FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Lokalizační systém PROWiLOS v1.0
Student:	Bc. Vít Medřický
Vedoucí:	Ing. Filip Křikava, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

PROWiLOS je systém na sledování polohy pacientů v pečovatelském domě pomocí lokalizačních zařízení.

Cílem této práce je navázat na prototyp aplikace vypracovaný v bakalářské práci stejného autora, přidat podporu lokalizačních zařízení a dodělat systém do stavu nasaditelného do testovacího prostředí.
1. Za pomoci metodiky softwarového inženýrství analyzujte aktuální požadavky na systém a s pomocí výstupu z bakalářské práce "Webová aplikace PWiL - Systém pro lokalizaci pacientů" navrhněte kroky vedoucí k splnění nových požadavků.
2. Navrhněte a implementujte komunikační protokol mezi systémem PROWiLOS a lokalizačními zařízeními pro pacienty.
3. Navrhněte a implementujte komunikační protokol mezi systémem a mobilní aplikací.
4. Dle návrhu v prototypu realizujte komunikaci systému s lokalizačním serverem.
5. Navrhněte a implementujte nezbytné chybějící části uživatelského rozhraní.
6. Zhotovenou aplikaci včetně jednotlivých komunikací podrobte vhodnému testování.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 23. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Lokalizační systém PROWiLOS v1.0

Bc. Vít Medřický

Katedra softwarového inženýrství
Vedoucí práce: Ing. Filip Křikava, Ph.D.

14. února 2019

Poděkování

Chtěl bych poděkovat mému vedoucímu Ing. Filipu Křikavovi, Ph.D. za velmi pozitivní přístup, podnětné připomínky a zkušené rady k obsahu a formě této práce. Důležité poděkování patří všem mým spolupracovníkům, kteří se mnou konzultovali technické zpracování tohoto projektu. V neposlední řadě děkuji své rodině, přítelkyni a všem přátelům za neuvěřitelnou morální podporu, díky které tato práce mohla vzniknout.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 14. února 2019 .....

Odkaz na tuto práci

Medřický, Vít. *Lokalizační systém PROWiLOS v1.0*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce navazuje na prototyp webového systému PWiL zhotoveného v bakalářské práci stejného autora. PWiL je systém lokalizace pacientů v pečovatelských domech umožňující efektivní přivolání pomoci pro pacienty v případě nouze. Z prototypu z bakalářské práce byl vytvořen hotový systém připravený k nasazení do testovacího provozu. Uživatelské rozhraní webové aplikace je doplněno o nezbytné chybějící části a jsou implementována a testována komunikační rozhraní. Aplikace je napsána v PHP s použitím frameworku Laravel.

Klíčová slova lokalizační systém, webová aplikace, SPoT API, Laravel, PHP

Abstract

This thesis follows on a prototype of web system PWiL created in author's bachelor thesis. PWiL is a system for patient localization in nursery homes allowing effective call for help when necessary. From the prototype in the bachelor thesis a functional system has been created and is now suitable for deployment into a trial run. Missing functionalities of the web apllication user interface are added and communication interfaces are implemented and tested. The application is implemented in PHP using the Laravel framework.

Keywords localization system, web application, SPoT API, Laravel, PHP

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V domech s pečovatelskou službou denně nastávají případy, kdy pacient potřebuje asistenci zdravotního pracovníka. V mnoha případech svůj účel splní signalizační tlačítka, zpravidla umístěná v pacientově pokoji. Problém nastává ve chvíli, kdy se pacient pohybuje mimo svou místnost a dostane se do stavu, kdy potřebuje pomoc[1]. Zejména v místech, kde trvá delší dobu, než si někdo z pečovatelů problému všimne, např. v koupelně, na toaletě, či v odlehlých prostorách budovy. V takovou chvíli je potřeba, aby existoval způsob, jak si pacient přivolá pomoc a pečovatelé budou ihned vědět, kde pacienta hledat. Okamžitá pomoc totiž může znamenat nejen předejití zdravotním komplikacím, ale v kritických případech i záchranu života.

Jedno z možných řešení je aplikace, která umožní pečovatelům lokalizaci a pacientům signalizaci. Problém konstrukce lokalizačně-signalizačního systému jsem začal řešit ve své bakalářské práci Webová aplikace PWiL - Systém pro lokalizaci pacientů[1]. Aplikace PWiL se skládá z následujících komponent:

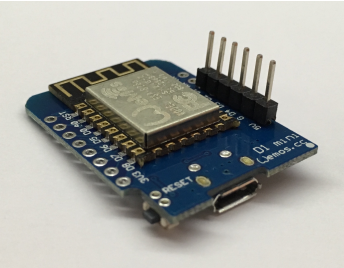
Webová aplikace - funkční jádro systému. Jejím úkolem je správa databáze (pacientů, uživatelů, náramků) a zprostředkování komunikace mezi lokalizačním serverem, mobilními aplikacemi a náramky.

Webové rozhraní - uživatelské rozhraní pro administraci a přehled systému. Slouží jako hlavní rozhraní pro správu systému, umožňuje uživateli práci se záznamy pacientů, náramků a uživatelů. V bakalářské práci bylo toto rozhraní připraveno pro základní operace nad databázovými entitami.

Lokalizační server (SPoT) - specializovaný server, který bude poskytovat webové aplikaci přehled o polohách lokalizovaných zařízení. Produkt poskytuje firma *Ruckus* [2], jako komunikační rozhraní používá *SPoT API* [3].

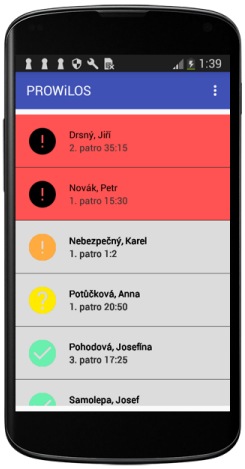
[1] Dle informací pečovatelského domu je běžné, že pacient upadne a nemůže se zvednout.

Náramek - zařízení pro lokalizaci a signalizaci pacienta. Připojením k bezdrátové síti jsou tyto náramky lokalizovány pomocí výše uvedeného SPoT systému. Signalizace (volání o pomoc) je zprostředkována stisknutím tlačítka na náramku. Konkrétní hardware představuje vývojová deska *Wemos D1 mini* [4] s přístupem k wi-fi.



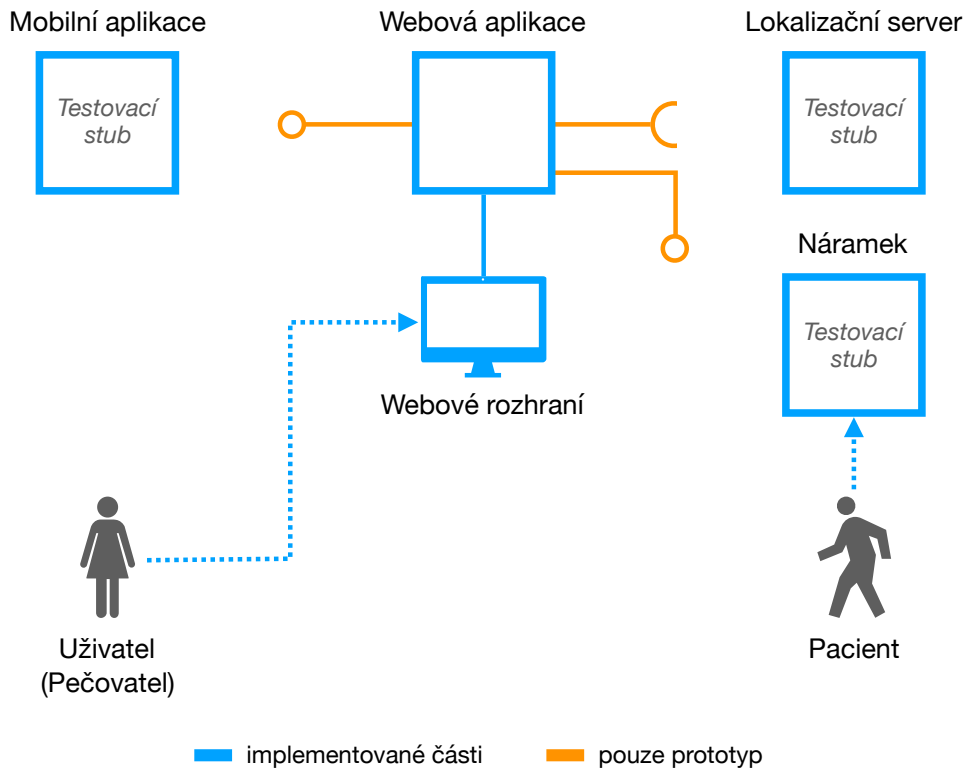
Obrázek 0.1: Hardware náramku - Wemos D1 mini

Mobilní aplikace - zjednodušená aplikace pro přehled nad stavem a polohami pacientů. Získává informace z webové aplikace a zobrazuje je uživateli (pečovateli). Zhotovení aplikace nebylo náplní této práce, pouze návrh a implementace komunikace s ní.



Obrázek 0.2: Mobilní aplikace PROWiLOS

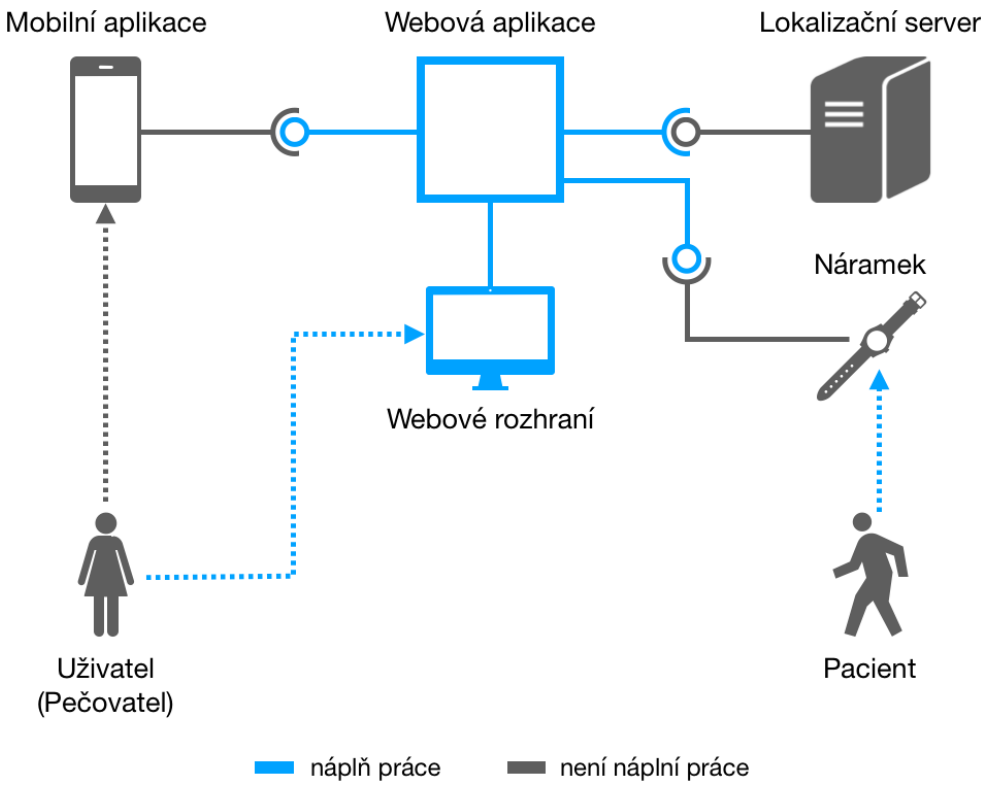
Komunikaci těchto komponent v prototypu popisuje následující schéma:



Obrázek 0.3: Schéma prototypu webové aplikace z bakalářské práce

Implementovány byly základní funkcionality pro obsluhu webového rozhraní a přiřazování náramku k pacientům. Komunikace s mobilní aplikací, lokalizačním serverem a náramky byly zhotoveny ve formě prototypu, na kterém bylo možné ověřit navrženou koncepci. Výstupem bakalářské práce byla první verze systému na kterou v této práci navážeme.

Ze schématu 0.3 jsou některé chybějící části systému zřejmé, jiné skryté. Tato diplomová práce posouvá vypracovaný prototyp k systému nasaditelnému do testovacího provozu. Nejpodstatnějším rozdílem mezi bakalářskou a diplomovou prací je doplnění komunikací jednotlivých komponent s webovou aplikací. Výsledný systém lze popsat schématem (0.4) na následující straně.



Obrázek 0.4: Schéma systému PROWiLOS v1.0

Cíl práce

Výstup bakalářské práce byl prvním krokem ve vývoji tohoto projektu. K zapojení sytému do produkce je třeba ještě provést mnoho nezbytných kroků. Cílem této diplomové práce je doplnit systém o podporu lokalizačních zařízení a mobilních aplikací, navázat komunikaci s lokalizačním serverem a dopracovat chybějící části uživatelského rozhraní.

Motivace

Tato diplomová práce vychází ze zadání mého zaměstnavatele, firmy Probox a logicky navazuje prototyp vypracovaný v rámci mé bakalářské práce. Díky jeho vypracování rozvíjím své dovedností v oblasti softwarového inženýrství a vývoje webových aplikací. Zároveň věřím, že výsledná aplikace přispěje k modernizaci služeb v místech, kde je potřeba.

Struktura práce

Kapitola 1 - Návaznost na prototyp, analýza nových požadavků a případů užití;

Kapitola 2 - Návrh a implementace komunikací webové aplikace s náramkem, mobilní aplikací a lokalizačním serverem a doplňujících částí uživatelského rozhraní;

Kapitola 3 - Testování komunikací webové aplikace s náramkem, mobilní aplikací a lokalizačním serverem a souvisejících mechanismů;

Kapitola 4 - Závěr a rozbor splnění zadání.

Kapitola 1

Analýza

1.1 Od prototypu k verzi 1.0

Verze 0.1 (prototyp) přinesl poznatky k možnostem technologií, které můžeme k řešení problému lokalizace a signalizace využít. Systém byl představen zadavateli formou prezentačního dema a byly konzultovány výsledky. Jako další krok byla zvolena příprava systému k testovacímu provozu a byly specifikovány nové funkční požadavky. K navázání na předchozí práci je potřeba tyto nové požadavky analyzovat, rozpracovat případy užití a na jejich základě navrhnout a implementovat aplikaci. V této kapitole rozebereme funkční požadavky, specifikujeme nefunkční požadavky, analyzujeme případy užití a zobrazíme mechanismy předávání informací v systému PROWiLOS v1.0.

1.2 Funkční požadavky

Pro novou verzi máme nové požadavky, které je třeba důkladně rozepsat a zhodnotit jejich překryv s prototypem z bakalářské práce. Jelikož zůstávají všechny vyhotovené součásti i v nové verzi systému, pro přehled stručně zobrazíme i požadavky na původní systém, které již byly splněny. Tak získáme lepší představu nad tím, které funkcionality je do systému třeba přidat či zdokonalit a které funkcionality už systém umí.

Za tímto účelem byla vytvořena tabulka 1.1 s přehledem všech funkčních požadavků s příslušnou úrovní splnění v prototypu.

FP-#	Název požadavku	Míra splnění
FP-1	Uživatelské účty	splněno v prototypu (3/3)
FP-2	Uživatelské role	splněno v prototypu (3/3)
FP-3	Přihlašování do webového rozhraní	splněno v prototypu (3/3)
FP-4	Správa databázových entit	splněno v prototypu (3/3)
FP-5	Zobrazení stavu a lokací pacientů	splněno v prototypu (3/3)
FP-6	Párování pacientů s náramky	rozpracováno (2/3)
FP-7	Notifikace volání o pomoc	nový požadavek (0/3)
FP-8	Řešení volání o pomoc	nový požadavek (0/3)
FP-9	Správa profilu	nový požadavek (0/3)
FP-10	Načítání lokací z lokalizačního serveru	hrubě rozpracováno (1/3)
FP-11	Komunikace s mobilní aplikací	část rozpracována (2/3)
FP-12	Komunikace s náramky	hrubě rozpracováno (1/3)
FP-13	Zpětná vazba webového rozhraní	nový požadavek (0/3)

Tabulka 1.1: Přehled funkčních požadavků; 0 - nesplněno, 3 - splněno

Požadavky 1–5 byly dostatečně splněny už v prototypu v bakalářské práci a nemá smysl je dále rozebírat. V následující části uvedeme bližší specifikaci zbylých požadavků s komentářem stavu rozpracování.

FP-6: Párování - Webové rozhraní umožní propojení záznamu pacienta se záznamem náramku.

Tento požadavek byl v prototypu rozpracován. Bylo implementováno propojení pacienta s náramkem, ale uživatelské rozhraní neposkytuje žádnou informaci, dle které by šlo určit, zda je náramek již někomu přidělen, nebo je volný. Obdobně je tomu s pacientem. Stránka také neposkytuje zpětnou vazbu při spárování záznamů pacienta a náramku. Bude nutné navrhnout a implementovat přehledné zobrazení volných náramků a nepřiřazených pacientů a doplnit potvrzující při úspěšně provedené operaci, případně varovné hlášky při spárování již připojeného náramku.

FP-7: Notifikace volání o pomoc ve web. rozhraní - Webové rozhraní bude notifikovat uživatele v případě volání o pomoc. Zobrazí upozornění v notifikačním centru pomocí zablikání, nebo jiného upozornění.

FP-8: Řešení volání o pomoc ve web. rozhraní - Webové rozhraní umožní uživateli vyřešit volání o pomoc. kliknutím na tlačítko bude možné *dokončit* volání. V ideálním případě s přístupem přímo přes notifikační centrum.

FP-9: Správa profilu - Webové rozhraní umožní každému uživateli (nezávisle na jeho roli - administrátor/uživatel) upravit údaje o vlastním uživatelském účtu.

FP-10: Načítání lokací náramků z lok. serveru - Webová aplikace bude získávat poslední známe lokace náramků od lokalizačního serveru.

Tento požadavek byl v prototypu hrubě rozpracován. Jelikož nebylo možné testovat komunikaci na reálném lokalizačním serveru, byla vytvořena komponenta webové aplikace, která má na starost získávání lokací na příslušné adrese. Tato implementace byla provedena na základě dostupné dokumentace SPoT API (api lokalizačního serveru). Je třeba komponentu upravit tak, aby se pravidelně dotazovala reálného SPoT API na lokace náramků.

FP-11: Komunikace s mobilní aplikací - Webová aplikace poskytne aplikační rozhraní pro mobilní aplikace, které umožní:

1. přihlášení k mobilní aplikaci pomocí uživatelského účtu
2. předání informací o změnách lokací
3. předání informací o volání o pomoc
4. předání informací o změnách údajů pacientů
5. vyřešení volání o pomoc z mobilní aplikace
6. odhlášení z mobilní aplikace

Bod b) byl v prototypu rozpracován, ale z následného testování vyplynulo, že bude muset být upraven. Body c) a d) byly v prototypu zpracovány velmi hrubě a s jejich pozdější optimalizací se počítalo. Bude zapotřebí navrhnout a implementovat komunikační protokol pro přihlašování k uživatelskému účtu, doplnit možnost řešení volání o pomoc a optimalizovat poskytované údaje.

FP-12: Komunikace s náramky - Webová aplikace poskytne náramkům aplikační rozhraní pro komunikaci. Toto rozhraní bude umožňovat:

1. připojení nového náramku - náramek se registruje do systému zasláním požadavku na připojení
2. předání informace o volání o pomoc - stiskem signalizačního tlačítka odešle náramek volání o pomoc
3. předání periodické informace - automaticky a periodicky odesílaná data z různých čidel na náramku

Z tohoto požadavku byl v prototypu rozpracován jen bod b) a jeho implementace byla primitivní. Tento způsob předávání volání o pomoc bude muset být upraven v souvislosti s dalšími požadavky na komunikaci s náramky.

FP-13: Zpětná vazba webového rozhraní - Uživatelské rozhraní bude podávat uživateli zpětnou vazbu na provedené operace. V případě úspěšných změn potvrdí provedenou úpravu, bude varovat v případě, kdy hrozí, že uživatel provedl nestandardní operaci a bude reagovat chybnou hláškou v případě, že uživatel nevalidně vyplní formulář.

1.3 Nefunkční požadavky

Nefunkční požadavky specifikují kritéria k hodnocení aplikace z ostatních pohledů, kromě funkcionality. Pro tuto práci zadavatel definoval následující požadavky, které je třeba splnit:

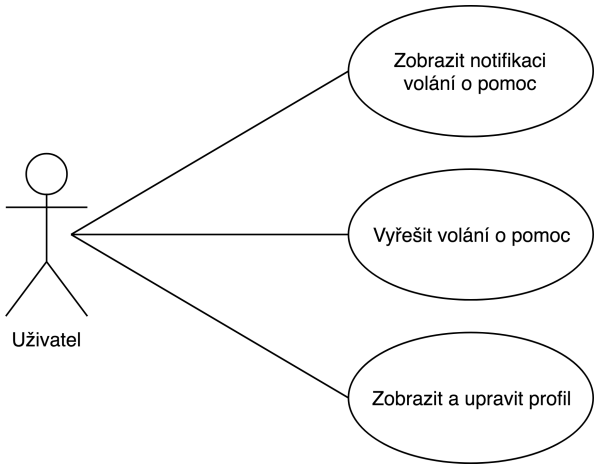
1. Implementace v PHP frameworku *Laravel* [5].
2. Verzování zdrojových kódů pomocí nástroje *git* v předem stanoveném firemním repozitáři.
3. Nahrání vyhotovené aplikace systému na firemní server.

1.4 Případy užití

V bakalářské práci byly detailně řešeny případy užití pokrývající především uživatelské (webové) rozhraní. Pro navazující práci nejsou podstatné a nebudeme je tedy upravovat, ani zmiňovat. Ukážeme si nové případy užití systému z pohledu všech aktérů: uživatele, náramku, mobilní aplikace i webové aplikace.

1.4.1 Nové případy užití pro uživatele

K případům užití pro uživatele přibyly 2 interakční skupiny s webovým rozhraním. Jedná se o řešení volání o pomoc a správu profilu.

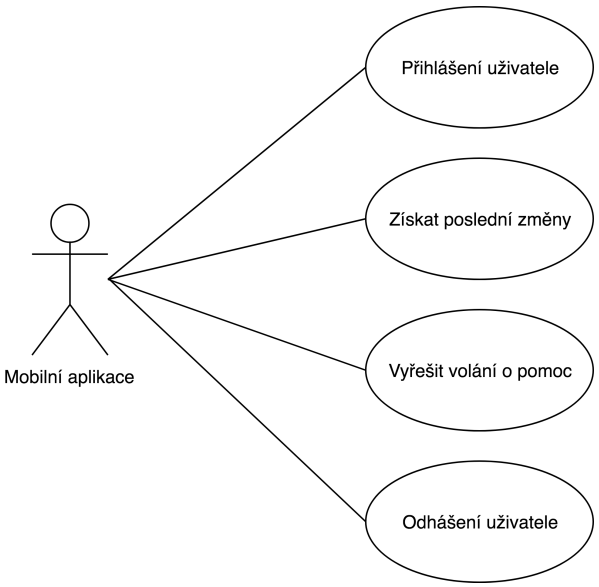


Obrázek 1.1: Diagram případů užití - uživatel

1. Uživatel ve webovém rozhraní zobrazí notifikaci kliknutím na notifikační centrum.
2. Uživatel vyřeší volání o pomoc kliknutím na tlačítko *Vyřešit volání o pomoc* v záznamu pacienta.
3. Uživatel zobrazí stránku s vlastními údaji, upraví je pomocí tlačítka *Upravit*.

1.4.2 Případy užití pro mobilní aplikaci

Mobilní aplikace bude vystupovat jako aktér v komunikaci s webovou aplikací. Cílem této komunikace je možnost přihlášení se k uživatelskému účtu, získávání potřebných informací o pacientech a řešení volání o pomoc.

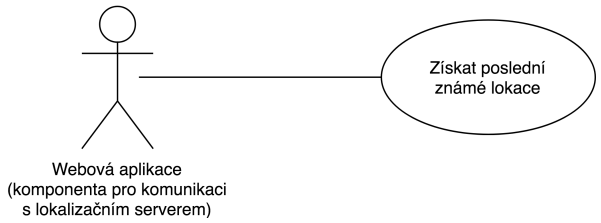


Obrázek 1.2: Diagram případů užití - mobilní aplikace

1. Mobilní aplikace přihlásí uživatele zasláním jeho přihlašovacích údajů webové aplikaci.
2. Mobilní aplikace získá výpis posledních změn zasláním dotazu webové aplikaci.
3. Mobilní aplikace vyřeší volání o pomoc zasláním požadavku webové aplikaci.
4. Mobilní aplikace odhlásí uživatele zasláním požadavku webové aplikaci.

1.4.3 Případy užití pro webovou aplikaci

V těchto komunikacích je aktérem i webová aplikace, konkrétně komponenta pro komunikaci s lokalizačním serverem. Její úkol je prostý, ale důležitý.

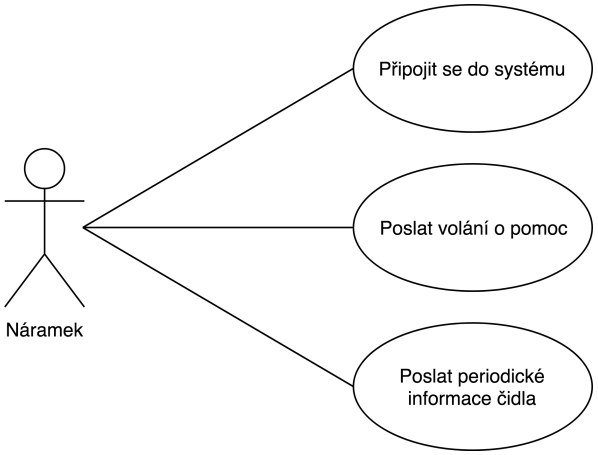


Obrázek 1.3: Diagram případů užití - webová aplikace

Webová aplikace získá pomocí dotazu na SPoT API poslední známé lokace náramků od lokalizačního serveru.

1.4.4 Případy užití pro náramky

Posledním aktérem v komunikaci je náramek, který bude potřebovat možnost registrace do systému, předání volání o pomoc a pravidelné posílání naměřených dat z příslušných senzorů.



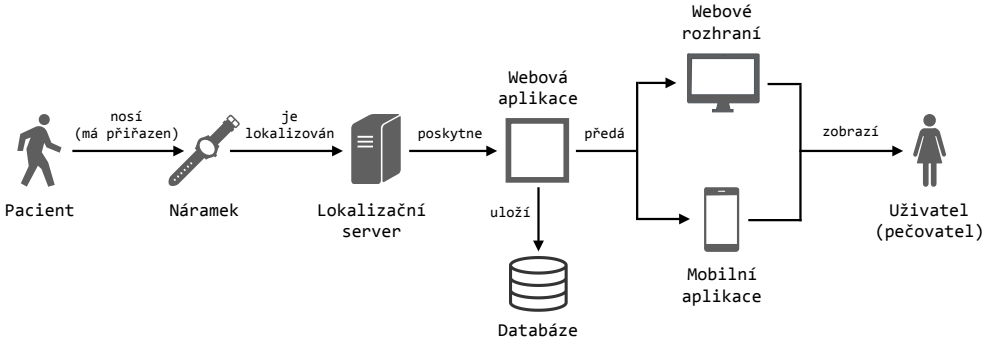
Obrázek 1.4: Diagram případů užití - náramek

1. Náramek se připojí do systému zasláním požadavku webové aplikaci.
2. Náramek předá volání o pomoc zaslání požadavku webové aplikaci.
3. Náramek předá periodické informace z čidla zasláním požadavku webové aplikaci.

1.5 Předávání informací v systému

Pro názornější ilustraci předávání informací v systému a komplexního nastínění případů užití byla vytvořena pomocná schémata.
První schéma (1.5) popisuje předávání informace o lokaci od lokalizovaného pacienta po uživatele mobilní aplikace, nebo webového rozhraní.

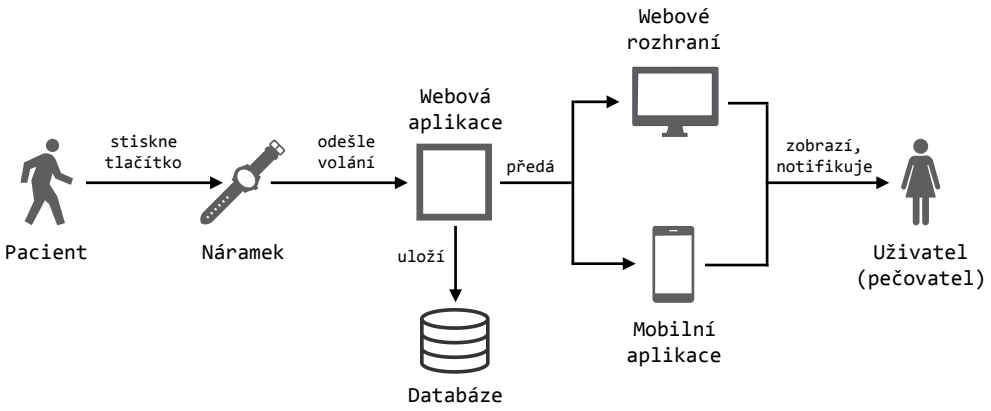
1. Pacient má v systému přiřazen náramek, který nosí stále s sebou.
2. Náramek je lokalizován SPoT systémem a údaje jsou ukládány na lokalizačním serveru.
3. Webová aplikace získává od lokalizačního serveru poslední známé lokace.
4. Webová aplikace lokace ukládá do vlastní databáze a předává je mobilní aplikaci a zobrazuje je pomocí webového rozhraní.
5. Mobilní aplikace i webové rozhraní zobrazují uživateli poslední známé lokace pacientů.



Obrázek 1.5: Schéma předávání lokací

Druhé schéma (1.6) popisuje předání informace o volání o pomoc od pacienta po uživatele (pečovatele).

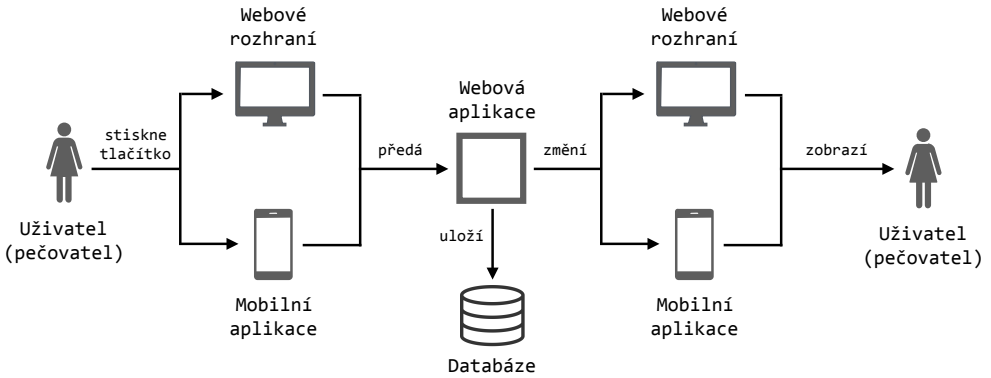
1. Pacient má v systému přiřazen náramek, na kterém pro zavolání pomoci zmáčkne tlačítko.
2. Náramek odešle informaci webové aplikaci.
3. Webová aplikace volání uloží do databáze předá informaci mobilní aplikaci a webovému rozhraní.
4. Mobilní aplikace i webové rozhraní notifikují uživatele a zobrazí volání o pomoc.



Obrázek 1.6: Schéma předávání volání o pomoc

Třetí schéma (1.7) popisuje vyřešení volání o pomoc uživatelem (pečovatelem). To nastává v případě, kdy uživatel dohledá pacienta, který volal o pomoc a poskytne mu potřebnou asistenci. V takovou chvíli je nutné do systému předat informaci, že o pacienta bylo postaráno a jeho stav již není v ohrožení.

1. Uživatel stiskne tlačítko pro vyřešení volání o pomoc v mobilní aplikaci, nebo webovém rozhraní.
2. Mobilní aplikace nebo webové rozhraní předají tuto informaci webové aplikaci.
3. Webová aplikace zpracuje požadavek, uloží změny do databáze a předá nové údaje opět mobilní aplikaci a webovému rozhraní.
4. Mobilní aplikace a webové rozhraní zobrazí uživateli nový stav pacienta, který již nevolá o pomoc.



Obrázek 1.7: Schéma vyřešení volání o pomoc

Kapitola 2

Návrh a implementace

Tato kapitola se zabývá návrhem a implementací:

1. komunikace webové aplikace s lokalizačním (SPoT) serverem,
2. komunikace webové aplikace s náramkem,
3. komunikace webové aplikace s mobilní aplikací,
4. uživatelského rozhraní[2]

Jelikož vytváříme nadstavbu prototypu z bakalářské práce, je nutné navrhovat nové části v návaznosti na něj. To přináší jistá omezení v předem stanovených technologiích, ale zároveň vede k lepší představě o konkrétní podobě dané části systému. Jednou z nejpodstatnějších použitých technologií, která byla předem určena kvůli návaznosti na předchozí prototyp, je Laravel framework.

2.1 Laravel framework

V prototypu byl k implementaci použit framework Laravel pro jazyk PHP. Zadavatel si tuto technologii vyžádal na základě vlastní analýzy [6] a z důvodu budoucí možnosti integrovat aplikaci PROWiLOS do vlastního širšího systému, v němž by tato diplomová práce byla jedním z modulů. Při práci na prototypu se neprojevily žádné výrazné nedostatky tohoto frameworku, naopak usnadnil spoustu procesů - vytváření a upravování databázových entit pomocí *migrací*, automatické plnění databáze testovacími daty pomocí *seederů*, generování nových tříd příkazem *artisan* ad. Značnou výhodou je i rozsáhlá nabídka pomocných nástrojů, instruktážních videí a diskusních fór díky široké komunitě uživatelů.

[2] Jeho chybějících a doplňujících částí.

V porovnání s předchozí prací, kde jsme implementovali základní funkcionality pro běžné operace nad databází, řeší tato práce specifičtější problematiku. Jak si ukážeme, adresářová struktura Laravelu neposkytuje žádné nativní umístění pro vlastní typy pomocných tříd, jejichž tvorbě se při implementaci rozsáhlejšího projektu nevyhneme. Je tedy třeba do organizované struktury vnést vlastní pravidla pro organizaci souborů.

2.2 Komunikace s lokalizačním (SPoT) serverem

Tato sekce se věnuje návrhu a implementaci komponent webové aplikace, jejichž odpovědností je pravidelné dotazování lokalizačního serveru na poslední známé polohy náramků a ukládání získaný údajů do databáze.

Komunikace probíhá přes SPoT API poskytované lokalizačním serverem. Komunikační rozhraní ze strany lokalizačního serveru je zde definováno přímo dokumentací SPoT API[3], řešíme tedy aplikační logiku pro komunikaci ze strany webové aplikace.

V prototypu z bakalářské práce již byl zhotoven mechanismus pro uložení odpovědi serveru, ale návazné funkcionality nikoliv. Je třeba systém doplnit o komponenty zajišťující:

1. stažení posledních známých lokací z rozhraní definovaného v dokumentaci SPoT API,
2. zpracování odpovědi ve formátu dle dokumentace SPoT API (data ve formátu json) a uložení dat do příslušných databázových struktur,
3. pravidelné opakování tohoto mechanismu za cílem udržování aktualizovaného systému, v daném časovém intervalu (v řádu sekund[3]).

2.2.1 Autentizace a dotazování na lokace

První částí této komunikace je komponenta stahující poslední známé lokace ze SPoT API. V prototypu tuto službu simulovala pomocná třída `SpotApiLocationsDownloader` voláním `file_get_contents()` ze simulační adresy na níž byl umístěn *stub*[4]. Prototyp ale při volání neřešil autentizaci. Dle dokumentace je při posílání dotazu třeba posílat API klíč, který SPoT server poskytne na základě zaslání přihlašovacích údajů. Je tedy nutné nejprve získat klíč a poté se jím autentizovat při každém dalším dotazu.

[3] Potřebujeme zajistit schopnost systému udávat aktuální pozice pacientů, čím delší intervaly dotazování budou, tím delší bude reakční doba systému na pohyb pacienta.

[4] Pomocná část kódu vracející hodnotu, která simuluje reálné chování.

Získávání posledních známých lokací od serveru se tedy skládá ze 2 základních kroků:

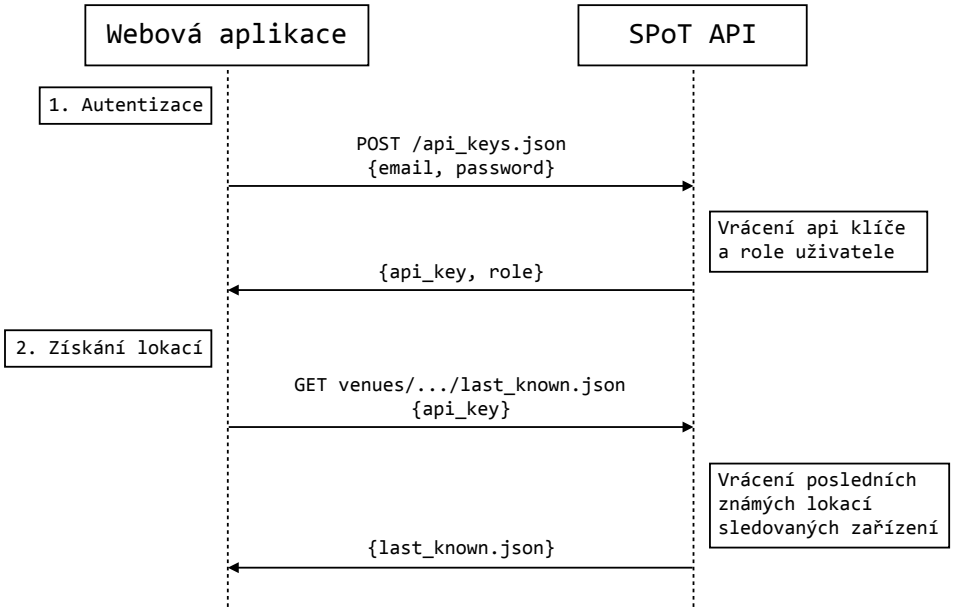
1. autentizace - zaslání přihlašovacích údajů na adresu `.../api_keys.json` dle dokumentace SPoT API a následné obdržení API klíče,

2. získání lokací - zaslání dotazu na `.../locations/last_known.json` dle dokumentace SPoT API,

kde 1. krok bude proveden pouze při prvním spuštění. V tomto případě máme na výběr implementovat automatické přihlášení do kódu aplikace, nebo necháme tento krok na administrátorovi, aby klíč získal manuálně např. použitím příkazu `curl`, jak je uvedeno v dokumentaci SPoT API:

```
curl -X POST
     -H "Content-Type: application/json"
     -d '{"email": USERNAME, "password": PASSWORD}'
     http://IP:PORT/api/v1/api_keys.json
```

Vzhledem k faktu, že klíč je třeba získávat pouze 1x na instalaci, postačí nám v tomto případě manuální řešení; 2. krok bude opakován v časovém úseku stanoveném dle potřeb služby. Detailněji vysvětluje komunikaci následující schéma:



Obrázek 2.1: Schéma komunikace se SPoT API

Získání lokací 2.1(2) je implementováno pomocí samostatné pomocné třídy `LocationsDownloader`, která má na starost dotazování a přijímání odpovědi od serveru. Stahuje data z adresy ve tvaru:

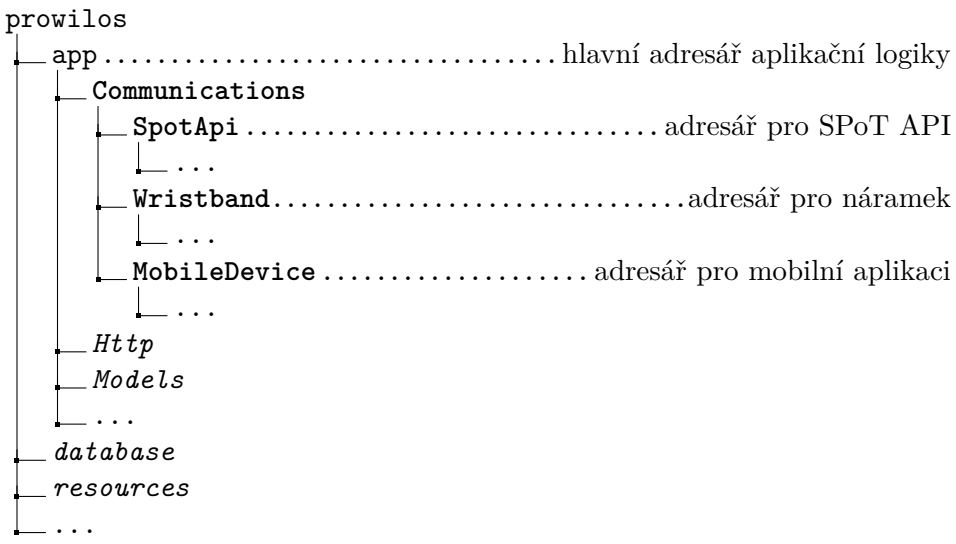
```
http://SRV_IP:PORT/api/v1/venues/vspot/locations/last_known.json
```

Ke zjednodušení zasílání HTTP požadavků s parametry (posíláme api klíč s každým požadavkem) využívá `Guzzle PHP HTTP` klienta[7].

2.2.1.1 Adresářová struktura pro pomocné třídy

Framework Laravel standardně řeší třídění příslušných tříd podle jejich funkcionalit v projektu. Každá část kódu má tak své přesné umístění, kde by se měla nacházet. Na rozdíl od předchozích implementací z BP zde vytváříme pomocnou třídu, jejíž použití je specifické právě pro náš projekt, tudíž ji nelze zařadit do již existujících adresářů. Obecně lze strukturovat takovéto pomocné třídy dvěma hlavními způsoby (nebo jejich kombinací). Buď bude třída umístěna ve stejném, nebo sousedním adresáři, jako komponenta, které se třída týká; nebo vytvoříme oddělený adresář s danou tématikou v hierarchicky nejvyšší úrovni aplikace, do které lze funkcionalitu zařadit.

Jelikož se tento projekt věnuje několika různým komunikacím s vnějšími systémy, nabízí se vytvoření adresáře `Communications/` na nejvyšší úrovni aplikační logiky, který bude obsahovat veškeré pomocné třídy pro tyto typy komunikace. Vzhledem k dalším plánovaným implementacím byla připravena následující struktura:



Obrázek 2.2: Ukázka vytvořené adresářové struktury

2.2.2 Databázový model pro ukládání lokací

Lokace získané od SPoT serveru je třeba ukládat do databáze. Prototyp za tímto účelem ukládal odpovědi serveru v podobě, jaké je SPoT prezentuje (mac adresa zařízení, časové razítko a souřadnice `x`, `y`).

Naším cílem je ukládat lokace náramků v návaznosti na propojené pacienty. Za tímto účelem bude aplikace užívat tabulku `last_known_locations`, do které jsou kvůli jednoznačnému určení lokace, náramku, přiděleného pacienta a času, kdy se na daném místě náramek nacházel, ukládány následující údaje:

`id` - identifikátor záznamu (primární klíč)

`wristband_id` - identifikátor náramku (cizí klíč)

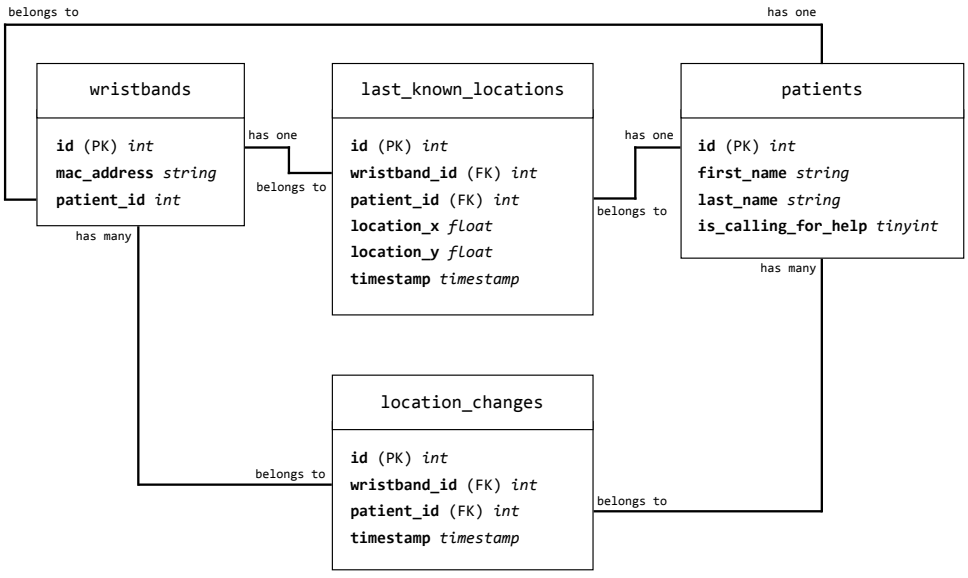
`patient_id` - identifikátor pacienta (cizí klíč)

`location_x` - souřadnice na ose x

`location_y` - souřadnice na ose y

`timestamp` - čas výskytu v dané lokaci

Návaznost tabulky posledních známých lokací na další databázové entity znázorňuje následující schéma 2.3.



Obrázek 2.3: DB schéma - `last_known_locations`

K implementaci této části databáze lze využít předpřipravených nástroje Laravelu. Použijeme rozhraní pro příkazovou řádku *Artisan* pro vygenerování *modelu*, *migrace* a *seederu*. Model nám umožní objektově relační mapování[5], migrací vygenerujeme databázové schéma[6] a seederem naplníme novou tabulku vzorovými daty (v tomto případě za testovacími účely v průběhu vývoje, později bude aplikace získávat tato data od lokalizačního serveru). Implementací těchto částí připravíme úložiště pro získané lokace a jelikož se jedná o standardní postupy použití Laravelu, nemá smysl je hlouběji rozebírat.

2.2.3 Zpracování odpovědí a průběžné dotazování

K doplnění funkčnosti mechanismu pro komunikaci s lokalizačním serverem zbývá:

1. transformovat odpověď serveru do podoby, v jaké chceme údaje o lokace ukládat,
2. ukládat transformované údaje do připravené databáze,
3. zajistit pravidelné volání tohoto procesu.

2.2.3.1 Procesor odpovědí serveru

SPoT server odpovídá na dotazy posledních známých lokací ve formátu json:

```
[
  { mac, timestamp, x, y, floor_number, located_inside, zones },
  ...
],
```

kde `mac` - mac adresa lokalizovaného zařízení, `timestamp` - časové razítko výskytu v lokaci na souřadnicích `x, y`. Ostatní parametry v této verzi zatím nevyužijeme. Struktura odpovídající našemu ukládání musí odpovídat tvaru:

```
[
  { wristband_id, patient_id, timestamp, location_x, location_y },
  ...
],
```

[5] ORM slouží k mapování objektů programovacího jazyka na relační databázi a tak výrazně usnadňuje práci s databází v kódu[10].

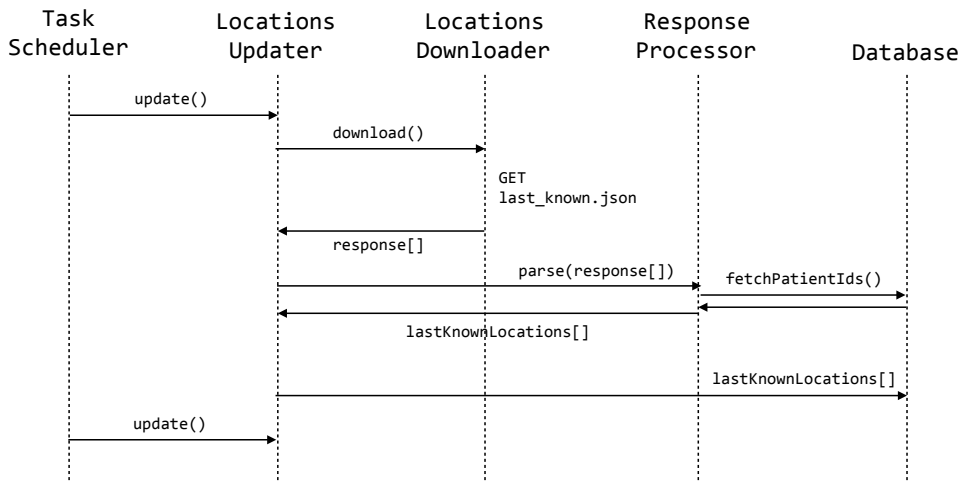
[6] Migrace také slouží k *verzování* databáze[11]. Specifikuje totiž změny, které mají nad danou částí databáze nastat. Lze tak měnit strukturu již existující databázové entity bez nutnosti mazání uložených dat.

kde `wristband_id` je id náramku s příslušnou mac adresou, `patient_id` je id pacienta přiřazeného k tomuto náramku.

Protože se tato funkcionalita netýká přímo stahování dat, neměl by ji vykonávat `LocationsDownloader`. Je proto vhodné vytvořit další pomocnou komponenty, která bude zpracovávat příchozí odpovědi serveru do formy vhodné pro další práci s těmito daty. Tuto komponentu implementujeme jako samostatnou pomocnou třídu `ResponseProcessor`.

2.2.3.2 Pravidelné spouštění updateru

Všechny vytvořené komponenty je třeba propojit a celý proces iniciovat pravidelným voláním. Tuto funkci má na starost komponenta `LocationsUpdater`, jejímž úkolem bude využít připravené mechanismy k udržování uložených lokací v aktualizovaném stavu. Roli této třídy v komunikaci s ostatními komponentami popisuje následující schéma (2.4).



Obrázek 2.4: Schéma komunikace updateru s ostatními komponentami

`LocationsUpdater` využívá `LocationsDownloader` ke stahování lokací, odpověď ze SPoT serveru nechává zpracovat `ResponseProcessorem` a zpracované údaje o lokacích ukládá do databáze.

Pravidelné volání updateru je řešeno pomocí softwarového démona *cron* v kombinaci s nástrojem *Laravel Task Schedulerem*[12]. Cron je jednoduchý časový plánovač úkolů, který lze využít k automatickému volání dané funkce, v našem případě k pravidelnému spouštění updatu lokací. Laravel zároveň obsahuje vlastní scheduler, který umožňuje časové plánování bez nutnosti specifikace každého volání přímo v cronu a přenechává tak plánovací logiku

ve zdrojovém kódu aplikace. K jeho spuštění stačí přidat do *crontabu*[7] jediný příkaz s voláním scheduleru.

```
* * * * * cd /path-to-your-project && php artisan schedule:run >> /dev/null 2>&1
```

Obrázek 2.5: Přidání volání scheduleru do crontabu

Pojem *scheduler* representuje v Laravelu třída `Kernel`, která obsahuje metodu `schedule()`, ve které můžeme volat naše plánované úlohy s nastavitelnou časovou periodicitou. V praxi to tedy znamená, že:

1. cron aktivuje scheduler,
2. scheduler zavolá updater,
3. updater vykoná proces stahování, procesování a uložení lokací, jak jsme rozepsali výše.

Problém, který představují oba plánovače spočívá v jejich nejkratším možném intervalu pro opětovné volání, které je limitováno na maximálně 1x za minutu. Jelikož se náš požadavek na dotazování pohybuje v rámci sekund, potřebujeme najít způsob, jakým interval volání zkrátit. Pro tyto účely spouští aplikace (pomocí scheduleru) každou minutu cyklus, který volá updater a po skončení práce se na krátký čas uspí. Aby ale systém udržoval dotazování pravidelné, je mechanismus implementován tak, aby doba uspání činnosti reflektovala čas provedení operací updaterem. Tím jsme schopni nastavit přesný časový interval mezi voláními. Následující pseudokód vysvětluje proces, který je každou minutu spuštěn.

```
updateTime = 3 // čas mezi voláními v sekundách
callsPerMinute = 60 / updateTime

for (i < callsPerMinute) {
    start = microtime(now)
    updater->update()
    duration = microtime(now) - start

    if (duration < updateTime && i < callsPerMinute - 1) {
        usleep((updateTime - duration) * 1000000)
    }
}
```

[7]Soubor obsahující příkazy a časy, kdy má cron tyto příkazy provést.

Uspání na konci cyklu je spuštěno pouze za předpokladu, že:

1. doba vykonání updatu nepřesáhla interval mezi voláními,
2. se nejedná o poslední volání v cyklu - nemá smysl uspávat vlákno, pokud už nic vykonávat nebude.

Při takovémto řešení se vystavujeme riziku, že bude proces updatu trvat déle, než interval mezi voláními. Tím pádem by aplikace další volání spouštěla okamžitě po dokončení předchozího, což by vedlo ke zbytečnému volání několika *okamžitých* updatů v řadě. V horším případě by se nestihla všechna volání provést před iniciací dalšího volání schedulerem (při začátku další minuty) a všechna následná volání by se odsouvala na později. Pro mitigaci tohoto rizika se zaměřme na povahu prováděných operací v průběhu updatu. Z hlediska doby běhu jednotlivých částí procesu je časově nejnespolehlivějším článkem této operace *komunikace* mezi naší webovou aplikací a lokalizačním serverem, která závisí na úspěšném doručení požadavku a obdržením odpovědi; a délce odezvy serveru. Tento problém za nás řeší HTTP klient (Guzzle PHP HTTP), který umožňuje nastavení časového limitu pro čekání na odpověď. Jedná se o ošetření výjimečných případů, průměrná doba odezvy serveru při opětovném dotazování činí přibližně 50 milisekund[8].

Dále je nutné zajistit, aby systém dokázal upozornit, pokud v tomto updatovacím procesu nastane chyba, která zapříčiní že se aplikaci nepodaří aktualizovat polohy pacientů. Do scheduleru byl implementován blok pro odchytávání výjimek, který zajišťuje, že v případě selhání updatu poloh pacientů je vyhozena výjimka do `.log` souboru. Pro případ opakující se chyby je vhodné v takovémto místě informovat správce systému, případně samotného uživatele, že updatovací systém selhává a je třeba takový problém řešit. Vzhledem k 3 sekundovém intervalu aktualizace poloh, se nabízí po 10 neúspěšných pokusech (= půl minuty) zaslat upozorňující email, případně sms administrátorovi systému. Lze očekávat, že nasazení systému do testovacího ukáže jaké nastavení těchto parametrů a metod oznámení poruchy bude nejvhodnější.

[8] Průměrná doba odezvy při 1000 dotazech v řadě.

2.3 Komunikace s náramkem

Veškeré informace potřebné k lokalizaci pacienta zařizuje náramek lokalizovaný SPoT serverem. Za účelem umožnění přivolání pomoci musí ale náramky komunikovat přímo s webovou aplikací. Zároveň se nabízí navrhnout komunikační rozhraní tak, aby bylo možné přidat funkcionalitu pro posílání i dalších informací kromě volání o pomoc, jako např. tělesnou teplotu nebo srdeční tep, které mohou zdravotním pracovníkům umožnit předcházení nebezpečných zdravotních situací pacientů.

2.3.1 Prototyp rozhraní v BP

V bakalářské práci jsme připravili *Controller*, který přijímal POST požadavek na domluvené adrese. Náramek zasílal jako součást požadavku svou mac adresu, díky které systém dohledal propojeného pacienta a vyhodnotil proces jako volání o pomoc. Přidání náramku zahrnovalo nutnost manuálního zadání do systému přes webové rozhraní včetně vyplnění mac adresy, kterou jsme nejprve museli zjistit libovolným způsobem. Zároveň nebylo řešeno zabezpečení, ani možnost volání jiných akcí, než přivolání pomoci.

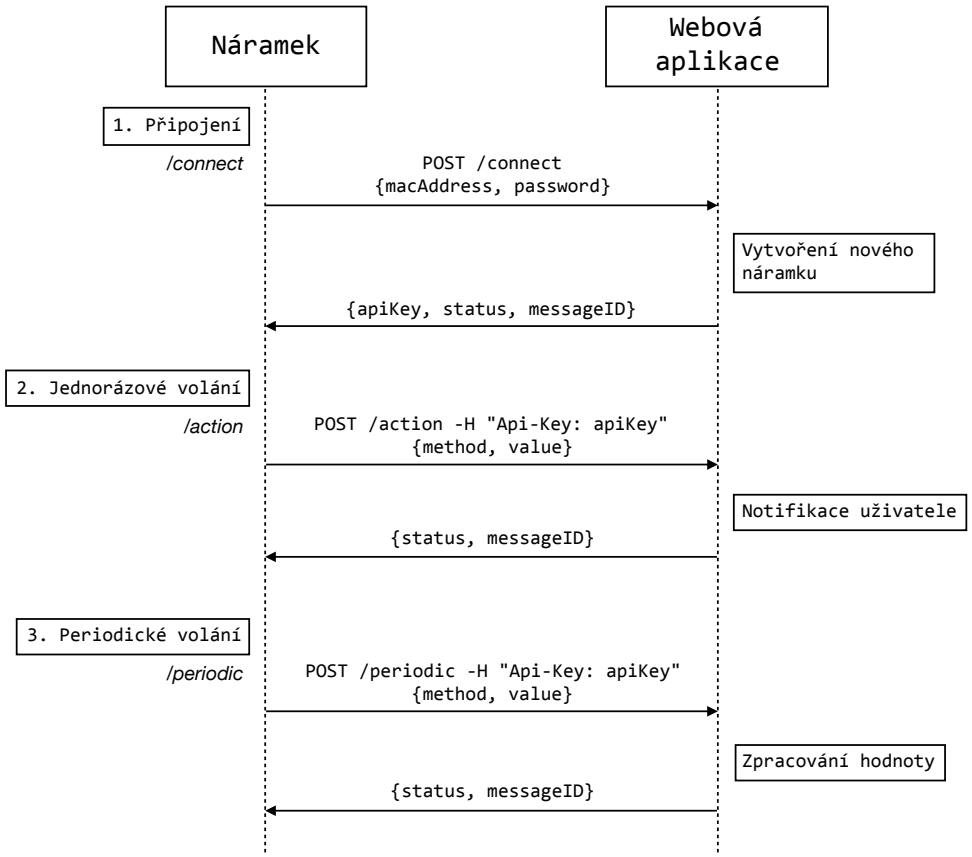
2.3.2 Web API pro náramky

Na základě funkčních požadavků z analýzy, posyktuje webová aplikace webové API na adrese `/api/v1/wristband-communication`, s metodami:

`/connect` - připojení náramku do systému,

`/action` - předání informace, že bylo stisknuto tlačítko volání o pomoc,

`/periodic` - předání informace z periodicky se hlásícího čidla na náramku.



Obrázek 2.6: Metody komunikace webové aplikace s náramkem

1. krokem je nutná autentizace náramku. Jako odpověď na požadavek na adresu /connect obdrží náramek API klíč, kterým se bude identifikovat v následné komunikaci. 2. krok popisuje jednorázové volání, které vyvolá náramek při stisknutí tlačítka. Na subadrese /action, bude náramek zasílat *metodu* a její *hodnotu*. V této práci řešíme pouze volání o pomoc, ale zadavatel zmínil, že chce mít možnost přidat další tlačítka, což takto definovaným rozhraním umožníme. Stačí definovat novou metodu či hodnotu, kterou bude systém vyhodnocovat novým způsobem (např. tlačítko pro zrušení volání o pomoc by volalo stejnou metodu s jinou/opačnou hodnotou). 2. a 3. krok může náramek opakovat v libovolném pořadí, dokud nebude odebrán ze systému. Poté bude muset opět zažádat o klíč.

2.3.2.1 Připojení náramku do systému

Před tím, než náramek poprvé začne komunikovat s webovou aplikací, musí se do systému zaregistrovat. K tomu slouží adresa /connect, na kterou náramek

zašle HTTP POST požadavek s 2 povinnými parametry:

1. *mac adresou* - systém potřebuje adresu znát kvůli budoucímu spárování náramku s lokalizačními daty ze SPoT serveru, který reprezentuje lokalizovaná zařízení pomocí jejich mac adres,
2. *počátečním heslem* - pro účely základního zabezpečení je v kódu aplikace nastaveno počáteční heslo zamezující libovolnému jinému zařízení v předstírání připojovaného náramku.

Oba parametry pošle náramek v těle požadavku jako application/json data, viz vzorový požadavek:

```
POST /api/v1/wristband-communication/connect HTTP/1.1
Host: prowilos.local
Content-Type: application/json
...
{
    "password": "wfoije67jo82",
    "macAddress": "A1:B2:C3:D4:E5:F6"
}
```

Webová aplikace uloží přijatou zprávu do databáze, ověří počáteční heslo, vytvoří nový záznam náramku a vygeneruje API klíč. Jako odpověď vrátí webová aplikace kromě api klíče také *status* vyřízení požadavku, zpravidla `success` v případě úspěchu a `fail` s příslušnou chybovou hláškou v případě selhání. Pro účely přehlednějšího řešení případných potíží v komunikaci odpověď obsahuje i *identifikátor zprávy*, pod kterým uložila webová aplikace požadavek do databáze, viz vzorová odpověď:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "status": "success",
    "messageId": 14,
    "apiKey": "Wf1oH7KFG3NBI4uSPACULqrcFCTkaWGFhaF151BN"
}
```

2.3.2.2 Volání o pomoc

Nejdůležitější částí této komunikace je předání volání o pomoc. Na adrese /action přijímá webová aplikace požadavky vyvolané stisknutím tlačítka na náramku. HTTP požadavek typu POST musí obsahovat:

1. *API klíč* - získaný z předchozího připojení,
2. *metodu* - specifikující typ volané akce,
3. *hodnotu* - specifikující hodnotu zvolené metody.

API klíč je třeba specifikovat v hlavičce požadavku, 2. a 3. parametr očekává aplikace v těle požadavku jako `application/json` data, viz vzorový požadavek:

```
POST /api/v1/wristband-communication/action HTTP/1.1
Host: prowilos.local
Content-Type: application/json
Api-Key: 9zhKuF8L72tqyndqipu7NCVo4feqokCypKMbhLXC
...
{
    "method":"callingForHelp",
    "value":true
}
```

a vzorová odpověď:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "status": "success",
    "messageId": 12
}
```

2.3.2.3 Periodické volání a návrh funkce heartbeat

Jelikož zadavatel nespecifikoval žádné konkrétní požadavky na využití této komunikace, je naším cílem především navrhnout způsob předávání takovéto informace. Způsob komunikace pro předání jednorázové akce byl záměrně navržen tak, aby byl použitelný i pro účely periodických volání. Finální verze může být tedy identická.

Do oddělených adres jsou typy komunikace rozděleny kvůli separovanosti kódu z pohledu na jejich využití. Nabízí se totiž možnost v budoucnu přizpůsobit tuto funkcionalitu k monitoringu činnosti náramků, kde bude od periodických volání, na rozdíl od jednorázových, očekávána jistá úroveň pravidelnosti. Jedním z klíčových využití periodického volání náramku může být například *heartbeat* indikující aktivitu náramku. Náramek může posílat v daném časovém intervalu jednoduchou zprávu *alive*, případně *stav baterie*, který

by zároveň umožnil předejít vybití baterie za provozu. Ale hlavně bude systém na základě nepříchozích zpráv schopen vyhodnotit, že náramek selhal a je třeba ho restartovat, opravit, či vyměnit. Systém by tak dokázal včas odhalit nefunkční/nekomunikující náramky, upozornit pečovatele a tím předejít nebezpečí, že si pacient nebude moci přivolat pomoc.

2.3.2.4 Implementované komponenty

K reprezentaci jednotlivých URL v API využívá aplikace standardního routovacího systému Laravelu, tzn. že každá adresa je propojena s odpovídajícím *Controllerem* a jeho příslušnou metodou.

```
Route::post(
    '/wristband-communication/connect', '\\' .
    WristbandCommunicationController::class . '@connect'
);
```

Hlavní roli v této komunikaci hraje `WristbandCommunicationController`, který obsahuje metody `connect()`, `action()` a `peridic()`. Ke zpracování požadavků využivá controller několik pomocných tříd.

`MessageSaver` - ukládá do databáze příchozí zprávy od náramků. Každý záznam má unikátní ID, čas příchodu, api klíč zařízení, volanou metodu a posíláná data. Ukládání zpráv provádí aplikace především kvůli možnosti debugování případně nefunkční komunikace.

`WristbandActionHandler` - řeší logiku jednorázových požadavků (volání o pomoc). Je připraven tak, aby bylo možné přidávat další typy volání.

`WristbandPeriodicHandler` - obdobný jako ActionHandler, ale pro periodické požadavky.

`ResponseComposer` - připravuje odesílanou odpověď. Přidává do odpovědi id uložené příchozí zprávy, status vykonání požadavku a případné chybové hlášky.

Smyslem těchto pomocných tříd je především udržování přehlednějšího kódu v controlleru v souladu s metodikou co nejvyššího oddělení zodpovědnosti (Separation of Concerns [14]).

2.3.2.5 Ukládání API klíče

Jelikož náramek při prvním připojení klíč obdrží a autentizuje se jím po celou dobu své komunikace až do smazání, nemá pro toto použití smysl vytvářet samostatnou tabulku udržující vazby mezi klíči a náramky. API klíč je tedy

uložen přímo v záznamu náramku. Přidání atributu `api_key` do již existující databázové tabulky `wristbands` v Laravelu usnadňuje mechanismus migrací, který umožňuje přidání atributu do již existující databáze entity bez nutnosti přesunu/mazání uložených dat. Stejným postupem lze migraci stornovat a vrátit tak systém do stavu před jejím provedení. Svým způsobem se jedná o *verzování* databáze, které je především užitečné při práci s již nasazeným systémem, kdy si nemůžeme dovolit přemazání již uložených dat a zároveň potřebujeme provést potřebné úpravy [11].

2.4 Komunikace s mobilní aplikací

Další požadovanou částí systému je komunikace s mobilní aplikací. Jejím účelem je umožnit pečovatelům mít přehled nad nejdůležitějšími informacemi souvisejícími s lokalizací, administrační stránka je ponechána webovému rozhraní aplikace. Zpracování samotné mobilní aplikace není náplní této diplomové práce, avšak její implementace přímo závisí na definici komunikačního rozhraní s webovou aplikací.

2.4.1 Prototyp rozhraní v BP

V analýze jsme rozebrali stav implementovaných části této komunikace. V bakalářské práci byl zhotoven prototyp, jehož cílem bylo ověřit možnost takovýmto způsobem komunikovat s mobilní aplikací. Jelikož systém ale nepracoval s průběžně aktualizovanými lokalizačními údaji, nebyl kladen důraz na důkladné testování správnosti výpočtů při změnách lokací pacientů. Nedostatky se projevily při testováních souvisejících s implementovaným časovým plánovačem (schedulerem) v předešlé sekci. Systém nesprávně vyhodnocoval změnu polohy a výsledkem bylo, že na některé dotazy odpovídal posíláním lokací všech pacientů, čímž by mechanismus výpočtu změn za tímto účelem postrádal smysl. Dále nebylo v prototypu řešeno rozhraní pro předání informace o vyřešení volání o pomoc. V mobilní aplikaci, stejně jako ve webovém rozhraní, bude mít pečovatel možnost vyřešit volání o pomoc tlačítkem. V prototypu také nebyla řešena autentizace uživatele mobilní aplikace. Tato sekce se věnuje opravě mechanismu pro výpočet změn, poskytnutí rozhraní pro předání informace o vyřešení volání o pomoc a doplnění autentizace uživatelského účtu v mobilní aplikaci.

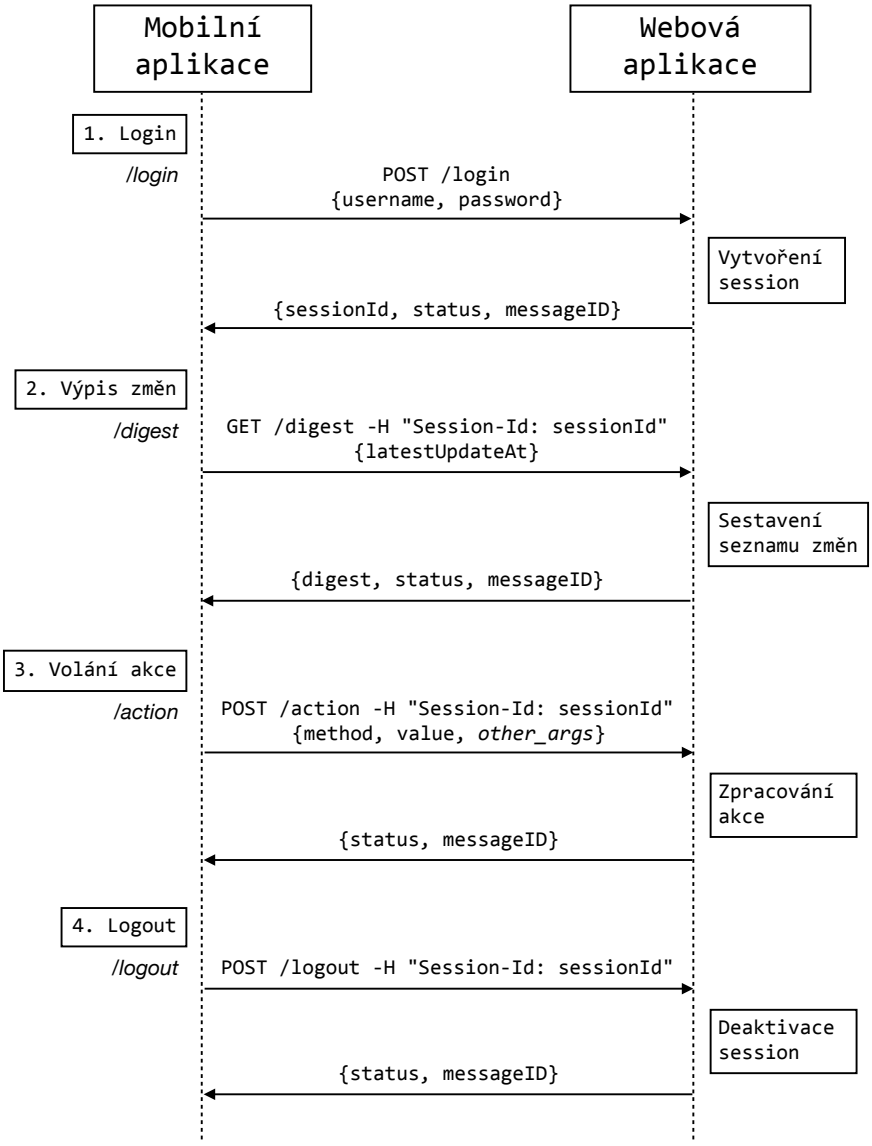
2.4.2 Web API pro mobilní aplikace

Za účelem komunikace s mobilní aplikací poskytuje webová aplikace API na adrese `/api/v1/mobile-device-communication` s následujícími metodami:

/login, /logout - přihlášení a odhlášení uživatele v mobilní aplikaci;

/digest - získání strukturovaného seznamu změn lokací, údajích o pacientech a volání o pomoc;

/action - provedení specifické akce - v tomto případě vyřešení volání o pomoc.



Obrázek 2.7: Metody komunikace webové aplikace s mobilní aplikace

1. krok - přihlášení, je nezbytným požadavkem pro mobilní aplikace. Bez zaslání přihlašovacích údajů na adresu /login a úspěšného ověření uživatel-

ského účtu nebude mít aplikace oprávnění k získávání informací z webové aplikace. Odpovědí serveru je sessionId, které musí aplikace používat k autentizaci při každém dalším požadavku. 2. krok je nejdůležitější částí této komunikace. Pomocí požadavku na adrese /digest dostává aplikace veškeré nové informace o pacientech. Odpověď má tvar změn od posledního dotazu do současnosti, je tedy nutné uvést v požadavku čas posledního updatu. 3. krok umožňuje např. vyřešit volání o pomoc. Obdobně jako u náramku je tato adresa ale přizpůsobena na rozšíření o další metody. 4. krokem je odhlášení uživatele z mobilní aplikace. Při zaslání požadavku na adresu /logout je deaktivována session a tím i zrušen přístup aplikace do komunikačního rozhraní.

2. a 3. krok může aplikace volat v libovolném pořadí a opakovaně podle vlastních potřeb. Lze ale předpokládat, že výpis změn bude aplikace volat periodicky a řešení volání o pomoc pouze v případě potřeby.

Součástí rozhraní je i adresa `/patients` z prototypu komunikace vytvořeného v bakalářské práci, poskytující výpis všech pacientů. Používání této metody je v současné implementaci není nutné, neboť její funkcionalitu plně zastupuje `/digest`.

2.4.2.1 Autentizace uživatele

Přihlášení do mobilní aplikace je řešeno pomocí stejného uživatelského účtu jako do webového rozhraní. Vzhledem k omezeným funkcím mobilní aplikace webová aplikace neřeší rozdíl v rolích uživatele, administrátorský účet zde má stejná práva jako prostý uživatel.

Před zahájením komunikace musí mobilní aplikace zaslat HTTP POST požadavek na adresu /login. V těle musí požadavek obsahovat parametry `username` a `password` ve formátu `application/json`, viz vzorový požadavek:

```
POST /api/v1/mobile-device-communication/login HTTP/1.1
Host: prowilos.local
Content-Type: application/json
...
{
    "username": "uzivatel",
    "password": "heslo"
}
```

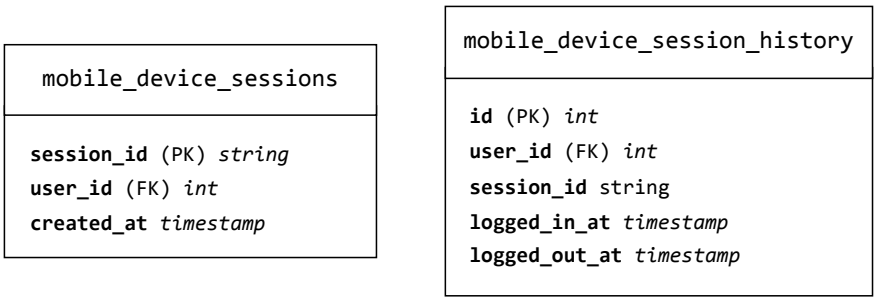
Webová aplikace v případě úspěšného ověření přihlašovacích údajů vygeneruje `session_id`. Společně s `user_id` a časovým razítkem `created_at` uloží záznam do tabulky `mobile_device_sessions`, ve které záznam setrvá do odhlášení. Zde se nabízí standardní nastavení *timeoutu* pro vypršení platnosti session, tudíž i automatického odhlášení uživatele. Jelikož je pro nás ale důležité, aby nebyl uživatel odhlášen v průběhu pracovní doby, kterou neumíme

předem určit, necháme tuto funkcionalitu stranou, dokud se neprokáže, že je opravdu potřeba.

Odpovědí webové aplikace je zpráva obsahující nově vytvořené session id, status a id zprávy se stejnými vlastnostmi jako v komunikaci s náramky, viz vzorová opdověď:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "status": "success",
    "messageId": 14,
    "sessionId": "HL6gaLNiYKVW028L90VFVZzzb4vC2kv6KB9voQ4w"
}
```

Pro odhlášení na adrese `/logout` nevyžaduje v těle požadavek žádná data, pouze v hlavičce musí být specifikované `sessionId`. Struktura poslané zprávy je odvoditelná z metod popsaných níže. Při odhlášení smaže webová aplikace session z databáze a zapíše záznam do historie mobilních přihlášení, viz následující schémata:



Obrázek 2.8: DB schémata aktivních a archivovaných session

2.4.2.2 Výpis posledních změn

K získávání veškerých potřebných aktualit poskytuje aplikace metodu `/digest`, která vrací strukturovaný seznam posledních změn. Součástí HTTP *GET* požadavku na tuto adresu musí být *timestamp* `latestUpdateAt`, který specifikuje čas posledního předchozího dotazu aplikace. Při prvním požadavku je nutné, aby aplikace zasílala timestamp s hodnotou 0. Důvodem pro neukládání tohoto časového údaje do session, namísto zasílání s každým požadavkem je ošetření případu, kdy se odpověď nedoručí mobilní aplikaci. V takovém případě by mohlo dojít ke ztrátě dat, webová aplikace by totiž aktualizovala čas

posledních nahlášených změn, zatímco mobilní aplikace by se chovala jako by žádná změna nenastala.

K zjištění změn využívá aplikace timestampů u uložených záznamů. Laravel umožňuje využití metody `timestamps()`, která se stará o ukládání časů *vytvoření* a *modifikace* databázových záznamů. Při tvorbě listu změn, které se mají poslat v odpovědi *digestu*, prochází aplikace databázi posledních známých lokací a v případě, že je timestamp úpravy časově novější, než zaslaná hodnota `latestUpdateAt`, přidá záznam do seznamu změn. Stejným způsobem prochází aplikace změny u pacientů, včetně jejich vytvoření nebo smazání.

Výpis změn (digest) se skládá z:

1. `latestChangeAt` - časový údaj poslední změny, který použije mobilní aplikace v příštím dotaze, jako již popsaný parametr `latestUpdateAt`;
2. `locationChanges` - lokalizační změny pacientů;
3. `patientChanges` - změny osobních údajů pacientů, rozdělené na nové, upravené a smazané záznamy;
4. `callingForHelp` - `id` pacientů volajících o pomoc.

Vzorový požadavek mobilní aplikace:

```
GET /api/v1/mobile-device-communication/digest HTTP/1.1
Host: prowilos.local
Content-Type: application/json
Session-Id: HL6gaLNiYKVW028L90VFVZzzb4vC2kv6KB9voQ4w
...
{
    "latestUpdateAt":"2018-09-13 11:21:13"
}
```

Vzorová odpověď webové aplikace:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "status":"success",
    "messageId":16,
    "digest":{
        "latestChangeAt":"2019-01-08 03:25:06",
        "locationChanges":{
            "1":{
                "x":220,
```

```
                "y":498
            },
            "23":{
                "x":465,
                "y":-233
            }
        },
        "patientChanges":{
            "created":[
                "25":{
                    "firstName":"Oldřich",
                    "lastName":"Nový"
                }
            ],
            "updated":[
                "1":{
                    "firstName":"Ladislav",
                    "lastName":"Hruška"
                }
            ],
            "deleted":[]
        },
        "callingForHelp":[
            2
        ]
    }
}
```

2.4.2.3 Řešení volání o pomoc

Posledním požadavkem na rozhraní je vyřízení volání o pomoc. Obdobně jako u náramku, na adrese `/action` poskytuje aplikace službu vyžadující HTTP POST požadavek s parametry:

1. `method` - specifikující typ volané akce,
2. `value` - specifikující hodnotu zvolené metody,
3. `patientId` - id pacienta, kterému byla poskytnuta pomoc.

Tato metoda je opět definována obecně pro umožnění budoucího rozšíření o další funkcionality. Zasílání hodnoty u metody *volání o pomoc* je zachováno kvůli možnosti obrátit proces. Např. standardně bude mobilní aplikace zasílat metodu `solveCallingForHelp` s `value = true`, ale v případě implementace

funkce "Vzít zpět"(z důvodu použitelnosti uživatelského rozhraní) může mobilní aplikace zasílat opačnou hodnotu `value = false` a tím vrátit zpět stav volání o pomoc. Hodnota parametru `value` samozřejmě nemusí být binární, záleží na potřebách metody.

Vzorový požadavek:

```
POST /api/v1/mobile-device-communication/action HTTP/1.1
Host: prowilos.local
Content-Type: application/json
Session-Id: HL6gaLNiYKVW028L90VFVZzzb4vC2kv6KB9voQ4w
...
{
    "method":"solveCallingForHelp",
    "value":true,
    "patientId":23
}
```

Vzorová odpověď:

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "status": "success",
    "messageId": 19
}
```

2.4.2.4 Implementované komponenty

Obdobně jako v komunikaci s náramkem hraje ústřední roli tohoto rozhraní `MobileDeviceCommunicationController`, s metodami `login()`, `logout()`, `digest()` a `action()` volanými při požadavku na příslušných adresách, využívající standardního routování.

K ukládání zpráv (požadavků) do databáze využívá controller již existující pomocná třída `MessageSaver`, stejně tak pro vytváření odpovědí je znovu použita třída `ResponseComposer`.

Řešení požadavku typu *action* je obdobně jako v komunikaci s náramky separováno do pomocné třídy `MobileDeviceActionHandler`. Na základě názvu volané meotdy se provede kód řešící příslušný problém.

Tvorbu *digestu* (výpisu změn) zajišťuje třída `DigestAssembler`, jejímž jediným vstupním parametrem je čas poslední známé změny `latestUpdateAt`. Třída obsahuje metodu `assemble()`, která volá privátní metody pro tvorbu všech potřebných částí výpisu:

`getLatestChangeAt()` - získá nejnovější čas úpravy mezi změnami lokací, pacientů a volání o pomoc;

`getLocationChanges()` - získá všechny lokace jejichž změna nastala po uvedeném čase poslední známé změny;

`getPatientChanges()` - získá všechny změny mezi údaji pacientů, jejichž změna nastala po uvedeném čase poslední známé změny;

`getCallingForHelp()` - získá všechna id pacientů, kteří volají o pomoc.

Takto implementovaná třída umožňuje jednoduché přidání dalších položek do výpisu. Návratovou hodnotou funkce `assemble()` je objekt `Digest`, který je v Laravelu při poslání v odpovědi automaticky reprezentován ve formátu *json*.

2.5 Doplnění uživatelského rozhraní

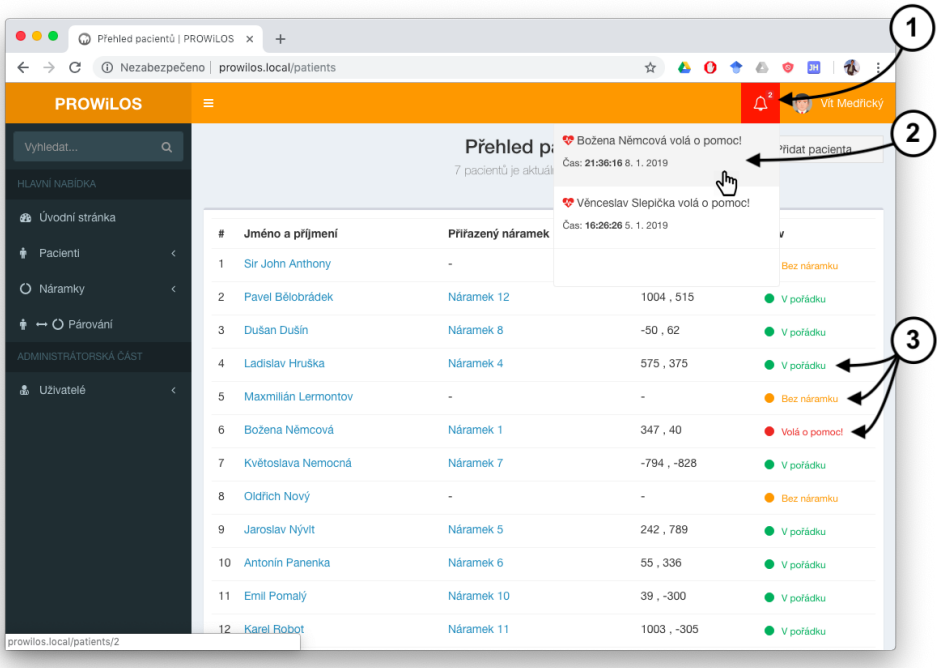
Prototyp uživatelského rozhraní z bakalářské práce byl vyhotoven tak, aby byly umožněny úpravy záznamů pacientů, uživatelů a náramků, párování pacienta s náramkem. Úkolem této sekce je navrhnout a implementovat chybějící funkcionality uživatelského rozhraní a vylepšit *user experience*[9] aplikace na základě funkčních požadavků stanovených v analýze.

2.5.1 Notifikace a vyřešení volání o pomoc

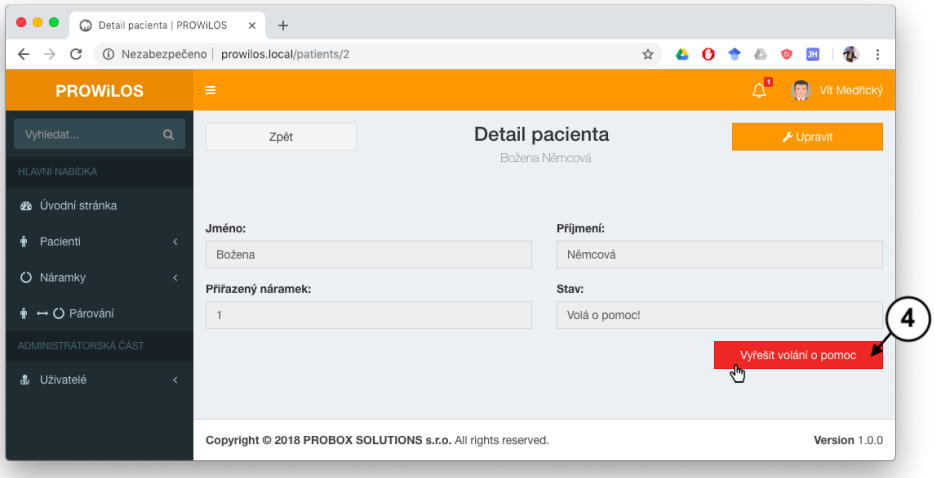
Podstatnou funkcionalitou systému je poskytování možnosti zavolat si pomoc. Je důležité zajistit, aby se informace dostala k pečovatelům. Primárním nástrojem pro obdržení notifikace by pro pečovatele měla být mobilní aplikace, kterou má vždy nablízku. Nicméně webové rozhraní je jedním z přístupových bodů systému, ve kterém by taková informace neměla být přehlédnuta.

Kromě stránky s přehledem pacientů, na které je vidět *stav* všech pacientů, poskytuje webová aplikace notifikační centrum pro důležitá oznámení. Standardním přístupem při vývoji webové stránky je poskytnutí ikonky zobrazující upozornění, dostupné, pokud je to možné, ve všech částech systému. Jelikož webové rozhraní obsahuje lištu fixního záhlaví (`header`) je notifikační centrum umístěno do něj.

[9] UX - jaký dojem nebo prožitek zanechá uživatelské rozhraní na uživateli, jedná se o důležitý aspekt celkové použitelnosti aplikace.



Obrázek 2.9: Notifikace volání o pomoc



Obrázek 2.10: Vyřešení volání o pomoc

Volání o pomoc, které obdrží webová aplikace od náramku, je zobrazeno uživateli pomocí ikonky zvonku, s číselným indexem počtu notifikací. Ikona bliká červeně, dokud není notifikace zobrazena (1).

Po rozkliknutí notifikací je zobrazen seznam aktuálních volání o pomoc (2). Každé oznámení je zároveň odkazem na detail pacienta, kde lze volání vyřešit tlačítkem `Vyřešit volání o pomoc` (4).

Dále byla upravena tabulka s přehledem, která nyní zobrazuje aktuální *stav* pacienta (3). Implementovány byly 3 stupně, se zbarvením dle důležitosti. Zelená barva (*V pořádku*) signalizuje, že pacient má přiřazen náramek a nevolá o pomoc. Oranžová barva (*Bez náramku*) znamená, že pacientovi zatím nebyl přidělen náramek. Na tento stav musí uživatelské rozhraní upozorňovat, protože pacient bez náramku není ani lokalizován, ani si nemůže přivolat pomoc. Červená barva (*Volá o pomoc*) signalizuje, že pacient stiskl tlačítko na náramku a vyžaduje asistenci pečovatele.

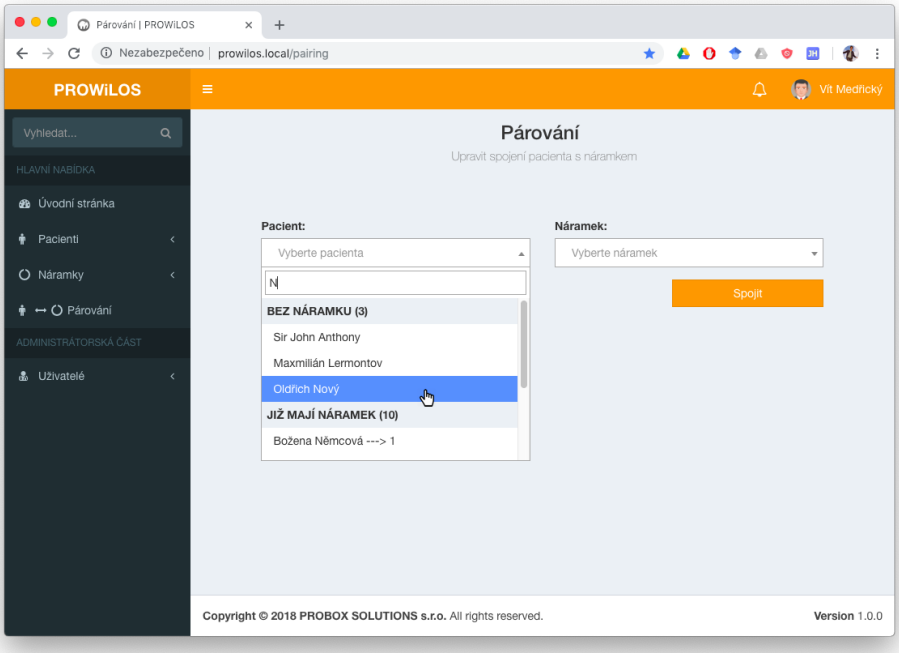
Obdobným způsobem jako přehled pacientů je řešena tabulka s přehledem náramků. Na nepřiřazený náramek ale systém neupozorňuje, stav je zobrazen šedou barvou (*Nepřiřazen*), protože na rozdíl od pacienta nám z takového stavu nevyplývá žádné nebezpečí. Naopak volný náramek lze přiřadit dalšímu pacientovi.

2.5.2 Úprava stránky s párováním

Webové rozhraní umožňuje přiřazení náramku k pacientovi. V prototypu byla k tomuto účelu vyhotovena samostatná stránka *párování*, na které lze vybrat pacienta i náramek a záznamy spárovat. Nedostatkem této stránky byla především absence nápovědy uživateli. Nebylo zřejmé, zda je náramek volný k přidělení, či zda pacient nemá už přidělen jiný náramek. Stránka umožňovala propojit libovolného pacienta s libovolným náramkem, aniž bychom se dozvěděli, že byl jinému pacientovi náramek odebrán.

Cílem uživatelského rozhraní na této stránce by mělo být předejití omylům, tedy lepší než vysvětlit uživateli, jaké chyby se dopustil, je navrhnout rozhraní tak, aby měl uživatel co nejvyšší pravděpodobnost se chybě vyhnout [15].

Za tímto účelem byla nabídka pacientů a náramků přepracována, aby zobrazovala v seřazeném seznamu nejprve nepřiřazené náramky a pacienty. V oddělené skupině nabídky pak lze vidět i přiřazené záznamy, vždy společně se spárovaným pacientem/náramkem, viz následující obrázek:



Obrázek 2.11: Párování s nápovědou

Implementace této nabídky využívá *select2* [16] dropdown menu s vyhledáváním. Tělo rozbaleného menu bylo rozděleno pomocí HTML tagů `<optgroup>` na 2 oddělené seznamy hodnot `<option>`. Data jsou předávána do blade šablony pomocí objektů `PatientSelectValues` a `WristbandSelectValues`, která obsahují oddělená pole hodnot se jmény volných a přiřazených náramku/pacientů.

Důležitým prvkem použitelnosti této stránky je také zpětná vazba provedených operací. Jelikož se tato problematika týká celého webového rozhraní, rozebereme ji v samostatné sekci.

2.5.3 Zpětná vazba aplikace

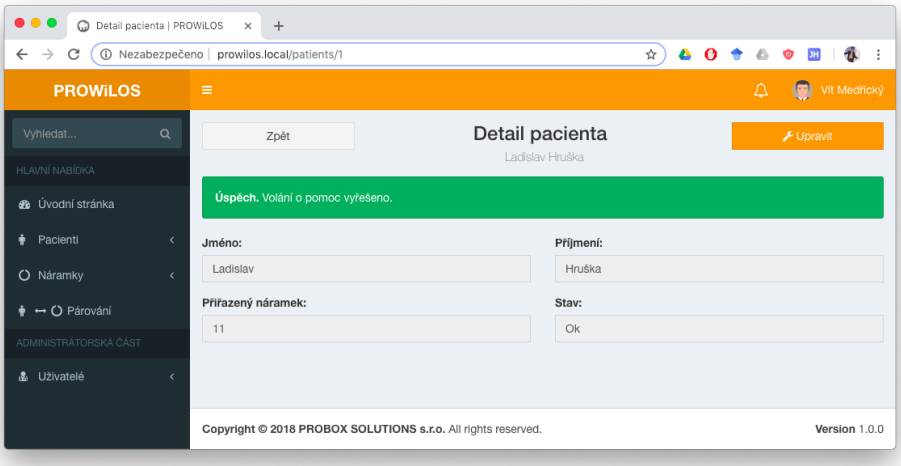
Zpětná vazba uživatelského rozhraní je důležitým prvkem použitelnosti. Uživatel by měl vědět, v jakém stavu se aplikace nachází [15]. Pokud jsou uživatelem provedeny změny, měla by aplikace poskytnout potvrzení provedeného úkonu, případně varování, či chybovou hlášku. To se týká veškerých operací, kdy uživatel upravuje data uložená v databázi. V tomto rozhraní se jedná o vyřešení volání o pomoc, párování náramku s pacientem, veškeré CRUD[10] operace nad entitami pacientů, náramků a uživatelů, včetně úpravy uživatelského profilu.

[10] **C**reate, **R**ead, **U**pdate, **D**elete - vytváření, zobrazení/čtení, úprava a smazání zdroje.

Webové rozhraní zobrazuje 3 typy zpráv:

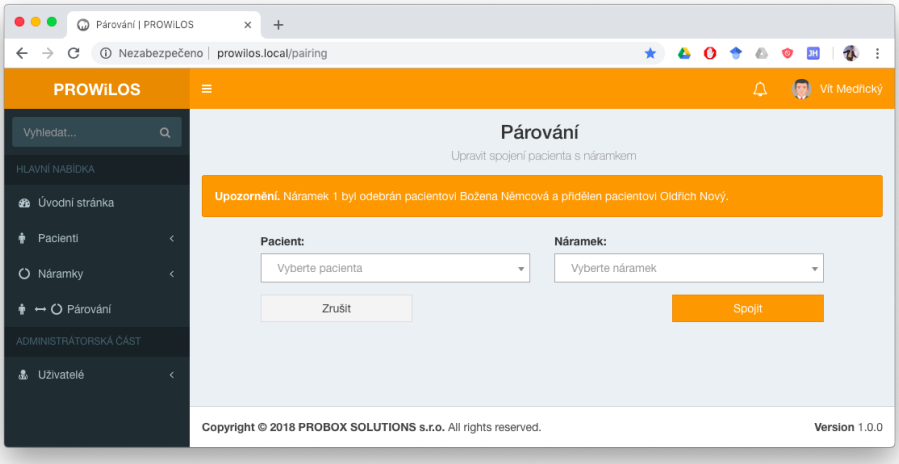
1. Úspěch (zelená zpráva) - potvrzení provedené úpravy (2.12);
2. Varování (oranžová zpráva) - potvrzení provedené úpravy s varováním, že se provedla operace, kterou uživatel nemusel mít v úmyslu (2.13);
3. Chyba (červená zpráva) - došlo k chybě, nejčastěji při špatném vyplnění formuláře. Těchto zpráv se při jedné operaci může zobrazit více dle počtu problémů, které nastaly (2.14).

K implementaci těchto zpráv je v aplikaci v příslušném controlleru definována zpráva uložením potřebných dat (*typ*, *hlavička* a *text* zprávy) do aktuální session. Třída `FlashMessagesViewComposer` vytvoří z dat uložených v session zprávu a zobrazí jí pomocí samostatného view na stránku. Struktura zprávy je definována ve vlastní blade šabloně `_flash-message.blade`[11].

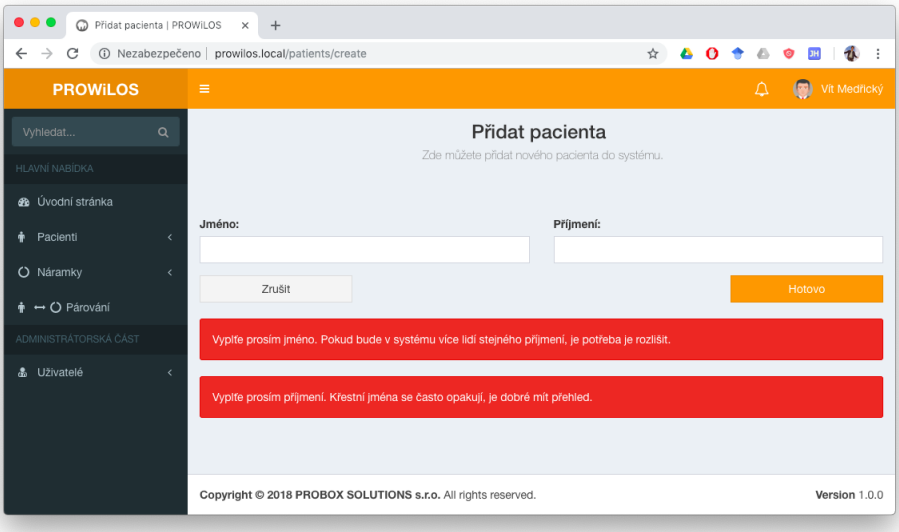


Obrázek 2.12: Flash message - Úspěch

[11] Znakem "_"před názvem jsou v projetku označovány views, které samostatně nevytváří vlastní stránku, jsou vkládány do jiných šablon - tzv. *partials*.



Obrázek 2.13: Flash message - Varování

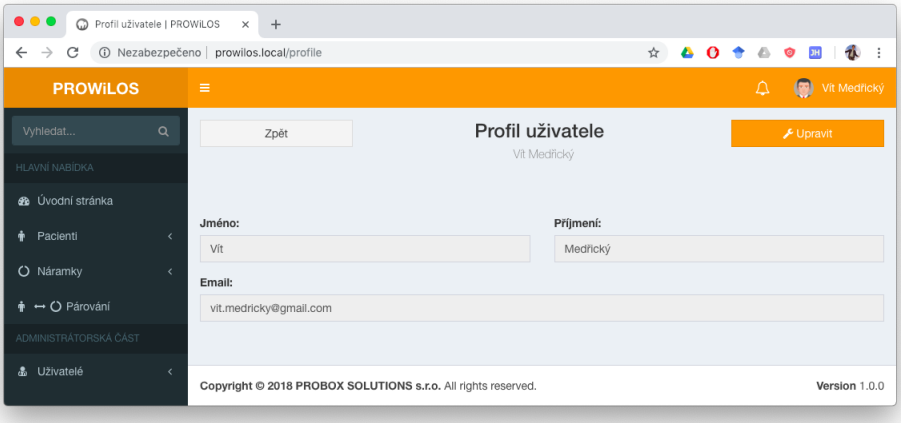


Obrázek 2.14: Flash message - Chyba

2.5.4 Profil uživatele

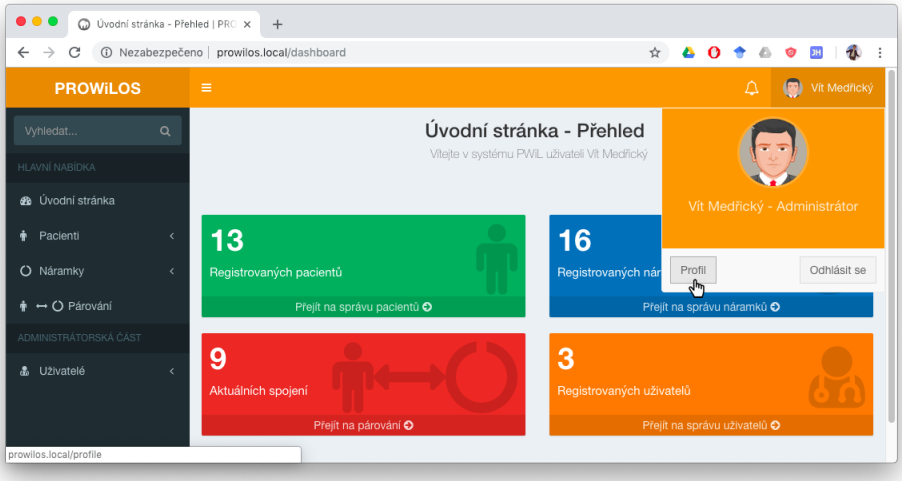
Poslední chybějící částí uživatelského rozhraní je úprava vlastního profilu. Z datového pohledu je stránka s profilem zobrazením detailu uživatele, které již byl implementován v prototypu, ale jako detail vybraného uživatele. Z autorizačního hlediska je zde rozdíl, v prototypu i současné verzi systému má přístup k uživatelským účtům pouze administrátor. Profil přihlášeného uživatele je tedy výjimkou, ve které aplikace umožní úpravu dat jiným uživatelským rolím, než administrátorovi.

Narozdíl od routování uživatelů, ke kterým lze přistupovat pomocí třídy `UserController` na adrese `/users/id` je profil uživatele přístupný pomocí `ProfileControlleru` na stránce `/profile`, čímž je jasně vymezen přístup uživatele pouze ke svému účtu.



Obrázek 2.15: Profil uživatele

V rozhraní byl přístup k profilu umístěn pod nabídku uživatele v záhlaví stránky, kde se nyní lze odhlásit nebo spravovat svůj účet. K implementaci byla využita nabídka uživatelského účtu z veřejně dostupné šablony *AdminLTE* [17], na jejímž základě stojí vzhled celého uživatelského rozhraní.



Obrázek 2.16: Nabídka uživatelského účtu

2.6 Další úpravy

V průběhu této práce bylo provedeno i několik drobnějších úprav, které vyplynuly z poznatků nabytých při implementaci nových součástí. Tyto úpravy nejsou pro finální produkt kritické a jejich řešení byla primitivní. Jedním z důležitějších bodů bylo odebrání funkce manuálního přidávání náramků pomocí uživatelského rozhraní. Díky novému způsobu přidávání náramků do systému pomocí automatického připojení přes API se manuální přidání náramku stalo redundantním a zbytečně složitým. Uživatel by musel zjišťovat mac adresu náramku a přepisovat ji do systému. Musela by také být přidána možnost nahrání api klíče do paměti náramku. Ani jeden z těchto postupů nebyl vyhodnocen jako žádoucí a stránka s přidáním náramku byla odstraněna. Také byla odebrána možnost měnit id náramku. Současná verze aplikace zobrazuje názvy náramků jako "Náramek č. *id_náramku*"a v případě, že nasazení ukáže potřebu uživatelů (pečovatelů) rozlišovat náramky jiným způsobem, může být v budoucnu přidán atribut *název náramku*. Toto řešení bude záležet na finální podobě náramků a jejich vizuálního rozlišení.

Kapitola 3

Testování

Testování je důležitou součástí vývoje softwarového produktu. Ověřuje správnou funkcionalitu zhotovených částí systému a pomáhá nalézt chyby [19]. V bakalářské práci jsme testovali především funkcionality uživatelského rozhraní a základní operace s databázovými entitami. V této práci jsme navázali na předchozí prototyp a tedy i z pohledu testování je výsledný produkt je nutno vnímat jako jednotný celek. Testy, které byly zhotoveny v předchozí práci bylo nejprve třeba upravit tak, aby odpovídaly aktuální podobě systému. Kromě hlavních implementačních celků, které jsme v této práci rozebrali, bylo totiž provedeno i mnoho drobných úprav, které způsobily, že aplikace neprošla některými z testů (např. test přidávání náramku pomocí uživatelského rozhraní, které jsme uvedli v předchozí kapitole).

Tato práce se zaměřovala především na komunikaci webové aplikace s lokalizačním serverem, náramky a mobilními aplikacemi. Proto musela být některá testování prováděna již v průběhu vývoje pro ověření funkcionality a správnosti návrhu. Jelikož veškeré navržené komunikační protokoly pracují s HTTP požadavky, byly vytvořeny pomocné skripty využívající příkazu `curl` [18], jejichž spuštěním bylo možné simulovat požadavky jednotlivých komunikačních stran. Webová aplikace tak reagovala na stejný požadavek, který by obdržela od reálného klienta (náramku/mobilní aplikace). Obdobným způsobem bylo možné testovat dotazy na SPoT API, jehož dokumentace tento postup přímo doporučovala.

3.1 Unit a feature testy

Dále bylo potřeba testovat interní mechanismy aplikace zastoupené řadou pomocných tříd. Za tímto účelem byly vytvořeny automatické *unit* testy prověřující funkcionalitu jednotlivých tříd a jejich metod [20]. K testování komunikací byly vytvořeny tzv. *feature* testy, které prověřují funkcionalitu větších funkčních celků [21]. Původní skripty užívající příkaz curl, které bylo nutné zadávat manuálně byly převedeny na automatizovanou formu a umožňují okamžitou

komplexní kontrolu funkčnosti. Kromě testování komunikací byla pomocí feature testů kontrolována i funkcionalita uživatelského rozhraní. Laravel nabízí funkce pro simulaci průchodu uživatelským rozhraním a lze tak testovat scénáře užití systému, ověřit funkcionality tlačítek, vyplnění formulářů, či související stav databáze, viz ukázka:

```
/** @test */
public function it_solves_calling_for_help_through_notif()
{
    ...
    $this->actingAs($user)
        ->visit('/')
        ->click('notifications-center')
        ->click($patient->id . '_calling-for-help-notif')
        ->press('Vyřešit volání o pomoc')
        ->see('Volání o pomoc vyřešeno')
        ->seeInDatabase('patients', [
            'id' => $patient->id,
            'is_calling_for_help' => false
        ]);
}
```

Obrázek 3.1: Ukázka testu průchodu uživatelským rozhraním

Rozdělení testovacích případů na unit a feature vyplývá z před připraveného testovacího prostředí v Laravelu, který již v základní instalaci poskytuje nástroj PHPUnit a umožňuje tak testování bez nutnosti instalace externího testovacího frameworku.

Výsledná aplikace byla testována 21 testovacími sadami (tzv. *test suites*, 12 sad unit testů, 9 sad feature testů) v celkovém rozsahu 71 testů, dohromady zahrnujících 375 asercí.

3.2 Testování komunikace s lokalizačním (SPoT) serverem

V bakalářské práci bylo testováno získávání lokací od pomocného stubu, který vracel vzorovou odpověď SPoT serveru. Tím byly aplikaci jednorázově předány poslední známé lokace několika vzorových náramků. V této práci již byly implementovány komponenty pro stažení lokací z reálného SPoT API, a tudíž bylo hlavním úkolem otestovat připojení k serveru, stažení lokací a následné spuštění průběžného updateru vyvolávající periodické opakování dotazů. Pomocí skriptů zasílajících HTTP požadavky prostřednictvím nástroje curl byla otestována komunikace s reálným SPoT serverem, který poskytl zadavatel. Dle

postupu uvedeného v online dokumentaci SPoT API [3] bylo nejprve zažádáno o api klíč potřebný pro následnou komunikaci. Tento postup je rozveden již v předchozí kapitole, jedná se o autentizační požadavek zasláním přihlašovacích údajů a naslédné obdržení api klíče.

S obdrženým api klíčem byl zaslán požadavek na poslední známé lokace, který proběhl v pořádku. V odpovědi server ovšem posílá pouze prázdné, viz ukázka:

```
curl -u *****:X http://*******
/api/v1/venues/vspot/locations/last\_known.json

HTTP/1.1 200 OK
< Content-Type: application/json
< Content-Length: 2
< Connection: keep-alive
< Status: 200 OK
< X-Content-Type-Options: nosniff
< Date: Wed, 09 Jan 2019 06:59:48 GMT
< X-Powered-By: Phusion Passenger 5.0.30
< Server: nginx/1.10.2 + Phusion Passenger 5.0.30
<
* Curl_http_done: called premature == 0
* Connection #0 to host ******** left intact

[]%
```

Prázdné pole v odpovědi je vráceno proto, že aktuální SPoT server zatím nelokalizuje žádné náramky. Za účelem otestování zpracování zpráv a průběžného updatingu byl proto vytvořen vlastní simulátor lokalizačního serveru.

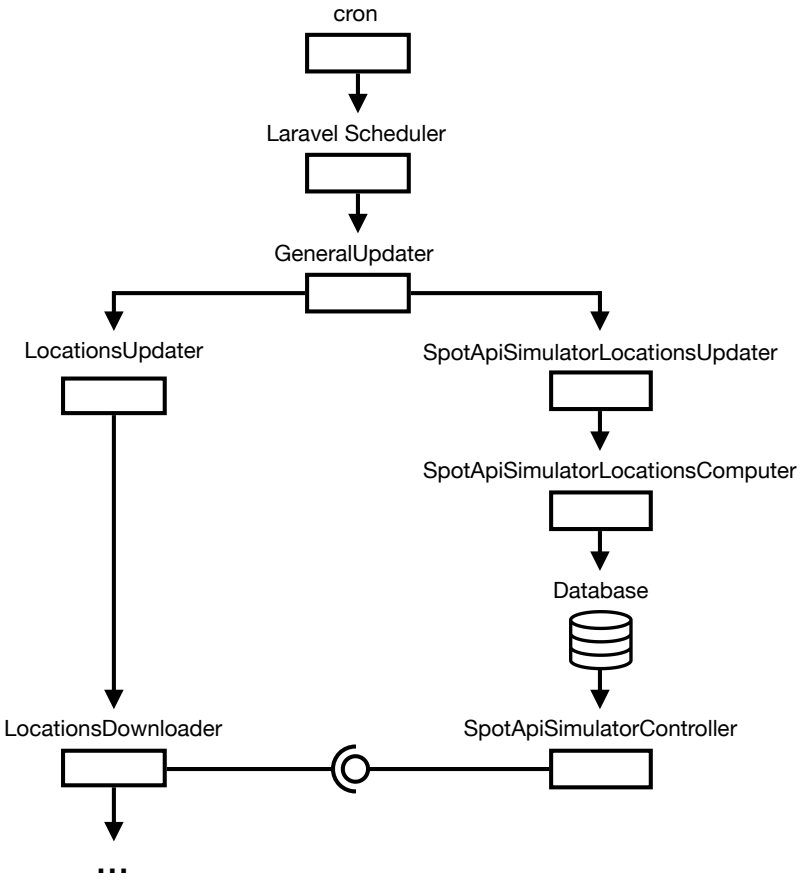
3.2.1 SPoT API simulátor

SPoT API simulátor generuje nové pozice náramků (simuluje pohyb pacientů s náramky) a odpovídá na dotazy stejným způsobem, jako reálný SPoT server (dle dokumentace).

Tuto funkcionalitu zastupuje třída `SpotApiSimulatorController`. Pro napojení updatovacího systému složeného z komponent, které jsme rozepsali v kapitole 2, obr. 2.4, stačí změnit adresu pro stahování ve třídě `Locations-Downloader` na adresu simulátoru:

```
/spot-api-simulator/venues/vspot/locations/last_known.json
```

Simulátor využívá ke generování nových lokací pomocnou třídu `SpotApiSimulatorLocationsComputer`. Pro automatické obnovování lokací (simulace měnících se lokací pacientů s náramky) je tato třída periodicky volána třídou `SpotApiSimulatorLocationsUpdater`, která je vyvolávána již dříve implementovaným systémovým updaterem.



Obrázek 3.2: Využití simulátoru SPoT API

Díky této pomocné komponentě jsme schopni simulovat práci systému s reálným serverem. Průběžné změny lokací jsou generovány tak, aby se podobaly reálným, pohybujícím se pacientům. Pro každý pohyb pacienta je vypočítána pravděpodobnost pohybu v závislosti na předchozím pohybu. V případě, že se pacient v předchozím kroku pohyboval, je pravděpodobnost jeho následujícího kroku výrazně vyšší, než v případě, že stál na místě. V případě generátor lokací vyhodnotí, že se má pacient pohybovat, následuje náhodné vygenerování pohybu po osách x, y, vždy je ale pohyb limitován maximální velikostí kroku. Tímto způsobem generujeme pohybující se pacienty, kteří jsou vždy po nějakou dobu v pohybu a na cílovém místě chvíli setrvají.

Napojení `LocationsDownloaderu` na simulátor nám umožňuje pozorovat přímo v uživatelském rozhraní průběžně se měnící lokace pohybujících se pacientů, což zároveň znamená, že data byla úspěšně uložena do databáze a jsou připravena na předání mobilní aplikaci. Pro komunikaci s reálným SPoT serverem bude třeba pouze změnit zdrojovou adresu pro zasílání požadavků.

3.3 Testování API pro náramky a mobilní aplikace

V průběhu vývoje API pro náramky a mobilní aplikace bylo nutné testovat jednotlivé požadavky pomocí příkazu cURL. Zasláním požadavku byla spuštěna konkrétní služba a bylo možné sledovat, zda server vrací očekávanou odpověď, nebo vyhazuje výjimku, případně error.

Na straně webové aplikace byl implementován systém zpětné vazby, který ukazuje, zda požadavek proběhl v pořádku, nebo nastal nějaký problém. Součástí `json` odpovědi webové aplikace je `status`, značící úspěch, nebo chybu. V případě chybného stavu obsahuje zpráva navíc `statusInfo`, specifikující typ problému (chybová hláška) Pro případné řešení potíží je navíc součástí každé odpovědi `messageId`, pomocí kterého lze dohledat zprávu, kterou aplikace od klienta obdržela.

Aby bylo možné tuto funkcionalitu hromadně testovat, byly vytvořeny feature testy komunikací. Pro každý typ klienta (náramek, mobilní zařízení) byla připravena testovací sada prověřující funkcionalitu každého z koncových bodů příslušného API.

U validních požadavků bylo testováno, zda byly provedeny požadované změny (stav databáze), zda byly vráceny požadované informace (api klíč, session id) a zda měla odpověď správný formát (digest, patients). U nevalidních požadavků byly testovány správné formáty odpovědí (statusů a chybových hlášek) dle konkrétní chyby v požadavku, případně zda systém neprovedl nežádoucí změny.

adresa	specifikace testu	status	statusInfo (chybová hláška)
connect	validní požadavek	success	-
	bez zaslaných dat	fail	No data sent.
	bez hesla	fail	Data | `"password"` not specified.
	špatné heslo	fail	Data | incorrect password.
	bez mac adresy	fail	Data | `"macAddress"` not specified.
action	validní požadavek	success	-
	bez api klíče	fail	Header | `"Api-Key"` not specified.
	nesprávný api klíč	fail	Header | invalid Api-Key.
	bez zaslaných dat	fail	No data sent.
	bez metody	fail	Data | `"method"` not specified.
	bez hodnoty	fail	Data | `"value"` not specified.
	nesprávná metoda	fail	Data | invalid method.

Tabulka 3.1: Testované chybové hlášky - API pro náramky

adresa	specifikace testu	status	statusInfo (chybová hláška)
login	validní požadavek	success	-
	bez zaslaných dat	fail	No data sent.
	nesprávné přihl. údaje	fail	Data | incorrect credentials.
logout	validní požadavek	success	-
	bez session id	fail	Header | `"Session-Id"` not specified.
	nepsrávné session id	fail	Header | invalid Session-Id.
action	validní požadavek	success	-
	bez session id	fail	Header | `"Session-Id"` not specified.
	nepsrávné session id	fail	Header | invalid Session-Id.
	bez zaslaných dat	fail	No data sent.
	bez id pacienta	fail	Data | `"patientId"` not specified.
	pacient neexistuje	fail	Data | patient does not exist.
	bez metody	fail	Data | `"method"` not specified.
	bez hodnoty	fail	Data | `"value"` not specified.
	nesprávná metoda	fail	Data | invalid method.
digest	validní požadavek	success	-
	bez session id	fail	Header | `"Session-Id"` not specified.
	nesprávné session id	fail	Header | invalid Session-Id.
	bez zaslaných dat	fail	No data sent.
	bez času posl. updatu	fail	Data | `"latestUpdateAt"` not specified.

Tabulka 3.2: Testované chybové hlášky - API pro mobilní aplikace

K odesílání požadavků byl využit PHP HTTP klient Guzzle 6 [6], který umožňuje jednoduchou specifikaci hlavičky i json dat posílaných v těle požadavku.

```
/** @test */
public function it_tries_to_send_action_with_invalid_method()
{
    ...
    $url = 'pwil.local/api/v1/wristband-communication/action';

    $response = $client->request('POST', $url, [
        'headers' => [
            'Accept' => 'application/json',
            'Content-Type' => 'application/json',
            'Api-Key' => $wristband->api_key
        ],
        'json' => [
            'method' => 'someInvalidMethod',
            'value' => true
        ]
    ]);

    // assertions
}
```

Obrázek 3.3: Ukázka testu zaslání požadavku

Závěr

Splnění zadání

1. **Za pomoci metodiky softwarového inženýrství analyzujte aktuální požadavky na systém a s pomocí výstupu z bakalářské práce „Webová aplikace PWiL - Systém pro lokalizaci pacientů“ navrhněte kroky vedoucí k splnění nových požadavků.**

   V úvodní kapitole byly analyzovány funkční a nefunkční požadavky, zmíněny splněné požadavky z předchozí práce a popsány kroky vedoucí ke splnění nově stanovených požadavků. Pro přiblížení problematiky byly rozepsány případy užití všemi aktéry v systému a zobrazena komplexní schémata předávání informací o lokacích, volání o pomoc a vyřešení volání o pomoc.

2. **Navrhněte a implementujte komunikační protokol mezi systémem PROWiLOS a lokalizačními zařízeními pro pacienty.**

   V druhé kapitole bylo navrženo a implementováno komunikační rozhraní mezi webovou aplikací a lokalizačními zařízeními pro pacienty (náramky). Na základě možností zvoleného hardwaru (Wemos D1 mini) bylo navrženo a implementováno webové API s oddělenými adresami pro zasílání příslušných požadavků náramků. Byla implementována autentizace náramku a následná možnost předání volání o pomoc.

3. **Navrhněte a implementujte komunikační protokol mezi systémem a mobilní aplikací.**

   Dalším bodem druhé kapitoly byl návrh implementace webového API pro mobilní aplikace. Byl zdokonalen návrh komunikačního protokolu z bakalářské práce, přidána autentizace uživatele, implementováno předávání výpisu posledních změn v lokacích, údajích pacientů a volání o pomoc; a přidána podpora řešení volání o pomoc z mobilní aplikace.

4. **Dle návrhu v prototypu realizujte komunikaci systému s lokalizačním serverem.**

   Pro komunikaci s lokalizačním serverem byl rozšířen návrh systému komponent pro stahování posledních známých lokací. Byl navržen a implementován mechanismus pro pravidelné vyvolávání updatu, a tím udržování databáze v aktualizovaném stavu. Byla vytvořena komponenta zasílající požadavky na příslušnou adresu SPoT API dle jeho dokumentace.

5. **Navrhněte a implementujte nezbytné chybějící části uživatelského rozhraní.**

   Uživatelské rozhraní webové aplikace bylo rozšířeno o zpětnou vazbu aplikace v podobě potvrzujících, varovných a chybových zpráv při interakci s databází. Bylo přidáno notifikační centrum pro upozornění uživatele, která nyní upozorňuje na volání o pomoc. Byla rozšířena funkcionalita stránky pro párování pacientů s náramky, která nyní zobrazuje nabídku nepřiřazených náramků a pacientů bez náramku. Tím byla zvýšena přehlednost nad přiřazovanými záznamy bez nutnosti přepínání obrazovek a současně snížena pravděpodobnost chybného spárování pacienta s náramkem. Byla přidána možnost úpravy vlastního uživatelského profilu a rozšířena stránka s přehledy pacientů a náramků zobrazující jejich stav.

6. **Zhotovenou aplikaci včetně jednotlivých komunikací podrobte vhodnému testování.**

   Ve třetí kapitole byly popsány mechanismy testování implementovaných části systému. Byly vytvořeny unit testy prověřující funkcionalitu zhotovených tříd a feature testy prověřující funkcionalitu API pro mobilní aplikace a náramky; získávání dat od SPoT serveru a průchody uživatelským rozhraním. Byly testovány odpovědi webové aplikace na validní a nevalidní požadavky klientů (náramků a mobilních aplikací) a ověřováno provedení požadovaných úprav v návaznosti na příslušné požadavky. Pro otestování periodického dotazování na poslední známé lokace a průběžné aktualizace systému byl implementován simulátor zastupující reálný lokalizační server v přesném formátu dle jeho dokumentace.

Výsledný stav a budoucnost projektu

Jak mobilní aplikace, tak softwarová část náramku byly vyvíjeny současně s tvorbou této práce. Princip jejich funkcionality byl ověřen již po dokončení bakalářské práce. Proběhlo prezentační demo, na kterém bylo zadavateli prezentováno předání volání o pomoc mezi náramkem (prototypem Wemos D1 mini) a webovou aplikací. Návrh dalších funkcionalit probíhal vždy na základě vzájemné komunikace s dalšími členy týmu, zodpovědnými za příslušnou část vývoje. Aktuální jádro systému (webová aplikace) je připraveno k nasazení do testovacího provozu, ovšem pro kompletně funkční lokalizační systém je nejprve potřeba dokončit implementace mobilní aplikace a softwaru pro náramek v návaznosti na nově přidané specifikace webového API. Podstatným krokem pro spuštění testovacího provozu je nasazení hardwaru pro lokalizaci (SPoT systému) a dokončení prototypu náramku. Jakmile budou tyto kroky splněny, lze začít s testováním systému v reálném prostředí pečovatelského domu.

Osobně tento projekt vnímám jako prospěšné využití informačních technologií v pečovatelství a doufám, že čas strávený vývojem tohoto systému přinese efektivní pomocný nástroj jak do rukou lidí, kteří kvalitní péči potřebují, tak i těch, kteří tuto péči vykonávají.

Literatura

[1] MEDŘICKÝ, Vít. Webová aplikace PWiL - Systém pro lokalizaci pacientů [online]. Praha, 2016 [cit. 2019-02-11]. Dostupné z: https://alfresco.fit.cvut.cz/share/proxy/alfresco/api/node/content/workspace/SpacesStore/5658f3dc-2d0c-4afd-959b-f8b6d773f329. Bakalářská práce. ČVUT v Praze - Fakulta informačních technologií.

[2] Ruckus [online]. Sunnyvale, CA 94089 USA: Ruckus Networks [cit. 2019-01-09]. Dostupné z: https://www.ruckuswireless.com/

[3] RUCKUS WIRELESS, INC. SPoT API Dokumentace. Ruckus API Explorer [online]. [cit. 2019-01-09]. Dostupné z: https://us-sys.ruckuslbs.com/api_explorer

[4] Wemos Electronics Wiki. Https://wiki.wemos.cc/ [online]. 2018/12/28 04:41 [cit. 2019-01-09]. Dostupné z: https://wiki.wemos.cc/products:d1:d1_mini

[5] OTWELL, Taylor. Laravel. Laravel.com [online]. [cit. 2019-01-09]. Dostupné z: https://laravel.com/

[6] BRADÁČ, Jan. *Mojeskolka.info – tvorba uzavřené komunitní sítě pro školky*, Praha, 2015. Diplomová práce. Str. 93-98. České vysoké učení technické v Praze. Fakulta informačních technologií. Vedoucí práce Jiří HUNKA.

[7] Guzzle. Guzzlephp.org [online]. [cit. 2018-12-18]. Dostupné z: http://docs.guzzlephp.org/en/stable/overview.html

[8] SPoT API | API Explorer. Ruckuslbs.com [online]. [cit. 2018-12-18]. Dostupné z: https://us-sys.ruckuslbs.com/api_explorer

[9] Dokumentace Laravel | Artisan Console. Laravel.com [online]. [cit. 2018-12-18]. Dostupné z: https://laravel.com/docs/5.7/artisan

[10] O'NEIL, Elizabeth J. Object/relational mapping 2008: hibernate and the entity data model (edm). In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008. p. 1351-1356.

[11] Dokumentace Laravel | Database: Migrations. Laravel.com [online]. [cit. 2018-12-18]. Dostupné z: `https://laravel.com/docs/5.7/artisan`

[12] Dokumentace Laravel | Task Scheduling. Laravel.com [online]. [cit. 2018-12-20]. Dostupné z: `https://laravel.com/docs/5.7/scheduling`

[13] AGUILERA, Marcos Kawazoe; CHEN, Wei; TOUEG, Sam. Heartbeat: A timeout-free failure detector for quiescent reliable communication. In: International Workshop on Distributed Algorithms. Springer, Berlin, Heidelberg, 1997. p. 126-140.

[14] DIJKSTRA, Edsger W. On the role of scientific thought. In: Selected writings on computing: a personal perspective. Springer, New York, NY, 1982. p. 60-66.

[15] NIELSEN, Jakob. 10 usability heuristics for user interface design. Nielsen Norman Group, 1995, 1.1.

[16] BROWN, Kevin, WEISSMAN, Alexander, ed. Select2. Select2: Dokumentace [online]. [cit. 2019-01-09]. Dostupné z: `https://select2.org/`

[17] ALMSAEED, Abdullah. AdminLTE. Adminlte. [online]. [cit. 2019-01-09]. Dostupné z: `https://adminlte.io/themes/AdminLTE/index2.html`

[18] CURL. Curl.haxx.se [online]. [cit. 2019-01-09]. Dostupné z: `https://curl.haxx.se/`

[19] MATHUR, Aditya P. Foundations of software testing, 2/e. Pearson Education India, 2013.

[20] HUIZINGA, Dorota; KOLAWA, Adam. Automated defect prevention: best practices in software management. John Wiley & Sons, 2007.

[21] Dokumentace Laravel | Testing. Laravel.com [online]. [cit. 2019-02-05]. Dostupné z: `https://laravel.com/docs/5.7/testing`

Příloha **A**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF