ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Návrh back-end řešení pro snížení digitálních závislostí

Jakub Lhoták
Otevřená informatika

Květen 2022
Vedoucí práce: doc. Ing. Daniel Novák, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lhoták** Jméno: **Jakub** Osobní číslo: **474732**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh back-end řešení pro snížení digitálních závislosti

Název diplomové práce anglicky:

Design of Back-end Application for Digital Addiction Reduction

Pokyny pro vypracování:

1) Seznamte se s problematikou digitálních závislostí
2) Navrhněte back-end aplikaci, která bude komunikovat s front-indovou aplikací
3) Aplikaci otestujte na 5 uživatelích

Seznam doporučené literatury:

K. Lukavska, The effects of parental control and warmth on problematic internet use in adolescents: A prospective cohort study, Journal of Behavioral Addictions, 2020
K. Lukavska, The immediate and long-term effects of time perspective on Internet gaming disorder, Journal of Behavioral Addictions, 2020
Vondrackova, P., & Gabrhelık, R. (2016). Prevention of internetaddiction: A systematic review.Journal of Behavioral Addic-tions,5(4), 568–579

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Daniel Novák, Ph.D. Analýza a interpretace biomedicínských dat FEL

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

Datum zadání diplomové práce: **14.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **19.02.2024**

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

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu práce panu doc. Ing. Danielovi Novákovi, Ph.D. za velké množství užitečných podnětů, návrhů a připomínek, které mi v průběhu mé práce na projektu poskytl.

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

V Praze dne 19. května 2022

.....

Abstrakt / Abstract

Tato práce se zabývá návrhem a implementací back-end řešení pro mobilní aplikaci sloužící ke snížení digitálních závislostí u dětí. Back-end řešení bylo implementováno tak, aby odpovídalo funkčním požadavkům klientské aplikace a také aby byl umožněn sběr a analýza uživatelských dat. Výsledkem práce je funkční a již nasazený server, který vyřizuje požadavky klientských aplikací. V rámci práce bylo provedeno i testování aplikace na uživatelích, při kterém se odhalily některé nedostatky systému.

Klíčová slova: digitální závislosti, rodičovská kontrola, klient–server model, back-end, sběr dat

This thesis deals with the design and implementation of a back-end service for a mobile application, which was made to deal with the digital addiction of children. The back-end service implementation was based on client application functional requirements. The service also allows users' data to be collected and later analyzed. The result of this work is a functional and already deployed server, which can process client applications requests. Part of the work is also the user testing of the application, which revealed some of the system's flaws.

Keywords: digital addiction, parental control, client–server model, back-end, data collection

Title translation: Design of Back–end Application for Digital Addiction Reduction

Obsah /

/ Obrázky

Kapitola 1
Úvod

V dnešním moderním světě se téměř každý den setkáváme s nejrůznějšími formami mobilních aplikací. Některé používáme pro zábavu, jiné pro komunikaci či například k práci. Jedním z nejdůležitějších aspektů kvalitní mobilní aplikace je propracované uživatelské rozhraní, které umožní uživatelům pohodlné ovládání aplikace. Některé funkce aplikací ovšem nelze vyřešit pouze klientskou částí aplikace a je potřeba implementovat i serverové řešení, aby taková aplikace mohla fungovat naplno. Mobilních aplikací, které potřebují ke svému plnému fungování vzdálený server, je celá řada. E-mailoví klienti, sociální sítě či třeba aplikace pro předpověď počasí potřebují server, se kterým mohou komunikovat a předávat si tak důležitá data.

Zmíněné typy aplikací ale zdaleka nejsou jedinými aplikacemi, které využijí back-end řešení. Edukační nebo terapeutické aplikace také využijí vzdálený server, ať už pro sběr důležitých informací nebo například k synchronizaci uživatelských profilů a dat. Příkladem takové aplikace může být nově vznikající aplikace *Digital Parenting*, které se tato diplomová práce týká.

1.1 Cíl projektu

Práce popisuje proces návrhu, vývoje a integrace back-end řešení pro mobilní terapeutickou aplikaci Digital Parenting. Čtenáře v ní seznámím s hlavními myšlenkami celého projektu. Pokusím se o vysvětlení jednotlivých komponent systému a také vysvětlím jak tyto komponenty mezi sebou komunikují.

Práce je rozdělena do kapitol, které se věnují konkrétní problematice. V kapitolách se čtenář dočte o současné problematice digitálního rodičovství, o samotné aplikaci Digital Parenting a především o implementaci serverové části aplikace, která je nyní nedílnou součástí celého systému.

Kapitola 2
Rodičovství v digitální éře

V této kapitole se pokusím shrnout jakou problematikou se zabývá takzvané *digitální rodičovství*. Jedná se však pouze o velmi stručné shrnutí komplexního problému dnešní společnosti.

2.1 Riziko technologií

Svět informačních technologií se neustále vyvíjí. To, co dnes považujeme za běžnou součást našich životů, bylo dříve naprosto nemyslitelné. Většina lidí dnes vlastní chytrý telefon, na kterém tráví nemalou část dne. Zařízení používají ke čtení e-mailů, zpráv či třeba ke sledování on-line videí. Mobilní telefony a další digitální zařízení (tablety, počítače, konzole) jsou stále častěji používány malými dětmi. Děti na těchto zařízeních tráví stejně, někdy dokonce i více času, než jejich rodiče. Nadměrné používání těchto zařízení může ovšem u dětí způsobovat vážná zdravotní rizika jako jsou například obezita, problémy se spánkem, poruchy pozornosti a další [1].

Kromě zdravotních rizik tu jsou ovšem i další problémy. Jedním z nich může být škodlivý či návykový obsah, ke kterému se dítě (zejména skrze internet) může dostat. Problematický a návykový obsah na internetu může vést k vytvoření špatných návyků, či dokonce závislostí. Jedná se pak o takzvané *problematické používání internetu* [2].

2.2 Digitální výchova

Každý zodpovědný rodič chce pro své dítě zajistit co nejkvalitnější životní podmínky. Ovšem prostředí, ve kterém děti vyrůstají dnes (tedy mobily, internet, sociální sítě), je naprosto odlišné od toho, co si pamatují jejich rodiče. Některé studie ukazují, že děti přijdou do styku s digitálními zařízeními již v prvním roce jejich života [3].

Rodiče dost často netuší, jakým způsobem by se svými dětmi měli mluvit o používání digitálních zařízení. Neví jaký maximální čas u obrazovek by jejich dítě mělo strávit, nebo jaké správné návyky by měli své děti učit. Tato problematika je v dnešní době velmi aktuální a stále roste počet rodičů, kteří by ocenili rady ohledně rodičovství v dnešním digitálním světě.

2.3 Edukace rodičů

Zásadním krokem, ke zlepšení celé této situace, je edukace rodičů. Rodiče si sice uvědomují, že jejich dítě nejspíše tráví u obrazovek příliš mnoho času, ovšem sami neví jak tento čas regulovat. Také dost často nemají představu jak by se svým dítětem o tomto tématu měli mluvit.

Existují studie, které se touto problematikou zabývají a přinášejí různé poznatky ohledně správných a špatných rodičovských přístupů. Problém ovšem zůstává v tom, jakým způsobem, lze tyto informace předat rodičům.

Jedním z řešení mohou být edukativní aplikace do telefonu, které by svým uživatelům, tedy rodičům, pomáhaly s výchovou a dodávaly jim cenné informace ohledně výchovy dětí v digitálním světě. Tato myšlenka stála ze vytvořením aplikace Digital Parenting.

Kapitola 3
Digital Parenting

Tato kapitola se věnuje aplikaci Digital Parenting, zejména její klientské části. Čtenáři vysvětlím jak aplikace funguje, co uživateli poskytuje a jaké všechny funkcionality nabízí. Zmíním se zde také o tom, jaké benefity přináší rozšíření systému o back-end řešení.

3.1 Seznámení s aplikací

Aplikace Digital Parenting je naučná mobilní aplikace, která se snaží vést rodiče k lepšímu porozumění problematiky digitálního rodičovství. Celý projekt má za cíl usnadnit rodičům přístup k informacím ohledně digitální výchovy jejich dětí a také je v tomto směru motivovat k lepšímu rodičovskému přístupu. Aplikace má jednoduché a přehledné uživatelské rozhraní, které by mělo vyhovovat většině uživatelů.

Aplikace slouží pouze jako informativní pomůcka pro rodiče. Součástí systému tedy není žádný modul, který by přímo sledoval nebo omezoval činnosti, které dítě na jeho zařízení (či zařízení rodiče) smí provádět. Takové aplikace dnes již existují. Za zmínku stojí například *Kaspersky Safe Kids*[1], *Norton Family*[2] nebo *Family Link*[3]. Implementace podobné funkcionality do naší aplikace byla zvažována, ovšem v době psaní této práce součástí systému není.

3.2 Používání aplikace

Při prvním spuštění aplikace proběhne takzvaný *onboarding* uživatele. Nejprve se v aplikaci zobrazí krátký strukturovaný text, který uživateli v rychlosti vysvětlí, jaký je smysl aplikace a co má od ní očekávat. Dále následuje prvotní přihlášení do systému, při kterém proběhne registrace nového uživatele, případně přihlášení již existujícího uživatele. Do registračního (přihlašovacího) formuláře uživatel zadá svoji e-mailovou adresu společně s heslem a tím se do aplikace přihlásí. V následujícím kroku uživatel vytvoří pod svým uživatelským účtem své první dítě. Zde pouze zadá jméno (případně zdrobnělinu nebo přezdívku) a datum narození dítěte. Po úspěšném vytvoření dítěte se uživatel dostává do aplikace, kde má přístup k veškerému obsahu. V aplikaci se nachází 3 důležité komponenty: *Sezení*, *Pravidla* a *Profil*.

3.2.1 Komponenta Sezení

Komponenta Sezení je nejdůležitější částí celé aplikace. Komponenta se skládá z celkem 5 edukativních interaktivních prezentací (*sezení*), které rodiče postupně seznamují s jednotlivými výzvami digitálního rodičovství. Kromě informování rodičů je komponenta Sezení zodpovědná také za vytvoření takzvaných *pravidel*. Jako pravidlo se v systému

[1] `https://www.kaspersky.cz/safe-kids`
[2] `https://family.norton.com/web/`
[3] `https://families.google.com/intl/cs/familylink/`

označuje jakási dohoda mezi rodičem a dítětem, která vznikla na základě doporučení z jednotlivých sezení. Rodič by se měl v ideálním případě snažit o to, aby jeho dítě tato pravidla dodržovalo.

Každé sezení se zaměřuje na jinou problematiku. V sezeních je zahrnut následující seznam témat:

- Monitoring času před obrazovkami
- Omezení času před obrazovkami
- Zóny a činnosti bez obrazovek
- Mediální obsah
- Bezpečnost na internetu

Další sezení jsou teprve ve fázi přípravy a vývoje.

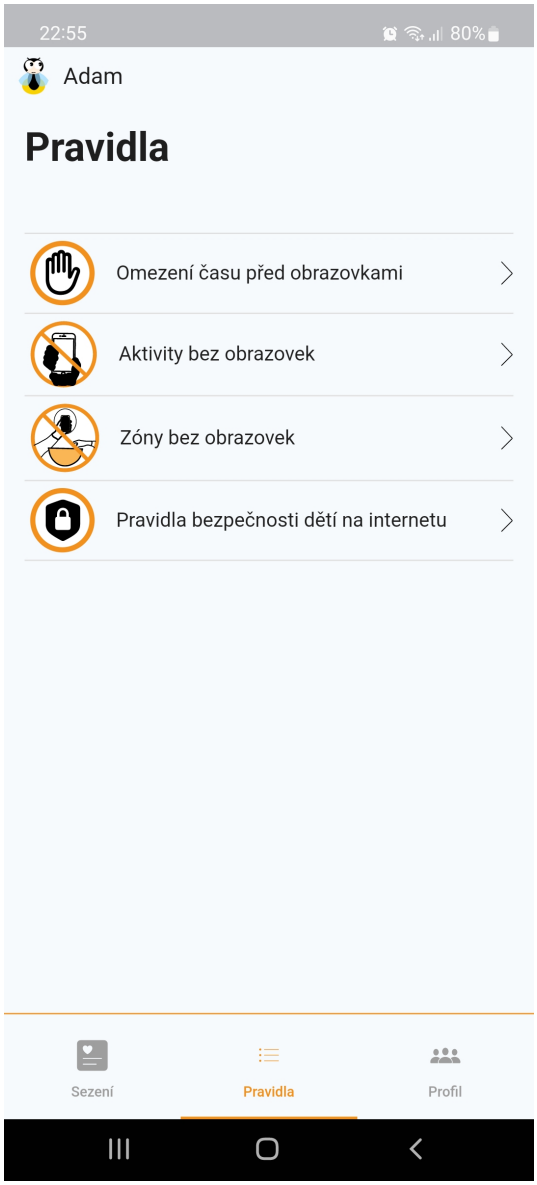
Pro lepší představu přikládám snímek obrazovky (obrázek 3.1) zachycující obsah druhého sezení.

Obrázek 3.1. Digital Parenting – část obsahu druhého sezení

3.2.2 Komponenta Pravidla

Komponenta Pravidla slouží jako rychlá rekapitulace všech pravidel, které si uživatel v aplikaci nastavil. Uživatel si v této komponentě může pravidla přidat nebo odebrat.

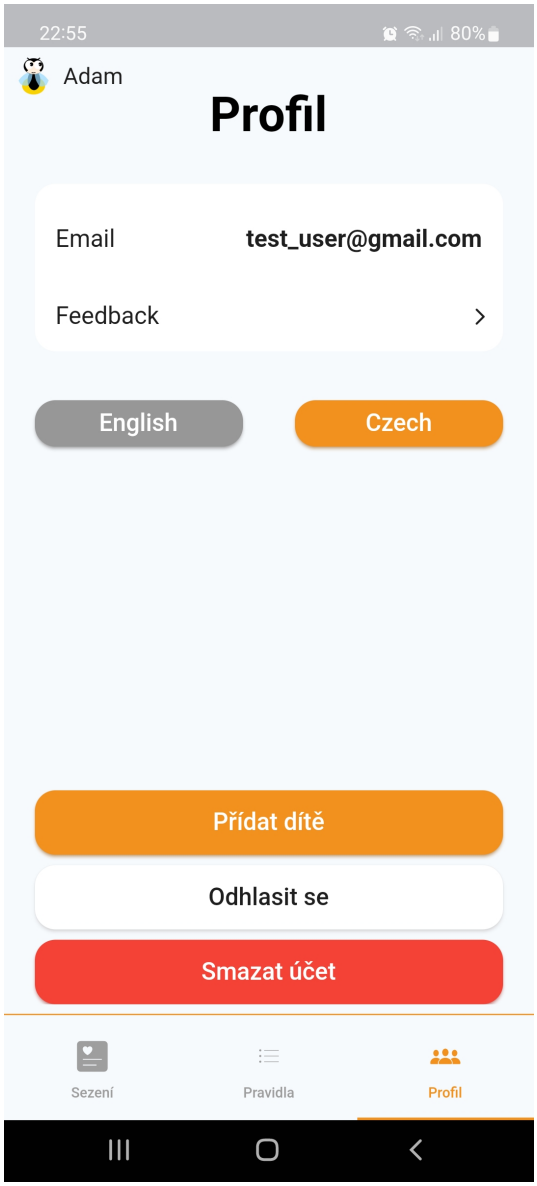
Na obrázku 3.2 se nachází snímek obrazovky, na kterém lze vidět seznam jednotlivých pravidel.



Obrázek 3.2. Digital Parenting – komponenta Pravidla

3.2.3 Komponenta Profil

V komponentě Profil nalezne uživatel všechny důležité informace a funkce týkající se jeho uživatelského účtu. Vidí zde svoji e-mailovou adresu, může do systému přidat nové dítě, může se odhlásit a pro úplnost je zde k dispozici i funkce sloužící ke smazání uživatelského účtu. Dále zde uživatel najde také přepínání jazyku aplikace. Uživatel má na výběr mezi češtinou a angličtinou. Verze pro operační systém Android má v komponentě



Obrázek 3.3. Digital Parenting – komponenta Profil

Profil také odkaz k jednoduchému vytvoření e-mailové zprávy pro zpětnou vazbu uživatelů (*Feedback*). Komponentu Profil si lze prohlédnout na přiloženém snímku obrazovky (obrázek 3.3).

3.3 Multiplatformní vývoj

Pokud má být mobilní aplikace úspěšná, musí být podporována co nejvíce zařízeními a co nejvíce operačními systémy. Při vývoji Digital Parenting se na toto myslelo a aplikace je nyní ve fázi vývoje pro 2 nejběžnější mobilní operační systémy: Android a iOS.

3.3.1 Vývoj pro operační systém iOS

Původně byla aplikace Digital Parenting pouze klientská aplikace pro telefony s operačním systémem iOS. Tato verze aplikace byla napsána v programovacím jazyce *Swift*[4]

[4] `https://developer.apple.com/swift/`

za použití frameworku *SwiftUI*[5]. O vývoji této aplikace pojednává bakalářská práce *Design of Therapeutical Application for Digital Addiction Reduction* [4], která může posloužit jako technická zpráva či dokumentace k iOS klientské aplikaci.

3.3.2 Vývoj pro operační systém Android

Většina uživatelů mobilních telefonů (v roce 2022 přibližně 70% [5]) používá operační systém Android. Aby bylo možné používat naši aplikaci i v těchto zařízeních, bylo nutné zahájit vývoj aplikace i pro tento operační systém.

Verze aplikace pro operační systém Android začala vznikat přibližně půl roku po zahájení vývoje iOS verze. Android verze se svým vzhledem a ovládáním snaží přiblížit verzi pro iOS. V době psaní této práce ještě nejsou některé nejnovější funkcionality pro Android verzi implementovány. Směr vývoje Android aplikace se však snaží být téměř totožný s její iOS variantou.

3.4 Benefity back-end řešení

Klientské aplikace Digital Parenting jsou sice tím nejdůležitějším prvkem celého systému, některé funkce ovšem ke svému plnému využití vyžadují i serverové (back-end) řešení. V následujících dvou podsekcích jsou vysvětleny největší výhody, které implementace back-end řešení přináší.

3.4.1 Uchování uživatelských dat

Samotná klientská aplikace má všechna svá data pevně vázána pouze k danému zařízení, v našem případě k mobilnímu telefonu. Pokud by tedy uživatel zadal do aplikace nějaká data, pak všechny tyto informace zůstanou pouze v jeho zařízení. Není tedy možné, aby si uživatel s přechodem na nový telefon svoje data z předchozího telefonu obnovil.

Pokud bude existovat serverová služba, která se může starat o data uživatelů, pak je jednoduché implementovat například uživatelské profily, se kterými budou data skrze server spojena. Toto byla jedna z hlavních motivací za vznikem back-end řešení.

3.4.2 Sběr a analýza dat

Další problém čistě klientských aplikací je, že systém nemůže sbírat žádná data zadaná uživateli. Zanalyzovaná uživatelská data by se hodila například ke zkvalitnění klientských aplikací. Pokud designer aplikace ví, které funkce uživatelé používají a které naopak ne, může aplikaci upravit a posunout aplikaci směrem, který si uživatelé přejí.

Sběr uživatelských dat ale přináší další užitečnou vlastnost a tou je možnost prezentování informací a dat ze systému specializovaným odborníkům. Ti pak mohou z nasbíraných dat od rodičů usuzovat různé závěry, na základě kterých mohou podávat nová cílená doporučení.

Díky této myšlence vnikla kromě back-end řešení také jakási informativní nástěnka, na které může specializovaný odborník, či správce systému sledovat agregovaná data od uživatelů. Více se o této nástěnce čtenář dozví v kapitole 8.

[5] `https://developer.apple.com/swiftui/`

Kapitola 4
Analýza požadavků na aplikaci

Během vývoje softwarových produktů velmi často dochází k vzájemnému nepochopení jednotlivých stran podílejících se na vývoji. Toto nepochopení může vést klidně i k zanesení relativně zbytečných chyb do již fungujícího systému. Zainteresované osoby, jako jsou například programátoři, vedoucí projektu či zadavatelé, se musí shodnout na výsledném produktu a musí mít upevněnou představu o finálním výsledku nebo směru vývoje. Chyby, které vznikají nedostatečnou specifikací projektů, jsou velmi časté. Dle některých studií 40 až 50% všech chyb zanesených do systému způsobuje právě nedostatečná specifikace požadavků [6].

Při vývoji back-end řešení pro aplikaci Digital Parenting se na problém nespecifikování požadavků myslelo a celý tým podílející se na vývoji se mu snažil předejít pořádáním týdenních schůzek, na kterých se domlouvaly další kroky vývoje naší aplikace. Design systému a s tím i spojené požadavky na aplikaci tedy vznikaly postupně na jednotlivých schůzkách.

Požadavky na aplikaci se zpravidla dělí na 2 typy: *Funkční požadavky* a *Nefunkční požadavky*. Funkční požadavky na aplikaci jsou specifické tím, že odpovídají na otázku: *„Co by měl systém dělat?“*. Specifikují funkce, které systém uživatelům umožňuje provádět. Nefunkční požadavky na druhou stranu odpovídají na otázky typu: *„Jaký by měl systém být?“*. Nefunkční požadavky se zaměřují spíše na kvalitu funkcí poskytovaných systémem a to zejména z technického hlediska.

4.1 Funkční požadavky na back-end řešení

Jak již bylo zmíněno výše, design architektury i samotný vývoj celého systému kolem aplikace Digital Parenting probíhal postupně. Směr vývoje naší aplikace se každou další schůzkou blíže specifikoval. Následujících několik bodů popisuje všechny funkční požadavky, které byly kladeny na mnou implementované back-end řešení systému Digital Parenting:

- **FR01 – Sběr uživatelských dat**
  Tento požadavek je víceméně primárním důvodem vzniku back-end řešení. Tímto požadavkem je specifikováno, že data zadaná uživateli do systému musí být uložena na vzdáleném serveru.

- **FR02 – Vytváření uživatelských účtů**
  Systém umožňuje rodiči založit si vlastní uživatelský účet.

- **FR03 – Synchronizace uživatelského účtu**
  Systém automaticky synchronizuje uživatelská data po přihlášení do aplikace.

- **FR04 – Správa uživatelského účtu**
  Uživatel může upravovat svůj uživatelský účet (včetně změny hesla).

- **FR05 – Smazání uživatelského účtu**
  Uživatel může smazat svůj uživatelský účet.

- **FR06 – Zobrazení agregovaných dat**
  Systém umožňuje správci či specializovanému expertovi zobrazit agregovaná data ze systému.

4.2 Nefunkční požadavky na back-end řešení

Součástí požadavků na aplikaci by měly být i požadavky nefunkční. Při návrhu back-end systému se jich specifikovalo hned několik. Nefunkční požadavky jsou zaznamenány v následujícím seznamu bodů:

- **NFR01 – Korektnost uživatelských dat**
  Všechna uživatelská data uložená na serveru musí být validní a v korektním formátu.

- **NFR02 – Smazání uživatelského účtu odstraní všechna uživatelská data**
  V případě smazání uživatelského účtu musí být všechna data, která jsou s daným účtem spojena, odstraněna.

- **NFR03 – Rozšiřitelnost datového modelu**
  Datový model reprezentující entity sytému musí být snadno rozšiřitelný.

- **NFR04 – Konfigurace**
  Systém lze konfigurovat skrze konfigurační soubory – konfigurační parametry nesmí být součástí kódu.

- **NFR05 – Šifrování komunikace**
  Komunikace s back-end řešením musí probíhat skrze šifrovaný kanál. Tímto se zabrání nechtěnému úniku citlivých uživatelských dat jako je e-mailová adresa nebo heslo.

- **NFR06 – Hašování uživatelských hesel**
  Hesla v databázi nesmí být uložena pouze jako prostý text. Pro ukládání hesel musí být použita *kryptografická hašovací funkce*.

- **NFR07 – Nasazení back-end řešení**
  Součástí projektu je i nasazení back-end řešení tak, aby bylo ihned k použití klientskými aplikacemi.

Kapitola 5
Analýza architektury aplikace

Tato kapitola se zabývá analýzou softwarových architektur a následným výběrem konkrétní architektury pro back-end řešení aplikace Digital Parenting. Čtenáře seznámím s nejčastějšími typy softwarových architektur a provedu mezi nimi porovnání. Dále vysvětlím pojmy jako jsou *klient–server model* či *REST API*. Jelikož náš systém pracuje s citlivými daty, je součástí kapitoly také sekce o zabezpečení uživatelských dat.

5.1 Softwarová architektura

Pro pojem *softwarová architektura* neexistuje žádná oficiální definice. Na druhou stranu většina definic tohoto pojmu má společné jádro a ve svém významu se často liší pouze svým záběrem, důrazem na určitý aspekt či svou detailností. Jako definici softwarové architektury zde uvedu tuto definici, která je výsledkem diskusní skupiny věnované problematice softwarové architektury v Institutu softwarového inženýrství při Carnegie Mellon University v Pittsburghu:

Softwarová architektura je struktura komponent programu/systému, jejich vzájemné vazby, principy a předpisy určující jejich návrh a vývoj v průběhu času. [7]

Softwarová architektura tedy popisuje co všechno je součástí systému, jaké jsou mezi jednotlivými částmi systému vztahy a jakým způsobem tyto části a vztahy vznikají.

Volba ideální softwarové architektury je klíčová při vývoji každé aplikace. Málokdy existuje pouze jedna zřejmá *„správná“* cesta a z toho důvodu je důležité dopředu zvážit, který směr se má pro vývoj daného softwaru zvolit.

5.1.1 Typy softwarových architektur

Softwarových architektur existuje celá řada. Často se však rozlišují na 2 základní typy. Jedná se o architektury *monolitické* a *distribuované*. Oba z těchto typů architektur nacházejí své uplatnění v rozdílných typech projektů. V následujících podsekcích se čtenář o obou typech architektur dozví více podrobností. Poté následuje jejich vzájemné porovnání.

5.1.2 Monolitická architektura

Monolitická architektura se často označuje za konvenční metodu vývoje softwaru. Za monolitickou architekturu můžeme označit takové systémy, které jsou vyvíjeny jako jeden ucelený balíček. Celý systém se sice může skládat z více logicky oddělených celků, ovšem finální produkt (aplikace, server...) se rozdělit nedá. [8].

5.1.3 Distribuovaná architektura

Distribuované architektury jsou na rozdíl od monolitických rozděleny do několika samostatných a nezávisle na sobě fungujících jednotek (modulů), které spolu komunikují. Každá z těchto samostatných jednotek je pak zodpovědná za určitou část funkcionalit

celého systému. Komunikace modulů probíhá zpravidla po síti pomocí předem definovaných aplikačních rozhraních. Tato skutečnost umožňuje modulům, aby pracovaly na oddělených strojích, což s sebou přináší řadu výhod i nevýhod.

V poslední době se stala velmi populární distribuovanou architekturou architektura *Microservices*. V případě architektury Microservices je výsledný produkt (softwarové řešení) složen z velkého počtu velmi malých samostatných modulů. Tyto moduly se starají vždy pouze o jednu konkrétní funkcionalitu. Jedním z důvodů velké popularity architektury Microservices je také open source projekt *Docker*[1], který vývojářům umožňuje snadno provozovat jednotlivé moduly aplikace v vzájemně oddělených kontejnerech.

5.1.4 Srovnání monolitické a distribuované architektury

Rozdílů mezi zmíněnými typy architektur je mnoho. Vývoj monolitických architektur bývá zpravidla jednodušší a přímočařejší. Provoz a správa systémů s monolitickou architekturou je v porovnání se systémy s distribuovanou architekturou levnější a jednodušší. Na druhou stranu distribuované architektury mají své výhody zejména ve škálovatelnosti, rozšiřitelnosti a dostupnosti. Škálovatelnost a dostupnost systémů s distribuovanou architekturou je zajištěna především tím, že jednotlivé moduly mohou fungovat na různých oddělených počítačových systémech. Výkon systému tedy není omezen na výkon jednoho stroje, jako tomu bylo u architektur monolitických. Kromě výkonu má rozdělení systému na více výpočetních zařízení také pozitivní vliv na dostupnost. Pokud máme stejný modul na více strojích, tak v případě výpadku jednoho konkrétního počítače může systém fungovat dále za použití zbylých počítačů v systému.

Distribuované architektury jsou v posledních letech velmi populární, a to zejména z výše zmíněných důvodů jako je škálovatelnost a dostupnost. Jejich popularita také vzrůstá díky rozvoji takzvaných *cloudových technologií*. Jako cloudové technologie můžeme označit různá online softwarová řešení, která umožňují svým klientům využít sdílený výpočetní výkon či datové úložiště. Pro distribuované architektury často platí, že výsledný systém může být vyvíjen a nasazován nezávisle na sobě.

Velkou nevýhodou distribuovaných architektur je jejich zvýšená komplexita. U monolitických architektur často platí, že celý systém je psán v jednom, případně v několika málo programovacích jazycích. V případě distribuovaných architektur mohou být jednotlivé moduly vytvořeny pomocí naprosto odlišných technologií. Pochopení celé architektury a vnitřního fungování systému se tedy u distribuovaných architektur stává mnohem složitější.

Pro snazší srovnání obou architektur shrnu v několika následujících bodech jejich hlavní výhody a nevýhody:

- **Výhody monolitických architektur:**
  - snazší návrh
  - rychlejší vývoj (alespoň ze začátku)
  - lehčí pochopení celého systému
  - levnější na údržbu/správu
  - snazší nasazení systému
  - lehčí testování systému

- **Nevýhody monolitických architektur:**
  - omezená škálovatelnost
  - s rostoucí velikostí projektu se zvyšuje komplexita

[1] `https://www.docker.com/`

- složitý přechod na nové technologie (musí se změnit celý systém)
- omezené paralelní zpracování (pouze jeden stroj)

- **Výhody distribuovaných architektur:**
  - velmi dobrá škálovatelnost projektu
  - dostupnost
  - nezávislé nasazování jednotlivých modulů
  - projekt je lehce rozšiřitelný o další funkcionality
  - snadné testování samostatných modulů
  - rychlejší adaptace na změnu technologií

- **Nevýhody distribuovaných architektur:**
  - složitý návrh systému
  - přidána komplexita navíc při vývoji
  - dražší a složitější provoz (při velkém množství modulů)
  - složité testování celého systému (musíme mít k dispozici všechny moduly)

5.1.5 Výběr softwarové architektury pro projekt Digital Parenting

Ačkoli jsou distribuované architektury pro vývoj softwarových řešení stále populárnější volbou, rozhodl jsem se pro back-end server aplikace Digital Parenting zvolit architekturu monolitickou. Jedním z hlavních důvodů je, že vývoj a návrh monolitické architektury probíhá rychleji a v případě menších projektů je to preferovaná volba architektury. Dalším důvodem pro tuto volbu byla skutečnost, že s vývojem systémů s distribuovanou architekturou nemám dostatečné znalosti a zkušenosti.

5.2 Klient–server model

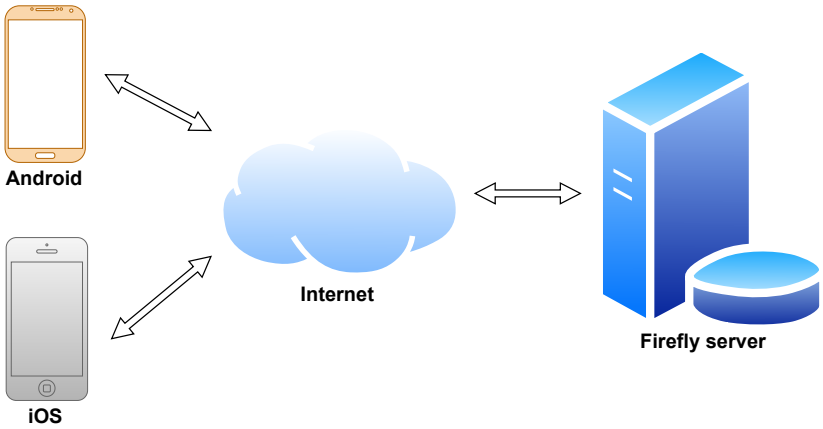
Samotný vznik back-end řešení by pro stávající projekt Digital Parenting žádnou hodnotu nepřidal. Aby mohly spolu jednotlivé části projektu (klientské aplikace a nově vzniklý server) vzájemně komunikovat, musely se obě strany vývoje shodnout na jednotném klient–server modelu, který se postupně implementoval.

Klient–server model je v informatice označení pro síťovou architekturu, která odděluje klientské části aplikace, jako je například webový prohlížeč či aplikace na mobilním telefonu, od serverové části, která zpracovává uživatelské požadavky. Klienti se serverem komunikují zpravidla skrze počítačovou síť. [9]

V případě projektu Digital Parenting je klient–server model relativně přímočarý. V systému existují 2 typy klientských aplikací – verze pro operační systém iOS a verze pro Android. Obě verze komunikují skrze internet se vzdáleným serverem, který vyřizuje požadavky klientských aplikací. Server vystavuje pouze jedno aplikační rozhraní a obě verze aplikace skrze toto rozhraní komunikují. Klient–server model systému Digital Parenting si lze prohlédnout na obrázku 5.1.

5.3 REST API

REST API (také RESTful API) je aplikační rozhraní (API), které umožňuje klient–server architektuře výměnu dat pomocí internetového protokolu *HTTP*. Jako REST API nazýváme takové rozhraní, které dodržuje sadu omezení REST. REST navrhl a popsal v roce 2000 Roy Fielding, který je jedním ze spoluautorů protokolu HTTP [10].



Obrázek 5.1. Digital Parenting – klient–server model

REST se používá pro jednotný a snadný přístup ke zdrojům. Zdrojem mohou být data, nebo také stavy aplikace (stavy musí jít popsat daty). Všechny zdroje mají vlastní identifikátor URI, pomocí kterého se jednoznačně určí daný zdroj. REST ke komunikaci využívá HTTP metody. Zpravidla se jedná o tyto 4:

- **GET** – vrátí datovou reprezentaci dotazovaného zdroje
- **PUT** – upraví data dotazovaného zdroje
- **POST** – vytvoří nová data
- **DELETE** – smaže data z dotazovaného zdroje

Rozhraní REST není protokol ani standard. Z tohoto důvodu existuje mnoho způsobů jak toto rozhraní navrhnout a implementovat. Data, která jsou skrze toto rozhraní posílána, mohou být v různých datových formátech. Nejčastěji se používají formáty XML nebo JSON. Komunikace ovšem může probíhat i pomocí prostého textu. Nedílnou součástí REST API jsou také hlavičky a metadata obsažena v HTTP požadavcích (*HTTP request*) a odpovědích (*HTTP response*). Obsahují totiž důležité informace jako jsou URI identifikátor, autorizační hlavička, cookies a další [11].

Součástí REST jsou také návratové status kódy protokolu HTTP. Pomocí nich lze například určit, zda při vykonávání požadavku nedošlo k chybě, případně o jakou chybu se jedná.

Back-end řešení aplikace Digital Parenting ke své klient–server komunikaci využívá REST API. Posílaná data jsou ve formátu JSON a ke komunikaci se využívá také autorizační hlavička HTTP požadavků, pomocí které se v systému řeší autentizace a autorizace uživatele.

5.4 Zajištění bezpečnosti uživatelských dat

V dnešním světě jsme již zvyklí vyřizovat různé činnosti online pomocí internetu. Často se ale v těchto požadavcích na vzdálené systémy nacházejí citlivá data, která nechceme sdílet s nikým jiným. Dobrým příkladem by mohlo být heslo k internetovému bankovnictví. Je naprosto zásadní, aby se k heslu nemohl dostat nikdo jiný než informační systém dané bankovní instituce. Aby byla tato bezpečnost systémů zajištěna, musely se navrhnout a implementovat různé bezpečnostní protokoly. Obor informatiky, který se tímto problémem zabývá, se nazývá *Počítačová bezpečnost*.

5.4.1 Počítačová bezpečnost

Počítačovou bezpečnost lze definovat jako zabezpečení počítačových systémů a informací proti poškození, krádeži dat nebo také neautorizovanému přístupu. Jedná se o proces, pomocí kterého se snažíme o prevenci a detekci neautorizovaného použití počítačového systému. [12]

Počítačová bezpečnost je relativně široký pojem, pod který spadá velké množství různých výzev a problémů. V této práci se budu věnovat 2 konkrétním problémům, které se přímo vztahují k návrhu architektury aplikace Digital Parenting, jsou jimi *hašování hesel* a *komunikace zabezpečeným kanálem*.

5.4.2 Hašování hesel v databázi

Hesla uživatelů jsou často těmi nejdůležitějšími a zároveň nejzranitelnějšími daty. Uživatelská hesla fungují jako autorizační prvek k uživatelským účtům. Heslo by měl znát pouze sám uživatel a nikdo jiný. Dokonce ani serverová služba, se kterou uživatel komunikuje, by neměla znát jeho heslo. V případě, že by došlo k úniku dat a uživatelské účty by byly zveřejněny spolu s jejich hesly, mohlo by to mít katastrofické následky.

K únikům dat dochází velmi často. Některé úniky dat dosahují enormních rozměrů. Je běžné, že v případě útoku na databázové systémy některých velkých společností mohou být ukradeny desítky až stovky milionů uživatelských údajů a dat [13]. Aby se při úniku dat zabránilo zneužití uživatelských hesel, neukládají se v databázových systémech hesla přímo. Ukládají se pouze takzvané otisky hesel (neboli *hashe*), vytvořené pomocí speciálních kryptografických hašovacích funkcí. Pokud na prostý text použijeme hašovací funkci, pak je prakticky nemožné z jejího výstupu rekonstruovat původní text. Toto je rozdíl oproti klasickému šifrování, kdy se pomocí klíče můžeme dostat k původní zprávě.

Ukládání otisků hesel v databázi úplně stačí k poskytnutí služby sloužící pro identifikaci uživatelského účtu. Je to dáno tím, že pokud uživatel odešle na server své heslo, server heslo zpracuje předem danou hašovací funkcí, a poté výsledek porovná s výsledkem uloženým v databázi. Implementace tohoto kryptografického modelu se v dnešní době stala jakýmsi standardem pro většinu informačních systémů, jejichž uživatelské účty jsou zabezpečeny hesly. Výjimkou není ani back-end řešení aplikace Digital Parenting, i zde je implementováno hašování uživatelských hesel.

5.4.3 Komunikace přes zabezpečený kanál

Díky internetu dnes mohou spolu komunikovat různé systémy napříč celým světem. Tato komunikace je sice velmi užitečná, může ovšem být i nebezpečná. Při komunikaci 2 systémů v počítačové síti internet nedochází k přímému spojení. Spojení je vytvořeno pomocí velkého množství různých, navzájem spojených uzlů, které si vzájemně přeposílají data na sítí. Tím zajišťují komunikaci jednotlivých systémů na internetu. Komunikace mezi jednotlivými uzly často není nijak chráněna proti neoprávněnému čtení posílaných dat. Z tohoto důvodu je velmi nebezpečné používat nezabezpečené internetové protokoly, jako je například HTTP. Data, která se při HTTP komunikaci posílají skrze internet, nejsou nijak šifrovaná a kdokoli se připojí k uzlu, skrze který komunikace probíhá, může tato data číst a zneužít.

V dnešní době existuje velké množství protokolů, které relativně bezpečnou komunikaci na internetu umožňují. Bývá to ale za cenu snížení výkonu a průtoku dat. To ovšem v dnešní době není až takový problém, jelikož výkon počítačů (i ostatních digitálních zařízení) se oproti době, kdy byly navrženy první internetové protokoly, mnohonásobně

zvýšil. Jedním z nejpopulárnějších internetových protokolů zajišťující zabezpečenou komunikaci v počítačové síti je protokol HTTPS. HTTPS je rozšířením protokolu HTTP o protokol *SSL* nebo *TLS*. Tyto 2 zmíněné protokoly patří do rodiny kryptografických protokolů a fungují na principu *asymetrické kryptografie*[2]. Klientská aplikace se serverem v takovém případě komunikuje pomocí *soukromých* a *veřejných klíčů*, které zajišťují takzvané *asymetrické šifrování*. [14]

Protokol HTTPS je v dnešní době víceméně standardem pro komunikaci webových prohlížečů s webovými servery. HTTPS se ovšem také dá využít k zabezpečení REST API. Pokud budou jednotlivé REST endpointy serverového řešení přístupné pouze skrze protokol HTTPS, pak je v takovém systému do jisté míry zaručena bezpečná komunikace po internetu.

Jedním z nefunkčních požadavků back-end řešení systému Digital Parenting – NFR05, je požadavek na použití šifrovaného komunikačního kanálu ke zprostředkování klient–server komunikace. Pro mne to znamenalo, že jsem musel vybrat takovou dostupnou hostovací službu, která umožňuje komunikovat pomocí protokolu HTTPS. Více se o výběru hostingu čtenář dozví v kapitole 6 v sekci 6.5.

[2] `https://www.techtarget.com/searchsecurity/definition/asymmetric-cryptography`

Kapitola 6
Návrh a implementace

Pravděpodobně nejdůležitějším krokem vývoje nového softwaru je jeho vytvoření, tedy implementace. Implementace ovšem není jedinou fází takzvaného *životního cyklu vývoje softwaru*[1]. Analýza a také návrh softwarového řešení bývají často stejně tak důležité jako samotná implementace.

Implementace řešení může ovšem začít až ve chvíli, kdy už má programátor představu o finálním produktu. Požadavky na back-end řešení a analýzu architektury aplikace Digital Parenting jsem už rozebral v kapitolách 4 a 5. V této kapitole se zaměřím na návrh a implementaci řešení. Součástí kapitoly je také sekce o nasazení řešení na online hostovací službu. Na konci kapitoly se zmíním o dokumentaci mého back-end řešení, která hrála klíčovou roli při implementaci klient–server komunikace s klientskými aplikacemi Digital Parenting.

6.1 Použité technologie

Před samotnou implementací je důležité, aby si programátor stanovil, jaké nástroje a technologie k vývoji použije. Výběr nástrojů pak často určuje, jakým směrem se bude vývoj a implementace řešení ubírat.

Stěžejní volbou při vývoji softwaru je volba programovacího jazyku či frameworku, ve kterém bude řešení napsáno. Dnes existují stovky jazyků a frameworků, které mohou programátoři k implementaci svých řešení použít. Výběr ideálního jazyku pro daný projekt bývá nelehký úkol. Programovací jazyky jsou často uzpůsobené k řešení specifických problémů a málokdy je ideální volba jazyku či frameworku zřejmá hned na začátku vývoje.

6.1.1 Vlastnosti a porovnání technologií

Při vývoji back-end řešení mají programátoři opravdu velké množství dostupných technologií, které mohou pro své projekty použít. Aby si programátor mohl zvolit ideální jazyk, musí nejdříve porovnat dostupné technologie z hlediska různých vlastností.

Při výběru jazyka či frameworku se často hledí na následující seznam vlastností. Tyto vlastnosti určují, na jaký typ projektu se daná technologie hodí a k čemu se používá. [15]

- **Rychlost vývoje**

  Čas, strávený implementací řešení, je jedním z hlavních aspektů, na které bychom se při vývoji softwarových řešení měli zaměřit. Pro programovací jazyky platí, že stejnou funkcionalitu může programátor ve dvou odlišných jazycích tvořit různě dlouho. Některé jazyky umožňují velmi rychlé psaní funkčního kódu. Tyto jazyky mají v sobě implementováno množství užitečných nástrojů, datových struktur či paradigmat, které mohou programátoři využívat a zkrátit si tak čas, který by trávili vytvářením těchto předem hotových nástrojů.

[1] celý proces vývoje softwaru – od požadavků až po instalaci a údržbu

Tento flexibilní programovací model ovšem přináší i velkou nevýhodu. Psaním kódů v programovacích jazycích zaměřených na rychlé vytvoření řešení může programátor do systému zanést mnoho chyb, které se poté složitě opravují. Často tedy platí, že vysoká rychlost vývoje řešení je poskytnuta na úkor *robustnosti* řešení.

- **Rychlost výsledného programu**

  Další důležitou vlastností programovacích jazyků je rychlost vykonávání výsledného programu. Programovací jazyky fungují na rozdílných principech a rychlost vykonávání stejného programu v odlišných jazycích se může znatelně lišit.

  Na výsledné rychlosti programu se nejvíce podílí to, zda je jazyk *kompilovaný*, či *interpretovaný*. Kompilované jazyky se před spuštěním musí takzvaně přeložit do strojového kódu, se kterým už umí procesor počítače pracovat. Překlad probíhá pomocí překladače (neboli kompilátoru), jimž bývá program, který převede zdrojový kód aplikace do spustitelného strojového kódu. Naproti tomu interpretované jazyky ke svému běhu využívají speciální program (interpret), který přímo vykonává instrukce zdrojového kódu, řádek po řádku.

  Kompilované jazyky bývají zpravidla několikanásobně rychlejší než interpretované jazyky. Důvodem je především přímé využití procesorových instrukcí a také možnost vyšší úrovně optimalizace při překládání.

  Pro back-end řešení aplikace Digital Parenting nebyla rychlost vykonávání kódu brána jako klíčový aspekt při rozhodování o použití dostupných technologií. Důvodem je, že součástí systému nejsou žádné výpočetně složité operace a zároveň systém používá relativně malý objem dat.

- **Robustnost kódu**

  Robustnost lze popsat jako schopnost kódu škálovat s rostoucí velikostí projektu. Za jazyky specializované na vysokou robustnost lze označit takové jazyky, které umožňují vývojářům psát velmi striktně kontrolovaný kód. Tyto jazyky nejsou zaměřeny na rychlý vývoj. Jak už jsem psal výše v bodě *Rychlost vývoje*, mezi rychlostí vývoje a robustností výsledného řešení se musí vytvořit jakési kompromisy.

- **Dostupnost vývojářů**

  Poslední bod seznamu se týká dostupnosti vývojářů. Není to sice technický aspekt volby programovacího jazyka, ovšem i tento bod by měl být při výběru jazyka či frameworku zahrnut. Dostupnost vývojářů pro danou technologii určuje, jak složité je rozšířit tým o nového programátora, který s touto technologií umí pracovat.

  Vzhledem k tomu, že se na celém projektu Digital Parenting podílí více vývojářů, bylo nutné pro back-end řešení vybrat takové technologie, které jsou konvenční, známé a často používané.

Na základě těchto bodů jsem postupoval při výběru technologií pro back-end řešení. Jelikož se jedná o webový server s vystaveným REST API rozhraním, existuje hned několik vhodných technologií, které jsem mohl pro projekt použít.

6.1.2 Programovací jazyk

Jako vhodné programovací jazyky pro tento projekt se nabízely tyto 2: *Java* a *Python*. S vývojem aplikací v obou těchto jazycích mám již určité zkušenosti. Pro oba tyto jazyky existuje hned několik frameworků pro vývoj webových serverů. Back-end řešení by tedy šlo vytvořit v obou těchto jazycích.

Programovací jazyk Python je populární interpretovaný programovací jazyk. Vytvořil jej *Guido van Rossum* a vydaný byl již v roce 1991 [16]. Python je velmi univerzální jazyk a používá se v různých odvětvích informatiky. Python je multiplatformní a funguje na většině moderních operačních systémech. Má jednoduchou syntax, která připomíná angličtinu. Programy v Pythonu vznikají velmi rychle, především díky jeho bohaté sadě věstavěných funkcí. Python je multiparadigmatický jazyk. To znamená, že programátor může psát kód hned několika způsoby. Jsou jimi *procedurální programování*, *objektově orientované programování* a také *funkcionální programování*. Velkou nevýhodou Pythonu je jeho rychlost. Programy napsané v Pythonu jsou ve srovnání s ostatními programovacími jazyky velmi pomalé.

Jak už jsme psal výše, Python je velmi univerzální jazyk a vývoj webových serverů je v Pythonu možný. Pro Python existuje hned několik dostupných webových frameworků, které velmi usnadňují vývoj webových serverů. Mezi nejpopulárnější takové framworky patří *Django*[2] nebo *Flask*[3].

Jazyk Java je také velmi populární programovací jazyk. První verzi Javy vyvinula firma *Sun Microsystems* a představila ji 23. května 1995. Stejně jako Python je i Java interpretovaný jazyk. Na rozdíl od Pythonu se však zdrojový kód Javy nejdříve překládán do takzvaného *bajtkódu*, který se pak dále vykonává na javovském interpretu – *Java Virtual Machine* (JVM). Novější verze JVM dokonce neintepretují bajtkód přímo, ale před svým prvním provedením dynamicky zkompilují bajtkód do strojového kódu daného počítače. Tento proces se nazývá *Just-in-time kompilace* (JIT). Díky tomu je Java v porovnání s Pythonem mnohem rychlejší co se týče samotného vykonávání programu. [17]

I pro Javu exstistuje velké množství webových knihoven a frameworků. Jako příklad uvedu framework *Spring Boot*[4] či knihovnu *Javalin*[5].

Oba jazyky, Python i Javu, lze pro back-end řešení použít. Výhodou Pythonu je rychlý a snadný počáteční vývoj, kdežto hlavní výhodou Javy je vyšší rychlost vykonávání programu a lepší škálovatelnost projektu. Pro back-end server aplikace Digital Parenting jsem se rozhodl použít jazyk Python a to zejména kvůli jeho rychlému a pohodlnému vývoji. Dalším důvodem pro jeho výběr byly také mé předešlé zkušenosti s psaním webových serverů v Pythonu.

6.1.3 Flask vs. Django

Frameworky Flask a Django jsou nejpopulárnější webové framoworky pro programovací jazyk Python. Oba fungují na velmi podobném principu, ovšem v pár přístupech se liší.

Django je komunitou velmi oblíbený, webový, open source framework umožnující rychlý vývoj webových aplikací. Obsahuje mnoho vestavěných knihoven, které usnadňují implementaci často používaných služeb. Velkou výhodou Djanga je také vestavěný modul umožňující *objektově relační mapovaní* (ORM). Tento modul umožňuje vytvořit automatickou konverzi dat mezi relační databází a Python objekty. V datovém modelu aplikace Digital Parenting se ovšem nacházejí komplexní vnořené objekty (více v sekci 6.2) a perzistentní data aplikace je vhodné ukládat spíše do *dokumentově orientované databáze*.

Framework Flask je oproti Djangu minimalističtější. Obsahuje méně vestavěných knihoven a přenechává vývojáři vlastní rozhodnutí nad použitím dalších služeb jako

[2] `https://www.djangoproject.com/`
[3] `https://flask.palletsprojects.com/`
[4] `https://spring.io/projects/spring-boot`
[5] `https://javalin.io/`

jsou například *NoSQL databáze*. Vzhledem k tomu, že back-end řešení systému bude pouze poskytovat REST API endpointy a bude využívat dokumentově orientovanou databázi, rozhodl jsem se raději pro použití frameworku Flask. Velkým argumentem pro použití Flasku byla také moje zkušenost z předešlých projektů, kde se mi Flask osvědčil.

6.2 Modelování systému

Předtím než se mohlo back-end řešení implementovat, musely se obě programátorské strany (tedy klienti a server) dohodnout, jakým způsobem se vymodelují jednotlivé entity v systému. V případě, kdy se chce zajistit klient–server komunikace dat, musí se vytvořit společný model, který obě strany implementují. Původní, čistě klientská aplikace Digital Parenting si v pozadí žádný model entit neudržovala. Bylo tedy nutné tento model vytvořit a držet se ho. Původní verze aplikace také nepočítala s uživatelskými účty a s vytvářením více dětí pod jedním rodičem. Tyto nedostatky se postupně odstranily a vznikl tedy model celého systému, podle kterého se vyvíjelo.

6.2.1 Entity v systému

Aplikace z pohledu dat rozlišuje 2 entity. První entita reprezentuje uživatelská data. Tato entita je v modelu označena jako *User*. User entita v sobě nese všechny informace ohledně jednoho konkrétního uživatele – rodiče. Druhou entitou je entita *Kid*. Ta v sobě nese veškeré informace a data, která se v aplikaci vztahují ke konkrétnímu dítěti. Kid entita v sobě nese také informace ohledně pokroku uživatele v aplikaci v rámci jednoho dítěte. Pomocí této informace lze určit, ve které fázi aplikace se rodič u konkrétního dítěte nachází, tedy kolik sezení má již za sebou.

6.2.2 Datový model

Spolčeně s definicí jednotlivých entit vznikl i datový model aplikace. Datový model určuje názvy a datové typy jednotlivých atributů obou entit v systému. Klientské aplikace i serverové řešení pracuje se stejným datovým modelem. Datový model reprezentovaný diagramem lze vidět na obrázku 6.1.

Datový model definuje 2 hlavní datové typy: *User* a *Kid*. Následující seznam obsahuje všechny atributy datových typů User a Kid i s vysvětlením, k čemu daný atribut v systému slouží.

- **User:**
  - **_id** – *textový řetězec*
    Identifikátor User entity
  - **lastUpdate** – *číslo*
    Datum a čas poslední změny User entity (Unix time[6])
  - **lastAccess** – *číslo*
    Datum a čas posledního přístupu k User entity (Unix time)
  - **creationTime** – *číslo*
    Datum a čas vytvoření User entity (Unix time)
  - **email** – *textový řetězec*
    E-mailová adresa rodiče
  - **emailLower** – *textový řetězec*
    E-mailová adresa rodiče převedená na malá písmena

[6] `https://en.wikipedia.org/wiki/Unix_time`

- **token** – *textový řetězec*
  Krátký textový řetězec (*token*) sloužící k autorizaci uživatele v aplikaci
- **key** – *textový řetězec*
  Uživatelovo zahašové heslo
- **lang** – *textový řetězec*
  Textový řetězec reprezentující zvolený jazyk aplikace (cs/en)
- **kidIdList** – *seznam textových řetězců*
  Seznam _id identifikátorů uživatelovo dětí

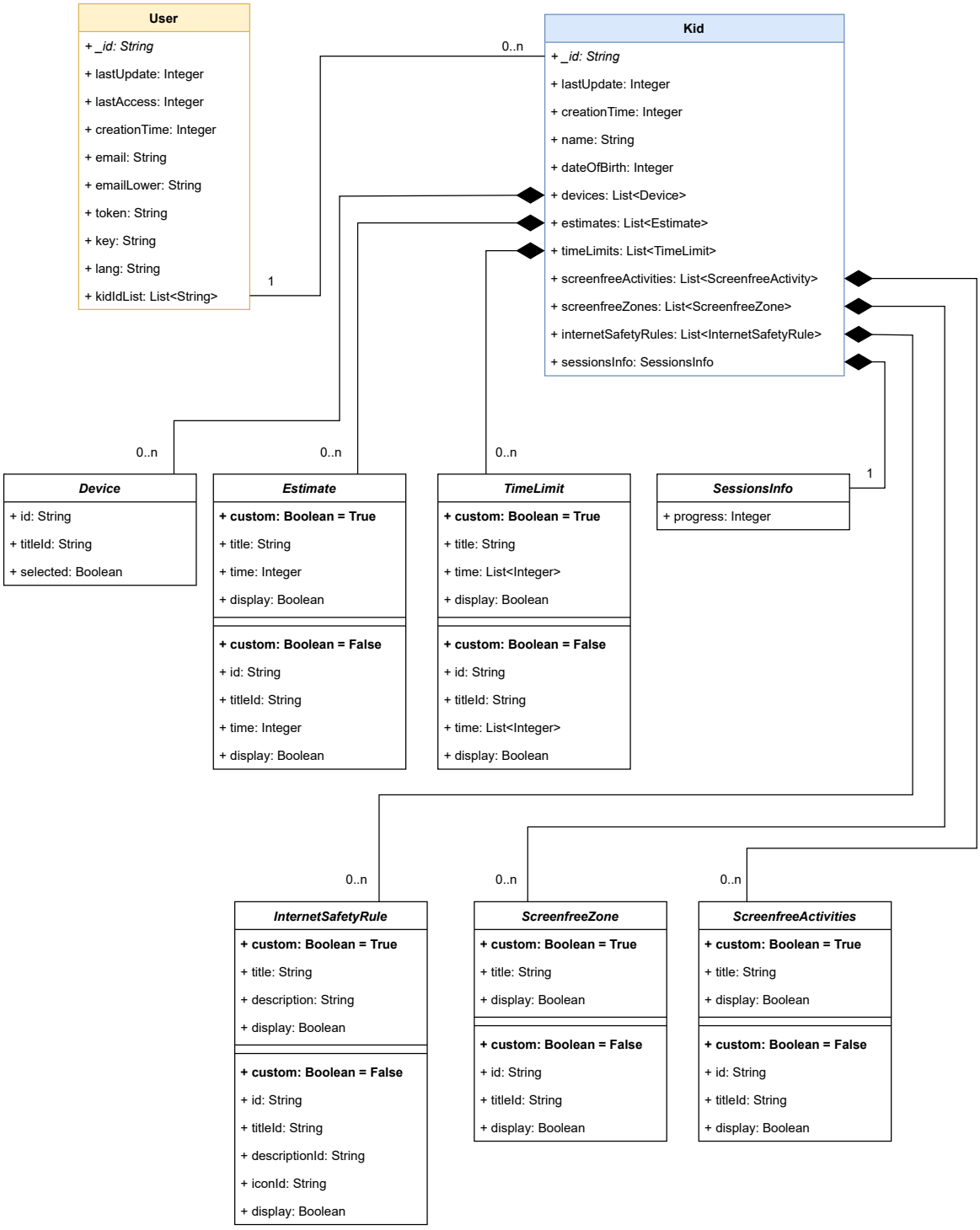
- **Kid:**
  - **_id** – *textový řetězec*
    Identifikátor Kid entity
  - **lastUpdate** – *číslo*
    Datum a čas poslední změny Kid entity (Unix time)
  - **creationTime** – *číslo*
    Datum a čas vytvoření Kid entity (Unix time)
  - **name** – *textový řetězec*
    Jméno dítěte
  - **dateOfBirth** – *číslo*
    Datum a čas narození dítěte (Unix time)
  - **devices** – *seznam typu Device*
    Seznam zařízení, které dítě používá.
  - **estimates** – *seznam typu Estimate*
    Seznam odhadů času stráveného provozováním různých aktivit.
  - **timeLimits** – *seznam typu TimeLimit*
    Seznam nastavených časových limitů pro provozování jednotlivých aktivit.
  - **screenFreeActivities** – *seznam typu ScreenFreeActivity*
    Seznam aktivit, pro které je zakázané používat digitální zařízení.
  - **screenfreeZones** – *seznam typu ScreenfreeZone*
    Seznam míst, kde je zakázané používat digitální zařízení.
  - **internetSafetyRules** – *seznam typu InternetSafetyRule*
    Seznam vybraných bezpečnostních pravidel při používání internetu.
  - **sessionsInfo** – *SessionsInfo*
    Datový typ reprezentující postup v aplikace (počet vykonaných sezení).

Uživatelské entity User jsou s Kid entitami spojeny *one-to-many* vazbou, která je realizovaná pomocí User atributu *kidIdList*. V tomto atributu se nachází seznam *_id* atributů různých Kid entit. Platí tedy, že uživatel (rodič) může skrze svoji User entitu odkazovat na 0 a více Kid entit. Na druhé straně Kid entita je pevně vázána k jedné konkrétní User entitě. Nemůže se tedy stát, že by v systému bylo *„dítě bez rodiče“*.

V celém systému pro určování konkrétního data a času používáme takzvaný *Unixový čas*. Tento přístup jsme zvolili pro svou jednoduchost a praktičnost. Pro většinu moderních programovacích jazyků existují knihovny, které tento údaj umí oboustranně převádět do svých datových typů reprezentující datum a čas.

User entity si u sebe drží 2 atributy pro e-mail. Atribut *email* je přesná textová reprezentace e-mailu, kterou uživatel zadal při registraci do systému. Na druhé straně atribut *emailLower* (uživatelův e-mail převedený na malá písmena) slouží k vyhledávání User entit podle emailu. Protokol e-mail jako takový totiž nerozlišuje velká a malá písmena.

Datový model aplikace Digital Parenting



Obrázek 6.1. Digital Parenting – datový model

Součástí datového modelu jsou i další nově vytvořené datové typy. Datový typ *Device* slouží pro uchování informací ohledně vybraných digitálních zařízení, které dítě používá. Device obsahuje následující atributy: *_id*, *titleId* a *selected*. Atribut *_id* slouží k identifikaci konkrétního typu zařízení, *titleId* slouží k přiřazení správného názvu používaného zařízení (požadavek od vývojářů klientských aplikací). Atribut *selected* je booleovská hodnota určující, zda dítě dané zařízení používá či ne.

Typ *SessionsInfo* v sobě obsahuje číselný atribut *progress*, pomocí kterého klientská aplikace pozná, ve kterém sezení se rodič u konkrétního dítěte nachází.

Zbývající datové typy (*Estimate*, *TimeLimit*, *InternetSafetyRule*, *ScreenfreeZone* a *ScreenfreeActivities*) jsou si svou strukturou velmi podobné a drží v sobě atributy pro uchování informací ohledně pravidel a limitů nastavených u konkrétního dítěte. Systém rozděluje tyto datové typy do 2 skupin: *výchozí* a *uživatelem vytvořené*. Rozlišit je lze podle booleovského atributu *custom*, který je nastavený na hodnotu *false* pro výchozí a *true* pro vytvořené uživatelem. Zmíněné datové typy obsahují také atribut *display*, podle kterého se pozná, zda si rodič dané pravidlo vybral (a tudíž se má v aplikaci zobrazit). Všechny výchozí datové typy mají společné atributy *_id* a *titleId*, které slouží k identifikaci a k přiřazení správného názvu. Na druhou stranu limity a pravidla vytvořené uživatelem obsahují místo toho atribut *title*, kde si své pravidlo mohou pojmenovat sami podle svého uvážení. V původním návrhu aplikace zaznělo, že by si uživatel mohl vytvářet svá vlastní pravidla. Tato funkce ovšem v klientských aplikacích prozatím implementována není.

V Android verzi aplikace Digital Parenting není implementováno dotazování uživatele na dítětem používaná zařízení. Také v aplikaci chybí odhady časů, které dítě tráví jednotlivými aktivitami. Z tohoto důvodu Android verze nevyužívá atributy *devices* a *estimates* v Kid entitě. Tyto nedostatky se ovšem plánují do budoucna odstranit.

6.3 Volba databázového systému

Velmi důležitou komponentou back-end řešení aplikace Digital Parenting je databázový systém, který se stará o data aplikace. V dnešní době existuje velmi mnoho databázových systému, které programátoři mohou ke svým projektům využít. Výběr databázového systému určuje, jakým způsobem budou data uvnitř databáze uchována a jakým způsobem k nim server může přistupovat.

6.3.1 Relační vs. NoSQL databáze

Databázové systémy lze rozdělit na 2 typy: *Relační databáze* a takzvané *NoSQL databáze*. Mezi těmito koncepty jsou značné rozdíly, ať už ve způsobu ukládání dat, rychlosti či přístupu k databázovým entitám.

Relační databáze jsou konvenčními typy databázových systémů založené na relačním modelu. Databáze jsou složené z tabulek, jejichž řádky reprezentují jednotlivé záznamy v databázi. V některých sloupcích databáze se mohou nacházet takzvané *cizí klíče*), které uchovávají informace o relacích mezi jednotlivými záznamy [18]. K manipulaci dat nad relačními databázemi se zpravidla používá jazyk *SQL*, případně některé jeho dialekty. Nevýhodou Relačních databází je nutnost vytvoření schématu, které přímo určuje strukturu uložených dat. Další nevýhodou je také omezená škálovatelnost. [19]

Jako NoSQL databáze lze označit jiné databázové systémy než jsou relační databáze. Existuje jich několik typů, přičemž každý typ NoSQL databáze má své přednosti v jiných aspektech a hodí se do jiných typů projektů. Velkou výhodou NoSQL databází

bývá dynamické databázové schéma, která umožňuje rychlejší adaptaci změn v datovém modelu. Některé NoSQL databáze mají také velmi dobře vyřešenou problematiku se škálováním a hodí se k operacím nad *velkými daty* (*big data*). [19]

Jedním z typů NoSQL databází jsou *dokumentově orientované databáze*. Tyto databázové systémy umožňují s daty pracovat v podobě dokumentů. Z pravidla se jedná o dokumenty ve formátu JSON nebo XML. Výhodou těchto databází je především jednoduchost co se týče manipulace s daty. S objekty v databázi se totiž pracuje jako s dokumenty. Dokumentově orientované databáze se používají v systémech, kde jsou objekty a entity příliš komplexní a není u nich možné dodržet pevnou strukturu (což je nutné u relačních databází).

V datovém modelu aplikace Digital Parenting se nachází relativně komplexní entita Kid, která se přímo skládá z více menších složených datových typů. Z tohoto důvodu se zdálo ideální pro back-end řešení projektu použít některou z dokumentově orientovaných databází.

6.3.2 MongoDB

Velmi oblíbenou dokumentově orientovanou databází je databáze *MongoDB*[7]. MongoDB k ukládání dat používá dokumentový formát *BSON*[8], což je lehce upravená verze populárního formátu JSON. MongoDB má dynamické databázové schéma. To programátorům umožňuje jednoduše a rychle ukládat data, případně měnit strukturu těchto dat. Tato vlastnost databáze se pro back-end řešení aplikace Digital Parenting velice hodila. Důvod je ten, že data v systému jsou relativně různorodá a datový model aplikace se často měnil. S databázovým systémem MongoDB jsem navíc již měl nějaké zkušenosti. Volba databázového systému pro back-end řešení tedy padla na MongoDB.

6.3.3 MongoDB Atlas

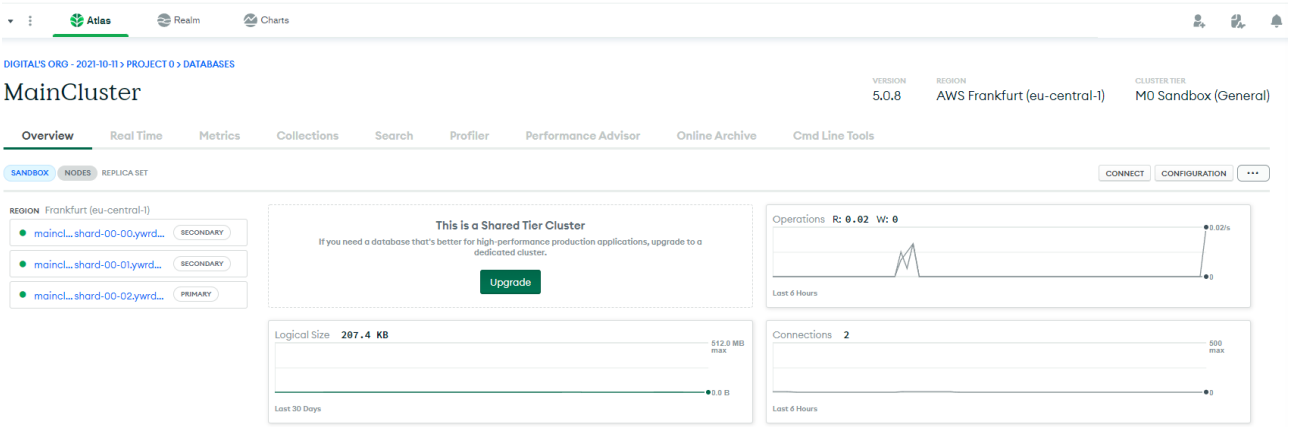
MongoDB nabízí registrovaným uživatelům přístup k multi–cloudové *DBaaS*[9] platformě *MongoDB Atlas*[10]. Ta umožňuje založení veřejně přístupných MongoDB databázových clusterů. Tyto databáze jsou hostované pomocí cloudových online služeb jako jsou *AWS*[11], *Google Cloud*[12] nebo *Azure*[13] [20]. Ve svém bezplatném režimu MongoDB Atlas nabízí úložiště o velikosti 512 MB. Pro správu dat v systému Digital Parenting takové úložiště prozatím stačí. Online hostování zdarma a dostatečná velikost úložiště byly hlavními důvody pro využití služby MongoDB Atlas.

Služba MongoDB Atlas má přehledné uživatelské rozhraní implementované ve webovém prohlížeči (obrázek 6.2). Kromě seznamu jednotlivých clusterů a databází jsou součástí uživatelského rozhraní také užitečné statistiky ohledně využití sdílené paměti či statistiky ohledně přístupu aplikací a serverů k danému clusteru.

6.4 Implementace serveru

Z důvodů zmíněných v sekci 6.1 probíhala implementace řešení v programovacích jazyku Python za použití frameworku Flask. Projekt jsem si rozdělil na několik částí (logické komponenty programu), které se navzájem volají a tvoří tak funkcionalitu celého

[7] `https://www.mongodb.com/`
[8] `https://www.mongodb.com/basics/bson`
[9] `https://www.forpsicloud.cz/database-as-a-service.aspx`
[10] `https://www.mongodb.com/cloud-database/benefits`
[11] `https://aws.amazon.com/`
[12] `https://cloud.google.com/`
[13] `https://azure.microsoft.com/`



Obrázek 6.2. MongoDB Atlas – uživatelské rozhraní

back-end řešení. Python kód řešení jsem rozdělil do celkem 4 *Python modulů* (*Python package*). Každá logická komponenta programu odpovídá jednomu Python modulu. Diagram zobrazující jednotlivé komponenty i jejich vzájemnou komunikaci lze vidět na přiloženém obrázku (obrázek 6.3).

Hlavní částí diagramu komponent je kontejner označený jako *Digital Parenting Server*. Uvnitř lze vidět 4 logické komponenty: *API*, *Services*, *Model* a *Utils*.

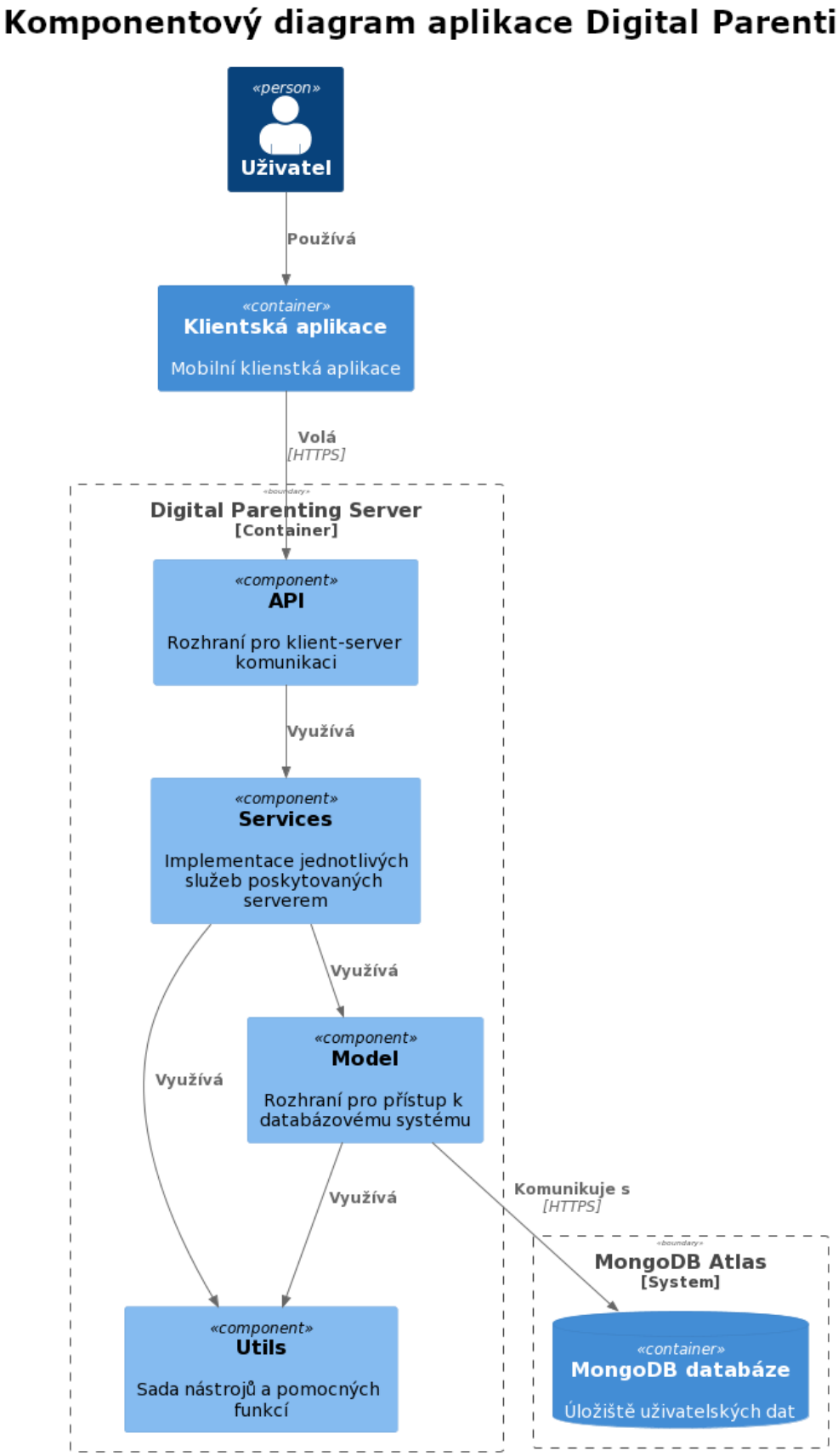
Komponenta API se stará o komunikaci s klientskými aplikacemi. Vystavuje veřejné REST API rozhraní a je implementovaná zejména pomocí frameworku Flask. Komponenta API ke své funkci využívá komponentu Services. V té se nacházejí implementace všech služeb, které serverové řešení nabízí. Kromě provádění jednotlivých klientských požadavků je komponenta Services zodpovědná i za validaci uživatelských dat. Komponenta kontroluje, zda se jedná o správné datové formáty a také zda jsou data z klientských aplikací validní. V případě že v programu dojde k chybě vlivem špatně formulovaného požadavku nebo kvůli požadavku s chybnými daty, je komponenta Services zodpovědná za řádné zpracování dané chyby a vygenerování příslušné chybové hlášky. Komponenta Services využívá komponentu Model. Ta se stará o manipulaci s uživatelskými daty v databázi. Je přímo napojena k MongoDB databázi, která je hostovaná službou MongoDB Atlas. Komponenty Services a Model ke svým činnostem využívají také komponentu Utils. Ta je jakousi sadou nástrojů a pomocných funkcí, které se využívají například při manipulaci s daty.

Pro ucelený pohled na celý systém Digital Parenting, je tento komponentový diagram navíc rozšířený o entity *Uživatel* a *Klientská aplikace*. Uživatel používá klientskou aplikaci, která skrze protokol HTTPS komunikuje s REST API rozhraním vystaveným na serveru.

6.4.1 Komunikace s klientskými aplikacemi

Klientské aplikace se snaží o to, aby byla jejich interní uživatelská data synchronizovaná s back-end řešením. Data zároveň musí být aktuální. Pokud uživatel používá 2 telefony, pak se změny provedené na jednom telefonu musí projevit i na telefonu druhém. Toto je v systému zajištěno pomocí atributu *lastUpdate*, který nalezneme uvnitř datových typů User a Kid. Tento atribut se na serveru vždy nastaví na aktuální čas, při kterém ke změně dat došlo. Klientské aplikace tedy vždy poznají, zda u sebe mají aktuální data. V případě že ne, mohou si data ze serveru stáhnout a tím je u sebe aktualizovat.

Aktualizace dat serveru probíhá inkrementálně, vždy po dokončení jednoho sezení. Tento model se vybral z toho důvodu, že uživatel zpravidla dokončí celé sezení a až



Obrázek 6.3. Digital Parenting – komponentový diagram

poté aplikaci ukončí . Aktualizace dat také probíhá při změně pravidel v komponentě Pravidla či při změnách provedených v komponentě Profil.

6.4.2 Struktura REST API

Server aplikace Digital Parenting musel být navržený tak, aby s ním mohlo komunikovat více různých typů klientských aplikací. Pro komunikaci mezi klienty a serverem se používá REST API rozhraní, které obsahuje několik připravených endpointů. Komunikace dat probíhá za použití formátu JSON.

Z důvodů usnadnění vývoje jsem se rozhodl pro následující formát výměny dat. Odpovědi serveru se dělí do dvou skupin: *validní odpovědi* a *chybné odpovědi*. Pokud je odpověď serveru validní, má následující formát:

```
{
    "status": 0,
    "target": <DATA>
}
```

Odpovědí serveru je JSON objekt, který obsahuje číselný atribut *status*, jehož hodnota je vždy nastavena na 0. Objekt dále obsahuje atribut *target*, ve kterém se nachází dotazovaná data, uložená jako JSON objekt. Formát a obsah tohoto JSON objektu může být odlišný. Vždy záleží na konkrétním endpointu rozhraní.

Chybné odpovědi serveru se drží tohoto formátu:

```
{
    "status": <STATUS_CODE>,
    "message": <ERROR_MESSAGE>
}
```

Odpovědí je opět JSON objekt. Chybná odpověď serveru obsahuje dva atributy. Tím prvním je opět číselný atribut *status*. Jeho hodnota nyní identifikuje konkrétní chybu, ke které v rámci zpracování požadavku došlo. Hodnota atributu *status* je vždy různá od 0. Dále se v objektu nachází atribut *message*, který reprezentuje konkrétní chybovou hlášku. Datový typ tohoto atribut je vždy textový řetězec. Atribut *message* umožňuje vývojářům snazší identifikaci chyb při implementaci klient–server komunikace. Atribut ovšem neslouží k přímému sdělování chyb v klientských aplikacích. V klientských aplikacích je tedy nutné implementovat správné zobrazování chybových hlášek na základě číselného kódu chyby v odpovědi serveru.

Jako příklad chybné odpovědi serveru přikládám následující JSON objekt, který byl serverem vrácen při chybném pokusu o registraci.

```
{
    "message": "E-mail is in wrong format",
    "status": 2
}
```

Programátor, který implementuje klientskou aplikaci nyní ví, že pokud při registraci uživatele bude v odpovědi serveru hodnota atributu *status* 2, pak musel uživatel zadat neplatný e-mail. Pokud tedy taková situace při reálném používání aplikaci nastane, pak aplikace uživatele upozorní na chybně zadanou e-mailovou adresu.

6.4.3 Zajištění bezpečnosti

Jelikož se v aplikaci pracuje s citlivými daty, musel se pro back-end řešení navrhnout dostatečně bezpečný model. Bezpečností dat se v řešení zabývám ve 2 případech.

Prvním případem je ukládání hesel. Hesla v databázi nesmí být uložena jako prostý text (viz kapitola 5, sekce 5.4.2). K vyřešení tohoto problému jsem použil vestavěný Python modul *hashlib*, který nabízí různé kryptografické hašovací funkce. Pro hašování hesel v databázi jsem zvolil algoritmus *SHA-256*[14]. Pro hašování hesel používám 100000 iterací tohoto algoritmu. Součástí hašovacího procesu je i generování takzvané *soli*. Jako sůl se při hašování označují náhodná data, která se k heslu před hašováním přidají. Tato sůl, která z pravidla bývá krátký textový řetězec, se poté přiřadí k výslednému výstupu hašovací funkce do databázového úložiště. Tímto se v případě uniku dat zajistí ochrana v případě použití duplicitních hesel – stejná hesla mají totiž díky náhodně vygenerované soli rozdílné výstupy hašovací funkce. [21]

Druhá oblast, ve které řeším zabezpečení dat, je síťová komunikace. Aby se mohl uživatel skrze svou klientskou aplikaci dostat ke svým datům, musí nejprve prokázat, že se jedná o správného uživatele (*autentizace*) a že má k dané operaci dostatečná práva (*autorizace*). Jedním z možných přístupů jak v REST API aplikaci problém s autorizací vyřešit, je posílání uživatelského hesla spolu s e-mailem v každém požadavku na server. Komunikace serveru by sice měla probíhat pomocí zabezpečeného komunikační protokolu HTTPS, ovšem i tak je tento přístup vnímán jako problematický. Z tohoto důvodu jsem se rozhodl pro použití *tokenů*, které nahradí nutnost posílání e-mailu a hesla.

Token je 32 znaků dlouhý náhodný řetězec, který generuje server a který se používá k autentizaci a autorizaci uživatele. Tokeny nemají expirační dobu a lze je používat do té doby, než se server požádá o vytvoření a přiřazení nového tokenu. Tento přístup má další užitečnou vlastnost a tou je udržení relace na uživatelském zařízení bez nutnosti ukládání uživatelského hesla.

6.5 Hosting serveru

Nedílnou částí projektu bylo také nasazení a zpřístupnění řešení pro okamžité používání a testování. Bylo nutné vybrat vhodnou hostovací službu, která by splňovala následující seznam požadavků:

- **Podpora Flask aplikací**
- **Dostupnost** – ideálně služba běžící 24 hodin denně
- **Přijatelná cena**

6.5.1 Heroku

Jedním z řešením bylo použití cloudové *Platform as a Service* (*PaaS*)[15] služby *Heroku*. Heroku byla vyvinuta již v roce 2007 a jedná se o jednu z prvním cloudových služeb na světě. Heroku umožňuje nasazování různých typů aplikací napsaných v programovacích jazycích Ruby, Java, Node.js, Scala, Clojure, Python, PHP nebo Go. [22]

Heroku nabízí několik možných variant hostování aplikace. Je mezi nimi dokonce i bezplatná varianta, která se pro projekt Digital Parenting zdála dostačující. Nevýhodou této varianty je relativně omezený výpočetní čas procesoru, který aplikace může použít. Pro náš projekt toto ovšem příliš nevadí, jelikož back-end řešení ke své práci

[14] `https://www.n-able.com/blog/sha-256-encryption`
[15] `https://www.rascasone.com/cs/blog/co-je-paas-vyuziti-vyhody`

nepotřebuje počítat žádné složité operace a tudíž se příliš procesorového času nespotřebuje. V případě, že by se počet uživatelů příliš zvýšil a tento bezplatný hosting by nestačil, muselo by se buď přejít na některý z placených režimů, nebo by se musel zvážit přechod na jinou formu hostování aplikace.

Kapitola 7
Dokumentace projektu

Součástí většiny projektů by měla být i adekvátní dokumentace řešení. Vytváření takové dokumentace má 2 zásadní důvody. Prvním je efektivnější komunikace programátorů. Pokud má programátor po někom převzít projekt, či na projektu začít spolupracovat, je nahlédnutí do dokumentace jedním z prvních kroků, které programátor učiní. Tento typ dokumentace se označuje jako *technická dokumentace*. Jsou jimi například komentáře v kódu, nebo automaticky vygenerované dokumenty z kódu řešení.

Druhým typem dokumentace řešení je *uživatelská dokumentace*. Tato forma dokumentace je psána pro uživatele daného softwaru. Uživateli mohou být dokonce i programátoři, kteří se například snaží o zprovoznění určité knihovny. Dokumentace jim poté může posloužit jako jakýsi manuál k dané knihovně. Dokumentace by měla obsahovat popis všech důležitých funkcí, které může uživatel používat. [23]

Dokumentaci back-end řešení aplikace jsem při práci nemohl podcenit. Programátoři implementující klient–server funkcionalitu klientských aplikací museli být alespoň částečně obeznámeni s tím, jak funguje mé back-end řešení a jaké jeho funkce mohou používat.

7.1 REST API dokumentace

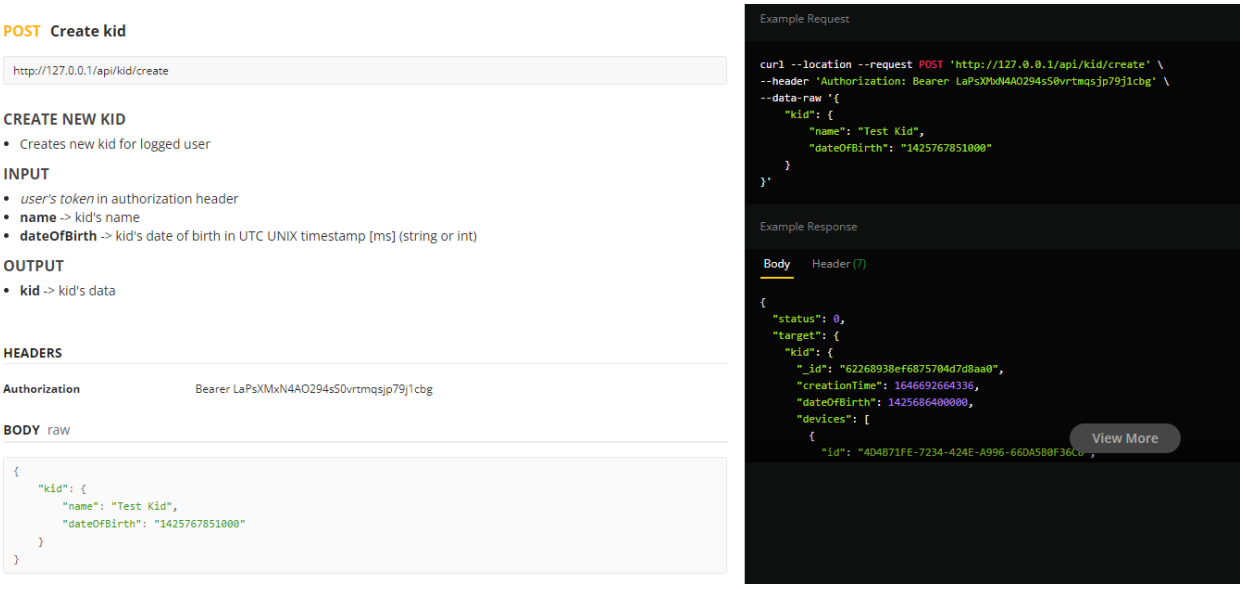
Nejdůležitější informací, kterou programátoři klientských aplikací pro připojení k serveru potřebují, je seznam jednotlivých přístupových bodů (endpointů) vystaveného REST API rozhraní. Pro dokumentaci endpointů jsem použil populární nástroj *Postman*[1]. Postman se používá zejména k návrhu a testování webových API. Kromě toho se dá v tomto nástroji vygenerovat i jednoduchá, přehledná dokumentace, která umožňuje vytvoření okomentovaného seznamu přístupových bodů. K jednotlivým endpointům lze také přiřadit i příklad jeho použití. Těmito příklady se poté může programátor implementující daný endpoint inspirovat, což mu často ulehčí práci s pochopením dokumentace.

V době psaní této práce je v Postman dokumentaci zdokumentováno celkem 13 endpointů. Endpointy v dokumentaci mají následující formát:

- HTTP metoda – GET, POST nebo DELETE
- URI – identifikátor endpointu
- Popis endpointu
- Parametry endpointu
- Příklad volání daného endpointu

Pro lepší představu o zdokumentovaných endpointech přikládám výstřižek obrazovky, na kterém je zachycena dokumentace k přístupovému bodu s názvem *Create kid* 7.1.

[1] `https://www.postman.com/`

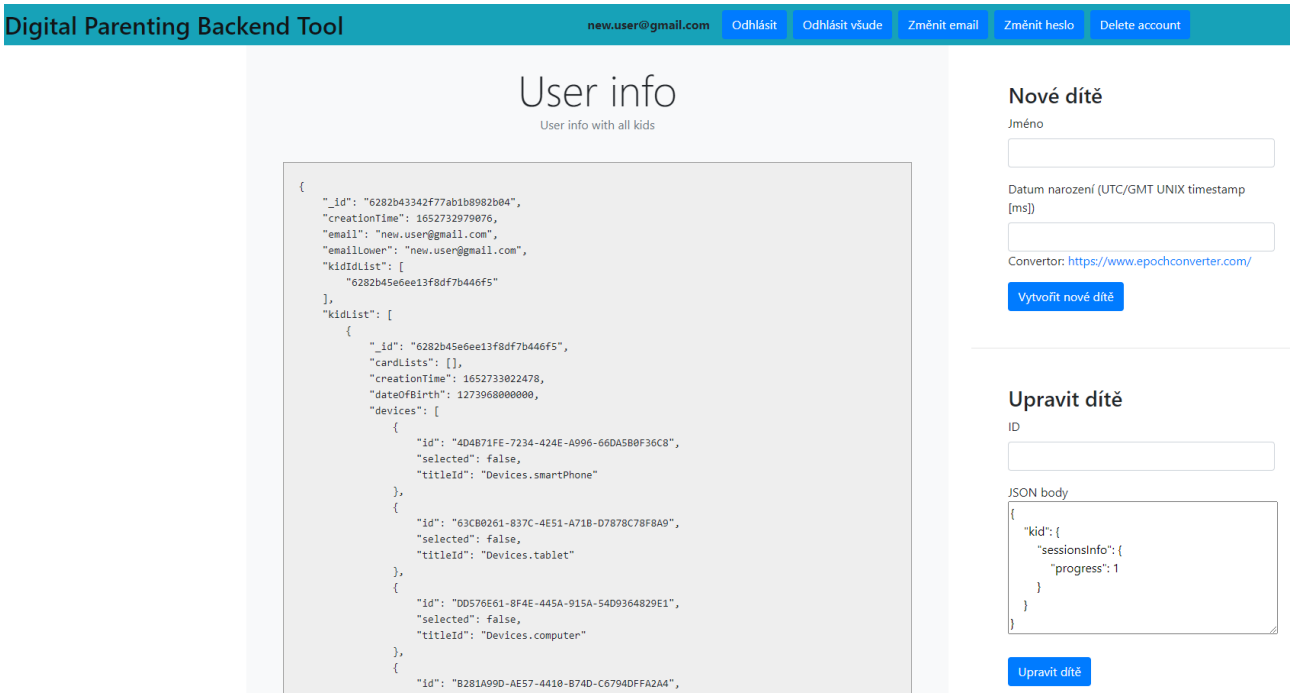


Obrázek 7.1. Dokumentace endpointu *Create kid*

7.2 Digital Parenting Backend Tool

Paralelně s Postman dokumentací vznikl také nástroj *Digital Parenting Backend Tool* (dále jen Backend Tool). Tento nástroj má formát webové aplikace a slouží pouze k testování REST API rozhraní back-end serveru. Programátoři klientských aplikací si tedy mohli před implementací endpointů komunikaci nejdříve vyzkoušet pomocí tohoto nástroje.

Backend Tool nabízí stejné funkcionality jako mají klientské aplikace. Má jednoduché, uživatelsky ovšem velmi nepřívětivé uživatelské rozhraní. Jakákoli operace se v nástroji provádí pomocí několik předem připravených funkcí, určených k manipulaci s uživatelskými daty. Funkce jako svůj vstup přijímají textové řetězce, které přímo reprezentují obsahy jednotlivých JSON objektů, které se následně odesílají na server k dalšímu zpracování. Na obrázku 7.2 lze vidět uživatelské rozhraní aplikace Backend Tool při manipulaci dat uživatele s e-mailem *new.user@gmail.com*. Uprostřed výřezu obrazovky se nachází *User info*. User info je souhrnný přehled o veškerých datech spojených s konkrétním uživatelem v systému. Tato uživatelská data obsahují také informace o všech dětech uživatele. Obsah panelu User info je naformátovaný JSON dokument.



Obrázek 7.2. Digital Parenting Backend Tool

Kapitola 8
Digital Parenting Dashboard

Back-end řešení systému Digital Parenting sbírá mnoho užitečných dat od uživatelů. Aby se ovšem z dat mohly usuzovat nějaké závěry, musí se data nejprve vizualizovat. Z důvodu vizualizace dat vznikla kromě back-end řešení také jednoduchá webová aplikace *Digital Parenting Dashboard* (dále jen Dashboard), která je nyní nedílnou částí celého systému.

Dashboard slouží správcům a specializovaným odborníkům pro rychlou analýzu uživatelských dat uvnitř systému. Dashboard zpracuje data ze systému Digital Parenting a vytvoří z toho agregované výsledky, které si pak může uživatel Dashboardu zobrazit odkudkoli chce. Vizualizace jednotlivých grafů je v aplikaci zajištěna pomocí JavaScriptové knihovny *Chart.js*[1].

Dashboard je rozdělený na 2 části. *User Dashboard* a *Kid Dashboard*. Přepínat mezi nimi lze v navigačním panelu, který se v aplikaci nachází vlevo.

8.1 User Dashboard

User Dashboard je velmi stručný. Nachází se v něm nejdůležitější agregovaná data spjatá s User entitami v systému. V User Dashboardu se nachází následující seznam informací:

- **Celkový počet uživatelů**
- **Počet nových uživatelů za poslední měsíc**
- **Počet nových uživatelů za posledních 12 měsíců**
- **Graf přírůstku nových uživatelů za posledních 12 měsíců**
- **Graf počtu dětí na 1 uživatele**
- **Průměrný počet dětí na 1 uživatele**

Výstřižek obrazovky User Dashboardu lze vidět na přiloženém obrázku 8.1

8.2 Kid Dashboard

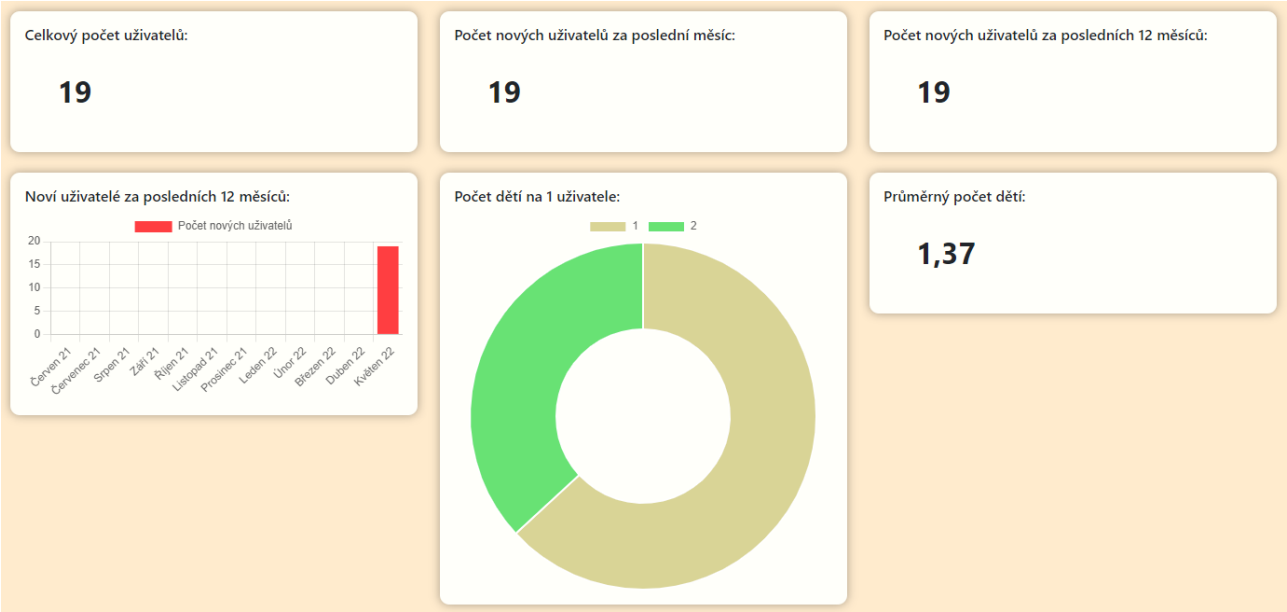
Kid Dashboard obsahuje v porovnání s User Dashboardem informací více. V Kid Dashboardu se nachází všechny agregovaná data týkající se Kid entit systému Digital Parenting. Uživatel Dashboardu se v něm dozví následující seznam statistik a přehledů:

- **Celkový počet dětí**
- **Počet nových dětí za poslední měsíc**
- **Počet nových dětí za posledních 12 měsíců**
- **Histogram věku dětí**
- **Graf používaných zařízení**
- **Volba Internet Safety Rules**
- **Volba Screenfree Zones**

[1] `https://www.chartjs.org/`

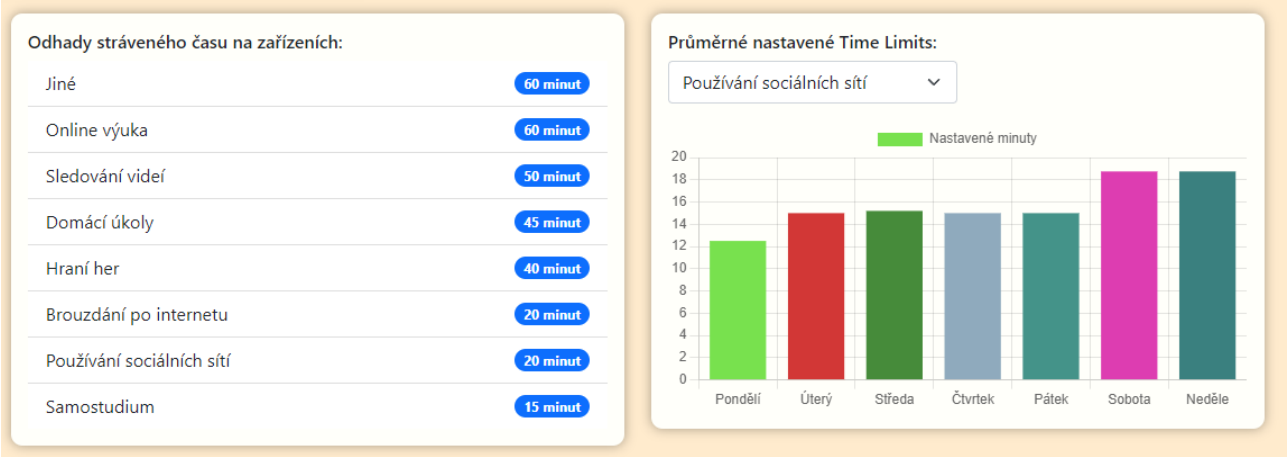
- **Volba Screenfree Activities**
- **Odhady stráveného času na zařízeních**
- **Graf průměrných nastavených Time Limits** (pro konkrétní činnost)

Grafické zpracování Kid Dashboardu lze vidět na přiložených obrázcích 8.2 a 8.3.



Obrázek 8.1. Digital Parenting Dashboard – User Dashboard

Obrázek 8.2. Digital Parenting Dashboard – Kid Dashboard (část 1)



Obrázek 8.3. Digital Parenting Dashboard – Kid Dashboard (část 2)

8.3 Vyhledávání uživatele

Vizualizace agregovaných dat sice může posloužit jako jakýsi stručný přehled o uživatelských datech, ovšem v případě nutnosti kontroly konkrétního uživatele aplikace Digital Parenting, je tato forma prezentace dat poněkud nedostačující.

Z tohoto důvodu je nutné do Dashboardu zakomponovat také vyhledávač konkrétního uživatele. Ten by umožňoval expertovi (např. pracovníkovi zdravotnické organizace) vyhledat konkrétního rodiče a na základě jeho dat by mohl poskytovat další doporučení.

Vyhledávání by probíhalo pomocí uživatelovy e-mailové adresy, skrze kterou je jeho účet propojený s jeho uživatelskými daty. Následná vizualizace by probíhala podobným způsobem jako je tomu v User a Kid Dashboardu.

Požadavek na tuto funkcionalitu byl ovšem zformulován až ve velmi pozdní fázi vývoje systému a momentálně tato funkcionalita ještě není součástí Digital Parenting Dashboardu.

Kapitola 9
Uživatelské testování aplikace

Součástí této práce je také uživatelské testování aplikace. Abychom poznali, zda se návrh a vývoj aplikace posouvá tím správným směrem, museli jsme náš systém otestovat na reálných uživatelích již v relativně brzké fázi jeho vývoje.

K testování jsme potřebovali rodiče s dětmi ve věku ideálně do 15 let, kteří by byli ochotni si naši aplikaci pro účely testování vyzkoušet.

Testovací model byl následující. Rodiče, kteří o testování aplikace projevili zájem, od nás obdrželi informační e-mail, ve kterém se nacházely instrukce k instalaci aplikace. Instrukce k instalaci byly vždy zasílány pouze pro konkrétní operační systém daného uživatele. Dále se v e-mailu nacházely pokyny k testování aplikace. Poslední část informačního e-mailu měla formát prosby o vyplnění *Google Forms*[1] dotazníku.

9.1 Pokyny k testování

Pokyny k testování aplikace jsou důležité, protože uživatelé, *testeři*, musí přesně vědět, co se od nich očekává. Pokyny seznámí testery s hlavním cílem testování a vysvětlí jim, co se od nich při průchodu aplikací očekává. Součástí pokynů je také zmínka o termínu, do kdy je testování nutné ukonči. Celé znění testovacích pokynů je následující:

„Hlavním cílem testování je ověřit funkčnost aplikace. Vedlejším cílem je prevence negativních vlivů digitálních technologií u dětí za pomoci rodiče. Aplikace vás provede celkem 5 interaktivními edukačními prezentacemi, které nazýváme sezení. V rámci testování bychom vás rádi požádali o dokončení všech 5 zmíněných sezení (samozřejmě oceníme pokud budete testovat i další funkcionality aplikace). Aplikaci používejte ideálně ve spolupráci s vaším dítětem/dětmi.

Testování bude probíhat od 07.05.2022 až do 11.05.2022. Aplikaci je možné používat i po skončení testování.“

9.2 Testovací dotazník

Po dokončení testování aplikaci byli testeři vyzváni k vyplnění krátkého online dotazníku. Cílem dotazníku bylo zjištění celkové spokojenosti uživatelů s aplikací. Skrze dotazník jsme se snažili především o zjištění pokud možno co nejvíce nedostatků a chyb aplikace, které bychom v průběhu dalšího vývoje aplikace mohli opravit.

Dotazník se skládá z celkem 9 uzavřených a 4 otevřených otázek. Součástí otazníku jsou také 2 doplňující otázky: dotaz na e-mailovou adresu testera a volba operačního systému zařízení, na kterém byla aplikace testována.

[1] `https://www.google.com/forms/about/`

9.2.1 Uzavřené otázky

Uzavřené otázky jsou formulovány jako jednoduché výroky, u kterých uživatel vyjadřuje míru souhlasu pomocí 5-stupňové Likertovy škály. Odpovědi škály vypadají takto:

- Souhlasím
- Spíše souhlasím
- Ani souhlasím, ani nesouhlasím
- Spíše nesouhlasím
- Nesouhlasím

Seznam jednotlivých otázek i s jejich vysvětlením je následující:

1. **Instalace aplikace proběhla bez problémů.**
   Zde se ptáme na to, zda se uživatelům podařilo nainstalovat a spustit aplikaci bez problému. Klientské aplikace v době psaní této práce zatím nejsou dostupné na *Google Play* a *App Store*. Z tohoto důvodu jsme chtěli zjistit, zda spuštění poskytnuté aplikace proběhlo u všech respondentů v pořádku.

2. **První přihlášení do aplikace a vytvoření nového dítěte bylo snadné.**
   Zjišťovali jsme, zda spuštění aplikace a následný onboarding proběhly bez problémů.

3. **Používání aplikace bylo jednoduché a intuitivní.**
   Zde nás zajímalo, jestli uživatelům přijde aplikace jednoduchá a intuitivní k použití. Kvalitní *user experience* je základem každé aplikace.

4. **Aplikace fungovala bez problémů.**
   Ptáme se, zda v aplikace nedocházelo k chybám.

5. **Tipy a rady z jednotlivých sezení shledávám jako užitečnou pomůcku pro vytváření pravidel se svým dítětem.**
   Zajímalo nás, zda uživatelům přijdou informace, rady a poučení z aplikace užitečné.

6. **Pokud bych chtěl/a poučit své dítě o škodlivosti digitálních technologií, použil/a bych tuto aplikaci.**
   Tato otázka byla formulována poněkud nešťastně. Cílem aplikace totiž není *poučit dítě*, ale *poučit rodiče*. Uživatele tento logický omyl pravděpodobně pochopili a na otázku odpovídali tak, jak bylo původně zamýšleno. Otázkou jsme chtěli zjistit, zda si rodiče dokáží představit běžné používání naší aplikace.

7. **Funkce přidání více dětí do aplikace je užitečná.**
   V této otázce jsme chtěli zjistit, jakým způsobem uživatelé vnímají možnost přidání více svých dětí do aplikace a zda jim tato funkce přijde užitečná. V případě negativní nebo převážně negativní zpětné vazby bychom museli pozměnit směr vývoje aplikace a tuto funkci upravit, či z důvodů nevyužití úplně odstranit.

8. **Uživatelský účet a možnost přihlášení z jiného zařízení shledávám jako užitečné.**
   Zjišťujeme, zda mělo vytvoření back-end řešení pro uživatele aplikace smysl.

9. **Aplikaci bych doporučil/a svým známým.**
   Zajímá nás, zda by testeři doporučili aplikaci jiným rodičům.

9.2.2 Otevřené otázky

Otevřené otázky dávají respondentům více prostoru pro vyjádření svých odpovědí. Otevřené otázky ve formuláři se zaměřují zejména na celkový dojem z používání aplikace. Součástí formuláře jsou tyto 4 otázky:

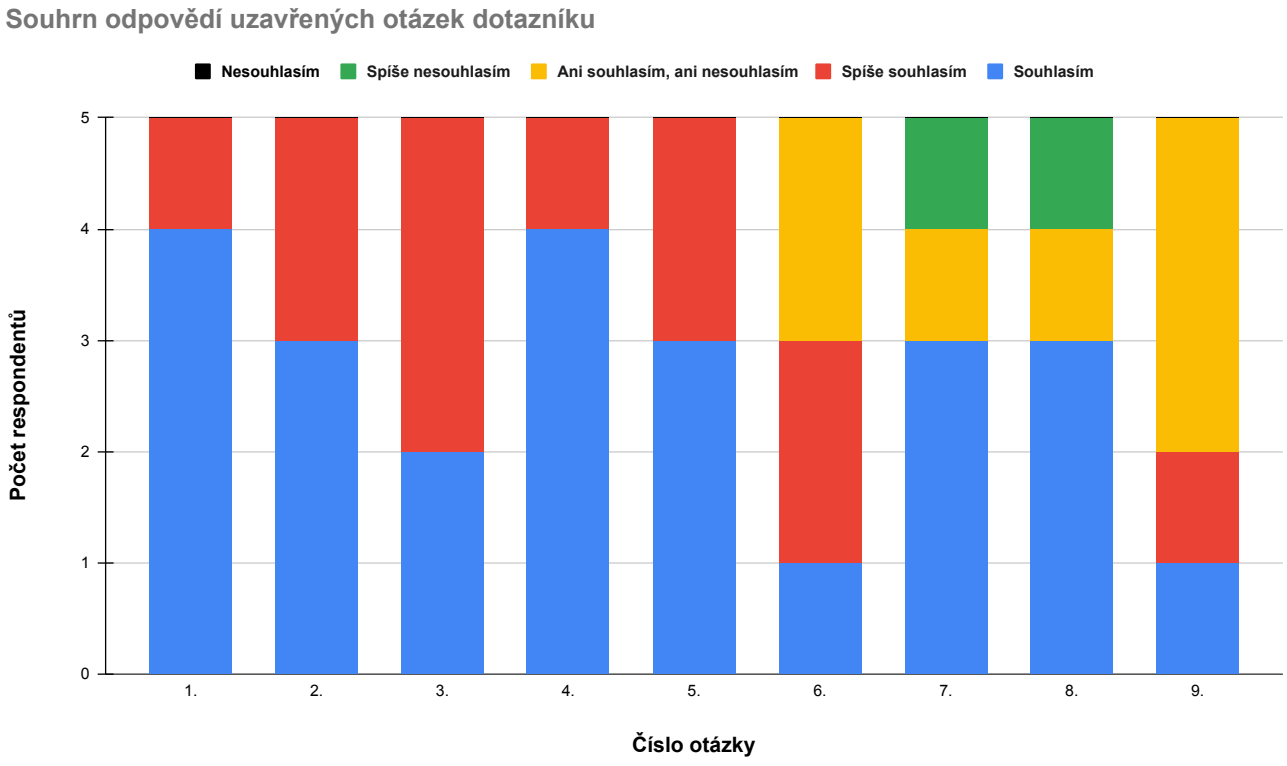
1. Co se Vám na aplikaci líbilo nejvíce?
2. Co se Vám naopak na aplikaci nelíbilo?
3. Co byste na aplikaci do budoucna vylepšil/a?
4. Co Vám v aplikaci chybělo? Co bylo naopak zbytečné?

9.3 Vyhodnocení odpovědí

Celkem jsme ve formuláři dostali odpovědi od 5 respondentů. Vzhledem k tomu, že počet respondentů nebyl příliš veliký, nemělo smysl pro vyhodnocování odpovědí používat komplexní vyhodnocovací nástroje statické analýzy. Výsledky testovacího formuláře jsem tedy zpracoval ručně.

9.3.1 Uzavřené otázky

Hodnocení aplikace v uzavřených otázkách bylo převážně pozitivní. Většina respondentů s výroky *souhlasila*, nebo *spíše souhlasila*. Graf na obrázku 9.1 zachycuje odpovědi respondentů pro všech 9 uzavřených otázek. Jedná se o složený sloupcový graf, ze kterého lze lehce zjistit, jak se na jednotlivé otázky odpovídalo, případně v jakém poměru jednotlivé odpovědi byly.



Obrázek 9.1. Souhrn odpovědí uzavřených otázek dotazníku

Žádná otázka neměla ve svých odpovědích od respondentů odpověď *Nesouhlasím*. Na druhou stranu, otázky 7 a 8 měly mezi svými odpovědmi vždy jednu *Spíše nesouhlasím* odpověď. Konkrétně se jedná o tyto otázky:

- Funkce přidání více dětí do aplikace je užitečná.
- Uživatelský účet a možnost přihlášení z jiného zařízení shledávám jako užitečné.

Z odpovědí respondentů tedy plyne, že ne všechny funkcionality aplikace byly přijaty kladně. Funkce přidávání více dětí ke svému uživatelskému účtu je samozřejmě zbytečná pro rodiče, kteří mají pouze jedno dítě. Ovšem i rodiče, kteří mají dětí více, tuto funkcionalitu mohou chápat jako přebytečnou. Při založení nového dítěte musí rodič projít těmi stejnými sezeními jako u prvního dítěte. To může uživatelům přijít zdlouhavé, nebo dokonce zbytečné. Možná by tedy stálo za promyšlení, jak tuto funkcionalitu upravit tak, aby uživatelům dávala větší smysl.

Přihlašování ke svému uživatelskému účtu z jiného zařízení je pravděpodobně funkcionalita, kterou testeři při testování aplikace nevyužili. Tato funkcionalita se hodí v případě, kdy bude uživatel aplikace přecházet na nový telefon a bude chtít, aby si na svém novém telefonu uchoval dříve zadaná pravidla a časové limity. Může se možná zdát, že tato funkcionalita nebude často využívána, ovšem myslím si, že by zachována být měla. Kromě toho by bez uživatelských účtů nebylo možné analyzovat data konkrétního uživatele, čímž by se celý systém připravil o velmi cennou funkci.

Kromě těchto 2 výše zmíněných otázek byly v dotazníku hůře hodnocené otázky číslo 6 a 9. Toto by se dalo vysvětlit tím, že v aplikaci zřejmě ještě není dostatek kvalitního edukačního materiálu a je nutné aplikaci do budoucna rozšířit o další poučné informace.

9.3.2 Otevřené otázky

Odpovědi v otevřených otázkách nelze shrnout žádným číselným vyjádřením a tedy všechny odpovědi respondentů bylo nutné projít manuálně.

- **Co se Vám na aplikaci líbilo nejvíce?**

  Většina respondentů oceňovala design, někteří zmiňovali i jednoduchost aplikace. Víme, tedy, že design i formát aplikace byly zvoleny správně.

- **Co se Vám naopak na aplikaci nelíbilo?**

  Testeři si stěžovali na chybějící překlady některých tlačítek a také na určité menší grafické chyby (*„Některé otázky nebyly přes postupové šipky či kvůli malému rámečku úplně vidět.“*).

  Jeden z respondentů si také stěžoval na to, že moc nepochopil smysl některých funkcionalit aplikace, jako je třeba vytváření dítěte nebo vytváření pravidel. Toto mohlo vzniknout tím, že respondent správně nepochopil účel naší aplikace. Bylo by tedy vhodné zapracovat na vysvětlení, aby k takovýmto nedorozumění nedocházelo.

- **Co byste na aplikaci do budoucna vylepšil/a?**

  V odpovědích byly zmíněné gramatické chyby, špatná formulace některých textů, či chyby v grafickém rozložení. Na všech těchto nedostatcích by se v příštích verzích aplikace mělo zapracovat.

  V jedné odpovědi se jeden z respondentů zmiňoval o tom, že nejspíše správně nepochopil smysl aplikace. V odpovědi se ptá, proč se vlastně jednotlivá pravidla nastavují a jakým způsobem by měla probíhat kontrola těchto pravidel. Dle jeho odpovědi respondent nejspíše očekával, že dodržování pravidel bude kontrolováno nějakou aplikací na zařízení, které používá jeho dítě. Jak už jsem psal výše, toto nedorozumění nejspíše vzniklo nedostatečným vysvětlením funkcionalit a smyslu aplikace.

- **Co Vám v aplikaci chybělo? Co bylo naopak zbytečné?**
  Odpovědi vesměs zmiňují jednu konkrétní věc a tou je nedostatečné vysvětlení jednotlivých částí aplikace. Respondenti by tedy nejspíše očekávali, že jim aplikace nejdříve vysvětlí k čemu je, jak se s ní má zacházet a také (dle jedné z odpovědí) jakým způsobem by by měl rodič se svým dítětem pracovat při nastavování limitů a pravidel.

Odpovědi z otevřených otázek přinesly velmi cenné podněty, které jistě poslouží ke zlepšení aplikace. Opravení gramatických chyb, chybějících překladů, či chyb v grafickém rozložení aplikace je samozřejmostí. Mnohem zajímavějšími podněty jsou ovšem připomínky ohledně nedostatečného, či chybějícího obsahu, který by uživateli lépe vysvětlil smysl či správné použití naší aplikace.

Podněty z odpovědí jsem rozdělil do několika bodů, které by v budoucnu mohly posloužit jako možný směr vývoje aplikace:

- Uvedení aplikace do kontextu digitálního rodičovství
- Seznámení uživatele s aplikací
- Lepší vysvětlení smyslu nastavování pravidel a limitů
- Vysvětlit rodiči jak o tématu mluvit se svým dítětem

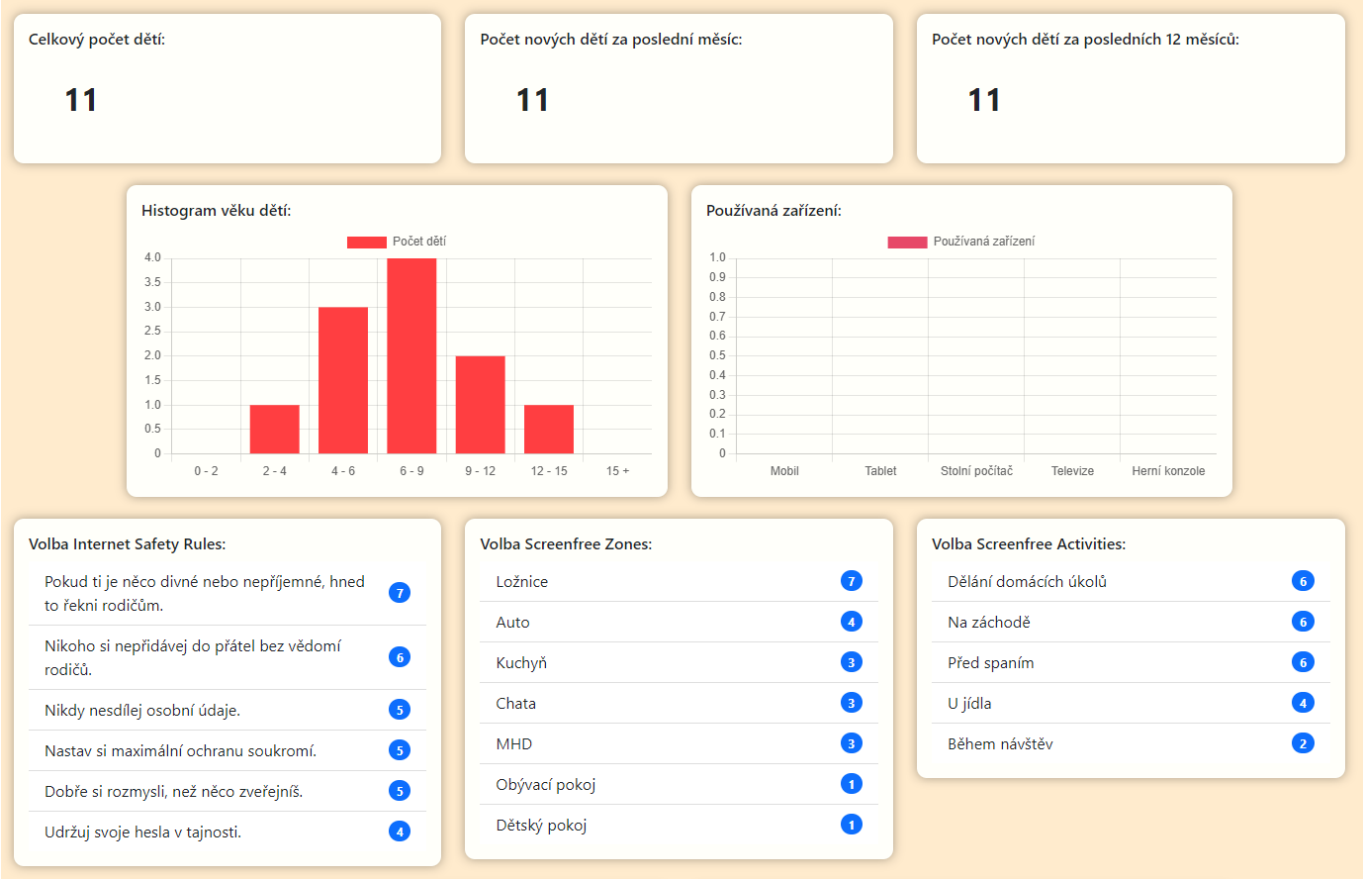
9.4 Analýza dat uživatelů

Díky nástroji Digital Parenting Dashboard jsem byl schopný po dokončení uživatelského testování analyzovat data, která testeři do aplikace zadali.

Z User Dashboardu jsem vyčetl, že v aplikaci bylo vytvořeno celkem 7 uživatelských účtů. To je tedy o 2 více, než kolik jsme měli respondentů v testovacím dotazníku. 3 uživatelé si v aplikaci založili pouze 1 dítě, kdežto zbývající 4 si založili děti 2. Někteří testeři tedy testovali také funkci přidávání více dětí.

Zajímavější informace se ovšem nacházely v Kid Dashboardu. Nejčastější zastoupenou věkovou skupinou dětí byly děti ve věku 6 až 9 let. V systému byly celkem 4 děti v tomto věkovém rozmezí. Dalším zajímavým údajem z testovaní byla volba bezpečnostních internetových pravidel, volba bez-obrazovkových míst a volba bez-obrazovkových aktivit. Nejvíce vybíranou volbou internetových pravidel byla tato: *„Pokud ti je něco divné nebo nepříjemné, hned to řekni rodičům“*. Tuto volbu si u jednotlivých dětí zvolili testeři celkem 7krát. Naopak nejméně voleným pravidlem bylo *„Udržuj svoje hesla v tajnosti“* (zvoleno 4krát). Nejvíce vybíraným bez-obrazovkovým místem byla *Ložnice* (7), nejméně *Dětský pokoj* (1). Z bez-obrazovkových aktivit byla nejvíce vybírána možnost *Dělání domácích úkolů* (6), nejméně *Během návštěv* (2). Kompletní data z Kid Dashboardu lze vidět na přiložených obrázcích 9.2 a 9.3.

Aplikace Digital Parenting pro operační systém Android v sobě nemá implementovaný výběr zařízení, které děti používají. Všichni respondenti v dotazníku uvedli, že používali verzi pro operační systém Android. Z toho důvodu je výsledný graf *Používaná zařízení* prázdný.



Obrázek 9.2. Kid Dashboard – výsledek testování (část 1)

Obrázek 9.3. Kid Dashboard – výsledek testování (část 2)

Kapitola 10
Závěr

Cílem této práce byl návrh, implementace, nasazení a také následné uživatelské testování back-end řešení pro již existující klientskou aplikaci Digital Parenting. Nejdříve bylo nutné se seznámit s problematikou digitální rodičovství a pochopit na jakých principech fungovala již vytvořená mobilní aplikace. Poté proběhl sběr a analýza požadavků pro nově vznikající back-end řešení. Na základě toho se vybrala odpovídající softwarová architektura. Při návrhu a implementaci jsem musel spolupracovat s dalšími vývojáři v týmu, kteří implementovali a rozšiřovali klientské aplikace. Během práce na projektu se původní aplikace obohatila o několik užitečných funkcionalit, jako jsou například synchronizovaný uživatelský účet, nebo možnost vytvoření více dětí pod jedním uživatelem. Aplikační server byl nasazen na hostovací platformu Heroku, odkud vystavuje veřejné aplikační rozhraní pro použití v klientských aplikacích. Nedílnou částí práce bylo také testování celé aplikace (včetně back-end řešení) na reálných uživatelích. Testeři nám poskytli užitečnou zpětnou vazbu, která může posloužit jako výchozí bod pro následné rozšiřování a upravování celého systému.

Nad rámec zadaní vznikla také jednoduchá webová aplikace Digital Parenting Dashboard. Ta slouží pro vizualizovaný pohled do agregovaných uživatelských dat v systému. Digital Parenting Dashboard prozatím poskytuje pouze souhrnné pohledy do dat v databázi. Plánuje se ovšem tuto aplikaci rozšířit o vyhledávací systém, ve kterém by šlo vyhledat konkrétního uživatele a analyzovat pouze jeho data odděleně od zbytku uživatelů.

Literatura

[1] Lukavská, Kateřina, Jaroslav Vacek, Ondřej Hrabec, Michal Božík, Michaela Slussareff, Martina Píšová, David Kocourek, Lucie Svobodová a Roman Gabrhelík. Measuring Parental Behavior towards Children's Use of Media and Screen-Devices: The Development and Psychometrical Properties of a Media Parenting Scale for Parents of School-Aged Children. *International Journal of Environmental Research and Public Health*. 2021, ročník 18, č. 17. ISSN 1660-4601. Dostupné na DOI 10.3390/ijerph18179178. Dostupné na `https://www.mdpi.com/1660-4601/18/17/9178`.

[2] Lukavská, Katerina, Jaroslav Vacek a Roman Gabhelík. *The effects of parental control and warmth on problematic internet use in adolescents: A prospective cohort study*. Dostupné na DOI 10.1556/2006.2020.00068. Dostupné na `https://akjournals.com/view/journals/2006/9/3/article-p664.xml`.

[3] Radesky, Jenny S., Heidi M. Weeks, Rosa Ball, Alexandria Schaller, Samantha Yeo, Joke Durnez, Matthew Tamayo-Rios, Mollie Epstein, Heather Kirkorian, Sarah Coyne a Rachel Barr. Young Children's Use of Smartphones and Tablets. *Pediatrics*. 07, 2020, ročník 146, č. 1. ISSN 0031-4005. Dostupné na DOI 10.1542/peds.2019-3518. Dostupné na `https://doi.org/10.1542/peds.2019-3518`. e20193518.

[4] Gadireddi, Erik. *Design of Therapeutical Application for Digital Addiction Reduction*. Jugoslávských partyzánů 1580/3, Praha, 160 00: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2022 [vid. 2022-05-09]. Bakalářská práce. Dostupné na `http://hdl.handle.net/10467/99216`.

[5] G., Nick. *Android: Market Share & Other Stats for 2022 [Infographic*. [vid. 2022-05-09]. Dostupné na `https://techjury.net/blog/android-market-share/`.

[6] Wiegers, K. a J. Beatty. *Software Requirements*. Pearson Education, 2013. Developer Best Practices. ISBN 9780735679627. Dostupné na `https://books.google.cz/books?id=nbpCAwAAQBAJ`.

[7] Smolík, Tomáš. *Softwarová architektura*. [vid. 2022-05-12]. Dostupné na `https://profinit.eu/univerzity/material-sweng/architecture-and-design/`.

[8] *Microservice vs monolitické*. [vid. 2022-05-12]. Dostupné na `https://cs.education-wiki.com/1235570-microservice-vs-monolithic`.

[9] Wikipedie. *Klient–server — Wikipedie: Otevřená encyklopedie*. [vid. 2022-05-11]. Dostupné na `https://cs.wikipedia.org/w/index.php?title=Klient%E2%80%93server&oldid=21196886`.

[10] Wikipedie. *Representational State Transfer — Wikipedie: Otevřená encyklopedie*. [vid. 2022-05-12]. Dostupné na `https://cs.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=18851475`.

[11] *What is a REST API?* [vid. 2022-05-12]. Dostupné na `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.

[12] Choudary, Archana. *What is Computer Security and Its Types? Introduction to Computer Security*. [vid. 2022-05-12]. Dostupné na https://www.edureka.co/blog/what-is-computer-security/.

[13] Hron, Martin. *10 největších úniků dat v roce 2018*. [vid. 2022-05-12]. Dostupné na https://blog.avast.com/cs/10-nejvetsich-uniku-dat-v-roce-2018.

[14] *What is HTTPS?* [vid. 2022-05-13]. Dostupné na https://www.cloudflare.com/learning/ssl/what-is-https/.

[15] Uusivaara, Christophe Riolo. *How to choose a programming language for your project? Avoid these biases.* [vid. 2022-05-13]. Dostupné na https://qvik.com/news/how-to-choose-a-programming-language-for-project/.

[16] *Python Introduction*. [vid. 2022-05-13]. Dostupné na https://www.w3schools.com/python/python_intro.asp.

[17] Wikipedie. *Java (programovací jazyk) — Wikipedie: Otevřená encyklopedie*. [vid. 2022-05-13]. Dostupné na https://cs.wikipedia.org/w/index.php?title=Java_(programovac%C3%AD_jazyk)&oldid=20781848.

[18] Wikipedie. *Relační databáze — Wikipedie: Otevřená encyklopedie*. [vid. 2022-05-14]. Dostupné na https://cs.wikipedia.org/w/index.php?title=Rela%C4%8Dn%C3%AD_datab%C3%A1ze&oldid=21163187.

[19] *Difference between SQL and NoSQL*. [vid. 2022-05-14]. Dostupné na https://www.geeksforgeeks.org/difference-between-sql-and-nosql/.

[20] *MongoDB Cloud Services*. [vid. 2022-05-14]. Dostupné na https://www.mongodb.com/cloud.

[21] contributors, Wikipedia. *Salt (cryptography) — Wikipedia, The Free Encyclopedia*. [vid. 2022-05-15]. Dostupné na https://en.wikipedia.org/w/index.php?title=Salt_(cryptography)&oldid=1087581909.

[22] contributors, Wikipedia. *Heroku — Wikipedia, The Free Encyclopedia*. [vid. 2022-05-15]. Dostupné na https://en.wikipedia.org/w/index.php?title=Heroku&oldid=1085566739.

[23] Wikipedie. *Dokumentace k softwaru — Wikipedie: Otevřená encyklopedie*. [vid. 2022-05-16]. Dostupné na https://cs.wikipedia.org/w/index.php?title=Dokumentace_k_softwaru&oldid=20769507.