče, ktoré sú neexportovateľné. Tieto kľúče môžu byť viazané na autentifikáciu používateľa. Pri ukladaní citlivých údajov (napríklad tokenov) sa tieto údaje zašifrujú pomocou kľúča uloženého v Keystore. Pred použitím kľúča musí používateľ prejsť autentifikáciou (PIN, vzor, heslo alebo biometria). Týmto spôsobom je zabezpečené, že ani samotná aplikácia nemá prístup k údajom bez overenia identity používateľa. Ak je autentifikácia vyžadovaná, systém zabráni použitiu kľúča, pokiaľ používateľ nepreukáže svoju identitu. Autentifikácia môže byť spustená pomocou `BiometricPrompt` alebo iného vhodného mechanizmu. Po úspešnej autentifikácii aplikácia môže vykonať kryptografické operácie (napr. šifrovanie alebo dešifrovanie) pomocou kľúča uloženého v Keystore. Avšak, samotný kľúč nikdy neopustí bezpečné úložisko.[51] Tento proces ja znázornený na obrázku 2.5.



Obrázek 2.5 Tok komunikácie medzi klientom (mobilnou aplikáciou alebo systémom) a bezpečným prostredím Keymaster

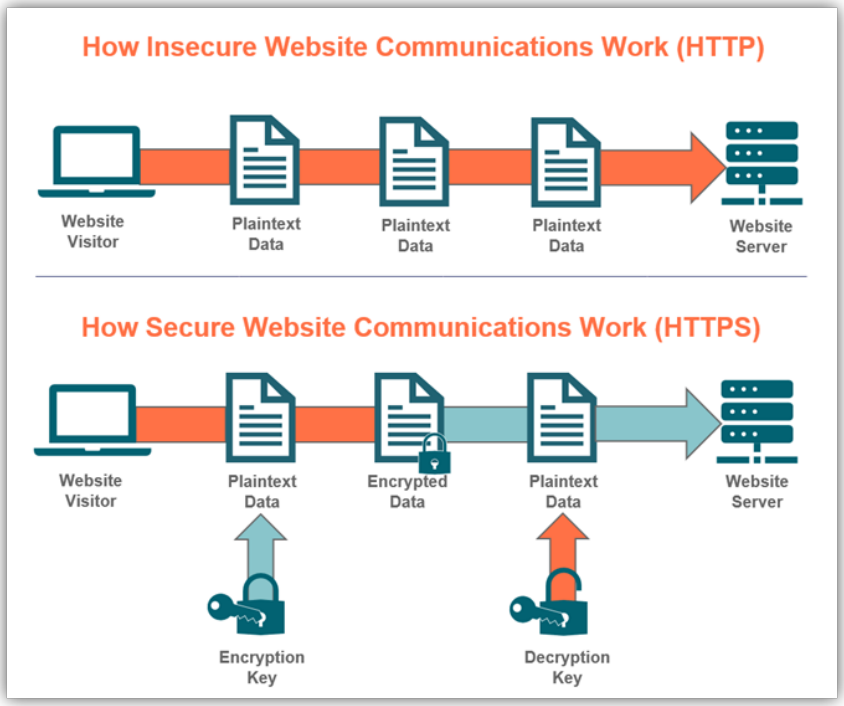
(zdroj: 51)

2.3 Zabezpečenie aplikácie SSL/TLS certifikátom

HTTPS je zašifrovaná verzia HTTP protokolu. Na šifrovanie komunikácie medzi klientom a serverom používa TLS protokol. Použitie HTTPS zaručuje bezpečnú komunikáciu medzi serverom a klientom a teda bezpečnú výmenu citlivých informácií.[52]

Pri používaní HTTP informácia posielaná týmto protokolom je rozdelená do paketov dát, ktoré sú vystavené nebezpečenstvu odhalenia pomocou voľne dostupných softvérov. Dáta posielané ako čistý text sú jednoducho dosiahnuteľné každému, kto používa správny nástroj. To je nebezpečné hlavne pri používaní verejnej Wi-Fi. Na druhej strane **HTTPS** zabraňuje vystaveniu dát, ktoré môžu byť takto jednoducho zneužité.

HTTPS chráni webovú aplikáciu pred vložením obsahu do webovej stránky bez súhlasu majiteľa. Tento obsah je zväčša nejaký typ reklamy, ktorý je vložený poskytovateľom internetovej služby za účelom zisku, ktorý nie je zdieľaný s majiteľom webovej aplikácie.[53] Porovnanie komunikácie pomocou HTTP a HTTPS je možné vidieť na obrázku 2.6.



Obrázek 2.6 Porovnanie komunikácie pomocou HTTP a HTTPS

(zdroj: 54)

2.3.1 TLS a SSL protokol

Za HTTPS sa skrýva TLS protokol, ktorý sa stará o zabezpečenú komunikáciu medzi serverom a klientom. Jeho predchodcom bol SSL protokol, ktorý nahradil práve TLS. Avšak, "SSL"sa stal tak zaužívaným pojmom pre túto technológiu, že aj ked bol nahradený, termín zostal a dodnes sa používa SSL/TLS.[55]

Na to, aby klient a server spolu komunikovali zabezpečene - pomocou SSL/TLS, musí prebehnúť SSL/TLS *handshake*. Ten sa skladá z niekoľkých krokov ako autentifikácia medzi sebou, zhodnutie sa na šifrovacích štandardoch a vytvorenie bezpečného kanálu na prenos dát. Ešte pred zahájením SSL/TLS *handshake* dochádza k *TCP three-way handshake* - klient začne nadväzovať spojenie so serverom. Kroky *TCP three-way handshake*:

1. **SYN**: Klient pošle serveru SYN paket. Ten obsahuje náhodne vygenerované počiatočné číslo sekvencie (x) a TCP príznaky a voliteľné nastavenia.

2. **SYN-ACK**: Po prijatí SYN packetu server vracia náhodne vygenerované počiatočné číslo sekvencie (y) a (x+1). Pre ACK je to x+1 čo značí, že server potvrdzuje paket poslaný klientom. SYN je rovnako náhodne zvolené počiatočné číslo sekvencie (y), ktorú vyberá server na svojej strane. Tento SYN-ACK packet je poslaný naspäť klientovi.

3. **ACK**: Po prijatí SYN-ACK paketu, klient podobne ako server pri SYN-ACK ho potvrdí tak, že navýši číslo sekvencie SYN (y+1) a ACK (x+1). To pošle serveru naspäť.[56]

Po nadviazaní TCP spojenia dochádza k SSL/TLS *handshake*, ktorý overuje, že používateľ nie je pripojený k nedôveryhodnej strane - útočníkovi. Uisťuje sa, že žiadna tretia strana nemôže meniť alebo získať informácie, ktoré putujú naprieč internetom, ktorého kroky sú:

1. **ClientHello** posiela klient ako správu na server keď je vytvorená žiadosť na bezpečné spojenie. Správa "ClientHello"obsahuje dôležité informácie ako klientom podporované SSL/TLS verzie a šifrovacie súpravy, ktoré klient môže použiť.

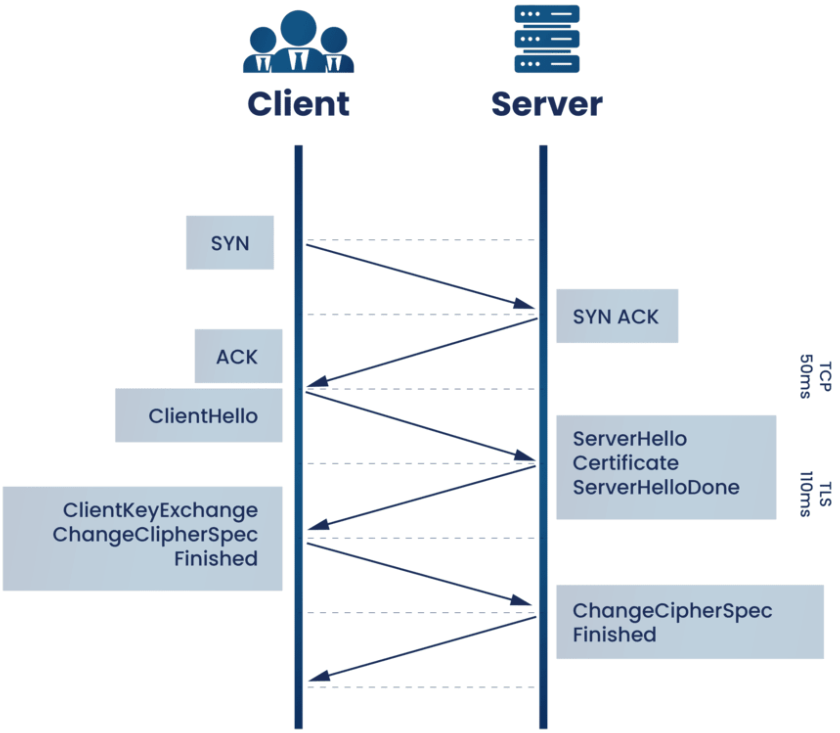
2. **ServerHello** je odpoveď servera na "ClientHello". Odpoveď obsahuje najvyššiu verziu SSL/TLS a šifrovacie súpravy, ktoré podporujú obe strany.

3. **Overovacie údaje servera**, kedy server predstavuje jeho digitálny certifikát overený Certifikačnou Autoritou.

4. **Verifikácia klientom a generovanie kľúča.** Klient validuje certifikát servera a po jeho potvrdení použije verejný kľúč servera na šifrovanie *premaster secret* - unikátny session kľúč. *Premaster secret* je poslaný naspať serveru.

5. **Vytvorenie bezpečného spojenia** tak, že server dešifruje *premaster secret* svojím privátnym kľúčom. Tak server dostane session kľúč, ktorý je použitý na symetrické šifrovanie celej nasledujúcej komunikácie medzi klientom a serverom.

Bez SSL/TLS *handshake* sú citlivé informácie vystavené naprieč online priestorm.[57] Celý tento proces je možné vidieť na obrázku 2.7.



Obrázek 2.7 TCP a TLS *handshake*

(zdroj: 58)

3 ZÁSADY GDPR PRI TVORBE APLIKÁCIÍ

Táto kapitola je prevzatá z textu seminárnej práce autorky.

GDPR (General Data Protection Regulation) je Nariadenie Európskeho parlamentu a Rady (EU) 2016/679 z roku 2016 o ochrane fyzických osôb v súvislosti so spracovaním osobných údajov a o voľnom pohybe týchto údajov.[59] GDPR sa teda zaoberá ochranou osobných údajov občanov Európskej Únie, Nórska, Islandu a Lichtenštajnska. Každá firma, ktorá vykonáva transakcie v rámci Európskej Únie musí dodržiavať pravidlá GDPR.[60]

3.1 Pojmy v GDPR

Prevádzkovateľ osobných údajov je zvolená entita, ktorej používateľ povolil zhromažďovať svoje údaje na vykonávanie transakcií.[60] Je to teda osoba, organizácia alebo iný subjekt, ktorý rozhoduje o účele a prostriedkoch spracúvania osobných údajov. V podstate to znamená, že prevádzkovateľ určuje, ako a prečo budú osobné údaje spracovávané.[61] Obecne ním býva majiteľ webovej stránky či aplikácie.[60]

Spracovateľ údajov je subjekt alebo osoba, ktorá spracúva údaje v mene správcu. Napríklad služby tretích strán ako Google, Amazon, iTunes alebo PayPal. V niektorých prípadoch sa za spracovateľov údajov môžu považovať aj spolupracujúce vývojové spoločnosti. [60]

Subjekt údajov je osoba, ktorej osobné údaje sú spracúvané. V kontexte GDPR to znamená jednotlivca, ktorého osobné informácie zhromažďuje, spracováva a uchováva prevádzkovateľ osobných údajov. Subjekt údajov má v rámci GDPR viacero práv.[61] Je to teda používateľ webovej, či inej aplikácie.[60]

Zodpovedná osoba pre ochranu osobných údajov (DPO) je osoba, ktorá má v rámci organizácie na starosti monitorovanie súladu s pravidlami ochrany osobných údajov podľa nariadenia GDPR. DPO má na starosti dohľad nad spracovaním osobných údajov, školenie zamestnancov a zabezpečenie, že organizácia spĺňa všetky povinnosti spojené s ochranou údajov. Táto osoba musí mať dostatočné odborné znalosti a nezávislosť pri výkone svojej funkcie.[61]

3.2 Požiadavky GDPR

Medzi kľúčové požiadavky tohto nariadenia patrí:

- Firma by mala vyžiadať explicitný súhlas od užívateľov predtým, ako zhromaždí ich údaje.
- Zabezpečenie údajov má byť prioritou od začiatku a automaticky aktívne.
- Používatelia majú mať prístup k svojim dátam.
- Každý používateľ by mal mať právo na prenesenie údajov.
- Každý používateľ by mal mať právo na výmaz svojich údajov.
- Striktná implementácia pravidiel.
- Firma by mala informovať používateľa v prípade úniku osobných údajov.[60]

Tieto požiadavky popisujú, ako by firma mala pristupovať k téme ochrane osobných údajov. Každý občan Európskej Únie má právo vedieť, čo sa s jeho citlivými údajmi deje.

GDPR jasne vraví o tom, ktoré informácie *prevádzkovatelia osobných údajov* a *spracovatelia údajov* sú povinní poskytnúť na žiadosť zákazníka. A to:

- kto je spracovateľom údajov,
- prečo má ich dáta,
- odkiaľ má ich dáta,
- ako dlho bude mať ich dáta,
- ako a kde sa ich dáta spracovávajú.

Všetky údaje o *subjekte údajov* musia byť na jeho žiadosť vymazané a to všade, kde sa spomínajú. *Subjekt údajov* musí byť absolútne odstránený do mesiaca od žiadosti.

Rovnako podľa GDPR musí byť súhlas na spracovávanie a uchovávanie dát jasne daný a teda, používateľ musí udeliť slobodný a špecifický súhlas so spracovaním jeho údajov. Mlčanie neznamená súhlas.

Nariadenie jasne stanovuje osobné údaje, ktoré sú priveľmi citlivé a na ich spracovanie, použitie a zdieľanie je potrebný súhlas *subjektu údajov*. Týmito údajmi sú:

- náboženské alebo filozofické presvedčenia,
- fyzické a duševné zdravie,
- údaje o sexuálnom živote,
- rasový alebo etnický pôvod,
- členstvo v odboroch,
- politické názory,
- biometrické a genetické údaje.

Ak by došlo k úniku osobných údajov, GDPR udáva povinnosť kontaktovať *subjekt údajov*. Ak by došlo k úniku informácií, ktoré môžu spôsobiť škodu, či nejakú formu ublíženia, je povinnosťou informovať *subjekt údajov* do 72 hodín.[62]

3.3 Technické aspekty implementácie GDPR

V prvom rade je nutné zoznámiť sa s hlavnými princípmi GDPR, ktoré musia byť v aplikácii zohľadnené a implementované. Taktiež by malo byť pred samotnou implementáciou určené, ktoré dáta je nutné uchovávať. Najlepším prístupom je uchovávať len dáta, ktoré sú nevyhnutné.

Implementácia GDPR do webovej aplikácie vyžaduje niekoľko technických opatrení, ktoré zabezpečia súlad s požiadavkami na ochranu osobných údajov. Medzi hlavné technické aspekty patrí šifrovanie, získavanie súhlasu, práva subjektov údajov a zabezpečenie dát pri prenose aj uložení.

3.3.1 Šifrovanie a zabezpečenie dát

Samotné nariadenie GDPR odporúča a dokonca kladie dôraz na šifrovanie dát, i keď nie je uvedené ako povinné. Šifrovanie a zabezpečenie dát pre aplikácie ako také (webové či mobilné) je vykonávané pri **prenose dát** rovnako ako sa šifrujú aj **uložené dáta**. Popis pravdepodobnosti možných útokov v závislosti na zabezpečení je možné vidieť na obrázku 3.2.

		Attacker			
		Provider (or its employees)		Hacker	
		Plausibility (Motivation)	Success probability	Plausibility (Motivation)	Success probability
Approach	1. Local, on-premises storage of encrypted data	Low	Low	Moderate	Significant
	2. Cloud storage without client-side encryption	Moderate	Large	Large	Moderate
	3. Cloud storage with client-side, end-to-end encryption	Low	Low	Moderate	Moderate

Obrázek 3.1 Možnosti zabezpečenia a pravdepodobnosť útoku

(zdroj: 62)

End-to-end šifrovanie na strane klienta (dáta sú zašifrované už pred odoslaním do cloud) minimalizuje riziko narušenia dôvernosti dát zo strany poskytovateľa aj hackera. Preto je v kontexte GDPR najvhodnejšie.

Šifrovanie pri prenose dát

Jednou z metód využívaných na šifrovanie pri prenose dát je **SSL/TLS certifikát**.[60] Je to objekt, ktorý umožňuje nadviazať zabezpečené (šifrované) spojenie na sieti pomocou SSL/TLS protokolov. Tieto certifikáty sa používajú v rámci kryptografického systému známeho ako PKI. PKI poskytuje spôsob, ako môže jedna strana zistiť identitu druhej strany pomocou certifikátov, ak obe dôverujú tretej strane – známej ako certifikačná autorita. SSL/TLS certifikáty teda fungujú ako digitálne identifikačné karty na zabezpečenie sieťovej komunikácie, stanovenie identity webových stránok cez internet, ako aj zdrojov v súkromných sieťach. Tento certifikát má niekoľko benefitov, ktoré úzko súvisia aj s požiadavkami kladenými na aplikácie GDPR. Nimi sú podpora dodržiavania predpisov a ochrana súkromných dát. Technológia SSL/TLS pomáha zabezpečiť šifrovanie celej komunikácie medzi prehliadačom a webovou stránkou. Kľúčovými princípmi v tejto technológii sú:

Šifrovanie, ktoré používa verejný a súkromný kľúč. Verejný kľúč je kryptografický kľúč, ktorý webový server dáva prehliadaču v certifikáte SSL/TLS. Prehliadač používa kľúč na zašifrovanie informácií pred ich odoslaním na webový server. Súkromný kľúč má

iba webový server. Súbor, ktorý je zašifrovaný súkromným kľúčom, možno dešifrovať iba verejným kľúčom a naopak. Tieto kľúče teda tvoria pár.

Autentifikácia. Server odošle verejný kľúč v certifikáte SSL/TLS do prehliadača. Prehliadač overí certifikát od dôveryhodnej tretej strany. Tým môže overiť, že webový server je tým, za koho sa vydáva.

Digitálny podpis predstavuje unikátne číslo pre každý SSL/TLS certifikát. Používa sa na uistenie, že nikto nemanipuloval s certifikátom ako putoval po sieti a to tak, že príjemca vygeneruje nový digitálny podpis, ktorý porovná s tým pôvodným.[63]

Tento prístup používa HTTPS, ktorý je už v dnešnej dobe povinný pre stránky, ktoré spracovávajú citlivé údaje.[60]

HTTP Strict Transport Security (HSTS) patrí medzi pokročilejšie bezpečnostné opatrenia pri prenose dát. Tento mechanizmus zabezpečuje, že prehliadače komunikujú so serverom len cez HTTPS, čím predchádza možnosti útokov typu man-in-the-middle.[64]

PFS (Perfect Forward Secrecy) rovnako patrí medzi pokročilejšie bezpečnostné opatrenia pri prenose dát. Zabezpečuje, že relácia bude chránená aj v prípade kompromitácie serverového privátneho kľúča. PFS používa dynamické šifrovacie kľúče pre každú reláciu, čo znamená, že prelomenie jedného kľúča neohrozí ostatné relácie.[65]

Kroky potrebné pre zavedenie šifrovania v praxi a teda splnenie určitých bezpečnostných požiadaviek, ktoré GDPR popisuje, je možné zhrnúť do nasledujúcich krokov.

Získanie SSL/TLS certifikátu od dôveryhodnej certifikačnej autority, ako je Let's Encrypt (bezplatný) alebo iné komerčné cerfitikačné autority.

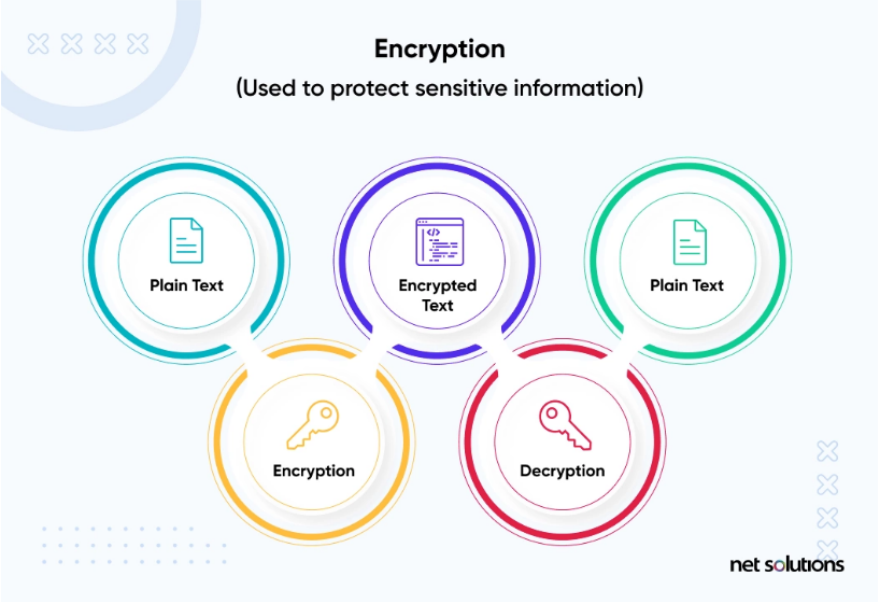
Nastavenie HSTS hlavičky v konfigurácii servera, aby sa vynútilo používanie HTTPS.

Pravidelná aktualizácia certifikátov a prechod na vyššie verzie TLS s podporou moderných šifrovacích algoritmov.

Testovanie SSL/TLS konfigurácie pomocou nástrojov, ako je SSL Labs, na kontrolu zraniteľností a zabezpečenie najvyššej úrovne ochrany.[61]

Šifrovanie uložených dát

Šifrovanie dát, ktoré sa ukladajú je ďalším dôležitým aspektom pri ochrane osobných údajov používateľov aplikácie, ktoré nariadenie GDPR zdôrazňuje. Šifrovanie takýchto údajov znamená, že je vykonaná transformácia digitálnych informácií z obyčajného (*plain*) textu do nezrozumiteľného formátu - šifrovaného textu. Transformácia je vykonávaná pomocou kľúča - zväčša textu a/alebo čísla, ktorý je generovaný kryptografickým algoritmom.[66]



Obrázek 3.2 Proces šifrovania a dešifrovania

(zdroj: 66)

Osobné informácie spomenuté v GDPR by mali byť (odporúčanie GDPR) v databáze šifrované, rovnako ako heslo, ktoré by malo byť hashované.[66]

3.3.2 Pseudonymizácia

V prípade, že z nejakých pádnych dôvodov nie je možné použiť šifrovanie, je možné implementovať pseudonymizáciu. Ako legitímny dôvod môže byť napríklad nákladová efektívnosť, či pokles výkonu.[60]

Pseudonymizácia je proces, kedy sú informácie, ktoré identifikujú osobu nahradené pseudonymami. Pseudonym je jedinečný umelý identifikátor, ktorý nahradí osobné údaje a tým sa stanú údaje v obmedzenom kontexte anonymné. Prinášajú však riziko opätovnej identifikácie údajov. GDPR je voči tejto metóde benevolentnejšie, pretože je menej pravdepodobné, že by tieto dáta mali významný vplyv na dotknuté osoby v prípade úniku.

Implementácia pseudonymizácie má určité úskalia a výzvy. Lacnejšou verziou je stále šifrovanie.[67]

3.4 Cookies a sledovacie technológie

Cookies sú krátke textové súbory - užívateľské stopy. V dnešnej dobe pri návšteve webovej stránky sú cookies uložené na akomkoľvek podporovanom užívateľskom zariadení.

Cookies sa používajú na sledovanie užívateľa a jeho aktivity v danej webovej aplikácii. Obsahujú napríklad informácie o preferovanom jazyku alebo detailoch o prihlásení. Môžu byť užitočné ako pre užívateľa aplikácie tak pre jej prevádzkovateľa. Vďaka nim je možné webovú aplikáciu personalizovať a teda zlepšiť dojem z danej webovej aplikácie.

Na základe GDPR nariadenia, by používateľ mal byť upovedomený o používaní cookies, a to aké dáta sú spracovávané a ako sú použité. Aby boli tieto požiadavky GDPR splnené je používaná kombinácia dvoch nástrojov - Cookies Policy a cookies lišta či okno.

Cookies Policy je informačný dokument o spracúvaní osobných údajov prostredníctvom cookies. Vraví o tom, ktoré dáta sú pomocou cookies spracúvané a ako sú použité. Tento dokument by mal byť vždy prístupný na webovej stránke a po jeho prečítaní sa používateľ môže rozhodnúť, či so sledovaním súhlasí, poprípade s ktorými časťami. A teda v rámci tohto dokumentu by mali byť aj informácie o tom, ako môžu používatelia uplatniť svoje práva v súvislosti so spracúvaním ich osobných údajov podľa GDPR. Používatelia by mali byť informovaní o tom, že majú právo na prístup k svojim osobným údajom, právo na opravu týchto údajov, právo na vymazanie osobných údajov a ďalšie práva v súvislosti so spracovaním osobných údajov. Z toho vyplýva, že Cookies Policy by mal byť neustále aktualizovaný a podávať užívateľom aplikácie vždy aktuálne informácie.

Cookie lišta či okno slúži na informovanie užívateľov webovej aplikácie o používaní cookies a mal by obsahovať možnosť ich správy. Je veľmi dôležitý pre súlad s GDPR a musí byť správne implementovaný na to, aby bol efektívny. Mal by obsahovať určité elementy ako:

1. ľahko čitateľná textová správa informujúca používateľa o používaní cookies vo webovej aplikácii. Musí obsahovať jasný popis účelu súborov cookies a spôsobu ich použitia;

2. interaktívne tlačítka, ktoré používateľovi umožňujú spravovať svoje preferencie cookies. Mali by používateľom umožniť odmietnuť používanie voliteľných súborov cookies, umožniť používateľom upraviť svoje preferencie týkajúce sa používania súborov cookies na danej webovej lokalite, akceptovať všetky súbory cookies naraz.

Táto lišta respektíve okno, by malo byť na webovej stránke jednoznačne viditeľné a ľahko lokalizovateľné. Pre užívateľa by mala byť prístupná a viditeľná dovtedy, dokým nevyjadrí svoje preferencie. Pri implementácii lišty cookies by sa malo zabezpečiť, aby sa zobrazovala vo všetkých častiach webovej aplikácie, kde sa používajú cookies, a aby obsahovala relevantné informácie o ich používaní.[68]

3.5 Kroky pri tvorbe aplikácie vyhovujúcej GDPR

V nasledujúcej časti sú zhrnuté kroky, ktoré by mali byť nasledované, aby webová, či mobilná aplikácia spĺňala požiadavky GDPR.

1. **Poznať údaje, ktoré sú vlastnené**
   Prvým krokom pre implementáciu požiadaviek GDPR do webovej aplikácie je poznanie vlastnených údajov. To znamená, aké osobné údaje sú vlastnené, kde sú uložené a kto k nim má prístup. Existuje niekoľko otázok, ktoré môžu pomôcť k tomuto poznaniu. Sú to:

   - Ktoré osobné údaje vlastníme?
   - Sú to citlivé osobné údaje? Ak áno, ako sú chránené?
   - Zbiera aplikácia osobné údaje od maloletých?
   - Prečo aplikácia potrebuje tieto dáta?
   - Ako je uchovaný súhlas so spracovaním týchto osobných údajov?
   - Kde sú tieto osobné údaje uložené?
   - Kto má k týmto dátam prístup?
   - Vlastnia tieto dáta tretie strany? Ak áno, ako je kontrolované spracovávanie týchto údajov?
   - Sídlia tieto tretie strany mimo Európskeho hospodárskeho priestoru? Ak áno, aký mechanizmus majú zavedený na ochranu vašich osobných údajov pred prístupom

cudzích orgánov alebo pred použitím na iné účely, ako sú tie, ktoré sú povolené v zmluve s touto treťou stranou?

- Ako dlho je potrebné tieto osobné údaje uchovávať? Je možné niektoré z týchto informácií vymazať či anonymizovať?

2. **Zabezpečiť aplikáciu**

   Údaje uložené aplikáciou musia byť chránené. Samotná aplikácia musí byť chránená pred vonkajšími útokmi, pretože sú často vystavované hackerským útokom. Odporúčané kroky pre zabezpečenie webovej, či mobilnej aplikácie:

   - Inštalácia certifikátu SSL, ktorý zašifruje akékoľvek zdieľanie informácií medzi webom a serverom.
   - Používanie silných hesiel pre administrátorské účty.
   - Rozšírenie servera o ďalšie vrstvy ochrany pre prípad, že je používateľom umožnené zdieľať informácie o platbe.
   - Použitie poskytovateľa CDN, ktorý môže zlepšiť bezpečnosť, napr. ochranou webových aplikácií pred DDoS.
   - Uchovávať, zhromažďovať a používať len osobné údaje, ktoré sú potrebné pre danú webovú aplikáciu.
   - Vyvarovanie sa (ak to nie je nevyhnutné) posielaniu a zdieľaniu osobných údajov, najmä citlivých typov, službám tretích strán.
   - Použitie pseudonymizácie alebo anonymizácie osobných údajov pred ich uložením.
   - Odstránenie osobných údajov, keď ich aplikácia už nepotrebuje.
   - Zálohovanie údajov na viacerých miestach.

3. **Aktualizácia zásad ochrany osobných údajov**

   Zásady ochrany osobných údajov by mali byť jednoducho prístupné užívateľom aplikácie. Hlavným účelom je informovať návštevníkov aplikácie o tom, ako zhromažďuje, používa, uchováva a zverejňuje ich osobné údaje spolu s vysvetlením, prečo sa tak deje. Zásady ochrany osobných údajov by mali byť jasne popísané a teda jednoducho pochopiteľné pre každého užívateľa.

4. **Získanie súhlasu pre e-maily**

   Ak je uchovávaný zoznam adresátov, ktorí sú z Európskej Únie, je nutné skontrolovať, či je v súlade s GDPR.

   Ak sú zasielané marketingové kampane, či bulletiny, je nutné získať povolenie od užívateľa. Rovnako musí mať používateľ právo odhlásiť sa z odberu týchto e-mailov.

   Odporúčaná metóda je použiť verifikáciu e-mailovej adresy, kedy používatelia musia overiť svoju e-mailovú adresu po jej zdieľaní s aplikáciou.

5. **Použitie cookie lišty/okna**

   Ak webová aplikácia používa cookies, je nutné pridať lištu/okno, vďaka ktorému môže používateľ spravovať používanie cookies. Ako už bolo spomenuté, táto lišta či okno musí byť jasne viditeľné.

6. **Kontrola formulárov webovej aplikácie**

   Ak má aplikácia akékoľvek formuláre, napr. kontakt, ktoré zhromažďujú osobné údaje, musí byť zabezpečené:

   Zahrnutie vyhlásenia o ochrane osobných údajov vysvetľujúce prečo sú nutné dané údaje, ako sa s nimi bude nakladať a objasnenie, že používateľ môže kedykoľvek tento súhlas odobrať.

   Pridanie možností, ako je napríklad nezaškrtnuté políčko alebo neaktivovaný prepínač, aby bol získaný súhlas užívateľa so zhromažďovaním údajov.

   Súhlas s dostávaním korešpondencie od danej webovej aplikácie či partnera, pomocou zaškrtávacieho políčka alebo inej podobnej možnosti.

   Prednostné pridanie odkazu na zásady ochrany osobných údajov ak je nutné/chcené získať ďalšie informácie.

7. **Kontrola spracovateľov údajov alebo služieb tretích strán**

   Je nevyhnutnosťou poznať služby a firmy, ktoré prevádzkovateľ aplikácie používa a zistiť ich zásady ochrany osobných údajov a či sú v súlade s GDPR. Rovnako musia byť v súlade s GDPR a so zásadami ochrany osobných údajov, ak vykonávajú prácu v mene spoločnosti prevádzkovateľa aplikácie.

8. **Kontrola medzinárodného prenosu údajov**

   V prípade, že aplikácia prenáša dáta mimo Európskej Únie, malo by byť zabezpečené:

   - Vykonanie potrebných hodnotení rizík pred prenosom údajov.
   - Kontrola, či poskytuje prijímajúca krajina alebo služba primeranú úroveň systému ochrany údajov.
   - Kontrola všetkých potrebných zmlúv s prijímajúcou spoločnosťou/službami.

9. **Poskytnutie práva na údaje**

   Používatelia aplikácie majú právo získať informácie o svojich osobných údajoch, ktoré sú uchovávané, a kedykoľvek požiadať o ich úpravu alebo vymazanie. Toto by malo byť zabezpečené jednoduchým prístupom k správnym možnostiam na uplatnenie týchto práv. Ako sú tieto požiadavky naplnené by malo byť popísané v zásadách ochrany osobných údajov dostupných v aplikácii.

10. **Analýza a zmiernenie úniku údajov**

    Jednotlivé kroky na pripravenie systému na prípadný únik údajov:

    Uchovávať záznamy o spracovateľských činnostiach.

    Zablokovanie prístupu do aplikácie, dokým nie je opravená zraniteľnosť.

    Vykonanie dôkladného vyšetrovania po úniku – kde, kedy a ako sa to stalo, o aké údaje išlo a kto bol ovplyvnený a ako.

    Informovať príslušný dozorný orgán o porušení do 72 hodín so všetkými informáciami, ktoré sú k dispozícii. Zvyčajne musí oznámenie o narušení obsahovať kategórie a približný počet dotknutých užívateľov; kategórie a približný počet dotknutých záznamov osobných údajov; akékoľvek opatrenia prijaté alebo plánované spoločnosťou v reakcii na porušenie, vrátane opatrení na zmiernenie jeho možných nepriaznivých účinkov.

    Upozornenie dotknutých užívateľov, ak existuje zvýšené riziko pre práva a slobody užívateľov v dôsledku porušenia. Informovať užívateľov, čo môžu urobiť na ochranu svojich údajov.

    Aktualizovať zásady a postupy, aby sa predišlo budúcim narušeniam bezpečnosti.

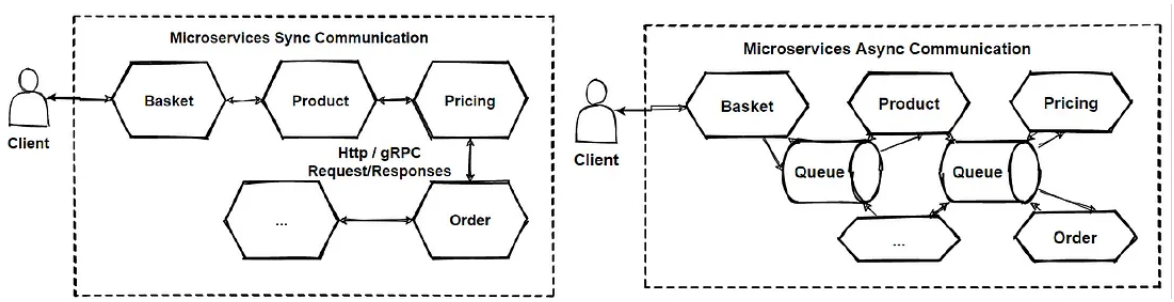
Pripravenie akčného plánu, ak dôjde alebo pravdepodobne v budúcnosti dôjde k ďalšiemu narušeniu údajov.[69]

4 KOMUNIKAČNÉ MECHANIZMY MEDZI MIKROSERVISAMI A NOTIFIKAČNÝ SYSTÉM

4.1 Komunikačné mechanizmy medzi mikroservisami

Firmy v dnešnej dobe siahajú po mikroservisách a prechádzajú z monolitov práve na mikroservisy. Tento architektonický návrh potrebuje mechanizmus komunikácie medzi jednotlivými mikroservisami. Táto komunikácia môže byť synchrónna, asynchrónna, jednoduchá či komplexná. RESTful API, gRPC alebo fronty správ sa radia medzi najčastejšie používané prístupy.[70]

Pretože mikroservisy sú distribuované a každá mikroservisa má vlastnú inštanciu a beží v samostatnom procese, komunikujú medzi sebou na sieťovej úrovni prostredníctvom medziservisovej komunikácie. Preto je nevyhnutné, aby servisy medzi sebou komunikovali pomocou protokolov, ako sú HTTP, gRPC alebo message brokery využívajúce napríklad AMQP protokol. Typy komunikácie medzi mikroservisami je možné rozdeliť následovne:



Obrázek 4.1 Asynchrónna a synchrónna komunikácia medzi mikroservisami

(zdroj: 71)

Synchrónna komunikácia zväčša používa HTTP alebo gRPC protokol - vracajú synchrónnu odpoveď. Synchrónna pretože klient odošle požiadavku a dokým nedostane odpoveď zo strany servera, blokuje dané vlákno.

Asynchrónna komunikácia je presný opak synchrónnej. Klient taktiež odošle požiadavku na server, ale nečaká na odpoveď. To znamená, že neblokuje vlákno dokým nepríde odpoveď zo strany servera. Tento typ najčastejšie využíva fronty správ ako Kafka.

Asynchrónnu komunikáciu je ešte možné ďalej deliť podľa implementácie - mód *one-to-one* (fronta) alebo mód *one-to-many* (téma). V prípade *one-to-one* je jeden odosielateľ a jeden

príjemca kdežto v *one-to-many* je viacero príjemcov.[71]

REST

REST je momentálne najbežnejším prístupom v kontexte komunikácie medzi mikroservisami. Siahajú po ňom vývojári práve pre jeho jednoduchosť a všadeprítomnosť. Používa HTTP metódy ako GET, PUT, POST, DELETE a na komunikáciu používa JSON formát. Tento formát je jednoduchý na čítanie a ľahko pochopiteľný. Medzi výhody REST zaraďujeme:

- nezávislosť od programovacieho jazyka,
- podpora naprieč programovacími jazykmi a frameworkami,
- čitateľné edpointy,
- jednoduchosť uľahčuje prototypovanie,
- podpora nástrojmi ako Postman či Swagger

Avšak, s REST sú spojené aj nevýhody ako:

- textovo orientovaný a bezstavový (pri vysokom zaťažení a požiadavkách na nízku latenciu môže viesť k výkonnostným problémom),
- znižovanie flexibility v čase,
- synchrónny (menej vhodný pre oddelené a reaktívne (asynchrónne) pracovné toky).

Fronty správ

Narozdiel od REST sú fronty správ asynchrónne. Používajú komunikáciu medzi mikroservisami riadenú udalosťami, kedy technológie ako Apache Kafka, RabbitMQ a NATS umožňujú mikroservisám komunikovať pomocou správ, bez ohľadu na čas alebo umiestnenie. Výhody tohto prístupu sú:

Nezávislý chod producentov a konzumentov - prijímanie a odosielanie správ je asynchrónne, bez vzájomnej závislosti.

Podpora voľného prepojenia (loose coupling) - znižovanie závislosti medzi mikroservisami (ľahší vývoj a údržba).

Zvyšujú odolnosť systému - zlyhanie mikroservisy nespôsobí stratu správy, tie zostanú vo fronte.

Jednoduchšie škálovanie - pridávanie/odoberanie konzumentov podľa potreby zaťaženia.

Podporujú *eventual consistency* - výhodné keď odozva nemusí byť okamžitá.

Podporujú reaktívnu architektúru.

Zvyšovanie flexibility - nové mikroservisy môžu odoberať už existujúce správy.

Medzi nevýhody patrí:

- asynchrónny prístup zvyšuje komplexnosť ladenia, monitorovania a riešenia problémov a chýb,
- zložitejšia diagnostika chýb
- nutnosť dôkladne riešiť poradie správ a spoľahlivosť doručenia,
- prevádzková náročnosť (napríklad pri správe brokerov, particií, nastavení uchovávania správ a škálovaní konzumentov).[71]

Používaným prístupom zo skupiny fronty správ je **Apache Kafka**. Apache Kafka funguje na báze štyroch základných API.

Producer API: *aplikácii umožňuje publikovať prúd záznamov do jedného alebo viacerých Kafka tém.*

Consumer API: *umožňuje aplikácii prihlásiť sa na odber jednej alebo viacerých tém a spracovávať prichádzajúce záznamy.*

Streams API: *slúži na transformáciu dátových prúdov z vstupných tém do výstupných tém.*

Connector API: *umožňuje prepojenie Kafka tém s existujúcimi aplikáciami pomocou opakovane použiteľných producentov a konzumentov.*

Témy (topics) sú kategórie, do ktorých Kafka ukladá prúdy dátových záznamov. Záznamy sú postupne ukladané za sebou v rámci každej témy. Tieto záznamy sú uchované podľa retenčnej politiky.[72]

REST vs Apache Kafka

V nasledujúcej tabuľke 4.1 je možné vidieť porovnanie Apache Kafka a REST.

Tabulka 4.1 Porovnanie Apache Kafka a REST

Vlastnosť	**Apache Kafka**	**REST**
Odolnosť voči chybám	Vysoká	Stredná
Prepojenie systémov	Nízka závislosť	Vysoká závislosť
Priepustnosť (výkon)	Vysoká	Relatívne nižšia priepustnosť
Spracovanie dát	V reálnom čase	Zvyčajne nie v reálnom čase
Škálovateľnosť	Horizontálne škálovateľný	Obmedzená škálovateľnosť
Štýl komunikácie	Asynchrónny	Synchrónny

(zdroj: 72)

4.2 Notifikačný systém v mobilných aplikáciách

Ako notifikačný systém v mobilných aplikáciách sa používajú push notifikácie. Sú to malé správy, ktoré vyskočia používateľovi v ich zariadení po tom, čo sú odoslané. Používateľ ich môže vidieť aj na zamknutej obrazovke. Oproti iným komunikačným kanálom ako email, je ich účelom cieliť na používateľa v reálnom čase.[73]

Prvýkrát boli predstavené svetu v roku 2008 Matiasom Duarte a jeho dizajnovým tímom, ktorý bol predchodcom Androidu. V roku 2009 túto technológiu predstavila aj firma Apple. Používajú sa na rôzne účely ako:

- Marketing
- Občianska komunikácia
- Bezpečnostná autentifikácia[74]

Push notifikácie môžme rozdeliť do dvoch základných skupín. Delia sa na základe obsahu správy a ich cielenia na:

Transakčné sa používajú na informovanie používateľa o dôležitých informáciach súvisiacich s aplikáciou. Vznikajú na základe interakcie používateľa s aplikáciou a používateľ ich očakáva. Podávajú informácie napríklad o stave objednávky (potvrdená, odoslaná a iné), stave platby či informácie o doručení. Zväčša sú vyvolané akciou používateľa (vytvorenie objednávky, platby). Poskytujú používateľom potrebné informácie keď ich potrebujú, v reálnom čase, kedy informácia vznikla a tak lepšie spájajú používateľa s aplikáciou. Môžme sem radiť obnovenie hesla, potvrdenia, dvojfaktorovú autentifikáciu, potvrdenie rezervácie a mnohé iné.

Marketingové sa nazývajú aj propagačné notifikácie. Nevravíme o nich ako o nevyhnutných notifikáciach pre používateľa a jeho informovanosť ako v prípade transakčných. Ich hlavným cieľom je zaujať, potešiť používateľov a teda udržanie aplikácie v povedomí používateľa a v neposlednom rade, na generovanie zisku. Môžu obsahovať správy o špeciálnych ponukách a propagovať aplikáciu, aby sa do nej užívateľ vrátil.[73]

4.2.1 Architektúra notifikačného systému

Push notifikácie pre iOS a Android fungujú na rovnakom princípe s malými rozdielmi v použitej technológii. Základnými prvkami architektúry push notifikácií sú mobilné zariadenie, aplikácia, server, servisa operačného systému pre push notifikácie (OSPNS). Pre iOS aplikácia komunikuje s Apple Push Notification Service (APNS) a v prípade Android aplikácia komunikuje s Firebase Cloud Messaging (FCM). Všeobecná architektúra push notifikácií vyzerá nasledovne:

1. **Registrácia zariadenia a získanie tokenu**
   Popritom, ako užívateľ po prvýkrát otvorí aplikáciu, aplikácia si vyžiada povolenie posielať push notifikácie. Pokiaľ toto povolenie používateľ nedá, žiadne push notifikácie mu z aplikácie chodiť nebudú. Ak je tento súhlas udelený, zariadenie nadviaže komunikáciu s APNS v prípade iOS alebo FCM v prípade Android. Toto spojenie je dôležité práve pre generovanie unikátneho tokenu pre každé zariadenie. Tento token je unikátny globálne a špecifický pre dané zariadenie a aplikáciu. Pokiaľ by používateľ nainštaloval rovnakú aplikáciu na iné zariadenie a udelil súhlas s push notifikáciami, token bude odlišný od toho na predchádzajúcom zariadení. Pokiaľ používateľ odinštaluje a opätovne nainštaluje aplikáciu, udelí súhlas na notifikácie, potom bude tento

token opäť iný. Spôsob generovania tohto tokenu je pre každú platformu špecifický, ale výsledkom je vždy unikátny identifikátor zariadenia a aplikácie - token.[75]

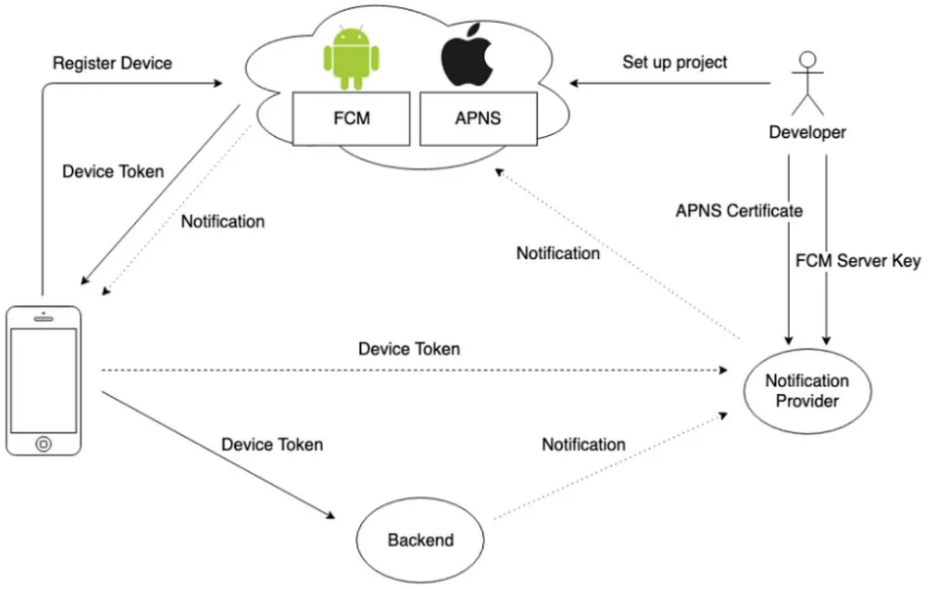
2. **Uloženie tokenu na serveri**

   Token je odoslaný na backendový server aplikácie, kde je uložený v databáze. Je dôležité pravidelne aktualizovať a validovať tokeny, pretože sa môžu meniť pri určitých udalostiach, ako je preinštalovanie aplikácie alebo aktualizácia systému. Ukladanie tokenu, ktorý odoslal APNS alebo FCM je označované ako *best practice*. Rovnako sa odporúča ukladať čas vytvorenia alebo úpravy.[76]

3. **Odoslanie notifikácií**

   Na základe určitých udalostí, ktoré vyvolávajú odoslanie notifikácie, backendový server rozhodne o odoslaní notifikácie. Server vytvorí payload správy a odosiela ju poskytovateľovi push služieb spolu s príslušným tokenom zariadenia. Notifikačný payload je objekt typu JSON, ktorý reprezentuje celú notifikáciu. Obsahuje správu, nadpis a ďalšie údaje (napríklad URL na presmerovanie po kliku na notifikáciu). Poskytovateľ push notifikácií smeruje payload do cieľového zariadenia podľa jeho tokenu. Dané zariadenie push notifikáciu obdrží a zobrazí.[77]

Na obrázku 4.2 je možné vidieť opísaný spôsob fungovania push notifikácií.



Obrázek 4.2 Architektúra notifikačného systému

(zdroj: 78)

4.2.2 Poskytovatelia služieb pre push notifikácie a ich porovnanie

Existuje mnoho poskytovateľov služieb pre push notifikácie. Výber poskytovateľa záleží na požiadavkách aplikácie a preferenciách vývojárov. Pri výbere sa zohľadňujú:

- vlastnosti, ktoré sú v aplikácii potrebné,
- rozpočet,
- podporované platformy.

Medzi najpoužívanejších poskytovateľov služieb radíme Firebase Cloud Messaging (FCM), APNS pre iOS, Firebase pre Android, Amazon Simple Notification Service (SNS), OneSignal a mnoho ďalších. V tabuľke 4.2 je porovnanie základných vlastností poskytovateľov pre viacero platforiem.

Tabulka 4.2 Porovnanie poskytovateľov push notifikácií

Funkcia	**Firebase Cloud Messaging (FCM)**	**OneSignal**	**Amazon SNS**
Cena	Bezplatné (obmedzenia)	Bezplatné, (obmedzenia), platené	Platené podľa použitia
Jednoduchosť integrácie	Vyžaduje konfiguráciu	Jednoduchá integrácia	Vyžaduje AWS konfiguráciu
Podporované platformy	Android, iOS, Web	Android, iOS, Web, Email, SMS	Android, iOS, Email, SMS
Spoľahlivosť doručenia	Vysoká	Vysoká	Vysoká

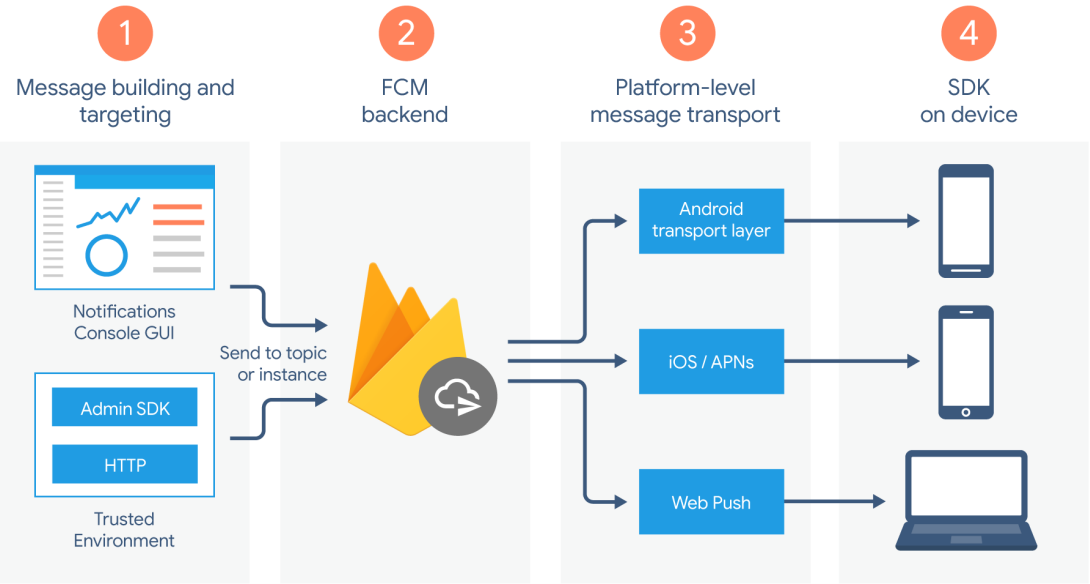
(zdroj: 79, 80)

4.2.3 Firebase Cloud Messaging (FCM)

FCM je multiplatformové riešenie na posielanie správ medzi zariadeniami používané na notifikáciu používateľov alebo posielanie okamžitých správ. Používanie FCM je podmienené dvoma bodmi, ktoré je nutné splniť:

1. Dôveryhodné prostredie (Cloud Functions pre Firebase alebo server) kde sa budú vytvárať, cieliť a posielať správy.
2. Zariadenie (Apple, Android) alebo web (JavaScript) klient, ktoré odoslané správy prijmú.[81]

FCM architektúra je zložená z komponent, ktoré vytvoria, dopravia a prijmú správu. Týmito komponentami sú nástroje na zostavenie alebo vytváranie žiadostí o správu, FCM backend, transportná vrstva na úrovni platformy a SDK zariadenia. Na obrázku 4.3 je možné vidieť architektúru FCM s jednotlivými komponentami.



Obrázek 4.3 Architektúra FCM

(zdroj: 82)

1. **Nástroje na zostavenie alebo vytváranie žiadostí o správu**

   Tieto nástroje vytvárajú správu. Je možné použiť *Notification composer* na tvorbu žiadosti o notifikáciu, alebo zvoliť automatickú možnosť. Automatický prístup pre všetky typy možností znamená, že je nutné vystavať požiadavku o správu v nejakom dôveryhodnom serverovom prostredí, ktorý podporuje Firebase Admin, SDK pre FCM server protokol. Medzi takéto prostredia radíme Cloud Functions pre Firebase, App Engine alebo vlastný aplikačný server.

2. **FCM backend**

   Backend má viacero funkcií, kde jednou z nich je prijímanie požiadavky na správy. Generuje metadáta správy a rozposiela jednu správu viacerým prijímateľom naraz pomocou tzv. tém (topics).

3. **Transportná vrstva na úrovni platformy**

   Táto vrstva smeruje správu do cieľového zariadenia. Spravuje doručovanie správ ako aj špecifické konfigurácie pre jednotlivé platformy (pokiaľ sú nevyhnutné). Pre každú platformu má vlastnú transportnú vrstvu. Pre Android zariadenia Android transport layer (ATL), pre Apple zariadenia Apple Push Notification Service (APNS) a pre webové aplikácie Web push protokol.

4. **SDK zariadenia**

   Zariadenie, kde sa notifikácia zobrazí má FCM SDK.

FCM má životný cyklus, ktorý sa skladá z registrácie zariadenia (aby bolo možné na tomto zariadení prijímať správy - získanie tokenu zariadenia) a odosielanie, prijímanie správ. Poslanie správy:

1. Zostavenie správy (v Notification composer, dôveryhodné prostredie) a poslanie žiadosti do FCM backendu.

2. FCM backend prijíme žiadosť o správu, vygeneruje metadáta a posiela správu transportnej vrstve pre konkrétnu platformu.

3. Správa prichádza na zariadenie (pokiaľ je online) pomocou transportnej vrstvy platformy zariadenia.

4. Aplikácia na danom zariadení dostáva správu.[82]

5 NASADENIE APLIKÁCIE

Nasadenie aplikácie je proces, kedy je aplikácia (súbor aplikácií) sprístupnená na používanie. Tento proces sa skladá z inštalácie, konfigurácie, aktualizácie a sprístupnenia. Proces nasadenia aplikácie musí prekonať priepastné rozdiely medzi vývojom a používaním aplikácie. Efektívny prístup k nasadeniu aplikácie je taký, ktorý zaistí, že softvér je dodaný rýchlo, spoľahlivo a bezpečne. Tým sa maximalizuje jeho hodnota. Pri správnom postupe by sa mala udržiavať konzistentnosť a teda znižovanie pravdepodobnosti chýb, nezrovnalostí medzi vývojovým, testovacím a produkčným prostredím.

Kľúčové pojmy v procese nasadenia aplikácie sú:

Deployment Pipeline je postup automatických krokov ktoré uľahčujú nasadenie. Tieto kroky sú kompilovanie, testovanie a nasadenie. Zväčša sa jedná o integráciu aplikácie do už bežiacej (nasadenej) aplikácie (CI/CD systém).

Prostredie nasadenia je prostredie kde, bude aplikácia bežať (bude nasadená). Môže to byť produkčné, predprodukčné alebo vývojové prostredie.

Automatizácia nasadenia je proces, kedy sa nasadenie automatizuje pomocou rôznych nástrojov a scriptov. Tento proces sa používa na zjednodušenie celého procesu, konzistencie a vyvarovaniu sa chybám v dôsledku ľudského faktora.

Mechanizmus obnovenia predchádzajúcej verzie v prípade, že nastala chyba v nasadzovaní alebo nastali neočakávané udalosti. Je to možnosť vrátiť aplikáciu do predchádzajúcej verzie.

Blue-Green nasadenie (Blue-Green deployment) je stratégia nasadenia. Využíva dve identické produkčné prostredia, jedno je označované Green a druhé Blue. Blue je aktuálne používané (živé) prostredie, kam chodia všetci používatelia. Green je nové prostredie, kde je nasadená nová verzia aplikácie – zatiaľ ju nikto nevidí. Nová verzia sa najprv nasadí do samostatného prostredia („green“), a až po otestovaní sa prevádzka presmeruje z pôvodného prostredia („blue“) na nové. [83]

5.1 Spôsoby nasadenia backend časti aplikácie

Existuje niekoľko spôsobov nasadenia aplikácie. Je možné využiť rôzne nástroje a ich výber záleží na špecifikách aplikácie, technológiách v nej použitých.[83] Rovnako je možné vybrať si z viacerých metód, medzi tri najznámejšie (hlavné) métody zaraďujeme:

Fyzický server je zväčša navrhnutý pre potreby aplikácie, ktorú hostí. Zvyčajne je vyhradený pre jedného konkrétneho nájomcu, kedy všetky zdroje (výpočtový výkon, procesorová kapacita, úložisko) sú vyhradené pre tohto nájomcu - aplikáciu, ktorú nájomca nasadzuje. S tým je spojený vysoký výkon pre špecifické prípady použitia definované hosťujúcou aplikáciou. Pokiaľ by chcel nájomca nasadiť inú aplikáciu, ktorá má inú konfiguráciu, musí vytvoriť celý nový server.

Výhody:

- prispôsobiteľný hardvérový a softvérový stack,
- najvyšší dostupný výkon bez latencie (pretože neobsahuje viacero vrstiev softvérovej abstrakcie),
- vysoká úroveň bezpečnosti (jeden používateľ, server sa nezdieľa medzi viacerými používateľmi)

Nevýhody:

- cena a využitie zdrojov (vyhradené hardvérové a softvérové konfigurácie),
- horšia škálovateľnosť (na škálovanie je nutné pridať ďalší identický fyzický server),
- slabá odolnosť (pri zlyhaní hardvéru, zlyhá celý stroj).

Virtuálny stroj (*Virtual Machine - VM*) je napodobenina (emulácia) skutočného počítača, ktorá beží ako program vo vnútri iného počítača - virtualizácia. Využívajú sa vrstvy softvérovej abstrakcie vďaka čomu môže na jednom fyzickom počítači bežať viacero virtuálnych strojov s rôznymi operačnými systémami. Na to, aby na jednom fyzickom počítači mohlo bežať viacero operačných systémov sa používa softvér nazývaný *hypervisor*. Ten zabezpečuje správu jednotlivých virtuálnych strojov a beží buď na hostiteľskom operačnom systéme, alebo na fyzickom hardvéri. Operačný systém vo

vnútri VM sa označuje ako hosťovský operačný systém, zatiaľ čo ten, ktorý beží na fyzickom stroji, je hostiteľský operačný systém. Virtuálne stroje sa primárne používajú v cloudových prostrediach.

Výhody:

- cena,
- škálovateľnosť,
- odolnosť (pri výpadku fyzického servera je jednoduché presunúť virtuálny stroj na iný server)
- flexibilita

Nevýhody:

- bezpečnosť (zdieľanie jedného fyzického servera s mnoho používateľmi - zraniteľnosť ovplyvňujúca jedného z nich môže potenciálne ohroziť aj ostatných),
- výkon (zdieľanie fyzických možností servera s ďalšími používateľmi),
- využitie zdrojov (preťažený virtuálny stroj nedokáže odovzdať nadbytočnú výpočtovú záťaž hostiteľskému procesoru).

Kontajnery sú najľahšou variantou virtualizácie. Jedná sa o malé balíky, ktoré sú samostatné a obsahujú aplikáciu so všetkými jej závislosťami (knižnice, frameworky, runtime prostredie). Využívajú funkcionalitu a systémové zdroje hostiteľského operačného systému. Virtualizuje samotný operačný systém (virtuálny stroj virtualizuje hardvér) a vďaka tomu umožňuje spúšťať viacero kontajnerov (každý z nich má izolované aplikačné prostredie).

Výhody:

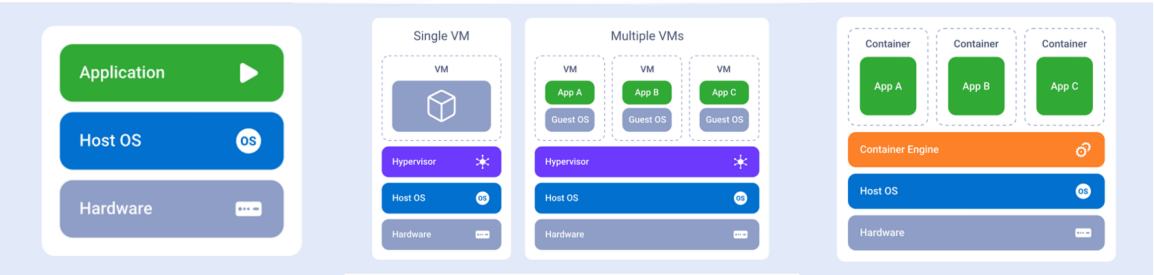
- cena,
- prenositeľnosť,
- výkon.

Nevýhody:

- bezpečnosť (zraniteľnosť v jednom kontajneri môže potenciálne ohroziť aj ostatné kontajnery na tom istom hoste),

- funkčnosť (spoliehajú vo veľkej miere na funkcionalitu hostiteľského operačného systému),
- spravovanie (rozdelenie fyzického servera na desiatky až stovky kontajnerov vytvára množstvo samostatných prvkov, ktoré je potrebné spravovať).

Tieto tri technológie nie sú úplne nezávislé jedna od druhej. Virtuálne stroje a kontajnery bežia na fyzických serveroch. Kontajnery môžu byť nasadené vo vnútri virtuálnych strojov.[84]



Obrázek 5.1 Porovnanie architektúr nasadenia aplikácií:
Fyzický server, Virtuálne stroje, Kontajnery

(zdroj: 85)

5.2 Spôsoby nasadenia mobilnej aplikácie

Nasadenie mobilnej aplikácie znamená sprístupnenie mobilnej aplikácie koncovým používateľom. Nasadenie sa skladá z viacerých krokov:

- pripravenie aplikácie pre distribúciu,
- nastavenie serverov,
- konfigurácia nastavení.

Všetky tieto kroky vedú k cieľu nastaviť a sprístupniť aplikáciu na danú platformu/platformy ako Android či iOS.

Pripravenie aplikácie pre distribúciu znamená dokončenie kódu, testovanie a vytvorenie distribučného balíčka. Aplikácia a všetky jej závislosti sa zabalia do jedného celku, ktorý možno spustiť alebo nasadiť na iné prostredie. Následne sa aplikácia distribuuje. Znamená to sprístupnenie aplikácie cez oficiálne kanály ako Apple App Store či Google Play, alebo cez

privátny kanál. V rámci distribúcie oficiálnymi kanálmi môže prebiehať kontrola aplikácie a následné schválenie/zamietnutie platformou.[86]

5.2.1 Google Play

Google Play je oficiálnym poskytovateľom aplikácií pre platformu Android. Ponúka nástroje na nasadenie, distribúciu a proces kontroly aplikácie. Je možné aplikáciu spoplatniť rôznymi spôsobmi, ako reklamy, predplatné či nákupy v rámci aplikácie. Rovnako je možné využiť analytické nástroje na sledovanie výkonu aplikácie či využívanie aplikácie používateľmi.[86]

Proces nasadenia aplikácie cez Google Play zahŕňa:

- vytvorenie vývojárskeho účtu v Google Play Console,
- sprostredkovanie metadát aplikácie (názov, ikona, popis, kategória)
- poskytnutie snímok obrazovky pri používaní aplikácie (prehľad pre potencionálneho používateľa),
- vyplnenie dotazníka na určenie vekovej vhodnosti aplikácie.[87]

5.2.2 Apple App Store

Apple App Store je oficiálnym poskytovateľom aplikácií pre platformu iOS. Ponúka nástroje na nasadenie, distribúciu a proces kontroly (rovnako ako Google Play). Rovnako ponúka monetizáciu aplikácie rôznymi spôsobmi, ktoré ponúka aj Google Play.[86]

Proces nasadenia aplikácie cez Apple App Store zahŕňa:

- vytvorenie vývojárskeho Apple Developer účtu,
- sprostredkovanie metadát aplikácie (názov, ikona, popis, kategória)
- poskytnutie snímok obrazovky pri používaní aplikácie (prehľad pre potencionálneho používateľa),
- vytvorenie App ID pre aplikáciu,
- odovzdať aplikáciu na schválenie.

V prípade Apple App Store distribúcia aplikácie je náročnejšia a môže trvať dlhší čas pretože Apple musí najprv posúdiť jej kvalitu a zhodu s ich pravidlami. Je nutné naštudovať si a držať sa ich predpisov, inak môže byť aplikácia zamietnutá, pokiaľ Apple zhodnotí, že aplikácia porušuje ich pravidlá.[87]

II PRAKTICKÁ ČÁST

6 ANALÝZA SÚČASNÝCH RIEŠENÍ PODOBNÝCH MOBILNÝCH APLIKÁCIÍ

6.1 Mobilná aplikácia Dôvera

Mobilná aplikácia *Dôvera* je určená pre zákazníkov zdravotnej poisťovne Dôvera. Ponúka prehľad receptov predpísaných od lekára, novinky zo sveta zdravotnej poisťovne Dôvera a ponúk ako napríklad príspevky pre určité úkony. Rovnako upozorňuje používateľa na jemu predpísané lieky, vydané lieky a napríklad nutnosť navštíviť lekára, kedy je prehliadka nutná v určitom časovom rozmedzí.

Pokiaľ chce poistenec Dôvery používať ich mobilnú aplikáciu, musí si vytvoriť účet. Je nutné zadať jedinečnú emailovú adresu a jedinečné telefónne číslo. Jeden kontakt tak nie je možné zdieľať na viacerých účtoch. Ak má používateľ vo svojom účte dieťa, je v poriadku mať jeden kontakt. Prvotné prihlásenie je cez emailovú adresu zadanú pri registrácii a na poskytnuté telefónne číslo je poslaný overovací kód. Tento kód je nutné zadať. Následne sa aplikácia opýta, či sa chce používateľ prihlasovať pomocou kódu (pri tejto možnosti je nutné zadať daný kód dvakrát) alebo biometrikou.

Na obrazovke prihlasovania je možné deaktivovať aplikáciu a zobraziť zdravotný preukaz používateľa. Po prihlásení aplikácia presmeruje používateľa na domovskú stránku. V spodnej časti je navigačný panel vo forme záložiek. Tieto záložky sú *Domov*, *Zdravie*, *Benefity*, *Platby* a *Viac* (menu). V hornej časti je ikona podľa pohlavia a krstné meno používateľa.

Domov obsahuje pohybujúci sa panel s novinkami. Nasledujú správy (ak nejaké sú), sekcie *Zdravie*, *Benefity*, *Preukaz poistenca* a *Kontakt*. Sekcia *Zdravie* má tri panely, ktoré navigujú používateľa do ďalších častí aplikácie. Prvým panelom je *Recepty* nasleduje *Lekári a preventívky* a *Zdravotná karta*. Sekcia *Benefity* obsahuje karty s ikonkami a popisom rôznych benefitov. Tieto karty je možné posúvať do strán. *Preukaz poistenca* obsahuje obrázok preukazu poistenca používateľa a tlačítko, ktorým sa preukaz zobrazí na celej obrazovke s možnosťou zobraziť čiarový kód a tlačítkom späť reprezentovaným šípkou doľava. Poslednou sekciou je *Kontakt* kde sú opäť dve karty s možnosťou zavolať do poisťovne a zobraziť notifikácie (správy).

Obrazovka *Recepty* má dve záložky v hornej časti, pod ikonou s krstným menom, tlačítkom späť a nadpisom *Recepty*. Prvá záložka *Nové* sú momentálne nevybrané recepty a druhá *Vybrané* kde sú už vybrané recepty. Obe záložky zobrazujú recepty. Recept je vyob-

razený vo forme karty, ktorú je možné otvoriť a zobraziť doplňujúce informácie. Základné informácie vravia o názve lieku, stave (vybraný/nevybraný), dátume a mieste vybrania lieku. Doplňujúce informácie sú meno lekára, ktorý predpísal liek, dávkovanie, doplatok, počet balení a koľko z ceny lieku zaplatila poisťovňa.

Obrazovka *Lekári a preventívky* je dizajnovo podobná obrazovke *Recepty*. Vo forme kariet sú lekári používateľa. Pre každého lekára na samostatnej karte je jeho špecializácia, dátum poslednej preventívnej prehliadky a meno. Rovnako je každá preventívka u lekára označená zeleným panelom s informáciou, že používateľ zodpovedne absolvuje preventívne prehliadky u daného lekára.

Obrazovka *Zdravotná karta* má rovnako ako predchádzajúce identický dizajn rozloženia obrazovky. Pod nadpisom je vyhľadávací panel na hľadanie záznamov. Nasleduje informácia, koľko poisťovňa zaplatila za daného používateľa za posledných 24 mesiacov. Nasledujú zdravotné záznamy oddelené názvom mesiaca a roku. Každý záznam má názov, kde bol liek vydaný a informácie o ňom. V detaile lieku je možné požiadať o reklamáciu.

Na obrazovku *Zdravie* je možné sa dostať zo spodného panelu. Opäť obsahuje odkazy na *Recepty*, *Lekári a preventívky* a *Zdravotná karta*. Nasledujú rôzne kampane poisťovne, napríklad samodiagnostika, duševné zdravie a podobne.

Obrazovka *Benefity*, ktorú je možné zobraziť zo spodného panelu obsahuje rovnaké odkazy na jednotlivé benefity ponúkané poisťovňou ako na domovskej obrazovke.

Obrazovka *Platby* podáva informácie o stave účtu poistenca. Informuje, odkedy je používateľ platiteľom poistného, či má automatické platby a možnosti nastavenia platieb.

Posledná možnosť v dolnom paneli je *Viac*. V tejto časti je profil používateľa s jeho celým menom a identifikačným číslom. Nasledujú informácie ako *E-mail*, *Mobil*, *Bankové účty* a *Súhlasy*. Všetky tieto informácie je možné meniť. Nasleduje časť *Nastavenia*, kde je možné zobraziť a nastaviť *Podmienky používania aplikácie*, *Zmena PIN*, prepnutie režimu tmavý/svetlý, *Ikona aplikácie*, *Prihlásenie pomocou biometrie* vo forme *switch*. Posledná sekcia *Aplikácia* obsahuje informácie o aplikácii, možnosť ohodnotiť aplikáciu a *kód zdieľania obrazovky*. Nasleduje možnosť deaktivovať aplikáciu a číslo verzie aplikácie.

Mobilná aplikácia spĺňa svoj účel a ponúka dostatok informácií používateľovi o stave jeho poistenia, liekov a lekárov. Avšak, aplikácia je pomerne pomalá a občas zmätočná pri prechádzaní jednotlivými sekciami, či navigáciou medzi jej časťami.

6.2 Mobilná aplikácia Medevio

Mobilná aplikácia *Medevio* je určená na objednávanie pacienta k svojim lekárom. Na prvotnej stránke prihlásenia je v ľavom hornom rohu možnosť prepnúť jazyk, nasleduje názov (ťažko čitateľný, pretože je bielou farbou na skoro bielom pozadí), informácie o aplikácii (v českom jazyku, aj keď zvyšok aplikácie je po slovensky), tlačítko *Prihlásiť sa*, *Vytvoriť účet* a *Pokračovať bez prihlásenia*.

Možnosť *Pokračovať bez prihlásenia* zavedie používateľa na obrazovku so žiadosťou o povolenie notifikácií. Tu je možnosť len povoliť, pokiaľ chce používateľ odmietnuť, musí skúsiť tlačítko späť integrované v mobilnom zariadení, aplikácia niečo také neponúka. Následne sa zobrazí domovská obrazovka. Obsahuje názov aplikácie v ľavom hornom rohu, tlačítko prihlásiť sa v pravom hornom rohu. V dolnej časti je navigačný panel s možnosťami *Domov*, *Lekári*, *Požiadavky*, *Liečba* a *Účet*. V pravom dolnom rohu je vznášajúce (*floating*) sa tlačítko plus. Po kliknutí na toto tlačítko sa zobrazí popup s možnosťami nahrať dokument z galérie, fotoaparátu, skenera či súbor. Na domovskej obrazovke je možnosť vyhľadať lekára, kampane spoločnosti a *Sledovanie chorôb* s výberom bežných problémov ako migréna, psoriáza a podobné. Po kliknutí na chorobu je možné začať liečebný plán alebo nájsť bližšie informácie o danom ochorení.

Obrazovka *Lekári* zobrazuje používateľových lekárov a možnosť pridať lekára. Obrazovka *Požiadavky* je rozdelená na tri časti pomocou panelu v hornej časti. Možnosti sú *Všetky*, *Aktívne* a *Vybavené*. Požiadavku je možné vytvoriť len pokiaľ má používateľ aktívnych lekárov v aplikácii. Obrazovka *Liečebné plány* ponúka aktívne používateľove liečebné plány a možnosť vytvoriť ďalší z ponúkaného zoznamu ochorení. Poslednou obrazovkou je *Účet* ktorý pre neprihláseného používateľa ponúka možnosť prihlásiť sa a následne možnosti *Jazyk aplikácie*, *Podmienky používania* a *Kontaktovať podporu*.

Pokiaľ na úvodnej stránke používateľ klikne na možnosť *Vytvoriť účet*, je presmerovaný na formulár, kde je nutné zadať emailovú adresu. Po zadaní emailu sa zobrazí nová obrazovka *Vaše nové heslo*. Je potrebné zadať heslo, ktorým sa bude používateľ prihlasovať.

Možnosť *Prihlásiť sa* zavedie používateľa na obrazovku prihlásenia kde je nutné zadať emailovú adresu a heslo. Rovnako je tu možnosť *Zabudnuté heslo*, kde je možné obnoviť a nastaviť nové heslo. Po prvotnom prihlásení nasleduje obrazovka s možnosťou nastaviť prihlasovanie pomocou biometriky. Následne je používateľ presmerovaný na ďalšiu obra-

zovku, kde musí zadať meno a priezvisko. Táto obrazovka má bledé pozadie a písmo je rovnako bielej farby, takže je priam nemožné prečítať, čo od používateľa aplikácia očakáva. Po úspešnom prihlásení sa opäť zobrazí obrazovka so žiadosťou o povolenie notifikácií. Po tomto kroku je používateľ na domovskej obrazovke, ktorá je identická ako pre neprihláseného používateľa. Jediný rozdiel medzi prihláseným a neprihláseným používateľom je sekcia *Účet*.

Obrazovka *Účet* pre prihláseného používateľa má niekoľko sekcií:

- *Správa informácií*, ktorá obsahuje *Pacienti*, *Moji lekári*, *Platenie*,
- *Nastavenia*, ktorá obsahuje *Nastavenia účtu*, *Oznámenia* a *Biometrické prihlásenie* s možnosťou zapnúť/vypnúť,
- *Právne záležitosti* s *Jazyk aplikácie* a *Podmienky používania*,
- *Ďalšie možnosti* s *Kontaktovať podporu* a *Odhlásiť sa*.

Možnosť *Pacienti* zobrazuje pacientov v prípade, že používateľ je aj lekárom. Možnosť *Moji lekári* zobrazí zoznam lekárov a tlačítko na pridanie lekára. *Platenie* ukazuje informácie o pravidelných platbách. Z každej časti je možné vrátiť sa späť pomocou šípky v ľavom hornom rohu.

Nastavenia účtu ponúkajú úpravu mena a priezviska, telefónneho čísla, emailovej adresy, jazyka aplikácie, zmenu hesla a odstránenie účtu. Z tejto sekcie nie je možné vrátiť sa späť pomocou tlačítka, je nutné využiť natívne tlačítko späť alebo spodný navigačný panel.

Mobilná aplikácia *Medevio* oproti ich webovej aplikácii nie je tak dotiahnutá. Sú v nej viditeľné grafické chyby a pre prihlásenie či registráciu je nutné prejsť príliš veľa krokov. Avšak, aplikácia rozhodne spĺňa svoj účel.

7 NÁVRH MOBILNEJ APLIKÁCIE

Návrh mobilnej aplikácie sa odvíjal od vzoru už existujúcej webovej aplikácie. Mobilná aplikácia preberá funkcionalitu webovej aplikácie a dopĺňa ďalšie možnosti určené výhradne pre mobilnú aplikáciu alebo rozšírenie už existujúcej webovej aplikácie. Rozširované boli funkčné požiadavky, model prípadov užitia, architektonický model, architektúra systému, dátový model aj model tried. Aktérom sú pridané nové funkčné požiadavky.

7.1 Funkčné požiadavky

Funkčné požiadavky navrhnuté podľa požiadaviek spoločnosti Diatop v rámci bakalárskej práce pre webovú aplikáciu sú registrácia, prihlásenie užívateľa, možnosť zmeny údajov užívateľom zadaných pri registrácii, vytvorenie objednávky užívateľom, poslanie objednávky vytvorenej užívateľom systémom vo forme emailu výdajni Diatop, zobrazenie prehľadu doposiaľ vytvorených objednávok užívateľovi, administrátorská časť s možnosťou nahrania dokumentu, zobrazenie registrovaných užívateľov administrátorovi, zobrazenie nahraných súborov (článkov užívateľovi k čítaniu, nie však k stiahnutiu, či ku kopírovaniu) a odhlásenie.[88] Nové funkčné požiadavky sú:

- verifikácia emailovej adresy po registrácii,
- zadanie nového hesla pri zabudnutí,
- vymazanie účtu používateľa,
- možnosť označiť objednávku stavom *odoslaná* v administrátorskej časti aplikácie,
- vytvorenie štatistík na základe údajov z tretej strany o využívaní rozpočtu na zdravotné pomôcky za daný rok a minulý rok,
- vytvorenie štatistík pre administrátora na základe údajov z tretej strany.

Rovnako pribudli funkčné požiadavky len pre mobilnú aplikáciu:

- notifikácia používateľa keď je objednávka odoslaná,
- notifikácia administrátora pri vytvorení novej objednávky,

- zmena režimu dizajnu aplikácie na tmavý/svetlý,
- zmena povolení notifikácií,
- možnosť prihlásenia sa biometrickými údajmi (pokiaľ to zariadenie dovoľuje).

Nové funkčné požiadavky sú zapracované do prípadov užitia a doplnené do aktérov.

7.2 Aktéri

Rovnako ako pre webovú aplikáciu, aj v mobilnej aplikácii sú dvaja aktéri, používateľ a administrátor. Majú rovnaké funkcionality ako boli popísané v bakalárskej práci v sekcii *Aktéri*[88]. Jediným rozdielom okrem nových funkcionalít je, že v mobilnej aplikácii používateľ nemôže vidieť dokumenty nahrané výdajňou a administrátor ich nemôže nahrávať či meniť. Dôvodom je, že výdajňa si túto funkcionalitu nepriala mať v mobilnej aplikácii. Doplnené možnosti aktérov sú:

Používateľ po registrácii musí overiť svoju emailovú adresu. Následne musí povoliť alebo odmietnuť notifikácie. Môže spravovať svoj účet, dizajn mobilnej aplikácie, nastavovať push notifikácie, odstrániť účet a vidieť míňanie svojho rozpočtu na pomôcky a porovnať to s míňaním v predchádzajúcom roku.

Administrátor rovnako musí povoliť/zamietnuť používanie push notifikácií. Následne vidí štatistiky ako počet užívateľov aplikácie, množstvo objednávok v danom roku a porovnanie počtu objednávok v priereze 5 rokov. Rovnako označuje objednávku ako odoslanú.

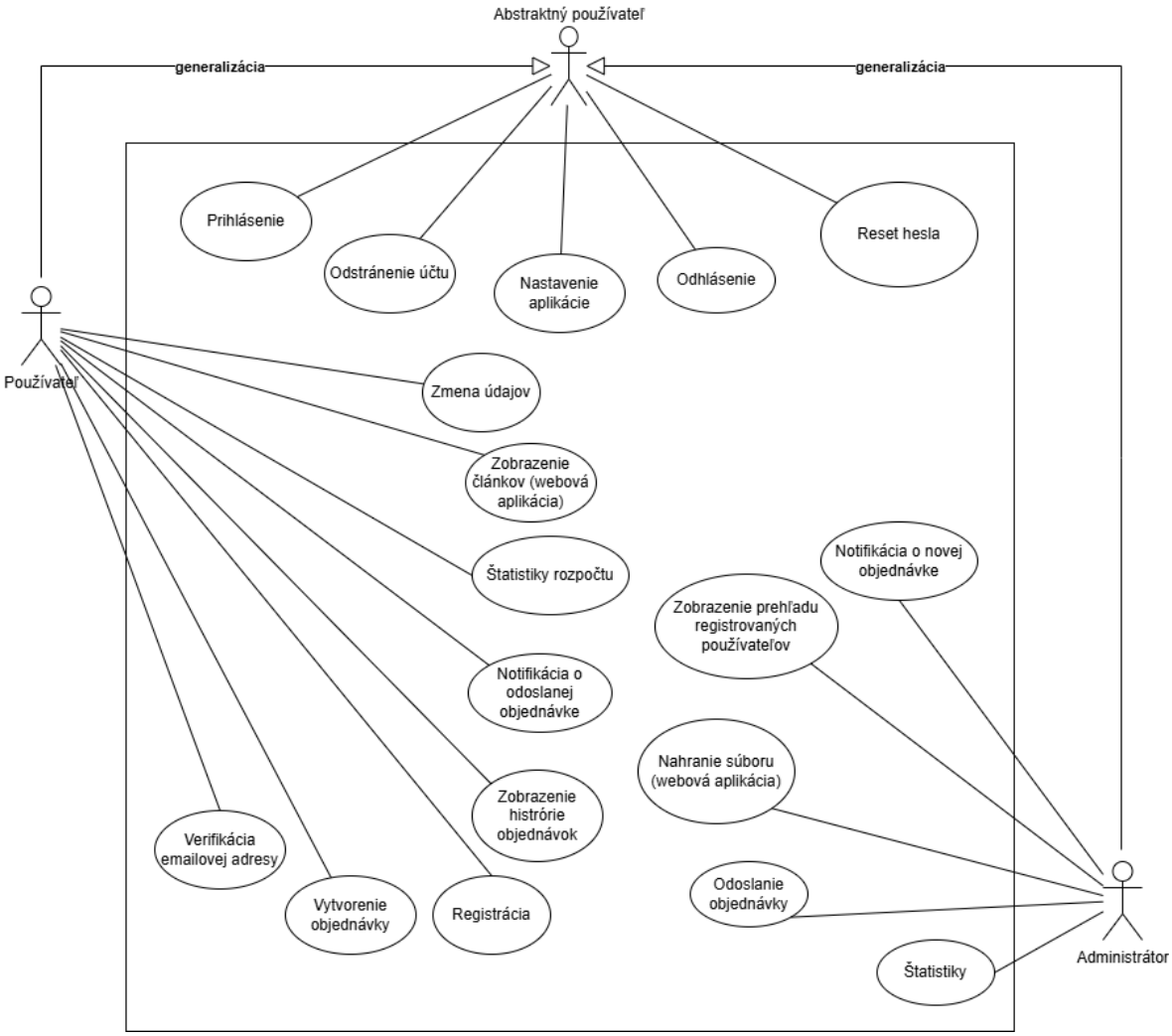
7.3 Model prípadov užitia

Model prípadov užitia reprezentuje, akým spôsobom aktéri používajú daný systém. Okrem už existujúcich prípadov sú pridané nové prípady užitia. Niektoré prípady užitia ako **prihlásenie**, **odstránenie účtu**, **odhlásenie**, **nastavenie aplikácie** - tmavý/svetlý režim, **obnovenie hesla pri zabudnutí** má používateľ a administrátor spoločné.

Používateľ má po prihlásení nové funkcionality ako **notifikácia o odoslanej objednávke** a **štatistiky rozpočtu**. Po registrácii musí overiť svoju emailovú adresu.

Administrátorovi pribudli možnosti používania aplikácie ako **štatistiky**, **odoslanie objednávky** a **notifikácia o novej objednávke**.

Všetky prípady použitia je možné vidieť na obrázku 7.1 v prehľadnom diagrame.



Obrázek 7.1 Model prípadov užitia

(zdroj: 88, vlastný)

7.4 Architektonický model aplikácie

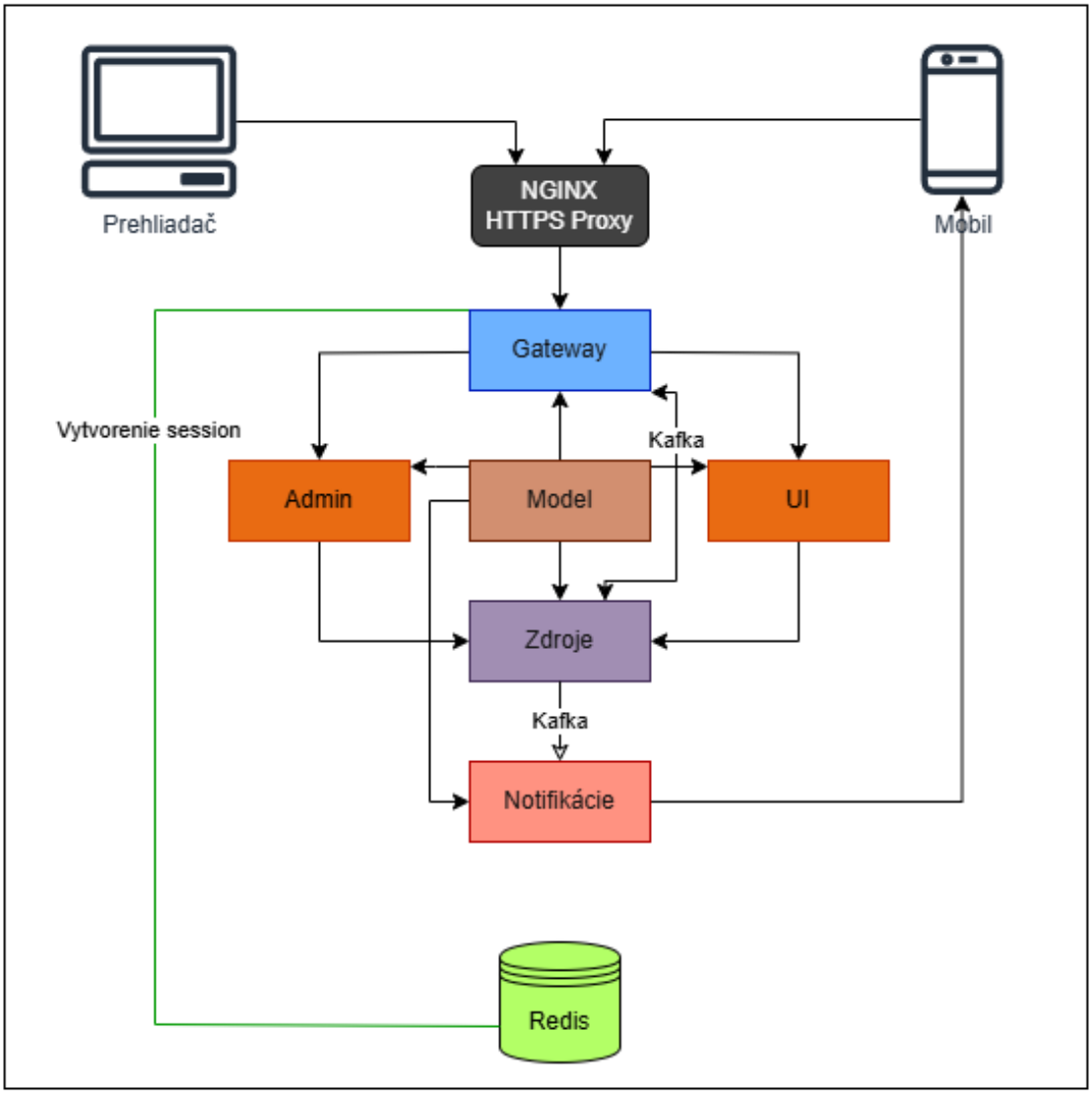
Architektonický model bol oproti pôvodnému modelu rozšírený o niekoľko nových častí. Bola pridaná nová mikroservisa pre notifikácie, reverzný proxy pre HTTPS a Apache Kafka ako spôsob komunikácie medzi mikroservisami.

Mikroservisa **Notifikácie** je rovnako ako ostatné mikroservisy Maven modulom. Jej hlavnou činnosťou je posielať notifikácie a udržiavať model pre notifikácie. Pomocou Kafka komunikuje s mikroservisou Zdroje a to tak, že mikroservisa Zdroje pošle pomocou Kafka správu mikroservise Notifikácie, aby poslala určitú push notifikáciu používateľovi.

Reverzný proxy server **NGINX** zabezpečuje HTTPS ochranu celého systému. Slúži na termináciu TLS spojení a všetky požiadavky smeruje do mikroservisy Gateway, s ktorou výlučne spolupracuje.

Mikroservisa Zdroje a Gateway rovnako komunikujú medzi sebou pomocou Kafka. Gateway obsahuje metódy pre ukladanie používateľa do databázy a teda aj šifrovanie citlivých údajov. Niektoré tieto údaje potrebuje aj mikroservisa Zdroje a preto spolu s Gateway komunikuje, aby využila jej šifrovacie/dešifrovacie algoritmy.

Celý architektonický model je možné vidieť na obrázku 7.2, pôvodné mikroservisy ako Gateway, Model, Admin, UI, Zdroje a Redis sú popísané v bakalárskej práci [88].



Obrázek 7.2 Architektonický model

(zdroj: 88, vlastný)

7.5 Architektúra systému

Architektúra systému obsahuje dátový model a model tried. Dátový model je uložený v databáze. Tento model bol od pôvodného rozšírený. V modeli tried boli pridané nové metódy a triedy. Dátový model zostal uložený v mikroslužbe Model (s výnimkou modelu pre push notifikácie) ako aj väčšia časť tried aplikácie.

7.5.1 Dátový model

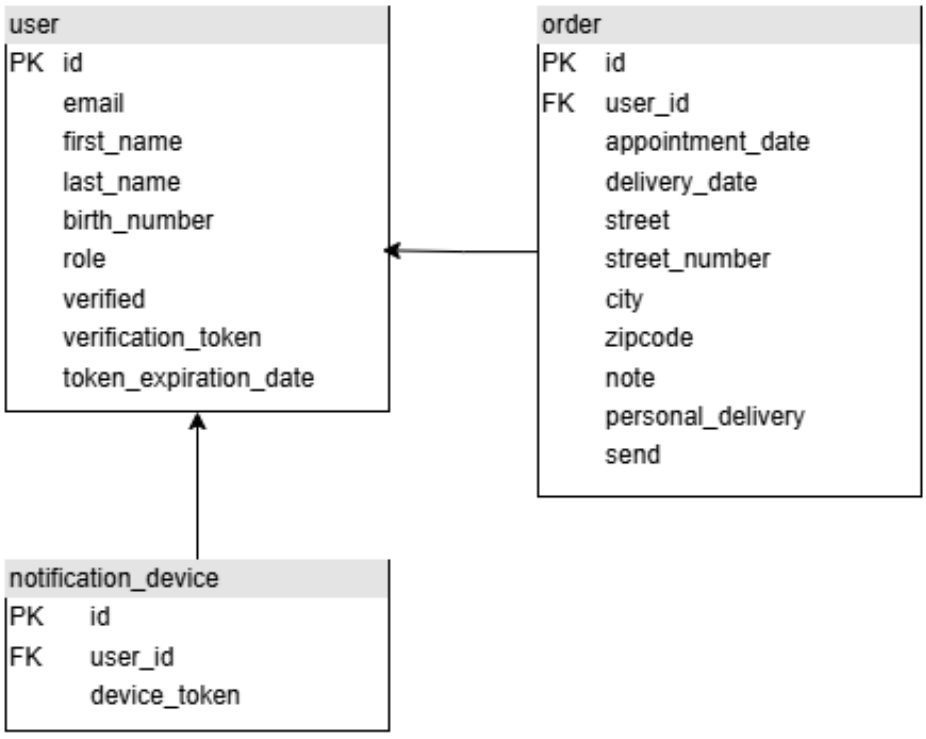
Dátový model aplikácie sa skladá z `user`, `order`, `notification_device`, `subscription` a `document`.

Do modelu `user` boli pridané:

verified hovorí o tom, či bol používateľov email verifikovaný,

verification_token je token pre verifikáciu emailu,

token_expiration_date expirácia daného verifikačného tokenu.



Obrázek 7.3 Dátový model *user*, *order* a *notification_device*

(zdroj: 88, vlastný)

Do modelu `order` bol pridaný iba `send`, ktorý znázorňuje, či bola objednávka odoslaná.

Model `notification_device` predstavuje mobilné zariadenie používateľa. Primárnym kľúčom je `id`. Cudzím kľúčom je `user_id` podľa ktorého je prepojený používateľ s konkrétnym zariadením. Používateľ teda môže mať zaregistrovaných viacero zariadení. Na správne fungovanie push notifikácie je nutný `device_token`, jedinečný token mobilného zariadenia.



Obrázek 7.4 Dátový model *subscription*

(zdroj: vlastný)

Model `subscription` reprezentuje, akú tému chce používateľ odoberať. Obsahuje `id` ako primárny kľúč, `device_token` - jedinečný identifikátor zariadenia a `topic`, ktorú chce používateľ odoberať.

Model `document` zostal nezmenený.[88]

7.5.2 Model tried

Model tried znázorňuje implementačnú štruktúru aplikácie, ako jednotlivé triedy pracujú s dátami z databázy a aké operácie nad nimi vykonávajú. Kým dátový model predstavuje konceptuálny návrh ukladania údajov, model tried zachytáva ich skutočné spracovanie v rámci aplikačnej logiky. Tento model je teda priamo previazaný s implementáciou a napomáha zabezpečiť, aby aplikácia plnila požadované funkcie podľa návrhu systému.

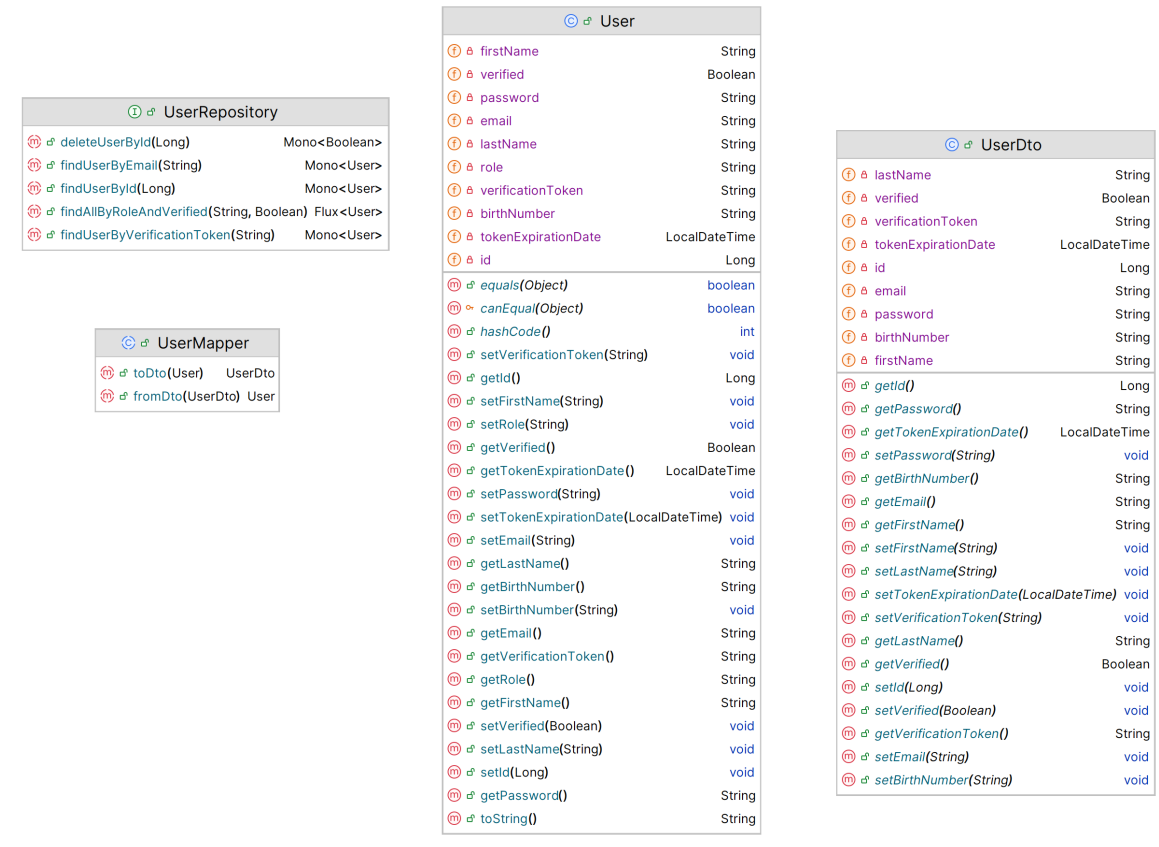
Model a vzťahy s inými mikroservisami

Väčšina tried sa nachádza v mikroservise Model. Tá je ako *Maven dependency* vložená do iných mikroservís.

Pre reprezentáciu používateľa sú triedy `User`, `UserDto`, servisa `UserRepository` a *mapper* `UserMapper`. Na obrázku 7.5 je možné vidieť metódy a atribúty jednotlivých tried. Trieda `User` a jej kópia `UserDto` boli rozšírené o atribúty:

- `verified`
- `verificationToken`
- `tokenExpirationDate`

S tým súvisia aj pridané metódy v `UserRepository` `findAllByRoleAndVerified` a metóda `findUserByVerificationToken`, ktoré sa používajú pri verifikácii emailovej adresy používateľa. Úplne novou triedou je `UserMapper`, ktorá sa stará o mapovanie z triedy `User` do `UserDto` a opačne. Model týchto tried je možné vidieť na obrázku 7.5.



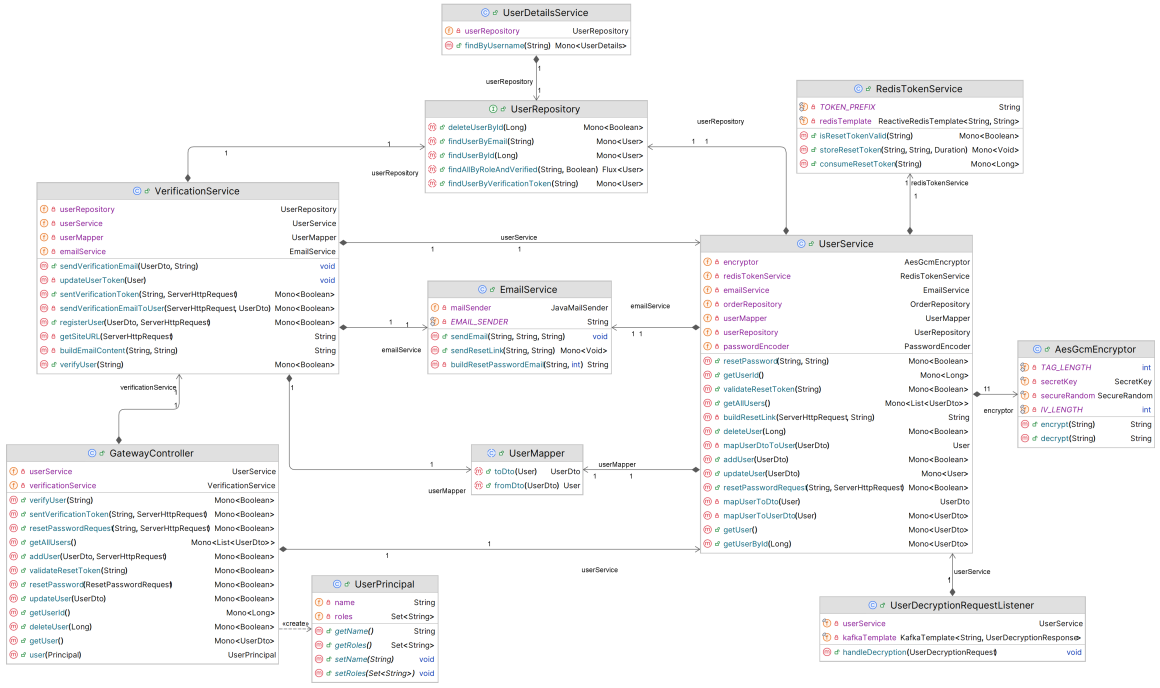
Obrázek 7.5 Model tried reprezentujúci používateľa

(zdroj: 88, vlastný)

Triedy reprezentujúce používateľa sa viažu primárne s mikroservisou `Gateway`, pretože

táto mikroservisa riadi autentifikáciu. Obrázok 7.6 zobrazuje vzťahy medzi `User` a `Gateway`. Do pôvodnej implementácie boli pridané nové triedy:

- `RedisTokenService` udržuje a pracuje s token pre obnovenie zabudnutého hesla používateľa.
- `AesGcmEncryptor` šifruje a dešifruje citlivé údaje používateľa.
- `UserDecryptionRequestListener` rieši žiadosti z iných mikroservís pre dešifrovanie zašifrovaných údajov používateľa.
- `VerificationService` verifikácia emailovej adresy - spojenie práve s atribútmi pridanými do triedy `User`.



Obrázek 7.6 Model tried reprezentujúci používateľa vo vzťahu k Gateway

(zdroj: 88, vlastný)

Model triedy `Order` bol rozšírený o atribút `send`. Vo vzťahu k objednávke vznikli ďalšie triedy:

- `OrderYearCountDto`,
- `OrderMonthCountDto`,

ktoré slúžia na tvorbu štatistík pomocou dát z tretej strany. Na obrázku 7.7 sú zobrazené všetky triedy v spojení s objednávkou s ich atribútmi a metódami. Model tried pre `Order` je



Obrázek 7.7 Model tried reprezentujúci objednávku a s ňou zviazané pomocné triedy

(zdroj: 88, vlastný)

spojený s mikroservisou `Resource`. Ich vzťahy je možné vidieť na obrázku 7.8. Rovnako je vidieť, že servisy pre žiadosť o dešifrovanie údajov využíva aj `OrderService`. Je to z dôvodu, že je nutné odoslať rodné číslo na email výdajne (požiadavka výdajne Diatop). Nová trieda `OrderEventService` sa stará o poslanie správy mikroservise `PushNotifications`,

ktorá následne odošle notifikáciu. Poslednou pridanou triedou je `ThirdPartyService`, ktorá sa stará o komunikáciu s API tretej strany.



Obrázek 7.8 Model tried reprezentujúci objednávku vo vzťahu k Resource

(zdroj: 88, vlastný)

Posledným významným celkom v `Model` sú triedy pre komunikáciu s treťou stranou. Jedná sa o triedy typu *request* a *response*. Tie reprezentujú požiadavky na API tretej strany a následné odpovede. Na obrázku 7.9 je možné vidieť prehľad všetkých tried, ktoré sa používajú na komunikáciu medzi treťou stranou a aplikáciou. Triedy sú vytvorené na základe dokumentácie, ktorá bola poskytnutá treťou stranou. Jedná sa o triedy:

- `GetCustomersRequestDto`,
- `IndividualPersonCustomerData`,
- `GetOrdersRequestDto`,
- `LoginRequestDto`,
- `RefreshTokenRequestDto`,
- `CustomersResultDto`,
- `OrderDetailResponseDto`,
- `OrdersResponseDto`,

- `CountResponseDto`,
- `LoginResponseDto`,
- `RefreshTokenResponseDto`.



Obrázek 7.9 Model tried pre komunikáciu s treťou stranou

(zdroj: vlastný)

Tieto triedy sa používajú v mikroservise `Resource`, ktorá obsahuje celú logiku komunikácie a spracovania dát z tretej strany. Jednotlivé vzťahy medzi servisami starajúcimi sa o celú komunikáciu a spracovanie je možné vidieť na obrázku 7.10. Skladá sa zo servís na prihlásenie

sa do API tretej strany, obnovovanie tohto spojenia cez *recaptcha token*, volania API tretej strany a spracovania týchto dát.



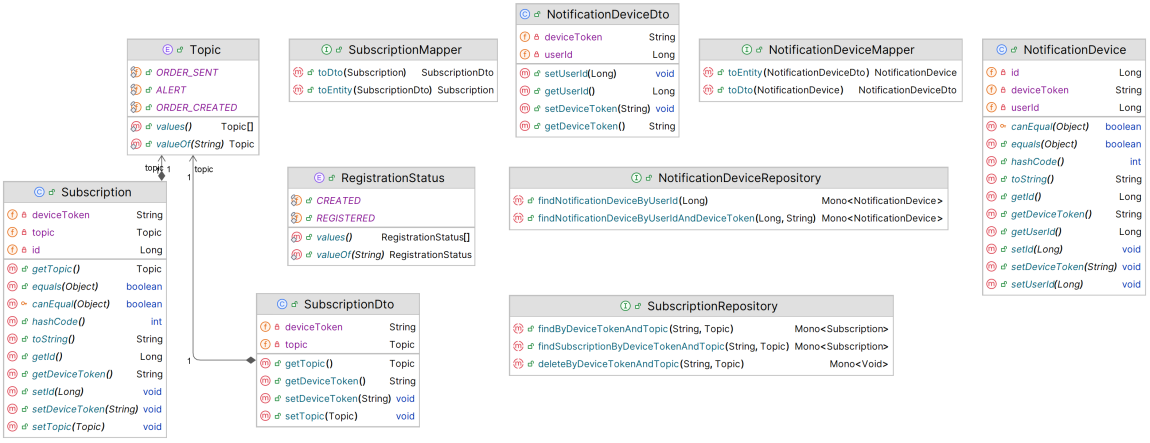
Obrázek 7.10 Model servís pre komunikáciu s treťou stranou

(zdroj: vlastný)

Notifikačná mikroservisa

Táto mikroservisa sa stará o posielanie push notifikácií. Po prijatí správy cez Apache Kafka spracuje požiadavku (o aký typ notifikácie sa jedná) a odošle ju. Na ukladanie a prácu s notifikáciami a odoberaním rôznych tém (*topics*) slúžia triedy `NotificationDevice`, `NotificationDeviceDto`, `Subscription` a `SubscriptionDto`. Pre vyťahovanie dát z databázy slúži `NotificationDeviceRepository` a `SubscriptionRepository`. Pre mapovanie z `NotificationDevice` do `NotificationDeviceDto` slúži `NotificationMapper`.

Rovnako má aj `Subscription` a `SubscriptionDto` triedu, ktorá sa stará o mapovanie `SubscriptionMapper`. Všetky triedy s ich atribútmi a metódami je možné vidieť na obrázku 7.11. Tento model sa používa v servisách, ktoré sa starajú o registráciu zariadenia po-



Obrázek 7.11 Model push notifikácií

(zdroj: vlastný)

užívateľa, prihlásenie k odberu jednotlivých tém a samotné posielanie notifikácií. Prehľad celkovej štruktúry tried pre posielanie notifikácií je vidieť na obrázku 7.12.



Obrázek 7.12 Model na posielanie push notifikácií

(zdroj: vlastný)

8 POPIS IMPLEMENTÁCIE

Implementácia rozšírenia systému výdajne zdravotníckych pomôcok pre diabetikov pre firmu Diatop prebiehala v niekoľkých fázach. Prvou fázou bolo rozšírenie registrácie o verifikáciu emailovej adresy. Nasledovalo zabezpečenie obnovy hesla v prípade zabudnutia. Nasledovalo šifrovanie citlivých údajov a implementácia mobilnej aplikácie a jej spojenie s backend časťou. Implementácia mobilnej aplikácie prebiehala v dvoch častiach, používateľská časť a admin časť. K tomu patrí aj tvorba novej mikroservisy pre push notifikácie. Následne bola aplikácia napojená na tretiu stranu a z dát od tretej strany boli tvorené štatistiky.

8.1 Verifikácia emailovej adresy

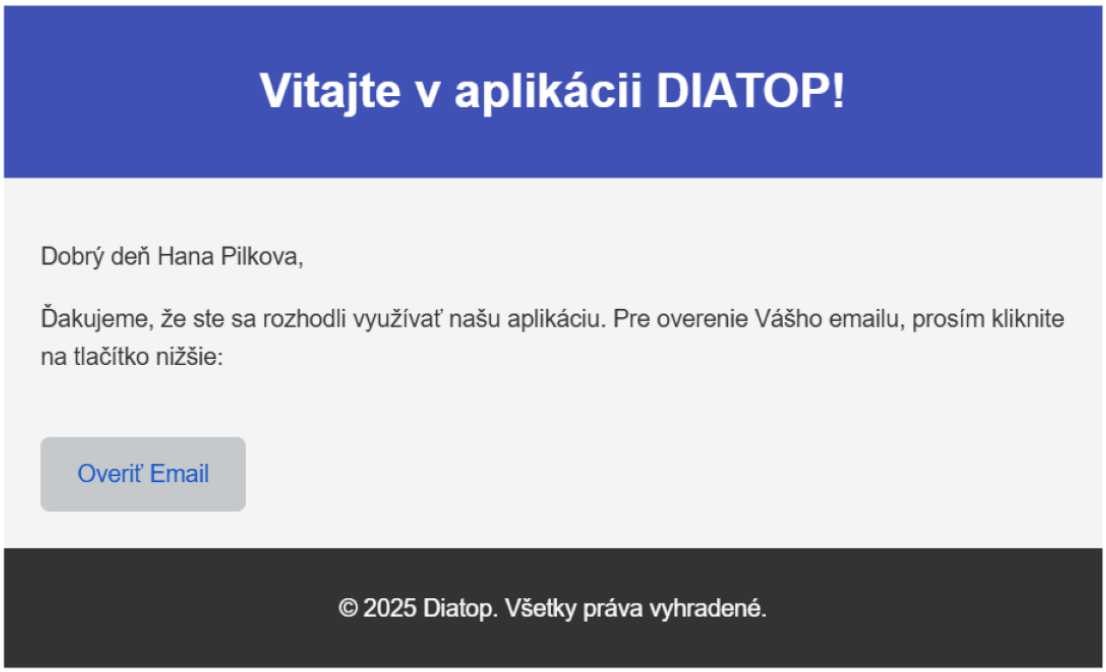
Verifikácia emailovej adresy je zabezpečená pomocou tokenu. Tento token má určitú platnosť, v databáze pre každého používateľa ako `verification_token_expiration`. Či je používateľova emailová adresa overená, sa kontroluje cez atribút `verified`. Používateľ overuje emailovú adresu cez email, ktorý mu je poslaný.

Proces verifikácie

Proces verifikácie má niekoľko krokov. Najprv je nutné, aby používateľ vyplnil registračný formulár. Následne sa zavolá metóda `registerUser`, kedy sa vygeneruje token, nastaví sa jeho expirácia a skontroluje sa, že daný používateľ už neexistuje. Pokiaľ neexistuje, je pridaný, ale nie overený. Nasleduje poslanie emailu používateľovi. Celý proces je možné vidieť v ukážke kódu 8.1.

```
1  public Mono<Boolean> registerUser(UserDto userDto,
       ServerHttpRequest request) {
2    userDto.setVerified(false);
3    String token = UUID.randomUUID().toString();
4    userDto.setVerificationToken(token);
5    userDto.setTokenExpirationDate(LocalDateTime.now().
       plusHours(24));
6    return userService.addUser(userDto)
7      .hasElement()
8      .flatMap(userExists -> {
9        if (!userExists) {
10         return Mono.just(false);
11       }
12       return sendVerificationEmailToUser(request, userDto);
13     });
14 }
```

Výpis 8.1 Verifikačný proces: registrácia (zdroj: vlastný)



Obrázek 8.1 Overovací email

(zdroj: vlastný)

Po kliknutí na tlačítko *Overiť Email*, je používateľ presmerovaný na stránku, kde je mu oznámené, či verifikácia prebehla úspešne, alebo nie. Neúspech je zväčša spôsobený vypršaním platnej doby tokenu. Overenie je možné vidieť v kóde 8.2.

```
1  public Mono<Boolean> verifyUser(String token) {
2    return userRepository.findUserByVerificationToken(token)
3      .flatMap(user -> {
4        if (user != null) {
5          if (user.getVerified()) {
6            return Mono.just(true);
7          }
8          if (user.getTokenExpirationDate().isAfter(
               LocalDateTime.now())) {
9            user.setVerified(true);
10           return userRepository.save(user).hasElement();
11         }
12       }
13       return Mono.just(false);
14     });
15 }
```

Výpis 8.2 Verifikačný proces: overenie (zdroj: vlastný)

Pokiaľ doba platnosti tokenu vypršala, používateľ môže rovno zažiadať o zaslanie nového tokenu. To znamená, že celý proces generovania, pridelenia, nastavenia platnosti a odoslania emailu sa zopakuje (kód 8.3).

```
1  public Mono<Boolean> sentVerificationToken(String token,
       ServerHttpRequest request) {
2    return userRepository.findUserByVerificationToken(token)
3      .flatMap(user -> {
4        if (user != null) {
5          updateUserToken(user);
6          return userRepository.save(user)
7            .flatMap(savedUser -> {
8              UserDto userDto = userMapper.toDto(savedUser);
9              return sendVerificationEmailToUser(request,
                   userDto);
10           });
11       }
12       return Mono.just(false);
13     });
14 }
15
16 private void updateUserToken(User user) {
17   user.setVerified(false);
18   String newToken = UUID.randomUUID().toString();
19   user.setVerificationToken(newToken);
20   user.setTokenExpirationDate(LocalDateTime.now().plusHours
       (24));
21 }
```

Výpis 8.3 Verifikačný proces: obnovenie verifikačného tokenu (zdroj: vlastný)

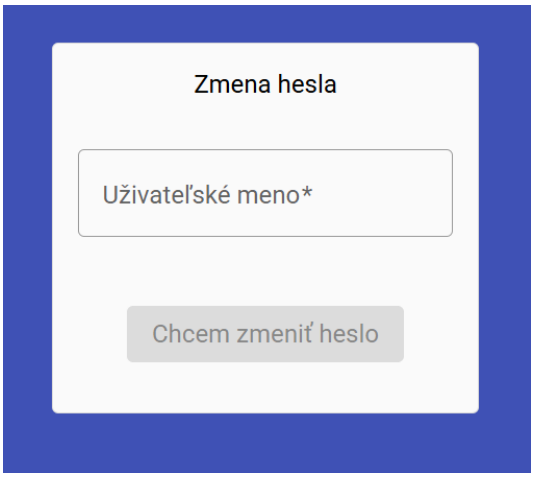
8.2 Obnova zabudnutého hesla

Zabudnutie hesla je bežnou záležitosťou pri používaní aplikácie, a preto je nutné túto možnosť ošetriť. V prípade zabudnutia hesla používateľ na stránke/obrazovke prihlásenia môže kliknúť na tlačítko *Zabudli ste heslo?*. Pokiaľ tak urobí, je presmerovaný na obrazovku, kde musí zadať svoje používateľské meno – emailovú adresu. Po tom, čo používateľ zadá emailovú adresu a zažiada o obnovenie hesla, príde mu email. Token pre obnovenie sa ukladá do Redis (8.4).

```
1  public Mono<Boolean> resetPasswordRequest(String userEmail,
       ServerHttpRequest request) {
2    return userRepository.findUserByEmail(userEmail)
3      .flatMap(user -> {
4        String token = UUID.randomUUID().toString();
5        Duration ttl = Duration.ofMinutes(15);
6        String resetLink = buildResetLink(request, token);
7        return redisTokenService.storeResetToken(token, user.
             getId().toString(), ttl)
8          .then(emailService.sendResetLink(userEmail,
             resetLink))
9          .thenReturn(true);})
10     .switchIfEmpty(Mono.just(false));
11 }
```

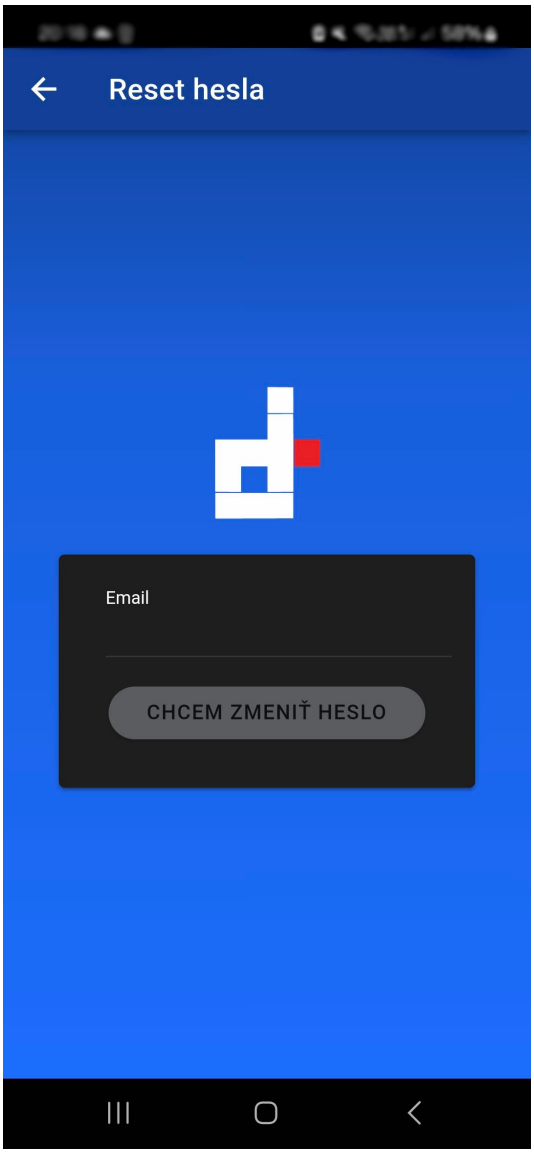
Výpis 8.4 Žiadosť o zmenu hesla (zdroj: vlastný)

Obrazovku žiadosti o zmenu hesla pre webovú aplikáciu je možné vidieť na obrázku 8.2 a pre mobil 8.3. Email zaslaný používateľovi ho odkáže na obrazovku, kde môže zadať



Obrázek 8.2 Žiadosť o zmenu hesla - webová aplikácia

(zdroj: vlastný)



Obrázek 8.3 Žiadosť o zmenu hesla - mobilná aplikácia

(zdroj: vlastný)

nové heslo. Tam sa overuje token, ktorý mu bol pridelený. Ak je token v poriadku, môže používateľ zmeniť heslo. Po tom, ako nové heslo zadá a odošle, je jeho heslo zmenené aj v systéme. Validácia prebieha v `RedisTokenService` metódou `isResetTokenValid`. Samotnú zmenu hesla zobrazuje ukážka 8.5.

```
public Mono<Boolean> resetPassword(String token, String
    newPassword) {
  return redisTokenService.consumeResetToken(token)
    .flatMap(userId -> userRepository.findUserById(userId))
    .flatMap(user -> {
      String hashed = passwordEncoder.encode(newPassword);
      user.setPassword(hashed);
      return userRepository.save(user).thenReturn(true);
    })
    .defaultIfEmpty(false);
}
```

Výpis 8.5 Zmena hesla (zdroj: vlastný)

Email, ktorý príde používateľovi s možnosťou zmeniť heslo, je možné vidieť na obrázku 8.4.



Obrázek 8.4 Email s možnosťou zmeniť heslo

(zdroj: vlastný)

8.3 Šifrovanie citlivých údajov

Citlivé údaje musia byť maximálne chránené. Šifrovanie týchto dát zabezpečuje ich maximálnu ochranu. V systéme je heslo hashované pomocou `PasswordEncoder`, ktorý používa

algoritmus BCrypt. Rodné číslo je šifrované AES šifrou. Na šifrovanie bola použitá trieda `Cipher` z `javax.crypto.Cipher`. Šifrovanie a dešifrovanie je možné vidieť v 8.6.

```
1  public String encrypt(String plaintext) {
2    try {
3      byte[] iv = new byte[IV_LENGTH];
4      secureRandom.nextBytes(iv);
5
6      Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding")
           ;
7      GCMParameterSpec spec = new GCMParameterSpec(TAG_LENGTH
           , iv);
8      cipher.init(Cipher.ENCRYPT_MODE, secretKey, spec);
9
10     byte[] encrypted = cipher.doFinal(plaintext.getBytes())
           ;
11
12     byte[] combined = new byte[IV_LENGTH + encrypted.length
           ];
13     System.arraycopy(iv, 0, combined, 0, IV_LENGTH);
14     System.arraycopy(encrypted, 0, combined, IV_LENGTH,
           encrypted.length);
15
16     return Base64.getEncoder().encodeToString(combined);
17   } catch (Exception e) {
18     throw new RuntimeException("Encryption failed", e);
19   }
20 }
21
22 public String decrypt(String base64Ciphertext) {
23   try {
24     byte[] combined = Base64.getDecoder().decode(
           base64Ciphertext);
25
26     byte[] iv = new byte[IV_LENGTH];
27     byte[] ciphertext = new byte[combined.length - IV_
           LENGTH];
28     System.arraycopy(combined, 0, iv, 0, IV_LENGTH);
29     System.arraycopy(combined, IV_LENGTH, ciphertext, 0,
           ciphertext.length);
30
31     Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding")
           ;
32     GCMParameterSpec spec = new GCMParameterSpec(TAG_LENGTH
           , iv);
33     cipher.init(Cipher.DECRYPT_MODE, secretKey, spec);
34
35     byte[] decrypted = cipher.doFinal(ciphertext);
36     return new String(decrypted);
37   } catch (Exception e) {
38     throw new RuntimeException("Decryption failed", e);
39   }
40 }
```

Výpis 8.6 Šifrovanie a dešifrovanie (zdroj: vlastný)

Šifrovanie sa používa pri ukladaní nového používateľa, zmeny dát používateľa a pri odosielaní emailu výdajni s novou objednávkou. Avšak, servisa pre posielanie emailov je v mikroservise `Resource` a šifrovanie (keďže sa jedná o implementáciu spojenú s používateľom) v mikroservise `Gateway`. Riešenie je pomocou `UserDecryptionRequestListener`, ktorý prijíma správy na dešifrovanie rodného čísla. V žiadosti o dešifrovanie sa nachádza `userId`, servisa `UserService` sa postará o nájdenie používateľa na základe jeho id v databáze, de-

šifruje rodné číslo a vráti dáta používateľa ako objekt triedy `UserDto`. Tam už sa nachádzajú údaje namapované z `User` do `UserDto` s dešifrovaným rodným číslom. Implementáciu je možné vidieť v 8.7.

```
@KafkaListener(topics = "user-decryption-request", groupId
    = "gateway-decryption-group")
public void handleDecryption(UserDecryptionRequest request)
    {
  UserDecryptionResponse response = new
     UserDecryptionResponse();
  response.setRequestId(request.getRequestId());
  UserDto userDto = userService.getUserById(request.
     getUserId())
    .switchIfEmpty(Mono.error(new RuntimeException("User
       not found")))
    .block();
  response.setUser(userDto);
  kafkaTemplate.send("user-decryption-response", response);
}
```

Výpis 8.7 Žiadosť o dešifrovanie informácií o používateľovi (zdroj: vlastný)

8.4 Mobilná aplikácia

Mobilná aplikácia má dve hlavné časti - časť pre štandardného používateľa a administrátorskú časť. Tie sú oddelené tak, že pri prihlásení sa na základe roly rozhodne, ktorá časť sa zobrazí. Navigácia naprieč aplikáciou, či už v používateľskej alebo administrátorskej časti, je riešená pomocou `Routes`, ktoré definujú jednotlivé stránky (pages) spojené s cestou (path).

8.4.1 Prihlásenie

Používateľ, respektíve administrátor (v tejto kapitole ďalej len používateľ) sa môže prihlásiť niekoľkými spôsobmi. Prvotné prihlásenie musí prebehnúť cez používateľské meno (emailová adresa) a heslo. Po úspešnom prihlásení môže ďalej používať biometrické prihlasovanie, pokiaľ to jeho zariadenie umožňuje a túto možnosť bude mať v nastavení oprávnení mobilného zariadenia povolenú.

Počas prvotného prihlásenia sa používateľské meno a heslo uložia do bezpečného úložiska. Toto úložisko je plugin `@aparajita/capacitor-secure-storage`, ktorý sa stará o bezpečné uloženie natívnymi funkciami jednotlivých platforiem (8.8).

```
authenticate(credentials: Credentials, callback: (isAdmin:
    boolean) => void) {
  this.httpService.login(credentials).then(async response
     => {
```

```
        this.authenticated = response && response.name !=
            null;
        const isAdmin = this.authenticated && response.roles.
            length > 0 && response.roles.indexOf('ROLE_ADMIN')
             > -1;

        if (this.authenticated) {
          await SecureStorage.set('username', credentials.
              username);
          await SecureStorage.set('password', credentials.
              password);

          await this.userService.getUser(isAdmin);
          await this.httpService.get('/notification/ping');
          await this.notificationService.
              initializePushNotifications(this.userStore.user
              ().id);
        }

        return callback && callback(isAdmin);
      }).catch(error => {
        console.warn('Authentication failed: ', error.
            toString());
      });
  }
```

Výpis 8.8 Prvotné prihlásenie (zdroj: vlastný)

Ďalšie prihlasovania je možné robiť pomocou biometriky. Najprv sa skontroluje, či je táto možnosť povolená a ak áno, spustí sa biometrické prihlasovanie pomocou pluginu `@aparajita/capacitor-biometric-auth`. V realite po biometrickom prihlásení (biometrika používateľa sa zhoduje a zariadenie potvrdí zhodu) sa stále volá klasické prihlasovanie volaním backendu s prihlasovacími údajmi používateľa (8.9).

```
  async authenticateWithBiometric(): Promise<boolean> {
    const isBiometricAvailable = await this.
        isBiometricAvailable();
    if (!isBiometricAvailable) {
      console.info('Biometric authentication is not available
          on this device');
      return false;
    }

    const isAuthenticated = await this.
        authenticateWithBiometrics();
    if (!isAuthenticated) {
      console.warn('Biometric authentication failed');
      return false;
    }
    return true;
  }

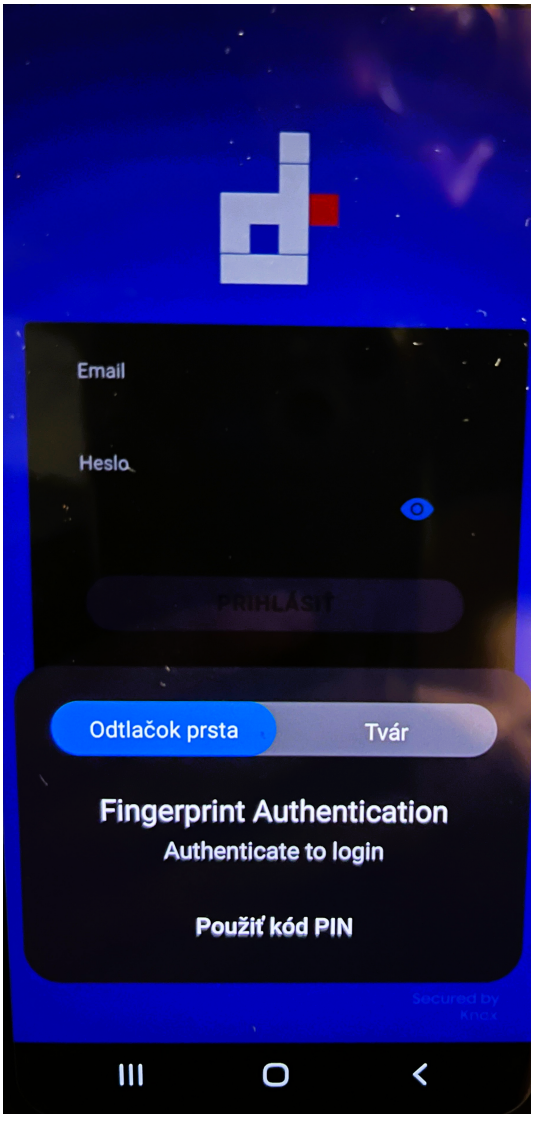
    async authenticateWithBiometrics(): Promise<boolean> {
    try {
      await BiometricAuth.authenticate({
        reason: 'Authenticate to login',
        allowDeviceCredential: true, // Allows fallback to
            PIN/Pattern on Android
        iosFallbackTitle: 'Use Passcode', // Fallback button
            text for iOS
        cancelTitle: 'Zrušiť'
      });

      return true;
    } catch (error) {
      if (error instanceof Error) {
        console.error('Biometric authentication failed:',
            error.message);
```

```
29
30          if ((error as any).code === BiometryErrorType.
                userCancel) {
31            console.warn('User canceled biometric
                authentication');
32          }
33        }
34        return false;
35      }
36    }
```

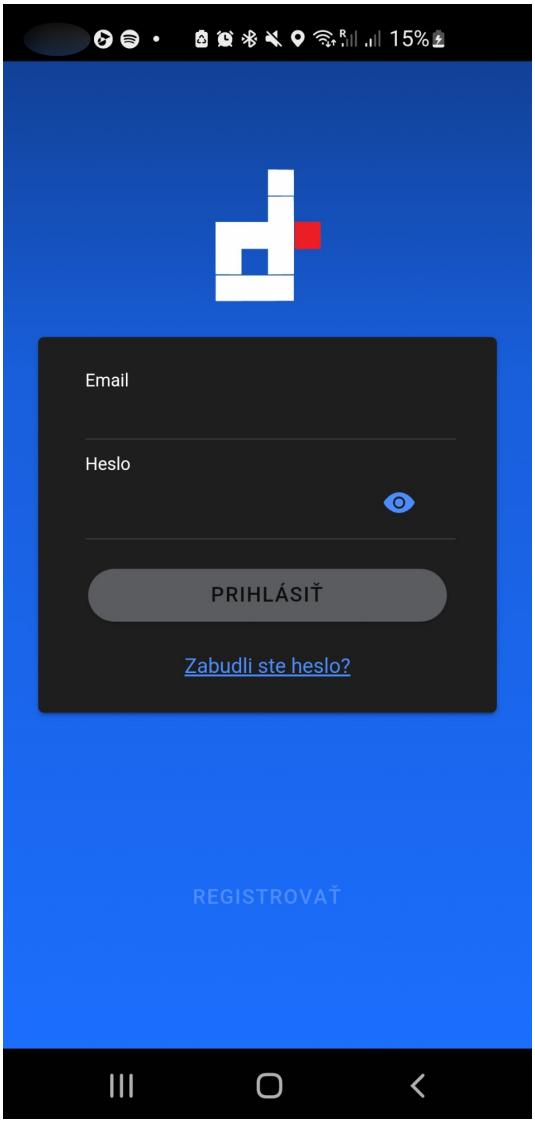
Výpis 8.9 Prihlasovanie biometrikou (zdroj: vlastný)

Obrazovky prihlásenia je možné vidieť na obrázkoch 8.6 (autentifikácia používateľským menom a heslom) a 8.5 (autentifikácia biometrikou).



Obrázek 8.5 Prihlasovanie cez biometriku

(zdroj: vlastný)

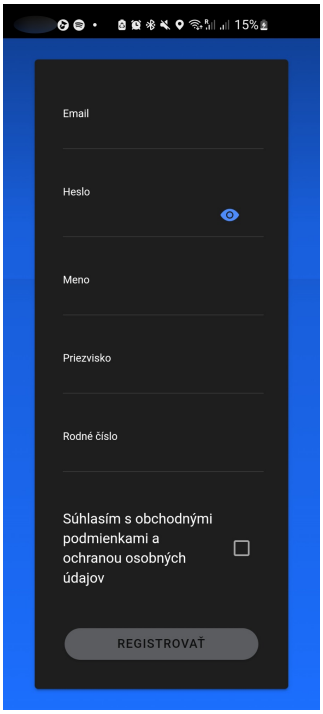


Obrázek 8.6 Prihlasovanie

(zdroj: vlastný)

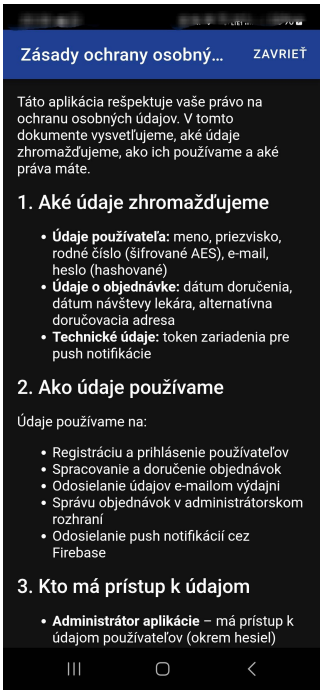
8.4.2 Registrácia

Registrácia používateľa je klasickým formulárom, kedy je nutné vyplniť všetky povinné polia a zaškrtnúť súhlas s obchodnými podmienkami a ochranou osobných údajov. Následne



Obrázek 8.7 Registrácia

(zdroj: vlastný)



Obrázek 8.8 Zásady ochrany osobných údajov a obchodné podmienky

(zdroj: vlastný)

je možné kliknúť na tlačítko registrácie. Súhlas s obchodnými podmienkami a ochranou osobných údajov si môže používateľ prečítať. Text je zobrazený v samostatnom okne (8.10).

```
<ion-modal trigger="open-modal">
    <ng-template>
      <ion-header>
        <ion-toolbar>
          <ion-title>Zásady ochrany osobných údajov</ion-title>
          <ion-buttons slot="end">
            <ion-button (click)="close()">Zavrieť</ion-button>
          </ion-buttons>
        </ion-toolbar>
      </ion-header>
      <ion-content class="ion-padding">
        <app-privacy-policy-text></app-privacy-policy-text>
      </ion-content>
    </ng-template>
</ion-modal>
```

Výpis 8.10 Okno s podmienkami ochrany osobných údajov a obchodnými podmienkami (zdroj: vlastný)

8.4.3 Používateľská aplikácia

Po prihlásení používateľ vstupuje do aplikácie. Tá sa skladá z navigačného panelu, ktorý sa nachádza v spodnej časti. Obsahuje záložky *Domov*, *Objednávka*, *História* a *Účet*. Navigácia je zabezpečená pomocou `Routes` (8.11) a komponenty záložiek `TabComponents`.

```
{
    path: 'ui',
    component: TabsComponent,
    children: [
      { path: 'home', loadComponent: () => import('./user/
        home/home.page').then((m) => m.HomePage) },
      { path: 'order', loadComponent: () => import('./user/
        order/order.page').then((m) => m.OrderPage) },
      { path: 'history', loadComponent: () => import('./user/
        history/history.page').then((m) => m.HistoryPage) },
      { path: 'account', loadComponent: () => import('./user/
        account/account.page').then((m) => m.AccountPage)},
      { path: 'account/profile', loadComponent: () => import(
        './user/account/profile/profile.page').then((m) => m
        .ProfilePage) },
      { path: 'account/notifications', loadComponent: () =>
        import('./user/account/notifications/notifications.
        page').then((m) => m.NotificationsPage) },
      { path: 'account/customization', loadComponent: () =>
        import('./user/account/customization/customization.
        page').then((m) => m.CustomizationPage) },
      { path: 'account/privacy', loadComponent: () => import(
        './user/account/privacy/privacy.page').then((m) => m
        .PrivacyPage) },
      {
        path: '',
        redirectTo: '/ui/home',
        pathMatch: 'full',
      },
    ]
  }
```

Výpis 8.11 Navigácia v používateľskej časti (zdroj: vlastný)

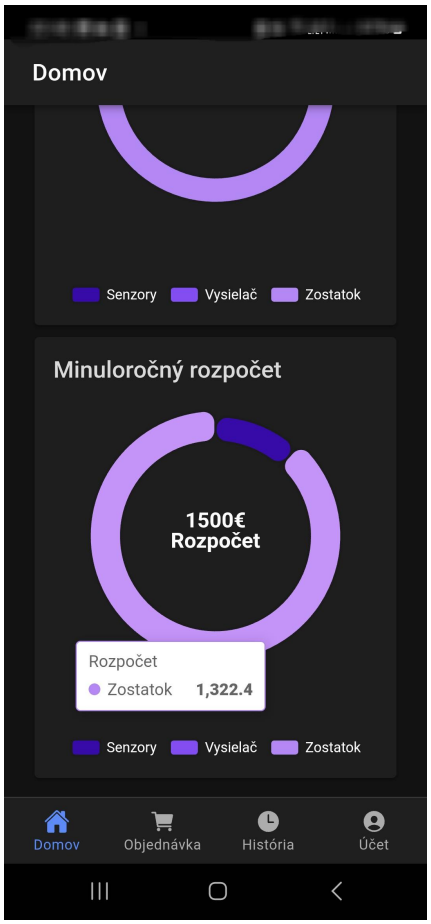
Domov

Prvá obrazovka aplikácie - *Domov* ponúka prehľad štatistík o tom, ako používateľ čerpá (za daný rok) a čerpal (minulý rok) peniaze na zdravotnícke pomôcky. Týmito pomôckami sú len senzory a vysielač. Štatistiky sú tvorené z dát zo systému tretej strany, ktorý výdajňa používa na zadávanie objednávok. Zobrazenie štatistík je pomocou koláčových grafov. Grafy sú z knižnice `ngx-echarts` a `echarts`, ktoré dokopy ponúkajú jednoducho použiteľné a upravovateľné grafy. Obrazovku *Domov* je možné vidieť na obrázkoch 8.9 a 8.10.



Obrázek 8.9 Domov

(zdroj: vlastný)



Obrázek 8.10 Domov

(zdroj: vlastný)

Zobrazenie rozpočtu je pomocou komponenty `pie-chart.component`. Využíva `echarts` a nastavuje reprezentáciu dát, farby a zmenu farieb pri zmene režimu z tmavého na svetlý a opačne. Táto zmena sa zachytáva pomocou `effect`, pretože preferencia používateľa je v signal store `CustomizationStore`. Tento proces je možné vidieť v ukážke kódu 8.12.

```
1  private init() {
2    effect(() => {
3      const dark = this.customizationStore.darkMode();
4      this.textColor = dark ? '#ffffff' : '#262626';
5      this.pieBorderColor = dark ? '#1c1c1d' : '#ffffff';
6      if (this.data().items != null) {
7        this.initChartOptions();
8      }
9    });
10 }
```

Výpis 8.12 Zachytenie zmeny režimu (zdroj: vlastný)

Nastavenie dát do echarts je pomocou JSON štruktúry. Je nutné dáta vložiť do štruktúry, ktorú echarts používa (8.13).

```
1  data: this.data().items.map(item => ({
2    value: item.value,
3    name: item.description,
4    itemStyle: { color: item.color }
5  }))
```

Výpis 8.13 Spracovanie dát do echarts (zdroj: vlastný)

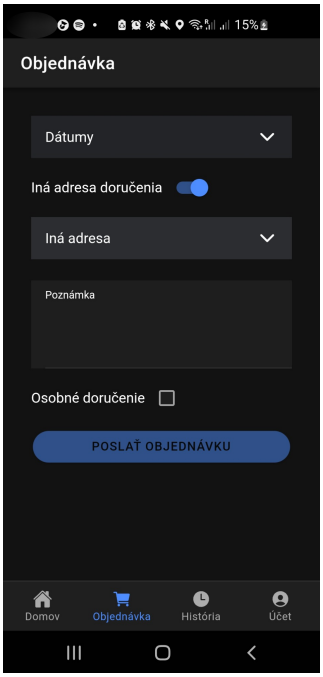
Rovnako je možné nastaviť vzhľad legendy, vyskakovacieho okna pre detail či grafického zobrazenia celkovo.

Vytvorenie objednávky

Vytvorenie objednávky je kľúčovou funkciou aplikácie. Objednávka má povinné polia, ktoré musí používateľ zadať ako *dátum návštevy lekára* a *dátum doručenia*. Pokiaľ sa používateľ rozhodne, že chce doručiť objednávku inam, ako je zvykom, musí zadať všetky informácie o adrese doručenia. Osobné doručenie je možné len v okolí Bratislavy, na čo je upozornený vyskakovacím oknom. Doručovanie nie je možné počas víkendov (víkendy sú zakázané v kalendári). Nová objednávka je následne odoslaná na backend, kde je spracovaná (uloženie do databázy). Objednávka sa odosiela do mikroservisy `Resource`, do ktorej ju nasmeruje `Gateway`. Formulár na zadanie novej objednávky je možné vidieť na obrázkoch 8.11, 8.12, 8.13 a 8.14.

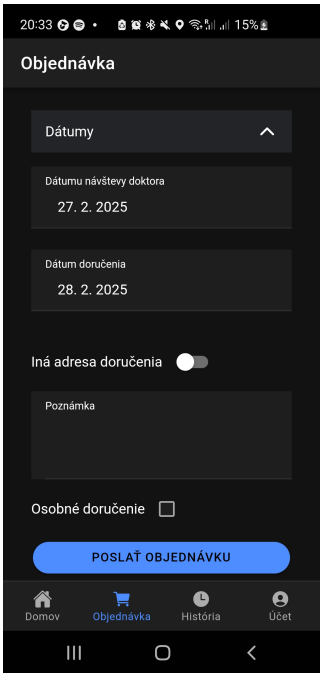
```
1 async createOrder(order: Order): Promise<void> {
2 order.userId = this.userStore.user().id;
3
4 this.http.post<Order>('/resource/addOrder', order, false).
    then(order => {
5   if (order) {
6     this.orderStore.addOrder(order).then();
7     this.showToast('Objednávka odoslaná!', 'success').then();
8   } else {
9     this.showToast('Niečo sa pokazilo. Skúste to ešte raz.',
        'danger').then();
10  }
11 }).catch (error => {
12   console.error('Order error: ', error);
13   this.showToast('Niečo sa pokazilo. Skúste to ešte raz.', '
       danger').then();
14 });
15 }
```

Výpis 8.14 Vytvorenie objednávky (zdroj: vlastný)



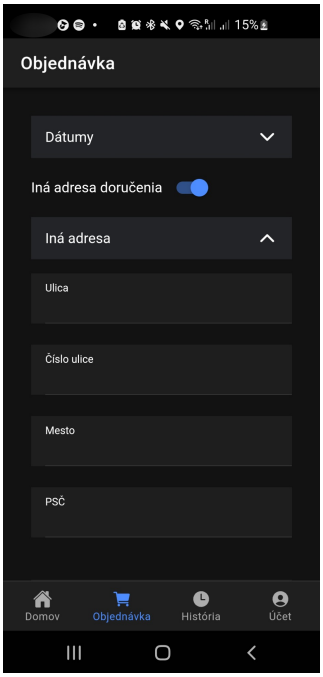
Obrázek 8.11 Vytváranie objednávky

(zdroj: vlastný)



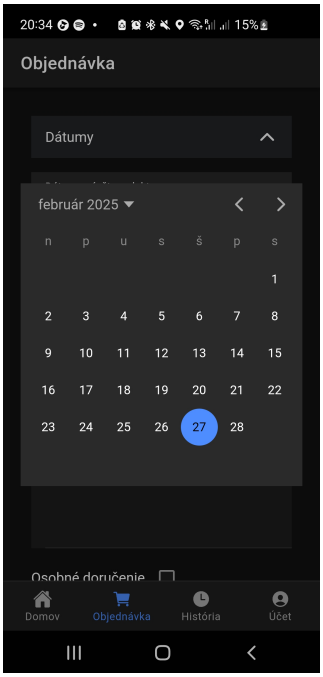
Obrázek 8.12 Vytváranie objednávky

(zdroj: vlastný)



Obrázek 8.13 Vytváranie objednávky

(zdroj: vlastný)



Obrázek 8.14 Vytváranie objednávky

(zdroj: vlastný)

História objednávok

Objednávky sú okrem databázy ukladané počas behu aplikácie do `order.store` ktorý je `signalStore` od `@ngrx/signals`. Počas používania aplikácie sú objednávky používateľa vždy dostupné v tomto úložisku a je jednoduchšie k nim pristupovať z jedného miesta. Po tom, čo je aplikácia vypnutá/uspaná, úložisko zanikne (dáta sú vymazané) a je nutné načítať ich znovu. NgRx Signal Store prináša jednoduchší a prehľadnejší spôsob práce so stavom v Angular aplikáciách. Signal Store umožňuje pristupovať k dátam cez tzv. signály, ktoré sa správajú predvídateľne a reagujú okamžite na zmeny. Nie je nutné riešiť odoberanie sa z dátových prúdov (unsubscribe) a zároveň sú k dispozícii všetky výhody tradičného NgRx, ako je správa akcií, efektov a reducerov.

Úložisko objednávok `order.store` sa vytvára s počiatočným stavom `initialState`. Môže obsahovať `withComputed`, ktorý vytvorí odvodený (computed) signál z existujúcich stavov v signal store. V ukážke 8.15 je to `emptyOrders` na určenie, či existujú nejaké objednávky.

```
withMethods((store) => ({
    async loadOrders(orders: Order[], year: number) {
      const ordersByYearArray = [...store.orders()];
      orders.sort((a, b) => new Date(b.deliveryDate).getTime
         () - new Date(a.deliveryDate).getTime());

      let existing = ordersByYearArray.find(o => o.year ===
         year);
      if (existing) {
        const combined = [...existing.orders, ...orders];
        const unique = Array.from(new Map(combined.map(o => [
           o.id, o])).values());
        unique.sort((a, b) => new Date(b.deliveryDate).
           getTime() - new Date(a.deliveryDate).getTime());
        existing.orders = unique;
      } else {
        ordersByYearArray.push(new OrderByYear(year, orders))
           ;
      }

      ordersByYearArray.sort((a, b) => b.year - a.year);
      patchState(store, {orders: ordersByYearArray});
    }
})
```

Výpis 8.15 order.store vytvorenie (zdroj: vlastný)

Rovnako je možné používať metódy vo forme `withMethods`, napríklad metóda `loadOrders`, ktorá načíta objednávky v danom roku a uloží ich do signal store.

```
async loadOrders(orders: Order[], year: number) {
      const ordersByYearArray = [...store.orders()];
      patchState(store, {orders: ordersByYearArray});
    }
```

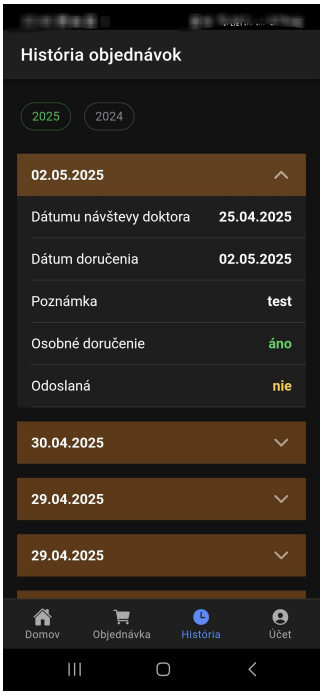
Výpis 8.16 order.store vytvorenie (zdroj: vlastný)

Ako je možné vidieť v `order.store`, objednávky sa načítavajú po rokoch. Je to z dôvodu optimalizácie aplikácie. Na obrazovke *Objednávky* sú *chips* (8.17) - tlačítka s rokom, takže používateľ najprv vidí aktuálny rok a objednávky v danom roku. A teda, objednávky sa načítavajú postupne podľa roku, ktorý si používateľ zvolí a nezaťažuje sa tak aplikácia.

```
<ion-chip outline *ngFor="let year of years"
          [color]="activeYear() === year ? 'success' : '
            medium'"
          (click)="onYearSwitch(year)">
  {{ year }}
</ion-chip>
```

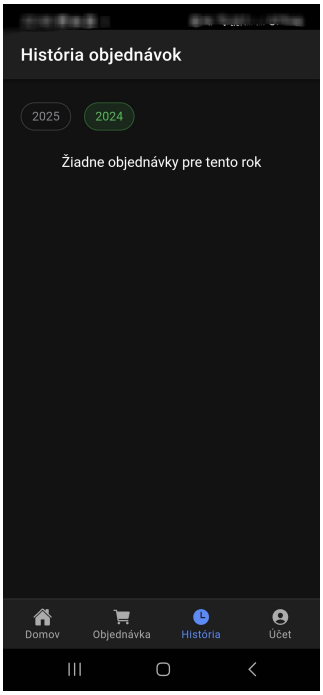
Výpis 8.17 Tlačítka pre históriu podľa rokov (zdroj: vlastný)

Na obrázkoch 8.15 a 8.16 je zobrazená obrazovka histórie. Každá objednávka je označená



Obrázek 8.15 História objednávok

(zdroj: vlastný)



Obrázek 8.16 História objednávky

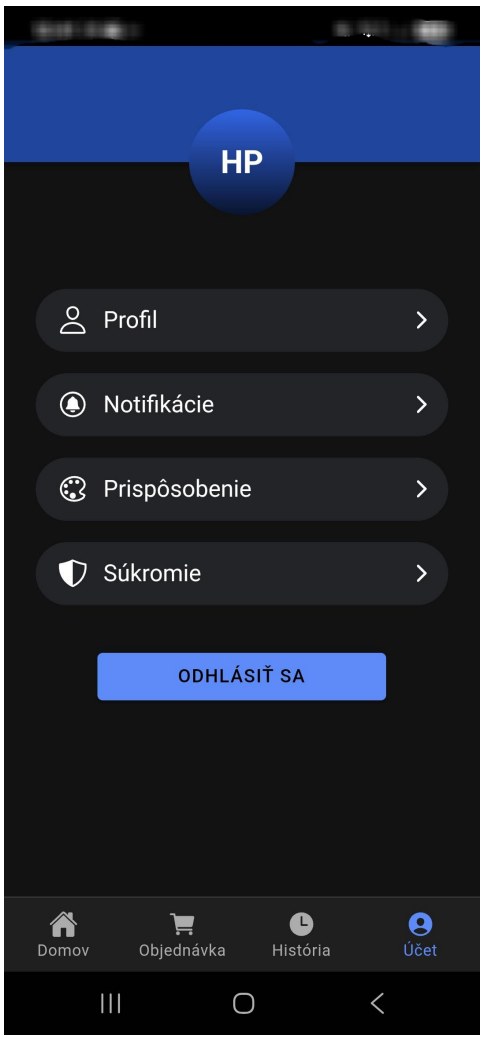
(zdroj: vlastný)

dátumom doručenia a farbou. Pokiaľ je farba oranžová, objednávka ešte nebola odoslaná,

odoslaná objednávka je zelenou farbou. Pri každej objednávke je možné otvoriť jej detail. Detailné zobrazenie ponúka základné informácie o objednávke.

Používateľský profil

Používateľský profil dáva možnosť používateľovi nastaviť jednotlivé časti aplikácie a jeho preferencie. V hornej časti je vždy prítomný kruh s iniciálkami používateľa. Prvou možnosťou je *Profil*, kde používateľ môže meniť svoje údaje zadané pri registrácii. Nasledujú *Notifikácie*, ktoré umožňujú prihlásiť sa na odber, alebo sa z odberu odhlásiť. *Prispôsobenie* ponúka zmenu vzhľadu - režimu tmavý/svetlý. Poslednou možnosťou je *Súkromie* s textom o obchodných podmienkach a ochrane osobných údajov.



Obrázek 8.17 Účet

(zdroj: vlastný)

Nastavenie účtu

Nastavenie účtu ponúka zmenu osobných údajov. Rodné číslo je pseudoanonymizované, takže je možné ho vidieť len po kliknutí na tlačítko vo forme ikony oka. Heslo je možné meniť, ale používateľ musí potvrdiť túto zmenu heslom alebo biometrikou.

```
changePassword() {  this.userService.changePassword(this.
    passwordForm).then((success) => {
      if (success) {
        this.changePasswordModal.dismiss().then();
        SecureStorage.set('password', this.userStore.user().
            password).then();
        this.userService.showToast('Vaše informácie boli úspe
            šne zmenené', 'success').then();
      } else {
        this.userService.showToast('Niečo sa pokazilo. Skúste
            to ešte raz.', 'danger').then();
      }
    });
}
```

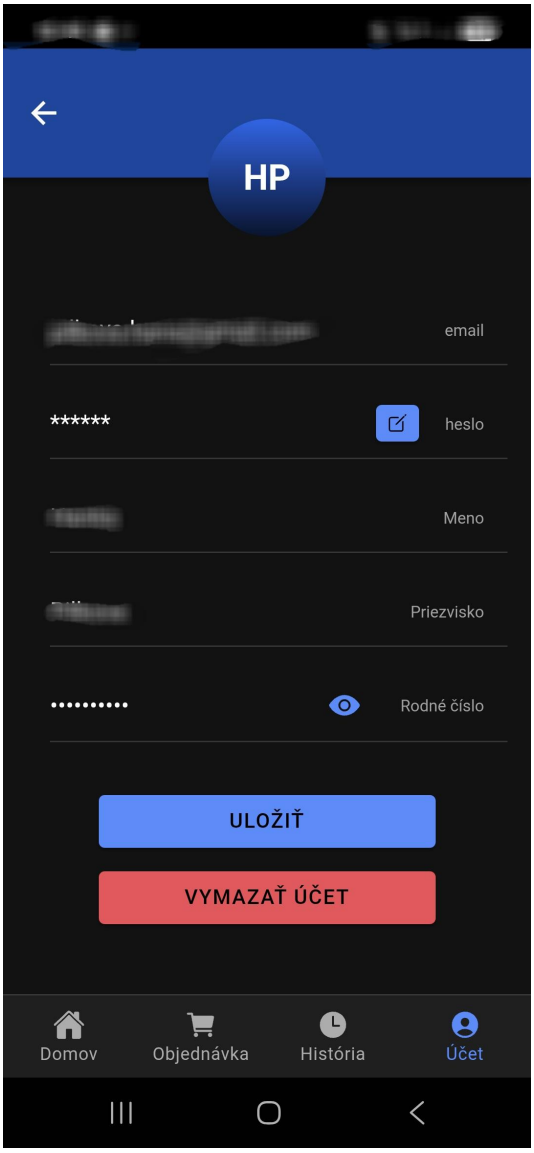
Výpis 8.18 Zmena hesla (zdroj: vlastný)

Zmena hesla sa vykonáva tak, že sa najprv overí, že je používateľ prihlásený. Následne sa z `FormGroup` vytiahne nové heslo a používateľove informácie sa uložia do databázy.

```
async changePassword(password: FormGroup): Promise<boolean>
    {
  const authenticated = await this.bioAuthService.
      authenticateWithBiometric();
  if (authenticated) {
    this._user.password = password.get('password')?.value;
    return this.updateUserProfile();
  } else {
    return false;
  }
}
```

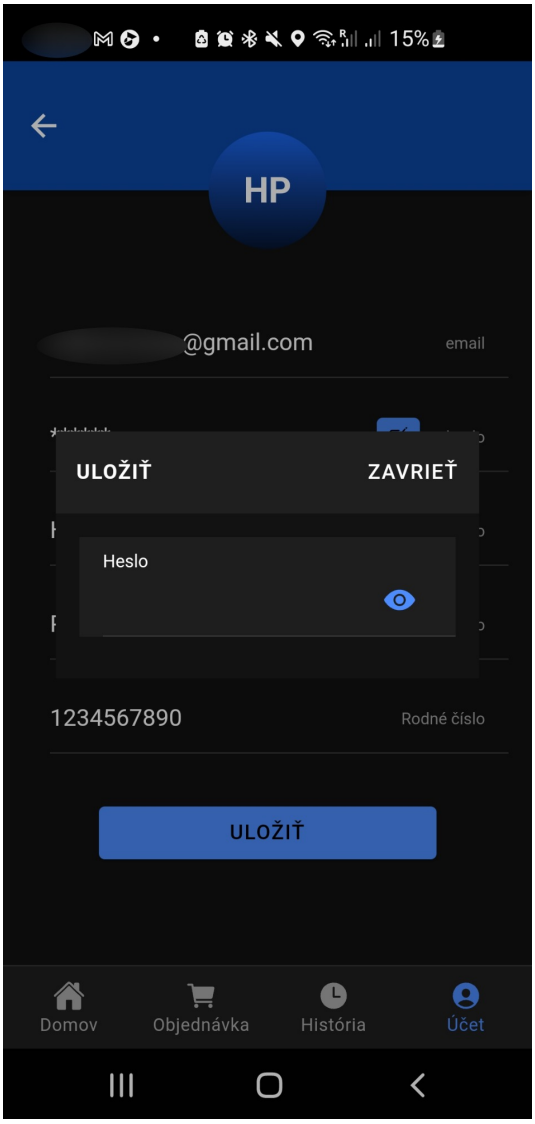
Výpis 8.19 Zmena hesla s overením (zdroj: vlastný)

Obrázok 8.18 zobrazuje, ako vyzerá obrazovka nastavenia účtu - *Profil*. Obrázok 8.19 ukazuje okno na zmenu hesla. Aplikácia nedisponuje takýmito oknami, jediné prípady sú zmena hesla a vymazanie účtu. Tieto operácie sú citlivé a je nutné ich potvrdenie, aby sa nestalo, že si používateľ omylom zmení heslo alebo vymaže účet.



Obrázek 8.18 Profil

(zdroj: vlastný)



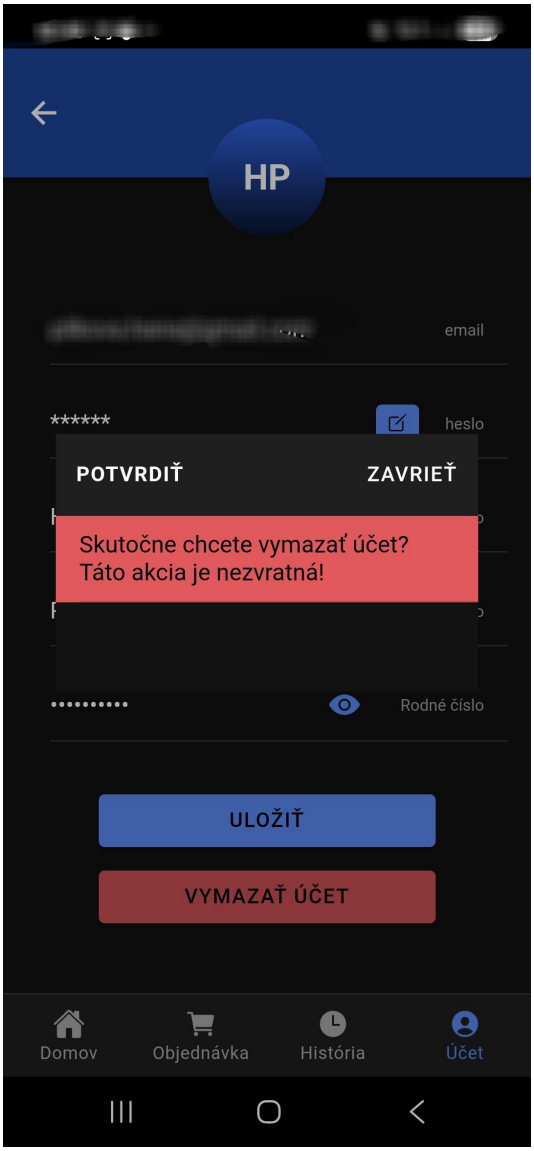
Obrázek 8.19 Zmena hesla

(zdroj: vlastný)

Pokiaľ používateľ zmení iné údaje ako heslo, musí ich potvrdiť tlačítkom *Uložiť*. Rovnako má používateľ možnosť vymazať celý svoj účet. Tlačítko na mazanie účtu je červenou farbou, aby indikovalo nevratnú udalosť. Mazanie účtu znamená, že všetky dáta spojené s používateľom sú vymazané. Takže objednávky používateľa aj jeho registrované zariadenie pre notifikácie sú vymazané z databázy. Rovnako je jeho používateľské meno a heslo vymazané z `SecureStorage` a teda nie je už možné prihlasovať sa biometrikou (8.20). Pokiaľ by sa chcel používateľ opätovne prihlásiť s vymazaným účtom, zobrazí sa chyba (vyobrazené na obrázku 8.21).

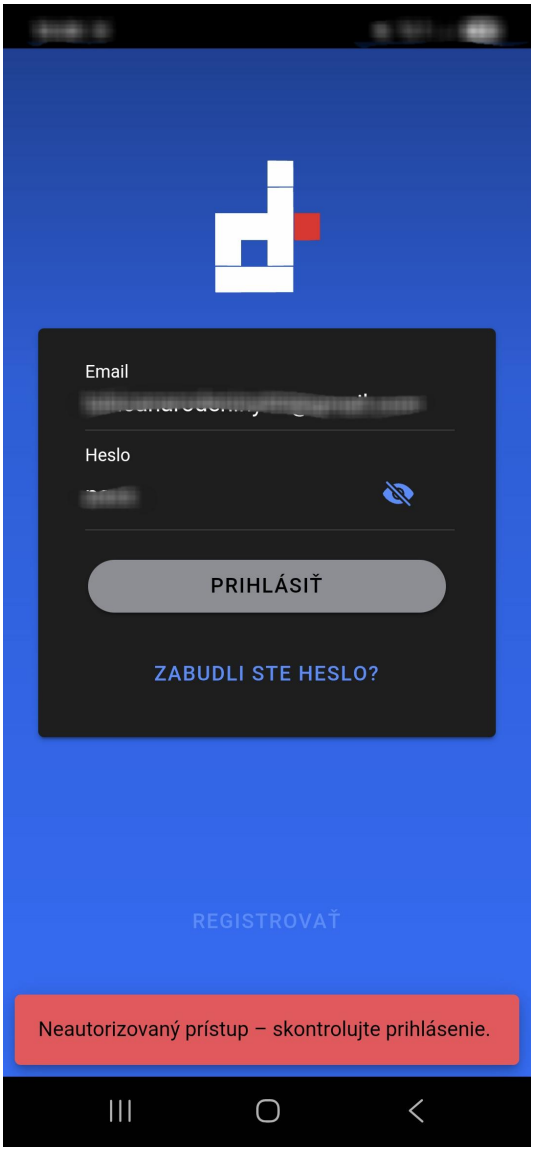
```
removeAccount() {
  this.removeAccountModal.dismiss().then(_ => {
    this.userService.deleteUserAccount().then(async success
      => {
      if (success) {
        await SecureStorage.remove('username');
        await SecureStorage.remove('password');
        this.router.navigate(['/login']).then();
      }
    })
  });
}
```

Výpis 8.20 Vymazanie účtu (zdroj: vlastný)



Obrázek 8.20 Vymazanie účtu

(zdroj: vlastný)



Obrázek 8.21 Prihlásenie s vymazaným účtom

(zdroj: vlastný)

Nastavenie notifikácií

Používateľ má možnosť odhlásiť sa z odberu notifikácií. Notifikácie sú rozdelené na dva typy: *Objednávky* - notifikácia o zmene stavu objednávky a *Upozornenia* - upozornenia rôzneho typu od výdajne. Zmena sa ošetruje pomocou `notification.service.ts` metódou `changeSubbscription`. Volanie tejto metódy je na zmenu stavu *switch* (8.21).

```
changeAlert(event: Event): void {
  this.alert = (event as CustomEvent).detail.checked;
  this.notificationService.changeSubscription(Topic.ALERT,
     this.alert).then();
}

changeOrder(event: Event): void {
  this.order = (event as CustomEvent).detail.checked;
  this.notificationService.changeSubscription(Topic.
     ORDER_SENT, this.order).then();
}
```

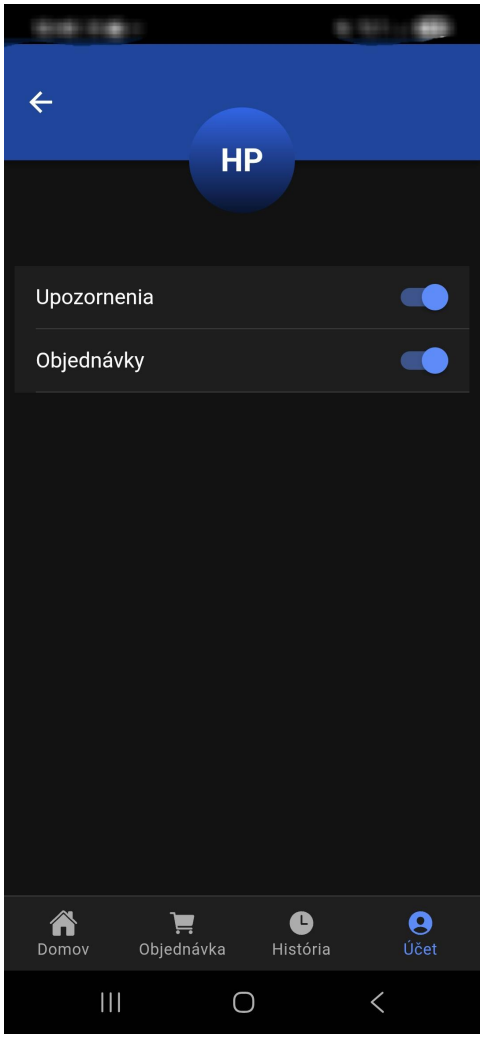
Výpis 8.21 Zmena odoberania notifikácií (zdroj: vlastný)

Zmena odoberania notifikácií sa ukladá do databázy. Na to slúži volanie backendu, konkrétne `NotificationController`. Prihlásenie na odber notifikácií - `/subscribe` alebo odhlásenie - `/unsubscribe`. Na vykonanie tejto akcie slúži `deviceToken` uložený v `Preferences`. Vďaka nemu je možné spárovať používateľa, zariadenie a tému notifikácie. Kód je možné vidieť v ukážke 8.22.

```
async changeSubscription(topic: Topic, subscribed: boolean)
    {
  const deviceToken = await Preferences.get({ key: '
     pushToken' });
  if (deviceToken.value != null) {
    const subscription = new SubscriptionDto();
    subscription.topic = topic;
    subscription.deviceToken = deviceToken.value;
    if (subscribed) {
      this.http.post<void>('/notification/subscribe',
         subscription).then();
    } else {
      this.http.post<void>('/notification/unsubscribe',
         subscription).then();
    }
  }
}
```

Výpis 8.22 Zmena odoberania notifikácií (zdroj: vlastný)

Obrazovka zmeny notifikácií je zobrazená na obrázku 8.22.



Obrázek 8.22 Obrazovka úpravy notifikácií

(zdroj: vlastný)

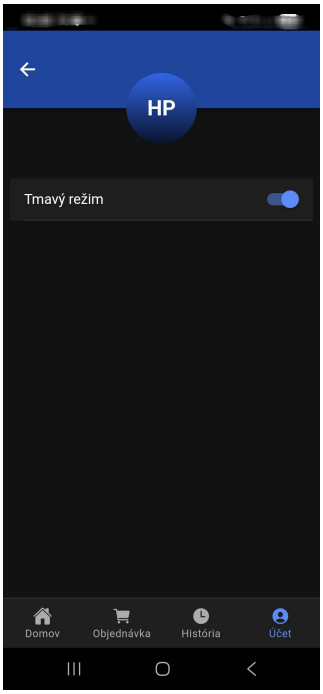
Prispôsobenie vzhľadu aplikácie

Používateľ si môže vzhľad aplikácie prispôsobiť tak, že vie zmeniť aplikáciu na tmavý či svetlý režim. Prvotné nastavenie je podľa systémového nastavenia. O nastavovanie režimu a jeho prepínanie sa stará Signal Store `CustomizationStore`. Preferencia je uložená v `Preferences`, takže sa nemôže stať, že používateľ bude mať po otvorení vždy systémové nastavenie, namiesto jeho voľby, ktorú zadal v aplikácii. Systémová preferencia sa určuje podľa konkrétneho štýlu `prefers-color-scheme`. Pokiaľ neexistuje uložená preferencia v `Preferences`, vezme sa systémové nastavenie. Ako bolo možné vidieť v 8.12, `CustomizationStore` sa používa na viacerých miestach na zistenie aktuálneho re-

žimu a prispôsobenie farieb napríklad v grafoch a legendách grafov. Kód `CustomizationStore` je možné vidieť v ukážke 8.23.

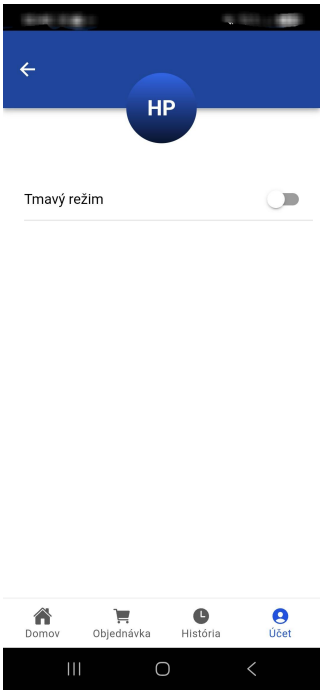
```
export const CustomizationStore = signalStore(
  { providedIn: 'root' },
  withState(initialState),
  withMethods((store) => ({
    async loadDarkModePreference() {
      const savedPreference = await Preferences.get({ key: 'darkMode' });
      const systemPreference = window.matchMedia(PREFER_COLOR_SCHEME_MEDIA_QUERY).matches;
      const initialDarkMode = savedPreference.value ? JSON.parse(savedPreference.value) : systemPreference;
      patchState(store, {darkMode: initialDarkMode});
      document.documentElement.classList.toggle('ion-palette-dark', initialDarkMode);
    },
    async toggleDarkMode(darkModeEnabled: boolean) {
      patchState(store, {darkMode: darkModeEnabled});
      await Preferences.set({ key: 'darkMode', value: JSON.stringify(darkModeEnabled) });
      document.documentElement.classList.toggle('ion-palette-dark', darkModeEnabled);
    },
  }))
);
```

Výpis 8.23 Signal Store pre farebný režim (zdroj: vlastný)



Obrázek 8.23 Zmena farebného režimu

(zdroj: vlastný)



Obrázek 8.24 Svetlý režim

(zdroj: vlastný)

8.4.4 Administrátorská aplikácia

Aplikácia pre administrátora má menej možností ako používateľská aplikácia. Rovnako ako používateľská aplikácia, má navigačný panel v spodnej časti obrazovky. Tento panel sa skladá z *Domov*, *Objednávky* a *Účet*. Navigácia je riešená opäť cez `Routes` a `TabsComponent` (8.24).

```
{
  path: 'admin',
  component: TabsComponent,
  children: [
    { path: 'admin-home', loadComponent: () => import('./admin/home/home.page').then(m => m.HomePage)},
    { path: 'orders', loadComponent: () => import('./admin/orders/orders.page').then( m => m.OrdersPage)},
    { path: 'admin-account', loadComponent: () => import('./admin/account/account.page').then( m => m.AccountPage)},
    { path: 'admin-account/customization', loadComponent: () => import('./admin/account/customization/customization.page').then( m => m.CustomizationPage)},
    { path: 'admin-account/profile', loadComponent: () => import('./admin/account/profile/profile.page').then( m => m.ProfilePage)},
    {
      path: '',
      redirectTo: '/admin/home',
      pathMatch: 'full',
    },
  ]
}
```

Výpis 8.24 Navigácia v administrátorskej časti (zdroj: vlastný)

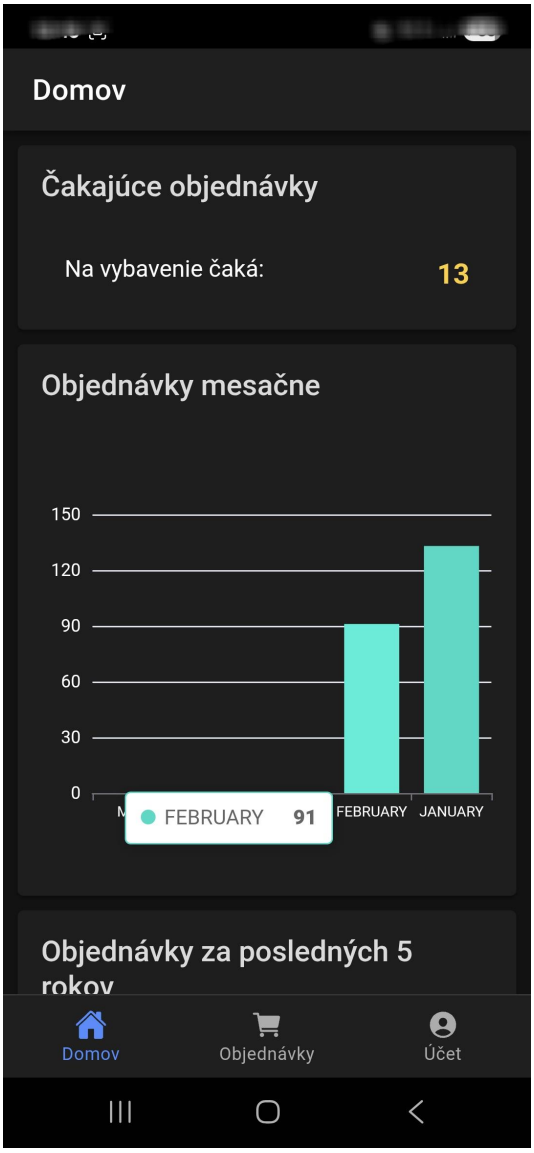
Na základe role sa rozhoduje, či je používateľ administrátorom alebo nie. Tento stav sa následne ukladá do `Preferences` a Signal Store `UserStore`, aby aj pri opätovnom načítaní tieto informácie zostali uložené (8.25).

```
export const UserStore = signalStore(
  { providedIn: 'root'},
  withState(initialState),
  withMethods((store) => ({
    async loadUser(user: User, isAdmin: boolean) {
      await Preferences.set({ key: 'isAdmin', value: JSON.stringify(isAdmin) });
      const initials = user.firstName.charAt(0) + user.lastName.charAt(0);
      patchState(store, {user: user, userInitials: initials, isAdmin: isAdmin});
    }
  }))
);
```

Výpis 8.25 Spracovanie role používateľa (zdroj: vlastný)

Domov

Domovská obrazovka administrátorskej časti sa skladá z niekoľkých častí. Hneď ako prvé je možné vidieť počet objednávok, ktoré čakajú na vybavenie. Potom nasledujú štatistiky počtu objednávok v danom roku rozdelené po mesiacoch, počtu objednávok rozdelených po rokoch (posledných 5 rokov) a počet používateľov aplikácie (webovej a mobilnej). Počet objednávok čakajúcich na vybavenie a prvé štatistiky je možné vidieť na obrázkoch 8.25 a 8.26.



Obrázek 8.25 Domovská obrazovka administrátorskej časti

(zdroj: vlastný)



Obrázek 8.26 Domovská obrazovka administrátorskej časti

(zdroj: vlastný)

Štatistiky objednávok sú vytvorené z dát, ktoré sa berú z tretej strany. Grafická reprezen-

tácia je pomocou komponenty `BarChartComponent`, ktorá používa `echarts` (8.26).

```
<ion-card>
  <ion-card-header>
    <ion-card-title>{{title()}}</ion-card-title>
  </ion-card-header>
  <ion-card-content class="chart-wrapper ion-no-padding">
    <div echarts class="chart" [options]="chartOptions"></div
        >
  </ion-card-content>
</ion-card>
```

Výpis 8.26 Stĺpcový graf (zdroj: vlastný)

Keďže `BarChartComponent` zobrazuje rôzne štatistiky, je to generická komponenta, ktorá zobrazuje a inicializuje dáta na základe vstupov `data` a `title`.

```
  data = input.required<PeriodCount[]>();
  title = input<string>('');
```

Výpis 8.27 BarChartComponent vstupy (zdroj: vlastný)

Samotné dáta, ktoré sa v grafe zobrazujú, sa vytvárajú pomocou štruktúry JSON. Tie sa následne predávajú ako vstup do komponenty `echarts`.

```
  private initChartOptions() {
    this.chartOptions = {
      grid: {
        left: '15%', right: '5%'
      },
      tooltip: { trigger: 'item' },
      xAxis: {
        type: 'category',
        data: this.data().map(orderYear => orderYear.period),
        axisLabel: {
          color: this.textColor,
          fontSize: '10px',
          interval: 0
        }
      },
      yAxis: {
        type: 'value',
        axisLabel: {
          color: this.textColor
        }
      },
      series: [
        {
          itemStyle: { color: '#03DAC6' },
          data: this.data().map(orderYear => orderYear.count)
          type: 'bar'
        }
      ]
    };
  }
```

Výpis 8.28 BarChartComponent tvorba dát (zdroj: vlastný)

Pre reprezentáciu množstva používateľov aplikácie sa používa `LineChartComponent` rovnako s pomocou `echarts`. Pre účely prezentácie sú dáta v grafe na obrázku 8.27 umelo vytvorené. Dané štatistiky boli implementované aj do webovej aplikácie.



Obrázek 8.27 Domovská obrazovka administrátorskej časti

(zdroj: vlastný)

Objednávky

Obrazovka *Objednávky* zobrazuje všetky objednávky, ktoré ešte neboli spracované - odoslané. Táto časť je zjednodušená oproti webovej aplikácii, pretože výdajňa nechcela vidieť všetky objednávky, ale jednoducho notifikovať používateľa o zmene stavu. Zmena stavu sa

deje pomocou potiahnutia karty s objednávkou doľava. Karta obsahuje informácie o objednávke a jej zadávateľovi, ktoré sú uložené v Signal Store. Tento Signal Store `AdminStore` drží informácie o používateľoch a ich objednávkach, spracúva tieto dáta do takej formy, aby boli jednoducho zobraziteľné v karte.

```
  withMethods((store) => {
    const restApiService = inject(RestApiService);
    return {
      async loadAllData() {
        restApiService.loadOrders().then(orders => {
          restApiService.loadUsers().then(users => {
            const usersOrders = this.mapUserOrders(users,
               orders);
            const usersStat = this.computeUserStats(users);
            patchState(store, {usersOrders: usersOrders,
               users: usersStat});
          })
        })
      },
      mapUserOrders(users: User[], orders: Order[]):
         UserOrders[] {
        return users.map(user => {
          const userOrders = new UserOrders();
          userOrders.user = user;
          userOrders.orders = orders.filter(order => order.
             userId === user.id);
          return userOrders;
        })
      }
    }
  })
```

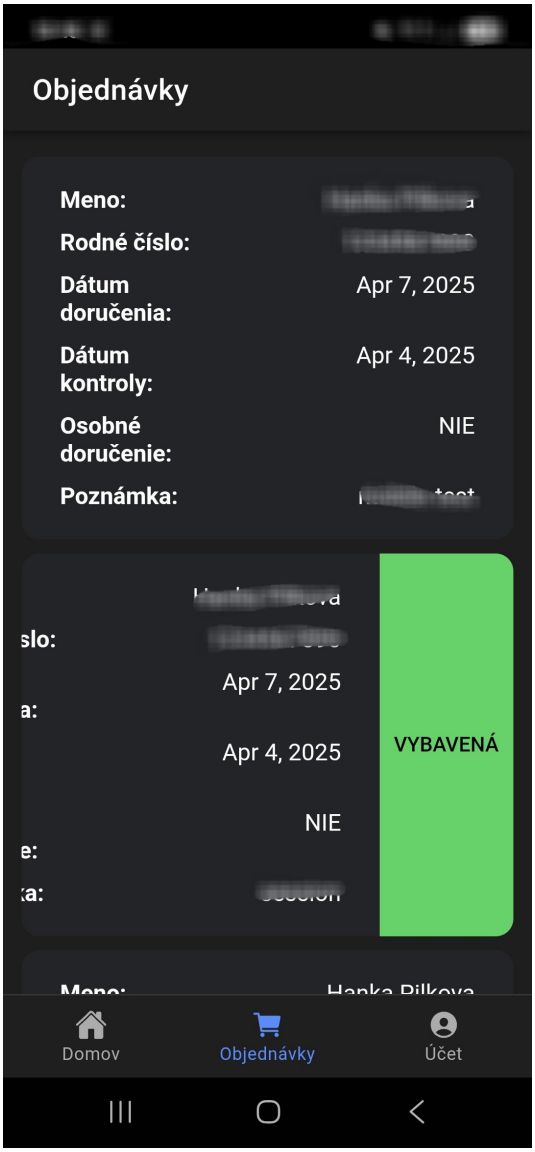
Výpis 8.29 AdminStore: načítanie dát (zdroj: vlastný)

Samotné odoslanie objednávky pomocou gesta doľava je implementované cez Ionic komponentu `ion-item-sliding`. Odoslanie je možné vidieť na obrázku 8.28.

```
<ion-list class="orders-list" lines="none">
  @for (userOrder of adminStore.pendingOrders(); track $index
     ) {
    <div class="order-wrapper">
      <ion-item-sliding #slidingItem>
          <app-order
            [userOrder]="userOrder">
          </app-order>
        <ion-item-options side="end">
          <ion-item-option color="success" expandable (click)
             ="onOrderDone(userOrder, slidingItem)">Vybavená<
             /ion-item-option>
        </ion-item-options>
      </ion-item-sliding>
    </div>
  }
</ion-list>
```

Výpis 8.30 Odoslanie objednávky (zdroj: vlastný)

Vo webovej aplikácii je táto funkcionalita implementovaná pomocou zaškrtávacieho okna, kde je možné zaškrtnúť a odoslať viacero objednávok naraz.

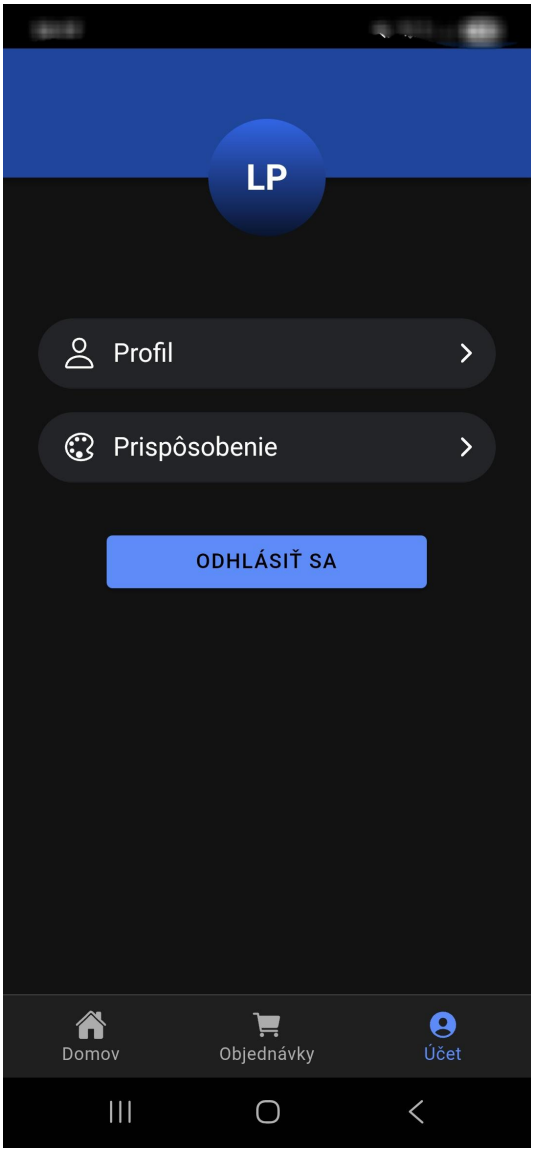


Obrázek 8.28 Odoslanie objednávky

(zdroj: vlastný)

Účet

Účet administrátora je rovnako koncipovaný ako používateľský účet s tým rozdielom, že má menej funkcionalít. Nachádza sa v ňom iba *Profil*, *Prispôsobenie* a možnosť odhlásiť sa. Je to preto, lebo administrátor má vždy dostávať upozornenia s novou objednávkou a ochrana osobných údajov sa ho netýka. Obrazovku účtu je možné vidieť na obrázku 8.29.



Obrázek 8.29 Účet administrátora

(zdroj: vlastný)

8.5 Push notifikácie

Spravovanie a implementácia notifikácií je v dvoch častiach. Backend je vo forme mikroservisy `push-notifications` a frontend je implementovaný v mobilnej aplikácii. Notifikácie sú posielané pomocou Firebase.

Nastavenie Firebase

Na to, aby bolo možné používať Firebase je nutné registrovať sa na ich platforme a vytvoriť si projekt. Projekt znamená registrácia aplikácie ako jej meno, balík či prezývka aplikácie.

Následne je nutné vygenerovať `google-services.json`, ktorý slúži ako konfiguračný súbor pre Firebase v Android aplikácii. Obsahuje:

- identifikátory projektu Firebase ako `project_id`, `mobilesdk_app_id`, `storage_bucket`, `api_key` a iné,
- konfiguráciu pre konkrétnu Android aplikáciu ako `package_name`, `app_id`, prípadne nastavenia analytiky, cloud messagingu a ďalších služieb,
- údaje potrebné na inicializáciu Firebase SDK pri spustení aplikácie.

Slúži na to, aby Firebase SDK zistil, ku ktorému projektu sa má pripojiť. Gradle plugin google-services tento súbor spracuje a vygeneruje z neho interné konfiguračné triedy. Slúži ako most medzi aplikáciou a Firebase backendom – bez neho sa appka nepripojí k správnemu Firebase projektu. Pre iOS platformu slúži ekvivalentný súbor `GoogleService-Info.plist`.

Mikroservisa push-notifications

Mikroservisa `push-notifications` je Maven modul, ktorej úlohou je posielanie push notifikácií. Ako jediná mikroservisa nemá svoj model v `Model`, ale riadi si aj túto časť samostatne. Obsahuje dve konfigurácie, `SecurityConfiguration` a `FirebaseConfiguration`, ktorá slúži na nastavenie a inicializáciu Firebase.

```
@Configuration
public class FirebaseConfiguration {
  @Bean
  public FirebaseMessaging firebaseMessaging() throws
     IOException {
    ClassPathResource serviceAccount = new ClassPathResource(
       "diatop-key.json");

    GoogleCredentials googleCredentials = GoogleCredentials
            .fromStream(serviceAccount.getInputStream())
            .createScoped("https://www.googleapis.com/auth/
               firebase.messaging");

    FirebaseOptions firebaseOptions = FirebaseOptions.builder
       ()
            .setCredentials(googleCredentials)
            .build();

    FirebaseApp.initializeApp(firebaseOptions);

    return FirebaseMessaging.getInstance();
  }
}
```

Výpis 8.31 Firebase konfigurácia (zdroj: vlastný)

Na komunikáciu používa `RestController`, ktorý prijíma požiadavky a ďalej ich spracováva. Na ďalšie spracovanie používa `NotificationService`. *Controller* obsahuje *endpoints* ako:

GET ping na overenie, že mikroservisa beží,

POST register na registrovanie zariadenia používateľa,

POST subscribe prihlasuje používateľa k odberu,

POST unsubscribe odhlasuje používateľa z odberu,

GET isSubscribed vráti, či je daný používateľ prihlásený k odberu konkrétnej témy.

Registrácia sa udeje vždy, keď používateľ vstúpi do aplikácie. Skontroluje sa, či je dané zariadenie a používateľ registrovaný a ak nie, zaregistruje ho.

```
1  public Mono<RegistrationStatus> registerDevice(
       NotificationDeviceDto notificationDevice) {
2    return notificationDeviceRepository.
       findNotificationDeviceByUserIdAndDeviceToken(
       notificationDevice.getUserId(),
3                  notificationDevice.getDeviceToken())
4            .flatMap(device -> {
5              logger.info("Device with token is already
                 registered {}", device.getDeviceToken());
6              return Mono.just(RegistrationStatus.REGISTERED)
                 ;
7            })
8            .switchIfEmpty(Mono.defer(() -> {
9              var newDevice = notificationDeviceMapper.
                 toEntity(notificationDevice);
10             logger.info("New device registered with token
                 {}", newDevice.getDeviceToken());
11             return notificationDeviceRepository.save(
                 newDevice)
12                   .thenReturn(RegistrationStatus.CREATED)
                       ;
13           }));
14 }
```

Výpis 8.32 Registrácia zariadenia a používateľa (zdroj: vlastný)

Posielanie notifikácií je cez `NotificationEventListener` narozdiel od ostatných žiadostí spojených s notifikáciami, ktoré idú cez `NotificationController`. Momentálne existuje jediný typ notifikácií - *Objednávka* (topic `Order`). Tie sa posielajú buď do administrátorskej časti - zariadení, kde je prihlásený administrátor po tom, čo používateľ vytvorí objednávku, alebo do používateľovho zariadenia, keď je objednávka odoslaná. Na základe typu udalosti `OrderEventType` sa rozhodne, ktorá notifikácia je poslaná.

```
@KafkaListener(topics = "order-events", groupId = "
    notification-service-group")
public void handleOrderEvent(OrderEvent event) {
  if (OrderEventType.CREATED.equals(event.getEventType())
      {
    notificationService.sendOrderNotificationToTopic(event)
        ;
  } else if (OrderEventType.SENT.equals(event.getEventType
      ())) {
    notificationService.sendOrderNotificationToUser(event);
  }
}
```

Výpis 8.33 Žiadosť o poslanie notifikácie (zdroj: vlastný)

Ak je táto udalosť `OrderEventType` typu `CREATED` - nová objednávka, notifikácia sa posiela všetkým administrátorom s témou `ORDER_CREATED`.

```
public void sendOrderNotificationToTopic(OrderEvent event)
    {
  Message message = Message.builder()
          .setTopic(Topic.ORDER_CREATED.toString().
              toLowerCase())
          .setNotification(Notification.builder()
                  .setTitle("Nová objednávka!")
                  .setBody("Klikni pre zobrazenie objedná
                      vky.")
                  .build())
          .putAllData(Map.of(
                  "orderId", event.getOrderId().toString(),
                  "topic", Topic.ORDER_CREATED.toString()))
          .build();
  sendNotification(message);
}
```

Výpis 8.34 Poslanie notifikácie administrátorovi o vytvorení novej objednávky (zdroj: vlastný)

Pokiaľ administrátor odošle objednávku a používateľ odoberá danú tému a jeho zariadenie je registrované, pošle sa na toto konkrétne zariadenie notifikácia o odoslaní objednávky.

```
public void sendOrderNotificationToUser(OrderEvent event) {
  notificationDeviceRepository.
      findNotificationDeviceByUserId(event.getUserId())
    .flatMap(notificationDevice ->
            isSubscribed(notificationDevice.
                getDeviceToken(), Topic.ORDER_SENT)
                    .filter(isSubscribed -> isSubscribed)
                    .doOnNext(__ ->
                        sendOrderSentNotification(event,
                        notificationDevice.getDeviceToken
                        ())))
    .subscribe();
}
```

Výpis 8.35 Poslanie notifikácie používateľovi o odoslaní objednávky (zdroj: vlastný)

Nastavenie push notifikácií v mobilnej aplikácii

Na používanie push notifikácií v mobilnej aplikácii sa používajú dva pluginy:

- `@capacitor/push-notNotificationServiceifications` - základný plugin na prácu s notifikáciami, funguje keď je aplikácia v popredí aj na pozadí
- `@capacitor-firebase/messaging` - špecializovaný Firebase plugin (zjednodušuje prácu s tokenmi, a iné).

Akonáhle je používateľ prihlásený, zavolá sa metóda `initializePushNotifications` v servise, ktorá sa stará o notifikácie - `NotificationService`.

```
async initializePushNotifications(userId: number) {
  const permission = await PushNotifications.
     requestPermissions();
  if (permission.receive !== 'granted') {
    console.warn('Push notifications not granted.');
    return;
  }

  await PushNotifications.register();

  await PushNotifications.addListener('
     pushNotificationReceived', (notification:
     PushNotificationSchema) => {
    this.handlePush(notification);
  });

  await PushNotifications.addListener('
     pushNotificationActionPerformed', (notification:
     ActionPerformed) => {
    this.handlePush(notification.notification);
  });

  let existingToken;
  await PushNotifications.addListener('registration', async
      (token: Token) => {
    existingToken = token.value;
    await this.registerDeviceToken(userId, token.value);
  });

  await FirebaseMessaging.addListener('notificationReceived
     ', (message) => {
    this.handlePush(message);
  });

  await FirebaseMessaging.addListener('
     notificationActionPerformed', (action) => {
    this.handlePush(action.notification);
  });

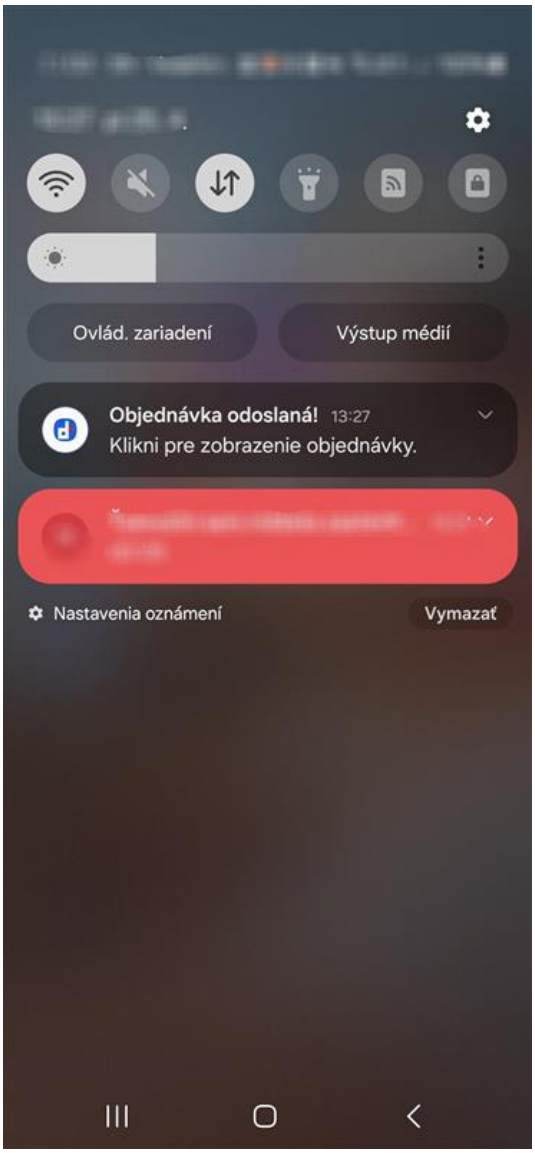
  const firebaseToken = await FirebaseMessaging.getToken();
  if (firebaseToken?.token && firebaseToken?.token !==
     existingToken) {
    await this.registerDeviceToken(userId, firebaseToken.
       token);
  }
}
```

Výpis 8.36 Inicializácia notifikácií (zdroj: vlastný)

Táto servisa sa stará nie len o registráciu zariadenia, ale aj o prijímanie notifikácií:

- `PushNotifications.addListener`,
- `FirebaseMessaging.addListener`,

a následných akcií s tým spojených - `handlePush`. Na obrázku 8.30 je možné vidieť push notifikáciu - odoslaná objednávka, ktorá dorazí používateľovi.



Obrázek 8.30 Push notifikácia odoslaná používateľovi

(zdroj: vlastný)

9 POPIS NASADENIA

Nasadenie aplikácie prebiehalo v niekoľkých krokoch. Keďže sa nasádza webová aj mobilná aplikácia.

9.1 Nasadenie webovej aplikácie

V prípade nasadenia webovej aplikácie bolo nutné vykonať niekoľko krokov. V prvom rade bolo nutné upraviť kód a *properties* tak, aby neobsahovali rôzne citlivé údaje ako heslá a podobne.

9.1.1 Príprava Gateway mikroservisy

Mikroservisa `Gateway` obsahuje smerovanie (route) jednotlivých REST volaní. V priebehu vývoja sa používali url ako `http://localhost:` a číslo portu danej servisy. To v prípade bežiacej aplikácie v produkcii už neplatí. Boli vytvorené špeciálne `RoutesProperties`.

```
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "routes")
public class RoutesProperties {

  private String resource;
  private String ui;
  private String admin;
  private String notification;
}
```

Výpis 9.1 Adresy pre smerovanie v Gateway (zdroj: vlastný)

Rovnako bol vytvorený špeciálny súbor `application-prod.properties` predstavujúci nastavenia pre produkciu. Smerovanie pre produkciu v súbore `application-prod.yaml` je možné vidieť v ukážke 9.2.

```
routes.resource=http://resource:9000
routes.ui=http://ui:8081
routes.admin=http://admin:8082
routes.notification=http://notification:9001
}
```

Výpis 9.2 Nastavenie adries smerovania v Gateway pre produkciu (zdroj: vlastný)

Následne sú tieto nastavenia používané v smerovaní. Príklad smerovania je možné vidieť v ukáže 9.3.

```
    .route("resource", p -> p.path("/resource/**")
            .filters(f -> f.rewritePath("/resource/(?<segment
               >.*)", "/${segment}"))
            .uri(routes.getResource()))
}
```

Výpis 9.3 Gateway smerovanie (zdroj: vlastný)

Pre mikroservisu `Gateway` bol vytvorený súbor `.env` na všetky citlivé údaje ako heslá, používateľské mená a podobne. Tento súbor nie je súčasťou verzovacieho systému ani *Docker image*. Príklad použitia premennej zo súboru `.env` je v ukážke

```
spring.r2dbc.username=${DB_USER}
spring.r2dbc.password=${DB_PASSWORD}

spring.mail.username=${MAIL_USERNAME}
spring.mail.password=${MAIL_PASSWORD}
```

Výpis 9.4 Príklad použitia premennej zo súboru .env (zdroj: vlastný)

9.1.2 Príprava Diatop mikroservisy

Rovnako ako v `Gateway` bolo nutné upraviť nastavenia v mikroservise `Diatop` (UI). Bol vytvorený `application-prod.properties`. V tomto súbore sa nastavil *host* pre Redis.

```
spring.redis.host=redis
```

Výpis 9.5 Nastavenie Redis v mikroservise Diatop (zdroj: vlastný)

9.1.3 Príprava Admin mikroservisy

Taktiež v mikroservise `Admin` boli upravené *properties*. V súbore `application-prod.properties` sa nastavil *host* pre Redis.

```
spring.redis.host=redis
```

Výpis 9.6 Nastavenie Redis v mikroservise Admin (zdroj: vlastný)

9.1.4 Príprava Model mikroservisy

Mikroservisa `Model` reprezentuje model celej aplikácie, takže pracuje s databázou. V tomto prípade je nutné vytvoriť `.env` súbor, kde sú uložené prihlasovacie údaje k databáze ako v prípade mikroservisy `Gateway`.

```
spring.r2dbc.username=${DB_USER}
spring.r2dbc.password=${DB_PASSWORD}
```

Výpis 9.7 Použitia premennej zo súboru .env v Model (zdroj: vlastný)

Rovnako bol vytvorený súbor `application-prod.properties`, kvôli r2dbc url.

```
spring.r2dbc.url=r2dbc:pool:mysql://mysql:3306/diatopdb
spring.r2dbc.username=${DB_USER}
spring.r2dbc.password=${DB_PASSWORD}
```

Výpis 9.8 Použitia premennej zo súboru .env v Model (zdroj: vlastný)

9.1.5 Príprava Resource mikroservisy

Mikroservisa `Resource` má okrem databázy a emailu citlivé údaje na prihlasovanie do API tretej strany. Tieto údaje boli pridané do novovytvoreného súboru `.env` pre túto mikroservisu. Ich použitie je vidno v ukážkach 9.9 pre `application.properties` a novovytvorených `application-prod.properties`.

```
third-party-api.base-url=${THIRD_PARTY_API_BASE_URL}
third-party-api.username=${THIRD_PARTY_API_USERNAME}
third-party-api.password=${THIRD_PARTY_API_PASSWORD}
third-party-api.recaptcha=${THIRD_PARTY_API_RECAPTCHA}
```

Výpis 9.9 Použitie premenných pre prihlásenie do tretej strany zo súboru .env v application.properties (zdroj: vlastný)

9.1.6 Príprava PushNotifications mikroservisy

Poslednou mikroservisou na nastavenie je `PushNotifications`. Tá obsahuje rovnaké nastavenia, ktoré boli spomenuté vyššie. Týmito nastaveniami sú Redis, databáza a Kafka. Rovnako bol vytvorený súbor `.env` a `application-prod.properties`.

9.1.7 Kúpa domény

Pre doménu bol vybraný názov *diatop-vydaj.sk*, pretože zahŕňa názov výdajne spolu s jej účelom vytvárať objednávky a teda zabezpečenie výdaja materiálu. Doména *diatop-vydaj.sk* bola kúpená z *https://www.websupport.sk/*. Táto stránka je oficiálnym poskytovateľom domén a má výborné hodnotenia.

9.1.8 Kúpa a nastavenie VPS

Server bol kupovaný od Hetzner (*https://www.hetzner.com/cloud*). Nastavenie servera:

Lokácia: Nuremberg Nemecko,

Operačný systém: Ubuntu 22.4,

Typ: zdieľaný vCPU s:

- 2 Intel CPU,
- 4GB RAM,
- 40GB SSD

Sieť: Public IPv4, Public IPv6.

Po kúpe servera bol server nastavený. Nastavenie zahŕňalo firewall (povolenie portov), vytvorenie ssh, aktualizácie systému, nainštalovanie Java, Maven, Docker, git. Následne bol z gitu stiahnutý repozitár. Po úspešnom `maven clean install` bol spustený `docker build -t diatop-gateway .` kde:

- `-t diatop-gateway` je názov výsledného image
- bodka (".") znamená, že Dockerfile je v aktuálnom priečinku.

Dockerfile pre gateway je možné vidieť v 9.10.

```
1 FROM openjdk:19-jdk-slim
2 WORKDIR /app
3 COPY target/*.jar app.jar
4 ENV SPRING_PROFILES_ACTIVE=prod
5 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Výpis 9.10 Dockerfile pre gateway (zdroj: vlastný)

Následne bol vytvorený `docker-compose.yaml`. Súbor `.env` v koreňovom adresári projektu bol skopírovaný na server na rovnaké miesto. Potom bol za účelom otestovania, či gateway, databáza, redis a kafka bežia ako majú, spustený `docker compose up -d`. Ostatné mikroservisy majú rovnaký `Dockerfile` ako gateway.

IP adresa VPS servera (verejná) bola nastavená k doméne ako cieľová IP adresa.

9.1.9 Reverzný proxy server NGINX

Najprv bol vytvorený testovací `nginx.conf` pre generovanie certifikátu (9.11).

```
events {}

http {
    server {
        listen 80;
        server_name diatop-vydaj.sk www.diatop-vydaj.sk;

        location ^~ /.well-known/acme-challenge/ {
            root /usr/share/nginx/html;
            default_type "text/plain";
            try_files $uri =404;
        }

        location / {
            return 301 https://$host$request_uri;
        }
    }
}
```

Výpis 9.11 Súbor nginx.conf (zdroj: vlastný)

Na serveri bol vytvorený certifikát pomocou *Let's Encrypt*. Súbor `docker-compose` bol upravený na používanie nginx (9.12).

```
events {}

http {
    server {
        listen 80;
        server_name diatop-vydaj.sk www.diatop-vydaj.sk;

        location ^~ /.well-known/acme-challenge/ {
            root /usr/share/nginx/html;
            default_type "text/plain";
            try_files $uri =404;
        }

        location / {
            return 301 https://$host$request_uri;
        }
    }
}
```

Výpis 9.12 Docker pre nginx (zdroj: vlastný)

Po vytvorení certifikátu cez *Let's encrypt* bol `nginx.conf` upravený, aby fungoval s certifikátom (9.13).

```
events {}
http {
    server {
        listen 80;
        server_name diatop-vydaj.sk www.diatop-vydaj.sk;

        location ^~ /.well-known/acme-challenge/ {
            root /usr/share/nginx/html;
            default_type "text/plain";
            try_files $uri =404;
        }

        location / {
            return 301 https://$host$request_uri;
        }
    }
    server {
        listen 443 ssl;
        server_name diatop-vydaj.sk www.diatop-vydaj.sk;

        ssl_certificate /etc/letsencrypt/live/diatop-vydaj.sk
           /fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/diatop-
           vydaj.sk/privkey.pem;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;

        location / {
            proxy_pass http://gateway:8080;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_
               forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

Výpis 9.13 Súbor nginx.conf s certifikátom (zdroj: vlastný)

Posledným krokom bola úprava `docker-compose` do finálnej podoby (9.14).

```
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./nginx/html:/usr/share/nginx/html:ro
      - /etc/letsencrypt:/etc/letsencrypt:ro
      - /var/lib/letsencrypt:/var/lib/letsencrypt
      - /usr/share/nginx/html:/usr/share/nginx/html
    restart: always
```

Výpis 9.14 Súbor docker-compose nginx nastavenie (zdroj: vlastný)

Aplikácia beží na adrese *www.diatop-vydaj.sk*. Pomocou `CommandLineRunner` boli pridané tri administrátorské účty a jeden demo účet. Migrácie v databáze a prvotné vytvorenie tabuliek sú pomocou *flyway*.

9.2 Nasadenie mobilnej aplikácie

Nasadenie mobilnej aplikácie bolo rozdelené do dvoch častí:

1. Nasadenie aplikácie na Android platformu
2. Nasadenie aplikácie na iOS platformu

9.2.1 Nasadenie aplikácie na Android

Pre nasadenie aplikácie na Android sa používa *Google Play Console*. Predtým, ako je možné aplikáciu nahrať, je nutné ju podpísať. Podpísanie sa deje pomocou *release-key.jks*. Ten je nutné vygenerovať a nastaviť heslá. Následne je možné tieto závislosti pridať do Android Studio `build.gradle (:app)`. Potom je možné vygenerovať výsledný `.app`, ktorý sa nahráva do Google Play Console.

Na zverejnenie aplikácie je nutné vyplniť údaje o aplikácii ako predvolený záznam v obchode, pravidlá ochrany súkromia, prístup k aplikáciám, reklamy a iné. Následne je nutné zverejniť aplikáciu 14 testerom. Až po ukončení tohto procesu je možné ju nasadiť cez produkčný kanál.

9.2.2 Nasadenie aplikácie na iOS

Na nasadenie aplikácie do Apple store je potrebný Mac a Xcode. Následne je nutné vytvoriť zložku s `ios` pomocou príkazov v (9.15).

```
ionic build
npx cap sync ios
npx cap add ios
npx cap open ios
```

Výpis 9.15 Príkazy pre vystavanie aplikácie a synchronizáciu projektu pre Apple (zdroj: vlastný)

V Xcode bol z Firebase stiahnutý `GoogleService-Info.plist` a umiestnený do priečinku ios/App/App/. V Xcode *Signing & Capabilities* boli pridané dve *Capabilities* pre push noti-

fikácie:

- Push Notifications,
- Background Modes, kde bola zaškrtnutá možnosť *Remote notifications*.

Týmito nastaveniami sú funkčné push notifikácie pomocou Firebase na iOS.

Na zachovanie vzhľadu aplikácie bola vygenerovaná ikona a *splash screen* pomocou `@capacitor/assets` pluginu.

Posledným krokom bolo vytvorenie archívu, ktorý sa nahráva do Apple App Store s použitím *Apple Distribution* certifikátu pod menom vývojára a jeho účtu. Tento archív bol distribuovaný do Apple App Store.

V Apple App Store po dohode s výdajňou Diatop najprv prebehne testovanie (rovnako ako v prípade Android) pomocou *TestFlight*. Pre tieto účely bola vytvorená skupina, kde boli pridaní testeri, ktorí boli vybraní výdajňou.

ZÁVĚR

Výsledkom diplomovej práce je komplexný systém pre výdajňu zdravotníckych pomôcok pre diabetikov. Tento systém sa skladá z webovej a mobilnej aplikácie. Mobilná aplikácia ponúka registráciu, prihlásenie, obnovu hesla, zadávanie objednávok, ktoré sú formou emailu zaslané výdajni a notifikačným systémom je výdajňa upozornená o zadaní novej objednávky. Obsahuje informácie o čerpaní finančných prostriedkov, ktoré boli používateľovi pridelené na zdravotné pomôcky. Rovnako bola vytvorená mobilná aplikácia určená pre administrátorov, ktorí sú pracovníkmi výdajne. Taktiež bola rozšírená existujúca webová aplikácia o nové funkcie pre klientov výdajne a jej zamestnancov.

Systém bude naďalej rozširovaný o nové možnosti, ktoré pomôžu výdajni a jej klientom. Sú plánované rozšírenia pre zobrazovanie čerpania rôznych typov rozpočtu, podľa aktuálnej situácie používateľa. Rovnako výdajňa zamýšľa rozšíriť štatistiky o presnejšie grafy, s možnosťami filtrovania a väčšej interakcie.

Tvorba diplomovej práce ma obohatila o poznatky a skúsenosti v tvorbe mobilných aplikácií a nasadenia webovej a mobilnej aplikácie. Tieto vedomosti budem ďalej využívať v pracovnom prostredí.

Ďalším krokom je pilotné testovanie mobilnej aplikácie na platformách Android a iOS a jej finálne nasadenie. Rovnako sa naďalej plánuje pracovať na jej vylepšovaní a doplňovaní nových funkcií.

SEZNAM POUŽITÉ LITERATURY

[1] RAVULAVARU, Arvind, 2017. *Learning Ionic*. Packt Publishing. 2. vyd. ISBN 978-1-78646-605-1.

[2] GRIFFITH, Chris, 2017. *Mobile App Development with Ionic, Revised Edition: Cross-Platform Apps with Ionic, Angular, and Cordova*. O'Reilly Media, Inc. 1. vyd. ISBN 978-1-49199-809-0.

[3] CHRIS GRIFFITH, [b.r.]. *What is Hybrid Mobile App Development?* Online. Dostupné z: `https://ionic.io/resources/articles/what-is-hybrid-app-development`. [cit. 2025-04-14].

[4] ADITYA JANI, 2023. *Native vs Hybrid App: Best Choice for Mobile App Development*. Online. Dostupné z: `https://medium.com/@adityajani1/native-vs-hybrid-app-best-choice-for-mobile-app-development-4163b497d273`. [cit. 2025-04-14].

[5] CYBELLIUM LTD, Kris Hermans, 2023. *Mastering Ionic: A Comprehensive Guide to Learn Ionic*. Cybellium Ltd. 1. vyd. ISBN 979-8-87015-193-9.

[6] IONIC, 2025. *Capacitor: Cross-platform Native Runtime for Web Apps*. Online. Dostupné z: `https://capacitorjs.com/docs`. [cit. 2025-04-21].

[7] IONIC, 2025. *Web View*. Online. Dostupné z: `https://ionicframework.com/docs/core-concepts/webview`. [cit. 2025-04-21].

[8] IONIC TEAM, 2025. *How Capacitor Works*. Online. Dostupné z: `https://ionic.io/blog/how-capacitor-works-2`. [cit. 2025-04-21].

[9] MARTIN DONADIEU, 2025. *How Capacitor Bridges Web and Native Code*. Online. Dostupné z: `https://capgo.app/blog/how-capacitor-bridges-web-and-native-code/#quick-overview`. [cit. 2025-04-21].

[10] IONIC, 2025. *Capacitor Plugins*. Online. Dostupné z: `https://capacitorjs.com/docs/plugins`. [cit. 2025-04-21].

[11] IONIC, 2025. *CapacitorHttp*. Online. Dostupné z: `https://capacitorjs.com/docs/apis/http`. [cit. 2025-04-21].

[12] IONIC, 2025. *CORS Errors*. Online. Dostupné z: https://ionicframework.com/docs/troubleshooting/cors. [cit. 2025-04-21].

[13] IONIC, 2025. *@capacitor/push-notifications*. Online. Dostupné z: https://capacitorjs.com/docs/apis/push-notifications. [cit. 2025-04-21].

[14] IONIC, 2025. *@capacitor/preferences*. Online. Dostupné z: https://capacitorjs.com/docs/apis/preferences. [cit. 2025-04-21].

[15] APARAJITA, 2024. *@aparajita/capacitor-biometric-auth*. Online. NPM. Dostupné z: https://www.npmjs.com/package/@aparajita/capacitor-biometric-auth. [cit. 2025-04-22].

[16] APARAJITA, 2024. *@aparajita/capacitor-secure-storage*. Online. NPM. Dostupné z: https://www.npmjs.com/package/@aparajita/capacitor-secure-storage. [cit. 2025-04-22].

[17] CAPGO, [b.r.]. *capacitor-secure-storage*. Online. Capgo. Dostupné z: https://capacitor-tutorial.com/plugins/capacitor-secure-storage/. [cit. 2025-04-22].

[18] APARAJITA, [b.r.]. *capacitor-secure-storage*. Online. Github. Dostupné z: https://github.com/aparajita/capacitor-secure-storage-demo. [cit. 2025-04-22].

[19] MARVIN, 2024. *Cordova vs Capacitor: A Comparison in Building Ionic Framework Apps*. Online. INFOSOFT. Dostupné z: https://sdk.docutain.com/blogartikel/cordova-versus-capacitor. [cit. 2025-04-22].

[20] MILE ZIVKOVIC, 2024. *Capacitor vs Cordova - Which Framework is Better?* Online. MobiLoud. Dostupné z: https://www.mobiloud.com/blog/capacitor-vs-cordova. [cit. 2025-04-22].

[21] MATT NETKOW, [b.r.]. *Ionic vs Flutter: Best Platform for Hybrid App Development*. Online. Ionic. Dostupné z: https://ionic.io/resources/articles/ionic-vs-flutter-comparison-guide#h-how-flutter-works. [cit. 2025-04-23].

[22] FERUZA SAIDKOMOLOVA, 2025. *Flutter vs Ionic in 2023: Which is Best for Your App?* Online. Ptolemay. Dostupné z: https://www.ptolemay.com/post/flutter-vs-ionic-in-2023-which-is-best-for-your-app. [cit. 2025-04-23].

[23] PETER MAGNUSSON, Emilie Barse, 2024. *2023 Retrospective: Web and Mobile Application Security*. Online. Assured AB. Dostupné z: https://www.assured.se/posts/retrospective-2023-web-mobile-api-application-security. [cit. 2025-04-27].

[24] AWAN, Haseeb, [b.r.]. *Mobile Web App Security Compared | NowSecure*. Online. NowSecure. Dostupné z: https://www.nowsecure.com/resource/mobile-app-security-vs-web-app-security/?utm_source=chatgpt.com. [cit. 2025-04-27].

[25] AWAN, Haseeb, 2024. *Mobile App Security vs Mobile Website Security*. Online. Dostupné z: https://www.efani.com/blog/mobile-app-security-vs-mobile-website-security?utm_source=chatgpt.com. [cit. 2025-04-27].

[26] ANDROID, 2025. *Authentication*. Online. Dostupné z: https://source.android.com/docs/security/features/authentication. [cit. 2025-04-27].

[27] SHUEH, Pat, 2020. *Mobile App Security vs. Web App Security | How They Differ*. Online. Zimperium. Dostupné z: https://zimperium.com/blog/mobile-app-security-vs-web-app-security-how-they-differ. [cit. 2025-04-27].

[28] SHUEH, Pat, [b.r.]. *Application attacks*. Online. Contrast Security. Dostupné z: https://www.contrastsecurity.com/glossary/application-attacks. [cit. 2025-04-27].

[29] OWASP, [b.r.]. *Session hijacking attack*. Online. OWASP. Dostupné z: https://owasp.org/www-community/attacks/Session_hijacking_attack. [cit. 2025-04-30].

[30] HRANICKÝ, Jan, [b.r.]. *Lekce 9 - Session hijacking a jak se proti němu bránit*. Online. itnetwork.cz. Dostupné z: https://www.itnetwork.cz/php/bezpecnost/session-hijacking-a-jak-se-proti-nemu-branit. [cit. 2025-04-30].

[31] OWASP, [b.r.]. *A01:2021 – Broken Access Control*. Online. OWASP. Dostupné z: https://owasp.org/Top10/A01_2021-Broken_Access_Control/. [cit. 2025-04-30].

[32] OWASP, [b.r.]. *A07:2021 – Identification and Authentication Failures*. Online. OWASP. Dostupné z: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/. [cit. 2025-04-30].

[33] OWASP, [b.r.]. *API1:2023 Broken Object Level Authorization*. Online. OWASP. Dostupné z: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/. [cit. 2025-05-02].

[34] OWASP, [b.r.]. *API3:2023 Broken Object Property Level Authorization*. Online. OWASP. Dostupné z: https://owasp.org/API-Security/editions/2023/en/0xa3-broken-object-property-level-authorization/. [cit. 2025-05-02].

[35] SHARMA, Atharv, 2024. *OWASP Top Ten — Injection*. Online. Medium. Dostupné z: https://atharvvvsharma.medium.com/owasp-top-ten-injection-3321452baebc. [cit. 2025-05-02].

[36] OWASP, [b.r.]. *M1: Improper Credential Usage*. Online. OWASP. Dostupné z: https://owasp.org/www-project-mobile-top-10/2023-risks/m1-improper-credential-usage.html. [cit. 2025-05-02].

[37] OWASP, [b.r.]. *M9: Insecure Data Storage*. Online. OWASP. Dostupné z: https://owasp.org/www-project-mobile-top-10/2023-risks/m9-insecure-data-storage.html. [cit. 2025-05-02].

[38] OWASP, [b.r.]. *M9: Reverse Engineering*. Online. OWASP. Dostupné z: https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering. [cit. 2025-05-02].

[39] MOZILLA, [b.r.]. *Types of attacks*. Online. Mozilla. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Types_of_attacks#stored_xss_attacks. [cit. 2025-05-02].

[40] STANO, 2022. *M9: Reverse Engineering*. Online. bart.sk. Dostupné z: https://blog.bart.sk/bezpecnost-na-internete-5-xss-utoky-skryte-za-pluginy-appky/?utm_source=chatgpt.com. [cit. 2025-05-02].

[41] ESET, [b.r.]. *Understanding Mobile Security: The Basics of Biometric Authentication*. Online. ESET. Dostupné z: https://www.eset.com/za/about/newsroom/press-releases-za/blog/understanding-mobile-security-the-basics-of-biometric-authentication/?srsltid=AfmBOopOMequii_u9z27MIcG__fRJcBqVNaCdnGZHo0qva4YsUtgYDDl. [cit. 2025-04-25].

[42] INC, Cyphercor, [b.r.]. *Biometric Authentication*. Online. Cyphercor Inc. Dostupné z: https://www.logintc.com/types-of-authentication/biometric-authentication/. [cit. 2025-04-26].

[43] NG, Vishwas, 2018. *Biometric Authentication in iOS*. Online. Medium. Dostupné z: https://medium.com/@vishwasng/bibiometric-authentication-in-ios-36f51a65b39b. [cit. 2025-04-27].

[44] APPLE, [b.r.]. *Local Authentication*. Online. Apple. Dostupné z: https://developer.apple.com/documentation/localauthentication. [cit. 2025-04-27].

[45] APPLE, 2024. *Biometric security*. Online. Apple. Dostupné z: https://support.apple.com/guide/security/biometric-security-sec067eb0c9e/web. [cit. 2025-04-27].

[46] SAIBAA, Omar, 2023. *Local Storage in iOS: Keychain*. Online. Medium. Dostupné z: https://medium.com/@omar.saibaa/local-storage-in-ios-keychain-668240e2670d. [cit. 2025-04-27].

[47] APPLE, 2024. *Keychain data protection*. Online. Apple. Dostupné z: https://support.apple.com/en-ke/guide/security/secb0694df1a/web. [cit. 2025-04-27].

[48] APPLE, [b.r.]. *Accessing Keychain Items with Face ID or Touch ID*. Online. Apple. Dostupné z: https://developer.apple.com/documentation/localauthentication/accessing-keychain-items-with-face-id-or-touch-id. [cit. 2025-04-27].

[49] ANDROID, 2025. *Show a biometric authentication dialog*. Online. Android. Dostupné z: https://developer.android.com/identity/sign-in/biometric-auth. [cit. 2025-04-27].

[50] ANDROID, 2025. *Hardware-backed Keystore*. Online. Android. Dostupné z: https://source.android.com/docs/security/features/authentication/fingerprint-hal. [cit. 2025-04-27].

[51] ANDROID, 2025. *Fingerprint HIDL*. Online. Android. Dostupné z: https://source.android.com/docs/security/features/keystore. [cit. 2025-04-27].

[52] MOZILLA, [b.r.]. *HTTPS*. Online. Mozilla. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/HTTPS. [cit. 2025-05-03].

[53] CLOUDFLARE, [b.r.]. *What is HTTPS?* Online. Cloudflare. Dostupné z: https://www.cloudflare.com/learning/ssl/what-is-https/. [cit. 2025-05-03].

[54] SECURITY, Savvy, 2021. *HTTP vs HTTPS Security: The Differences Between These Protocols*. Online. Savvy Security. Dostupné z: https://cheapsslsecurity.com/blog/http-vs-https-security-the-differences-between-these-protocols/. [cit. 2025-05-03].

[55] CLOUDFLARE, [b.r.]. *How does SSL work? | SSL certificates and TLS*. Online. Cloudflare. Dostupné z: https://www.cloudflare.com/learning/ssl/how-does-ssl-work/. [cit. 2025-05-03].

[56] CHAN, Alysa, 2022. *TCP and TLS handshake: What happens from typing in a URL to displaying a website? (Part 2)*. Online. Medium. Dostupné z: https://medium.com/@alysachan830/tcp-and-tls-handshake-what-happens-from-typing-in-a-url-to-displaying-a-website-part-2-243862438cd9. [cit. 2025-05-03].

[57] TEAM, SSL Support, 2023. *SSL/TLS Handshake: Ensuring Secure Online Interactions*. Online. SSL.com. Dostupné z: https://www.ssl.com/article/ssl-tls-handshake-ensuring-secure-online-interactions/. [cit. 2025-05-03].

[58] BHATT, Hemant, 2025. *Insight Into TLS Handshake For Building Secure Communications Over The Internet*. Online. Encryption Consulting LLC. Dostupné z: https://www.encryptionconsulting.com/what-is-a-tls-handshake-and-how-does-it-work/. [cit. 2025-05-03].

[59] MINISTERSTVO VNITRA ČESKÉ REPUBLIKY, [b.r.]. *Co je GDPR*. Online. Ministerstvo vnitra České republiky. Dostupné z: https://www.mvcr.cz/gdpr/clanek/co-je-gdpr.aspx. [cit. 2025-04-24].

[60] SELLEO, 2023. *How To Meet GDPR Requirements In Mobile And Web Applications*. Online. Selleo. Dostupné z: https://selleo.com/blog/how-to-meet-gdpr-requirements-in-mobile-and-web-applications#what-is-gdpr. [cit. 2025-04-24].

[61] LOUREIRO, Nuno, 2023. *What is GDPR Compliance in Web Application and API Security?* Online. Probely. Dostupné z: https://probely.com/blog/what-is-gdpr-compliance-in-web-application-and-api-security/. [cit. 2024-04-24].

[62] NESTEROV, Oleg, 2024. *15 key steps to make your software GDPR compliant in 2024*. Online. MindK Inc. Dostupné z: https://www.mindk.com/blog/how-to-make-your-software-gdpr-compliant/. [cit. 2025-04-24].

[63] SERVICES, Amazon Web. *What is an SSL/TLS Certificate?* Online. Amazon Web Services. Dostupné z: https://aws.amazon.com/what-is/ssl-certificate/. [cit. 2025-04-24].

[64] FOUNDATION, Mozilla, 2024. *Strict-Transport-Security*. Online. Mozilla Foundation. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security. [cit. 2025-04-25].

[65] LEWIS, Sarah, 2018. *perfect forward secrecy (PFS)*. Online. TechTarget. Dostupné z: https://www.techtarget.com/whatis/definition/perfect-forward-secrecy. [cit. 2025-04-24].

[66] SHARMA, Rajnish Kumar, 2024. *How To Protect Data In Mobile Web Apps Using Encryption online*. Online. Net Solutions. Dostupné z: https://www.netsolutions.com/hub/mobile-app-development/data-encryption. [cit. 2025-04-24].

[67] COOS, Andrada, 2021. *GDPR Data Encryption Requirements*. Online. Endpoint Protector by CoSoSys Ltd. Dostupné z: https://www.endpointprotector.com/blog/gdpr-data-encryption-requirements/. [cit. 2025-04-24].

[68] MGR. TOMÁŠ MAUX, Tereza Benešová M. J. M., 2023. *Cookies and GDPR: what do you need to know as a website provider?* Online. PEYTON legal. Dostupné z: https://www.peytonlegal.cz/en/cookies-gdpr-website/. [cit. 2025-04-24].

[69] SHREYA, 2024. *GDPR Compliance Checklist: 10 Key Steps (With Infographic)*. Online. CookieYes. Dostupné z: https://www.cookieyes.com/blog/gdpr-checklist-for-websites/. [cit. 2025-04-24].

[70] ENGINEERS, Platform, 2025. *A Deep Dive into Communication Styles for Microservices: REST vs. gRPC vs. Message Queues*. Online. Medium. Dostupné z: https://medium.com/@platform.engineers/a-deep-dive-into-communication-styles-for-microservices-rest-vs-grpc-vs-message-queues-ea72011173b3. [cit. 2025-05-03].

[71] OZKAYA, Mehmet, 2025. *Microservices Communications*. Online. Medium. Dostupné z: https://medium.com/design-microservices-architecture-with-patterns/microservices-communications-f319f8d76b71. [cit. 2025-05-03].

[72] CODEMIA, [b.r.]. *Kafka instead of Rest for communication between microservices*. Online. Codemia. Dostupné z: https://codemia.io/knowledge-hub/path/kafka_instead_of_rest_for_communication_between_microservices. [cit. 2025-05-04].

[73] ONESIGNAL, [b.r.]. *Push Notifications: The Ultimate Guide*. Online. OneSignal. Dostupné z: https://onesignal.com/what-are-push-notifications. [cit. 2025-05-04].

[74] IBM, [b.r.]. *What is a push notification?* Online. IBM. Dostupné z: https://www.ibm.com/think/topics/push-notifications. [cit. 2025-05-04].

[75] EVERHART, Jeff, 2024. *How push notifications work on Apple and Android*. Online. knock. Dostupné z: https://knock.app/blog/how-push-notifications-work-on-apple-and-android. [cit. 2025-05-07].

[76] FIREBASE, [b.r.]. *Best practices for FCM registration token management*. Online. Firebase. Dostupné z: https://firebase.google.com/docs/cloud-messaging/manage-tokens. [cit. 2025-05-07].

[77] ALERTZY, [b.r.]. *Understanding Push Notifications: Technical and Technological Overview*. Online. Alertzy. Dostupné z: https://alertzy.app/articles/understanding-push-notifications-technical-and-technological-overview. [cit. 2025-05-08].

[78] CHOWDRY, Bali, 2024. *Push Notification with FCM (High-Level Architecture) Key Considerations*. Online. Medium. Dostupné z: https://medium.com/%40balichowdry/push-notification-with-fcm-high-level-architecture-key-considerations-df60369307c0. [cit. 2025-05-07].

[79] ABLY, [b.r.]. *Firebase vs OneSignal*. Online. Ably. Dostupné z: https://ably.com/compare/firebase-vs-onesignal. [cit. 2025-05-08].

[80] ABLY, [b.r.]. *Amazon SNS vs Firebase*. Online. Ably. Dostupné z: https://ably.com/compare/amazon-sns-vs-firebase. [cit. 2025-05-08].

[81] FIREBASE, 2025. *Firebase Cloud Messaging*. Online. Firebase. Dostupné z: https://firebase.google.com/docs/cloud-messaging. [cit. 2025-05-08].

[82] FIREBASE, 2025. *FCM Architectural Overview*. Online. Firebase. Dostupné z: https://firebase.google.com/docs/cloud-messaging/fcm-architecture. [cit. 2025-05-08].

[83] COMPUTING, Scale, 2024. *What is Application Deployment?* Online. Scale Computing. Dostupné z: https://www.scalecomputing.com/resources/application-deployment-tools-techniques-and-best-practices. [cit. 2025-05-08].

[84] HAWKES, Alex, 2023. *Bare metal vs virtual machines vs containers: Which is the right infrastructure for me?* Online. PCCW Global. Dostupné z: https://blog.consoleconnect.com/bare-metal-vs-virtual-machines-vs-containers-which-is-the-right-infrastructure-for-me. [cit. 2025-05-09].

[85] NFWARE, 2024. *Exploring Deployment Options: Hardware vs Bare Metal vs Virtualization vs Containerization*. Online. NFWare. Dostupné z: https://blog.consoleconnect.com/bare-metal-vs-virtual-machines-vs-containers-which-is-the-right-infrastructure-for-me. [cit. 2025-05-09].

[86] NETGURU, [b.r.]. *What is Mobile App Deployment: Mobile Development Explained*. Online. Netguru S.A. Dostupné z: https://www.netguru.com/glossary/mobile-app-deployment. [cit. 2025-05-09].

[87] TEAM, Research, [b.r.]. *Mobile app deployment: here's what it is and how to plan for it*. Online. Global App Testing. Dostupné z: https://www.globalapptesting.com/mobile-app-deployment. [cit. 2025-05-09].

[88] PILKOVÁ, Anna Hana, 2023. *Webová aplikace pro výdejnu zdravotnických pomůcek pro diabetiky*. Online, bakalářská práce. Univerzita Tomáše Bati ve Zlíně. Fa-

kulta aplikované informatiky, Ústav informatiky a umělé inteligence. Dostupné z: http://hdl.handle.net/10563/54066. [cit. 2025-05-10].

SEZNAM OBRÁZKŮ

Obrázek 1.1 Natívna vs aplikácia s Capacitor 18
Obrázek 1.2 Capacitor komponenty 19
Obrázek 1.3 Capacitor natívny most 20
Obrázek 1.4 Capacitor Runtime 21
Obrázek 2.1 Proces autentifikácie v Apple zariadeniach 43
Obrázek 2.2 Proces autentifikácie v Apple zariadeniach s keychain a biometrikou 45
Obrázek 2.3 Proces autentifikácie v Android zariadeniach 46
Obrázek 2.4 Proces autentifikácie v Android zariadeniach pomocou odtlačku prsta .. 48
Obrázek 2.5 Tok komunikácie medzi klientom a bezpečným prostredím Keymaster .. 49
Obrázek 2.6 Porovnanie komunikácie pomocou HTTP a HTTPS 50
Obrázek 2.7 TCP a TLS *handshake* 52
Obrázek 3.1 Zabezpečenie a pravdepodobnosť útoku 56
Obrázek 3.2 Proces šifrovania a dešifrovania 58
Obrázek 4.1 Asynchrónna a synchrónna komunikácia medzi mikroservisami 65
Obrázek 4.2 Architektúra notifikačného systému 70
Obrázek 4.3 Architektúra FCM 72
Obrázek 5.1 Porovnanie architektúr nasadenia aplikácií: Fyzický server, Virtuálne stroje, Kontajnery 77
Obrázek 7.1 Model prípadov užitia 87
Obrázek 7.2 Architektonický model 88
Obrázek 7.3 Dátový model *user*, *order* a $notification_device$ 89
Obrázek 7.4 Dátový model *subscription* 90
Obrázek 7.5 Model tried reprezentujúci používateľa 91
Obrázek 7.6 Model tried reprezentujúci používateľa vo vzťahu k Gateway 92
Obrázek 7.7 Model tried reprezentujúci objednávku a s ňou zviazané pomocné triedy 93
Obrázek 7.8 Model tried reprezentujúci objednávku vo vzťahu k Resource 94
Obrázek 7.9 Model tried pre komunikáciu s treťou stranou 95
Obrázek 7.10 Model servís pre komunikáciu s treťou stranou 96
Obrázek 7.11 Model push notifikácií 97
Obrázek 7.12 Model na posielanie push notifikácií 97

Obrázek 8.1 Overovací email 99
Obrázek 8.2 Žiadosť o zmenu hesla - webová aplikácia 101
Obrázek 8.3 Žiadosť o zmenu hesla - mobilná aplikácia 101
Obrázek 8.4 Email s možnosťou zmeniť heslo 102
Obrázek 8.5 Prihlasovanie cez biometriku 106
Obrázek 8.6 Prihlasovanie 106
Obrázek 8.7 Registrácia 107
Obrázek 8.8 Zásady ochrany osobných údajov a obchodné podmienky 107
Obrázek 8.9 Domov 109
Obrázek 8.10 Domov 109
Obrázek 8.11 Vytváranie objednávky 111
Obrázek 8.12 Vytváranie objednávky 2 111
Obrázek 8.13 Vytváranie objednávky 3 111
Obrázek 8.14 Vytváranie objednávky 4 111
Obrázek 8.15 História objednávok 113
Obrázek 8.16 História objednávky 113
Obrázek 8.17 Účet 114
Obrázek 8.18 Profil 116
Obrázek 8.19 Zmena hesla 116
Obrázek 8.20 Vymazanie účtu 117
Obrázek 8.21 Prihlásenie s vymazaným účtom 117
Obrázek 8.22 Obrazovka úpravy notifikácií 119
Obrázek 8.23 Zmena farebného režimu 120
Obrázek 8.24 Svetlý režim 120
Obrázek 8.25 Domovská obrazovka administrátorskej časti 122
Obrázek 8.26 Domovská obrazovka administrátorskej časti 122
Obrázek 8.27 Domovská obrazovka administrátorskej časti 124
Obrázek 8.28 Odoslanie objednávky 126
Obrázek 8.29 Účet administrátora 127
Obrázek 8.30 Push notifikácia odoslaná používateľovi 132

SEZNAM TABULEK

Tabulka 1.1 Porovnanie hybridného a natívneho prístupu 16
Tabulka 1.2 Porovnanie Flutter a Ionic 29
Tabulka 2.1 Zraniteľnosti pri ukladaní dát v mobilnej aplikácii s návrhmi na ich zmiernenie/riešenie 37
Tabulka 4.1 Porovnanie Apache Kafka a REST 68
Tabulka 4.2 Porovnanie poskytovateľov push notifikácií 71

SEZNAM VÝPISŮ

Výpis 8.1 Verifikačný proces: registrácia (zdroj: vlastný) 98
Výpis 8.2 Verifikačný proces: overenie (zdroj: vlastný) 99
Výpis 8.3 Verifikačný proces: obnovenie verifikačného tokenu (zdroj: vlastný) 100
Výpis 8.4 Žiadosť o zmenu hesla (zdroj: vlastný) 100
Výpis 8.5 Zmena hesla (zdroj: vlastný) 102
Výpis 8.6 Šifrovanie a dešifrovanie (zdroj: vlastný) 103
Výpis 8.7 Žiadosť o dešifrovanie informácií o používateľovi (zdroj: vlastný) 104
Výpis 8.8 Prvotné prihlásenie (zdroj: vlastný) 104
Výpis 8.9 Prihlasovanie biometrikou (zdroj: vlastný) 105
Výpis 8.10 Okno s podmienkami ochrany osobných údajov a obchodnými podmienkami (zdroj: vlastný) 107
Výpis 8.11 Navigácia v používateľskej časti (zdroj: vlastný) 108
Výpis 8.12 Zachytenie zmeny režimu (zdroj: vlastný) 109
Výpis 8.13 Spracovanie dát do echarts (zdroj: vlastný) 110
Výpis 8.14 Vytvorenie objednávky (zdroj: vlastný) 110
Výpis 8.15 order.store vytvorenie (zdroj: vlastný) 112
Výpis 8.16 order.store vytvorenie (zdroj: vlastný) 113
Výpis 8.17 Tlačítka pre históriu podľa rokov (zdroj: vlastný) 113
Výpis 8.18 Zmena hesla (zdroj: vlastný) 115
Výpis 8.19 Zmena hesla s overením (zdroj: vlastný) 115
Výpis 8.20 Vymazanie účtu (zdroj: vlastný) 117
Výpis 8.21 Zmena odoberania notifikácií (zdroj: vlastný) 118
Výpis 8.22 Zmena odoberania notifikácií (zdroj: vlastný) 118
Výpis 8.23 Signal Store pre farebný režim (zdroj: vlastný) 120
Výpis 8.24 Navigácia v administrátorskej časti (zdroj: vlastný) 121
Výpis 8.25 Spracovanie role používateľa (zdroj: vlastný) 121
Výpis 8.26 Stĺpcový graf (zdroj: vlastný) 123
Výpis 8.27 BarChartComponent vstupy (zdroj: vlastný) 123
Výpis 8.28 BarChartComponent tvorba dát (zdroj: vlastný) 123
Výpis 8.29 AdminStore: načítanie dát (zdroj: vlastný) 125

Výpis 8.30 Odoslanie objednávky (zdroj: vlastný) 125
Výpis 8.31 Firebase konfigurácia (zdroj: vlastný) 128
Výpis 8.32 Registrácia zariadenia a používateľa (zdroj: vlastný) 129
Výpis 8.33 Žiadosť o poslanie notifikácie (zdroj: vlastný) 130
Výpis 8.34 Poslanie notifikácie administrátorovi o vytvorení novej objednávky (zdroj: vlastný) 130
Výpis 8.35 Poslanie notifikácie používateľovi o odoslaní objednávky (zdroj: vlastný) 130
Výpis 8.36 Inicializácia notifikácií (zdroj: vlastný) 131
Výpis 9.1 Adresy pre smerovanie v Gateway (zdroj: vlastný) 133
Výpis 9.2 Nastavenie adries smerovania v Gateway pre produkciu (zdroj: vlastný) .. 133
Výpis 9.3 Gateway smerovanie (zdroj: vlastný) 134
Výpis 9.4 Príklad použitia premennej zo súboru .env (zdroj: vlastný) 134
Výpis 9.5 Nastavenie Redis v mikroservise Diatop (zdroj: vlastný) 134
Výpis 9.6 Nastavenie Redis v mikroservise Admin (zdroj: vlastný) 134
Výpis 9.7 Použitia premennej zo súboru .env v Model (zdroj: vlastný) 135
Výpis 9.8 Použitia premennej zo súboru .env v Model (zdroj: vlastný) 135
Výpis 9.9 Použitie premenných pre prihlásenie do tretej strany zo súboru .env v application.properties (zdroj: vlastný) 135
Výpis 9.10 Dockerfile pre gateway (zdroj: vlastný) 136
Výpis 9.11 Súbor nginx.conf (zdroj: vlastný) 137
Výpis 9.12 Docker pre nginx (zdroj: vlastný) 137
Výpis 9.13 Súbor nginx.conf s certifikátom (zdroj: vlastný) 138
Výpis 9.14 Súbor docker-compose nginx nastavenie (zdroj: vlastný) 138
Výpis 9.15 Príkazy pre vystavanie aplikácie a synchronizáciu projektu pre Apple (zdroj: vlastný) 139

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CI/CD	Continuous Integration and Continuous Deployment
CSRF	Cross-site request forgery
CSS	Cascading Style Sheets
CORS	Cross-Origin Resource Sharing
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
REST	Representational State Transfer
SDK	Software Development Kit
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VPS	Virtual Private Server
Wi-Fi	Wireless Fidelity
XSS	Cross-site scripting

SEZNAM PŘÍLOH

PŘÍLOHA P I: Súbor prilohy

PŘÍLOHA P I: SÚBOR PRILOHY

Súbor obsahuje:

- zložka **diatop-mobile-app** - obsahuje implementáciu mobilnej aplikácie,
- zložka **diatop-web-app** - obsahuje implementáciu webovej aplikácie,
- súbor **Readme.pdf** - návod na spustenie mobilnej aplikácie a spôsoby testovania webovej a mobilnej aplikácie.