FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Systém pro hierarchickou evidenci poznámek pro Android

Lukáš Runt

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Systém pro hierarchickou evidenci poznámek pro Android

Lukáš Runt

Vedoucí práce
Ing. Ladislav Pešička

PLZEŇ **2024**

Citace v seznamu literatury:
RUNT, Lukáš. *Systém pro hierarchickou evidenci poznámek pro Android*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Ladislav Pešička.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš RUNT**
Osobní číslo:	**A20B0226P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Systém pro hierarchickou evidenci poznámek pro Android**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prozkoumejte vybrané aplikace pro správu poznámek na mobilních zařízeních.
2. Analyzujte vhodné technologie a frameworky pro realizaci hierarchické evidence poznámek na mobilní platformě Android.
3. Navrhněte systém pro hierarchickou evidenci poznámek pro platformu Android. Systém bude umět synchronizovat poznámky z více zařízení.
4. Navržený systém realizujte, ověřte jeho funkcionalitu pomocí netriviální sady testů a navrhněte možná rozšíření.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Ladislav Pešička**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 02. května 2024

. . . . . .
Lukáš Runt

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Cílem bakalářské práce je navrhnout a implementovat mobilní aplikaci pro tvorbu hierarchicky strukturovaných poznámek. Práce zkoumá vybrané existující aplikace pro tvorbu poznámek. V další části se práce zabývá existujícími technologiemi pro vývoj mobilních aplikací a možnostmi uložení dat. Praktická část obsahuje specifikaci funkcionality vyvíjené aplikace, její návrh a samotnou implementaci. Následně je vytvořená aplikace otestována netriviálními testy a jsou navržena možná rozšíření.

Abstract

This bachelor thesis aims to design and implement a mobile application for creating and editing hierarchically structured notes. The paper reviews selected existing applications and explores technologies used in mobile application development as well as data-storing technologies. The practical section covers the specification of the application that will be developed, its design, model, and its implementation. The final section of the thesis includes testing of the implemented application and proposes potential enhancements.

Klíčová slova

mobilní aplikace • Flutter • Firebase • Android • tvorba poznámek • multiplatformní vývoj • testování

Poděkování

Tímto bych rád poděkoval vedoucímu práce Ing. Ladislavu Pešičkovi, za cenné rady, vstřícnost, trpělivost a za jeho energii a čas, který mi byl ochoten věnovat.

Dále bych rád poděkoval rodině, přátelům, přítelkyni a všem ostatním, kdo mě při průběhu mého studia podporovali.

Obsah

Úvod 1

Aplikace pro tvorbu poznámek umožňují uživatelům efektivně a flexibilně zaznamenávat důležité informace, myšlenky, úkoly či termíny schůzek. Vytvořené poznámky mohou mnohdy obsahovat formátovaný text, obrázky, videa, zvukové nahrávky, dokumenty a skici, které zvyšují univerzálnost a uživatelský zážitek.

V současnosti existuje mnoho aplikací, které se snaží zaujmout uživatele svým přístupem k organizaci poznámek a velkým množství funkcí, které v některých případech zahrnují integraci s kalendářem, umělou inteligenci a spolupráci v týmech. Aby měl uživatel poznámky stále po ruce, je snaha vytvářet tyto aplikace na co nejvíce platforem. Vzhledem k velkému konkurenčnímu prostředí se vývojáři snaží vytvořit uživatelsky přívětivé prostředí, které bude uživatele motivovat k používání aplikace, v některý případech může být uživatelům za používání aplikace měsíčně účtován nemalý finanční obnos.

Cílem této bakalářské práce je vytvořit jednoduchou aplikaci pro zapisování poznámek. Aplikace bude zaměřena na vytváření školních poznámek v textovém formátu s možností vkládat hypertextové odkazy. Důležitou vlastností této aplikace bude libovolné zanoření poznámek, synchronizování poznámek mezi více zařízeními, rozšiřitelnost aplikace o další funkce a bezplatné použití.

V úvodu naší práce vyzkoušíme některé z existujících aplikací, které jsou určeny pro tvorbu hierarchicky strukturovaných poznámek. Zjistíme, jak tyto aplikace vypadají, jaká jsou jejich pozitiva a negativa, funkce a řešení některých důležitých aspektů. Z těchto poznatků následně specifikujeme plánovanou funkcionalitu naší aplikace. V další části práce provedeme analýzu technologií vhodných pro vývoj námi specifikované mobilní aplikace. V této části se zaměříme, které frameworky a programovací jazyky je vhodné použít pro tvorbu naší práce. Dále také prozkoumáme možnosti ukládání dat mobilních aplikací. Na konci této části následně specifikujeme, jaké technologie budou ve vývoji použity. Další fáze naší práce se bude zabývat návrhem a implementací samotné aplikace námi specifikovanými vlastnostmi. Vytvořená aplikace bude následně systematicky otestována netriviálními testy. Na závěr provedeme zhodnocení vytvořené aplikace a předložíme návrhy na možná zlepšení a rozšíření.

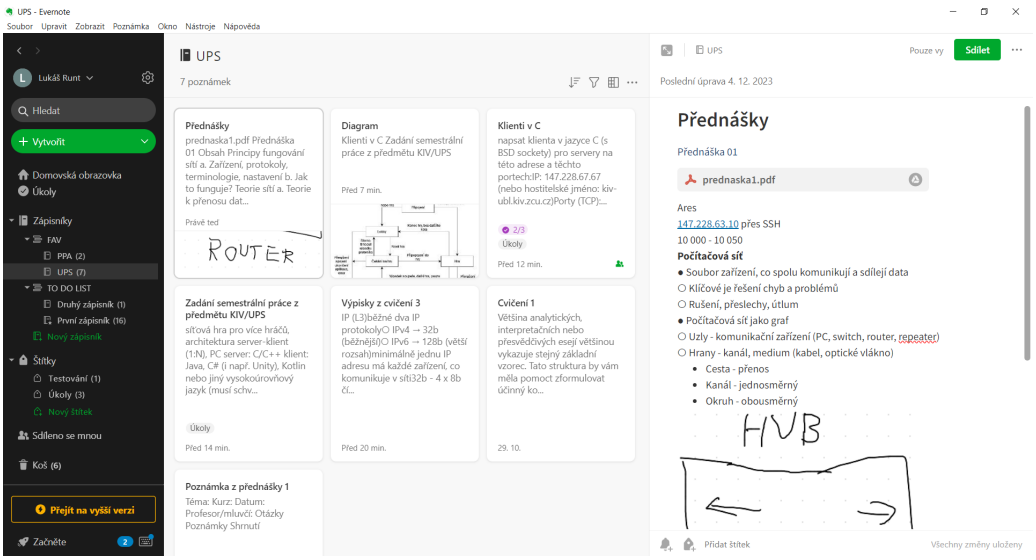
Analýza vybraných aplikací

2

V rámci analýzy vybraných aplikací se zaměříme na známé aplikace s velkým počtem aktivních uživatelů, které jsou kompatibilní s platformou Android. Tyto aplikace by měly umožňovat organizování poznámek do hierarchické struktury, nabízet možnost synchronizace mezi různými zařízeními, spolupráci s dalšími uživateli a možnost šifrování dat pro zajištění bezpečnosti.

2.1 Evernote

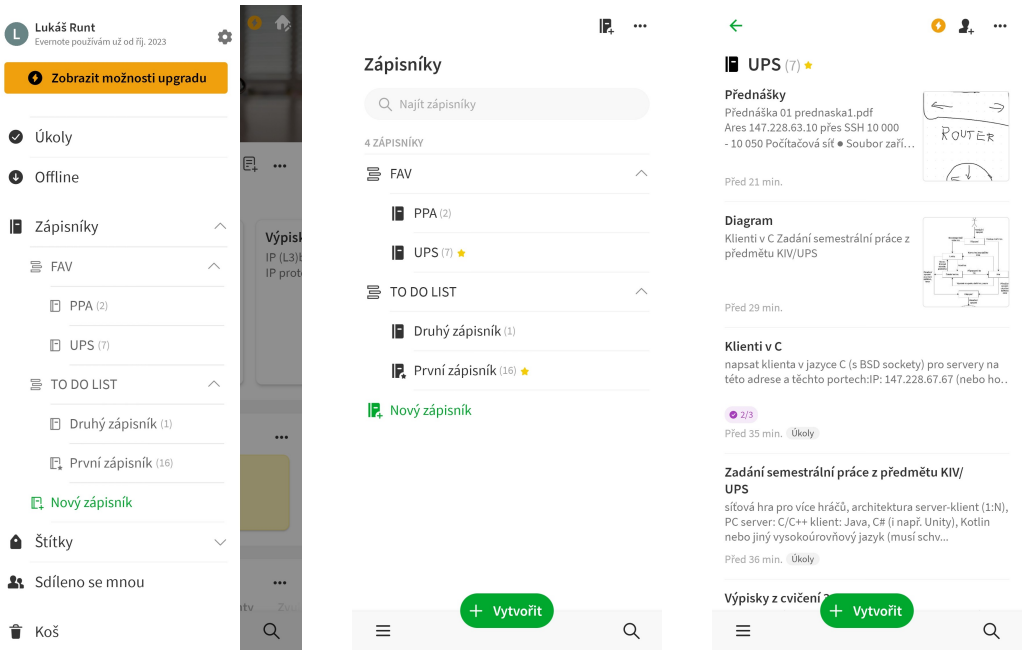
Evernote od společnosti Evernote Corporation (viz obrázek 2.1) je jedna z nejpopulárnějších a nejpoužívanějších aplikací pro tvorbu hierarchicky organizovaných poznámek. Hlavní předností aplikace je dostupnost na všech běžných platformách, snadné ovládání a velký počet funkcí. [1]



Obrázek 2.1: Počítačová verze aplikace Evernote

Evernote nabízí čtyři verze předplatného Free, Personal, Professional a Teams. Základní verze je bezplatná, avšak je pro uživatele velmi limitující. V bezplatné verzi můžeme synchronizovat své poznámky jen mezi dvěma zařízeními (web se počítá jako jedno zařízení), poznámky se dají stahovat pro offline použití pouze na počítači[1], omezeny jsou funkce vyhledávání v PDF (Portable Document Format) souborech a obrázcích, měsíční upload dat je omezen na 60 MB a maximální velikost poznámky na 25 MB a tím bohužel výčet ani zdaleka nekončí. Ceny prémiových účtů se pohybují od 12,99 €/měsíčně[2].

Aplikace umožňuje vytvářet poznámky a úkoly. Do obsahu poznámek a úkolů je možné vkládat formátovaný text, skici, obrázky, soubory i zvukové nahrávky. Při vytváření nových poznámek lze využít předdefinovaných šablon. Poznámky lze organizovat pomocí štítků, zkratek a hierarchické struktury, která je zobrazena seznamem a stromovou strukturou ve vysouvacím menu (viz obrázek 2.2). Maximální hloubka hierarchie je omezena na tři úrovně. Pro každou úroveň je definován jeden typ souboru, a to svazek (úroveň 1), zápisník (úroveň 2) a poznámka (úroveň 3). Vzhledem k tomu, že soubory existují pouze v rámci své vlastní úrovně, není možné vložit soubor stejného typu do identického typu souboru.



Obrázek 2.2: Organizace poznámek v mobilní aplikaci Evernote. Lišta se stromovou strukturou (vlevo), seznam se zápisníky (uprostřed), seznam poznámek (vpravo)

[1] Aplikace uchovává v paměti jen poznámky, které jsme zobrazili nebo upravili. S placenou verzí můžeme zvolit, které poznámky budou staženy pro režim bez připojení k internetu.

[2] Cena předplatného k 19. 11. 2023

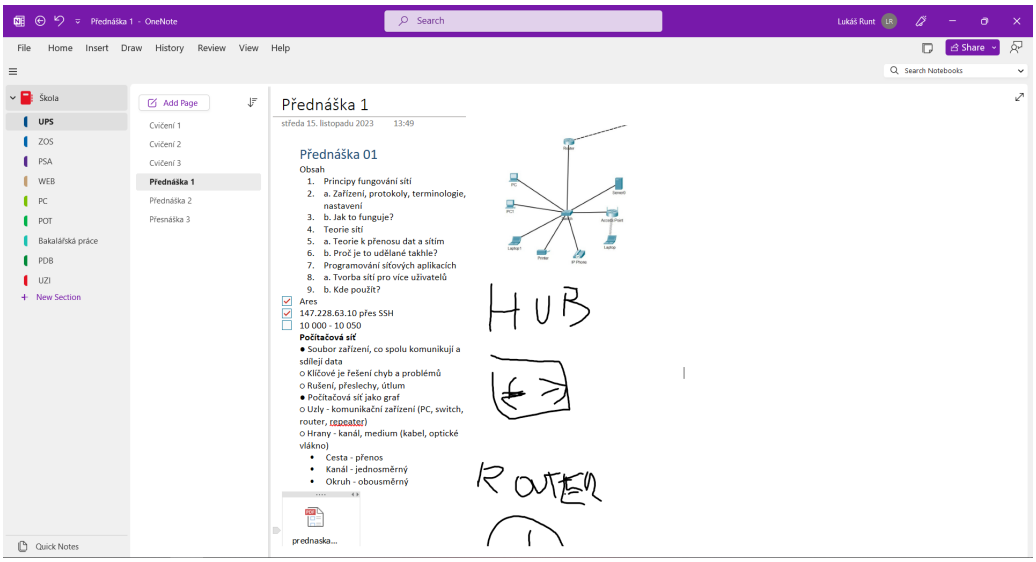
Synchronizace poskytuje uživatelům přístup ke svým poznámkám z více zařízení. Evernote využívá real-time synchronizaci, tedy změny provedené na jednom zařízení se okamžitě promítají na ostatní zařízení. Uživatelé mohou své poznámky upravovat i bez připojení k internetu. Jakmile je připojení k internetu obnoveno, provedené změny jsou automaticky synchronizovány. Kromě synchronizace mezi zařízeními lze také sdílet data mezi účty. Sdílet můžeme zápisníky a poznámky pomocí odkazu nebo poslání pozvánky. V rámci sdílení je možné nastavovat práva ostatním uživatelům. Kdo může data zobrazit, editovat a rozesílat pozvánky. Všechna data jsou uložena na interních serverech společnosti Evernote. Bezpečnost dat při přenosu mezi zařízeními je zajištěna moderními průmyslovými standardy šifrování TLS (Transport Layer Security), SSL (Secure Sockets Layer) a HSTS (HTTP Strict Transport Security).

Evernote se snaží vytvořit uživatelsky přívětivé a produktivní prostředí. Každý uživatel má možnost editace domovské stránky, organizace dat, vytváření štítků a zkratek. Rychlejší vyhledávání požadovaných informací ulehčuje řazení, filtrování, standardní vyhledávání, vyhledávání v dokumentech a obrázcích s použitím metod OCR (Optical Character Recognition) a vyhledávání s použitím umělé inteligence, které vrací odpovědi na dotazy psané v přirozeném jazyce. Umělá inteligence je obsažena také ve funkci AI Note Cleanup, která formátuje, opravuje chyby a upravuje věty, přičemž zanechává původní význam. Myšlenky lze rychle zachytit bez otevření aplikace díky rychlým poznámkám, webovému rozšíření pro ukládání a vytváření výstřižků webových stránek, vestavěné funkci fotoaparátu a nahrávání hlasu. Dále aplikace nabízí notifikace a několik druhů widgetů, které se dají přidat na plochu. Při práci uživatelé rovněž ocení historii změn poznámek. Tato funkce umožňuje sledovat historii úprav a vrátit se k libovolné verzi, nebo obnovit smazaný soubor.

Bohatý je také počet integrací s ostatními aplikacemi. Uživatelé mohou propojit například Google Disk a Dropbox, pro vkládání souborů přímo z cloudu. Uživatelé s předplatným mohou spojit aplikaci se svým kalendářem a e-mailovým účetem, čímž získat možnost sledovat a aktualizovat události přímo z Evernote. Dále lze připojit například Microsoft Teams, Trello, Slack, Zapier a další užitečné aplikace.

2.2 OneNote

OneNote je software pro vytváření poznámek od společnosti Microsoft, který je součástí kancelářského balíku Microsoft Office. V současné době je možné tento software používat bezplatně na všech platformách včetně mobilního zařízení a webového prohlížeče. Uživatelé bezplatného účtu jsou omezeni 5 GB úložného prostoru služby OneDrive, kterou OneNote používá pro ukládání dat. Pokud uživatel potřebuje větší úložný prostor, je možno zvýšit tento limit na 100 GB za 1,99 $/měsíčně [2].



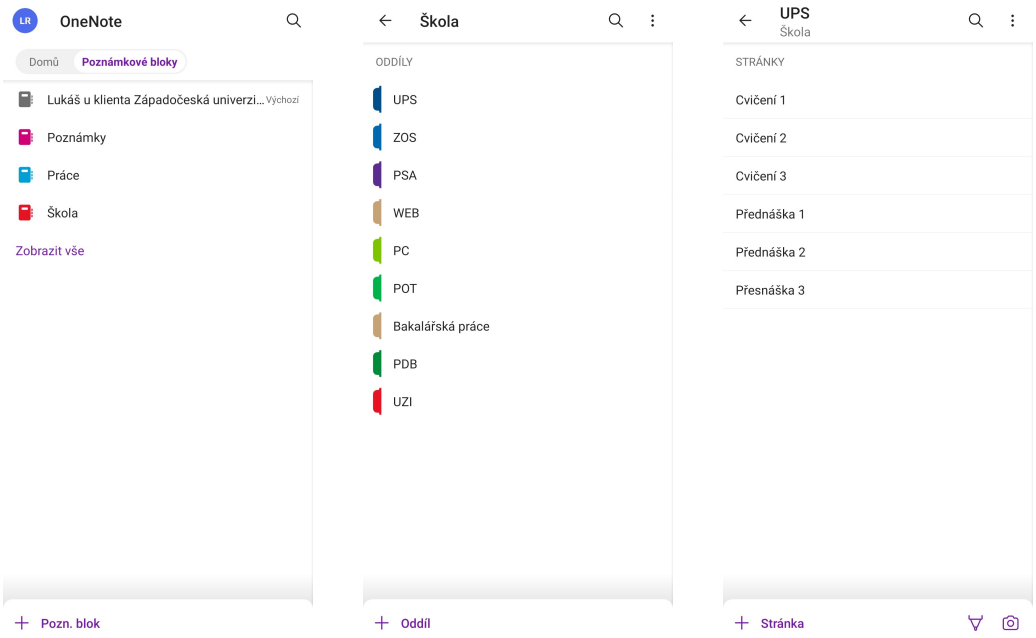
Obrázek 2.3: Aplikace OneNote na počítači

Uživatelské rozhraní aplikace OneNote připomíná ostatní produkty od společnosti Microsoft (viz obrázek 2.3). Při vytváření poznámek máme k dispozici formátování textů, vytváření zaškrtávacích seznamů, vkládání obrázků, souborů, zvukových nahrávek, kreslení skic ani tím výčet nekončí. Poznámky jsou uspořádány do poznámkových sešitů, ve kterých je možno vytvořit libovolný počet oddílů, v nichž můžeme mít neomezený počet stránek[3]. Poznámky je možné organizovat do tří úrovní. Uspořádání poznámek je na počítači a webovém prohlížeči zobrazeno v levé části obrazovky. Na mobilním zařízení se hierarchií postupně proklikáváme v seznamem (viz obrázek 2.4), dokud se nedostaneme k požadované poznámce.

Přesouvání poznámek do jiných poznámkových bloků a je realizováno tlačítkem Přesunout, které se zobrazí při dlouhém podržení (mobilní zařízení), respektive pravým kliknutím na poznámku (počítač). Následně si vybíráme ze seznamu oddílů a poznámkových bloků, kam bude poznámka přesunuta. Tato funkce dokáže také poznámky kopírovat. Na počítači lze stránky přesouvat přetažením do jiného oddílu. Přesouvání oddílů do jiných poznámkových bloků není umožněno.

OneNote poskytuje automatickou synchronizaci v reálném čase a spolupráci mezi více uživateli. Poznámky jsou dostupné i v offline režimu, provedené změny se synchronizují po připojení k internetu. Synchronizace a ukládání dat probíhá prostřednictvím cloudové služby OneDrive. Uživatelé mají možnost spravovat přístup ostatních uživatelů k jejich poznámkám. Poznámkové bloky a oddíly se dají zabezpečit heslem, což způsobí uzamčení a zašifrování všech stránek, dokud uživatel nezadá správné heslo. Všechny soubory jsou zkontrolovány antivirovou ochranou a zakódovány unikátním AES (Advanced Encryption Standard)-256 klíčem. Šifro-

[3] Počet stránek a oddílů je omezen velikostí úložného prostoru



Obrázek 2.4: Hierarchie v mobilní aplikaci OneNote

ván je také přenos mezi zařízeními a servery Microsoftu, který využívá protokolu TLS a HTTPS (Hypertext Transfer Protocol Secure). Žádný vývojář nemá přístup k datům a do datacenter má přístup jen určený personál.

Kromě již zmíněných funkcí nabízí OneNote také widgety na plochu a Web Cliper, který dokáže ukládat články, výstřižky, odkazy a anotovat obsah. Z běžné funkcionality nechybí ani hledání, filtrování, řazení poznámek, historie úprav, vytváření odkazů mezi poznámkami, používání šablon, práce s matematickými vzorci a tabulkami, správa úkolů a podpora citací a další. Pro uživatele dotykových zařízení je k dispozici podpora pera, korekce a rozpoznávání rukou napsaného textu. Přítomné je také rozeznávání textu OCR v dokumentech a obrázcích.

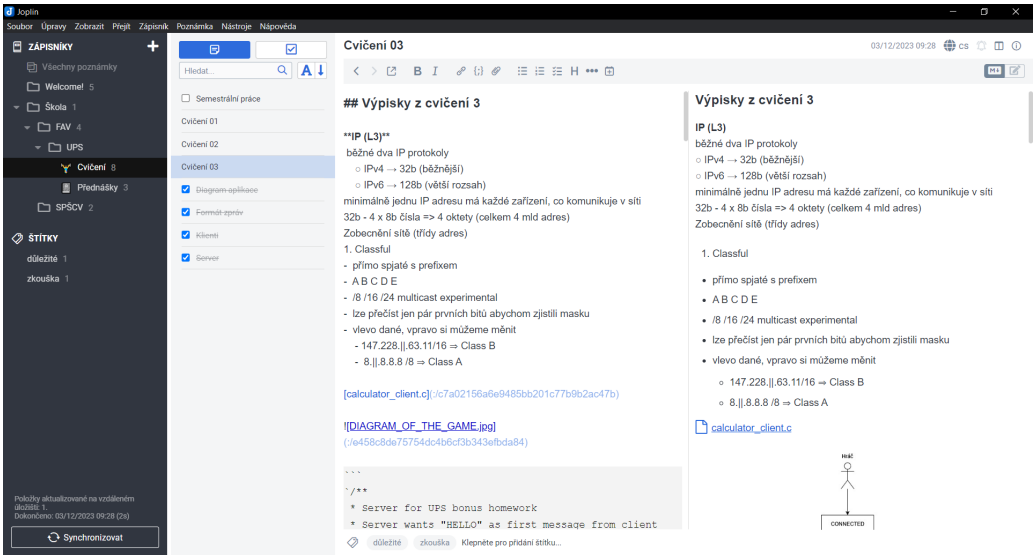
Nechybí ani integrace s dalšími aplikacemi balíčku Microsoft Office, přes aplikaci Newton je možné posílat důležité e-maily a přes Zapier je možné propojit OneNote s dalšími aplikacemi, jako je Twitter, Google kalendář, Trello a mnohé další.

2.3 Joplin

Joplin je open-source aplikace pro zaznamenávání poznámek a úkolů, jejíž zdrojový kód je veřejně dostupný na platformě GitHub[4] [3]. Aplikace je dostupná na mobilních zařízeních a počítačích (viz obrázek 2.5), kde je dostupná i verze bez grafického rozhraní v konzoli. Webová verze není oficiálně k dispozici. Kód aplikace je napsán

[4] `https://github.com/laurent22/joplin`

v jazycích TypeScript a JavaScript. Zatímco mobilní aplikace je vyvinuta pomocí frameworku React Native, počítačová verze využívá framework Electron. Aplikace Joplin je bezplatná, avšak synchronizace dat prostřednictvím služby Joplin Cloud vyžaduje placené předplatné. Ceny Joplin Cloud se pohybují od 2,99 € za měsíc.



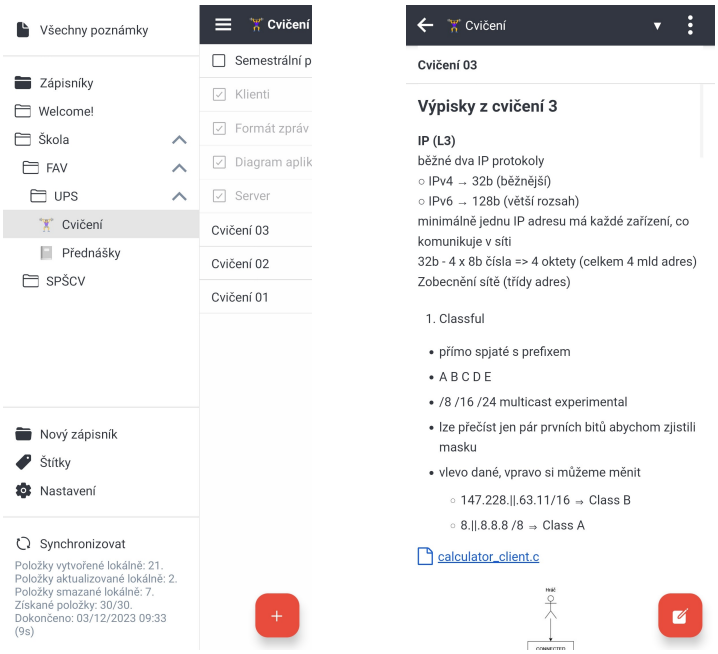
Obrázek 2.5: Aplikace Joplin na počítači

Aplikace nám umožňuje vytvořit poznámky a úkoly. Úkoly se od poznámek liší zaškrtávacím polem, které lze označit jako dokončené. Obsah tvoří formátovaný text s obrázky, soubory a odkazy. Každá poznámka obsahuje také metadata o času a poloze vytvoření, času posledního upravení a verze úprav. Vytváření skic přímo v aplikaci chybí[5]. Poznámky se zobrazují ve dvou režimech, a to v režimu čtení a režimu úprav. V režimu úpravy si uživatel může přizpůsobit editor podle svého vkusu, jestli má raději WYSIWYG (What You See Is What You Get) editor nebo raději píše text se syntaktickými značkami[6]. Náročnější uživatelé mohou připojit svůj oblíbený textový editor a upravovat v něm poznámky. Poznámky lze organizovat pomocí štítků a hierarchie ve stromové struktuře, která je umístěna v levé části obrazovky. Hierarchie obsahuje poznámky a zápisníky a umožňuje libovolný počet úrovní (viz obrázek 2.6). Zápisníky plní funkci složek, lze do nich vkládat poznámky i zápisníky. Přesun zápisníků a poznámek je v mobilní verzi aplikace implementován nastavením rodičovského zápisníku ze seznamu. V desktopové verzi je navíc implementováno přesouvání přetahováním souborů ve stromové struktuře.

Primárně je aplikace navržena pro použití bez přístupu k internetu, a tak každé zařízení, na kterém je Joplin nainstalován, uchovává svou vlastní kopii dat. Své po-

[5]Testování probíhalo v aplikaci bez nainstalovaných pluginů

[6]Joplin povoluje více formátu poznámek, mezi které patří například Markdown nebo HTML (Hypertext Markup Language)



Obrázek 2.6: Hierarchická organizace dat (vlevo) a editace poznámky (vpravo)

známky může uživatel také exportovat a importovat v několika možných formátech. Pro automatickou synchronizaci, která se provádí v časových intervalech, si uživatel nastaví časový interval a platformu, na které se budou data ukládat. Je možno zvolit OneDrive, DropBox, NextCloud, Joplin server, Joplin Cloud nebo jinou platformu, kterou Joplin nabízí. Poslední zmiňované služby Joplin Cloud a Joplin Server jsou placené. Oproti ostatním platformám umožňují tyto platformy zveřejnění poznámek na internetu a sdílení dat mezi účty. Vzhledem k časovým intervalům mohou vzniknout při synchronizování dat konflikty, když uživatel současně edituje stejný soubor na více zařízeních. V tomto případě Joplin detekuje konflikt a vytvoří poznámkový blok s konflikty, do kterého nahraje problematické soubory, dále jsou data přepsána. Uživatel si poté může konflikty zobrazit a případně obnovit všechny potřebné změny. Data lze šifrovat koncovým šifrováním, to znamená, že aplikace vygeneruje klíč, kterým jsou data šifrována a dešifrována až u koncového zařízení, na kterém jsou data zobrazována. Šifrování je ve výchozím nastavení vypnuté.

Aplikace obsahuje téměř kompletní českou lokalizaci [7] a kontrolu českého pravopisu. Vyzdvihnout lze také webové rozšíření pro Firefox a Google Chrome, které umožňuje vytváření výstřižků, ukládání URL (Uniform Resource Locator) adres, uložení celé nebo části webu v několika možných formátování (text, HTML a Markdown). V aplikaci nechybí ani řazení, filtrování a hledání v poznámkách. Pokud by uživateli nevyhovoval vzhled nebo ovládání aplikace, může si aplikaci přizpůsobit podle svého vkusu v nastavení. To samé platí i o instalaci a nastavení rozšíření.

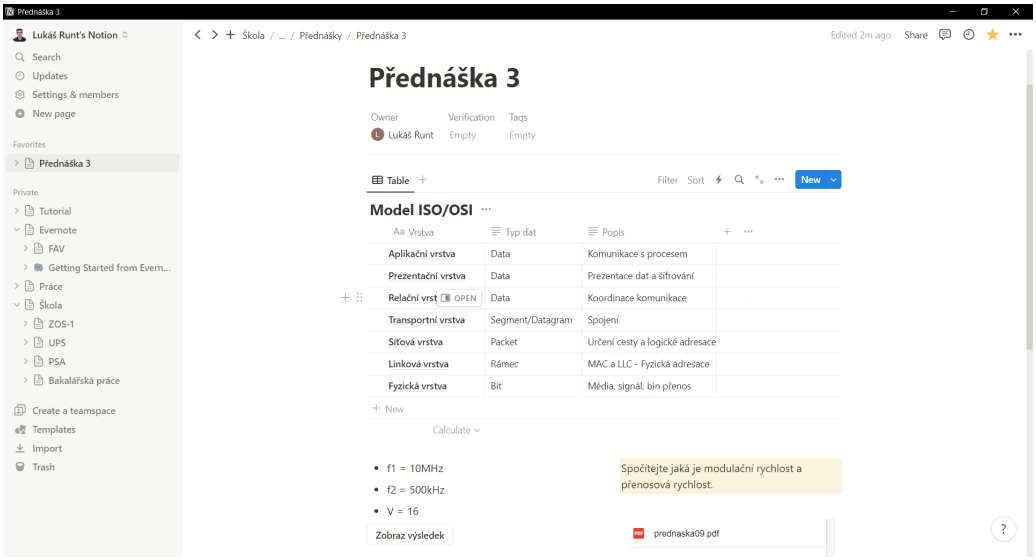
[7] K datu 3. 11. 2023 je přeloženo 96% textů

Vzhledem k tomu, že má Joplin veřejný kód a zveřejněnou vývojářskou dokumentaci na webových stránkách, má komunita možnost pomoc s vývojem aplikace. V současné době existuje mnoho rozšíření, které dokáže vylepšit pracovní prostředí Joplinu, jde například o přidání kalendáře, nástrojů pro vytváření diagramů, widgety, podpora LaTeX, integraci s aplikacemi a mnoho dalších funkcí.

2.4 Notion

Notion je multifunkční aplikace s širokým využitím pro spolupráci a organizaci poznámek. Díky rozsáhlé funkcionalitě a integracím s ostatními aplikacemi jde o opravdu silný nástroj, který lze použít ve firemním prostředí pro řízení projektů a týmů. Aplikace je dostupná na Windows (viz obrázek 2.7), macOS, iOS, Android a ve webovém prohlížeči. [4]

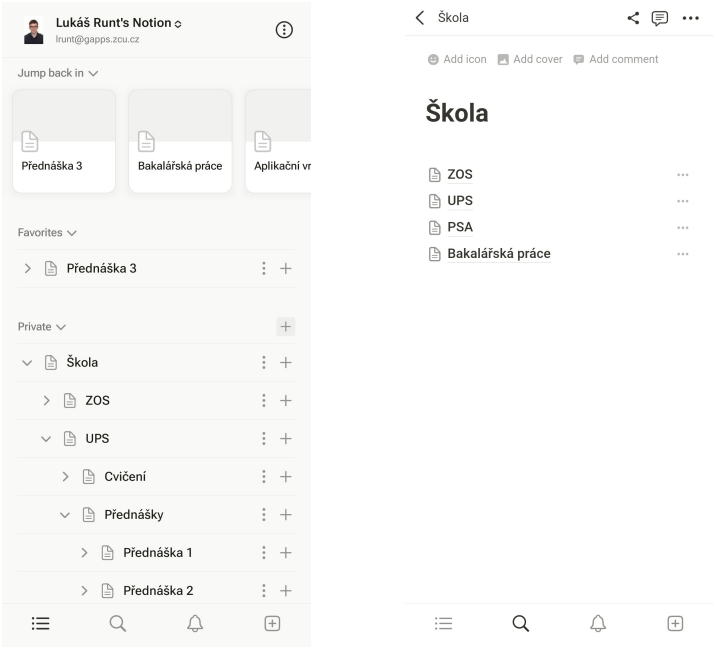
Základní verze aplikace je dostupná zdarma. Náročnější uživatelé si mohou vybrat z tří tarifů předplatného Plus, Business a Enterprise, jejichž cena začíná na 8 $/Měsíčně. Základní verze nabízí většinu funkcionality, která stačí většině nenáročných uživatelů. Oproti placeným verzím je zde třeba omezena velikost nahraných souborů 5 MB/Soubor, spolupráce s více uživateli, historie úprav, exportování poznámek do PDF formátu a mnohé další. Pokud se tedy očekává práce v týmu je vhodné si upgradovat na jednu z placených verzí. Studenti a vyučující mohou bezplatně získat některé funkce tarifu plus.



Obrázek 2.7: Notion na počítači

Poznámky se píší a ukládají do stránek. Obsah stránek je strukturován do bloků, které můžeme libovolně uspořádat přetahováním pod sebe, vedle sebe nebo do sebe. Typ obsahu bloků stránek je velice rozmanitý. Kromě textu, obrázků a videí nabízí

také vkládání databází, matematických vzorců, kalendářových bloků, map, internetových stránek, komentářů a další typy obsahů. Do obsahu poznámky lze také vložit další stránku. Tato stránka poté zobrazí v hierarchickém stromě (viz obrázek 2.8). Hierarchické uspořádání umožňuje libovolný počet úrovní. Uspořádání stránek lze upravovat přetahováním do příslušného místa ve stromě nebo pomocí tří teček u stránky a následného zvolení „move to", kde si uživatel může vybrat místo, kam bude stránka přesunuta.



Obrázek 2.8: Hierarchická struktura (vlevo) a odkazy na další stránky v obsahu poznámky (vpravo) v mobilní aplikaci Notion

Notion je primárně určen pro používání online. Každá změna, kterou uživatel udělá, se automaticky odesílá na cloudovou službu Amazon Web Service, ve které se data ukládají ve formátu JSON (JavaScript Object Notation). Všechna data se v reálném čase promítají na všechna ostatní zařízení. Aplikace mimo jiné ukládá data do mezipaměti v zařízení uživatele. Do této paměti se však ukládají jen data poznámek, které uživatel otevřel. Takže pokud chce uživatel prohlížet a upravovat některou stránku bez připojení k internetu, musí ji otevřít ještě když má připojení k internetu a poté nechat aplikaci spuštěnou. Data se po připojení k internetu synchronizují jen za předpokladu, že uživatel nezavřel aplikaci a stránku, ve které proběhlo k úpravám. V případě, že uživatel stránku zavře nebo obnoví obsah, dojde ke ztrátě dat. Synchronizace není dostupná jen mezi zařízeními, ale je možné sdílet blok i v rámci více stránek. Notion nabízí různé úrovně sdílení a práv, které umožňují uživatelům spolupracovat na obsahu a projektech. Sdílené stránky umožňují vytvářet a sdílet konkrétní stránky a databáze s ostatními. Pracovní prostory pak

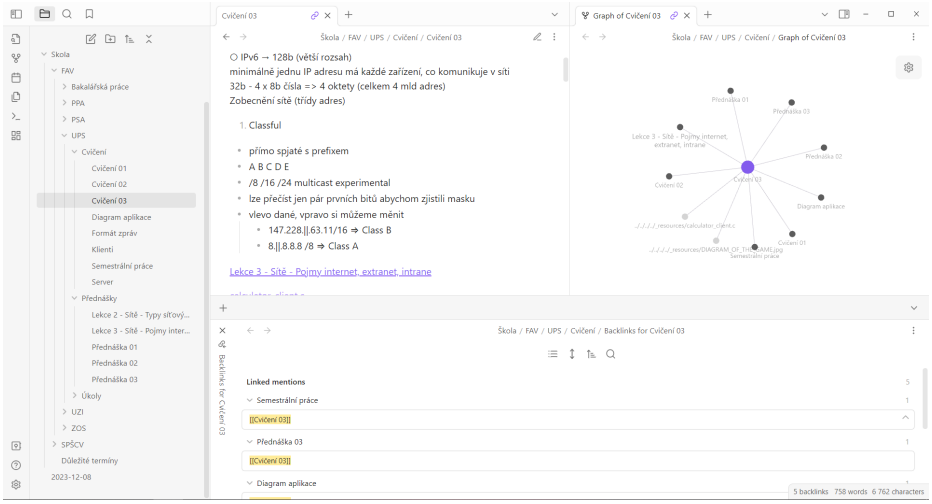
poskytují prostředí, kde uživatelé mohou spolupracovat na více projektech s více uživateli. Bezpečnost dat je zajištěna logováním, zálohováním a šifrováním. Uložená data jsou šifrována algoritmem AES-256 a TLS při přenosu dat. Pokud dojde k úniku dat, je o tom uživatel informován do 72 hodin.

Notion obsahuje opravdu velké množství funkcí a integrací s ostatními aplikacemi, čímž vytváří velmi produktivní pracovní prostředí. Mezi funkcemi nechybí označování štítky, vyhledávání, řazení, historie změn. Jednou z předností aplikace je nespočet různých typů obsahu, které může být uložen na jedné stránce. Jedním z nestandardních typů jsou databáze, které lze navzájem propojovat, používat v nich matematické funkce, automatizovat s nimi procesy a zobrazovat je v různých rozdílných zobrazení. Dále v obsahu stránky můžeme vytvořit tlačítka, kterým můžeme nadefinovat akce. V aplikaci je přítomná umělá inteligence pro vygenerování určitého textu, shrnutí bloků a vyhledávání. Importovat lze data několik formátů a aplikací. V aplikaci je přítomná kontrola českého pravopisu a gramatiky Česká lokalizace bohužel není dostupná. Bohatý počet integrací s dalšími nástroji a službami, ať už oficiálními nebo pomocí API (Application Programming Interface), poskytuje možnost propojit Notion s externími aplikacemi a rozšířit tak jeho již existující bohatou funkcionalitu.

2.5 Obsidian

Obsidian je software pro tvorbu a organizaci poznámek, který se odlišuje od konkurence propojováním poznámek přes odkazy a následné zobrazení vazeb formou grafu (viz obrázek 2.9). Aplikace je napsána v jazycích TypeScript a JavaScript s využitím frameworku Electron. Aplikace je dostupná na mobilních zařízeních a počítačích. Obsidian je bezplatný pro personální použití. Komerční užití je zpoplatněno 50 $/rok za každého uživatele. Další placené služby aplikace jsou Obsidian Sync a Obsidian Publish, které stojí 8$/měsíc. Obsidian Sync nabízí synchronizování dat do velikosti 10 GB a Obsidian Publish publikaci poznámek na webovou stránku. Poslední placenou položkou je Catalyst, pro předčasný přístup k novým verzím, komunitní odznaky a přístup k VIP kanálu za jednorázovou platbu 25 $. [5]

Aplikace umožňuje vytvářet textové poznámky a plátna (`canvas`), ve kterých nelze kreslit tužkou. Tato plátna zastupují funkci myšlenkových map. Do plátna lze vkládat textové karty, obrázky a kopie poznámek, které se dají jednoduše propojit šipkami. Do obsahu poznámek je možno vložit formátovaný text, obrázky, URL adresy, soubory a odkazy. Vytváření skic přímo v programu chybí. Poznámky i plátna se dají uspořádat do složek s neomezených počtem úrovní. Strom se soubory je zobrazen v levé části obrazovky (viz obrázek 2.10). Veškerou manipulaci se soubory, plátny a poznámkami lze provést pomocí přetažením příslušného souboru do požadovaného místa ve stromě. Pokud bychom chtěli využít jiný způsob, tak mů-

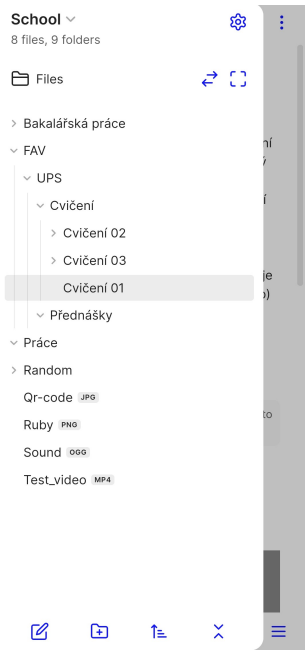


Obrázek 2.9: Počítačová verze aplikace Obsidian

žeme jednotlivé soubory přesouvat přes podrobnosti o souboru nebo v souborovém systému zařízení.

Obsidian ukládá všechna data ve fyzické paměti zařízení v souborech .md a .canvas, což je speciální formát Obsidianu obsahující JSON s informacemi o rozložení a obsahu karet. Místo, kam se data budou ukládat se nazývá trezor a uživatel si jej volí sám. Na místě, které si uživatel zvolí se vytvoří, pokud trezor ještě neexistuje, inicializační složka .obsidian s daty ve formátu JSON. Vzhledem k tomu, že jsou poznámky uložené v paměti zařízení, je možné poznámky upravovat kdykoliv i bez připojení k internetu, avšak automatická synchronizace je dostupná jen s placenou službou Obsidian Sync. Pokud chce mít uživatel data i na jiném zařízení, musí všechna data na toto zařízení překopírovat a v aplikaci pak vybrat správnou cestu k trezoru. Při zakoupení Obsidian Sync se pak poznámky synchronizují automaticky v pozadí aplikace a lze je sdílet s ostatními uživateli. V rámci synchronizace si můžeme vybrat, které soubory budou při synchronizování ignorovány a máme k dispozici historii až rok starých změn. Při synchronizování se data ukládají na server Obsidianu a šifrují se koncovým šifrováním s použitím algoritmu AES-256.

Obsidian nabízí podobně jako ostatní testované aplikace funkce vyhledávání, filtrování, řazení a navíc ještě slučování více poznámek do jedné a organizaci do bloků. Jedinečnou funkcí Obsidianu je graficky zobrazený graf, který zobrazuje závislosti mezi poznámkami. Uživatel si tak může vytvořit vlastní sémantickou síť nebo myšlenkovou mapu. Pro rozšíření základní funkcionality a vyladění pracovního prostředí podle svého vkusu, mají uživatelé možnost si vybrat některé z tisíce rozšíření. Pokud by uživatel nenašel vhodný vzhled nebo rozšíření, které potřebuje, má možnost se podívat do dokumentace API, ve které je podrobně popsáno, jak si vytvořit vlastní rozšíření nebo upravit vzhled aplikace.



Obrázek 2.10: Hierarchie v mobilní aplikaci Obsidian

2.6 Srovnání vybraných aplikací

V rámci analýzy byl prozkoumán trh s aplikacemi pro tvorbu poznámek. Bylo vyzkoušeno a otestováno pět z nich. Jak již bylo řečeno v úvodu, na trhu existuje opravdu spoustu aplikací, které se zaměřují na tvorbu poznámek. K dalším aplikacím patří například Google Keep, která umožňuje vytváření poznámkových lístečků. Dále můžeme zmínit poznámkové aplikace Apple's Notes, Ulysses a Bear vytvořené primárně pro zařízení společnosti Apple a aplikace Confluence, Slite a Coda určené především pro dokumentaci a kooperaci ve firemním prostředí.

Při zkoušení jednotlivých aplikací jsme si mohli všimnout jejich odlišností, pozitiv, negativ a speciálních funkcí. Jednou z odlišností byla dostupnost na platformách. Z vybraných aplikací byly všechny dostupné na Android a poznámky se daly zobrazit a upravit i na počítači. Přehled o dostupnosti na platformách je uveden v tabulce 2.1.

Tabulka 2.1: Tabulka dostupnosti aplikací na platformách

	Android	iOS	Windows	Linux	macOS	Web
Evernote	✓	✓	✓	✓	✓	✓
OneNote	✓	✓	✓	✗	✓	✓
Joplin	✓	✓	✓	✓	✓	✗
Notion	✓	✓	✓	✗	✓	✓
Obsidian	✓	✓	✓	✓	✓	✗

Z pohledu finanční strategie jsme si mohli všimnout dvou modelů. Některé aplikace využívaly freemium marketingového plánu, ve kterém je základní verze aplikace s omezenou funkcionalitou zdarma a další balíčky s více funkcemi se musí měsíčně platit. Další aplikace naopak preferovaly strategie bezplatné aplikace, kde jsou placené jen některé služby nebo velikost úložiště, bez kterého se v některých případech lze bez problémů obejít.

Z hlediska organizace jsme si mohli všimnout štítků, shlukování do skupin, tvoření zkratek a hierarchické struktury. Hierarchie byla rovněž implementována různými způsoby. V některých případech jsme se setkali s omezenou hloubkou, kde každý typ souboru měl stanovenou úroveň, ve které se může nacházet, což může být celkem omezující. Jiné aplikace naopak umožňovaly hloubku neomezenou. Typy souborů, které se objevují v hierarchii, byly též velmi různorodé. Hierarchie byla nejčastěji zobrazena stromovou strukturou a seznamem. V některých případech jsme se setkali se spojením poznámek pomocí odkazů. Přesouvání je nejčastěji implementováno přesouváním drag-and-drop nebo zvolením cesty ze seznamu.

Další odlišnosti aplikací bylo uložení uživatelských dat a přístup k datům dat bez připojení k internetu. Pro synchronizaci bývá využito buď cloudového úložiště nebo serveru. Při sdílení dat mezi více uživateli, může vlastník nastavit práva dalším uživatelům. Data jsou nejčastěji šifrována algoritmem AES-256. Zabezpečení dat při přesunu bylo často implementováno koncovým šifrováním. Ve větších společnostech se ve většině případů využívá kombinace moderních standardů TLS, HTTPS, HSTS a SSL. Funkcionalita aplikací bývá také velice odlišná. Některé aplikace nabízí pouze základní funkce s možností instalace některých rozšíření. Další mají v základní verzi nespočet funkcionalit, ze kterých běžný uživatel využije jen malé procento. Srovnání nabízených funkcí je uvedeno v tabulce 2.2

Tabulka 2.2: Shrnutí funkcí jednotlivých aplikací

	Evernote	OneNote	Joplin	Notion	Obsidian
Synchronizace	✓	✓	✓	✓	$
Sdílení poznámek	✓	✓	$	✓	$
Offline přístup	$	✓	✓	✗	✓
Ukládání verzí	✓	✓	✓	✓	$
Úložný prostor	60 MB[8]	5 GB	∞[9]	∞	∞
Max počet úrovní	3	3	∞	∞	∞

[8] Maximální velikost nahraných dat za měsíc
[9] Omezeno fyzickými parametry zařízení

2.7 Shrnutí

Zjistili jsme, jak vypadají a fungují vybrané aplikace pro tvorbu poznámek. Z hlediska vývoje naší aplikace je vhodné se inspirovat dobře navrženými funkcemi a zároveň se vyhnout nedostatkům, které by mohli by uživatele limitovat. Vyvíjená aplikace bude nabízet vytváření poznámek s neomezenou[10] úrovní zanoření, aby bylo například umožněno organizování výpisků z přednášek dle ročníku, předmětu, a rozlišovat mezi přednáškami a cvičeními. Aplikace bude také umožňovat synchronizaci poznámky mezi různými zařízeními. Poznámky by mělo, být možné synchronizovat i mezi počítačem a mobilním zařízením. Je důležité, aby byla aplikace bezplatná a jednoduše rozšiřitelná. V kapitole 5 tyto nápady zohledníme a provedeme detailnější návrh specifikace aplikace.

[10] Počet úrovní bude limitován fyzickými parametry zařízení.

Technologie pro vývoj mobilních aplikací — 3

V této kapitole se podíváme na technologická řešení a nástroje, které lze využít pro tvorbu mobilních aplikací. Prozkoumáme různé přístupy vývoje a jejich prostředky. Vyhodnotíme vlastnosti, přednosti a omezení každé technologie. Na závěr kapitoly vybereme technologii, která bude využita pro tvorbu naší aplikace.

3.1 Vývoj mobilních aplikací

Existuje více způsobů vývoje mobilních aplikací. V základu můžeme typy vývoje rozdělit na nativní a multiplatformní vývoj. Multiplatformní vývoj pak můžeme dále rozdělit na nativní multiplatformní, hybridní a PWA (Progressive Web Apps). Hlavním rozdílem mezi nativním a multiplatformním vývojem je cílení aplikace na jednu konkrétní platformu s čímž se pojí vyšší optimalizace u nativních aplikací. Multiplatformní vývoj má obvykle jeden zdrojový kód pro více platforem, což zkracuje dobu vývoje a snižuje náklady aplikací vyvíjených pro více platforem. [6]

3.2 Nativní vývoj

Nativní vývoj představuje proces, při kterém vývojáři vyvíjejí aplikace pro konkrétní platformu nebo zařízení. Pokud tedy budeme vyvíjet aplikaci pro Android a iOS, budeme muset vytvořit dvě samostatné verze aplikace. Výhodou nativního vývoje je možnost plně využít funkcí, možností a předností daného operačního systému. To vede k lepšímu výkonu, optimalizaci, bezpečnosti, plnému přístupu k hardwarovým funkcím (GPS, fotoaparátu, mikrofonu a další), typickému vzhledu a chování pro daný operační systém. Nevýhodou je již zmíněný vývoj pouze na jednu platformu a s tím spojený delší čas vývoje, náklady a nutnost znát více technologií. Nativní vývoj se upřednostňuje zejména u aplikací, ve kterých je důležitý vysoký výkon, časté využívání hardwarových funkcí a pokročilé interakce s uživatelem. [7] [8]

Pro vývoj aplikací pro platformy iOS se nejčastěji používají programovací jazyky Objective-C a Swift, společně s využitím vývojového prostředí Xcode. Na rozdíl od platformy Android, má operační systém iOS neveřejný zdrojový kód, vytvořený výhradně pro hardware společnosti Apple. Pro vývoj pro platformu iOS je tedy nezbytné použití zařízení Mac. Po dokončení vývoje lze aplikaci distribuovat prostřednictvím obchodu App Store. [9]

Aplikace pro platformu Android se nejčastěji vyvíjí v jazycích Java a Kotlin, přičemž od roku 2017 je Kotlin společností Google označen jako oficiální jazyk pro vývoj aplikací na platformu Android[1]. Pro vývoj je doporučováno používat Android Studio. Používání tohoto vývojového prostředí není povinné, avšak v nabídce je široká škála funkcionalit zahrnující návrh uživatelského rozhraní, kompilaci zdrojového kódu a v neposlední řadě emulátory mobilních zařízení, které dokážou vývojářům značně zjednodušit vývoj. Aplikace je možno vydávat v obchodě Google Play, odkud si ji mohou stáhnout nebo zakoupit ostatní uživatelé. [10] [9]

3.2.1 Kotlin

Kotlin je open-source[2] programovací jazyk, vyvinutý společností JetBrains. Jedná se o staticky typovaný a objektově orientovaný programovací jazyk, který je spustitelný na JVM (Java Virtual Machine), s plnou podporou knihovních tříd Java a platformou Android. Původně byl Kotlin navržen s cílem poskytnout vývojářům jednodušší a stručnější syntaxi ve srovnání s jazykem Java. V současné době je Kotlin určený především pro vývoj mobilních aplikací. Díky své univerzálnosti lze však Kotlin použít též k vytváření serverů, webových stránek nebo aplikací pro datové analýzy. [11] [12]

Oproti jazyku Java představuje Kotlin mladší programovací jazyk, který nabízí moderní funkce, stručnou a čitelnou syntaxi. Kotlin zavádí koncept takzvaných datových tříd, které jsou primárně určeny k uchovávání dat. S použitím této deklarace jsou třídy automaticky vybaveny metodami `equals()`, `hashCode()`, `toString()` a `copy()`. Gettery a settery jsou nahrazeny přístupem k vlastnosti viz zdrojový kód 3.1.

Zdrojový kód 3.1: Definování třídy v jazyce Kotlin

```
1 data class MyClass(private var myField: Int) {
2     fun getMyField() = myField
3     fun setMyField(value: Int) { myField = value }
4 }
```

V Kotlinu rozlišujeme dva hlavní typy proměnných `val` a `var`, kterým nemusíme explicitně zadávat datový typ. Tyto proměnné se liší dle toho, zda lze hodnotu

[1] https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/
[2] https://github.com/JetBrains/kotlin

po nadefinování změnit. Proměnná `val` definuje hodnotu, která nemůže být změněna, zatímco `var` definuje proměnnou, jejíž hodnota může být při běhu programu změněna. Kotlin při definici proměnné nabízí funkci „Null safety“, která má za cíl zabránit chybě `NullPointerException`, která je častým problémem při vývoji Java aplikací. Vývojáři mohou touto funkcí explicitně deklarovat, zda proměnná může obsahovat hodnotu `null`, pomocí znaménka otazníku viz zdrojový kód 3.2.

Zdrojový kód 3.2: Proměnné v jazyce Kotlin

```
val title = "Kotlin" // Proměnnou po definování nelze změnit
var title = "Kotlin" // Proměnná, kterou lze v průběhu změnit
var title: Any = "Kotlin" // Proměnná může být libovolný
    datový typ
var name: String = "Kotlin" // Proměnná nemůže být null
var name: String? = null // Proměnná může být null
```

Dalším významným rozdílem oproti Javě je podpora funkcionálního programování. Kotlin podporuje lambda výrazy, funkce vyššího řádu a rozšiřující funkce viz zdrojový kód 3.3. Kotlin rovněž podporuje korutiny, které poskytují jednodušší realizaci asynchronního programování oproti složitější práci s vlákny v Javě. Kotlin nevyžaduje ošetření výjimek, zpracovává primitivní typy jako objekty, dále můžeme najít mnoho dalších vylepšení oproti Javě. [13] [14] [15]

Zdrojový kód 3.3: Funkcionální programování v jazyce Kotlin

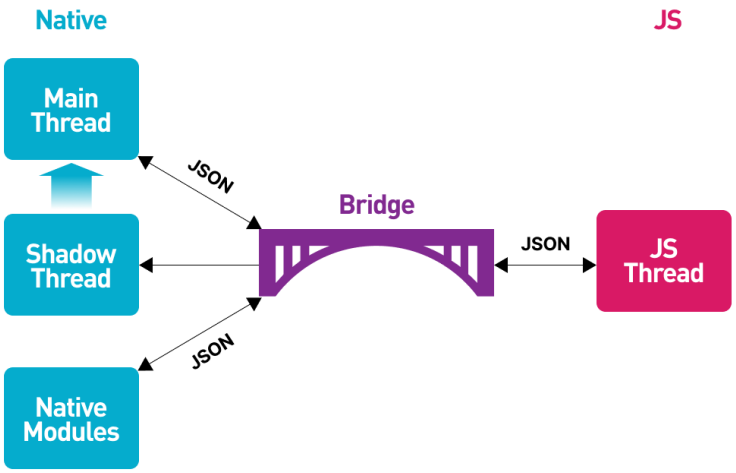
```
// Lambda výrazy
val list = listOf(1, 2, 3)
// Zdvojnásobení každého prvku v seznamu
val doubledList = list.map { it * 2 }

// Funkce vyššího řádu, která přijíma dva celočíselné
    argumenty a jednu funkci
fun higherOrderFunc(x: Int, y: Int, f: (Int, Int) -> Int):
    Int {
    return f(x, y)
}
// Provedení sečtení dvou čísel
val result = higherOrderFunc(3, 4) { x, y -> x + y }

// Rozšiřující funkce umožňuje existující třídy bez nutnosti
    dědění
fun Int.isEven() = this \% 2 == 0
val isFourEven = 4.isEven()
```

3.3 Nativní multiplatformní vývoj

Multiplatformní vývoj využívá technologie, ve kterých lze napsat část nebo dokonce celý kód pro více platforem. Tento kód je následně zkompilován do nativní verze kódu pro různé platformy. V tomto aspektu se odlišují od hybridních aplikací. Vzhledem k tomu, že se využívá nativního vykreslování, je optimalizace srovnatelná s nativním vývojem. Výkon však záleží na frameworku, který se vývojář rozhodne použít. Mezi populární nativní multiplatformní frameworky patří React Native, Flutter a .NET MAUI. [6] [16]

3.3.1 React Native

React native je open-source framework umožňující vyvíjet mobilní aplikace s použitím JavaScriptu a Reactu. Tento framework vznikl v létě 2013 jako interní projekt společnosti Facebook v rámci hackathlonu. V březnu 2015 pak společnost Facebook zveřejnila zdrojový kód na platformě GitHub[3] [17]. React Native obsahuje předdefinovanou sadu komponent, jako jsou například View, Text nebo Image, které se přímo mapují na nativní komponenty platformy. Tyto komponenty lze též kombinovat a sestavovat tak komplexnější komponenty, které lze použít v aplikaci. [18] [19]



Obrázek 3.1: Architektura bridge v React Native [20]

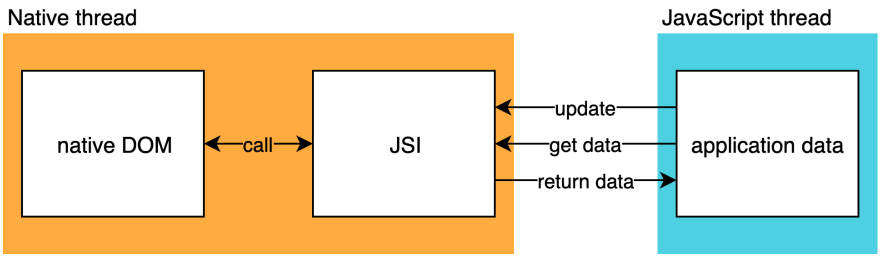
React Native se na rozdíl od hybridních frameworků nespoléhá na webové zobrazení (WebView), ale na skutečné komponenty a přístup k API poskytnutý nativními platformami. Z počátku React Native používal koncept „mostu", který umožňoval

[3] https://github.com/facebook/react-native

asynchronní komunikaci mezi prvky JavaScriptu a nativními prvky viz obr. 3.1. Logika mostu je zde stejná jako v případě webových aplikací, kdy o sobě vrstvy frontendu a backendu nemusí nic vědět, ale musí spolu umět komunikovat. Pro vzájemnou komunikaci se používá zpráv ve formátu JSON. Samotný React Native závisí na čtyřech vláknech:

- **Hlavní vlákno:** jedná se o vlákno, ve kterém je spuštěna aplikace. Toto vlákno je zodpovědné za interakci s uživatelem a vykreslování uživatelského rozhraní,
- **JavaScriptové vlákno:** je zodpovědné za provádění logiky aplikace,
- **Stínové vlákno:** je vlákno, které je spuštěné společně s JavaScript vláknem. Jeho úkolem je vypočítat pozice rozvržení (View) a zkonstruovat strom komponent (Layer),
- **Vlákno nativních modulů:** stará se o přístup k API dané platformy.

Od verze 0.68 je možné používat novou architekturu React Native, která upouští od mechanismu mostu ve prospěch JavaScriptového rozhraní. JSI (JavaScript Interface) je odlehčená univerzální vrstva, která může navázat přímé spojení s nativními rozhraními API viz obr. 3.2. To umožňuje synchronní komunikaci mezi JavaScriptem a nativními prvky, což vede ke zvýšení výkonu. [19] [21]



Obrázek 3.2: Architektura JSI v React Native [20]

Framework React Native nabízí některé výhody, mezi které například patří:

- **Společný kód:** Využití stejné kódové základny pro různé platformy urychluje vývoj, zkracuje dobu uvedení na trh a usnadňuje údržbu. Stejný kód lze využít pro webové a mobilní aplikace, pokud používají React Native,
- **Rozsáhlá komunita:** React Native má rozsáhlou aktivní komunitou vývojářů. Díky této komunitě existuje velké množství knihoven, nástrojů, komponent a návodů. Rozsáhlá komunita může zrychlit řešení problému, protože je vyšší šance, že někdo narazil na stejný problém,

- **Výkon:** V porovnání s Hybridními aplikacemi poskytuje React Native výkon podobný nativnímu,
- **Aktualizace:** React Native je populární framework, který dostává pravidelné aktualizace a vylepšení, které společnost Facebook vydává,
- **Jazyk:** Při implementaci se používají populární jazyky JavaScript a TypeScript, které se používají i u webových aplikací,
- **Hot Reloading:** Tato funkce umožňuje vývojářům vidět provedené změny bez nutnosti znovu kompilovat kód. [19]

Mezi nevýhody React Native naopak patří:

- **Mladá technologie:** Jedná se o relativně mladou technologii, která má jistá omezení, chyby a problémy, které je třeba vyřešit. Některé moduly ve frameworku neexistují, což znamená, že vývojáři mohou potřebovat více času na vytvoření vlastních modulů,
- **Animace a interakce:** React Native není přizpůsobený na vytváření komplexních animací a interakci se složitými gesty, důvodem je JavaScriptová část aplikace,
- **Výkonnost:** Přestože je výkon React Native lepší než u hybridních aplikací, stále je zde rozdíl mezi nativním řešením.

Co se týče komponent, má vývojář na výběr z oficiálních komponent, které jsou uvedeny v dokumentaci React Native[4], nebo využít jedné z mnoha komunitních knihoven, díky rozsáhlé komunitě, kterou React Native má. Poslední možností je vytvořit si vlastní komponenty.

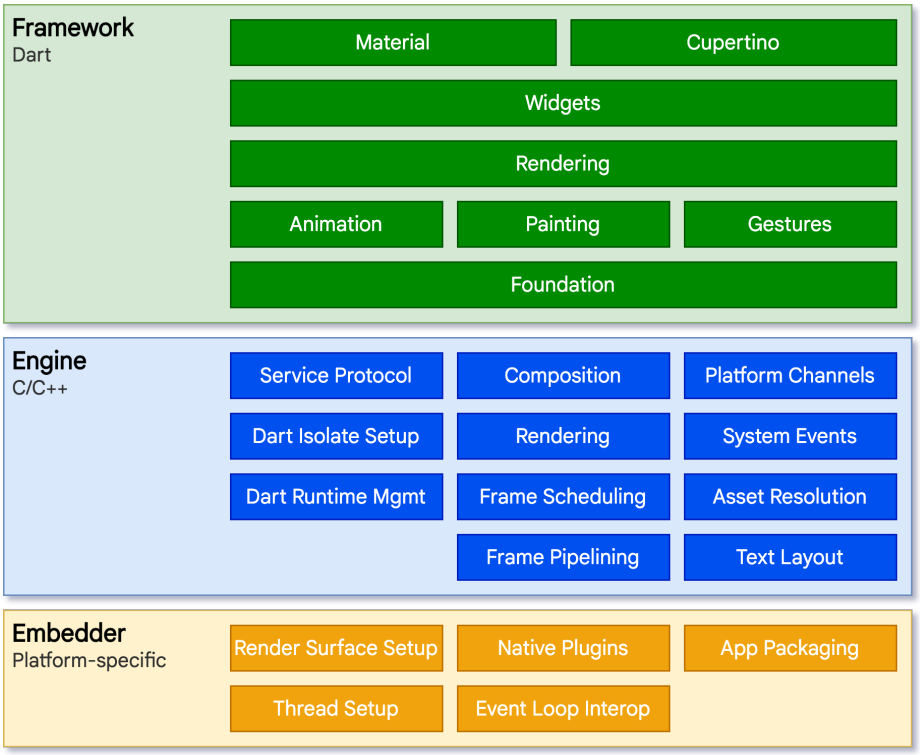
3.3.2 Flutter

Flutter je open-source technologie společnosti Google pro vytváření mobilních, desktopových a webových aplikací. Jedná se o celkem mladou technologii, který byl oficiálně jako projekt spuštěn v roce 2017 a následně v prosinci 2018 byla zveřejněna první stabilní verze. Pomocí Flutteru je vytvořeno mnoho známých aplikací. Jedná se například o eBay, Alibaba.com, Google Classroom a Google Pay. [22] [23]

Flutter není programovací jazyk, framework ani knihovna, ale kompletní SDK (Software Development Kit) obsahující vykreslovací jádro, widgety, testovací a integrační API a mnohé další viz obr. 3.3. Toto SDK je tvořeno třemi hlavními částmi: [24] [23]

[4] https://reactnative.dev/docs/components-and-apis

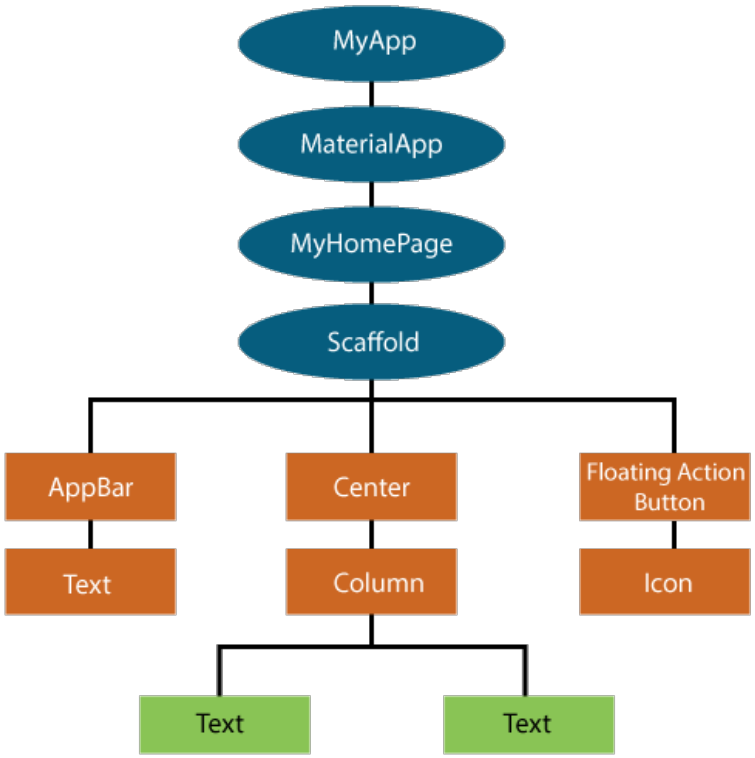
- **Embedder** používá jazyk specifický pro danou platfromu a zajišťuje bezproblémový běh aplikace na různých operačních systémech,
- **Engine**, napsaný v C/C++, poskytuje nízkoúrovňovou implementaci základních rozhraní API aplikace Flutter, které se stará o vykreslování vizuálních prvků vždy, když se něco změní. Zároveň také zajišťuje veškerou základní funkcionalitu, mezi kterou patří kompilace jazyku Dart, rozložení textu, systémové eventy, vstupy/výstupy a další,
- **Framework** je založen na programovacím jazyku Dart. Jedná se o bohatou sadu knihoven a funkcí, které Flutter poskytuje vývojářům.



Obrázek 3.3: Architektura technologie Flutter [22]

Pro vývoj aplikací se využívá jazyk Dart. Jedná se o objektově orientovaný jazyk, který svou syntaxí připomíná jazyk Java nebo C++. Na rozdíl od jiných frameworků (např. React Native) nepotřebuje Dart žádný mezilehlý krok k přeložení a propojení nativního zobrazení platformy. Dart podporuje dva způsoby kompilace AOT (Ahead Of Time) a JIT (Just In Time). AOT kompiluje zdrojový kód do nativního před spuštěním aplikace. JIT překládá zdrojový kód až v okamžiku, když je aplikace spuštěna, čehož se využívá hlavně při vývoji, kdy vývojář vidí změny které provádí. [23] [24]

Ve Flutteru je vše složeno z widgetů. Jedná se o základními stavebními prvky uživatelského rozhraní, které se do sebe dají vkládat a tím tvořit složité widgety. Při vytváření uživatelského rozhraní se definuje strom widgetů viz obr. 3.4, který popisuje, jak vypadá vzhled naší aplikace. Widgety kromě vzhledu aplikace také zohledňují její aktuální stav. Flutter zavádí dvě hlavní třídy widgetů, a to bezstavové (`Stateless`) a stavové (`Statefull`). Widgety, které nemění v průběhu své existence svůj stav (ikony, popisky, tlačítka a další) jsou definovány `Stateless` widgety. Pokud se vlastnosti a stav widgetu v průběhu běhu aplikace mění na základě interakce s uživatelem nebo jiných faktorů, jedná se o `Statefull` widget. Oba typy widgetů obsahují metodu pro překreslení build, která překresluje obsah při změně.



Obrázek 3.4: Widget tree [25]

Mezi výhody Flutteru patří: [26] [24]

- **Společný zdrojový kód:** Flutter je multiplatformní, umožňuje vytvářet aplikace pro více platforem zároveň. Flutter podporuje celkem sedm platforem: Android, iOS, Windows, Linux, macOS, web a Google Fuchsia (operační systém vyvíjený společností Google),
- **Rychlý vývoj uživatelského rozhraní:** Flutter poskytuje bohatou sadu komponent, které umožňují vytvářet přizpůsobitelná uživatelská rozhraní za krátkou dobu,

- **Stálá podpora:** Společnost Google garantuje dlouhodobou podporu technologie Flutter,
- **Vysoký výkon:** Flutter využívá přímé kompilace díky tomu je výkon Flutteru srovnatelný s nativními aplikacemi,
- **Hot reload:** Změny, které provede vývojář v kódu se ihned promítnou v aplikaci, což vede ke zvýšení produktivity programování,
- **Snadná internacionalizace:** Flutter nativně podporuje widgety založené na knihovně intl[5], která zjednodušuje proces internacionalizace. V současné době je podporováno 78 jazyků, různé měny, měrné jednotky, zobrazování textu zprava doleva a mnohá další specifika lokalizací,
- **Rychlé učení:** Naučit se jazyk Dart by neměl být velký problém vzhledem k rozsáhlé dokumentaci a stále rostoucí komunitě. Začínající programátor nemusí mít velké zkušenosti s programováním mobilních aplikací.

Mezi nevýhody Flutteru můžeme zařadit:

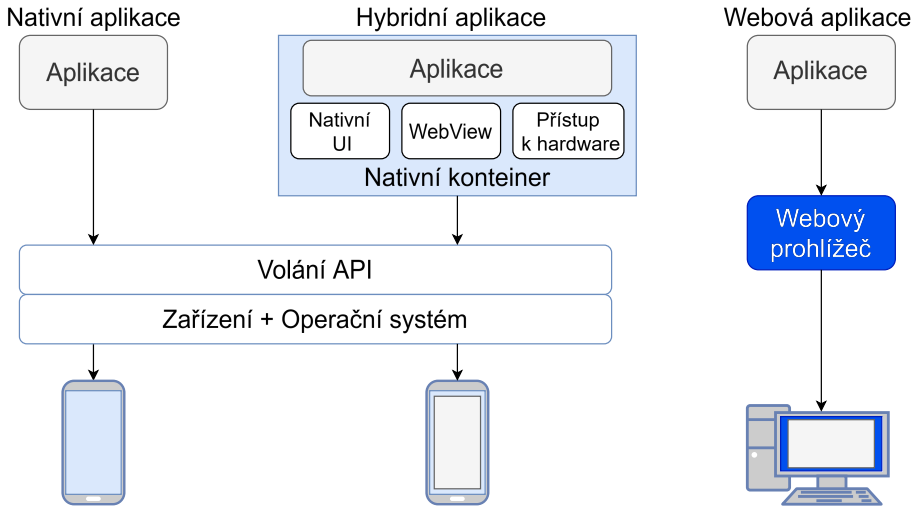
- **Nedostatek knihoven třetích stran:** Pro Flutter je vytvořeno méně komunitních knihoven než je tomu třeba u React Native nebo jiných starších technologií,
- **Velikost aplikace:** Vzhledem k tomu, že má Flutter vestavěné widgety, minimální velikost aplikace přesahuje 4 MB, což je více než u nativních aplikací,
- **Rozšíření jazyku Dart:** Z hlediska popularity Dart zaostává například za JavaScriptem, Javou nebo Kotlinem. Vývojáři, kteří se rozhodou vytvořit Flutter aplikaci jsou nuceni se naučit nový programovací jazyk, který s velkou pravděpodobností nepoužijí v žádné další technologii.

Při hledání vhodných komponent má vývojář možnost zkusit najít vhodné widgety v katalogu s widgety[6], kde lze najít například widgety Material design pro Android nebo Cupertino pro iOS. Pokud zde vývojář nenalezne widgety, které potřebuje, má možnost zkusit prohledat stránku pub.dev[7], což je oficiální stránka s Flutter knihovnami. V případě, že ani zde není to, co vývojář potřebuje, lze ještě zkusit najít požadované knihovny na platformě GitHub, webové stránce Flutter Awesome a dalších online zdrojích.

[5] https://pub.dev/packages/intl
[6] https://docs.flutter.dev/ui/widgets
[7] https://pub.dev/

3.4 Hybridní vývoj

Hybridní aplikace kombinují prvky webových aplikací s prvky nativních aplikací. Aplikace se vyvíjejí s využitím standardních webových technologií HTML, CSS (Cascading Style Sheet) a Javascriptu. Na rozdíl od webových aplikací, které fungují přímo ve webovém prohlížeči, se hybridní aplikace spouští jako nativní, využívající vlastní vestavěný prohlížeč WebView (Android) nebo WKWebView (iOS). Aplikace je také zabalena do nativního kontejneru viz obr. 3.5, který umožňuje přistupovat k funkcím daného operačního systému skrze pluginy.



Obrázek 3.5: Porovnání nativní, hybridní a webové aplikace (překresleno z [27])

Hybridní aplikace přinášejí výhodu jedné kódové základny, která snižuje náklady, údržbu a zaručuje stejnou funkcionalitu napříč zařízeními. Aby však aplikace efektivně komunikovala se systémovými API a konzistentně fungovala na dané platformě, je často nutné napsat část nativního kódu. Tato implementace může vést k rozdílům v uživatelském rozhraní a ovládání aplikace na různých platformách. Hlavní nevýhodou hybridních aplikací je nižší výkon ve srovnání s nativními aplikacemi, jakožto důsledek závislosti na vestavěném prohlížeči. Mezi hybridní technologie se řadí Ionic a Apache Cordova. [28] [29]

3.4.1 Ionic

Ionic framework je open-source[8] sada nástrojů určená pro vývoj hybridních mobilních aplikací. První verze byla publikována v roce 2013 a byla postavena na technologiích Angular a Apache Cordova. V dnešní době tento framework poskytuje sadu

[8] Zdrojový kód Inonic zveřejněn zde: https://github.com/ionic-team/ionic-framework

webových komponent, které umožňují uživatelům vyvíjet aplikaci ve frameworku Angular, React nebo Vue.js. [30]

Mezi výhody frameworku Ionic patří nezávislost na platformě a s tím spojený společný kód, bohatá nabídka komponent a pluginů, možnost vyvíjet v prohlížeči, dobře napsaná dokumentace a podpora komunity. Mezi nevýhody můžeme zařadit nižší výkon ve srovnání s nativními technologiemi a bezpečnostní problémy, musí se řešit specifické bezpečnostní rizika všech platforem, které však lze řešit správnými bezpečnostními opatřeními. [31]

Ionic poskytuje velké množství optimalizovaných komponent navržených pro mobilní zařízení. Tyto komponenty mohou být dále přizpůsobeny pro každou platformu pomocí `Platform Styles`, ve kterých má každá platforma nastavený typický vzhled a chování. Pokud se pokusíme najít vhodné komponenty pro náš projekt, máme na výběr z velkého množství oficiálních komponent, které jsou uvedeny v dokumentaci[9]. Stromová struktura bohužel v seznamu oficiálních komponent chybí, avšak v Ionic Market place[10], ve kterém můžeme najít bezplatné i zpoplatněné šablony, motivy, komponenty a pluginy navržené pro vývoj Ionic aplikací, lze najít několik implementací této struktury. Další možností pro hledání užitečných komponent a knihoven je například server GitHub, knihovny npmjs a další webové stránky jako syncfusion[11], určené k nahrání komponent.

3.5 Progresivní webové aplikace

PWA je typ webové aplikace, který může fungovat jako webová stránka i mobilní aplikace. Tyto aplikace se nedají považovat za plnohodnotné mobilní aplikace, neboť jde jen o obal webového prohlížeče, který tvoří technologie Service Workers a JSON manifest:

- **Service Workers:** jsou skripty, které umožňují práci s mezipamětí. To umožňuje uživatelům práci i bez připojení k internetu a současně přispívá k rychlejšímu běhu aplikace,
- **JSON manifest:** Obsahuje základní informace, jako je název aplikace, použité ikony nebo barvy a zajišťuje přístup k hardware. Jeho cílem je zvýšit zážitek podobný nativní aplikaci.

PWA se chovají a vypadají stejně jako běžné webové stránky, které poskytují responzivní design a lze je vyhledat v internetovém prohlížeči. Navíc bývají rychlejší

[9] https://ionicframework.com/docs/components
[10] https://market.ionicframework.com
[11] https://www.syncfusion.com

než klasické webové stránky, mají možnost fungovat i bez internetového připojení a lze je přidat na domovskou obrazovku přímo z prohlížeče. Stejně jako mobilní aplikace mohou být publikovány v obchodech s mobilními aplikacemi App Store a Google Play. Přestože PWA mohou využívat hardwarové funkce, tento přístup nemusí být ke všem mobilním senzorům, aplikace nemusejí mít nativní vzhled a v případě iOS může nastat problém s instalací aplikace na domovskou obrazovku. Zdrojový kód je jednotný pro všechna zařízení, což výrazně usnadňuje a zrychluje vývoj pro různé operační systémy. Příkladem PWA je například Pinterest, Spotify, Uber, X (bývalý Twitter) nebo TikTok. [32] [33]

3.6 Shrnutí

V rámci kapitoly technologií pro vývoj mobilních aplikací jsme prozkoumali různé způsoby a technologie používané pro vytváření mobilních aplikací. S ohledem na charakteristiky aplikace pro vytváření poznámek, je praktické, aby byla aplikace přístupná z více typů zařízení, umožňující uživatelům prohlížet a upravovat poznámky jak na mobilním zařízení, tak na počítači. Dále též v případě změny operačního systému z Android na iOS, je dobré, aby uživatelé mohli dále spravovat bez problémů své poznámky. Aplikace též nepotřebuje složité animace a velký výpočetní výkon. Z těchto důvodu jsem se rozhodl při vývoji využít nativní multiplatformní vývoj.

Při rozhodování mezi nativními multiplatformními technologiemi jsem se rozhodl upřednostnit Flutter. Hlavními faktory mého rozhodnutí byla rostoucí popularita, což znamená vyšší použitelnost technologie v budoucnu, a vysoký výkon v porovnání s ostatními multiplatformními technologiemi. Flutter umožňuje kompilaci do nativního kódu, tím se eliminuje mezikrok komunikace s nativními komponentami. Další důvod rozhodnutí je technologie widgetů, která nabízí unikátní způsob, který vývojářům zjednodušuje vytváření uživatelského rozhraní.

Ukládání dat 4

V této kapitole se zaměříme na různé způsoby a technologie ukládání dat mobilních aplikací. Začátek kapitoly se zabývá ukládáním dat do lokálního úložiště mobilního zařízení. Další část kapitoly se věnuje technologiím, které lze využít pro synchronizaci dat. Tato část analyzuje nejdříve možnosti serverových a cloudových úložišť. V závěru kapitoly jsou vybrány technologie, které budou použity při vývoji aplikace.

4.1 Lokální ukládání

Jedna z klíčových funkcionalit vytvářené aplikace je umožnění práce s daty bez připojení k internetu. K dosažení této vlastnosti budeme muset ukládat uživatelská data do lokálního úložiště zařízení. Bez lokálního ukládání by všechna data po ukončení práce s aplikací zmizela a uživatel by se při opětovném spuštění aplikace setkal pouze s inicializačními daty. V následující části budou popsány možnosti lokálního ukládání dat na mobilním zařízení.

Ukládání dat aplikace na mobilním zařízení funguje rozdílně, než je tomu u desktopových aplikací. Android poskytuje několik základních možností ukládání:

- **Vnitřní úložiště:** Soubory, které se ukládají do soukromého adresáře aplikace. Tato data jsou určena pro použití konkrétní aplikací a jsou smazána společně s odinstalováním aplikace,
- **Externí úložiště:** Uložení dat do veřejné části paměti. Může se jednat o SD (Secure Digital) kartu i pevně zabudovanou paměť v zařízení. Typicky to jsou fotky, videa, zvukové nahrávky a dokumenty, které byly vytvořeny v aplikaci,
- **Shared preferences:** Uložení soukromých dat základních primitivních typů ve formě klíč-hodnota,
- **Databáze:** Uložení dat do databáze s pomocí některé z knihoven (např. SQLite, Hive, Isar Database). [34] [35]

V další části se podíváme na použitelné knihovny pro ukládání lokálních dat ve Flutter aplikacích.

4.1.1 SharedPreferences

SharedPreferences je nejjednodušší forma ukládání pro ukládání jednoduchých dat aplikace. Tato třída umožňuje ukládání malého množství jednoduše strukturovaných dat ve formě perzistentní mapy s páry klíč-hodnota. Pro Flutter je vytvořena knihovna `shared_preferences`, která obaluje úložiště specifické pro konkrétní platformy (např. SharedPreferences pro Android, NSUserDefaults pro iOS a macOS). Knihovna je kompletně napsána v jazyku Dart a je kompatibilní s platformami Android, iOS, Linux, macOS, Windows a webovými aplikacemi. Podporované datové typy jsou `int`, `double`, `bool`, `String` a `List<String>`. [35] [36]

4.1.2 SQLite

SQLite je malý a rychlý SQL (Structured Query Language) databázový engine, který je napsán v jazyce C. Jedná se o nejčastěji používanou relační databázi na světě, která je integrována do mobilních zařízení, počítačů a mnoha běžně používaných aplikací. Kompletní SQL databáze má podporu funkcionalit klasických databázových serverů, jako je podpora více tabulek, indexování, podpora triggerů a vytváření pohledů. Veškerá data této databáze jsou uložena v jednom souboru. Pro Flutter je vytvořena knihovna `sqflite`[1], která je kompletně přepsána do jazyka Dart. [37] [35]

4.1.3 Hive

Hive je rychlá a nenáročná NoSQL databáze ukládající data ve formátu klíč-hodnota, která je kompletně napsána v jazyce Dart. Tato databáze ukládá data s možností zabezpečení dat šifrovacím algoritmem AES-256. [38]

Všechna data, uložena knihovnou Hive, jsou organizována do boxů, což je obdoba tabulky v SQL. Aplikace může mít libovolný počet boxů. Před použitím se musí box otevřít. Při otevření klasického boxu se načtou všechna data z lokálního úložiště do paměti pro okamžitý přístup. Pokud máme aplikaci, která uchovává větší množství dat, lze využít líného načítání dat, které si zapamatuje jen všechny klíče a adresy, kde jsou data jednotlivých klíčů uložena. Boxy mají operace pro zapisování, čtení a mazání dat (viz zdrojový kód 4.1). [39]

Zdrojový kód 4.1: Operace s Hive boxy

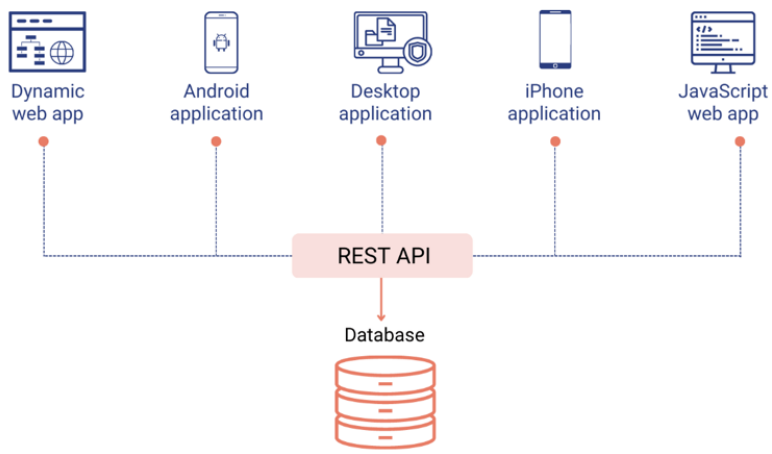
```
1 T value = box.get('key');     // Čtení dat
2 box.put('key', 'value');      // Ukládání dat
3 box.delete('key');            // Mazání dat
```

[1] https://pub.dev/packages/sqflite

Hive podporuje kromě primitivních a složených datových typů (String, DateTime, List, Map) také objekty vytvořené uživatelem. V případě uživatelem vytvořených objektů je potřeba registrovat `TypeAdapter`, který převede objekt do binární podoby. Type adapter lze automaticky vygenerovat, pokud v definici našeho objektu doplníme anotace `@HiveType` na začátku třídy a `@HiveField`, které doplníme k atributům třídy. Automatické vygenerování adaptéru pak provedeme příslušným příkazem.

4.2 Serverové úložiště

Na serverech lze vytvářet databáze, ve kterých se mohou uchovávat data. Při vytváření serveru lze zvolit mezi pořízením vlastního serveru nebo využitím služeb hostování serverů od externích poskytovatelů. Vytvoření vlastního serveru dává úplnou kontrolu nad aktualizacemi, zálohováním, zabezpečením citlivých dat a mnohými dalšími aspekty. To znamená, že se server může plně přizpůsobit požadavkům aplikace. Při vytváření serverové databáze je na výběr mezi relační (SQL) a NoSQL (Not Only SQL) databází. [40]



Obrázek 4.1: Princip fungování REST API [41]

Při vývoji serverové části je vhodné vytvořit takzvané REST (REpresentational State Transfer) API, což je architektonický styl, který usnadňuje komunikaci mezi klientem a serverem viz obr. 4.1. REST API funguje jako propojení mezi databází a klientem, přičemž jsou obě strany na sobě navzájem nezávislé a komunikace je bezstavová, tedy server se nezajímá o stav klienta. Komunikace využívá efektivní výměny dat pomocí definovaných URL adres a HTTP (Hypertext Transfer Protocol)

metod (GET, POST, PUT a DELETE). Mezi nejpoužívanější technologie pro psaní REST API patří Node.js, framework Java Spring Boot, Go a Ruby. [42] [41] [43]

4.2.1 Relační databáze

Relační databáze jsou systémy ukládání dat, které organizují data do tabulek. Každá tabulka je složena z atributů (sloupců) a záznamů (řádků), kde každý záznam je unikátní. Mezi tabulkami mohou být pomocí klíčů vytvořeny relace. Pro operace s daty se využívá jazyk SQL, který umožňuje vytváření, manipulaci a vyhledávání dat. Typickými zástupci relačních databází jsou MySQL, PosgreSQL a Microsoft SQL Server. [44] [45]

4.2.2 NoSQL databáze

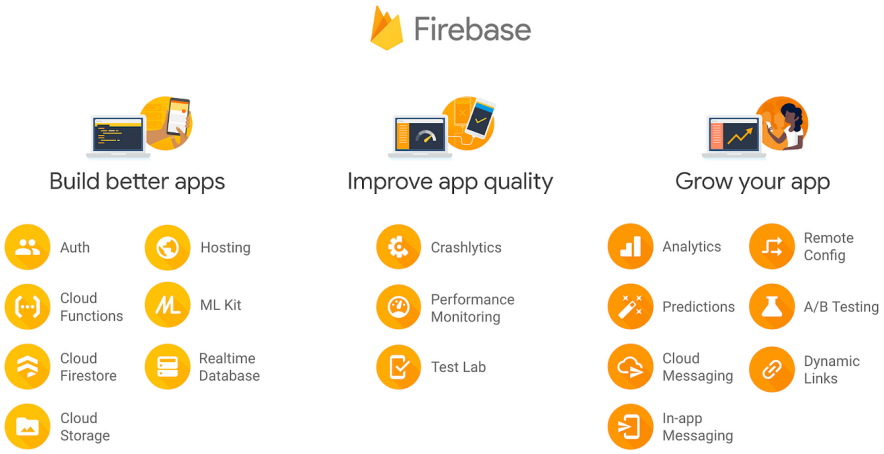
NoSQL databáze nepoužívají tradiční relační tabulkový model pro ukládání dat. Místo toho ukládají data v jiné podobě, jako je například dvojice klíč-hodnota, dokumentově orientovaná data (JSON, XML) nebo ve formě grafů (sémantické sítě). Toto řešení se běžně používá pro ukládání velkých objemů nestrukturovaných dat, jako jsou obrázky, videa nebo příspěvky na sociálních sítích. Typickými zástupci NoSQL databází jsou MongoDB, Cassandra a Neo4j. [44] [46]

4.3 Cloudové úložiště

Cloudové technologie představují vzdálené servery umístěné v datových centrech. Cloudové úložiště umožňuje uživatelům přistup ke svým datům odkudkoliv a kdykoliv z libovolného zařízení prostřednictvím internetu. Data mohou být replikována na více serverech, pokud je pak některý ze serverů odstaven, uživatel má stále možnost se dostat ke svým datům. Využitím cloudových služeb se eliminuje realizace a údržba vlastní infrastruktury serverů, což může vést k rychlejší implementaci, nižší režii a provozním nákladům. Mezi výhody se řadí také škálovatelnost úložiště, jehož velikost se může přizpůsobit aktuálním potřebám. Přestože většina provozovatelů cloudových služeb klade nemalé úsilí na používání nejmodernějších bezpečnostních standardů a technologií, uživatel nemá nad daty plnou kontrolu, ať už se jedná o fyzickou dostupnost dat nebo způsob jejich ukládání. Existuje zde také riziko nepoctivých zaměstnanců nebo jiné bezpečnostní incidenty, které mohou vést ke ztrátě nebo vyzrazení citlivých dat. Mezi nejvýznamnější cloudové úložiště a služby patří například Firebase, AWS (Amazon Web Services) a Microsoft Azure. [47] [48] [49]

4.3.1 Firebase

Firebase je produkt společnosti Google, který pomáhá vývojářům snadno vytvářet, spravovat a rozvíjet jejich aplikace. Jedná se o rozsáhlou sadu backendových cloudových služeb, která eliminuje nutnost vytvářet vlastní API pro propojení s databází, což je běžně vyžadováno u jiných technologií. Firebase nabízí bezplatný plán Spark[2], který umožňuje použití většiny služeb. Tento plán je omezený maximálním počtem uživatelů, velikostí dat a počtem volání za časové období. Placený plán Blaze odstraňuje všechna omezení, přičemž se cena odvíjí od rozsahu využití cloudových služeb dle ceníku Google Cloud. Mezi služby Firebase patří například autentizace, databáze, hosting, systém odesílání zpráv, nástroje strojového učení, možnosti monetizace a některé další služby viz obrázek 4.2: [50] [51] [52]



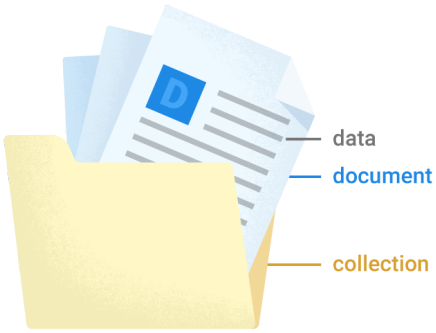
Obrázek 4.2: Služby platformy Firebase [52]

Firebase Authentication umožňuje identifikaci uživatelů pomocí různých identifikátorů jako je e-mail a heslo, telefonní číslo, účet Google, Facebook, Twitter, GitHub nebo Apple a vytváření dočasných anonymních účtů. Tato služba umožňuje též pokročilou funkcionalitu jako je změna hesla, ověření přes e-mailovou adresu či telefonní číslo nebo slučování účtů.

Cloud Firestore je flexibilní cloudová NoSQL databáze, která poskytuje služby pro ukládání dat, synchronizaci a offline podporu. Data jsou uspořádána do hierarchického datového modelu, ve kterém jsou data uložena v dokumentech. Dokumenty jsou pak zařazeny do kolekcí viz obr. 4.3. Dokumenty podporují mnoho datových typů od čísel po složité vnořené objekty.

Firebase Realtime Database je cloudová NoSQL databáze, která poskytuje podobnou funkcionalitu jako Firestore. Hlavní rozdíl oproti Cloud Firestore spočívá ve struktuře, ve které jsou data uložena v podobě jednoho rozsáhlého JSON souboru.

[2] https://firebase.google.com/pricing



Obrázek 4.3: Model uložení dat v Cloud Firestore [53]

Firebase Hosting poskytuje rychlý a bezpečný hosting, který může být použit ke statickým nebo dynamickým webovým stránkám a PWA s použitím bezplatného SSL certifikátu.

Google Analytics for Firebase slouží k analýze návštěvnosti a chování uživatelů v aplikaci. Poskytuje podrobné statistiky o skupině uživatelů a její interakce s aplikací.

Firebase Predictions používá algoritmů umělé inteligence pro analýzu chování uživatelů a následné předpovídání jejich budoucích akcí.

Firebase Cloud Messaging je bezplatná služba, která umožňuje odesílání notifikací do mobilních a webových aplikací napříč platformami.

Firebase A/B Testing je nástroj, který umožňuje provádět experimenty s různými variantami uživatelského rozhraní, zpráv a funkcí, čímž pomáhá k optimalizaci uživatelského zážitku.

Firebase Machine Learning je nástroj, který umožňuje snadné využití balíčků strojového učení bez potřeby velkých znalostí z oblasti umělé inteligence. V nabídce jsou předtrénované modely pro rozpoznávání textu, překlady, detekce objektů a další. Zkušeným vývojářům umožňuje použití vlastních TensorFlow Lite modelů.

4.3.2 AWS Amplify

AWS je komplexní, široce používanou cloudovou platformu poskytovanou společností Amazon, která nabízí služby výpočetního výkonu, databázového úložiště a síť pro doručování obsahu. Cloudové služby AWS využívají modelu „Pay-as-you-go", kde uživatel platí za využívání služeb dle času a objemu uložených dat. Noví uživatelé mají aktivovaný Free Plan, který je bezplatný na 12 měsíců, pro vyzkoušení služeb, které AWS nabízí. [54] [55]

AWS Amplify je vývojová platforma, která usnadňuje vývojářům vývoj mobilních a webových aplikací. Tato služba představuje ucelený balík nástrojů a služeb,

které pomáhají snadno vytvářet, hostovat a monitorovat běh aplikací. Mezi funkce patří například autorizace, ukládání a synchronizování dat, UI (User Interface) komponenty, strojové učení a mnoho dalšího. AWS Amplify je vývojový framework založený na JavaScriptu, který se skládá z knihoven, komponentů uživatelského rozhraní a nástrojů CLI (Command Line Interface): [56] [57]

- **Knihovny:** umožnění připojení, integraci a interakci s cloudovými službami AWS. Knihovny umožňují snadno přidat do aplikací bezpečné ověřování, ukládání souborů, ukládání dat, analytiku, notifikace, a další,
- **Komponenty UI:** Amplify bylo vytvořeno tak, aby integrovalo UX (User Experience) napříč různými platformami, UI komponenty jsou pak předpřipravené komponenty uživatelského rozhraní,
- **Nástroje CLI:** Tyto nástroje umožňují efektivní správu a aktualizace backendové struktury AWS.

4.4 Shrnutí

V této kapitole jsme prozkoumali možnosti pro lokální a online ukládání dat. Pro potřeby synchronizace jsem se rozhodl využít cloudové platformy Firebase viz sekce 4.3.1, která eliminuje potřebu vytvářet vlastní API. Zároveň tato platforma nabízí velký počet funkcí, jako je autorizace uživatele, zasílání notifikací a ukládání dat, které usnadní a zrychlí vývoj aplikace. Ze dvou typů úložišť (Cloud Firestore, Firebase Realtime Database), které Firebase nabízí, jsem vybral službu Cloud Firestore z důvodu podpory ukládání komplexnějších dat. Vzhledem k velikosti plánované aplikace by měl stačit bezplatný Spark plan. Při větším počtu uživatelů a míře použití lze přejít na placený Blaze plan.

Pro ukládání uživatelských dat do lokálního úložiště jsem se rozhodl použít knihovnu Hive viz sekce 4.1.3, která je mezi vývojáři Flutter aplikací velmi populární. Vzhledem k tomu, že naše data nejsou strukturovaná a Firebase rovněž ukládá data do formátu NoSQL, je použití Hive lepší volbou než SQLite. Hive navíc podporuje líné načítání a šifrování, které se též může při vytváření aplikace hodit.

Návrh aplikace 5

Obsahem této kapitoly je specifikace funkcionality, návrh datového modelu a uživatelského rozhraní aplikace. Zpočátku kapitoly specifikujeme funkce, které aplikace nabídne svým uživatelům. V další části se zaměříme na organizaci dat, způsoby ukládání a jejich datový model. V závěru kapitoly navrhneme vzhled uživatelského rozhraní a navigaci mezi obrazovkami.

5.1 Specifikace funkcionality

Před samotným vývojem aplikace je nutné specifikovat funkce, které bude vyvíjená aplikace obsahovat. Tato aplikace se bude zaměřovat na tvorbu hierarchicky organizovaných poznámek s možností synchronizace dat mezi více zařízeními. Aplikace bude primárně vytvářena pro platformu Android.

Hierarchická organizace poskytne uživateli libovolný[1] počet úrovní, neboť konkrétní počet úrovní může být omezující. Hierarchie by měla být přehledná a umožňovat snadnou organizaci a práci s poznámkami. Poznámky bude též možno přesunout na libovolné místo v hierarchii.

Vzhledem k tomu, že primárně cílíme na mobilní zařízení, je velmi pravděpodobné, že potenciální uživatelé budou chtít používat aplikaci i bez připojení k internetu (například při cestě vlakem). Je tedy důležité zajistit, aby aplikace bezproblémově fungovala bez internetového připojení. Mnoho uživatelů má též více zařízení, ze kterých chtějí své poznámky spravovat. Kromě práce v módu bez připojení k internetu, je také důležité, umožnit uživatelům synchronizovat svá data mezi více zařízeními. Synchronizace dat bude probíhat prostřednictvím účtu, který si uživatel v aplikaci vytvoří. Je možné, že uživatel bude upravovat data bez připojení k internetu. V tomto případě mohou vzniknout konflikty, neboli několik různých verzí poznámek, ze kterých nelze jednoznačně určit správnou verzi. Aplikace bude tyto konflikty odhalovat a následně řešit tak, aby uživatel neztratil data.

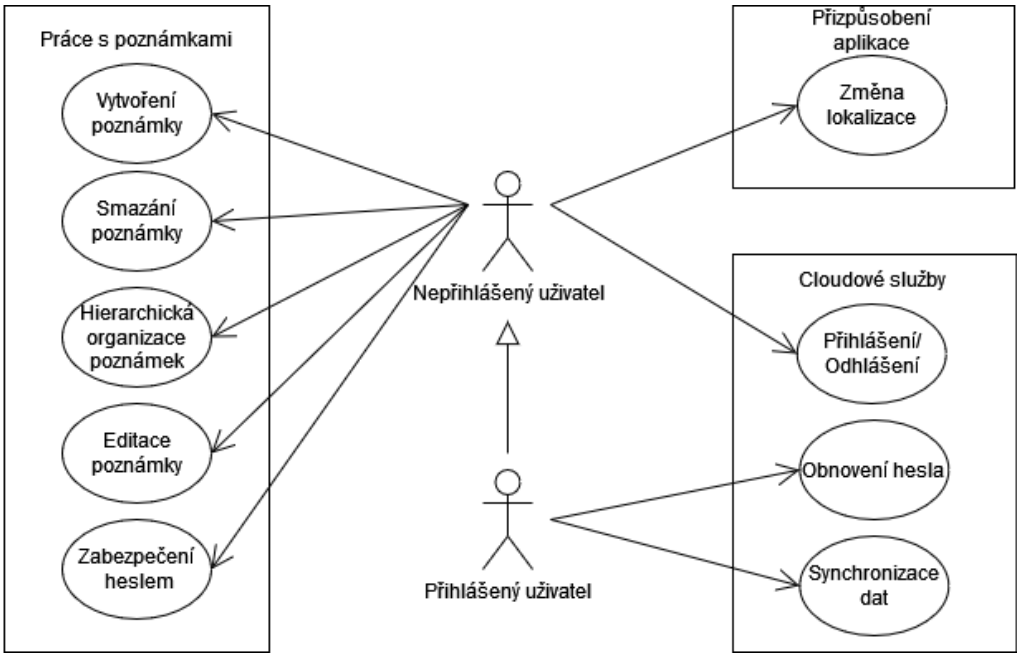
[1] Hierarchie bude omezena fyzickými parametry úložiště

I když to není doporučováno, mohou uživatelé uchovávat v poznámkách citlivá data. Je tedy adekvátní tato data bezpečně uchovávat pomocí moderních bezpečnostních standardů. V případě potřeby poznámky uzamknout pinem nebo heslem, kdyby například uživatel někomu půjčil telefon.

Aplikace bude mít veřejně přístupný zdrojový kód a všechny její funkcionality budou dostupné bezplatně. Dalším cílem je vytvořit dobře strukturovat zdrojový kód aplikace tak, aby byla dobře modifikovatelná a nabízela tak dalším vývojářům možnost přidání nové funkcionality.

5.2 Případy užití

V předchozí sekci jsme specifikovali funkce, které by měla vyvíjená aplikace obsahovat. Následující diagram 5.1 reprezentuje všechny funkční požadavky z pohledu uživatele. Aplikace bude plně podporovat režim bez připojení k internetu, uživatelé nemusejí být při používání aplikace registrováni. Aplikaci tedy budou moci používat dva typy uživatelů, a to přihlášení a nepřihlášení uživatelé. Základní funkcionalita zahrnující práci s poznámkami a přizpůsobení aplikace pomocí lokalizace je dostupná oběma typům uživatelů. Přihlášený uživatel má navíc možnost synchronizace poznámek.

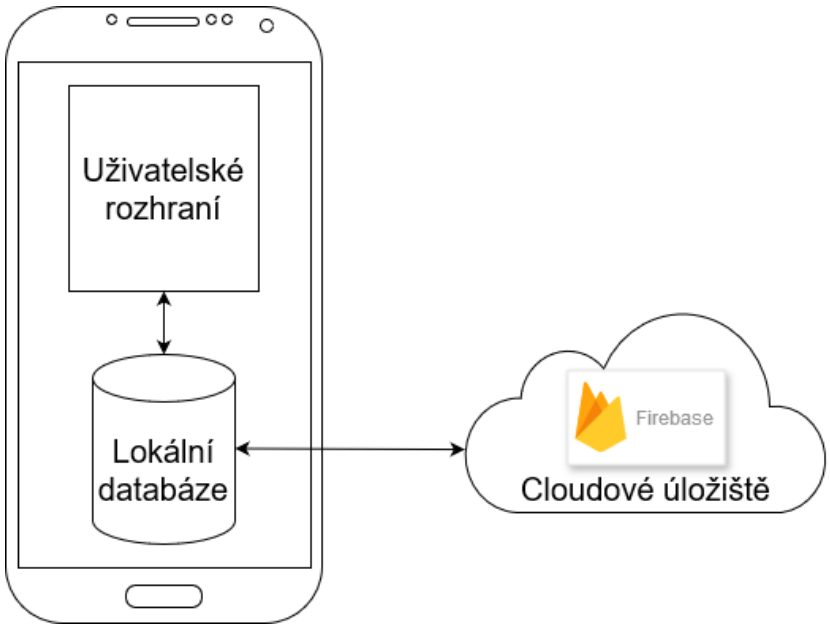


Obrázek 5.1: Případy užití aplikace

5.3 Offline-first návrh

Jak je zmíněno ve specifikaci aplikace. Je velmi důležité, aby aplikace fungovala i bez připojení k internetu. Z tohoto důvodu bude vytvořena offline-first aplikace. V této aplikaci se uživatel nebude muset přihlásit, aby mohl používat aplikaci. Přihlášení bude povinné až v případě, že by uživatel chtěl synchronizovat nebo zálohovat svá data. Pro získání těchto možností se uživatel bude muset přihlásit.

Offline-first je způsob, který zajišťuje bezproblémové fungování softwaru i bez připojení k internetu. Pro dosažení této funkčnosti je potřeba ukládat data na straně klienta, tedy do lokálního úložiště zařízení. Toto bude dosaženo využitím lokální databáze Hive (viz sekce 4.1.3). Při synchronizaci budou data nejdříve uložena do lokální databáze a až poté zobrazena v uživatelském rozhraní viz obrázek 5.2.

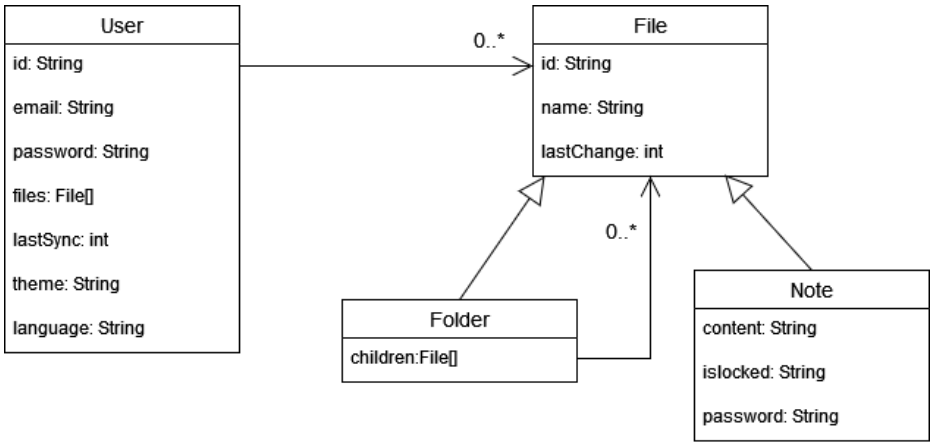


Obrázek 5.2: Offline first model ukládání dat

5.4 Datový model

Na následujícím obrázku 5.3 je znázorněný datový model, který by mohl být využit ve vytvářené aplikací. V návrhu jsou znázorněny entity, jejich atributy a relace mezi nimi. První entitou je uživatel, který bude obsahovat identifikátor, dle kterého bude uživatel ověřovat svou identitu. Dalšími atributy uživatele bude údaj o vlastněných souborech, poslední synchronizaci, preferovaném tématu a jazyce. Každý uživatel pak může vlastnit libovolný počet souborů, což je druhá entita diagramu. Soubor může být dvou druhů, a to typu adresář nebo poznámka. Soubory mají atributy

unikátního identifikátoru, jména a čas poslední změny. Poznámka má navíc oproti souboru ještě atributy obsahu poznámky, jestli je poznámka zamčena a případné heslo. Adresář obsahuje seznam odkazů na soubory, které jsou jeho potomky.



Obrázek 5.3: Plánovaný datový model aplikace

V kapitole 4 jsme určili, jaké technologie budou využity pro ukládání dat. Konkrétně jsme zvolili databázi Hive pro lokální ukládání dat a cloudové úložiště Firestore. Ačkoliv obě databáze patří do kategorie NoSQL databází, jejich styl ukládání dat se liší. Při implementaci tedy bude nezbytné adaptovat tento datový model na specifika dané technologie tak, aby byla zajištěna konzistence dat v obou úložištích.

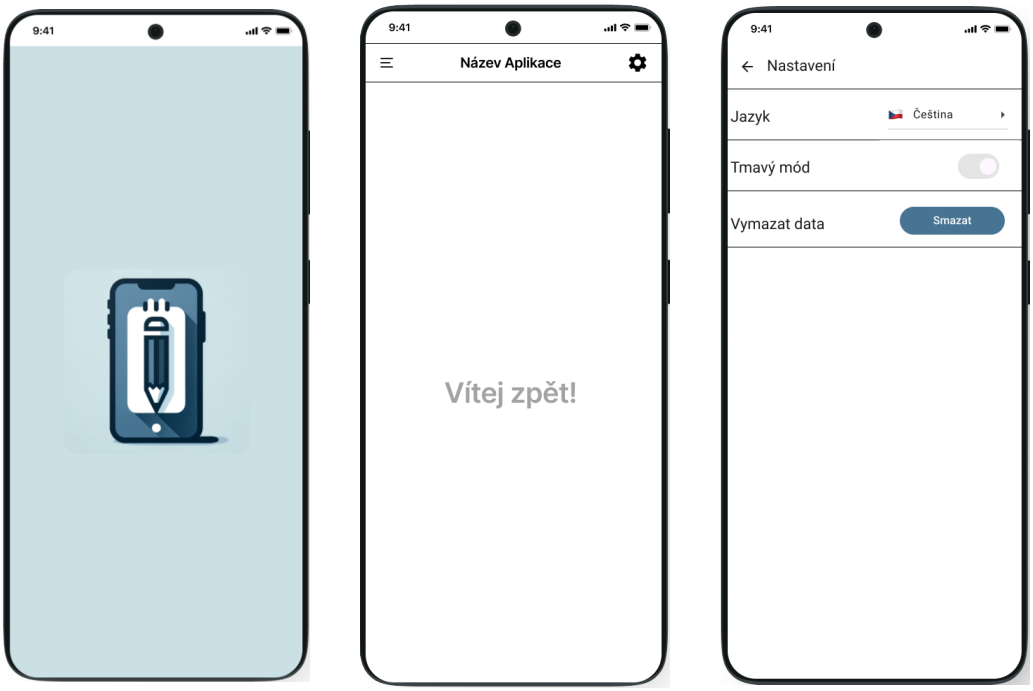
5.5 Uživatelské rozhraní

Před zahájením vývoje je též vhodné navrhnout vzhled a uspořádání komponent na obrazovkách. Zároveň je důležité zvážit, jak bude uživatel interagovat s aplikací a jaká bude navigace mezi obrazovkami. Cílem návrhu uživatelského rozhraní je vytvořit prostředí, které bude uživatelsky přívětivé a ovládání bude pro uživatele intuitivní.

Pro návrh uživatelských rozhraní jsem využil aplikace Figma[2], ve které jsem vytvořil návrhy rozložení jednotlivých obrazovek. Jednotlivé návrhy jsou zobrazeny na obrázcích 5.4, 5.6 a 5.7. Při návrhu jsem využil knihovny Material 3 Design kit, jejíž komponenty budou použity při tvorbě aplikace. Pro návrh uspořádání jednotlivých obrazovek jsem vytvořil drátový model mobilní aplikace viz obrázek 5.8, ve kterém je zobrazeno, jak bude možné se pohybovat mezi obrazovkami.

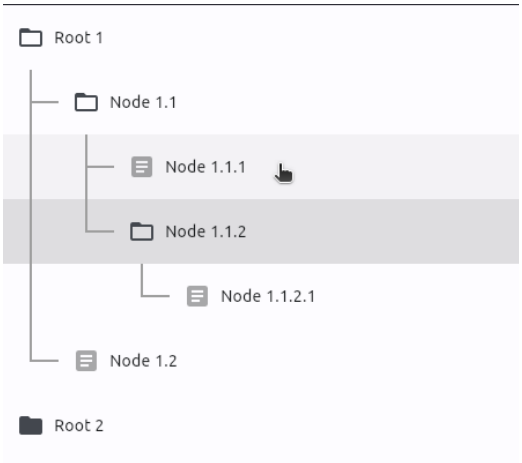
Při spuštění aplikace se uživateli zobrazí inicializační obrazovka s logem (obrázek 5.4 vlevo). Potom, co se aplikace inicializuje, se uživateli zobrazí úvodní obrazovka (obrázek 5.4 uprostřed). Na této obrazovce bude mít uživatel možnost otevřít

[2] https://www.figma.com/



Obrázek 5.4: Návrh uživatelského rozhraní: inicializační obrazovka, úvodní obrazovka a nastavení

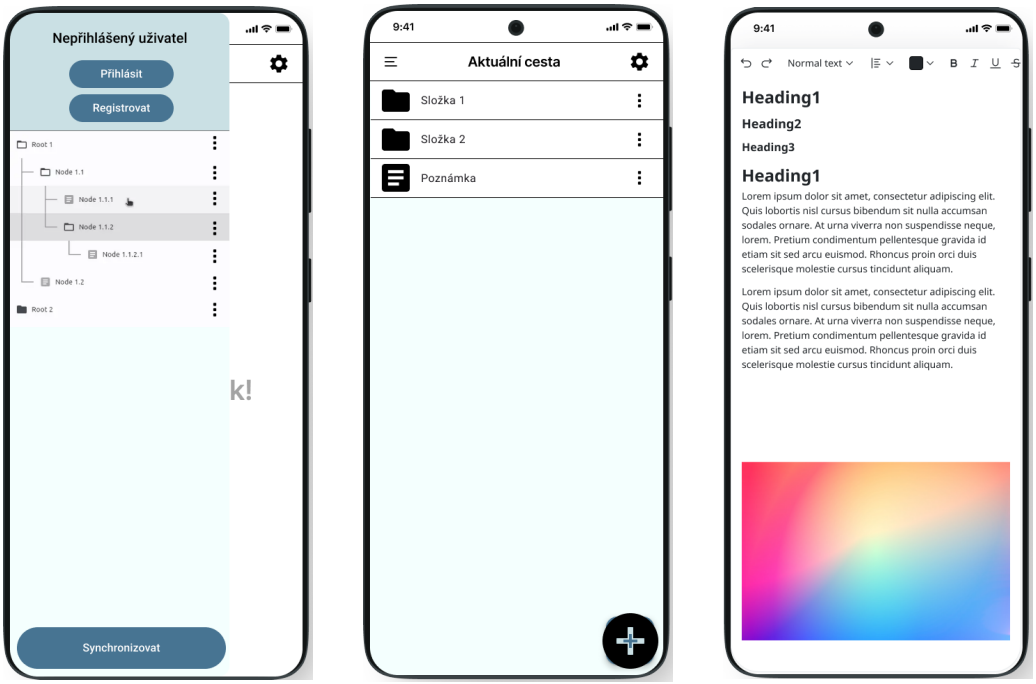
nastavení (obrázek 5.4 vpravo) a ovládací menu (obrázek 5.6 vlevo), ve kterém bude zobrazena hierarchická struktura poznámek s možností přihlášení uživatele. Hierarchická struktura může být realizováno knihovnou `flutter_fancy_tree_view` na obrázku 5.5.



Obrázek 5.5: Stromové zobrazení hierarchické struktury [58]

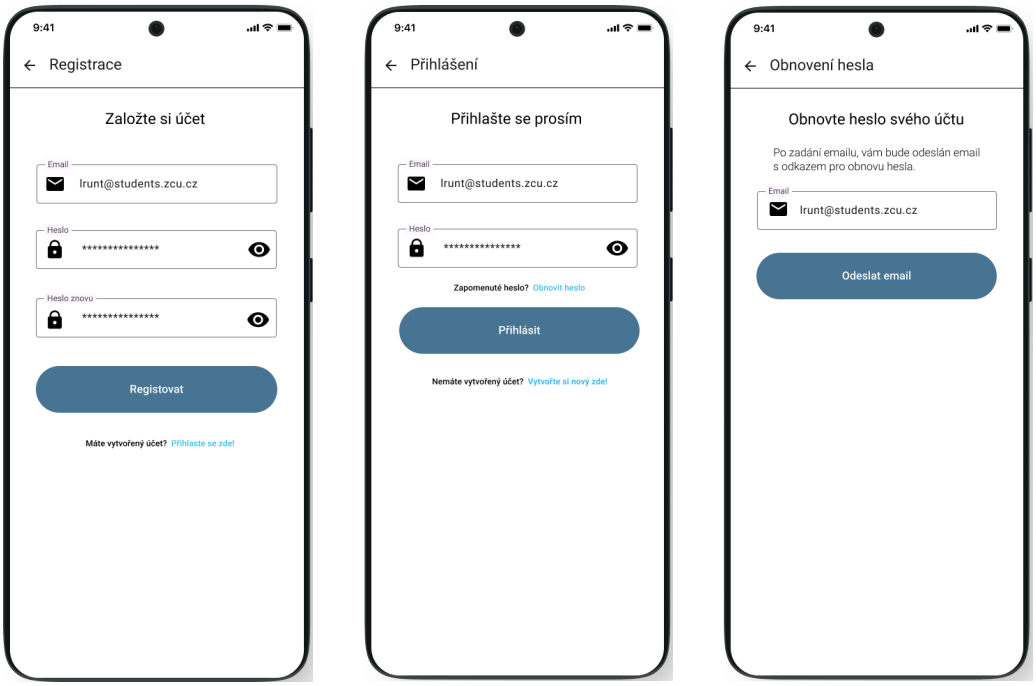
Hierarchii poznámek bude mít uživatel možnost organizovat již ve stromové struktuře. Soubory budou moci být přesunuty, přejmenovány, smazány a v adresářích bude možné vytvářet nové adresáře a poznámky. Tento strom bude též slou-

žit pro navigaci. Při kliknutí na adresář bude uživatel přesměrován na příslušnou stránku s načteným obsahem souboru (viz obrázek 5.6). Při otevření adresáře bude zobrazen seznam potomků adresáře s možností úprav a vytváření nových souborů. V případě poznámku bude uživatel přesměrován to editoru formátovaného textu, kde bude načten obsah poznámky.

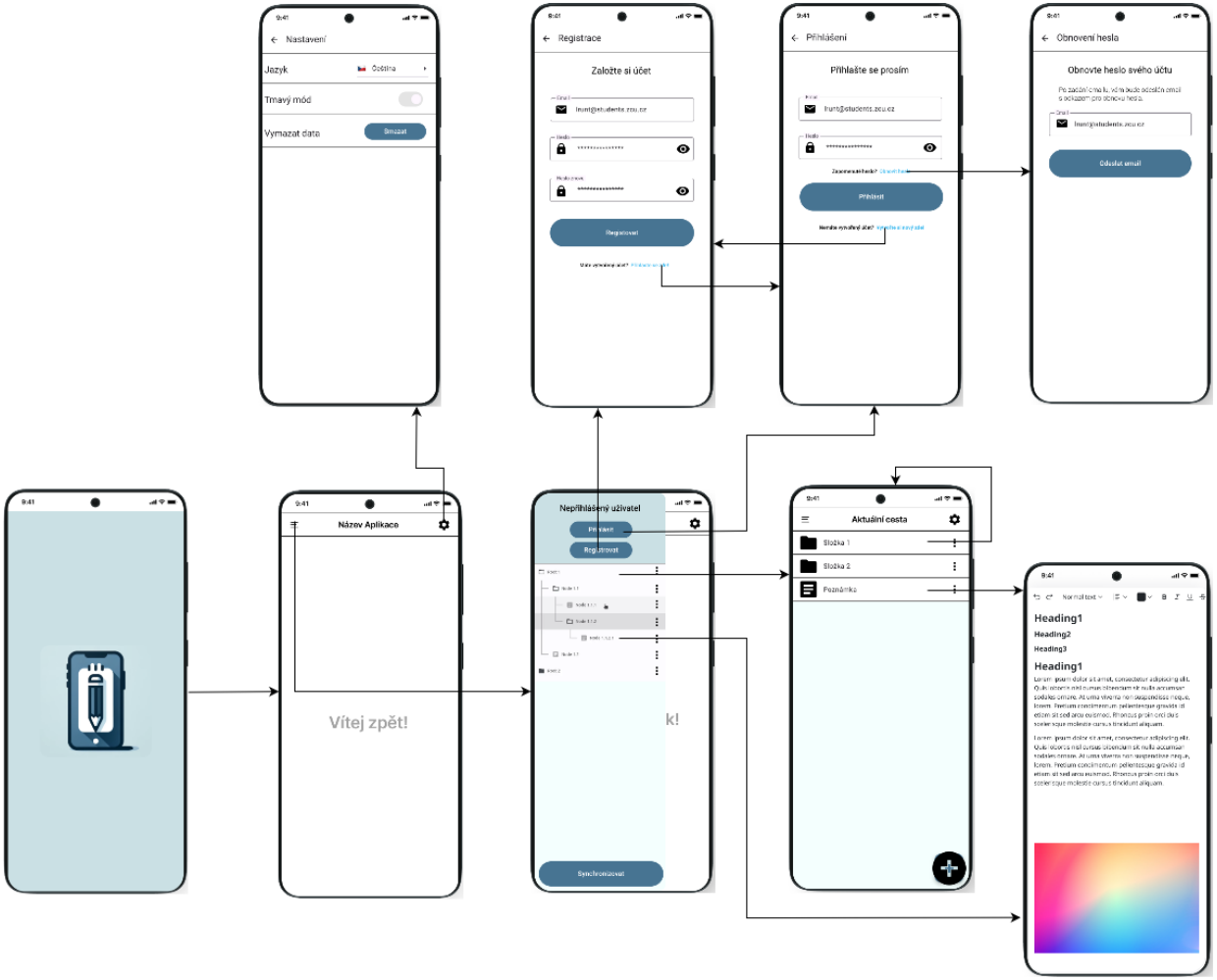


Obrázek 5.6: Návrh uživatelského rozhraní: otevřené vysouvací menu, obsah složky a editace poznámky

Pro přihlášení se budou ve vysouvacím menu nacházet tlačítka Přihlášení a Registrace. Tato tlačítka budou přesouvat uživatele na příslušné obrazovky pro přihlášení a registraci, mezi kterými bude možno se přesouvat (viz obrázek 5.7). Na obrazovce přihlášení bude odkaz na stránku, která bude odesílat e-mail pro obnovu hesla, pokud by uživatel své heslo zapomněl.



Obrázek 5.7: Návrh uživatelského rozhraní: registrace nového uživatele, přihlášení a obnova hesla



Obrázek 5.8: Drátový model mobilní aplikace

Implementace 6

Následující kapitola se zabývá implementací vyvíjené aplikace. Na začátku kapitoly si popíšeme strukturu projektu. Následně se přesuneme k organizaci a ukládání lokálních dat. Od lokálního ukládání dat se přesuneme k využití cloudových služeb Firebase a synchronizaci dat. V závěru kapitoly se budeme věnovat vzhledu aplikace a platformám, na kterých je aplikace zveřejněna.

6.1 Struktura projektu

Projekt má charakteristickou Flutter strukturu. V následující části si popíšeme důležité části této struktury:

- `assets/` je adresář, ve kterém jsou obsaženy všechny multimediální soubory, které aplikace využívá,
- `android/` je adresář obsahující vše potřebné pro sestavení Androidové aplikace včetně Gradle skriptů a Android manifestu,
- `web/` je adresářem, kde Flutter ukládá všechny soubory potřebné pro běh webové aplikace,
- `lib/` je hlavní adresář, ve kterém je vývojářem napsaný zdrojový kód aplikace,
- `test/` je složka, která obsahuje všechny automatické testy. Více o testech v kapitole 7,
- `firebase.json` je soubor popisující konfiguraci hostování webové aplikace,
- `package.json` je soubor, který Node.js používá k zaznamenání verze všech balíčků,
- `pubspec.yaml` je soubor, ve kterém jsou definovány všechny knihovny a závislosti.

6.1.1 Adresář lib

Adresář `lib` hraje ve Flutter projektech důležitou roli, jelikož obsahuje většinu zdrojového kódu, který vývojář napíše. Tento kód je společný pro všechny platformy. Během procesu sestavování aplikace pro konkrétní platformu se tento multiplatformní kód kompiluje do nativního kódu jednotlivých platforem.

Při tvorbě rozsáhlých projektů je vhodné rozhodnout, jak budeme náš projekt strukturovat. Struktura adresáře `lib` má zásadní vliv na přehlednost a udržitelnost projektu. Existuje více způsobů, jak projekt přehledně strukturovat. V našem případě byla zvolena struktura, ve které jsou zdrojové soubory uspořádány dle jejich typu na komponenty, obrazovky, servisy, lokalizační soubory, model a data:

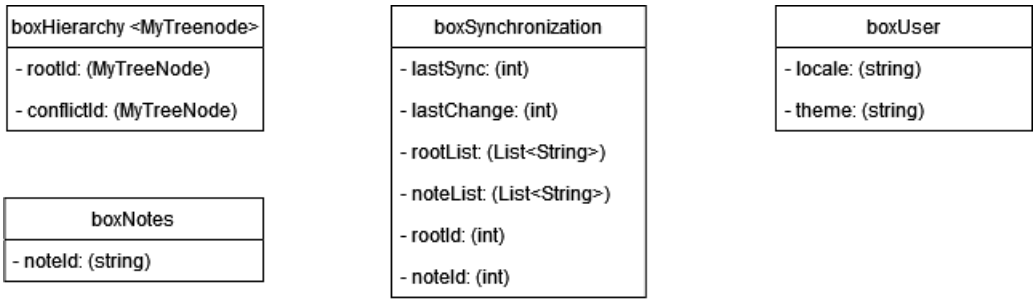
- `components/` je adresář obsahující všechny předdefinované komponenty použité v obrazovkách aplikace,
- `data/` v tomto adresáři se nachází část zdrojového kódu, který manipuluje s lokální databází. Tyto třídy čtou, zpracovávají a ukládají data do lokálního úložiště zařízení,
- `l10n/` (zkratka pro lokalizaci) obsahuje soubory potřebné pro lokalizaci aplikace. Každý jazyk má v tomto adresáři vlastní soubor s příslušnými překlady,
- `screens/` zde se nachází zdrojový kód definující jednotlivé obrazovky aplikace,
- `services/` tento adresář obsahuje logiku aplikace zahrnující autentifikaci, editaci hierarchické struktury, synchronizace a mnohé další,
- `boxes.dart` je soubor obsahující všechny Hive boxy pro lokální ukládání lokálních dat,
- `firebase_options.dart` obsahuje konfiguraci připojení ke službě Firebase pro různé platformy,
- `constants.dart/` je složka, ve které jsou uloženy všechny konstanty použité v aplikaci,
- `main.dart` je soubor, ve kterém se Flutter aplikace inicializuje a spouští.

6.2 Ukládání a editace lokálních dat

V této části kapitoly se zaměříme na základní funkcionalitu aplikace bez připojení k internetu. Nejdříve si vysvětlíme, jak jsou ukládaná lokální data. V dalších sekcích probereme implementaci hierarchické struktury a editoru poznámek.

6.2.1 Lokální ukládání

Pro lokální ukládání byla použita již zmíněná databáze Hive, která pracuje s boxy. Box je schopen uložit primitivní datové typy nebo objekty, pokud je k boxu připojený vygenerovaný TypeAdapter příslušného objektu. Ukládání objektů jsem využil v případě ukládání hierarchické struktury. V následujícím schématu 6.1 je zobrazeno, jaká data jsou uložena v konkrétních boxech.



Obrázek 6.1: Model pro ukládání lokálních dat

Lokální data jsem uložil do čtyř boxů, podle toho, s jakým účelem se data používají. Každý box ovládá jedna třída v adresáři `data/`. Hierarchickou strukturu ukládá `boxHierarchy` a `boxNotes` ukládá obsah poznámek. Data v těchto boxech se ukládají pomocí identifikátoru (cesty) objektů. Data, která jsou potřeba pro správnou synchronizaci dat, jsou uložena v `boxSynchronization`. Pro uložení uživatelských preferencí slouží `boxUser`.

6.2.2 Hierarchická struktura

Hierarchická struktura hraje v naší aplikaci klíčovou roli. V rámci implementace jsem se rozhodl tuto strukturu znázornit dvěma způsoby. Prvním způsobem je pomocí treeView ve vysouvacím menu. Pro treeView neexistuje žádná defaultní knihovna Flutteru, a tak jsem využil knihovny `flutter_fancy_tree_view`[1], která nabízí vykreslení hierarchického stromu. K použití této knihovny je potřeba přizpůsobit datový model, přesněji adresáře a poznámky, potřebám této knihovny. Tato knihovna pracuje s instancemi, které obsahují atributy `title` a `children` viz zdrojový kód 6.1. Při inicializaci widgetu se pak vytváří instance `TreeControlleru` do které se inicializuje `List` kořenů stromu viz zdrojový kód 6.2.

[1] https://pub.dev/packages/flutter_fancy_tree_view

Zdrojový kód 6.1: MyTreeNode - uzel hierarchické struktury

```
class MyTreeNode {
    final String title;
    final List<MyTreeNode> children;

    const MyTreeNode({
        required this.title,
        this.children = const <MyTreeNode>[],
    });
}
```

Druhým způsobem hierarchického zobrazení je seznam potomků, který je implementován pomocí `ListView`. Pokud uživatel v hierarchické struktuře klikne na adresář, jsou na hlavní stránce zobrazeny potomci adresáře, který uživatel stiskl. Na této stránce má uživatel možnost procházet adresáře i poznámky.

Zdrojový kód 6.2: Vytvoření instance TreeControlleru

```
final List<MyTreeNode> roots = hierarchyDb.getRoots();
final treeController = TreeController<MyTreeNode>(
  roots: roots,
  childrenProvider: (MyTreeNode node) => node.children,
);
```

V hierarchické struktuře jsou definovány dva typy uzlů, jejíž typ určuje atribut `isNote`. Každý uzel je instancí `MyTreeNode`, které jsem přidal několik atributů, aby šla hierarchická struktura snadno editovat a byla poskytnuta příslušná funkce. Instance `MyTreeNode` obsahuje následující atributy:

- **id:** Jedinečný identifikátor uzlu, který byl navržen pro efektivní prohledávání a organizaci hierarchické struktury. Identifikátor instance obsahuje cestu ve formátu `|<id_kořenu>|<id_uzlu>|<id_listu>`, kde každé ID odděluje vertikální čára,
- **title:** Název poznámky nebo adresáře, který je zobrazen v uživatelském rozhraní. Nesmí obsahovat znak '|' (oddělovač v cestě) a '/' (znak se speciální funkcí ve Firestore),
- **isNote:** Značí, zda je uzel adresářem či poznámkou. Na základě této vlastnosti se určují funkce, dostupné pro daný uzel,
- **children:** Seznam potomků uzlu. Do seznamu se ukládají celé instance uzlů,
- **isLocked:** Určuje, zda je uzel chráněn heslem,
- **password:** Hash hesla zamčeného uzlu, vytvořený algoritmem SHA-256. V odemčených uzlech nastavena hodnota `null`.

Metody pro organizaci hierarchické struktury jsou implementovány v souboru `services/node_service.dart`. V rámci naší aplikace je implementována metoda pro odstranění, přejmenování, vytvoření nové složky a poznámky (jen u adresářů), přesunutí a uzamknutí/odemknutí (jen u poznámek). Při operacích se kontroluje jméno souboru tak, aby jméno souboru nebylo prázdné, neobjevovaly se nepovolené znaky '|' a '/' a aby díky operaci nevznikly dva uzly se stejným jménem ve stejném adresáři a tím pádem i se stejným identifikátorem. Při odstraňování souborů je nutno zkontrolovat jeho typ. V případě poznámky musíme odstranit i obsah samotné poznámky. Při existenci potomků musíme tento postup opakovat u každého následujícího potomka. U přejmenování a přesouvání musíme zajistit správnou změnu cest v identifikátorech uzlů.

6.2.3 Editace poznámek

Většina aplikací pro tvorbu poznámek nabízí rozsáhlé možnosti formátování a obsahu poznámek. Pro poskytnutí srovnatelné úrovně uživatelské přívětivosti formátování textu, jsem se rozhodl využít knihovnu `flutter_quill`[2], pro kterou jsou pravidelně vydávány nové verze. Tento editor formátovaného textu pracuje s daty ve formátu JSON. Aby uživatel neztratil žádná data, která v aplikaci vytvoří, ukládám obsah poznámky při každé změně do lokálního úložiště zařízení. Nová data přepisují starou verzi poznámky společně s aktualizací časové značky poslední úpravy.

6.3 Integrace služeb Firebase

Vytvářená aplikace využívá cloudových služeb Firebase. V této sekci si představíme služby, které vytvářená aplikace využívá. V rámci této sekce si také vysvětlíme, jak funguje synchronizace dat, která s těmito službami úzce souvisí.

6.3.1 Autentizace

Aby uživatelé mohli synchronizovat svá data, je nutné uživatele nějakým způsobem identifikovat. V naší aplikaci je k tomuto účelů využita cloudová služba Firebase Authentication (viz sekce 4.3.1), která dokáže umožnit přístup k datům pouze autorizovaným uživatelům.

Autentizační proces naší aplikace podporuje přihlášení kombinací e-mailu a hesla a přihlášení účtem Google. Pro případ, že uživatel zapomene své heslo, jsem využil služby Firebase odeslat e-mail s odkazem na stránku, kde uživatel může své heslo změnit. Obsah tohoto e-mailu lze libovolně změnit, dle potřeb vytvářené aplikace.

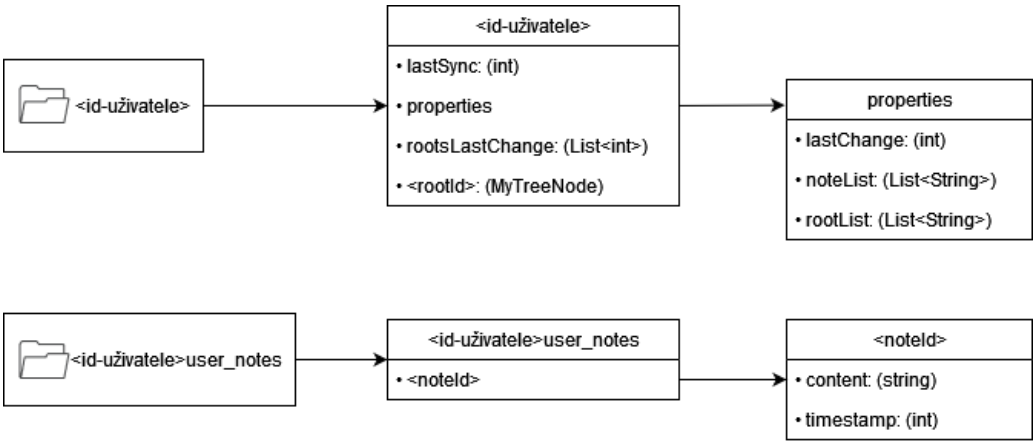
[2] https://pub.dev/packages/flutter_quill

Implementace Firebase Auth v našem projektu vyžadovala přidání závislosti `firebase_auth` a konfigurování propojení s vytvořeným Firebase projektem (návod na konfiguraci viz příloha C). Po konfiguraci a přidání závislosti lze využívat instanci `Firebase.auth`, která zprostředkovává přístup ke všem funkcím a udržuje stav uživatelova přihlášení. Tato instance je inicializována již na začátku běhu aplikace a postupně předávána v parametrech jednotlivých tříd, pro lepší usnadnění testování a lepší udržitelnost kódu.

V logické vrstvě aplikace ve třídě `auth_service.dart` jsou implementovány metody pro registraci, přihlášení, odhlášení a odeslání e-mailu pro změnu hesla. Pokud nastane při přihlašování chyba (například uživatel nebude mít přístup k internetu) vrátí metody chybový kód, který je následně namapován na chybové hlášení srozumitelné pro uživatele. K zobrazení chybových hlášení jsem použil komponentu `SnackBar`.

6.3.2 Cloud Firestore

Pro ukládání uživatelských dat do cloudové databáze bylo nutné přizpůsobit datový model požadavkům databáze Firestore, která pracuje s kolekcemi a dokumenty. Vytvořený model využívá údaje z Firebase Auth tak, že každý uživatel vlastní kolekce s vlastním identifikačním číslem. Tím má uživatel zajištěn přístup ke svým datům. Datový model databáze Firestore je zobrazen na následujícím obrázku 6.2. Komunikaci s databází zajišťuje třída `firestore_service.dart`, která mimo jiné obsahuje logiku synchronizace a s tím spojené detekování konfliktů.



Obrázek 6.2: Datový model databáze Firestore

6.3.3 Synchronizace dat

Před implementací synchronizace bylo rozmýšleno mezi synchronizací s užitím časových značek a historií změn. Z těchto dvou metod byla zvolena metoda časových značek. Tento způsob uchovává časový údaj o poslední synchronizaci a poslední změně každého souboru. Dle časových údajů se pak rozhoduje, zda budou data stažena do lokálního zařízení nebo nahrána na cloudové úložiště. Pomocí časových značek lze také odhalit potenciální konflikty. Pokud je poslední změna souboru provedena před poslední synchronizací, byl soubor nezměněn a může se přepsat novými daty. Pokud je však poslední změna souboru datována až po synchronizaci u lokálního i na cloudu uloženém souboru, je velmi pravděpodobné, že se jedná o konflikt.

6.3.4 Řešení konfliktů

Pokud je při synchronizaci odhalen potencionální konflikt, je lokální verze těchto dat uložena s časovou značkou ve formátu <`jméno souboru`><`časová značka`> do části lokální databáze pro správu konfliktů. Následně jsou data, která se zobrazují uživateli v hierarchickém stromě přepsána verzí dat z databáze Firestore. Po synchronizaci má tak uživatel přístup k oběma verzím poznámek. Data, kterých se konflikt týká, lze zobrazit v nastavení, stisknutím tlačítka Konflikty.

6.4 Přizpůsobení a vzhled

Při vývoji jsem se snažil zapracovat též na příjemném a konzistentním vzhledu, který se bude uživatelům líbit a bude charakterizovat vytvářenou aplikaci. Snahou také bylo umožnit uživatelům volbu preferovaného jazyka a výběr mezi světlým a tmavým režimem. V další části si rozebereme, jakým způsobem byl navržen vzhled a přepínání jednotlivých preferencí.

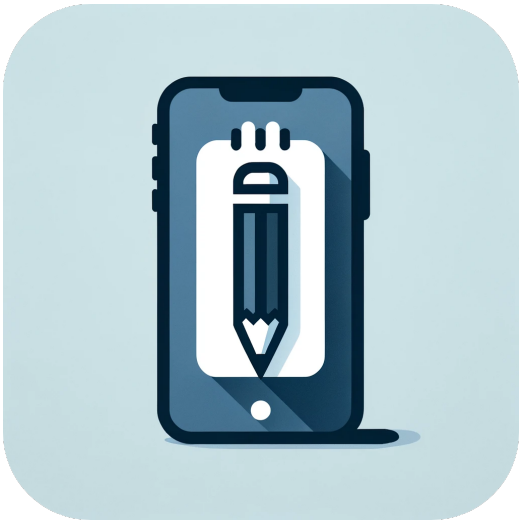
6.4.1 Logo a úvodní obrazovka

Každá aplikace by měla mít své logo a název, který ji bude charakterizovat a odlišovat od ostatních aplikací. Aplikaci jsem pojmenoval názvem *Notes*, což je slovo, které má svůj význam v češtině i angličtině. Vzhledem k tomu, že nemám dostatečné umělecké cítění, využil jsem pro vytvoření loga, zobrazeného na obrázku 6.3, umělé inteligence DALL-E. Při založení Flutter projektu se aplikaci nastaví výchozí Flutter logo. Pro změnu loga aplikace byla použita knihovna `flutter_launcher_icons`[3], pro kterou jsem v souboru `pubspec.yaml` definoval verzi a obrázek, ze kterého

[3] https://pub.dev/packages/flutter_launcher_icons

se ikony vytvoří. Následně jsem použil příkaz pro vygenerování ikony pro všechny distribuce.

Při spouštění se aplikace inicializuje. Při inicializování je implicitně nastavena čistě bílá obrazovka. Uživatel pak má pocit, že aplikace nefunguje správně. Tuto stránku jsem tedy nechtěl ponechat bílou, a proto jsem na tuto stránku přidal logo aplikace s použitím knihovny `flutter_native_splash`[4]. Pro nadefinování úvodní obrazovky je opět potřeba upravit `pubspec.yaml`, ve kterém můžeme nastavit obrázek a pozadí, které se uživateli při inicializaci zobrazí. S použitím příkazu lze pak vygenerovat úvodní obrazovku pro všechny distribuce.



Obrázek 6.3: Logo aplikace

6.4.2 Lokalizace

Vytvořená aplikace obsahuje podporu a překlady anglického a českého jazyka. Pro lokalizaci je využita knihovna `intl 0.18.1`[5], která je přidána v seznamu závislostí s povolením automatického generování zdrojového kódu. Lokalizační soubory s překlady jsou uloženy v adresáři `lib/l10n/`. Pro každý jazyk je vytvořen samostatný lokalizační soubor ve formátu ARB (Application Resource Bundle), který obsahuje mapu klíčů a překladů. V případě vyvíjené aplikace se jedná o soubory `app_en.arb` pro angličtinu a `app_cs.arb` pro češtinu.

Pro používání internacionalizace je dále potřeba vytvořit konfigurační soubor `l10n.yaml` viz zdrojový kód 6.3, ve kterém první řádka specifikuje cestu, kde se nacházejí lokalizační soubory. Na druhé řádce je určen šablonový soubor. Tento soubor obsahuje všechny definice textových řetězců s popisy, kde se daný textový řetězec

[4] https://pub.dev/packages/flutter_native_splash
[5] https://pub.dev/packages/intl

používá. Jako šablonový soubor se nejčastěji používá anglický lokalizační soubor. Poslední řádkou v konfiguraci definujeme název souboru, kam se nám budou generovat jednotlivé překlady. Překlady lze generovat příkazem: `flutter gen-l10n`.

Zdrojový kód 6.3: Konfigurační soubor l10n.yaml

```
// Cesty k lokalizačním souborům.
arb-dir: lib/l10n
// Určení šablonového souboru.
template-arb-file: app_en.arb
// Jméno souboru, kam se budou generovat překlady.
output-localization-file: app_localizations.dart
```

K přepínaní jazykových preferencí byla vytvořena třída `language.dart`, která je uložena v adresáři `model`. Instance třídy obsahuje atribut emotikonu vlaky (`flag`), název jazyka (`name`) a kód jazyka dle normy ISO 639-1 (`langCode`). Výchozím jazykem aplikace je angličtina. Při změně jazyka je kód preferovaného jazyku uložen do lokální databáze.

6.4.3 Barevné palety a přepínání témat

Všichni uživatelé nemusejí preferovat světlá témata. Zejména pak ve večerních hodinách může být příjemnější sledovat displej méně osvětlený než přesvícený. Z těchto důvodů jsem se rozhodl do aplikace přidat výběr mezi světlým a tmavým módem.

Při definování palety jednotlivých zobrazení jsem se snažil využít barev, které byly užity při vytváření loga aplikace, aby bylo docíleno konzistence. Při vytváření barevných palet pro oba módy jsem využil stránky Material Theme Builder[6], která zobrazuje, jak barvy ovlivní vzhled jednotlivých komponent. Vývojář pak může z definovaných barev vygenerovat barevné palety pro obě témata. Tyto palety jsem vygeneroval a následně uložil jako konstanty do souboru `constants.dart`.

Pro umožnění změny tématu jsem vytvořil třídu `theme.dart`, která využívá funkce poskytovatele stavu (Provideru) pro centralizovanou správu režimu. Tato třída uchovává informace o aktuálním tématu, světlém tématu a tmavém tématu a obsahuje metody `setThemeMode`, `setLightScheme` a `setDarkScheme`, kterými lze režim měnit. Instance `ThemeProvider` je vytvořena při inicializaci aplikace s použitím `ChangeNotifierProvider` a `Customer` (viz zdrojový kód 6.4), což poskytuje správné téma napříč widgetovým stromem. Pokud pak nastane změna, všichni posluchači jsou informováni o změnách, které nastaly.

V souboru `main.dart` je pak mód inicializován na systémový mód uživatele. Při změně preferencí se uživatelské preference ukládají do lokální databáze aplikace a při opětovném spuštění programu se inicializuje hodnota z lokální databáze.

[6] https://m3.material.io/theme-builder#/custom

Zdrojový kód 6.4: Inicializace ChangeNotifierProvider instance ThemeProvider

```
@override
 Widget build(BuildContext context) {
   return ChangeNotifierProvider(
     create: (context) => ThemeProvider(themeMode:
  initialThemeMode, userDb: db),
     child: Consumer<ThemeProvider>(
       builder: (context, themeProvider, child) {
         return MaterialApp(
           title: 'Notes',
           theme: ThemeData(useMaterial3: true, colorScheme:
   themeProvider.lightScheme),
           darkTheme: ThemeData(useMaterial3: true,
  colorScheme: themeProvider.darkScheme),
           themeMode: themeProvider.themeMode,
           home: MainScreen(
           ),
         );
       },
     ),
   );
 }
```

6.5 Distribuce

S použitím technologie Flutter se mi podařilo vytvořit verzi aplikace pro platformu Android a progresivní webovou aplikaci. Obě verze sdílí většinu zdrojového kódu. Většina změn se týká vizuálního aspektu aplikace, konfigurace propojení se službou Firebase a změny závislostí knihoven.

6.5.1 Android

Pro mobilní verzi s Androidem byl vygenerován APK (Android Application Package) soubor pomocí příkazu `flutter build apk`. Aplikace vyžaduje alespoň Android 6.0 Marshmallow (API level 23). Návod na instalaci je uveden v příloze B bakalářské práce.

6.5.2 Webová aplikace

Pro distribuci webové aplikace bylo potřeba upravit inicializaci spojení se službou Firebase a změnit závislostí v rámci přihlášení pomocí účtu Google. Následně byl vygenerován nativní kód webové aplikace příkazem `flutter build web`. Vygenerované zdrojové soubory byly umístěny na hostovaný Firebase server. Postup konfigurace a nastavení hostování je popsán v příloze C.

Testování Aplikace 7

Nezbytnou součástí vývoje je testování. Testy nám sice neříkají nic o tom, že je software bezchybný, avšak mohou nám pomoci odhalit některé z chyb. V této kapitole se zaměříme na testovací strategie a zvýšení kvality vytvářené Flutter aplikace. Na začátku kapitoly se budeme věnovat automatickému testování. Následně bude aplikace otestována testovacími scénáři na různých zařízeních. V závěru kapitoly zkontrolujeme kvalitu aplikace dle měřítek Google Play.

7.1 Automatické testování

Automatické testování zefektivňuje proces testování softwaru tím, že snižuje úsilí a potřebný čas na vykonání testů. Obecně také platí, že čím dříve najdeme chybu, tím menší bude vyvinuté úsilí na opravu této chyby. Toto testování se hodí zejména v případě kontrolování, že nově přidané řádky kódu negativně neovlivňují funkcionalitu aplikace.

7.1.1 Unit testy

Jednotkové testy představují malé cílené testy, které ověřují správné fungování konkrétní části kódu (například funkce, metody nebo třídy). Cílem jednotkových testů je ověřit logickou správnost za rozdílných podmínek a na různých vstupech. V naší aplikaci byly jednotkovými testy otestovány především metody zabývající se logickou složkou (organizace hierarchie a autentizace uživatele) a zpracováním dat (model, lokální ukládání a synchronizace).

Průběh jednotkového testu můžeme rozdělit do tří hlavních fází. V první fázi se v metodě `setUp` inicializuje vývojové prostředí, nastavují se mockované objekty a připravují se všechna potřebná data pro vykonání testu. Další fází je samotné vyvolání testované funkce nebo metody s připravenými daty. V poslední fázi se ověřuje, zda výstup odpovídá očekávaným hodnotám nebo byly vyvolány očekávané výjimky. Následující zdrojový kód 7.1 ukazuje výše popisovaný průběh jednotkového testu.

Zdrojový kód 7.1: Příklad jednotkového testu

```
test('getFormattedDate returns formatted date string', () {
      // Příprava dat
      int millisecondsSinceEpoch = 1713016004370000;
      String expectedDateString = '15:46 13.04.2024';

      // Vyvolání testované funkce
      final formattedDate = UtilService.getFormattedDate(
    millisecondsSinceEpoch);

      // Ověření očekávaných výsledků
      expect(formattedDate, expectedDateString);
    });
```

7.1.2 Mockování objektů

Jednotkové testy jsou izolované tím, že nečtou žádná data z disku ani serveru. Velké procento metod však pracuje s daty ze serveru, čte data z disku a používá externí knihovny. V tomto případě je potřeba tato data simulovat. Ve Flutteru se k simulaci objektů používá knihovna Mockito[1], která umožňuje vytvářet mockované objekty tříd a rozhraní, které dokáží předdefinovanými odpověďmi napodobit chování skutečných objektů.

Zdrojový kód 7.2: Příklad mockování knihovny Firebase

```

class MockFirebaseAuth extends Mock implements FirebaseAuth {
  // Přetížení funkce, aby nevykonávala žádnou činnost
  @override
  Future<UserCredential> createUserWithEmailAndPassword({
    required String? email,
    required String? password,
  }) => super.noSuchMethod(Invocation.method(#
    createUserWithEmailAndPassword, [email, password]),
    returnValue: Future.value(MockUserCredential()));
}

// Mockování návratového objektu
class MockUserCredential extends Mock implements
    UserCredential {}
```

Nastavit testovací prostředí je mnohdy těžší než psaní samotné funkce, kterou zrovna testujeme. Aby byla zajištěna nezávislost testované funkce, je nutné simulovat všechny vnější závislosti. Zdrojový kód 7.2 ukazuje, jak je možné mockovat objekt knihovny `FirebaseAuth`, aby volaná metoda nevykonávala žádnou

[1] https://pub.dev/packages/mockito

činnost. V následující ukázce kódu 7.3 je příklad přípravy testovaného prostředí metodou `setUp()` a následné provedení testu. Pro registrace pomocí knihovny `FirebaseAuth` bylo potřeba vytvořit mockovaný objekt lokalizace, kontextu, Firebase funkce pro autentizaci a objekt, který tato metoda vrací. V rámci testování byla otestována úspěšná i neúspěšná registrace.

Zdrojový kód 7.3: Příklad přípravy a provedení jednotkového testy s mock objekty

```
1
2  void main() {
3    TestWidgetsFlutterBinding.ensureInitialized();
4    late MockBuildContext mockContext;
5    late MockAppLocalizations mockLocalizations;
6    late MockFirebaseAuth mockAuth;
7    late MockUserCredential mockUserCreditial;
8    late AuthService authService;
9    late MockGoogleSignIn mockGoogleSignIn;
10
11   // Příprava testového prostředí
12   setUp(() {
13     mockContext = MockBuildContext();
14     mockLocalizations = MockAppLocalizations();
15     mockAuth = MockFirebaseAuth();
16     mockUserCreditial = MockUserCredential();
17     mockGoogleSignIn = MockGoogleSignIn();
18     authService = AuthService(
19       auth: mockAuth,
20       localizationProvider: (_) => mockLocalizations,
21   );
22
23   test('Registration - simple test (succesfull)', () async {
24     // Příprava dat
25     const String email = "test@gmail.com";
26     const String password = "password123";
27     // Definování návratové hodnoty mockované funkce
28     when(mockAuth.createUserWithEmailAndPassword(email: email
       , password: password)).thenAnswer((realInvocation) =>
       Future.value(mockUserCreditial));
29     // Vyvolání testované funkce
30     final result = await authService.register(email, password
       );
31     // Ověření očekávaných výsledků
32     expect(result, mockUserCreditial);
33   });
34 }
```

7.1.3 Widget testy

Widget testy bývají komplexnější než jednotkové testy. Tyto testy testují fungování jednoho widgetu bez závislosti na okolí. Cílem je ověřit, zda se widget chová a vypadá podle očekávání. Toto testování mnohdy vyžaduje více tříd pro poskytnutí příslušného kontextu životního cyklu aplikace. Příslušný widget by měl být schopen reagovat na akce uživatele, provádět rozvržení a instancovat podřízené widgety. Stejně jako u jednotkových testů je prostředí testu simulováno a je tedy mnohem jednodušší než plnohodnotný systém uživatelského rozhraní. [59]

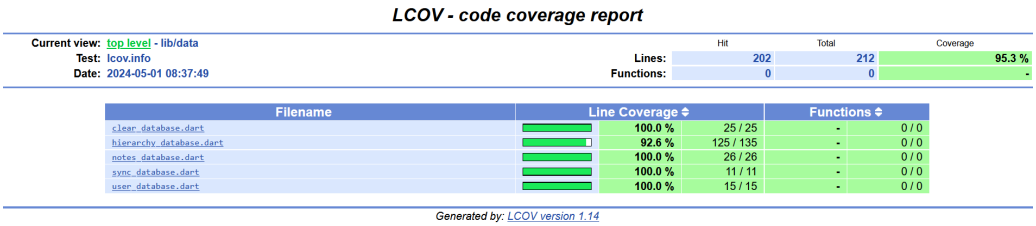
Vzhledem k tomu, že widget testy testují primárně vzhled a fungování widgetů, byly tyto testy použity primárně na třídy komponent jako jsou tlačítka, dialogová okna v adresáři `components/`. Průběh widget testů je velmi podobný jednotkovým. Na začátku testu nastavíme testovací prostředí a metodou `pumpWidget` vytvoříme testovaný widget. V následující fázi se simulují uživatelské interakce s widgetem, jako je například stisknutí, posunutí nebo vstupy do textových polí. V poslední fázi testu se ověřuje, zda jsou výsledky interakcí správné. V následujícím výpisu kódu 7.4 je zobrazen widget test textového pole pro zadání hesla.

Zdrojový kód 7.4: Validace funkčnosti textového pole pro zadání hesla

```
1   testWidgets('changes␣password␣visibility␣functionality',
    (WidgetTester tester) async {
2     // Příprava widgetu
3     final controller = TextEditingController();
4     await tester.pumpWidget(MaterialApp(
5       home: Scaffold(
6         body: StyledTextField(
7           isPasswordField: true,
8           hint: 'Password',
9           controller: controller,
10          pefIcon: const Icon(Icons.key),
11        ),
12      ),
13    ));
14    // Heslo je skryté
15    expect(find.byIcon(Icons.visibility), findsOneWidget);
16    // Změna viditelnosti hesla
17    await tester.tap(find.byIcon(Icons.visibility));
18    // Simulace překreslení a aktualizace UI
19    await tester.pump();
20    // Ověření, že heslo je viditelné
21    expect(find.byIcon(Icons.visibility_off),
    findsOneWidget);
22  });
```

7.1.4 Měření pokrytí kódu

Pokrytí kódu je metrika používaná v softwarovém vývoji. Tato metrika měří, kolik procent zdrojového kódu je testováno automatickými testy. Cílem kódového pokrytí je identifikace, jak kvalitně jsou otestovány jednotlivé části kódu, což napomáhá k identifikaci méně testovaných nebo dokonce zcela netestovaných oblastí kódu.



Obrázek 7.1: Statistika pokrytí kódu jednotlivých tříd

Při vývoji Flutter aplikací lze pokrytí kódu zjistit příkazem `flutter test --coverage`. Tento příkaz provede všechny automatické testy a vygeneruje soubor `lcov.info`, umístěný v adresáři `coverage/`, který obsahuje informace o pokrytí ve formátu LCOV (Linux/Graphical Code Coverage). V operačních systémech Linux a Mac pak lze z vygenerovaného souboru `lcov.info` pomocí `genhtml` vygenerovat HTML stránku, která přehledně zobrazuje, jak velké procento jednotlivých řádků bylo otestováno viz obrázek 7.1. Při rozkliknutí konkrétní třídy jsou světle modrou barvou zvýrazněny řádky, které byly otestovány. Řádky, které nebyly otestovány, jsou zvýrazněny oranžově viz obrázek 7.2.

```
202              :
203            0 :   List<String> getAllFolders() {
204            0 :     List<String> folders = [];
205            0 :     for (MyTreeNode node in HierarchyDatabase.roots) {
206            0 :       getFolders(folders, node);
207              :     }
208              :     return folders;
209              :   }
210              :
211            1 :   void getFolders(List<String> folders, MyTreeNode node) {
212            1 :     if (!node.isNote) {
213            2 :       folders.add(node.id);
214              :     }
215            2 :     for (MyTreeNode child in node.children) {
216            1 :       getFolders(folders, child);
217              :     }
218              :   }
```

Obrázek 7.2: Označení otestovaných a neotestovaných řádek kódu

7.2 Manuální testování

Automatické testy dokážou rychle a opakovaně otestovat velkou část kódu, nezaručují však, že je otestovaný kód bezchybný. Některé aspekty programu se navíc testují automaticky velice obtížně. Proto naší aplikaci prověříme několika testovacími scénáři, při kterých si budeme všímat reakcí a vzhledu systému.

Jelikož se aplikace může chovat odlišně na různých zařízeních. Otestujeme aplikaci na množině mobilních zařízení, emulátorech i v několika webových prohlížečích, abychom si byli jisti, že aplikace bude bez problému fungovat na většině zařízení.

7.2.1 Zařízení

Pro manuální testování byla aplikace otestována na několika mobilních zařízeních a emulátorech s různou verzí Androidu. Podrobný přehled zařízení je dostupný v tabulce 7.1. Abychom si byli jisti s funkčností webové verze aplikace, otestovali jsme webovou verzi aplikace v prohlížečích Google Chrome, Mozilla Firefox a Microsoft Edge na počítači i mobilním zařízení Xiaomi Redmi Note 7, abychom věděli, že se aplikace chová a zobrazuje správně.

Tabulka 7.1: Seznam zařízení, na kterých byla mobilní verze aplikace testována

Název zařízení	Verze Androidu	Rozlišení	Typ zařízení
Pixel 3a	14 (API 34)	1080x2220	virtuální
Pixel 8 Pro	14 (API 34)	1344x2992	virtuální
Pixel Fold	11 (API 30)	1840x2208	virtuální
Redmi Note 7	10 (API 29)	2340x1080	fyzické
Redmi Note 11	13 (API 33)	2400x1080	fyzické

7.2.2 Testovací scénáře

Následujícími testovacími scénáři byla ověřena základní funkcionalita aplikace:

Vytvoření hierarchie a editace poznámky

Popis: Cílem tohoto testu je otestovat, zda lze vytvořit hierarchickou strukturu. V hierarchické struktuře bude následně vytvořena poznámka, ve které bude napsán formátovaný text. Následně bude zkontrolováno, zda jsou data aplikace uchována i po zavření aplikace. Jedinou prerekvizitou pro tento test je česká lokalizace aplikace.

Postup:

1. Zobrazíme vysouvací menu a stiskneme tlačítko *Nová složka*. Po stisknutí tlačítka by se mělo zobrazit dialogové okno s titulkem *Nová složka*.
2. Zadáme do textového pole text *Škola* a stiskneme tlačítko *Vytvořit*. Výsledkem by měla být nově vytvořená složka v hierarchickém stromě.
3. Stiskneme tři tečky na konci řádku složky. Mělo by se zobrazit menu obsahující položky *Smazat, Přejmenovat, Nová poznámka, Nová složka, Přesunout*.
4. Stiskneme položku v menu s názvem *Nová složka* a obdobným způsobem jako v bodě 1 vytvoříme složku s názvem *UPS*. V hierarchickém stromě by se měla zobrazit složka s názvem *UPS*, která je potomkem složky *Škola*.
5. Klikneme na název složky *UPS*. Měla by se zobrazit obrazovka s tlačítkem o úroveň výš a textový řetězec *Tato složka je prázdná*. V horní liště by měla být zobrazena aktuální cesta |*Škola*|*UPS*.
6. Stiskneme plovoucí tlačítko v pravém dolním rohu obrazovky a zvolíme možnost *Nová poznámka* (ikona článku). Objeví se dialogové okno, do kterého zadáme text *Přednáška 1* a stiskneme tlačítko *Vytvořit*. Dialogové okno by se po stisknutí tlačítka mělo zavřít a v seznamu by se měla objevit poznámka s názvem *Přednáška 1*.
7. Stiskneme položku seznamu *Přednáška 1*. Měli bychom být přesměrováni do editoru formátovaného textu.
8. Zadáme libovolný text a aplikaci zavřeme.
9. Spustíme aplikaci a otevřeme vysouvací menu. Ve stromové struktuře bychom měli najít poznámku *Přednáška 1*.
10. Otevřeme poznámku kliknutím na jméno poznámky a zkontrolujeme, zda obsah poznámky souhlasí s tím, co jsme napsali před zavřením aplikace.

Výsledek testu: V rámci testu bylo zjištěno, že při tvorbě kořenového adresáře se tento soubor vytvořil, avšak stromová struktura tento soubor nezobrazovala. Při vytváření podřízených souborů a editace poznámky nebyly nalezeny žádné chyby.

Úprava a smazání poznámky

Popis: Cílem testu je otestovat, zda lze poznámku přejmenovat a přesunout bez ztráty dat. Na konci testu se pokusíme poznámku smazat. Prerekvizitou testu je hierarchie vytvořena v předešlém testu.

Postup:

1. Otevřeme vysouvací menu a v hierarchickém stromě najdeme poznámku s názvem *Přednáška 1*.
2. Zmáčkneme tři tečky na konci řádku. Zobrazí se dialogové okno s titulkem *Přejmenovat Přednáška 1*. Do dialogového okna zadáme nový název *Poznámka* a stiskneme tlačítko *Přejmenovat*. Dialogové okno by se mělo zavřít a ve stromě by se mělo změnit jméno souboru.

3. Stiskneme tlačítko pro vytvoření nové složky a vytvoříme novou složku, kterou pojmenujeme *Práce*.
4. Stiskneme tři tečky u poznámky, která byla přejmenována v prvním kroku, a v menu zvolíme možnost *Přesunout*. Zobrazí se dialogové okno s rozbalovacím seznamem, ve kterém by se měly nacházet 2 položky: |*Škola* a |*Práce*. Zvolíme možnost |*Práce*. Dialogové okno by se mělo uzavřít a poznámka by měla být přesunuta do složky *Práce*.
5. Přesunutou poznámku otevřeme a zkontrolujeme, zda se obsah této poznámky nezměnil.
6. Otevřeme vysouvací menu, stiskneme tři tečky u poznámky a stiskneme volbu *Smazat*. Mělo by se zobrazit potvrzovací dialogové okno.
7. V dialogovém okně stiskneme tlačítko *Smazat*. Dialogové okno by se mělo zavřít a poznámka by měla být odebrána ze stromové struktury.

Výsledek testu: Během testu došlo k přejmenování, přesunutí a smazání poznámky. Nebyly nalezeny žádné chyby.

Registrace uživatele

Popis: V tomto testovacím scénáři ověříme, že uživateli je umožněno založit si nový účet. Současně zkusíme některé chybové stavy, které mohou při registraci nastat. Prerekvizitou je nastavení české lokalizace, stabilní internetové připojení, nepřihlášený uživatel a neexistující účet *test@gmail.com*.

Postup:

1. Otevřeme vysouvací menu a klikneme na tlačítko *Registrovat*. Po této akci bychom měli být přesměrováni na obrazovku registrace.
2. Nevyplníme žádné textové pole a stiskneme tlačítko *Registrovat*. Mělo by se zobrazit chybové hlášení: *Jedno nebo více textových polí je prázdné!*
3. Do textového pole pro e-mailovou adresu zadáme *test@gmail.com*, do textového pole určeného pro heslo zadáme textový řetězec *heslo*, do textového pole pro ověření hesla zadáme *Heslo123* a stiskneme tlačítko *Registrovat*. Výsledkem bude chybové oznámení: *Hesla se neshodují!*
4. Do obou textových polí pro zadání hesla zadáme *heslo* a stiskneme tlačítko pro registraci. Výsledkem bude chybové oznámení: *Heslo je příliš krátké. Heslo musí obsahovat nejméně 6 znaků!*
5. Do obou textových polí pro zadání hesla zadáme *Heslo123*, obsah textového pole e-mailu změníme na *test* a stiskneme tlačítko *Registrovat*. Výsledkem bude chybové oznámení: *Špatný formát e-mailové adresy!*
6. Změníme opět e-mailovou na původní hodnotu *test@gmail.com* a stiskneme tlačítko *Registrovat*. Mělo by se zobrazit načítání. Po krátké době bychom měli být přesměrováni na obrazovku s otevřeným vysouvacím menu. V hlavičce vysouvacího menu by se mělo zobrazit: *Přihlášený uživatel: test@gmail.com* a tlačítko *Odhlásit*.

Výsledek testu: V rámci testování byla ověřena funkce registrace nového uživatele. Program se choval dle uvedeného popisu.

Přihlášení e-mailem a heslem

Popis: V tomto testu ověříme, že uživatel je schopen se přihlásit ke svému existujícímu účtu. Prerekvizitou testovacího scénáře je nastavení české lokalizace, nepřihlášený uživatel, stabilní internetové připojení, neexistující účet *notexist@gmail.com* a existující účet *test@gmail.com* s heslem *Heslo123*.

Postup:

1. Otevřeme vysouvací menu a klikneme na tlačítko *Přihlásit*. Po této akci bychom měli být přesměrováni na obrazovku přihlášení.
2. Nevyplníme žádné textové pole a stiskneme tlačítko *Přihlásit*. Mělo by se zobrazit chybové hlášení: *Jedno nebo více textových polí je prázdné!*
3. Do prvního textového pole zadáme e-mailovou adresu *test@gmail.com* a stiskneme tlačítko *Přihlásit*. Výsledkem bude chybové oznámení: *Špatná e-mailová adresa nebo heslo!*
4. Do textového pole hesla zadáme textový řetězec *Heslo123* a stiskneme tlačítko *Přihlásit*. Měli bychom vidět indikátor načítání a následně bychom měli být přesměrováni na obrazovku s otevřeným vysouvacím menu. V hlavičce vysouvacího menu by se mělo zobrazit: *Přihlášený uživatel: test@gmail.com* a tlačítko Odhlásit. Data, která byla uložena na účtu by měla být uložena a zobrazena v aplikaci.

Výsledek testu: Během testování došlo k přihlášení k existujícímu účtu. Uživatelská data byla uložena. Test byl úspěšný.

Přihlášení účtem Google

Popis: V tomto testovacím scénáři ověříme, zda je uživatel schopen se přihlásit pomocí účtu Google. Prerekvizitou testovacího scénáře je nastavení české lokalizace, nepřihlášený uživatel, stabilní internetové připojení, existující Google účet, ke kterému má tester přístup.

Postup:

1. Otevřeme vysouvací menu a klikneme na tlačítko *Přihlásit*. Po této akci bychom měli být přesměrováni na obrazovku přihlášení.
2. Klikneme na ikonu Google ve spodní části obrazovky. Po stisknutí by se mělo zobrazit přihlašovací okno pro přihlášení Google účtem.
3. Zvolíme svůj Google účet. Aplikace by nás měla přihlásit a přesměrovat zpět na vysouvací menu. Zároveň stáhnout uložená data, která jsou na cloudu uložena.

Výsledek testu: Ve webové verzi aplikace byla nalezena chyba při přihlášení účtem Google. V mobilní verzi aplikace bylo přihlašování účtem Google úspěšné.

Odhlášení přihlášeného uživatele

Popis: V tomto testu se ověříme funkcionalitu odhlášení. Prerekvizitou je přihlášený uživatel a data vytvořená v testovacím scénáři: Vytvoření hierarchie a editace poznámky.

Postup:

1. Zobrazíme vysouvací menu. V hlavičce vysouvacího menu by měl být text *Přihlášený uživatel* s e-mailovou adresou přihlášeného uživatele. Pod textem by se mělo nacházet tlačítko *Odhlásit*. V hierarchickém stromě by se měla nacházet uložená data.
2. Stiskneme tlačítko *Odhlásit*. Hlavička vysouvacího menu by měla změnit svůj obsah na text *Nepřihlášený uživatel* a zobrazit tlačítka *Přihlásit* a *Registrovat*. Všechna data v hierarchickém stromě by měla být smazána.

Výsledek testu: Uživatel byl úspěšně odhlášen a data byla smazána.

Změna zapomenutého hesla

Popis: V tomto testu ověříme, zda je uživatel schopen změnit si heslo, pokud jej zapomene. Prerekvizitou testovacího scénáře je nastavení české lokalizace, nepřihlášený uživatel, existující účet *lrunt@students.zcu.cz* (nebo jiný, ke kterému má tester přístup) a otevřená obrazovka pro přihlášení uživatele.

Postup:

1. Na obrazovce pro přihlášení uživatele stiskneme odkaz *Obnovit heslo*. Měli bychom být přesměrováni na stránku pro zaslání e-mailu s odkazem pro obnovu hesla s jedním textovým polem a jedním tlačítkem.
2. Do textového pole zadáme e-mailovou adresu *lrunt@students.zcu.cz* a stiskneme tlačítko *Odeslat e-mail*. Výsledkem bude zobrazený Toast s textem *E-mail pro obnovení hesla byl odeslán* a přesměrování na obrazovku pro přihlášení uživatele.
3. Zkontrolujeme e-mailovou schránku. Měl by nám přijít e-mail s předmětem *Reset your password for Notes* a odkazem na stránku pro změnu hesla (odesílání e-mailu může trvat několik minut).
4. Otevřeme hypertextový odkaz, který nám přišel v e-mailu. Měli bychom být přesměrováni na stránku pro změnu hesla. Do textového pole na otevřené stránce zadáme textový řetězec *password* a stiskneme tlačítko *Save*.
5. Nyní se zkusíme v aplikaci Notes přihlásit s novými přihlašovacími údaji e-mailem *lrunt@students.zcu.cz* a heslem *password*. Měli bychom být přihlášeni a přesunuti na obrazovku s otevřeným vysouvacím menu.

Výsledek testu: Odesílání e-mailu ve většině případů trvalo trochu déle. Heslo bylo ve všech případech změněno dle předpokladů.

Synchronizace dat bez konfliktu

Popis: Tento testovací scénář testuje, že je aplikace schopna synchronizovat uživatelská data bez vyvolaných konfliktů. Prerekvizitou jsou dvě zařízení, na kterých je aplikace spuštěna, přihlášení na obou zařízení ke stejnému účtu.

Postup:

1. Na zařízení 1 vytvoříme složku, kterou nazveme *Synchronizace* a stiskneme tlačítko *Synchronizace*.
2. Na zařízení 2 stiskneme tlačítko *Synchronizace*. Složka se jménem *Synchronizace* by se měla zobrazit v hierarchickém stromě.
3. Na zařízení 2 vytvoříme ve složce *Synchronizace* poznámku s libovolným jménem. Poznámku otevřeme a zadáme libovolný text. Po úpravě stiskneme tlačítko *Synchronizace*.
4. Na zařízení 1 stiskneme tlačítko *Synchronizace* a zkontrolujeme správnost synchronizovaných dat.

Výsledek testu: Během testování nebyly objeveny žádné chyby. Data byla bez problémů synchronizována.

Synchronizace dat s vyvoláním konfliktů

Popis: Tento testovací scénář testuje detekci a řešení konfliktů. Prerekvizitami jsou dvě zařízení, na kterých je aplikace spuštěna, přihlášení na obou zařízeních ke stejnému účtu a nějaká počáteční data.

Postup:

1. Na zařízení 1 vytvoříme složku s libovolným jménem a stiskneme tlačítko *Synchronizace*.
2. Na zařízení 2 stiskneme tlačítko *Synchronizace*. Složka se jménem *Synchronizace* by se měla zobrazit v hierarchickém stromě.
3. Na zařízení 2 v námi vytvořené složce vytvoříme poznámku libovolně pojmenovanou poznámku. Poznámku otevřeme a zadáme libovolný text. Tím na zařízení 2 skončíme.
4. Na zařízení 1 vytváříme v námi vytvořené složce poznámku, definujeme její obsah a stiskneme tlačítko *Synchronizace*.
5. Na zařízení 2 stiskneme tlačítko *Synchronizace*. Měl by se objevit Toast s textem, že se byly detekovány konflikty. V hierarchickém stromě by se měl objevit obsah zařízení 1.
6. Na zařízení 2 otevřeme nastavení a stiskneme možnost *Konflikty*. V konfliktech by měla být uložena námi vytvořená struktura s časovou značkou, kdy byla synchronizace provedena.
7. Stiskneme tři tečky na konci řádku a zkusíme tento konflikt smazat. Konflikt by měl být smazán.

Výsledek testu: Během testu byl detekován konflikt dat a data byla uložena na očekávaném místě s časovou značkou. Smazání konfliktních dat proběhlo taktéž bez problémů.

Změna lokalizace

Popis: V tomto testu zkontrolujeme, zda se všechny texty správně překládají. Testovací scénář nemá žádné prerekvizity.

Postup:

1. Zkoušíme procházet všechny obrazovky a dialogová okna aplikace. Všímáme si, zda jsou všechny texty ve správném jazyce.
2. Otevřeme menu aplikace (Ozubené kolo v pravém horním rohu).
3. V poli Jazyk zvolíme druhý jazyk.
4. Opět procházíme celou aplikaci a sledujeme, zda jsou všechny textové řetězce ve správném jazyce.

Výsledek testu: V aplikaci bylo nalezeno pár řetězců, které byly buď umístěny špatně (například *název poznámky* v dialogovém okně pro založení adresáře), nebo neexistoval jejich překlad.

Změna barevného módu

Popis: V tomto testu ověříme správný vzhled a přepínání témat. Testovací scénář nemá žádné prerekvizity.

Postup:

1. Zkoušíme procházet všechny obrazovky a dialogová okna aplikace. Všímáme si, zda všechny komponenty mají správnou barvu a správné rozložení.
2. Otevřeme menu aplikace (Ozubené kolo v pravém horním rohu).
3. V poli Tmavý mód stiskneme přepínač, kterým by se mělo změnit téma aplikace.
4. Procházíme celou aplikaci a kontrolujeme, zda všechny komponenty odpovídají aktuálně zvolenému barevnému tématu.

Výsledek testu: V rámci testu byl zkontrolován vzhled aplikace. Rozložení a velikost některých komponent (například rozložení tlačítek *Přihlásit* a *Registrovat* ve vysouvacím menu) bylo změněno.

7.3 Hodnocení kvality aplikace

V při vývoji je vhodné vyhodnotit kvalitu vytvářené aplikace. Kvalita aplikací většinou ovlivňuje uživatelskou spokojenost a funkčnost aplikace. Normy „Core App Quality"[2] poskytované společností Google definují kritéria pro hodnocení aplikací. Tyto standardy umožňují systematicky vyhodnotit kvalitu aplikace.

Tabulka 7.2: Kritéria „Core App Quality", které aplikace splňuje

Kategorie	**ID splněného kritéria**
Vizuální zážitek	VX-N1, VX-N2, VX-N3, VX-S1, VX-S2, VX-V1, VX-V2, VX-V3, VX-A1, VX-A2
Funkcionalita	FN-B1
Výkon a stabilita	PS-S1, PS-P1, PS-T1, PS-T2, PS-T5, PS-T6
Ochrana soukromí a zabezpečení	SC-P1, SC-P2, SC-P3, SC-P4, SC-DF1, SC-DF2, SC-DF3, SC-N1, SC-C1

V tabulce 7.2 jsou uvedena kritéria, která vyvíjená aplikace splňuje. Tato kritéria jsou rozdělena dle kategorií. Kritérium VX-V3 v kategorii vizuální zážitek například znamená, že aplikace a veškerý webový obsah, na který aplikace odkazuje, podporuje tmavý režim.

[2] https://developer.android.com/docs/quality-guidelines/core-app-quality

Možná rozšíření 8

Od začátku vývoje se aplikace vytvářela se snahou na bezproblémové přidání dalších funkcí. V následující kapitole jsou popsány možná rozšíření již námi vytvořené aplikace. Jedná se o rozšíření zavádějící novou funkcionalitu, zlepšující komfort uživatelů a rozšíření podpory na více platforem.

8.1 Spolupráce více uživatelů

V současné verzi aplikace může uživatel synchronizovat poznámky mezi svými zařízeními v rámci svého účtu. Někdy je však potřeba sdílet poznámky s více uživateli nebo spolupracovat v týmu. Dalším rozšířením by mohlo být sdílení a spolupráce více uživatelů na jedné poznámce. Uživatelé by měli možnost sdílet libovolný soubor s dalším uživatelem, kterému by mohli přidávat a odebírat oprávnění, jak může s daným souborem zacházet.

Pro tuto funkci by bylo potřeba rozšířit datový model informacemi obsahující data o sdílení poznámek. Každý uživatel by mohl mít ve své kolekci informace o souborech a poznámkách, které jsou s ním sdíleny. V klientovi by byla přidána do menu položka sdílet s uživatelem, která by zobrazovala dialog s nastavením práv a uživatelem, se kterým by byly poznámky sdíleny. V tomto kroku by se muselo kontrolovat, zda už poznámka není s uživatelem sdílena, aby nedošlo k duplikaci informací.

8.2 Obsah poznámek

Aktuální verze aplikace poskytuje ukládání poznámek ve formě formátovaného textu, což by některým uživatelům nemuselo stačit. Dalším krokem by tedy mohlo být rozšíření typů obsahu. Kromě textu by aplikace mohla podporovat ukládání obrázků, videí, audio nahrávek nebo skic. Tato média by mohla být vkládána ze souborového systému nebo by se příslušné digitální soubory mohli vytvářet přímo v aplikaci s použitím hardwarového vybavení zařízení.

8.3 Historie změn

Pro zvýšení kontroly nad daty by mohla být zavedena historie změn na daném zařízení, která by uživatelům umožňovala vrátit se k libovolné verzi svých poznámek. Tato funkcionalita by mohla přispět k efektivnějšímu řešení konfliktů.

8.4 Rozšíření na více platforem

Aplikace je vytvořená s použitím technologie Flutter, která nabízí multiplatfomní vývoj aplikací. Tato technologie umožňuje rozšířit aplikaci na další platformy jako jsou iOS, macOS, Windows, Linux a Google Fuchsia. Pro zveřejnění aplikace na těchto platformách by bylo nutné udělat specifické úpravy ve vzhledu (například použití Cupertino Design pro iOS, upravení rozložení komponent), konfiguraci služby Firebase a úprava závislostí knihoven. Před zveřejněním by bylo vhodné aplikaci znovu otestovat manuálními testy k ověření správné funkčnosti.

Závěr 9

Cílem této bakalářské práce bylo navrhnout a implementovat mobilní aplikaci pro zařízení Android, která bude umožňovat tvorbu hierarchických poznámek s libovolným počtem úrovní. Další požadovanou vlastností aplikace byla synchronizace dat mezi více zařízeními, bezplatné použití a dobrá rozšířitelnost aplikace. Vzhledem k tomu, že většina dostupných aplikací kombinaci těchto požadavků nesplňuje, rozhodli jsme se k vytvoření vlastní aplikace.

V úvodu práce byl proveden podrobný průzkum existujících aplikací, ve kterém bylo zjištěno, jaké funkce tyto aplikace nabízejí. Z těchto poznatků byly nastíněny funkce, které by měla výsledná aplikace obsahovat. V další části proběhla analýza technologií pro vývoj mobilních aplikací a ukládání dat, na základě kterého byly zvoleny technologie Flutter a Firebase. Po výběru konkrétních technologií byly specifikovány případy použití vytvářené aplikace, které byly zohledněny v následujícím návrhu aplikace. V tomto návrhu byl navržen datový model a uživatelské rozhraní vytvářené aplikace. V další fázi byla navržená aplikace úspěšně realizována.

Vytvořená aplikace je dostupná nejen na mobilních zařízeních s operačním systémem Android, ale také jako progresivní webová aplikace, která je veřejně dostupná. Aplikace umožňuje vytvářet a organizovat poznámky s hierarchickou strukturou. Vytvořené poznámky lze synchronizovat mezi více zařízeními s detekcí potenciálních konfliktů. Mezi další zajímavé funkce a vlastnosti aplikace můžeme zařadit tmavý mód, vícejazyčnou podporu, rozšiřitelnost kódu a bezplatné použití. Pro podporu uživatelů byla vytvořena webová stránka s návodem, jak aplikaci ovládat. V závěru práce byla aplikace otestována komplexní sadou testů a byla navržena možná rozšíření.

Slovník

AES Advanced Encryption Standard 10, 16, 17, 19, 34, 75

AOT Ahead Of Time 27, 75

API Application Programming Interface 16, 17, 24–27, 30, 35–37, 39, 58, 64, 75, 107

APK Android application PacKage 58, 75, 107

ARB Application Resource Bundle 56, 75

AWS Amazon Web Services 36, 38, 39, 75

CLI Command Line Interface 39, 75, 110, 115

CSS Cascading Style Sheet 30, 75

GPS Global Positioning System 21, 75

HSTS HTTP Strict Transport Security 9, 19, 75

HTML Hypertext Markup Language 12, 13, 30, 63, 75

HTTP Hypertext Transfer Protocol 35, 75

HTTPS Hypertext Transfer Protocol Secure 11, 19, 75

JIT Just In Time 27, 75

JSI JavaScript Interface 25, 75, 85

JSON JavaScript Object Notation 15, 17, 25, 31, 36, 37, 53, 75

JVM Java Virtual Machine 22, 75

LCOV Linux/Graphical Code Coverage 63, 75

NoSQL Not Only SQL 35–37, 39, 44, 75

OCR Optical Character Recognition 9, 11, 75

PDF Portable Document Format 8, 75, 119

PWA Progressive Web Apps 21, 31, 32, 38, 75

REST REpresentational State Transfer 35, 36, 75

SD Secure Digital 33, 75

SDK Software Development Kit 26, 75

SHA Secure Hash Algorithm 52, 75, 86, 112, 113

SQL Structured Query Language 34–36, 75

SSL Secure Sockets Layer 9, 19, 38, 75

TLS Transport Layer Security 9, 11, 16, 19, 75

UI User Interface 39, 75

URL Uniform Resource Locator 13, 16, 35, 75, 117

UX User Experience 39, 75

WYSIWYG What You See Is What You Get 12, 75

XML Extensible Markup Language 36, 75

Bibliografie

1. EVERNOTE CORPORATION. *Evernote* [online]. 2023. [cit. 2023-11-20]. Dostupné z: https://evernote.com/.
2. MICROSOFT. *OneNote help* [online]. 2023. [cit. 2023-11-21]. Dostupné z: https://support.microsoft.com/en-us/onenote.
3. COZIC, Laurent. *Joplin* [online]. 2023-10-30. [cit. 2023-11-27]. Dostupné z: https://joplinapp.org/help/.
4. NOTION LABS INC. *Notion* [online]. 2023. [cit. 2023-12-07]. Dostupné z: https://www.notion.so/help/reference.
5. OBSIDIAN. *Obsidian* [online]. 2023. [cit. 2023-12-09]. Dostupné z: https://obsidian.md/.
6. BAVOSA, Alan. *Native vs Non-Native Mobile Apps - What's the Difference?* [online]. The startup, 2021-09-09 [cit. 2024-02-02]. Dostupné z: https://medium.com/swlh/native-vs-non-native-mobile-apps-whats-the-difference-b3a641e06f52.
7. KOFFER, Peter. *What is a Native Mobile App Development?* [online]. 2023-09-05. [cit. 2024-02-09]. Dostupné z: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development-.
8. MURUGESH, Yogesh. *What is Native mobile app development? Pros and cons of building applications natively.* [online]. 2023-09-29. [cit. 2024-02-05]. Dostupné z: https://mallow-tech.com/blog/what-is-native-mobile-app-development-pros-and-cons-of-building-applications-natively/.
9. MONUS, Anna. *The 2023 guide to native app development* [online]. 2023-01-31. [cit. 2024-03-26]. Dostupné z: https://raygun.com/blog/native-app-development/.
10. ZOPP, Michal. *Cross-platform Mobile Application Development Comparison*. 2020. Diplomová práce. Masarykova univerzita, Fakulta informatiky.
11. KOTLIN. *Kotlin docs* [online]. [cit. 2024-02-06]. Dostupné z: https://kotlinlang.org/docs/home.html.

12. LUTKEVICH, Ben. *What is Kotlin?* [online]. 2022-12-28. [cit. 2024-03-25]. Dostupné z: https://www.techtarget.com/whatis/definition/Kotlin?vgnextfmt=print.

13. MOSKALA, Marcin; WOJDA, Igor. *Android Development with Kotlin*. Birmingham, UK: Packt Publishing Ltd, 2017. ISBN 978-1-78712-368-7.

14. JEMEROV, Dmitry; ISAKOVA, Svetlana. *Kotlin in action*. Simon a Schuster, 2017. ISBN 9781617293290.

15. CHIDERA, Edeh Israel. *Kotlin VS Java – What's the Difference?* [online]. 2023-03-31. [cit. 2024-03-26]. Dostupné z: https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/.

16. HAVRYLEVSKYJ, Vladyslav. *All You Need To Know About Cross Platform App Development* [online]. 2023-10-25. [cit. 2024-02-03]. Dostupné z: https://yojji.io/blog/all-you-need-to-know-about-cross-platform-app-development.

17. WARCHOLINSKI, Matt. *What is React Native? All You Need to Know for 2024* [online]. 2024-01-11. [cit. 2024-03-18]. Dostupné z: https://brainhub.eu/library/what-is-react-native.

18. REACT NATIVE. *React Native* [online]. [cit. 2024-03-17]. Dostupné z: https://reactnative.dev/.

19. PATERSKA, Patrycja. *What is React Native And When to Use It For Your App in 2024 (Updated)* [online]. 2024-02-23. [cit. 2024-03-17]. Dostupné z: https://www.elpassion.com/blog/what-is-react-native-and-when-to-use-it.

20. GONCHARENKO, Oleg. *How To Hire React Native Developer - A Guide For The Founder* [online]. 2021-06-17. [cit. 2024-03-18]. Dostupné z: https://brocoders.com/blog/hire-react-native-developer/.

21. KOSMAL, Jakub. *How does React Native work? Understanding the architecture* [online]. 2022-11-11. [cit. 2024-03-18]. Dostupné z: https://medium.com/front-end-weekly/how-does-react-native-work-understanding-the-architecture-d9d714e402e0.

22. FLUTTER. *Flutter documentation* [online]. [cit. 2024-02-09]. Dostupné z: https://docs.flutter.dev/.

23. BARTOSIŃSKA, Inez; DEMBNY, Michał. *What is flutter and is it the best solution to build your app?* [online]. 2023-08-07. [cit. 2024-02-16]. Dostupné z: https://www.thedroidsonroids.com/blog/what-is-flutter-app-development.

24. ALTEXSOFT. *The Good and the Bad of Flutter App Development* [online]. 2022-08-04. [cit. 2024-02-16]. Dostupné z: https://www.altexsoft.com/blog/pros-and-cons-of-flutter-app-development/.

25. JAVAPOINT. *Flutter Widgets* [online]. [cit. 2024-03-26]. Dostupné z: https://www.javatpoint.com/flutter-widgets.

26. INEZ, Bartosińska; BUCZKOWSKI, Tymoteusz. *Pros and Cons of Flutter App Development* [online]. 2023-07-06. [cit. 2024-03-19]. Dostupné z: https://www.thedroidsonroids.com/blog/flutter-pros-and-cons.

27. SHIOTSU, Yoshitaka. *What Is a Hybrid App? (Detailed Guide for 2024)* [online]. 2021-11-22. [cit. 2024-03-26]. Dostupné z: https://www.upwork.com/resources/hybrid-app.

28. APPSFLYER LTD. *Hybrid app* [online]. 2023-09-13. [cit. 2024-02-11]. Dostupné z: https://www.appsflyer.com/glossary/hybrid-app/.

29. GRIFFITH, Chris. *What is Hybrid Mobile App Development?* [online]. 2019-02-06. [cit. 2024-03-17]. Dostupné z: https://ionic.io/resources/articles/what-is-hybrid-app-development.

30. FRAMEWORK, Ionic. *Ionic framework* [online]. [cit. 2024-02-06]. Dostupné z: https://ionicframework.com/.

31. JOANNA, Pokorska. *What is Ionic and how does it work?* [online]. 2022-10-20. [cit. 2024-02-08]. Dostupné z: https://appstronauts.co/blog/what-is-ionic-and-how-does-it-work/.

32. KOĎOUSKOVÁ, Barbora. *Co jsou progresivní webové aplikace (PWA) a jaké mají výhody* [online]. 2023-07-23. [cit. 2024-02-08]. Dostupné z: https://www.rascasone.com/cs/blog/progresivni-webova-aplikace-vyhody.

33. VUE STOREFRONT. *What is PWA? Progressive Web Apps in eCommerce Explained*. 2023-08-23. Dostupné také z: https://vuestorefront.io/blog/pwa.

34. DEVELOPERS, Android. *Data and file storage overview*. 2024-03-12. Dostupné také z: https://developer.android.com/training/data-storage.

35. LACKO, Ľuboslav. *Mistrovství-Android*. 1. vydání. Brno: Computer Press, 2017. ISBN 978-80-251-4875-4. Překlad: Herodek, Martin.

36. FLUTTER.DEV. *shared_preferences 2.2.2* [online]. 2023-10-10. [cit. 2024-02-29]. Dostupné z: https://pub.dev/packages/shared_preferences.

37. SQLITE. *What Is SQLite?* [online]. 2024-01-23. [cit. 2024-02-18]. Dostupné z: https://www.sqlite.org/index.html.

38. CHOI, Simon. *hive 2.2.3* [online]. 2022-01-20. [cit. 2024-02-19]. Dostupné z: https://pub.dev/packages/hive.

39. CHOI, Simon. *Hive Docs* [online]. 2022-08-11. [cit. 2024-02-19]. Dostupné z: https://docs.hivedb.dev/#/.

40. BIG DATA TRUNK. *In-house IT servers Vs Cloud Computing: Which is The Better Alternative for Organizations?* [online]. 2024-01-25. [cit. 2024-02-24]. Dostupné z: https://bigdatatrunk.medium.com/in-house-it-servers-vs-cloud-computing-which-is-the-better-alternative-for-organizations-18711d8c2533.

41. MANZOOR, Ahsan. *Understanding REST API: A comprehensive guide* [online]. 2023-03-30. [cit. 2024-03-11]. Dostupné z: https://datasciencedojo.com/blog/understanding-rest-api/.

42. TEAM, Codecademy. *What is REST?* [online]. [cit. 2024-03-13]. Dostupné z: https://www.codecademy.com/article/what-is-rest.

43. BLAIR, Ian. *How to Create a RESTful API For Your Mobile App (Fast)* [online]. 2021-09-13. [cit. 2024-03-11]. Dostupné z: https://buildfire.com/create-restful-api-mobile-app/.

44. SHELDON, Amyra. *Data Management and Storage Strategies for Mobile App Backend Development* [online]. 2023-06-27. [cit. 2024-03-10]. Dostupné z: https://medium.com/quick-code/data-management-and-storage-strategies-for-mobile-app-backend-development-f0779ccf6bc.

45. LUTKEVICH, Ben. *What is a relational database?* [online]. 2021-01-24. [cit. 2024-02-28]. Dostupné z: https://www.techtarget.com/searchdatamanagement/definition/relational-database?vgnextfmt=print.

46. AYUSHARMA0698. *Introduction to NoSQL*. Dostupné také z: https://www.geeksforgeeks.org/introduction-to-nosql/.

47. SHASHWAT, Saurabh. *Cloud storage: What is it and how does it work?* [online]. 2020-02-18. [cit. 2024-02-10]. Dostupné z: https://medium.com/@anukritisaurabh/cloud-storage-what-is-it-and-how-does-it-work-87265f4ebc65.

48. IBM. *What is cloud storage?* [online]. [cit. 2024-02-10]. Dostupné z: https://www.ibm.com/topics/cloud-storage.

49. AWS. *What is Cloud Storage?* [online]. [cit. 2024-02-10]. Dostupné z: https://aws.amazon.com/what-is/cloud-storage/.

50. FIREBASE. *Firebase* [online]. [cit. 2024-03-10]. Dostupné z: https://firebase.google.com/.

51. JAGDALE, Devang. *Firebase – Introduction* [online]. 2021-07-15. [cit. 2024-03-10]. Dostupné z: https://www.geeksforgeeks.org/firebase-introduction.

52. STEVENSON, Doug. *What is Firebase? The complete story, abridged.* [online]. 2018-08-25. [cit. 2024-03-10]. Dostupné z: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0.

53. FIREBASE. *Cloud Firestore* [online]. [cit. 2024-03-10]. Dostupné z: https://firebase.google.com/docs/firestore.

54. KIRVAN, Paul. *Amazon Web Services (AWS)* [online]. 2024-01-31. [cit. 2024-03-11]. Dostupné z: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services.

55. AMAZON WEB SERVICES. *Start building on AWS today* [online]. [cit. 2024-03-11]. Dostupné z: https://aws.amazon.com/.

56. AMAZON WEB SERVICES. *Amplify Documentation* [online]. [cit. 2024-03-12]. Dostupné z: https://docs.amplify.aws/flutter/.

57. JAVATPOINT. *What is AWS Amplify?* [online]. [cit. 2024-03-12]. Dostupné z: https://www.javatpoint.com/what-is-aws-amplify.

58. BAUMGARTEN, Matheus. *flutter_fancy_tree_view* [online]. 2024-04-24. [cit. 2024-04-29]. Dostupné z: https://pub.dev/packages/flutter_fancy_tree_view.

59. GOOGLE. *Testing Flutter apps* [online]. 2024-03-01. [cit. 2024-03-02]. Dostupné z: https://docs.flutter.dev/testing/overview.

Seznam obrázků

Seznam tabulek

Seznam výpisů

Uživatelská příručka

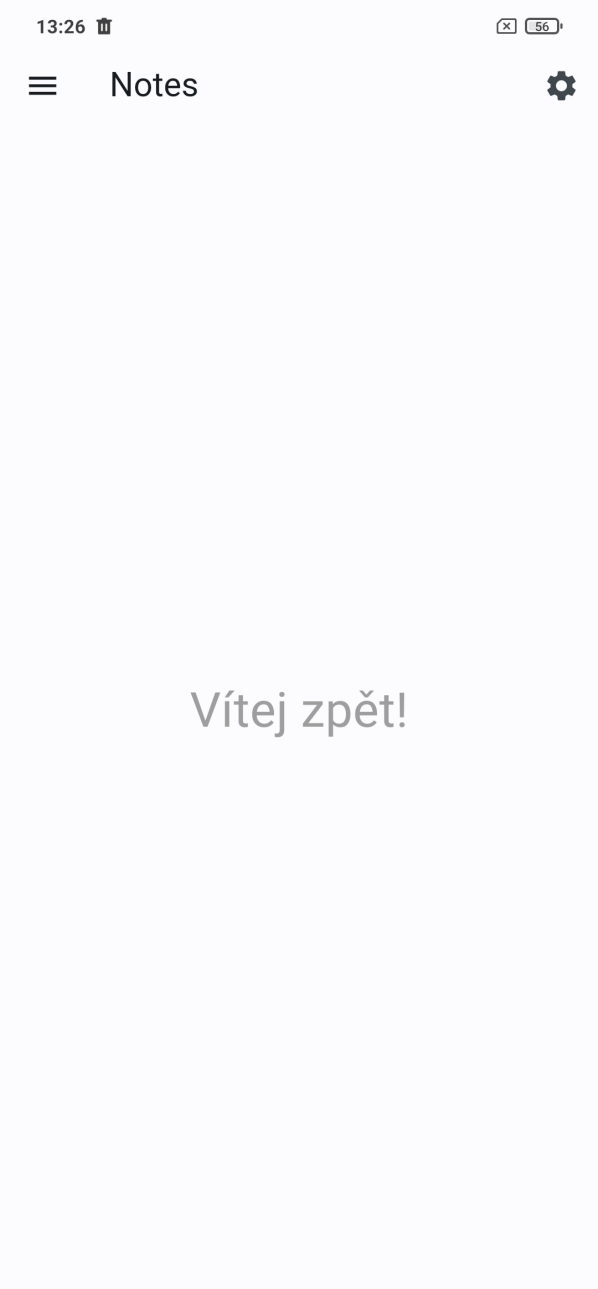
Tato příloha obsahuje uživatelskou příručku aplikace Notes. Cílem této příručky je poskytnout uživatelům jasné a přehledné informace o tom, jak ovládat aplikaci. Podrobnější návod je dostupný na webové stránce: `https://notes-manual.web.app/manual-cz.html`

A.1 Spuštění aplikace

Při spuštění aplikace se zobrazí inicializační obrazovka s logem aplikace viz obrázek A.1.

Obrázek A.1: Inicializační obrazovka

Když je aplikace inicializována, zobrazí se uživateli domovská obrazovka viz obrázek A.2.



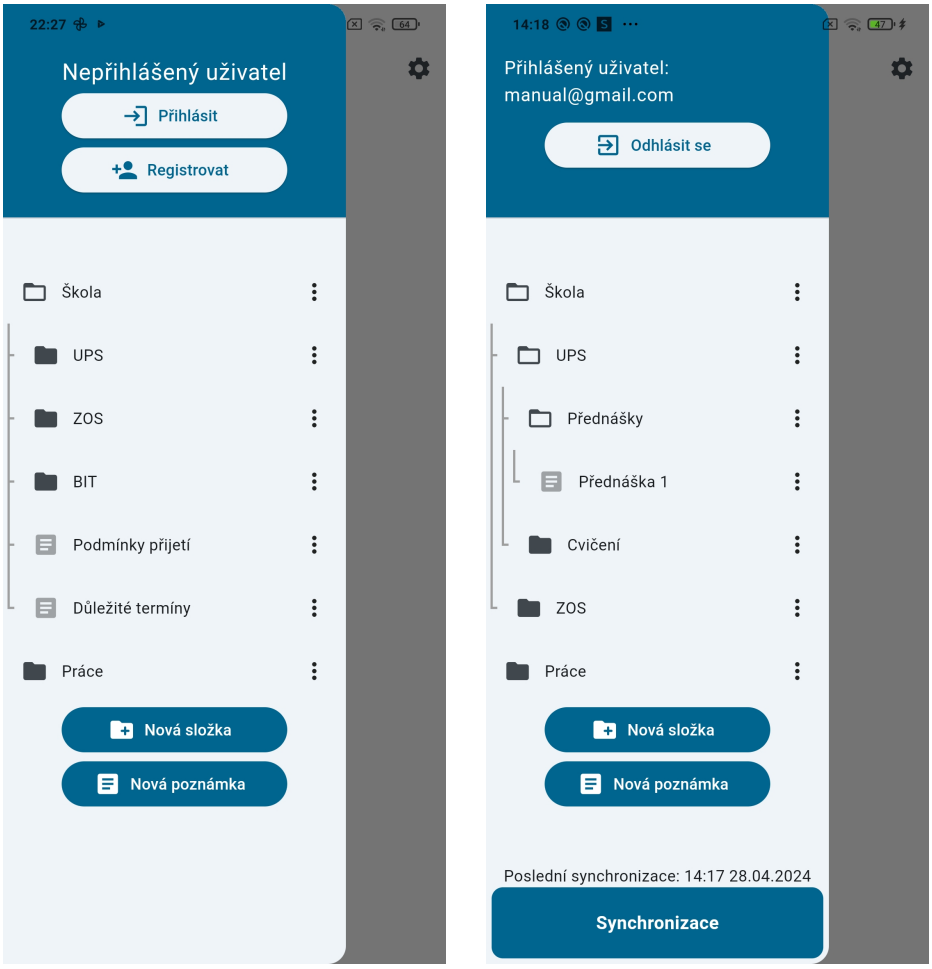
Obrázek A.2: Snímek úvodní obrazovky aplikace

Na obrazovce se nacházejí dvě tlačítka. V levém horním rohu jsou tři vodorovné čáry, které otevírají vysouvací menu popsané v sekci A.2. V pravém horním rohu se nachází ikona ozubeného kola, která otevírá nastavení aplikace. Stránka nastavení je popsána v sekci A.8.

A.2 Vysouvací menu

Vysouvací menu zobrazuje stav přihlášení uživatele. Pokud je uživatel přihlášen, nachází se v horní části menu informace o přihlášeném účtu a tlačítko pro odhlášení uživatele. Přihlášenému uživateli se zároveň ve spodní části menu zobrazuje tlačítko pro synchronizaci a čas poslední synchronizace. Nepřihlášení uživatelé mají možnost využít tlačítka *Přihlásit* a *Registrovat*.

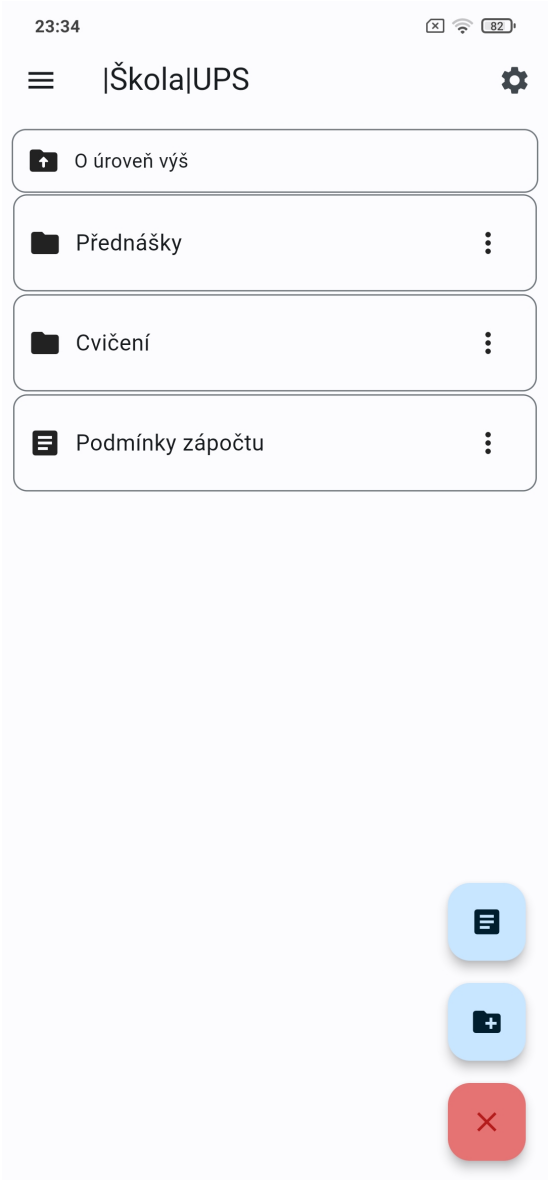
Ve vysouvacím menu je kromě stavu uživatele zobrazena také hierarchická struktura ve formě stromové komponenty viz obrázek A.3. Kliknutím na ikonu adresáře lze rozbalit nebo skrýt podřízené adresáře a poznámky. Pokud uživatel stiskne název souboru nebo ikonu poznámky, je automaticky přesunut na obrazovku adresáře (viz sekce A.3) či poznámky (viz sekce v sekce A.4). Každá složka ve stromové struktuře má na konci řádku tři tečky, které zobrazují menu s možnostmi úpravy poznámky či adresáře.



Obrázek A.3: Obrazovky s vysouvacím menu: vlevo vysouvací menu nepřihlášeného uživatele, vpravo vysouvací menu přihlášeného uživatele

A.3 Obrazovka adresáře

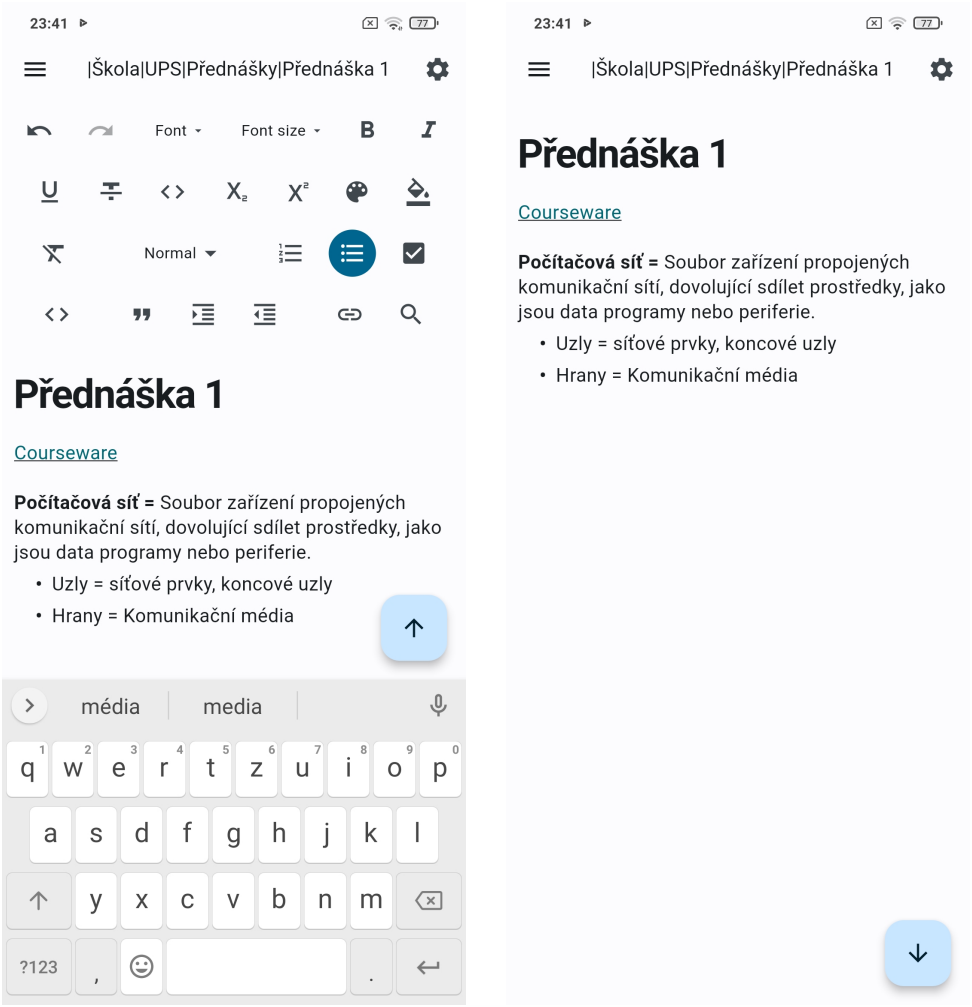
Na dalším obrázku A.4 je zobrazena obrazovka s obsahem adresáře. V aplikačním panelu je cesta adresáře, ve kterém se uživatel nachází. Podřízené soubory adresáře jsou zobrazeny v seznamu uprostřed obrazovky. Pokud uživatel stiskne některý ze souborů, je následně přesměrován do příslušného adresáře či poznámky. V pravém dolním rohu se nacházejí plovoucí tlačítka, kterými má uživatel možnost vytvářet nové adresáře a poznámky v aktuálním adresáři. Červené tlačítko s křížkem slouží ke skrytí plovoucích tlačítek nad ním.



Obrázek A.4: Snímek obrazovky, která zobrazuje obsah adresáře formou seznamu

A.4 Editace poznámky

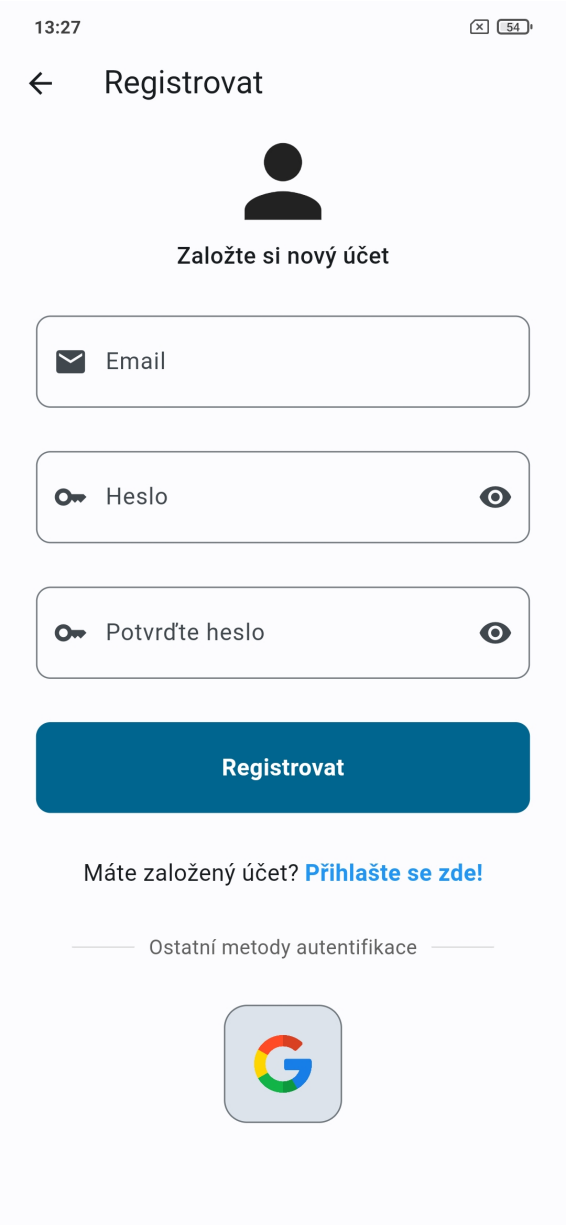
Na dalších snímcích A.5 je zobrazen editor formátovaného textu. V aplikačním panelu má uživatel informaci o tom, jakou poznámku zrovna edituje. Pomocí plovoucího tlačítka má uživatel možnost schovat či zobrazit panel nástrojů.



Obrázek A.5: Obrazovka úpravy poznámky, vlevo s viditelným panelem nástrojů, vpravo se skrytým panelem nástrojů

A.5 Registrace nového uživatele

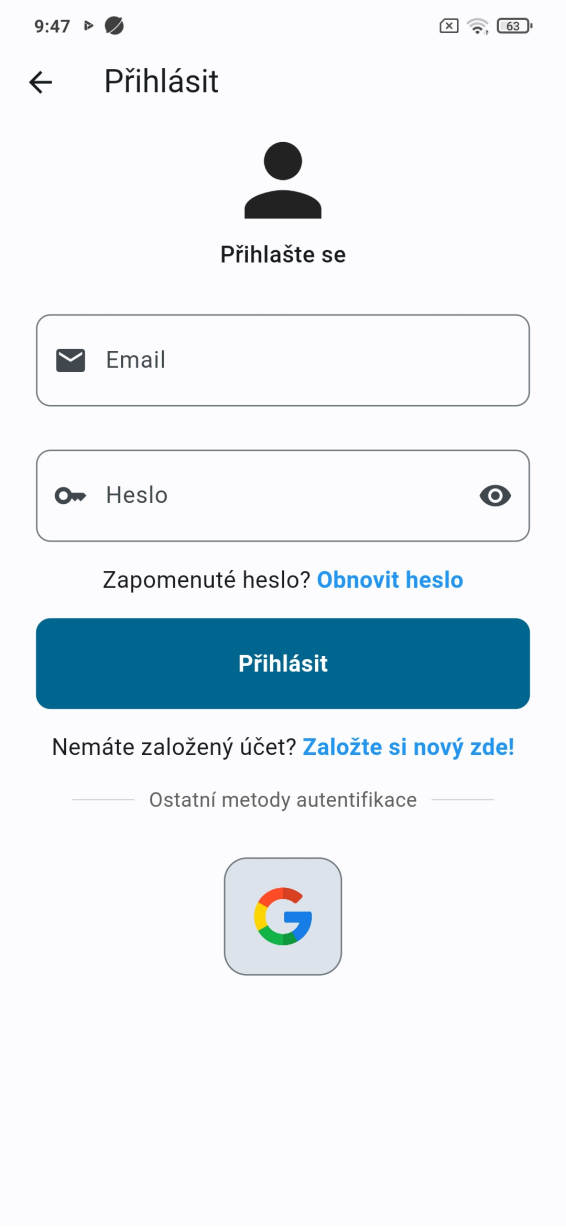
Na obrázku A.6 se nachází obrazovka pro registraci nového uživatele. Uživatel má možnost se registrovat kombinací e-mailu a hesla nebo pomocí účtu Google. Pokud má uživatel již založený účet, může se pomocí odkazu *Přihlašte se zde!* dostat na stránku s přihlášením.



Obrázek A.6: Snímek obrazovky pro registraci nového uživatele

A.6 Přihlášení uživatele

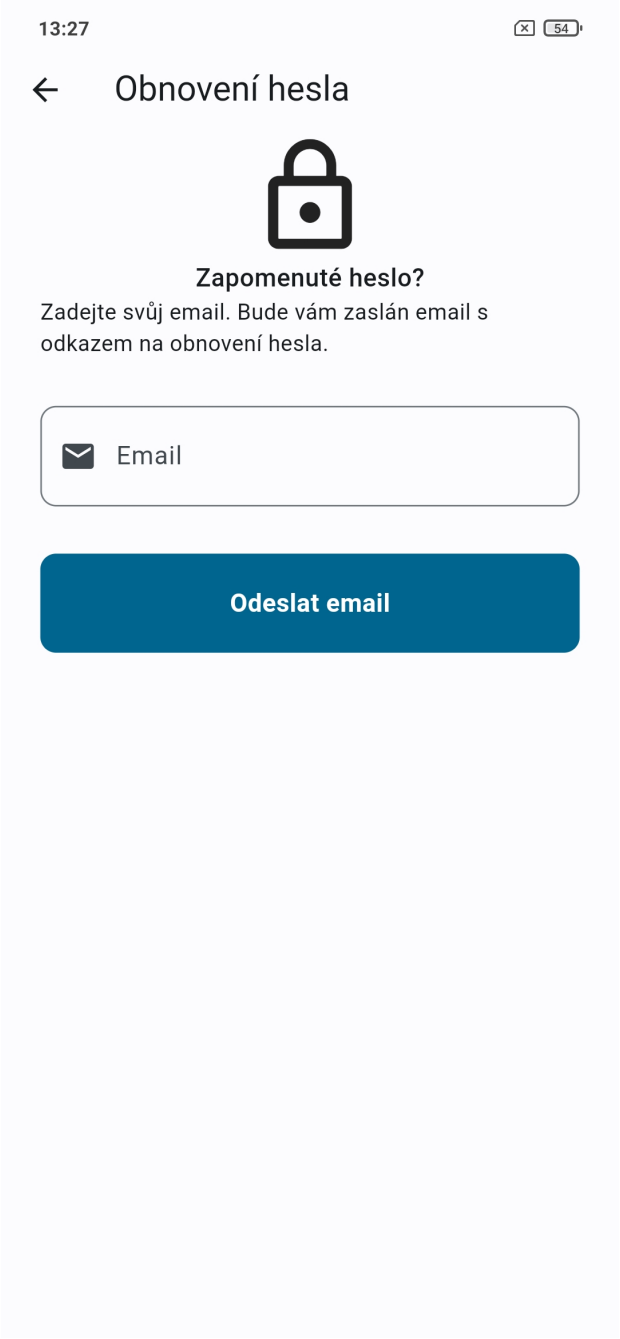
Uživatelé se mohou přihlásit buď použitím e-mailu a hesla, nebo prostřednictvím účtu Google. V případě zapomenutí hesla mají uživatelé možnost kliknout na odkaz *Obnovit heslo*, který je přesměruje na stránku pro obnovení hesla. Ti, kteří ještě nemají založený účet, mohou využít odkazu *Založte si nový zde!* pro přesměrování na stránku s registrací. Obrázek A.7 ukazuje stránku pro přihlášení.



Obrázek A.7: Snímek přihlašovací obrazovky

A.7 Obnovení hesla

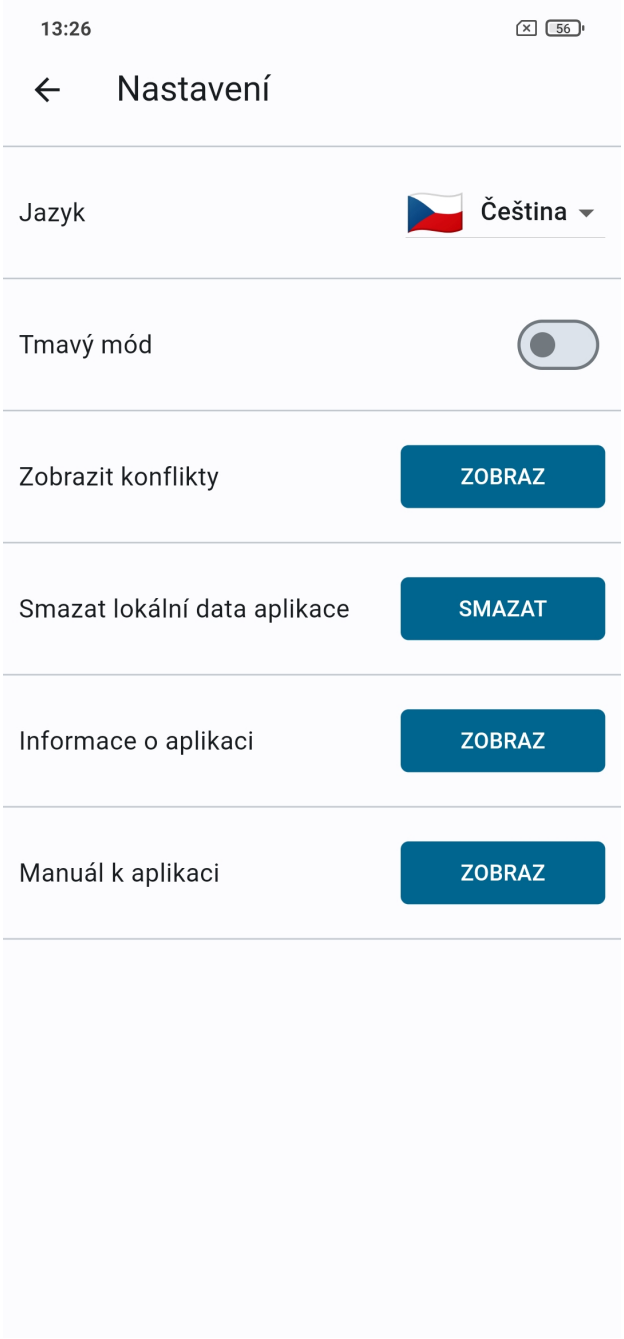
Na následujícím obrázku A.8 se nachází obrazovka pro obnovení hesla. Tato obrazovka obsahuje textové pole pro zadání uživatelova e-mailu a tlačítko, kterým se odešle e-mail s odkazem pro změnění hesla.



Obrázek A.8: Snímek obrazovky pro obnovu hesla

A.8 Nastavení

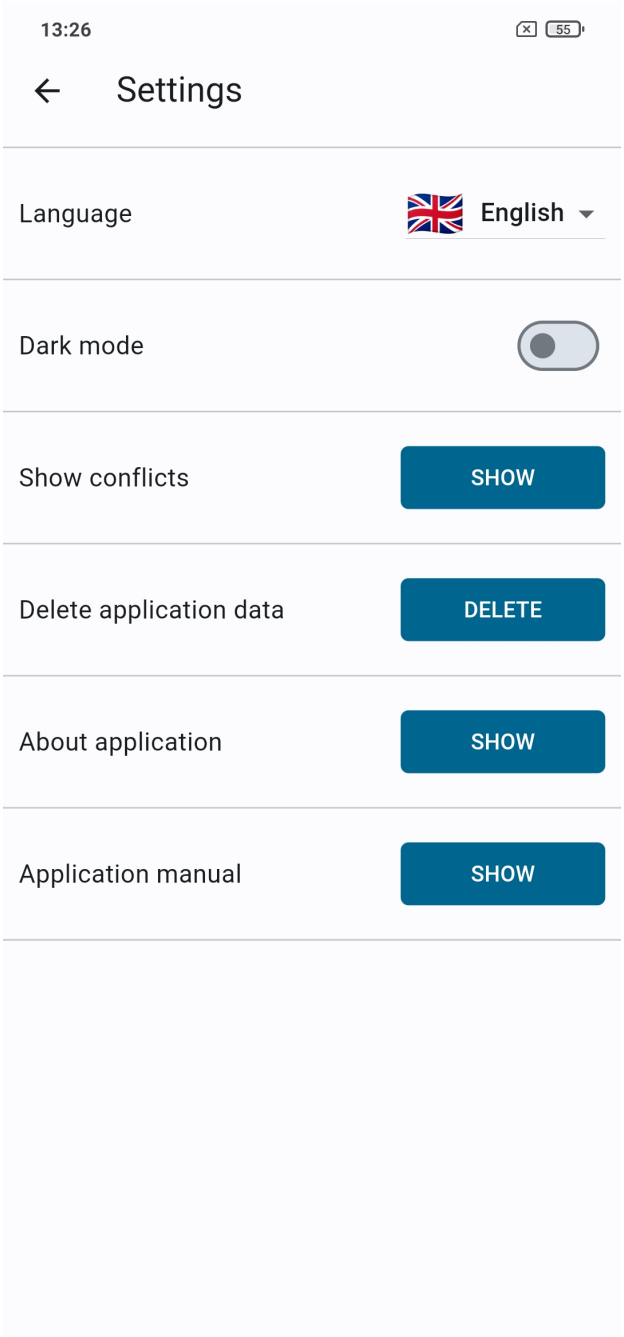
Na následujícím obrázku A.9 je zobrazena stránka nastavení. V dalších sekcích bude popsáno, jaké možnosti toto nastavení poskytuje.



Obrázek A.9: Snímek obrazovky s nastavením aplikace

A.8.1 Nastavení jazyka

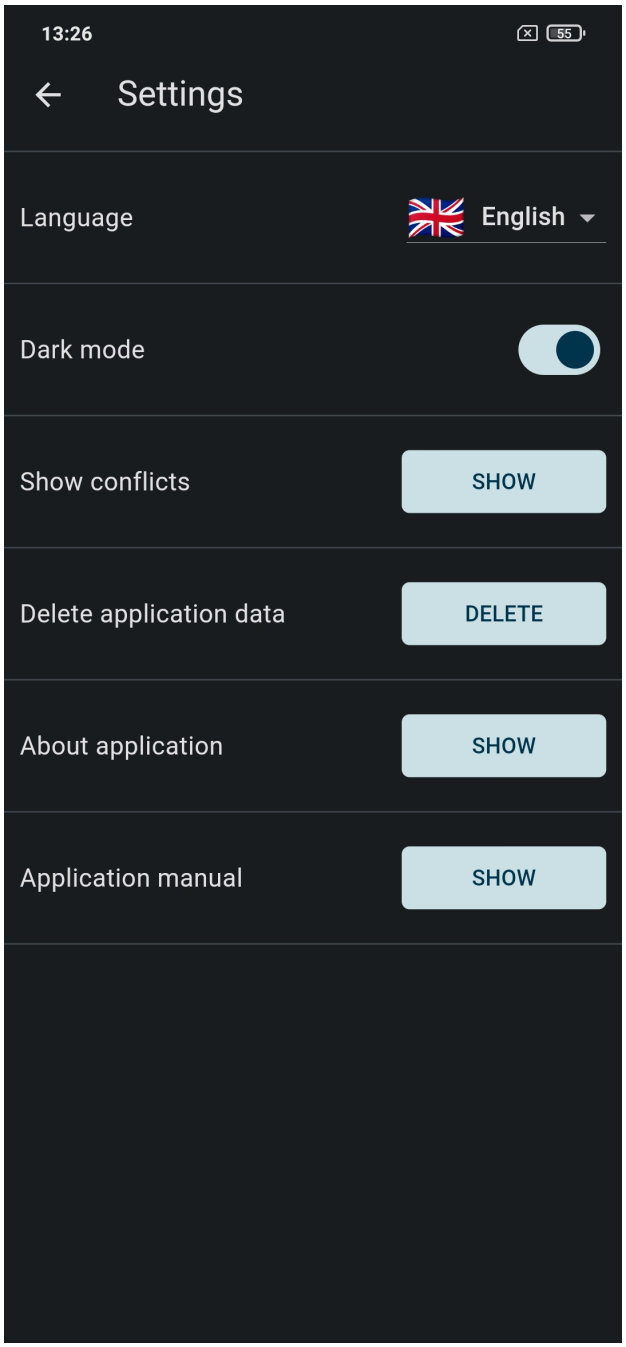
Jazyk lze nastavit v nastavení kliknutím na rozbalovací seznam s aktuálním jazykem. Následně je možné si vybrat ze dvou jazyků, a to z češtiny a angličtiny. Přepnutí aplikace do anglického jazyka je ukázáno na obrázku A.10.



Obrázek A.10: Snímek zobrazující anglickou lokalizaci aplikace

A.8.2 Tmavý mód

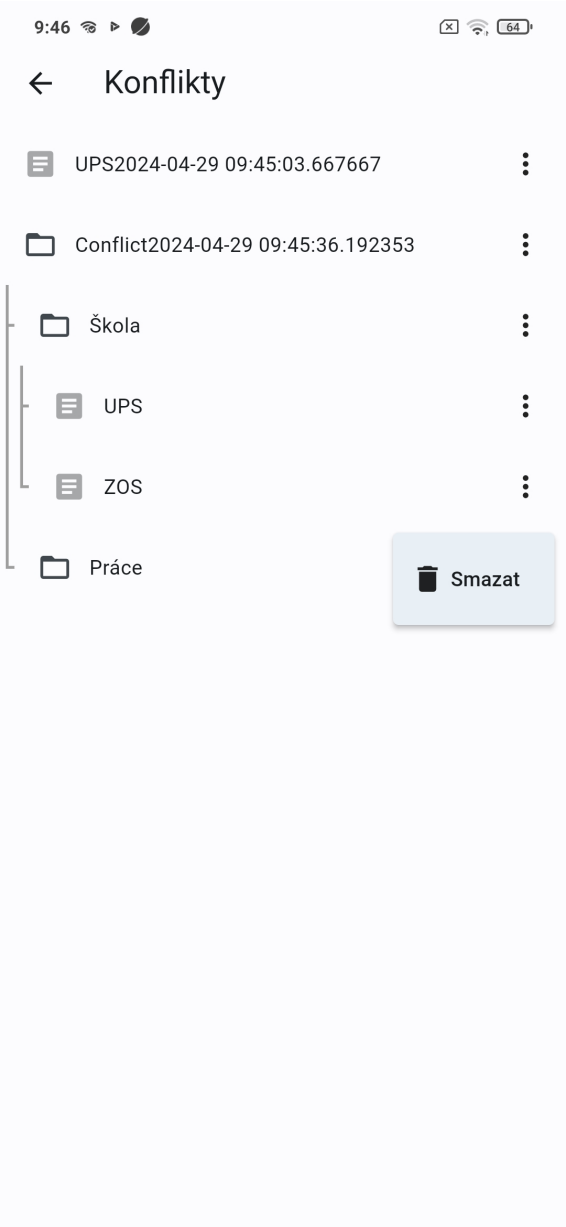
Tmavý mód lze v nastavení zapnout pomocí přepínače v řádce *Tmavý mód*. Vzhled tmavého módu je zobrazen na obrázku A.11.



Obrázek A.11: Snímek ukazující vzhled tmavého režimu

A.8.3 Zobrazení konfliktů

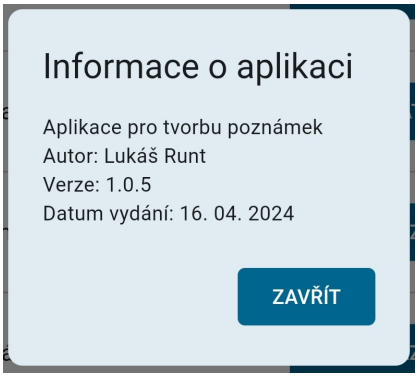
Konflikty lze zobrazit kliknutím na tlačítko *ZOBRAZ* v řádku *Zobrazit konflikty*. Na následujícím obrázku A.12 je zobrazena stránka s konfliktními daty. Konfliktní data jsou zobrazena ve stromovém zobrazení. Každý konflikt má v názvu přesný čas, kdy ke konfliktu došlo. Konfliktní data lze smazat kliknutím na tři tečky a následným stisknutím možnosti *Smazat*.



Obrázek A.12: Snímek zobrazující stránku s konfliktními daty

A.8.4 Informace o aplikaci

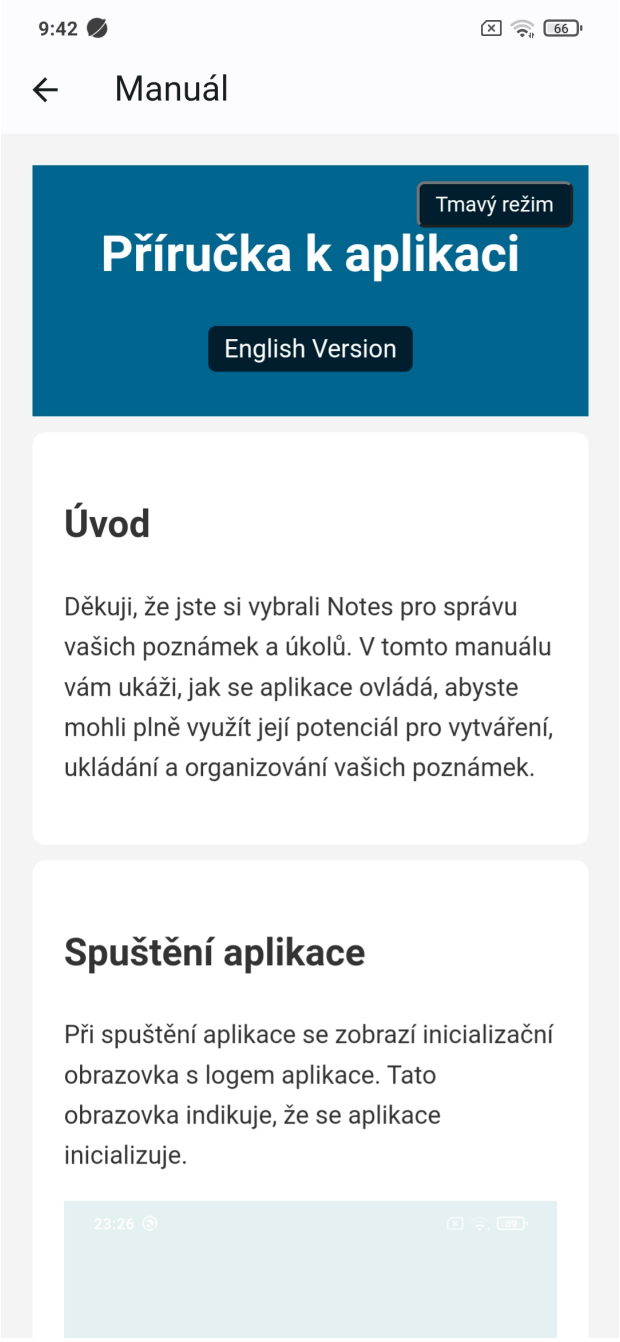
Po stisknutí tlačítka *ZOBRAZ* v řádku *Informace o aplikaci* se zobrazí dialogové okno s informacemi o vytvořené aplikaci (viz obrázek A.13), jako je autor, verze aplikace a datum vydání. Na následujícím obrázku je zobrazen snímek dialogového okna s informacemi o aplikaci.



Obrázek A.13: Snímek zobrazující stránku s konfliktními daty

A.8.5 Webová stránka manuálu

Při stisknutí tlačítka v řádku *Manuál k aplikaci* je uživatel přesměrován na webovou stránku s manuálem, která je zobrazena v aplikaci. Na obrázku A.14 je tento manuál zobrazen.

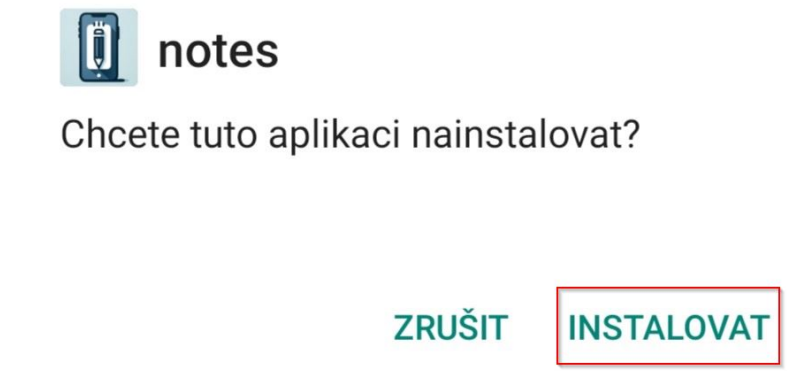


Obrázek A.14: Snímek zobrazující stránku s webovým zobrazením manuálu

Instalační příručka B

B.1 Android

Aplikace požaduje minimální verzi Androidu 6.0 Marshmallow (API level 23). Pro nainstalování aplikace na mobilní zařízení je nutné stáhnout nebo překopírovat APK soubor aplikace do svého mobilního zařízení. Dalším krokem je vyhledání a spuštění souboru. Je možné, že mobilní zařízení nebude mít povolené instalování aplikací z neznámých zdrojů. V tomto případě se zobrazí dialogové okno, ve kterém je nutno povolit instalaci z neznámých zdrojů. Po spuštění APK souboru a povolení instalace z neznámých zdrojů by se mělo zobrazit dialogové okno viz obrázek B.1, ve kterém stačí zmáčknout tlačítko instalovat a aplikace by se měla nainstalovat.

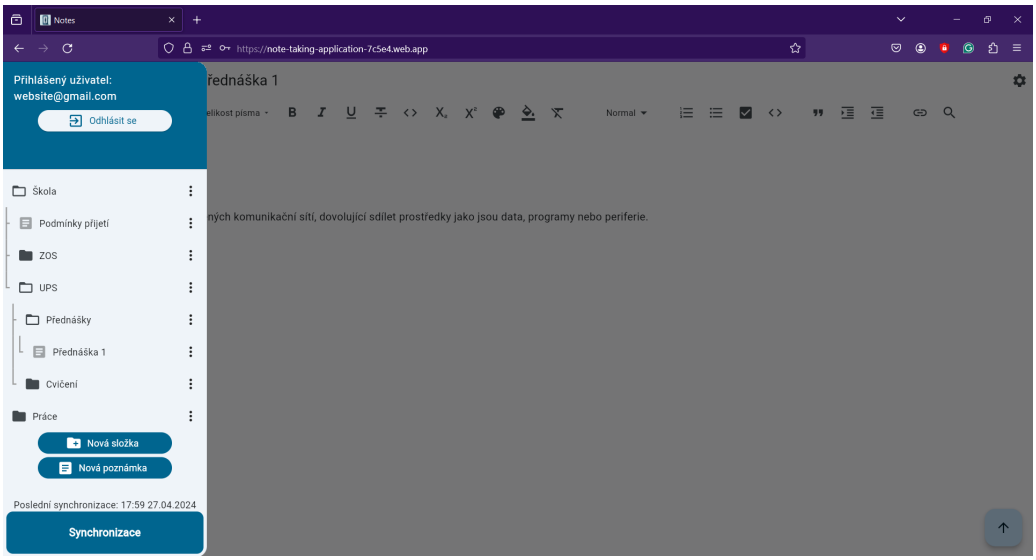


Obrázek B.1: Instalační dialog

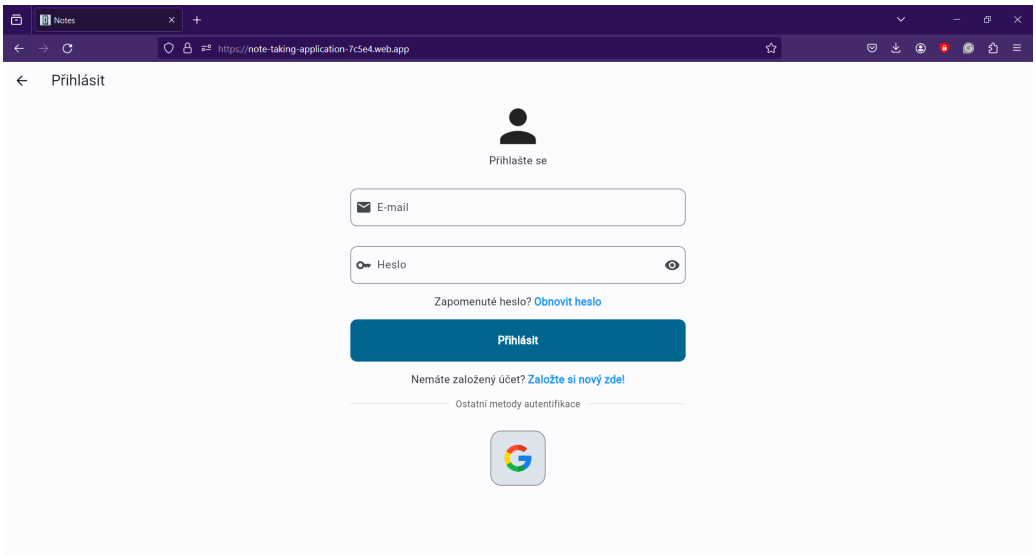
B.2 Web

Webová aplikace je dostupná na adrese: `https://note-taking-application-7c5e4.web.app`

Na následujících obrázcích B.2 a B.3 je zobrazen vzhled webové aplikace.



Obrázek B.2: Vysunuté menu ve webové verzi aplikace



Obrázek B.3: Přihlašovací stránka ve webové verzi aplikace

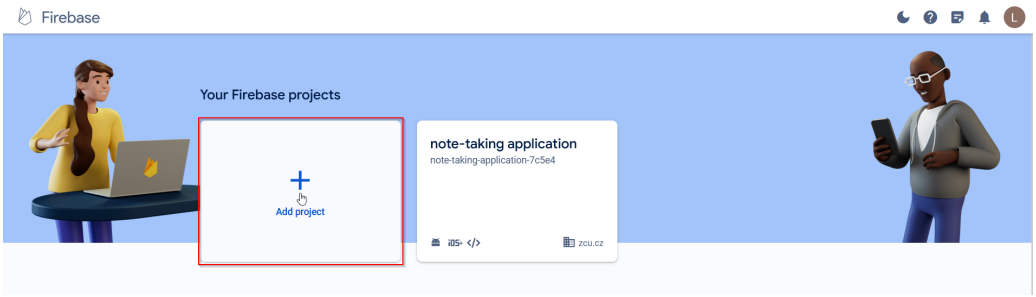
Konfigurace Firebase

C

V následující příloze se nachází příručka, jak založit, konfigurovat a propojit Firebase projekt s Flutter aplikací.

C.1 Vytvoření Firebase projektu

Pro vytvoření projektu se musíme na stránce Firebase[1] přihlásit účtem Google. Po přihlášení klikneme na tlačítko pro založení nového projektu viz obrázek C.1.



Obrázek C.1: Založení nového projektu

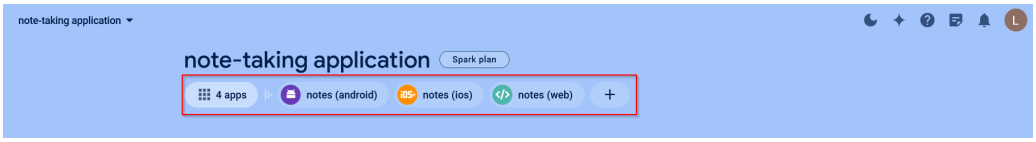
Vyplníme jméno projektu a stiskneme tlačítko *Next*. Na další stránce je nastavení Google Analytics, které můžeme zapnout nebo jej nechat vypnuté. Po vytvoření projektu se budeme nacházet na hlavní stránce projektu.

C.2 Propojení Firebase s Flutter projektem

Firebase projekt lze s Flutter projektem propojit různými způsoby. Následující návod se řídí instrukcemi z oficiální dokumentace[2]. Otevřeme hlavní stránku projektu, kde nalezneme logo Flutteru. Stiskneme logo Flutteru. Zobrazí se návod, dle kterého budeme dále postupovat:

[1] https://firebase.google.com/
[2] https://firebase.google.com/docs/flutter/setup?platform=android

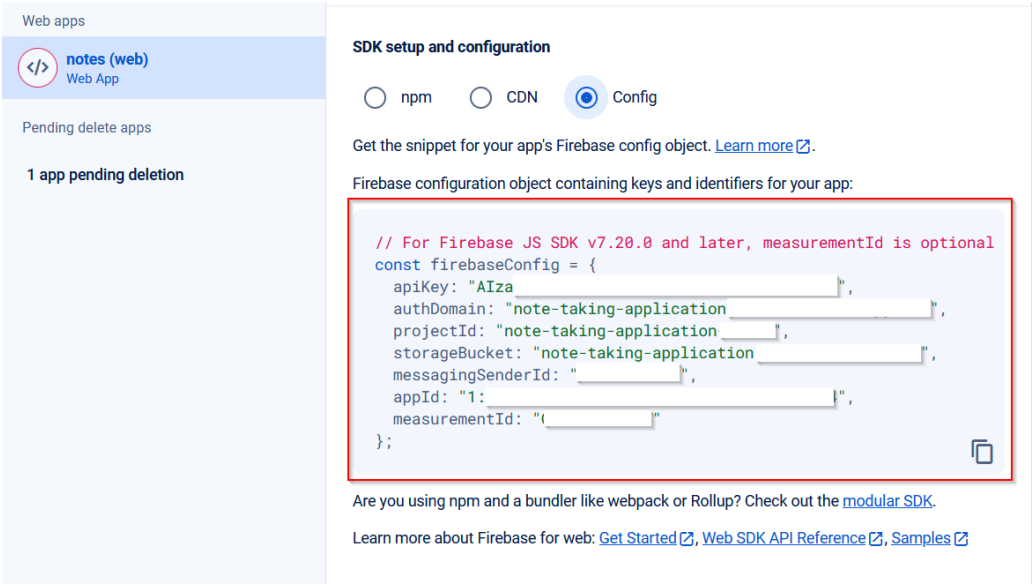
1. Nainstalujeme na své zařízení Firebase CLI. Návod na instalaci: `https://firebase.google.com/docs/cli#setup_update_cli`,
2. Otevřeme příkazovou řádku a zadáme příkaz: `firebase login`,
3. Přihlásíme se účtem Google, na kterém máme založený projekt, který chceme propojit s naší aplikací,
4. Zadáme příkaz: `dart pub global activate flutterfire_cli`,
5. Zadáme příkaz: `flutterfire configure`,
6. Objeví se všechny Firebase projekty, které máme na našem účtu založeny. Vybereme ten, který chceme propojit s naší aplikací,
7. Zobrazí se seznam platforem, na které můžeme Firebase konfigurovat. Při stisknutí klávesy enter se konfigurují všechny zobrazené platformy,
8. Spojení by se mělo automaticky konfigurovat. V adresáři `lib/` by se měl vytvořit soubor `firebase_options.dart`. Na hlavní stránce Firebase projektu by se mělo objevit propojení s platformou webu, Androidu, macOS a iOS viz obrázek C.2,
9. Pro dokončení spojení s webovou aplikací je nutné změnit ve zdrojovém kódu `main.dart` část definující spojení s webovou aplikací viz zdrojový kód C.1. Do této části kódu je nutno doplnit konfigurační údaje, které získáme v nastavení webového spojení viz obrázek C.3.



Obrázek C.2: Úspěšné propojení s aplikací

Zdrojový kód C.1: Konfigurace propojení s webovou aplikací

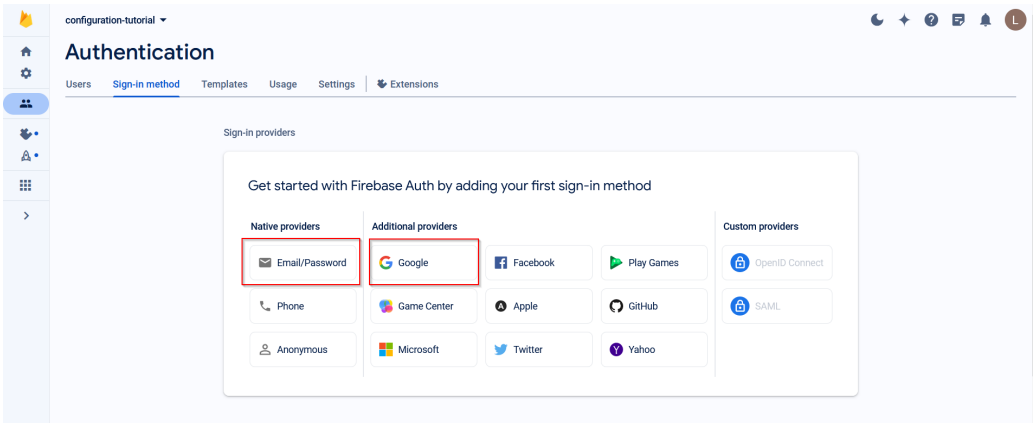
```
if (kIsWeb) {
      await Firebase.initializeApp(
        options: const FirebaseOptions(
          apiKey: "<vaše-apikey>",
          appId: "<vaše-appId>",
          messagingSenderId: "<vaše-messageSenderId>",
          projectId: "<vaše-projectId>",
        ),
      );
    }
```



Obrázek C.3: Konfigurační údaje webové aplikace

C.3 Firebase Authentication

Následující část se zabývá nastavením autentizačních služeb projektu Firebase. Vyvíjená aplikace využívá k autentizaci uživatele kombinací e-mailu a hesla a prostřednictvím účtu Google viz obrázek C.4. Obě metody budeme muset nastavit.



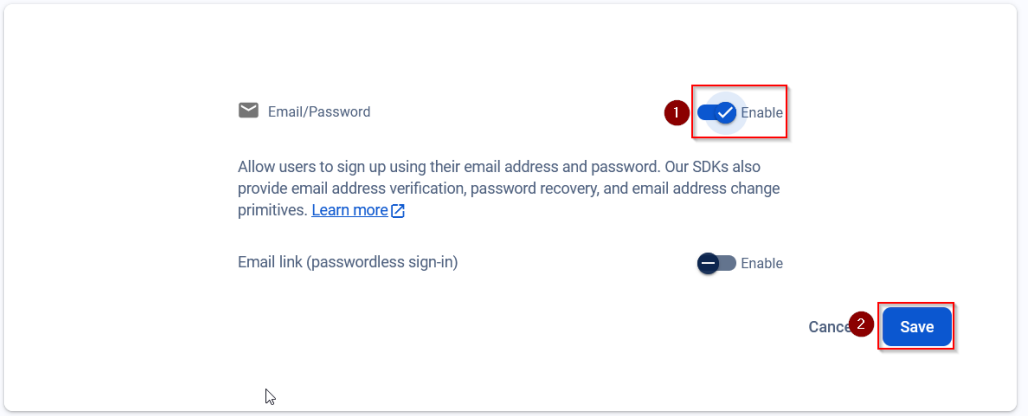
Obrázek C.4: Způsoby autentizace

C.3.1 Autentizace e-mailem a heslem

1. Klikneme na přidání nového způsobu ověření identity a zvolíme způsob e-mailu a hesla,

2. Povolíme tento způsob a klikneme na tlačítko *Save* viz obrázek C.5,

3. Způsob ověření kombinací e-mailem a heslem by se měl objevit mezi povolenými způsoby.

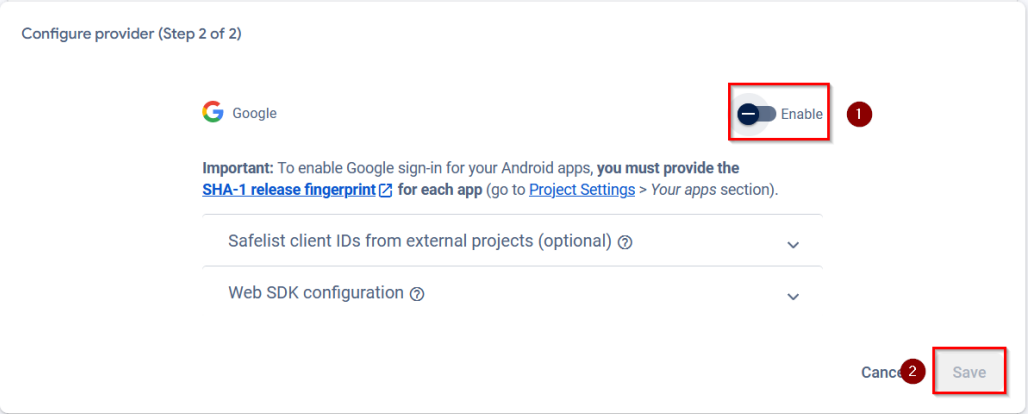


Obrázek C.5: Povolení autentizace pomocí e-mailu a hesla

C.3.2 Autentizace účtem Google

Následující sekce poskytuje postup, jak lze nastavit autentizaci pomocí účtu Google:

1. Ve vašem Firebase projektu se přesuňte do nastavení autentizace. Zde přidejte nový způsob autentizace účtem Google. Zobrazí se dialog, ve kterém je tuto metodu ověření nutno povolit viz obrázek C.6,



Obrázek C.6: Povolení ověření identity prostřednictvím účtu Google

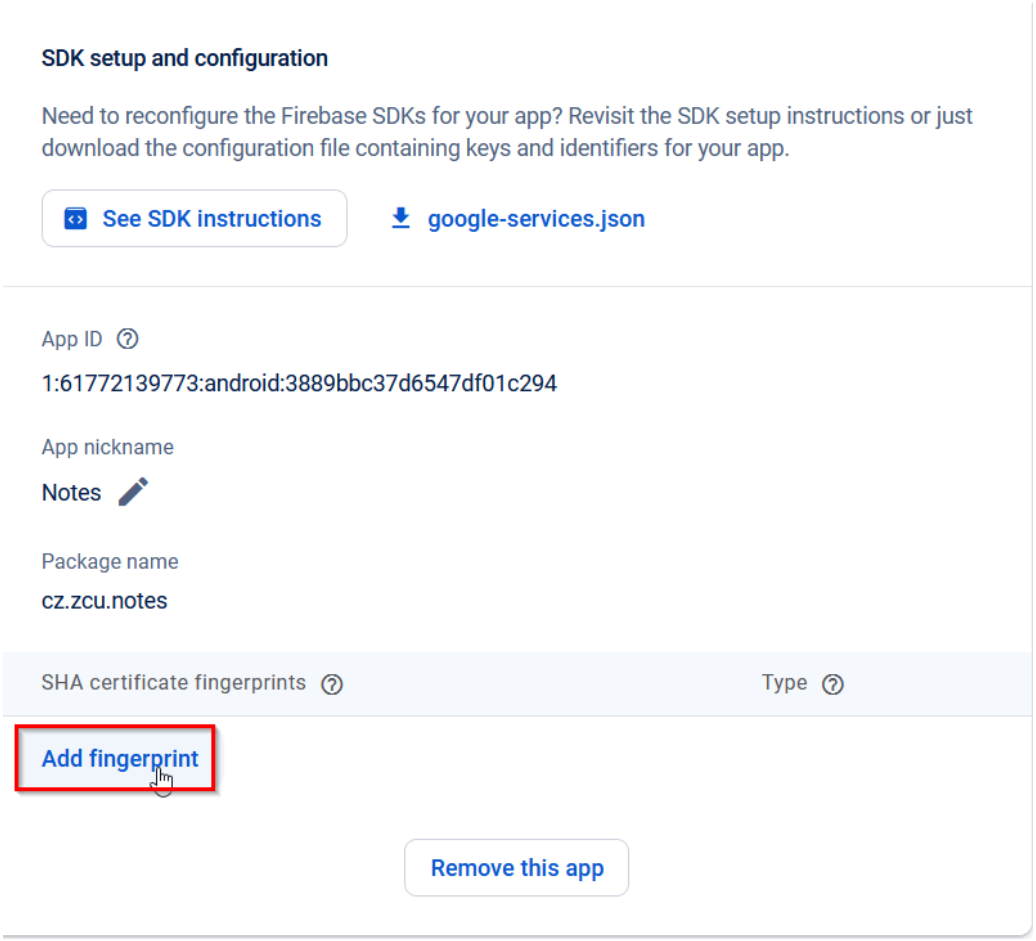
2. Otevřete nastavení propojení s Android aplikací a nastavte otisk certifikátů SHA-1:

a) Otevřeme konzoli a zadáme příkaz:
`keytool -list -v -alias androiddebugkey -keystore %USERPROFILE%/.android/debug.keystore`,

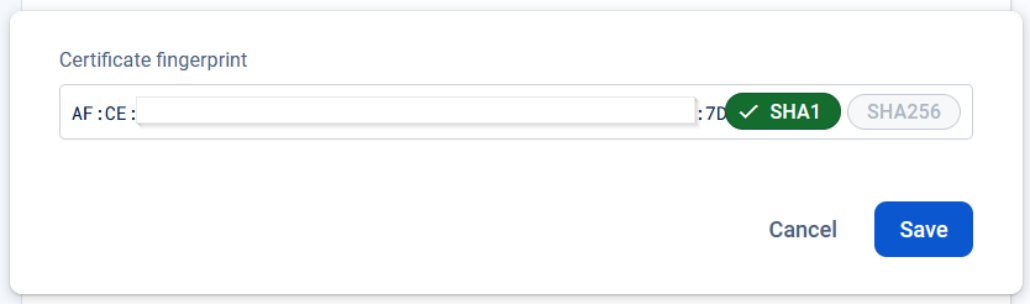
b) Jako heslo zvolíme: `android`,

c) Tento příkaz vygeneruje otisky SHA-1 a SHA-256, V nastavení propojení pak stiskneme tlačítko Add fingerprint viz obrázek C.7,

d) Zobrazí se dialogové okno, do kterého zkopírujeme vygenerované otisky SHA-1 a SHA-256 viz obrázek C.8.



Obrázek C.7: Přidání SHA certifikátů

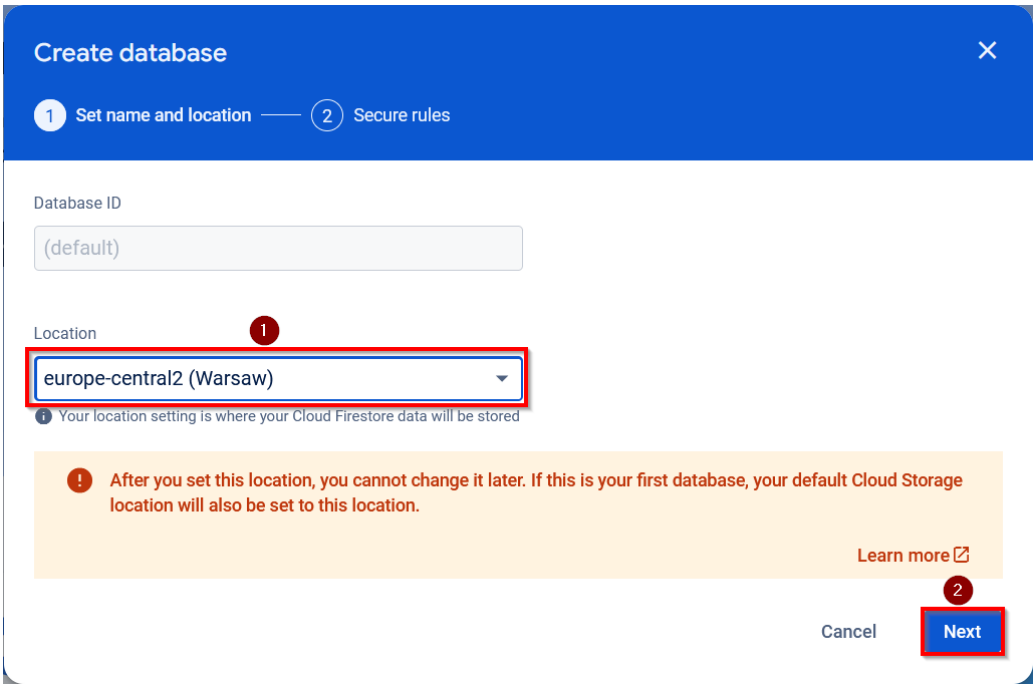


Obrázek C.8: Zadání otisku do dialogového okna

C.4 Firebase Firestore

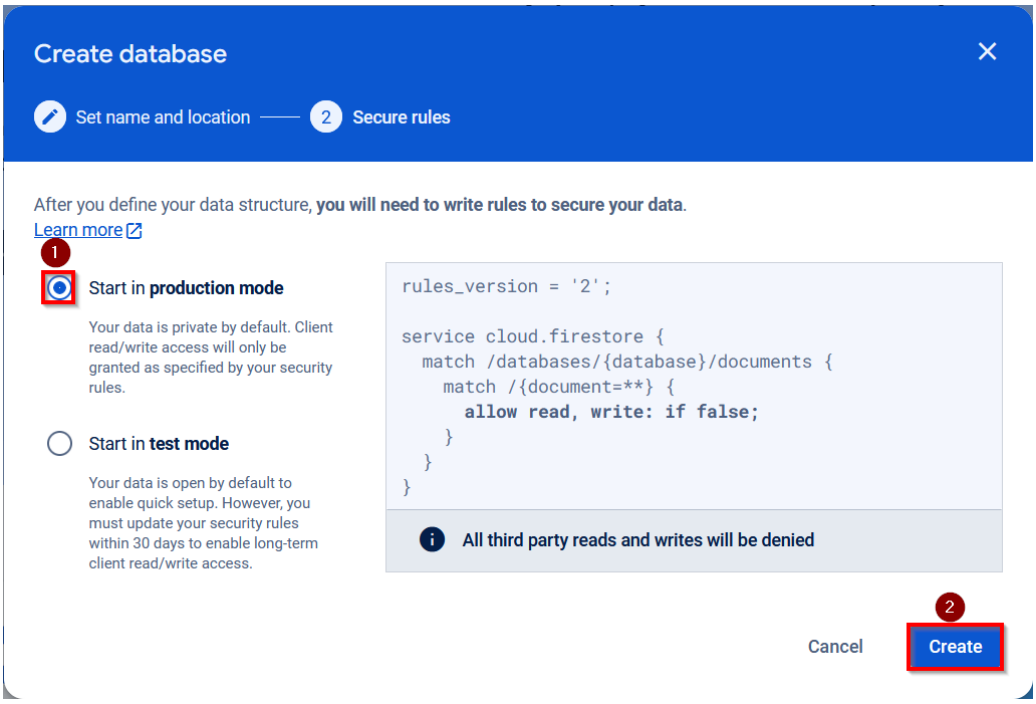
Tato sekce poskytuje návod, jak vytvořit a nakonfigurovat databázi Firestore k ukládání uživatelských dat:

1. Vytvořte databázi ve svém Firebase projektu. Objeví se dialogové okno, ve kterém zvolte server, na kterém budou data uložena (viz obrázek C.9) a stiskněte tlačítko *Next*. Na další stránce zvolte *Start in production mode* (viz obrázek C.10) a volbu potvrďte stiskem tlačítka *Create*,



Obrázek C.9: Založení Firestore databáze - volba serveru

2. Vytvořená databáze není připravena pro čtení a zápis dat. Je potřeba upravit pravidla (*Rules*) a povolit čtení a zápis dat viz zdrojový kód C.2.



Obrázek C.10: Založení Firestore databáze - založení v produkčním módu

Zdrojový kód C.2: Nastavení pravidel databáze Firestore

```
1 rules_version = '2';
2
3 service cloud.firestore {
4   match /databases/{database}/documents {
5     match /{document=**} {
6       allow read, write: if request.auth != null;
7     }
8   }
9 }
```

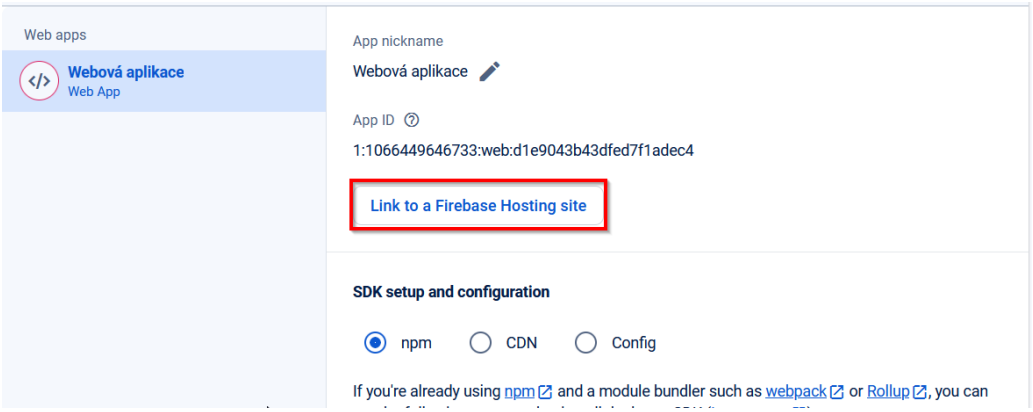
C.5 Firebase Hosting

Následující část příručky obsahuje návod, jak vytvořit hosting pro webové aplikace, které byly vytvořeny technologií Flutter. Předpokladem tohoto návodu je vytvořený a spárovaný Firebase projekt a nainstalované Firebase CLI:

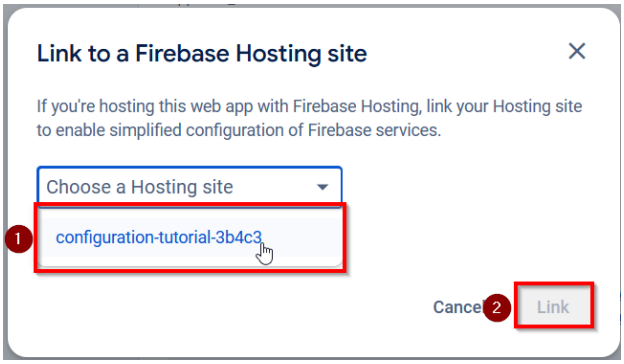
1. Otevřete příkazovou řádku v kořenovém adresáři Flutter projektu a vytvořte build webové aplikace příkazem: `flutter build web`.
   Tento build se uloží do složky `build/web`,

2. Dále je potřeba nastavit hostování pro váš projekt na webových stránkách Firebase. Ujistěte se, že vaše webová aplikace je propojena s projektem Firebase.

Pokud není, konfigurujte spojení dle návodu C.2. Pokud toto propojení existuje, přejděte do nastavení webové aplikace a zvolte hosting, na který bude web odkazovat. Tento postup je zobrazen na obrázcích C.11 a C.12,



Obrázek C.11: Tlačítko nastavení hostingu v nastavení webové aplikace



Obrázek C.12: Volba hostingu, na který bude web odkazovat

3. Přesuňte se zpět do příkazové řádky a přesvědčte se, že jste přihlášeni k Firebase příkazem `firebase login`. Případně se přihlaste účtem Google, který používáte,
4. Pomocí příkazu: `firebase init hosting` proveďte inicializaci hostování,
5. Předchozí příkaz vytvoří ve Flutter projektu soubor `firebase.json` viz zdrojový kód C.3. V tomto souboru nastavte hodnotu klíče "public"na cestu k buildu webu, který jsme vytvořili v prvním kroku,

Zdrojový kód C.3: Konfigurace propojení s webovou aplikací

```
{
  "hosting": {
    "public": "build/web",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

6. Proveďte nasazení aplikace na Firebase Hosting příkazem: `firebase deploy`. Po dokončení nasazení se automaticky vygeneruje URL adresa, na které bude vaše webová aplikace dostupná.

Obsah ZIP souboru


```
A20B0226P_prilohy
├── Text_prace
│   ├── LaTeX_zdrojove_soubory ..... Zdrojové soubory PDF
│   └── A20B0226P_BP.pdf ..... PDF verze bakalářské práce
├── Aplikace_a_knihovny
│   ├── Notes_zdrojove_soubory ..... Zdrojové soubory aplikace
│   ├── Notes.apk ..... Instalační soubor aplikace
│   ├── Dartdoc ..... Dokumentace zdrojového kódu
│   ├── Navod_na_sestaveni.txt ..... Návod na sestavení aplikace
│   └── Manual_zdrojove_soubory ..... Zdrojové soubory webové stránky
│       manuálu
└── Readme.txt ..... Detailní informace o obsahu adresáře
```