FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Aplikace pro osobní evidenci zdravotní péče

Studijní program: B0613A140005 Informační technologie
Studijní obor: Aplikovaná informatika

Autor práce: **Vojtěch Voleman**
Vedoucí práce: Ing. Jana Vitvarová, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Aplikace pro osobní evidenci zdravotní péče

Jméno a příjmení:	**Vojtěch Voleman**
Osobní číslo:	M20000081
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2022/2023

Zásady pro vypracování:

1. Proveďte rešerši existujících aplikací pro evidenci osobní zdravotní péče.
2. Analyzujte potřeby pacientů při vybraných akutních i chronických zdravotních problémech.
3. Analyzujte využitelná veřejně dostupná zdravotnická data, jako například informace o lécích, či kódové označení nemocí. Dále zdroje dat, které má pacient k dispozici, jako například přehledy zdravotní péče od zdravotní pojišťovny nebo papírové lékařské zprávy.
4. Vytvořte funkční specifikaci aplikace dle zjištěných potřeb.
5. Navrhněte technické řešení a vyberte platformu s důrazem na bezpečné nakládání s citlivými daty.
6. Vytvořte interaktivní prototyp uživatelského rozhraní.
7. Implementujte a testujte prototyp aplikace procesem CI/CD.
8. Proveďte uživatelské testování a výsledky vyhodnoťte.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 až 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] ECKEL, Bruce a Svetlana ISAKOVA. Atomic Kotlin. Mindview, 2021. ISBN 978-0-9818725-5-1.
[2] MARTIN, Robert C. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson, 2008. ISBN 0132350882.
[3] MKN-10 klasifikace [online]. [cit. 2022-10-12]. Dostupné z: https://mkn10.uzis.cz

Vedoucí práce:	Ing. Jana Vitvarová, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2022
Předpokládaný termín odevzdání:	15. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 12. října 2022

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

14. května 2024 Vojtěch Voleman

Aplikace pro osobní evidenci zdravotní péče

Abstrakt

Tato práce se zabývá návrhem a vývojem mobilní aplikace pro osobní evidenci zdravotní péče. Návrh řešení vychází z rešerše problematiky přístupu ke zdravotnickým datům pacienta a konzultací s cílovými uživateli. Jedním ze cílů této práce bylo prozkoumat veřejně přístupné datové zdroje ve zdravotnictví a jejich využití v aplikaci. Mobilní aplikace byla vyvíjena v jazyce Kotlin podle systémové architektury Clean Architecture. Pro zpracování a zpřístupnění veřejných datových zdrojů byl vytvořen vzdálený API server v jazyce PHP a frameworku Symfony.

Klíčová slova: mobilní aplikace, Android, PHR, Clean Architecture, veřejné datové zdroje, PHP

Personal healthcare app

Abstract

This thesis deals with the design and development of a mobile app for personal healthcare records. The design is based on the research of the patient's medical data access and consultations with targeted users. One of the objectives was to explore open data in the healthcare and use them in the app. The mobile app was developed in Kotlin following the Clean Architecture system architecture. A remote API server was created for processing and accessing open data, developed in PHP and Symfony framework.

Keywords: mobile app, Android, PHR, Clean Architecture, open data, PHP

Poděkování

Rád bych poděkoval paní Ing. Janě Vitvarové, Ph.D, za cenné rady, skvělé připomínky a trpělivost při vedení této práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam zkratek

PHR	Personal Healthcare Records (osobní zdravotnické záznamy)
MVVM	Model-View-ViewModel
DI	Dependency Injection
DLP	Databáze léčivých přípravků
MKN-10	Mezinárodní klasifikace nemocí a souvisejících zdravotních problémů, verze 10
NRPZS	Národní registr poskytovatelů zdravotních služeb
SÚKL	Státní úřad pro kontrolu léčiv
ÚZIS	Ústav zdravotnických informací a statistiky
DASTA	Český národní datový standard pro výměnu informací ve zdravotnictví

1 Úvod do problematiky

1.1 Zdravotnická dokumentace

Ve svém životě člověk navštíví různé poskytovatele zdravotní péče. Každý poskytovatel má povinnost uchovávat o pacientovi záznamy, které se poskytované péče týkají. Mezi tyto záznamy patří údaje o pacientovi a jeho zdravotním stavu, informace o diagnóze, návrh léčebného postupu, záznam o rozsahu poskytnutých nebo vyžádaných služeb, záznam o předepsaných či podaných lécích. (1) Všechny tyto záznamy, dokumenty a informace se obecně nazývají zdravotní dokumentace a její specifika a vedení jsou stanoveny v části 6 zákona č. 372/2011 Sb. Zdravotní dokumentaci lze dle § 54 daného zákona vést v listinné nebo elektronické podobě nebo v kombinaci obou těchto podob. (1) Doba uchovávání je upravena přílohou č. 3 k vyhlášce č. 98/2012 Sb., například:

- 10 let od změny praktického lékaře,
- 5 let po posledním poskytnutí zdravotních služeb pacientovi v oboru zubní lékařství, gynekologie nebo porodnictví,
- 40 let od ukončení poslední hospitalizace pacienta v nemocnici, či 10 let od jeho úmrtí.

Zdravotní stav je komplexní záležitost, problémy pacienta často nespadají jen do jedné či druhé lékařské disciplíny a je tedy potřeba součinnost více lékařských pracovníků. Každý z nich si o pacientovi vede vlastní záznamy, posílá ho na testy, jejichž výsledky má u sebe. Problém nastává při předávání zdravotnické dokumentace mezi poskytovateli.

Často se pro předávání zpráv a jiných informací využívají poštovní služby. Nejedná se však o pravidlo, a nezřídkakdy dochází k předání části zdravotnické dokumentace přes pacienta, tzn. že pacient je nucen sebou nosit své zprávy a výsledky. Praktický lékař nemusí ani vědět o všech vyšetřeních a testech pacienta, což může vést ke zhoršení zdravotní péče a docházet k duplicitnímu provádění vyšetření či laboratorních testů. To vede některé pacienty, zejména ty s dlouhodobými zdravotními komplikacemi, k archivaci svých zdravotních dokumentů a nošení jich sebou k lékaři.

Pro výměnu obrazové dokumentace existuje v České republice od roku 2008 výměnná síť ePACS. Poskytovatelé, kteří jsou k této síti připojeni, si mezi sebou mohou obrazovou dokumentaci vyměňovat v řádu sekund. V dubnu 2024 bylo k síti

připojeno 716 poskytovatelů. (2) Bohužel, stále se lze setkat se situací, kdy se obrazové záznamy mezi poskytovateli vyměňují např. prostřednictvím CD.

Samostatnou kapitolou je pak využití zdravotnické dokumentace pro účely záchranné zdravotnické služby. V současné době neexistuje jednotný způsob, jak by si záchranáři mohli zjistit detaily o zdravotním stavu pacienta. V rámci projektu eMeDOcS, vyvíjený krajem Vysočina, si může ZZS vyžádat urgentní informace o pacientovi prostřednictvím tzv. Emergency Card. Tyto informace jsou zprostředkovány jednotlivými zapojenými nemocnicemi do sanitního vozu v řádu několika sekund. (3) Projekt je to cenný, ale vzhledem k tomu, že většina zapojených poskytovatelů jsou nemocnice, nelze čekat detailní informace o pacientovi jako od praktického lékaře.

Většinou tedy záchranáři získávají informace přímo od pacienta, jeho blízkého okolí, či ze seznamu alergií a nemocí, které někteří lidé mohou mít u sebe (například pacienti s dlouhodobými zdravotními komplikacemi).

1.2 Přístup ke zdravotnickým datům

Právo nahlížet do zdravotnické dokumentace konkrétního poskytovatele má dle § 65 zákona 372/2011 Sb. sám pacient či jeho zákonný zástupce, popřípadě osoba, která k tomu byla určena. (4) Může tak učinit po domluvě se zdravotníky a za přítomnosti osoby pověřené poskytovatelem zdravotních služeb. Pacient si také může udělat kopie v elektronické podobě (pomocí fotoaparátu nebo mobilního telefonu), tato varianta je zdarma. Pacient má také možnost si vyžádat kopii dokumentace, která musí být vyhotovena do 30 dnů od vyžádání. Poskytovatel může po pacientovi požadovat náhradu, která však nesmí překročit náklady spojené s pořízením.

Určitý způsob získání informací o svém zdravotním stavu nabízí Portál občana. V záložce "Zdravotnictví" lze najít možnost "Zdravotnická dokumentace". Ta nabízí výpis ze zdravotní dokumentace od poskytovatelů, kteří jsou připojeni k Národnímu kontaktnímu místu pro elektronické zdravotnictví (NCPeH), který zabezpečuje bezpečný přenos zdravotnické dokumentace mezi poskytovateli v ČR i přeshraničně v EU (projekt MyHealth@EU). V dubnu 2024 bylo k NCPeH připojeno 16 poskytovatelů a dalších deset je v přípravě. (5)

Je nutné si ale uvědomit, že formát zdravotnické dokumentace, která je přes NCPeH k dispozici, se liší podle poskytovatele. Většina z nich sdílí tzv. Pacientský souhrn, viz příloha A.1, jiní jen propouštěcí zprávu či záznam o výjezdu.

Jako možný způsob přístupu ke svým zdravotnickým datům je portál eRecept, který je také přístupný přes Portál občana. V něm lze dohledat své existující či starší recepty, nebo aplikované vakcíny v sekci očkování. Do té ovšem mohli lékaři zapisovat až od 1. ledna 2022, očkování provedené v minulosti tam tedy často nejsou k dohledání (pacient má možnost lékaře zažádat o zpětné doplnění). Od 30. června 2022 je zápis do eOčkování povinný. (6)

Mimo tyto státem spravované řešení existují také portály zdravotních pojišťoven. Stát sice zamýšlel mít jednotný portál, kde by pojištěnec viděl informace o vykázané zdravotní péči (7, s. 53), ale neučinil k tomu žádné kroky. Možnosti portálů se liší

dle pojišťoven, ale základní funkce jsou velmi podobné. Jedná se výpisy péče (jací poskytovatelé si zažádali o úhradu jakých činností), podání žádostí (jak administrativních úkonů, tak o příspěvek z fondu prevence) a komunikace s pojišťovnou.

I když v České republice neexistuje jednotné řešení elektronizace zdravotnictví, jednotliví poskytovatelé mohou svým pacientům podobné služby nabízet, samozřejmě v omezeném rozsahu. Například Krajská zdravotní, a.s., provozovatel sedmi nemocnic v Ústeckém kraji, dokáže sdílet data pacientů mezi svými nemocnicemi. Provozuje také "Portál pacienta Ústeckého kraje", kde si pacient po osobní registraci může zobrazit svůj souhrn, výsledky vyšetření EKG a příjmové, propouštěcí i ambulantní zprávy. (8)

Dalším podobným příkladem je poskytovatel primární péče Moje Ambulance, která má přes 30 poboček praktických lékařů. Výhodou tohoto spojení je mobilní aplikace, která je názornou ukázkou tzv. Personal Healthcare Records aplikace. Umožňuje pacientovi nahlédnout do své zdravotnické dokumentace, výsledků laboratorních testů, vidět předepsané léky, kartu svého zdravotního stavu. Mimo to také umožňuje řešit administrativních úkony přímo z aplikace.

1.3 PHR

Z výše uvedených podkapitol vyplývá, že poskytování zdravotní péče v České republice může trpět nedostatečným propojením zdravotních dokumentací pacienta. Řešením tohoto problému může být PHR (Personal Healthcare Records) – v češtině také osobní evidence zdravotní péče.

Pojmy PHR a „patient-oriented" zdravotní péče se začaly ve světě používat v průběhu 70. let minulého století (9, s. 2), ale větší pozornosti se dočkaly až počátkem tohoto tisíciletí s rozvojem výpočetní techniky a legislativních změn – ve Spojených státech například HITECH Act z roku 2009. (10)

Pro pojem neexistuje jednotná definice, ale mezi klíčové body patří souhrnné informace o zdravotním stavu jednotlivců, jejich lékařské záznamy a další indikátory zdravotního stavu. V odborné literatuře se lze například setkat s rozdělením PHR aplikací na několik typů – Tethred, Stand-alone a Interconnected. (11, s. 11-13)

Aplikace odpovídající typu Tethred (česky provázané) jsou přímo provázány s konkrétním poskytovatelem zdravotní péče. Všechna data do aplikace proudí jen od tohoto poskytovatele a nelze ji využívat bez něho.

Typ Stand-alone (v češtině samostatná) je specifický tím, že není závislý na systémech poskytovatelů péče, pro výměnu dat však může existovat rozhraní. Tyto aplikace přímo nahrazují fyzické složky, které si pacienti mohou ze svých záznamů vytvářet.

Jako Interconnected (česky navzájem propojené) lze označit aplikace, které komunikují s vícero systémy poskytovatelů zdravotních služeb. Typicky tyto aplikace mohou být jednotným řešením vyvíjeným státní správou.

1.4 Existující řešení

1.4.1 Moje Ambulance

Moje Ambulance je síť poskytovatelů primární péče v České republice, která pro své pacienty nabízí mobilní aplikaci „Moje Ambulance“ (dostupné pro Android i iOS). (12) Prostřednictvím této aplikace mají pacienti přístup ke svým lékařským zprávám, předepsaným lékům, naměřeným hodnotám a dalším částem zdravotní dokumentace. Mohou také provádět administrativní úkony bez návštěvy lékaře, jako například zažádat o potvrzení k řidičskému oprávnění, o invalidní důchod či o neschopenku.

Tuto aplikaci lze označit jako typ Tethred, jelikož je přímo vázaná na data a záznamy praktického lékaře.

1.4.2 Health (iOS)

Health (česky „Zdraví“) je výchozí mobilní aplikace pro zařízení s operačním systémem iOS. Ta uživatelům umožňuje automatické sledování dat o jejich zdraví – sledování kroků, tepové frekvence, kvality spánku, krevního tlaku apod. Nabízí také možnost ručního zadávání hodnot – hladina glukózy v krvi, váha, jiné tělesné míry a změny nálad. Uživatele také může upozorňovat na pobírané léky, ve Spojených státech dokonce zvládne aplikace rozpoznat tzv. „lékové interakce“ (situaci, kdy dva či více přípravků ovlivňuje působení a účinky alespoň jednoho z nich) (13).

Významnou přidanou hodnotou aplikace je možnost vytvořit tzv. zdravotní ID. Jedná se o sadu informací majitele zařízení, která má za cíl poskytnout důležité informace v případě krizové situace. Lze zde uvést údaje o majiteli, zdravotní problémy, alergie, léky či kontakt na blízkou osobu. Zdravotní ID je po zapnutí k dispozici na zamykací obrazovce zařízení pod tlačítkem „krizová situace“.

Aplikaci lze označit jako typ Stand-alone, jelikož ke své funkčnosti nepotřebuje připojení k žádnému poskytovateli péče.

2 Specifikace požadavků

Pro efektivní vypracování projektu je potřeba si správně určit cílové uživatele a jejich priority. S těmito informacemi je možné určit požadavky pro aplikaci – ať už z hlediska funkčnosti či přístupnosti. Při vytváření těchto požadavků byly konzultování možní cíloví uživatelé, podle jejich zpětné vazby byly požadavky upravovány.

2.1 Cíloví uživatelé

2.1.1 Uživatel se zdravotní indispozicí

Jako „Uživatel se zdravotní indispozicí“ se považuje uživatel s chronickými či jinými dlouhodobými problémy. Tato skupina uživatelů často dochází do lékařských zařízení, je jim předepisováno větší množství léků a je od nich vyžadována vlastní evidence zdravotního stavu (typicky krevní tlak, tep apod.).

Tito uživatelé si přirozeně v těchto materiálech a úkonech dělají vlastní systém a pořádek. Jelikož dochází k více lékařským pracovníkům, je pro ně důležité rozdělovat lékařské zprávy, léky a měření podle konkrétních lékařských pracovníků či dle zdravotního problému.

Je důležité brát v potaz, že větší část těchto uživatelů nemusí být zdatná v moderních technologiích. Proto je pro ně důležité, aby uživatelské rozhraní a struktura aplikace byla co nejsrozumitelnější.

2.1.2 Běžný uživatel

Jako „Běžný uživatele“ se považuje uživatel, který nemá žádné dlouhodobé zdravotní problémy, za to ale má rád systematičnost a pořádek ve svých datech.

Rád si lékařské dokumenty, které se mu dostanou do rukou, archivuje a při potřebě je podle některých kritérií vyhledá.

Uživatel si zapisuje termíny svých pravidelných kontrol, případně ho zajímá historie jeho návštěv u konkrétního lékaře.

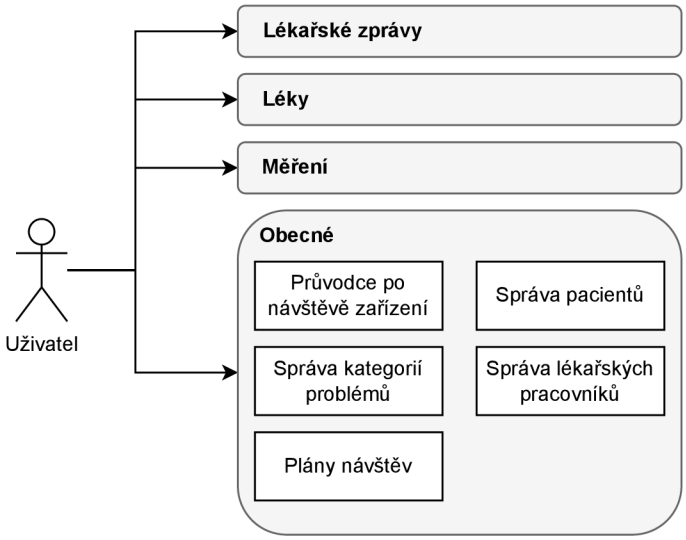
2.1.3 Pečovatel

Jako „Pečovatel“ se považuje uživatel, který se stará například o rodinného příslušníka. Potřebuje mít po ruce přehled léků pro příslušníka, chce být upozorňován, že je čas léky podat, či chce znát seznam léků s dávkováním v určitém časovém intervalu.

Dále potřebuje evidovat zdravotní stav příslušníka (teplota, tlak apod.) a nechat si měření připomenout. Pečovatel se ale takto musí starat i o více lidí (např. dětí) – je tedy pro něj důležité, aby měl data každého pacienta odděleně.

2.2 Funkční požadavky

Ze specifikace cílových uživatelů vyplynuly potřeby, které jsou v aplikaci žádané. Dalším krokem ve specifikaci požadavků je definice konkrétních případů užití (Use-case). Tyto případy se dělí dle funkčnosti do celků – v rámci této práce označovaných jako moduly. Požadavky, které jednoznačně nepatří jen k jedné funkčnosti, tvoří jeden obecný modul.



Obrázek 2.1: Rozdělení požadavků na moduly

2.2.1 Lékařské zprávy

V rámci modulu "Lékařské zprávy" vyplynuly tyto případy (obrázek A.2):

1. UC-01: Přidání lékařské zprávy
2. UC-02: Filtrování lékařských zpráv
3. UC-03: Export lékařských zpráv

Modul zahrnuje kromě uživatele i druhého aktéra v podobě API serveru. Tento aktér poskytuje seznam diagnóz ve strojově čitelném formátu.

2.2.2 Léky

V rámci modulu "Léky" vyplynuly tyto případy (obrázek A.3):

1. UC-05: Přehled všech léků

2. UC-06: Výpis léků v časovém okně
3. UC-07: Zobrazení detailu léku
4. UC-08: Přidání léku
5. UC-09: Přepnutí upozorňování léku
6. UC-10: Export seznam léků

Modul zahrnuje dva dodatečné aktéry: API server, který poskytuje seznam léků ve strojově čitelném formátu, a Správce upozornění, který nabízí všem aplikacím rozhraní pro práci s upozorněními systému (AlertManager).

2.2.3 Měření

V rámci modulu "Měření" vyplynuly tyto požadavky (obrázek A.4):

1. UC-11: Vytvoření skupiny měření
2. UC-12: Přidání hodnoty měření
3. UC-13: Přehled výsledků měření
4. UC-14: Přehled skupin měření
5. UC-15: Export měření

Modul obsahuje jednoho dalšího aktéra Správce upozornění, který nabízí všem aplikacím rozhraní pro práci s upozorněními systému (AlertManager)

2.2.4 Průvodce po návštěvě zařízení

Tento podmodul je součástí obecného modulu, do kterého patří případy užití, které nelze jednoznačně zařadit do jednoho modulu. Úkolem tohoto podmodulu je nabídnout uživateli možnost zadat všechna data, která se týkají konkrétní návštěvy lékařského zařízení, viz A.5

1. UC-16: Průvodce po návštěvě zařízení

2.2.5 Správa pacientů

Případy spadající do tohoto obecného podmodulu nabízejí uživateli možnost spravovat „pacienty" – v kontextu aplikace uživatele, viz A.6

1. UC-17: Výpis všech pacientů
2. UC-18: Přidání/Úprava uživatele
3. UC-19: Přepnutí uživatele

2.2.6 Správa kategorií problémů

Kategorie problémů slouží jako "obálka", do které lze přiřadit lékařské zprávy, léky či měření. Umožňuje uživateli mít informace o konkrétním problému na jednom místě, a případně je i jednotně exportovat, viz A.7

1. UC-20: Výpis všech kategorií
2. UC-21: Přidání/Úprava kategorie
3. UC-22: Přehled konkrétní kategorie

2.2.7 Správa lékařských pracovníků

O tomto podmodulu lze uvažovat jako o „seznamu kontaktů" – v případě potřeby lze rychle dohledat kontakt na lékařského pracovníka.

Lékařský pracovník nemusí být nutně fixován na jednom konkrétním pracovišti, aplikace by měla tuto skutečnost reflektovat. Měla by umožnit uživateli evidovat si lékařské pracovníky a jejich kontaktní údaje vázány na lékařské zařízení, viz A.8

1. UC-23: Výpis lék. pracovníků
2. UC-24: Přidat/Upravit pracovníka
3. UC-25: Detail pracovníka

2.2.8 Plány návštěv

Tento podmodul slouží jako zápisník budoucích lékařských návštěv, které si pacient může zaznamenat. Při zápisu lze uvést konkrétního lékařského pracovníka, kategorii problému a frekvenci upozornění, viz A.9

1. UC-26: Kalendář návštěv
2. UC-27: Přidat návštěvu
3. UC-28: Exportovat návštěvu do externí kalendářové aplikace

2.3 Nefunkční požadavky

Nefunkční (z angličtiny non-functional) požadavky jsou ty, které se netýkají funkcionality, ale vlastností systému – spolehlivosti, výkonu, údržby, závislosti na jiný software či hardware, uživatelské přívětivosti, přístupnosti apod. (14, s. 2). Při definování těchto požadavků se často pokládají otázky: *K čemu software slouží? Kdo tento software bude využívat? Jaká jsou jeho specifika?*

2.3.1 Přístupnost

S ohledem na zamýšlenou skupinu uživatelů se musí brát v potaz i možné zdravotní indispozice, které uživatelé mohou mít. Operační systém Android poskytuje řadu služeb pro zlepšení přístupnosti.

Mezi takovou službu patří TalkBack, který umožňuje předčítání prvků a textů viditelné v zařízení. Pro správné fungování je potřeba řádně splňovat některé „best practices“, mezi které patří například řádné popisování prvků v aplikaci. (15)

2.3.2 Srozumitelné UI

Správně navrženou aplikaci by uživatel měl být schopen rychle pochopit a využívat i bez návodu. Toho lze docílit konzistentním uživatelským rozhraním, správně zvolenou barevnou paletou (definování primární a sekundární barvy včetně odstínů, barevné odlišení rozdílných funkcí) a také prostřednictvím vhodné typografie a ikonografie.

2.3.3 Konzistentní UI

Konzistence lze dosáhnout tím, že prvky s podobnou funkcí mají stejný nebo velmi podobný vzhled a chování. Typicky se jedná o jednotný vzhled formulářů, seznamů, přehledů a dalších prvků. Výsledkem toho je, že uživatel bude prvky, se kterými se ještě nesetkal, jednodušeji využívat. Poskytne mu to také důvěru ve fungování aplikace a zlepší celkový zážitek.

Dále lze také tvrdit, že konzistentní prvky uživatelského rozhraní poskytují solidní základ pro lepší přístupnost aplikace.

2.3.4 Barevná paleta

Klíčovým úkolem při tvorbě aplikace je také výběr vhodné barevné palety. Ta by měla vycházet z toho, čím se aplikace zabývá.

Aplikace vyvíjena v této práci má zdravotnickou tématiku. Ve zdravotnickém prostředí často potkáme bílou, modrou, a zelenou barvu. Aplikace by to tedy měla zohlednit a zvolit variantu jedné z těchto barev jako primární.

2.4 Požadavky na systém

Mobilní aplikace vyvíjená v této práci počítá s následujícími minimálními parametry zařízení:

1. mobilní zařízení s operačním systémem Android,
2. verze operačního systému alespoň Android 8 (SDK 26),
3. lokální úložiště v zařízení, velikost je úměrná využívání aplikace a
4. internetové připojení.

3 Návrh řešení

Po specifikaci požadavků nyní přichází na řadu vytvoření návrhu řešení aplikace. Rozumí se tím definování klíčových aspektů vzhledu, chování a funkčnosti aplikace. Jde o poslední část před samotným vývojem.

3.1 UI/UX

Pojmy UX (User eXperience) a UI (User Interface) se sice často používají společně ale každý znamená něco jiného. UX design se zaměřuje na interakci programu a uživatele. Cílem je jít uživateli „naproti“ a co nejvíce mu zjednodušit práci s aplikací. Uživatel by se měl přirozeně v aplikaci orientovat a vědět, co se děje.

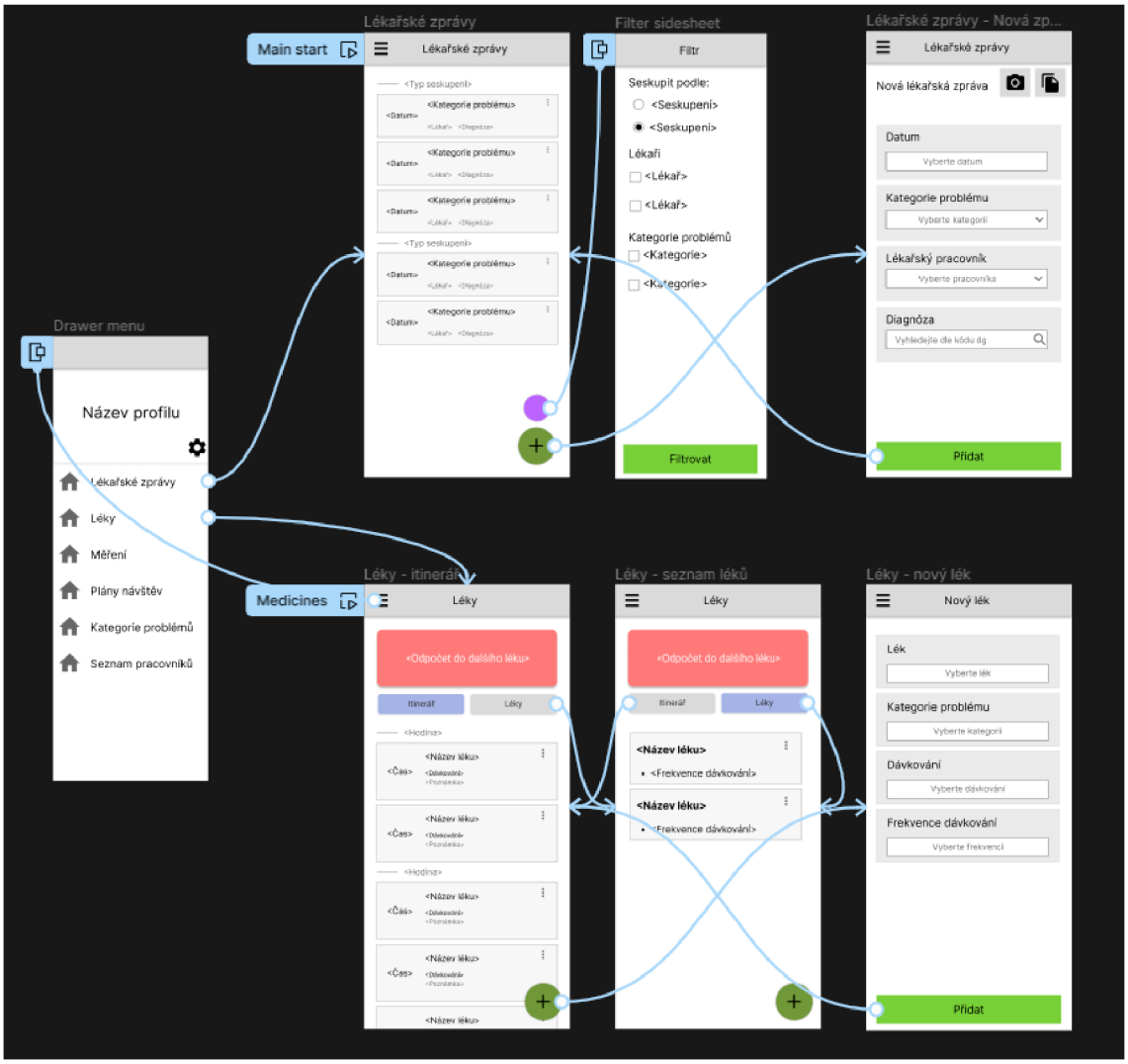
3.1.1 Wireframe

Wireframe je typ návrhu UX, který slouží k definování základní kostry aplikace. Neklade se zde důraz na barvy, fonty a další vizuální aspekty, ale čistě na strukturu a rozložení prvků. Výsledkem může být klikatelný prototyp, který umožňuje jak klientovi, tak vývojáři potvrdit si vzájemné pochopení konceptu aplikace. Po schválení wireframu následuje tvorba prototypu uživatelského rozhraní ve formě tzv. mockupu.

Pro tvorbu wireframu může postačit tužka a papír, ale lze využít řadu dostupných nástrojů (Adobe XD, Balsamiq…). Pro tuto práci byla využita aplikace Figma, která umožňuje i další kroky návrhu uživatelského rozhraní.

V průběhu definování funkčních požadavků byl vypracováván orientační wireframe. Ten umožňoval kontrolu, jak by bylo možné dané požadavky v aplikaci provést a zda mezi požadavky něco nechybí. Výsledkem byl klikatelný prototyp obsahující hlavní moduly aplikace. Moduly jsou si co do struktury podobné, vždy obsahují výpis informací, interakci s položkami a formulář přidávající nové a upravující existující položky.

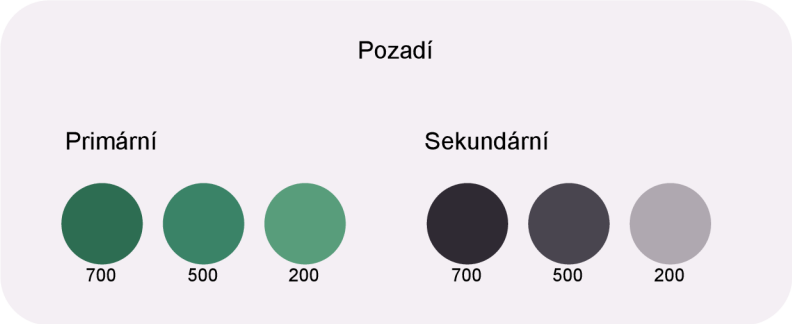
Při aplikování funkčních požadavků také vyplynula potřeba zahrnout různé navigační prostředky – hlavní postranní menu, menu karet (tabs) a plovoucí akční tlačítka (floating action buttons).



Obrázek 3.1: Ukázka části wireframu

3.1.2 Vizuální identita

Při tvorbě barevné palety byla dodržena dostatečná kontrastnost mezi primární a sekundární barvou. Obě tyto barvy se skládají ze 3 odstínů – 700 (nejtmavší), 500 a 200 (nejsvětlejší). Primární barva je v aplikaci využívána pro prvky, které mají získat pozornost uživatele. Sekundární barva se poté vyskytuje například u výčtu položek.



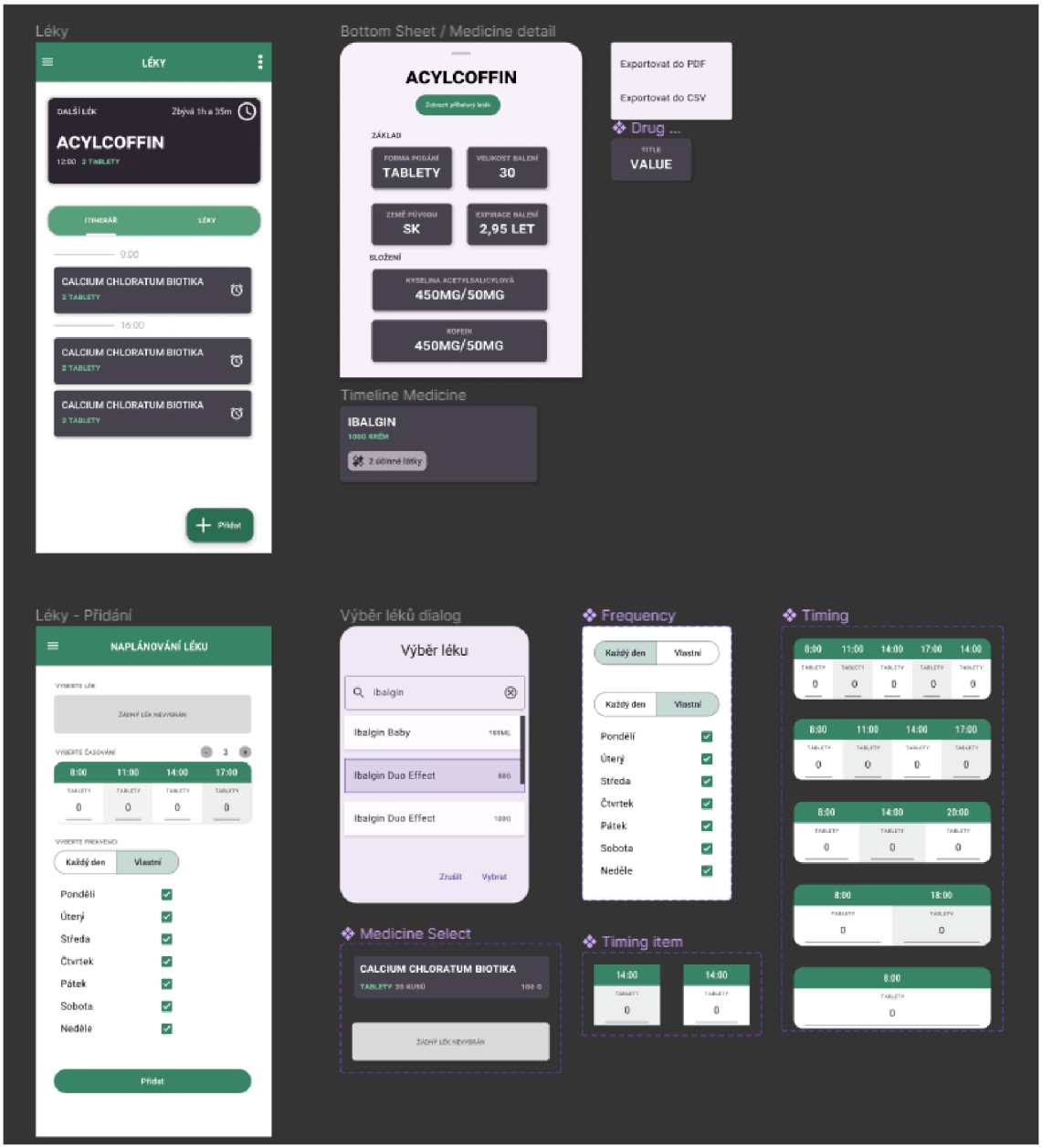
Obrázek 3.2: Barevná paleta

Jelikož aplikace vytvářená v této práci slouží hlavně jako prototyp, nebylo pro ni vytvářeno logo a jako prozatímní název je využíváno jednoduché „PHR“. V případě možného budoucího vývoje se tyto části vizuální identity samozřejmě doplní.

3.1.3 Mockup

Dalším krokem při vývoji je vytvoření mockupu. Jeho cílem je stanovení finálního vzhledu aplikace ještě před tím, než se na uživatelském vzhledu v aplikaci začne pracovat. To dává všem zúčastněným stranám možnost se vyjádřit k vizuálním konceptům. Dobře vytvořený mockup zásadně ulehčuje vývoj UI. Klíčové je definování a pevné ukotvení barev, typografie a znovupoužitelné komponenty.

Pro vývoj mockupu byl znovu využit nástroj Figma, zejména jsem zde ocenil možnost definování komponent. Rozdělení UI na znovupoužitelné komponenty je mocným nástrojem, pomocí nějž se z návrhu mockupu stává v podstatě stavebnice. Rozdělení uživatelského rozhraní je možné vidět na obrázku 3.3.

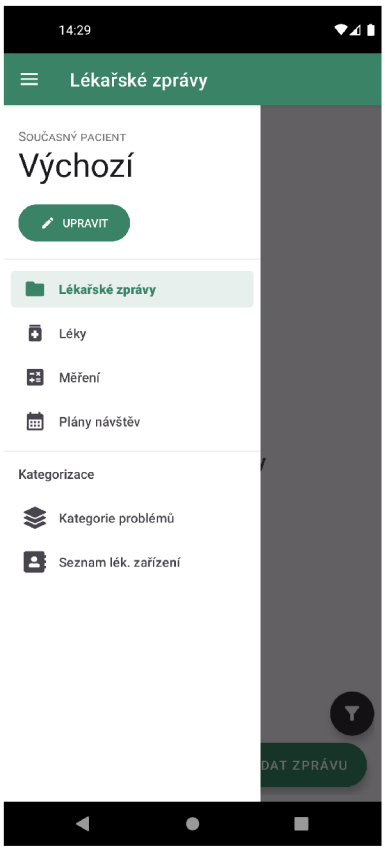


Obrázek 3.3: Ukázka využití komponent pro návrh UI

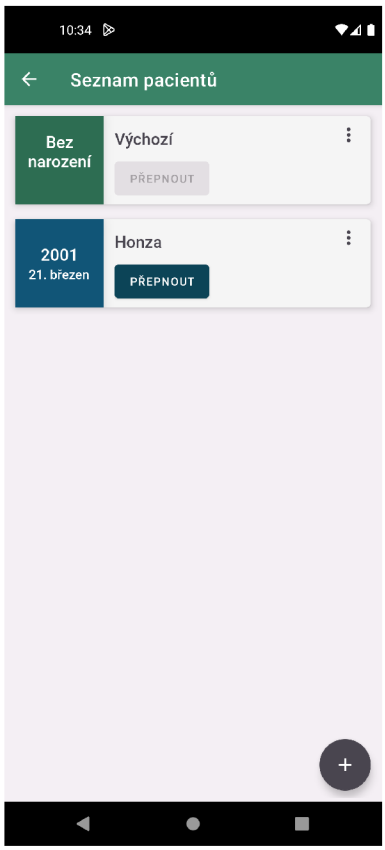
Pro ověření, zda mockup vhodně splňuje požadavky a zda je struktura aplikace správně navržená, byl z mockupu znovu vytvořen prototyp. Ten umožňuje interaktivní procházení návrhem a nabízí možnost, jak na testovacích uživatelích ověřit správnost řešení ještě před začátkem vývoje.

3.2 Popis fungování aplikace

Po zapnutí aplikace uživatele přivítá hlavní obrazovka prvního modulu. Pro průchod aplikací slouží postranní vysouvací menu, které se dělí na hlavičku, odkazy na hlavní části a odkazy na dodatečné položky (zde nazvané "Kategorizace"). Hlavička aplikace informuje o aktuálním vybraném pacientovi. Pacienti v této aplikaci fungují jako profily, všechna uživatelem zadaná data se vážou ke konkrétnímu pacientovi. Kliknutí na tlačítko "Upravit" vede uživatele na seznam všech pacientů (UC-17). Po instalaci aplikace existuje jen jeden pacient - "Výchozí". Pokud je vytvořen i jiný pacient (UC-18), lze na něj přejít tlačítkem "Přepnout" (UC-19).



Obrázek 3.4: Postranní menu aplikace



Obrázek 3.5: Výpis pacientů

3.2.1 Lékařské zprávy

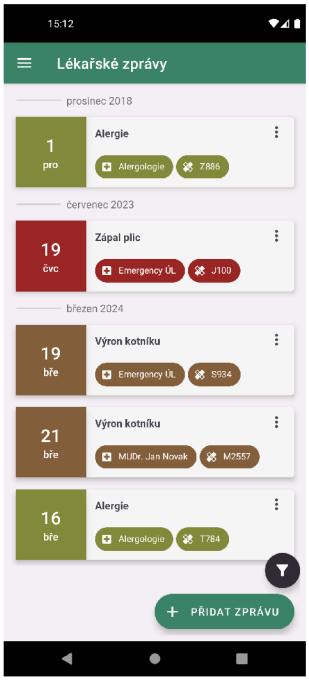
Hlavní obrazovkou modulu "Lékařské zprávy" je výpis lékařských zpráv (UC-02). Zprávy jsou ve výchozím stavu seskupeny podle data přidání. Po kliknutí na tlačítko s více možnostmi (tři tečky) můžeme konkrétní zprávu upravit, exportovat (UC-03) či smazat.

Kliknutím na hnědé tlačítko (pravý dolní roh) se zobrazí možnosti filtrace. To umožňuje nastavit seskupování (datum, kategorie problému, lékař) a omezit výpis zpráv dle kategorií a lékařů. Výběr je potřeba potvrdit tlačítkem "Filtrovat".

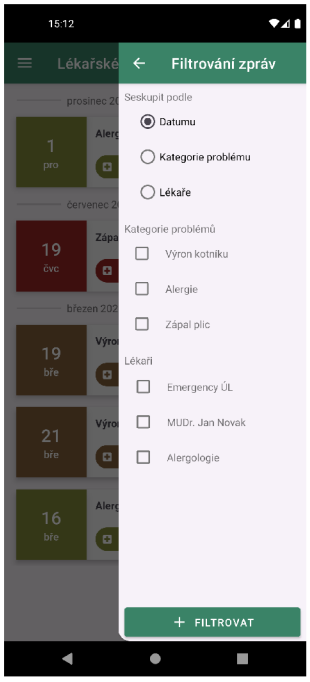
Na obrazovce pro novou lékařskou zprávu má uživatel možnost přidat (vyfotit/vybrat ze zařízení) přílohy (UC-01). Přílohy se analyzují a aplikace, pokud dokázala přečíst text, nabídne předvyplněné datum zprávy a diagnózu. Tato možnost je podmíněna připojením k internetu pro kontrolu kódu diagnózy.

K lékařské zprávě lze přiřadit i určitou kategorii problému a lékařského pracovníka. Tyto dvě informace jsou součástí sekce "Kategorizace", takže je potřeba nejdříve kategorii a lékařského pracovníka do aplikace přidat.

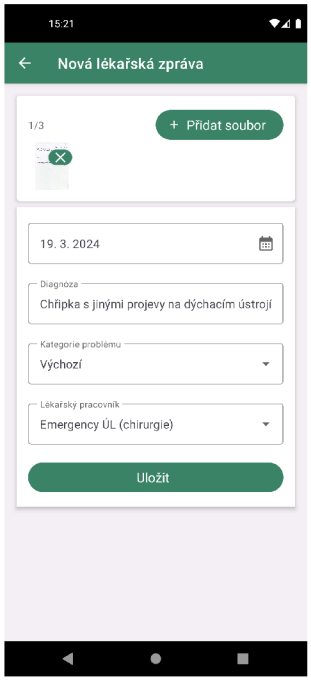
Po vytvoření či úpravě lékařské zprávy je uživatel přesměrován zpět na výpis zpráv.



Obrázek 3.6: Výpis zpráv



Obrázek 3.7: Filtrování



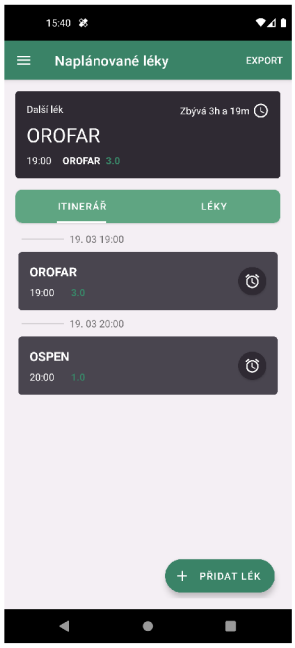
Obrázek 3.8: Nová zpráva

3.2.2 Léky

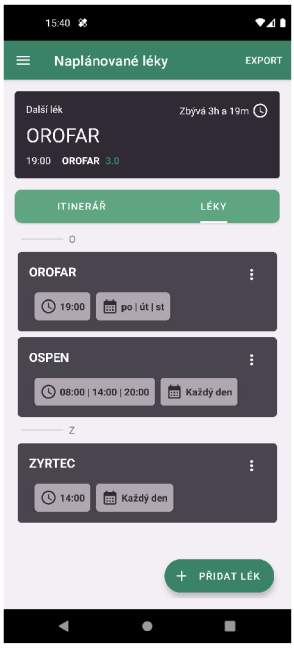
Hlavní obrazovkou modulu "Léky" je přehled léků pacienta. Obrazovka obsahuje export podávaných léků (UC-10), odpočet do dalšího podání, přehled dnešních podání (UC-06) a výpis všech léků (UC-05).

Léky v itineráři jsou seskupeny dle hodiny a zobrazují se jen ty, které se do konce dne ještě budou podávat. U každého léku je možnost vypnout či znovu zapnout upozornění (tlačítko s ikonou budíku) (UC-09). Léky v záložce "Léky" jsou seskupeny abecedně, obsahují informaci o časech a dnech podání. Kliknutím na lék se zobrazí detail léku se základními informacemi o něm (UC-07). V tomto detailu je možné si otevřít příbalový leták léku - jedná se o přesměrování na webové stránky SÚKLu, je tedy potřeba přístup k internetu.

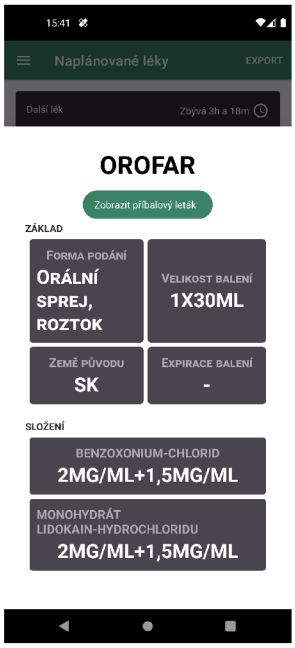
Na obrazovce pro přidání nového léku (UC-08) je potřeba vybrat podávaný lék, a to kliknutím na tlačítko “Žádný lék nevybrán”. V otevřeném dialogovém okně lze vyhledávat podle názvu léku, po vybrání léku a potvrzením tlačítkem “OK” se lék přidá do formuláře. Následuje vybrání počtu podání (v kolik hodin se podává jaké množství) a vybrání frekvence (v jaké dny upozorňovat). Po uložení přesměruje aplikace uživatele zpět na přehled léků.



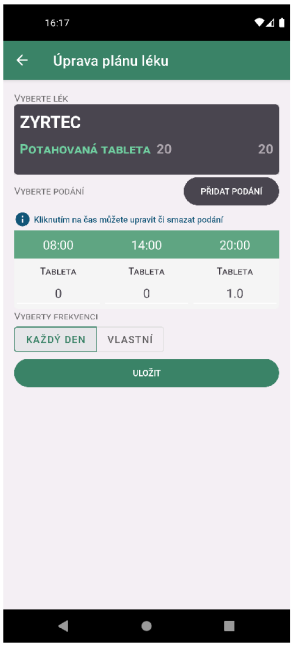
Obrázek 3.9: Itinerář léků



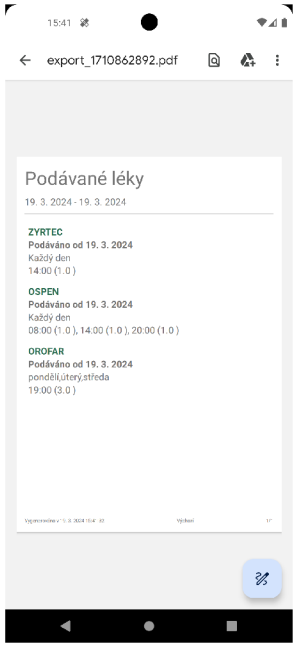
Obrázek 3.10: Seznam léků



Obrázek 3.11: Detail léku



Obrázek 3.12: Nový lék



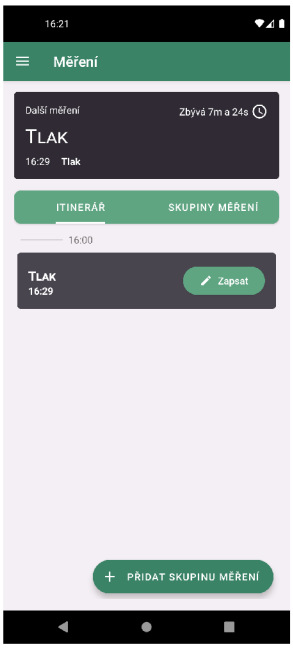
Obrázek 3.13: Export léků

3.2.3 Měření

Hlavní obrazovkou modulu “Měření” je přehled měření (UC-14), velmi podobný přehledu léků. Obsahuje odpočet do dalšího měření, itinerář a všechny skupiny měření.

Přidat nový záznam lze v záložce “Itinerář” po kliknutí na tlačítko “Zapsat”. Formulář obsahuje datum a čas zápisu a následně políčko pro zápis hodnot měření (UC-12).

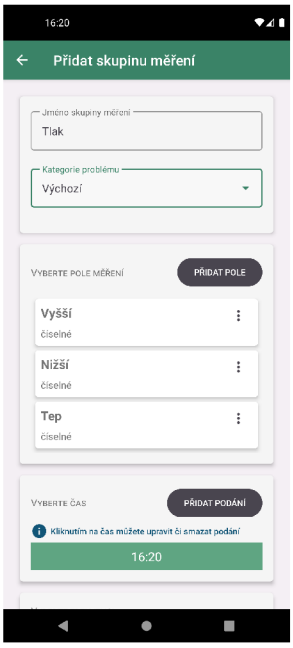
V záložce “Skupina měření” lze kliknutím na položku otevřít detail skupiny (UC-13). Ta obsahuje tabulku všech zápisů a graf vývoje hodnot pro každé políčko skupiny. Mimo to lze odtud přidat nový záznam či skupinu exportovat (UC-15).



Obrázek 3.14: Itinerář měření

Obrázek 3.15: Detail skupiny

Formulář pro přidání nové skupiny (UC-11) měření obsahuje jméno skupiny, kategorii problému, pole měření (co za hodnoty budeme měřit – skupina ”Tlak” může obsahovat políčka “Nižší”, “Vyšší” a “Tep”). Podobně jako u léku, i zde lze nastavit čas a dny upozorňování. Zápis hodnot však není limitován jen pro tyto zvolené časy. Hodnoty měření lze zapisovat kdykoliv, v nastavené časy jen přijde uživateli upozornění.

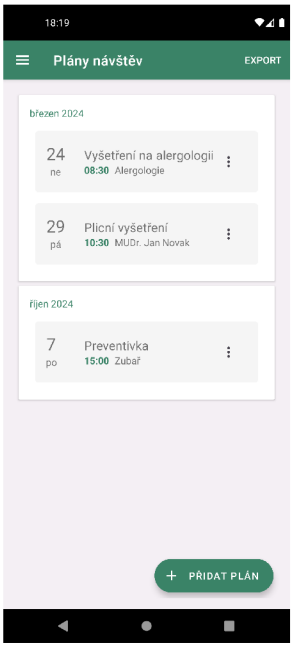


Obrázek 3.16: Nová skupina

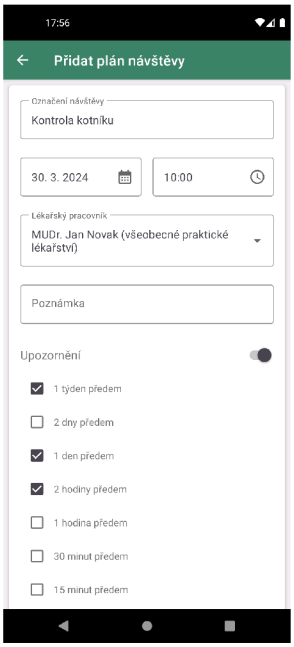
Obrázek 3.17: Nový zápis hodnot

3.2.4 Plány návštěv

Poslední hlavní částí jsou “Plány návštěv”, jejichž hlavní obrazovka obsahuje výpis všech plánovaných návštěv seskupených dle měsíců (UC-26). Původně byla tato část zamýšlena jako podmodul, při řešení však byly plány návštěv vypracovány jako samostatný modul. Formulář nových návštěv (UC-27) se skládá z označení návštěvy, data a času, lékařského pracovníka a případné poznámky. K dispozici je také možnost vybrat, jak dopředu má aplikace upozorňovat na návštěvu (týden, den, hodinu...).



Obrázek 3.18: Výpis plánů návštěv



Obrázek 3.19: Nový plán

4 Implementace

Tato kapitola se zabývá implementací návrhů z předchozí kapitoly a pokouší se je řešit vhodně zvoleným způsobem.

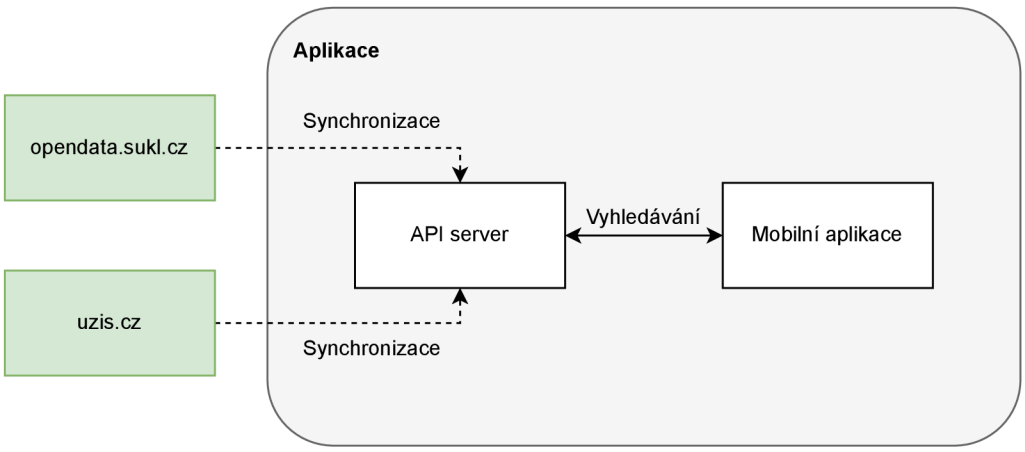
4.1 Celkové řešení

Jelikož si práce klade za cíl využívat v aplikaci data dostupná z veřejných zdrojů, je potřeba se nejprve zamyslet, jak aplikace nakládá se svými daty a odkud data z těchto veřejných zdrojů získává.

Všechny zdroje vydávají svá data ve formátu ZIP, periodičnost se liší - jeden měsíc, šest měsíců či rok. Je tedy potřeba uchovávat na jednom místě všechna aktuální data a starat se o synchronizování dat. K tomu v tomto případě slouží vlastní API server. Ten nabízí aplikaci přístup a možnost se dotazovat na tyto data, je ale důležité říct, že tím jeho role končí. Na server neputují žádná data z mobilní aplikace, z tohoto hlediska jsou uživatelská data v bezpečí. Na druhou stranu, nelze dotazování API serveru omezit jen pro uživatele aplikace.

Mobilní aplikace využívá přístup k API pro vyhledávání léků, diagnóz a lékařských zařízení. Pokud nemá aplikace přístup k internetu, nelze tyto funkce využívat, pokud však již záznamy v aplikaci jsou, není problém ji využívat offline.

Pro tuto práci existují dva GitHub repositáře - pro mobilní aplikace (16) a pro API server (17).

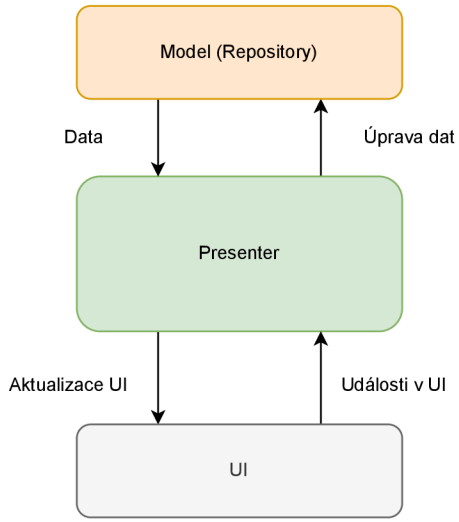


Obrázek 4.1: Celkové řešení této práce

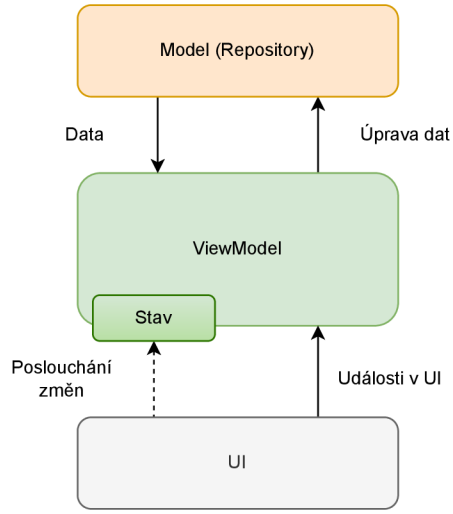
4.2 Architektura aplikace

Při vývoji aplikací pro Android se využívá návrhový vzor MVVM (Model–View–ViewModel). Cílem vzoru je separace logiky spojené s uživatelským rozhraním od datového modelu. Velice se podobá vzoru MVP, liší se však tím, jakým směrem proudí data při změně v modelu.

V návrhovém vzoru MVP (Model-View-Presenter) se při změně dat o aktualizaci uživatelského rozhraní stará Presenter, čím se však stává na View závislý. Ve vzoru MVVM je tomu naopak, View je závislé na řídící jednotce ViewModel. Ta si uchovává „stav obrazovky“, který dle potřeb aktualizuje a který je k dispozici uživatelskému rozhraní. View pak reaguje na změny ve stavu a aktualizuje uživatelské rozhraní sám. (18)



Obrázek 4.2: Model-View-Presenter



Obrázek 4.3: Model-View-ViewModel

V jazyce Kotlin je stav obrazovky realizován datovou třídou (data class), která se ve ViewModelu „uchovává“ ve StateFlow. Ten patří mezi Flow, prostředek poskytující tok hodnot (stream) vysílaných za běhu aplikace. Nové hodnoty v toku dat je možné „poslouchat“ a reagovat na ně. StateFlow je pak specifický tím, že si vždy uchovává poslední vysílanou hodnotu.

V Androidu je ViewModel důležitý tím, že dokáže uchovávat svá data i během konfiguračních změn (rotace obrazovky, přepnutí na noční režim a podobně). U View komponent dojde po konfigurační změně k znovu vykreslení, při kterém dochází ke ztrátě dat těchto komponent. V aplikaci to může vypadat třeba tak, že po otočení zařízení zmizí z vyplněných formulářů data. ViewModely si data uchovávají a jsou tudíž využívány jako „držitelé“ stavu obrazovek. Z tohoto důvodu je tedy View závislý na ViewModelu, využívá data, které ViewModel uchovává a při událostech v UI (kliknutí, gesto apod.) volá metody, které poskytuje ViewModel.

Lze si všimnout, že zde přirozeně dochází k dělení na vrstvy – v tomto případě na UI (View) a prezentační (ViewModel). Pokud by bylo cílem vytvořit pouze malou

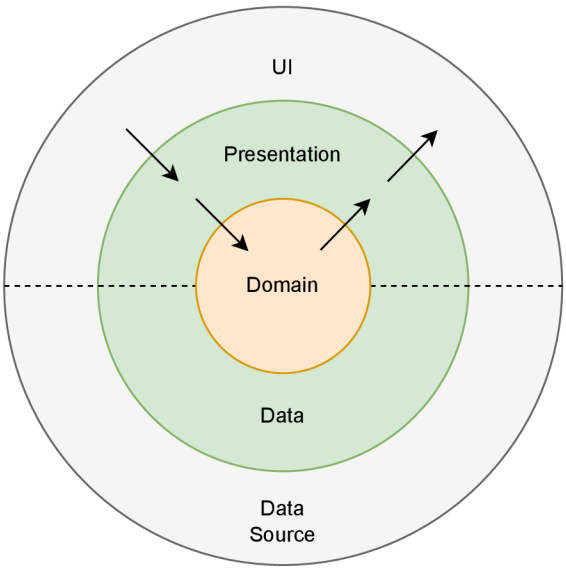
aplikaci s velmi omezeným rozsahem, toto rozdělení by stačilo. V rámci ViewModelů by se volaly repositáře provádějící databázové operace, zpracovávala by se data a uchovávaly stavy. Bohužel při růstu požadavků na aplikaci by tímto způsobem došlo k situaci, kdy by ViewModely byly příliš velké, obsahovaly by duplicitní části a obecně by se hůře udržovaly.

4.3 Clean Architecture

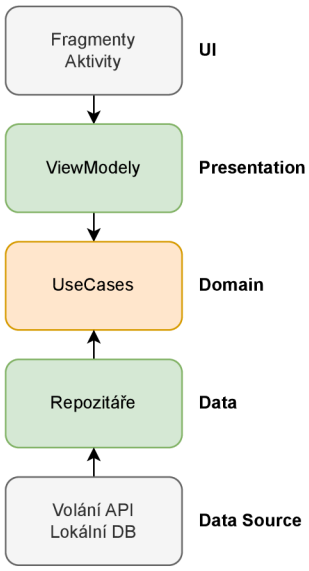
Z těchto důvodů byla pro práci využita vrstevnatá systémová architektura „Clean Architecture“. Pro potřeby práce bylo čerpáno z knihy „Clean Architecture for Android“ (Boundjnah, Eran). (19) Výhodou této architektury je větší škálovatelnost a udržitelnost, ovšem za cenu většího úsilí při vývoji.

Architektura dělí aplikaci na pět vrstev: vrstvu datových zdrojů (data source), datovou vrstvu (data), doménovou vrstvu (domain), prezentační vrstvu (presentation) a vrstvu uživatelského rozhraní (user interface).

Autor knihy tuto architekturu pojímá velice striktně – každá vrstva by měla mít přístup jen k vrstvě nižší (platí pro user interface a presentation) či vyšší (platí pro data a data source) a pracovat s datovými modely dané vrstvy, viz obrázek 4.5. To znamená, že každá vrstva má svojí verzi datových modelů, které se mohou lišit a při jejichž změně není nutné zasahovat do ostatních vrstev. Mezi vrstvami se modely mapují pomocí „mapperů“, které byly pro daný model a dané dvě vrstvy vytvořeny. Tento krok sice přináší spoustu „boilerplate“ kódu, dává ale vývojářům manévrovací prostor při změně požadavků na dané vrstvě. Zároveň mapování modelů zajišťuje, že daná vrstva nemá závislosti na implementaci ve vrstvě jiné (toto platí třeba pro entity ve vrstvě datových zdrojů).



Obrázek 4.4: Vrstvy architektury



Obrázek 4.5: Svislý řez vrstev

4.3.1 Doménová vrstva

Jak je zřejmé z obrázku 4.4, doménová vrstva je jádrem aplikace a není na ostatních vrstvách vůbec závislá. Vrstva slouží k vykonávání tzv. „bussiness logic“, důvodu, proč aplikace existuje – přímé požadavky klienta na funkce aplikace (jak často upozorňovat na léky, podle čeho řadit a seskupovat data, výběr léků do přehledu, kdy notifikovat uživatele…). Tato logika je definována v tzv. Use-casech. Use-case funguje jako izolovaná jednotka, která své závislosti přijímá prostřednictvím konstruktoru.

Mezi závislosti patří hlavně repositáře doménové vrstvy. Ty jsou tvořeny rozhraním (interface) s metodami, se kterými počítá Use-case a je mu jedno kdo a kde je implementuje. Tato abstrakce významně ulehčuje testování logiky skrze unit testy. Lze říct, že tuto vrstvu nezajímá, jestli tvoříme mobilní aplikaci, nástroj pro příkazovou řádku nebo REST API. Je proto důležité ji „odstřihnout“ od jakýkoliv specifikací zařízení.

Use-casy v této aplikaci implementují jednotné rozhraní UseCase, respektive abstraktní třídu BackgroundExecutingUseCase, a jsou vykonávány pomocí třídy UseCaseExecutor, která daný UseCase spustí asynchronně – v Kotlinu jsou pro tyto potřeby využívány tzv. Coroutiny.

Níže uvedený Use-case (1) má za úkol zjistit jaké léky (a kdy) se podávají v časovém intervalu. Přijímá datovou třídu SchedulesInRangeRequest, která obsahuje ID pacienta a časový interval, a vrací seznam všech plánovaných podání léků v časovém intervalu.

Přes konstruktor UseCase přijímá repositář doménové vrstvy GetSchedulesByPatientRepository. Jedná se o rozhraní, které je implementováno v datové vrstvě a do Use-case se dostává pomocí předávání závislostí (dependency injection). Samotná logika se vykonává v metodě executeInBackground.

```
class GetScheduledInTimeRangeUseCase(
    private val getSchedulesByPatientRepository: GetSchedulesByPatientRepository,
    coroutineContextProvider: CoroutineContextProvider
) : BackgroundExecutingUseCase<
        SchedulesInRangeRequest,
        List<ScheduleItemWithDetailsDomainModel>
    >(
    coroutineContextProvider
) {

    override suspend fun executeInBackground(
        request: SchedulesInRangeRequest
    ): List<ScheduleItemWithDetailsDomainModel> {
        // Logika pro vykonávání zde
    }
}
```

Kód 1: Struktura Use-case třídy

4.3.2 Prezentační vrstva

Prezentační vrstva má za úkol říct uživatelskému rozhraní co prezentovat uživateli, ne jak. Podobně jako u doménové vrstvy, prezentační vrstva nemá žádné závislosti na implementaci uživatelského rozhraní a je proto nutné některé akce abstrahovat.

Hlavním bodem této vrstvy je ViewModel, pro každou obrazovku vlastní. Ten obsahuje view state neboli datovou třídu s daty definující aktuální stav obrazovky. Typicky to jsou seznamy modelů pro výpis, aktuální pacient a podobně. Jak již bylo zmíněno výše, view state je „uchováván“ prostřednictvím prostředku StateFlow, který umožňuje UI vrstvě poslouchat na změny.

Dalším hlavním důvodem existence ViewModelu jsou metody volané uživatelským rozhraním po nějaké interakci, např. onFormSubmit, onEdit, onSort. Jelikož uživatelské rozhraní může být naprosto cokoliv, je důležité pojmenovávat metody podle toho na co reagují a ne podle akcí v UI (např. místo onSearchClick použít onSearch).

U každé obrazovky je potřeba definovat možné destinace, na které může ViewModel navigovat. Jsou pro to využívány sealed classes, speciální typ tříd, který omezuje hierarchii dědičnosti a podobně jako enumerátor se využívá pro definování možných hodnot. Navigování ve ViewModelu poté funguje zavoláním metody navigate s instancí destinace.

```
sealed class ListMedicineDestination : PresentationDestination {
    data class EditSchedule(val id: String) : PresentationDestination
    object CreateSchedule : PresentationDestination
}

// .. ve ViewModelu
fun onCreate(){
    navigate(ListMedicineDestination.CreateSchedule)
}
```

Kód 2: Třída s destinacemi v prezentační vrstvě

V poslední řadě je potřeba „notifikovat“ UI vrstvu, že došlo k nějaké události (formulář se úspěšně uložil, obsahuje chyby…). Pro tyto účely je vytvořena sealed class s možnými notifikacemi pro každou obrazovku.

ViewModely v této aplikaci dědí od vytvořeného předka BaseViewModel. Předek definuje toky hodnot pro view state, notifikace a destinaci. Pro přístup k aktuální hodnotě view state definuje currentViewState, pro aktualizaci view state pak metodu updateViewState. Pro vyslání nové notifikace definuje metodu notify, pro navigování na jinou obrazovku metodu navigateTo. Jako poslední definuje abstraktní metodu initState, která vytváří a vrací počátní view state.

4.3.3 UI vrstva

Pro tuto práci byl zvolen způsob vytváření uživatelského rozhraní pomocí XML šablon. Základním stavebním kamenem uživatelského rozhraní v Androidu je třída Activity, která drží všechny prvky uživatelského rozhraní. Historicky se pro každou obrazovku aplikace vytvářela vlastní aktivita, v současné době se však častěji setkáme s tím, že aplikace má jednu (nebo malý počet) aktivit a konkrétní obrazovky jsou tvořeny pomocí Fragmentů.

Fragmenty v Androidu reprezentují část či celé uživatelské rozhraní obrazovky. Při vytvoření Fragmentu je potřeba „nafouknout“ (inflate) XML šablonu a dosadit do ní data aplikace (binding).

Fragmenty v naší aplikaci potřebují ke správné funkčnosti

- ViewModel, u kterého poslouchá změny ve view state, notifikace a navigování,
- DestinationUiMapper, který mapuje cíle z prezentační vrstvy na volání navigační komponenty Androidu a
- Binder, který při změně ve view state „binduje“ aktuální data na layout.

Při inicializaci fragmentu se také registrují listenery na interakce s UI prvky (tlačítka, textové pole,...), které při události volají metody ViewModelu.

V aplikaci existuje mnou definovaný předek BaseFragment. Ten definuje abstraktní atributy, která musí potomek implementovat; view model, destination mapper a view state binder. Předek nabízí také řadu metod pro zjednodušení práce s uživatelským rozhraním, např. showSnackbar, showConfirmDialog a metody pro definování menu pro horní lištu aplikace.

Pro přesměrování uživatele na jiné obrazovky existují v aplikaci třídy DestinationUiMapper, které mapují přesměrování z prezentační vrstvy na konkrétní operace s navigační komponentou prostřednictvím třídy NavManager.

Poslední třída pro uživatelské rozhraní je ViewStateBinder, který se stará o propisování dat do šablony obrazovky. Tyto Bindery dědí od předka BaseViewStateBinder, který definuje metody

- init – zavedení binderu před prvním vykreslením,
- firstBind – volá se jen při prvním vykreslení a
- bind – volá se při každé změně ve view state.

4.3.4 Datová vrstva

Úkolem datové vrstvy je implementovat rozhraní repositářů doménové vrstvy a tím odstínit doménovou vrstvu od konkrétních implementací. V těchto konkrétních implementacích se využívají prostředky z vrstvy datových zdrojů, v této aplikaci zejména Data Access Objekty (DAO) knihovny Room. Ta slouží pro objektově relační mapování lokální SQLite databáze.

Jak je možná zřejmé, datová vrstva a vrstva datových zdrojů mají hodně společného, a často je lze vidět seskupeny do jedné vrstvy – to platí i pro tuto aplikaci. Autor zmiňované knihy pohlíží na datovou vrstvu jako nezávislou na platformě a realizuje to tím, že v datové vrstvě vytváří další repositáře (interface) pro práci s daty, které pak implementuje vrstva datových zdrojů.

Pro účely této aplikace bylo od tohoto náhledu upuštěno, jelikož by to přineslo další komplexnost, která se mi nezdá příliš opodstatněna. Místo toho je v aplikaci vrstva datových zdrojů brána jako podvrstva datové vrstvy.

V doménové vrstvě máme definovány rozhraní DeletePatientRepository, GetPatientByIdRepsitory a GetSelectedPatientRepository, se kterými pracují některé UseCasy.

```
interface DeletePatientRepository {
    suspend fun deletePatient(id: String): Boolean
}

interface GetPatientByIdRepository {
    suspend fun getById(id: String): PatientDomainModel?
}

interface GetSelectedPatientRepository {
    fun getSelectedPatient(): Flow<PatientDomainModel>
}
```

Kód 3: Rozhraní repositářů doménové vrstvy

V datové vrstvě vznikla třída PatientRepository, která všechna tato rozhraní implementuje. Třída přijímá v konstruktoru DAO pro práci s databází, DataStore pro uchovávání ID aktuálně vybraného pacienta a mapper pro mapování modelu pacienta mezi vrstvami.

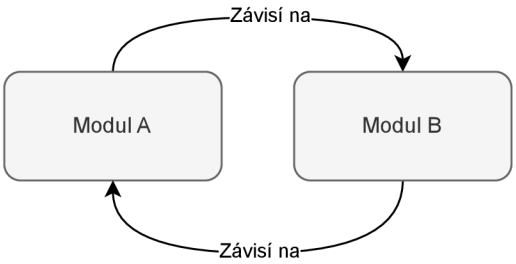
4.3.5 Rozdělení modulů

Jelikož z funkčních požadavků vyplynulo rozdělí na moduly, bylo rozhodnuto toto rozdělení zachovat i při implementaci v aplikaci.

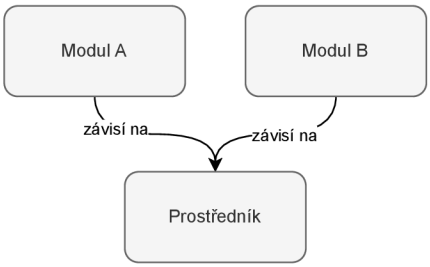
Zde přichází na řadu Gradle, nástroj pro automatizaci sestavení projektů (build management systém) postavených na JVM (Java, Kotlin, Scala…). Pomocí Gradle můžeme definovat závislosti na knihovny, konfigurovat specifika sestavení (například nastavení kompilátoru a verze JVM). Zároveň také umožňuje rozdělení projektu do více modulů, které mají vlastní závislosti a sestavují se samostatně. Takovéto rozvržení Gradle nazývá jako „Multi-Project build“, moduly označuje jako subprojekty. (20)

Moduly mohou mít závislosti na jiných modulech, je však důležité si pamatovat, že nelze vytvořit závislost na modulu, který již na prvním modulu závisí. Tím by vznikla kruhová závislost a Gradle by nebyl schopný projekt sestavit (obrázek 4.6).

Možným řešením je vytvoření modulu, který slouží jako prostředník pro prostředky, které kruhovou závislost způsobují (obrázek 4.7).

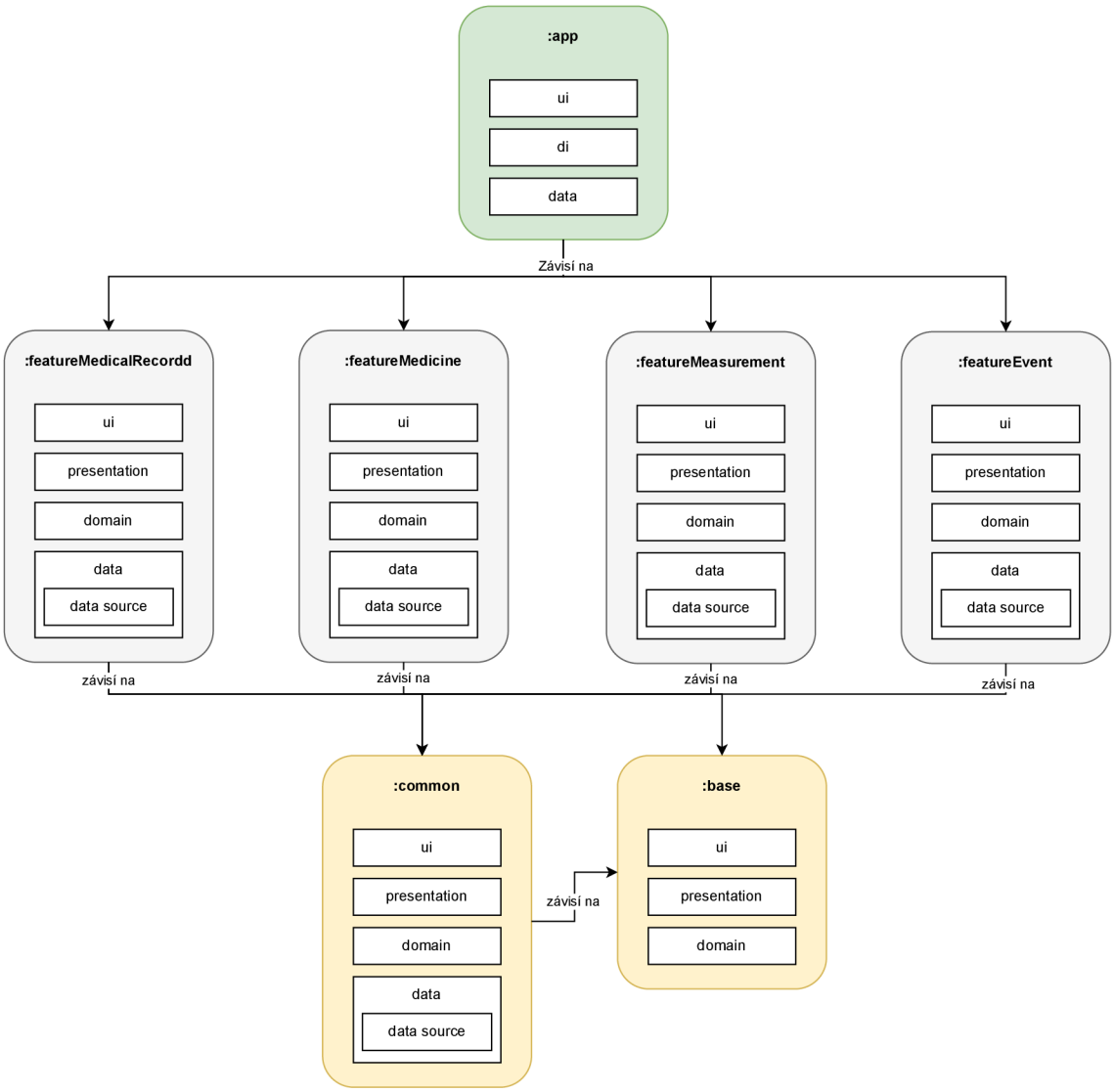


Obrázek 4.6: Kruhová závislost



Obrázek 4.7: Řešení kruhové závislosti

V této aplikaci se k modulům přistupuje následovně. Moduly, které se týkají specifické funkce, se označují prefixem „feature“. V aplikaci tedy jsou moduly „featureMedicalRecord“ pro lékařské zprávy, „featureMedicine“ pro léky, „featureMeasurement“ pro měření hodnot a „featureEvent“ pro plány návštěv. Všechny tyto moduly potřebují vázat svá data na konkrétního pacienta a potřebují vědět, jaký pacient je aktuálně vybraný. Nedávalo by smysl, aby prostředky pro práci s pacienty byly v každém tomto modulu, tudíž tyto prostředky se přidaly do dalšího modulu „common“. Ten je s modulem „base“, který obsahuje předky a prostředky pro fungování Clean Architecture, v závislostech každého ”feature” modulu. Výsledné moduly a jejich závislosti jsou zřetelné na diagramu 4.8.



Obrázek 4.8: Diagram modulů a vzájemných závislostí

Podobně jako pacient, i kategorie problémů je prostředek využívaný v dalších ”feature” modulech. Součástí funkčních požadavků je i detail kategorie, tedy výpis všech lékařských zpráv, léků a měření, které jsou ke kategorii uloženy. To přináší další problém, jelikož je potřeba z „common“ modulu přistupovat k prostředkům (repositáře, datové modely…), které jsou součástí ”feature” modulů, čím by se však vytvořila kruhovou závislost.

Tento problém jsem v aplikaci vyřešil využitím návrhového vzoru „Observer“. V aplikaci byla vytvořena třída EventBusChannel, která udržuje seznam všech registrovaných „posluchačů“, které upozorní na nově příchozí událost. V „common“ modulu existuje objekt CommonEventBus se všemi kanály, které jsou v modulu využívány. Pokud chce ”feature” modul poslouchat a reagovat na událost, musí si vytvořit potomka třídy ModuleListener. V metodě onInit se poté registruje ke kanálu, které jsou potřeba.

V této aplikaci je tento prostředek využíván například pro získání detailů o záznamech, které se vážou ke kategorii problému.

4.4 Vybrané technologie

4.4.1 Programovací jazyk

Aplikace pro Android se historicky vyvíjely v Javě, v roce 2017 však Google oznámil „first-class“ podporu jazyka Kotlin pro Android a od roku 2019 je Kotlin preferovaným jazykem pro vývoj aplikací pro Android. (21)

Po předchozích zkušenostech v Javě, které však nezahrnovaly vývoj aplikací, jsem pro práci vybral jazyk Kotlin. Přechod z Javy na Kotlin byl velmi jednoduchý a velmi „pohodlný“. Výhodou Kotlinu je, že se stejně jako Java překládá do Java Bytecode, lze tedy využít existující knihovny původně vytvořené pro Javu.

4.4.2 Vývojové prostředí

Pro vývoj aplikace jsem využil vývojové prostředí (IDE) Android Studio přímo od společnosti Google. Android Studio nabízí širokou škálu nástrojů pro návrh a vývoj aplikací – integrovaný emulátor, debugger, analyzátor výkonu (Profiler), nadstavbu pro správu závislostí (Gradle), preview při vytváření XML šablon a mnoho dalších.

4.4.3 Lokální databáze

Android aplikace disponují možností vytvořit si SQLite databázi pro své potřeby. Je možné s databází pracovat na bázi přímých dotazů, mnohem jednodušší je však využít knihovnu Room pro objektové mapování relační databáze. Zároveň také nabízí ověření správnosti SQL dotazů při kompilaci.

Základem je definování entit pomocí datových tříd a anotací. Z té se pak při kompilaci vytvoří databázová entita. V základu lze v entitě uvést jen triviální datové typy, existuje ale možnost vytvořit a použít konvertory, které složitější data převádějí na textový řetězec.

```
@Entity(tableName = "medicine_schedule")
data class MedicineScheduleDataSourceModel(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    @ColumnInfo(name = "patient_id") val patientId: Int,
    @ColumnInfo(name = "medicine_id") val medicineId: String,
    val createdAt: LocalDateTime,
    @ColumnInfo(name = "is_alarm_enabled") val isAlarmEnabled: Boolean = true,
)
```

Kód 4: Definice databázové entity

Pro manipulaci s daty slouží tzv. Data Acces Objects (DAO). Jedná se o rozhraní, kde metody obsahují anotaci s SQL dotazem. Lze tedy dosáhnout i komplexnějších dotazů.

Takto definované modely jsou skvělým příkladem, proč se v aplikaci mapují modely mezi vrstvami. Jelikož modely obsahují anotace specifické pro knihovnu Room, nesla by se tato závislost i do vyšších vrstev aplikace.

4.4.4 Komunikace s API serverem

Vyvíjená aplikace pro některé své funkce požaduje přístup k vzdálenému API serveru. Pro tyto potřeby v aplikaci používám knihovnu Retrofit. Ta umožňuje definování „endpointů“ pomocí rozhraní a mapování výsledku HTTP požadavku na objekty. Jedná se o druhý typ datového zdroje, který v aplikaci je.

4.4.5 Generování PDF

V každém modulu aplikace je zapotřebí exportovat uživatelská data ve formátu PDF. Pro tyto účely se využívá třída PdfDocument, která je součástí Androidu. (22) Ta umožňuje detailní práci s PDF soubory, práce je to však nízkoúrovňová. Je tedy využíváno vlastní řešení, které umožňuje definici stránek pomocí potomků mnou vytvořené třídy DocumentPage a pracovat se šablonou dokumentu takřka stejně jako v případě Fragmentu. PDF dokument je posléze poskládán z jedné či více stránek.

```
class MedicineExportPage(
    private val context: Context,
    private val schedules: List<MedicineScheduleUiModel>
) : DocumentPage() {

    override fun bind(binding: ViewBinding) {
        // Bind data to the view of the page
    }

    override fun getViewBinding(
        inflater: LayoutInflater
    ): DocumentPageMedicineBinding {
        return DocumentPageMedicineBinding.inflate(inflater)
    }
}
```

Kód 5: Třída reprezentující stránku PDF

Pro uložení dokumentu je potřeba využít kontrakt. Kontrakt je způsob, jak zjednodušit interakce mezi různými komponentami, službami či zcela jinými aplikacemi. (23) Mnou využívaný kontrakt ActivityResultContracts.CreateDocument umožňuje

říct, že si aplikace přeje uložit soubor. Operační systém kontrakt přijme, otevře aplikaci průzkumníka souborů a po uložení původní aplikaci vrací cestu k uloženému souboru. Je důležité pamatovat, že kontrakty je nutné registrovat před tím, než je Fragment vytvořen (před voláním metody onCreate).

Kvůli těmto úkonům jsem vytvořil předka BaseExportFragment, který všechny tyto úkony dělá sám a potomkům dává k dispozici metodu launchExport, která přijímá seznam objektů DocumentPage a generuje PDF soubor.

4.4.6 OCR

Při brainstormingu funkcí v rámci přidávání lékařské zprávy vzešel nápad automaticky přečíst ze zprávy základní informace jako je diagnóza a datum návštěvy, a uživateli je předvyplnit. Jelikož však neexistuje jednotný formát lékařských zpráv, bylo potřeba se vydat cestou automatické detekce textu a jeho následného zpracovávání.

Pro rozpoznávání textu na fotografii je v aplikaci využíván Google ML Kit, který umožňuje snadněji integrovat funkce jako skenování čárového kódu, detekci obličejů, detekci objektů nebo právě rozpoznávání textu. (24)

ML Kit po rozpoznání textu vrací objekt skládající se ze skupiny textových bloků, ze kterých je poskládán nalezený text. Na řadu přichází mnou vytvořené rozpoznávání, kdy pro každou požadovanou informaci existuje vlastní rozpoznávač (FieldProcessor), třída, která má za úkol z obyčejného textového řetězce informaci najít. Je nutné podotknout, že úspěšnost nalezení informací vychází ze správného detekování textu. V budoucnu lze uvažovat o implementování operací, které by fotografii před detekcí upravily a zlepšily tím úspěšnost.

Diagnóza

V lékařských zprávách ve většině případů nalezneme i kódy usuzované diagnózy. Tyto kódy se odkazují na Mezinárodní klasifikaci nemocí a diagnóz (MKN-10) a mají přesně stanovený formát. Pomocí regulárního výrazu je lze v textu vyhledávat. (25)

```
"[A-TV-Z][0-9][0-9AB]\\.?[0-9A-TV-Z]{0,4}"
```

Kód 6: Regulární výraz pro získání diagnózy

Datum návštěvy

Rozpoznávač pro datum návštěvy v textu vyhledává podle definovaných formátů zápisu data.

```
mapOf(
    "dd.MM.yyyy" to "(\\d{2})[.|-]\\s*?(\\d{2})[.|-]\\s*?(\\d{4})",
    "d.MM.yyyy" to "(\\d{1})[.|-]\\s*?(\\d{2})[.|-]\\s*?(\\d{4})",
    "dd.M.yyyy" to "(\\d{2})[.|-]\\s*?(\\d{1})[.|-]\\s*?(\\d{4})",
    "d.M.yyyy" to "(\\d{1})[.|-]\\s*?(\\d{1})[.|-]\\s*?(\\d{4})",
    "yyyy-MM-dd" to "(\\d{4})[.|-]\\s*?(\\d{2})[.|-]\\s*?(\\d{2})",
    "yyyy-M-d" to "(\\d{4})[.|-]\\s*?(\\d{1})[.|-]\\s*?(\\d{1})",
    "yyyy-MM-d" to "(\\d{4})[.|-]\\s*?(\\d{2})[.|-]\\s*?(\\d{1})",
    "yyyy-M-dd" to "(\\d{4})[.|-]\\s*?(\\d{1})[.|-]\\s*?(\\d{2})"
)
```

Kód 7: Vyhledávané formáty dat

Pacient

Ve dříve nalezených datumech se rozpoznávač (PatientFieldProcessor) snaží najít datum, které by se shodovalo s datem narození některého z pacientů. V průběhu vývoje byl tento rozpoznávač zamýšlen pro to, aby uživateli nabídl případné přepnutí pacienta na toho správného, pokud by zjistil rozdílné datum narození. Ve finální verzi tento processor sice existuje, ale jeho výsledek není využíván.

4.4.7 Předávání závislostí

V aplikaci se využívá technika zvaná „předávání závislostí“ (v angličtině Dependency Injection – DI). Ta spočívá v tom, že třída přijímá své závislosti (UseCase, repozitář, mapper, …) skrze parametry a není odpovědná za inicializaci těchto závislostí, což vede k větší modularitě kódu.

Pro implementaci DI je v aplikaci využívána knihovna Hilt. Pro fungování je potřeba vytvořit moduly (jedná se o třídy, neplést s moduly aplikace), které definují, jak se inicializují jednotlivé závislosti. Jelikož se jedná o knihovnu specifickou pro Android, je přímo využívaná jen v UI a prezentační vrstvě. Pro předání závislosti je potřeba využít anotace, ve Fragmentech je potřeba označit třídu s @AndroidEntryPoint, ve ViewModelu je potřeba @HiltViewModel.

4.5 Zabezpečení

Jak již bylo uvedeno, aplikace obsahuje citlivá data uživatelů a je tudíž potřeba dbát na její bezpečnost. Důležitým faktem je, že aplikace nepřenáší žádná data uživatelů na server a striktně zůstávají v jejich zařízeních. Aplikace sice využívá vzdálený API

server, ale jen pro dotazování dat z číselníků léčivých přípravků, diagnóz a zdravotnických zařízení. Na server nejsou odesílána žádná data sloužící k identifikaci uživatele. Tento přístup sebou nese i úskalí, konkrétně nejsou data aplikace nijak zálohovaná. V současné podobě neexistuje možnost exportovat data ve strojově čitelné podobě, lze o tom uvažovat při možném pokračování vývoje. Při odinstalaci aplikace, či při poškození zařízení, jsou data nenávratně ztracena.

V aplikaci se uchovávají jak obrazové záznamy lékařských zpráv, tak i data aplikace ve strojově čitelném formátu (databáze aplikace). Obrazové záznamy jsou uchovávány v interním uložišti aplikace, tedy ve složce, ke které operační systém uděluje přístup pouze aplikaci, jíž složka patří. (26) Od verze Android 7 (API level 24) je podporován tzv. File-Based Encryption (FBE). Od verze Android 10 (API level 29) je tato funkce povinná ve všech zařízeních.

Co se týče bezpečnosti databáze, lze říct, že se za normálních okolností k datům nelze dostat mimo aplikaci. To ovšem neplatí, pokud je zařízení tzv. rooted (uživatelský účet má udělen nejvyšší oprávnění – superuser). V tomto případě se lze dostat k datům všech aplikací, i když by za normálních okolností nebyla přístupná.

Pokud bychom chtěli data této aplikace zabezpečit i na této úrovni, bylo by nutné databázi enkryptovat. K tomu by šlo využít například projekt SQLCipher, který lze integrovat s knihovnou pro ORM Room. (27) Pro potřeby této práce nebyl tento krok zvolen, jedná se o možné rozšíření bezpečnosti do budoucna.

4.6 Propojení na veřejné datové zdroje

Jak již bylo uvedeno dříve, jedním z cílů této práce je prozkoumat veřejně dostupné zdroje dat ze zdravotnického prostředí.

4.6.1 Databáze léčivých přípravků

Databáze léčivých přípravků (DLP) je poskytována v rámci otevřených dat Státním ústavem pro kontrolu léčiv (SÚKL). Jedná se o archiv formátu ZIP, který obsahuje řadu číselníků a seznamů ve formátu CSV. Periodicita aktualizací dat je 1 měsíc. (28)

Hlavním souborem, ze kterého je čerpáno, je seznam všech léčivých přípravků (dlp_lecivepripravky.csv), který sčítá přes 64 tisíc položek. U každého přípravku jsou dodatečná data, která často odkazují na číselníky z jiných souborů.

V mobilní aplikaci jsou data z tohoto zdroje využívána k zapisování a připomínání léčivých přípravků. Mezi využívané informace patří název, složení, balení, forma podání a země původu. Původně bylo zamýšleno využít i informaci o velikosti balení přípravku pro výpočet zbývajících podání, při analýze dat však vyšla najevo skutečnost, že nejsem schopný data správně a jednoznačně interpretovat, viz tabulku 4.1. Necelá polovina přípravků má jako balení zapsán jen počet dávek. U pětiny je počet dávek doplněn o informaci velikosti jedné dávky. U přípravků, které jsou jednorázové, může být i jen označení velikosti dávky. Zbylých 29,2 % přípravků má u balení informaci, kterou nejsem schopný jednoznačně určit. Příklady pro jednotlivé

formáty lze vidět v tabulce 4.2. Z časových důvodů bylo od využití informací o balení upuštěno, při možném budoucím vývoji se lze k problému vrátit po konzultacích se specialisty.

Formát	Četnost výskytu	Dokážu zpracovat?
Počet dávek	48,2 %	ANO
Počet dávek a velikost dávky	20,3 %	ANO
Velikost dávky	02,3 %	ANO
Čísla s římskými čísly	14,8 %	NE
Ostatní	14,42 %	NE

Tabulka 4.1: Typy zápisů balení přípravků

Formát	Příklad
Počet dávek	5
Počet dávek a velikost dávky	1X20ML
Velikost dávky	15G
Čísla s římskými čísly	5 II
Ostatní	7X(1+1X1ML ISP+AD+J)

Tabulka 4.2: Příklady zápisů

4.6.2 Mezinárodní klasifikace nemocí

Mezinárodní klasifikace nemocí (MKN-10, v angličtině ICD-10) je systém označování a klasifikace lidských poruch, zdravotních problémů a dalších příznaků. (29) Pacient se s ní může často setkat v lékařských zprávách, kde se využívá např. pro kódové označení stanovené diagnózy.

Klasifikaci lze získat z webu Ústavu zdravotnických informací a statistiky ČR (ÚZIS) buď jako tabelární část (publikace MKN ve formátu PDF) nebo v ZIP archivů. Archiv obsahuje popis číselníků ve formátu DOCX, tabelární část ve formátu XLSX, rozdílový soubor a složky s daty, které se liší kódováním souborů (cp_utf8 a cp_w1250). V těch lze najít CSV soubor obsahující všechny záznamy pohromadě (01_MKN10_*_*_utf8.csv). Periodicita aktualizací dat je 1 rok. (30)

V aplikaci je klasifikace využívána pro označení lékařských zpráv konkrétní diagnózou.

4.6.3 Národní registr poskytovatelů zdravotních služeb

ÚZIS také nabízí veřejně přístupná data v rámci Národního registru poskytovatelů zdravotních služeb (NRPZS). (31) Díky tomu lze pracovat se soupisem všech poskytovatelů, jejich adres, specializací, kontaktních údajů a dalších informací. V aplikaci lze k poskytovatelům získaných z toho registru uvádět i uživatelem zadané informace o pracovnících. Jedná se o jeden CSV soubor o velikosti zhruba 39,5 MB, který obsahuje přes 63 tisíc záznamů. Periodicita aktualizací dat je 1 měsíc.

4.6.4 Další zdroje

DASTA

V České republice se pro předávání dat mezi systémy využívá standard DASTA, který se mimo Slovensko jinde v zahraničí nevyužívá. (32) Součástí standardu je mimo jiné i široká škála číselníků. Pro současné potřeby aplikace tento zdroj vybrán nebyl, je však možné jeho využití při teoretickém pokračování vývoje aplikace přehodnotit.

Výpis péče od pojišťoven

Při rešerši byla prozkoumána i možnost využívat výpisy zdravotní péče od zdravotních pojišťoven pro evidenci v rámci aplikace. V České republice však neexistuje jednotné řešení přístupu k datům zdravotních pojišťoven (jak bylo popsáno v 1.2), tudíž by využití tohoto zdroje dat pro aplikaci zahrnovalo překlad přehledů od rozdílných zdravotních pojišťoven. Z tohoto důvodu nebyl zdroj dat využit.

2D kód balení léčiv

Pro jednodušší identifikaci léků vznikl nápad načítat informace o pobíraných léčivých přípravcích z vyfoceného balení přípravku. To obsahuje nejen kód SÚKL (33), ale i 2D kód, ze kterého lze získat GTIN (identifikátor produktu, liší se od kódu SÚKL), sériové číslo, použitelnost do a číslo šarže (34). Bohužel identifikátory GTIN pro jednotlivá léčiva nejsou veřejně dostupný, z toho důvodu nebyl nápad dále rozvíjen.

4.7 Zpracování dat na serveru

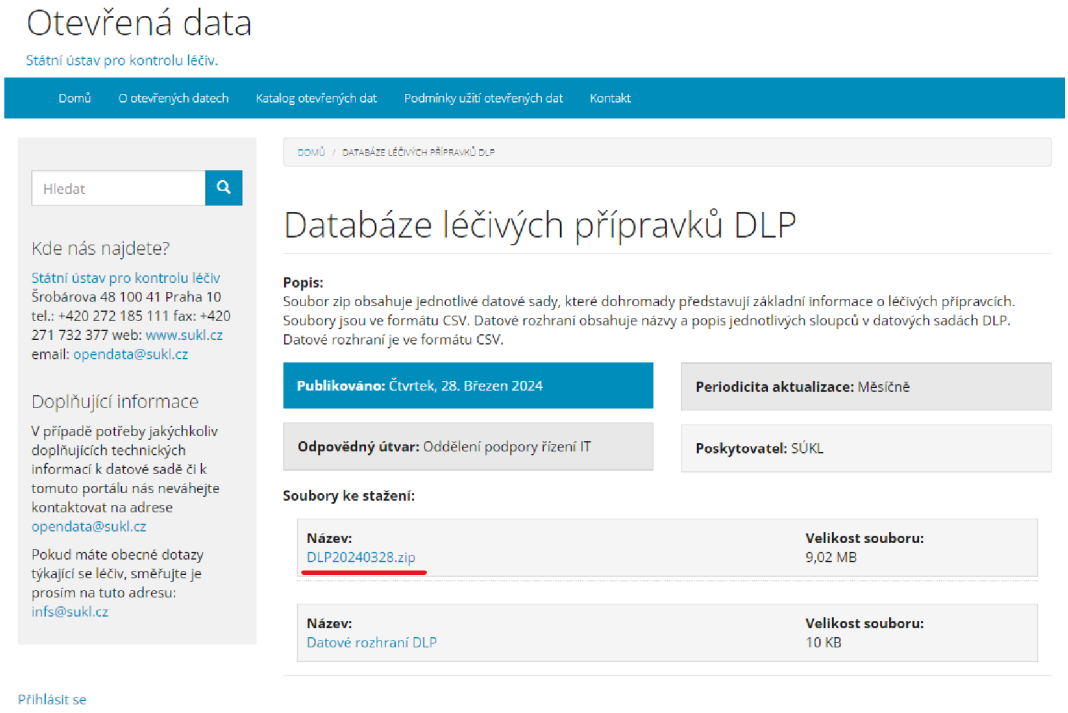
Pro zpracování a zpřístupnění těchto zdrojů aplikaci je potřeba vzdálený API server, se kterým bude aplikace komunikovat. Pro implementaci API serveru byl využit jazyk PHP 8.1 a framework Symfony 6.2. Pro vývoj i nasazení se využívají kontejnery (Docker), které umožňují snadné definování konzistentního prostředí. API server je nasazena u poskytovatele Hetzner Cloud – pro aplikaci se využívá typ se sdílenými vCPU CPX21 s operačním systémem Ubuntu 22.04. (35)

Server má dva úkoly:

1. Synchronizovat data z datových zdrojů do databáze
2. Poskytovat mobilní aplikaci data pomocí API

4.7.1 Synchronizace dat

Pro synchronizaci je potřeba nejdříve stáhnout aktuální data (v archivu ZIP či souboru CSV) z webových stránek poskytovatelů. Jelikož každý aktualizovaný soubor má jinou URL adresu, je potřeba nejprve tuto adresu získat. Samotné odkazy se sice mění, ale na webové stránce jsou vždy k dispozici na stejném místě, viz obrázek 4.9. Stačí tedy na serveru stáhnout HTML dané stránky a pomocí CSS selektorů odkaz na aktuální soubor získat. Pro procházení HTML pomocí CSS selektorů se na serveru využívá knihovna symfony/dom-crawler. (36) Následuje stažení, a případné rozbalení, souboru do lokálního uložiště.



Obrázek 4.9: Odkaz na aktuální data (označeno červeně)

Stavebním kamenem pro synchronizaci stažených dat s databází je tzv. syncer. Ten se váže k jednomu konkrétnímu CSV souboru a řádek po řádku synchronizuje data. Tyto syncery dědí z předka AbstractSyncer a mohou definovat závislosti na jiné syncery (tedy CSV soubory). Příkladem takové závislosti je soubor dlp_lecive_pripravky.csv, který u přípravku uvádí další informace (balení, forma, cesta, indikační skupina atd.), a které se odkazují na jiné soubory. V praxi to znamená, že před spuštěním synceru pro tento soubor (MedicalProductSyncer) je potřeba spustit syncery, na kterých závisí.

Každý datový zdroj má také třídu dědící z předka CsvSyncer, který definuje místo uložení souborů a syncery, které se mají spustit. Součástí předka je metoda sync, která si seřadí syncery dle jejich závislostí a postupně je spouští.

Pro práci s databází je na serveru využívána knihovna pro ORM Doctrine. Pomocí této knihovny jsou definovány třídy, resp. entity, a operace nad nimi jsou prováděny prostřednictvím repositářů. Operace jsou do databáze posílány po dávkách (batch), velikost dávky se ustálila na 5000. Původně probíhala synchronizace čistě skrze Doctrine, to však značně zpomalovalo provádění, jelikož při nastavování cizích klíčů k entitě dochází k vyhledávání entity v databázi. Z toho důvodu se v syncerech pro soubory s větším počtem záznamů místo toho využívají „raw“ SQL dotazy. Syncer tedy vytvoří dotaz ve formě řetězce a ten vrací. V metodě předka AbstractSyncer sync se pak dotazy seskupují do dávek a provádějí najednou.

Při vývoji se vyskytl problém, kdy kvůli logování operací způsobovala knihovna Doctrine memory leak. Tento problém se však týká jen vývojářského prostředí, kde to lze vyřešit přidáním vlajky (flag) „—no-debug“ při spouštění synchronizace pomocí příkazu.

Pro spouštění synchronizace pomocí příkazu se využívá knihovna symfony/console. Synchronizaci všech datových zdrojů lze spustit najednou jedním příkazem, existují i příkazy pro spuštění konkrétních zdrojů:

```
# Spuštění všech zdrojů
php bin/console syncer:all:run

# Spuštění individuálního zdroje
php bin/console syncer:mkn:run
php bin/console syncer:nrpzs:run
php bin/console syncer:sukl:run
```

Kód 8: Příkazy pro spuštění synchronizace

Pro periodickou aktualizaci dat běží na serveru Cron, který jednou měsíčně ve 3:00 ráno spustí synchronizaci dat. Pro testování trvání synchronizací jsem napsal bash skript, který N-krát spustí příkaz a do souborů ukládá dobu trvání a maximální využití RAM. Před každým spuštěním jsou vymazána data z databáze. Z tabulky 4.3 vychází, že synchronizace na lokálním stroji trvá průměrně 58,22 sekund. Z tabulky 4.4 pak vychází, že na vzdáleném stroji je trvání průměrně 93,84 sekund.

Lokální stroj	NRPZS		DLP		MKN		Celkem
	Čas	Paměť	Čas	Paměť	Čas	Paměť	Čas
Měření #1	24,04 s	49,99 MB	30,79 s	25,00 MB	4,45 s	14,56 MB	59,28 s
Měření #2	23,87 s	47,75 MB	29,62 s	25,28 MB	4,00 s	19,40 MB	57,49 s
Měření #3	24,42 s	47,75 MB	29,56 s	25,28 MB	3,90 s	19,40 MB	57,88 s
Průměr	24,11 s	48,50 MB	29,99 s	25,19 MB	4,12 s	17,79 MB	58,22 s

Tabulka 4.3: Doba trvání na lokálním stroji

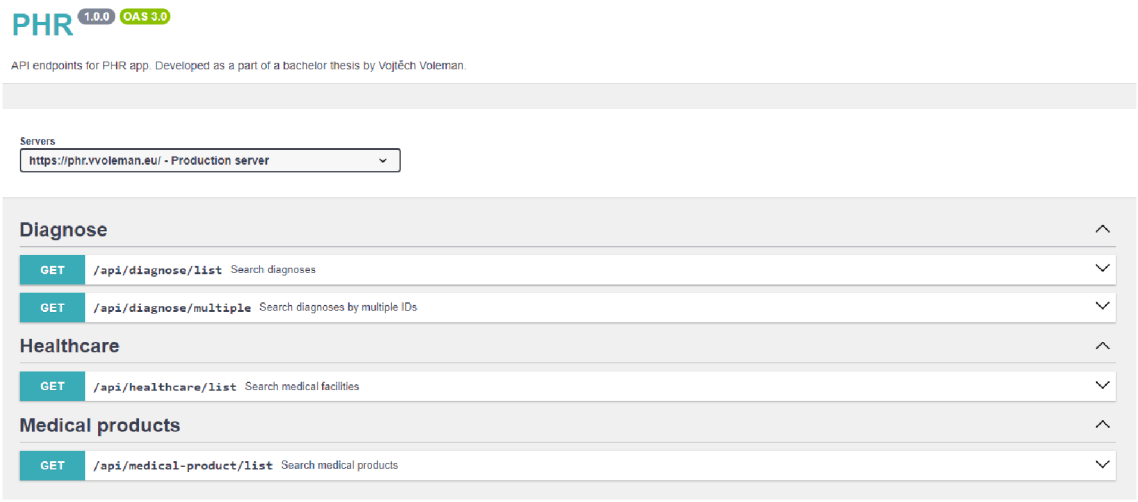
Vzdálený stroj	NRPZS		DLP		MKN		Celkem
	Čas	Paměť	Čas	Paměť	Čas	Paměť	Čas
Měření #1	31,53 s	47,85 MB	51,67 s	25,48 MB	9,70 s	19,53 MB	92,90 s
Měření #2	31,60 s	47,85 MB	55,17 s	25,48 MB	10,08 s	19,53 MB	96,85 s
Měření #3	31,53 s	47,85 MB	50,81 s	25,48 MB	9,42 s	19,53 MB	91,76 s
Průměr	31,55 s	47,85 MB	52,55 s	25,48 MB	9,73 s	19,53 MB	93,84 s

Tabulka 4.4: Doba trvání na vzdáleném stroji

4.7.2 Poskytování dat pomocí API

Pro vytváření koncových bodů (endpoint) se využívají klasické Symfony kontrolery, které dědí z mnou vytvořeného předka BaseApiController. Tato abstraktní třída poskytuje základní metody pro jednotné odesílání odpovědí a chyb. Data z endpointů se vrací ve formátu JSON.

V aplikaci existují tři kontrolery, každý z nich obsahuje endpoint */list*, který slouží pro výpis dat dle použitého filtru. Pro dokumentování API endpointů jsem využil bundle NelmioApiDocBundle, který do Symfony aplikace přináší možnost dokumentování kontrolerů, generování schéma podle OpenApi specifikací a vizualizaci dokumentace pomocí Swagger UI. Dokumentace je dostupná v rámci nasazeného API serveru. (37) Schéma je k dispozici v repositáři pro tento projekt v souboru schema.json. (17)



Obrázek 4.10: Ukázka dokumentovaných endpointů

5 Testování

Při tvorbě jakékoliv aplikace vzniká potřeba ověřit, zda splňuje požadavky zadané klientem, či zda dokáže ustát neočekávatelné chování uživatele. Existuje spousta způsobů testování aplikací, v této kapitole bude pozornost věnována těm, které byly prováděny v průběhu vývoje této aplikace a následnému testování a zhodnocení konkrétními uživateli.

5.1 Při vývoji

5.1.1 Unit testy

Základním principem při vývoji aplikací je rozdělování komplexních problémů na menší, lépe zvládnutelné úkoly. Tím se snižuje riziko výskytu chyb, případné problémy lze řešit na konkrétních místech. Z tohoto důvodů se provádí tzv. unit testing, který testuje právě bloky kódu. Pro jednodušší tvorbu testů je důležitá modulárnost kódu, tedy že testovaný kód není přímo provázán s jiným kódem (coupling).

V mobilní aplikaci vyvíjené v rámci této práce se využívá pro unit testing knihovna JUnit, pro mockování jsou využívány knihovny Mockito a Mockk.

V aplikaci se testují zejména Use-casy, které jsou klíčové pro správné chování aplikace - tedy plánování léků, filtrování výsledků, přidávání nových zpráv. Mimo ty jsou testovány i některé fasády (facade) či továrny (factory).

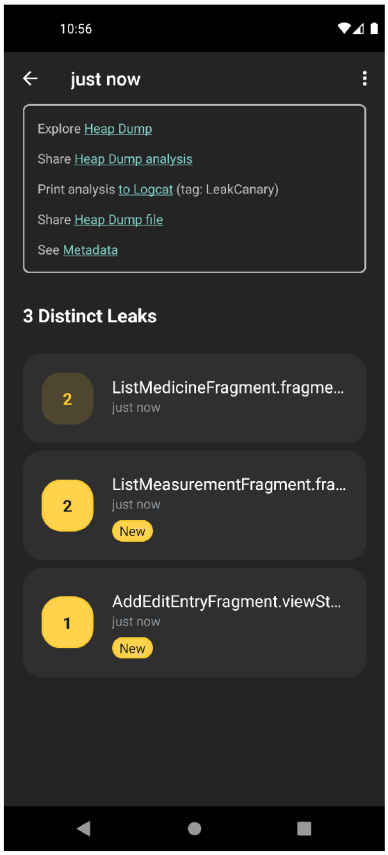
5.1.2 Únik paměti

Únik paměti (memory leak) je situace, kdy není prostředek po konci využívání uvolněn, vlastník k prostředku ztratí referenci a Garbage Collector není schopen prostředek uvolnit z paměti. To má za následek postupné zaplňování paměti, které ovlivňuje výkon aplikace a může způsobovat případné pády. V Androidu se úniky paměti často objevují kvůli špatnému nakládání s prostředky v lifecycle-aware komponentách (Activity, Fragment…). (38)

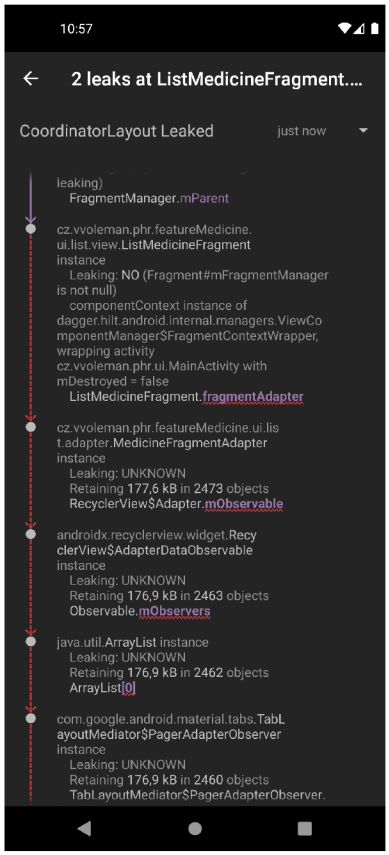
Možný únik paměti lze poznat zhoršením výkonu aplikace. V Android Studiu lze nalézt nástroj pro analýzu výkonu a paměti (Profiler), který dokáže vypsat a uložit výpis haldy (heap dump). Z výpisu pak Android Studio dokáže odhadnout Aktivity či Fragmenty, kde může k únikům docházet. (39)

Při testování úniků v této aplikaci jsem kromě Profileru využil i knihovnu LeakCannary. Ta za běhu aplikace analyzuje, zda někde dochází k úniku a vy-

generuje zprávu s informacemi pro řešení situace, jak se zřemé na obrázku 5.1. Po rozkliknutí konkrétního úniku lze pomocí stacktrace určit místo, kde k úniku dochází, viz obrázek 5.2.



Obrázek 5.1: Výpis úniků paměti



Obrázek 5.2: Stacktrace úniku

5.2 Uživatelské testování

Uživatelské testování je tím nejdůležitějším, protože se zde zjistí, zda je aplikace navržena dobře, zda je uživatelsky přívětivá a zda dokáže odolat neočekávanému chování uživatelů. Testování probíhalo ve dvou fázích – testování aplikace na emulátoru a testování na reálném zařízení.

5.2.1 Testování na emulátoru

První forma uživatelského testování byla provedena na dálku za pomoci softwaru pro vzdálené ovládání počítače a zúčastnili se ho dva uživatelé – jeden běžný uživatel a jeden uživatel se zdravotní indispozicí. Uživatelé pomocí myši ovládali aplikaci v emulátoru. Uživatelům byla v krátkosti vysvětlena premisa aplikace – mobilní aplikace, která slouží pro záznam jejich zdravotních informací. Poté již bylo na nich, aby aplikaci prozkoumávali a pozorovat jejich chování.

Uživatelé začali v modulu „Lékařské zprávy“. První podstatnou zpětnou vazbou byla žádost, aby při vytváření nové lékařské zprávy šlo z formuláře rovnou i vytvořit novou kategorii problému či přidat lékařského pracovníka. Tato žádost se později objevila i v ostatních modulech. Další zpětnou vazbou byla i žádost o nějakou formu nápovědy či vysvětlení funkcí na obrazovce. Po dalších dotazech na formu či rozsah byla uživatelem navrženo tlačítko v horní liště aplikace, které by po kliknutí zobrazilo dialogové okno s popisem konkrétních prvků na obrazovce. Při přidávání příloh k lékařské zprávě vadilo, že ne vždy aplikace správně přečte informace z fotografie. Jeden uživatel navrhl funkci pro výřez dokumentu z fotografie. Mimo tyto věci uživatelé pochopili funkce tohoto modulu, a kromě některých vizuálních nedostatků (malé dialogové okno v přidání nové lékařské zprávy, skromný detail zprávy) neměli s modulem problém.

V rámci první obrazovky modulu „Léky“ jednomu uživateli vadilo označení „itinerář“ pro výpis dnes podávaných léků. Stěžoval si na to, že si pod označením „itinerář“ nedokáže prvotně nic představit a stěží to dokáže vyslovit. Navrhl změnit označení na „dnešní podání“. Ve formuláři pro přidání léku měli oba uživatelé problém pochopit, co po nich aplikace chce – zejména v sekci pro výběr podání. Po vysvětlení, že se jedná o nastavení dávkování pro konkrétní časy to uživatelé pochopili, ale navrhli přidání již zmiňované nápovědy a graficky bloky lépe oddělit.

V modulu „Měření“ měli uživatelé opět problém s pojmenováním prvků a funkcí. Místo „skupin měření“, které slouží jako předpis pro měření, navrhli pojmenování „moje měření“ či jenom „měření“. Při vytváření nové skupiny měření uživatelům chyběla nápověda, vadilo jim označení „pole měření“. Po vysvětlení uživatelé vše pochopili.

Modul „Plány návštěv“ nezpůsoboval uživatelům žádný problém, jediná připomínka byla k možné změně „Označení“ návštěvy na „Název“ návštěvy.

U seznamu zdravotnických pracovníků v sekci „Kategorizace“ si uživatelé stěžovali na to, že při dlouhém názvu dochází ve výpisu k nevzhlednému formátování textu. V detailu pracovníka chyběla uživatelům mapa adresy lékařského zařízení či možnost prokliku do mapové aplikace. U kategorií problémů chyběla uživatelům možnost výběru vlastní barvy při vytváření kategorie.

Při vyplňování dat uživatelům chyběla možnost přidat všechna data týkající se jedné návštěvy najednou. Tato funkce byla prozkoumána v požadavku 2.2.4 ”Průvodce po návštěvě zařízení”. Z časových důvodů jsem se rozhodl tento požadavek nevypracovávat, jelikož se ale jedná o nadstavbu již existujících funkcí, nepřichází uživatel o žádné funkce. Při pokračování vývoje lze o tuto funkcionalitu aplikaci rozšířit.

Zhodnocení

Během testování vyplynuly některé problémy či možná rozšíření. Pro mě nejdůležitější je žádost o přidání integrované nápovědy do aplikace a možnost rychlého přidání kategorie či pracovníka. V tabulce 5.1 jsou nálezy vypsány i s očekávanou náročností.

Název	Popis	Typ	Náročnost
Rychlé přidání kategorie či pracovníka	Ve formulářích přidat možnost vytvořit kategorii/pracovníka, aniž by bylo nutné opustit současný formulář	Rozšíření	Realizovatelné, časově náročné
Nápověda	U složitějších obrazovek přidat tlačítko pro dialogové okno popisující prvky	Rozšíření	Realizovatelné, časově nenáročné
Chybovost image-to-text	Aplikace ne vždy úspěšně přečte informace z fotografie.	Chyba	Nutné prozkoumat
Výřez dokumentu	Umožnit uživateli vybrat rohy papíru a fotografii dle toho upravit	Rozšíření	Realizovatelné, časově náročné
Označení "Itinerář"	Najít lepší označení pro výpis všech položek pro současný den	Chyba	Jednoduché
Označení "Skupiny měření"	Najít lepší označení pro předpis budoucích měření	Chyba	Jednoduché
Přetékající text	U seznamu pracovníků lépe pracovat s delšími názvy	Chyba	Jednoduché
Mapa adresy zařízení	V detailu přidat mapu pro adresu lékařského zařízení	Rozšíření	Nutné prozkoumat
Vlastní barva pro kategorii	Při úpravy kategorie přidat možnost vybrat vlastní barvy	Rozšíření	Jednoduché
Průvodce po návštěvě zařízení	Požadavek 2.2.4	Rozšíření	Realizovatelné, časově náročné

Tabulka 5.1: Zpětná vazba od uživatelů

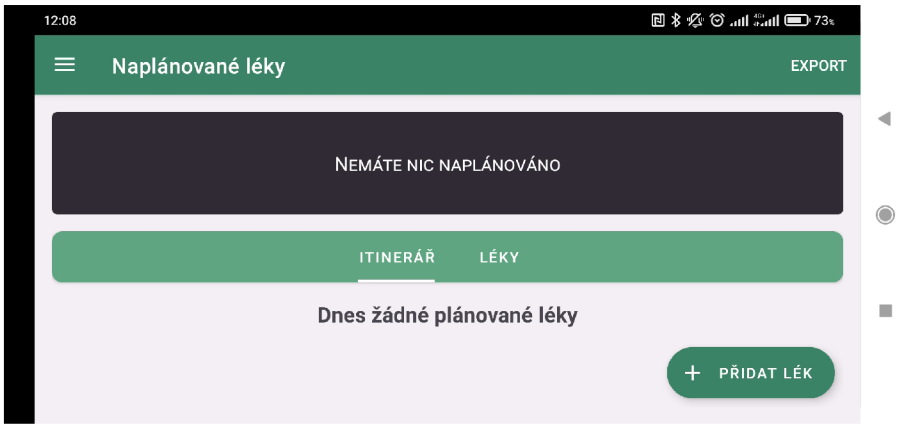
5.2.2 Testování na zařízení

Pro testování na fyzických zařízeních byl vybrán mobilní telefon Xiaomi Note 8T – Android 11, obrazovka 6.3“ a rozlišení 2340 x 1080 px, a tablet Lenovo Tab M10 FHD Plus – Android 10, obrazovka 10.3“ a rozlišení 1920 x 1200 px. V rámci testování na zařízeních bylo testováno především chování uživatelského rozhraní a náročnost aplikace na prostředky zařízení.

Nainstalovaná aplikace zabírá na mobilním telefonu 33 MB a na tabletu 29,24 MB. Pokud není zařízení připojené k internetu, je aplikace stále funkční, ale nelze vyhledávat konkrétní léky, diagnózy či lékařská zařízení. To může představovat možný problém pro uživatele, kteří si chtějí zapsat nové léky hned po předepsání, ale nevyužívají mobilní datové připojení. Z hlediska využití baterie nepředstavuje aplikace žádný problém.

Při procházení aplikace na tabletu si lze všimnout toho, že uživatelské rozhraní bylo primárně vytvářeno pro mobilní telefony. Jednotlivé prvky se roztáhnou na plnou šířku, ale jejich výška se nemění, viz obrázky 5.3 a 5.4. Lze tedy uvažovat o alternativním zobrazení prvků pro zařízení se širokou obrazovkou. U mobilu zabírá část obrazovky komponenta "Další lék", pro samotný výpis léků pak nezbývá moc prostoru. Řešením by bylo možnost scrollovat celou obrazovku.

Menších problémů si lze všímat u notifikací, konkrétně u Alarmů sloužících pro probuzení aplikace po uplynulé době. V implementaci byl zvolen typ alarmu pro opakované provádění, který však není vždy přesný. Odchylka od požadovaného času se liší dle vytížení zařízení, většinou ale bývá zpoždění v řádech minut. Řešením tohoto problému by bylo využití jednorázových přesných alarmů (exact alarm), které by vyžadovaly spouštění po uplynutí předchozího.



Obrázek 5.3: Výpis léků v landscape módu na mobilu

Obrázek 5.4: Výpis léků v landscape módu na tabletu

6 Nasazení

Jedním z posledních kroků tvorby aplikací je postarat se, že aplikace je připravena pro koncového uživatele či zákazníka. Mezi tyto úkoly patří otestování aplikace, kontrola kvality kódu a připravení konečného sestavení (build).

6.1 CI/CD

Pojem CI/CD se skládá ze dvou principů. Jako continual integration (CI) se rozumí využívání nástrojů pro automatizaci úkolů v průběhu vývoje. Pro tuto aplikaci se využívá verzovací systém Git a platforma GitHub, která umožňuje definovat sadu akcí (Github Actions), které se mají vykonat po nějaké události (commit, merge, pull request…). V rámci této aplikace existují akce pro kontrolu unit testů a pro statickou analýzu kódu pomocí nástroje Detekt. Ty se spouštějí při událostech pull request a push do hlavní větve (main).

Continual deliverence (CD) má za cíl automatické sestavení či nasazení aplikace. Stejně jako u CI, i zde je využíván Github Actions. Pro mobilní aplikaci existuje akce pro sestavení instalačního souboru a vytvoření „vydání“ (release). Tato akce je spuštěna po přidání štítku (tag) ve formátu zápisu verze (vX.Y.Z). Sestavená aplikace je poté digitálně podepsána.

Pro API server se akce spustí po události push do větve deploy. Akce přes SSH spustí skript na vzdáleném zařízení, který stáhne změny z repositáře a znovu nastartuje kontejnery (Docker).

6.2 Distribuce aplikace

V operačním systému Android lze distribuovat aplikace prostřednictvím platformy Google Play. Pro přidání aplikace na tuto platformu je nutné mít aktivní vývojářský účet, který je potřeba potvrdit uhrazením jednorázového poplatku ve výši 25 USD. (40) Od listopadu 2023 platí pro nově vytvořené vývojářské účty povinnost provést uzavřené testování aplikace s minimálně 20 testery po minimální dobu 14 dnů. (41) Z těchto důvodů není pro tuto práci zvoleno nasazení aplikace na platformu Google Play.

Na místo toho je pro distribuci aplikace zvoleno vydávání verzí s instalačním souborem prostřednictvím GitHubu. Pro tento způsob instalace je potřeba u koncových uživatelů povolit instalaci aplikací z neznámých zdrojů.

7 Závěr

Prvním cílem této práce bylo provést rešerši způsobů a možností, pomocí kterých mohou pacienti přistupovat a pracovat se svojí zdravotnickou dokumentací. Při této rešerši vyšlo najevo, že neexistuje jednotný způsob přenosu dokumentace a některé skupiny pacientů jsou nuceny fyzicky přenášet dokumentaci mezi poskytovateli zdravotní péče. Byl představen termín PHR a ukázány existující řešení, která mohou pacientům se zdravotní indispozicí pomoci.

Na základě konzultací s osobami, které podobné problémy trápí, jsem vypracoval specifikaci požadavků, která definovala skupiny cílových uživatelů – uživatelé se zdravotní indispozicí, běžní uživatelé a pečovatelé. Při zpracování všech požadavků konzultovaných osob byly tyto požadavky rozděleny na čtyři hlavní moduly, tedy „Lékařské zprávy“, „Léky“, „Měření“ a „Plány návštěv“, a několik podmodulů – „Správa pacientů“, „Správa kategorií problémů“ a „Správa lékařských pracovníků“.

Při návrhu řešení jsem nejprve vypracoval wireframe, na kterém se již při definici požadavků zkoušelo, zda a jak lze dané požadavky do aplikace zařadit. Na jeho základě jsem posléze vytvořil barevnou paletu a mockup aplikace v nástroji Figma, který již obsahoval konkrétní vzhledy obrazovek a komponent. Pro testování konceptu byl vytvořen prototyp, pomocí kterého šlo mockup interaktivně procházet. Závěrem zde bylo popsáno fungování aplikace na jednotlivých obrazovkách.

Pro implementaci aplikace jsem vybral programovací jazyk Kotlin. Aplikace byla vyvíjena podle systémové architektury Clean Architecture, konkrétní implementace byla řešena dle knihy „Clean Architecture for Android“ (Boudjnah, Eran, 2022). Implementace z této knihy však není využívána 1:1, při vývoji byly prostředky upravovány dle potřeb a problémů v této práci. Pro ukládání dat na lokálním zařízení byla využita knihovna Room, která slouží jako ORM nad SQLite databází.

Dalším cílem této práce bylo prozkoumat a vhodně využít veřejně dostupná data v oblasti zdravotnictví. Pro současné potřeby aplikace jsem využil databázi léčivých přípravků, Mezinárodní klasifikaci nemocí a Národní registr poskytovatelů zdravotních služeb. Vytvořil jsem API server, který má za cíl synchronizovat data z těchto zdrojů v časové periodě a nabízet přístup k datům pomocí koncových bodů, které vrací data ve formátu JSON. Tento server byl vyvíjen v jazyce PHP a frameworku Symfony. Aplikace s API serverem komunikuje pomocí knihovny Retrofit.

V závěru práce byla aplikace testována uživateli. Při něm se kromě drobných problémů přišlo i na potřebná a žádaná rozšíření. Nejvíce bylo žádáno přidání dialogových oken s nápovědou. Tyto chyby a rozšíření byly popsány v kapitole 5.2.

V současné podobě může být aplikace užitečná pro zamýšlené cílové uživatele, kterým umožní uchovávat si část svých zdravotnických dat na jednom místě. Při možném pokračování vývoje lze uvažovat o lepší analýze lékařských zpráv, kde bych rád zlepšil úspěšnost extrakce informací nebo udělal komplexnější extrakci dat ze samotného textu zprávy. To by však vyžadovalo konzultace s odborníky ze zdravotnictví. Dalším možným rozšířením může být možnost sdílení dat a synchronizace aplikace mezi více zařízeními či využití dat z dalších externích zdrojů (např. data zdravotních pojišťoven). Mimo to si dokáži představit podrobněji využívat již existující zdroje, například pracovat s velikostí balení léků zmíněné v 4.6.1.

Použitá literatura

[1] *Zdravotnická dokumentace*. 2024. Dostupné také z: https://www.nzip.cz/clanek/1074-zdravotnicka-dokumentace.

[2] *EPACS*. 2024. Dostupné také z: http://www.epacs.cz/epacs/faces/pages/index.xhtml;jsessionid=1uskuv1pmlm1w1hygtuexcnik2.

[3] *Technický popis projektu*. 2024. Dostupné také z: https://www.emedocs.cz/technicky-popis-projektu-c180.

[4] *Právo na nahlížení do zdravotnické dokumentace*. 2024. Dostupné také z: https://www.nzip.cz/clanek/242-pravo-na-nahlizeni-do-zdravotnicke-dokumentace.

[5] *Poskytovatelé*. 2024. Dostupné také z: https://www.nixzd.cz/poskytovatele.

[6] *Novinky v eReceptu: Nabídne evidenci očkování a elektronické poukazy na zdravotnické prostředky*. 2021. Dostupné také z: https://www.sukl.cz/sukl/novinky-v-ereceptu-nabidne-evidenci-ockovani-a-elektronicke.

[7] ČESKÉ REPUBLIKY, Ministerstvo zdravotnictví. Národní strategie elektronického zdravotnictví České republiky 2016 – 2020. [B.r.]. Dostupné také z: https://ncez.mzcr.cz/sites/default/files/Attachment/Narodni_strategie_elektronickeho_zdravotnictvi_v1.0_1.pdf.

[8] *Často kladené otázky*. 2024. Dostupné také z: https://portal.kzcr.eu/Pages/Home/View/faq.

[9] BOUAYAD, Lina, Anna IALYNYTCHEV a Balaji PADMANABHAN. Patient Health Record Systems Scope and Functionalities: Literature Review and Future Directions. *Journal of Medical Internet Research*. 2017, roč. 19, č. 11, s. -. ISSN 1438-8871. Dostupné z DOI: 10.2196/jmir.8073.

[10] MENACHEMI, Nir a Taleah H COLLUM. Benefits and drawbacks of electronic health record systems. *Taylor*. 2011. Dostupné také z: https://www.tandfonline.com/doi/abs/10.2147/RMHP.S12985.

[11] KOSKINEN, Jani a Minna RANTANEN. What is a PHR? Definitions of Personal Health Record (PHR) Used in Literature—A Systematic Literature Review. In: 2020/08/20, s. 24–49. ISBN 978-3-030-57846-6. Dostupné z DOI: 10.1007/978-3-030-57847-3_2.

[12] *Mobilní aplikace Moje Ambulance*. 2024. Dostupné také z: https://www.mojeambulance.cz/mobilni-aplikace-moje-ambulance-1/.

[13] *Interakce léčiv*. 2024. Dostupné také z: https://www.wikiskripta.eu/w/Interakce_l%5C%C3%5C%A9%5C%C4%5C%8Div.

[14] GLINZ, M. On Non-Functional Requirements. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. 2007, s. 21–26. ISSN 2332--6441. Dostupné z DOI: 10.1109/RE.2007.45.

[15] *Principles for improving app accessibility*. 2024. Dostupné také z: https://developer.android.com/guide/topics/ui/accessibility/principles.

[16] *Repositář pro mobilní aplikaci* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://github.com/vvoleman/phr_android.

[17] *Repositář pro API server* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://github.com/vvoleman/phr.

[18] HARTINGER, David. *Lekce 3 - Třívrstvá architektura a další vícevrstvé architektury* [online]. 2024. [cit. 2024-05-13]. Dostupné z: https://www.itnetwork.cz/trivrstva-architektura-a-dalsi-vicevrstve-architektury.

[19] BOUNDJNAH, Eran. *Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps (English Edition)*. 1. vyd. BPB Publications, 2022. ISBN 9355510497.

[20] *Multi-Project Build Basics*. 2024. Dostupné také z: https://docs.gradle.org/current/userguide/intro_multi_project_builds.html.

[21] *Kotlin on Android. Now official*. 2017. Dostupné také z: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.

[22] *PdfDocument* [online]. 2024. [cit. 2024-05-13]. Dostupné z: https://developer.android.com/reference/android/graphics/pdf/PdfDocument.

[23] *Get a result from an activity* [online]. 2024. [cit. 2024-05-13]. Dostupné z: https://developer.android.com/training/basics/intents/result.

[24] *ML Kit* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://developers.google.com/ml-kit.

[25] D., Cook John. *Regular expression for ICD-9 and ICD-10 codes*. 2019. Dostupné také z: https://www.johndcook.com/blog/2019/05/05/regex_icd_codes/.

[26] *Security guidelines*. 2024. Dostupné také z: https://developer.android.com/privacy-and-security/security-tips.

[27] KHAMBHAYTA, Jai. *Android: Encrypting Existing Room Databases with SQLCipher: Encrypting Existing Room Databases with SQLCipher*. 2024. Dostupné také z: https://medium.com/@khambhaytajaydip/encrypting-an-existing-room-database-with-sqlcipher-in-android-50cdc98fe6c.

[28] *Databáze léčivých přípravků*. 2024. Dostupné také z: https://opendata.sukl.cz/?q=katalog/databaze-lecivych-pripravku-dlp.

[29] *O MKN-10*. 2024. Dostupné také z: https://mkn10.uzis.cz/o-mkn.

[30] *MKN Mezinárodní statistická klasifikace nemocí a přidružených zdravotních problémů*. 2024. Dostupné také z: https://www.uzis.cz/index.php?pg=vystupy--knihovna%5C&id=3371.

[31] *Data ke stažení*. 2024. Dostupné také z: https://nrpzs.uzis.cz/index.php?pg=home--download.

[32] *DASTA a projekty e-Health, další rozvoj*. 2024. Dostupné také z: https://ciselniky.dasta.mzcr.cz/hypertext/201540/hypertext/MZAXB.htm.

[33] *Principy identifikace humánních léčivých přípravků v ČR*. 2024. Dostupné také z: https://www.sukl.cz/leciva/principy-identifikace-humannich-lecivych-pripravku-v-cr.

[34] *Ochranné prvky pro humánní léčivé přípravky - Otázky a odpovědi*. 2018. Dostupné také z: https://www.czmvo.cz/file.php?id=282.

[35] *Hetzner Cloud*. 2024. Dostupné také z: https://www.hetzner.com/cloud/.

[36] *The DomCrawler Component*. 2024. Dostupné také z: https://symfony.com/doc/current/components/dom_crawler.html.

[37] *Dokumentace pro API* [online]. 2024. [cit. 2024-04-30]. Dostupné z: https://phr.vvoleman.eu/api/doc.

[38] AMALFITANO, Domenico et al. Do Memories Haunt You? An Automated Black Box Testing Approach for Detecting Memory Leaks in Android Apps. *IEEE Access*. 2020, roč. 2020, č. vol. 8, s. 12217–12231. ISSN 2169-3536. Dostupné z DOI: 10.1109/ACCESS.2020.2966522.

[39] *Inspect your app's memory usage with Memory Profiler*. 2024. Dostupné také z: https://developer.android.com/studio/profile/memory-profiler.

[40] *How to get started with Play Console*. 2024. Dostupné také z: https://support.google.com/googleplay/android-developer/answer/6112435?hl=en.

[41] *App testing requirements for new personal developer accounts*. 2024. Dostupné také z: https://support.google.com/googleplay/android-developer/answer/14151465?hl=en.

A Přílohy

A.1 Pacientský souhrn

Krajská zdravotní nemocnice Ústeckého kraje

Pacientský souhrn
Voleman Vojtěch

A - Identifikace a kontaktní údaje pacienta

Jméno pacienta:
Rodné číslo:
Číslo dokladu:
Datum narození:
Pohlaví
Adresa trvalého bydliště:
Resortní identifikátor:
E-mail:
Mobilní telefon:
Státní občanství:

B - Identifikační a kontaktní údaje registrujícího poskytovatele

Název a adresa:
Kontakty:

C - Identifikační a kontaktní údaje zákonných zástupců, opatrovníků nebo kontaktních osob

Vztah	Info	Jméno	Adresa	Email	Mobil	Telefon
Příbuzný						

D - Informace o zdravotním pojištění

Kód ZP:

E - Urgentní informace

a) Alergie – údaj není znám

b) Ostatní rizikové faktory – údaj není znám

F - Souhrnná anamnéza

a) Provedená očkování – údaj není znám

b) Minulé zdravotní problémy a diagnózy – údaj není znám

c) Významné chirurgické výkony se vztahem k minulým zdravotním problémům a diagnózám – údaj není znám

G - Současné zdravotní problémy a diagnózy

a) Současné zdravotní problémy a diagnózy – údaj není znám

b) Významné chirurgické výkony se vztahem k současným zdravotním problémům a diagnózám – údaj není znám

c) Zdravotnické prostředky, na kterých závisí nebo může záviset pacientův zdravotní stav – údaj není znám

d) Léčebná doporučení, která nezahrnují medikamenózní léčbu – údaj není znám

e) Soběstačnost nebo invalidita pacienta – údaj není znám

H - Užívané léky – údaj není znám

I - Faktory životního stylu – údaj není znám

J - Těhotenství – údaj není znám

K - Fyzikální nález

a) Krevní tlak systolický a diastolický a tepová frekvence – údaj není znám

b) Výška a hmotnost

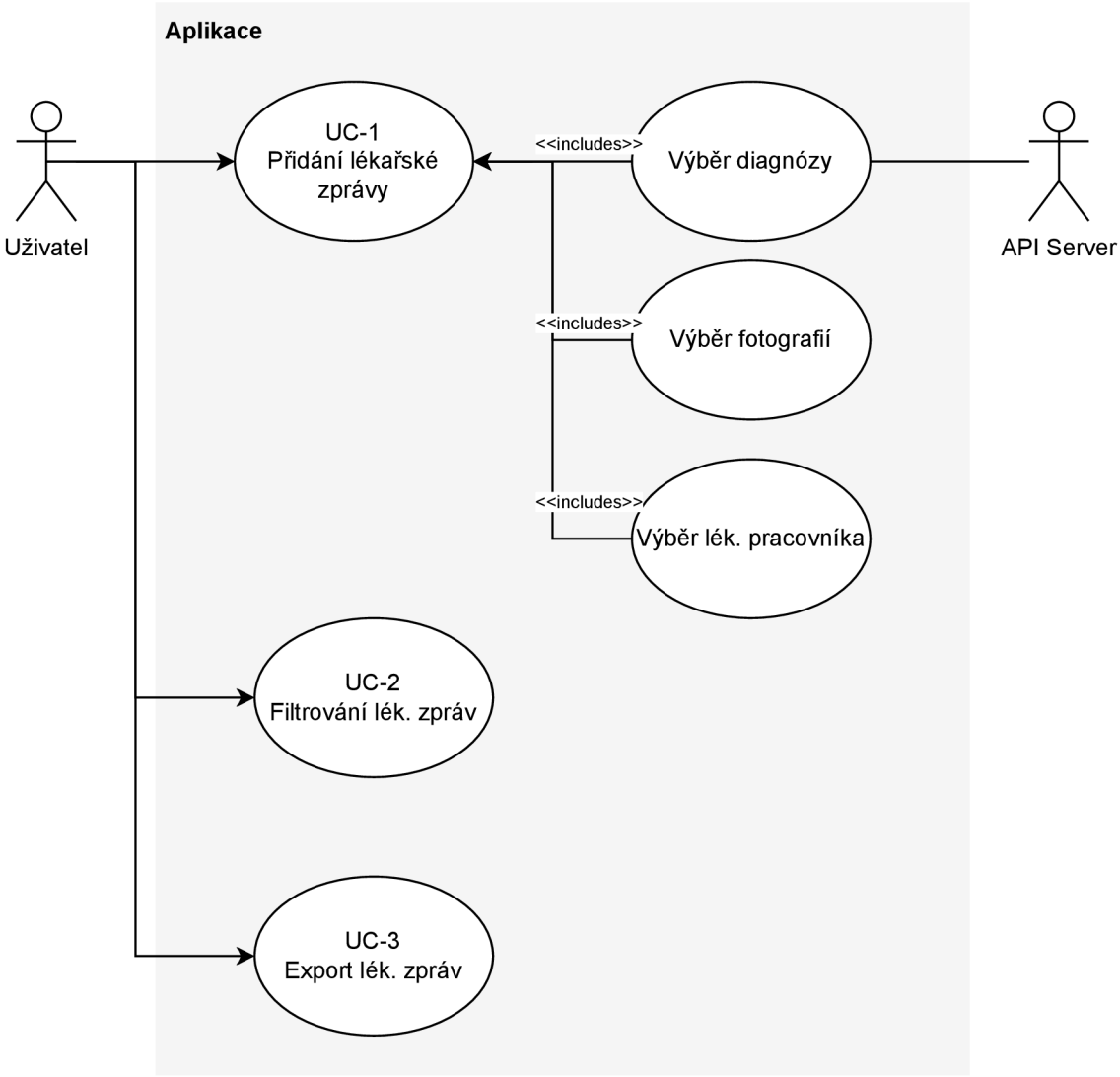
Dat. měření	Výška (cm)	Hmotnost (kg)

L - Diagnostické testy

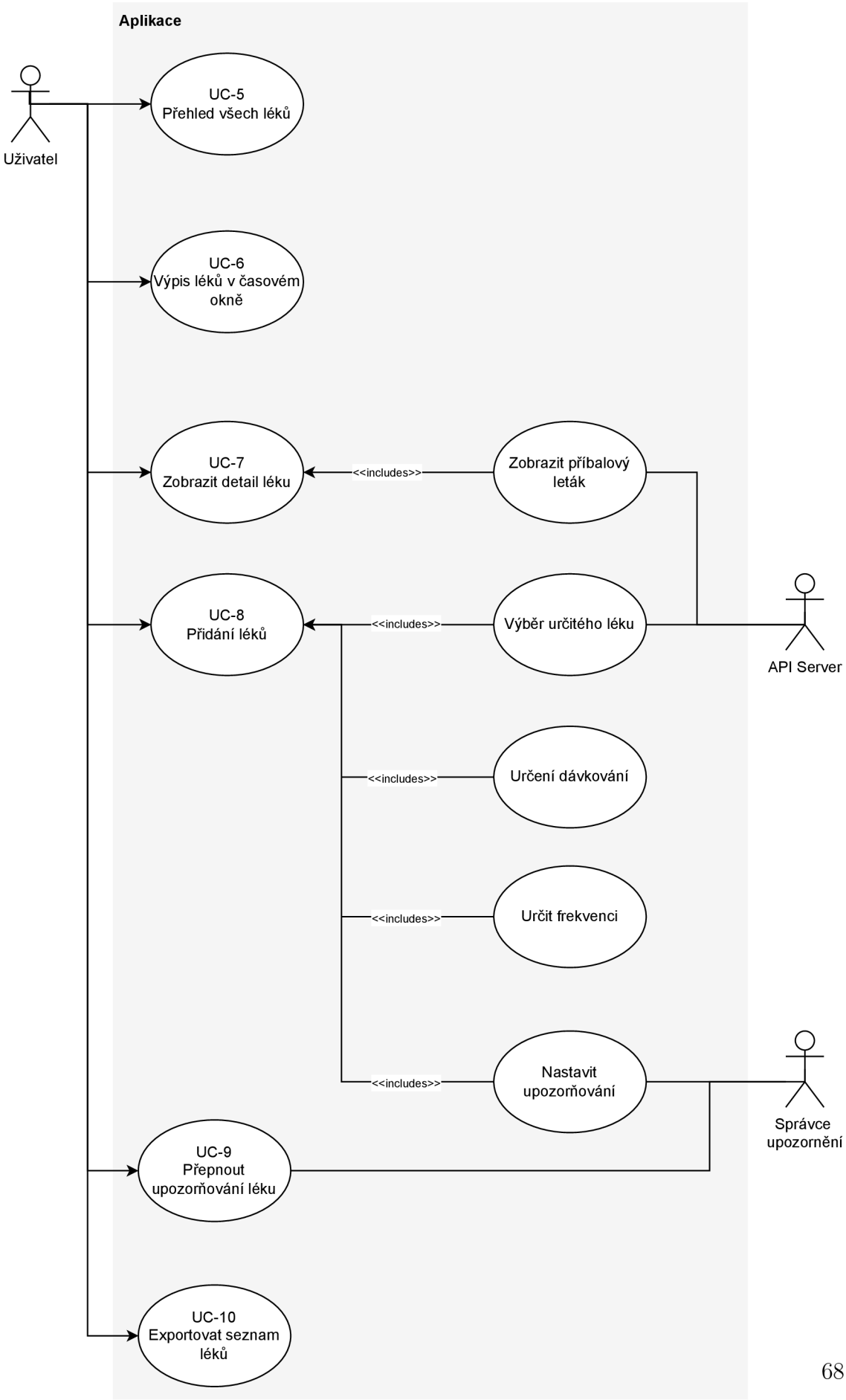
a) Krevní skupina a Rh faktor – údaj není znám

Obrázek A.1: Ukázka pacientského souhrnu od Krajská zdravotní, a.s.

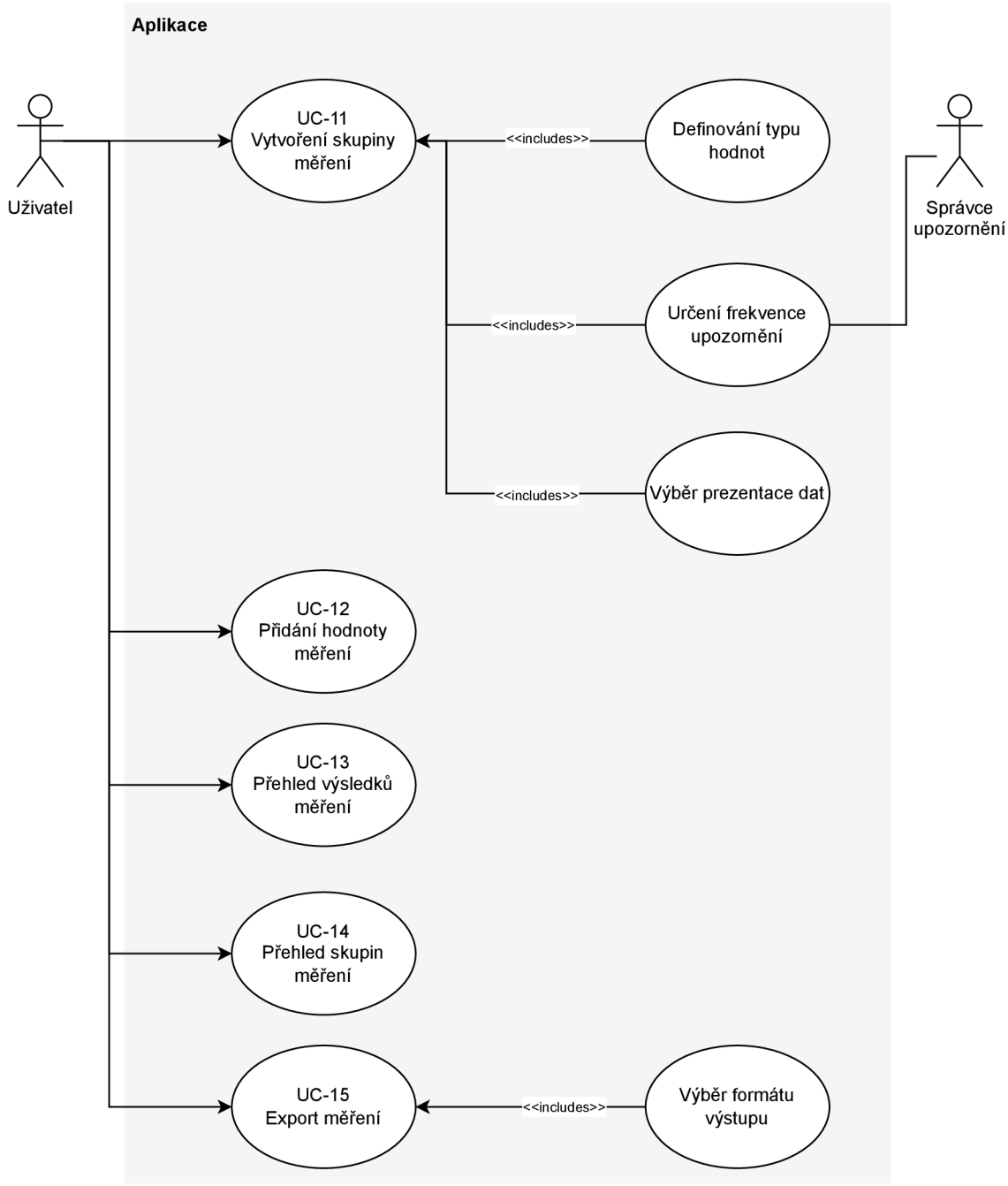
A.2 Funkční požadavky



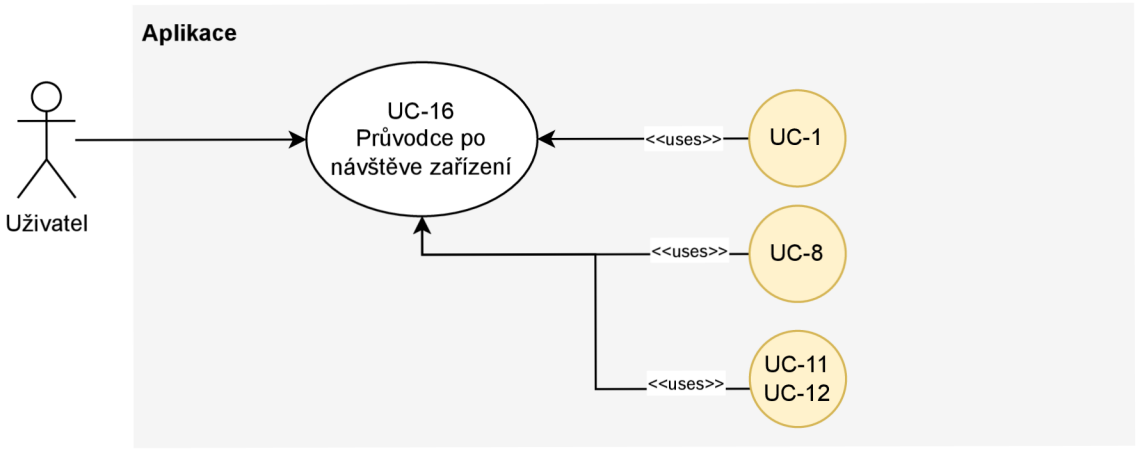
Obrázek A.2: Funkční požadavky modulu „Lékařské zprávy“



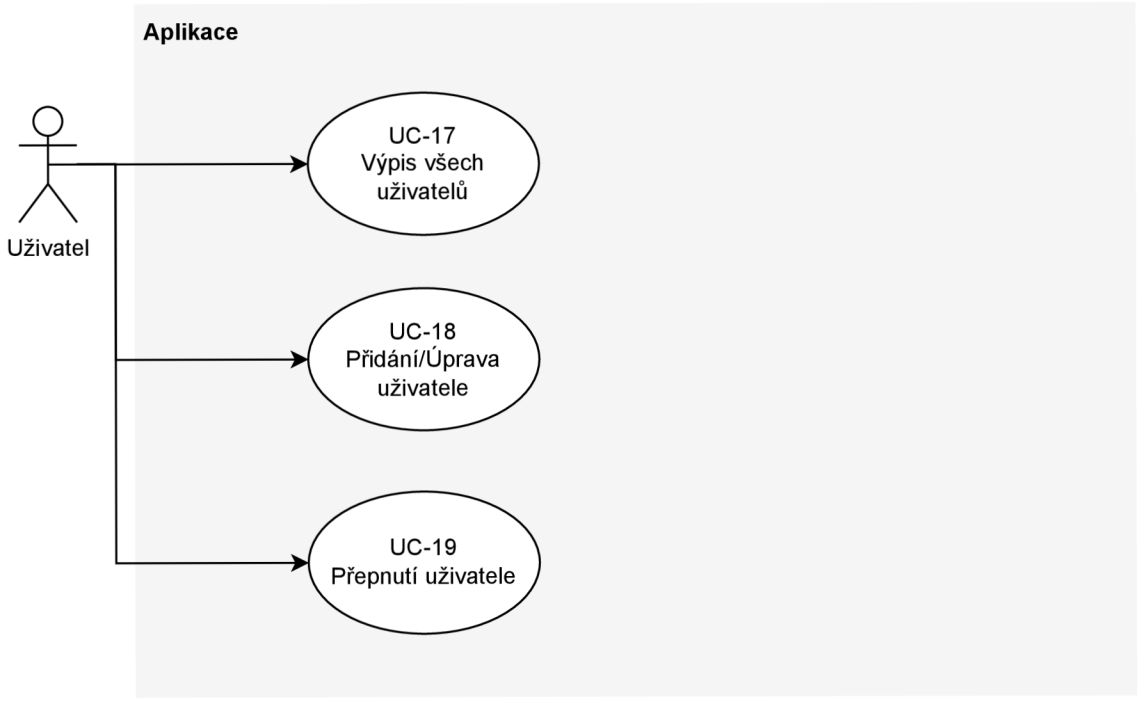
Obrázek A.3: Funkční požadavky modulu „Léky“



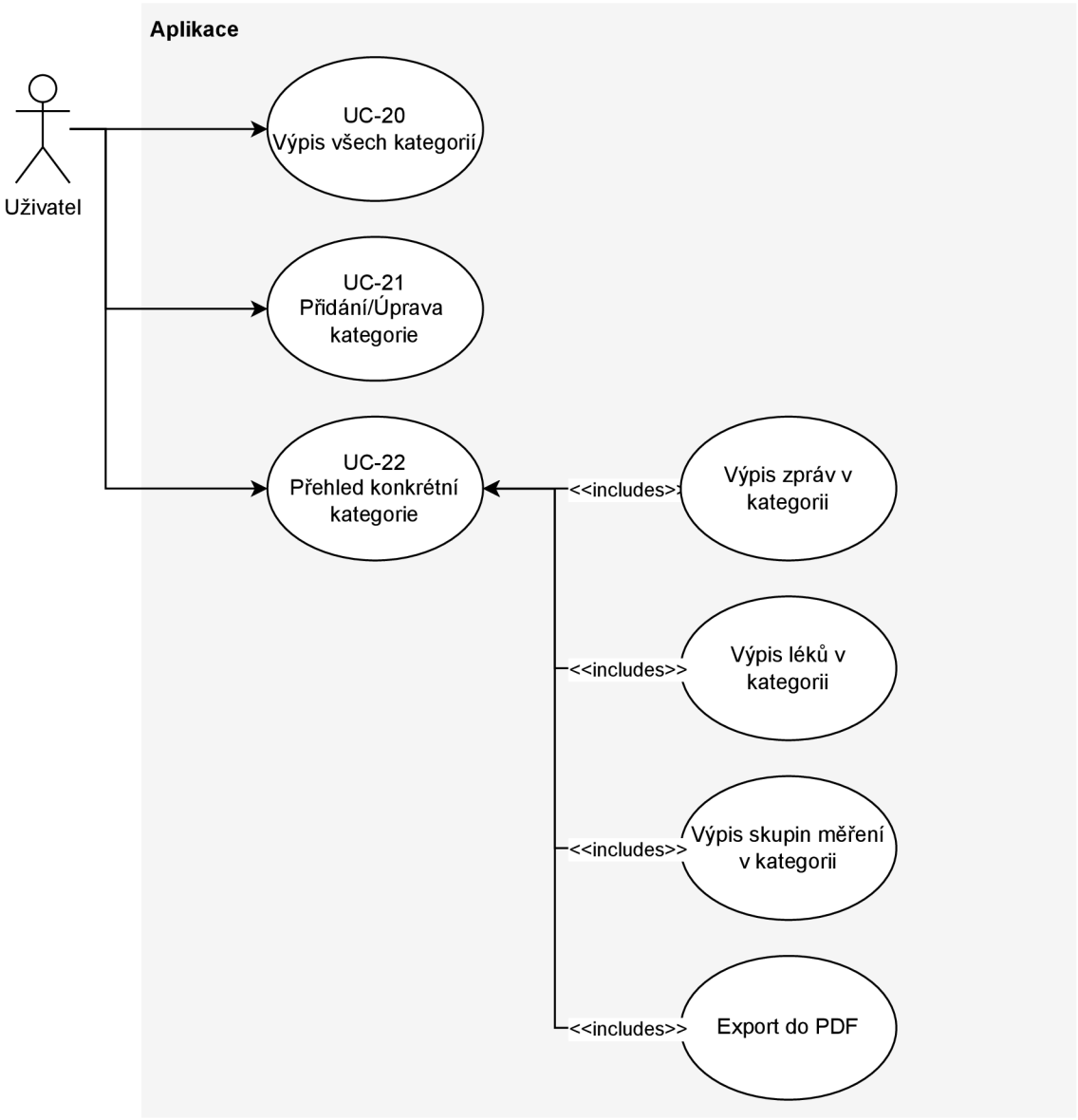
Obrázek A.4: Funkční požadavky modulu „Měření“



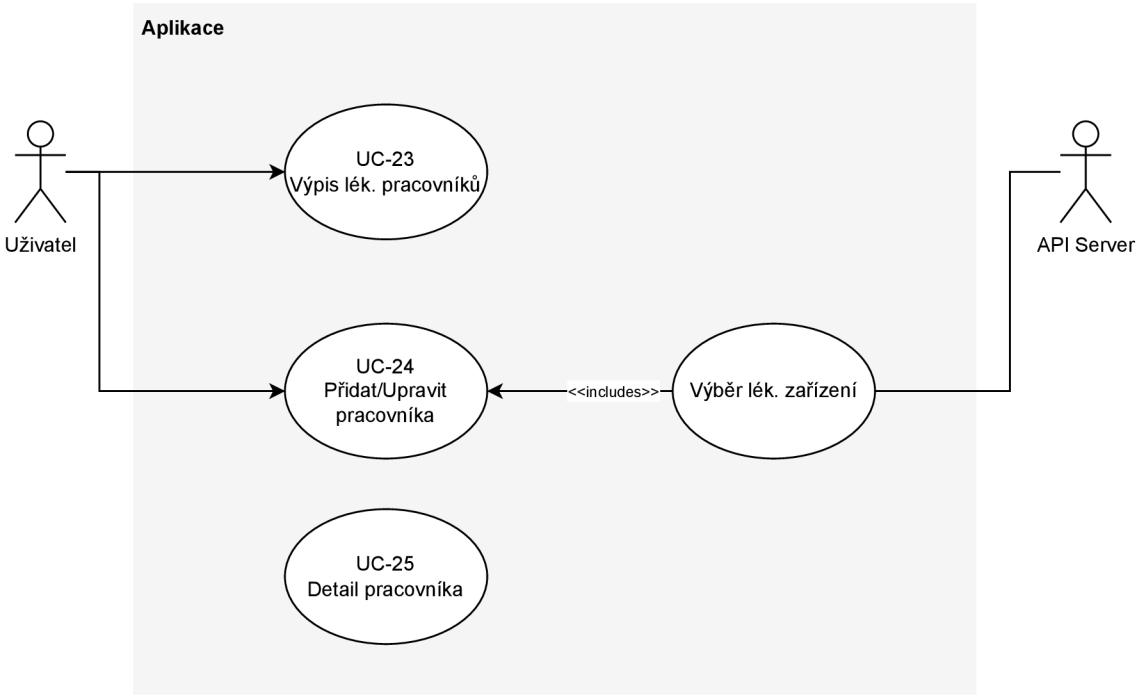
Obrázek A.5: Funkční požadavky modulu „Obecné - Průvodce po návštěvě zařízení“



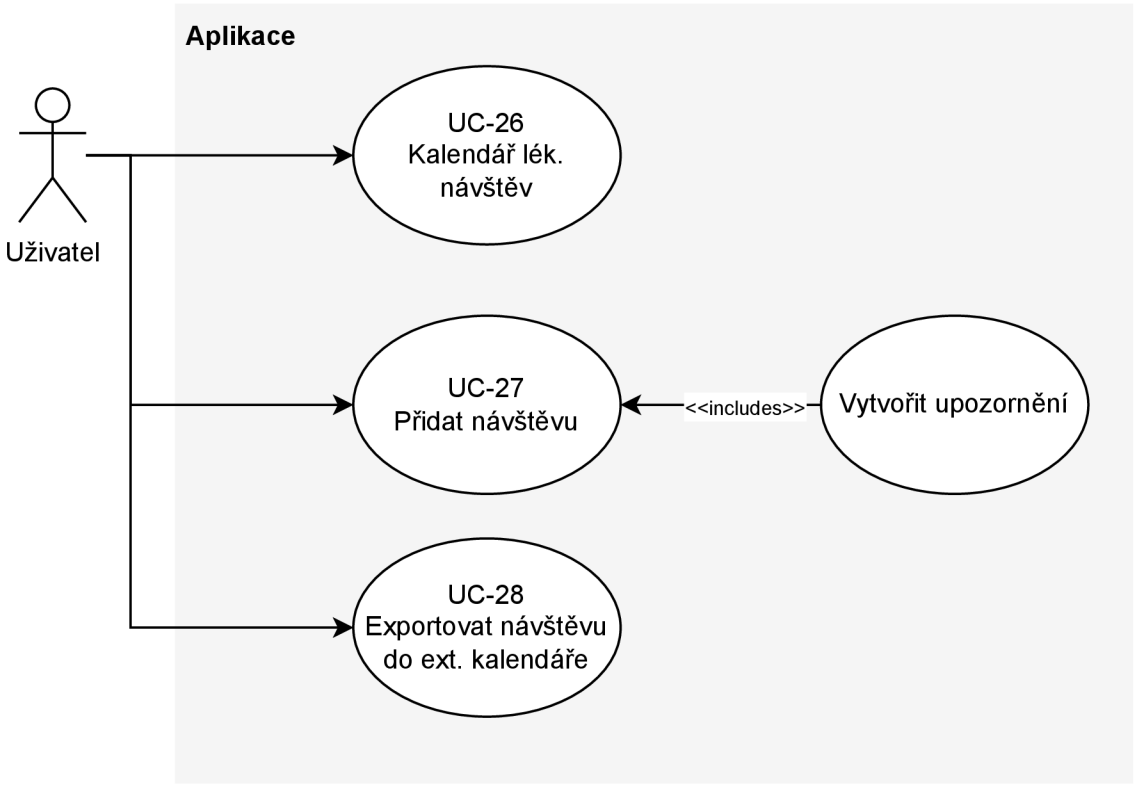
Obrázek A.6: Funkční požadavky modulu „Obecné - Správa pacientů“



Obrázek A.7: Funkční požadavky modulu „Obecné - Správa kategorií problémů“



Obrázek A.8: Funkční požadavky modulu „Obecné - Správa lékařských pracovníků“



Obrázek A.9: Funkční požadavky modulu „Obecné - Plány návštěv“