ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Sběr dat z platformy Garmin pro použití v telerehabilitaci

Diplomová práce

Bc. Václav Šídlo

Květen 2022
Vedoucí práce: doc. Ing. Miroslav Bureš, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šídlo** Jméno: **Václav** Osobní číslo: **474410**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Sběr dat z platformy Garmin pro použití v telerehabilitaci

Název diplomové práce anglicky:

Data collection from Garmin platform for telerehabilitation purposes

Pokyny pro vypracování:

Seznamte se s možnostmi sběru dat z platformy Garmin (počet kroků, srdeční tep, aktivita, atd.) pro použití v telerehabilitaci a telecoachingu. Navrhněte a implementujte technickou infrastrukturu pro sběr dat z fitness náramků Garmin. Následně navrhněte a implementujte přenos sesbíraných dat do již existující aplikace TERESA (TEleREhabilitation Self-training Assistant). Vytvořenou infrastrukturu otestujte podle zadání školitele.

Seznam doporučené literatury:

Kanitthika Kaewkannate a Soochan Kim. A comparison of wearable fitness devices. BMC Public Health. 2016, 16 (1), DOI 10.1186/s12889-016-3059-0.
Peter Duking, Andreas Hotho, Hans-Christer Holmberg, Franz Konstantin Fuss and Billy Sperlich. Comparison of Non-Invasive Individual Monitoring of the Training and Health of Athletes with Commercially Available Wearable Technologies. Frontiers in Physiology. 2016, 7 DOI 10.3389/fphys.2016.00071.
Bruno Nascimento, Tiago Oliveira a Carlos Tam. Wearable technology: What explains continuance intention in smartwatches?. Journal of Retailing and Consumer Services. 2018, 43 157–169. DOI 10.1016/j.jretconser.2018.03.017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Miroslav Bureš, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **30.01.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání	Podpis studenta

Poděkování / Prohlášení

Úvodem bych rád poděkoval své rodině a přátelům za podporu během celého studia. Dále děkuji svému vedoucímu doc. Ing. Miroslavu Burešovi, Ph.D. za veškerou podporu a cenné rady při vypracovávání této práce.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 19. 5. 2022

.....

Abstrakt / Abstract

Tato práce se zabývá sběrem dat (počet kroků, srdeční tep, informace o spánku, typ aktivity, intenzita aktivity, atd.) pro použití v telerehabilitaci a telecoachingu. Data jsou sbírána z několika platforem různých výrobců - Garmin, Fitbit, Xiaomi.

V rámci této práce je na základě požadavků navrhnuta a implementována vhodná technická infrastruktura pro sběr dat z těchto platforem, stejně jako implementována logika sběru dat ze zvolených platforem. Data jsou následně přenášena do existující aplikace TERESA (TEleREhabilitation Self-training Assistant), která byla v rámci této práce rozšířena o rozhraní pro sběr těchto dat a o detailní reprezentaci sesbíraných dat pro jejich využití lékaři v rámci telerehabilitace a telecoachingu.

Klíčová slova: telerehabilitace, telecoaching, senzory, COVID-19, garmin, fitbit, xiaomi, mikroslužby, kafka, zasílání zpráv

This thesis deals with the collection of data (step count, heart rate, sleep information, activity type, activity intensity, etc.) for use in telerehabilitation and telecoaching. Data are gathered from multiple platforms from multiple vendors - Garmin, Fitbit, Xiaomi.

In this thesis, based on the requirements, technical infrastructure for collecting data is well designed and implemented, as well as the logic for gathering data from the selected platforms. Collected data are then transfered to the existing TERESA (TEleREhabilitation Self-training Assistant) application, in which was as part of this thesis implemented interface for collecting the data and the logic for detailed representation of collected data for usage by doctors in telerehabilitaion and telecoaching procedures.

Keywords: telerehabilitation, telecoaching, sensors, COVID-19, garmin, fitbit, xiaomi, microservices, kafka, messaging

Title translation: Gathering data from the Garmin platform for the use in telerehabiliation (Diploma thesis)

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

V posledních třiceti letech došlo k výraznému vývoji technologií v oblasti internetu a elektrotechniky. Spotřební elektronika prudce zlevnila a společně s internetem se stala v západním světě běžnou a dostupnou součástí života[1]. V roce 2007 byl představen první chytrý telefon iPhone[2] a společně s ním došlo k překotnému růstu technologií, které většina lidí používá na každodenní bázi a se kterými jsou prakticky neustále ve fyzickém kontaktu. Díky přenositelným zařízením tohoto typu a bezdrátovému připojení se lidé stali nepřetržitě dostupnými prakticky odkudkoliv na světě.

Online spojení a menší, úspornější technologie daly vzniknout novému segmentu elektroniky - takzvané nositelné elektronice, neboli wearables. Zpravidla se jedná o elektroniku malých rozměrů, která slouží k záznamu aktivit jako je chůze, běh, či kvalita spánku. Stejně tak je lze použít k měření základních životních funkcí pomocí neinvazivních způsobů. Takto lze měřit například srdeční tep, dechová frekvence, nebo okysličení krve. Pro pohodlné používání je zařízení zpravidla synchronizováno bezdrátově s mobilním telefonem a umí pracovat samostatně. Tedy člověk si například nemusí vzít mobilní telefon s sebou pro to, aby si mohl zaznamenat svou jízdu na kole[3].

Měření základních životních funkcí a aktivit, prakticky v reálném čase, dává prostor pro využití těchto dat k medicínským účelům. Vznikl tak obor telemedicíny. Telemedicína je souhrnné označení pro medicínskou aktivitu, která je poskytována na dálku[4]. Může se jednat o telefonické konzultace, či dlouhodobé sdílení komplexních dat s lékaři jedinců. Díky tomu je možné poskytnout zdravotní péči jak v odlehlých oblastech, tak v případech, kdy je pro pacienta obtížné se dostat fyzicky k lékaři, např. v pokročilém věku[5]. S rozvojem telemedicíny a tedy sdílení velmi citlivých dat prostřednictvím internetu je nutné dbát na dostatečné zabezpečení celé komunikace a ochranu citlivých dat před jejich zneužitím[6].

1.1 Motivace

Poté, co na jaře roku 2020 vypukla celosvětová pandemie COVID-19, vznikl po celém světě velký počet výzkumných skupin zabývajících se následky této nemoci. Jednou z oblastí jsou plicní rehabilitace pro pacienty, kteří prodělali těžký průběh onemocnění COVID-19 a potýkají se s dýchacími obtížemi, které jim narušují každodenní život[7]. Jednou z těchto skupin je pracovní skupina týmu odborníků z Českého vysokého učení technického v Praze (ČVUT), Fakultní nemocnice Hradec Králové (FN HK), Univerzity Palackého v Olomouci (UPOL) a Univerzity obrany (UNOB), jejíž členové spolupracují na projektu TERESA (TEleREhabilitation Self-training Assistant)[8].

Hlavním cílem projektu je snaha umožnit rehabilitaci pacientů s přetrvávajícími následky po prodělaném onemocnění COVID-19 v domácím prostředí, stejně jako zlepšit terapii tím, že terapeuti budou moci využívat přesnější a dlouhodobější data díky sběru pohybové aktivity přímo u pacientů. V neposlední řadě je snaha o zvýšení kapacity zdravotnických zařízení díky zefektivnění terapie.

K tomu, aby bylo možné zařadit do plánu rehabilitací co nejvíce účastníků, se tým rozhodl ustoupit od drahých profesionálních medicínských zařízení a rozhodl se využít běžně dostupnou wearables elektroniku, konkrétně fitness náramky.

V rámci průběhu projektu byla provedena pilotní studie. Studie měla za cíl otestování a vyhodnocení spolehlivosti a funkčnosti technického řešení, stejně jako ověření vlivu na zlepšení zdravotního stavu zapojených pacientů. Díky pozitivním výsledkům této studie bylo rozhodnuto o rozšíření zaměření tohoto projektu a to nejen na pacienty s následky onemocnění COVID-19, ale i ostatní pacienty, kteří mají jiné komplikace s dýchací soustavou po jiných typech onemocnění jako například po bronchitidě.

1.2 Cíle této práce

Fitness náramky nejsou běžně využívány pro sběr dat za účelem telemedicíny, včetně telerehabilitací a telecoachingu, kterými se projekt TERESA zabývá. Z těchto důvodů je cílem této práce analyzovat požadavky a na jejich základě vhodně navrhnout, implementovat a otestovat technickou infrastrukturu pro sběr dat z několika platforem. Konkrétně se jedná o platformy Garmin, Fitbit a Xiaomi (Gadgetbridge).

Sesbíraná data je následně potřeba agregovat v existující aplikaci TERESA (TEle-REhabilitation Self-training Assistant), kterou je v rámci této práce nutné o tuto logiku vhodně rozšířit.

V zadání této práce je požadován sběr dat pouze z platformy Garmin. Nad tento stanovený rozsah byl rozsah práce navíc rozšířen i o sběr dat z platformy Fitbit, společně s úpravou existujícího sběru dat z platformy Gadgetbridge. Rozšíření o úpravu sběru z platformy Gadgetbridge je z toho důvodu, aby sběr z této platformy odpovídal technické infrastruktuře navržené a vyvinuté v rámci této práce.

Kapitola 2
Nositelná zařízení, Wearables

Nositelná zařízení, nebo z angličtiny také wearables, v dnešním smyslu jsou elektronická zařízení, která obsahují množství senzorů. Od ostatních elektronických zařízení se odlišují tím, že je uživatel používá tak, že jsou v kontaktu s tělem uživatele. Může se jednat o hodinky, šperky, oblečení, ale i podkožní implantáty. Tato zařízení jsou schopna uživateli, který je nosí na svém těle, poskytovat informace o jeho aktivitě a fyziologických funkcích, či mu přinášet jiné benefity, jako zlepšení smyslového vnímání či motoriky[9].

2.1 Historie

Nositelná elektronika by se mohla zdát jako nový vynález. Ale technika, kterou by člověk nosil pro to, aby zlepšil svou kvalitu života, je zde již velmi dlouho. Při pohledu do historie již v 16. století německý vynálezce Peter Henlein vytvořil malé hodiny, které nosil jako řetízek 2.1. V 17. století se z nich pak stal doplněk, převážně pro ženy[10].

Obrázek 2.1. První hodinky na světě z počátku 16. století[11]

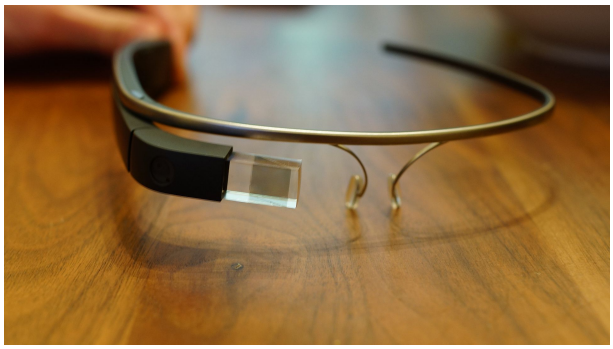
V 19. století bylo vyvinuto první ušní naslouchátko[12]. Největší rozmach nositelných zařízení však byl umožněn díky rozvoji technologií a jejich miniaturizací od druhé poloviny dvacátého století, nejvíce však po roce 2000 a pokračuje dodnes. V sedmdesátých letech dvacátého století byly populární např. hodinky s kalkulačkou. V roce 2009 pak představila společnost Fitbit svůj první krokoměr viz obrázek 2.2 a v roce 2013 byly představeny chytré brýle od společnosti Google viz obrázek 2.3.

V poslední dekádě pak došlo k výraznému růstu všech možných chytrých náramků, hodinek, zdravotních pomůcek jako jsou měřiče hladiny cukru v krvi, dále například hrudní pásy pro přesné měření srdeční aktivity, měřiče spánku a další.

2.2 Senzory

Se zlevňováním, zpřesňováním a snižováním energetické náročnosti jednotlivých senzorů dochází k čím dál tím větší senzorové vybavenosti zařízení, která jsou běžně dostupná na

Obrázek 2.2. První krokoměr Fitbit z roku 2009[3]



Obrázek 2.3. Chytré brýle Google Glass z roku 2013[13]

trhu. Zároveň výrobci představují jedinečné senzory, či nové funkcionality díky zpřesnění a vylepšení algoritmů, které by jejich produkty odlišily od ostatních. Naprostá většina zařízení však obsahuje několik standardních senzorů jako akcelerometr, gyroskop, senzor srdečního tepu, či GPS, které budou detailněji popsány v následujících podkapitolách.

2.2.1 Akcelerometr a gyroskop

Akcelerometr a gyroskop jsou senzory, které jsou součástí téměř každého zařízení. Tyto senzory slouží k detekci pohybu ve všech směrech a k měření zrychlení[14]. Data z nich mohou být využita hned v několika oblastech. Algoritmy je mohou zpracovat k detekci počtu kroků, výpočtu intenzity pohybové aktivity, ale i k záznamu kvality spánku a spánkové aktivitě. Pokročilými funkcemi pak může být např. detekce epileptických záchvatů[15].

2.2.2 Senzor srdečního tepu

Existuje několik druhů senzorů srdečního tepu. Nejčastěji používaným senzorem na běžně dostupné nositelné elektronice, hlavně náramcích, je optický senzor. Ten funguje na principu toho, že kůže je osvětlena zeleným nebo červeným světlem a následně je odečítán tok krve, díky čemuž může být detekován srdeční tep[15]. Kromě srdečního tepu poskytují moderní zařízení i další údaje jako variabilitu rytmu srdce, která může sloužit jako jeden z ukazatelů stresové zátěže, či únavy[16].

2.2.3 GPS

Velká část (převážně dražších) zařízení disponuje senzorem GPS - globálního pozičního systému. Tento senzor je používán k určení polohy, stejně jako je používán pro záznam trasy pohybu a jejích parametrů, jako je vzdálenost či převýšení[15]. Uživatelé jsou tak schopni dostávat zajímavé statistiky o jejich aktivitě a trasy, po kterých se pohybovali. Výhodou toho, že je senzor přímo v zařízení a nikoliv pouze v telefonu, je hlavně to, že si uživatel s sebou např. při běhání nemusí brát svůj telefon a stačí mu pouze nositelné zařízení. Nevýhodou je, že senzor GPS má relativně velkou spotřebu energie a proto může významně snižovat dobu, po kterou zařízení vydrží bez nabíjení[15].

2.3 Rozhraní

Zařízení jsou zpravidla propojena bezdrátově s mobilním telefonem, který naměřená data z nich agreguje a dále zpracovává. K bezdrátovému spojení je zpravidla používána technologie Bluetooth, ale u některých typů zařízení může být použita i technologie RFID, nebo NFC[17].

2.4 Typy zařízení

V dnešní době je nositelná elektronika běžně dostupná v mnoha různých provedeních. Co se týče zastoupení na trhu, pak lídry v sekci komerčních konzumních produktů jsou jak internetoví giganti jako Google, Apple a Microsoft, tak výrobci elektroniky jako Samsung, LG, Garmin, stejně jako výrobci, kteří se zaměřují primárně na nositelnou elektroniku, jako např. Fitbit[18].

2.4.1 Chytré hodinky a náramky

Hodinky a náramky jsou jedním z nejběžnějších typů nositelné elektroniky. Jedná se o zařízení, které uživatel nosí na zápěstí a které je vybaveno výpočetní jednotkou, různými senzory viz kapitola 2.2 a rozhraním viz kapitola 2.3, přes které je zařízení schopno bezdrátově komunikovat s dalšími zařízeními v jeho blízkosti. Zařízení jsou zpravidla vodotěsná a prachotěsná tak, aby se dala používat při všech běžných denních i sportovních aktivitách, tedy například plavání a sprchování[19].

Nejběžnější senzory byly popsány v předešlé kapitole 2.2. Dražší modely hodinek a náramků mohou mít zařízení i senzory pro měření okysličení krve či EKG[20].

Co se týče komunikace s okolním světem, pak zařízení mají zpravidla připojení pomocí Bluetooth, některá zařízení umí využívat Wi-Fi, nebo i mobilní sítě. Pro možnost zaznamenávat trasu při fyzické aktivitě jsou některá zařízení vybavena GPS.

Rozdíly mezi hodinkami a náramky nemusí být vždy patrné, hodinky na rozdíl od náramků mají vždy displej, aby mohly zobrazit čas a další informace. Naopak náramky žádný displej mít nemusí. Výrobci těchto zařízení se snaží uživatelům přinášet i další funkcionality, tedy např. zobrazování notifikací z mobilního telefonu přímo na hodinkách, detekci pádu při sportu a případné zavolání záchranných složek, denní přehledy aktivit a množství dalších funkcionalit[21]. Díky tomu a díky své velké senzorové vybavenosti a široké škále využití se jedná o jeden z nejprodávanějších typů nositelné elektroniky[18].

2.4.2 Šperky

Digitální šperky jsou jednou z kategorií nositelné elektroniky. Svou oblibu si získaly díky tomu, že plní nejen svou funkci z pohledu nositelné elektroniky, ale jedná se především o šperk. Často jsou vyrobeny z chirurgické oceli a většinou se jedná o prsteny viz obrázek 2.4, náušnice a další běžné šperky[22].

V poslední době je velmi populární prsten Oura finského výrobce viz obrázek 2.4, jehož data se často využívají i při vědeckých výzkumech, např. při studiích o kvalitě spánku[23].

Obrázek 2.4. Prsten Oura[24]

2.4.3 Oblečení

Jednou z dalších aplikací může být zabudování elektroniky do oblečení. U oblečení je nutné se potýkat s tím, že prádlo je třeba prát, a tudíž musí být veškerá elektronika a senzory odolné vůči praní. Díky přímému kontaktu s větší plochou těla je pak možné sledovat přesněji např. svalovou aktivitu nebo činnost srdce[25].

2.4.4 Zařízení upevňující se na hlavu

V posledních letech vzrůstají na popularitě zařízení upevňující se na hlavu. Jak již bylo zmíněno, v roce 2013 uvedla společnost Google na trh chytré brýle viz obrázek 2.3. Tento segment se nadále neustále rozvíjí. Kromě brýlí se jedná o zařízení pro rozšířenou (augmentovanou) realitu, či pro virtuální realitu, např. brýle HoloLens 2 od společnosti Microsoft viz obrázek 2.5.

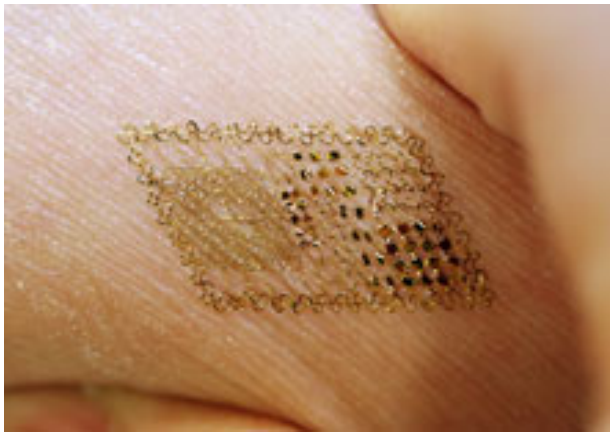
Obrázek 2.5. Brýle HoloLens 2[26]

Pokud se podíváme na medicínské aplikace v oblasti nositelné elektroniky, pak jako jedno z využití můžeme uvést naslouchátka na podporu sluchu.

2.4.5 Implantáty

Podkožní implantáty jsou speciální kategorií nositelné elektroniky. Jde o elektroniku, která je implantována pod lidskou pokožku a stává se tudíž součástí uživatele. Formy jsou různé, může se jednat o malé čipy aplikované pod pokožku, nebo např. chytré tetování viz obrázek 2.6, které je složeno z drobných součástí tenčích než lidský vlas, které může monitorovat zdravotní stav uživatele[27].

Využití je různé, ale jako u ostatních kategorií se zpravidla jedná o monitorování fyziologických funkcí uživatele jako jsou srdeční a svalová aktivita.



Obrázek 2.6. Chytré tetování[27]

2.5 Budoucí vývoj nositelné elektroniky

V budoucnu se dá očekávat pokračující trend zmenšování, snižování spotřeby energie a zpřesňování jednotlivých senzorů, s čímž je očekáván další rozvoj nositelné elektroniky. Již nyní dochází k experimentům s novými typy senzorů, které mohou snímat např. intenzitu a složení potu a na základě těchto dat pak například zařízení automaticky dávkují léky[17].

Obecně se předpokládá, že pomocí komplexnějších algoritmů, zvýšení dostupnosti nositelné elektroniky a zvýšení komfortu při jejím používání, bude docházet k tomu, že jednotliví výrobci budou schopni poskytnout celkový obrázek o stavu jedince na základě agregace dat ze všech zařízení, které pacient používá a bude díky tomu možné velmi brzy rozpoznat potenciální příznaky onemocnění[28].

2.6 Využití wearables v medicíně

Wearables mají díky obrovskému rozvoji v poslední době potenciál ovlivnit kvalitu, rychlost a dostupnost zdravotní péče. Jejich možnosti využití v oblastech zdravotní péče jsou velmi široké. Jedná se jak o využití v době, kdy je pacient přímo v nemocnici, tak v případě ambulantní péče, ale i v rámci prevence[17].

Výhodou, ale i nevýhodou využití wearables v oblasti medicíny je to, kolik dat jsou zařízení schopna sesbírat. Lékař je pak schopen dostat detailní obrázek nejen o aktuálním, ale i dlouhodobém trendu pacientova stavu. Vzhledem k tomu, že je ale sesbíraných

dat velké množství, je nutné vyvíjet komplexní algoritmy, které jsou schopny agregovat data z různých zařízení a na základě jejich analýzy určit odchylky od normálů a označit tak oblasti, na které by se měl lékař zaměřit[28].

Po vytvoření dostatečně komplexních a přesných algoritmů je pak možné z naměřených dat pomocí běžných zařízení v předstihu, či v zárodku upozornit na začínající onemocnění. Může se jednat o spánkové deprivace, depresivní stavy, těhotenství, plicní poruchy, poruchy srdce, problémy s motorikou a další[28].

Překážkou v širším využívání wearables v případě lékařské péče jsou v nemálo případech regulatorní omezení. Jedná se o to, že například v případě, kdy by chtělo medicínské zařízení používat nějakou elektroniku pro monitoring srdečního tepu, pak daný senzor musí být certifikován podle stejných regulací jako ostatní vybavení. Tyto certifikace běžně dostupná komerční zařízení zpravidla nemají a nemůžou z těchto důvodů být běžně používána pro medicínské účely[28].

Další výzvou je pak přesnost a spolehlivost senzorů na běžně dostupné elektronice, která dost často není ověřena ve velkých nezávislých studiích a z tohoto důvodu se na ní nemůže spoléhat jako na přesný ukazatel[17].

2.7 Volba zařízení pro účely projektu TERESA

V počátcích projektu TERESA bylo nutné určit, který typ nositelných zařízení bude vhodný pro účely tohoto projektu. Tedy zvolit z jednotlivých typů nositelné elektroniky, detailně popsaných v kapitole 2.4, takový typ zařízení, který bude nejvíce vyhovovat potřebám tohoto projektu a bude vhodný pro cílovou skupinu pacientů, kteří budou zařízení používat. Cílová skupina pacientů jsou zejména senioři se zpravidla malou technickou znalostí. Důraz při volbě byl kladen na následující oblasti:

- komfort při dlouhodobém nošení, neboť data je třeba sbírat nepřetržitě a dlouhodobě
- jednoduchost ovládání a používání
- dostupnost na trhu
- přepoužitelnost zařízení pro více pacientů
- příznivá cena, neboť zařízení bude potřeba řádově vyšší desítky až stovky kusů

Po zhodnocení těchto kritérií se nejvhodnějším typem zařízení pro tento projekt ukázaly být chytré hodinky a náramky. Faktory, které přispěly k jejich volbě, byly ty, že splňují veškeré požadavky, které byly od zařízení očekávány. Jedná se o nejběžnější typ nositelné elektroniky, tudíž existuje velké množství produktů na trhu, ze kterých je možné vybírat. Díky možnosti výběru a velké konkurence je u těchto zařízení i příznivá cena. Díky tomu, že se jedná o hodinky či náramky, tak se i pro budoucí pacienty jedná o známé zařízení, neboť klasické hodinky nosil prakticky každý. Druhým kandidátem byly prsteny, zejména prsten Oura viz kapitola 2.4, avšak tento typ zařízení nebyl nakonec vybrán a to zejména z důvodu toho, že prsten je závislý na velikosti prstu a každý pacient by potřeboval jiný, zatímco náramky a hodinky může používat kdokoliv (pokud nemá obvod zápěstí extrémně velký, nebo malý).

Kapitola 3
Sběr a analýza požadavků

V rámci životního cyklu vývoje softwaru dochází velmi často k nedorozumění mezi tím, co je zákazníkem požadováno a očekáváno a tím, co programátor reálně naprogramoval a jak se software reálně chová. Tyto chyby jsou natolik časté, že mají za následek až 50% všech chyb, které jsou v softwaru nalezeny[29].

Z těchto důvodů je vždy nutné před začátkem vývoje sesbírat, analyzovat a definovat požadavky tak, aby s nimi byly všechny zainteresované osoby seznámeny a aby s nimi všichni souhlasili. Díky takovému seznamu požadavků je pak ukotvena očekávaná funkčnost systému, stejně jako je následně možné vhodně navrhnout architekturu, design a chování vyvíjené aplikace. V rámci softwarového vývoje vnímáme termín požadavek jako specifikaci toho, co má být implementováno. Tedy je to popis toho, co má systém dělat, za jakých okolností a jaké jsou omezující podmínky[29].

3.1 Stávající stav aplikace TERESA

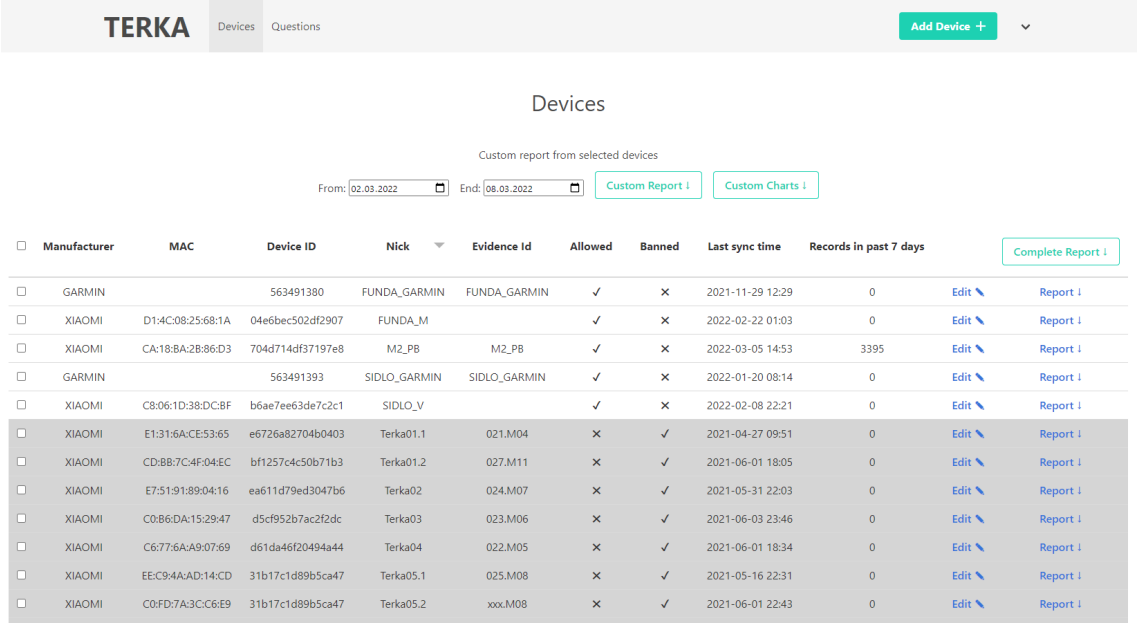
Jak již bylo řečeno v předešlých kapitolách, v době psaní této práce již existovala aplikace TERESA, která byla navrhnuta a vyvinuta členy v rámci pracovní skupiny v počáteční fázi celého projektu.

Aplikace TERESA sestává z rozhraní, které umožňuje sbírat data o počtu kroků, srdečním tepu a typu pohybové aktivity ze zařízení Mi Band 5[30] od výrobce Xiaomi. Data jsou z náramků výrobce Xiaomi sbírána za využití aplikací Gadgetbridge a Konektor. Aplikace Gadgetbridge je open-source aplikace umožňující přímé napojení na náramky výrobce Xiaomi a následný export sesbíraných dat pro využití dalšími aplikacemi. Aplikace Konektor je studentská práce vyvinutá výhradně pro účely tohoto projektu za účelem sběru dat z náramků výrobce Xiaomi skrze aplikaci Gadgetbridge.

Komunikace mezi aplikací Konektor a aplikací TERESA je řešena pomocí protokolu HTTPS s použitím REST (*REpresentational State Transfer*). Aplikace Konektor slouží i k dalším účelům, jako jsou notifikace uživatelů o tom, co mají dělat a též ke sbírání odpovědí na dotazníky, které jsou jim každý den předkládány k vyplnění. Zadání na dotazníky i notifikace jsou řízeny aplikací TERESA, která je pomocí rozhraní komunikuje do aplikace Konektor.

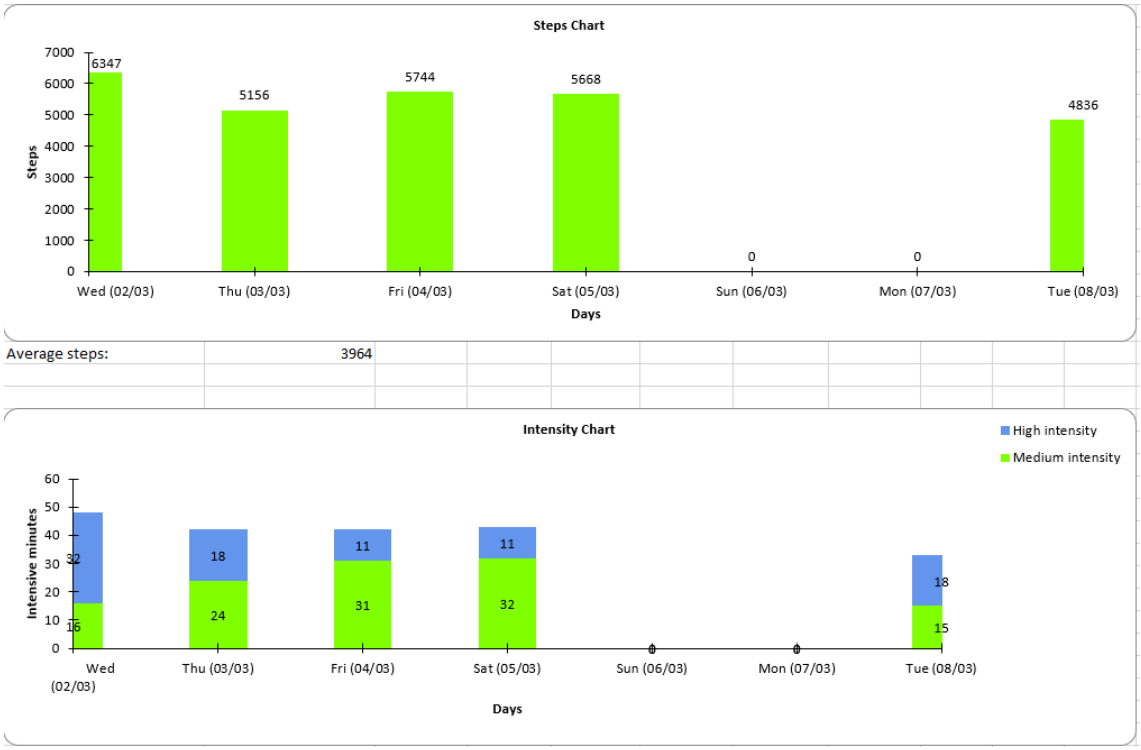
Poslední částí aplikace TERESA je uživatelské rozhraní, které umožňuje spravovat životní cyklus jednotlivých zařízení. Kromě toho umožňuje lékařům upravovat a nastavovat obsah dotazníků, stejně jako typ a frekvenci notifikací zobrazovaných pacientům v aplikaci Konektor. Sesbíraná data je následně možné reportovat do CSV souborů pro následné využití lékaři v telerehabilitaci.

Ukázky stávajícího stavu aplikace TERESA a některých funkčností jsou zobrazeny na obrázcích 3.1 a 3.2.



	Manufacturer	MAC	Device ID	Nick	Evidence Id	Allowed	Banned	Last sync time	Records in past 7 days		
☐	GARMIN		563491380	FUNDA_GARMIN	FUNDA_GARMIN	✓	×	2021-11-29 12:29	0	Edit ✎	Report ↓
☐	XIAOMI	D1:4C:08:25:68:1A	04e6bec502df2907	FUNDA_M		✓	×	2022-02-22 01:03	0	Edit ✎	Report ↓
☐	XIAOMI	CA:18:BA:2B:86:D3	704d714df37197e8	M2_PB	M2_PB	✓	×	2022-03-05 14:53	3395	Edit ✎	Report ↓
☐	GARMIN		563491393	SIDLO_GARMIN	SIDLO_GARMIN	✓	×	2022-01-20 08:14	0	Edit ✎	Report ↓
☐	XIAOMI	C8:06:1D:38:DC:BF	b6ae7ee63de7c2c1	SIDLO_V		✓	×	2022-02-08 22:21	0	Edit ✎	Report ↓
☐	XIAOMI	E1:31:6A:CE:53:65	e6726a82704b0403	Terka01.1	021.M04	×	✓	2021-04-27 09:51	0	Edit ✎	Report ↓
☐	XIAOMI	CD:BB:7C:4F:04:EC	bf1257c4c50b71b3	Terka01.2	027.M11	×	✓	2021-06-01 18:05	0	Edit ✎	Report ↓
☐	XIAOMI	E7:51:91:89:04:16	ea611d79ed3047b6	Terka02	024.M07	×	✓	2021-05-31 22:03	0	Edit ✎	Report ↓
☐	XIAOMI	C0:B6:DA:15:29:47	d5cf952b7ac2f2dc	Terka03	023.M06	×	✓	2021-06-03 23:46	0	Edit ✎	Report ↓
☐	XIAOMI	C6:77:6A:A9:07:69	d61da46f20494a44	Terka04	022.M05	×	✓	2021-06-01 18:34	0	Edit ✎	Report ↓
☐	XIAOMI	EE:C9:4A:AD:14:CD	31b17c1d89b5ca47	Terka05.1	025.M08	×	✓	2021-05-16 22:31	0	Edit ✎	Report ↓
☐	XIAOMI	C0:FD:7A:3C:C6:E9	31b17c1d89b5ca47	Terka05.2	xxx.M08	×	✓	2021-06-01 22:43	0	Edit ✎	Report ↓

Obrázek 3.1. Stávající uživatelské rozhraní aplikace TERESA



Obrázek 3.2. Ukázka reportů s počtem kroků generovaných do Excel souborů

3.2 Cíle rozšíření v rámci této práce

Požadavky na řešení, které má vzniknout v rámci této práce, byly sbírány kontinuálně. Jak na počátku, tak i v průběhu projektu TERESA při pravidelných schůzkách s pracovní skupinou odborníků, kteří se na tomto projektu podílí. Všechny sesbírané požadavky, které jsou od řešení očekávány a které je nutné splnit v rámci této práce, jsou popsány v následujících sekcích.

Všechny vzniklé požadavky jsou ovlivněny tím, že řešení vyvíjené v rámci této práce má vhodně rozšířit a doplnit již existující aplikaci TERESA, jež byla vyvinuta dříve ostatními členy pracovní skupiny 3.1.

3.3 Funkční požadavky

- **FR01 – Typ sbíraných dat**
  Řešení umožní sběr dat o srdečním tepu, počtu kroků, pohybové intenzitě a kvalitě spánku. Pokud to daná platforma výrobce umožní, je vyžadováno sbírat data s minutovou granularitou.
- **FR02 – Seznam výrobců a platforem pro sběr dat**
  Řešení umožní sběr dat z platforem vybraných výrobců – Fitbit, Garmin.
- **FR03 – Ukládání dat v centrální databázi TERESA**
  Řešení umožní ukládání sesbíraných dat z jednotlivých platforem v jednotném formátu v centrální databázi aplikace TERESA.
- **FR04 – Evidence zařízení různých výrobců v aplikaci TERESA**
  Řešení rozšíří existující webové rozhraní aplikace TERESA tak, aby bylo možné evidovat pacienty využívající náramky různých výrobců.
- **FR05 – Detailní přehledy dat v aplikaci TERESA**
  Řešení rozšíří existující webové rozhraní aplikace TERESA o detailní přehledy sesbíraných dat, tedy detailní denní přehledy počtu kroků, tepu a kvality spánku pro jednotlivé pacienty.
- **FR06 – Sběr dat poskytovaných aplikací Konektor v novém řešení**
  Řešení rozšíří, případně upraví stávající logiku sběru dat poskytovaných aplikací Konektor tak, aby byla součástí nově vzniklého řešení.

3.4 Nefunkční požadavky

- **NFR01 – Počet uživatelů**
  Řešení musí být schopné zpracovat a archivovat kompletní denní data o srdečním tepu, počtu kroků, pohybové intenzitě a kvalitě spánku od minimálně 500 uživatelů. *Poznámka – je očekáváno, že uživatelé budou svá data synchronizovat 1x denně zpravidla ve večerních hodinách*
- **NFR02 – Rozšiřitelnost o nové typy dat**
  Řešení musí být jednoduše rozšiřitelné o další typy sbíraných dat.
- **NFR03 – Rozšiřitelnost o platformy dalších výrobců**
  Řešení musí být jednoduše rozšiřitelné o platformy dalších výrobců, ze kterých budou data sbírána. *Poznámka – v budoucnosti je očekáváno, že bude systém rozšířen o sběr z platforem dalších výrobců.*
- **NFR04 – Šifrování**
  Vzhledem k citlivosti dat musí veškerý síťový provoz v rámci systému probíhat v šifrované podobě.
- **NFR05 – Částečná dostupnost**
  Řešení musí být schopné pracovat i v případě, že sběr dat z jedné platformy bude částečně nedostupný – např. v případě technické odstávky, atd.
- **NFR06 – Stabilita**
  Řešení nesmí ztratit data ani v případě, kdy dojde k výpadkům některých částí aplikace, jako je centrální databáze, připojení k internetu, pád aplikace, atd.
- **NFR07 – Konfigurace**

Konfigurace systému bude umožněna skrze konfigurační soubory tak, aby konfigurační parametry nebyly součástí kódu.

- **NFR08 – Kontejnerizace**

  Systém bude kontejnerizován tak, aby byl umožněn jednoduchý provoz a lehká přenositelnost napříč prostředími.

3.5 Případy užití

Na základě funkčních požadavků byl vytvořen detailnější rozpad těchto požadavků na jednotlivé případy užití. Případy užití byly rozděleny dle jejich logického zařazení do několika kategorií. Zároveň pro úpravy v aplikaci TERESA byl jako hlavní aktor definován *Přihlášený uživatel*. Pro ostatní kategorie byl jako hlavní aktor definován *Systém*.

- **Úpravy v aplikaci TERESA**
  - **UC01 – Registrace pacienta s náramkem od Garmin, Fitbit, Xiaomi**
    - 1. Přihlášený uživatel zvolí možnost registrace nového zařízení
    - 2. Systém zobrazí registrační formulář
    - 3. Uživatel vyplní požadované hodnoty - výrobce zařízení, MAC adresa zařízení, ID zařízení, evidenční číslo zařízení
    - 4. Uživatel potvrdí registraci zařízení
    - 5. Systém zkontroluje vyplnění povinných dat - ID zařízení a jeho výrobce. Pokud není některé pole vyplněno, uživatel zobrazí varovnou hlášku a scénář pokračuje bodem 3
    - 6. Uživatel uloží data o novém zařízení do databáze a přesměruje uživatele na hlavní obrazovku aplikace
  - **UC02 – Získání přístupu k datům z pacientova náramku od Garmin, Fitbit, Xiaomi**
    - 1. Přihlášený uživatel zvolí možnost editace zařízení
    - 2. Systém zobrazí editovatelná pole zařízení - výrobce zařízení, MAC adresa zařízení, ID zařízení, evidenční číslo zařízení a tlačítko `Aktivace zařízení`
    - 3. Uživatel stiskne tlačítko `Aktivace zařízení`
    - 4. Systém provede aktivaci zařízení v rámci aplikace v závislosti na výrobci zařízení - UC05 (Garmin), UC10 (Fitbit), UC15 (Xiaomi)
  - **UC03 – Export Excel souboru s detailními grafy obsahující přehledy o počtu kroků, tepu a kvalitě spánku za zvolené časové období pro jednoho pacienta**
    - 1. Systém načte sesbíraná data za daný interval pro dané zařízení
    - 2. Systém vytvoří nový Excel soubor s názvem `<DEVICE_ID>.xlsx` obsahující záložku s agregovaným přehledem dat a následně záložku pro každý den intervalu s detailními grafy
    - 3. Každá záložka s detailními grafy obsahuje následující položky: čas a hodnota naměřených kroků, čas a hodnota naměřeného srdečního tepu, čas a hodnota naměřené kvality spánku, graf naměřeného počtu kroků s minutovou granularitou (vždy 6:00–22:00), graf naměřeného srdečního tepu s minutovou granularitou (vždy 6:00–22:00), graf naměřené kvality spánku s minutovou granularitou
    - 4. Systém vrátí vytvořený soubor k dalšímu zpracování
  - **UC04 – Hromadný export Excel souborů z UC03 pro zvolené uživatele za zvolené období**
    - 1. Přihlášený uživatel vyplní časový interval, pro který si přeje generovat detailní grafy
    - 2. Uživatel zvolí všechna zařízení pro která chce grafy generovat a následně možnost generování detailních grafů

- 3. Systém pro každé zařízení vygeneruje detailní report, dle UC03
- 4. Systém všechny vygenerované soubory zabalí do archivu a vrátí je uživateli ke stažení. Archiv bude mít název `DetailedCharts.zip`

- **Sběr dat z platformy Garmin**
  - **UC05 – Autorizace zařízení vůči Garmin Connect Cloud**
    - 1. Systém přijme identifikátor zařízení, které má autorizovat vůči Garmin Connect Cloud a uloží si jej do databáze
    - 2. Systém zahájí autorizační proces s Garmin Connect Cloud a přesměruje uživatele na vyplnění přístupových údajů k zařízení
    - 3. Systém přijme autorizační tokeny z Garmin Connect Cloud a uloží si je k danému zařízení pro budoucí použití
    - 4. Systém přesměruje uživatele na hlavní obrazovku aplikace TERESA
  - **UC06 – Sběr dat o srdečním tepu z platformy Garmin**
    - 1. Systém přijme skrze rozhraní data o srdeční aktivitě
    - 2. Systém přijatá data rozdělí dle jednotlivých uživatelů, data pro uživatele, kteří nejsou systému známy, nejsou dále zpracovávány
    - 3. Systém pro každého uživatele, pro kterého přišla data, tato data rozdělí na části s minutovou granularitou
    - 4. Systém odešle pro každého uživatele všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
  - **UC07 – Sběr dat o počtu kroků z platformy Garmin**
    - 1. Systém přijme skrze rozhraní data o počtu kroků
    - 2. Systém přijatá data rozdělí dle jednotlivých uživatelů, data pro uživatele, kteří nejsou systému známy, nejsou dále zpracovávány
    - 3. Systém pro každého uživatele, pro kterého přišla data, tato data rozdělí na části s minutovou granularitou
    - 4. Systém odešle pro každého uživatele všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
  - **UC08 – Sběr dat o intenzitě pohybové aktivity z platformy Garmin**
    - 1. Systém přijme skrze rozhraní data o intenzitě pohybové aktivity
    - 2. Systém přijatá data rozdělí dle jednotlivých uživatelů, data pro uživatele, kteří nejsou systému známy, nejsou dále zpracovávány
    - 3. Systém pro každého uživatele, pro kterého přišla data, tato data rozdělí na části s minutovou granularitou
    - 4. Systém odešle pro každého uživatele všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
  - **UC09 – Sběr dat o kvalitě spánku z platformy Garmin**
    - 1. Systém přijme skrze rozhraní data o kvalitě spánku
    - 2. Systém přijatá data rozdělí dle jednotlivých uživatelů, data pro uživatele, kteří nejsou systému známy, nejsou dále zpracovávány
    - 3. Systém pro každého uživatele, pro kterého přišla data, tato data rozdělí na jednotlivé intervaly kvality spánku
    - 4. Systém odešle pro každého uživatele všechna data rozdělená do jednotlivých intervalů do aplikace TERESA
- **Sběr dat z platformy Fitbit**
  - **UC10 – Autorizace zařízení vůči Fitbit Cloud**

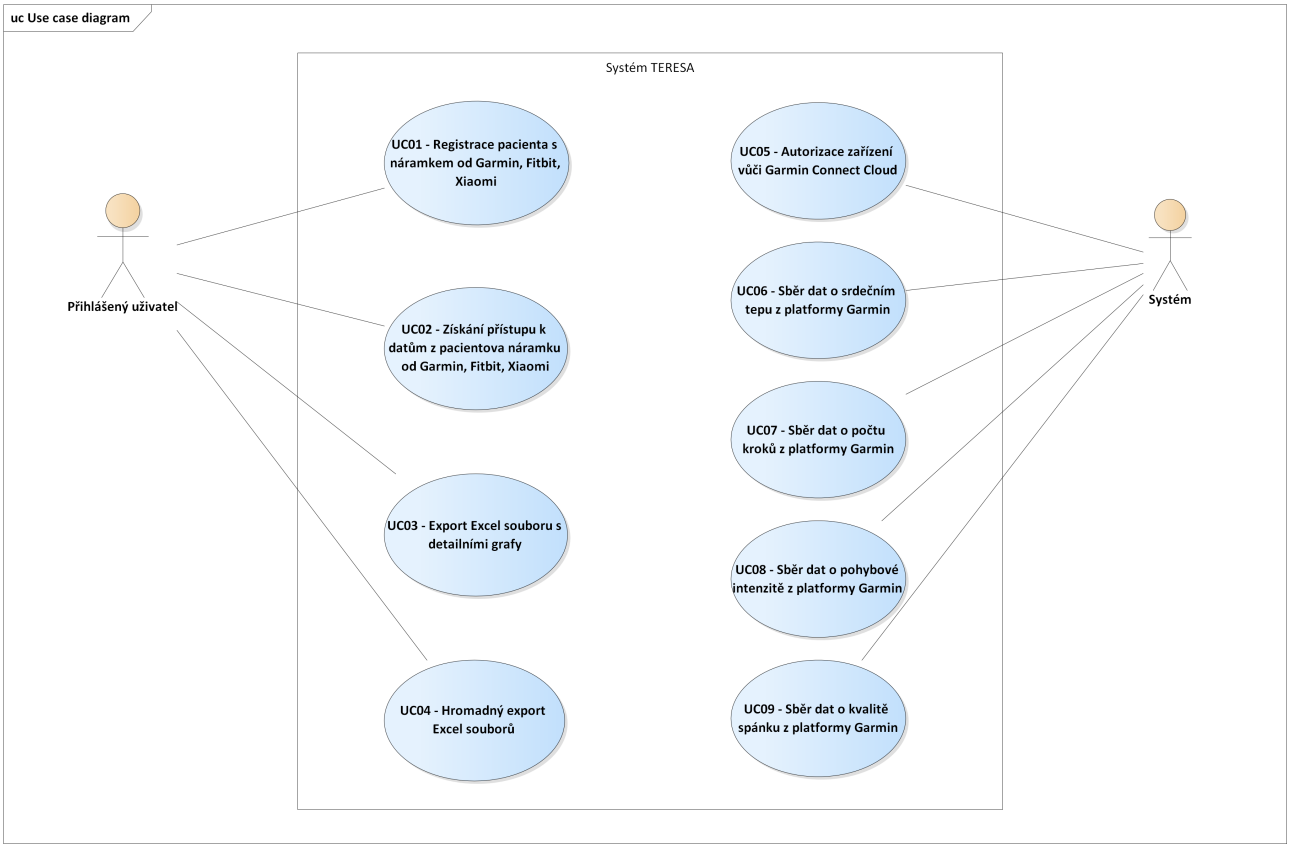
- 1. Systém přijme identifikátor zařízení, které má autorizovat vůči Fitbit Cloud a uloží si jej do databáze
- 2. Systém zahájí autorizační proces s Fitbit Cloud a přesměruje uživatele na vyplnění přístupových údajů k zařízení
- 3. Systém přijme autorizační tokeny z Fitbit Cloud a uloží si je k danému zařízení pro budoucí použití
- 4. Systém přesměruje uživatele na hlavní obrazovku aplikace TERESA

- **UC11 – Sběr dat o srdečním tepu z platformy Fitbit**
  - 1. Systém přijme skrze rozhraní notifikaci o dostupnosti nových dat o srdečním tepu pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém načte data z externího rozhraní Fitbit Cloud o srdečním tepu pro uživatele a den, pro který přišla notifikace o dostupnosti nových dat
  - 4. Systém přijatá data rozdělí na části s minutovou granularitou
  - 5. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC12 – Sběr dat o počtu kroků z platformy Fitbit**
  - 1. Systém přijme skrze rozhraní notifikaci o dostupnosti nových dat o počtu kroků pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém načte data z externího rozhraní Fitbit Cloud o počtu kroků pro uživatele a den, pro který přišla notifikace o dostupnosti nových dat
  - 4. Systém přijatá data rozdělí na části s minutovou granularitou
  - 5. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC13 – Sběr dat o intenzitě pohybové aktivity z platformy Fitbit**
  - 1. Systém přijme skrze rozhraní notifikaci o dostupnosti nových dat o intenzitě pohybové aktivity pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém načte data z externího rozhraní Fitbit Cloud o intenzitě pohybové aktivity pro uživatele a den, pro který přišla notifikace o dostupnosti nových dat
  - 4. Systém přijatá data rozdělí na části s minutovou granularitou
  - 5. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC14 – Sběr dat o kvalitě spánku z platformy Fitbit**
  - 1. Systém přijme skrze rozhraní notifikaci o dostupnosti nových dat o kvalitě spánku pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém načte data z externího rozhraní Fitbit Cloud o kvalitě spánku pro uživatele a den, pro který přišla notifikace o dostupnosti nových dat
  - 4. Systém přijatá data rozdělí na jednotlivé intervaly
  - 5. Systém odešle všechna data rozdělená do jednotlivých intervalů do aplikace TERESA

- **Sběr dat z platformy Xiaomi**
  - **UC15 – Autorizace zařízení**
    - 1. Systém přijme identifikátor zařízení, které má autorizovat a uloží si jej do databáze.
    - 2. Systém přesměruje uživatele na hlavní obrazovku aplikace TERESA

- **UC16 – Sběr dat o srdečním tepu z platformy Xiaomi**
  - 1. Systém přijme skrze rozhraní data o srdeční aktivitě pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém přijatá data rozdělí na části s minutovou granularitou
  - 4. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC17 – Sběr dat o počtu kroků z platformy Xiaomi**
  - 1. Systém přijme skrze rozhraní data o počtu kroků pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém přijatá data rozdělí na části s minutovou granularitou
  - 4. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC18 – Sběr dat o intenzitě pohybové aktivity z platformy Xiaomi**
  - 1. Systém přijme skrze rozhraní data o intenzitě pohybové aktivity pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém přijatá data rozdělí na části s minutovou granularitou
  - 4. Systém odešle všechna data rozdělená do jednotlivých části s minutovou granularitou do aplikace TERESA
- **UC19 – Sběr dat o souhrnném záznamu konkrétní aktivity z platformy Xiaomi**
  - 1. Systém přijme skrze rozhraní data o souhrnném záznamu aktivity pro daného uživatele
  - 2. Pokud uživatel není systému znám, pak scénář končí
  - 3. Systém odešle souhrnný záznam aktivity do aplikace TERESA

Případy užití jsou názorně zobrazeny na obrázku 3.3. Pro přehlednost byly vynechány UC10 - UC19, které jsou stejné jako UC05, UC06, UC07, UC08, UC09 s tím rozdílem, že se týkají jiných výrobců a platforem.



Obrázek 3.3. Diagram případů užití

Kapitola 4
Architektura řešení

V této kapitole je popsáno, co je to softwarová architektura. Dále jsou detailně popsány jednotlivé typy architektur. Na závěr je zvolena a navržena vhodná architektura pro tento systém.

4.1 Co je to softwarová architektura

Celé softwarové odvětví se dosud nebylo schopno shodnout na přesné definici toho, co je to softwarová architektura. Často se softwarová architektura definuje jako obecná předloha toho, jak by měla vypadat struktura softwaru, případně to, jakým způsobem je ke tvorbě softwaru přistupováno a jaký je celkový proces tvorby softwaru[31].

Dále v tomto textu je na architekturu nahlíženo dle následující definice: Softwarová architektura je soubor těchto vlastností:

- jak je systém strukturován
- jaké má architektura charakteristiky
- jaká rozhodnutí jsou při tvorbě architektuře zvolena
- jaké návrhové principy jsou použity[31].

Pokud jsou spojeny tyto čtyři oblasti, pak vznikne celkový pohled na architekturu řešení.

4.1.1 Struktura

Pokud se budeme dívat na architekturu z pohledu struktury, pak známe několik hlavních typů. Může se jednat o vrstevnatou architekturu, ta se vyznačuje tím, že máme jeden monolitický systém rozdělený do jednotlivých logických vrstev, které spolu komunikují. Dalším příkladem může být vnitřní dělení na komponenty, což jsou funkční logické celky, které spolu navzájem komunikují a spolupracují[31].

4.1.2 Charakteristiky

Pokud se díváme na architekturu z pohledu charakteristik, zajímají nás vlastnosti systému, které se liší na základě volby konkrétní architektury. Charakteristik, které volba konkrétní architektury ovlivňuje, je nespočet. Pokud vyjmenujeme ty hlavní, podle kterých se nejčastěji rozhodujeme, pak se jedná o následující: dostupnost, spolehlivost, testovatelnost, schopnost provádět změny, výkonnost, udržovatelnost, rozšiřitelnost, schopnost systému zotavit se z chyb a učící křivka pro vývojáře. Všechny tyto aspekty bychom při volbě a návrhu architektury pro konkrétní systém měli zhodnotit, určit jejich priority a dle nich zvolit vhodné řešení[31].

4.1.3 Rozhodnutí ovlivňující architekturu

Vzhledem k tomu, že při volbě architektury není pouze jediná možnost správná a existuje vždy několik variant řešení, mají naše konkrétní rozhodnutí vliv na výslednou architekturu. Příkladem může být např. to, že pokud mluvíme o vrstevnaté architektuře, tak máme několik možností, jak povolíme vrstvám mezi sebou komunikovat. Například u vrstevnaté architektury se můžeme rozhodnout, že povolíme jednotlivým vrstvám komunikovat pouze se sousedícími vrstvami. Tím zvýšíme izolaci vrstev. Na druhou stranu toto rozhodnutí může ovlivnit výkonnost tím, že nepůjde z nejvyšší prezentační vrstvy volat databázi přímo. Tato jednotlivá rozhodnutí tedy mohou významně ovlivnit to, jak finální architektura bude vypadat a jaké bude mít vlastnosti[31].

4.1.4 Návrhové principy

Poslední oblastí jsou návrhové principy. Návrhové principy nám říkají, jaké principy a zásady bychom měli respektovat při vývoji systému nad danou architekturou. Nejedná se o striktní nařízení, spíše doporučení a osvědčené postupy toho, jak se rozhodovat ve volbě řešení, pokud jich architektura nabízí několik. Příkladem může být to, jak by spolu, v případě vývoje systému s architekturou mikroslužeb, měly jednotlivé mikroslužby komunikovat. Možností je více, ať už se jedná o synchronní komunikaci přes REST rozhraní, která je na implementaci jednoduchá, či pouze asynchronní komunikace pomocí front zpráv pro větší výkonnost a škálovatelnost, ale zároveň přidanou komplexitu při vývoji a provozu[31].

4.2 Typy architektur

Při analýze architektur rozlišujeme dva základní typy architektur. Jedná se o architektury monolitické a distribuované. V následujících kapitolách jsou popsány rozdíly mezi monolitickými a distribuovanými architekturami. Následuje popis a srovnání několika aktuálně nejpoužívanějších typů softwarových architektur. Je popsána jejich topologie, výhody, nevýhody a pro jaké účely se dané architektury nejlépe hodí.

4.2.1 Monolitické architektury

Monolitické architektury, často označovány jako konvenční, jsou takové architektury, kde všechny komponenty systému ve výsledku organizujeme do jednoho spustitelného artefaktu[32].

4.2.2 Distribuované architektury

Oproti tomu, distribuované architektury, jsou takové, ve kterých je výsledná aplikace složena z několika samostatně fungujících prvků. Jednotlivé prvky jsou zpravidla rozděleny do logických celků dle jejich funkčnosti a tyto celky spolu navzájem synchronně nebo asynchronně komunikují, zpravidla skrze síť. Výsledná aplikace pak může být spuštěna na několika strojích, kdy každá komponenta může běžet na jiném stroji[32].

4.2.3 Srovnání monolitických a distribuovaných architektur

Rozdíly mezi monolitickými a distribuovanými architekturami jsou velké. Monolitické architektury mají velkou výhodu v tom, že jsou mnohem jednodušší, srozumitelnější a rychlejší na vývoj. Jejich provoz je většinou levnější a jednodušší. Distribuované architektury jsou velmi populární v posledních letech a to zejména díky rozvoji cloudových

technologií, tlaku na škálovatelnost a dostupnost. Škálovatelnost a dostupnost je jednou z hlavních výhod distribuovaných architektur, neboť aplikace není omezena na výkon jednoho stroje, ale může běžet na několika strojích. Na druhou stranu distribuované architektury s sebou přináší velkou komplexitu navíc, tedy problémy s komunikací mezi komponentami, mnohem složitější údržbou, verzováním jednotlivých komponent a komunikací. Velkým problémem také může být, pokud je nutná implementace databázových transakcí napříč více komponentami. Z těchto důvodů je třeba si dopředu rozmyslet, jestli je přidaná komplexita díky použití distribuovaného typu architektury oprávněná a nezbytná, či nikoliv[31].

- **Monolitické architektury**
  - **Výhody**
    - jednoduchost na vývoj
    - jednoduchý provoz
    - snadné testování napříč aplikací
  - **Nevýhody**
    - omezená škálovatelnost
    - špatné paralelní zpracování
    - velká složitost vývoje v případě, že je aplikace velmi komplexní
    - složitý přechod na nové standardy a technologie v případě, že je chceme změnit v jednotlivých částech - je nutné zasáhnout do celé aplikace
- **Distribuované architektury**
  - **Výhody**
    - velmi dobrá škálovatelnost
    - snazší zajištění vysokého procenta času dostupnosti
    - snadné paralelní zpracování
    - jednodušší testování jednotlivých komponent, které jsou menší, než v případě monolitické architektury
  - **Nevýhody**
    - velká přidaná komplexita při vývoji
    - dražší a složitější provoz
    - složitější testování celého systému - např. složité simulování výpadků komunikace, nedostupnosti, atd.

4.3 Monolitické architektury

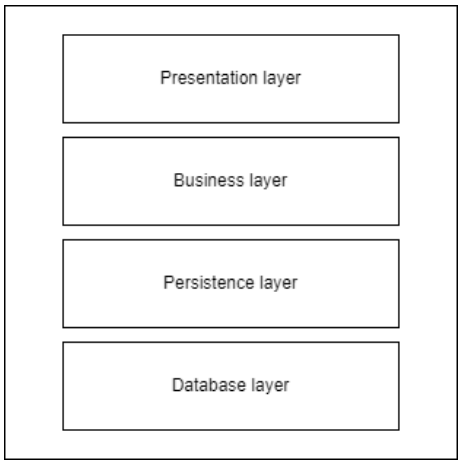
Monolitické architektury, jsou takové architektury, kde všechny komponenty systému ve výsledku organizujeme do jednoho spustitelného artefaktu viz kapitola 4.2.

4.3.1 Vrstevnatá architektura

Vrstevnatá architektura je jedna z nejjednodušších, nejpopulárnějších a nejznámějších typů architektur. Někdy se nazývá i n-vrstvá architektura. Její název odpovídá tomu, jak je řešení strukturováno. Celé řešení je rozděleno do několika horizontálních vrstev. Jednotlivé vrstvy jsou jsou definovány tím, jakou mají roli a zodpovědnost. Jedná se o typický příklad monolitické architektury.

Oddělením vrstev vznikají izolované celky, které spolu komunikují pouze skrze definované rozhraní. Tato vlastnost poskytuje výhodu v tom, že je možné relativně jednoduše jednu z vrstev nahradit jiným řešením splňujícím stejnou funkčnost, neboť vazby mezi vrstvami jsou pevně dány jejich rozhraním[33]. Další výhodou je oddělení zodpovědnosti

jednotlivých vrstev. V každé vrstvě je totiž možné jasně určit, co jsou její povinnosti, jaká data potřebuje a co s nimi může dělat. Zároveň lze na každé vrstvě kontrolovat, jestli je daná operace validní a neporuší konzistenci. Topologii vrstevnaté architektury je možné vidět na obrázku 4.1.



Obrázek 4.1. Topologie vrstevnaté architektury[31]

Pokud se podíváme na samotné vrstvy a jejich členění, pak jejich počet není z definice omezen. Zpravidla se však jedná o čtyři vrstvy. Pro čtyřvrstvé členění jsou vrstvy následující: prezentační vrstva, vrstva aplikační a business logiky, doménová vrstva, databázová vrstva[31].

Prezentační vrstva je zodpovědná za interakci s uživatelem a zobrazení dat. Vrstva aplikační a business logiky je zodpovědná za samotnou logiku, kterou aplikace má. Zpravidla se jedná o to, že tato vrstva naplňuje funkční požadavky na chování celého systému. Doménová vrstva je zodpovědná za reprezentaci a strukturu doménových dat, stejně jako za ověřování, že jsou dodržována omezení, která jsou na data kladena. Poslední vrstvou je databázová vrstva. Databázová vrstva je ve většině případů samotná databáze a zajišťuje perzistentní uložení dat[34].

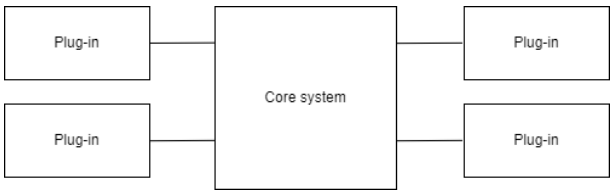
Častým fenoménem při použití vrstevnaté architektury je v rámci izolace a oddělení zodpovědnosti to, že každá z vrstev může komunikovat pouze tak, že data získává od vrstvy pod sebou a výsledky předává vrstvě nad sebou. Díky tomu se pak nestane, že by např. prezentační vrstva komunikovala přímo s databázovou vrstvou a naopak. Zajistíme tím to, že pokud např. uděláme zásah do databázové vrstvy, pak se změna nedotkne jiné vrstvy, než vrstvy doménové[31].

- **Výhody**
  - jednoduchost – jak na vývoj, tak na pochopení
  - izolace vrstev a oddělení zodpovědnosti
  - možnost nahradit kompletně jednu vrstvu, bez dopadu na další, pokud je zachováno rozhraní
- **Nevýhody**
  - jednoduchá oprava může mít dopad do všech vrstev
  - při použití mnoha vrstev zhoršený výkon
  - horší testovatelnost – v případě jednoduché změny se musí přetestovat celá aplikace

4.3.2 Mikrokernel architektura

Mikrokernel architektura je jedna z populárních monolitických architektur. Někdy je nazývána i jako architektura zásuvných modulů (plug-in). Její název odpovídá tomu, jak je řešení strukturováno. Celé řešení je rozděleno na mikrokernel, tedy hlavní komponentu, která je schopna fungovat samostatně a zajišťuje základní funkcionalitu celého systému. Další částí jsou pak dříve zmíněné zásuvné moduly, které zajišťují modulární rozšiřitelnost funkcionality celého řešení[35].

Jedná se o oblíbenou architekturu, hlavně v oblastech, kdy je cílem mít velmi modulární architekturu s velkým počtem zásuvných modulů a tím pádem možností aplikaci libovolně rozšiřovat. Jedním z příkladů použití mohou být aplikace pro vývoj software typu Eclipse, nebo Intellij IDEA, které umožňují vytvářet a instalovat doplňky, které mohou původní aplikaci libovolně rozšířit a poskytnout tak novou funkcionalitu i po vydání samotného softwaru. Topologii mikrokernel architektury je možné vidět na obrázku 4.2.



Obrázek 4.2. Topologie mikrokernel architektury

Mikrokernel, tedy hlavní komponenta, zpravidla zajišťuje pouze minimální nutnou funkcionalitu aplikace. Zároveň je zodpovědná za poskytnutí rozhraní pro zásuvné moduly, jejich správu, komunikaci a zabezpečení.

Zásuvné moduly jsou samostatné komponenty, které zpravidla zajišťují vylepšení nebo rozšíření funkcionality původní aplikace. Zpravidla jsou na sobě nezávislé a mohou být vůči sobě úplně transparentní. Zásuvné moduly komunikují s hlavním mikrokernelem pomocí zpráv ve formátu XML, JSON, nebo jiného rozhraní, dle konkrétní volby.

Díky tomu, že je možné přidávat libovolné zásuvné moduly, je tato architektura oblíbena pro svojí vysokou modularitu a rozšiřitelnost i poté, co byla aplikace sestavena. Zároveň se jedná o relativně jednoduchou architekturu, která je díky tomu snadná na pochopení i na vývoj[31].

- **Výhody**
  - modularita, rozšiřitelnost
  - testovatelnost – můžeme testovat jednotlivé moduly
  - v případě správné implementace je jednoduché vytvářet další zásuvné moduly
- **Nevýhody**
  - může být složité zajistit bezpečnost v případě použití doplňků třetích stran
  - špatná škálovatelnost
  - počáteční návrh mikrokernelu a způsobu komunikace se zásuvnými moduly může být náročnější

4.4 Distribuované architektury

Distribuované architektury, jsou takové, ve kterých je výsledná aplikace složena z několika samostatně fungujících prvků. Výsledná aplikace pak může být spuštěna na několika strojích, kdy každá komponenta může běžet na jiném stroji viz kapitola 4.2.

4.4.1 Architektura orientovaná na služby

Architektura orientovaná na služby je jedna z nejpopulárnějších distribuovaných architektur. Základním rysem této architektury je to, že aplikace je rozdělena do několika velkých logických celků, které zpravidla sdílejí jednu databázi a mohou sdílet i jedno uživatelské rozhraní[35].

To, že se jedná o distribuovanou architekturu, s sebou přináší obecně výhody jako je lepší výkonnost, snazší testovatelnost a přidanou flexibilitu. Na druhou stranu se ale nejedná o složitou architekturu a na poměry ostatních distribuovaných architektur se jedná o jednu z nejjednodušších.

Architekturu orientovanou na služby můžeme rozdělit do několika částí. První z nich je uživatelské rozhraní. Jedná se zpravidla o samostatnou aplikaci, která ke svému chodu využívá rozhraní několika služeb a data z nich konsoliduje a zobrazuje uživatelům.

Další částí jsou samotné služby. Služby jsou děleny dle jejich domény a logiky tak, aby každá služba zajišťovala jednu oblast funkcionalit a pokud možno aby navzájem nebyly moc provázány. Díky tomu jsou služby relativně veliké a je jich celkově většinou několik jednotek. Samotné služby jsou pak interně většinou navrhnuty jako vrstevnaté aplikace.

Poslední částí je databáze. Databáze je zpravidla společná pro všechny služby. Databáze je pak většinou logicky dělena podle jednotlivých služeb a je nutné zajistit to, aby nedocházelo ke kolizím mezi jednotlivými službami[31].

Celkově je tato architektura velmi oblíbená. Vede k tomu několik důvodů. Vzhledem k hrubému dělení služeb podle domén se tato architektura velmi hodí v případě, že je použit doménově řízený návrh aplikace. Zároveň, jak již bylo řečeno dříve, tato architektura poskytuje většinu výhod distribuovaných typů architektur bez toho, aby příliš narůstala komplexita celého řešení a jeho správy, stejně jako se lze vyvarovat problémům, které se vyskytují např. u mikroslužeb viz kapitola 4.4.3, kde je třeba řešit častěji například databázové transakce napříč mikroslužbami, nebo složitější nasazení a provoz[35].

Slabinou této architektury může být zejména centrální sdílená databáze. Jedná se jednak o úzké hrdlo a zároveň při změnách databázového schématu můžeme ovlivnit a potenciálně narušit všechny služby najednou, což je nežádoucí. Hrubé dělení služeb podle jednotlivých domén pak může u velkých aplikací, kde jsou i jednotlivé služby samy o sobě velmi komplexní, ztěžovat jejich rozšiřitelnost, modularitu a testovatelnost[31].

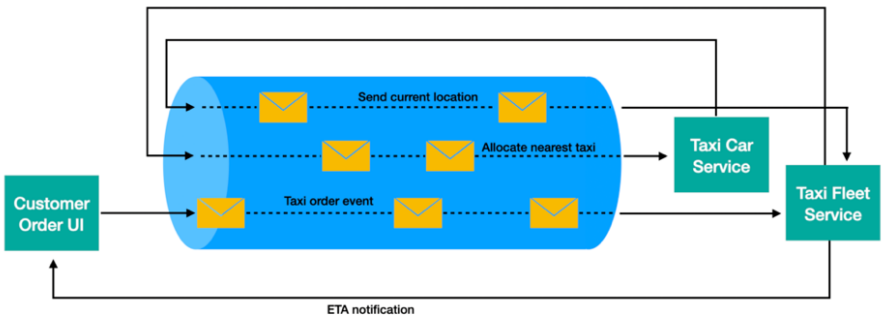
- **Výhody**
  - modularita, snazší rozšiřitelnost
  - testovatelnost – můžeme testovat jednotlivé služby samostatně
  - možnost nasazovat služby nezávisle na sobě
  - na poměry distribuovaných architektur nízká složitost a tím pádem i náklady na vývoj a údržbu
- **Nevýhody**
  - pro velmi velké aplikace, kde jsou služby veliké, náročná údržba a testovatelnost
  - přidaná komplexita oproti monolitickým architekturám
  - úzké hrdlo v případě sdílené databáze

4.4.2 Událostmi řízená architektura

Událostmi řízená architektura je dalším typem populárních distribuovaných architektur. Její název naznačuje, že její chování je propojeno s událostmi. Událostí je například uživatelská akce, či obdržení dat a podobně. Architektura je postavena tak, že po vzniku

události dojde k jejímu postupnému zpracování různými komponentami napříč systémem. Každá komponenta na událost buď zareaguje, či nikoliv a pošle jí ke zpracování dále, dokud není kompletně zpracována[31].

Událostmi řízená architektura se skládá z několika základních komponent. Jedná se o událost (*event*), což je zpravidla určitá uživatelská akce, ale může se např. jednat i o zprávu o změně stavu některého z okolních systémů, či časovou událost, apod. Tato událost následně putuje mezi komponentami zpracovávajícími události (*event processor*). Komponenty zpracovávající události jsou zpravidla samostatné komponenty, které umí události přijímat a provádět nad nimi konkrétní definované akce. Následně, pokud je to potřeba, je událost zaslána další komponentě zpracovávající události, dokud nejsou nad událostí provedeny veškeré potřebné akce[31]. Základní schéma topologie architektury řízené událostmi je zobrazeno na obrázku 4.3.



Obrázek 4.3. Topologie událostmi řízené architektury[36]

Důležitou vlastností událostmi řízené architektury je to, že jednotlivé komponenty zpracovávající události jsou navzájem propojeny asynchronním rozhraním, zpravidla nějakým typem zasílání zpráv, jako např. Apache Kafka nebo RabbitMQ. Toto asynchronní rozhraní poskytuje prostor pro spolehlivost, rozšiřitelnost, modularitu a škálovatelnost. Spolehlivost je dána tím, že software pro zasílání zpráv zpravidla poskytuje možnost perzistence zpráv, tedy informace nejsou ztraceny ani v případě výpadku jedné z komponent. Rozšiřitelnost a modularita je dána tím, že je vždy možné připojit další komponentu zpracovávající události, či existující komponentu odebrat. Škálovatelnost je pak další ze silných stránek této architektury, neboť v případě správného návrhu může být komponenta spuštěna vícekrát, tedy je možné v případě zvýšené zátěže spustit vícekrát stejnou komponentu a tím pádem zvětšit propustnost celým systémem. Škálovatelnosti samozřejmě pomáhá i samotné asynchronní rozhraní a fronty zpráv, neboť okolní komponenty nejsou závislé na tom, aby musely čekat na to, až bude konkrétní událost zpracována. Principy a prvky této architektury se též často používají jako součást jiných architektur[31].

Nevýhodou této architektury je zejména velmi složité testování chování napříč celým systémem. To je dáno tím, že celý systém není deterministický, případně existuje velké množství možných cest, kterými může událost projít. Z tohoto důvodu je velmi složité simulovat všechny možné situace a podmínky. Další značnou nevýhodou je relativní složitost architektury, kdy je nutné na rozdíl od monolitických typů architektur implementovat několik komponent a infrastrukturu zajišťující asynchronní rozhraní[31].

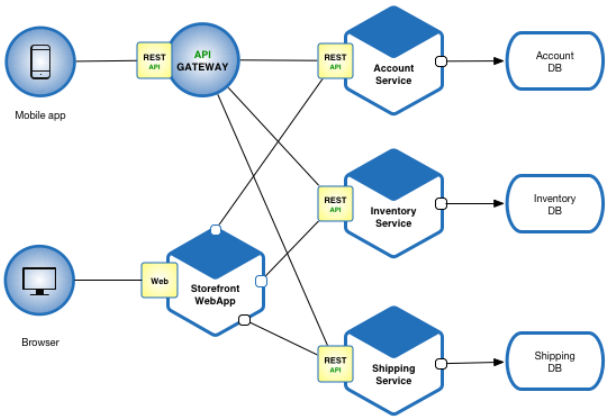
- **Výhody**
  - modularita, velmi snadná rozšiřitelnost
  - škálovatelnost

- spolehlivost

- **Nevýhody**
  - složité testování chování celého systému – chování je nedeterministické
  - složitější nasazení, správa a provoz
  - složité zajištění zpracování událostí v pořadí, ve kterém přišly (pokud je vyžadováno)

4.4.3 Architektura mikroslužeb

Architektura mikroslužeb je v poslední době velmi oblíbená distribuovaná architektura. Jedná se o architekturu, kdy je řešení děleno na velké množství relativně malých komponent. Architektura mikroslužeb vyniká svou škálovatelností, výkonností a modularitou. Základní schéma topologie architektury mikroslužeb je možné vidět na obrázku 4.4



Obrázek 4.4. Topologie architektury mikroslužeb[37]

Architektura je sestavena z několika prvků. Sjednocující vrstvou je nepovinná vrstva aplikačního rozhraní (*API Layer*). Vrstva aplikačního rozhraní, pokud je použita, zajišťuje rozhraní mezi těmi, kdo systém používají a samotným systémem. Tato vrstva slouží k přesměrovávání požadavků, může zde být implementován systém balancování požadavků napříč instancemi mikroslužeb, sledování metrik, nebo například zabezpečení[31].

Dále jsou v architektuře přítomné samotné mikroslužby. Mikroslužby jsou samostatné komponenty, které by měly být co nejvíce granulární a v rámci aplikace jsou zpravidla rozděleny podle domén, případně samostatných bloků funkcionalit. Důležitým rozdílem oproti architektuře orientované na služby je nejen velikost jednotlivých služeb, ale hlavně to, že každá mikroslužba by měla být schopna fungovat samostatně, tedy služby spolu nesdílí ani databázi a každá mikroslužba by měla mít databázi svojí[31].

Pojem mikroslužeb také nutně neznamená, že komponenty jsou malé, malé jsou z tradičního pohledu. Granularita mikroslužeb je větší, než v případě architektury orientované na služby, ale zároveň by měla být taková, aby bylo například potřeba co nejméně provádět databázové transakce napříč více mikroslužbami. Databázové transakce napříč mikroslužbami jsou totiž jednou z hlavních nevýhod této architektury, neboť je nutné tyto transakce navzájem koordinovat.

Komunikace mezi mikroslužbami je většinou asynchronní a velmi často se k ní používá komunikace pomocí zasílání zpráv. Díky tomu, že jsou na sobě mikroslužby nezávislé, je možné mít spuštěno více instancí stejné mikroslužby nezávisle na sobě i nezávisle na

ostatních mikroslužbách a tím pádem velmi dobře řídit škálování a celkovou výkonnost celého systému.

Díky rostoucí popularitě virtualizace a cloudových technologií došlo k velkému růstu nástrojů, bez kterých by nebylo možné architekturu mikroslužeb efektivně provozovat. Dnešní nástroje umožňují jednoduše sestavovat, nasazovat, provozovat a orchestrovat velké množství mikroslužeb. Architektura mikroslužeb s využitím těchto technologií počítá a nepřímo na nich závisí. To je také důvod, proč vznikla až s nástupem těchto technologií. Díky jejich efektivnímu využití jsou pak systémy postavené na architektuře mikroslužeb výkonné, spolehlivé a škálovatelné. To je hlavním důvodem obrovské popularity této architektury v poslední době a její největší přidanou hodnotou.

Velkou nevýhodou této architektury je její složitost. Díky velkému množství různých mikroslužeb dochází k přidané komplexitě v prakticky všech oblastech. Jedná se o nutnost vyřešit problémy s databázovými transakcemi napříč mikroslužbami, pokud jsou vyžadovány. Další problémy pramení z testování, kde stejně jako u událostmi řízené architektury je velmi náročné simulovat chování napříč systémem. Další přidanou komplexitou je nutnost používat a provozovat velké množství nástrojů, které provoz takto vysokého počtu komponent vůbec umožňuje.

- **Výhody**
  - snadné testování mikroslužeb
  - velmi malé vazby mezi mikroslužbami navzájem
  - škálovatelnost
  - spolehlivost
  - velmi snadná rozšiřitelnost i pro velké systémy
  - výkonnost
- **Nevýhody**
  - složité zajišťování transakcí napříč mikroslužbami
  - složitost nasazení a provozu
  - složité testování chování celého systému

4.5 Návrh architektury

V této kapitole je popsán návrh konkrétní architektury vhodné pro tento systém. Návrh je ovlivněn konkrétními požadavky kladenými na systém a jeho architekturu, stejně jako částí již existujícího systému.

4.5.1 Požadavky na architekturu

Vzhledem k tomu, že návrh a volba architektury má významný vliv na vlastnosti celého systému, je nutné před tím, než dojde k návrhu architektury znát vlastnosti a požadavky, které jsou od chování systému požadovány a očekávány. V návaznosti na sběr a analýzu požadavků, které jsou detailně popsány v kapitole 3, je níže seznam požadavků, jejichž naplnění může významně ovlivnit zvolený návrh architektury. Jedná se o požadavky týkající se očekávaného počtu uživatelů, plánovaných rozšíření, dostupnosti aplikace, či objemu sledovaných dat.

- **FR01 – Typ sbíraných dat**

  Řešení umožní sběr dat o srdečním tepu, počtu kroků, pohybové intenzitě a kvalitě spánku. Pokud to daná platforma výrobce umožní, je vyžadováno sbírat data s minutovou granularitou.
- **FR02 – Seznam výrobců a platforem pro sběr dat**

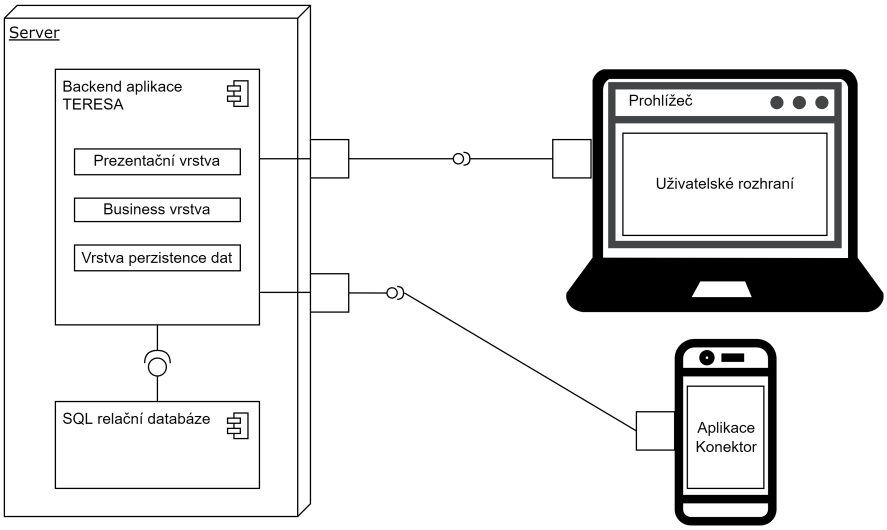
Řešení umožní sběr dat z platforem vybraných výrobců – Fitbit, Garmin.

- **FR06 – Sběr dat poskytovaných aplikací Konektor v novém řešení**
  Řešení rozšíří, případně upraví stávající logiku sběru dat poskytovaných aplikací Konektor tak, aby byla součástí nově vzniklého řešení.
- **NFR01 – Počet uživatelů**
  Řešení musí být schopné zpracovat a archivovat kompletní denní data o srdečním tepu, počtu kroků, pohybové intenzitě a kvalitě spánku od minimálně 500 uživatelů. *Poznámka – je očekáváno, že uživatelé budou svá data synchronizovat 1x denně zpravidla ve večerních hodinách.*
- **NFR02 – Rozšiřitelnost o nové typy dat**
  Řešení musí být jednoduše rozšiřitelné o další typy sbíraných dat.
- **NFR03 – Rozšiřitelnost o platformy dalších výrobců**
  Řešení musí být jednoduše rozšiřitelné o platformy dalších výrobců, ze kterých budou data sbírána. *Poznámka – v budoucnosti je očekáváno, že bude systém rozšířen o sběr z platforem dalších výrobců.*
- **NFR05 – Částečná dostupnost**
  Řešení musí být schopné pracovat i v případě, že sběr dat z jedné platformy bude částečně nedostupný – např. v případě technické odstávky, atd.
- **NFR06 – Stabilita**
  Řešení nesmí ztratit data ani v případě, kdy dojde k výpadkům některých částí aplikace, jako je centrální databáze, připojení k internetu, pád aplikace, atd.

4.5.2 Stávající architektura aplikace TERESA

Funkčnost aplikace TERESA je detailně popsána v kapitole 3.1. V této podkapitole je popsán stávající stav z architektonického pohledu. Jedná se o jednoduchou vrstevnatou monolitickou aplikaci. Pro perzistenci dat je využita SQL relační databáze. Aplikace má vystavené uživatelské rozhraní pro práci v prohlížeči. Druhým vystaveným rozhraním je rozhraní pro komunikaci s aplikací Konektor, která zajišťuje sběr dat pro zařízení od výrobce Xiaomi.

Architektura je zobrazena na obrázku 4.5.



Obrázek 4.5. Stávající architektura aplikace TERESA

Při volbě architektury bylo nutné rozhodnout, zda pouze nerozšířit stávající architekturu popsanou v kapitole 4.5.2. Vzhledem k tomu, že v době začátku psaní této práce byla původní aplikace TERESA poměrně malá, byl prostor i k velkým změnám v celkové architektuře, zejména kvůli benefitům volby správné architektury do budoucna. Z těchto důvodů bylo rozhodnuto, že bude architektura navrhnuta nezávisle na stávajícím řešení a stávající řešení bude pokud možno v co největší míře přepoužito.

4.5.3 Volba monolitické versus distribuované architektury

Na základě analýzy různých typů architektur z kapitoly 4.2 a na základě seznamu požadavků, které ovlivní zvolená architektura, viz předchozí kapitola 4.5.1, bylo nutné zvolit vhodnou architekturu řešení, jež bude v rámci této práce implementována.

První částí volby je, zdali zvolit monolitický, nebo distribuovaný typ architektury. Rozdílem mezi monolitickou a distribuovanou architekturou je, kromě jiného, zejména výkonnost a škálovatelnost celého řešení.

Dle požadavku NFR01 je očekáváno relativně velké zatížení celého systému. Konkrétně se jedná o požadavek na zpracování dat od minimálně 500 uživatelů s tím, že je očekáváno, že uživatelé budou svá data synchronizovat převážně ve večerních hodinách. Pokud je vzato v potaz očekávání sběru 4 typů dat (srdeční tep, počet kroků, pohybová intenzita, kvalita spánku) a jejich granularita po minutách, pak vychází, že dle očekávání přijde denně minimálně 2 880 000 informací o stavu pacientů. Zároveň je očekáváno, že většina informací dorazí během večera, kdy budou pacienti svá zařízení synchronizovat.

```
      1440     *           4           *     500     = 2 880 000
(minut denně) * (typů sbíraných dat) * (uživatelů) = informací denně
```

V závislosti na tomto objemu informací, které musí umět řešení denně zpracovat, asymetrii zatížení celého řešení napříč dnem a předpokládaným růstem do budoucna, došlo k rozhodnutí, že bude zvolen distribuovaný typ architektury. Výhodou bude její větší výkonnost a potenciální škálovatelnost, hlavní nevýhodou pak větší komplexita řešení a náročnější implementace.

4.5.4 Volba a návrh konkrétní architektury

Poté, co došlo k rozhodnutí, že bude zvolen distribuovaný typ architektury, bylo nutné zvolit a navrhnout konkrétní typ distribuované architektury pro toto konkrétní řešení.

Dle analýzy jednotlivých typů distribuovaných architektur z kapitoly 4.2 přicházely v úvahu následující typy architektur, případně jejich kombinace: architektura orientovaná na služby, událostmi řízená architektura a architektura mikroslužeb.

Z těchto třech architektur a jejich vlastností byla zvolena architektura mikroslužeb v kombinaci s událostmi řízeným asynchronním rozhraním mezi těmito mikroslužbami. Takto zvolená architektura respektuje a naplňuje požadavky, jež jsou na systém kladeny.

Architektura mikroslužeb byla zvolena pro její modularitu a tedy snadnou rozšiřitelnost, spolehlivost a výkonnost. Konkrétně modularita a snadná rozšiřitelnost respektuje požadavky FR02 a FR06, kdy již v základní implementaci je očekáván sběr dat ze tří různých platforem a zároveň požadavek NFR03, kdy je do budoucna očekáváno rozšíření o sběr dat z dalších platforem.

Výkonnost a spolehlivost architektury mikroslužeb respektuje požadavky NFR01, NFR05 a NFR06, které se týkají očekávané zátěže a nutnosti, aby řešení bylo stabilní a alespoň částečně dostupné i v případě, že dojde k výpadkům některých platforem, či částí této aplikace.

Propojení jednotlivých mikroslužeb je řešeno událostmi řízeným asynchronním rozhraním. Událostmi řízené rozhraní a komunikace přirozeně vyplývá z povahy dat, které systém bude zpracovávat. Bude se jednat o sběr sledovaných dat z jednotlivých platforem pro vybrané uživatele. Událostí v kontextu tohoto systému pak bude získání sledovaných dat z dané platformy pro daného uživatele a komunikace těchto dat k agregaci v systému TERESA.

Pro účely asynchronního rozhraní byla zvolena možnost komunikace skrze fronty zpráv. Toto řešení bylo zvoleno z několika důvodů. Komunikace pomocí zpráv totiž přináší řadu výhod zejména v oblasti minimalizace vazeb mezi jednotlivými mikroslužbami, škálovatelnosti a stability.

Minimalizace vazeb mezi jednotlivými mikroslužbami pak přispívá k lehké rozšiřitelnosti celého řešení a případné náhradě, kdy při zachování rozhraní mohou být zdrojové komponenty nahrazeny, nebo rozšířeny bez nutnosti upravovat konzumenty zpráv. Je tím tak podpořena realizace požadavku NFR03.

Řešení implementující asynchronní komunikaci pomocí front zpráv jsou schopna jednotlivé zprávy perzistovat. Perzistence zpráv a fakt, že se jedná o asynchronní komunikaci pomocí front zpráv má přímý vliv i na škálovatelnost a výkonnost celého řešení. V případě velkého množství získaných dat v krátkém čase, např. ve večerních hodinách, je totiž možné zprávy pouze zaslat do jednotlivých front a zpracovávat je postupně se zpožděním. To se týká i zachování funkčnosti systému v případě výpadků částí systému. V případě výpadku agregátoru nejsou data ztracena, neboť jsou zprocesována z front zpětně po opětovném nastolení funkcionality agregátoru. V případě výpadku některé mikroslužby sbírající data z některé platformy, pak není ovlivněn sběr dat z ostatních platforem. Díky těmto vlastnostem je podpořena realizace požadavků NFR01, NFR05 a NFR06.

Veškerá rozhraní jednotlivých mikroslužeb budou přístupná skrze reverzní proxy. Reverzní proxy je komponenta, která sdružuje, filtruje a směruje požadavky, které přichází z vnějšku systému. Slouží k několika účelům, těmi hlavními pro tento systém jsou sjednocení rozhraní celého systému, zabezpečení a možnost měnit jednotlivé části systému bez toho, aby tím byli ovlivněni konzumenti těchto služeb[38].

4.5.5 Finální návrh architektury

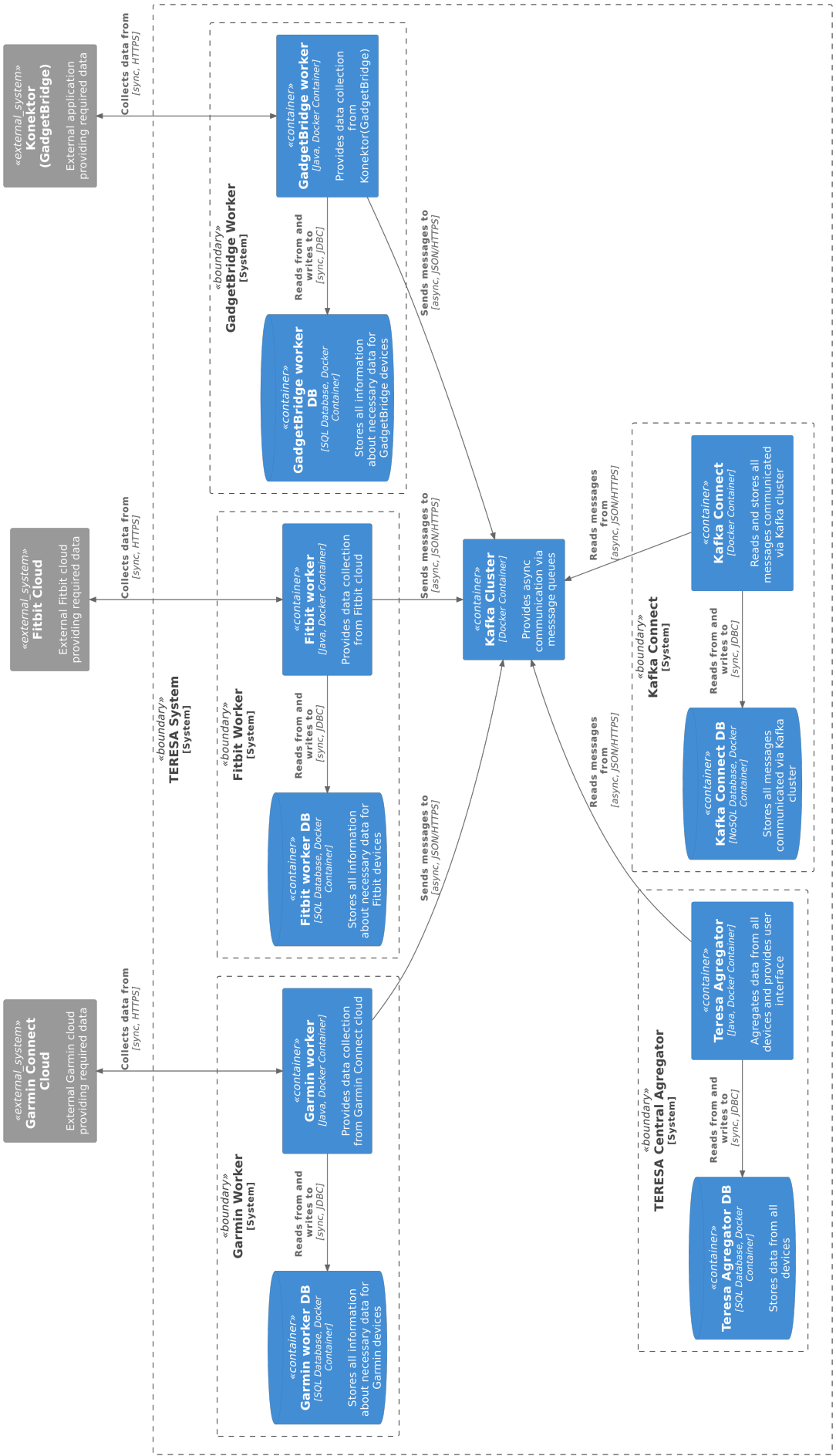
Na základě návrhu konkrétní architektury z kapitoly 4.5.4 byl vytvořen návrh architektury systému.

V rámci návrhu jsou následující mikroslužby:

- načítač dat z platformy Garmin
- načítač dat z platformy Fitbit
- načítač dat z platformy Xiaomi
- mikroslužba perzistující veškerou komunikaci pomocí zpráv
- centrální agregátor TERESA společně s uživatelským rozhraním
- reverzní proxy

Díky tomuto návrhu může být přepoužita i většina stávající aplikace TERESA. Změny pro stávající aplikaci TERESA jsou takové, že z aplikace bude vyjmut stávající načítač dat z aplikace Konektor, jež získává data ze zařízení od výrobce Xiaomi a bude nahrazen obecným načítačem dat, který bude schopen přijímat jednotná data ze všech platforem z nového asynchronního rozhraní pomocí zpráv.

Na následujícím diagramu 4.6 je zobrazen finální návrh architektury systému. Na diagramu není pro přehlednost záměrně zakreslena reverzní proxy.



Obrázek 4.6. Finální návrh architektury

Kapitola 5
Technologie

Po analýze požadavků a návrhu architektury je nutné zvolit vhodné technologie pro implementaci jednotlivých částí systému. Pro účely této práce byly zvoleny běžně dostupné technologie se širokou základnou uživatelů, standardně používané při vývoji komerčních aplikací. Dalším kritériem výběru bylo to, že by technologie měly být pro účely tohoto projektu pokud možno bezplatné. Volba byla také ovlivněna osobní zkušeností a znalostí těchto technologií autorem práce. Volba běžných technologií s velkou podporou komunity má velkou výhodu v tom, že je díky tomu docíleno snadnější udržovatelnosti celé aplikace.

5.1 Asynchronní komunikace pomocí front zpráv

Pro zprostředkování asynchronní komunikace pomocí front zpráv bylo nutné zvolit vhodné řešení. Softwarových řešení pro tento účel je mnoho. Je to dáno zejména stále rostoucí popularitou architektur postavených na mikroslužbách viz kapitola 4.4.3 a s tím spojenou rostoucí oblibou asynchronní komunikace pomocí zpráv. Komunikace pomocí zpráv je též využívána pro architektury řízené událostmi 4.4.2.

Mezi nejpopulárnější software v této kategorii patří RabbitMQ[1] a Apache Kafka[2].

5.1.1 RabbitMQ vs. Apache Kafka

RabbitMQ je open source distribuovaný zprostředkovatel pro zasílání zpráv. Má velkou uživatelskou základnu a tudíž i velkou škálu technologií, které podporuje. RabbitMQ je vysoce škálovatelný a výkonný. Poskytuje široké možnosti pro směrování jednotlivých zpráv napříč systémem. Distribuce zpráv je založena na principu front zpráv. Pokud tedy producent zprávy zprávu odešle, RabbitMQ se postará o to, že je zpráva doručena do fronty příjemce[39].

Apache Kafka je stejně jako RabbitMQ open source řešení. Opět se jedná o velice populární systém se širokou základnou a díky velkému zastoupení v komerční sféře i velké množství doplňků na enterprise úrovni kvality. Na rozdíl od RabbitMQ se označuje jako distribuovaná platforma pro streamování událostí. Opět se jedná o vysoce škálovatelný a výkonný systém schopný zpracovat i miliony zpráv za vteřinu. Na rozdíl od RabbitMQ funguje na principu *publish-subscribe*, tedy že je producent zpráv a následně příjemce zpráv. Příjemci se pak mohou zaregistrovat k odběru daných typů zpráv a Apache Kafka následně zajistí jejich doručení[39].

Dá se říci, že pro potřeby tohoto projektu jsou technologie srovnatelné a ani jedna nepřináší zásadní nevýhody oproti té druhé. Zvoleno bylo řešení Apache Kafka a to zejména díky její popularitě napříč všemi segmenty komerční sféry a také díky osobní zkušenosti autora práce s tímto softwarem.

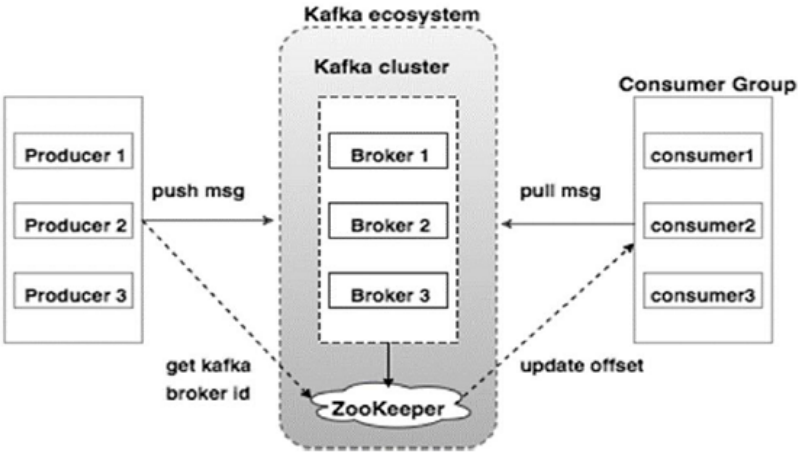
[1] https://www.rabbitmq.com/
[2] https://kafka.apache.org/

5.1.2 Apache Kafka

Apache Kafka je rozdělena na několik různých komponent tak, aby byl umožněn efektivní a spolehlivý provoz celého řešení. V následujícím seznamu jsou popsány jednotlivé části celého clusteru[40].

- **Kafka Broker** Broker je základní komponenta celého Kafka ekosystému. Jedná se o bezstavovou mikroslužbu zajišťující přijímání a odesílání zpráv. Zpravidla jich je spuštěno více instancí kvůli redundanci a rovnoměrnému rozdělení zátěže. Tyto instance pak společně tvoří celý cluster.
- **ZooKeeper**
  ZooKeeper je komponenta, která udržuje stav všech brokerů v clusteru a koordinuje jejich činnost. Je to zároveň komponenta, která pravidelně kontroluje stav a dostupnost jednotlivých brokerů a v případě přetížení, nebo nedostupnosti přesměrovává producenty a konzumenty na jiné brokery.
- **Producent**
  Producent je aplikace, která vytváří zprávy a zasílá je brokerům.
- **Konzument**
  Konzument je aplikace, která zprávy přijímá od brokerů a následně je zpracovává podle své logiky.

Strukturu Kafka clusteru je možné vidět na obrázku 5.1.

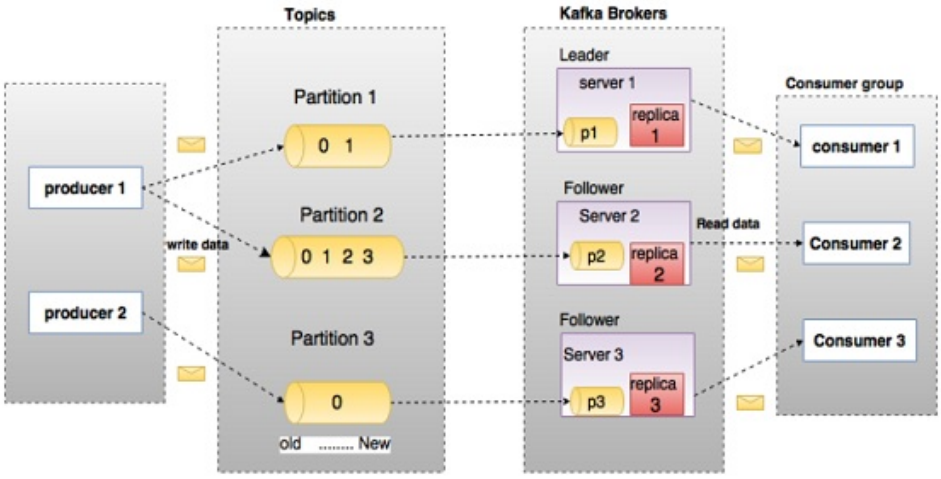


Obrázek 5.1. Struktura Kafka clusteru[40]

Jednotlivé fronty zpráv se označují jako *topic* a dělí se do více částí nazývaných *partition*.

- **Topic**
  Topic je označení v Kafka terminologii pro frontu zpráv. Topicy mohou být rozděleny do více partition, vždy ale obsahují alespoň 1 partition. Pořadí zpráv v topicu není zaručeno, zaručeno je pouze pořadí zpráv uvnitř jednotlivých partition
- **Partition**
  Partition je část topicu. Každý topic obsahuje alespoň 1 partition. Počet partition určuje, jaká je maximální paralelizace na straně konzumentů, neboť každá partition může být přidělena maximálně jednomu konzumentu.

Tato struktura je znázorněna na obrázku 5.2



Obrázek 5.2. Struktura front zpráv v Apache Kafka[41]

5.2 Načítače dat

Pro načítače dat bylo zvoleno řešení napsané v jazyce Java[3]. Java je open source objektově orientovaný programovací jazyk. Programovací jazyk Java může být použit pro široké spektrum aplikací, počínaje malými mobilními aplikacemi až po velké enterprise systémy. Uživatelská základna je obrovská, stejně jako počet doplňků a knihoven, které se dají použít.

Jazyk Java byl zvolen ve verzi Java 11, a to z toho důvodu, že v době vývoje projektu se jednalo o nejnovější verzi s tzv. dlouhodobou podporou. Dlouhodobá podpora je výhodná v tom, že výrobce dlouhodobě (roky) garantuje vydávání bezpečnostních záplat a oprav chyb. Pro standardní verze Javy bez dlouhodobé podpory jsou tyto záplaty vydávány pouze po dobu 6 měsíců. Systémy pak musí být často aktualizovány v rámci údržby.

Programovací jazyk Java byl zvolen zejména díky tomu, že se jedná o standard pro tento typ aplikací, v tomto jazyce již je napsána existující část aplikace TERESA, stejně jako díky osobní zkušeností autora práce s prací v tomto programovacím jazyce.

5.2.1 Spring Boot

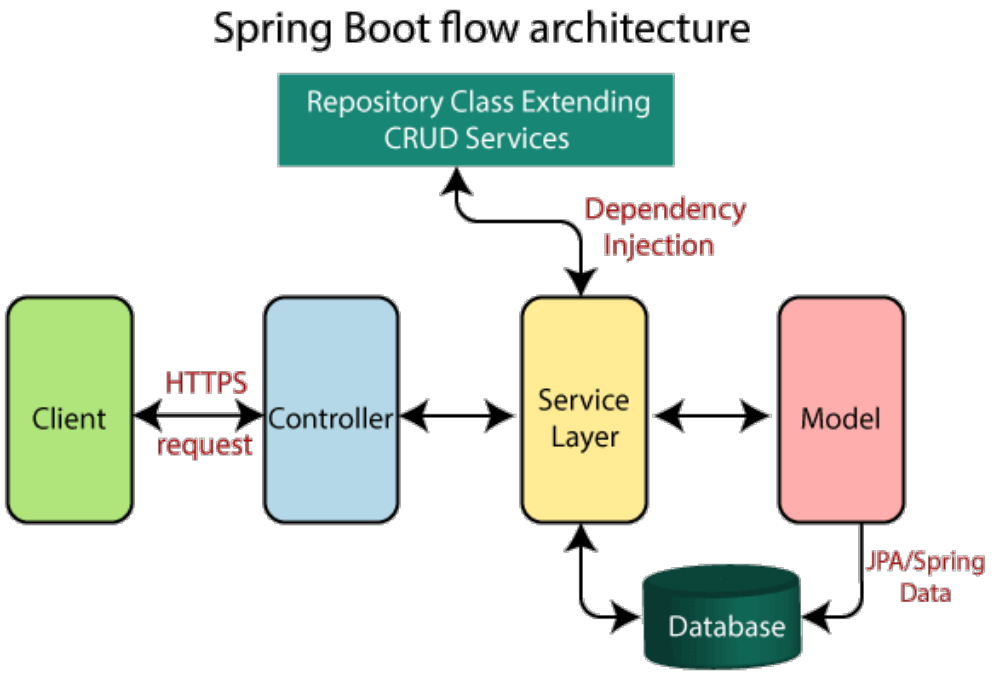
Spring Boot[4] je populární rozšíření známého a široce používaného frameworku Spring. Výhod použití Spring Boot oproti frameworku Spring je několik. Aplikace postavené nad Spring Boot totiž automaticky obsahují zabudovaný server, tudíž je mnohem snazší je spustit a provozovat, neboť není potřeba podstupovat proces nasazení WAR souborů na servery. Zatímco při použití samotného Springu musí uživatel vše ručně nakonfigurovat pro konkrétní aplikaci, Spring Boot nabízí přednastavenou konfiguraci, která je vhodná pro nejběžnější použití. Pro další urychlení vývoje je pro Spring Boot poskytnut nástroj Spring Initializr[5], pomocí kterého lze vybrat jaké knihovny mají být součástí aplikace a nástroj následně vygeneruje kostru aplikace i s těmito knihovnami.

Spring Boot, stejně jako Spring, poskytuje široké množství knihoven, které umožňují velmi snadné používání nástrojů třetích stran, tedy konektory k databázím, rozhraní

[3] https://www.java.com/en/
[4] https://spring.io/projects/spring-boot
[5] https://start.spring.io/

pro Apache Kafka, knihovny pro snadné testování, atd. Na obrázku 5.3 je možné vidět základní strukturu Spring Boot aplikací.

V rámci načítačů dat byla použita aktuální verze 2.5.2.



Obrázek 5.3. Struktura Spring Boot aplikace[42]

5.2.2 PostgreSQL relační databáze

Pro ukládání dat o pacientech, jejich zařízeních a dalších nezbytných informací nutných pro kolekci dat konkrétním načítačem, bylo nutné zvolit vhodnou databázi pro ukládání těchto dat.

Jako vhodná databáze byla zvolena relační databáze PostgreSQL[6]. Jedná se o populární open source databázi se širokou podporou. Vzhledem k tomu, že se neočekává, že by načítače dat potřebovaly složité databázové procedury, velké kvantity dat, ani žádné další náročné použití, nebylo potřeba volit databázi dle žádných specifických požadavků.

Relační databáze PostgreSQL byla tedy zvolena zejména díky tomu, že již je použita jako databáze v existující databázi TERESA. Tedy díky použití stejného typu databáze bude celý systém technologicky homogenní, co se týče použitých technologií. Databáze PostgreSQL je použita ve verzi 13.2.

5.2.3 Gradle

Pro automatické sestavování aplikace je použit nástroj Gradle[7]. Jedná se o multiplatformní a multijazykový nástroj. Při použití nástroje Gradle je sestavení aplikace rozděleno do několika úkolů, které jsou postupně provedeny a aplikace je tímto způsobem sestavena.

V rámci načítačů dat je použita verze 7.0.

[6] https://www.postgresql.org/
[7] https://gradle.org/

5.2.4 Napojení na Apache Kafka

Pro připojení ke clusteru Apache Kafka byla využita knihovna Spring for Apache Kafka[8], která výrazně zjednodušuje připojení aplikací ke Kafka clusteru. Knihovna je použita v aktuální verzi 2.7.6.

5.3 Mikroslužba perzistující veškerou komunikaci pomocí zpráv

Pro mikroslužbu perzistující veškerou komunikaci pomocí zpráv bylo nutné zvolit technologii pro implementaci řešení této logiky a následně databázi, do které by byl obsah zpráv perzistován.

5.3.1 Kafka Connect

Při analýze vhodných řešení, která by byla vhodná pro implementaci perzistence všech zpráv, byly na výběr dvě možnosti.

- návrh a implementace vlastní mikroslužby
- použití existujícího řešení Kafka Connect

Myšlenka za návrhem a implementací vlastní mikroslužby byla taková, že daná mikroslužba by se připojila jako konzument ke všem topicům a zprávy by perzistovala do vlastní databáze.

V rámci analýzy bylo zjištěno, že již existuje řešení Kafka Connect, které kromě jiného lze nakonfigurovat tak, aby perzistovalo všechny zprávy ze zvolených topiců do zvolených tabulek v databázi.

Vzhledem k tomu, že řešení Kafka Connect lze použít přesně dle požadavků, bylo zvoleno toto řešení namísto napsání vlastní mikroslužby. To zejména díky větší robustnosti a spolehlivosti tohoto řešení v porovnání s vlastním vývojem. Další výhodou použití tohoto řešení je absence vývoje nové mikroslužby a zlepšení udržovatelnosti celého systému v budoucnu díky široké podpoře celé komunity.

5.3.2 MongoDB dokumentová databáze

Pro perzistenci zpráv, které jsou zasílány do jednotlivých topiců a následně mají být perzistovány, bylo nutné zvolit vhodnou databázi. Na výběr bylo mezi standardní relační databází a některou z NoSQL databází. Vzhledem k tomu, že jednotlivé zprávy, které jsou zasílány skrze Apache Kafka jsou ve formátu JSON, byla jako vhodné řešení zvolena dokumentová databáze.

Vzhledem k tomu, že primárním cílem této databáze je perzistence dat pro zpětný audit a časový sled zpráv, byly požadavky na tuto databázi nenáročné. Díky osobní zkušenosti autora práce, nenáročnosti na zdroje a velké oblibě v komunitě a díky tomu velké podpoře, byla zvolena dokumentová databáze MongoDB[9]. Tato databáze ukládá dokumenty ve formátu JSON. Zprávy díky tomu nemusí být nijak konvertovány a mohou být rovnou perzistovány.

[8] https://spring.io/projects/spring-kafka
[9] https://www.mongodb.com/

5.4 Centrální agregátor TERESA

Existující aplikace TERESA je postavena na velmi podobných technologiích, jako ty, které byly zvoleny pro nově vytvářené mikroslužby. Aplikace je postavena na Spring Boot ve verzi 2.4.3. Programovacím jazykem je jazyk Java ve verzi 11. Databáze je použita PostgreSQL ve verzi 13.2.

Sestavování je na rozdíl od nových mikroslužeb prováděno pomocí Apache Maven[10] ve verzi 3. Apache Maven je obdobou sestavovacího nástroje Gradle. Jedná se o velmi známou technologii se širokou podporou a oblibou. Vzhledem k tomu, že sestavování pomocí Maven zde již bylo nakonfigurováno a Maven nemá oproti Gradle v případě tohoto projektu žádné zřejmé nevýhody, nebylo nutné stávající řešení měnit.

5.4.1 Framework Thymeleaf

Existující uživatelské rozhraní mikroslužby TERESA je napsáno ve frameworku Thymeleaf[11]. Jedná se o framework, který má za cíl usnadnit a urychlit vývoj webových aplikací. Základem frameworku jsou dynamické šablony, které se integrují se Spring MVC a jsou pak schopné zobrazovat zvolená data.

5.4.2 Napojení na Apache Kafka

Stejně jako v načítačích dat 5.2.4 je pro komunikaci s Kafka clusterem použita knihovna Spring for Apache Kafka ve verzi 2.7.6.

5.5 Reverzní proxy Nginx

Jako reverzní proxy je použita Nginx[12]. Jedná se o velmi oblíbený open source software. Řešení je schopné kromě reverzní proxy pracovat i jako webový server, případně balancovat zátěž. Výhodou tohoto řešení je vysoký výkon a nízké požadavky na zdroje. Nginx byla použita již v původní aplikaci, proto byla v rámci implementace pouze rozšířena o dodatečnou konfiguraci související s novými mikroslužbami.

5.6 Kontejnerizace, Docker

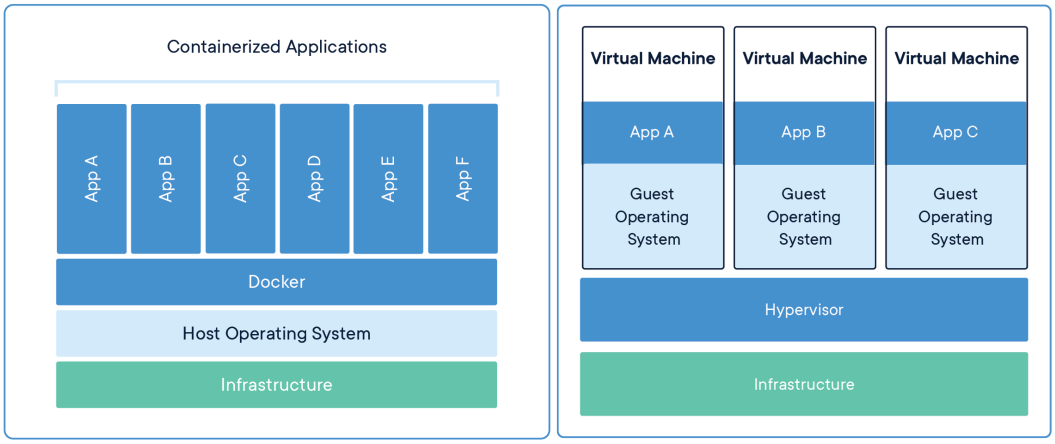
Celý systém obsahuje velké množství aplikací. Jedná se například o jednotlivé mikroslužby, komponenty Apache Kafka clusteru, databáze či reverzní proxy. Kvůli takto komplexní kombinaci se provoz celého systému stává náročným. Každá aplikace totiž potřebuje vlastní konfigurace, nastavení, specifické kroky nutné k jejímu spuštění apod. Velkého zjednodušení provozu systému lze docílit kontejnerizací jednotlivých komponent.

Pro kontejnerizaci byla zvolena platforma Docker[13]. Docker je populární open source platforma s cílem zajistit multiplatformní prostředí pro běh a správu kontejnerů. Samotný kontejner obsahuje pouze vybrané aplikace a soubory nutné pro jeho provoz. Na rozdíl od virtuálních strojů tedy neobsahuje operační systém a tudíž velké množství nadbytečných souborů. Kontejnery jsou tedy oproti virtuálním strojům velikostně menší, potřebují méně zdrojů a jsou tak levnější na provoz. Stále ale zajišťují výhody,

[10] https://maven.apache.org/
[11] https://www.thymeleaf.org/
[12] https://www.nginx.com/
[13] https://www.docker.com/

jako je izolace kontejnerů od hostitelského systému nebo izolaci napříč jednotlivými kontejnery, a tudíž zvýšenou bezpečnost.

Rozdíly mezi kontejnery a virtuálními stroji lze vidět na obrázku 5.4.



Obrázek 5.4. Kontejner versus virtuální stroj[43]

Konfigurace jednotlivých kontejnerů a jejich spuštění je prováděno pomocí nástroje docker compose[14]. Docker compose je nástroj, který umožňuje snadnou konfiguraci a běh aplikací sestávajících z většího počtu Docker kontejnerů. Konfigurace celého systému se pak provádí zápisem do souboru, který je ve struktuře YAML a zpravidla se jmenuje *docker-compose.yml*.

[14] https://docs.docker.com/compose/

Kapitola 6
Implementace

V této kapitole je detailně popsána implementace jednotlivých částí systému v souladu s navrženou architekturou.

6.1 Fronty a struktura zasílaných zpráv v rámci Kafka

V rámci implementace bylo nutné navrhnout a vytvořit členění zpráv do jednotlivých front (topiců) a navrhnout vhodný formát a strukturu zpráv, které do těchto front budou zasílány.

6.1.1 Fronty zpráv

Na základě funkčních požadavků vyplývajících z analýzy viz kapitola 3.3, konkrétně požadavku FR01, bylo navrhnuto několik druhů zpráv a jejich členění do jednotlivých front (topiců). Druhy zpráv, potažmo jejich členění do jednotlivých front, byly navrhnuty primárně dle doménové příslušnosti dat, která budou ve zprávách obsažena.

Zprávy jsou členěny do následujících front.

- **Activity**
  Fronta zpráv Activity s daty o činnostech uživatele vykonávaných v daném čase. Zpravidla se jedná o informaci, jestli pacient seděl, chodil, běhal či spal.
- **ActivityDLT**
  Fronta zpráv ActivityDLT je zkratkou pro *Activity Dead Letter Queue*. Jedná se o frontu zpráv, kam se zasílají zprávy, které se nepodařilo úspěšně zpracovat na straně konzumenta. Tato fronta je následně kontrolována administrátorem v rámci provozu aplikace. Pokud je to možné, mohou být zprávy znovu přehrány, případně je nutné opravit problém, který zamezuje jejich zpracování. Výhodou zasílání neúspěšně zpracovaných zpráv do *DLT* front je možnost mít přehled o tom, jestli a případně která data nebyla zpracována.
- **ActivitySummary**
  Fronta zpráv ActivitySummary obsahuje zprávy s daty o souhrnech pohybové aktivity.
- **ActivitySummaryDLT**
  Fronta zpráv ActivitySummaryDLT obsahuje neúspěšně zpracované zprávy z fronty ActivitySummary.
- **HeartRate**
  Fronta zpráv HeartRate obsahuje zprávy s daty o naměřené tepové frekvenci.
- **HeartRateDLT**
  Fronta zpráv HeartRateDLT obsahuje neúspěšně zpracované zprávy z fronty HeartRate.
- **SleepLevelTimeRange**
  Fronta zpráv SleepLevelTimeRange obsahuje zprávy s daty o kvalitě spánku v daném časovém intervalu. Zařízení totiž rozlišují více typů spánku jako je lehký, hluboký, REM, případně informaci o tom, že pacient byl v daném intervalu vzhůru.

- **SleepLevelTimeRangeDLT**
  Fronta zpráv SleepLevelTimeRangeDLT obsahuje neúspěšně zpracované zprávy z fronty SleepLevelTimeRange.
- **SleepSummary**
  Fronta zpráv SleepSummary obsahuje zprávy se souhrnem informací o spánku během daného dne. Zpravidla se jedná o délku a kvalitu spánku během jedné noci.
- **SleepSummaryDLT**
  Fronta zpráv SleepSummaryDLT obsahuje neúspěšně zpracované zprávy z fronty SleepSummary.
- **GarminRawRequest**
  Fronta zpráv GarminRawRequest obsahuje zprávy s celým tělem odpovědi vrácené externím rozhraním z Garmin Connect cloudu. Vytvoření této fronty je snadný způsob, jak si zachovat veškerou historii přijatých dat. Díky této frontě je možné zpětně zpracovávat data v případě, že v jejich zpracování byla zanesena chyba. Možnost zachování historie přijatých dat a zpětného zpracování bude zajištěna tím, že zprávy z této fronty budou perzistovány do dokumentové databáze MongoDB. Další výhodou je, že zpětně mohou být zpracována data, která dosud zpracovávána nebyla.
- **GarminRawRequestDLT**
  Fronta zpráv GarminRawRequestDLT obsahuje těla takových odpovědí vrácených externím rozhraním z Garmin Connect cloudu, která byla neúspěšně zpracována. Neúspěšné zpracování zpravidla může znamenat to, že odpověď neměla očekávanou strukturu, obsahovala nevalidní data, nebo se z jiného důvodu nepodařilo namapovat a rozdělit přijatá data do jednotlivých zpráv, případně se tyto zprávy nepodařilo odeslat do jednotlivých front.
  V rámci monitoringu aplikace je tedy nutné tyto fronty kontrolovat. V případě, že se při kontrole ve frontě objeví zpráva, je nutné se zaměřit na zjištění příčiny chyby. Může se např. jednat o změnu rozhraní nebo struktury dat, která může mít potenciálně dopad na sběr dat od všech pacientů využívajících danou platformu.
- **FitbitRawRequest**
  Fronta zpráv FitbitRawRequest obsahuje stejný typ zpráv jako fronta GarminRawRequest s tím rozdílem, že se jedná o těla odpovědí vrácených z Fitbit Cloudu.
- **FitbitRawRequestDLT**
  Fronta zpráv FitbitRawRequestDLT obsahuje stejný typ zpráv jako fronta GarminRawRequestDLT s tím rozdílem, že se jedná o neúspěšně zpracovaná těla zpráv vrácených z Fitbit cloudu.
- **GadgetBridgeRawRequest**
  Fronta zpráv GadgetBridgeRawRequest obsahuje stejný typ zpráv jako fronta GarminRawRequest s tím rozdílem, že se jedná o těla odpovědí vrácených z aplikace Konektor sbírající data z náramků Xiaomi pomocí platformy GadgetBridge.
- **GadgetBridgeRawRequestDLT**
  Fronta zpráv GadgetBridgeRawRequestDLT obsahuje stejný typ zpráv jako fronta GarminRawRequestDLT s tím rozdílem, že se jedná o neúspěšně zpracovaná těla zpráv vrácených z aplikace Konektor sbírající data z náramků Xiaomi pomocí platformy GadgetBridge.

6.1.2 Struktura zpráv

Struktura jednotlivých zpráv byla navrhnuta na základě analýzy dat, která jsou poskytována jednotlivými rozhraními. Po prostudování rozhraní byla vytvořena struktura zpráv obsahující parametry, které jsou poskytovány většinou platforem a jsou relevantní

pro tento projekt. V rámci analýzy bylo přihlíženo i k existujícímu schématu dat, která již byla aplikací TERESA sbírána skrze rozhraní z aplikace Konektor.

Cílem je mít granularitu zpráv nastavenou vždy na 1 minutu, pokud je to vhodné. Struktura zpráv je následující.

- **ActivityMessage**
  - datum a čas
  - počet kroků
  - intenzita aktivity
  - MET - metabolický ekvivalent[1]
  - typ aktivity Garmin
  - typ aktivity GadgetBridge
- **ActivitySummary**
  - datum a čas začátku aktivity
  - datum a čas konce aktivity
  - typ aktivity GadgetBridge
  - zeměpisná délka
  - zeměpisná šířka
  - zeměpisná výška
  - záznam trasy
  - textový souhrn aktivity
- **HeartRateMessage**
  - datum a čas
  - srdeční tep
- **SleepLevelTimeRangeMessage**
  - kalendářní datum, ke kterému se data o spánku vztahují
  - datum a čas začátku intervalu
  - datum a čas konce intervalu
  - úroveň spánku v tomto intervalu
- **SleepSummaryMessage**
  - kalendářní datum, ke kterému se data o spánku vztahují
  - datum a čas počátku spánku
  - datum a čas konce spánku
  - celková doba trvání
  - celková doba hlubokého spánku
  - celková doba lehkého spánku
  - celková doba spánku REM
  - celková doba, kdy byl pacient vzhůru
  - celková doba, kdy měření neprobíhalo
- **GarminRawRequestMessage**
  - tělo odpovědi vráceno externím rozhraním
- **FitbitRawRequestMessage**
  - tělo odpovědi vráceno externím rozhraním
- **GadgetBridgeRawRequestMessage**
  - tělo odpovědi vráceno externím rozhraním

Zprávy dále obsahují časovou značku zaslání do fronty a identifikátor pacienta, ke kterému daná data patří. To neplatí pouze pro zprávy, které obsahují pouze těla odpovědí. Identifikátor je následně použit pro účely spárování dat s jednotlivými pacienty na straně agregátoru TERESA. Formátem všech zpráv je JSON.

[1] https://en.wikipedia.org/wiki/Metabolic_equivalent_of_task

6.2 Načítač dat z platformy Garmin

Pro načítání dat z cloudu Garmin Connect byla v souladu s navrženou architekturou v kapitole 4.5.5 vyvinuta samostatná aplikace, dále označovaná jako Garmin Worker.

6.2.1 Získání přístupu k platformě Garmin Connect

Aby aplikace třetích stran mohly mít přístup k datům v rámci Garmin Connect cloudu, musí být tyto aplikace ověřeny společností Garmin a musí jim být zřízen přístup. K tomuto účelu slouží speciální registrační formulář[2], ve kterém je nutné vyplnit údaje o organizaci, způsobu zpracování dat a účel, pro který jsou data zpracovávána. Tato žádost je následně posouzena společností Garmin a pokud vyhovuje, je aplikaci zřízen testovací přístup.

Testovací přístup je omezen na množství dat, které je skrz něj možné stahovat a je určen pouze pro vývojové a testovací účely. Po úspěšné implementaci kolekce dat a splnění podmínek je možné zažádat o produkční přístup.

Podmínky pro získání produkčního přístupu jsou následující.

- implementace sběru alespoň jednoho typu poskytovaných dat
- umožnit uživatelům zrušení přístupu ke kolekci dat aplikaci třetí strany
- úspěšný sběr dat (bez chyb) paralelně alespoň ze dvou zařízení v posledních 24 hodinách

Zřízení přístupu k datům pro tuto aplikaci, stejně jako produkční přístup, probíhalo rychle a bez problémů. Společnost Garmin byla v rámci celého procesu proaktivní, dokonce byla projektu nabídnuta součinnost pro případ řešení chyb, kterou nebylo nakonec nutné využít.

6.2.2 Autorizace nových zařízení

Pro získání přístupu k datům konkrétního uživatele je nutné, aby daný uživatel autorizoval přístup této aplikace. Celý proces autorizace a následný přístup k datům uživatele je řešen pomocí protokolu OAuth ve verzi 1.0a[3].

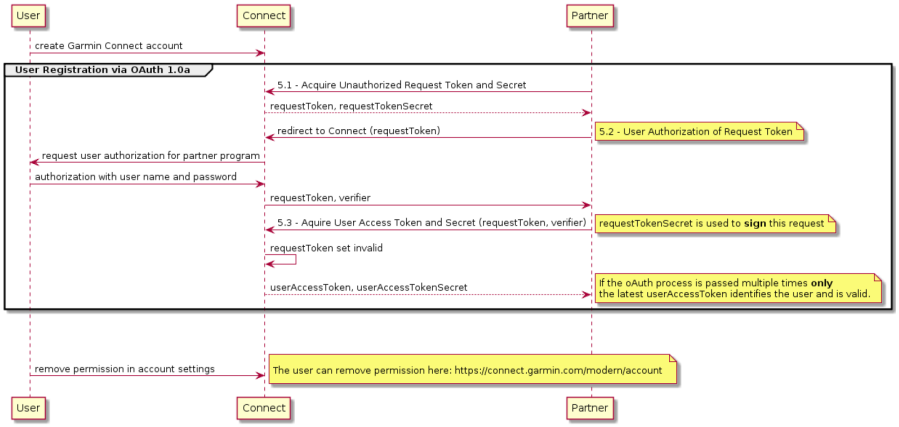
Výhodou protokolu OAuth je, že umožňuje aplikacím třetích stran přistupovat k citlivým datům uživatelů bez toho, aniž by uživatelé museli těmto aplikacím poskytovat své přihlašovací údaje.

Protokol určuje přesný průběh procesu autorizace, jehož výsledkem je tzv. UserAccessToken sloužící k autorizaci při komunikaci s Garmin Connect cloudem. Zároveň tento token využívá Garmin Connect jako identifikátor při zasílání dat.

Proces ve verzi protokolu 1.0a je následující.

- Aplikace Načítač získá tzv. RequestToken a RequestTokenSecret z Garmin Connect
- Aplikace přesměruje uživatele na stránku Garmin Connect
- Uživatel autorizuje přístup aplikaci Načítač přihlášením se do svého uživatelského účtu a potvrzením přístupu k datům
- Aplikace Načítač obdrží od Garmin Connect tzv. verifier
- Aplikace Načítač využije verifier k získání userAccessToken a userAccessTokenSecret, které následně může využívat pro přístup k datům

[2] https://www.garmin.com/en-US/forms/GarminConnectDeveloperAccess/
[3] https://oauth.net/core/1.0a/



Obrázek 6.1. OAuth 1.0a autorizační proces

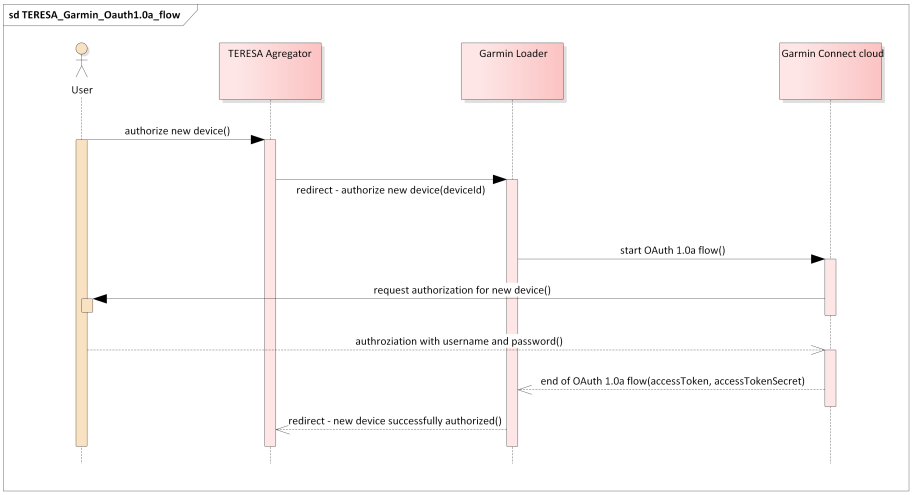
Celý proces, jakým je získán tento token, je znázorněn obrázku 6.1.

Aplikace Garmin Worker nemá žádné uživatelské rozhraní, ze kterého by mohl uživatel obsluhující celý systém TERESA zahájit tento proces. Proto vystavuje Garmin Worker rozhraní sloužící k nastartování a provedení OAuth autorizace.

Rozhraní přijímá jeden parametr, kterým je identifikátor pacienta v agregátoru TERESA. Jedná se o identifikátor pacienta, jež je následně zasílán ve všech zprávách s obdrženými daty a skrz který si data agregátor TERESA páruje s konkrétním pacientem.

Na toto rozhraní je uživatel přesměrován z uživatelského rozhraní agregátoru TERESA a aplikace Načítač provede celý proces OAuth 1.0a, kde po úspěšném obdržení a uložení přístupových tokenů přesměruje uživatele zpět do uživatelského rozhraní agregátoru TERESA. Aplikace Garmin Worker je tedy pro uživatele transparentní a uživatel sám s ní nemusí nikterak interagovat.

Celý proces autorizace přístupu a získání tokenů z pohledu systému TERESA je znázorněn na obrázku 6.2. Na obrázku je záměrně zjednodušen samotný proces výměny tokenů, který je detailně znázorněn na obrázku 6.1.



Obrázek 6.2. Získání přístupových tokenů z Garmin Connect v rámci systému TERESA

Vzhledem k tomu, že OAuth protokol ve verzi 1.0a je standardem, existuje velké množství knihoven, které slouží ke zjednodušení implementace jak na serverové, tak

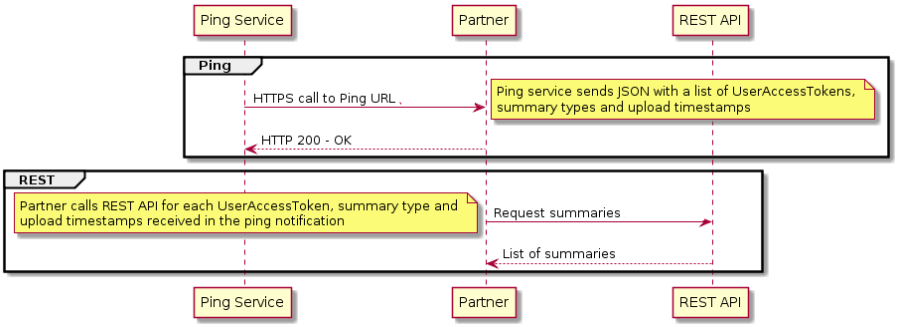
na klientské straně. V rámci aplikace Garmin Worker byla využita knihovna Google OAuth Client Library for Java[4] ve verzi 1.31.5.

6.2.3 Získávání a zpracování dat z Garmin Connect

Garmin Connect cloud je schopen poskytovat data vícero způsoby. Data je možné získat buď na vyžádání, nebo je Garmin Connect cloud schopen generovat notifikace o tom, že si uživatel synchronizoval své zařízení a jsou dostupná nová data.

Kvůli úspoře požadavků vznášených na Garmin Connect cloud je doporučeno data získávat na základě notifikací. Díky tomu jsou data získávána pouze pro zařízení, pro která jsou nová data dostupná. Další výhodou je to, že aplikace obdrží opakovaně až do úspěšného zpracování a to po dobu až 7 dnů. Tím pádem i v případě krátkodobého výpadku aplikace Garmin Worker jsou veškerá data stažena.

Garmin Connect cloud poskytuje 2 typy notifikací. Prvním typem je tzv. PING notifikace, kdy přijde na dané rozhraní (podle typu dat) zpráva se seznamem userAccessTokenů (a tedy uživatelů), pro která jsou nová data dostupná. Aplikace si pak může na základě této notifikace data vyžádat. Proces je znázorněn na obrázku 6.3.



Obrázek 6.3. Notifikace o dostupnosti nových dat z Garmin Connect

Druhým typem notifikací je tzv. PUSH notifikace. Jedná se o HTTP POST požadavek na dané rozhraní, jehož tělo obsahuje konkrétní nová data. Aplikace pak nemusí vznášet žádné požadavky vůči Garmin Connect cloudu. Tento typ notifikací je použit i v rámci této aplikace.

Pro samotný příjem dat jsou v rámci aplikace vystaveny 4 REST rozhraní. Jedná se o následující rozhraní.

- **Příjem dat o srdečním tepu**

  Struktura přijatých dat o srdečním tepu přichází z rozhraní ve struktuře, kdy je čas jednotlivých naměřených hodnot relativně posunutý vůči počátečnímu času měření v rámci každé notifikace. Tedy například počátek měření konkrétní notifikace je 11.11.2021 10:00:00 a naměřené hodnoty mají relativní posun 60, 120 a 180 vteřin. Časové značky je tedy nutné při přijetí transformovat na 11.11.2021 10:01:00, 10:02:00, 10:03:00 a naměřená data v jednotlivých zprávách odeslat.

- **Příjem dat o pohybové aktivitě**

  Příjem dat o pohybové aktivitě je v rámci částečné anonymizace dat ze strany Garmin Connect cloudu řešen tak, že chodí pouze souhrny aktivity v 15 minutových intervalech, tzv. epochách.

  Data jsou tedy strukturována tak, že pro každou 15 minutovou epochu přijde informace o tom, kolik vteřin v rámci dané epochy uživatel vykonával jakou aktivitu.

[4] https://github.com/googleapis/google-oauth-java-client

Navíc, jednotlivé aktivity z každé epochy mohou přijít rozdělené do více notifikací. Kvůli tomuto formátu se následně nedá replikovat přesná aktivita pacienta každou minutu. To pro účely této aplikace nevadí, protože v rámci agregátoru TERESA se sledují primárně trendy aktivity v průběhu času.

Vzhledem k tomu, že centrální agregátor očekává pro každou minutu pouze jeden záznam, je nutné data vhodně transformovat a rozčlenit skrze celou epochu.

Aplikace si tedy pro každou epochu v případě, že je její součástí více aktivit, udržuje informaci o posunu, tedy o kolik minut mají být následující zprávy z dané epochy posunuty. V případě, že jsou odeslány všechny části dané epochy, je pak z důvodu úspory dat v databázi daný záznam smazán, neboť již není potřeba.

Pro názornost je uveden následující příklad práce s epochami.

```
Start epochy: 14:00, 25.11.2021
Údaje o aktivitě: 5 minut běh, 7 minut chůze, 3 minuty sezení

Generování zpráv:
14:00 - 14:04 běh
14:05 - 14:11 chůze
14:12 - 14:14 sezení
```

- **Příjem dat o spánkové aktivitě**

  Data o celkovém souhrnu spánku za předchozí noc, i data o jeho kvalitě v jednotlivých časových intervalech, se nachází v rámci struktury přijímaných dat v rozhraní pro příjem dat o spánkové aktivitě.

  Data jsou rozdělena na zprávu o souhrnu spánku a následně na zprávy o jednotlivých časových intervalech s fázemi spánku.

- **Příjem dat o zrušení souhlasu se stahováním dat**

  Pro splnění podmínek využívání Garmin Connect pro produkční účely bylo nutné implementovat rozhraní, na které přijde notifikace, že se uživatel rozhodl zrušit souhlas s poskytováním dat. Neočekává se, že by bylo toto rozhraní používáno vzhledem k tomu, že aplikace pracuje s účty pacientů. Každopádně kvůli splnění podmínek bylo implementováno.

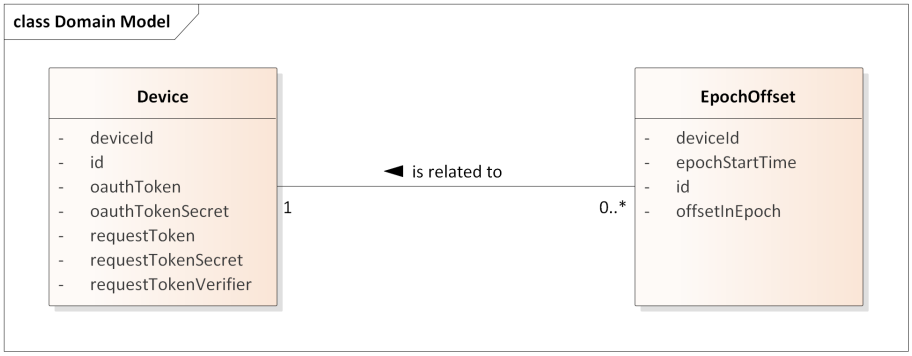
6.2.4 Doménový model

Vzhledem k tomu, že aplikace většinu přijatých dat pouze transformuje a přeposílá zprávami dále, je doménový model velmi jednoduchý. Obsahuje entitu Device, která slouží k obsluze dat spojených s jednotlivými zařízeními, tedy jejich identifikátor v agregátoru TERESA, a následně data o přístupových tokenech ke sběru dat z Garmin Connect cloudu. Druhou entitou je EpochOffset, což je entita sloužící k perzistenci informací o tom, kolik dat v dané epoše bylo doposud zpracováno. Podrobnější logika tohoto zpracování je popsána v kapitole 6.2.3.

Doménový model je zobrazen na obrázku 6.4.

6.3 Načítač dat z platformy Fitbit

Pro zpracování dat z cloudu Fitbit byla stejně jako pro načítač z platformy Garmin Connect v souladu s navrhnutou architekturou v kapitole 4.5.5 vyvinuta samostatná aplikace, dále označovaná jako Fitbit Worker.



Obrázek 6.4. Doménový model načítače dat z platformy Garmin

6.3.1 Získání přístupu k platformě Fitbit

Stejně jako pro přístup ke Garmin Connect cloudu, bylo potřeba získat ověření a přístup k datům v rámci Fitbit cloudu. K tomuto účelu slouží speciální registrační formulář[5]. Celý proces je velmi podobný tomu pro Garmin Connect viz kapitola 6.2.1. Rozdílem je, že pro přístup k datům s minutovou granularitou (kromě denních přehledů) je nutné vyplnit speciální žádost s důkladnějším odůvodněním nutnosti sběru těchto dat a důkladnějším popisem, jak bude s těmito daty zacházeno.

Zřízení přístupu ke všem datům, včetně těch s minutovou granularitou, pro tuto aplikaci, probíhalo rychle a bez problémů. Společnost Fitbit byla v rámci celého procesu proaktivní, byla poskytnuta i doporučení, jak s daty zacházet a jak je chránit před únikem.

6.3.2 Autorizace nových zařízení

Pro získání přístupu k datům konkrétního uživatele je nutné, aby daný uživatel autorizoval přístup této aplikace. Celý proces autorizace a následně přístup k datům uživatele je, podobně jako u Garmin Connect viz kapitola 6.2.2, řešen pomocí protokolu OAuth, tentokrát ale v novější verzi 2.0[6].

Proces ve verzi protokolu 2.0 je jednodušší, než ve verzi 1.0a, a je následující.

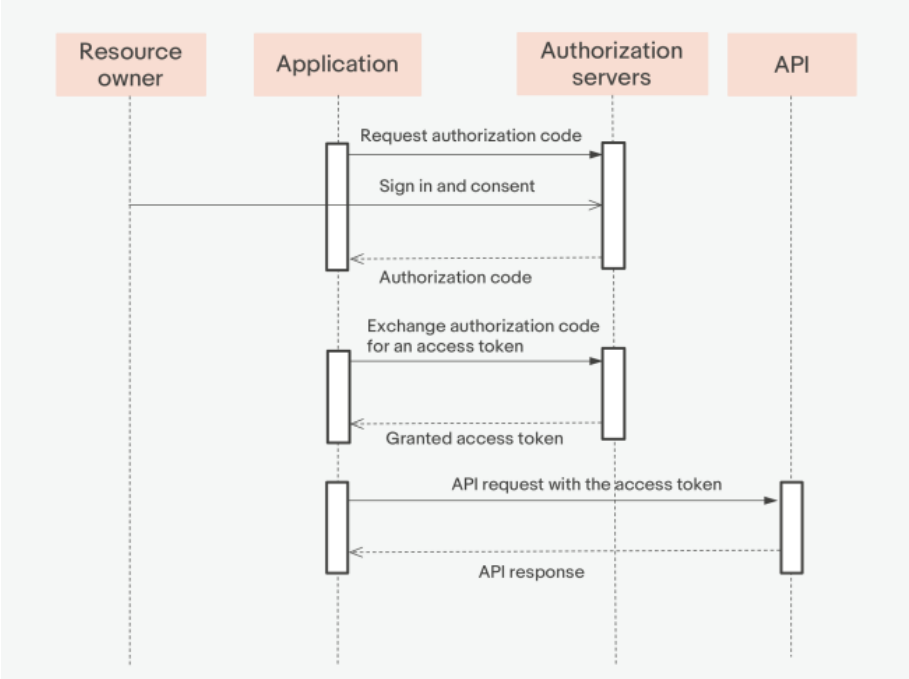
- Aplikace přesměruje uživatele na stránku Fitbit
- Uživatel autorizuje přístup aplikaci Fitbit Worker přihlášením se do svého uživatelského účtu a potvrzením přístupu k datům
- Aplikace Fitbit Worker obdrží od Fitbit autorizační kód
- Aplikace Fitbit Worker využije autorizační kód k získání accessToken a refreshToken, které následně může využívat pro přístup k datům

  Celý proces, jakým jsou tokeny získány, je znázorněn obrázku 6.1.

  Fitbit využívá dle standardu 2 typy přístupových tokenů.
- **přístupový token – accessToken**

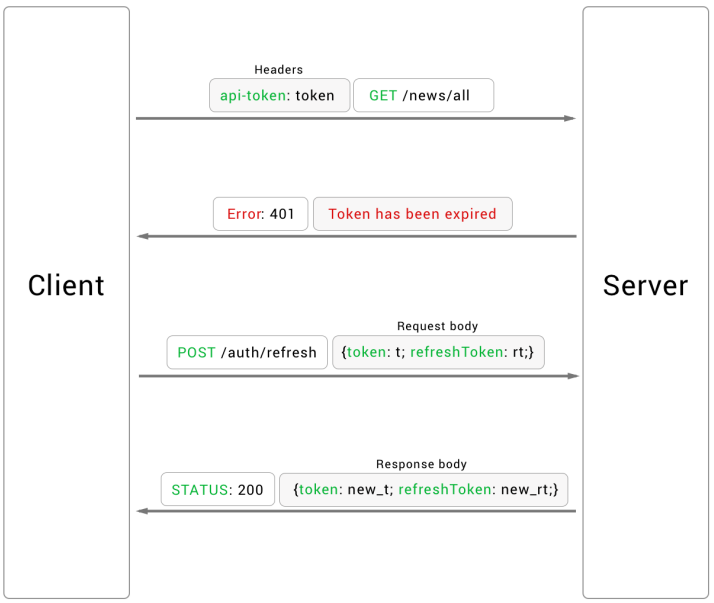
  Přístupový token umožňuje přístup k uživatelským datům, kde ke každému požadavku je přidána hlavička `Authorization` ve tvaru `Bearer <accessToken>`. Díky tomu je zajištěna autorizace přístupu k datům. Vzhledem k tomu, že token není v tomto případě nijak šifrován, a tedy v případě odposlechnutí může být zneužit někým jiným, má accessToken krátkodobou (8 hodin) platnost. Po jeho vypršení musí dojít k obnovení tokenu za jiný pomocí refreshTokenu.
- **obnovovací token – refreshToken**

[5] https://dev.fitbit.com/apps/new
[6] https://oauth.net/2/



Obrázek 6.5. OAuth 2.0 proces autorizace[44]

RefreshToken je jednorázový token sloužící k vygenerování nového accessTokenu. V rámci vygenerování nového accessTokenu aplikace obdrží společně s ním i nový refreshToken pro příští přegenerování.
Proces výměny tokenů je znázorněn na obrázku 6.6.



Obrázek 6.6. OAuth 2.0 proces obnovení přístupového tokenu[45]

Aplikace Fitbit Worker nemá žádné uživatelské rozhraní, ze kterého by mohl uživatel obsluhující celý systém TERESA zahájit tento proces. Proto vystavuje Fitbit

Worker rozhraní sloužící k nastartování a provedení OAuth autorizace. Proces je stejný jako v případě aplikace Garmin Worker, jež je detailně popsán v kapitole 6.2.1.

Stejně jako OAuth protokol ve verzi 1.0a je i protokol ve verzi 2.0 standardem. Existuje velké množství knihoven pro zjednodušení implementace. V rámci aplikace Fitbit Worker byla využita knihovna ScribeJava[7] ve verzi 8.3.1.

6.3.3 Získávání a zpracování dat z Fitbit

Fitbit cloud je schopen, stejně jako Garmin Connect, poskytovat data vícero způsoby. Data je možné získat buď na vyžádání, nebo prostřednictvím notifikací o tom, že si uživatel synchronizoval své zařízení a jsou dostupná nová data.

Kvůli úspoře požadavků vznášených na Fitbit cloud je doporučeno data získávat na základě notifikací, stejně jako u Garmin Connect. Díky tomu jsou data získávána pouze pro zařízení, pro která jsou nová data dostupná.

Pro příjem notifikací je nutné vystavit REST rozhraní schopné přijímat HTTP POST požadavky. V těle požadavků je pak seznam notifikací o tom, o jaký typ dat se jedná a za jaký den jsou k dispozici. Aplikace Fitbit Worker si všechny notifikace uloží do své databáze pro následné zpracování, které je popsáno dále v textu. Aby Fitbit generoval notifikace pro jednotlivé uživatele, je potřeba se přihlásit k odběru těchto notifikací pro každého z nich. K tomu dochází vždy v rámci autorizace nového zařízení.

```
[
    {
        "collectionType": "foods",
        "date": "2010-03-01",
        "ownerId": "USER_1",
        "ownerType": "user",
        "subscriptionId": "1234"
    }, {
        "collectionType": "activities",
        "date": "2010-03-01",
        "ownerId": "X1Y2Z3",
        "ownerType": "user",
        "subscriptionId": "2345"
    }
]
```

Data jsou stahována v reakci na výše zobrazenou strukturu notifikací o dostupnosti nových dat. Každou hodinu je spuštěna procedura, která se pokusí získat data týkající se všech notifikací, jež dosud nebyla úspěšně zpracována. V případě, že se data týkající se některé notifikace nepodaří zpracovat, zůstává tato notifikace označena jako nezpracovaná a následující hodinu dojde k novému pokusu o její zpracování.

Jedním z požadavků Fitbit cloudu vůči aplikacím třetích stran je to, že počet dotazů na data týkající se jednoho uživatele nesmí přesáhnout 150 za hodinu. Z tohoto důvodu je v aplikaci Fitbit Worker implementována logika, která zabraňuje v rámci stahování dat přesažení tohoto limitu.

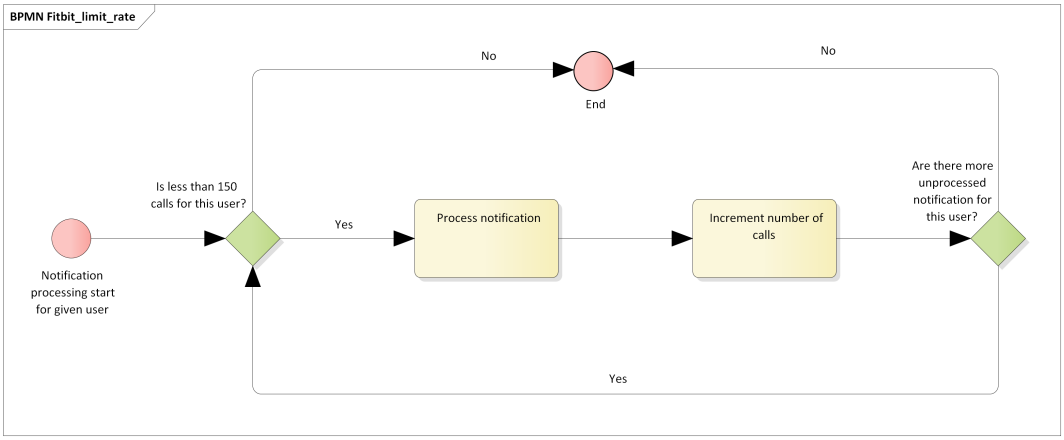
Logika je taková, že v rámci procedury, která každou hodinu zpracovává notifikace, je nejprve vynulován počet požadavků v uplynulé hodině. Následně je s každým

[7] https://github.com/scribejava/scribejava

požadavkem na stažení dat nezpracovaných notifikací, či případné obnovy tokenů, tento čítač pro jednotlivá zařízení inkrementován.

V případě, že čítač dosáhne limitní hodnoty, je zpracování notifikací pro dané zařízení pozastaveno. Zbylé notifikace jsou pak staženy v rámci dalšího běhu procedury. Tím je zajištěno to, že nemůže dojít k překročení požadovaného limitu.

Logika zajištění nepřekročení limitního počtu požadavků je znázorněna na obrázku 6.7.



Obrázek 6.7. Logika zajištění nepřekročení počtu požadavků vůči Fitbit cloudu

Samotné získávání jednotlivých typů dat je řešeno napojením na REST rozhraní Fitbit cloudu[8] a je popsáno níže.

- **Příjem dat o srdečním tepu**

  Přijatá data o srdečním tepu jsou rozdělena dle naměřených hodnot s granularitou 1 minuta. Data jsou tedy rozdělena na jednotlivé zprávy a odeslány do Apache Kafka clusteru.

- **Příjem dat o pohybové aktivitě**

  Data o pohybové aktivitě a její energetické náročnosti jsou získávána ze dvou různých rozhraní. Z tohoto důvodu je nutné pro získání kompletních dat nejprve získat data z obou rozhraní a data následně zkombinovat a rozdělit s minutovou granularitou do jednotlivých zpráv.

- **Příjem dat o spánkové aktivitě**

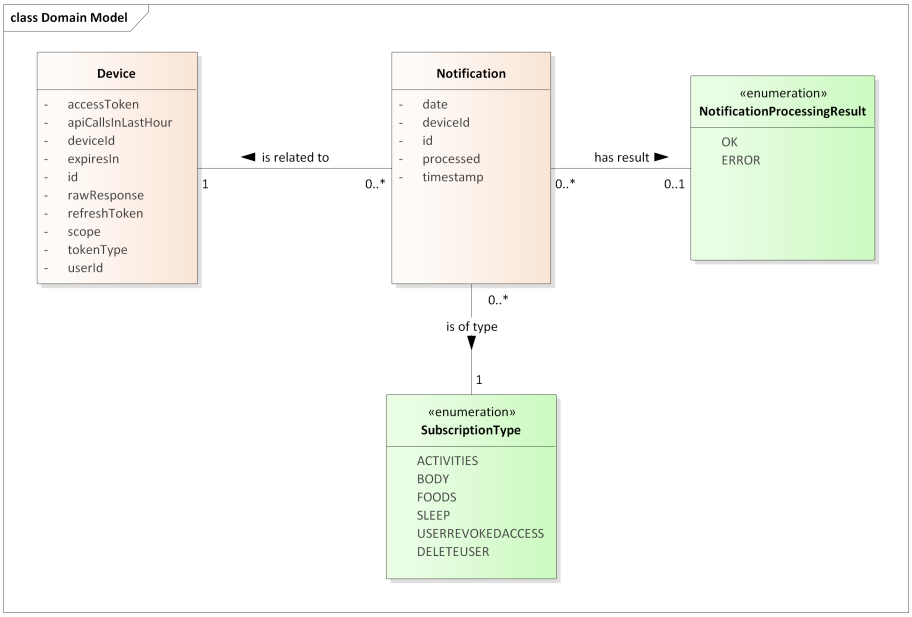
  Z rozhraní poskytujícím data o spánkové aktivitě jsou získávána, podobně jako v Garmin Connect, data jak ohledně celkového souhrnu spánku, tak ohledně jednotlivých časových intervalů jednotlivých fází spánku.

  Data jsou tudíž rozdělena na zprávu o souhrnu spánku a následně na zprávy o jednotlivých časových intervalech s fázemi spánku.

6.3.4 Doménový model

Stejně jako ostatní načítače, aplikace většinu dat pouze transformuje a přeposílá zprávami dále, tudíž je doménový model velmi jednoduchý. Obsahuje entitu Device, která slouží k perzistenci dat spojených s jednotlivými zařízeními, tedy jejich identifikátor v agregátoru TERESA a následně data o přístupových tokenech ke sběru dat z Fitbit cloudu. Druhou entitou je Notification, což je entita sloužící k perzistenci informací o přijatých notifikacích a stavu jejich zpracování. Podrobná logika zpracování v souvislosti s notifikacemi je popsána v kapitole 6.3.3.

[8] https://dev.fitbit.com/build/reference/web-api/developer-guide/



Obrázek 6.8. Doménový model načítače dat z platformy Fitbit

Doménový model je zobrazen na obrázku 6.8.

6.4 Načítač dat z platformy Gadgetbridge

V rámci architektury, která byla navrhnuta v kapitole 4.5.5 bylo nutné upravit stávající řešení pro příjem dat z mobilní aplikace Konektor, potažmo z platformy Gadgetbridge. Existující řešení bylo součástí agregátoru TERESA a proto bylo v souladu s navrhnutou architekturou vyjmuto. Byla pro něj vyvinuta samostatná aplikace, dále označovaná jako Gadgetbridge Worker.

6.4.1 Získání přístupu ke Gadgetbridge

Aplikace Gadgetbridge[9] je mobilní aplikace pro platformu Android umožňující sběr dat ze zařízení různých výrobců. V případě systému TERESA jsou sbírána data z chytrých náramků výrobce Xiaomi. Aplikace umožňuje vytvářet SQL exporty pro další aplikace, které je mohou využívat. Tyto exporty jsou v mobilních telefonech zpracovávány aplikací Konektor, jež je vyvíjena v rámci pracovní skupiny jako další studentská závěrečná práce. Aplikace Konektor následně data zasílá na vystavené rozhraní systému TERESA.

Díky tomu, že jsou data poskytována aplikací, která je vyvíjena touto skupinou, není nutné získávat žádný speciální přístup pro sběr těchto dat.

Při vytváření této mikroslužby musela být zároveň zachována funkcionalita tak, aby nedošlo k žádnému dopadu na aplikaci Konektor. Díky tomu byla jak logika autorizace, tak rozhraní pro příjem dat zachovány v plném rozsahu.

6.4.2 Autorizace nových zařízení

Stejně jako u ostatních načítačů má i tato aplikace vystavené rozhraní pro autorizaci nového zařízení. Vzhledem k tomu, že autorizace zařízení je na straně tohoto systému, je jeho proces jednoduchý. Stačí si pouze uložit identifikátor nového zařízení

[9] https://gadgetbridge.org/

v agregátoru TERESA tak, aby byla aplikace Gadgetbridge Worker schopna data přijatá z tohoto zařízení odesílat dále do centrálního agregátoru.

Autorizace přijímaných dat je řešena následujícím způsobem. Data jsou přijímána pouze za podmínky, že jsou svázána se zařízením, které je aplikaci Gadgetbridge Worker známé. Další podmínkou je, že všechny požadavky, které jsou vůči aplikaci Gadgetbridge Worker vzneseny, musí obsahovat hlavičku `Authorization` s hodnotou `Bearer <SECRET>`, kde `SECRET` je předsdílený klíč známý pouze mobilní aplikaci a této aplikaci Gadgetbridge Worker.

6.4.3 Získávání a zpracování dat z Gadgetbridge

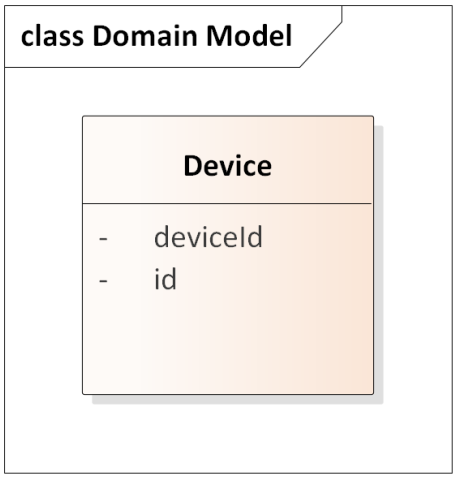
Pro přenos dat vystavuje aplikace Gadgetbridge worker rozhraní, na které se několikrát denně přenáší data z jednotlivých mobilních zařízení, respektive z nainstalované aplikace Konektor v mobilním zařízení. V případě neúspěšného odeslání jsou data na zařízeních stále uložena a jsou odeslána při dalším pokusu o přenos dat. Tímto je ošetřeno to, že nedochází ke ztrátám dat ani v případě výpadku aplikace Gadgetbridge Worker.

Příjem dat je pro tuto aplikaci řešen jediným rozhraním, které přijímá data o aktivitě, srdečním tepu i souhrnu aktivity. Data jsou vždy zasílána s minutovou granularitou, a tak je není potřeba transformovat. Data o kvalitě spánku aplikace Konektor nezasílá.

6.4.4 Doménový model

Stejně jako ostatní načítače, aplikace většinu dat pouze transformuje a přeposílá zprávami dále, tudíž je doménový model velmi jednoduchý. Obsahuje entitu Device, která slouží k perzistenci známých zařízeních a jejich identifikátoru v agregátoru TERESA.

Doménový model je zobrazen na obrázku 6.9.



Obrázek 6.9. Doménový model načítače dat z platformy Gadgetbridge

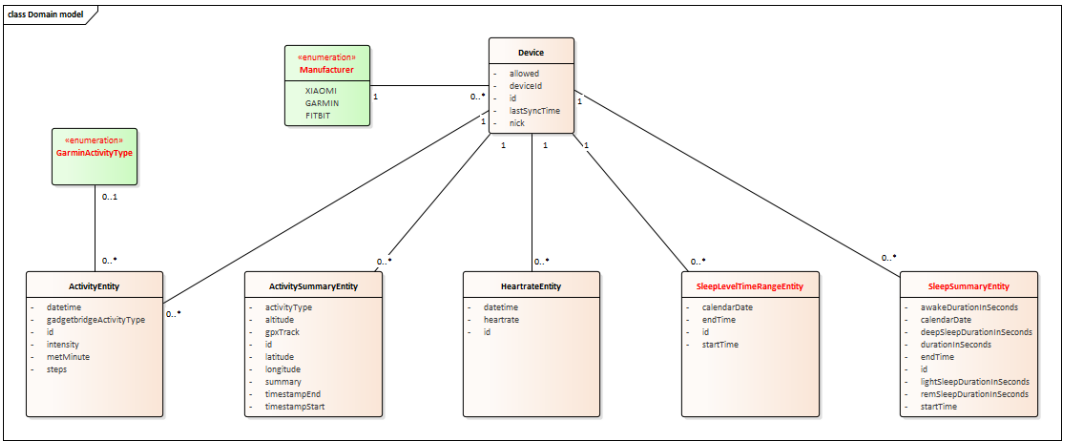
6.5 Agregace dat v aplikaci TERESA

V rámci implementace byla aplikace TERESA rozšířena v několika částech. Tato rozšíření budou popsána v následujících kapitolách. Jedná se o následující rozšíření:

- rozšíření datového modelu
- vytvoření rozhraní pro příjem zpráv s daty z Apache Kafka
- modifikace správy náramků o možnost pracovat se zařízeními různých výrobců
- rozšíření exportů dat o detailní grafy aktivity, srdečního tepu a kvality spánku

6.5.1 Rozšíření doménového modelu

Doménový model aplikace TERESA byl v rámci úprav rozšířen o nové typy sbíraných dat, jako jsou např. data o spánku a dále o výrobce daného zařízení. Zjednodušený doménový model, ve kterém jsou vynechány některé části nerelevantní pro tuto práci, je zobrazen na obrázku 6.10. Rozšíření v rámci této práce jsou označeny červeně.



Obrázek 6.10. Doménový model aplikace TERESA

6.5.2 Příjem a zpracování dat z Apache Kafka

Aplikace TERESA byla rozšířena o rozhraní pro příjem zpráv z Apache Kafka. Pro připojení ke Kafka clusteru je podobně jako v ostatních aplikacích použita knihovna Spring for Apache Kafka viz kapitola 5.2.4.

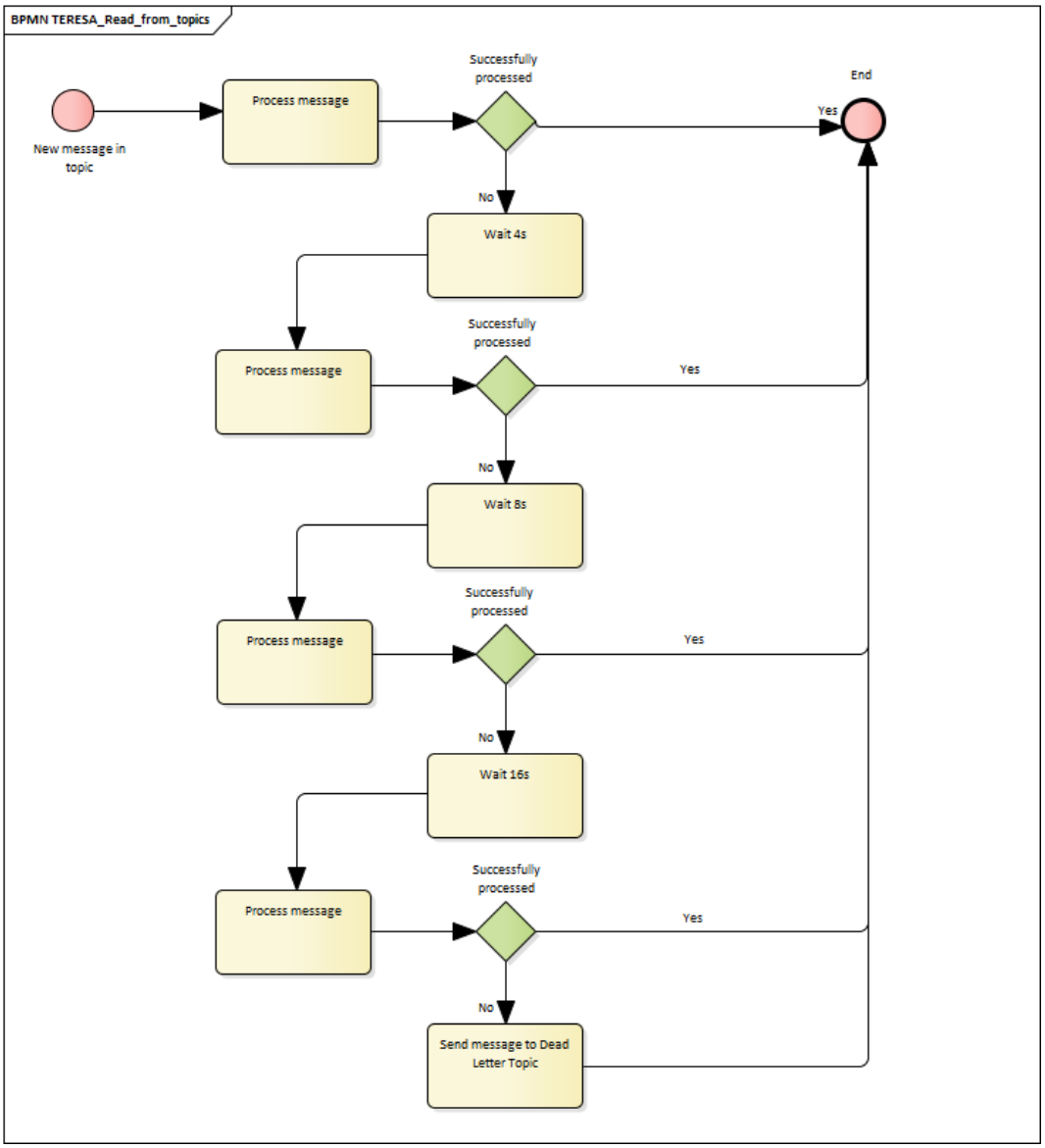
Knihovna poskytuje rozhraní pro napojení se na jednotlivé fronty zpráv. Pro každou frontu zpráv, ze které chce aplikace odebírat data, je vytvořen a nakonfigurován posluchač. Každý posluchač zajišťuje periodickou kontrolu obsahu front a případné načítání nových zpráv. Toto napojení lze vhodně nakonfigurovat tak, aby bylo schopné zajistit korektní zpracování dat i v případě krátkodobých výpadků databáze, případně kolizí databázových transakcí, apod.

Pokud zpráva není korektně zpracována, neputuje rovnou do fronty nekorektně zpracovaných zpráv. Posluchače jsou totiž nakonfigurovány tak, aby se aplikace pokusila zprávu zpracovat opakovaně. Konfigurace je taková, že se zpráva pokusí zpracovat 4x se zvětšujícím se časový rozestupem – 4 vteřiny, 8 vteřin a 16 vteřin. Pokud není ani na jeden pokus zpráva úspěšně zpracována, pak je odeslána do fronty neúspěšně zpracovaných zpráv. Tuto frontu je nutné v rámci provozu pravidelně kontrolovat a v případě, že se v ní některé zprávy objeví, pak je nechat znovu zpracovat.

Proces opakovaného zpracování zpráv z jednotlivých front je zobrazen na obrázku 6.11.

Ukázka konfigurace posluchače zpráv.

```
@RetryableTopic(
    attempts = "4",
```



Obrázek 6.11. Logika robustního zpracování zpráv z Apache Kafka

```
    backoff = @Backoff(delay = 4000, multiplier = 2.0),
    kafkaTemplate = "<KAFKA_TEMPLATE>",
    dltStrategy = DltStrategy.FAIL_ON_ERROR
@KafkaListener(id = "<LISTENER_ID>", topics = "<TOPIC_NAME>",
    containerFactory = "<CONTAINER_FACTORY>")
```

Vzhledem k tomu, že zprávy jsou koncipovány tak, aby obsahovaly data s takovou granularitou, aby mohly být perzistovány v rámci jediné entity, je samotné zpracování zpráv jednoduché. Úkolem aplikace TERESA je tedy data z jednotlivých zpráv transformovat do entit a ty následně perzistovat k jednotlivým zařízením, dle identifikátoru, který je součástí zprávy.

6.5.3 Úpravy uživatelského rozhraní

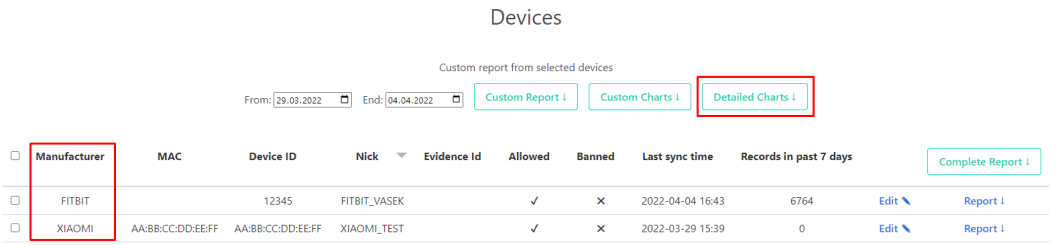
Uživatelské rozhraní bylo rozšířeno v části vytváření, editace a přehled jednotlivých zařízení, stejně jako o možnost exportu detailních přehledů naměřených dat.

Z úprav datového modelu viz kapitola 6.5.1 vyplývá, že změny v oblasti zařízení jsou svázány s rozšířením o výrobce zařízení. O výrobce zařízení jsou tedy rozšířeny obrazovky pro vytváření a editaci zařízení, stejně jako přehled všech zařízení.

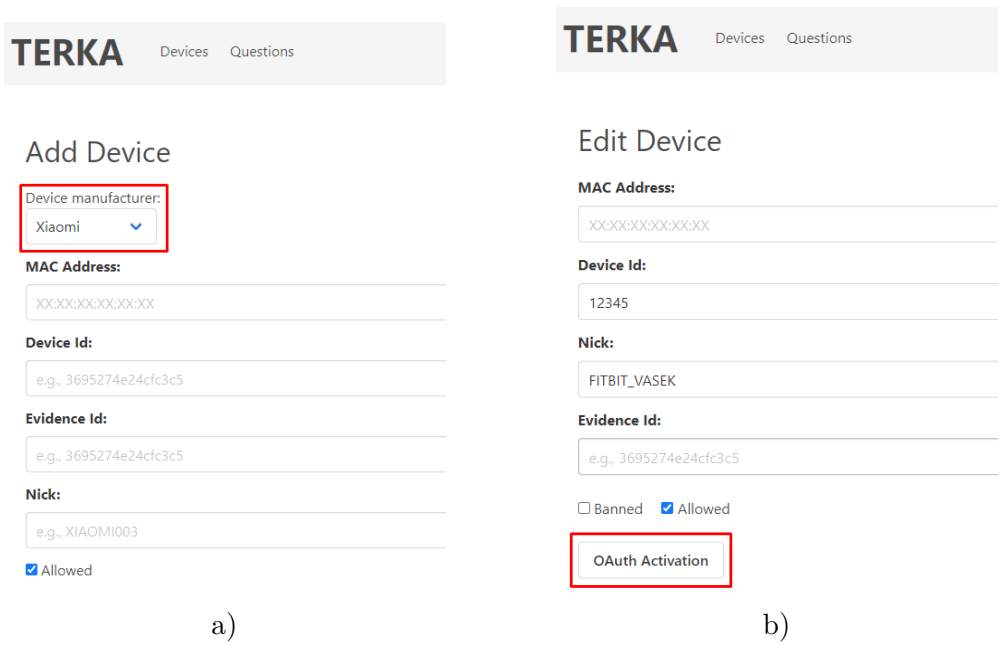
Obrazovka editace zařízení je v případě výrobců Garmin a Fitbit dále rozšířena o tlačítko, které umožňuje spustit OAuth proces autorizace jednotlivých zařízení. Proces samotného OAuth procesu autorizace je detailně popsán v kapitolách 6.2.2 a 6.3.2.

Poslední úpravou uživatelského rozhraní je přidání tlačítka pro generování detailních přehledů naměřených dat. Tlačítko je na hlavní obrazovce a umožňuje generovat přehledy pro zvolená zařízení za zvolené období. Logika generování a struktury těchto reportů je detailně popsána v kapitole 6.5.4. Logika volby zařízení a období je stejná jako pro ostatní reporty pro zachování logiky napříč celou aplikací.

Veškeré změny jsou vidět na obrázcích 6.12 a 6.13. Úpravy jsou zvýrazněny červenými rámečky.



Obrázek 6.12. Úpravy uživatelského rozhraní TERESA – přehled zařízení



a) b)

Obrázek 6.13. Úpravy uživatelského rozhraní TERESA – vytvoření a editace zařízení

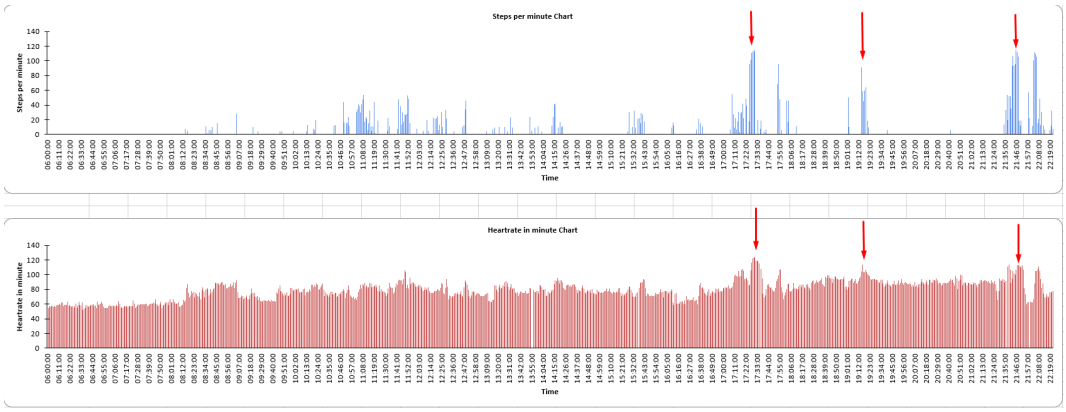
6.5.4 Export detailních přehledů naměřených dat

V rámci rozšíření v aplikaci TERESA bylo vytvořeno generování souhrnných přehledů naměřené aktivity (počet kroků za den) o detailní přehledy srdeční aktivity, počtu kroků a spánkové aktivity v průběhu dne.

Pro generování detailních přehledů je nutné, aby uživatel zvolil zařízení, pro která chce data exportovat. Stejně tak je důležité, aby zvolil periodu, za kterou chce data generovat. Následně aplikace pro jednotlivá zařízení vygeneruje soubory, které se nazývají `<DEVICE_NICK>.xlsx` a vrátí všechny soubory v jednom archivu s názvem `DetailedCharts.zip`. Uživatel může následně procházet jednotlivé soubory a případně nad nimi vytvářet další statistiky a reporty přímo v aplikaci Excel.

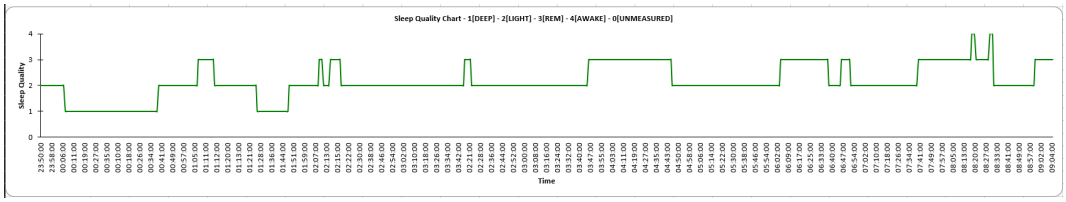
Každý soubor je členěn do několika karet. První kartou je vždy karta s názvem `<NICK>`. Tato karta byla přepoužita z již existujících exportů a jedná se o kartu se souhrnným přehledem počtu kroků v jednotlivých dnech zvolené periody. Následují, v rámci této práce nově vytvořené, karty s detailními grafy. Jedná se o jednu kartu pro každý den ze zvolené periody. Karta má vždy název ve tvaru `<dd.MM.yy>`.

Jednotlivé karty obsahují na prvních 6 řádcích naměřená data s minutovou granularitou pro kroky, srdeční tep a typ spánkové aktivity. Data o počtu kroků a srdeční aktivitě jsou pro přehlednost zobrazena pouze v rozmezí 6-22 hodin. Data o obou veličinách bylo nutné zobrazit ve stejném počtu a měřítku tak, aby bylo možné vidět vliv počtu kroků na změny v srdeční aktivitě. Detailní grafy jsou pak schopny lékařům poskytnout vhled do toho, jak moc je daná aktivita pro pacienta náročná. Ukázka grafu reálně naměřené aktivity, tedy kroků a srdeční aktivity, stejně jako vlivu zvýšené aktivity na srdeční tep je zobrazena na obrázku 6.14.



Obrázek 6.14. Ukázka grafu s naměřeným počtem kroků a srdeční aktivitou

Třetím grafem je graf kvality spánku. Z grafu lze vyčíst odkdy, dokdy a jak dlouho pacient spal, stejně jako naměřené spánkové fáze. U spánkových fází se odlišují následující fáze: lehký spánek, hluboký spánek, fáze REM, pacient je vzhůru a neměřeno[46]. Reálně naměřená data a ukázka reprezentace těchto dat je zobrazena na obrázku 6.15.



Obrázek 6.15. Ukázka grafu s naměřenými daty kvality spánku

Veškeré generování dokumentů typu Excel s příponou .xlsx je řešeno pomocí knihovny Apache POI[10] ve verzi 4.1.0, která slouží ke generování dokumentů ve formátu Microsoft Office[11].

6.6 Konfigurace Kafka Connect

Jak již bylo popsáno v kapitole 5.3.1, řešení Kafka Connect slouží v systému TERESA k perzistenci všech zpráv zaslaných skrze Apache Kafka.

Jedná se o nástroj, který lze libovolně konfigurovat. Pro účely tohoto systému jej bylo nutné nakonfigurovat tak, aby při komunikaci s Kafka clusterem pracoval se šifrováním pomocí SSL. Následně bylo nutné vytvořit konfigurace pro napojení na jednotlivé topicy a perzistenci zpráv, které obsahují do zvolených kolekcí v MongoDB.

Konfiguraci lze provést zasláním HTTP POST požadavku s konfigurací zaslanou v těle tohoto požadavku na rozhraní, které Kafka Connect vystavuje. Ukázka takové konfigurace je znázorněna níže.

```
{
   "name": "mongodb-sink-activity",
   "config": {
      "connector.class":
         "at.grahsl.kafka.connect.mongodb.MongoDbSinkConnector",
      "tasks.max": 1,
      "topics": "activity",
      "mongodb.connection.uri":
         "mongodb://pc2e:<PASSWORD>@mongo-db:27017/kafkaData",
      "mongodb.collection": "Activity",
      "key.converter":
         "org.apache.kafka.connect.storage.StringConverter",
      "key.converter.schemas.enable": false,
      "value.converter":
         "org.apache.kafka.connect.storage.StringConverter",
      "value.converter.schemas.enable": false
   }
}
```

Požadavek lze zaslat následujícím způsobem, kdy konfigurace je uložena v souboru configuration.json a Kafka Connect je nasazen lokálně na portu 8083.

```
curl -d @./kafka-connect/configuration.json \
   -H "Content-Type: application/json" \
   -X POST http://localhost:8083/connectors
```

[10] https://poi.apache.org/index.html
[11] https://www.office.com/

Kapitola 7
Testování

Pro zajištění kvality systému a ověření jeho spolehlivosti bylo nutné celý systém řádně otestovat. Testování a verifikace probíhala v rámci všech fází tohoto projektu s cílem odhalit případné chyby co nejdříve. Čím dříve jsou chyby odhaleny, tím je jejich oprava snazší a tedy i levnější[47]. Jednotlivé typy provedených testů, jejich návrh, implementace a výsledky jsou popsány v následujících kapitolách.

V rámci efektivity testování aplikací je nutné v rámci testování stanovit prioritní oblasti, na které by se testování mělo zejména zaměřit[48]. Zpravidla jsou to oblasti, jejichž špatná funkčnost by byla kritická buď pro stabilitu systému, nebo pro vzniklou škodu v závislosti na nefunkčnosti dané části systému. V rámci systému TERESA je prioritou zpracování dat a správnost takto zpracovaných dat. Na tyto oblasti bylo testování v rámci tohoto systému koncentrováno.

7.1 Automatické testy

Pro účely co nejdřívějšího odhalení chyb byly implementovány v průběhu vývoje jak jednotkové, tak integrační testy. Tyto testy byly doplněny o statickou analýzu kódu.

7.1.1 Jednotkové testy

Jednotkové testování je velmi běžný typ automatického testování. Jedná se o ověřování správnosti chování a implementace atomických částí systému. Jednotkové testování je velmi dobré pro rychlé odhalení chyb z nedomyšlení okrajových podmínek, nepozornosti, atd[49].

Jednotkové testy se zpravidla píší ihned po implementaci dané části systému, nebo ještě před samotnou implementací. Pokud jsou napřed napsány jednotkové testy a až pak implementace, pak tomuto stylu vývoje říkáme testy řízený vývoj[50].

V rámci této práce jsou v rámci všech jednotlivých mikroslužeb napsány jednotkové testy zaměřující se primárně na správné chování systému při zpracování přijatých dat, tedy jejich rozdělení dle uživatelů, rozdělení dat na minutovou granularitu a další logiku spojenou se zpracováním těchto dat.

Pro implementaci jednotkových testů jsou využity Spring knihovny Spring-boot-starter-test[1] ve verzi 2.6.5 a Spring-boot-kafka-test ve verzi 2.8.4, které obsahují předkonfigurované knihovny k testování, tedy knihovny JUnit, AssertJ, Mockito, Hamcrest a další.

7.1.2 Integrační testy

Integrační testy slouží, na rozdíl od jednotkových testů, k otestování větších částí systému a jejich vzájemné spolupráce. V rámci tohoto systému byl kladen důraz na otestování aplikací při spolupráci s Apache Kafka, stejně jako ověření správnosti chování vystavených rozhraní a díky tomu celého zpracování dat.

[1] https://docs.spring.io/spring-boot/docs/1.5.7.RELEASE/reference/html/boot-features-testing.html

Knihovny využité pro integrační testování byly stejné jako pro jednotkové testování viz kapitola 7.1.1.

7.1.3 Statická analýza kódu

Statická analýza kódu je technika sloužící k odhalení potenciálních chyb softwaru bez toho, aby musel být spuštěn. Jedná se o efektivní metodu jak nalézt některé z chyb hned po jejich vzniku, tedy naprogramování, ještě před tím, než je program spuštěn[51].

Nástroje hledají chyby jako nekonečné smyčky, větvení programu s tím, že do některé větve se program nikdy nemůže dostat, potenciální přetečení hodnot, potenciálně špatné přetypování, a další. Nástroje odhalují chyby přímou prací s kódem, případně sestavením modelu nad kódem a jeho následnou kontrolou[51].

V rámci vývoje byl kromě zabudovaného statického analyzátoru kódu ve vývojovém prostředí IntelliJ IDEA[2] použit nástroj SonarLint[3].

7.2 Testování zátěže

V rámci celého návrhu a implementace byl kladen velký důraz na výkonnost a robustnost celého systému. Z tohoto důvodu bylo přistoupeno k zátěžovému testování celého systému.

7.2.1 Použité nástroje

Pro zátěžové testování byla zvolena cloudová služba loader.io[4], která v rámci bezplatné licence umožňuje simulovat zátěž tisíců požadavků dvou typů najednou a její použití je jednoduché a uživatelsky přívětivé.

7.2.2 Návrh testu a testovací data

Dle požadavku NFR01 z kapitoly 3.4 je vyžadováno, aby byl systém schopen odbavit 500 uživatelů denně s největší zátěží ve večerních hodinách.

Z tohoto důvodu bylo pro ověření výkonnosti a kapacity zpracování zvoleno, že bude systém otestován na přijetí celodenních dat od 250, 500 a 1000 uživatelů v časovém rozmezí 1 minuty. V reálném provozu je takto krátký časový interval nepravděpodobný. Přesto pokud jej systém zvládne, naroste důvěra v kvalitu řešení.

Jako testovací data byla zvolena reálná data, která byla reálně obdržena z platformy Garmin Connect, pro co nejrealističtější test. Vzhledem k tomu, že služba loader.io umožňuje zasílat najednou pouze 2 různé požadavky, byly využity požadavky s daty o počtu kroků a srdečním tepu, jejichž data jsou nejobjemnější. Testovací data jsou součástí zdrojových kódu ve složce `test_data`.

7.2.3 Testovací prostředí

Z důvodu stability produkčního prostředí byl test proveden vůči testovacímu prostředí, tedy `https://dev.pc2e.felk.cvut.cz/`. Jedná se o virtuální stroj jehož technické specifikace jsou v tabulce 7.1. Na prostředí byly v době testu spuštěny aplikace Garmin Worker, Apache Kafka a proxy Nginx.

[2] https://www.jetbrains.com/idea/
[3] https://www.sonarlint.org/
[4] https://loader.io/

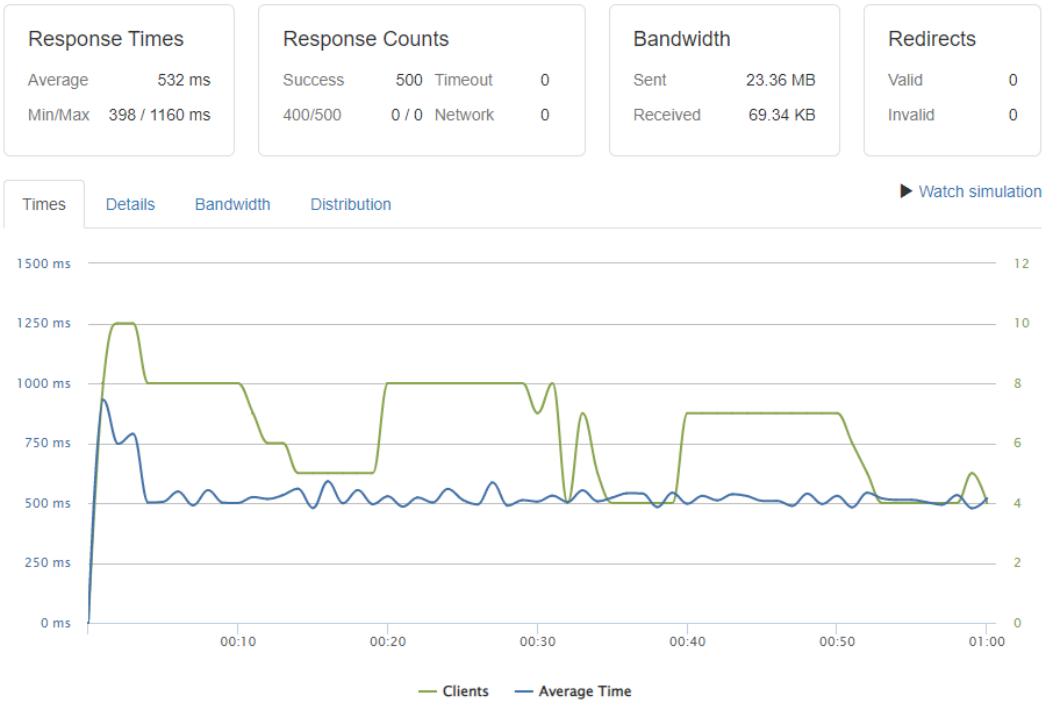
RAM	6GB
Disk	10GB
CPU	Intel, 2 jádra, 3GHz
OS	Debian 11

Tabulka 7.1. Přehled technických specifikací vývojového prostředí

7.2.4 Vyhodnocení

Pro všechny hodnoty zátěže byly úspěšně odbaveny a zpracovány všechny požadavky a na všechny požadavky přišla úspěšná odpověď HTTP 200 OK. Nejvyšší průměrná doba odezvy byla dle očekávání pro nejvyšší zátěž. Jednalo se o průměrnou odezvu 605 ms. Výsledky s přehledem minimální, maximální a průměrné doby odezvy, distribuce zátěže v čase a další statistiky jsou zobrazeny na obrázcích 7.1, 7.2 a 7.3.

Test proběhl bez potíží a prokázal schopnost systému odbavovat množství požadavků nad rámec očekávání i v krátkých časových intervalech.



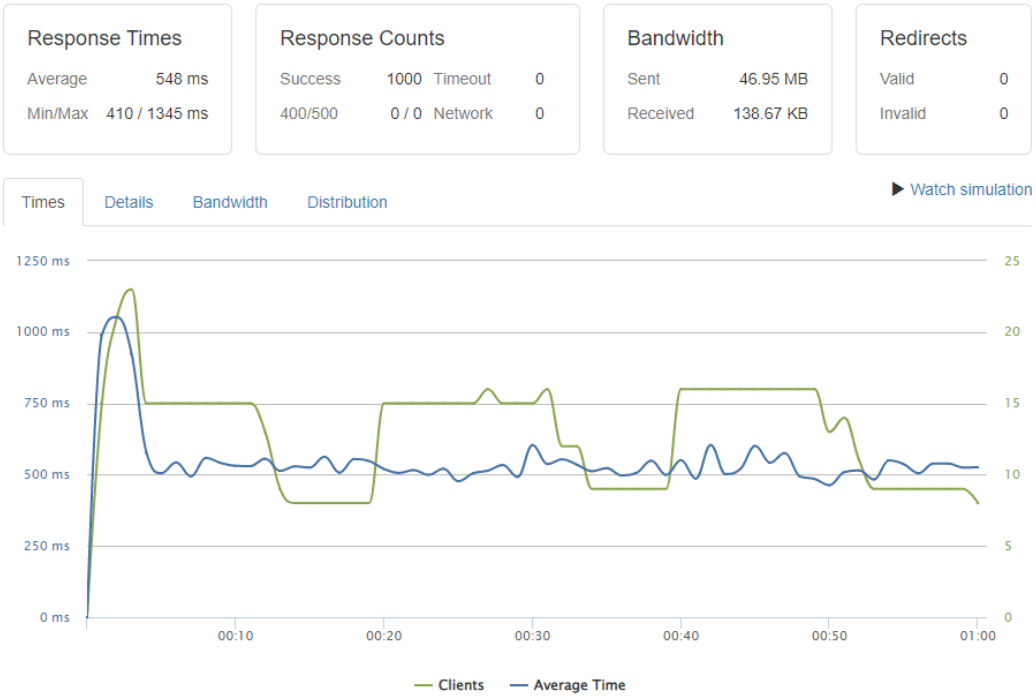
Obrázek 7.1. Výsledky testu pro zátěž 250 zařízení

7.3 Uživatelské testování

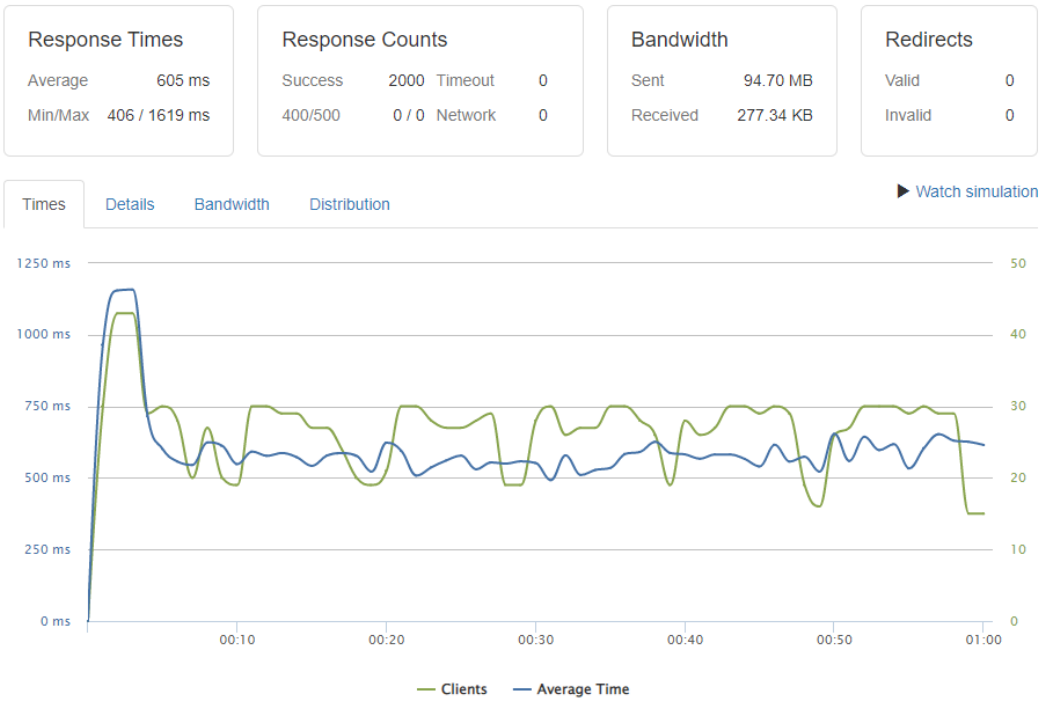
V rámci testování funkčnosti aplikace bylo provedeno testování s uživateli. Testování mělo za cíl ověřit dlouhodobou funkčnost sběru dat a jejich agregace v aplikaci TERESA.

7.3.1 Průběh testu

Pro každého výrobce byly pracovní skupinou zakoupeny 2 kusy zařízení.



Obrázek 7.2. Výsledky testu pro zátěž 500 zařízení



Obrázek 7.3. Výsledky testu pro zátěž 1000 zařízení

- Xiaomi – Mi Smart Band 5
- Garmin – Vívosport S/M
- Fitbit – Charge 4

Tato zařízení byla v aplikaci nakonfigurována a následně předána uživatelům k otestování. Před začátkem testu byl každý uživatel proškolen, jak zařízení nosit na

ruce, jak jej nabíjet a jak jej synchronizovat. Zároveň mu byla do jeho mobilního telefonu nainstalována příslušná mobilní aplikace a proběhlo spárování zařízení s mobilním telefonem pro účely sběru dat.

Následovalo 7 dní, při kterých každý uživatel náramek používal za podobných podmínek, jako jej budou používat potenciální pacienti. Tedy uživatel měl za úkol nosit zařízení po celou dobu na ruce s výjimkou nabíjení a dále 1x denně otevřít aplikaci a nechat data synchronizovat.

7.3.2 Vyhodnocení

Celkově uživatelské testování proběhlo bez větších potíží. V rámci testu byly pro jednotlivé výrobce odhaleny drobné chyby, které jsou popsány níže a které byly následně v aplikaci opraveny.

- **Garmin**
  - V rámci typů aktivit přichází hodnota GENERIC, která není popsána v API
  - Data o kvalitě spánku jsou zasílána opakovaně a mohou být zpětně modifikována – logika jejich zpracování byla upravena tak, aby s touto skutečností uměla pracovat
- **Fitbit**
  - Při přihlašování aplikace k odběru notifikací o dostupnosti nových dat může být odpovědí i HTTP kód 204 CREATED, nejen HTTP kód 200 OK
- **Xiaomi**
  - V rámci testování nebyly odhaleny žádné chyby

Kapitola 8
Provoz

Díky agilní metodice vývoje, kdy jsou jednotlivé funkcionality dodávány pravidelně v rámci iterativního vývoje, je možné systém postavený na nové architektuře kontinuálně provozovat od září 2021.

8.1 Vývojová prostředí

K zajištění zachování funkčnosti produkční aplikace, kdy jsou v průběhu času kontinuálně sledovaní různí pacienti, je nutné, aby byly na produkční prostředí nasazovány pouze stabilní verze.

K tomu slouží vytvořené vývojové prostředí. Na vývojovém prostředí je možné testovat například integrace s cloudy třetích stran. Další užití vývojového prostředí je možnost vyzkoušení konfigurací a nastavení a jejich fungování na prostředí, které je velmi podobné tomu produkčnímu.

8.2 Monitoring

V rámci provozu byl pravidelně sledován celkový stav systému, tedy jak jeho vytížení, využití operační paměti, či místa na disku, tak monitoring logů a chování systému.

V rámci monitoringu logů a chování systému byly sledovány samotné logy, i `DLT` topicy v Kafka se zprávami, které se nepodařilo zpracovat. Díky monitoringu bylo odhaleno několik chyb týkajících se nepřesné specifikace struktury dat ze strany Garmin, kdy některé např. číselníkové hodnoty aktivit neodpovídají těm, jež jsou specifikovány.

V rámci monitoringu zatížení bylo vyhodnoceno, že stávající zdroje virtuálního stroje (2GB RAM, 10GB filesystém, 1 výpočetní jádro), na kterém běží veškeré kontejnery a tedy celý systém, jsou nedostatečné. Na základě vzneseného požadavku byly virtuálnímu stroji přiděleny silnější zdroje (10GB RAM, 80GB filesystém, 2 výpočetní jádra).

8.3 Aktualizace knihoven

Systém by neměl zastarávat, měl by být dlouhodobě udržovatelný a zůstávat bezpečný díky bezpečnostním záplatám. Proto je nutné pravidelně aktualizovat jednotlivé části systému, na kterých je systém postaven a které používá.

Největší změnou v rámci provozu byla aktualizace použité verze programovacího jazyka Java. Java byla povýšena z verze 11, což byla v počátcích vývoje nejnovější verze s tzv. dlouhodobou podporou na verzi 17. Verze 17 je další verzí s dlouhodobou podporou na několik let.

V rámci aktualizace verze Java bylo nutné aktualizovat i sestavovací nástroj Gradle na aktuální verzi 7.4, neboť nejnižší verze, která podporuje Java 17 je verze 7.3.

knihovna / verze	původní verze	poslední aktualizovaná verze
Java	11	17
Gradle	7.0	7.4.1
Spring Boot	2.5.2	2.6.5
Spring Boot Web	2.5.2	2.6.5
Spring Boot JPA	2.5.2	2.6.5
Spring for Apache Kafka	2.7.6	2.8.4
PostgreSQL	42.3.1	42.3.3
AssertJ	3.21.0	3.22.0

Tabulka 8.1. Přehled verzí aktualizovaných knihoven

V rámci dalších aktualizací byly v průběhu provozu pravidelně aktualizovány také další použité knihovny a nástroje. U některých knihoven se jednalo o několik aktualizací, pro přehlednost jsou však v tabulce 8.1 zobrazeny pouze počáteční a koncové verze jednotlivých knihoven.

Po jakékoliv změně ve verzích knihoven je samozřejmostí, že vždy musí proběhnout kompletní regresní přetestování celé aplikace, neboť úpravy v knihovnách mohou mít dopad do všech částí systému.

Kapitola 9
Závěr

Cílem této práce bylo navrhnout a implementovat technickou infrastrukturu pro sběr a agregaci dat z chytrých náramků výrobce Garmin v existující aplikaci TERESA (TEleREhabilitation Self-training Assistant). Tento cíl byl v průběhu práce rozšířen o sběr dat z chytrých náramků výrobců Fitbit a Xiaomi(Gadgetbridge), stejně jako o tvorbu detailních grafů sesbíraných dat, které mohou být využívány lékaři pro telerehabilitaci a telecoaching.

Prvním krokem této práce byla rešerše nositelné elektroniky. Následoval sběr a analýza požadavků, na jejichž základě byla navrhnuta vhodná architektura.

Navržená architektura je postavena na mikroslužbách s asynchronní komunikací pomocí front zpráv. Toto řešení bylo zvoleno zejména díky požadavkům na výkonnost, robustnost a lehkou rozšiřitelnost v budoucnu.

Celý systém, včetně načítačů dat od jednotlivých výrobců, byl následně implementován. Součástí implementace je:

- vytvoření 2 nových mikroslužeb (Garmin Worker, Fitbit Worker)
- rozšíření a modifikace existující aplikace (agregátor TERESA)
- vyjmutí modulu pro načítání dat z platformy Gadgetbridge do samostatné mikroslužby (Gadgetbridge Worker) tak, aby celé řešení respektovalo navrhnutou architekturu a mělo požadované vlastnosti
- implementace exportu detailních grafů o počtu naměřených kroků, srdeční aktivitě a kvalitě spánku

Celé řešení podstoupilo 3 typy testování. Prvním typem je otestování řešení automatickými jednotkovými a integračními testy pro ověření správné funkčnosti logiky. Následně bylo provedeno zátěžové testování pro ověření výkonnosti a stability celého řešení v případě očekávané zátěže. Zátěžové testování potvrdilo výkonnost a stabilitu řešení při vyšší než očekávané zátěži. Následovalo uživatelské testování, kdy pro každého výrobce (Garmin, Fitbit, Xiaomi) používali 2 uživatelé dané zařízení po dobu 7 dnů tak, jak je očekáváno, že bude používáno v reálném provozu. V rámci uživatelského testování byly zjištěny drobné nedostatky u zpracování dat výrobců Fitbit a Garmin. Všechny zjištěné nedostatky byly opraveny a zapracovány do aplikace.

Řešení by mělo být v budoucnu lehce rozšiřitelné a škálovatelné díky typu zvolené architektury. Vzniklé řešení je tedy připraveno na budoucí plánovaný rozvoj projektu, jako je například rozšíření o další poskytovatele dat, stejně jako k rozšíření o sběr dalších typů dat, která výrobci poskytují.

Řešení je plně funkční a nasazené na vývojovém prostředí v rámci projektu TERESA. Je připraveno pro použití v ostrém provozu v rámci projektu TERESA pro reálné pacienty. Toto použití je plánováno v rámci další skupiny pacientů, jejíž sledování je plánováno na podzim roku 2022.

Literatura

[1] Althaf, Shahana. *Sustainability Implications of Consumer Electronics Adoption in the United States*. Rochester Institute of Technology, 2019.

[2] inc., Apple. *Apple Reinvents the Phone with iPhone*. [vid. 2021-12-09]. Dostupné na `https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/`.

[3] Kaewkannate, Kanitthika a Soochan Kim. A comparison of wearable fitness devices. *BMC Public Health*. Springer Science and Business Media LLC, may, 2016, ročník 16, č. 1. Dostupné na DOI 10.1186/s12889-016-3059-0. Dostupné na `https://doi.org/10.1186/s12889-016-3059-0`.

[4] Wootton, R. Recent advances: Telemedicine. *BMJ*. BMJ, sep, 2001, ročník 323, č. 7312, s. 557–560. Dostupné na DOI 10.1136/bmj.323.7312.557. Dostupné na `https://doi.org/10.1136/bmj.323.7312.557`.

[5] Keating, Andrew, Annemarie Lee a Anne E Holland. What prevents people with chronic obstructive pulmonary disease from attending pulmonary rehabilitation? A systematic review. *Chronic Respiratory Disease*. 2011, ročník 8, č. 2, s. 89-99. Dostupné na DOI 10.1177/1479972310393756. Dostupné na `https://doi.org/10.1177/1479972310393756` . PMID: 21596892.

[6] Nittari, Giulio, Ravjyot Khuman, Simone Baldoni, Graziano Pallotta, Gopi Battineni, Ascanio Sirignano, Francesco Amenta a Giovanna Ricci. Telemedicine Practice: Review of the Current Ethical and Legal Challenges. *Telemedicine and e-Health*. 02, 2020, ročník 26. Dostupné na DOI 10.1089/tmj.2019.0158.

[7] Salawu, Abayomi, Angela Green, Michael Crooks, Nina Brixey, Denise Ross a Manoj Sivan. A Proposal for Multidisciplinary Tele-Rehabilitation in the Assessment and Rehabilitation of COVID-19 Survivors. *International Journal of Environmental Research and Public Health*. 07, 2020, ročník 17, s. 4890. Dostupné na DOI 10.3390/ijerph17134890.

[8] ČVUT. *ČVUT FEL Aktuality Projekt TERESA umožní rehabilitaci pacientů po COVID-19 v domácím prostředí*. [vid. 2021-12-10]. Dostupné na `https://fel.cvut.cz/cz/aktuality/2021/teresa`.

[9] Duking, Peter, Andreas Hotho, Hans-Christer Holmberg, Franz Konstantin Fuss a Billy Sperlich. Comparison of Non-Invasive Individual Monitoring of the Training and Health of Athletes with Commercially Available Wearable Technologies. *Frontiers in Physiology*. Frontiers Media SA, mar, 2016, ročník 7. Dostupné na DOI 10.3389/fphys.2016.00071. Dostupné na `https://doi.org/10.3389/fphys.2016.00071`.

[10] Guler, Sibel Deren, Madeline Gannon a Kate Sicchio. *Crafting Wearables*. Apress, 2016. Dostupné na DOI 10.1007/978-1-4842-1808-2. Dostupné na `https://doi.org/10.1007/978-1-4842-1808-2`.

[11] Club, Your Watch. *The oldest watch in the world (is maybe from 1505)*. Dostupné na https://www.yourwatchhub.com/watches/oldest-watch-in-the-world/.

[12] Alexander, Howard. *Hearing Aids: Smaller and Smarter*. [vid. 2021-12-09]. Dostupné na https://www.nytimes.com/1998/11/26/technology/hearing-aids-smaller-and-smarter.html.

[13] Rahman, Shah Atiqur, Christopher Merck, Yuxiao Huang a Samantha Kleinberg. *Unintrusive eating recognition using Google Glass*. Dostupné na DOI 10.4108/icst.pervasivehealth.2015.259044.

[14] Goodrich, Ryan. *Accelerometers: What They Are How They Work*. [vid. 2022-03-01]. Dostupné na https://www.livescience.com/40102-accelerometers.html.

[15] Aroganam, Gobinath, Nadarajah Manivannan a David Harrison. Review on Wearable Technology Sensors Used in Consumer Sport Applications. *Sensors*. MDPI AG, apr, 2019, ročník 19, č. 9, s. 1983. Dostupné na DOI 10.3390/s19091983. Dostupné na https://doi.org/10.3390/s19091983.

[16] Brown, Simon. *Modular Monoliths*. [vid. 2022-03-03]. Dostupné na https://files.gotocon.com/uploads/slides/conference_12/515/original/gotoberlin2018-modular-monoliths.pdf.

[17] Yetisen, Ali K., Juan Leonardo Martinez-Hurtado, Barş Unal, Ali Khademhosseini a Haider Butt. Wearables in Medicine. *Advanced Materials*. Wiley, jun, 2018, ročník 30, č. 33, s. 1706910. Dostupné na DOI 10.1002/adma.201706910. Dostupné na https://doi.org/10.1002/adma.201706910.

[18] Arriba-Perez, Francisco de, Manuel Caeiro-Rodriguez a Juan Manuel Santos-Gago. Towards the use of commercial wrist wearables in education. In: *2017 4th Experiment@International Conference (exp.at17)*. IEEE, 2017. Dostupné na DOI 10.1109/expat.2017.7984354. Dostupné na https://doi.org/10.1109/expat.2017.7984354.

[19] Nascimento, Bruno, Tiago Oliveira a Carlos Tam. Wearable technology: What explains continuance intention in smartwatches?. *Journal of Retailing and Consumer Services*. Elsevier BV, jul, 2018, ročník 43, s. 157–169. Dostupné na DOI 10.1016/j.jretconser.2018.03.017. Dostupné na https://doi.org/10.1016/j.jretconser.2018.03.017.

[20] inc., Apple. *Apple Watch*. [vid. 2021-12-12]. Dostupné na https://www.apple.com/cz/watch/.

[21] Kaewkannate, Kanitthika a Soochan Kim. A comparison of wearable fitness devices. *BMC Public Health*. 05, 2016, ročník 16. Dostupné na DOI 10.1186/s12889-016-3059-0.

[22] Miner, Cameron S., Denise M. Chan a Christopher Campbell. Digital jewelry. In: *CHI 01 extended abstracts on Human factors in computing systems - CHI 01*. ACM Press, 2001. Dostupné na DOI 10.1145/634067.634098. Dostupné na https://doi.org/10.1145/634067.634098.

[23] Koskimaki, Heli, Hannu Kinnunen, Teemu Kurppa a Juha Roning. How Do We Sleep. In: *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and*

Wearable Computers. ACM, 2018. Dostupné na DOI 10.1145/3267305.3267697. Dostupné na https://doi.org/10.1145/3267305.3267697.

[24] RING, Oura. *Oura ring*. [vid. 2021-12-14]. Dostupné na https://ouraring.com/.

[25] FINNI, Taija, Min HU, Pasi KETTUNEN, Toivo VILAVUO a Sulin CHENG. Validity, reliability and feasibility of measuring muscle activity with textile electrodes embedded into clothing. *Journal of Biomechanics*. New York: Pergamon Press, 1968-, 2007, ročník 40, č. 2, s. S217.

[26] MICROSOFT. *HoloLens 2*. [vid. 2022-03-01]. Dostupné na https://www.microsoft.com/en-us/d/hololens-2/91pnzzznzwcp?activetab=pivot:overviewtab.

[27] COMMUNITY, Biomedical Engineering. *Wearable Implantable Technologies*. [vid. 2021-12-14]. Dostupné na https://www.embs.org/about-biomedical-engineering/our-areas-of-research/wearable-implantable-technologies/.

[28] DUNN, Jessilyn, Ryan RUNGE a Michael SNYDER. Wearables and the medical revolution. *Personalized Medicine*. Future Medicine Ltd, sep, 2018, ročník 15, č. 5, s. 429–448. Dostupné na DOI 10.2217/pme-2018-0044. Dostupné na https://doi.org/10.2217/pme-2018-0044.

[29] WIEGERS, Karl a Joy BEATTY. *Software requirements*. Pearson Education, 2013.

[30] XIAOMI. *Mi Band 5*. [vid. 2022-03-09]. Dostupné na https://www.xiaomi.cz/xiaomi-mi-band-5/.

[31] RICHARDS, Mark. *Fundamentals of software architecture : an engineering approach*. Sebastopol, CA: O'Reilly Media, 2020. ISBN 9781492043454.

[32] STRIMBEI, Catalin, Octavian DOSPINESCU, Roxana STRAINU a Alexandra NISTOR. Software Architectures – Present and Visions. *Informatica Economica Journal*. 12, 2015, ročník 19, s. 13-27. Dostupné na DOI 10.12948/issn14531305/19.4.2015.02.

[33] SAVOLAINEN, Juha a Varvana MYLLARNIEMI. Layered architecture revisited — Comparison of research and practice. 2009, s. 317-320. Dostupné na DOI 10.1109/WICSA.2009.5290685.

[34] BAELDUNG. *Layered Architecture*. [vid. 2022-03-03]. Dostupné na https://www.baeldung.com/cs/layered-architecture.

[35] RICHARDS, Mark. *Software architecture patterns*. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA ..., 2015.

[36] GRACE JANSEN, Johanna Saladas. *Advantages of the event-driven architecture pattern*. [vid. 2022-05-04]. Dostupné na https://developer.ibm.com/articles/advantages-of-an-event-driven-architecture/.

[37] RICHARDSON, Chris. *Pattern: Microservice Architecture*. [vid. 2022-05-04]. Dostupné na https://microservices.io/patterns/microservices.html.

[38] SOMMERLAD, Peter. Reverse Proxy Patterns.. In: *EuroPLoP*. 2003. s. 431–458.

[39] INSTACLUSTR. *RabbitMQ vs. Apache Kafka: Key Differences and Use Cases*. [vid. 2022-03-11]. Dostupné na https://www.instaclustr.com/blog/rabbitmq-vs-kafka/.

[40] TutorialsPoint. *Apache Kafka - Cluster Architecture*. [vid. 2022-03-11]. Dostupné na https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm.

[41] TutorialsPoint. *Apache Kafka - Fundamentals*. [vid. 2022-03-12]. Dostupné na https://www.tutorialspoint.com/apache_kafka/apache_kafka_fundamentals.htm.

[42] javaTpoint. *Spring Boot Architecture*. [vid. 2022-03-11]. Dostupné na https://www.javatpoint.com/spring-boot-architecture.

[43] Docker. *Use containers to Build, Share and Run your applications*. [vid. 2022-03-14]. Dostupné na https://www.docker.com/resources/what-container.

[44] Program, Ebay Developers. *The authorization code grant flow*. [vid. 2022-03-17]. Dostupné na https://developer.ebay.com/api-docs/static/oauth-authorization-code-grant.html.

[45] Monkov, Anthony. *Angular: Using httpinterceptor for token refreshing*. Dostupné na https://medium.com/@monkov/angular-using-httpinterceptor-for-token-refreshing-3f04ea2ccb81.

[46] Boe, Alexander, Lori Koch, Megan O'Brien, Nicholas Shawen, John Rogers, Richard Lieber, Kathryn Reid, Phyllis Zee a Arun Jayaraman. Automating sleep stage classification using wireless, wearable sensors. *npj Digital Medicine*. 12, 2019, ročník 2, s. 131. Dostupné na DOI 10.1038/s41746-019-0210-1.

[47] Bures, Miroslav, Miroslav Renda, Michal Dolezel a others. *Efektivn testovan softwaru: kl cove otazky pro efektivitu testovac ho procesu*. Grada Publishing as, 2016.

[48] Bures, Miroslav, Tomas Cerny a Matej Klima. Prioritized Process Test: More Efficiency in Testing of Business Processes and Workflows. In: Kuinam Kim a Nikolai Joukov, editoři. *Information Science and Applications 2017*. Singapore: Springer Singapore, 2017. s. 585–593. ISBN 978-981-10-4154-9.

[49] Olan, Michael. Unit Testing: Test Early, Test Often. *J. Comput. Sci. Coll.* Evansville, IN, USA: Consortium for Computing Sciences in Colleges, dec, 2003, ročník 19, č. 2, s. 319–328. ISSN 1937-4771.

[50] George, Boby a Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*. 2004, ročník 46, č. 5, s. 337-342. ISSN 0950-5849. Dostupné na DOI https://doi.org/10.1016/j.infsof.2003.09.011. Dostupné na https://www.sciencedirect.com/science/article/pii/S0950584903002040. Special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003.

[51] Louridas, P. Static code analysis. *IEEE Software*. 2006, ročník 23, č. 4, s. 58-61. Dostupné na DOI 10.1109/MS.2006.114.