VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SLEDOVÁNÍ TRÉNINKU S VYUŽITÍM TECHNOLOGIE NFC

TRAINING TRACKING USING NFC TECHNOLOGY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VOJTĚCH BLAŽÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

157213

Ústav: Ústav počítačových systémů (UPSY)
Student: **Blažík Vojtěch**
Program: Informační technologie
Název: **Sledování tréninku s využitím technologie NFC**
Kategorie: Mobilní aplikace
Akademický rok: 2023/24

Zadání:

1. Seznamte se s problematikou návrhu mobilních aplikací a technologií NFC. Proveďte rešerši zaměřenou na mobilní aplikace zaměřené na monitorování a plánování osobních tréninků a systémy pro sledování vytížení a údržbu fitness strojů.
2. Navrhněte aplikaci využívající technologie NFC, která bude uživatelům umožňovat plánovat, sledovat a analyzovat průběh tréninku a provozovatelům sledovat vytížení strojů. Při návrhu dbejte na ergonomii aplikace.
3. Navrženou aplikaci implementujte ve vhodném multiplatformním frameworku (PWA, Flutter, React Native, apod.).
4. Vyhodnoťte a diskutujte parametry a možnosti dalšího rozšíření. V rámci vyhodnocení proveďte dotazníkové šetření mezi uživateli.

Literatura:
- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
- Splnění bodu 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 2.5.2024

Abstrakt

Tato práce se zabývá vývojem mobilní aplikace využívající technologii NFC pro sledování tréninků a monitorování využití strojů v posilovnách. V práci jsou shrnuty základní důvody a přístupy ke sledování tréninků a vytížení strojů, shrnuje také technologii NFC a vývoj mobilních aplikací se zaměřením na multiplatformní framework Flutter. Výsledná aplikace je založená na architektuře klient-server s podporou offline režimu a k identifikaci stroje a cviků využívá identifikátory rozmístěných NFC štítků. Přínosem práce je ukázka funkčního systému, který využívá tuto technologii netradičním způsobem k dosažení vyššího uživatelského komfortu.

Abstract

This bachelor thesis describes the development of mobile application which uses NFC technology for training tracking and monitoring of gym equipment usage. The work summarizes the basic motivation and approaches to training and equipment tracking as well as NFC technology and mobile app development focused on the multiplatform framework Flutter. The resulting app is based on the client-server architecture with offline mode support and uses the identifiers of deployed NFC tags to identify the machine and exercises. The goal of this work is to show a functional system, which incorporates this technology in an unconventional way to achieve a higher user comfort.

Klíčová slova

NFC, tag, RFID, mobilní aplikace, Flutter, Dart, PocketBase, framework, multiplatformní, Android, iOS, klient-server, backend, frontend, API, posilovna, cvik, trénink, widget, logování, vytížení, monitorování, UI, localstorage

Keywords

NFC, tag, RFID, mobile application, app, Flutter, Dart, PocketBase, framework, multiplatform, Android, iOS, client-server, backend, frontend, API, gym, exercise, trainings, widget, logging, usage, monitoring, UI, localstorage

Citace

BLAŽÍK, Vojtěch. *Sledování tréninku s využitím technologie NFC*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Sledování tréninku s využitím technologie NFC

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Josefa Strnadela. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Blažík
7. května 2024

Poděkování

Chtěl bych poděkovat především panu Zdeňkovi Vašíčkovi za umožnění realizace mého nápadu v rámci bakalářské práce a panu Josefovi Strnadelovi za pomoc s realizací práce. Dále bych chtěl poděkovat Michalovi Uhrovi za možnost vyzkoušet systém v jeho provozovně a také všem zúčastněným v rámci testování aplikace.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Návštěva posiloven či tělocvičen je jednou z oblíbených sportovních aktivit dnešní doby, především díky široké škále možností a vybavení, které tato zařízení poskytují. Pro dosahování fitness cílů je klíčový výběr sportovních aktivit, který nemusí být vždy jednoduchý. S rozvojem mobilních technologií se otevírají nové možnosti, jak tento výběr zefektivnit a zpříjemnit. Tato práce se zaměřuje na vývoj mobilní aplikace, která těchto technologií využívá pro řešení potřeb moderního fitness světa. Aplikace umožňuje uživatelům zapisovat a následně analyzovat jejich tréninkové aktivity. Využívá technologii NFC k načtení dat o stroji či vybavení a tím k rychlejšímu nalezení požadovaného cviku, který právě hodlá vykonávat. Dále uživateli poskytuje historii jeho cvičení, čímž přispívá k lepšímu rozhodování o dalších aktivitách. Vedle toho poskytuje správcům zařízení informace o použití jejich vybavení, čímž dodává majiteli potřebná data k zefektivnění a zlepšení provozu jejich zařízení. Jedním z primárních cílů je také vytvořit funkční *proof of concept*, který by později mohl být zakomponován i do jiných již existujících řešení.

První část práce (kapitola 2) je zaměřená na shrnutí aktuálně dostupných řešení společně s technologiemi, které jsou v následující kapitole 3 zkombinovány do návrhu o nové aplikaci. V následující kapitole 4 je popsána implementace této aplikace nástrojem, díky kterému je možné aplikaci používat na zařízeních Android i iOS. Práce je zakončena otestováním daného řešení v praxi (kapitola 5) a zhodnocením dalšího možného využití celého systému (kapitola 6).

Kapitola 2

Rešerše

2.1 Sledování tréninku

Tato práce se zabývá sledováním tréninku, především silových tréninků v posilovnách s vybavením. Pojmem „sledování“ je zde myšleno zapisování (často také jako „logování“) jistých veličin a následné sledování jejich vývoje a využití těchto hodnot v budoucnosti.

2.1.1 Přínosy sledování tréninků

Při posilování nebo obecně cvičení je hlavním cílem zlepšování se a dosahování lepších výsledků, takových výsledků však není možné dosáhnout bez neustálého posouvání hranic [31]. V případě posilování a cvičení jsou těmito hranicemi zvedané váhy, uběhnutá vzdálenost či počet opakování daného cviku. To jsou však velmi jednoduše změřitelné veličiny. Trasování tréninků nám tedy umožňuje sledovat vývoj těchto hranic, kterých člověk zvládne dosáhnout, a zajistit, že se tyto hranice budou i nadále posouvat. Mezi hlavní přínosy sledování můžeme zařadit:

- výběr vhodného cviku,
- výběr vhodné váhy, počtu opakování a dalších veličin na základě minulosti,
- sledování statistik pro účely zajištění vývoje,
- vyšší motivace a další.

Jako příklad běžného případu užití sledování je situace, kdy nový návštěvník posilovny začne pravidelně cvičit, ovšem nepamatuje si z paměti, jaké váhy je vhodné si na různých strojích zvolit. Jako primitivní nástroj pro trasování si zvolí obyčejný sešit, do kterého si po každém cviku zapíše, které váhy mu nejvíce vyhovovaly. Při příští návštěvě posilovny se poté podívá do sešitu na posledně zapsané váhy, které si zvolí.

2.1.2 Možnosti sledování

Sledovat je možné mnoho veličin, mezi které patří například:

1. zvedané váhy na jednotlivých strojích/cvicích v posilovně,
2. počty opakování daného cviku (např. klik),

3. zvolený cvik a procvičené svalové partie,
4. způsob provedení cviku,
5. uběhnutá/ujetá vzdálenost,
6. doba strávená nějakou aktivitou (např. prkno – výdrž ve zpevněné poloze),
7. počet úspěšných pokusů (např. střelba na branku),
8. fyziologické veličiny (např. tep).

Pro silové tréninky je běžné cvičení po sériích, u kterých je nejčastější sledování veličin 1 až 3, na které je tato práce zaměřená, ovšem v závislosti na tréninku je možné sledovat prakticky cokoli. Dále je možné u série rozlišovat její typ, který může být běžnou sérií, rozcvičkou, tzv. *dropsetem*[1] či např. úplným vyčerpáním s konstantní vahou [31]. Prezentace těchto informací může poté probíhat různými způsoby:

- zobrazení celého tréninku,
- zobrazení posledních zapsaných hodnot,
- graf vývoje jednotlivých veličin,
- shrnující statistiky, jako je *1RM*[2] či *volume*[3],
- rozdělení svalových partií v rámci tréninku,
- celkový počet tréninků, cviků, sérií apod.

Následují různé typy přístupů ke sledování tréninků. Tyto typy ovšem nejsou nijak oficiálně definované – názvy byly vytvořeny a jsou použity pouze pro účely této práce na základě výzkumu přibližně 20 populárních mobilních aplikací různých typů na obchodě Google Play.

Pasivní sledování

Pohled pasivní je zde brán z pohledu uživatele, tedy takové trasování se děje "samo". Uživatel pouze spustí aplikaci na mobilním telefonu nebo chytrých hodinkách a aplikace sleduje trénink sama. Tyto aplikace využívají k měření technologie zabudované v chytrých telefonech či hodinkách, jako je např. poloha či tep. Dále využívají předem zadané hodnoty uživatelem, jako je váha, výška či typ tréninku (běh, jízda na kole, plavání apod.). Díky těmto datům dokáží uživateli dodat širokou škálu informací o jeho trénincích – rychlost běhu, spálená energie, tep, tempo a další. Vzhledem k povaze celého systému a absenci sofistikovanějších měřících technologií jsou tyto aplikace poněkud omezené z hlediska využití, neboť dodávají především shrnující informace o tréninku. Proto nejčastější využití tohoto způsobu měření jsou tréninku typu již zmíněné běh či jízda na kole.

[1] Postupné snižování váhy do vyčerpání.

[2] 1RM, neboli one-repetition max, představuje odhad maximální možné váhy, kterou je člověk schopen zvednout na základě předchozího výkonu. Existuje více způsobů výpočtu. Více o 1RM na Wikipedii: odkaz.

[3] Celková zvedlá váha, počítá se pro každou sérii jako váha×opakování.

Příkladem této aplikace je *Strava* [4]. Strava umožňuje pasivně sledovat širokou škálu sportů. Dále umožňuje připojení různých senzorů, díky kterým je možné měřit fyziologické veličiny. Bez těchto senzorů je však aplikace pro silové tréninky – na které je tato práce zaměřená – nepoužitelná.

Plánování

Plánování – jak už název napovídá – spočívá v naplánování celého tréninku dopředu. Takové plánování má smysl v případě, kdy je trénink komplexní a skládá se z více částí či cviků. Tento fakt je typický pro silové tréninky, ovšem může se vyskytnout prakticky u jakéhokoli sportu. V praxi si uživatel ještě před započetím tréninku naplánuje cvik po cviku celý trénink a podle připraveného plánu se poté řídí. Mnohé plánovací aplikace také nabízí již předpřipravené tréninky pro své uživatele, které ani nemusí skládat.

Tento způsob sledování tréninku má řadu výhod. V první řadě může dodat začátečníkům směr, kterým se mají v posilovně vydat. Plán může nahradit trenéra, neboť přesně určuje, co bude uživatel v příštím kroku dělat. Předpřipravené tréninky se dají po zadání osobních informací (jako je váha, úroveň pokročilosti apod.) připravit na míru uživateli, je možné také zohlednit jeho historii cvičení. Hlavní nevýhoda tohoto způsobu ovšem spočívá v nižší flexibilitě, která se může projevit v například v situaci, kdy potřebné vybavení je obsazené, mimo provoz či se v posilovně vůbec nenachází. Tento způsob také nemusí být vhodný v případě cvičení s partnerem či osobním trenérem. Změny plánu při tréninku mohou uživatele vyrušovat více, než mu při tréninku pomáhá samotný plán. Poslední hlavní nevýhodou je fakt, že uživatel tento plán opravdu musí mít dopředu připravený, a to tak, aby mu vyhovovala délka i náplň plánu. Při nevyhovující délce či náplni musí opět tento plán měnit. Sestavení vlastního plánu nemusí být vůbec jednoduché i z důvodu výběru cviků, který je popsán v následující podkapitole.

Příklad plánovací aplikace je *Gym Workout Tracker: Gym Log*[5].

Logování

Logování spočívá v dynamickém zapisováním tréninku přímo při cvičení. Z uvedených způsobů trasování tréninků je tento způsob nejvíce obecně použitelný a je možné jej realizovat i bez použití jakékoli specializované aplikace (např. papírový sešit či poznámkový blok v telefonu). Uživatel vždy vybere cvik či činnost a následně zapisuje svůj výkon (u silového tréninku se často jedná o použitou váhu a počet opakování).

Výhodou a hlavním přínosem logovacích aplikací je jednoduchá použitelnost, přehlednost a možnost zobrazení historie. Taková aplikace by měla obsahovat nabídku cviků, ze kterých si uživatel vybere, což mu může i dodat inspiraci při tréninku. Velká nevýhoda těchto aplikací vychází z principu veškerého zapisování, kdy uživatel vše dělá až během tréninku a tím pádem je od samotného cvičení rušen. Důležitou vlastností těchto aplikací by tedy měla být jednoduchost a rychlost zapisování a výběru cviků. Existuje řada aplikací, které mají obě části – různými způsoby ač dobře – propracované. Všechny aplikace se však shodují ve výběru cviku, kde pro zvolení a přidání nového cviku je nutné vyhledat v seznamu obsahující až stovky cviků. Tyto seznamy sice obsahují prostředky pro vyhledávání mezi cviky, jako jsou filtry společně s obrázky či popisky cviků, i přesto je tento způsob patrně nepohodlný a může velmi rušit od samotného tréninku. Tato skutečnost hrála klíčovou roli

[4] Strava na obchodě Google Play: odkaz.

[5] Gym Workout Tracker: Gym Log na obchodě Google Play: odkaz.

u inspirace pro tuto bakalářskou práci a pro návrh logovací aplikace, která bude využívat technologii NFC ke zrychlení vyhledávání cviků.

Příkladem logovací aplikace je *Hevy*[6]. Tato aplikace tvořila velkou inspiraci při tvorbě této práce, neboť způsob zapisování je velmi intuitivní.

2.2 Sledování vytížení strojů

2.2.1 Přínosy sledování vytížení strojů

Sledovat a monitorovat je možné nejen samotný trénink, ale i použité stroje jako takové. V tomto případě však data nevyužívá uživatel posilovny, nýbrž její majitel či správce. Tato data mohou být například:

- celkové použití daného stroje či vybavení,
- dostupnost či nedostupnost daného stroje či vybavení pro návštěvníky,
- nefunkčnost či závada na stroji či vybavení;

tato data poté může využít např. na:

- výběr adekvátního počtu daného stroje či vybavení,
- rozložení strojů a vybavení za účelem rovnoměrného rozprostření návštěvníků po zařízení,
- detekci a opravu nefunkčností.

2.2.2 Možnosti trasování vytížení strojů

Někteří výrobci, jako je např. Precor nebo Matrix, nabízí vybavení se zabudovaných monitorováním strojů včetně upozornění na případné závady na stroji [14]. Toto řešení může mít řadu výhod, neboť výrobce může rozmístit monitorovací zařízení přesně na míru. Nevýhoda očividně spočívá v nutnosti koupě celého kusu vybavení od daného výrobce. Vybavení posilovny, které by nebylo od dané společnosti, by tím pádem nemohlo být nijak sledované.

Pro stroje či vybavení, které nemají monitorování zabudované, tu existuje řešení například od společnosti GYMetrix [24]. Její systém spočívá v připevnění monitorovacích zařízení na stroje a vybavení, které po dobu jednoho nebo dvou týdnů sledují, a současně provedou dotazníkové řízení s návštěvníky posilovny. Na základě získaných dat společnost vytvoří provozovně plán pro zlepšení počtu a rozmístění vybavení. Problém zde ovšem spočívá ve vyšší ceně a především krátké době sledování, která může vést k nepřesnostem a po změně návštěvnosti či přidání nového vybavení by musel celý proces být zopakován.

Dále existují například systémy jako je upace [13], který slouží spíše jako centrála pro všemožné informace o posilovně a rezervační systém, ovšem data o využití strojů či jejich závady je nutné zadávat ručně. Takový systém má využití spíše u velkých provozoven.

2.3 Technologie NFC

Near field communication (zkráceně NFC) je sada protokolů umožňující bezkontaktní komunikaci dvou zařízení na krátkou vzdálenost, standardně jednotky centimetrů a méně [32].

[6] Hevy na obchodě Google Play: odkaz.

Komunikace vždy obsahuje dvě zařízení – iniciátor, který začne komunikaci, a cílové zařízení. Tato zařízení mohou být obě aktivní (např. dva telefony), tedy obě jsou napájené a jsou schopna aktivně vysílat, nebo jedno zařízení aktivní a druhé pasivní, tedy jedno zařízení (typicky cílové) není stále napájené a energii získává z aktivního zařízení pouze na dobu komunikace.

Technologie je určena především pro přenos malého objemu dat, standardy určují rychlosti do 424 kbit/s [32]. Dále je technologie často využívána ve formě tzv. NFC tagů, tedy malých pasivních štítků, které v sobě mají zabudovaný NFC čip s anténou. Tyto tagy jsou vybavené malou pamětí (typicky desítky až tisíce bajtů) [32]. Uvedené vlastnosti dělají z NFC tagů levnou a nenáročnou technologii, která je vhodná k využití v masové produkci i případně pro jednorázové použití.

2.3.1 Případy použití

V dnešní době většina chytrých telefonů disponuje touto funkcí [5], tedy mnoho z nás nosí běžně NFC čtečku v kapse. Tento fakt dělá z NFC velmi široce využitelný prostředek.

Emulace karty mobilním telefonem

V dnešní době nejznámějším využitím NFC je placení mobilním telefonem. Uživatel zadá do mobilní aplikace údaje o své platební kartě a poté při placení je tato technologie využita na výměnu platebních informací mezi telefonem a terminálem. Tímto způsobem je možné mít v telefonu uloženo několik (i neplatebních) karet, mezi kterými si uživatel vždy vybere, kterou právě použije. Mezi aplikace umožňující tyto platby se řadí např. Google Pay nebo Apple Pay.

Oproti platební kartě má tento způsob bezhotovostního placení mnoho výhod. V prvé řadě je to pohodlnost, neboť uživatel aplikace nemusí u sebe nosit fyzické karty. Další výhodou je bezpečnost, neboť aplikace pro správu karet často vyžadují odemknutí či potvrzení platby heslem či otiskem prstu, tedy při ztrátě telefonu nebude možné provést jakoukoli platbu bez prolomení hesla [22]. To ovšem u běžné karty může být problém, neboť při jejím odcizení má neoprávněný držitel karty do jisté výše částky přístup k bezkontaktním platbám či platbám přes internet.

Stejným způsobem je možné naemulovat jakoukoli kartu, která může sloužit např. pro identifikaci osoby či odemykání domu, brány od parkoviště atd. Technologii je však možné využít podobným způsobem i bez nutnosti emulace v mobilním telefonu, tedy výše uvedené přístupové nebo platební karty je možné využít na fyzické kartě či čipu. Tímto způsobem je možné např. platit bezkontaktně na festivalu v oblasti se slabým signálem, kde by nebylo možné platit běžnou kartou. V takovém případě bude uživateli nahrána na NFC kartu částka, kterou zaplatil hotově, a tuto částku poté může utrácet bezhotovostně v areálu.

Bootstrapping jiné technologie a automatizace

Jak bylo řečeno v úvodu kapitoly 2.3, NFC nedisponuje vysokými rychlostmi přenosu, neboť k tomu ani není určena. Kromě toho pracuje na krátkou vzdálenost. Těchto vlastností je však možné využít k vytvoření spojení (bootstrapping) u jiné rychlejší technologie, jako je např. WiFi (pro představu řádově 1000krát rychlejší než NFC [10]) nebo Bluetooth. Některé technologické společnosti nabízí přímo speciální aplikace, které pomocí NFC spustí přenos

dat a automaticky se přepnou na jinou technologii. Jako příklad je dnes již nepoužívaný Android Beam[7].

Pro ruční připojení k WiFi je nutné na NFC tag zapsat záznam typu připojení k WiFi a zadat informace o síti, jako je název, heslo a typ zabezpečení. Při přečtení tagu se telefon k WiFi připojí [35]. Konkrétní využití pak může spočívat v NFC tagu, který návštěvník domácnosti načte a rychle a pohodlně se tak připojí k WiFi.

Pomocí Bluetooth je možné zařízení spárovat buď přes záznam na NFC tagu s MAC adresou (podobně jak u WiFi) či přímo kontaktem dvou zařízení, které podporují peer-to-peer Bluetooth párování [29].

Kromě spuštění jiné komunikační technologie je možné NFC tagy dále využít na různé automatizované úlohy, jako je posílání zpráv, otevření webové adresy či spuštění aplikace [34].

Zabudování do okolního světa

Zabudováním do okolního světa je myšleno využití NFC tagu v reálném světě, kdy je tag připevněný na konkrétní věc a data na tagu s touto věcí nějakým způsobem souvisí. Částečně se toto využití kryje s automatizací popsanou v předchozí podkapitole. Takové využití se najde ve chvíli, kdy není možné nějaké informace vyčíst přímo z dané věci jako takové, chceme si nějakým způsobem zapamatovat návštěvu této věci apod. Konkrétní využití může být například:

1. odkaz na YouTube video nebo dodatečné informace u výstavního exponátu,
2. kontaktní informace uložené na vizitce,
3. aktuální jídelní lístek v restauraci,
4. kontrolní bod u orientačního běhu,
5. odkaz na ohodnocení navštíveného místa a další.

Mnoho těchto úkolů by bylo možné vyřešit i přes jinou technologii, jako je např. QR kód. NFC tagy mají však oproti QR kódům výhodu v pohodlnosti, neboť pro přečtení NFC tagu stačí mít odemknutý telefon. Také není možné u QR kódu trasovat, kolikrát byl přečten a jestli byl opravdu přečten kontaktně (bod 4. by nebylo možné přes QR kód zkontrolovat). Jsou ovšem samozřejmě i případy, kdy je QR kód vhodnější, především když není možný kontakt na blízko. Často je vhodné tyto technologie kombinovat.

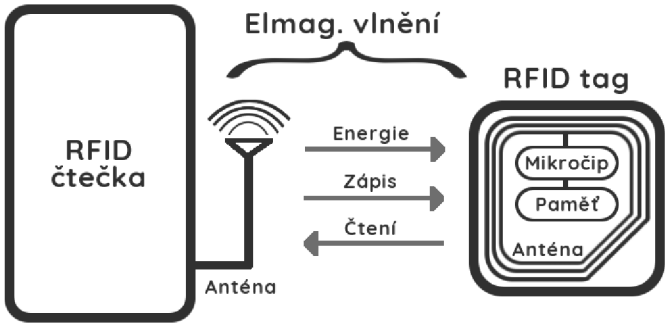
2.3.2 Technologie RFID

Celá technologie NFC je založena na technologii RFID – Radio Frequency IDentification (identifikace na rádiové frekvenci) [32]. Technologie by zabrala celou velkou kapitolu, ovšem tato práce není na hardwarovou stránku NFC zaměřena, proto je zde uveden jen krátký přehled.

RFID využívá elektromagnetické pole, jedná se tedy o bezdrátovou komunikaci, kde vzdálenost komunikujících objektů může dosahovat od několika centimetrů až po několik stovek metrů [8] a objekty nemusí být ve vzájemné přímé viditelnosti. Systém vždy obsahuje dva objekty – transpordér (též RFID štítek či tag) a čtecí zařízení. Při komunikaci poté

[7] Více o Android Beam na Wikipedii: odkaz.

čtecí zařízení snímá své okolí a hledá v něm transpordéry, které při nalezení začnou se čtečkou komunikovat. Navázaná komunikace je u RFID jednosměrná – pasivní prvek vždy odešle uložená data či zapíše přijatá data. Obrázek 2.1 zobrazuje zjednodušené schéma komunikace RFID.



Obrázek 2.1: Schéma komunikace RFID

RFID transpordéry

Transpordér slouží k označení objektů z okolního světa (např. položky v obchodě). Každý transpordér se skládá z antény a mikročipu, který disponuje pamětí; tyto části jsou pak obalené v papírové, plastové či jiné ochraně nebo jsou přímo zabudované do objektu. Mikročip se stará o zpracování signálu (modulace a demodulace) a práci s daty. Na každém tagu je uložen 96bitový identifikátor, tzv. Electronic Product Code (EPC) [8]. Tento kód je složen z více částí, které tvoří hierarchii identifikace – 8 bitů hlavička, 28 bitů organizace, 24 třída produktu a 36 bitů konkrétní položka. Často se při komunikaci používá právě tento unikátní kód, ovšem je možné také využít paměť v tagu, která může být v jakémkoli režimu práce s pamětí (read only, read write, write once,...).

Transpordéry se dělí na dva základní typy: aktivní a pasivní [8]. Aktivní tagy jsou napájené a aktivně vysílají elektromagnetické pulsy do svého okolí, čímž docílí vyššího dosahu a je možné používat pasivní čtečku pro jejich identifikaci. Napájení i vyšší složitost však dělá v mnoha případech z aktivních tagů drahé či nepoužitelné řešení. Z toho důvodu jsou častější pasivní tagy, které v blízkosti čtečky využijí elektromagnetické vlnění k nabití kondenzátoru pomocí elektromagnetické indukce a poté tuto energii využijí ke komunikaci. Nevýhodou u pasivních tagů však může být již zmíněná kratší vzdálenost, která dosahuje řádově jednotek centimetrů až jednotek metrů.

RFID čtecí zařízení

Čtečka naopak slouží k identifikaci objektu, který je označen RFID transpordérem (např. brána za pokladnou v obchodě). Stejně jako tagy se dělí na aktivní a pasivní – rozdíl mezi nimi je analogický, ovšem četnost výskytu je opačná (tedy aktivní čtečky jsou častější než pasivní), neboť komunikace vždy musí obsahovat alespoň jeden aktivní prvek, který bude napájen.

Využití RFID

Jak již bylo řečeno, nejčastějším využitím RFID je označení objektů a poté jejich trasování. Tyto objekty mohou být položky v obchodě, skladu, vybavení, vozidla nebo i lidé či zvířata.

Díky již zmíněné nízké ceně a malého rozměru pasivních tagů není problém tyto tagy využívat při masové výrobě. Konkrétně pak RFID tagy mohou být využity u [8]:

- výroby aktiv: každé aktivum je označeno tagem, čímž je odstraněna nutnost manuálního zadávání dat (např. počet vyrobených kusů),
- maloobchodních řetězců, knihoven, kasín apod.: štítky slouží ke kontrole odcizení položky pomocí čteček u východů ze zařízení,
- validace přístupu a identifikace osoby: přístupové karty, pasy či zařízení (např. umístěné ve vozidle a čtené u příjezdové brány)
- identifikace zvěře: implantované čipy slouží k jejímu trasování,
- sportu: např. časová kontrola velkých skupin běžců,
- kontroly polohy a mnoho dalšího.

Předchůdcem a také častou alternativou k RFID je čárový (případně QR) kód. Tyto jednoduché technologie však vyžadují přímou viditelnost potisku a čtečky, které často pracují pouze na krátkou vzdálenost. Dále čárové kódy umožňují pouze identifikaci třídy objektu, nikoli samotné instance objektu, krom toho po potištění objektu není možné nijak měnit data. Z těchto důvodů jsou vhodné pouze pro část případů užití RFID. Oproti RFID tagům jsou však velmi levné, neboť k výrobě stačí pouze výrobek či obal potisknout čárovým kódem, proto se dnes stále tento způsob identifikace hojně využívá.

2.3.3 Standardy a komunikační protokol

NFC je založeno na RFID standardu ISO/IEC 14443 a standardu FeliCa [7]. Tyto standardy definují komunikaci bezkontaktních RFID karet na frekvenci 13,56 MHz, tedy frekvenci, na které funguje také NFC. Samotné NFC je definováno dvěma standardy ISO/IEC 18092 (NFCIP-1) a ISO/IEC 21481 (NFCIP-2). NFCIP-1 definuje [25]:

- aktivní i pasivní komunikační režimy pro NFC,
- transportní protokol (aktivace komunikace a samotný přenos dat),
- modulační schémata, kódování, přenosové rychlosti, formát rámců a další nezbytné parametry.

Podle standardu ISO/IEC 14443 se NFC čipy dělí na dva typy A a B, které se liší způsobem kódování a typem modulace.

Organizace NFC Forum dále dělí NFC čipy na 5 [28] číslovaných typů (tabulka 2.1), lišící se pamětí, rychlostí přenosu a typ 5 odolností vůči poškození či kolizím signálu. [20]

Identifikátor tagu

Každý NFC tag přichází s unikátním identifikátorem (UID), který se nachází v read-only paměti čipu [15]. Jedná se o 4 až 10bajtové číslo, které je unikátní pro každý vyrobený tag. První bajt je přidělen výrobci tagů, který pak zodpovídá za unikátnost zbytku čísla (např. NXP Semiconductors má prefix 0x04 [23]). UID je posíláno při každém čtení tagu a je tedy možné jej využít k identifikaci objektu, se kterým je tag spojen (podobně jako u RFID v předchozí kapitole). Tato skutečnost je pro tuto práci klíčová, přestože by bylo možné vytvořit řešení i bez unikátního čísla.

Číslo	Založen na stan.	Paměť	Rychlost přenosu	Další vlastnosti
1	ISO14443A	96 B	106 kb/s	Levný, jednoduchý
2	ISO14443A	48 B až 2 kB	106 kb/s	Nejpoužívanější typ
3	FeliCa	až 9 kB	212 kb/s	Pro složitější aplikace
4	ISO14443A, ISO14443B	až 32 kB	106-424 kb/s	Odolný vůči datové kolizi
5	ISO 15693	až 2 kB	106 kbit/s	Odolný vůči poškození

Tabulka 2.1: Typy NFC tagů a jejich parametry.

Formát NDEF

NFC Forum dále nadefinovalo formát, ve kterém jsou přenášena aplikační data při komunikaci, nazývaný NDEF (NFC Data Exchange Format) [11]. Pomocí tohoto formátu je možné posílat mezi komunikujícími zařízeními libovolná data, jako je text, URL či libovolné MIME typy (např. obrázek). Jedna NDEF zpráva se skládá z jednoho či neomezeně mnoho záznamů, kde první záznam je označený příznakem Message Begin a poslední záznam příznakem Message End. Každý záznam kromě zmíněných a dalších příznaků obsahuje délku dat, typ dat, nepovinný identifikátor a samotná data. Vzhledem k tomu, že pro tuto práci není detail NDEF zprávy podstatný, není zde uveden.

2.3.4 Bezpečnost NFC

NFC jako takové bezpečnost většinou neimplementuje. Ačkoli komunikace probíhá na krátkou vzdálenost, existují u ní mnohá rizika. Mezi možné útoky na NFC se řadí například [7]:

1. odposlech komunikace – např. krádež osobních informací při bezkontaktním placení,
2. přepojovaný útok – podobný útoku Man in the middle,
3. modifikace dat – pomocí RFID rušičky je možné modifikovat vysílaný signál,
4. modifikace dat na tagu – data na tagu může přepsat libovolný uživatel pomocí mobilní aplikace.

Většina bezpečnostních opatření je u NFC realizováno na vyšších vrstvách, např. proti odposlechu je možné se bránit šifrováním provozu na komunikaci. Pro tuto práci je však důležitý především bod 4., neboť na tagu budou uložena data, která by mohl kdokoli přepsat. Ostatní rizika nás nemusí trápit, jelikož při komunikaci nedochází k výměně osobních dat, ani by nebyl problém, kdyby se někdo pokusil uživateli data podvrhnout. Navíc tato rizika vyžadují náročnější realizaci, na kterou běžný člověk nemá prostředky. Existují ovšem veřejně dostupné aplikace, které umožňují zapisovat na NFC tag libovolná data, jako je např. NFC Tools[8]. Pro prevenci tohoto rizika má většina NFC čipů implementován zámek nebo heslo [15]. Po zamčení tagu již není možné data přepsat ani zámek odstranit, neboť bit označující stav zámku se nachází ve write-once paměti čipu. Některé tagy, jako je např. NXP NTAG213, implementují možnost zamknutí zápisu heslem[9], které je možné odstranit

[8] NFC Tools v obchodě Google Play: odkaz.
[9] Detailní popis tohoto tagu se nachází v kapitole 4.1.

a data tedy mohou být znovu přepsána. Tento způsob ovšem má taky své nedostatky, jelikož – jak bylo psáno – pro přepsání dat je nutné heslo opravdu odstranit a poté na tag zapsat znovu, dále heslo je posíláno v prostém textu, není tedy odolné proti potenciálnímu odposlechu.

2.3.5 NFC tagy uzpůsobené na kovový povrch

Klasické NFC tagy by nebylo možné použít na kovovém povrchu, jelikož kov ruší signál a tím pádem by nebylo možné tag přečíst. Od toho existují tagy na kov (on-metal tagy), které obsahují vrstvu feritové fólie mezi anténou a lepící částí tagu [33]. Tato tenká vrstva vytváří elektromagnetickou mezeru mezi anténou a kovem, čímž zajišťuje správnou funkci tagu. Nevýhodou těchto tagů je ovšem nižší čtecí vzdálenost (ve srovnání se stejným tagem i anténou, ovšem bez modifikace na kov) a možnost čtení pouze ze strany, kde vrstva feritu není (to by bylo možné využít např. v situaci, kdy instalátor tagu chce tag schovat za kovovou stěnu). Na druhý nedostatek ovšem také existuje řešení, v této práci tuto vlastnost však nebudeme potřebovat. Důležitou vlastností těchto on-metal tagů je i fakt, že dokáží fungovat i na nekovovém povrchu, neboť tagy se v praxi mohou nacházet i mimo kovová místa, např. na plastové části stroje či zdi, proto je možné tyto tagy použít univerzálně na všechna místa.

2.4 Vývoj mobilních aplikací

Vývoj mobilní aplikace je proces vytvoření softwaru pro mobilní zařízení, jako jsou chytré mobilní telefony různých operačních systémů či chytré hodinky [4]. Mobilním aplikacím dodávají jejich specifičnost především hardwarové nároky, jako je výkon, tvar obrazovky, dostupné periferie či dostupné funkce, a softwarové nároky, neboť na mobilních zařízeních často běží speciální operační systém – u chytrých telefonů nejčastěji Android či iOS. Na vývoj je nutné zvolit vhodné prostředky, které reflektují tyto nároky a potřeby výsledné aplikace.

2.4.1 Proces vývoje

Stejně jako u každého jiného typu vývoje software není možné hned začít programovat. Vývoj by se měl řídit jedním z modelů životního cyklů vývoje softwaru. Tyto modely specifikují, jakým způsobem se má během vývoje postupovat, aby se dosáhlo co nejlepších výsledků za co nejkratší dobu a s nejnižším úsilím. Všechny modely spojují základní etapy: analýza požadavků, návrh, implementace, testování, vydání a údržba. Model životního cyklu říká, ve které etapě by se vývojáři měli nacházet na základě aktuálního stavu vývoje. Blíže se tímto tématem zabývá obor softwarového inženýrství, zde je uveden jen výtah základních modelů.

Vodopádový model
Tento model je nejjednodušší a spočívá v sekvenčním průchodu etapami od první do poslední (ve výše daném pořadí) [6]. Do další etapy se vždy vkročí až ve chvíli, kdy předchozí etapa je již hotová. Pokud se při práci v jedné etapě zjistí chyba v předchozí, vrátí se do této etapy a v ní se potřebné nedostatky opraví. Tento model může být velmi efektivní, neboť vyřešené nedostatky v brzkých etapách nezpůsobí velká zdržení v těch pozdějších. Důležitou prerekvizitou k této efektivnosti je ovšem nutnost dovést předchozí etapu k dokonalosti, což je spíše nepravděpodobné.

V-model

V-model rozšiřuje vodopádový model přiřazením testovací fáze ke každé vývojové fázi, čímž zdůrazňuje provázanost mezi těmito fázemi [3]. Diagram se poté kreslí do tvaru písmene V, což mimo jiné zdůrazňuje zaměření modelu na verifikaci a validaci.

Iterativní a inkrementální model

Jak název napovídá, model je založený na posloupnosti etap, které se během vývoje systému opakují [12]. V každé iteraci se vytvoří funkční část výsledného produktu, který se ovšem postupně rozšiřuje a upravuje na základě zpětné vazby zákazníka. Tento model přináší rychlejší odhalení chyb či nedorozumění mezi zákazníkem a vývojářem, ovšem je složitější na řízení a může zapříčinit nedostatky ve struktuře systému. Jednou z variant modelů je tzv. spirálový model [2], který vymezuje 4 kvadranty vymezující základní činnosti. Během každého cyklu se analyzují rizika, která by mohla projekt ohrozit, a reakce na tato rizika. Tento model je ovšem velmi komplexní a vyžaduje na řízení velký tým odborníků.

U každého typu softwaru je náplň jednotlivých etap trochu odlišná. U mobilních aplikací je náplň následující.

Analýza požadavků

Tato fáze je vesměs pro všechny typy vývoje softwaru stejná – hlavním cílem je získat co nejpřesnější představu o podobě výsledného systému. Způsob získání těchto znalostí se může lišit v závislosti na povaze celého projektu, tedy pokud se například jedná o zákaznický software, primárním zdrojem informací by měl být zákazník. Naopak v případě startupu či obecně vývoje „pro sebe“ je důležité, aby si sami vývojáři vytvořili přesnou představu o výsledném produktu, u kterého poté provedou validaci, zda by o takový produkt byl zájem. Klíčovou roli v této etapě hrají dotazníky či rozhovory s potenciálními uživateli. U tohoto druhu vývoje je také důležitý výzkum trhu, aby vývojáři zbytečně nevyvíjeli produkt, který již existuje a je funkční.

Návrh

Při návrhu se využijí data získaná při analýze k ujasnění celé koncepce, dekompozice systému, návrhu datového modelu a výběru vhodných prostředků pro vývoj. Během této fáze se hojně využívá jazyk UML pro modelování ERD (Entity Relationship Diagram), diagramů případů užití apod. Pro ujasnění cílového uživatele se používají persony (či osoby)[10], což je fiktivní člověk s konkrétními vlastnostmi, na kterého je produkt zaměřen.

U mobilních aplikací také hraje velkou roli návrh uživatelského rozhraní (anglicky *User Interface*, zkráceně *UI*). Často se začíná tzv. wireframem, neboli drátěným modelem, což je hrubý náčrt navigace a funkcí v aplikaci. Wireframe se poté transformuje na mockup, tedy grafický návrh reflektující výslednou aplikaci. Mockup poté může sloužit na uživatelské testování, neboť nástroje, jako je např. Figma[11], podporují interaktivní návrh. Upravený mockup se poté dá využít jako předloha pro implementaci.

Implementace

Podle návrhu je poté možné implementovat aplikaci. Součástí úspěšné implementace je také volba správného implementačního nástroje, která je blíže popsána níže. Mobilní aplikace je

[10] Více o personách na Wikipedii: odkaz.
[11] Více o nástroji Figma na oficiálních stránkách: odkaz.

možné ladit pomocí emulátoru chytrého telefonu v počítači – pro aplikace na Android slouží Android Studio, ve kterém je možné vybrat z desítek virtuálních zařízení. Další možností je připojení skutečného Android telefonu pomocí kabelu a ladit přímo na něm. U aplikací pro iOS podobnou funkci zajišťuje Xcode.

U implementace je také dobrou praxí využití nějakého návrhového modelu. U mobilních aplikací je asi nejčastějším návrhovým modelem MVC – Model, View a Controller. V této architektuře *model* uchovává data, *controller* s nimi pracuje a *view* pouze vykresluje. Tento systém dodává programátorovi představu, jak by měl kód modularizovat, aby byl lépe udržitelný a testovatelný.

Testování

Před vydáním každé verze aplikace by mělo proběhnout testování. Během něj se testuje především správná funkcionalita, uživatelská přívětivost, kompatibilita napříč cílovými platformami, výkonnost, bezpečnost a další. U mobilních aplikací je běžná kombinace manuálního a automatického testování. Manuální testování mohou provádět samotní vývojáři, ovšem nejlepší je vždy zapojit do testování uživatele, kteří budou aplikaci zkoušet v praxi a dodávat zpětnou vazbu. Různé funkce a části aplikace je pak možné testovat automaticky pomocí unit testů[12].

Vydání a údržba

Po dokončení testování a odladění celé aplikace je aplikace vypuštěna do světa. Pro Android aplikace je typický Google Play Store, pro iOS App Store. Pro vydání na oba obchody je třeba si vytvořit placený vývojářský účet a nahrát instalační soubor aplikace. Tím ovšem cyklus nekončí a nastává proces údržby, během kterého je vhodné opravovat chyby nalezené uživateli a udržovat celý systém aktuální a spolehlivý.

2.4.2 Implementační nástroje

Jak již bylo řečeno, pro aplikaci je důležité zvolit vhodný implementační nástroj, neboť má dopad na:

- způsob a náročnost implementace,
- čas a cenu vývoje,
- výkon a rychlost aplikace,
- cílové platformy, operační systémy a hardware,
- a mnoho dalšího.

Volba prostředků tedy závisí na těchto faktorech a podle nich se také dělí 4 základní typy přístupů k vývoji mobilní aplikace:

- Nativní aplikace: Jsou psané přímo pro konkrétní platformu, tedy využívají platformou daný jazyk, její API (*Application Programming Interface*, tedy rozhraní funkcí systému, které jsou volány) a poté běží přímo na operačním systému dané platformy. Výhodami psaní nativních aplikací je především jejich vysoký výkon a přímý přístup

[12] Více o unit testech na Wikipedii: odkaz.

k API. Nevýhodou ovšem je, že takové aplikace je nutné psát pro každou platformu kompletně odděleně, což zvyšuje časovou náročnost a tím pádem i cenu. Příkladem nativních programovacích jazyků pro Android je Java nebo Kotlin, iOS využívá vlastní jazyk Swift.

- Multiplatformní aplikace: Jsou psané pomocí frameworku, který umožňuje překlad do nativního kódu více platforem. Tímto způsobem je možné aplikaci napsat jen jednou a vydat na více platforem najednou. Frameworky často zjednodušují vývoj i jinými způsoby, např. Flutter disponuje funkcí hot reload, která během okamžiku promítne změny v kódu do již běžící aplikace. Více o přínosech frameworku Flutter je v kapitole 2.4.3. Vzhledem k další vrstvě kódu je ovšem u multiplatformních aplikací stále trochu nižší výkon, než-li u těch nativních. Dalším potenciálním nedostatkem jsou možná omezení přístupu k API cílové platformy, neboť frameworky často za účelem zjednodušení jejich volání některé funkce znepřístupní [26]. Příkladem může být již zmíněný Flutter či React Native [21] .

- Hybridní aplikace: Tyto aplikace jsou psané pomocí webových technologií a poté je výsledná aplikace obalena jádrem webového prohlížeče. Tímto způsobem se multiplatformnost aplikace rozšiřuje i na web, což může zjednodušit vývoj v případě, kdy je třeba vytvořit mobilní i webovou aplikaci. Hlavní nevýhodou je ovšem slabší výkon a větší velikost aplikace, které jsou ovlivněny přibaleným kódem. Příkladem může být obyčejné HTML, CSS a JavaScript.

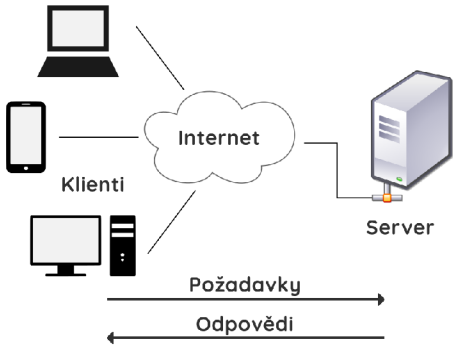
- Progresivní webové aplikace: Jsou podobné hybridním aplikacím, ale s rozdílem, že jsou přístupné pouze přes webový prohlížeč. Uživatel si může nastavit odkaz na stránku na domovskou obrazovku, díky čemuž se stránka bude poté chovat jako aplikace bez nutnosti instalace. Tento způsob je ovšem z hlediska přístupu k funkcím zařízení nejvíce omezený a podobně jako hybridní aplikace je výsledný výkon spíše nižší. Výhodou oproti ostatním způsobům je ovšem téměř nulová zátěž na úložiště zařízení, neboť je stále na nejnižší úrovni „jen“ webová stránka.

Architektura klient-server

Doposud zmíněné nástroje jsou určeny k tvorbě tzv. frontendu[13], tedy části systému, který je umístěn na zařízení uživatele a se kterým interaguje. Jedná se především o uživatelské rozhraní, navigaci v aplikaci a základní aplikační logiku. Některé jednoduché aplikace si s touto částí vystačí, ovšem většina používaných aplikací je součástí architektury klient-server [9]. V této architektuře je k frontendu (tedy klient) připojený ještě tzv. backend, který je umístěn na serveru a je společný pro všechny klienty. Oba koncové body spolu komunikují přes internet – klient posílá žádosti na server, ten požadavek zpracuje a odpoví. Tyto požadavky se posílají pomocí API a zpravidla se u tohoto typu komunikace využívá protokol HTTP[14]. Tato architektura se typicky využívá u aplikací, které ke své funkcionalitě vyžadují internetové připojení, ať se jedná o webový prohlížeč, internetové bankovnictví či sociální síť. Obrázek 2.2 zobrazuje schéma komunikace této architektury.

Hlavními důvody použití této architektury je centralizace, perzistence a ochrana dat. Například u internetového bankovnictví je částka na účtu uložena na jednom místě v databázi, ke které backend přistupuje a zajišťuje, že částka je při každé transakci či zobrazení

[13] Více o frontendu a backendu na Wikipedii: odkaz.
[14] Více o HTTP na Wikipedii: odkaz.



Obrázek 2.2: Schéma komunikace klient-server. Základ obrázku byl převzat z Wikipedie a upraven.

hodnoty aktuální. Kdyby tato částka byla uložena u klienta, nebylo by možné např. strhnout peníze z účtu, když klient není online. Navíc by tato metoda byla nebezpečná, neboť by uživatel mohl data ve svém telefonu přepsat či ztratit. Tato metoda také umožňuje jednoduché sdílení dat mezi dvěma klienty bez nutnosti spojení klientských zařízení způsobem peer-to-peer[15]. Díky tomu mohou fungovat např. aplikace na posílání zpráv. Backend má však mnoho dalších využití, která nemusí nutně souviset s daty – například hosting webové stránky, autentizace, odesílání emailů, přesun náročných výpočtů z klienta na server, automatické úlohy a podobně.

Při tvorbě backendu je výběr implementačního nástroje také důležitý, ovšem neplatí zde problém s cílovou platformou, neboť backend je na cílové platformě nezávislý. Pro tvorbu backendu je možné využít specializovaný framework (např. Node.js[16]) nebo již existující řešení typu BaaS[17] nebo SaaS[18]. Tato řešení fungují na principu již existujícího základního backendu, na kterém si vývojář – většinou pomocí grafického rozhraní – nastaví databázi a základní bezpečnostní pravidla pro její přístup. Většina těchto řešení disponuje možností rozšíření o vlastní část backendu, kterou může být například automatické zpracování dat bez požadavku klienta. Pro mnoho mobilních aplikací, kterým stačí pouze základní přístup k databázi bez nutnosti přídavné aplikační logiky, tyto nástroje postačí. Příkladem BaaS je Firebase[19] a příkladem SaaS je PocketBase[20], který je použit i v této práci a je blíže popsán v kapitole 2.4.4.

Backend je občas vnímán jako část systému, která pouze ukládá data. V takovém případě není u mobilních aplikací nutné využívat architekturu klient-server, ale je možné využít lokální úložiště zařízení (např. Android[21]). Aplikace s takovým backendem nekomunikuje přes HTTP, ale klasické API operačního systému podobně jako u ostatních funkcí. Takové využití má výhodu v nepotřebě internetového připojení, ovšem přináší řadu rizik co se týče bezpečnosti a ubírá na možné funkcionalitě celého systému.

[15] Více o peer-to-peer na Wikipedii: odkaz.
[16] Více o Node.js a dalších FW zde: odkaz.
[17] Více o BaaS na Wikipedii: odkaz.
[18] Více o SaaS na Wikipedii: odkaz.
[19] Více o Firebase na oficiálních stránkách: odkaz.
[20] Více o PocketBase níže či na oficiálních stránkách: odkaz.
[21] Více o local storage zde: odkaz.

2.4.3 Flutter

Flutter je open source SDK (Software Development Kit[22]) pro tvorbu multiplatformních aplikací. Je zaměřený na operační systémy Android a iOS, ovšem v dnešní době podporuje i webové a desktopové aplikace (Windows, Linux i macOS). Tento SDK přináší knihovnu widgetů (popsáno více níže) a základních prostředků pro psaní aplikace, renderovací engine, dokumentaci a nástroj pro překlad a správu projektu. Flutter využívá programovacího jazyka Dart (popsáno více níže).

Historie Flutteru

Flutter vyvíjí společnost Google a první verze se objevila v roce 2015 pod názvem "Sky", které ovšem cílilo pouze na OS Android. V květnu 2017 byla vydána alfa verze Flutteru společně s jeho první komerční aplikací a na konci roku 2018 poté verze 1.0. V březnu 2021 byla vydána verze 2, ve kterém mimo jiné přibyla poprvé předběžná podpora desktopových aplikací a lepší kompatibilita pro web. Ve stejné verzi vznikla také velká změna u jazyka Dart, u kterého přibyla podpora null safety (více v následující podkapitole). V květnu 2022 ve verzi Flutteru 3 byla podpora desktopových aplikací označena za stabilní.

V dnešní době je vydáno přes milion aplikací vyvinutých ve Flutteru [17]. Mezi patří především mnoho aplikací od Googlu, jako je např. Google Pay či Google Earth. Mezi další příklady můžeme zařadit např. eBay či BMW aplikaci [18].

Jazyk Dart

Programovací jazyk Dart je objektově orientovaný vysokoúrovňový jazyk vyvinutý Googlem [1]. První verze vydaná v roce 2011 byla určena především pro webový vývoj jako alternativa k jazyku JavaScript, kterému je mimo jiné i podobný syntaxí, částečně připomíná i C++. Dart se překládá přímo do strojového kódu cílového zařízení či právě do JavaScriptu. Velmi užitečnou funkcí jazyka je *hot reload*, která vyžaduje virtuální stroj (Dart VM) ke svému běhu, ovšem umožňuje velmi rychlé promítnutí nového kódu do výsledné aplikace bez nutnosti jejího kompletního překladu. Dart je možné kromě tvorby UI např. právě ve Flutteru využít i na tvorbu webových serverů (tedy backendu). Zde si uvedeme základní přehled tohoto jazyka.

Proměnné, typy a operátory

Dart je typovaný jazyk [16]. Při definici proměnných je možné typ vynechat, poté je odvozen od původní hodnoty (klíčové slovo `var`, `final` či `const`). Je možné ovšem deklarovat i dynamický datový typ pomocí `dynamic`, taková proměnná má poté iniciální hodnotu `null`. Zajímavostí o tohoto jazyka je již zmíněné klíčové slovo `final`, které se chová podobně jako konstanta, ovšem s tím rozdílem, že její hodnota není známá při překladu, ale až za běhu programu (často se využívá u parametrů tříd). Dart dále implementuje *null-safety*. Každá proměnná může být označená jako *nullable*, tedy je možné, aby nabývala hodnoty `null`. Nullable hodnota je označena otazníkem (např. `int?`) a Dart umožňuje odhalit chyby při práci s těmito hodnotami již při překladu. Již zmíněná `dynamic` proměnná je automaticky nullable a při neuvedení původní hodnoty je automaticky `null`. Dalším užitečným nástrojem jazyka je klíčové slovo `late`, které umožňuje deklarovat proměnnou bez její inicializace, která proběhne později. Inicializaci ovšem není možné zkontrolovat, hrozí tedy

[22] Více o SDK na Wikipedii: odkaz.

práce s neinicializovanou proměnnou, proto se využití tohoto klíčového slova nedoporučuje (alternativou je opět nullable hodnota).

Dart implementuje základní datové typy (`int`, `bool` apod.), datové struktury a objekty (např. seznam `List` či mapa `Map`) i pokročilé datové typy typické pro asynchronní operace (např. `Future`) [16].

Operátory jsou podobné jazyku C++, ovšem existují v Dartu další operátory. Za zmínku stojí např. operátory pro již zmíněné nullable hodnoty. Zde operátor `A ?? B` znamená „vrať hodnotu *A*, pokud není null, jinak hodnotu *B*“, operátor `A!` říká „hodnota *A*, přestože je nullable, určitě v tuto chvíli není null“ a operátor `A?.Volání()` říká „pokud je hodnota *A* null, nic nedělej, jinak proveď *Volání* na *A*“. Mezi další zajímavé operátory můžeme zařadit např. operátor `A is T`, který vrací `true` v případě, že hodnota *A* je typu *T*.

Třídy

Jak bylo řečeno, Dart je objektově orientovaný jazyk s širokou škálou možností dědičnosti a modifikace tříd [16]. Atributy třídy jsou implicitně veřejné, privátních atributů se docílí přidáním podtržítka před jejich jméno. Syntaxí se dá jazyk v tomto odvětví opět přirovnat k jazyku C++. Při předávání objektu parametrem se využívá metoda pass-by-reference, tedy předá se reference na objekt a je poté možné pracovat přímo s původním objektem. Užitečným nástrojem u tříd je klíčové slovo `extends`, tedy rozšíření nějaké třídy o vlastní metody či atributy. Využití najdeme především u vestavěných tříd, neboť tímto způsobem je možné např. nadefinovat vlastní metody pro `String` a není tedy nutné vytvářet nové třídy jako *MyString* apod.

Funkce

Funkce jsou v Dartu také objekty s typem Function. Není nutné uvádět návratový typ funkce ani typy parametrů, ovšem je to doporučeno. Anotace funkcí vypadají opět stejně jako u jazyka C++. Dart umožňuje anotaci anonymních (či lambda) funkcí, tedy funkcí beze jména definované přímo v místě použití (např. jako předaný parametr). Dále jazyk umožňuje zkrácený zápis funkce pomocí znaků `=>`. Uvedené dvě vlastnosti jsou při programování aplikací velmi užitečné, neboť předávání callback funkcí je velmi časté a často se jedná pouze o krátké kusy kódu, pro které by bylo zbytečné vytvářet plnohodnotnou funkci.

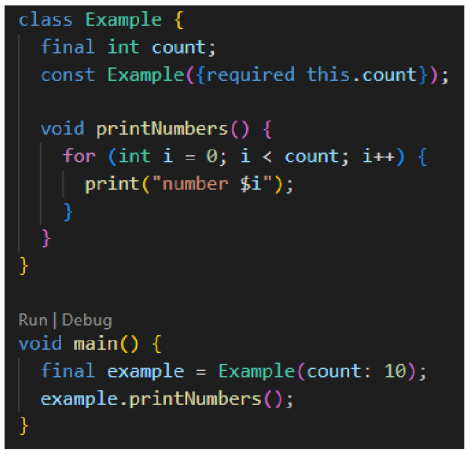
Dalším užitečným nástrojem u funkcí (i objektů) je možnost výběru mezi jmennými či pozicovými parametry. Pro jmenný parametr je nutné jej obalit do složených závorek, implicitně jsou parametry pozicové. Tyto způsoby je možné v kombinovat. Je také možné rozlišit, jestli jsou parametry povinné či ne – jmenných parametrů pomocí klíčového slova `required`, u pozicových je nutné tyto parametry obalit hranatými závorkami. U všech nepovinných parametrů je však třeba uvést implicitní hodnotu či tyto parametry označit jako nullable.

Jednoduchý program v Dartu

Pro základní představu je na obrázku 2.3 uveden velmi jednoduchý program v Dartu. Tento program obsahuje třídu `Example`, která požaduje povinný parametr `count` a po zavolání metody `printNumbers` vypíše – v tomto konkrétním případě – *number 0* ... *number 9*, každé číslo na samostatný řádek.

Tvorba uživatelského rozhraní pomocí widgetů

Existují dva základní způsoby tvorby uživatelského rozhraní – procedurální a deklarativní [27]. Alternativními řešeními může být například tvorba UI pomocí jiného grafického roz-



Obrázek 2.3: Příklad jednoduchého programu v Dartu

hraní (příkladem může být Qt[23].), ovšem tyto způsoby jsou často pouze grafickým rozhraním pro jeden ze dvou základních způsobů.

U procedurálního programování programátor krok po kroku popisuje, jak se rozhraní má sestavit. Příkladem procedurálního frameworku pro tvorbu UI je PyGTK[24]. Tento způsob může přinášet výhodu při ladění, ovšem může přinášet obtíže při orientaci v kódu a další nevýhody.

Deklarativní tvorba UI je častější, známým výskytem je HTML. Programátor u tohoto způsobu neřeší, jak a kdy se co vykreslí, ale pouze specifikuje, jak má výsledek vypadat. Pro tuto metodu je typické zanořování prvků UI do sebe, čímž vzniká stromová hierarchie, která se často reprezentuje tzv. Document Object Modelem (zkráceně DOM). HTML je kombinováno s CSS (Cascading Style Sheets), tedy specifikace stylů, jak má který prvek vypadat, napsané také deklarativně. Pro jisté imperativní prvky se poté starají skripty napsané většinou v jazyce JavaScript.

Flutter využívá k tvorbě UI widgety, což jsou předpřipravené upravitelné prvky uživatelského rozhraní. Část výsledného kódu je deklarativní, část imperativní, ovšem všechny části jsou napsané jedním jazykem – Dart (na rozdíl od HTML, CSS a JavaScriptu). Flutter se řídí myšlenkou „vše je widget“, tedy od běžných UI prvků, jako jsou tlačítka či texty, přes modifikátory rozložení, jako je zarovnání či roztažení, až po asynchronní operátory[25]. Každý widget je objekt a je tedy možné jej parametrizovat. Většina widgetů disponuje parametrem `child`, do kterého se přiřadí potomek, čímž vzniká ona stromová hierarchie. Většina widgetů má více parametrů, které poté slouží na upřesnění chování widgetu či nastavení stylu. Styly sice nejsou widgety, ovšem stále jsou to parametrizovatelné objekty.

Flutter samozřejmě umožňuje tvorbu vlastních upravitelných widgetů složením zabudovaných widgetů, čímž efektivně vznikají znovupoužitelné komponenty. Widgety se dělí na dva typy - *stateless* a *stateful*, tedy *bezstavové* a *stavové*. Stavové widgety umožňují programátorovi měnit stav widgetu za běhu, což má mnohé využití, např. při najetí myší na tlačítko se toto tlačítko jinak zabarví. Při vykreslení widgetu frameworkem se u každého widget volá metoda `build`, která vrací typ `Widget`. V této metodě je možné nejprve pro-

[23] Více o Qt na oficiálních stránkách: odkaz.

[24] Více o PyGTK: odkaz.

[25] Widgety, které řeší asynchronní operace, jako je načítání dat ze serveru, a na základě stavu budují uživatelské rozhraní.

vést operace nutné pro vykreslení (např. předpřipravení některých dat) a na konci je vždy návrat zanořených widgetů pomocí `return`. Každá `build` metoda požaduje jeden parametr `context` typu `BuildContext`. Tato proměnná popisuje aktuální stav stromu widgetů a během změny UI se kontext mění, proto je nutné tento kontext neustále předávat a za jistých okolností i kontrolovat, že tento kontext je aktuální a je možné do něj zasahovat.

Zde je uveden výtah základních widgetů a dalších prvků, které jsou důležité především pro tuto práci, ovšem pokrývají vše potřebné pro základ Flutteru. Následuje příklad tvorby vlastního widgetu.

Základní widgety

- `MaterialApp`: Slouží k obalení celé aplikace využívající Material Design a poskytuje základní funkcionalitu a výchozí styly a nastavení pro aplikaci.
- `Scaffold`: Poskytuje základní vizuální strukturu pro aplikaci a dodává možnost vykreslovat různé navigační prvky.
- `AppBar`: Vytváří nabídku v horní části obrazovky. Je možný v ní nastavit např. `Drawer`, tedy ikonku pro otevření nabídky pro navigaci a další akce v aplikaci.
- `BottomNavigationBar`: Vytváří navigační nabídku v dolní části obrazovky.
- `Text`: Klasický text s nastavitelným stylem.

Widgety rozložení

- `Row` a `Column`: Umístění seznamu widgetů do řádku, respektive sloupce. Obalením `Row` či `Column` pomocí `SingleChildScrollView` se dosáhne skrolovacího obsahu, v opačném případě by při velkém počtu položek v seznamu došlo k přetečení mimo obrazovku. Užitečným parametrem těchto widgetů je způsob rozmístění na hlavní a vedlejší ose.
- `Expanded`: Používá se v `Row` nebo `Column` a slouží k roztažení potomka na maximální možnou velikost.
- `Container`: Základní obdélníkový obal s nastavitelným pozadím, tvarem a stylem. Je možné specifikovat rozměry a tím pádem obálku pro jeho potomka, ovšem pro tento účel slouží spíše ořezaná verze Containeru `SizedBox`.
- `Align`: Zarovnání potomka v dané oblasti způsobem definovaným v parametru. Pro zarovnání na střed je možné využít widget `Center`.
- `Padding`: Přidá volný prostor okolo potomka.

Vstup

- `ElevatedButton`: Tlačítko s širokou škálou nastavení vzhledu a chování. Alternativou je `TextButton` či `IconButton`, tedy stisknutelný text či ikona.
- `GestureDetector`: Univerzální widget pro detekci dotyků obrazovky. Pomocí tohoto widgetu je možné nejen z jakéhokoli widgetu udělat stisknutelný, ale umožňuje detekovat i jiné vstupy, jako je dlouhé podržení či dvojité kliknutí. Umožňuje také rozlišit všechny fáze stisku, jako je začátek stisku, konec stisku či opuštění oblasti detekce během stisku, a na každou událost definovat reakci.

- `TextFormField`: Textové pole umožňující zadání hodnoty a následná kontrola či manipulace s hodnotou. Umožňuje širokou škálu nastavení vzhledu.
- `Checkbox`: Klasické tlačítko pro označení položek v seznamu.
- `DropdownButton`: Tlačítko pro výběr z předdefinovaných položek.

Ostatní widgety

- `FutureBuilder`: Slouží k provedení asynchronní operace a na základě stavu této operace vybudovat dané UI. Pomocí tohoto widgetu je možné zobrazit načítání, zatímco probíhá operace, a po dokonání operace zobrazit data, případně informaci, že se operaci nepodařilo provést (např. kvůli nefunkčnímu připojení k síti).

Třídy pro tvorbu stylů

Následující třídy sice nejsou widgety, ovšem hrají při tvorbě uživatelského rozhraní důležitou roli, zejména z grafického hlediska.

- `TextStyle`: Nastavení stylu textu, jako je velikost písma, font či barva.
- `Color`: Třída pro barvy. Nabízí možnost vytvoření barvy pomocí různých barevných modelů i nabídku předdefinovaných barev.
- `BoxDecoration`: Slouží k nastavení stylu `Container`u.
- `InputDecoration`: Dekorace vstupních widgetů, jako je `TextFormField` či `DropdownButton`.

Příklad vlastního widgetu

Tvorbu vlastního widgetu (respektive komponenty) si představíme na následujícím příkladu na obrázku 2.4. Jedná se widget přímo z výsledné aplikace, který zobrazuje danou hodnotu (`value`) a nad ní název této hodnoty (`label`) méně výrazným písmem. Dále je možné specifikovat nepovinný parametr `hint`, který se po kliknutí na pole zobrazí ve vyjížděcím dialogu (tělo funkce pro zobrazení dialogu je pro jednoduchost skryté).

`GestureDetector` zde slouží pro zobrazení nápovědy. `Container` s transparentním pozadím a maximální šířkou roztahuje plochu pro detekci dotyku na celý povrch widgetu (bez něj by `GestureDetector` detekoval pouze dotyky na samotný text). Parametr `crossAxisAlignment` určuje zarovnání na sekundární ose, konkrétně `start` zde udává zarovnání na začátek, tedy vizuálně doleva. Texty mají poté specifikované textové styly a pomocí metody `copyWith` upravené barvy textu.

Navigace a stav aplikace

Uživatel během používání aplikace neustále přechází mezi stránkami a mění na nich různá data, čímž dochází ke změnám stavu aplikace. Pro tyto účely má Flutter zabudovaných několik prostředků.

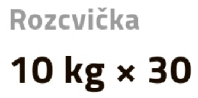
Nejjednodušším a nejuniverzálnějším způsobem správy stavu je volání funkce `setState`, která je dostupná ve stavových widgetech. Tato funkce požaduje jediný parametr, kterým je callback funkce, která provede změny stavu (např. se změní hodnota proměnné) a poté překreslí tento widget a jeho potomky s aktualizovanými daty. Tato metoda je vhodná především pro případ, kdy změny ve widgetu jsou pouze lokální a po zmizení widgetu

```dart
class LabeledValue extends StatelessWidget {
  final String label;
  final String value;
  final String? hint;

  const LabeledValue({
    super.key,
    required this.label,
    required this.value,
    this.hint,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: hint != null
          ? () {
              showHint(context);
            }
          : null,
      child: Container(
        color: Colors.transparent,
        width: double.infinity,
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              label,
              style: TextStyles.extraSmallBold.copyWith(
                color: GreyColors.dark,
              ),
            ), // Text
            Text(
              value,
              style: TextStyles.standardBold.copyWith(
                color: TextColors.main,
              ),
            ), // Text
          ],
        ), // Column
      ), // Container
    ); // GestureDetector
  }
```

(a) Implementace



(b) Výsledná podoba

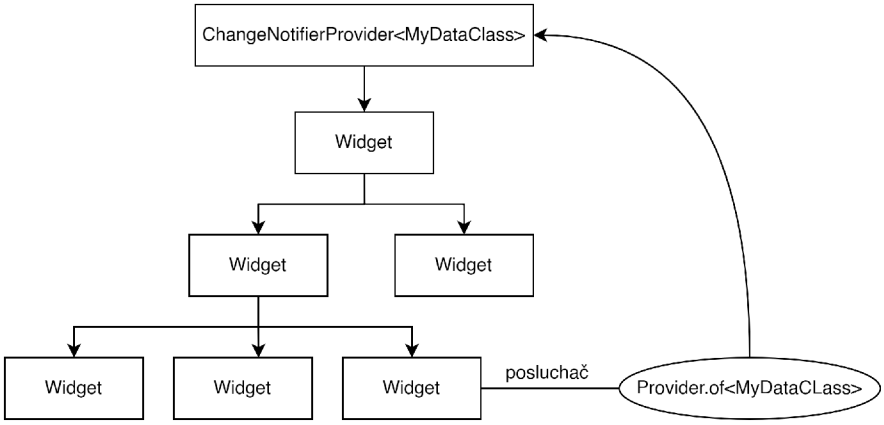
Obrázek 2.4: Příklad bezstavového widgetu ve Flutteru

můžeme tato data bez problémů zahodit. Může se jednat o změnu vzhledu widgetu po najetí myši nebo je také vhodná pro top-level navigaci (výběr aktuální stránky z menu v dolní části obrazovky).

Pro navigaci na samotných stránkách je vhodná např. třída `Navigator`, která pomocí metod `push` a `pop` vkládá stránky do popředí aplikace, respektive je od tam odebírá. Disponuje také možností nastavení animace pro přechod. Výhodou této metody je historie, neboť nová stránka se vždy vloží či odebere – jak názvy metod napovídají – na zásobník stránek. Je tedy možné se po otevření stránky vrátit na tu původní pomocí tlačítka *zpět* na zařízení. Alternativou k `Navigatoru` je např. *GoRouter*, který funguje na bázi URL. Tento přístup je vhodný především pro webový vývoj ve Flutteru, neboť dodává uživateli zkušenost z klasických webových stránek, s možností historie, kterou prohlížeč implementuje.

U mobilních aplikací je však nutné tuto historii naimplementovat zvlášť, což z GoRouteru dělá neuniverzální řešení.

Dalším prvkem stavu aplikace jsou kromě navigace také sdílená data. Příkladem sdílených dat může být nákupní košík. Uživatel si v aplikaci prohlíží produkty a když se mu nějaký zalíbí, přidá si jej do košíku. Tento košík ovšem nemůže být uložený přímo u tlačítka (které s košíkem manipuluje), protože tlačítka spolu nekomunikují a navíc by data byla po zmizení tlačítka ztracena. Tato situace by se dala vyřešit uložením košíku na vrcholu stromu widgetů a poté by byla tlačítkům předána callback funkce, která by ve vrchním widgetu volala zmíněný `setState`. Tento způsob má ovšem problém především v propagaci této callback funkce, která by u větší aplikace způsobovala chaos. Od toho ve Flutteru existuje řada řešení na správu stavu, jedním z nich je např. Provider [19]. Technika spočívá ve vytvoření třídy, která obsahuje sdílená data (např. `MyDataClass`) dědící od třídy `ChangeNotifier` a poté instanciace této třídy na vrcholu stromu widgetů s pomocí speciálního widgetu. Pro přístup k datům je pak možné na libovolném místě v aplikaci zavolat `Provider.of<MyDataClass>(context)`, čímž dostaneme instanci této třídy a můžeme s daty manipulovat. Tato metoda by se dala přirovnat ke globální proměnné např. u jazyka C. Výhodou této metody je také fakt, že je možné poslouchat změny, které na datech proběhly, a na základě změny aktualizovat stav widgetu. Třída, od které `MyDataClass` dědí, implementuje metodu `notifyListeners`, která automaticky aktualizuje všechny widgety, které změny na datech poslouchají. V našem příkladu s nákupním košíkem by tato funkce byla využita u ikonky nákupního košíku, která by po přidání položky automaticky zvýšila počet položek o jednu. Dalším možným využitím je stav přihlášení, neboť po odhlášení je záhodno uživatele automaticky přesměrovat na přihlašovací obrazovku a naopak po přihlášení na domovskou. Obrázek 2.5 ukazuje způsob funkce Provideru.



Obrázek 2.5: Příklad funkce Provideru ve stromě widgetů

Sestavení a spuštění aplikace

Framework přichází s velmi jednoduchým způsobem spuštění. Je možné využít zabudovaných funkcí vývojových prostředí, ovšem univerzální spuštění je možné přes příkazový řádek. Příkaz `flutter run -d device` sestaví a spustí aplikaci na zařízení `device` v režimu ladění. Toto zařízení může být kteréhokoli typu (emulátor, připojený telefon, webový prohlížeč, desktopový OS), ovšem musí být dostupné. Pro sestavení výsledné aplikace poté stačí zavolat `flutter build apk`, což sestaví instalační soubor ve verzi pro vydání (release).

2.4.4 Služba PocketBase

PocketBase je open source backend Saas (Software as a Service). Poskytuje základní funkce backendu, jako je autentizace uživatele, databáze s bezpečnostními pravidly a automatické odesílání emailů. PocketBase je možné využít i jako framework a tedy rozšířit již existující funkce o své vlastní. S relační databází[26] založené na SQLite se komunikuje přes REST API [30], tedy protokolem HTTP[27]. PB je tzv. self-hosted, tedy neposkytuje hosting a je nutné si jej zařídit samostatně stažením spustitelného souboru a zavoláním příkazu `pocketbase serve`, který spustí backend na lokální adrese na portu 8090, který je ovšem možné změnit. Poté na adrese a portu běží 3 služby:

- `http://ip_adresa:port` – veřejná webová stránka,
- `http://ip_adresa:port/_/` – konzole administrátora s grafickým rozhraním,
- `http://ip_adresa:port/api/` – adresa pro REST API.

Databáze a pravidla API

Databáze se skládá z tabulek, které se zde nazývají kolekce, které jsou složeny ze záznamů (records). Každá kolekce má – podobně jako u klasické relační databáze – název a záhlaví s atributy. Atributy mohou mít mnoho datových typů: *plain text*, *rich editor*, *number*, *bool*, *email*, *url*, *DateTime*, *select*, *file*, *relation* a *JSON*. Mnoho z těchto datových typů se u běžné relační databáze nevyskytuje, ovšem na nižší úrovni to jsou často stále atomické hodnoty. Velkou výhodou však tvoří datový typ *relation*, který umožňuje uložit odkaz na záznam z jiné tabulky (podobně jako cizí klíč u klasické relační databáze), ovšem s tím rozdílem, že databáze ví, na kterou kolekci se daná hodnota odkazuje. To umožňuje automatické rozšíření záznamu o odkazovaný při dotazování na daný záznam[28]. Další výhodou je možnost nastavení některé hodnoty jako vícenásobné. Typ *relation* je jedním z nich, což odstraňuje nutnost vazebních tabulek[29]. Další výhodou této databáze jsou operátory, které je možné při prohledávání využít. Za zmínku stojí operátor ∼, který je definován „jako“ či „obsahuje“ a je možné jej využít na vyhledávání podřetězců či ve vícenásobné hodnotě.

Další důležitou vlastností služby jsou pravidla přístupu k API. Vzhledem k tomu, že k API může přistoupit kdokoli, kdo jej zná, je nutné na backendu kontrolovat, jestli daný uživatel má patřičná oprávnění. K tomu slouží pravidla API, která jsou definovatelná pro každou kolekci zvlášť. U každé kolekce je pak možné nastavit samostatné pravidlo pro vyhledávání v kolekci a čtení, vytvoření, úpravu či smazání záznamu. Tato pravidla předchází především situaci, kdy by se uživatel pokusil přistoupit k datům někoho cizího, ovšem nejsou všemocná a pro pokročilejší aplikace či lepší úroveň zabezpečení je nutné řešení v podobě vlastního backendu. Většina pravidel využívá autentizační token, který nese informaci o stavu přihlášení. Příkladem je pravidlo `@request.auth.id = user` které říká, že pro povolení této operace (např. úprava) musí být přihlášený uživatel stejný, jako uživatel v záznamu (jinak řečeno záznam „patří“ uživateli). Ke konkrétní kolekci se pomocí API dostaneme přes již zmíněnou API adresu voláním `collections/nazev_kolekce/records` a ke konkrétnímu záznamu připojením `/:id` za tuto adresu. Typ požadavku je definován

[26] Více o relační databázi na Wikipedii: odkaz.

[27] Více o RESTu na Wikipedii: odkaz.

[28] Např. záznam z kolekce *posts*, tedy příspěvky, obsahuje odkaz na uživatele, který příspěvek přidal. Při dotazování na daný příspěvek je možné automaticky vrátit i data uživatele společně s daty z příspěvku.

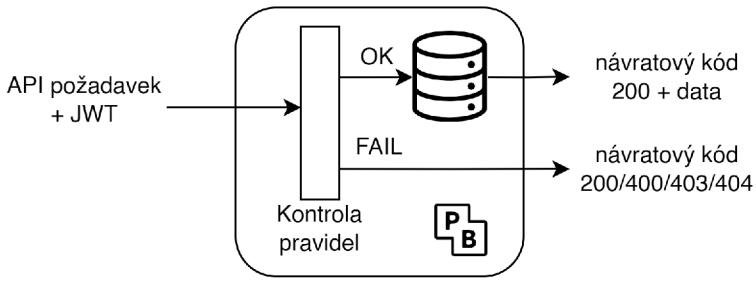
[29] Např. záznam z kolekce *trainings*, tedy tréninky, obsahuje seznam všech cviků, které byly cvičeny.

HTTP metodami, tedy GET provádí *list*, *search* nebo *view* (tedy čtení), POST provádí *create* (tedy vytvoření), PATCH provádí *update* (tedy úpravu) a DELETE provádí *delete* (tedy smazání) záznamu.

Autentizace

Důležitou funkcí i z hlediska zmíněného zabezpečení API je autentizace uživatele. PocketBase má zabudovanou speciální kolekci pro uživatele a umožňuje registraci několika způsoby – od klasického emailu a heslo po registraci přes jiného poskytovatele (např. Google) pomocí OAuth2[30]. Po přihlášení se uživateli vytvoří JWT (JSON Web Token)[31], který umožňuje jednoduchou a bezpečnou kontrolu autentizace uživatele, především u zmíněných bezpečnostních pravidel. PB mimo jiné disponuje možností nastavení emailového serveru, pomocí kterého je možné odesílat potvrzovací emaily či emaily na změnu hesla.

Obrázek 2.6 zobrazuje využití JWT u bezpečnostních pravidel při přístupu k API.



Obrázek 2.6: Schéma funkce pravidel API v PocketBase. V případě splnění pravidel (větev *OK*) je přistoupeno k databázi a jsou uživateli úspěšně vrácena data. V opačném případě (větev *FAIL*) je uživateli vrácen jeden z kódů, který závisí na typu chyby (popsáno v [30]). Obrázek obsahuje oficiální logo PocketBase.

[30] Více o OAuth na Wikipedii: odkaz.
[31] Více o JWT na Wikipedii: odkaz.

Kapitola 3

Návrh aplikace

3.1 Analýza a cíle

Jak bylo psáno v rešerši, většina aplikací zaměřené na zapisování tréninků používají k vyhledání cviku dlouhé seznamy obohacené o různorodé filtry. Cílem této práce je vytvoření alternativního a inovativního řešení problému vyhledávání cviků s pomocí technologie NFC, které funguje následujícím způsobem. Každý stroj a vybavení v posilovně bude mít na sobě připevněný NFC tag, který bude nést informaci o stroji. Primární informací budou dostupné cviky, které je možné na tomto stroji cvičit. Po načtení tagu bude uživateli zobrazen pouze seznam dostupných cviků (u většiny strojů se bude jednat o nízké jednotky) a uživatel si poté vybírá z tohoto seznamu. Tento systém dělá z dlouhého vyhledávání cviku v seznamu záležitost několika málo vteřin, má ovšem řadu dalších přínosů.
Výhody logovací aplikace využívající technologii NFC:

- rychlejší nalezení cviku v seznamu,
- možnost individuálního nastavení pro každý stroj i posilovnu a tím pádem přísun dodatečných informací a možností pro uživatele,
- vysoká flexibilita a pohodlnost zapisování při tréninku.

Nevýhody logovací aplikace využívající technologii NFC:

- nutnost rozmístění tagů a nastavení celého systému,
- nutnost používat speciální aplikaci, která systém podporuje,
- vyhledávání cviku neodstraňuje, v jistých případech stále musí uživatel hledat v dlouhém seznamu.

Rozmístěné NFC tagy a tato aplikace ovšem nemusí být uzavřený systém. Při návrhu systému bylo dbáno i na možnost využití aplikace a tagů zvlášť, tedy aby bylo možné použít aplikaci i bez přítomnosti NFC tagů, či naopak, aby bylo možné tagy načíst i v jiné aplikaci vybudované někým jiným.

Systém načítání NFC tagů přináší i možnost sledovat, které stroje byly právě uživatelem načteny. Tyto informace budou poté využity správcem posilovny, který může monitorovat využití jednotlivých strojů. Hlavními výhodami oproti alternativním možnostem monitorování je především fakt, že tato metoda je automatická, levná a univerzální. Je tedy možné

monitorovat jakýkoli stroj a to stále, navíc je tento systém jednoduše realizovatelný v jakkoli velké posilovně. Dále symbióza mezi uživatelem a správcem posilovny může podporovat rozšiřování aplikace mezi další uživatele a tím pádem i vyšší komerční potenciál v situaci, kdy by aplikace byla zmonetizována.

Primární cíle výsledného systému:

- funkční mobilní aplikace na zapisování tréninků,
- využívá NFC k načtení stroje,
- umožňuje správcům posiloven monitorovat vytížení strojů různými způsoby,
- aplikace je rychlá, pohodlná, ergonomická a uživatelsky přívětivá,
- systém NFC tagů je nezávislý na aplikaci.

V rámci analýzy také proběhlo malé dotazníkové šetření, které mělo za cíl validovat návrh a odhalit oblasti, na které by se aplikace měla zaměřit. Dotazník s výsledky je součástí přiloženého paměťového média.

3.1.1 Osoby

Za účelem upřesnění cílového uživatele vznikly dvě vymyšlené osoby či persony, které odráží především dva protipóly potenciálního uživatele.

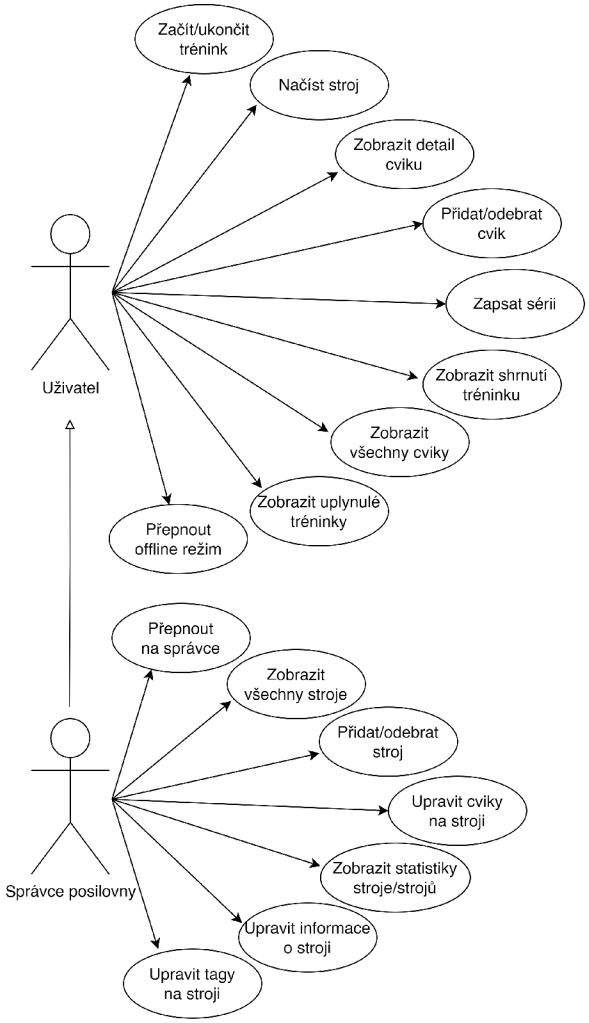
Petr
Věk: 22
Povolání: volejbalista
Zájmy: sport, čtení, vaření, astrologie
Petr je profesionální sportovec a cvičí v posilovně až 5krát týdně. Jsou na něj kladeny vysoké výkonnostní i časové nároky a tudíž si nesmí dovolit polevit. Petr má každý trénink jiný a během týdne procvičí všechny svalové partie s využitím mnoha strojů a vybavení. Vzhledem k množství cviků si nikdy nepamatuje, co minule cvičil, a rozhodl se své tréninky zapisovat. Také by rád věděl nějaké statistiky o svém tréninku, aby se mohl případně na něco zaměřit. Zkoušel různé mobilní aplikace, ovšem všechny jej od tréninku vyrušovaly neustálým hledáním cviků a chaotickým uživatelským rozhraním. Ocenil by mobilní aplikaci, která jej při tréninku nebude zdržovat ani vyrušovat a dodá mu potřebné prostředky pro sledování jeho vývoje.

Jana
Věk: 35
Povolání: účetní
Zájmy: finance, jízda na motorce, zahradnictví
Jana má sedavou práci a chce tento fakt občas vykompenzovat návštěvou posilovny. Sport ovšem není jejím zájmem a příliš informací o silových trénincích nemá. Také není největším nadšencem moderních technologií a nemá ráda, když si musí stahovat a používat nové mobilní aplikace. Vzhledem k nepravidelným návštěvám posilovny si však nepamatuje, jaké váhy si má na strojích zvolit. V posilovně si všimla tagů na strojích a zeptala se správce, o co se jedná. Na správcovo doporučení si aplikaci stáhla a začala používat. Díky přívětivému uživatelskému rozhraní pro zapisování a jednoduchému zobrazení historie začalo Janu cvičení v posilovně více bavit a navíc vždy ví, co přesně má cvičit.

3.2 Detailní specifikace

Výsledná aplikace bude mít dvě části – tréninkovou pro uživatele a monitorovací pro správce posilovny. Každý uživatel si vytvoří při prvním spuštění aplikace účet a poté se pomocí něj přihlásí. Každý uživatel může využívat tréninkovou část, autorizovaní správci posilovny budou mít možnost přejít do té monitorovací. Každá část má poté oddělenou navigaci, ovšem obě části mají společnou stránku pro profil, kde bude kromě možnosti odhlášení či přepnutí na druhou část i možnost ovládání offline režimu, neboť aplikace poběží primárně online.

Na obrázku 3.1 se nachází diagram základních případů užití. Tyto a další případy užití budou blíže popsány v následujících podkapitolách.



Obrázek 3.1: Diagram případů užití

3.2.1 Specifikace tréninkové části

V tréninkové části aplikace budou 3 hlavní navigační stránky: aktuální trénink, historie a profil. Vzhledem k tomu, že profil je společný pro obě části aplikace, budou v této části popsány pouze první dvě stránky.

Trénink

V této části aplikace bude uživatel začínat, zapisovat a ukončovat své tréninky. Po započetí vidí celý svůj trénink (tedy cviky a série, které cvičil) společně se shrnujícími statistikami o tréninku – celkové volume (popsáno v kapitole 2.1.2), počet sérií a čas od začátku tréninku. V tuto chvíli může kdykoli přiblížit telefon k NFC tagu a tím načíst stroj, k čemuž jej může navést nápověda na stránce. Alternativou pro výběr cviku je klasický seznam všech cviků. Po načtení stroje uživatel uvidí seznam všech cviků, které jsou na stroji či vybavení k dispozici a může jeden z nich zvolit. U strojů, u kterých je na výběr mnoho cviků, si může uživatel vypomoct filtry, které jsou:

- oblíbené či neoblíbené cviky,
- cviky, které již cvičil, případně které ještě necvičil,
- výběr svalové partie, s možností hledání pouze mezi primárními nebo i sekundárními partiemi.

Po výběru uživatel uvidí detail daného cviku, který zobrazuje:

- název a obrázek pro lepší identifikaci cviku,
- primární a sekundární partie, na které je cvik zaměřen,
- možnost přidat či přečíst poznámku,
- základní výběr z historie, jako jsou váhy a počty opakování z minulých tréninků,
- možnost pro přidání cviku do tréninku.

Po přidání se cvik objeví v seznamu a je možné přidávat a zapisovat série, které budou organizovány do tabulky. Cviky i série je možné i odstranit. U série je možné změnit její typ, na výběr je: normální série, rozcvička, dropset a vyčerpání (popsáno v kapitole 2.1.1). U série se vždy zapisuje váha a počet opakování, kterého je možné docílit čtyřmi způsoby:

- přímé zapsání do textového pole pomocí klávesnice,
- překopírováním hodnoty z předchozí série,
- překopírováním hodnot z minulého tréninku z odpovídající série (k tomu slouží speciální sloupec, který tuto hodnotu zobrazuje),
- pomocí posuvníku specializovaného pro daný stroj.

V případě, kdy chce uživatel cvičit více cviků najednou a přecházet mezi nimi, je možné cviky spojit do supersérie (cviky hned za sebou), případně rozpojit. Po dokončení daného cviku je možné jej minimalizovat. Během cvičení je také možné si cvik otevřít a znovu si prohlédnout jeho detail, podobně jako během přidávání.

Po dokončení tréninku je možné jej ukončit a tím uložit, případně smazat. Před ukončením tréninku uživatel uvidí shrnutí tréninku společně s grafem rozložení svalových partií, což může využít i během tréninku k rozhodnutí, který cvik dále zvolí, aby měl trénink vyvážený. Mimo to může trénink pojmenovat a přidat k němu poznámku.

Historie

Historie je rozdělena na dvě záložky: tréninky a cviky. V první záložce uživatel vidí seznam všech tréninků, které úspěšně ukončil, zobrazující základní informace o tréninku, jako je datum nebo jméno. Po zvolení konkrétního tréninku uvidí podobné shrnutí, jako při ukončování, rozšířené o podobný seznam cviků, jako u běžícího tréninku. Další funkcí této záložky jsou celkové shrnující statistiky všech tréninků, které zobrazují jejich celkový počet, týdenní průměr a rozložení svalových partií napříč tréninky. Záložka cviků zobrazuje podobný seznam jako je u výběru cviků u tréninku včetně možnosti filtrování. V detailu cviku je po otevření mimo jiné vidět i graf, který zobrazuje vývoj některé veličiny na daném cviku, jako je maximální váha či váha při rozcvičce. Tento graf je možné zobrazit i při přidávání cviků, ovšem implicitně je skrytý.

3.2.2 Specifikace společné části

Přihlášení a profil

V aplikaci se uživatel musí registrovat a přihlásit. Po úspěšném přihlášení stránka Profil zobrazuje aktuálně přihlášeného uživatele a možnost odhlášení. Správci posilovny se na této stránce zobrazuje tlačítko pro přepínání mezi částmi aplikace. Tato stránka také zobrazuje obchodní a kontaktní informace a základní informace o aplikaci.

Offline režim

Jak bude dále vysvětleno v kapitole 3.4, celý systém poběží primárně online. Již zmíněný offline režim umožňuje – jak název napovídá – používat aplikaci bez připojení k internetu. Hlavní motivací za vytvořením offline režimu je zpřístupnění aplikace i v posilovnách, kde nemusí být signál či WiFi. Data je možné zálohovat ručně nebo automaticky při přechodu zpět do online režimu.

3.2.3 Specifikace monitorovací části

Část aplikace pro správce posiloven má též 3 stránky: společný profil, správu vybavení a statistiky. Obě dvě stránky jsou velmi podobné, neboť obsahují seznam všech vytvořených strojů, každá stránka je však zaměřená na něco jiné.

Správa vybavení

Na stránce pro správu vybavení je možné opět načítat NFC tagy. Po načtení se v případě již přiřazeného stroje zobrazí jeho detail, v opačném případě se zobrazí nabídka pro vytvoření nového stroje. Stroj má následující parametry:

- název,
- minimální rozsah a krok závaží na stroji – tyto hodnoty slouží k vytvoření rozsahu dříve zmíněného posuvníku pro výběr váhy u cviku,
- fotka, která slouží primárně pro jednoduché rozpoznání strojů v seznamu správce, ovšem uvidí ji i běžný uživatel.

Po vytvoření stroje se zobrazí již zmíněný detail a správce může upravovat cviky. Dále je možné data o stroji upravit včetně přiřazení nových NFC tagů, což je užitečné zejména

u větších strojů, na kterých bude připevněno více tagů. Stroj je možné smazat. V detailu cviku je možné také zobrazit shrnující statistiky o stroji:

- celkové použití stroje,
- týdenní průměr použití stroje,
- graf s volitelným časovým úsekem zobrazující: celkové použití stroje, využití během týdne a využití během konkrétního dne v týdnu.

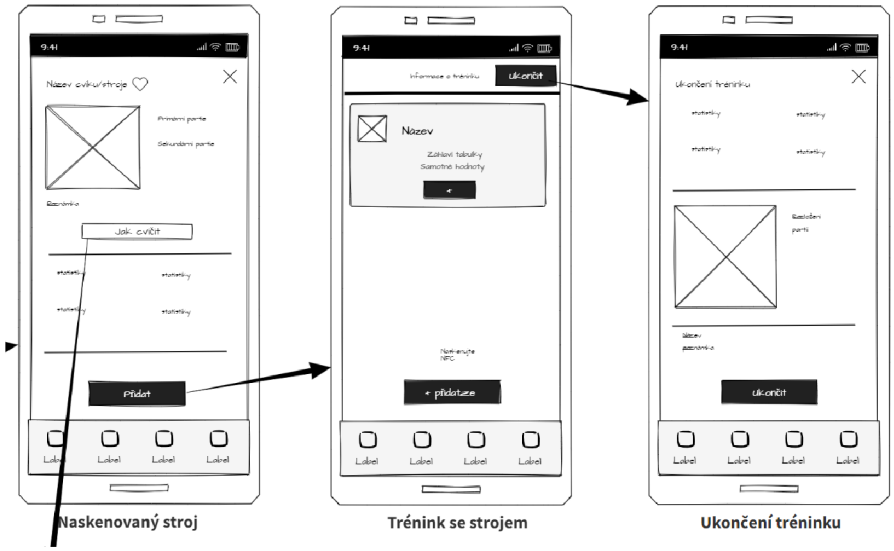
Statistiky

Stránka pro statistiky slouží k porovnání všech strojů mezi sebou. Obsahuje seznam všech strojů, které jsou seřazeny od těch nejpoužívanějších a tvoří takto sloupcový graf, který graficky zobrazuje použití strojů v porovnání k ostatním. Jednotlivé stroje je možné též rozkliknout a zobrazit tak jejich detail, tedy případné podrobné statistiky či stroj upravit.

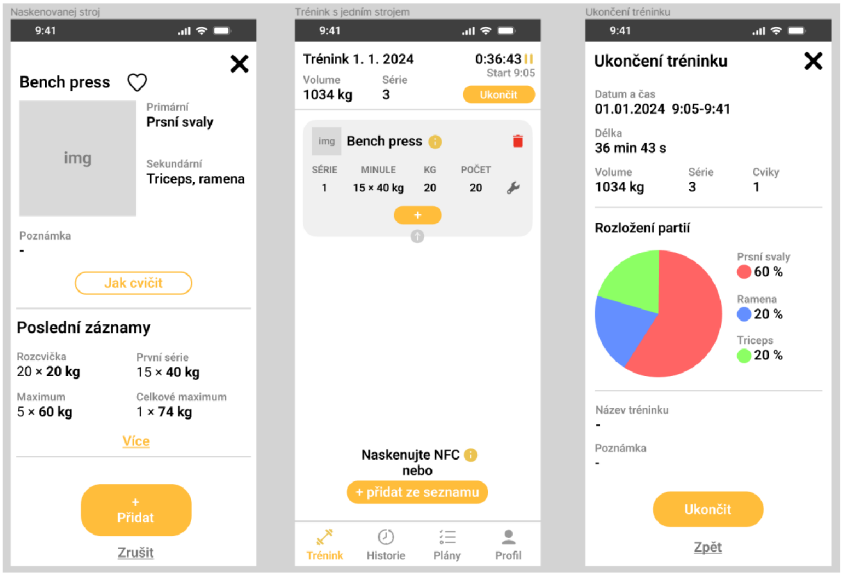
3.3 Grafický návrh

V rámci návrhu vznikl také wireframe pro celý systém. Vzhledem k zaměření aplikace na uživatelskou přívětivost a pohodlnost vznikl i mockup, tedy interaktivní grafický návrh v nástroji Figma, na kterém poté proběhlo testování na třech potenciálních uživatelích. Toto testování probíhalo formou 14 úkolů a cílů, jako např. „Spusť trénink a řekni, jak dlouho již trvá“ nebo „Přidej další sérii“. Na základě úspěšnosti a rychlosti dosažení těchto úkolů a dalších připomínek k návrhu byl poté finální design aplikace upraven. Velkou inspiraci u grafického designu hrála již zmíněná tréninková aplikace Hevy. Součástí návrhu grafického rozhraní byl také návrh navigace, který se nachází v příloze C.

Na obrázku 3.2 se nachází ukázka wireframu aplikace a na obrázku 3.3 grafický návrh této části aplikace. První obrazovka zobrazuje detail cviku po jeho zvolení, druhá zobrazuje stránku s tréninkem po vložení tohoto cviku a třetí shrnutí tréninku před ukončením. Kompletní wireframe i mockup jsou součástí paměťového média.



Obrázek 3.2: Ukázka wireframu části uživatelské aplikace



Obrázek 3.3: Ukázka mockupu části uživatelské aplikace

3.4 Architektura systému

Architektura aplikace funguje na dříve popsaném principu klient-server. Na obrázku 3.4 je zobrazen diagram vztahů entit, který reprezentuje datový model uložený na backendu, tedy serveru. Hlavní motivací za využitím této architektury je především možnost získávání dat pro správce posiloven, ovšem přináší i řadu dalších výhod, jako je centralizace a perzistence uživatelských a systémových dat.

3.4.1 Obecné principy

Načítání dat z NFC tagu

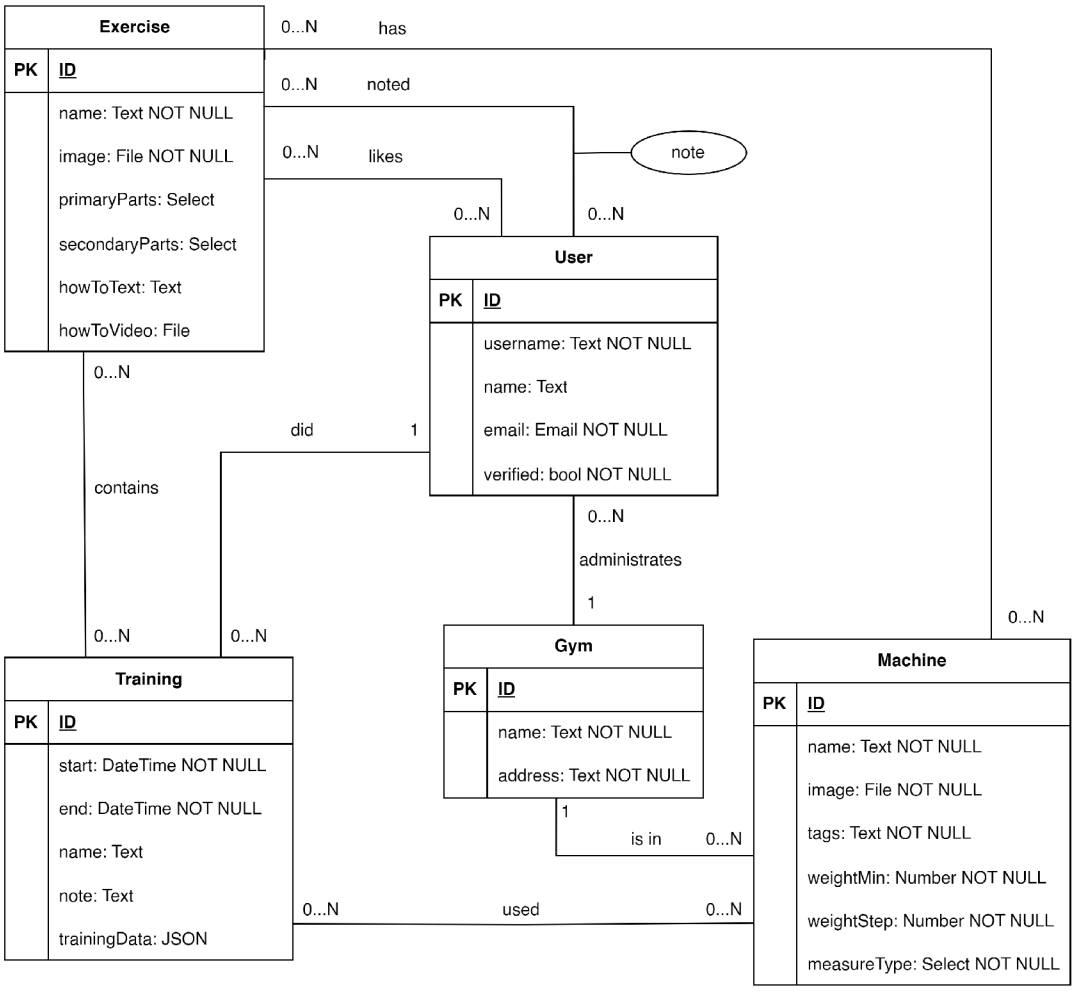
Při načtení tagu se z něj přečte identifikátor, který klient odešle na backend a podle ID se stroj vyhledá s využitím atributu *tags* v entitě *Machine*. Stroj uchovává seznam všech cviků, které jsou na něm k dispozici, a tento seznam se vrátí v rámci informací o stroji jako odpověď na požadavek. Na datové úložiště bude pomocí zmíněné aplikace NFC Tools nahrán odkaz na stažení aplikace a následně zamknut heslem, díky čemuž uživatel rychle získá přístup k aplikaci. Obrázek 3.5 zobrazuje princip komunikace systému.

Získání dat o využití strojů

Přístupů k rozhodnutí, zda byl stroj použit, je mnoho. V tomto systému bude za použití stroje bráno využití stroje v tréninku, který byl úspěšně ukončen a uložen. Tento způsob má především výhodu v jednoduché realizaci, která spočívá pouze načtení všech tréninků, které daný stroj obsahuje. K tomu v databázi slouží vztah mezi tréninkem (*Training*) a strojem (*Machine*) s názvem *used*. Za datum a čas použití stroje bude brán začátek tréninku.

Ukládání dat

Perzistentní data, jako jsou uplynulé tréninky, stroje či cviky, budou uložená na backendu. Data o probíhajícím tréninku budou uložena v mezipaměti, aby byla rychle přístupná, bu-



Obrázek 3.4: Diagram vztahů entit

dou však uložena i v lokálním úložišti, aby v případě vypnutí aplikace uživatel o trénink nepřišel. Důvodem ukládání do lokálního úložiště je především efektivita, neboť trénink se neustále mění (uživatel přidává série, zapisuje apod.) a tím pádem by v případě ukládání na backend bylo třeba neustále posílat požadavky na server. Dále takový trénink nebude potřeba ukládat na dlouhou dobu, ani třeba synchronizovat mezi zařízeními. Pomocí lokálního úložiště je řešen i offline režim.

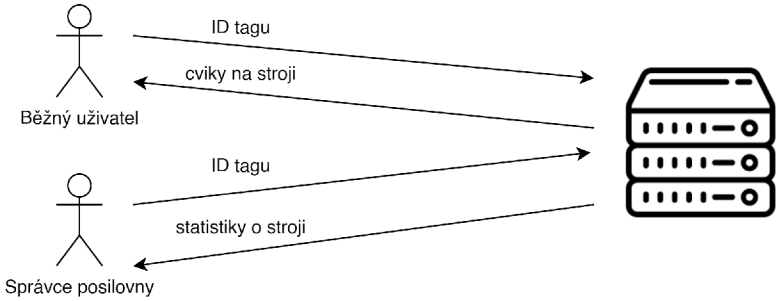
Získávání historie cviků

Důležitou vlastností aplikace je také získání historie o cviku. K tomuto účelu slouží vztah mezi tréninkem (*Training*) a cvikem (*Exercise*) s názvem *contains*. Při získávání historie daného cviku se na backend pošle dotaz na všechny tréninky, které obsahují daný cvik. Klient z těchto získaných tréninků pak vyextrahuje důležitá data a zobrazí uživateli.

Nabídka cviků

Nabídka cviků je vestavěná v systému a je uložená v databázi v tabulce *Exercises*. Při výběru cviku ze seznamu jsou tyto cviky načítány právě z této databáze. Tyto cviky byly přebrány

z databáze na webu *Muscle and Strength*[1] a obrázky ze serveru *Shutterstock* od autora Lioputra[2]. Každý cvik dále nese informaci o primárních a sekundárních svalových partiích. V systému je na výběr z devíti partií: břišní svaly, záda, biceps, lýtka, prsní svaly, předloktí, ramena, stehna a triceps.



Obrázek 3.5: Zjednodušený princip komunikace systému

3.4.2 Alternativní způsoby realizace

Architekturu aplikace a využití NFC tagu je možné realizovat více způsoby. Důvodem zahrnutí alternativ je ukázka, proč je použita právě navržená architektura.

Jedna z možných alternativ spočívá v jiném získávání dat o využití stroje. Tento systém by využíval vlastního rozšíření backendu o počítadlo načtení daného stroje. Tímto způsobem systém ví o všech načteních stroje přesně s datem a časem a dokáže takto statistiky předpočítat pro zobrazení správcem. Nevýhoda tohoto systému ovšem spočívá především v nemožnosti používat offline režim (respektive by tento systém musel být nakombinovaný s tím aktuálním) a také v náročnější realizaci. Dalším problémem je situace, kdy uživatel načte stroj a pak jej nepoužije (např. si chce pouze zobrazit dostupné cviky či historii na jednom ze cviků dostupným na stroji), čímž dodává do systému zavádějící informaci.

Další alternativou je převést celý systém do offline světa a backend by v takovém případě vůbec neexistoval. Veškeré informace o stroji by byly uložené v paměti NFC tagu v podobě záznamu, který by nesl seznam identifikátorů všech dostupných cviků a rozsah závaží na stroji. Díky možnosti uložit více záznamů na NFC tag by stále bylo možné, aby tag odkazoval na stažení aplikace či jinam. V tomto systému by všechna uživatelská i systémová data, jako jsou tréninky a cviky, byla uložena přímo na zařízení. Sledování načtení stroje by poté realizoval čítač na tagu, který např. NXP NTAG213 má zabudovaný, ovšem mohl by se vyskytovat i v podobě záznamu na tagu. Hlavní výhodou takového systému by byla především uživatelská přívětivost, neboť by bylo možné aplikaci používat neustále offline. Pro správce posiloven by však tento systém byl velmi nepohodlný, neboť by byl nucen pro sběr statistik obcházet všechny stroje a ručně z něj načítat data. Podobně by tomu bylo i při upravování dat na tagu, jako jsou cviky či další informace. Další problém by mohla přinášet omezená velikost paměti na čipu, která např. u použitého tagu dosahuje 144 bajtů. Data by tedy nutné ukládat ve velmi kompaktním formátu, tedy např. každý cvik by měl jedno nebo dvoubajtový identifikátor. I přesto je ale paměť omezená a nebylo by možné zaručit, že se na tag všechna potřebná data vlezou.

Další možnou alternativou či rozšířením je označení tagů QR kódy, které budou mít stejnou funkci, jako načtení tagu. Hlavní motivací za využitím NFC místo QR kódu je po-

[1] Databáze cviků z Muscle and Strength dostupná zde: odkaz.
[2] Obrázky cviků autora Lioputra dostupné zde: odkaz.

hodlnost, která by při načítání QR kódu byla značně snížena nutností pokaždé kód skenovat, což by přinášelo i další problémy.

3.5 Požadavky na prvky systému

Z uvedeného návrhu plynou základní požadavky na prvky systému, které musí prvky systému a implementační nástroje splňovat.

Požadavky na frontend

Cílem by měla být multiplatformní aplikace, je tedy důležité, aby použitý framework byl multiplatformní včetně všech použitých knihoven. Framework by měl být rychlý a spolehlivý. Měl by také podporovat všechny funkce potřebné pro výslednou aplikaci, buď v zabudované formě či v podobě knihovny. Tyto funkce jsou především práce s NFC a lokálním úložištěm, odesílání a příjem HTTP požadavků, práce s fotoaparátem, obrázky apod.

Požadavky na backend

Na backend nejsou kladené tak vysoké nároky, jako na frontend. Vzhledem k této nízké náročnosti by měl backend být vytvořen jednodušším nástrojem, který dodává pouze základní prostředky a nebude tedy třeba implementovat i ty nejzákladnější funkce. Primárně by měl umět především práci relační databází a měl by implementovat bezpečnostní pravidla pro přístup k API. Dále by měl autentizovat uživatele a ideálně podporovat odesílání emailů. Pro odevzdání v rámci práce je také vhodné, aby backend bylo možné odevzdat společně s prací a poté tento backend spustit lokálně.

Požadavky na NFC tag

- Kompatibilní s běžnými telefony, nalepovací, dostatečná paměť pro uložení URL, cykly čtení alespoň v řádu tisíců.
- Nízká cena: V každé posilovně se nachází řada strojů a vybavení. V závislosti na velikosti posilovny se mohou počty pohybovat od několik desítek až po stovky strojů. Při vysoké ceně tagu by celková částka velmi rychle vyrostla vzhůru.
- Funkčnost na kovovém povrchu: Většina strojů a vybavení v posilovnách jsou vyrobeny z kovových materiálů. Kov je ovšem materiál, který má tendenci rušit elektromagnetické vlnění, na kterém je technologie NFC založena. Vzhledem k tomu, že tagy budou umístěné přímo na stroji, je nutné, aby k tomu byly uzpůsobeny a komunikace nebyla kovem rušena.
- Protekce vůči přepsání: Z důvodu uložení dat na tag je podstatné, aby tento tag disponoval nějakou ochranou proti přepsání. Pro účely možného rozšíření do budoucnosti je vhodné, aby data bylo možné přepsat, tedy aby tento tag disponoval možností zaheslování.
- Pohodlné načtení tagu: Jedním z klíčových faktorů celého systému je pohodlnost při získávání dat ze stroje. K tomu je podstatné, aby uživatel nemusel při pokusu o načtení tagu několik sekund hledat polohu, ve které se mu tag načte. Obecně platí, že tagy s větší anténou mají větší dosah, proto je pro výběr tagu velikost antény podstatná.

Kapitola 4

Implementace aplikace

4.1 Výběr prostředků

Výběr nástrojů pro frontend a backend

Na implementaci klientské části aplikace byl zvolen dříve popsaný framework Flutter. Tento multiplatformní framework nabízí širokou řadu knihoven, mezi kterými se vyskytují všechny potřebné pro tento projekt. Je velmi rozšířený a rychlý.

Pro backend byl zvolen nástroj PocketBase, který je open-source a přináší všechny potřebné funkce pro běh systému. Je možné jej spustit lokálně pomocí jednoho spustitelného souboru, který ukládá data do jednoho lokálního adresáře.

Výběr NFC tagu

Všechny požadavky na NFC tag splňuje již dříve zmíněný NXP NTAG213 rozměrů 26,5×42 mm, konkrétně jeho on-metal verze, dostupný např. na ShopNFC. Konkrétní specifikace tagu je v tabulce níže.

NFC Standard	ISO/IEC 14443A
NFC Forum	Typ 2
UID	7 bajtů
Paměť	144 bajtů
Přenosová rychlost	106 kbit/s
Výdrž čtení/zápis	100 000 cyklů
Heslo	Ano
Zámek	Ano
Na kovový povrch	Ano
Čtecí vzdálenost	Až 10 cm
Cena za tag	≈ 28 Kč

Tabulka 4.1: Specifikace tagu NXP NTAG213 rozměrů 26,5×42 mm uzpůsobený na kovový povrch.

4.2 Frontend

Celý projekt se nachází v příloze, hlavní kód se pak nachází ve složce *frontend/lib*. Pro přehled je zde uveden souborový systém zdrojového kódu projektu:

- *screens*: adresář obsahující widgety pro všechny obrazovky včetně jejich unikátních komponent; je rozdělen na podsložky podle rolí a dále po jednotlivých obrazovkách,
- *widgets*: obsahuje znovupoužitelné widgety,
- *repository*: třídy pro práci s backendem či lokálním úložištěm,
- *models*: třídy reprezentující data na frontendu,
- *styles*: barvy a textové styly,
- *misc*: ostatní položky,
- *main.dart*: hlavní funkce pro běh aplikace.

4.2.1 Použité knihovny

- LocalStorage 4.0.1: Popsáno níže.
- PocketBase 0.18.1: Popsáno níže.
- Flutter NFC Kit: Popsáno níže.
- Provider 6.1.2: Popsáno v rešerši společně s Flutterem.
- Figma Squircle 0.5.3: Přidává speciální tvar obdélníku se zaoblenými a vyhlazenými rohy. Vychází z možnosti zmíněného nástroje Figma s názvem *Corner smoothing*.
- Pie Chart 5.4.0: Přidává koláčový graf využitý na zobrazení rozložení svalových partií na konci tréninku.
- Syncfusion Flutter Charts 24.2.9: Přidává tvorbu všemožných grafů. Je využit i historie cviku či statistik stroje.
- Flutter Spinkit 5.2.1: Přidává řadu nastavitelných načítacích widgetů.
- Flutter Palette 1.1.0: Nástroj pro tvorbu barevných palet, využitý u porovnání využití strojů.
- Další knihovny: intl 0.19.0, http 1.2.1, path_provider 2.1.2, camera 0.10.5, url_launcher 6.2.5.

Flutter NFC Kit

Tento balíček slouží k práci s NFC a je pro tuto práci klíčový[1]. Základní funkce balíčku jsou:

- `FlutterNfcKit.nfcAvailability`: Tato funkce zjistí stav NFC na telefonu, což umožňuje upozornění uživatele na zapnutí této funkce.

[1]Stránka balíčku zde: odkaz.

- `FlutterNfcKit.poll`: Blokující funkce poslouchající NFC tagy. Po nalezení tagu a přečtení jeho dat vrátí objekt `NFCTag`, který nese všechny jeho informace včetně identifikátoru. Úspěšný návrat z funkce započne session a pro další volání `poll` je nutné tuto session ukončit pomocí `FlutterNfcKit.finish`.
- `FlutterNfcKit.writeNDEFRecords`: Zapíše na tag data definovaná polem `NDEF` objektů.

LocalStorage

Tento pojem byl zmíněn již dříve, neboť každé mobilní zařízení obsahuje lokální úložiště, a zmíněný balíček slouží k přístupu k tomuto úložišti[2]. Balíček je velmi jednoduchý a obsahuje pouze základní prostředky pro inicializaci, zápis do a čtení ze souboru v úložišti. Data se ukládají ve formátu JSON[3].

- `LocalStorage`: Základní třída pro vytvoření souboru v úložišti. Použití např. `LocalStorage("file.json")`. Souborů může být v úložišti aplikace několik.
- `getItem(key)`: Vrátí data uložená pod klíčem `key`. Data jsou typu `Map<String, dynamic>`, který v Dartu reprezentuje JSON formát (`String` je klíč a `dynamic` je hodnota, která může být typu řetězec, číslo či další mapa).
- `setItem(key, data)`: Uloží JSON `data` pod klíčem `key`.

PocketBase

Přestože je možné na server odesílat požadavky obecným nástrojem či knihovnou, je vždy lepší využít speciálně vytvořených knihoven pro konkrétní backend, existuje-li. PocketBase nabízí vlastní SDK pro Dart[4], který definuje třídu `PocketBase`, u které se specifikuje adresa a poté je možné s backendem komunikovat. Při odesílání dat funkce požadují formát JSON (tedy v Dartu `Map<String, dynamic>`) a při čtení dat z backendu jsou data uložena v objektu `RecordModel`. Mezi základní metody třídy patří:

- `collection("kolekce").getFullList`: Vrátí seznam všech záznamů v kolekci. Tyto záznamy je možné filtrovat či seřadit pomocí parametrů `filter` a `sort`. K rozšíření záznamu o odkazované záznamy slouží parametr `expand`. Je také možné vyfiltrovat navrácené atributy pomocí `fields`.
- `collection("kolekce").getOne("ID")`: Vrátí jeden záznam s identifikátorem `ID`. Stejně funguje i metoda `delete`.
- `collection("kolekce").create(body: JSON)`: Vytvoří nový záznam v kolekci s daty uloženými v JSONu. Stejně funguje i metoda `update`.

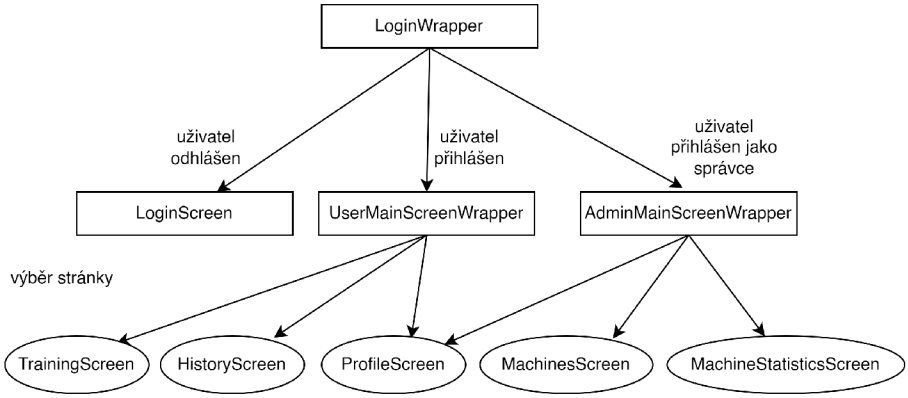
4.2.2 Uživatelské rozhraní

Tato kapitola a její podkapitoly se zabývají vzhledem a implementací uživatelského rozhraní. Při implementaci UI byl použit Material Design, který je ve Flutteru používaný jako primární designový jazyk.

[2]Stránka balíčku zde: odkaz.
[3]Více o formátu JSON na Wikipedii: odkaz.
[4]Stránka balíčku zde: odkaz.

Navigace po aplikaci

Na vrcholu celého stromu je stavový widget `LoginWrapper`, který zjistí stav přihlášení a na základě něj zobrazí přihlašovací obrazovku, nebo uživatele pustí do aplikace. Přihlašovací obrazovka se pomocí metody `setState` přepíná mezi přihlášením a registrací. Rozlišení mezi přihlášeným a nepřihlášeným uživatelem je popsáno v kapitole 4.2.3. V případě, kdy je uživatel přihlášen jako správce posilovny, vrátí tento widget `AdminMainScreenWrapper`, v opačném případě `UserMainScreenWrapper`. Tyto widgety zajišťují výběr hlavní stránky s pomocí zabudovaného widgetu `BottomNavigationBar`, který v dolní části obrazovky vytváří nabídku stránek. Zmíněné dva wrappery pro zvolení obsahu obrazovky užívají metody `setState`, která se volá po kliknutí na některou z ikonek v menu. Tato metoda je zde vhodná, neboť přepínání stránek probíhá jen a pouze z tohoto widgetu, tím pádem není třeba propagace funkce na aktualizaci widgetu stromem dolů. Navigaci na samotných stránkách zajišťuje zabudovaná třída `Navigator` pomocí vyskakovacích či vyjížděcích obrazovek a dialogů. Na obrázku 4.1 je vidět schéma základní navigace.



Obrázek 4.1: Schéma výběru aktuální obrazovky

Uživatelská přívětivost

K zajištění uživatelské přívětivosti aplikace obsahuje několik obecných prvků, jako jsou nápovědy a zpětná vazba uživateli.

Aplikace obsahuje obecný popis aplikace, kde je uživatel seznámen s celým konceptem aplikace pomocí textu a doprovodných obrázků. Vzhledem k netradičnímu způsobu zapisování tréninku je k dispozici podrobnější nápověda právě na tréninkové stránce. Dalším prvkem nápovědy je popis samotných prvků uživatelského rozhraní. Na mnoho z nich je možní kliknout, což zobrazí dialog s nápovědou pro daný prvek.

Aplikace samozřejmě implementuje i prvky zpětné vazby. Mezi nejzákladnější patří zobrazení načítání při čekání na dokončení nějaké akce a případné zobrazení chybové hlášky v případě, kdy se akci nepodařilo úspěšně vykonat. Načítání zpravidla provádí zabudovaný widget `FutureBuilder`. V jistých případech, např. při nedostupnosti funkce NFC, je uživatel vyzván k nápravě této skutečnosti. Dalším důležitým prvkem je tzv. snackbar, tedy malý informační dialog, který se objeví v dolní části obrazovky, např. při úspěšném přihlášení. Veškeré potenciální chyby v aplikační logice jsou obaleny blokem `try-catch`, který v případě výskytu některé chyby zobrazí snackbar s chybovou hláškou.

```
1 try {
2   appLogic(); // vykonani nejake aplikacni logiky, napr. pokus o prihlaseni
```

```
3   showSuccessSnackbar(); // v poradku, zobrazi se uspesny snackbar
4 } catch(_) {
5   showFailSnackbar(); // nastala chyba, zobrazi se neuspesny snackbar
6 }
```

Výpis 4.1: Příklad zobrazení (ne)úspěšného snackbaru

Poslední důležitou vlastností je potvrzení kritických akcí, jako je např. smazání tréninku. Uživatel je vždy při takové akci dotázán, zda si opravdu přeje akci provést. Aplikace neimplementuje žádné kroky zpět, tedy všechny akce smazání či přepsání jsou nevratné.

Znovupoužitelné widgety

Zde je popsáno pár základních widgetů, které jsem pro práci vytvořil a budou se objevovat v následujících stránkách. Dalo by se říct, že se jedná o znovupoužitelné komponenty. Pokud není explicitně řečeno jinak, nachází se v adresáři *lib/widgets*. Dále v adresáři *lib/styles* se nachází barvy a textové styly.

- `MyButton`: Nastavitelné tlačítko s jednotným stylem pro celou aplikaci. Je možné změnit styl na *outlined*. Alternativou pro tlačítko je `ClickableIcon` nebo `ClickableText`, tedy klikatelná ikonka nebo text. Tyto widgety jsou ve Flutteru vestavěné, jsou však mnou upravené a ořezané pro jednodušší a jednotnější styl a použití.
- `PopupScreen`: Vyskakovací obrazovka, které je možné nastavit směr vyskočení zespodu nebo zprava pomocí animací `Animations` umístěné v adresáři *lib/misc*. K otevření se používá `Navigator.of(context).push()` a je tedy možné se z nich vrátit tlačítkem *zpět* na zařízení, alternativně gestem. Kompaktnější alternativu tvoří `Dialog`, který zpravidla vyjíždí zespodu a zabírá pouze část obrazovky podle velikosti obsahu.
- `LabeledValue`: Hodnota a její popis méně výrazným písmem. Tento widget byl udán jako příklad tvorby vlastního widgetu v kapitole 2.4.3. Widget také disponuje volitelnou nápovědou, která se po kliknutí na hodnotu zobrazí jako popis dané hodnoty.
- `GreyBox`: Šedý obal, který může sloužit také jako komponenta typu karta, která má automatickou velikost podle velikosti potomka. Implicitně je vyplněný, je možné jej nastavit na ohraničený.
- `Loading`: Jednotný widget pro využití při načítání.
- `SnackBar`: Jednotný widget pro již zmíněný snackbar. Umožňuje nastavit jednotný styl podle typu obsahu, jako je úspěch, neúspěch nebo obecná informace.

4.2.3 Správa stavu, sdílená a lokální data

Stav aplikace souvisí s navigací, která byla popsána o kousek výše. O základní stav navigace se starají widgety typu wrapper pomocí volání `setState` a třída `Navigator`. V aplikaci je dále třeba řešit 2 stavové prvky:

- stav přihlášení: jestli je uživatel přihlášen, případně jestli je přihlášen jako správce,
- stav tréninku: jestli běží, jaké jsou v něm cviky a série apod.

Pro správu stavu aplikace je využit balíček *Provider*, který byl popsán v rešerši. Pro oba stavové prvky existuje speciální třída, která dědí od třídy `ChangeNotifier`.

AuthService

Tato třída nese informaci o stavu přihlášení. Třída je instanciovaná na vrcholu stromu widgetů a je tedy možné ji kdekoli ve stromě zpřístupnit. Tato třída implementuje základní metody, jako je přihlášení či odhlášení uživatele, které po svém dokonání volají metodu `notifyListeners`, která automaticky upozorní všechny posluchače, že došlo ke změně. Tyto změny poté poslouchá již zmíněný widget `LoginWrapper`, který po každé změně přečte aktuální stav přihlášení další metodou objektu `AuthService`. Třída po každém přihlášení uloží záznam do lokálního souboru *auth_state.json*, odkud se při restartování aplikaci načte stav přihlášení. Odhlášením se tento záznam smaže.

TrainingData

Tato třída nese informaci o tréninku a manipuluje s ním. Podobně jako `AuthService` je instanciovaná na vrcholu aplikace a je přístupná všude. Důležitým úkolem této třídy je i ukládání tréninku do lokálního úložiště po každé změně v tréninku. Tato data se ukládají do formátu JSON do souboru *training.json*. Privátní atributy třídy přístupné pomocí metody typu getter jsou:

- `localStorageReady`: asynchronně vrátí `true`, pokud je připraveno lokální úložiště pro ukládání tréninku,
- `exercises`: seznam všech cviků v tréninku včetně jejich sérií a typů,
- `isCurrentlyTraining`: vrací `true`, jestli právě probíhá trénink, jinak `false`,
- `trainingStart`: vrací čas začátku aktuálně probíhajícího tréninku,
- `total{Volume|Series|Exercises}`: vrací shrnující statistiky o tréninku – celkové volume, počet sérií a počet cviků.

Dále třída implementuje tyto metody:

- `startTraining`: Začne trénink, tedy nastaví potřebné proměnné a vytvoří záznam reprezentující prázdný trénink v lokálním úložišti.
- `endTraining`: Uloží trénink na backend a ukončí jej smazáním proměnných a dočasného tréninku v lokálním úložišti. Tato metoda po dokončení upozorní posluchače.
- `saveTraining`: Uloží trénink z mezipaměti do lokálního úložiště. Vzhledem k tomu, že tato metoda je volána vždy při změně na tréninku, je na konci opět voláno `notifyListeners` pro upozornění na tyto změny.
- `loadTraining`: Načte trénink z lokálního úložiště do proměnných v mezipaměti.
- `addExercise`: Přidá do tréninku nový cvik a uloží změny.
- `didSeeTutorial`: Metoda sloužící na rozhodnutí, zda uživatel již viděl nápovědu k tréninku či ne.

4.2.4 Komunikace s backendem a offline režim

Aplikace potřebuje komunikovat se vzdálenou databází na backendu. K tomu slouží speciální třída `RemoteDatabase`, která disponuje všemi potřebnými metodami pro práci s databází. Dostupné metody jsou již konkrétní požadavky na databázi s pár parametry, jako je např. `getMachineByTagID`, tedy „načti stroj s tímto identifikátorem tagu“. Třída tímto implementuje část aplikační logiky. Každá taková metoda vždy provede parametrizované API volání, počká na odpověď a převede navrácená data ve formátu JSON na Dart objekt reprezentující data na frontendu (třídy v adresáři *lib/models*). Každý model disponuje metodou pro převod do JSONu nebo z něj (typicky název `fromJson` nebo `toJson`). Předpřipravená volání vždy obsahují odkaz na kolekci a vybrané filtry a řazení, do kterých se při volání doplní konkrétní hodnoty, jako je ID záznamu či ID uživatele. Důležitým filtrem je symbol ∼, který slouží k vyhledávání ve vícenásobných hodnotách. Tímto je možné vyhledávat v seznamech odkazů, ovšem v mé aplikaci slouží i k vyhledání tagu. Jak bude později popsáno, identifikátory tagů se ukládají v textové podobě u stroje oddělené čárkami. Zmíněný symbol umí vyhledávat podřetězce v textu, čímž se implementuje hledání konkrétního stroje při načtení tagu[5].

Offline režim je naimplementovaný pomocí již zmíněného lokálního úložiště. Třída pro vzdálenou databázi disponuje metodou pro stažení všech uživatelských a důležitých systémových dat: tréninky, lajky, poznámky, stoje a cviky. Při přechodu do offline režimu jsou tato data stažena a předána třídě `LocalDatabase`, která si data uloží do souboru *local_database.json* a dále sem uloží informaci, že je právě aktivní tento režim. Dále tato třída disponuje funkcemi pro získání a smazání této lokální databáze, což je užitečné pro zálohování dat na server či přechodu zpět na online režim. V offline režimu se pro komunikaci s databází používá právě tato třída.

Třídy `RemoteDatabase` a `LocalDatabase` jsou obaleny třídou `DatabaseService`, která je návrhového vzoru strategie. Všechna volání týkající se práce s databází jdou přes tento objekt, který na základě aktuálního režimu rozhodne, zda se bude volat funkce ze vzdálené či lokální databáze. Následující úryvky kódu zobrazují vytvoření nového tréninku v lokální nebo vzdálené databázi, poté rozhodnutí o výběru z těchto metod a následně volání z aplikace. Některé metody jsou ovšem definovány pouze pro vzdálenou databázi (např. vytvoření uživatele či statistiky), proto by při volání těchto funkcí v offline režimu došlo k vyhození výjimky.

```
1 await pocketbase.collection('trainings').create(body: training.toJson());
```

Výpis 4.2: Tělo metody pro vytvoření tréninku na vzálené databázi (třída RemoteDatabase)

```
1 Map<String, dynamic> previousTrainings =
2   _localDatabase.getItem('trainings');
3
4 String newId = _getRandomString(15);
5 while (previousTrainings.containsKey(newId))
6   newId = _getRandomString(15);
7
8 Map<String, dynamic> allTrainings = {newId: training.toJson()};
9 allTrainings.addAll(previousTrainings);
```

[5]Jak vyplývá z popisu v kapitole 2.4.4, tento symbol hledá v textových záznamech podřetězce. V našem případě toto ovšem nevadí, neboť identifikátory tagů jsou navzájem unikátní a tím pádem nemůže nastat situace, kdy jeden identifikátor bude podřetězcem jiného.

```
10
11 await _localDatabase.setItem("trainings", allTrainings);
```

Výpis 4.3: Tělo metody pro vytvoření tréninku na lokální databázi (třída LocalDatabase)

```
1 if (LocalDatabase.isRemote) {
2   return await RemoteDatabase.createTraining(training);
3 } else {
4   return await LocalDatabase.createTraining(training);
5 }
```

Výpis 4.4: Vytvoření nového tréninku – rozhodnutí o strategii (třída DatabaseService)

```
1 await DatabaseService.createTraining(training);
```

Výpis 4.5: Volání metody na vytvoření tréninku

4.2.5 Implementace jednotlivých stránek

Trénink

Stránka pro trénink hojně využívá třídu pro sdílení dat `TrainingData`. Při otevření stránky tento objekt využije k načtení aktuálního tréninku a ten zobrazí, případně zobrazí možnost pro započetí nového. Stránka probíhajícího tréninku při budování zkontroluje, zda je dostupná funkce NFC voláním `FlutterNfcKit.nfcAvailability`. V případě vypnuté funkce NFC zobrazí tu informaci uživateli a nabídne možnost obnovení stránky. Po zapnutí NFC se zobrazí finální tréninková obrazovka, která je rozdělena na tři části: shrnutí, nápovědu a samotný trénink.

Shrnutí obsahuje datum tréninku, tlačítko ukončení a 3 statistické hodnoty o tréninku: celkové volume, počet sérií a délku tréninku. Pod shrnutím je text vyzývající naskenování NFC a ikonka pro otevření nápovědy ke tréninku. Je zde také možnost přidat cvik z kompletního seznamu, který disponuje filtry. Načítání NFC – jak text napovídá – probíhá automaticky bez nutnosti zásahu uživatele, je však nutné mít tuto stránku otevřenou. Při budování stránky se po již zmíněné kontrole dostupnosti spustí na pozadí následující funkce, která dané čtení implementuje.

```
1 while (true) {
2   var tag = await FlutterNfcKit.poll(timeout: const Duration(days: 1));
3   await loadMachine(tag.idnorm);
4   await Future.delayed(const Duration(seconds: 1));
5   await FlutterNfcKit.finish();
6 }
```

Výpis 4.6: Funkce implementující čtení NFC tagu

- Řádek 2: Blokující funkce, která čeká na komunikaci s tagem. Po úspěšném přečtení jsou data uložena v proměnné `tag`. Parametr `timeout` je zde povinný, ovšem není pro aplikaci podstatný.
- Řádek 3: Načte stroj a zobrazí jeho informace. Funkce skončí až po výběru cviku nebo zavření stránky. Atribut `idnorm` je mnou připsaná funke, která vrací ID tagu v normovaném formátu, tedy hexadecimální formát s dvojtečkami mezi čísly bez přebytečných nul. Implementace této funkce je v následujícím výpisu.

- Řádek 4: Krátká pauza, která slouží jako prevence proti mnohonásobnému přečtení tagu v případě, kdy se stroj nepovede načíst a tím pádem se funkce `loadMachine` vrátí dříve, než uživatel stihne oddálit telefon od tagu.
- Řádek 5: Ukončí session, kterou funkce `poll` začala. Je nutné ji volat před voláním další funkce `poll`.
- Řádek 1: Provádí celý uvedený proces dokola, dokud není stránka zahozena.

```
1  String idLower = id.toLowerCase();
2  List<String> bytes = [];
3  for (int i = 0; i < idLower.length; i += 2) {
4    String byte = idLower.substring(i, i + 2);
5    if (byte[0] == "0") {
6      byte = byte[1];
7    }
8    bytes.add(byte);
9  }
10 return bytes.join(":");
```

Výpis 4.7: Funkce implementující přečtení ID z tagu a konverzi

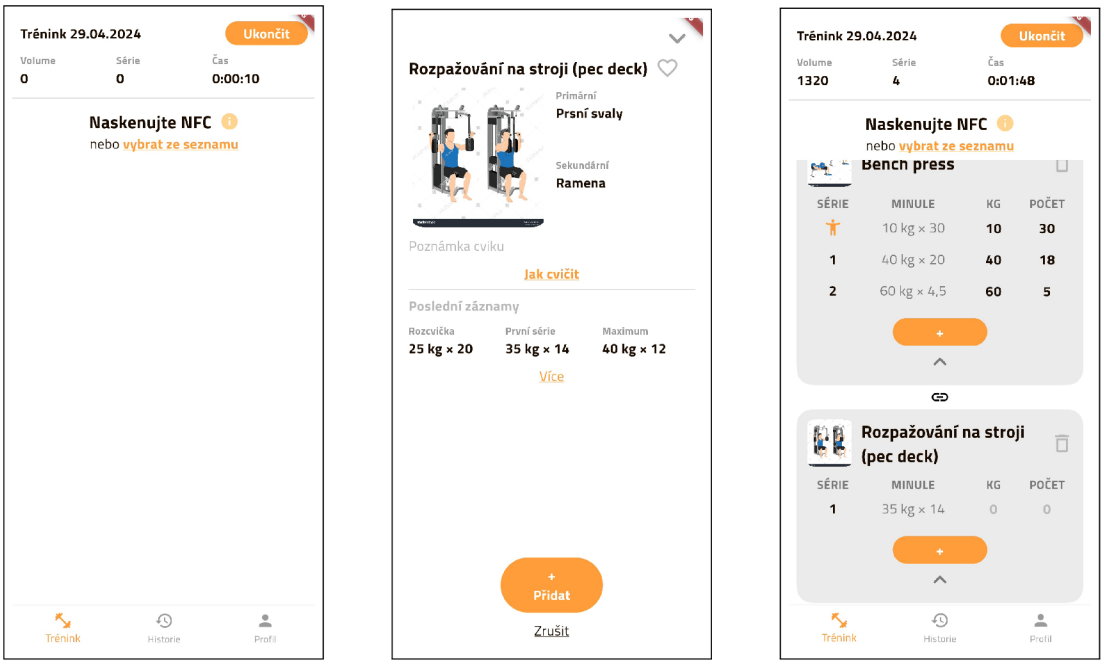
Po načtení NFC tagu výše popsaným způsobem je uživateli zobrazen seznam všech dostupných cviků s filtry (více v kapitole 4.2.5). Po kliknutí na cvik se zobrazí jeho detail, v případě výskytu pouze jednoho cviku na stroji je zobrazen přímo daný cvik. V dolní části obrazovky se nachází tlačítko *Přidat*, které cvik přidá do tréninku. Samotný trénink poté obsahuje skrolovací seznam všech cviků, které ukládá třída `TrainingExercise` a zobrazuje widget `TrainingExerciseWidget`. Tento widget zobrazuje obrázek a název cviku, možnost cvik odebrat a tabulku se sériemi včetně tlačítka pro přidání nové série. Tabulka obsahuje hodnoty v následujícím pořadí: typ/číslo série, hodnota minule, váha a počet. Váha a počet je implementováno textovým polem, které má implicitní hodnotu 0. Tato textová pole také zobrazují nápovědy v podobě šedého textu, které se při přidání série automaticky nakopírují do té nové. Dvojklikem na pole uživatel tuto hodnotu zaloguje. Dále je možné kliknutím na hodnotu ve sloupci *minule* překopírovat danou hodnotu do aktuální série. Poslední možností zalogování hodnoty je pomocí posuvníku, který se otevře podržením na textovém poli. Rozsah tohoto posuvníku je definován použitým strojem, který má rozsah od $minimum$ po $minimum + 20 \times krok$. Výběrem hodnoty tento posuvník zmizí a hodnota se zapíše. Aktuální cvik je přichycen ke spodní části obrazovky, což udržuje aktuální sérii vždy na stejném místě, které je ergonomické pro ovládání jednou rukou.

Při ukončování tréninku se uživateli zobrazí shrnutí společně s grafem. Tento graf využívá widget `PieChart` ze stejnojmenné knihovny. Vstupem grafu je mapa, která jednotlivým svalovým partiím (výčet `MusclePart`) přiřazuje jejich procentuální podíly v tréninku. Při výpočtu se využívají jednotlivé série a u každé má vždy primární partie cviku váhu 3 a sekundární váhu 1. Uživatel má možnost trénink pojmenovat, přidat poznámku a následně ukončit a tím uložit, případně ukončit smazáním.

Obrázek 4.2 zobrazuje ukázku tréninkové stránky.

Historie

Na stránce Historie v záložce Tréninky uživatel vidí statistiky ve vrchní části obrazovky a pod statistikami tréninky. Po rozkliknutí tréninku vidí jeho detail, který vypadá po-



(a) Prázdný trénink po spuštění

(b) Načtený cvik pro přidání

(c) Trénink se dvěma cviky

Obrázek 4.2: Ukázka stránky pro trénink

dobně, jako shrnutí při jeho ukončení. Pod shrnutím se nachází stejný seznam cviků, jako při tréninku, ovšem tabulka sérií neobsahuje položku *minule*. Také není možné tento trénink upravit. Ve spodní části seznamu je možnost trénink smazat.

V záložce pro cviky vidí uživatel filtry a pod nimi samotný seznam cviků. Filtry implementuje samostatná třída `Filters`, která při instanciaci požaduje seznam všech cviků. Implicitně má všechny filtry vypnuté a je s nimi možné manipulovat. Dále disponuje metodou `filteredExercises`, která na všechny cviky aplikuje vybrané filtry a následně vrátí vyfiltrovaný seznam cviků. Uživatelské rozhraní pro ovládání filtrů je implementováno widgetem `FiltersWidget` následovně:

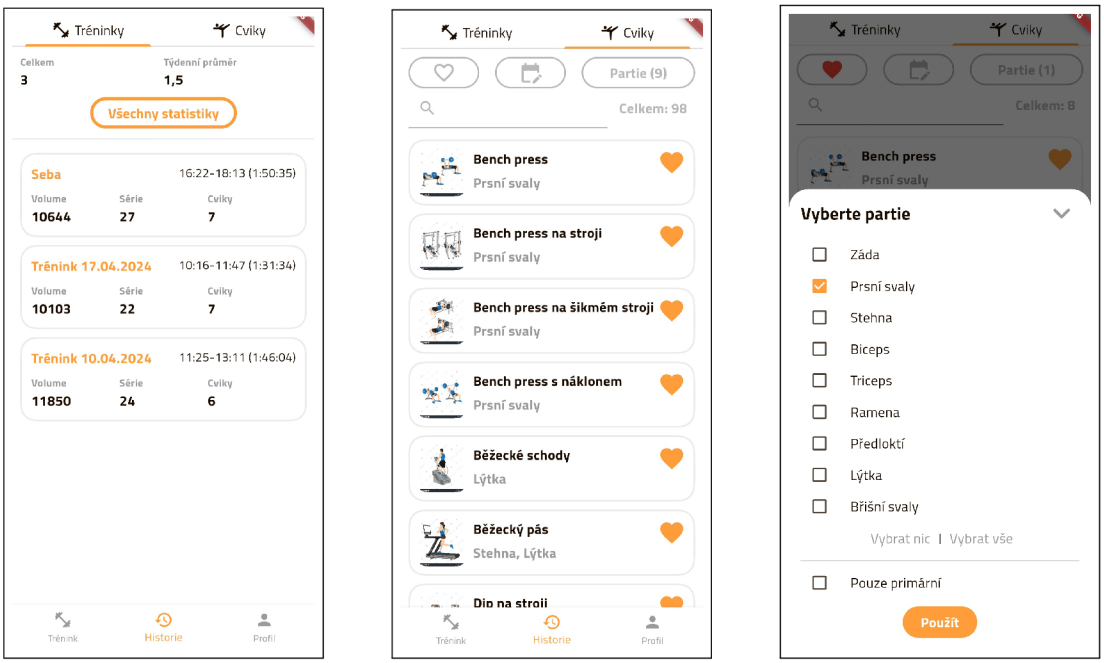
- Oblíbené cviky: Po prvním kliknutí je vybarvena ikonka srdce, které symbolizuje oblíbené cviky. Po druhém kliknutí je vybarveno její okolí, což symbolizuje neoblíbené cviky. Tímto způsobem uživateli umožňuje prohledávat nové cviky. Po třetím kliknutí filtr vypne.
- Použité cviky: Fungují na stejném principu, jako oblíbené cviky.
- Svalové partie: Uživatel vidí, kolik svalových partií se zobrazuje. Po kliknutí na tlačítko se mu zobrazí seznam všech partií, kde může vybrat jednotlivé partie, případně všechny nebo žádné. Dále má možnost vybrat mezi možností filtrovat pouze primární partie či zahrnout i sekundární.
- Vyhledávání: Uživatel může mezi cviky vyhledávat i pomocí textového pole, které vyhledává podle názvu cviku.

Po otevření cviku vidí jeho detail, kde může mimo jiné přidat poznámku, která se automaticky uloží při odchodu ze stránky. Důležitá je historie cviku, která je rozdělena na podčásti:

- Poslední záznamy: Zde vidí hodnoty *rozcvička*, *první série* a *maximum*.
- Další hodnoty: Zde vidí hodnoty *1RM*[6] a *nejlepší volume*.
- Graf: Historie výkonu na cviku, kde si může vybrat mezi hodnotami *maximální váha*, *nejvyšší volume*, *celkové volume*, *počet sérií* a *1RM*.

Princip způsobu získání dat byl již popsán v kapitole 3.4. Třída `DatabaseService` implementuje metodu `getAllTrainingsWithExercise`, která vrátí seznam všech tréninků, na kterých byl daný cvik uživatelem použit. Z těchto tréninků se pak jednoduše vyčtou data potřebná pro zjištění těchto historických hodnot. Tato metoda je také použita pro výpočet hodnot do sloupce *minule* u tréninku.

Obrázek 4.3 zobrazuje ukázku stránky historie.



(a) Seznam všech tréninků

(b) Seznam všech cviků s filtry

(c) Ukázka filtrů – pouze oblíbené a procvičující prsní svaly

Obrázek 4.3: Ukázka stránky historie

Profil

Ve vrchní části obrazovky se zobrazuje aktuální přihlášený uživatel a možnost se odhlásit. Odhlášení volá metodu z třídy `AuthService`, která po ohlášení upozorní na změny `LoginWrapper` a ten změní zobrazenou stránku.

[6] Zde je počítáno podle Brzyckiho vzorce, tedy $w \times \frac{36}{37-r}$, kde w je nejvyšší váha a r je počet opakování, kolikrát uživatel danou váhu zvedl.

Poté se v seznamu nachází sekce pro offline režim s nápovědou. Po kliknutí na přepínač je uživateli zobrazen seznam všech dostupných posiloven, ze kterých si musí vybrat. Tato funkce je zde především z důvodu, aby uživatel neměl zbytečně v telefonu celou databázi všech existujících posiloven. Po přechodu na offline je možné tento režim vypnout či pouze zálohovat svá data na server. Obě funkce zajišťuje opět třída `DatabaseService`, která implementuje `toggleOfflineMode` a `syncUserData`. Funkce pro nahrání dat na server zajišťuje `RemoteDatabase`, která je „chytrá“, tedy nezměněné položky ponechá a nové či naopak smazané upraví. Tímto se zachovává konzistenci databáze, neboť „hloupé“ smazání všech položek a následně vytvoření znovu by změnilo identifikátory záznamů. Pokud je aktuální uživatel i správce posilovny, synchronizuje i data o strojích. U běžného uživatele se jedná pouze o tréninky, lajky a poznámky cviků.

Pod sekcí pro offline režim se nachází tlačítko pro přepnutí na správcovskou verzi v případě, že uživatel je i správce. V zápatí stránky se nacházejí základní kontaktní a obchodní informace včetně základní nápovědy a popisu celé aplikace.

Správa strojů

Ve správě strojů vidí správce všechny aktivní stroje. Celá stránka slouží – podobně jako stránka pro trénink – jako skener NFC. Je tedy možné kdykoli přiblížit telefon k NFC tagu a tím zobrazit existující stroj či vytvořit nový. Stránka pro nový stroj obsahuje:

- Textové pole pro název stroje.
- Oblast pro zadání rozsahu závaží: Správce zadá minimum a krok závaží a z těchto hodnot se vypočítá maximum, které se mu zobrazí. Maximum není možné ovlivnit a je dáno vzorcem, který byl popsán dříve. Přesnost tohoto rozsahu ovšem není podstatná, neboť slouží primárně jako výpomoc při zadávání váhy a v případě nepřesného či nedostačujícího rozsahu může uživatel zvolit jinou metodu.
- Přidání fotky: Správce musí pro vytvoření přidat fotku stroje, na jejíž vyfocení má neomezeně pokusů.
- Možnost přidat tag již k existujícímu stroji: V tomto případě si správce vybere ze seznamu všech jeho strojů.

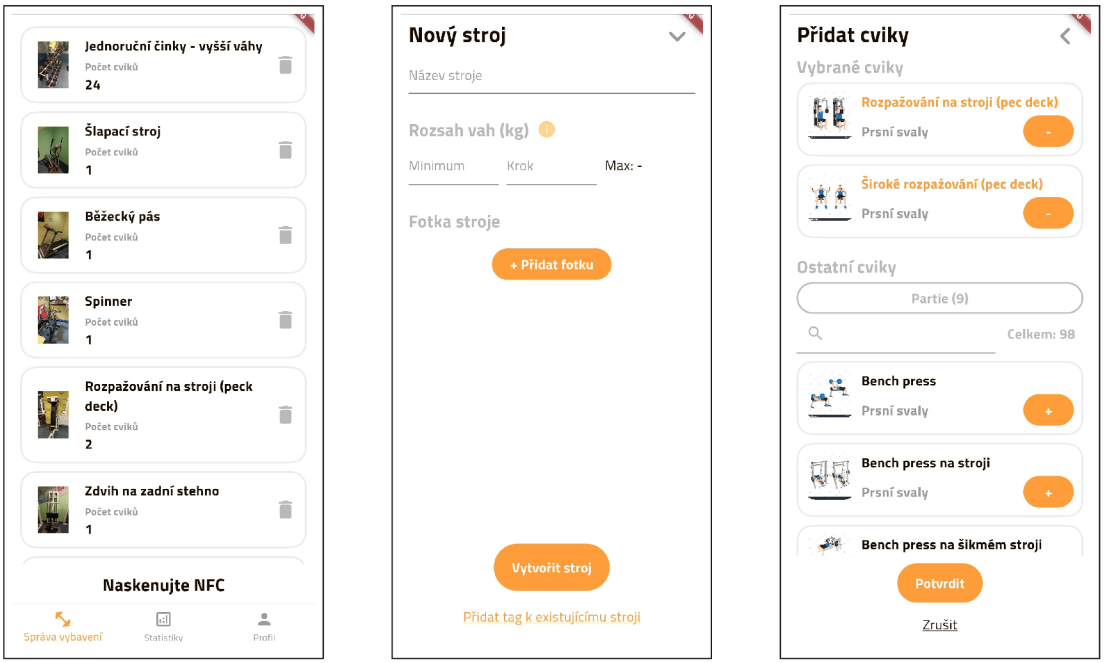
Po vytvoření stroje je správce přesměrován do sekce detailu stroje. Tato stránka obsahuje 3 záložky:

- Cviky: V této sekci vidí všechny přidané cviky na stroj a má možnost přejít na stránku pro úpravu cviků. Ve vrchní části této stránky vidí přidané cviky, následované podobným seznamem všech cviků, jako u historie cviků pro běžného uživatele, včetně filtrů pro partie a vyhledávání podle názvu cviku. Každý cvik disponuje tlačítkem „+“ či „-“ a je možné jej rozkliknout pro detail.
- Statistiky: Zde vidí shrnující statistiky pro daný stroj. Nabízí dvě hodnoty – celkové použití a průměrné denní použití – a graf. Tento graf nabízí 3 hodnoty: *celkové použití stroje*, *využití během týdne*, tedy rozbor jednotlivých dnů v týdnu, a *využití během dne v týdnu*, tedy konkrétní den v týdnu, který si vybere z přídavného výběru. Dále je možné vybrat časový úsek, ze kterého se statistiky čerpají. Konkrétně má na výběr *poslední týden*, *poslední měsíc*, *poslední 3 měsíce* a *celou dobu*.

- Nastavení: Stránka velmi podobná vytvoření nového stroje. Místo možnosti přidání tagu k existujícímu stroji je zde počet připojených tagů a možnost upravit existující tagy. Tato možnost otevře novou stránku, která je opět aktivním skenerem NFC. Správce může v tuto chvíli naskenovat jakýkoli tag a stránka mu zobrazí jeho stav, který může být:
  - přiřazený k tomuto stroji: je-li tagů na stroji více, je možné tento tag odebrat, v opačném případě to možné není,
  - přiřazený k jinému stroji: v tomto případě není možné s tagem nic dělat a je nutné přejít k danému stroji pro provedení některé z akcí,
  - nepřiřazený: v tomto případě je možné tag přiřadit k aktuálnímu stroji.

Ve všech zmíněných případech čtení tagu je použit stejný princip, jako u načítání tagu u tréninku, tedy čeká se na přečtení tagu, zjistí se jeho normovaný identifikátor a ním se poté dále pracuje.

Obrázek 4.4 zobrazuje ukázku stránky pro správu strojů.



(a) Seznam všech strojů

(b) Vytvoření nového stroje po načtení tagu

(c) Výběr cviků pro stroj

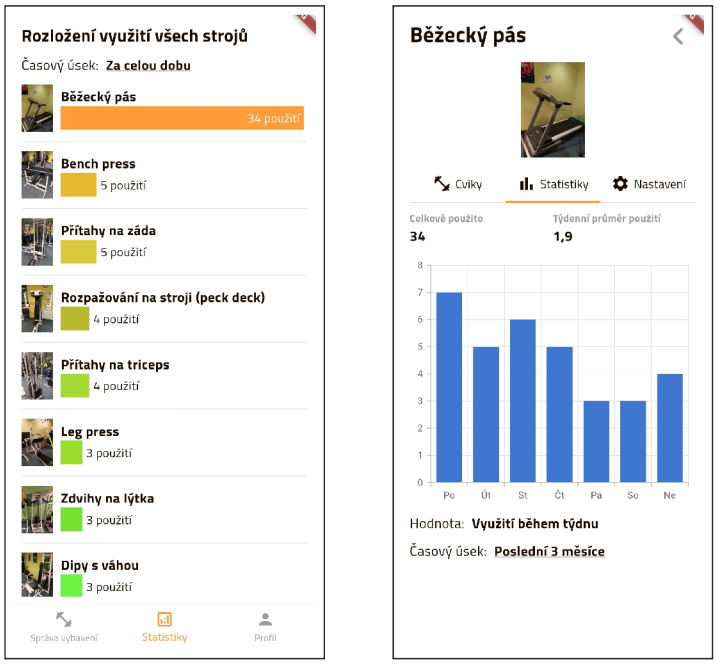
Obrázek 4.4: Ukázka správy strojů a vybavení

Statistiky

Statistiky odpovídají návrhu. Každý stroj je zobrazen obrázkem, názvem a sloupcem, který symbolizuje míru využití v rámci všech strojů. Dále je ke sloupci připojen text, který zobrazuje tuto hodnotu i číselně. Správce může – podobně jako u statistik strojů – vybrat dobu, za kterou se statistiky zobrazují. Po rozkliknutí některého stroje je správce přenesen do detailu stroje přímo ke statistikám. Načítání dat implementuje metoda `getAllMachineTrai-`

`nings`, která pro každý stroj vrátí tréninky. Statistiky strojů fungují pouze v režimu online, neboť by v offline režimu správce neměl aktuální data.

Obrázek 4.5 zobrazuje ukázku stránky statistik.



(a) Porovnání využití všech strojů

(b) Využití jednoho stroje během týdne

Obrázek 4.5: Ukázka statistik o strojích

4.3 Backend

Vzhledem k nenáročnosti backendové části je tato část krátká a obsahuje informace o převodu ERD do relační databáze nastavení bezpečnostních pravidel pro přístupu k API. Aktuálně backend běží na virtuálním privátním serveru (VPS) s parametry: 1 CPU (Intel Xeon E7), 1024 MB RAM, disk 20GB, konektivita 100 Mb/s.

Součástí této práce je ořezaná kopie této databáze, kterou je možné spustit lokálně a prozkoumat. Její popis se nachází v kapitole 5.2.1.

4.3.1 Transformace ERD do relační databáze

Jádrem celého backendu je databáze, jejíž kolekce reflektují navržený ER diagram. Přestože ER diagram je výsledné databázi velmi podobný, není možné jej využít 1 : 1 a je nutné provést transformaci do relačního modelu. Díky možnosti vícenásobných vztahů nebylo nutné vytvářet vazební tabulky. Jedinou skutečně vazební tabulkou je kolekce *likes*, která propojuje cvik s uživatelem (tedy který uživatel má oblíbený který cvik)[7]. Důvodem zavedení této tabulky je konzistence s tabulkou pro poznámky (*notes*), která je podobného charakteru, ale vzhledem k přidanému atributu textu poznámky by nebylo možné implementovat tabulku

[7] Alternativně by bylo možné ukládat seznam odkazů na cviky přímo k uživateli.

přes seznamy odkazů. Dalším důvodem je jednodušší přidávání, vyhledávání a odebírání oblíbených cviků. Záhlaví všech tabulek s popisy jsou součástí přílohy B.

4.3.2 API a bezpečnostní pravidla

PocketBase nabízí jednoduché API, které bylo popsáno v rešerši. Konkrétní adresa API pro tuto práci je

`http://blazik.me:8090/api/`
respektive
`http://89.203.249.43:8090/api/`.

Dále bylo naimplementováno několik typů bezpečnostních pravidel. Protože je třeba napsat pravidlo pro každý z pěti druhů operací a to pro každou kolekci, jsou zde vypsány základní pravidla a poté využití těchto pravidel pro jednotlivé kolekce.

1. `<admin only>`: Speciální vestavěné pravidlo, které umožňuje danou akci provést pouze skrze administrátorskou konzoli.
2. `<empty>`: Umožňuje komukoli bez omezení provést danou akci.
3. `@request.auth.id = user`: Akci může provést pouze přihlášený uživatel, kterému záznam patří.
4. `@request.auth.id = @request.data.user`: Akci může provést pouze přihlášený uživatel, který vytváří svůj záznam.
5. `@request.auth.gym_admin = @request.data.gym`: Akci může provést pouze přihlášený uživatel, který je správcem posilovny, do které vytváří záznam.
6. `@request.auth.gym_admin = gym`: Akci může provést pouze přihlášený uživatel, který je správcem posilovny, které záznam patří.
7. `@request.auth.id = user || @request.auth.gym_admin != ""`: Akci může provést pouze přihlášený uživatel, kterému záznam patří, nebo kterýkoli správce posilovny.

Pravidla jsou poté v kolekcích využita následovně:

	Výpis	Čtení	Vytvoření	Úprava	Smazání
exercises	2	2	1	1	1
gyms	2	2	1	1	1
likes	3	3	4	3	3
machines	2	2	5	6	6
notes	3	3	4	3	3
trainings	7	3	4	3	3

Tabulka 4.2: Tabulka použití pravidel pro přístup k API.

Kapitola 5

Testování

Testování aplikace probíhalo dvěma způsoby – uživatelské testování v praxi a pomocí unit testů. Oba způsoby byly doprovázené ručním testováním a laděním. Aplikace, přestože je napsaná plně multiplatformně, byla testována pouze na systému Android, neboť testování na systém iOS je poněkud komplikovanější.

5.1 Uživatelské testování

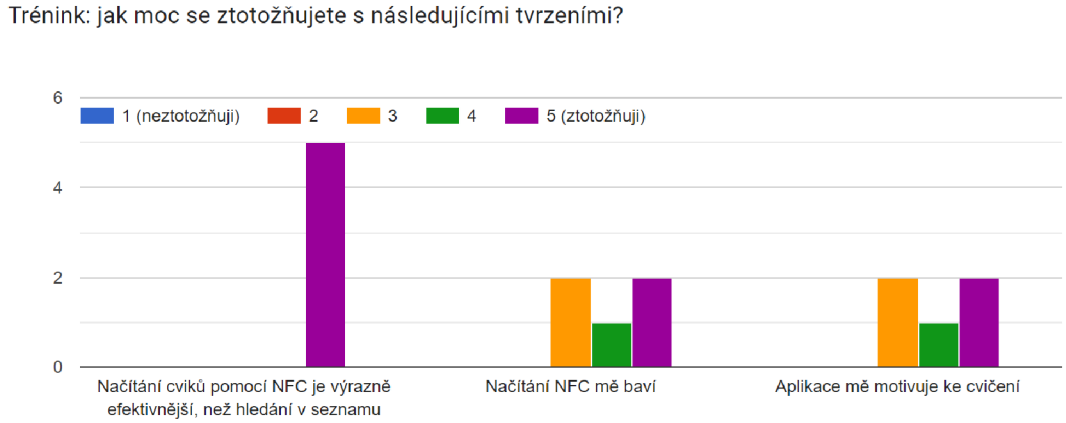
V rámci uživatelského testování byl celý systém nasazen v reálné posilovně. Bylo zde rozmístěno celkem 30 NFC tagů na 29 různých strojů či vybavení. Cílem této části bylo:

- otestovat registraci, odesílání potvrzovacích emailů a přihlášení,
- zjistit celkovou použitelnost a funkčnost systému v praxi,
- otestovat načítání NFC na více různých zařízeních,
- získat zpětnou vazbu na ergonomii a použitelnost aplikace z pohledu běžného uživatele,
- objevit případné nedostatky či chyby v aplikaci,
- otestovat funkčnost offline režimu, neboť ve vybrané posilovně není přístup k internetu.

Testování se zúčastnilo 5 uživatelů, někteří z nich i vícekrát. Forma testování spočívala v nainstalování aplikace do zařízení uživatele, který poté s minimálními instrukcemi začal aplikaci používat v praxi. Během testování jsem uživatele sledoval a dělal si poznámky na základě jejich akcí či připomínek. Každý uživatel na konci testování vyplnil detailní dotazník na jeho dojem z aplikace. Otázky společně s výsledky se nachází na paměťovém médiu, na obrázku 5.1 je ukázka jedné ze shrnujících otázek.

Uživatelské testování bylo úspěšné a obecný dojem uživatelů z aplikace byl kladný. Registrace, přihlášení a doručení emailu proběhlo u každého uživatele bez problému. Ukázalo se, že systém v praxi funkční a má svůj přínos. Načítání tagů fungovalo na všech zařízeních, ovšem na některých se za jistých okolností vypínalo poslouchání tagů. Uživatelé měli pár připomínek k uživatelskému rozhraní, které do budoucnosti plánuji upravit. Součástí uživatelského testování byl také rozhovor s majitelem posilovny zahrnující ukázku aplikace. Uživatelské testování pomohlo odhalit různé nedostatky, z nichž některé byly již napraveny. Příklady nalezených a opravených nedostatků:

- Chyba: za jistých okolností nepropisující se či mizející záznamy v tabulce sérií.
  - Důvod chyby: záznamy psané na klávesnici se ukládají při zavření klávesnice. Chyba se vyskytla při přidání další série ještě před jejím zavřením.
  - Oprava: při přidání další série se klávesnice automaticky zavře a tím pádem záznam uloží.
- Chyba: špatné zobrazení hodnot u grafu při ukončování tréninku.
  - Důvod chyby: při výpočtu poměru svalových partií byl celkový počet špatně vypočten.
  - Oprava: oprava výpočtu celkového počtu svalových partií.
- Nedostatek: zpoždění při otevírání textového pole pro zápis hodnoty váhy či počtu opakování.
  - Důvod chyby: textové pole je obaleno detektorem gest, které pro zapsání předchozí hodnoty implementuje funkci dvojitého kliknutí. Tato funkce vždy čekala na potenciální druhý klik, aby rozhodla, které z gest bylo vykonáno, čímž způsobovala zpoždění při otevírání pole.
  - Oprava: odstranění možnosti zapsání předchozí hodnoty tímto způsobem. Stále se tato hodnota ovšem zapíše v případě, kdy uživatel přidá další sérii bez zapsání dané hodnoty ručně.



Obrázek 5.1: Příklad otázky a výsledků u dotazníku

5.2 Unit testy

V rámci aplikace vznikly i unit testy, které mají za cíl otestovat aplikační logiku a práci s daty. Unit testy se nachází v adresáři projektu ve složce *frontend/test*. Spuštění testů je možné pomocí příkazu `flutter test test/unit_test.dart`. Byly testovány dva základní typy objektů:

- datové třídy: třídy udržující data z databáze, práce s daty, konverze z/do formátu JSON apod.,

- databázové třídy: odesílání a zpracování požadavků z databáze.

Vzhledem k tomu, že testy běží mimo aplikaci, nebylo možné testovat práci s lokálním úložištěm, ovšem na tuto část bylo zaměřené uživatelské testování. Celkem vzniklo 51 testů pokrývající úspěšné i neúspěšné pokusy o práci s daty. Pro testování byla vytvořena ořezaná kopie databáze, která je součástí přiložených souborů. Unit testy byly rovněž úspěšné a pomohly odhalit řadu spíše drobných nedostatků, jako je například:

- Chyba: chybějící důležitý parametr u načítání použitých cviků.
  - Oprava: přidání tohoto parametru do filtrů u dotazu.
- Chyba: nemazání poznámek z databáze.
  - Oprava: v případě prázdné poznámky se tato poznámka místo úpravy smaže.
- Nedostatek: zbytečné návratové hodnoty u některých databázových funkcí.
  - Oprava: odstranění těchto návratových hodnot. V případě chyby bude místo návratové hodnoty vyhozena výjimka, jako u ostatních funkcí.
- Nedostatek: zbytečné duplicitní funkce.
  - Oprava: odstranění této duplicity a ponechání pouze jedné univerzální verze funkce.

5.2.1 Testovací databáze

Pro úspěšný průběh testů je nutné spustit tuto lokální databázi pomocí příkazu `pocketbase serve` z adresáře *backend*. Při spuštění testů se program automaticky přihlásí na tuto lokálně běžící databázi, komunikuje s ní a na konci se zase odhlásí. Na tuto zkušební databázi je možné se připojit přes administrátorské rozhraní pomocí emailu *test@test.cz* a hesla *heslo12345*. Se stejnými údaji je i vytvořen uživatel na reálné databázi (pro přihlášení z aplikace). Databáze obsahuje:

- sadu 98 cviků, které systém aktuálně nabízí,
- všechny stroje z praktického nasazení se skrytými obrázky,
- dva testovací uživatele – *mockuser*, na kterém jsou demonstrována perzistentní data, a *testuser*, určeného na unit testy,
- ukázková data uživatele *mockuser* (testovací posilovnu, lajky, poznámky a tréninky).

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit mobilní aplikaci na zapisování a analyzování silových tréninků využívající technologii NFC na vyhledávání cviků, která bude rovněž pomáhat správcům provozoven sledovat vytížení strojů. Aplikace měla být ergonomická a pohodlná pro zajištění nerušeného tréninku. Aplikaci se podařilo s vytyčenými cíli vytvořit s pomocí multiplatformního frameworku, otestovat na zařízení Android a nyní je ve fázi vydání na obchod Google Play. Systém přinesl funkční ukázku směru, kterým by se mohly tréninkové aplikace v budoucnosti vyvíjet.

Samotnou aplikaci by v tuto chvíli bylo možné rozšířit o mnoho dalších funkcí, jako je například plánování tréninku dopředu či pokročilejší diagnostika využití strojů. Dalším možným rozšířením je verze pro chytré hodinky, které jsou v dnešní době čím dál populárnější a umožňují mimo jiné měření veličin, kterých mobilní telefon schopen není. Dále je díky veřejnému přístupu k databázi možné využít systém v aplikacích třetích stran, neboť NFC tagy je možné přečíst libovolnou aplikací. Dalším krokem k pohodlnosti a rychlosti systému je odstranění nutnosti ručního zapisování hodnot, například v podobě chytrého kolíku, který by pomocí senzoru měřil váhu a opakování, či chytrých strojů, které by navíc měly možnost automaticky nastavit závaží.

Práce mi pomohla s lepším porozuměním vývoje a testování mobilních aplikací, především s důrazem na architekturu klient-server. Také jsem se díky ní naučil pracovat s NFC tagy, které podle mého názoru mají mnohá využití v praxi, o kterých dnes ještě nevíme.

Literatura

[1] *Dart (programming language)* [online]. Wikipedia, The Free Encyclopedia., 16. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Dart_(programming_language)&oldid=1219222322.

[2] *Spiral model* [online]. Wikipedia, The Free Encyclopedia., 17. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Spiral_model&oldid=1219372191.

[3] *V-model (software development)* [online]. Wikipedia, The Free Encyclopedia., 17. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=V-model_(software_development)&oldid=1219370144.

[4] *Mobile app development* [online]. Wikipedia, The Free Encyclopedia., 2. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Mobile_app_development&oldid=1216942252.

[5] *List of NFC-enabled mobile devices* [online]. Wikipedia, The Free Encyclopedia., 20. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=List_of_NFC-enabled_mobile_devices&oldid=1214670918.

[6] *Waterfall model* [online]. Wikipedia, The Free Encyclopedia., 20. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Waterfall_model&oldid=1219904700.

[7] *Near-field communication* [online]. Wikipedia, The Free Encyclopedia, 27. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Near-field_communication&oldid=1220982780.

[8] *Radio-frequency identification* [online]. Wikipedia, The Free Encyclopedia., 27. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Radio-frequency_identification&oldid=1220982744.

[9] *Client–server model* [online]. Wikipedia, The Free Encyclopedia., 30. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=1221456053.

[10] *Wi-Fi* [online]. Wikipedia, The Free Encyclopedia., 30. 4. 2024. Dostupné z: https://en.wikipedia.org/w/index.php?title=Wi-Fi&oldid=1221493333.

[11] *NFC Data Exchange Format* [online]. Wikipedie: Otevřená encyklopedie., 5. 10. 2023. Dostupné z: https://cs.wikipedia.org/w/index.php?title=NFC_Data_Exchange_Format&oldid=23242805.

[12] *Iterative and incremental development* [online]. Wikipedia, The Free Encyclopedia., 9. 4. 2024. Dostupné z: hhttps://en.wikipedia.org/w/index.php?title=Iterative_and_incremental_development&oldid=1218026908.

[13] Attollo Technology LLC. *Upace* [online]. 2024. Dostupné z: https://www.upaceapp.com/.

[14] Cameron, C. *Equipment Usage Tracking Technologies* [online]. 24. 1. 2017. Dostupné z: https://www.athleticbusiness.com/facilities/fitness/article/15148725/equipment-usage-tracking-technologies.

[15] Carvalho Ota, F. K., Roland, M., Hölzl, M., Mayrhofer, R. a Manacero, A. Protecting Touch: Authenticated App-To-Server Channels for Mobile Devices Using NFC Tags. *Information*. 2017, sv. 8, č. 3. DOI: 10.3390/info8030081. ISSN 2078-2489.

[16] Dart. *Dart language* [online]. Dostupné z: https://dart.dev/language.

[17] Flutter. *Flutter FAQ* [online]. Dostupné z: https://docs.flutter.dev/resources/faq.

[18] Flutter. *Flutter showcase* [online]. Dostupné z: https://flutter.dev/showcase.

[19] Flutter. *Simple app state management* [online]. Dostupné z: https://docs.flutter.dev/data-and-backend/state-mgmt/simple.

[20] Future, R. *5 NFC Forum Tags Types You Need to Know* [online]. Dostupné z: https://www.rfidfuture.com/nfc-forum-tags-tyeps.html.

[21] GeeksForGeeks. *Flutter vs Native: Which is Best in 2024* [online]. 5. 2. 2024. Dostupné z: https://www.geeksforgeeks.org/flutter-vs-native/.

[22] Giese, D., Liu, K., Sun, M., Syed, T. a Zhang, L. Security Analysis of Near-Field Communication (NFC) Payments. *CoRR*. 1. vyd. 2019, abs/1904.10623.

[23] GoToTags. *NFC Chip UID* [online]. Dostupné z: https://gototags.com/nfc/chip/features/uid.

[24] Gymetrix. *Introduction to Gymetrix* [online]. Dostupné z: https://static1.gymetrix.co.uk/.

[25] ISO. *ISO/IEC 18092:2023* [online]. 2023. Dostupné z: https://www.iso.org/standard/82095.html.

[26] Jetbrains. *What is cross-platform mobile development?* [online]. 22. 3. 2024. Dostupné z: https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-mobile-development.html.

[27] MIT. *UI Software Architecture* [online]. Dostupné z: https://web.mit.edu/6.813/www/sp18/classes/05-ui-sw-arch/.

[28] NFC Forum. *NFC Forum Specifications* [online]. 2024. Dostupné z: https://nfc-forum.org/build/specifications.

[29] Phiaton. *NFC bluetooth pairing: What is it and what are the benefits?* [online]. 26. 5. 2017. Dostupné z: https://phiaton.com/blogs/audio/nfc-bluetooth-pairing-what-is-it-and-what-are-the-benefits.

[30] PocketBase. *PocketBase Documentation* [online]. Dostupné z: https://pocketbase.io/docs/.

[31] Rogers, P. *A Fundamental Guide to Weight Training* [online]. 19. 3. 2020. Dostupné z: https://www.verywellfit.com/weight-training-fundamentals-a-concise-guide-3498525.

[32] Shobha, N. S. S., Aruna, K. S. P., Bhagyashree, M. D. P. a Sarita, K. S. J. NFC and NFC payments: A review. In: *2016 International Conference on ICT in Business Industry and Government (ICTBIG)*. 2016, s. 1–7. DOI: 10.1109/ICTBIG.2016.7892683.

[33] Victoria, J., Alcarria, A., Mirasol, S., Molina, I., Suarez, A. et al. Study of a Novel High Permeability Ferrite Sheet Intended for Near Field Communication Systems. In: *2019 IEEE International Symposium on Electromagnetic Compatibility, Signal and Power Integrity (EMC+SIPI)*. 2019, s. 78–83. DOI: 10.1109/ISEMC.2019.8825260.

[34] Wakdev. *Wakdev FAQ* [online]. 2024. Dostupné z: https://www.wakdev.com/en/knowledge-base/troubleshooting/frequently-asked-question.html.

[35] XMINNOV. *How To Connect To WIFI Using NFC Tag?* [online]. 14. 8. 2023. Dostupné z: https://www.rfidtagworld.com/news/nfc-tag-wifi.html.

Příloha A

Obsah přiloženého paměťového média

Přiložené CD obsahuje:

- *frontend* – zdrojové soubory aplikace, unit testy a návod na překlad a spuštění,
- *backend* – soubor pro spuštění lokálního testovacího backendu a databáze,
- *dotazniky* – dotazníky a jejich výsledky,
- *navrh* – diagramy a další soubory související s návrhem (mockup, wireframe, ERD apod.),
- *obrazky* - obrázky výsledné aplikace (screenshoty),
- *apk* – přeložený instalační soubor aplikace na zařízení Android,
- *navod* – návod na instalaci a základní ovládání aplikace,
- *latex* – zdrojové soubory textu,
- *pdf* – text práce ve formátu PDF.

Příloha B

Záhlaví kolekcí v databázi

Název atributu	Datový typ	Povinný	Popis
username	Text	Ano	Automaticky generované uživatelské jméno
name	Text	Ne	Jméno uživatele
email	Email	Ano	Email uživatele pro přihlášení
gym_admin	Relation(gyms)	Ne	Odkaz na posilovnu, u které je správce

Tabulka B.1: Kolekce **users** – vestavěná kolekce uživatelů. Nejsou zde zmíněné všechny atributy, jelikož PocketBase je ukládá na pozadí.

Název atributu	Datový typ	Povinný	Popis
name	Text	Ano	Název cviku
image	File	Ano	Obrázek cviku
primaryParts	Select	Ne	Výčet primárních partií; na výběr je *back*, *chest*, *thigh*, *biceps*, *triceps*, *shoulder*, *forearm*, *calf* a *abs*
secondaryParts	Select	Ne	Výčet sekundárních partií; výběr je stejný, jako u primárních
howToText	RichEditor	Ne	Popis, jak cvik provést
howToVideo	File	Ne	Videonávod, jak cvik provést

Tabulka B.2: Kolekce **exercises** – všechny dostupné cviky. Jedná se pouze o systémová data, takže do ní uživatelé nijak nezasahují.

Název atributu	Datový typ	Povinný	Popis
name	Text	Ano	Název posilovny
address	Text	Ano	Adresa posilovny

Tabulka B.3: Kolekce **gyms** – všechny dostupné posilovny v systému.

Název atributu	Datový typ	Povinný	Popis
user	Relation(users)	Ano	Odkaz na uživatele
exercise	Relation(exercises)	Ano	Odkaz na cvik

Tabulka B.4: Kolekce **likes** – Ukládá oblíbené cviky uživatelů.

Název atributu	Datový typ	Povinný	Popis
name	Text	Ano	Název stroje
image	File	Ano	Obrázek stroje
exercises	MultiRelation(exercises)	Ne	Seznam všech dostupných cviků na stroji
gym	Relation(gyms)	Ano	Odkaz na posilovnu, ve které se stroj nachází
tags	Text	Ano	Seznam všech identifikátorů tagů, které jsou ke stroji přiřazeny; jednotlivé identifikátory jsou odděleny čárkou „,“
weightMin	Text	Ano	Minimum na závaží
weightStep	Text	Ano	Krok na závaží
measureType	Select	Ano	Typ měření, na výběr je: *kgCount*, *kmTime*, *kmCount*, *kgTime* a *other*

Tabulka B.5: Kolekce **machines** – ukládá stroje posiloven.

Název atributu	Datový typ	Povinný	Popis
user	Relation(users)	Ano	Odkaz na uživatele
exercise	Relation(exercises)	Ano	Odkaz na cvik
note	Text	Ano	Samotná poznámka

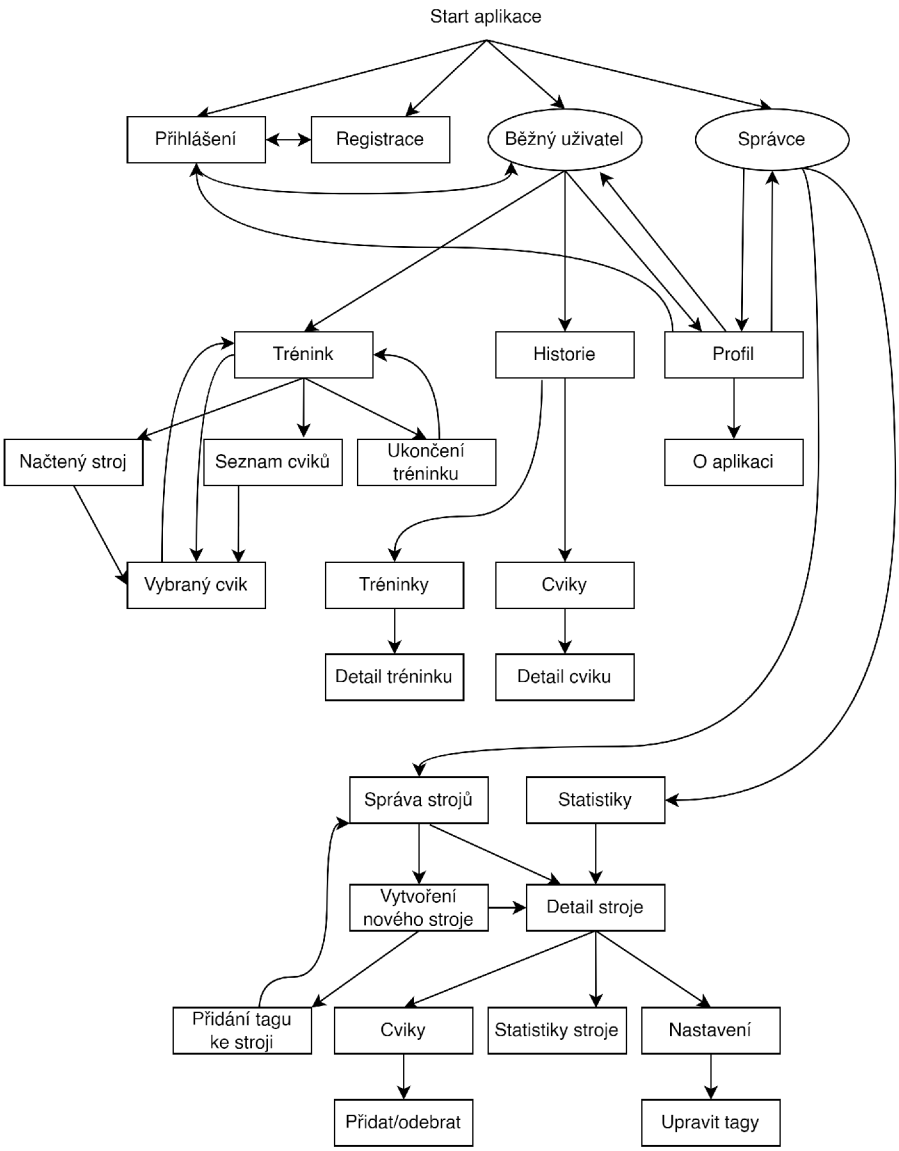
Tabulka B.6: Kolekce **notes** – ukládá poznámky uživatelů ke cvikům.

Název atributu	Datový typ	Povinný	Popis
user	Relation(users)	Ano	Odkaz na uživatele
start	DateTime	Ano	Začátek tréninku
end	DateTime	Ano	Konec tréninku
exercises	MultiRelation(exercises)	Ne	Proběhlé cviky; slouží pro vyhledávání v historii cviku
name	Text	Ne	Název tréninku
note	Text	Ne	Poznámka k tréninku
usedMachines	MultiRelation(machines)	Ne	Použité stroje; slouží pro vyhledávání v historii strojů
trainingData	JSON	Ano	Obsah tréninku; pro zachování konzistence musí obsahovat minimálně prázný JSON `{}`

Tabulka B.7: Kolekce **trainings** – ukládá tréninky.

Příloha C

Schéma navigace



Obrázek C.1: Schéma navigace v aplikaci