ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Mobilní aplikace pro sledování růstu, vývoje a zdravotních problémů dětí
Student:	Bc. Daniel Malý
Vedoucí:	Ing. Petr Špaček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem této práce je analyzovat, navrhnout a implementovat mobilní aplikaci pro operační systémy Android a iOS podle specifikace klienta. Aplikace bude umožňovat rodičům sledovat růst, zdraví a celkový vývoj jejich dětí. Musí podporovat automatickou synchronizaci uživatelských dat napříč všemi zařízeními jednoho uživatele, a také být schopna fungovat bez stabilního připojení k internetu. Musí také umožňovat sdílení dat mezi uživateli na dvou různých úrovních uživatelského oprávnění.

1) Analyzujte cílovou doménu.
2) Prozkoumejte možnosti řešení problému offline ukládání dat a synchronizace změn mezi různými uživateli.
3) Prozkoumejte a vyberte vhodné implementační technologie.
4) Navrhněte a implementujte řešení.
5) Otestujte a zdokumentujte řešení.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 7. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Mobilní aplikace pro sledování růstu, vývoje a zdravotních problémů dětí

Bc. Daniel Malý

Vedoucí práce: Ing. Petr Špaček, Ph.D.

9. ledna 2018

Poděkování

Chtěl bych poděkovat panu Lukáši Kovaříkovi za to, že mi zprostředkoval příležitost zhostit se tohoto úkolu a vyzkoušet na něm nové technologie. Dále děkuji vedoucímu práce, Ing. Petru Špačkovi, Ph.D., za cenné rady k obsahu a formě práce, za pomoc s administrativními úkony a za vše, co pro mě v průběhu celého mého studia udělal. V neposlední řadě chci poděkovat své rodině a blízkým za jejich stálou podporu a to i v dobách, kdy měli svých starostí více než dost.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 9. ledna 2018 .....

Odkaz na tuto práci

Malý, Daniel. *Mobilní aplikace pro sledování růstu, vývoje a zdravotních problémů dětí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá tvorbou mobilní aplikace pro systémy iOS a Android, která umožňuje rodičům zaznamenávat pokroky v růstu a vývoji jejich dětí, poskytuje jednoduchý způsob uchování jejich zdravotních informací a umožňuje sdílení těchto údajů s rodinnými příslušníky či s jinými osobami, kteří s dětmi přichází do styku. Mobilní aplikace je postavená na knihovně React Native, uchování dat na serveru a jejich synchronizace mezi uživateli jsou realizovány prostřednictvím databáze CouchDB. Práce popisuje proces analýzy, technického návrhu, implementace i testování systému a obsahuje praktická doporučení pro používání zvolených technologií.

Klíčová slova mobilní aplikace, děti, růst, zdraví, iOS, Android, React Native, CouchDB, synchronizace

Abstract

This work describes the development of a mobile application for the iOS and Android operating systems that lets parents track the growth of their children and save their important health information. It also allows sharing this data with other family members or persons which come into regular contact

with the children. The mobile application is built using the React Native framework and the cloud data storage and synchronization solution is realized using CouchDB. The work describes the process of analysis, technical design, implementation and testing of the system, and contains practical information regarding the use of the aforementioned technologies.

Keywords mobile application, children, growth, health, iOS, Android, React Native, CouchDB, synchronization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Motivace

„Deníček dítěte“ nebo také deníček miminka je papírový záznamník či fotoalbum, do kterého si rodiče mohou vkládat fotografie dětí a uchovávat důležité okamžiky z jejich vývoje, například kdy se jejich dítko poprvé usmálo, kdy mu vyrostl první zoubek, kdy vyřknulo první slovo a tak podobně. Je to velmi oblíbený nástroj pro uchovávání i ukazování cenných vzpomínek na první měsíce a roky života dítěte. Stejně oblíbené je i zaznamenávání růstu dětí, například pomocí čárek na rámu od dveří.

Tak jako mnoho dalších papírových předmětů v posledních deseti letech, i deníčky dítěte získaly svou digitální obdobu v podobě mobilních a webových aplikací zálohujících data v „cloudu“. Kvalitní aplikace cílená na české publikum a v českém jazyce však dosud chyběla.

Projekt aplikace BabyDiary

Na pozdim roku 2016 kontaktovala firma, které v této práci říkáme jen „klient“, softwarovou agenturu s poptávkou na vývoj právě takové mobilní aplikace, která měla sloužit jako digitální podoba deníčku dítěte pro české rodiče a rozšiřovat jeho možnosti o snadné sdílení dat mezi rodinnými příslušníky. Mimo zaznamenávání růstu a pokroků dětí měla aplikace umožňovat ukládání důležitých zdravotních informací o dětech, například alergie, podstoupená očkování, nebo prodělané nemoci. Všechny informace ukládané v aplikaci mělo být možné sdílet s lidmi, kteří s dětmi přijdou do styku, například s chůvami. Podmínkou bylo, aby aplikace fungovala jak na operačním systému iOS, tak na OS Android.

Zmíněná softwarová agentura se obrátila na mě, protože jsem již v minulosti vyvíjel mobilní aplikace pro obě platformy najednou a měl jsem zkušenosti s problémy, které při takovém vývoji vznikají. Byla to pro mě zároveň

příležitost vyzkoušet některé nové postupy a technologie, které vznikly za posledních několik let. Z toho vzešel námět na tuto práci. Funkční i nefunkční požadavky na aplikaci byly klientem předem dané, v technické realizaci jsem měl však volnou ruku. Samotný vývoj aplikace s pracovním názvem „BabyDiary“ probíhal přibližně šest měsíců čistého času, z nichž první dva byly věnovány zejména, výběru technologií a návrhu architektury systému.

Cíle a struktura práce

Tato práce dokumentuje tvorbu mobilní aplikace BabyDiary od analýzy požadavků až po testování výsledného produktu. První kapitola se věnuje požadavkům klienta a specifikuje, jak by měl cílový produkt vypadat. Druhá kapitola popisuje návrh a realizaci uživatelského rozhraní mobilní aplikace s přihlédnutím k rozdílům mezi oběma mobilními platformami. Třetí část se věnuje výběru hlavních technologií, pomocí kterých byla aplikace vytvořena, a představuje architekturu celého systému. Následuje detailní popis vybraných aspektů implementace aplikace a poslední kapitola se věnuje testovací strategii. Jako příloha jsou uvedeny snímky obrazovky z hotové aplikace, které ukazují rozložení uživatelského rozhraní.

Součástí práce je kompletní zdrojový kód obou částí systému (webového serveru a mobilní aplikace), který může posloužit jako inspirace pro další aplikace postavené na obdobných technologiích. Grafickou podobu díla, tedy barvy, písma, ikony a obrázky, měl na starosti klient, proto v této práci ukazujeme aplikaci bez některých z těchto prvků, popřípadě používáme volně dostupné náhrady.

Kapitola 1

Analýza požadavků

Tato kapitola analyzuje zadání mobilní aplikace a rozvádí ho do ucelené dokumentace pro návrhovou fázi. Popisuje cílovou doménu, uvádí obecné představy klienta na počátku projektu i jeho konkrétní funkční a nefunkční požadavky. Kromě požadavků klienta se do zadání promítají i prvky, které z nich logicky vychází a které by neměly být opominuty, aby byla výsledná aplikace použitelná a uživatelsky přívětivá.

Některé z požadavků na aplikaci nebyly známy hned od počátku, ale vyplynuly z vývoje a prvních iterací uživatelského testování. Mnohokrát se stalo, že musela být do návrhu uživatelského rozhraní přidána další obrazovka, aby navrhovaná funkce dávala smysl nebo aby pomohla použitelnosti aplikace. Vývoj probíhal zčásti iterativně, což byl i jeden z požadavků klienta. Formát této práce je však uzpůsoben spíše lineárnímu modelu vývoje, a tak zde nahlížíme na požadavky jako na ustálený celek.

Poslední část kapitoly se věnuje modelu cílové domény, který slouží jako předloha pro schéma databáze systému.

1.1 Doména a hlavní cíle

Cílová skupina uživatelů aplikace jsou rodiče dětí, jejich příbuzní a případně další osoby, které mají děti na starosti. Její nejobecnější účel je umožňovat uživatelům zaznamenávat a sdílet důležité okamžiky v životě dětí, sledovat jejich růst a pokroky a uchovávat informace o jejich zdravotním stavu. Jedná se o digitální obdobu deníčku dítěte, který jsme popisovali v úvodu práce. Aplikace rozšiřuje možnosti papírového záznamníku o zdravotní stránku a možnost jednoduchého sdílení mezi uživateli.

1.1.1 Klíčové zájmy

Klíčové zájmy aplikace můžeme rozdělit do následujících pěti oblastí. Ačkoli některé skutečné funkce budou pokrývat více než jednu oblast, tento výčet je

důležitý, neboť nám umožňuje mít na paměti, o co se vlastně aplikace snaží a čemu musí být jednotlivé funkční požadavky podřízeny. Jednotlivé oblasti pak detailněji popíšeme a rozvedeme na konkrétní funkční požadavky v následující sekci.

- Sledování vývoje dítěte
- Zdravotní karta dítěte
- Důležité kontakty
- Sdílení informací mezi uživateli
- Export dat z aplikace

1.2 Výčet případů užití a funkčních požadavků

V této sekci popíšeme každý okruh požadavků a doplníme ho o konkrétní případy užití. U každého případu užití uvedeme kód pro pozdější odkazování, slovní vysvětlení a případné doplňující funkční a nefunkční požadavky, pokud nejsou z vysvětlení ihned zřejmé.

Ve všech případech užití, kde uživatel do aplikace vkládá nějaká data, je implicitně zahrnuta i možnost tato data editovat a mazat, vyjma situací, kdy k tomu uživatel kvůli své úrovni přístupu nemá oprávnění. Výčet takových situací je uvedený v sekci 1.2.7 o uživatelských oprávněních.

1.2.1 Uživatelský účet

Tato skupina požadavků není uvedena jako klíčový zájem aplikace, nicméně je to samozřejmost, kterou uživatel od moderní softwarové služby očekává. Jedná se o možnost registrace a uchovávání uživatelských dat „v cloudu“, neboli na serverech poskytovatele služby, aby se uložená data mohla automaticky synchronizovat na každé další zařízení, na kterém se daný uživatel přihlásí.

UC1: Vytvoření uživatelského účtu Zadáním jména, příjmení, hesla a e-mailové adresy spolu s odsouhlasením podmínek použití se uživateli na serveru vytvoří účet. Po úspěšném provedení registrace se automaticky otevře hlavní část uživatelského rozhraní a už není potřeba se do aplikace přihlašovat.

UC2: Vytvoření dítěte při registraci Pojmem „vytvoření dítěte“ se zde rozumí založení profilu dítěte v aplikaci, aby k němu bylo možné přidávat záznamy. Ještě během registrace si bude moct uživatel rovnou zadat údaje o prvním dítěti (jméno, obrázek, datum narození, výška a váha), aby ho aplikace hned po registraci nemusela přesměrovávat na další formulář. Tento krok je nepovinný – uživatelé, kteří budou mít děti sdílené od někoho jiného (například chůvy), musí mít možnost se zaregistrovat pouze s vlastními údaji.

UC3: Přihlášení registrovaného uživatele Pokud už je uživatel registrovaný, ať už provedl registraci na aktuálně používaném zařízení nebo jinde, může se zadáním svého e-mailu a hesla dostat zpět ke svým vlastním datům.

UC4: Odhlášení přihlášeného uživatele V jedné instanci aplikace může být přihlášen pouze jeden uživatel a nesmí se stát, že úložiště na mobilním zařízení bude držet data od více než jednoho uživatele najednou. Aplikace musí tím pádem umožnit odhlášení uživatele a opětovné přihlášení někoho jiného.

UC5: Zažádání o obnovu hesla V případě ztráty hesla může uživatel projít běžný postup obnovy hesla spočívající v zadání své registrované e-mailové adresy a ověření své totožnosti pomocí otevření e-mailu zaslaného aplikací.

UC6: Obnova hesla Uživatel, který zažádal o obnovení hesla, obdrží na svou zaregistrovanou e-mailovou adresu ověřovací zprávu. Důležitý rozdíl oproti většině webových aplikací, které tento postup používají, je fakt, že aplikace BabyDiary nebude mít žádné webové rozhraní, na které by mohla v ověřovací zprávě odkázat. Tento problém jsem se rozhodl vyřešit tak, že bude v e-mailová zpráva obsahovat kód, který bude uživatel muset v aplikaci ručně zadat, popř. na svém mobilním zařízení kliknout na tzv. „deep link“. Ten otevře aplikaci na příslušné obrazovce a kód zadá automaticky. Odesláním tohoto kódu na server pak může aplikace ověřit totožnost uživatele a zpracovat změnu hesla.

Aby byl tento proces pro uživatele méně namáhavý, bude kód tvořit jen sedm číslic a zabezpečení proti útoku hrubou silou bude řešeno časovou prodlevou při zpracování požadavku na ověření. Server vždy počká alespoň pět vteřin, než na požadavek odešle odpověď. Spolu s omezením časové platností ověřovacího kódu na 10 minut je riziko úspěchu útoku hrubou silou zanedbatelné.

Bylo jedním z požadavků klienta, aby byla uživatelská registrace nutnou podmínkou k používání veškerých funkcí aplikace. Všechny následující případy užití jsou tedy podmíněny úspěšnou registrací nebo přihlášením do aplikace.

1.2.2 Profil dítěte

Děti jsou základní kámen celé cílové domény a všechna ostatní data ukládaná aplikací se váží právě na ně. Jejich správa je tedy hned po povinné uživatelské registraci nejdůležitější funkcí. Není nijak omezené, kolik dětí je možné jedním uživatelem spravovat. Počítáme však s tím, že pro běžného uživatele jejich počet nepřesáhne pět.

UC7: Vytvoření profilu dítěte Pokud si uživatel nezaloží profil dítěte při registraci (případ užití UC2), popřípadě má dětí víc, může profil založit i v hlavní části aplikace. Na obrazovce s formulářem vyplní jméno dítěte, datum narození a může mu přiřadit profilovou fotografii. Další údaje – aktuální výška a váha – jsou nepovinné.

UC8: Zobrazení profilu dítěte Všechny aktuální údaje o dítěti jsou dostupné na jedné obrazovce, kterou nazýváme profilem dítěte. Tento profil je rozdělen na dvě části, základní a zdravotní. Na základním profilu se nachází aktuální věk, výška a obvod hlavy dítěte, vypočítané podle nejnovějších záznamů s měřením (viz. případ užití UC14), pokud nějaké uživatel do aplikace vložil. Obsah zdravotního profilu je popsán v následující sekci. Profil dítěte lze i smazat, ovšem pouze tím uživatelem, který ho založil.

1.2.3 Zdravotní karta dítěte

Do aplikace si může rodič uložit důležité zdravotní informace o dítěti jako jsou prodělané dětské nemoci, alergie na potraviny či léky, krevní skupina, absolvovaná i nadcházející očkování a informace o zdravotním pojištění. Tyto informace mohou pomoci nejen při návštěvě lékaře, ale také například při vyplňování formulářů a přihlášek do kroužků nebo na dětské tábory. Všechny tyto zdravotní údaje jsou snadno přístupné z jediné obrazovky, „zdravotního profilu dítěte“.

UC9: Krevní skupina K dítěti je možné uložit a editovat údaj o jeho krevní skupině. Aplikace při zadávání dává na výběr z možností.

UC10: Alergie Každé dítě má v aplikaci seznam alergií, který může rodič upravovat. Alergie může mít tři různé stupně závažnosti: lehká, střední a těžká. Při výběru alergenu dává aplikace uživateli na výběr z několika přednastavených možností, lze ale zadat i úplně nový alergen. Ke každé alergii lze připsat poznámku, například přesně jakou reakci daný alergen u dítěte způsobuje.

UC11: Prodělané dětské nemoci Seznam prodělaných dětských nemocí je užitečný například v případě, že si opatrovník není jistý tím, jestli již dítě prodělalo plané neštovice nebo jinou nakažlivou dětskou nemoc. V aplikaci se jedná o jednoduchý seznam se jménem nemoci, měsícem prodělání a nepovinnou poznámkou.

UC12: Očkování Ve zdravotní kartě dítěte nesmí chybět ani seznam provedených a nadcházejících očkování. K očkování lze uložit název, datum, které může být v budoucnosti, a poznámku o průběhu. Pokud je datum očkování

v budoucnosti, uživatel si může nastavit, zda chce na událost upozornit lokální notifikací a pokud ano, kdy přesně se mu má notifikace zobrazit.

UC13: Zdravotní pojištění Do aplikace je možné uložit název zdravotní pojišťovny, u níž je dítě pojištěné. Jelikož plánujeme distribuovat aplikaci pouze v Česku, kde je seznam zdravotních pojišťoven omezený a nemění se příliš často, hodnotu tohoto údaje lze vybrat pouze z přednastaveného seznamu. Ten může správce aplikace na serveru upravit, pokud nějaká zdravotní pojišťovna zanikne, změní název nebo přibude nová.

1.2.4 Zaznamenávání pokroků a událostí

Aplikace BabyDiary umožňuje sledovat růstové a vývojové pokroky dětí. Konkrétně lze v aplikaci ukládat měření výšky, váhy a obvodu hlavy dítěte a zobrazovat graf vývoje těchto hodnot v čase. Dále lze zaznamenávat předem definované vývojové milníky a zobrazovat jejich historii. Je důležité, aby měly záznamy sentimentální hodnotu, proto je možné k záznamům přidávat textové poznámky a fotografie. Tato skupina požadavků naplňuje cíl aplikace poskytovat digitální podobu deníčku dítěte.

UC14: Zaznamenání měření výšky, váhy nebo obvodu hlavy Ke každému dítěti lze přidat jeden ze tří výše zmíněných druhů měření. Rodiče tak mohou od narození sledovat růst dítěte a případně ho porovnávat s jeho sourozenci či vrstevníky.

UC15: Zobrazení grafu s historií měření U každého druhu měření lze v aplikaci zobrazit historii jednotlivých záznamů a graf vývoje hodnot v čase. Graf umožňuje měnit časové rozmezí, které vykresluje, mezi jedním měsícem, jedním rokem a celou dobou od počátku měření.

UC16: Zaznamenání milníku Do aplikace lze uložit důležité vývojové milníky, které jsou v uživatelském rozhraní graficky odlišeny od jiných „běžných“ záznamů, spolu s datem, kdy bylo milníku dosaženo. Takovými událostmi jsou například první úsměv, první zoubek, první krůček nebo první slovo. Druh milníku lze vybrat z těchto čtyř předpřipravených možností, nebo si může uživatel vymyslet vlastní.

UC17: Uložení události Ne každá chvíle hodná uchování je nějaký životní milník. Do aplikace lze tudíž vložit vzpomínku na jakýkoli moment v podobě příspěvku sestávajícího z textové poznámky, data a jedné nebo více fotografií. Tento typ záznamu je v aplikaci a zbytku práce nazýván „událostí“, na diagramech v anglickém jazyce jako „post“.

UC18: Zobrazení záznamů na časové ose Aplikace umožňuje sbírat velké množství různých druhů záznamů o životě dítěte. Měření, milníky, události, fotografie, očkování, to vše může utvořit detailní přehled o prvních letech života. Jedním z centrálních prvků aplikace je proto časová osa, kde se zobrazují záznamy v chronologickém pořadí.

Záznamy jsou zobrazovány od nejnovějších po nejstarší, jako je tomu běžné například na sociálních sítích. Tento pohled je vhodný i pro ostatní členy rodiny, kteří sdílí přístup k dítěti (viz. sekce 1.2.7), aby mohli rychle zjistit, co je nového. Pro procházení vzpomínek od minulosti směrem k přítomnosti je ale možné na časové ose dorolovat až k nejstaršímu záznamu a vracet se směrem nahoru.

UC19: Zobrazení událostí v kalendáři Časová osa z předchozího požadavku, zvláště pokud uvážíme její logickou implementaci na mobilním zařízení jako prvek rolující odshora dolů, je vhodná pro prohlížení především událostí v minulosti. Aplikace ale umožňuje ukládání i nadcházejících událostí, jako například očkování nebo návštěv lékaře (viz. sekce 1.2.6). Pro jejich prohlížení je vhodnější prvek ve stylu kalendáře, kde může uživatel vidět celý měsíc dopředu a jsou na něm zvýrazněné dny, kdy se má něco stát.

Aplikace tedy kromě časové osy obsahuje přesně takový kalendář. Kromě perspektivy celého měsíce lze zobrazit detail konkrétního dne a všech událostí naplánovaných na daný den.

UC20: Filtrování záznamů Jak u kalendáře, tak u časové osy si může uživatel zapnout filtry, pomocí nichž si může zobrazit pouze záznamy určitého typu nebo záznamy vztahující se k určitému dítěti. Tyto filtry lze libovolně kombinovat.

1.2.5 Uchovávání důležitých kontaktů

Aplikace umí ukládat důležité kontakty týkající se péče o dítě, například telefonní čísla na chůvy, školky nebo adresy ordinací lékařů.

UC21: Uložení kontaktu Aplikace umožňuje uložit dva druhy kontaktu:

- *Obecný kontakt*, k němuž je možné uložit jméno, e-mail, telefonní číslo a/nebo fyzickou adresu.
- *Lékař*. Kromě stejných údajů jako u obecného kontaktu je u lékaře ještě možné evidovat specializaci a ordinační hodiny. Uložené lékaře lze přiřazovat k záznamům návštěvy lékaře (viz. sekce 1.2.6).

UC22: Využití uloženého kontaktu Má-li kontakt uložené telefonní číslo, lze ho z aplikace jednoduše vytočit. Stejně tak je možné po kliknutí na uloženou e-mailovou adresu kontaktu odeslat zprávu pomocí výchozí e-mailové aplikace.

UC23: Sdílení kontaktu Uložený kontakt lze sdílet mimo aplikaci BabyDiary pomocí standardního dialogu pro sdílení skrze různé komunikační aplikace.

Byla zvážena i možnost integrace s výchozím řešením pro ukládání kontaktů v mobilním operačním systému, byla však zamítnuta kvůli tomu, že by takovou funkci bylo obtížné správně navrhnout, implementovat i používat.

1.2.6 Sledování průběhu nemocí a teploty

Aplikace umožňuje sledovat průběh onemocnění a úrazů dětí, zaznamenávat měření tělesné teploty a ukládat informace o nadcházejících nebo proběhlých návštěvách lékaře. Na tuto skupinu funkcí lze pohlížet jako na nástroj pro „úzkostlivé“ rodiče; ti jsou jednou z cílových skupin uživatelů aplikace. Kromě toho ale můžou takové informace pomoct u lékaře, který tak získá přesnější představu o vývoji onemocnění a o zdravotních problémech, které mělo dítě v minulosti.

UC24: Záznam nemoci nebo úrazu Nemoc i úraz jsou z našeho pohledu úplně stejné a liší se pouze názvem. Kromě pojmenování a uložení poznámky spolu s počátečním datem je lze také ukončit, ale především jsou to jakési „schránky“ pro další záznamy jako měření teploty či návštěvy lékaře. Tyto další záznamy utvářejí detailní popis průběhu zdravotního problému, který si může uživatel zobrazit přehledně na jedné obrazovce. Součástí tohoto přehledu je i graf vývoje teploty.

UC25: Záznam příznaku Zdravotní příznak, jako například kašel, je z pohledu aplikace jednorázová událost a lze ho buď přiřadit nějaké nemoci či úrazu (a to i zpětně), nebo může stát sám o sobě. Kromě názvu příznaku lze přidat i poznámku nebo fotografie.

UC26: Měření teploty Jedná se pouze o záznam výsledku měření, nikoli měření samotné. Jelikož jsou změření teploty dítěte a uložení výsledku do aplikace pro uživatele dva různé úkony, je nutné, aby bylo ukládání co uživatelsky nejpřívětivější a nejjednodušší. Měření teploty lze přiřadit k nemoci nebo úrazu, a to i zpětně.

UC27: Návštěva lékaře a upozornění Návštěva lékaře je další typ záznamu v aplikaci. Je nutné ji propojit s konkrétním lékařem uloženým jako kontakt a je možné ji podobně jako záznam teploty přiřadit ke zdravotnímu problému. Dále si může uživatel nastavit, zda chce na návštěvu lékaře aplikací upozornit a pokud ano, kdy přesně se mu má notifikace zobrazit.

UC28: Záznam podávaných léků Důležitou funkcí aplikace BabyDiary je schopnost upozorňovat uživatele na to, že má dítěti podat nějaký lék. Na obrazovce zdravotního problému si proto může přidat lék užívaný dítětem a k němu pravidelné dávkování – jak velkou dávku v kolik hodin je potřeba podat. U každého léku si může zvolit, zda chce dostávat notifikace v časech podání. Názvy léků se ukládají mezi uživatelská data a při zadávání nových léků je aplikace schopna uživateli našeptávat již jednou použité hodnoty.

1.2.7 Sdílení informací

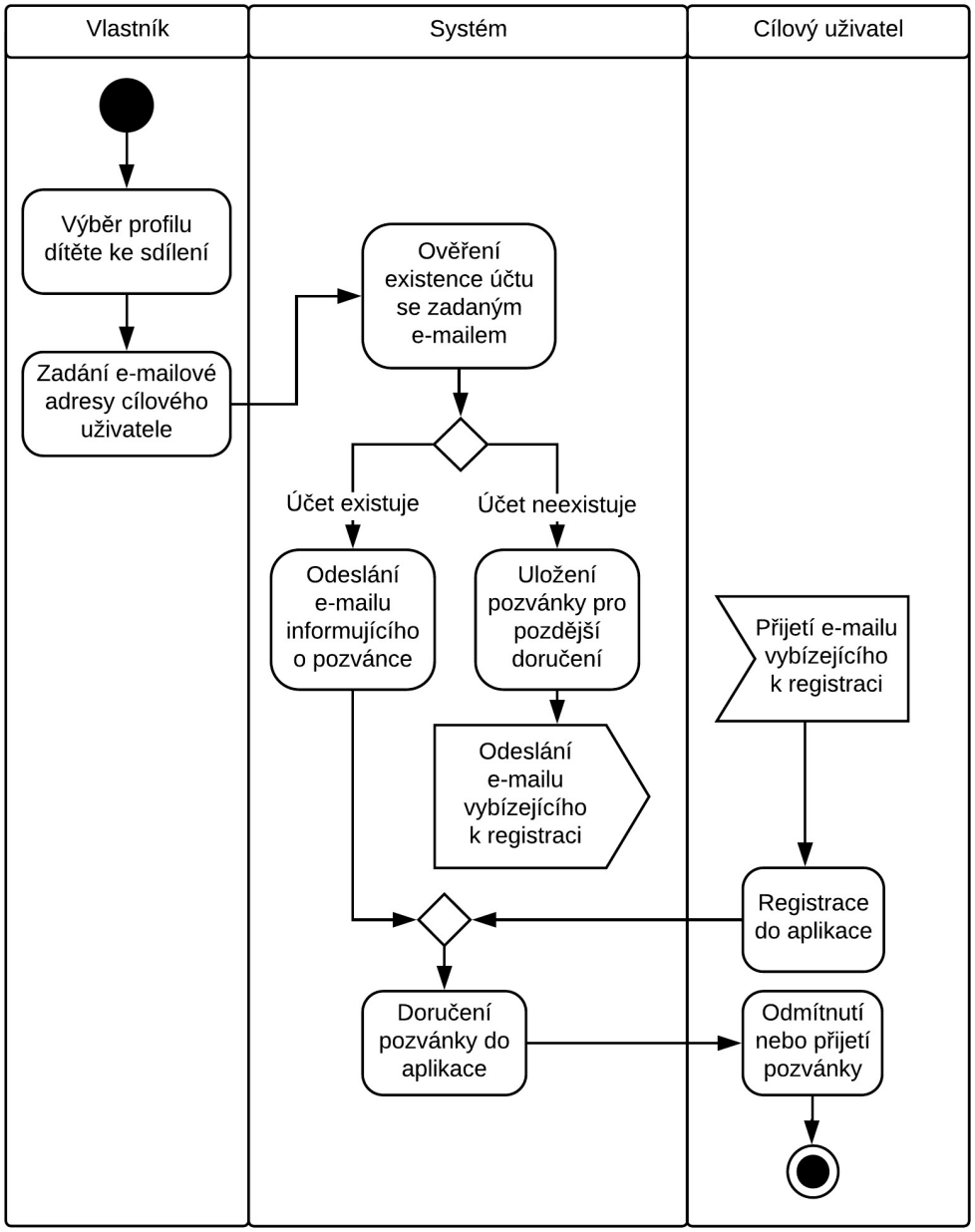
Důležité zdravotní informace, kontakty, události a další data o dětech je možné pomocí aplikace sdílet s jinými uživateli, především mezi rodiči a rodinnými příslušníky, ale také s chůvami či opatrovníky. Druhá jmenovaná skupina uživatelů by ale neměla mít možnost některé údaje měnit, pouze je mít k dispozici pro čtení. K tomuto účelu funguje v aplikaci systém pro udělení přístupu k datům dítěte na několika možných úrovních. Aby uživatelé nemuseli složitě nastavovat sdílení pro každý záznam nebo údaj zvlášť, sdílí se vždy celé dítě a všechna jeho data.

V aplikaci existují celkem tři úrovně přístupu k jednomu dítěti:

- *Vlastník*. Toto je uživatel, který založil profil dítěte v aplikaci. Má přístup ke všem datům a má povoleny všechny akce.
- *Rodič*. Ten má také přístup ke všem datům, nemůže však udílet přístup jiným uživatelům ani smazat dítě z aplikace. To může dělat jen vlastník.
- *Opatrovník*. Opatrovník nemůže měnit data na zdravotním profilu dítěte a nemůže editovat ani mazat záznamy, které vytvořil někdo jiný. Stejně jako rodič nemůže udílet přístup jiným uživatelům ani mazat profil dítěte.

Jeden uživatel může mít různý druh přístupu k různým dětem.

UC29: Odeslání pozvánky pro přístup k dítěti Vlastník dítěte zadá e-mailovou adresu uživatele, kterému chce přístup udělit, a požadovanou úroveň přístupu (rodič nebo opatrovník). V této fázi nezáleží na tom, zda je daná e-mailová adresa už v aplikaci zaregistrovaná, nebo ne. V obou případech přijde cílovému uživateli e-mail, který informuje o úmyslu sdílet přístup k dítěti.



Obrázek 1.1: Diagram aktivit procesu sdílení přístupu k profilu dítěte

Pokud je cílový uživatel již v aplikaci registrován, pozvánka se mu ihned zobrazí v aplikaci. Pokud není, e-mailová zpráva ho vybídne ke stažení aplikace a registraci. Jakmile se dotyčný registruje, pozvánka na něj již bude v aplikaci čekat. Celý proces je znázorněn na diagramu aktivit na obrázku 1.1.

UC30: Přijetí nebo odmítnutí pozvánky Přímo u zobrazené doručené pozvánky má cílový uživatel možnost pozvánku přijmout nebo odmítnout. V případě, že ji odmítne, pozvánka se smaže. V opačném případě aplikace stáhne příslušný profil dítěte do jeho mobilního zařízení a nastaví mu úroveň oprávnění zvolenou vlastníkem profilu.

UC31: Odebrání uděleného přístupu Vlastník dítěte může udělený přístup kdykoli odebrat. Pokud se to rozhodne udělat, aplikace ihned znepřístupní cílovému uživateli všechna nasdílená data a odstraní příslušný profil dítěte z jeho mobilního zařízení.

1.2.8 Export dat z aplikace

Oproti papírovému deníčku dítěte má mobilní aplikace tu nevýhodu, že na data v aplikaci si nelze „sáhnout“ a je možné o ně dokonce úplně přijít, pokud aplikace přestane být provozována. Pro zmírnění tohoto problému umožňuje naše aplikace exportovat všechny záznamy a informace vztahující se k jednomu dítěti do formátu PDF. Dokument si pak může rodič vytisknout a uložit v papírové podobě.

UC32: Export profilu a záznamů dítěte do formátu PDF Uživatel na profilu dítěte zvolí možnost exportu dat, zaškrtne, které druhy záznamů chce ve výsledném dokumentu mít, a zadá e-mailovou adresu, na kterou má být dokument zaslán. Požadavek je odeslán na server, který vygeneruje PDF dokument a pošle ho jako přílohu na zadanou adresu.

1.3 Nefunkční požadavky

Následuje krátký výčet požadavků na použitelnost, spolehlivost a bezpečnost aplikace. Od klienta pochází pouze první čtyři požadavky s velkým důrazem bezpečnost a ochranu citlivých údajů. Autorem tří posledních jsem já, neboť je mým přesvědčením, že by tyto požadavky měly být kladeny na *každou* mobilní aplikaci, která nějakým způsobem ukládá uživatelská data. Jejich splnění navíc představuje zajímavé technologické výzvy.

Dostupnost a lokalizace Mobilní aplikace bude dostupná v České republice a jen v českém jazyce.

Podpora platforem iOS a Android Aplikace musí podporovat oba nejrozšířenější mobilní operační systémy tak, aby aplikaci mohlo používat minimálně 90% všech uživatelů těchto platforem. V době stanovení požadavků (listopad 2016) se jednalo o tyto verze operačních systémů:

- iOS 9 a vyšší [1].

- Android 4.2 a vyšší [2].

Podporovaná zařízení Rozhraní aplikace nemusí být optimalizováno pro spouštění tabletech, pouze na mobilních telefonech.

Bezpečnost a ochrana citlivých údajů Jelikož si aplikace klade za cíl uchovávat data, která by většina lidí označila za citlivá a důvěrná, je logické, že je bude muset řádně zabezpečit. Nesmí být možné přes aplikaci získat přístup k datům, která nebyla do aplikace zadána dotyčnou osobou nebo je s ní nesdílí jejich vlastník. Aplikace nesmí pro účely trvalého přihlášení na mobilním zařízení ukládat uživatelské heslo.

Zálohování databáze Veškerá uživatelská data včetně fotografií je nutno zálohovat po dobu 30 dnů a ukládat zálohy do nějakého neveřejného cloudového řešení. Zálohování dat musí probíhat automaticky minimálně jednou denně.

Respektování UX konvencí platformy Návrh uživatelského musí respektovat zažité postupy a vzorce obou platforem. Rozložení prvků, navigace i animace musí působit přirozeně a plynule. Aplikace proto v prvé řadě nesmí vypadat úplně stejně na obou operačních systémech a nesmí si pomáhat „napodobeninami“ nativní navigace (viz. sekce 4.3.2).

Automatická a okamžitá synchronizace dat Jak již bylo řečeno v sekci 1.2.1, jeden z funkčních požadavků je, aby měl uživatel data k dispozici na jakémkoli mobilním zařízením přes přihlášení do svého uživatelského účtu. Zde navíc ale chceme, aby synchronizace probíhala automaticky bez jakéhokoli zásahu uživatele. Pokud je uživatel připojený k internetu, třeba i na více zařízeních najednou, musí mít vždy aktuální data, aniž by pro to musel cokoli dělat.

Vzhledem k tomu, že se data dají v aplikaci sdílet mezi uživateli, je tento požadavek rozšířen i na sdílená data. Pokud někdo jiný přidá nebo upraví data na profilu dítěte, který je s daným uživatelem sdílený, ten by měl úpravu ihned vidět ve své aplikaci. Jedinou výjimkou jsou nastavené notifikace. Ty si uživatel nastavuje pouze na jednom zařízení a s nikým jiným se nesdílí.

Použitelnost mimo připojení k internetu Aplikace musí být plně použitelná i mimo připojení k internetu. Výjimku tvoří pouze registrace, přihlášení, synchronizace dat a další procesy, které „přesahují“ zařízení, na kterém aplikace zrovna běží. Většina případů užití však musí být proveditelná offline.

Testovatelnost Jednotkové, neboli „unit“ testy by měly být samozřejmostí u jakéhokoli složitějšího kusu softwaru. Kód aplikace by měl obsahovat automatizované jednotkové testy na netriviální aplikační logiku. Kromě jednotkových testů musí existovat strategie i pro koncové testy, ať už automatické nebo manuální. Tyto testy by měly pokrývat všechny případy užití aplikace.

1.4 Doménový model

Jako poslední krok analýzy požadavků klienta sestavíme doménový model obsahující všechny druhy entit, které se v systému vyskytují, a vztahy mezi nimi. Do této chvíle jsme měli pouze roztroušené zmínky o typech dat v jednotlivých popisech případů užití, zde je konsolidujeme do přehledného schématu, které budeme využívat jak při technickém návrhu, tak při návrhu uživatelského rozhraní.

V této fázi se ještě nejedná o konkrétní datový model, nad kterým bychom přímo stavěli databázi, poskytne nám ale užitečný odrazový můstek. Nejedná se ani o vyčerpávající výčet všech datových položek, ten sestavíme až při návrhu databáze.

Data v aplikaci můžeme rozdělit do dvou skupin: administrativní a týkající se dítěte. Administrativní data jsou například informace o uživatelích aplikace, přístupy k profilům dětí a pozvánky. Vše ostatní jsou už data tvořící profil dítěte. Pro přehlednost rozdělíme náš doménový na dvě části odpovídající těmto dvou skupinám.

1.4.1 Administrativní data

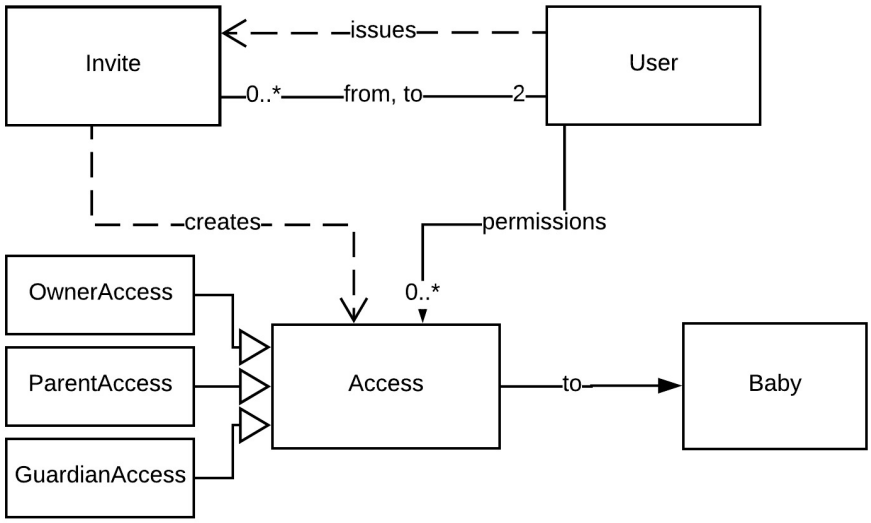
Diagram doménového modelu je na obrázku 1.2. Základní entitou je zde uživatel, který může mít různé druhy přístupů k profilu dítěte a může posílat či přijímat pozvánky, ze kterých vznikají další přístupy.

1.4.2 Data profilu dítěte

Diagram doménového modelu je na obrázku 1.3. Případ užití UC7 nám říká, že profil dítěte má mít dvě části – základní a zdravotní. Všechna další data jsou asociována s jedním z těchto profilů. Výjimku tvoří pouze kontakty, které stojí samy o sobě.

Entita „Entry“ je čistě abstraktní a představuje záznam, který má časový údaj, a tudíž se zobrazuje na časové ose a v kalendáři. Všechny záznamy jsou podtypem Entry kromě statických údajů na zdravotním profilu. Další důležitou entitou je „HealthProblem“, k níž jsou přiřazovány další události týkající se zdraví dítěte.

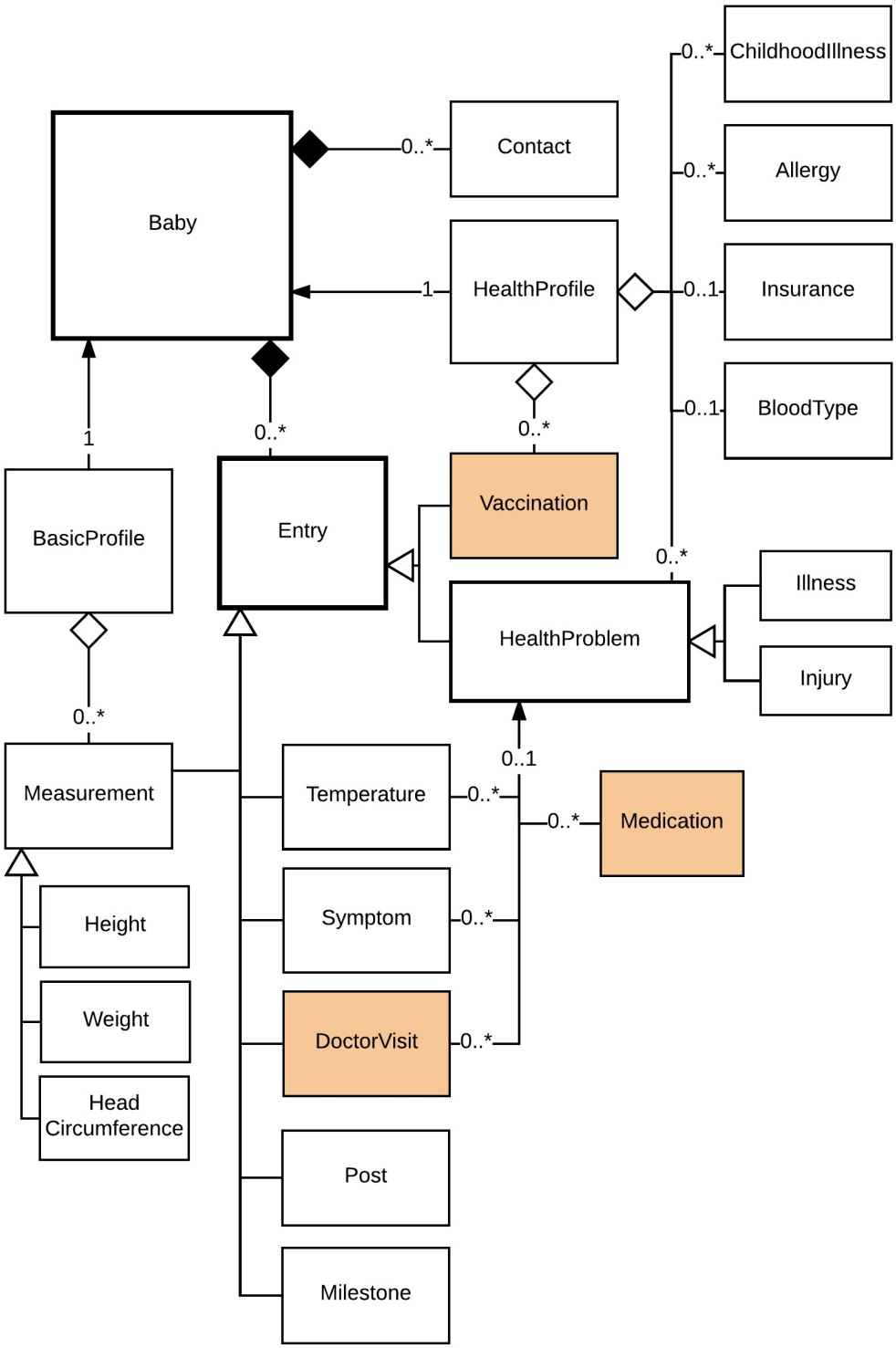
Oranžové podbarvení v diagramu označuje ty záznamy, na které si může uživatel nastavit notifikaci.



Obrázek 1.2: Doménový model: uživatelé a oprávnění

1.5 Shrnutí

V této kapitole jsme uvedli výčet klíčových zájmů spolu s funkčními a nefunkčními požadavky na aplikaci BabyDiary a sestavili přehledný model cílové domény. Následující kapitola se bude věnovat návrhu uživatelského rozhraní, které bude funkční požadavky realizovat.



Obrázek 1.3: Doménový model: děti a záznamy

Kapitola 2

Návrh uživatelského rozhraní

V této kapitole představíme návrh uživatelského rozhraní vyvíjené aplikace a popíšeme proces, kterým se k němu dospělo. Podobně jako u analýzy požadavků v předešlé kapitole a technického návrhu v kapitole následující budeme postupovat „shora dolů“, tedy od celkového pohledu až po jednotlivé detaily. Hotové rozhraní je možno vidět po spuštění aplikace, která je přiložena na USB disku. Příloha A pak obsahuje množství snímků obrazovek z mobilního zařízení, včetně těch, na které se v této kapitole odkazujeme.

Zatímco struktura rozhraní a filozofie za jeho návrhem je nutná k pochopení dalších implementačních rozhodnutí, nebudeme se zde nijak zabývat konkrétní grafickou podobou obrazovek, neboť ta nebyla samostatnou prací autora. Přiložená podoba aplikace žádné grafické prvky použité v produkční verzi neobsahuje, jak již bylo nastíněno v úvodu práce, snažil jsem se však hrubou podobu rozhraní zachovat, aby bylo možné výslednému produktu porozumět.

2.1 Požadavky na rozhraní

Jednou z nejpoužívanějších sad doporučení pro návrh interakce mezi člověkem a počítačovým programem je Nielsenova heuristická analýza z roku 1995 [3]. Ta definuje deset principů, kterých je potřeba se držet, aby byla interakce co nejpříjemnější. U mobilních aplikací není úplně možné splňovat všech deset pravidel do písmene – je například velmi obtížné poskytnout zkušenějším uživatelům rozhraní s více možnostmi a úplně nemožné implementovat klávesové zkratky. Přesto zde Nielsenova pravidla vypíšeme a uvedeme, jak chceme nebo proč nechceme docílit jejich splnění.

1. **Viditelnost stavu systému.** Pravidlo říká, že by měl systém vždy uživatele informovat o tom, co se děje. Ve většině situací je stav naší aplikace zcela řízen uživatelem a je dán tím, na jaké obrazovce se uživatel zrovna nachází, není třeba tedy na něj nijak zvlášť upozorňovat

kromě zobrazování nadpisu obrazovky. V několika málo případech, kdy je potřeba čekat na odpověď serveru, se uživateli zobrazuje načítací kolečko, které uživateli říká, že systém komunikuje se vzdáleným serverem.

Musíme ale mít také na paměti požadavek na automatickou a pro uživatele zcela skrytou synchronizaci dat na pozadí, který si do jisté míry protiřečí s touto heuristikou. Kompromisu docílíme tak, že budeme informovat uživatele vždy, když synchronizace přestane probíhat. To se stává většinou kvůli tomu, že se aplikace nemůže připojit k internetu. Zobrazíme v takovém případě notifikaci, že data, která uživatel vidí, nemusí být aktuální, a že jeho změny se projeví až po opětovném připojení k síti. Notifikace je ukázána na snímku obrazovky A.9.

2. **Propojení systému a reálného světa.** První výklad tohoto požadavku je takový, že by měl systém používat jazykové a grafické prvky, kterým uživatel rozumí. Aplikace sice má vlastní jazyk se slovy jako „záznam“, „profil“ a „milník“, jsou to ale všechno výrazy, kterým by měl cílový uživatel ihned porozumět. Žádné čistě systémové výrazy se v aplikaci neobjevují a grafickými prvky se zde nezabýváme.

   Dalším aspektem této heuristiky je, že by systém měl zobrazovat informace v přirozeném a logickém uspořádání. Zobrazení informací na časové ose a kalendáři zjevně tyto požadavky splňuje, posouzení tohoto kritéria u dalších obrazovek je ponecháno na čtenáři po přečtení zbytku kapitoly a prohlédnutí všech snímků obrazovky.

3. **Uživatelská kontrola a svoboda.** Pravidlo spočívá v tom, že každá uživatelská akce by měla jít jednoduše vrátit a měla by být minimalizována uživatelova zodpovědnost. Jako příklad se zde často uvádí tlačítko „zpět“ pro vrácení posledních úkonů. V naší aplikaci je toto pravidlo splněno možností libovolně editovat nebo mazat záznamy a minimalizací nevratných akcí. Jediný doopravdy destruktivní úkon je smazání záznamu nebo profilu dítěte. Pokud se uživatel chystá něco takto nevratně smazat, musí svůj úmysl ještě potvrdit ve vyskakovacím dialogu.

   Manipulace s přístupovými právy taktéž není nevratná. Pozvánky a přístupy lze nejen udílet, ale i libovolně rušit a znovu zakládat – zrušení přístupu nemá žádný vliv na data, jež kterýkoli uživatel k profilu dítěte už vložil.

4. **Jednoduché rozpoznání, pochopení a opravení chyb** Chybové hlášky by měly být v jazyce uživatele, srozumitelné a konstruktivní. V dnešní době je už těžké najít nějaký kus softwaru cílený na netechnické uživatele, který by tento princip porušoval, a aplikace BabyDiary jím také není.

5. **Prevence chyb.** Nielsen říká, že ještě lepší než mít v systému dobré chybové hlášky je, když chyby vůbec nemohou vzniknout. Systém by

uživateli měl dát možnost opravit stav, který by k chybě vedl, dříve, než nastane. To docílíme důslednou validací všech formulářů, aby uživatel nikdy nemohl dostat svá data do stavu, ze kterého by mohly vzejít nějaké chyby na úrovni databáze. Za všechny ostatní chyby může buď síť, nebo vývojář a těm bohužel uživatel předejít nemůže.

6. **Rozpoznání namísto vzpomínání.** Zde se jedná o to, že bychom měli minimalizovat nároky na uživatelovu paměť tak, aby byly všechny volby hned dostupné. Zároveň by uživatel neměl být nucen si pamatovat informace z jedné obrazovky po přechodu na jinou. Dostupnost většiny možností přímo na obrazovce je řešena spodní lištou záložek a globálně dostupným tlačítkem pro přidání záznamu (více v sekci 2.2.1). Samotné přidávání záznamů je vždy proces na jedinou obrazovku, a tak nemůže nastat situace, že by uživatel potřeboval více kroků pro provedení jedné akce.

   Prohlížení profilů dětí už je nutné realizovat hierarchickým uspořádáním obrazovek. Proto zde alespoň vždy v horní části obrazovky zůstane uvedené jméno dítěte, jehož profil si zrovna prohlížíme, jako například na snímku obrazovky A.10.

7. **Estetický a minimalistický design.** Rozhraní by nemělo obsahovat prvky ani informace, které uživatel nepotřebuje. Splnit toto pravidlo je o poznání jednodušší u mobilních aplikací, kde je prostoru na obrazovce mnohem méně, než u desktopových programů či webových stránek. I tak si ale můžeme dát za cíl, aby žádná obrazovka nebyla příliš „dlouhá“. Celkovou estetiku rozhraní lze vidět na všech snímcích obrazovky.

8. **Flexibilní a efektivní použití.** Toto pravidlo říká, že by systém měl nabízet jednodušší rozhraní pro začátečníky a pokročilejší rozhraní pro zkušené uživatele. Typický příklad je dostupnost klávesových zkratek. Kvůli omezením, které klade forma mobilní aplikace – malá obrazovka, absence klávesnice – toto pravidlo budeme muset porušit. Není v našich možnostech vyvíjet dvě různé verze rozhraní a není praktické schovávat některé možnosti, aby nemátly začátečníky. Místo toho se budeme snažit rozhraní vyvinout tak, aby bylo jednoduše použitelné pro všechny a umožňovalo nic víc a nic méně než to, co umožňovat má.

9. **Dostupnost nápověd a návodů.** Toto je další pravidlo, které zčásti porušíme. Obsáhlá uživatelská příručka k mobilní aplikaci, která není nijak složitá a ani se nevěnuje vysoce specializované doméně, není dle názoru autora nezbytná a neušetřila by uživatelům žádný čas.

   Na druhou stranu tam, kde by nemuselo být na první pohled jasné, co přesně aplikace myslí tím či oním pojmem, musí být přímo v dané

sekci uživatelského rozhraní uvedeno srozumitelné vysvětlení. Konkrétním příkladem je vysvětlení jednotlivých uživatelských rolí u procesu posílání pozvánky, ukázáno na snímku obrazovky A.17

10. **Standardizace a konzistence.** Stejná slova a grafické prvky by měly v celém systému odkazovat na tu samou věc a každý koncept by měl mít pouze jeden název či grafickou reprezentaci. Toho docílíme tak, že budeme důsledně dodržovat názvosloví z analýzy požadavků a doménového modelu a standardizujeme vzhled stejně se chovajících prvků. Například všechny formuláře se budou potvrzovat tlačítkem, které se bude nacházet vždy na tom samém místě a bude mít stejný popisek.

    Druhá část této heuristiky se zabývá dodržením konvencí platformy, na které uživatelské rozhraní běží. Je v zájmu tvůrce rozhraní, aby se toho musel uživatel co nejméně učit, a tak by se mělo chovat vše tak, jak to má uživatel naučené z jiných aplikací na dané platformě. Respektování konvencí platformy máme i jako jeden z nefunkčních požadavků v předchozí kapitole, tento bod je pro nás tedy obzvláště důležitý a stojí za to se na něj podívat blíže.

2.1.1 Multiplatformní vývoj a konzistence

Pro většinu projektů mobilních aplikací je dodržování zavedených vzorců jednoduché, neboť vývojová platforma nabízí standardizované UI komponenty a řešení pro navigaci. Vlastník platformy, tedy společnost Google v případě Androidu a Apple v případě iOS, tímto zároveň zjednodušuje vývoj a zajišťuje jistou kvalitu aplikací, protože „cesta nejmenšího odporu“ pro vývojáře je zároveň tou správnou cestou.

Naše rozhodnutí, že budeme aplikaci BabyDiary vyvíjet některou z multiplatformních technologií, vysvětlené a zdůvodněné v sekci 3.1), nás staví do zcela jiné pozice. Tyto technologie totiž nativní vývojovou platformu abstrahují a nabízí k ní buď obalující rozhraní, nebo dokonce úplně vlastní alternativy. Uživatelské rozhraní pak často vypadá jinak a někdy tento nesoulad může bít do očí a kazit dojem z celého produktu [4].

Musíme tedy akceptovat, že cenou vývoje pro oba operační systémy najednou je nutnost vynaložit zvláštní úsilí, aby uživatel neměl z aplikace pocit, že je s rozhraním něco v nepořádku. Jedná se o nalezení vhodného kompromisu mezi uniformitou zdrojového kódu a respektování platformy.

Tento kompromis se podařilo najít a implementovat především díky využití knihovny React Native Navigation (viz. sekce 4.3.2). Přestože práce s ní byla daleko náročnější než využití náhražek obsažených přímo v základní verzi React Native, podařilo se při uživatelském testování docílit toho, že žádný z testujících uživatelů nezapochyboval o tom, že byla aplikace vyvíjena čistě nativně.

2.2 Uspořádání rozhraní

V této sekci se podíváme na různé možnosti celkového uspořádání navigace v mobilním rozhraní a vybereme to nejvhodnější pro náš případ.

2.2.1 Navigace

Obecně existují tři typy rozhraní v mobilních aplikacích: hierarchické, ploché a obsahově řízené [5]:

- **Hierarchické** uspořádání se hodí tam, kde lze funkce aplikace jednoduše seskupit do stromové struktury podprocesů, ve které nedává smysl přecházet z jedné větve do jiné, aniž by se uživatel vrátil o úroveň výš. Nejlepším příkladem je aplikace pro nastavení telefonu, v níž se uživatel postupně zanořuje do podskupin nastavení.

- **Obsahově řízené** uspořádání dává smysl tehdy, když je z podstaty aplikace zřejmé, jak by se měl uživatel v rozhraní pohybovat. Navigace nemusí být nijak zvlášť strukturovaná, důležité je, aby byla podřízená obsahu. Typickým příkladem takové navigační struktury jsou hry. Uspořádání navigace je zde zcela závislé na tom, o jaký druh hry se jedná.

- Třetí uspořádání a to nejvhodnější pro naši aplikaci, je **ploché**. Tu využívá naprostá většina mobilních aplikací, protože je nejflexibilnější. Rozhraní je rozdělené do několika málo hlavních oblastí, mezi kterými jde jednoduše přepínat, ať už se uživatel nachází na kdekoliv. Každá oblast má vlastní navigační strukturu i stav, takže je možné například přepnout jinam, a pak se vrátit k rozpracovanému formuláři v jiné sekci.

Pro aplikaci BabyDiary jsme zvolili ploché uspořádání, především protože hierarchické ani obsahově řízené v našem případě nedává příliš smysl. Je pro nás také důležité, aby uživatel nebyl „uvězněn“ procesem, kterým právě prochází. Jelikož velká část interakce s aplikací se odehrává prostřednictvím formulářů pro přidávání záznamů, je důležité, aby si mohl uživatel přepnout a prohlížet existující data, aniž by o stav vyplnění formuláře přišel.

2.2.2 Sekce rozhraní

Abychom se mohli pustit do ukazování návrhu jednotlivých obrazovek, zbývá nám rozdělit naše ploché rozhraní do sekcí podle obsahu a navrhnout, jak se mezi nimi bude přepínat. S logickým rozdělením nám pomůže sekce 1.1.1 o klíčových zájmech aplikace a rozpis případů užití.

V prvé řadě bychom měli mít jednu celou sekci věnovanou zobrazování událostí na časové ose. Ta by měla být centrálním bodem aplikace, neboť na ní uživatel rychle najde nové a podstatné informace. Vlastní sekci by měl mít

i kalendář událostí, aby se na něj uživatel mohl rychle dostat, když v něm potřebuje něco najít. Další sekce by měla umožňovat prohlížení a správu profilů dětí. Do ní můžeme zakomponovat i funkce odesílání pozvánek a správy přístupu k profilům. Export dat do PDF můžeme také přidružit k profilu dítěte.

Logickým pokračováním těchto úvah by mohla být i sekce se zdravotními informacemi a zdravotními záznamy dětí. Je to přece jen jeden z klíčových záměrů aplikace a i velká část doménového modelu se točí kolem zdravotních údajů. Tento nápad se ale klientovi nelíbil, protože aplikaci nechtěl profilovat primárně jako nástroj pro sledování zdravotních problémů. Z marketingového hlediska je to pochopitelné. Mnohem důležitější pro klienta byla funkce deníčku a sledování pokroků.

Čtvrtou a tedy i poslední sekcí bude seznam a správa kontaktů. Na diagramu doménového modelu lze vidět, že kontakty stojí poněkud mimo všechny ostatní entity, a tak by bylo těžké jim najít místo někde jinde než ve vlastní sekci. Navíc chceme, aby se uživatel mohl ke kontaktům dostat rychle – pokud možno jedním kliknutím po spuštění aplikace.

Co se týče implementace přepínání mezi sekcemi, máme na výběr ze dvou standardních řešení. Tím prvním je tzv. šuplík, neboli menu schované za hlavním rozhraním, které lze zobrazit buď gestem, nebo kliknutím na ikonku. Druhou možností je lišta záložek, neboli „tab bar“, která je vždy přítomna v horní nebo dolní části rozhraní. Do šuplíku se vleze více položek, ale je potřeba ho gestem otevírat, což je o jeden úkon více než u lišty. Jelikož naše aplikace má sekce pouze čtyři, dává pro nás mnohem větší smysl lišta, zachycená na většině snímků obrazovek. U iOS verze aplikace je kromě záložek na liště také velké tlačítko „+“. Tímto tlačítkem lze odkudkoli v aplikaci spustit její centrální a nejpoužívanější funkci, a sice přidání nového záznamu.

Čtenář, který někdy používal oba mobilní operační systémy, ví, že na OS Android bývají obvykle záložky nahoře, zatímco na iOS jsou vždy přilepené na spodní okraj obrazovky. Po prostudování UX doporučení pro Android [6], které záložky ve spodním části obrazovky již povolují, a s přihlédnutím k tomu, že spodní část obrazovky je v dnešní době stále větších telefonů mnohem lépe dostupná palci, jsem se rozhodl pro nekonvenční umístění záložek dolů i ve verzi aplikace pro Android. Jeden rozdíl oproti iOS verzi zde však přece je. Velké centrální tlačítko na Androidu na liště není. Místo toho stejnou roli na obrazovce časové osy i kalendáře plní plovoucí tlačítko, neboli FAB („Floating Action Button“), což je jeden z nejpoužívanějších návrhových vzorů pro UX na celém systému. Vzhled lišty na Androidu i s plovoucím tlačítkem je na snímku obrazovky A.5.

2.3 Navigace a hlavní obrazovky

V této sekci popíšeme některé druhy obrazovek v aplikaci, jejich rozdělení do záložek a strukturu navigace. K tomu nám poslouží diagramy navigace na

obrázcích 2.1, 2.2 a 2.3. Hranaté obdélníky na diagramech představují obrazovky, obdélníky se zaoblenými rohy jsou důležité dílčí komponenty. Ztrojený obdélník indikuje, že daná obrazovka nebo komponenta ukazuje nějaký seznam stejných položek. Šipky indikují obvyklé směry navigace, na diagramech ale nezobrazujeme kroky zpět, které je jako u většiny dalších mobilních aplikací možné udělat kdykoliv. Šipky, které by na UML diagramu tříd znamenaly dědičnost, zde představují podobnou ideu, a sice určitý podtyp obrazovky.

2.3.1 Autentizace

Na diagramu 2.1 je ukázána struktura procesu autentizace uživatele. Tato část rozhraní není ploše uspořádaná, nýbrž je organizována jako proces se začátkem a koncem. Po spuštění aplikace, pokud se již uživatel jednou přihlásil a zůstal přihlášen, je celý proces přeskočen a zobrazí se mu rovnou hlavní rozhraní. Zbytek autentizace se příliš neliší od jiných mobilních či webových aplikací, které čtenář zná, snad s výjimkou procesu obnovy hesla. Zde bylo potřeba vyřešit problém, že systém nemá žádnou webovou aplikaci, na kterou by bylo možné v ověřovacím e-mailu odkázat. Řešení tohoto problému jsme popsali v kapitole o analýze požadavků (případ užití UC6).

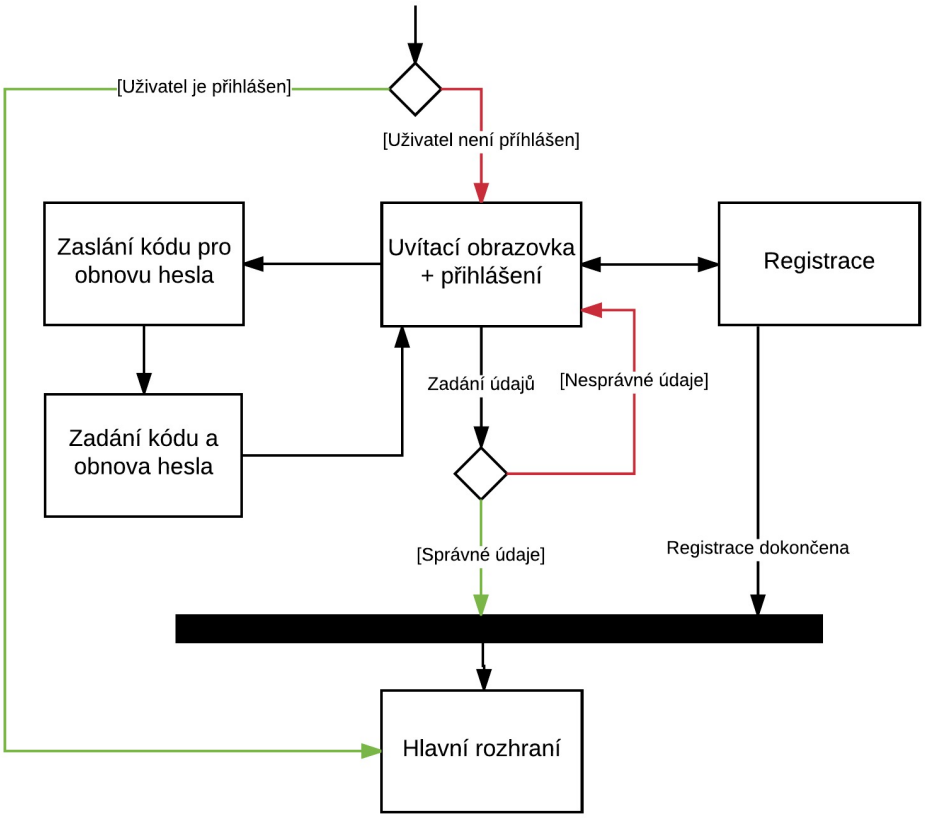
2.3.2 Časová osa a kalendář

Struktura rozhraní tří ze čtyř záložek v hlavním menu aplikace – časové osy, kalendáře a kontaktů – je ukázána na obrázku 2.2. Na rozdíl od autentizace je zde rozhraní uspořádáno jako různé způsoby pohledu na data a přecházení mezi nimi. Záložky na hlavní liště jsou v horní části diagramu. Každá má vlastní stav navigace a lze mezi nimi libovolně přepínat, pokud zrovna rozhraní nepřekrývá nějaká modální obrazovka, která má na diagramu bledě modré pozadí.

Časová osa i kalendář zobrazují úplně stejná data, jen různými způsoby, a z pohledu navigace plní stejnou funkci. Časová osa je zobrazena na snímku obrazovky A.5 a kalendář na A.6. Obě obrazovky mají v levém horním rohu tlačítko pro nastavení filtrů podle dětí nebo podle typu záznamů.

Záznamy na časové ose jsou seskupeny do nadepsaných segmentů podle dní, pokud jsou maximálně týden v minulosti nebo v budoucnosti, a podle delších intervalů, pokud jsou starší, resp. novější. Události, které jsou v budoucnosti, se zobrazují pouze v případě, že se uživatel pokusí rolovat směrem nahoru. Časová osa také využívá systém progresivního načítání, kde je zpočátku načteno jen několik prvních záznamů, aby na inicializaci aplikace nemusel uživatel příliš dlouho čekat. Pouze v případě, že osa dojede až na konec, jsou načteny a zobrazeny další záznamy.

V produkční verzi aplikace jsou různé druhy záznamů na časové ose i v kalendáři graficky odlišeny ikonami.

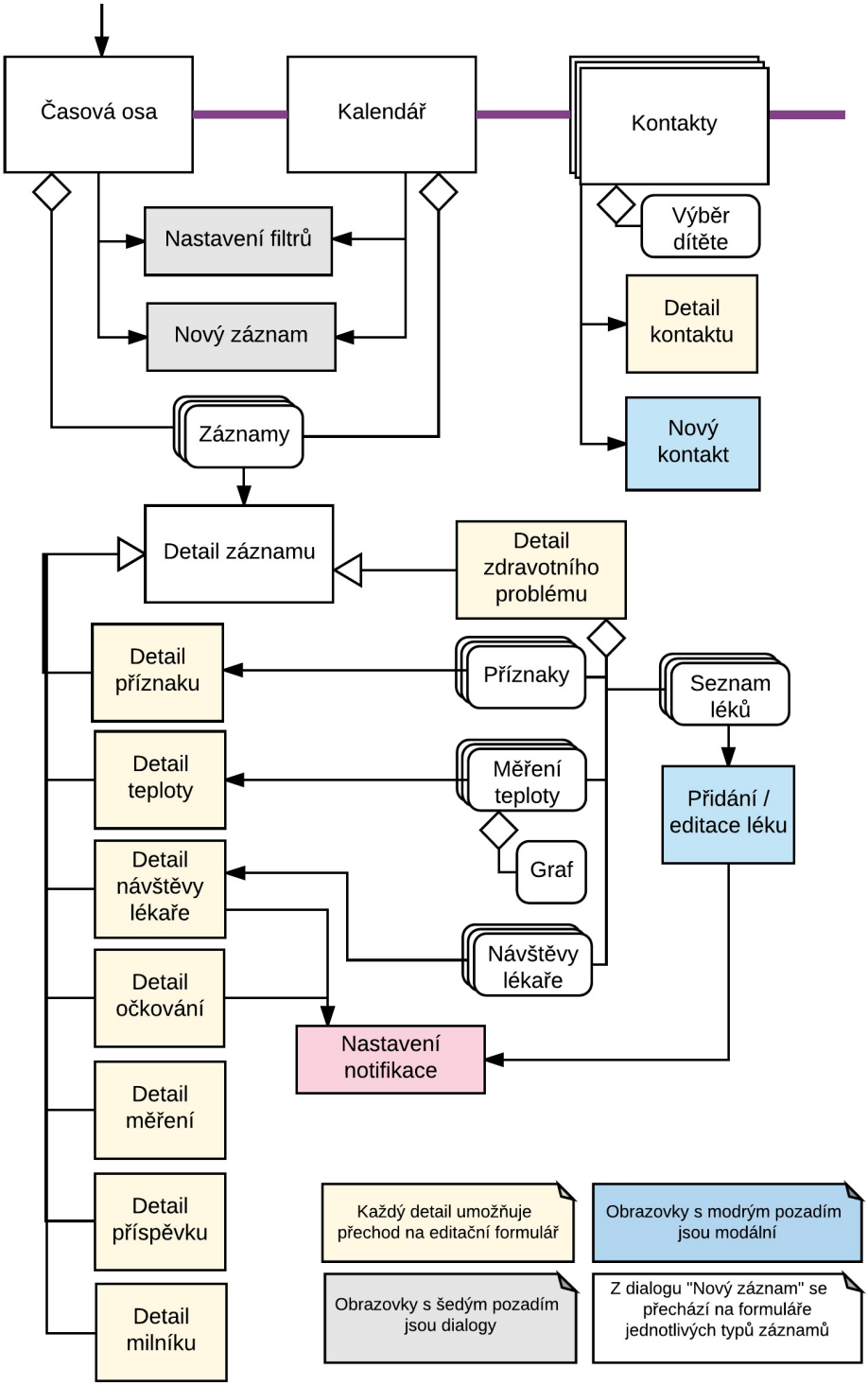


Obrázek 2.1: Schéma navigace – autentizace

2.3.3 Zdravotní problém

Obrazovka zdravotního problému, kterou lze vidět na snímcích obrazovky A.11 a A.12, seskupuje všechny informace a události týkající se jediné nemoci či úrazu dítěte. Najdeme zde seznam podávaných léků spolu s možností nastavit notifikace na jejich podání, graf naměřených teplot, seznam přiřazených příznaků a seznam návštěv lékaře. Z této obrazovky lze tyto dílčí záznamy také rovnou přidávat, aniž by je uživatel musel k danému zdravotnímu problému ručně přiřazovat.

Na snímku obrazovky A.13 je ukázáno přidávání nového zdravotního problému. Kromě nemoci a úrazu je zde na výběr ještě třetí typ, a sice příznak. Přestože jsou příznak a zdravotní problém odlišné entity na doménovém modelu, pro účely jejich přidávání jsou seskupeny dohromady. Pro uživatele by totiž mohlo být matoucí, že tlačítko pro přidání „zdravotního problému“ je odlišné od tlačítka pro přidání příznaku.



Obrázek 2.2: Schéma navigace – časová osa, kalendář a kontakty

2.3.4 Kontakty

Kontakty, jejichž struktura je také na diagramu 2.2, stojí odděleně od celého zbytku aplikace, vyjma jediné interakce – pokud uživatel přidává návštěvu lékaře v jiné části aplikace, může jako mezikrok na lékaře vytvořit nový kontakt. Seznam kontaktů je možné vidět na snímku obrazovky A.8.

2.3.5 Profily

Na záložce profilů je o něco více funkcí než na ostatních, proto ji máme na samostatném diagramu (obrázek 2.3). Z profilů se ale lze dostat i na některé obrazovky z předchozího diagramu, u těch jsou další směry navigace totožné a už je podruhé nezobrazujeme.

Základní obrazovka záložky, která je na snímku A.7, obsahuje informace o uživateli, možnost editace uživatelských informací, tlačítko pro odhlášení a seznam dětí, ke kterým má uživatel přístup. V tomto seznamu lze po kliknutí na tlačítko s ikonou rodičů spravovat přístupová práva. Ze seznamu se pak lze dostat na jednotlivé obrazovky dětí.

Profil dítěte je rozdělen na dvě části, základní (snímek obrazovky A.9) a zdravotní (snímek obrazovky A.10), mezi kterými jde přepínat. Každý druh údajů pak má vlastní obrazovku, například historie měření, seznam očkování nebo seznam alergií.

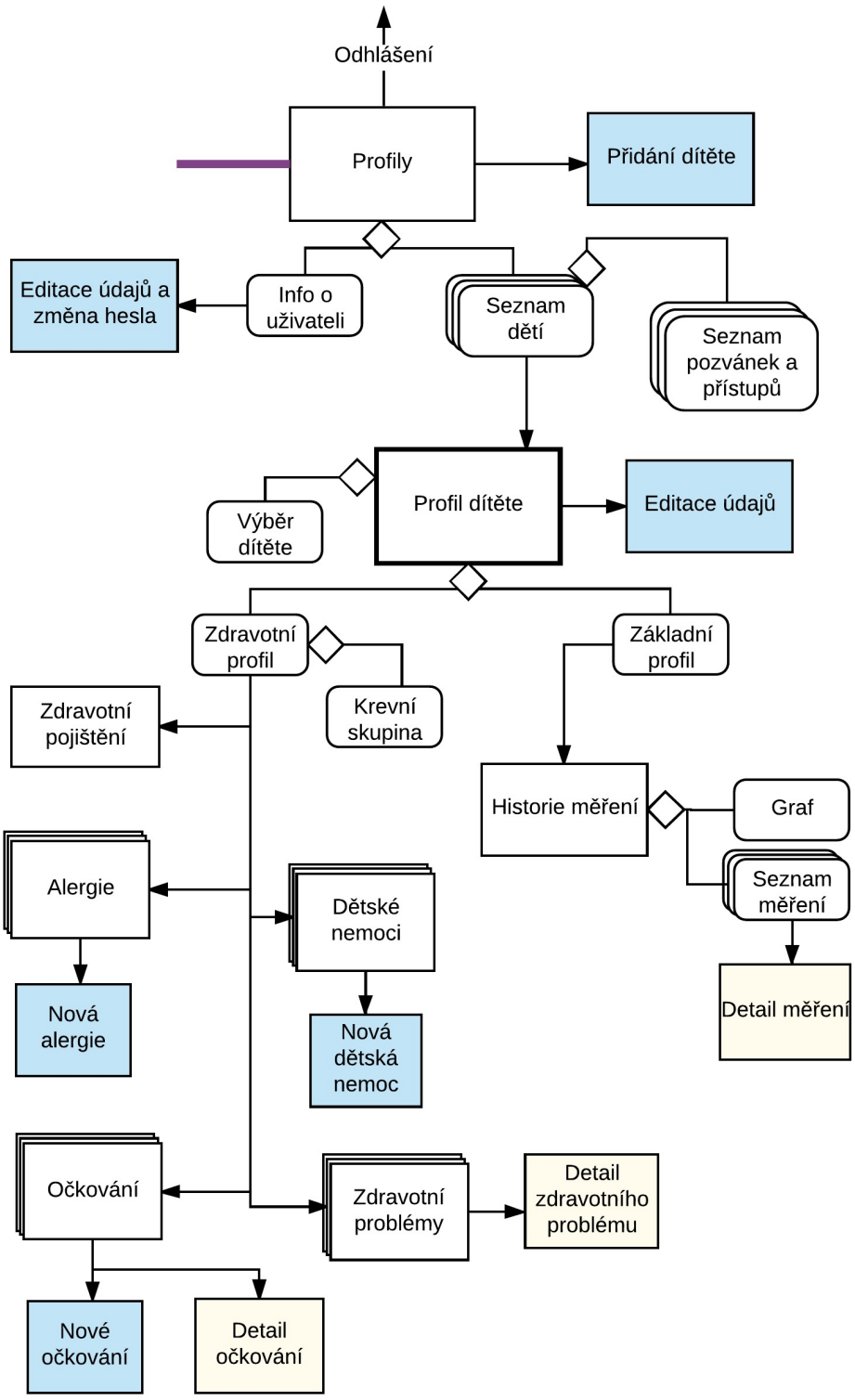
2.3.6 Přidávání nových záznamů

Poslední částí navigační struktury, kterou jsme ještě nepopsali, jsou formuláře pro přidávání nových záznamů, na které je možné se dostat přes tlačítko „+“ na spodní liště v iOS verzi nebo přes plovoucí tlačítko v Android verzi a dialog s výběrem typu záznamu ukázaném na snímku obrazovky A.4. Na tyto obrazovky nepotřebujeme vlastní diagram, neboť jsou všechny modální, takže z nich nelze odejít jinam než zpět. Zároveň je každé přidání jen na jeden krok.

Zbývá pouze vyřešit otázku, jakým způsobem má uživatel vybrat, ke kterému dítěti má záznam patřit. Přidávat mezikrok s obrazovkou či dialogem pro výběr dítěte nám přišlo jako příliš složité, rozhodli jsme se tedy navrhnout ovládací prvek, pomocí kterého lze mezi dětmi jednoduše přepínat. Na snímcích ho čtenář pozná podle šipek vedle obrázku dítěte – kromě klepnutí na šipku funguje i přejetí prstem doleva či doprava. Tento „přepínač“ je použit i na obrazovce s kontakty a na profilu dítěte. Příklady přidávání záznamu jsou na snímcích obrazovky A.15 a A.16.

2.4 Shrnutí

V této kapitole jsme popsali návrh a ukázali realizaci uživatelského rozhraní podle analyzovaných požadavků, které v co největší možné míře dodržuje prin-



Obrázek 2.3: Schéma navigace – profily

cipy dobrého návrhu tak, jak je udává Nielsenova heuristická analýza.

Výsledné rozhraní obsahuje celkem 47 různých obrazovek a dialogů, na kterých jsou používány desítky dílčích znovupoužitelných komponent. I přes relativně skromné klíčové cíle aplikace se tedy nejedná o malý kus softwaru, zvláště pokud přihlédneme k nefunkčním požadavkům z předchozí kapitoly. V následující kapitole se už podíváme na aplikaci ze softwarově-inženýrského pohledu a budeme řešit, jak popsaný návrh co nejlépe implementovat.

Kapitola 3

Návrh architektury

V této kapitole se podíváme na architekturu celého systému, tedy mobilní aplikace BabyDiary, její serverové části a databáze. Nebudeme se ještě zabývat konkrétními detaily implementace jednotlivých částí, ale zaměříme se především na výběr hlavních podpůrných technologií a jejich adaptaci pro náš konkrétní případ.

3.1 Výběr technologie pro vývoj aplikace

Základním cílem celého projektu je, aby vznikla mobilní aplikace pro operační systémy iOS i Android. Obě platformy jsou ale fundamentálně odlišné z hlediska vývoje, neboť používají jiné programovací jazyky a nástroje.

Již od počátku rozmachu aplikací se komunita mobilních vývojářů i softwarové firmy snaží najít způsob, jak tyto rozdíly přemostit a snížit tak náklady na vývoj. V současné době existuje již několik rozšířených knihoven, které lze pro multiplatformní vývoj použít. V této sekci prozkoumáme ty nejslibnější z nich a vybereme technologii, na které bude mobilní aplikace postavená.

3.1.1 Nativní vývoj

Začneme tou nejčastěji používanou metodou řešení problému, a sice vývojem dvou různých aplikací – jednu pro Android napsanou v Javě, druhou napsanou pro iOS v jazyce Swift. Obě budou vypadat i se chovat podobně, jen budou postavené odlišným způsobem. Je to sice nejméně zajímavá a nejnákladnější možnost, ale výsledek je zdaleka nejkvalitnější. To protože má vývojář přímý přístup ke všem možnostem platformy a vyvíjí způsobem, kterým to tvůrci operačního systému zamýšleli. Každá „zkratka“ oproti tomuto řešení je určitý druh kompromisu po stránce kvality, který lze zmírnit, ale nikdy ho nelze eliminovat.

Je také potřeba říct, že i v případě, že se rozhodneme pro nějakou z knihoven pro multiplatformní vývoj, nakonec stejně budeme muset pracovat s dvěma

různými softwarovými projekty, dvěma způsoby testování a dvěma systémy pro distribuci aplikace. Idea kódu, který je možné napsat jednou a spustit v mnoha prostředích, jak ji známe například z platformy Java Virtual Machine, není zdaleka tak dobře realizovatelná ve světě mobilních aplikací.

3.1.2 Apache Cordova a přidružené knihovny

Apache Cordova [7] je nejstarší a nejrozšířenější knihovnou pro multiplatformní vývoj. Koncept je téměř stejně starý jako mobilní aplikace samotné, první verze knihovny byla představena už v roce 2009. Od té doby se jí podařilo dostat na svou stranu výrobce mobilních platforem i firmy vyvíjející aplikace. Za dobu její existence na ní vznikly a zanikly desítky nadstaveb, jádro technologie je ale pořád stejné.

Cordova je založená na tom, že každá podporovaná mobilní platforma (vedle iOS a Android existuje podpora pro Windows Phone, ale také například Blackberry OS nebo Badu) obsahuje UI prvek, který se chová jako internetový prohlížeč. V angličtině se tento prvek nazývá „webview“ a tímto názvem ho budeme dále v textu označovat. Vývojář pomocí webview může zakomponovat prohlížení webu přímo do aplikace, aby z ní uživatel nemusel přepínat do vychozího prohlížeče systému. Webview má vlastní Javascriptový interpreter a dokáže vykreslovat HTML a CSS kód.

Aplikace postavená na Cordově je vlastně jeden velký webview, který svůj kód nenačítá z internetu, ale má ho přibalený přímo v archivu aplikace. Celý obsah aplikace je tedy napsaný jako webová stránka. Kromě toho nabízí Cordova Javascriptové rozhraní pro některé hardwarové funkce, které by jinak ve webview nebyly dostupné, například kompas, akcelerometr apod.

Apache Cordova a různé „mobilní webové knihovny“, které poskytují imitace nativního rozhraní – jako příklady uveďme jQuery Mobile, Sencha UI nebo moderní Ionic – se těší poměrně velké popularitě. Více než 6% dostupných mobilních aplikací pro Android je postavených na Cordově [8]. K vytvoření aplikace není potřeba nabírat ani zaučovat mobilní vývojáře, stačí využít webové vývojáře, kterých mají firmy mnohem víc. Pro manažery firem se koncept zdá být ideálním řešením dilematu mezi investicí do mobilního webu a investicí do mobilní aplikace, protože kombinuje výhody aplikací (lepší viditelnost na mobilním zařízením, více možností sledovat chování uživatele) s jednoduchostí vývoje webu.

Webview mají však jeden zásadní nedostatek. Provádění interpretovaného Javascriptového kódu je několikanásobně pomalejší než provádění nativního kompilovaného kódu. Grafické animace jsou také méně plynulé v prohlížeči než v obyčejné aplikaci, zvláště na zařízeních s OS Android. Když se k tomu přidá skutečnost, že celé uživatelské rozhraní se jen snaží kopírovat nativní prvky, vzniká recept na nízkou kvalitu aplikací a frustrované uživatele.

Existuje mnoho nadstavbových knihoven, které se snaží problémy s výkonem a s rozhraním řešit. Nejnovější a nejslibnější technologií v této oblasti

je už zmíněný Ionic Framework využívající populární webovou knihovnu Angular. Ionic má dle ohlasů zatím nejvěrnější reprodukci nativního UI i lepší výkon než jiné knihovny a pomalu se stává de facto automatickou volbou pro vývojáře používající Cordovu [9].

Vždy jde ale o citelný kompromis a čas strávený optimalizací může paradoxně spolknout veškerou časovou úsporu, kterou multiplatformní vývoj nabízí. Dle statistik uvedených v [8] nejsou žádné z nejpopulárnějších Cordova aplikací nějakými obrovskými hity, a tak ani po osmi letech vývoje mobilních aplikací ve webview je nelze postavit na stejnou úroveň jako aplikace vyvíjené nativně. Detailnější pojednání o stavu a budoucnosti této knihovny uvádí [10].

3.1.3 Xamarin

Xamarin (dříve Mono) začal jako technologie pro multiplatformní vývoj desktopových aplikací, po nástupu mobilních zařízení svou podporu rozšířil i na ně. Jedná se o open-source knihovnu, kterou v současnosti vlastní a vyvíjí společnost Microsoft.

Princip fungování je podobný jako například u Javy a virtuálního stroje Java VM. Základem je implementace platformy .NET od Microsoftu na různých operačních systémech, díky které může kód napsaný v jazyce C# běžet prakticky kdekoliv. Ne všechny funkce .NET jsou dostupné všude, ale nejpoužívanější komponenty ano. Na některých platformách, jako například iOS, je C# a .NET kompilován přímo do strojového kódu, na OS Android je spuštěn druhý virtuální stroj vedle toho nativního, který provádí „just-in-time“ kompilaci za běhu aplikace [11]. Přístup k rozhraním vývojové platformy je v Xamarinu řešen přímou vazbou na nativní knihovny, které lze volat z C# kódu úplně stejně jako z nativního. Není zde tedy potřeba psát žádné spojovací moduly jako je tomu u Cordovy i u React Native.

Díky tomu, že má Xamarin méně vrstev abstrakce než Cordova a nespoléhá na mobilní prolížeč, netrpí takovými problémy s výkonem a jeho použití neznamená automaticky, že bude výsledná aplikace pomalá. Jako další výhody uveďme integraci s populárním vývojovým prostředím Visual Studio, velmi podrobnou dokumentaci, podporu vyspělé a stabilní společnosti (Microsoft) a fakt, že na rozdíl od Cordovy se může chlubit velkými zákazníky i populárními aplikacemi [12].

Naopak hlavní nevýhoda Xamarinu spočívá v tom, že sám o sobě neřeší otázku uživatelského rozhraní. Poskytuje pouze způsob, jak provádět stejnou logiku na více platformách. Sice nehrozí, že by uživatelská rozhraní byla kvůli použité technologii nekvalitní, je však potřeba napsat dvě. Pro aplikace, u nichž většina náročnosti spočívá v tvorbě UI, Xamarin nepředstavuje žádnou velkou časovou úsporu. Částečné řešení nabízí nadstavbovou knihovnou Xamarin.Forms, která poskytuje společné rozhraní nad často používanými UI komponentami, ta se ale v prototypové fázi ukázala být málo flexibilní.

I přes tuto nevýhodu je Xamarin solidní volba pro multiplatformní vývoj, zvláště pokud už vývojáři aplikace mají zkušenosti s platformou .NET a jazykem C#.

3.1.4 React Native

React Native [13] je nejmladší ze zkoumaných technologií, společnost Facebook ho zveřejnila vývojářské komunitě teprve v roce 2015. V době psaní této práce je tento open-source projekt stále v relativně překotné beta fázi a nové verze knihovny jsou vydávány každý měsíc.

React Native se snaží vzít to nejlepší z modelu Cordova, aniž by musel zdědit všechny jeho neduhy. Kód aplikace se píše v jazyce Javascript, díky čemuž je knihovna přístupná široké komunitě webových vývojářů. Kód ale není spouštěn plnohodnotným prohlížečem, nýbrž jen „bezhlavým“ interpreterem, což je mnohem rychlejší. Uživatelské rozhraní se nevykresluje pomocí HTML a CSS, ale je realizováno čistě pomocí nativních komponent.

Na rozdíl od Xamarinu, který uživatelské rozhraní spíše neřeší, React Native naopak neřeší skoro nic jiného. Jeho myšlenka je taková, že nejnáročnější část vývoje mobilní aplikace je právě uživatelské rozhraní a tok dat mezi ním a aplikační vrstvou kódu. Pro bližší popis základních principů technologie je potřeba nejprve začít u sesterské knihovny React.js. Ta nabízí nový pohled na historický problém, jak bezbolestně dynamicky měnit obsah stránky napsaný v jazyce HTML pomocí Javascriptu. Zatímco jiné technologie berou HTML dokument a nějakým způsobem s ním manipulují, React vytváří strukturu celé stránky v Javascriptu a HTML obsah z ní vyplyne. Další klíčové koncepty Reactu a jejich protějšky v React Native jsou následující:

Stromová struktura a skládatelnost komponent. Aplikace v Reactu je jeden velký strom komponent, z nichž každá může mít vlastní dílčí komponenty. Těmito uzly jsou povětšinou prvky uživatelského rozhraní, které mají vlastní aplikační logiku a zároveň je možné nějak vykreslit na obrazovce. Vykreslování uživatelského rozhraní probíhá průchodem tohoto stromu od kořene dolů. Komponenty jsou vždy nezávislé na svém kontextu, tedy starají se pouze o sebe a o své podstromy. Lze tak jednoduše skládat a vnořovat komponenty jednu do druhé.

Tento systém je zvláště užitečný pro webové rozhraní a jazyk HTML, neboť to je také organizované stromově a vykreslení komponenty zde spočívá ve vygenerování HTML kódu na základě jejích dat. Ostatní rozhraní, na která jsme zvyklí, jsou ale také hierarchická.

U React Native jsme již říkali, že je rozhraní realizováno nativními komponentami. Funguje to tak, že místo HTML je výsledkem vykreslení komponenty XML kód, ve kterém figurují elementy, jež jsou pak knihovnou transformovány na nativní UI prvky. Pokud chceme definovat vlastní element, můžeme ho buď

poskládat z jiných, které už máme k dispozici, nebo napsat komponentu nativně a vytvořit spojovací rozhraní mezi ní a Javascriptem.

React Native také definuje způsob určení vzhledu, velikosti a pozice komponent, který se velmi podobá jazyku CSS, ale ve skutečnosti je to jen jeho malá podmnožina. Kód napsaný tímto systémem je opět knihovnou přeložen a aplikován na nativní komponenty.

Jednosměrný tok dat. Data v React aplikaci jsou předávána vždy z vrchu stromu dolů, nikdy opačným směrem. Data předávaná potomkům jsou neměnná a komponenta by správně ani neměla držet referenci na svého rodiče. Pokud vznikne v potomkovi událost, která by měla změnit data na vyšší úrovni než on sám, je potomkovi předána reference na funkci (tzv. callback) z vrstvy, která data změní sama a předá je zpět nejvyššímu uzlu, který o ně má zájem, aby na změnu mohl automaticky zareagovat celý podstrom.

Idea neměnných dat a funkčních referencí je převzatý z principu funkcionálního reaktivního programování, kterým je filozofie Reactu silně ovlivněna. Toto paradigma se snaží hrát stejnou roli jako problematický návrhový vzor Observer [14], který je v nějaké podobě přítomný téměř ve všem složitějším softwaru. Na rozdíl od Observeru, který pracuje se stavem konkrétních objektů, se FRP soustředí na jednosměrné asynchronní toky dat, což je flexibilnější a umožňuje volnější vazbu mezi entitami. Bližší popis reaktivního funkcionálního programování nezávislý na programovacím jazyce uvádí například [15].

Deklarativní programování namísto imperativního. React nutí vývojáře, aby popisoval, jak má komponenta za jakých okolností vypadat (z čeho se má skládat), spíše než jak se má chovat. Stav komponenty by měl být jednoznačně dán daty, která jsou uzlu předána. V zájmu pragmatičnosti a flexibility to v praxi neplatí úplně stoprocentně, nicméně ve většině případů je kód uživatelského rozhraní především funkce aktuálního stavu aplikace. To obrovským způsobem napomáhá přehlednosti, čitelnosti a udržovatelnosti zdrojového kódu.

Mimo tento specifický způsob tvorby uživatelských rozhraní se React Native příliš neliší od Cordovy. Až na několik základních výjimek neobsahuje přímé vazby na nativní knihovny, jako má Xamarin, takže pokud nějaké chceme použít, musíme buď najít, nebo napsat vlastní modul v nativním kódu.

Stejně jako ostatní realizace multiplatformní myšlenky není ani React Native bez vad na kráse. Tou největší z nich je nezralost celého ekosystému a paradoxně i jeho open-source povaha. Každá aktualizace knihovny naruší kompatibilitu velké části modulů třetích stran, které jsou často samy teprve ve fázi vývoje. Otevřenost a minimalistická povaha platformy nahrává štěpení nadstavbových projektů a výojářského úsilí, a tak používání této technolo-

gie je spojeno s jistým pocitem dobrodružství a inovace na jedné straně, ale také s rizikem velkého úsilí stráveného bojem s nedokončenými, nekompatibilními nebo nekvalitními knihovnami na straně druhé. Toto je však něco, co by se mělo postupem času zlepšovat, pokud se technologie vývojářské komunitě mezitím neomrzí.

3.1.5 Srovnání a výběr

Nyní je potřeba vybrat technologii, pomocí které implementujeme mobilní aplikaci BabyDiary. Pro začátek si shrňme výhody a nevýhody nativního vývoje:

- Výhody:
  - Není třeba slevit z kvality aplikací.
  - Vzniká méně závislostí na kódu třetích stran.
  - Lze čerpat znalostí obrovské vývojářské komunity a využívat obsáhlé dokumentace.
- Nevýhody:
  - Je nutné dobře znát oba programovací jazyky a obě vývojová prostředí.
  - Vyvíjí se dvakrát téměř stejný kus softwaru, jen různými způsoby.

Z tohoto seznamu by se mohlo zdát, že experimentovat s multiplatformním vývojem není vůbec dobrý nápad. Dle názoru autora, který má s touto problematikou několikaleté zkušenosti, v mnoha případech opravdu není. Aby bylo vůbec možné uspět, je potřeba dobře znát nejen výhody, nevýhody a omezení zvolené technologie, ale také je potřeba mít zkušenosti s nativním vývojem, kterému se v malém množství nejde vyhnout nikdy. V neposlední řadě je potřeba mít realistická očekávání a počítat s tím, že časová úspora rozhodně nebude poloviční.

U tohoto projektu bylo seznámení se s možnostmi multiplatformního vývoje jedním z vedlejších cílů klienta, neboť za předpokladu, že se budou tyto metody dále zdokonalovat, jsou zkušenosti s nimi do budoucna perspektivní investicí. Navíc měla aplikace vypadat na obou platformách téměř totožně a neměla využívat žádnou vysoce specializovanou technologii, pro kterou by nebylo možné vymyslet rozhraní společné oběma platformám. Byly to především tyto skutečnosti spolu se zkušenostmi autora s multiplatformním i nativním vývojem, díky kterým použití knihoven popsaných výše připadalo v úvahu.

Výhody a nevýhody jednotlivých multiplatformních řešení přehledně shrnujeme v tabulce 3.1.5. Data použitá pro tuto tabulku nepochází jen z dokumentace a internetových zdrojů, ale také ze tří prototypů, které byly před

	Cordova	**Xamarin**	**React Native**
Programovací jazyk	HTML, CSS, Javascript	C#	Javascript
Provádění kódu	Mobilní prohlížeč	Překlad nebo bytekód	JS interpreter
Výkon	Výrazně pomalejší než nativní	Srovnatelné s nativním	O něco pomalejší než nativní
UI komponenty	HTML	Nativní	Některé nativní, další skládáním
Využití nativního kódu	Přes adaptér	Přímo	Přes adaptér
Pokročilé vývojové nástroje	Hot swap, live reload	Pouze kompilace	Hot swap, live reload
Časová úspora	UI i aplikační logika	Pouze aplikační logika	Aplikační logika a struktura UI

Tabulka 3.1: Srovnání technologií pro multiplatformní vývoj mobilních aplikací

začátkem ostrého vývoje sestaveny. V prototypech jsem zkoumal plynulost uživatelského rozhraní, dostupnost důležitých knihoven či UI komponent a v neposlední řadě také celkový pocit z programovacího jazyka a vývojových nástrojů.

Apache Cordova už od počátku nebyla vážným kandidátem, hlavně díky negativním zkušenostem z minulosti. Rozhodování bylo především mezi Xamarinem a React Native. Nakonec i přes rizika a nevyspělost React Native jsem se rozhodl použít právě tuto technologii. Hlavní důvod byla knihovna PouchDB, kterou představíme v následující sekci, a sympatický vývojový model. Menší roli ale hrála také má neznalost platformy .NET a jazyka C# a naopak roky zkušeností s Javascriptem. Při učení se nové technologii je každá aplikovatelná zkušenost velkou časovou úsporou.

3.2 Návrh serverové části

Díky tomu, že musí být data vložená do aplikace dostupná nejen na zařízení, kde byla uložena, ale i na jiných zařízeních stejného uživatele, ba dokonce na

zařízení úplně jiných uživatelů (pokud je jim udělen přístup), musí systém nějak řešit síťovou komunikaci. Zvolíme klasickou architekturu klient-server, kde instance aplikace komunikují s centrálním webovým serverem pomocí protokolu HTTP, neboť je to u mobilních aplikací standard a cokoli jiného je pro nás zbytečně složité.

V této fázi bychom mohli mechanicky pokračovat výběrem platformy, na které webový server postavíme, a začít se zabývat komunikací mezi ním a aplikací. Odprostíme-li se ale od konvenčního myšlení a připustíme použití jiného modelu než webový server s relační databází, můžeme zkusit vyřešit více problémů najednou než jen otázku, kde budou ležet data. Největším oříškem z celého projektu je totiž problém offline synchronizace. Proto se nejprve zaměříme na to, jak ho vyřešíme, což výrazným způsobem ovlivní architekturu celého systému a požadavky na serverovou část aplikace. Zjistíme, že trochu více se zamyslet nad způsobem ukládání dat na serveru i v mobilním zařízení nám může vývoj značně usnadnit.

3.2.1 Problém offline synchronizace

Problém offline synchronizace spočívá v tom, jak zajistit přenos, dostupnost a konzistenci dat za podmínek, kdy mobilní zařízení může na dlouhou dobu fungovat mimo připojení k internetu. Aby mohl systém za takových okolností plnit svůj účel a nemusel uživateli blokovat přístup, dokud se nepřipojí k síti, musí být splněny následující podmínky:

- **Dostupnost dat.** Všechna kritická data, bez kterých aplikace nemůže mimo připojení k síti fungovat, musí být ukládána do pevné paměti zařízení a periodicky aktualizována, buď akcí uživatele, nebo automaticky na pozadí. Mobilní aplikace tedy nemůže dostávat jen výsledky výpočtů, jako to často dělají webové aplikace, ale musí mít k dispozici data, ze kterých výsledky může vypočítat sama.

  Nabízí se řešení mít všechna data vždy replikovaná všude. S tím ale mohou vzniknout problémy, pokud mají být některá data utajená a na klientské zařízení má dorazit jenom jejich podmnožina nebo nějaký agregovaný výstup ze serveru. Pokud se data na klientovi změní mimo připojení k síti, daný výstup už nebude možné znovu vygenerovat.

- **Ukládání změn.** Změny v uživatelských datech se musí také ukládat do trvalé paměti zařízení, protože není zajištěno, že bude do vypnutí aplikace server již dostupný. Po obnovení dostupnosti se musí všechny uživatelské změny nahrát na server v pořadí, v jakém byly provedeny. Ověřování autorizace ke změnám a validaci dat musí provádět nejen server, ale i klient, aby se nestalo, že změny po připojení k síti nebude možné do databáze zapsat.

- **Řešení konfliktů** Nakonec musí existovat způsob řešení konfliktních modifikací, pokud se jeden uzel pokusí aplikovat změny na data, která byla mezitím změněna nebo smazána jinde. Můžeme buď konflikty řešit automaticky, například tím, že vždy vyhraje nejnovější revize, nebo můžeme zodpovědnost přenést na uživatele a poskytnout mu rozhraní pro rezoluci konfliktů.

Tento problém existuje ve vývoji mobilních aplikací již od jejich zrození, bohužel však dodnes neexistuje jediné akceptované řešení a technologie, která by ho implementovala, a tak se problém řeší pořád dokola. Je to možná dáno tím, že každá aplikace má jiné požadavky na to, jak velká část funkcionality musí být dostupná offline a jiné představy o tom, jak by se měly řešit konflikty. Zajistit první dva body je relativně snadné, konflikty se ale řeší obtížně a je zde velký prostor pro selhání. Ztráta dat je ten nejlepší způsob, jak přijít o uživatele.

Návrh řešení problému offline synchronizace provedeme v několika krocích. Nejprve stanovíme, jak chceme přistupovat ke každému ze tří vyjmenovaných bodů. Dostupnost dat asociovaných s profilem dítěte můžeme řešit jejich replikací na všechna zařízení, kde je přihlášen uživatel mající k profilu přístup. Přestože máme více úrovní oprávnění, u žádné z nich nepotřebujeme tato data skrývat. Dostupnost dat na mobilním uzlu je tedy dána pouze existencí či neexistencí uživatelského přístupu, nikoli jejich povahou. Jiné je to u dat, která zajišťují bezpečnost aplikace, tedy hashe hesel, OAuth klíče, apod. Přihlášení, registrace ani změna hesla samozřejmě offline fungovat nemůžou.

Ukládání změn budeme muset implementovat přesně tak, jak říká naše vlastní doporučení, tedy včetně ověřování autorizace ke změnám u opatrovnických účtů a kompletní validace uživatelských dat v mobilní části. Bylo by pro nás složité nutit uživatele opravovat data dlouho poté, co je do aplikace vložil, jenom protože je server odmítnul. Můžeme nakonec i vypustit validaci některých druhů dat na serveru, neboť už budou validována z aplikace. Co však musí i nadále ověřovat aplikační server, jsou jakékoli akce vedoucí ke změně uživatelských oprávnění nebo přístupů, a to z bezpečnostních důvodů. V aplikaci nebude možné vytvářet ani mazat profily dětí, odesílat, mazat ani přijímat pozvánky a odebírat přístupy mimo připojení k internetu.

Co se týče řešení konfliktů, chceme tím co nejméně zatěžovat uživatele. Konflikty jsou do jisté míry okrajový případ, navíc u mnoha z nich bude správné řešení jednoduše nechat vyhrát nejnovější verzi. Nedává pro nás smysl navrhovat a implementovat uživatelské rozhraní pro řešení konfliktů. V situacích, kdy musí být jeden z údajů zahozen, použijeme ten nejnovější.

3.2.2 Role serverové části

Pokud se podíváme na systém jako celek a na roli backendu v něm, zjistíme, že po serverové části nemůžeme chtít víc než zajištění, aby byla správná data

dostupná na správných zařízeních. Kromě toho je zde také malé množství aplikační logiky jako řízení uživatelů a přístupů, vesměs je ale úkolem serverové části zařídit, aby všechny propojené uzly (mobilní zařízení a centrální databáze) měly k dispozici data, ke kterým mají mít přístup a která jsou aktuální. Žádnou další složitou logiku nemůžeme na server delegovat, neboť aplikace má být použitelná i mimo připojení k internetu a tudíž musí být soběstačná.

Můžeme tedy na systém pohlížet jako na jednu velkou distribuovanou databázi, ve které jsou různá data replikována na různé uzly. Je to zjednodušený pohled – nemůžeme dát uživatelům do ruky databázi a prohlásit práci za hotovou. Přidaná hodnota je ale hlavně v prezentační vrstvě, která musí být zajištěna mobilní aplikací, a v propojenosti uživatelů. Aplikace se koneckonců snaží plnit funkci papírového deníčku a ten nic jiného než databáze není. Deníčků je v našem systému ale mnoho, každý rodinný příslušník může mít svou kopii a zápisky se musí synchronizovat.

Máme-li data v systému replikovat, dává smysl, abychom použili stejný způsob ukládání jak na mobilním konci, tak na serverovém. Použitím stejného databázového řešení a stejné definice datového schématu docílíme toho, že bude automaticky zajištěna syntaktická i sémantická konzistence dat. Můžeme například použít stejný kód pro validaci modelů a nemusíme se start o to, jestli data přijatá od aplikace nezpůsobí při ukládání na serveru chybu.

Další krok v našich úvahách je výběr technologie, pomocí které distribuovanou databázi realizujeme. Naším cílem je, abychom museli co nejméně databázové logiky, zejména té synchronizační, vyvíjet sami.

3.2.3 Požadavky na distribuovanou databázi

Jelikož chceme používat stejnou databázi všude, prvním a nejdůležitějším požadavkem na databázovou technologii je její implementace na mobilních platformách. Tím se nám okruh kandidátů zúžil, ale jsou tu i další požadavky, které nám s výběrem pomohou.

Během posledních 17 let se pro popis různých distribuovaných systémů a zvláště databází stal velmi populární CAP teorém [16], který říká, že žádný distribuovaný systém nedokáže splnit požadavek na konzistenci, dostupnost a odolnost k přerušení zároveň. Při návrhu takového systému je potřeba minimálně z jedné vlastnosti z této trojice slevit. Je užitečné se na tyto tři vlastnosti podívat a stanovit, do jaké míry je systém musí splňovat.

Pro aplikaci BabyDiary a její serverovou část platí, že mezi konzistencí, dostupností a odolností k přerušení je to možná překvapivě konzistence, na které nám bude záležet nejméně. Jelikož nenakládáme s nijak kritickými daty, je pro nás mnohem důležitější dostupnost za špatných síťových podmínek a tím pádem i odolnost k přerušení. Stačí nám, pokud se správná data nakonec dostanou, kam mají, pokud možno bez konfliktních modifikací.

3.2.4 Výběr databázové technologie

Teď už se můžeme blíže podívat na jednotlivé databáze a vybrat z nich tu, kterou použijeme.

Relační a jiné nedistribuované databáze

Do kategorie nedistribuovaných databází zahrneme všechny, které nám s distribuovaností nepomohou a přinutily by nás od základu vymýšlet vlastní řešení datové synchronizace. Použijeme-li například na serveru relační databázi a část datové struktury zkopírujeme v aplikaci, budeme muset zajistit přenos dat vlastními silami.

Relační databáze jsou standardní volbou pro většinu aplikací a nebýt dostatku odvahy ze strany autora i klienta, byly by naší volbou i zde. Jsou seriózní, spolehlivá, snadno použitelná a dobře implementovaná technologie. Existují i použitelné implementace relačních databází na mobilních operačních systémech, například SQLite. Použití méně osvědčených prostředků k dosažení stejného cíle (perzistence dat) je z hlediska projektového řízení rizikem, ale výhody přinášené alternativními technologiemi byly vyhodnoceny jako větší.

Realm

Tato technologie je úplně nová a v době, kdy výběr probíhal, ještě její distribuovaná funkcionalita neexistovala. Přesto ji zde zmíníme, protože se zdá být slibná a je možné, že se časem stane definitivním řešením problému offline synchronizace.

Realm je v prvé řadě mobilní objektově orientovaná databáze pro Android a iOS, která slibuje zjednodušení práce i lepší výkon v porovnání s konvenčními způsoby ukládání dat na těchto platformách [17]. Je vyvíjená společností Facebook, a proto pro něj existuje podpora i v námi vybraném React Native. Kromě mobilní databáze obsahuje také serverovou verzi, která dokáže automaticky synchronizovat data mezi mobilním zařízením a serverem a dokáže i řešit konflikty.

CouchDB a PouchDB

Existují stovky různých alternativních databázových systémů a nemá smysl se o nich příliš rozepisovat, jelikož většina z nich je úzce specializovaná a náš případ do jejich specializace nespadá. Jedna z těchto úzce specializovaných technologií je ale pro nás velmi zajímavá, neboť se specializuje právě na synchronizaci dat přes nespolehlivou síť.

CouchDB je dokumentově orientovaná NoSQL databáze napsaná v jazyce Erlang. Díky tomu, že existuje její sesterská implementace jménem PouchDB, která je napsaná v Javascriptu, ji můžeme využít v mobilní aplikaci postavené

na React Native. Stačí jen připsat adaptér pro ukládání dat do pevné paměti zařízení.

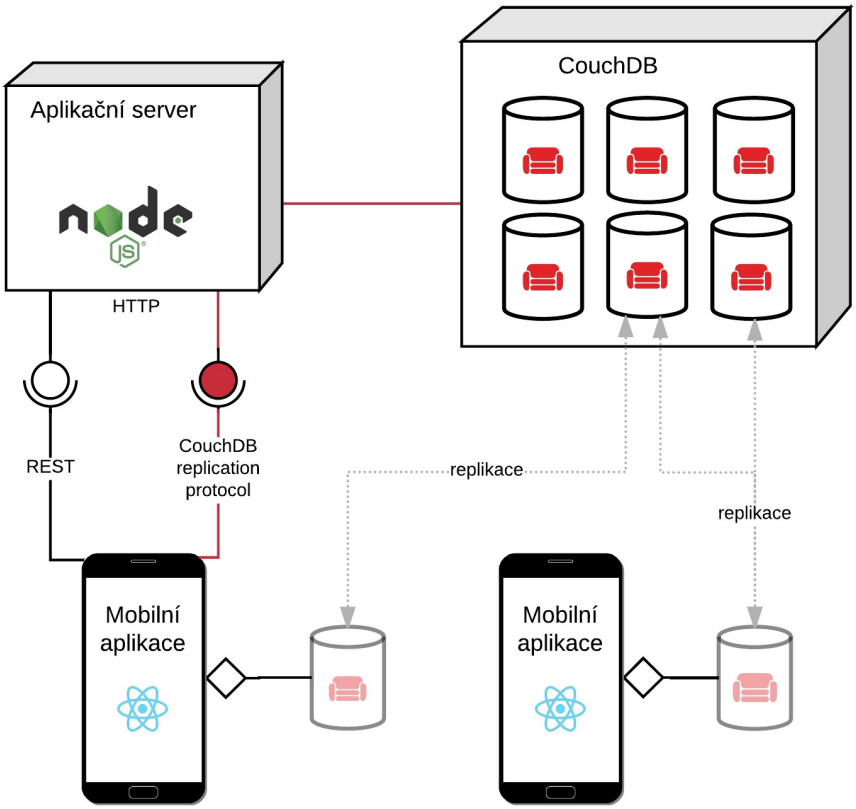
Tato databáze je specifická v tom, že více než o efektivní ukládání a vyhledávání dat se stará o jejich replikaci mezi různými instancemi [18]. Na CAP spektru dává silně přednost vysoké dostupnosti a odolnosti k přerušení před konzistencí. Díky těmto vlastnostem je blíže naším požadavkům než jiné databáze. Data jsou v CouchDB ukládána v podobě nezávislých dokumentů v textovém formátu JSON. Pokud chceme ukládat jiná data než textová, můžeme k dokumentu přidat binární přílohy. Databáze v CouchDB nemá schéma a až na výjimky v podobě ID dokumentu a čísla revize se systém o obsah dokumentů nijak nestará.

Dostupnost dat v distribuovaném prostředí je zajištěna verzováním dokumentů podobně, jako to dělají systémy pro verzování zdrojového kódu. Data se při zápisu nepřepisují, nýbrž se ukládá jejich nová verze. Pro čtení je pak dostupná verze z posledního zápisu. Každý požadavek na zápis musí kromě modifikovaných dat obsahovat i číslo verze, na kterou má být modifikace aplikována. Pokud přijde do databáze požadavek na modifikaci starší verze, než je ta aktuální, vznikne konflikt. Databáze uloží obě verze, označí dokument jako v konfliktu a je na klientovi, aby ho vyřešil, popřípadě ignoroval. Detailnější popis fungování databáze a jejího přístupu ke konzistenci udává například [19].

Chování databáze je možné modifikovat uložením speciálních dokumentů, tzv. „design docs“. Tyto dokumenty mohou například kontrolovat a odmítat změny v jiných dokumentech, takže opravdu nepostradatelnou validaci dat nebo ověřování vstupních podmínek lze do databáze připsat. Pomocí design docs se také ovládá replikace z jedné databáze do druhé. Replikovat lze buď celou databázi, nebo vybrané dokumenty a replikace může být buď jednorázová, nebo kontinuální, tedy spouštěná po každém zápisu do jedné z kopií.

Volnější definice toho, co může databáze ukládat, má své nevýhody. V databázi neexistuje koncept cizího klíče ani jiná reprezentace vztahů mezi daty. Pokud mají dokumenty odkazovat jeden na druhý, o nějakou referenční integritu se musí starat sám klient a efektivní vyhledávání je v databázi možné pouze podle ID dokumentu. Další pokročilé funkce, na které jsme zvyklí z relačních databází, buď úplně chybí, nebo jejich implementace spočívá ve vytvoření repliky databáze s trochu odlišnými vlastnostmi, což je sice často velmi flexibilní a užitečný přístup, nicméně s rostoucím počtem replik ještě více klesá schopnost systému dosáhnout konzistentního stavu bez konfliktů.

I přes tyto nedostatky jsem se rozhodl CouchDB na tomto projektu použít. Databáze dokáže sama řešit problém offline synchronizace a implementuje už v základu hlavní zodpovědnost naší serverové části, takže budeme moct věnovat více času vývoji samotné mobilní aplikace. Formát JSON navíc znamená jednoduchou integraci s Javascriptem. V neposlední řadě je celá databáze open-source, a tak si ji budeme moct přizpůsobit, pokud nám v něčem bude fatálně nevyhovovat.



Obrázek 3.1: Diagram architektury systému.

3.3 Architektura systému

Zde popíšeme navrhovanou architekturu z pohledu integrace serverové a mobilní části aplikace. Návrh jednotlivých komponent a detailnější popis technických řešení následuje v další kapitole.

Diagram architektury je na obrázku 3.1. Idea celého uspořádání je nechat CouchDB dělat co nejvíce práce a jen doplnit její funkcionalitu tam, kde jsou potřeba nějaké složitější procesy. Většina aplikační logiky bude napsaná v mobilní aplikaci a skoro každý proces bude končit úspěšným uložením, editací nebo smazáním dat v lokální databázi, která změny sama sesynchronizuje se serverovou replikou.

Implementaci webového serveru se nevyhneme, ale jeho role bude minimální. PouchDB v mobilní aplikaci a CouchDB na serveru musí komunikovat na přímo pomocí vlastního synchronizačního protokolu, který nechceme nijak komplikovat. Protože ale potřebujeme zabránit neoprávněnému přístupu

k datům a CouchDB na to nemá potřebné mechanizmy, bude webový server fungovat jako proxy v této komunikaci a všechny HTTP požadavky na databázi budou kontrolovány. Tímto způsobem budeme také moct kontrolovat, zda nějakým zápisem nevznikl konflikt, a případně ho zkusit opravit. Server bude napsaný v Javascriptu na platformě Node.js, aby byla integrace se zbytkem systému co nejjednodušší.

Kromě databázové replikace bude webový server obsluhovat požadavky na registraci a přihlášení uživatele, pomocí nichž získá mobilní klient ověřovací klíče pro následnou komunikaci. Další přidané služby serveru zahrnují funkce nahrávání obrázků, vytváření a mazání profilů dítěte, posílání a přijímání pozvánek, odebírání udělených přístupů a generování PDF dokumentů. Součástí těchto procesů může být odesílání e-mailových zpráv. Na všechny tyto služby vystaví server REST API, které bude využívat stejného autorizačního mechanizmu jako databázová proxy. Server bude napsaný v Javascriptu na platformě Node.js, aby byla integrace se zbytkem systému co nejjednodušší.

3.3.1 Uspořádání databází

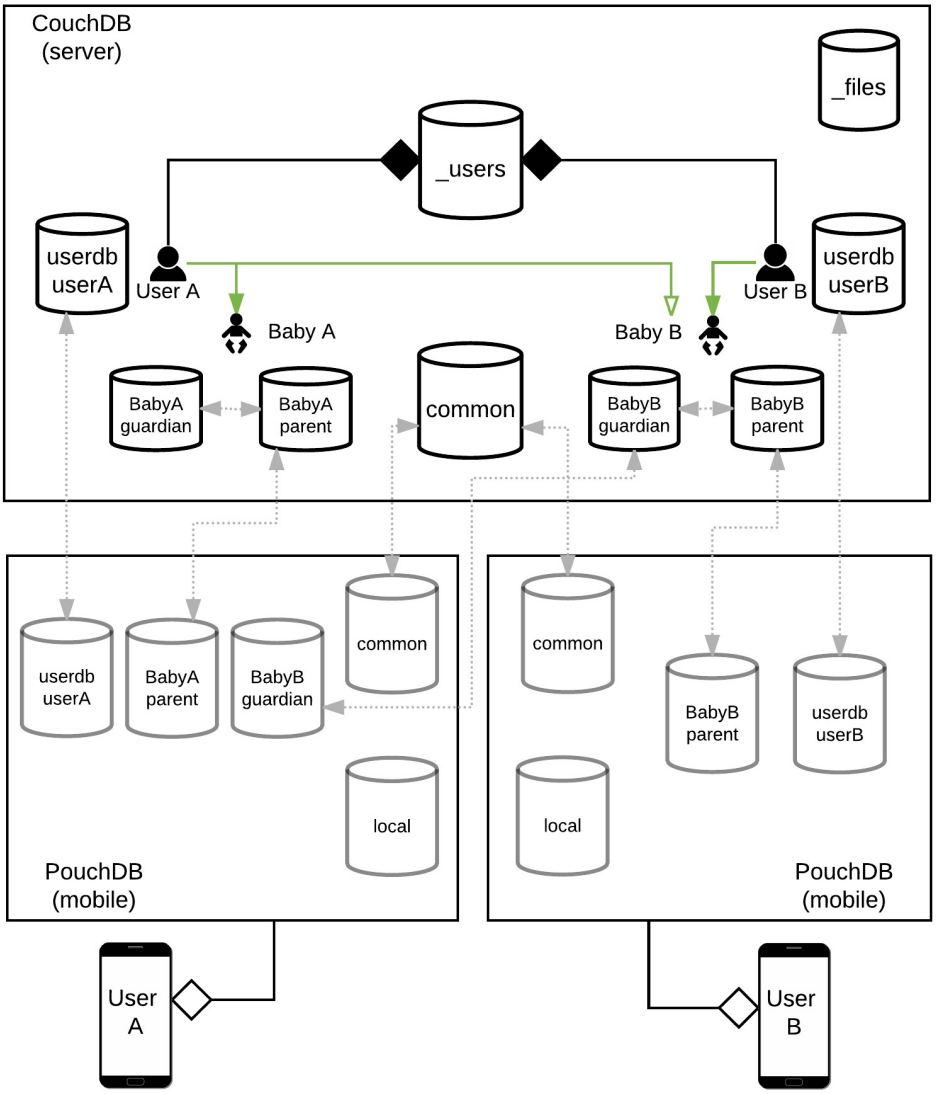
Popis využití CouchDB v předchozí sekci je poněkud zjednodušující. Jak chceme replikovat serverovou databázi, když každý uživatel má mít přístup pouze k malé podmnožině všech dat, která jsou na serveru uchovávána? Použijeme k tomu klasický návrhový vzor distribuovaných databází – vytvoříme repliky.

V CouchDB je totiž běžné a oproti jiným systémům výkonově nenáročné vytvářet spousty databází, mezi nimiž se mohou duplikovat data. Každá databáze pak představuje to, co bychom v relačním systému nazvali jako „pohled“. Proto je na diagramu architektury databázový systém zobrazen jako kontejner několika samostatných válců. Co se týče toho, jaké pohledy budeme vlastně potřebovat, máme tři možnosti, jak můžeme databázový systém pojmout:

- **Replikace z centrální databáze.** Na serveru bude jen jedna databáze a z ní nastavíme replikaci tak, aby se do mobilních kopií dostala jenom ta správná data. Velmi záhy se ukázalo, že tudy cesta nevede, neboť by v centrální databázi musel existovat jeden replikační dokument pro každé připojené zařízení a výsledná databáze by byla pomalá, neškálovatelná a nepřehledná.

- **Databáze pro každého uživatele.** Každý uživatel bude mít vlastní databázi, do které mu budou replikována správná data. Mobilní databáze pak bude kopie té uživatelovy. Databází bude víc, ale budou menší a přehlednější. Navíc bude v případě potřeby možné rozprostřít zátěž a ukládaná data mezi více fyzických strojů.

- **Databáze pro každé dítě.** Vlastní databázi bude mít každý profil dítěte. Udělený přístup k dítěti bude znamenat přístup k příslušné da-



Obrázek 3.2: Diagram uspořádání databází.

tabázi a možnost ji replikovat do mobilního zařízení. I takovou míru separace mezi daty si můžeme dovolit, protože v doménovém modelu prakticky neexistují vazby mezi objekty, které by patřily různým profilům. Kromě toho ale budeme mít i databázi pro každého uživatele, jen v ní budeme uchovávat data doopravdy specifická pro daného uživatele, například jméno, e-mailovou adresu, osobní nastavení apod.

Na počátku vývoje jsem zkoušel použít systém databáze pro každého uži-

vatele, ten se však rychle ukázal být nevyhovující. Nastavování replikace bylo příliš složité, protože u uživatelů, kteří měli přístup k více dětem, se v databázi míchaly záznamy od všech z nich a neexistovala jediná „autoritativní“ replika. Ještě větší problém bylo to, že každý záznam měl v systému tolik kopií, kolik uživatelů k němu mělo přístup. Databáze sice počítá s tím, že data existují na více místech, ale čím více kopií, tím složitější je dospět ke konzistenci. Největší problém nastal ve chvíli, kdy měl uživatel o nějaký přístup přijít. Museli bychom implementovat proces na selektivní mazání již replikovaných dokumentů, neboť CouchDB sama nemaže dokumenty po zrušení replikace.

Naproti tomu se systémem jedné databáze pro každý profil dítěte se dá pracovat mnohem přirozeněji. Na jednu stranu to znamená zesložitění kódu na mobilní straně, protože už není replikována jen jedna databáze, ale několik. Na druhou stranu to ale udržuje počet kopií jedněch samých dat na únosné úrovni a umožňuje udržovat jednu autoritativní repliku pro každý dokument. Navíc odpadá problém s odebraným přístupem, protože stačí jednoduše smazat celou repliku v mobilní aplikaci a proces je hotový. Úplně stejně je zjednodušen i proces vytváření přístupu.

Aplikace BabyDiary tedy používá systémy „databáze pro dítě“ i „databáze pro uživatele“. Je to ale ještě o něco složitější. Na ten samý profil dítěte existují v aplikaci dva možné pohledy – pohled rodiče a pohled opatrovníka a každý z nich používá jinou logiku pro ověřování operací. Opatrovník například nemůže smazat záznam, který sám nevytvořil. Oba pohledy sice zobrazují stejná data, ale nikde není záruka, že se to v budoucnosti nezmění. Proto potřebujeme mít i způsob, jak data oddělit, aniž bychom se vrátili ke konfigurační noční můře, kterou je selektivní replikace do mobilní databáze.

Problém je vyřešen rozdělením profilu dítěte na dvě databáze: jednu pro neomezený přístup (vlastník a rodič) a druhou pro omezený (opatrovník). Mezi těmito dvěma databázemi je nastavená kontinuální replikace. Rodičovská databáze je stále autoritativní kopií dat, ale uživatel s přístupem opatrovník do své mobilní aplikace replikuje opatrovnickou databázi.

Diagram výsledného návrhu je na obrázku 3.2. V něm je znázorněna situace, kdy máme dva uživatele a každý z nich vlastní profil jednoho dítěte. V aplikaci na telefonu na levé straně je přihlášen uživatel A, na pravé straně uživatel B. Zelená čára indikuje přístup k dítěti – plná šipka znamená rodičovský přístup, prázdná opatrovnický. Šedé čáry představují probíhající replikaci. Uživatel A má tedy opatrovnický přístup k dítěti uživatele B. Do mobilních zařízení je pak replikována jedna uživatelská databáze plus jedna databáze pro každý přístup k dítěti, který uživatel má. Technické detaily toho, jakým způsobem je takové množství replikací zvládáno po výkonové stránce, uvedeme v následující kapitole.

Kromě databází pro uživatele a pro profily dětí existuje ještě jediná databáze s názvem `common`. Ta v sobě obsahuje data, která jsou pro všechna zařízení s aplikací společná; jedná se především přednahrané možnosti výběrů různých hodnot, jako například názvy milníků. Tato data chceme mít možnost

aktualizovat na serveru a synchronizovat je do mobilních klientů bez nutnosti vydávat novou verzi aplikace.

Server má navíc dvě tajné databáze. Databáze (`_users`) drží seznam uživatelů a jejich autentizační informace, databáze (`files`) ukládá nahrané obrázky z aplikace. Kvůli optimalizaci výkonu a kontrole nad množství používaných dat nechceme přenášet binární data přes standardní replikaci, tudíž je nemůžeme ukládat jako přílohy k dokumentům v jiných databázích. Místo toho je bude server držet ve `files` a vystavovat je ke stažení přes zvláštní API.

3.4 Shrnutí

V této kapitole jsme rozpracovali analyzované požadavky do hrubého návrhu architektury systému, jehož je cílová mobilní aplikace součástí. Kromě multiplatformní mobilní aplikace postavené na knihovně React Native má systém i serverovou část, tvořenou zejména databází CouchDB, která zajišťuje autentizaci a vzdálené úložiště, ze kterého se replikačním protokolem databáze dle potřeby přihlášeným uživatelům do mobilních zařízení synchronizují data.

Následující kapitola se zabývá konkrétními implementačními detaily a popisuje, jak vývoj probíhal a jak byly vyřešeny problémy, které během implementace nastaly.

Kapitola 4

Implementace

Tato kapitola popisuje ty největší a nejzajímavější problémy, na které jsem během vývoje aplikace narazil, a způsob, jakým byly vyřešeny. Kombinace React Native a CouchDB/PouchDB je v době vzniku této práce stále relativně inovativní postup. Cílem kapitoly je poskytnutou materiál popisující řešení netriviálních problémů, které mohou při použití zmíněných technologii nastat.

Finální produkt je poměrně rozsáhlý projekt, a tak není možné zde popsat každé implementační rozhodnutí, omezíme se pouze na problémy, jejichž řešení není úplně evidentní. Jako na každém projektu, ne všechna technická rozhodnutí na tomto projektu byla nutně správná a proto v této kapitole narazíme i na kritiku některých použitých postupů. Dle názoru autora jsou i takové zkušenosti cenné a zasluhují si být popsány.

Kapitola je rozdělena na čtyři části. Nejprve rozvedeme doménový model ze sekce 1.4 a vytvoříme z něj databázové schéma, které bude sdílené pro server i aplikaci. V další sekci se budeme věnovat serverové části systému, komunikaci s mobilním aplikací a synchronizaci dat. Třetí sekce představí architekturu a implementaci mobilní aplikace samotné. Poslední sekce zhodnotí celkovou implementaci z technického hlediska a poskytne základ pro diskuzi o možných alternativních postupech či zlepšeních do budoucna.

Poznámka ke knihovnám třetích stran

Použití open-source knihoven třetích stran v softwarových projektech je dnes pro většinu projektů naprostou samozřejmostí, bez ohledu na konkrétní platformu nebo programovací jazyk. Ekosystém Javascriptu tento princip zavádí do extrému, jednak díky tomu, jak omezené jsou jeho možnosti v základu, jednak díky jednoduchosti, s jakou lze externí knihovny používat. Je běžné mít desítky malých externích závislostí, například na práci s textovými řetězci nebo s časovými údaji.

React Native jako platforma je už od počátku stavěna tak, aby mateřská společnost Facebook s jeho údržbou neměla příliš velké množství práce, a tak se za každou cenu snaží nerozšiřovat jeho možnosti. Předpoklad je takový, že si vývojářská komunita dovyvine, co bude potřeba. To se sice děje, ale výsledkem je fakt, že hotové projekty mívají desítky závislostí. Naše aplikace není výjimkou. Díky množství externích závislostí – přímých i nepřímých – zde není možné všechny zmínit, a tak se omezujeme na ty nejdůležitější. Kompletní seznam přímých externích závislostí se nachází v kořenové složce kódu mobilní aplikace v souboru `package.json`.

4.1 Schéma databáze

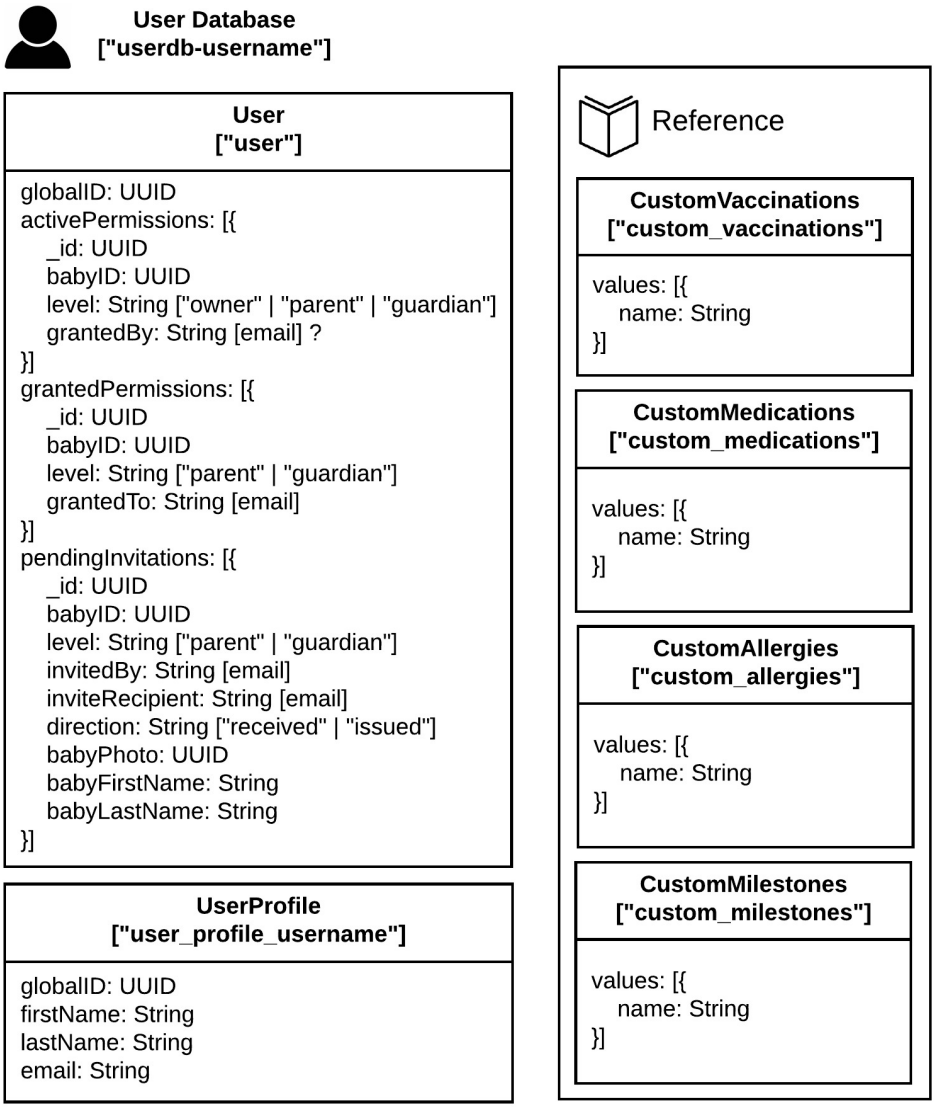
Přestože jsme říkali, že CouchDB nevynucuje schéma a bez řečí uloží cokoli, co je ve formátu JSON, tak pro účely vývoje bylo nezbytné se nějaké předem definované struktury držet, už jenom kvůli zajištění interoperability různých částí kódu. Jelikož Javascript, v němž je naprostá většina kódu aplikace napsaná, je dynamicky typovaný jazyk, nemohl jsem se spolehnout na žádnou automatickou kontrolu datových struktur a dodržování schématu bylo nutné hlídat ručně. Proto bylo důležité mít papírovou specifikaci.

Schéma databáze je rozděleno do několika diagramů. Diagramy jsou rozděleny podle toho, do jakého druhu databáze dokumenty patří. Aby se schémata vešla na stránku, jsou některá méně důležitá datová pole vypuštěna, zejména pole `modified`, které je obsaženo v každém dokumentu i vnořeném objektu a které slouží k prioritizaci verzí při řešení konfliktů.

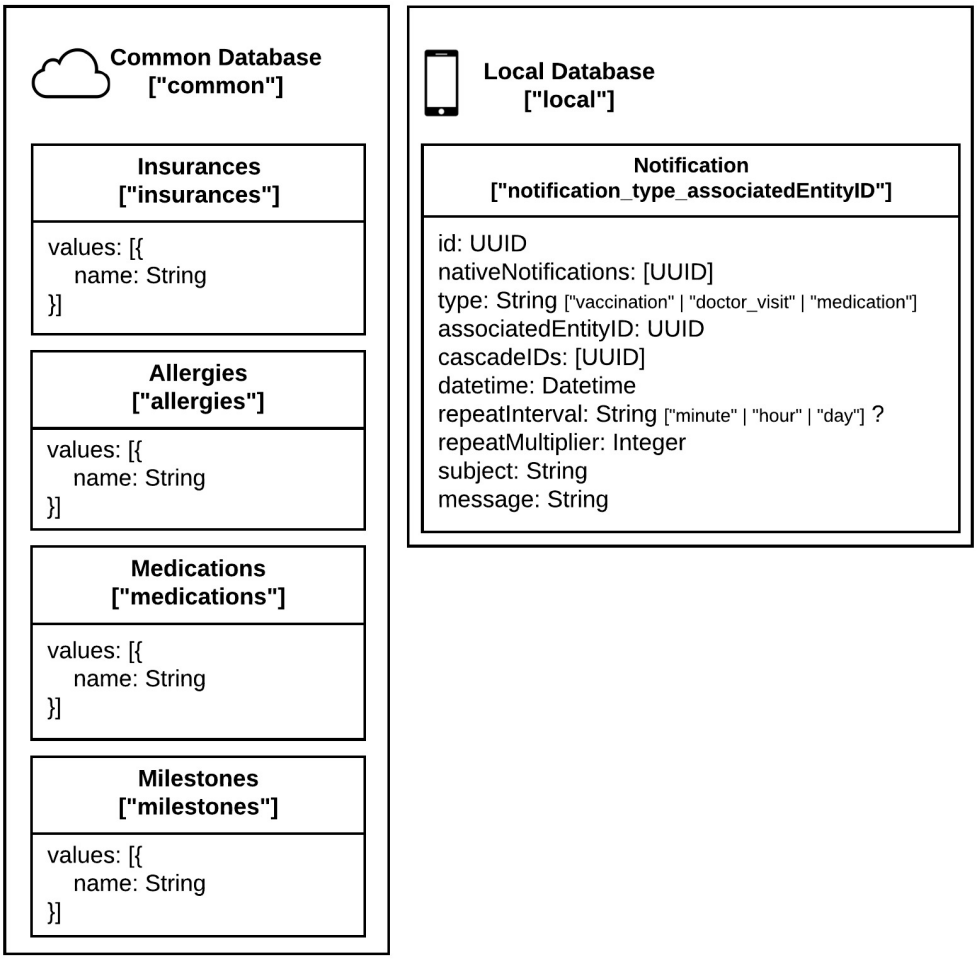
Na obrázku 4.1 jsou dokumenty patřící do databáze uživatele. Dokument `user` drží především informace o tom, ke kterým profilům dětí má daný uživatel přístup. Tato data slouží pouze jako vodítko mobilní aplikaci, aby věděla, jaké jiné databáze má replikovat, a nehrají roli v žádném bezpečnostním mechanizmu. Ostatní dokumenty v uživatelské databázi slouží k ukládání hodnot pro našeptávač. Pokaždé, když uživatel zadá název očkování, léku, alergie nebo milníku, se tato hodnota uloží do jeho osobní databáze, aby mu ji příště mohla aplikace napovědět. Díky synchronizaci databází se tyto hodnoty neztratí ani poté, co se přihlásí k jinému zařízení.

Na obrázku 4.2 je zobrazeno schéma databáze `common`. Tato databáze slouží k synchronizaci dat, která mají být sdílena mezi všemi uživateli, ale mohou být upravována pouze administrátory aplikace na serveru. Ideální využití takové databáze je pro vzdálenou konfiguraci pomocí nějakých nastavení, v průběhu implementace ale vyšlo najevo, že žádná taková konfigurace zatím není potřeba. V budoucnu budeme ale moci `common` využít například pro implementaci datových migrací. Prozatím databáze ukládá hodnoty, které jsou přednastaveny do našeptávačů. To jsou například základní milníky: první úsměv, první zoubek, první slovo a první krůček.

Rovněž na obrázku 4.2 je databáze `local`, která slouží jako privátní úlo-



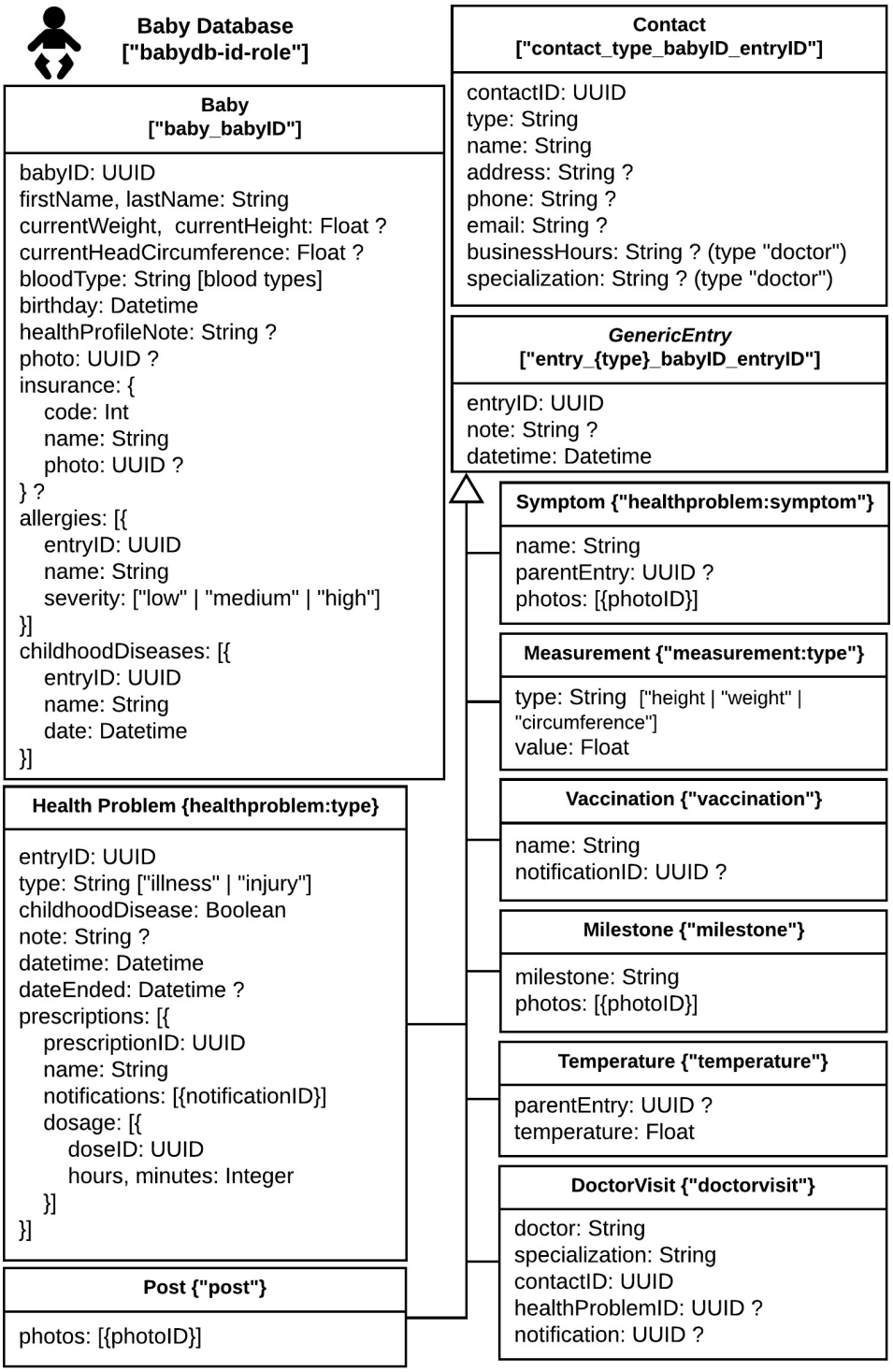
Obrázek 4.1: Schéma uživatelské databáze.



Obrázek 4.2: Schéma databází `common` a `local`.

žiště daného mobilního zařízení. Databáze se nikdy nesynchronizuje, a tak by tato data ani nemusela být v PouchDB uložena, ale zachování uniformního přístupu k datům vede k jednoduššímu kódu. V současné verzi databáze ukládá jenom jeden typ dokumentů, a sice metadata o aktivních lokálních notifikacích, které si přes aplikaci uživatel nastavil.

Na posledním databázovém diagramu na obrázku 4.3 je schéma databáze profilu dítěte. Šipka na tomto diagramu značí šablonu, kterou sdílí všechny asociované typy dokumentů, a její rozšíření. Hlavní dokument `baby` obsahuje základní informace o dítěti, a také několik dalších objektů, které vlastní dokument nepotřebují. Zbytek jsou jednotlivé záznamy, které mají všechny velmi podobnou strukturu. Na diagramu chybí ještě neměnná pole `authorName` a



Obrázek 4.3: Schéma databáze dítěte.

`createdBy`, která mají téměř všechny objekty.

Databáze `files` a `_users` z diagramu 3.2 nespravuje aplikace, nýbrž CouchDB, proto zde jejich schéma neuvádíme.

Identifikátory dokumentů

Každá entita v databázi, která je složená z více než jedné primitivní hodnoty, má vlastní identifikátor, který je unikátní napříč celým systémem. Díky tomu, že pravomoc vkládat objekty do databází je distribuována mezi všechny uzly v systému, je potřeba, aby se nemohlo stát, že dva odlišné objekty dostanou stejný identifikátor. Za tímto účelem je používán systém pseudo-náhodně generovaných UUID, u nichž je pravděpodobnost kolize prakticky nulová.

Každý dokument v CouchDB musí mít také pole `_id`, pomocí kterého je databáze indexována a který představuje jediný způsob vyhledávání. Hodnota tohoto pole je v naší aplikaci odlišná od identifikátoru entity, kterou dokument popisuje, neboť se v praxi velmi efektivní mít v tomto poli co nejvíce informací. Takto se minimalizuje nutnost vyhledávat pomocí hodnot, které jsou v těle dokumentu, což je neefektivní.

Databáze ukládá dokumenty jako B+ strom s řazením podle abecedy [19] a nejefektivnější využití této datové struktury je použití klíče, ve kterém postupují zleva doprava obecnější informace k méně obecným, podobně jako třeba cesta k adresáři v souborovém systému. Vyhledávání v databázi je možné nejen podle přesné shody, ale také pomocí rozpětí, stačí tedy specifikovat abecední začátek a konec dokumentů, které nás zajímají. Proto například v klíči pro záznamy ukládané v databázi dítěte máme na začátku typ a možné podtypy dokumentu, pak identifikátor dítěte a až nakonec identifikátor samotného objektu, který dokument představuje.

Ještě lepší by bylo, kdybychom mohli do identifikátoru záznamu vepsat i datum a čas pro ještě jednodušší vyhledávání, to už ale bohužel nezapadá do hierarchické struktury ID. Musíme si vybrat, zda chceme vyhledávat podle typu, nebo podle data. Datum tak musí zůstat jen v těle dokumentu a vyhledávání pomocí časového rozmezí bude vyžadovat přečtení obsahu.

4.2 Back-end a komunikace aplikace se serverem

V předchozí kapitole jsme rozhodli, že serverová aplikace bude tvořena především databází CouchDB a jejím systémem replikace, který přenáší data mezi mobilní databází a serverovou. Prostředníkem v této komunikaci je webový server napsaný v Javascriptu, který funguje jako proxy a provádí automatické řešení konfliktů. Zajišťuje také několik služeb, které databáze sama implementovat nedokáže. Tato sekce popisuje implementační rozhodnutí a řešení týkající těchto dvou back-endových komponent.

4.2.1 Uživatelský přístup a zabezpečení profilů

Jedna z nejdůležitějších otázek se týká zabezpečení dat v serverové části databáze. Z diagramu 3.2 vyplývá, že mezi jednotlivými databázemi a uživateli existuje vazba M:N a je tedy potřeba implementovat mechanizmus, který tento vztah bude podporovat. Je také potřeba zajistit, aby k příslušným databázím měl přístup pouze přihlášený uživatel. Úkol je o to složitější, že musíme zabezpečit i replikační protokol, jehož fungování nemáme pod vlastní kontrolou. Náš druhotný zájem je, aby se uživatel nemusel přihlašovat do mobilní aplikace při každém použití a zároveň aby aplikace nemusela heslo ukládat v nezměněné podobě.

Systém zabezpečení pro přístupu k datům na serveru nemůže předpokládat, že se k němu připojuje právě mobilní aplikace. Získat zdrojový kód mobilní aplikace spolu s daty, která jsou v zařízení uložena, je pro útočníka poměrně jednoduchá záležitost, pokud může se zařízením libovolně manipulovat. Stejně jednoduché je pak komunikovat se serverem a vydávat se za aplikaci. Všechny kroky ověřování oprávnění tedy musí probíhat na serveru stejně jako v aplikaci.

Při pokusu o přístup k libovolné databázi je potřeba ověřit, že platí následující dvě skutečnosti:

1. Uživatel je přihlášený a může k databázi přistoupit (autentizace).

2. Pokud zpracováváme požadavek na vytvoření, změnu či smazání dokumentu, uživatel má oprávnění tuto změnu provést v dané databázi provést (autorizace).

K implementaci prvního kroku byl využit protokol OAuth, se kterým CouchDB umí sama pracovat. Výhoda OAuth spočívá v tom, že umožňuje ověřovat identitu uživatele s každým požadavkem, aniž by k tomu bylo potřeba uživatelské heslo. V mobilní aplikaci tak můžeme ukládat jen OAuth klíče, které uživatel dostane jednorázovým přihlášením. Tyto klíče je možné po čase zneplatnit a donutit uživatele se přihlásit znovu, aniž by byla narušena platnost původního hesla. OAuth jako autentizační protokol kromě ověření uživatele umožňuje také identifikaci aplikace, přes kterou se uživatel připojuje. V našem případě existuje pouze jedna, a tak tento aspekt nevyužíváme.

O zajištění autentizace se částečně stará zabudovaná implementace OAuth v CouchDB. Uživatelé systému jsou ukládáni v databázi `_users`, kde má každý z nich vlastní dokument. Pokud do tohoto dokumentu uložíme při vytváření uživatele navíc OAuth klíče a nakonfigurujeme databázi, aby OAuth používala, je schopna autentizovat uživatele odesílajícího požadavek. Informace o uživateli jsou databází vloženy do kontextu požadavku, kde s nimi můžeme nadále pracovat.

Autorizace je vyřešena pomocí design dokumentů databáze, které jsme uvedli v minulé kapitole. Tyto dokumenty umožňují měnit chování databáze a

jsou schopny spouštět uživatelský kód v odpovědi na nějakou předem určenou událost. Do každé databáze uživatele i dítěte je při vytvoření vložen design dokument, který se spustí před vykonáním jakéhokoli požadavku na změnu. Ten zkontroluje, zda má uživatel dostatečná oprávnění tuto změnu provést; pokud nemá, požadavek je zamítnut a změna se neprovede. Oprávnění se odvíjí od toho, jaké role má uživatel definované v databázi `_users`. Seznam těchto rolí je dostupný v kontextu, který je databází vložen do informací o požadavku při vyhodnocování funkce.

Zatím jsme mluvili pouze o požadavcích směřujících přímo na databázi. Ostatní případy komunikace, počínaje přihlášením do aplikace a konče exportem do PDF, ale používají úplně stejný systém autentizace i autorizace, jen ověřování provádí sám webový server za pomoci dat uložených v databázi.

Proxy a OAuth

Kombinace OAuth a prostředníka v komunikaci vytváří jeden nepříjemný problém. Protokol byl navržen tak, aby pokud možno chránil komunikaci před útokem typu „man-in-the-middle“, tedy vložení třetí strany mezi dvě komunikující entity. Bohužel přesně tento druh činnosti náš proxy server provádí. Jelikož je v OAuth podpisu obsažena informace o odesilateli i cílová adresa požadavku, nelze požadavek jednoduše přijmout, prozkoumat a přeposlat jinam se stejným podpisem, neboť ten už bude neplatný.

V současnosti tento problém aplikace řeší tak, že každý přijatý požadavek server autentizuje sám pomocí klíčů, které si vyžádá od databáze. Pokud je původní podpis správný, vygeneruje nový podpis pro databázi a požadavek přepošle.

Toto řešení není elegantní a už od pohledu je méně efektivní, neboť je zde o požadavek na databázi víc, ale testování dosud neodhalilo potřebu ho měnit. Je to však oblast jistého technologického dluhu, který bude potřeba v budoucnu splatit implementací lepšího řešení.

4.2.2 Řešení konfliktů

Konflikt v databázi vzniká, kdykoli přijdou dva požadavky na změnu stejného dokumentu. V jazyce CouchDB je to vždy, když přijde požadavek na změnu revize dokumentu, která už není tou nejnovější. Databáze v takovém případě vrátí chybový kód `409` a historie revizí se rozvětví, nicméně „oficiální“ verze bude náhodně vybrána databází. V předchozí kapitole jsme rozhodli, že nebudeme dbát na stoprocentní konzistenci dat za existence konfliktních modifikací, neboť to by vyžadovalo intervenci uživatele. Úplně ignorovat však problém také nemůžeme.

Mnoho konfliktních změn totiž lze vyřešit automaticky jednoduchým algoritmem podobně, jako to dělají například verzovací systémy pro správu zdrojového kódu. Jde o případy, kdy jsou modifikovány různé části dokumentu (jiné

klíče v JSON reprezentaci). I v případech, kde se změny týkají stejné hodnoty, můžeme ponechat nejnovější verzi, která bude často tou fakticky nejaktuálnější. Pomáháme si tím, že každý dokument obsahuje pole `modified`, což je časové razítko poslední modifikace. Toto pole mají i případné vnořené objekty.

Výsledný algoritmus pro slučování verzí dokumentů funguje následovně:

1. Při přeposílání požadavku z mobilní aplikace na serverovou databázi a následném přeposílání odpovědi zpět aplikaci zkontroluje proxy server, zda výsledkem modifikace není kód `409`.

2. Pokud vznikl konflikt a databáze vrátila `409`, ID dokumentu je vloženo do fronty ke sloučení.

3. Samostatný proces si vyzvedne z fronty identifikátor dokumentu a dotáže se databáze na čísla revizí, která jsou v konfliktu. Pokud je v nějaké z nich dokument označen jako smazaný (`_deleted`), pak bude výsledek také smazán a algoritmus tímto skončí.

4. Oficiální revize je sloučena s konfliktní. Na každé pole dokumentu se aplikují následující pravidla:

   - Jsou-li obě hodnoty polí stejné, nic se nezmění.
   - Pokud jsou hodnoty odlišné a obě mají primitivní typ (číslo nebo řetězec) nebo pokud mají odlišné a neporovnatelné typy (například objekt a pole), použije se hodnota v novější verzi podle pole `modified`.
   - Je-li hodnotou vnořený objekt, použije se výsledek rekurzivního volání tohoto algoritmu.
   - Má-li hodnota typ pole, výsledkem je sloučení obou polí. Objekty uvnitř pole jsou opět sloučeny tímto algoritmem. Pokud v novější revizi nějaký prvek pole chybí, výsledek ho neobsahuje. Pokud chybí ve starší verzi, do výsledku je zahrnut.

   Pokud je konfliktních revizí víc, tento krok se opakuje pro každou z nich.

Na konci procesu máme sloučený dokument. Ten uložíme do databáze jako úplně novou revizi a ostatní smažeme. Jakmile se nová verze do databáze dostane, automaticky se replikuje do mobilní aplikace úplně stejným způsobem, jako bychom dokument modifikovali jakoukoli jinou cestu.

4.2.3 Speciální případy komunikace

V této sekci popíšeme procesy, které jsou složitější než jedna databázová operace. U všech těchto případů užití platí, že je nutné, aby byla aplikace připojena k internetu a schopna okamžité komunikace se serverem, a tedy tvoří

výjimky z požadavku na offline použitelnost aplikace. Pokud připojení k internetu není k dispozici, skončí proces chybovou hláškou na straně uživatele.

Speciální případy komunikace jsou implementovány pomocí endpointů serverového REST API. Autentizace požadavků na toto API probíhá za pomocí databáze (viz. sekce 4.2.1). Server si od databáze vyžádá informace o uživateli, od něhož požadavek je, a sám ověří, zda je OAuth podpis požadavku platný a zdá má uživatel příslušná oprávnění.

Vytvoření nebo smazání profilu dítěte

Pro vytváření a případné mazání profilů dětí slouží endpoint `/babies`. Po zavolání tohoto endpointu s požadavkem na vytvoření nového dítěte se nejprve vytvoří nová databáze pro rodiče spolu s databází pro opatrovníka (viz. sekce 3.3.1). Do obou databází je vložen design dokument, který zajišťuje ověřování autorizace ke změnám a mezi databázemi je nastavena kontinuální synchronizace. Uživateli, jenž profil vytváří, je v systémové databázi `_users` přidána role vlastníka nového dítěte a zároveň je mu udělen přístup k rodičovské databázi prostřednictvím dokumentu `_security`.

V posledním kroku je nově nabyté oprávnění zapsáno do osobní databáze uživatele, odkud se o něm replikací dozví mobilní aplikace. Jakmile tam aktualizovaný dokument s oprávněním dorazí, aplikace začne se synchronizací nové rodičovské databáze. Po prvním úspěšném provedení této replikace je proces dokončen.

Při mazání profilu je napřed všem uživatelům odebrán přístup k databázi dítěte, což způsobí, že ji mobilní aplikace přestane databázi replikovat a smaže svou uloženou kopii. Server pak smaže i vlastní databáze, které profilu patřily.

Odesílání a přijímání pozvánek

Tento proces je z uživatelského pohledu popsán na diagramu aktivit 1.1. Diagram se po odeslání pozvánky větví na dva případy:

- Je-li cílový uživatel již registrován, pozvánka se mu uloží do osobní databáze a replikuje do mobilní aplikace, kde ji může přijmout. Odeslaná pozvánka se zapíše do osobní databáze i zvoucímu uživateli, aby ji v aplikaci mohl případně zrušit.

- Pokud cílová e-mailová adresa nepatří žádnému registrovanému uživateli, uloží se informace o ní pouze do databáze zvoucího uživatele a do databáze `_users`. V ní je design dokument, který umožňuje zavolat dotaz na všechny takto nedoručené pozvánky. Při každé registraci nového uživatele je výsledek tohoto dotazu zkontrolován, a pokud je nalezena nějaká pozvánka, která má být doručena nově registrovanému uživateli, je proces v okamžiku registrace dokončen. Pozvánka tak již čeká na nového uživatele v aplikaci hned, jak do ní poprvé vstoupí.

Přijmutí pozvánky cílovým uživatelem je relativně jednoduchý proces: pozvánka se vymaže a nahradí ji udělený přístup. Postup udílení přístupu je stejný jako u vytváření nového dítěte s tou výjimkou, že jsou příslušné databáze již vytvořené. Odebrání přístupu spočívá v tom, že se cílovému uživateli odeberou práva na příslušnou databázi a zároveň se informace o přístupu smaže z databáze uživatele, na což mobilní aplikace zareaguje smazáním všech dat souvisejících s daným dítětem.

Export dat do PDF

Export dat do PDF dokumentu se provede zavoláním endpointu `/report` s požadovanými sekcemi, které mají být ve výsledném dokumentu zahrnuty (ty si uživatel navolí v UI aplikace), načež aplikační server přečte celý obsah rodičovské databáze příslušného dítěte. Samotné PDF je generováno skrze šablonu v jazyce HTML a CSS pomocí knihovny Jsreport [20] a odesláno na cílový e-mail jako příloha.

4.2.4 Výkonová optimalizace databáze

Zátěžové testování v průběhu vývoje odhalilo, že příliš mnoho běžících kontinuálních dokáže server zahltit. Tento problém se začal objevovat už u řádově stovek replikací, což bylo pro nás nepřijatelné vzhledem k tomu, že každý vytvořený profil dítěte spouští minimálně jednu mezi databází rodiče a databází opatrovníka.

Mechanizmus kontinuální replikace v CouchDB funguje tak, že se kopie jedna druhou opakovaně ptají na změny, a to i v případě, že v databázích už dlouhou dobu žádné změny neproběhly. Server je pak zahlcen prázdnými a zbytečnými dotazy.

Tuto komplikaci jsem vyřešil tak, že je na serveru periodicky spouštěna operace, která zkontroluje seznam aktivně se replikujících databází. U těch, které nezaznamenaly žádnou změnu v nějakém předem definovaném časovém úseku, je replikace vypnuta. Ve chvíli, kdy proxy server zaznamená nový požadavek od běžící mobilní aplikace na nějakou neaktivní databázi, je automaticky zařazena zpět do aktivních a replikace jí je opět zapnuta. Takto se minimalizuje počet zbytečných dotazů a zároveň uživatelé nepřijdou o žádné aktualizace.

Toto řešení se ukázalo být dostatečné pro předpokládaný počet aktivních uživatelů v řádu stovek. Pokud se v budoucnosti ukáže, že i tak probíhá příliš mnoho kontinuálních replikací, můžeme se jich úplně zbavit a přejít na systém jednorázových, podobně jako to má mobilní aplikace (viz. sekce 4.3.1).

4.2.5 Zálohování databáze

Zálohování databáze je pro nás triviální záležitost. CouchDB ukládá každou databázi do vlastního souboru, a tak stačí zkopírovat všechny tyto soubory,

které se všechny nachází v jediném adresáři. Pro obnovu ze zálohy je stačí zkopírovat zpátky – CouchDB ukládá kromě dat i celou svou konfiguraci jako uživatele, oprávnění, nastavení replikací apod. v databázích. Jednou denně je tedy na serveru spouštěn skript, který celou datovou složku zkomprimuje a zkopíruje do nakonfigurované destinace. Poté smaže všechny zálohy starší než nastavená doba uchovávání.

Existuje ještě jedno řešení, které je sice konfiguračně náročnější, ale zato elegantnější. Stačí vytvořit repliku každé databáze na odděleném serveru a spustit kontinuální synchronizaci. Takto mohou být data zálohována průběžně, ne jen jednou denně. V produkci z provozních důvodů ale používáme metodu kopírování souborů, protože druhý server s instalací CouchDB je dražší na provoz než jednoduché úložiště.

4.3 Implementace front-endu

V této sekci popíšeme implementaci hlavní části celého projektu, tedy mobilní aplikace jako takové. Podobně, jako jsme to udělali u serverové části, se zde podíváme na celkovou architekturu kódu a popíšeme největší vyřešené překážky. Je potřeba mít na paměti, že mobilní aplikace je násobně rozsáhlejší – jak v objemu kódu, tak v čase, který byl na implementaci vynaložen – než server. Proto zde pokryjeme menší část celku než v předchozí sekci.

Při psaní kódu mobilní aplikace jsem se snažil především docílit co největšího oddělení zodpovědností jednotlivých komponent a zabránit zbytečné duplikaci kódu. Nalezení vhodných abstrakcí a postupů jsem věnoval velké množství času, neboť mým cílem bylo vytvořit sadu principů, jakési „best practices“, které by bylo možné využít i při vývoji jiných aplikací. Zde jsou nejdůležitější výsledky mého snažení.

4.3.1 Mobilní databáze

Pro ukládání dat v mobilní aplikaci jsme v minulé kapitole zvolili PouchDB [21], což je implementace všech hlavních funkcí CouchDB v jazyce Javascript (CouchDB je psána v Erlangu). Databáze je v aplikaci používána podobně jako na serveru, s tím rozdílem, že k ní nemusíme přistupovat jako ke vzdálenému uzlu přes HTTP, ale můžeme její funkce volat napřímo. Díky systému replikací je kód, který ukládá nebo modifikuje data, velmi jednoduchý: stačí změnu provést v lokální databázi a systém se postará o zbytek.

PouchDB v prostředí React Native ale nelze použít přímo, protože aplikace neumí poskytnout ani jednu z možností fyzického ukládání dat do paměťového úložiště, kterou PouchDB podporuje. Základní mechanizmus pro ukládání dat v React Native je modul `AsyncStorage`, což je jednoduchá textová databáze typu klíč-hodnota. Ten sice není z nejrychlejších, ale pro naše účely je dostačující. Aby ho mohla PouchDB použít, je potřeba použít speciální adaptér [22].

Podobně jako na serveru, i zde jsem narazil na problém s počtem kontinuálních synchronizací. Aplikace průměrného přihlášeného uživatele s dvěma profily dětí musí udržovat repliky čtyř databází a už tento počet byl problematický. Neustálý polling, který databáze prováděla, se negativně podepisoval na výkonu celé aplikace. Bylo proto potřeba vymyslet efektivnější systém.

Nejlepší by samozřejmě bylo, aby si databáze vyměňovaly data pouze v případě, že mají k dispozici nějaké nové změny. To ovšem vyžaduje obousměrnou komunikaci prostřednictvím socketů a podpora pro sockety mezi CouchDB a PouchDB zatím neexistuje. Druhé nejlepší řešení je, aby se aplikace jen jednou za čas dotázala serveru, jestli pro ni jsou nějaké změny nachystané. Ani to pro nás není příliš praktické, neboť databáze jsou samostatné entity a k implementaci takového systému bychom museli rozšířit nejen replikační logiku v aplikaci, ale také serverovou část. Tento mechanizmus je ale na rozdíl od socketů proveditelný, a tak zůstává jako možná rezerva při budoucích problémech s výkonem.

Řešení, které nakonec aplikace používá, je následující algoritmus. Kontinuální synchronizace v PouchDB je vypnutá a není používána. Databáze, které mají být replikovány, jsou uloženy ve frontě. S určitou konfigurovatelnou frekvencí je v aplikaci vysílán signál k provedení jedné replikace. Po obdržení tohoto signálu modul `database.js`, který databáze spravuje, spustí jednorázovou („one-shot“) replikaci na tu databázi, která je ve frontě jako první. Po úspěšném dokončení synchronizace je zařazena zase na konec fronty. Kdykoli v pauze mezi replikacemi aplikace zaznamená lokální zápis, daná databáze je vložena na začátek fronty bez ohledu na její aktuální pozici. Takto je eliminována zbytečná prodleva při nahrávání změn na server.

Kromě zajišťování synchronizace poskytuje databázový modul jednoduché rozhraní pro zakládání a mazání databází a přímočaré hledání dokumentů, které se mohou nacházet v různých databázích. To usnadňuje načítání dat například pro časovou osu, kde se zobrazují data všech dětí najednou.

4.3.2 Nativní navigace

Navigace je ve vývojářské komunitě React Native horkým tématem již od vzniku knihovny. Konkrétně se jedná o otázku, jak docílit toho, aby byla aplikace strukturou obrazovek a přechody mezi nimi nerozeznatelná od aplikace vyvíjené nativně. Oba mobilní operační systémy řeší navigaci trochu jinak, nicméně navigace ve většině aplikací pro určitý systém vypadá úplně stejně. To je dáno tím, že aplikace používají standardní navigační prvky poskytované vývojovým prostředím dané platformy. Hybridní technologie jako React Native musí řešit, jak jejich využití zakomponují do vlastní filozofie, nebo nabídnout dostatečně dobrou alternativu. Otázka nativní navigace je pro nás silně spojena s požadavkem na respektování UX konvencí platformy (viz. sekce 1.3 a 2.1.1).

V době, kdy začínala implementace aplikace, se navigace v React Native nacházela v neutěšeném stavu. Oficiální podporované řešení navigace byla komponenta `Navigator`, ta byla ale bohužel pouze imitací nativního chování a z uživatelského pohledu působila nekvalitně. Byla navíc obtížně použitelná. V době psaní práce již existuje projekt React Navigation, se kterým se pracuje mnohem snadněji. Pořád je to ale náhražka, která musí animovat rozhraní na stejném vlákně, na němž je vykonáván kód aplikace, a tak nepůsobí dostatečně plynule.

Situaci zachránila knihovna React Native Navigation vyvíjená firmou Wix [23]. Ta poskytuje Javascriptové rozhraní k opravdovým nativním navigačním řešením na obou platformách. Na iOS tak bylo možné použít `UINavigationController`, zatímco na Androidu jsou to standardní přechody mezi aktivitami. V kódu knihovny bylo jen potřeba udělat jednu malou změnu, aby bylo možné implementovat velké tlačítko pro přidání záznamu na spodní liště se záložkami v iOS verzi.

Realizace nativní navigace však není vůbec tak jednoduchá, jak by se mohlo zdát. React Native totiž nepočítá s tím, že by kontext aplikace měl přesáhnout jednu nativní obrazovku (jeden `UIViewController` na iOS nebo jednu aktivitu na Androidu). React Native Navigation je tím pádem obrovský projekt, jehož použití s sebou nese několik vlastních problémů. Žádný z nich není natolik vážný, abych kvůli němu musel knihovnu zavrhnout, i tak jsou ale některé věci komplikovanější, než by byly bez ní.

Jeden z těchto problémů je například to, že React Native Navigation mění způsob inicializace aplikace a je potřeba měnit nativní kód, který je vygenerován automaticky a který by správně měněn být neměl. Při každé aktualizaci na novější verzi React Native je tak aktualizačním skriptem přepsán zase zpět. Knihovna navíc znemožňuje použití tzv. „hot reloading“, což je velmi užitečný vývojový nástroj, pomocí nějž není potřeba restartovat běžící aplikaci po každé změně kódu. Kromě toho jsem v průběhu vývoje narazil na několik podivných časově závislých chyb a pádů aplikace na určitých kombinacích verze OS Android a typu mobilního zařízení, jež zcela jistě vznikají kvůli použití nativní navigace. Nejedná se o nic vážného, ale stabilita aplikace není vždy stoprocentní. Lze jen doufat, že novější verze React Native Navigation tyto problémy vyřeší.

4.3.3 Redux a Redux-Saga

Další zásadní technologií, kterou naše aplikace, používá, je Redux. Redux je knihovna, ale především filozofie vývoje složitých aplikací s velkým množstvím vnitřního stavu, která vychází z principů funkcionálního programování [24]. Aplikace, které správně používají Redux, drží všechen svůj vnitřní stav – stav navigace, dat, komunikace se serverem – na jednom místě. Podoba uživatelského rozhraní a obsah paměti v jakýkoli moment je pak deterministickou funkcí tohoto stavu. Stav lze navíc měnit jen pomocí tzv. akcí. Platí zde po-

dobný princip: určitá kombinace aktuálního stavu a nějaké akce může vést jen na jeden konkrétní a pokaždé stejný stav.

Tento přístup se může zdát omezující a komplikovaný, jeho výhody jsou ale obrovské. Pokud v systému vznikne nějaká chyba, je triviální ji dohledat, neboť jedinou možnou příčinou je kombinace počátečního stavu a sekvence proběhlých akcí, kterou můžeme zaznamenávat a zpětně přehrát. Kód je navíc lépe testovatelný a přehlednější.

Kromě Reduxu aplikace BabyDiary používá i jeho rozšíření Redux-Saga [25], což je jeden z několika způsobů, jak do výše popsané metody zakomponovat asynchronní operace. Umožňuje definovat „ságy“, což jsou funkce, které se spouští v reakci na nějakou akci. Mohou provádět asynchronní volání a posílat do systému další akce. Detailní popis fungování Redux-Saga je nad rámec této práce, nicméně způsob, jakým ságy umožňují psát asynchronní logiku, značně napomáhá přehlednosti a čitelnosti kódu. Většina aplikační logiky v BabyDiary je realizována v ságách.

Aplikace používá Redux v největší možné míře. Ukázalo se velmi užitečné organizovat aplikační logiku, jako například inicializaci aplikace, modifikaci dat nebo komunikaci se serverem, jako řadu přesně definovaných a zapisovatelných akcí. Bohužel držet úplně všechen aplikační stav v Redux store je pro nás neproveditelné. Jednak by bylo příliš složité spojit Redux s automaticky probíhající replikací PouchDB, jednak kromě Javascriptu běží v aplikaci i množství nativního kódu, který na Redux nelze rozumně navázat. Přesto považuji přítomnost Reduxu a Redux-Saga v aplikaci za jeden z faktorů, který výrazně pomohl kvalitě a škálovatelnosti zdrojového kódu.

4.3.4 Tok dat v aplikaci

Tato sekce popisuje napojení mobilní databáze na uživatelské rozhraní. Cílem bylo vytvořit deklarativní systém, který se jednoduše používá, úplně abstrahuje interakci s databází a je plynule navázaný na způsob předávání dat v React Native.

React Native je postaven na myšlence jednosměrného toku dat v hierarchicky uspořádaném uživatelském rozhraní. To znamená, že kontext, podle kterého se nějaký kus rozhraní vykresluje, je zcela definován vlastnostmi, které dostane od svého rodiče (pole `props`), popřípadě vlastním vnitřním stavem, který ale může měnit jen on sám (pole `state`). Potomek nemůže přímo předávat data svému rodiči ani měnit jeho stav, protože o něm neví. Pokud chce rodič dostávat zprávy od svého potomka, musí mu explicitně předat funkci (callback) v `props`. Důvodem tohoto zapouzdřování je zajistit větší přehlednost i lepší testovatelnost kódu a eliminovat chyby pramenící z neočekávané interakce mezi komponentami.

Bylo potřeba vymyslet, jak do této hierarchie vložit dokumenty z PouchDB a jak zajistit, aby se při změně databáze nová data automaticky dostala na správná místa a spustilo se překreslování komponent, které je zobrazují.

Základní idea implementovaného systému stojí na tzv. „subscriptions“, neboli odběrech informací, pomocí nichž komponenta deklaruje, že má zájem o nějakou množinu dat z jedné nebo více databází. Jako příklad uveďme třeba obrazovku s detailem návštěvy lékaře. Ta bude mít jeden odběr pro dokument s informacemi o dítěti, jeden pro kontakt lékaře a ještě jeden pro dokument s návštěvou. Každá komponenta, která má být takto napojena na databázi, je obalena v prvku `<PouchConnectedComponent>`. Ten při inicializaci rozhraní odběry vytvoří, načte dokumenty a stará se o jejich aktualizaci v průběhu celého životního cyklu obalené komponenty. Jinak její vzhled ani chování nijak neupravuje.

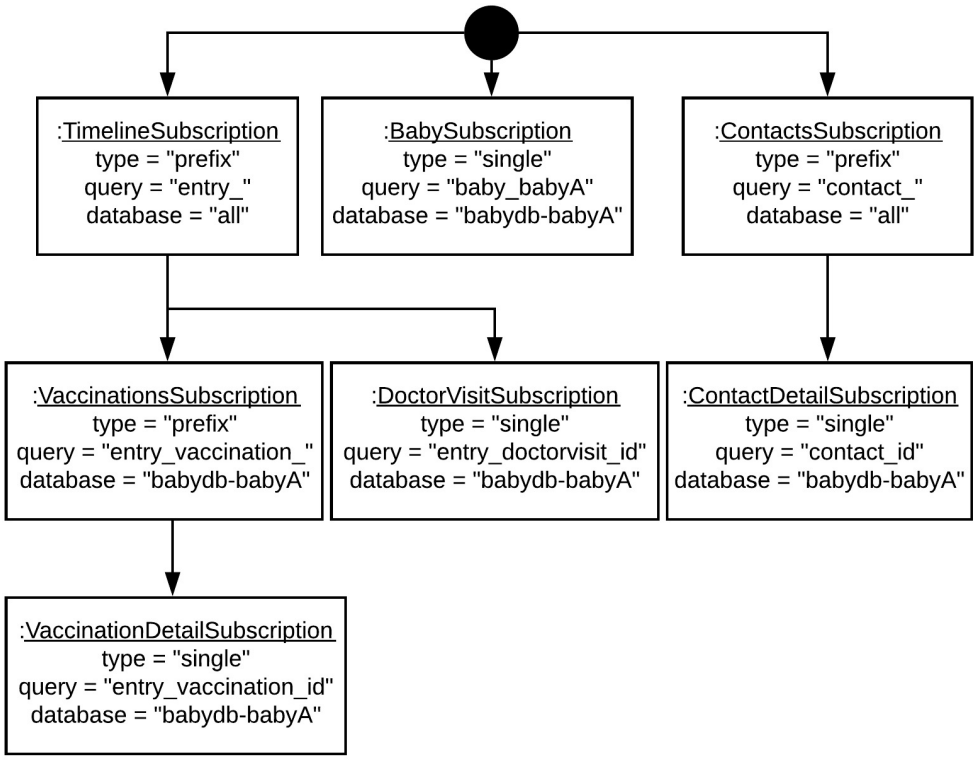
Všechny aktivní odběry jsou uloženy ve správci, který monitoruje změny ve všech databázích. Ve chvíli, kdy nějaká změna přijde, dostane správce identifikátor dokumentu, na němž byla modifikace provedena. Postupně se zeptá všech aktivních odběrů, zda je tato změna zajímá. Pokud ano, jsou dokumenty odběru načteny znovu a předány obalující komponentě jako nové `props`, což spustí překreslení odpovídající části rozhraní.

Je důležité připomenout, že `props` a tím pádem i dokumenty načtené z databáze jsou neměnné. Jediný způsob, jak docílit modifikace dat, je uložit novou verzi dokumentu do databáze. Tím odpadá nutnost jakkoli přemýšlet nad aktualizací jiných částí rozhraní než těch, kde modifikace vznikla, protože uložením do databáze se změna sama propíše celou aplikací. Ukládání dokumentů funguje pomocí akcí a ság popsaných v předchozí sekci. Komponenta uživatelského rozhraní jen pošle akci signalizující záměr vložit, změnit či smazat dokument a o zbytek se postará jiná část kódu.

Optimalizace dotazů

Takto popsaný systém sice dělá všechno, co od něj potřebujeme, ale je velmi neefektivní. Při běhu aplikace se totiž načítané množiny dokumentů překrývají a duplikují takovým způsobem, že jeden dokument může být z databáze načítán až sedmkrát, pokaždé jako součást jiného odběru. Takové množství zbytečných dotazů na databázi po každém zápisu se negativně podepisovalo na rychlosti aplikace v těch nejméně vhodných momentech.

Řešením je chytřejší režie při správě a načítání odběrů. Celý systém má na starosti modul `DocumentCache`, který registruje odběratele dat a seskupuje podobné odběry tak, aby počet dotazů na databáze minimalizoval. Metadata o odběrech mají přesně definovanou strukturu a načítání z databáze už není zodpovědností klientů, ale systému samotného. Podporovány jsou tři druhy dotazů: `all`, který načte všechny dokumenty, `single`, který načte jediný dokument se zadaným identifikátorem, a `prefix`, který načte všechny dokumenty, jejichž identifikátor začíná na zadaný řetězec. Kromě druhu a hledaného řetězce lze ještě specifikovat, zda se má hledat ve všech databázích, nebo jen v jedné. To umožňuje zavést následující dvě optimalizace:

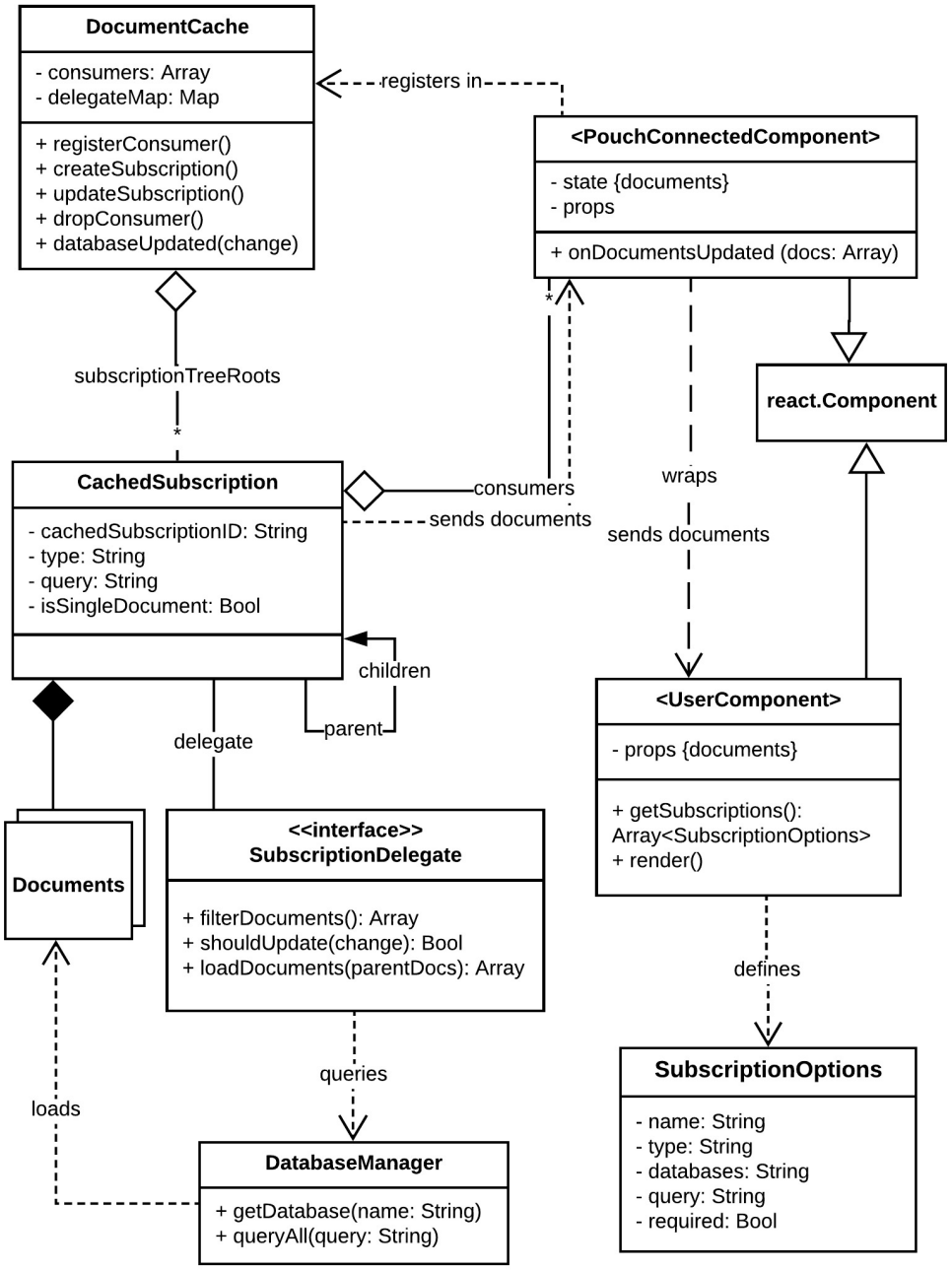


Obrázek 4.4: Příklad uspořádání odběrů ve stromové struktuře.

- Identické odběry jsou seskupeny do jednoho a načítány pouze jednou. Dokumenty se pak klientům jen nakopírují. Tím odpadá nutnost několikanásobného načítání například dokumentu `baby`, s nímž pracuje téměř každá komponenta.

- Je-li nějaká množina dokumentů zcela obsažena v jiném, již existujícím odběru, nenačítají se dokumenty z databáze, ale jsou zkopírovány z paměti. K tomu nám pomáhá i hierarchická struktura identifikátorů, viz. sekce 4.1. Například časová osa načítá velké množství záznamů a pro zobrazení detailu jednoho z nich stačí vzít už jednou načtený dokument.

Výsledkem těchto optimalizací je stromová, nebo spíše lesová struktura. Z databáze se načítají pouze data pro kořeny stromů, ostatním odběrům se dokumenty pouze předávají. Na obrázku 4.4 je ukázán příklad, jak může tato struktura při běhu aplikace vypadat. Například odběr `TimelineSubscription` načítá ze všech databází všechny dokumenty, jejichž identifikátor začíná na `entry_`. Tím pádem pro získání dokumentů pro `VaccinationsSubscription` stačí jen vyfiltrovat ty záznamy, které mají identifikátor začínající na `entry_vaccination_` a pochází z databáze `babydb-babyA`.



Obrázek 4.5: Diagram tříd navázání databáze na UI.

Celý systém pro načítání dat do komponent je ukázán na diagramu tříd na obrázku 4.5. K vlastnímu dotazování databáze slouží rozhraní `SubscriptionDelegate`, které je implementováno pro tři standardní typy odběrů popsané výše, nicméně klientský kód může definovat a registrovat vlastní implementaci pro nějaký jiný typ odběru nebo s vlastní filtrovací funkcí. Komponenta uživatelského rozhraní se zaregistruje do `DocumentCache` na začátku svého životního cyklu a po jeho skončení jsou její odběry zase zrušeny.

Výhoda této implementace spočívá v jednoduchosti jejího použití. Při psaní UI nás nemusí zajímat žádné databáze, aktualizace dat ani jejich kešování. Staráme se pouze o dolní pravý roh diagramu tříd, tedy definici informací o odběrech, což představovalo v aplikaci s několika desítkami komponent takto napojenými na data z databáze obrovskou časovou úsporu. Modul je otestovaný jednotkovými testy i provozem a použitelný i v dalších aplikacích postavených na PouchDB.

4.3.5 Práce s obrázky

Jak je již nastíněno v předchozí kapitole, uživatelem nahrané fotografie nechceme ukládat jako přílohy v PouchDB. Ta totiž stahuje všechna nová data bez ohledu na to, zda je už aplikace potřebuje, nebo ne. U textových dat to nevadí, protože jsou malá, obrázky jsou však něco jiného. Zbytečným stahováním fotografií nahraných někým jiným, které ani nemusí aplikace nikdy zobrazit, bychom mohli vyčerpat příliš velký objem dat. Proto v PouchDB aplikace ukládá jen textové identifikátory obrázků.

Samotná data vložených obrázků jsou ukládána v mobilním úložišti `AsyncStorage` a přenášena na server v pravidelném intervalu. Ten je ukládá do databáze `files` a servíruje v odpovědi na HTTP požadavek od autentizovaného uživatele. Obrázek s daným identifikátorem je neměnný a je možné ho pouze smazat, jinak bychom znovu od začátku řešili problém offline synchronizace. Jelikož je identifikátor obrázku generován zcela náhodně a je přístupný pouze přes dokumenty v databázích, které na něj odkazují, slouží zároveň jako jisté přístupové heslo.

Díky tomu, že aplikace umožňuje uživateli vložit do databáze dokument odkazující na obrázek, který ještě nebyl nahrán na server, se může stát, že u ostatních uživatelů bude nějakou chvíli obrázek chybět. To nám ale nevadí, neboť obrázky nejsou kritická data a po čase vše dorazí tam, kam má. Zobrazování uživatelem nahraných obrázků je implementováno komponentou `<LazyImage>`, které stačí předat identifikátor a ta obrázek automaticky načte z lokálního úložiště, pokud už byl někdy v minulosti stažen, nebo si ho vyžádá od serveru a zobrazí ve chvíli, kdy je stahování dokončeno.

4.4 Omezení a rizika

Na začátku kapitoly jsme řekli, že se na implementační rozhodnutí podíváme i kritickým okem. Několik omezení a vynucených kompromisů jsme již v textu uvedli, například problémy způsobené použitím React Native Navigation. Zde uvedeme ještě několik málo dalších.

Největším rizikem pro údržbu a další vývoj aplikace je rychlý posun a velká proměnlivost technologií, na nichž je aplikace postavená. React Native stále prochází překotným vývojem. Frekvence nových verzí knihovny se v průběhu vývoje snížila z jedné každé dva týdny na jednu měsíčně, v porovnání se stabilnějšími technologiemi je to však stále příliš rychlé. Aktualizovat React Native jako takový není tak obtížné, problém je s ostatními knihovnami, z nichž každá aktualizace rozbije alespoň jednu. Do budoucna je tedy potřeba počítat s nutností vynaložit o něco větší úsilí na to, aby byla aplikace stále aktuální. Samozřejmě je možné ji „zakonzervovat“ na starších verzích knihoven, ale v takovém případě se může stát, že jednoho dne přijde nové mobilní zařízení nebo nová verze mobilního operačního systému, kde aplikace už nebude fungovat správně.

Další problém je způsoben tím, že jsme se vzdali požadavku na nepřetržitou konzistenci databáze a navrhli systém tak, že jsou data duplikována i fragmentována mezi různými databázemi. Tento problém nijak nedopadá na běžné uživatele aplikace – systém funguje, jak má. Může ale může způsobit obtížnější aplikační podporu, protože je složitější data v databázích najít a zjistit, která verze je ta správná. Potenciálním rizikem je i množství kontinuálních databázových replikací, které systém vyžaduje. Pokud bude mít aplikace příliš mnoho aktivních uživatelů, bude nejspíš potřeba znovu optimalizovat databázi a ještě více běžící replikace omezit.

Posledním problémem implementace je fakt, že v aplikaci úplně chybí mechanizmus pro datové migrace. Ten prozatím nebyl potřeba, neboť existuje jen jediná verze aplikace. Pokud se však stane, že se změní struktura dat ukládaných v databázi nebo bude potřeba upravit rozhraní mezi aplikací a serverem, bude potřeba vymyslet způsob, jak umožnit souběžné fungování starší i novější verze aplikace a zároveň zajistit, aby dříve registrovaní uživatelé mohli snadno přejít na novou verzi.

4.5 Shrnutí

Tato kapitola se zabývala implementací obou částí systému a popsala nejdůležitější a nejzajímavější použitá řešení. I přes omezení zmiňovaná v předchozí sekci byla aplikace implementována v celém požadovaném rozsahu. Všechny funkční i nefunkční požadavky byly splněny a implementace byla otestována a akceptována klientem jako vyhovující.

Vývoj probíhal o něco déle, než byl původní plán, což bylo způsobeno ná-

ročností práce s React Native i CouchDB, se kterými jsem neměl předchozí zkušenosti. Výsledkem je však nejen mobilní aplikace, která funguje na operačních systémech iOS i Android, používá nativní navigaci, je schopna fungovat offline a její další rozšiřování je velmi jednoduché, ale také znovupoužitelná architektura pro vývoj podobných aplikací. V poslední kapitole popíšeme nasazování serverové části aplikace a představíme testovací strategii obou částí systému.

Kapitola 5

Testování a nasazení

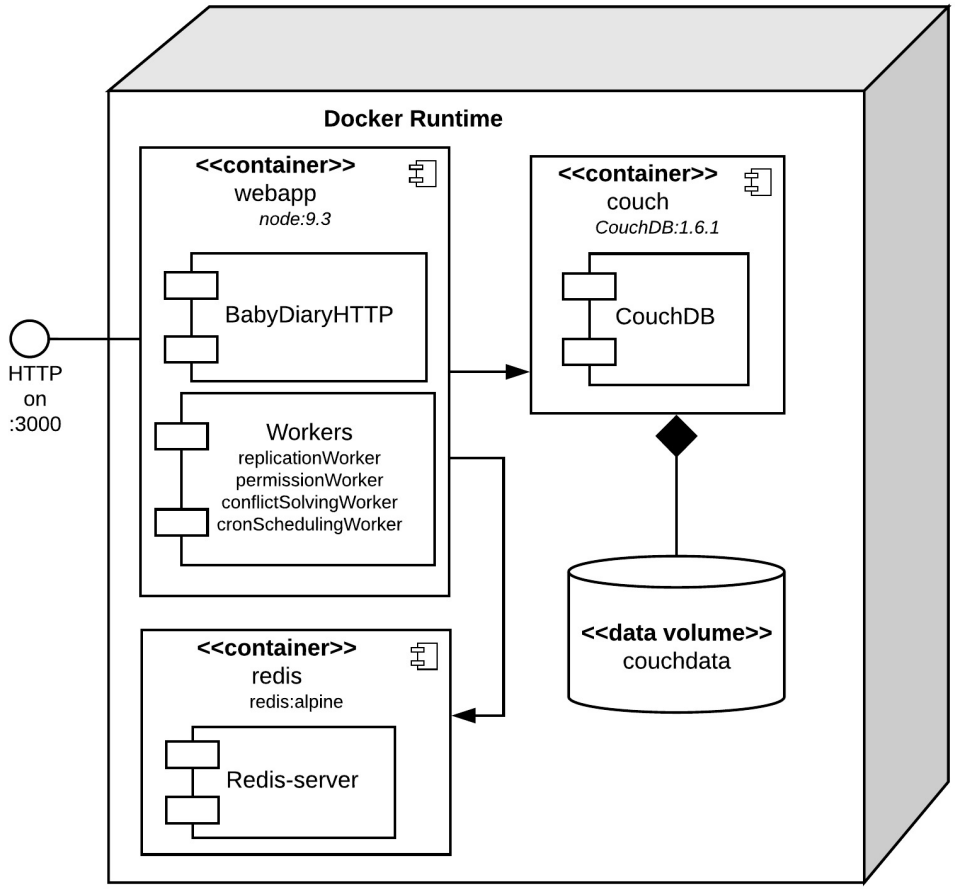
Tato kapitola stručně popisuje strategii a výsledky testování systému. V případě serverové části je otázka testování úzce spjata i s prostředím, ve kterém serverová aplikace a databáze běží, proto zde mluvíme i o způsobu jejich nasazování a propojení. Podrobný návod k instalaci aplikace pro testování i pro vývoj se pak nachází v příloze D.

5.1 Nasazení serverové části

Serverová část systému sestává ze tří hlavních komponent a několika vedlejších procesů, které musí kontinuálně běžet. Kromě databáze CouchDB a aplikačního HTTP serveru na Node.js je to ještě paměťová databáze Redis [26], která slouží jako fronta úkolů, a procesy pro řešení konfliktů v dokumentech, monitorování aktivních databází, nastavování oprávnění a zálohování databáze (viz. sekce 4.2). Těchto procesů a jejich závislostí je tolik, že z jejich instalace a spouštění ve správném pořadí se stal nelehký úkol.

Pro zjednodušení tohoto procesu jsem se rozhodl použít kontejnerový systém Docker [27], což je nástroj pro virtualizaci běhových prostředí. Docker umožňuje zabalit softwarový modul spolu se všemi jeho závislostmi, včetně operačního systému, do distribuovatelného kontejneru. Pro správný běh tohoto modulu je tak zapotřebí jen spustit daný kontejner. Docker má několik výhod oproti klasickému modelu virtualizace pomocí virtuálních strojů. Kontejnery neobsahují celý hostitelský operační systém, nýbrž jenom jeho nezbytnou část, a tak jsou fyzicky mnohem menší. Díky pokročilé technologii virtualizace navíc nemá použití kontejnerů negativní dopad na výkon serveru.

Nasazování serverové části BabyDiary je znázorněno na diagramu 5.1. Máme celkem tři samostatné Docker kontejnery, které spolu mohou komunikovat: jeden pro Redis, druhý pro CouchDB a třetí pro všechny procesy, které používají Javascriptovou platformu Node.js. Těmi jsou samotný HTTP server a čtyři procesy na pozadí. Kromě těchto tří kontejnerů je na diagramu znázorněn tzv. „data volume“, což je speciální kontejner, který je schopný per-



Obrázek 5.1: Schéma nasazení aplikačního serveru v Dockeru

zistentně ukládat data na hostitelském souborovém systému. Bez něj bychom při každém novém spuštění systému přišli o všechna data v databázi, neboť normální kontejnery nedokážou uchovat data po vypnutí.

O spouštění a propojování kontejnerů se stará nástroj Docker-compose, o správu běžících procesů uvnitř kontejneru `webapp` je to pak manažer procesů pro Node.js jménem PM2. Díky němu je instalace i spuštění celého serveru v prostředí Dockeru záležitostí jediného příkazu.

5.2 Automatické testy serveru

Chování serveru je úzce spjato s chováním databáze. Naprostá většina procesů, které aplikační server implementuje, má několik mezikroků komunikace s databází a velmi málo další logiky. Jednotkové testování bez použití databáze tak není příliš přínosné, protože bychom mohli maximálně ověřovat, že jsou všechny kroky prováděny ve správném pořadí, což je zřejmé na první pohled

ze zdrojového kódu. Všechny operace jsou navíc zcela asynchronní, s čímž je v klasických jednotkových testech problém.

Z těchto důvodů jsem se rozhodl testovat aplikační server společně s databází jako „black-box“ celek. Pro spuštění testů musí aplikační server i databáze běžet; testovací kód posílá na server požadavky a kontroluje odpovědi, popřípadě stav databáze. Správně bychom tedy tyto testy nazvaly integračními. Asynchronicita procesů je řešena čekáním určitého časového intervalu, než se výsledek operace dostaví.

Abychom mohli zakomponovat automatické testy do systému nasazování, musíme do našeho Dockerového prostředí přidat ještě další entitu, která bude testovací kód spouštět. Ta musí být spojena jak s kontejnerem `webapp`, tak s kontejnerem `couch`. Toho je docíleno pomocí mírně odlišné konfigurace pro testování než pro normální běh. Testovací prostředí obsahuje navíc čtvrtý kontejner, jenž má totožné prostředí jako `webapp`, ale jinou spouštěcí sekvenci a místo nastartování serveru začne provádět integrační testy. V testovací konfiguraci také není potřeba použít data volume, protože nám nevadí, že data z databáze po skončení běhu zmizí.

Serverová část obsahuje více než 60 aktuálních a udržovaných testů a pokrytí implementovaných procesů je téměř úplné. Jedinou významnou výjimkou je export do PDF, který testován není.

5.3 Testování mobilní aplikace

Automatické testování mobilní aplikace je složitější než testování serverové logiky, už jen protože největší část kódu se zabývá vykreslováním uživatelského rozhraní, jehož správnost se ověřuje jen obtížně. Samozřejmě i tady existují různé druhy speciálních testů, které se snaží tyto překážky překonat. Při vývoji BabyDiary jsem vyzkoušel celkem tři druhy automatických testů, které zde popíšu. Tyto pokusy ale nebyly příliš úspěšné, neboť ani jeden, s výjimkou skromných unit testů, nakonec neposkytoval dostatečné zhodnocení investovaného času.

5.3.1 Snapshot testy

React Native poskytuje možnost psaní tzv. „snapshot“ testů, jejichž úkolem je kontrolovat, že napsaný kód produkuje správnou strukturu uživatelského rozhraní po předání nějakých testovacích dat [28]. Běh testu spočívá v tom, že se testované komponentě předají data, následně je spuštěn vykreslovací systém React Native a výsledkem je kostra rozhraní, kde jsou opravdové nativní UI prvky nahrazeny pouze svými názvy. Kontrola správnosti výsledku se provede porovnáním této kostry s výsledkem nějakého referenčního (často prvního) běhu.

5.3.2 Jednotkové testy

Stejně jako pro serverovou část, i pro mobilní aplikaci lze napsat automatické jednotkové testy. Jejich síla je ale řádově nižší, protože už podle názvu se hodí hlavně pro testování oddělených modulů s minimem závislostí, jichž v programu orientovaném na UI nebývá tak mnoho.

Implementace BabyDiary několik málo jednotkových testů, které ověřují správnou funkčnost modulu `images.js` pro ukládání, nahrávání a stahování obrázků a modulu `database.js` pro správu aktivních databází. Tyto dvě komponenty jsou nejvíce testovatelné z celého projektu, protože nemají žádné vazby na okolí, které by se nedaly v testovacím prostředí nahradit.

5.3.3 Koncové testy

Obě mobilní platformy, jak iOS, tak Android, mají vlastní systém provádění koncových, neboli „end-to-end“ testů. Těmito testy se rozumí nějaký druh automatizace interakce s uživatelským rozhraní – simulace klikání a dívání se na obrazovku, zda je na ní vše, co tam má být. Pro nás je bohužel nešťastné, že jsou oba naprosto odlišné. Testovací kód bychom tak museli psát dvakrát.

Možným řešením je nástroj Appium [29], který k oběma způsobům testování poskytuje jednotné rozhraní. Appium funguje jako server, ke kterému existuje několik různých klientů pro různé programovací jazyky včetně Javascriptu. Klient serveru říká, se kterými částmi rozhraní má Appium interagovat, a server tyto příkazy vykonává na běžícím simulátoru. Interakce s UI prvky a kontrola jejich správnosti je závislá na tom, že se je podaří na vykreslené obrazovce správně identifikovat, což se ukázalo být nejsložitější stránkou použití Appia. Vyhledávání probíhá na obou platformách pomocí trochu jiných atributů, které je potřeba z React Native kódu dostat do fyzických nativních komponent. Podrobnější návod, jak toho dosáhnout, uvádí například [30].

5.3.4 Manuální testování

V konečném důsledku je nejdůležitějším testováním jakéhokoli systému testování v reálných podmínkách. V našem případě je to testování reálnými uživateli na opravdových mobilních zařízeních. Uživatelé z masa a kostí jsou jako jediní schopni doopravdy posoudit kvalitu výsledku.

Během implementace aplikace jsem měl k dispozici dedikovaného testera, který po každém mém větším zásahu do kódu (většinou po přidání každé nové funkcionality) aplikaci důkladně ozkoušel. Nakonec se tyto manuální testy, které postupně získávaly formální strukturu, staly jednou z nejdůležitějších částí projektu. Jenom díky důslednému testování všech případů užití po každé větší úpravě se podařilo odhalit desítky chyb, které by jinak odhalil až klient, nebo by se dokonce dostaly do produkce. Díky tomu nečekaly na klienta při odevzdání a schvalování žádná nepříjemná překvapení.

Závěr

Cílem této práce bylo navrhnout, vytvořit a otestovat implementaci mobilní aplikace pro systémy iOS a Android, do níž si rodiče mohou zaznamenávat zdravotní informace o svých dětech a důležité okamžiky v jejich životě. Nedílnou součástí aplikace měla být možnost sdílet data mezi uživateli. Koncept aplikace, požadované funkce a finální grafická podoba byly dodány klientem, na mne byl návrh struktury uživatelského rozhraní a technická realizace celého systému.

V rámci technického návrhu jsem se rozhodl, že bude aplikace postavena na nějaké z technologií pro multiplatformní vývoj. Po analýze nejvhodnějších možností byla vybrána knihovna React Native. Jelikož požadavky na aplikaci vynucovaly ukládání uživatelských dat v centrálním serverovém úložišti a jejich synchronizaci přes internet, součástí systému musela být i serverová část. Pro její realizaci byla vybrána dokumentová databáze CouchDB, především díky její schopnosti replikace a synchronizace. V mobilním zařízení pak funguje sesterská databáze PouchDB, která automaticky synchronizuje data se serverem. Složitější serverové funkce zajišťuje klasický HTTP server napsaný v Node.js.

Implementovaný systém splňuje všechny zadané funkční i nefunkční požadavky. Po dlouhém interním schvalovacím procesu u klienta byla aplikace vydána pro veřejnost na iOS App Store i obchod Google Play v prosinci roku 2017. V době psaní této práce je ještě příliš brzy na zhodnocení aplikace u reálných uživatelů v produkčním prostředí, akceptační kritéria byla však splněna.

Jedním z hlavních přínosů práce je vedle výsledného produktu také vytvoření podpůrných a znovupoužitelných softwarových modulů, které umožňují použití React Native a CouchDB/PouchDB na reálných projektech vyžadujících spolehlivost, stabilitu a dotaženost řešení. V tomto práce splnila můj vlastní cíl a jsem si jistý, že mi vytvořené postupy budou na budoucích projektech významně pomáhat.

Literatura

[1] Apple Inc.: App Store [online]. b. r., [cit. 2016-11-16]. Dostupné z: https://web.archive.org/web/20161116215649/https://developer.apple.com/support/app-store/

[2] Android Version Distribution History [online]. 2015-, [cit. 2016-11-16]. Dostupné z: https://www.bidouille.org/misc/androidcharts

[3] Nielsen, J.: Usability Heuristics for User Interface Design [online]. 1995, [cit. 2017-11-20]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

[4] Stehling, B.: PhoneGap vs Native [online]. 2013, [cit. 2017-11-22]. Dostupné z: http://blog.smallsharptools.com/post/69660117792/phonegap-vs-native

[5] Apple Inc.: Human Interface Guidelines – Navigation [online]. b. r., [cit. 2017-11-23]. Dostupné z: https://developer.apple.com/ios/human-interface-guidelines/app-architecture/navigation/

[6] Google: Material Design – Components [online]. b. r., [cit. 2017-11-23]. Dostupné z: https://material.io/guidelines/components/bottom-navigation.html

[7] Apache Foundation: Cordova Overview [online]. 2012-, [cit. 2017-11-28]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[8] AppBrain: PhoneGap / Apache Cordova [online]. 2018, [cit. 2017-01-06]. Dostupné z: https://www.appbrain.com/stats/libraries/details/phonegap/phonegap-apache-cordova

[9] Morony, J.: 8 Reasons Why I'm Glad I Switched to the Ionic Framework [online]. 2017, [cit. 2017-11-28]. Dostupné z:

https://www.joshmorony.com/8-reasons-why-im-glad-i-switched-to-the-ionic-framework/

[10] Yaskevich, A.: Apache Cordova: Is There A Future For It? [online]. 2017, [cit. 2017-11-28]. Dostupné z: https://www.scnsoft.com/blog/apache-cordova-is-there-a-future-for-it

[11] Xamarin Inc: Understanding the Xamarin Mobile Platform [online]. b. r., [cit. 2017-11-29]. Dostupné z: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/

[12] Xamarin Inc: Customers [online]. b. r., [cit. 2017-11-29]. Dostupné z: https://www.xamarin.com/customers

[13] Facebook Inc: React Native: A framework for building native apps using React [software]. b. r., [cit. 2017-11-30]. Dostupné z: https://facebook.github.io/react-native/

[14] Gamma, E.: *Design patterns : elements of reusable object-oriented software*. Reading, Massachusetts, USA: Addison-Wesley, 1995, ISBN 0-201-63361-2.

[15] Blackheath, S.; Jones, A.: *Functional Reactive Programming*. Shelter Island, New York, USA: Manning, 2016, ISBN 9781633430105.

[16] Brewer, E. A.: Towards Robust Distributed Systems. In *Symposium on Principles of Distributed Computing (PODC)*, 2000. Dostupné z: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[17] Realm [online]. b. r., [cit. 2017-12-01]. Dostupné z: https://realm.io/products/realm-database/

[18] Apache Foundation: CouchDB [software]. 2017, [cit. 2017-12-02]. Dostupné z: http://couchdb.apache.org/

[19] Anderson, J. C.; Lehnardt, J.; Slater, N.: *CouchDB: The Definitive Guide*. O'Reilly Media, 2010, ISBN 0596155891. Dostupné z: http://guide.couchdb.org/editions/1/en/index.html

[20] JSReport [software]. 2017, [cit. 2017-12-17]. Dostupné z: https://jsreport.net/

[21] PouchDB [software]. 2012-, [cit. 2017-12-05]. Dostupné z: https://pouchdb.com/

[22] Stock, C.: pouchdb-react-native [software]. 2016-, [cit. 2017-12-05]. Dostupné z: https://github.com/stockulus/pouchdb-react-native

[23] Wix: React Native Navigation [software]. 2015-, [cit. 2017-12-07]. Dostupné z: https://wix.github.io/react-native-navigation/

[24] Abramov, D.; Clark, A.: Redux [software]. 2015-, [cit. 2017-12-10]. Dostupné z: https://redux.js.org/

[25] Burzyński, M.: Redux-Saga [software]. 2016-, [cit. 2017-12-10]. Dostupné z: https://redux-saga.js.org/

[26] Redis Labs: Redis [software]. 2009-, [cit. 2018-01-02]. Dostupné z: https://redis.io/

[27] Docker Inc: Docker [software]. 2013-, [cit. 2018-01-02]. Dostupné z: https://docs.docker.com/

[28] Facebook Inc: Snapshot Testing With Jest [online]. b. r., [cit. 2018-01-07]. Dostupné z: https://facebook.github.io/jest/docs/en/snapshot-testing.html

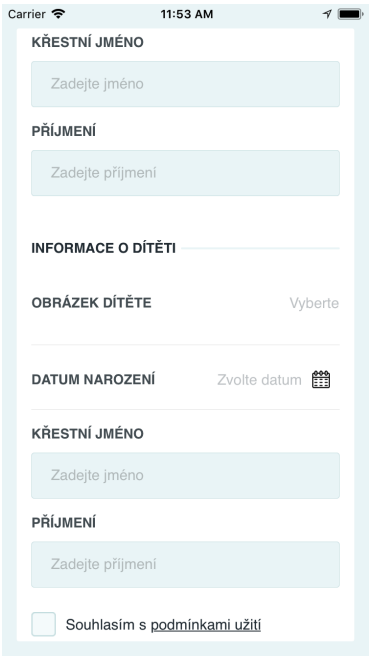
[29] Appium [software]. b. r., [cit. 2018-01-07]. Dostupné z: http://appium.io/docs/en/about-appium/intro/?lang=en

[30] Seibert, C.: Appium + React Native Quickstart [online]. 2017, [cit. 2018-01-07]. Dostupné z: https://chase-seibert.github.io/blog/2017/01/06/appium-react-native-quickstart.html

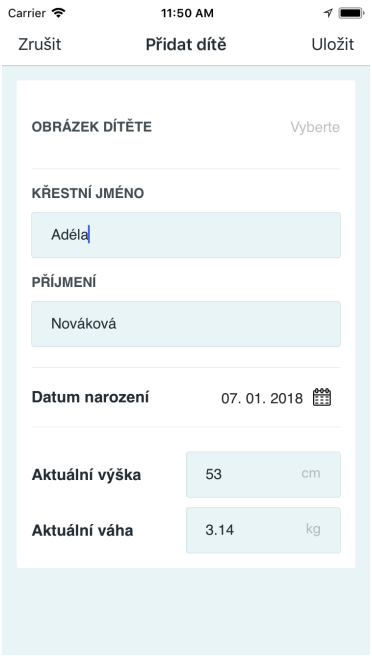
Příloha A

Snímky obrazovky

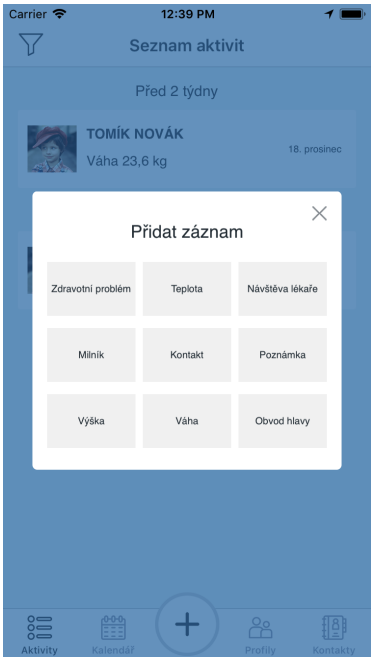
Obrázek A.1: Úvodní obrazovka (Android)



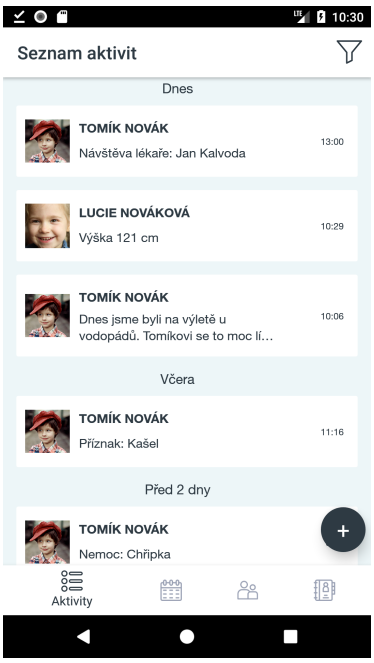
Obrázek A.2: Registrace uživatele (iOS)



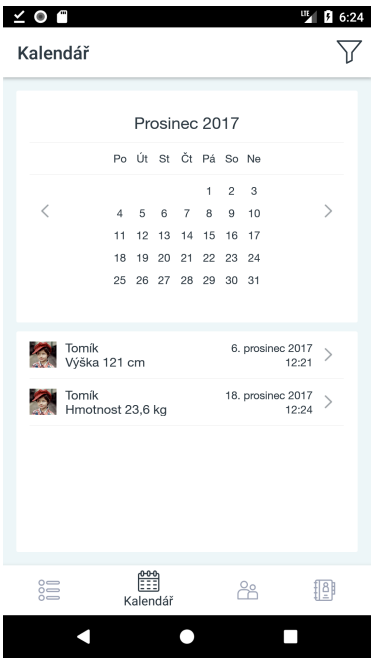
Obrázek A.3: Přidání dítěte (iOS)



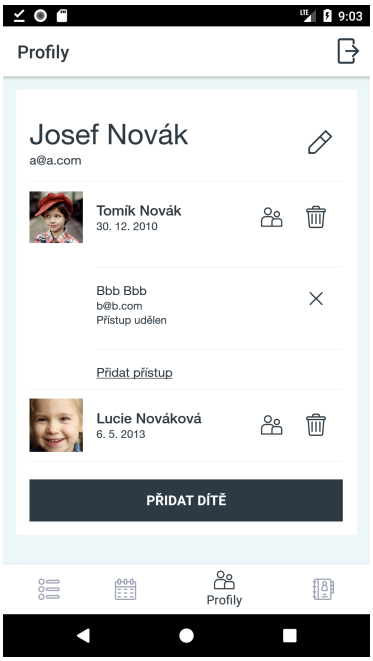
Obrázek A.4: Výběr typu přidávaného záznamu (iOS)



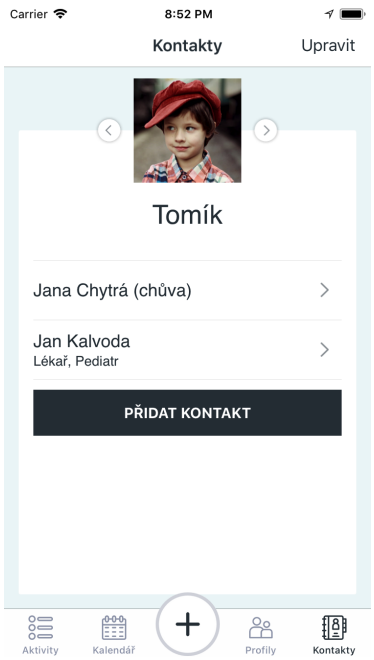
Obrázek A.5: Časová osa (Android)



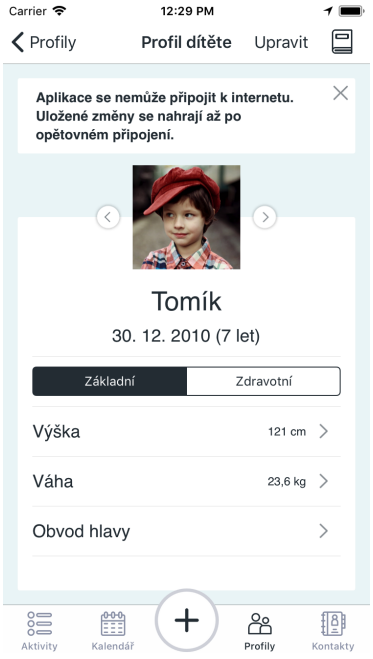
Obrázek A.6: Kalendář (Android)



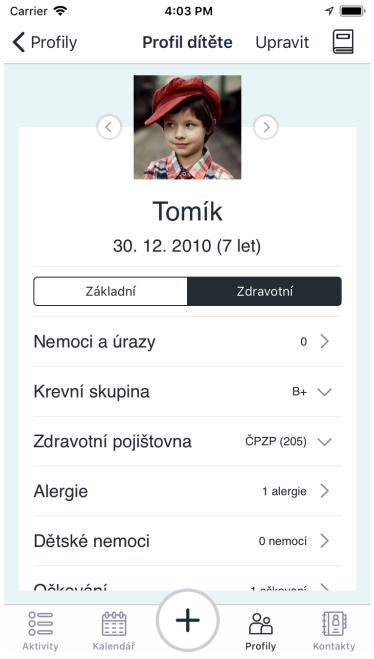
Obrázek A.7: Profily (Android)



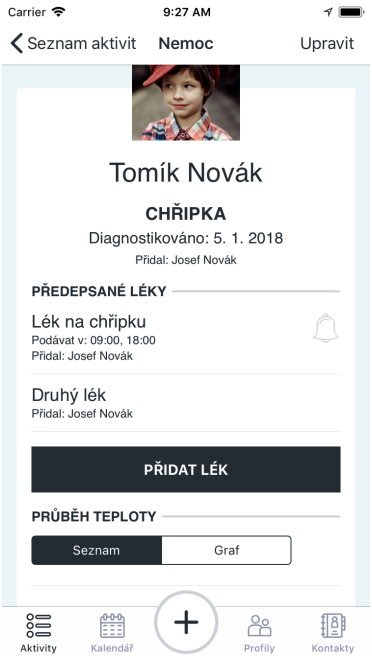
Obrázek A.8: Kontakty (iOS)



Obrázek A.9: Základní profil s upozorněním (iOS)

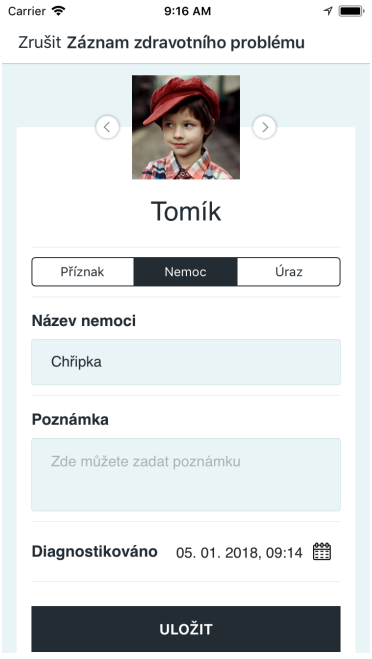


Obrázek A.10: Zdravotní profil (iOS)

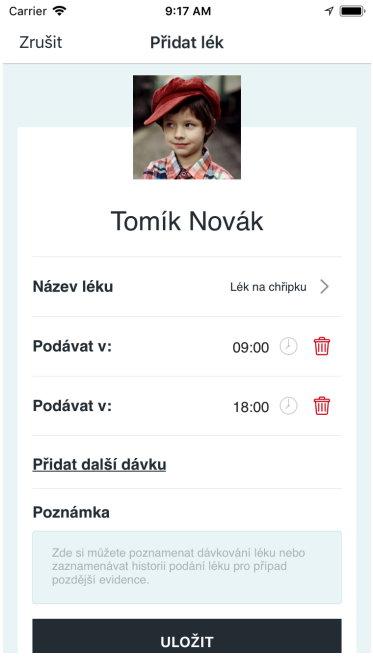


Obrázek A.11: Zdravotní problém část 1. (iOS)

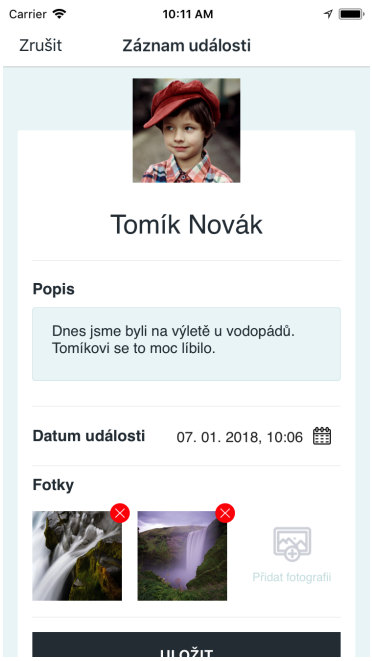
Obrázek A.12: Zdravotní problém část 2. (iOS)



Obrázek A.13: Přidání zdravotního problému (iOS)

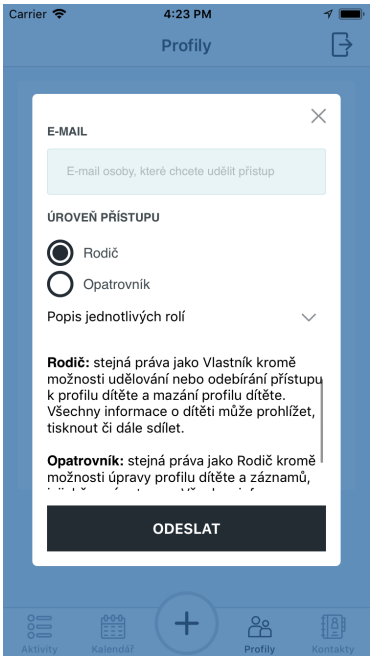


Obrázek A.14: Přidání léku (iOS)

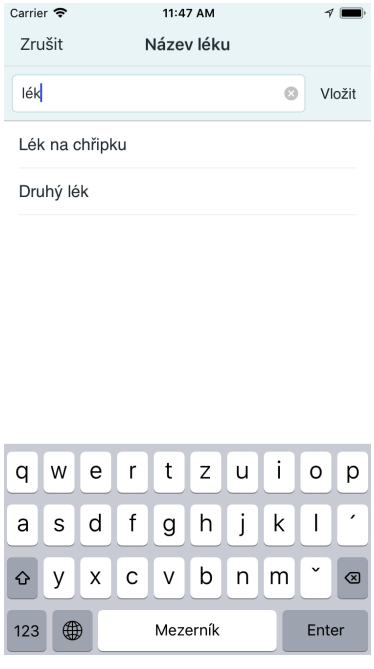


Obrázek A.15: Přidání události (iOS)

Obrázek A.16: Záznam teploty (Android)



Obrázek A.17: Odesílání pozvánky (iOS)



Obrázek A.18: Našeptávač (iOS)

Příloha B

Seznam použitých zkratek

UI User Interface, česky „uživatelské rozhraní“

UX User Experience, česky „uživatelský prožitek“

HTML Hypertext Markup Language

CSS Cascading Stylesheets

HTTP Hyper-Text Transfer Protocol

PDF Portable Document Format

XML Extensible Markup Language

JSON Javascript Object Notation

OS Operační systém

FRP Funkcionální reaktivní programování

Příloha C

Obsah přiloženého USB disku

- `readme.txt` ..... Stručný popis obsahu disku
- `BabyDiary` ..... Zdrojový kód aplikace
  - `Frontend` . Zdrojový kód mobilní části aplikace `.2 Backend` . Zdrojový kód serverové části aplikace `.1 Dist` . `.2 babydiary.apk` . Instalační soubor Android verze aplikace `.1 Paper` .... Zdrojová forma práce ve formátu LaTeX
- `assignment.pdf` ..... Zadání práce ve formátu PDF
- `thesis.pdf` ..... Text práce ve formátu PDF

Příloha D

Instalační příručka

Tato příloha slouží jako návod pro spuštění aplikace, jejíž zdrojový kód je na přiloženém USB disku, ve vývojovém prostředí. Všechny části aplikace by mělo být možné spustit na kterémkoli desktopovém operačním systému, s výjimkou iOS aplikace, kterou lze spustit pouze na Mac OS X. Instalace a spouštění byly testovány právě na Mac OS X, ale také na OS Windows.

Prvním krokem při instalaci kterékoli části systému je zkopírování adresáře `/BabyDiary/` mimo přiložený USB disk, neboť instalace bude potřebovat do svého adresáře stahovat závislosti, kompilovat zdrojový kód a ukládat pomocné soubory, což USB disk znečistí.

D.1 Serverová část

Zdrojový kód serverové části aplikace se nachází v adresáři `/BabyDiary/Backend/` a všechny cesty uvedené v tomto návodu jsou relativní k této složce.

Jak již bylo uvedeno v sekci 5.1, serverová část obsahuje velké množství komponent, které je všechny potřeba instalovat, a proto je pro snadnější spuštění zabalena do Docker kontejnerů. Aplikační server, databázi a všechny jejich závislosti tak jde spustit jedním příkazem, pokud má hostitelský počítač Docker nainstalován. Návod pro instalaci Dockeru se nachází na webové adrese zdroje [27].

D.1.1 Spuštění z Dockeru

Pro Unixové systémy jsem do složky `bin/` přibalil shell skripty, které by se měly o vše postarat, pokud jsou spuštěny z kořenového adresáře serverové části. Stačí pro aktuálního uživatele nastavit oprávnění k jejich spouštění pomocí příkazu `chmod`. Spuštění serveru se pak povede příkazem `./bin/start.sh`, zastavení serveru příkazem `./bin/stop.sh`.

Pokud je potřeba server nainstalovat na jiný operační systém nebo shell skripty nefungují, je možné provést spuštění přes nástroj docker-compose.

Slouží k tomu příkaz `docker-compose -f docker/docker-compose.yml up`. Pokud se přidá přepínač `-d`, poběží proces na pozadí. Pro vypnutí serveru pak stačí nahradit `up` za `down`.

Na OS Windows je potřeba Dockeru udělit povolení ke čtení a zapisování na příslušný disk pomocí vyskakovacího dialogu, proto může start serveru na první pokus selhat.

Po spuštění začne server poslouchat na adrese `localhost:3000` v hostitelském systému.

Odesílání e-mailů

Jedna z funkcionalit aplikačního serveru je i odesílání e-mailů. Pro účely této distribuce je zřízena e-mailová schránka na službě Gmail. Aplikace obsahuje v konfiguračním souboru `app/config/default.json` přístupové údaje k této schránce a bude ve výchozím nastavení odesílat e-maily přes ni. Konfiguraci lze změnit přidáním souboru s názvem `local.json` ve stejném adresáři a přepsáním příslušných klíčů. Obzvláště užitečné jsou následující hodnoty:

- Nastavením klíče `enableEmails` na hodnotu `false` bude odesílání e-mailů vypnuto.
- Nastavením klíče `sendToDefaultEmail` na hodnotu `true` bude odesílání veškerých zpráv přesměrováno na adresu uvedenou pod klíčem `defaultEmail` a nikam jinam se e-maily posílat nebudou.

D.1.2 Spuštění testů

Automatické testy aplikačního serveru jsou opět řešeny přes Docker. Slouží k tomu shell skript spustitelný pomocí `./bin/test.sh`. Před spuštěním samotných testů musí proces čekat asi půl minuty, než server naběhne a začne přijímat připojení.

V případě, že shell skript z nějakého důvodu nelze použít, stačí ručně spustit tři příkazy pro docker-compose, které jsou v něm obsaženy, podobně, jako jsme to popsali výše.

D.1.3 Dostupnost pro mobilní aplikaci

Ve výchozím nastavení server přijímá připojení na portu `3000`. Mobilní aplikace je v této distribuci nakonfigurovaná tak, aby se připojovala na `http://localhost:3000`, což funguje pouze v případě, že je spuštěna na jednom ze simulátorů, popřípadě na zařízení připojeném k počítači přes USB.

Pokud je mobilní aplikace nainstalována jinak, například pomocí přiloženého APK, nebude moci běžící server najít. Bez připojení k serveru bohužel není možné vytvořit uživatelský účet a tudíž ani používat žádnou funkci aplikace. Tento problém se dá vyřešit dočasným zpřístupněním portu `3000` na

počítači, kde běží server, pro externí připojení. Úvodní obrazovka mobilní aplikace (viz. snímek A.1) pak pod tlačítkem pro přihlášení obsahuje také tlačítko pro nastavení adresy aplikačního serveru, kde je možné zadat IP adresu a port ručně ve formátu `host:port`. Po potvrzení dialogu je potřeba aplikaci násilně vypnout a při dalším spuštění už by měla být k serveru připojená.

D.2 Mobilní aplikace

Mobilní aplikaci lze spustit buď přímo na Androidu z exportovaného souboru APK, nebo ze zdrojového kódu pomocí nástroje NPM. Na iOS bohužel díky restrikcím platformy není možné vytvořit instalační balíček, a tak jedinou možností je instalace ze zdrojového kódu.

D.2.1 Spuštění Android verze z APK

V adresáři `Dist/` se nachází soubor `babydiary.apk`, který stačí nainstalovat na jakékoli kompatibilní zařízení s OS Android 4.2 nebo vyšším. Nutnou podmínkou pro smysluplné použití aplikace je ovšem běžící server, jehož adresu je nutné v aplikaci nastavit, viz. sekce D.1.3.

D.2.2 Spuštění ze zdrojového kódu

Zdrojový kód serverové části aplikace se nachází v adresáři `/BabyDiary/Frontend/` a všechny cesty uvedené v tomto návodu jsou relativní k této složce.

Základní prerekvizitou pro instalaci aplikace jsou nástroje Git a NPM (součástí Node.js). Obojí lze jednoduše stáhnout pro jakýkoli operační systém na příslušných webových stránkách.

Jako další krok je potřeba vykonat následující příkazy:

```
$ npm install -g react-native-cli
$ npm install
```

Před spuštením aplikace ve vývojovém módu je ještě potřeba nastartovat React Native package server pomocí příkazu `npm start` a nechat ho běžet po celou dobu, co s aplikací pracujeme.

Jakmile instalace externích závislostí doběhne a je spuštěn package server, je aplikace připravena ke spuštění v XCode i v Android Studiu.

Spuštění iOS verze

Ve správném duchu produktů od společnosti Apple je zprovoznění iOS verze aplikace na jedno kliknutí, tedy pokud jsme na Mac OS X a máme nainstalované vývojové prostředí XCode. Stačí v XCode otevřít soubor

`ios/BabyDiary.xcodeproj` a kliknout na tlačítko „play“ pro spuštění v simulátoru nebo na připojeném testovacím zařízení.

Spuštění Android verze

Ze všeho nejdřív je potřeba nainstalovat Java Development Kit (JDK) od společnosti Oracle a správně nastavit proměnné prostředí `JAVA_HOME` a `PATH`. Postup při instalaci Javy se liší dle operačního systému, návod však není těžké najít.

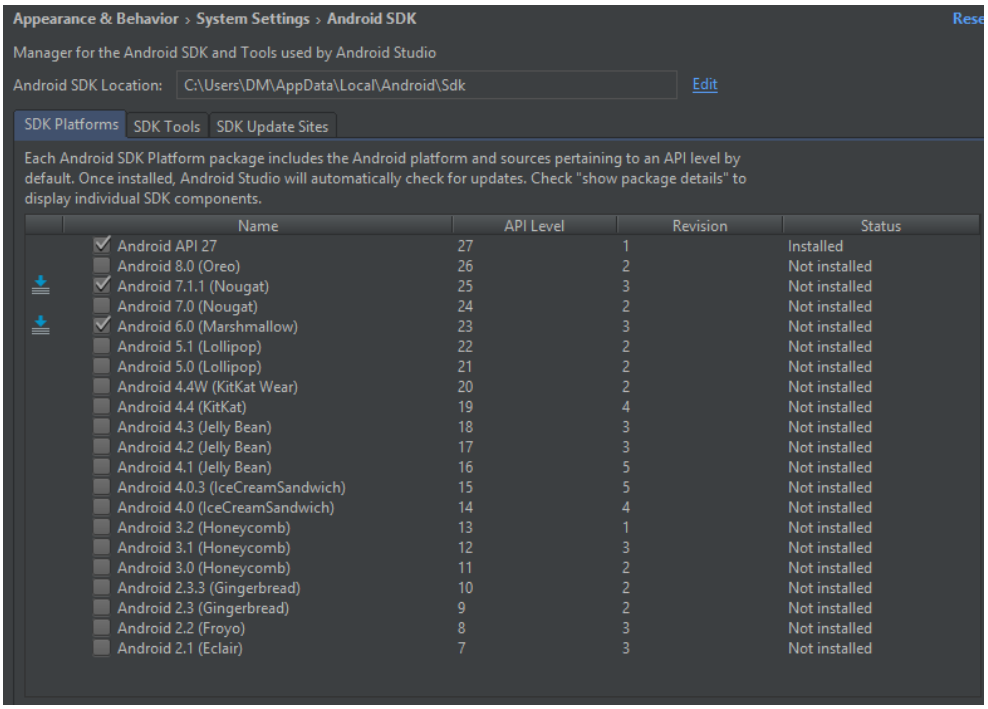
Dále je potřeba mít nainstalované vývojové prostředí Android Studio (ke stažení na `https://developer.android.com/studio/index.html`) a mít v něm nastavený nějaký Android emulátor s verzí operačního systému 4.2 nebo vyšší. Pro úspěšnou kompilaci vyžaduje aplikace následující komponenty platformy Android. Všechny z nich lze nainstalovat v Android Studiu pomocí nástroje „SDK manager“ (viz. snímky obrazovky D.1 a D.2).

- Android SDK verze 23 a 25
- Android Build Tools verze 23.0.1 a verze 25.0.1

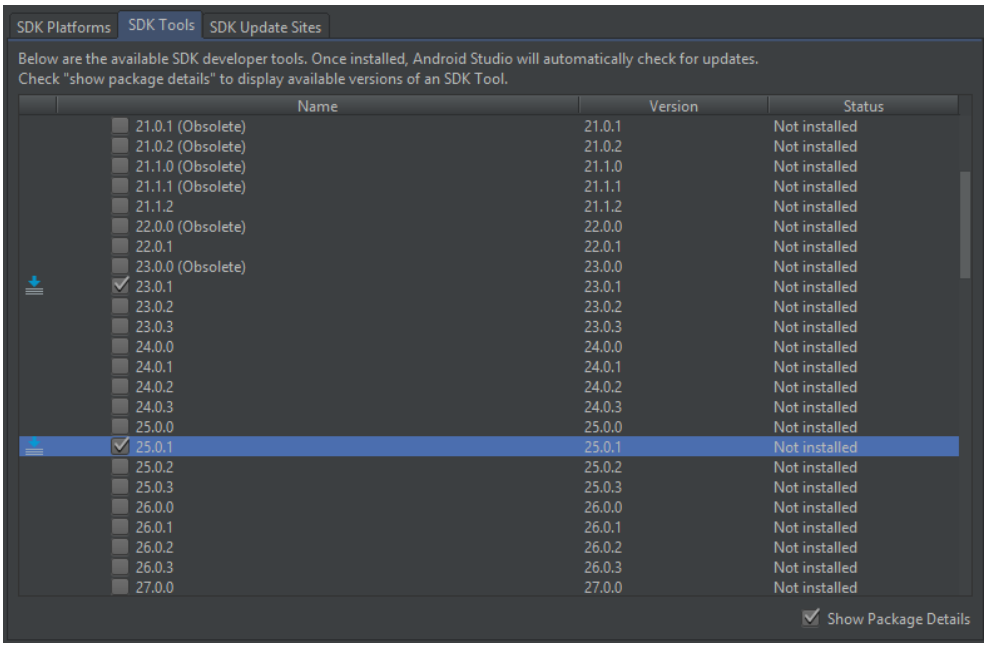
Před spuštěním aplikace na emulátoru nebo na testovacím zařízení z Android Studia je nutné provést následující dva příkazy, které zajistí, aby fungovala síťová komunikace mezi zařízením, package serverem a aplikačním serverem.

```
$ adb reverse tcp:8081 tcp:8081
$ adb reverse tcp:3000 tcp:3000
```

Spuštění je pak už jen záležitostí kliknutí na tlačítko „run“ či „debug“ v Android Studiu.



Obrázek D.1: Instalace Android SDK v Android Studiu



Obrázek D.2: Instalace Android Build Tools v Android Studiu