FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Zápisník s tvorbou vlastních šablon
Student:	Elena Patceva
Vedoucí:	Ing. Jan Blizničenko
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je vytvořit aplikaci – zápisník se šablonami pro různá témata, jako jsou recepty, zápisy pro cestovatele apod. Šablony budou některé předpřipravené s možností pro uživatele vytvořit šablonu novou – vlastní.

Aplikace musí být přístupná ze zařízení s aktuálními operačními systémemy Windows a Linux a to buď formou multiplatformní desktopové aplikace nebo aplikace webové. Nedílnou součástí práce je i výběr vhodných technologií a využití již existujících frameworků v co největší možné míře.

- Proveďte rešerši podobných nástrojů.
- Proveďte rešerši relevantních technologií, nástrojů, frameworků a knihoven.
- Vytvořte návrh aplikace a jejích součástí.
- Proveďte implementaci aplikace.
- Otestujte a zdokumentujte své řešení.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 25. ledna 2021 v Praze.

Bakalářská práce

ZÁPISNÍK S TVORBOU VLASTNÍCH ŠABLON

Elena Patceva

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Blizničenko
6. ledna 2022

Obsah

Seznam obrázků

Seznam výpisů kódu

Chtěla bych poděkovat svému vedoucímu Ing. Janu Blizničenko za jeho pomoc při vytváření bakalářské práce a taky svým rodičům a kamarádům za podporu během celého studia.

Prohlášení

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

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

V Praze dne 6. ledna 2022 .....

Abstrakt

Tato bakalářská práce se zabývá vývojem webové aplikace pro správu poznámek různého typu. Poznámkové bloky se vytváří podle šablon, a to buď veřejně předpřipravených nebo soukromě vytvořených. Začátek práce je věnován průzkumu existujících řešení a specifikaci funkčních a nefunkčních požadavků. Hlavní náplní této práce je návrh architektury aplikace a její implementace pomocí vhodných technologií. Práce je napsána s využitím frameworku React a vývojové platformy Firebase. Výsledkem je funkční webová aplikace, která umožňuje vytvářet šablony a poznámkové bloky.

Klíčová slova multiplatformní webová aplikace, zápisník, poznámkový blok, React, JavaScript

Abstract

This bachelor thesis focuses on the development of a web application for managing notes of different types. Notepads are created according to templates, either publicly prepared or privately created. At the beginning, existing software is evaluated and requirements are analysed. The main subject of this thesis is the design of the application architecture and its implementation with the help of appropriate technologies. Application is written using the React framework and the Firebase development platform. The output is a functioning web application that enables the user to create templates and notepads.

Keywords multiplatform web application, notepad, notes, React, JavaScript

Seznam zkratek

SPA	Single-Page Application
MPA	Multi-Page Application
HTML	HyperText Markup Language
AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
JSON	JavaScript Object Notation
DOM	Document Object Model
CRUD	Create, Read, Update, Delete
HTTPS	HyperText Transfer Protocol Secure
CDN	Content Delivery Network
ID	Identifier
UID	Unique Identifier

Úvod

Dávat přednost technologii před papírovými diáři se stalo součástí našeho života. Všechny koníčky a činnosti, které jsme dřív zapisovali do poznámkového bloku nebo zápisníku, se přesunuly do aplikací v mobilu nebo dokumentů v počítači. Snadno by se dala najít celá řada aplikací určených na evidování receptů, plánování dne nebo zkrátka pro jakoukoliv jinou potřebu. Hlavní nevýhodou tohoto přístupu je nutnost použít vždy samostatnou technologii. Proto jsem si jako téma bakalářské práce zvolila vytvoření aplikace, která uživateli umožňuje zaznamenat vše od receptů po filmy na jednom místě. Ve skutečnosti je to univerzální zápisník pro všechny příležitosti. K ulehčení použití bude v aplikaci předpřipraveno několik šablon jako například „Můj den“, „Kniha receptů“ atd. s tím, že si uživatel bude mít možnost vytvořit i šablony vlastní.

V mé bakalářské práci udělám rešerši podobných aplikací, navrhnu architekturu aplikace, implementuji a otestuji ji.

Cíle práce

Cílem bakalářské práce je vytvoření aplikace – zápisník s předpřipravenými šablonami pro různá témata jako jsou recepty, zápisy pro cestovatele apod. Uživatel si bude mít možnost vytvořit také šablony vlastní.

Cílem rešeršní části práce je získání přehledu o již existujících podobných aplikacích a prozkoumání relevantních technologií, nástrojů, frameworků a knihoven.

Implementačním cílem práce je vytváření samotné aplikací, která bude přístupná ze zařízení s operačními systémy Windows a Linux.

Další cíl bakalářské práce je návrh a otestování aplikace.

Členění práce

V kapitole 1 provedu rešerši již existujících aplikací a popíši klady a zápory jednotlivých systémů.

V kapitole 2 specifikuji funkční a nefunkční požadavky.

V kapitole 3 na základě požadavků zvolím vhodné technologie, formu aplikace a proberu návrh řešení.

V kapitole 4 se budu zabývat implementací.

V 5. kapitole se budu věnovat testování vytvořené aplikace.

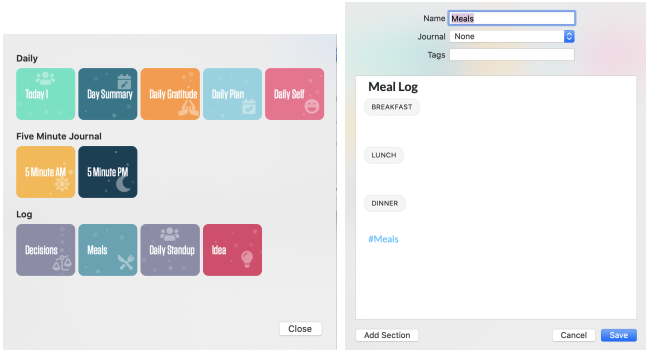
Kapitola 1

Zhodnocení existujících řešení

Nedílnou součástí bakalářské práce je rešerše již existujících řešení zadaného problému. V této kapitole se zaměřím na aplikace, které jsou té mojí nejvíce podobné. Pod pojmem „podobná aplikace“ se rozumí aplikace, která nějakým způsobem pracuje se šablonami pro různá témata, proto se nebudu zabývat systémy určenými pouze pro správu poznámek jednoho typu, například aplikace určené pouze na recepty nebo jenom pro sport. Vytvoření mobilní aplikace není cílem této bakalářské práce, proto budu rešerši provádět jen mezi aplikacemi dostupnými z počítače. V případě, že má aplikace placenou i neplacenou verzi, zaměřím se více na tu neplacenou.

1.1 Day One

Velmi podobná aplikace, kterou jsem našla, se jmenuje „Day One“ [3]. Stejně jako aplikace, kterou plánuji vytvořit, má několik předpřipravených šablon. V neplacené verzi lze vytvořit jenom jeden zápisník, který se skládá ze záznamů. Každý záznam může a nemusí obsahovat šablonu. Řazení dle šablony se provádí díky automaticky přidaným tagům.



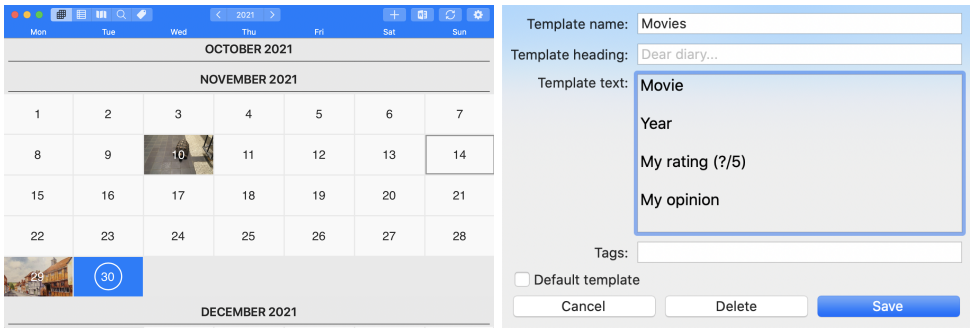
Obrázek 1.1 Aplikace Day One [3]

Nevýhody:

- Lze vytvořit pouze jeden zápisník.
- V jednom zápisníku se vyskytují záznamy s různými šablonami.
- Aplikace existuje pouze pro MacOS.

1.2 Diarium

Domovská stránka aplikace Diarium[4] je kalendář, do kterého uživatel pravidelně přidává své záznamy. Má také možnost vytvořit své vlastní šablony. Nevýhodou je, že se záznamy zobrazují pouze v pořadí přidání a nedají se tak řadit podle šablony.



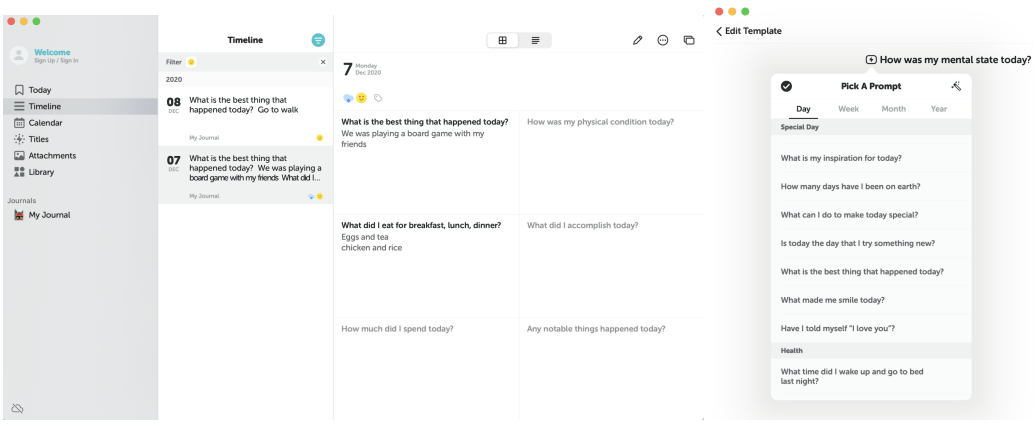
Obrázek 1.2 Aplikace Diarium [4]

Nevýhody:

- Není možnost řadit podle šablony.
- Neexistuje verze pro Linux.
- Bez placené verze nelze záznamy synchronizovat mezi různými zařízeními.

1.3 Grid Diary

Grid Diary [5] je zajímavá aplikace, ve které může uživatel každý den, týden, měsíc nebo rok (dle nastavení) zodpovědět několik otázek týkajících se různých částí jeho života. Seznam otázek zůstává stále stejný a díky tomu může uživatel sledovat změny ve svém životě. Při vytváření záznamu lze přidat i vlastní otázku, ta se ovšem do předpřipravené nabídky neuloží, proto se aplikace nehodí na správu seznamů jako jsou recepty nebo filmy. Navíc se předpokládá, že uživatel poznámky pravidelně přidává, ovšem například v případě knih nebo seriálů nechceme přidávat recenze každý den nebo měsíc.



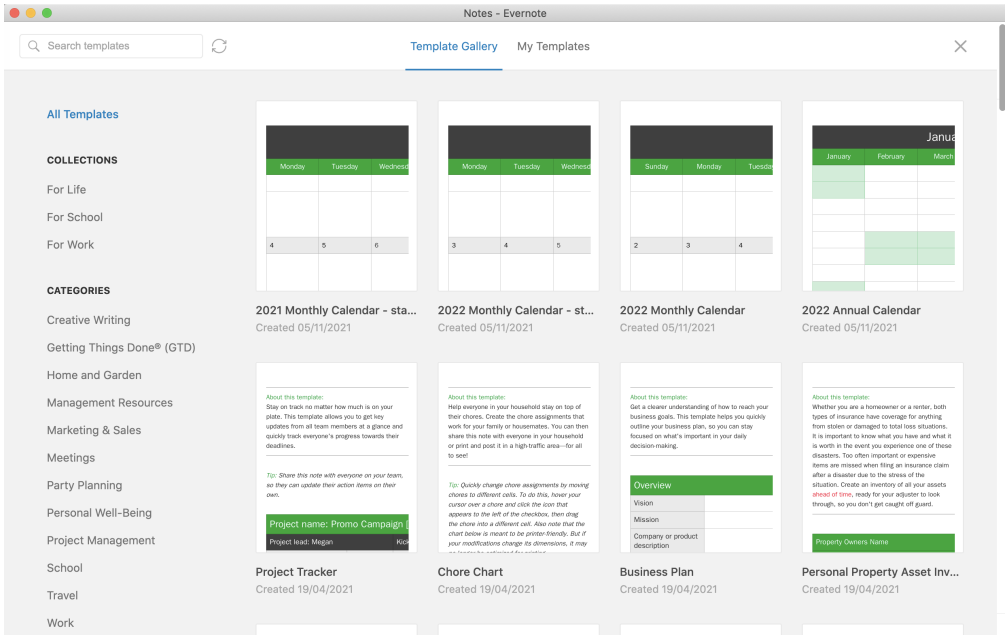
Obrázek 1.3 Aplikace Grid Diary [5]

Nevýhody:

- Nehodí se pro jednorázové záznamy.
- Lze vytvořit pouze jeden zápisník.
- Existuje verze pouze pro MacOS.

1.4 Evernote

Evernote [6] existuje v podobě webové, mobilní a desktopové aplikace. Stejně jako v Day One [1.1] každý záznam v zápisníku může, ale nemusí obsahovat šablonu. Zápisník se nemusí skládat pouze ze záznamů stejného typu. V takovém případě se řazení poznámek podle šablony provádí pomocí tagů, které si uživatel musí nastavovat ručně.



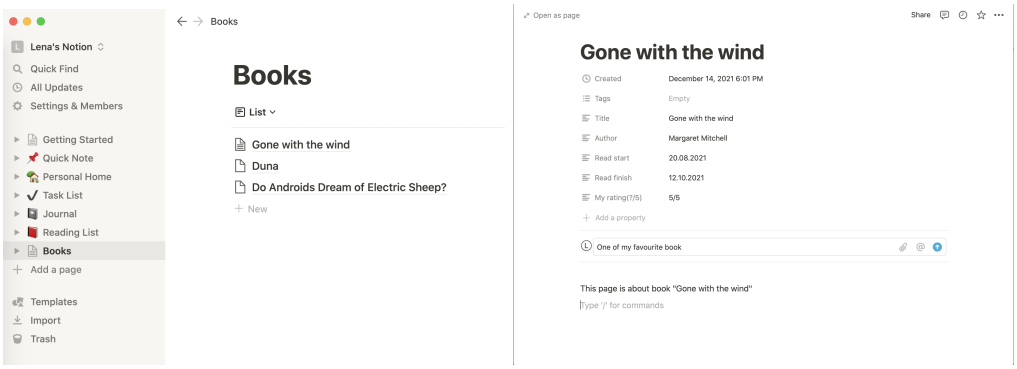
Obrázek 1.4 Aplikace Evernote [6]

Nevýhody:

- Vlastní šablony lze vytvářet pouze v placené verzi aplikace.
- Není možné automaticky řadit podle šablony.

1.5 Notion

Notion [7] je dostupné jako mobilní, webová a desktopová aplikace. Jedná se o komplexnější systém s rozsáhlou funkcionalitou. Mimo vytváření šablon a uchovávání poznámek všemožných typů umí tato aplikace například také importovat poznámky z jiných aplikací a formátů (jako Evernote [1.4], Word, HTML atd.), odesílat notifikace, sdílet poznámky, zobrazovat historii změn a mnoho dalšího. Pro správu jednoduchých poznámek má ovšem až moc velký záběr.



Obrázek 1.5 Aplikace Notion [7]

Nevýhody:

- Není vhodné pro správu menších poznámek.

Kapitola 2
Specifikace požadavků

V této kapitole specifikuji funkční a nefunkční požadavky své aplikace, a to na základě stanovených cílů bakalářské práce a také na základě rešerše již existujících aplikací z kapitoly 1.

2.1 Funkční požadavky

- Vytváření zápisníků podle šablony [F1]
- Správa záznamů v zápisníku [F2]
- Vytváření vlastních šablon [F3]
- Registrace nového uživatele [F4]
- Přihlášení uživatele [F5]
- Sdílení šablon mezi uživateli [F6]

F1: Vytváření zápisníků podle šablony

Uživatel bude mít možnost vytvářet zápisníky podle existujících šablon. Každý zápisník se bude skládat ze stránek, jejichž obsah bude uživatel schopen měnit (viz požadavek [F2]).

F2: Správa záznamů v zápisníku

Uživateli bude umožněno přidávat, editovat a mazat záznamy v zápisníku. Změny se budou průběžně ukládat.

F3: Vytváření vlastní šablony

V případě, že uživateli nevyhovuje žádná z předpřipravených nebo již vytvořených šablon, bude mít možnost vytvořit si vlastní šablonu na základě svých preferencí.

F4: Registrace nového uživatele

Na začátku si každý uživatel vytvoří vlastní účet, který bude uchovávat všechny jeho uložené zápisníky.

F5: Přihlášení uživatele

Po registraci se uživatel bude schopen přihlásit do svého učtu a prohlížet své záznamy z různých zařízení.

F6: Sdílení šablon mezi uživateli

Při vytváření vlastní šablony bude mít uživatel dvě možnosti: nechat šablonu soukromou, anebo ji zveřejnit také ostatním uživatelům. V prvním případě šablonu uvidí pouze její autor, v druhém případě se šablona objeví v seznamu veřejných šablon, odkud si ji bude moci stáhnout jakýkoliv uživatel.

2.2 Nefunkční požadavky

- Systém bude dostupný jako webová nebo desktopová aplikace pro operační systém Windows a Linux. [F1]
- Aplikace bude responzivní, tak aby umožňovala snadné prohlížení na mobilních zařízeních a tabletech. [F2]
- Citlivé údaje (jako jsou hesla) se budou ukládat šifrovaně. [F3]
- Aplikace bude během stahování dat ze serveru ukazovat ikonu načítání. [F4]

Kapitola 3
Návrh

V této kapitole se na základě požadavků z předchozí kapitoly budu zabývat návrhem samotného řešení. Na začátku zvolím vhodné moderní technologie, které využiji ve své implementaci, a poté popíši rozdělení aplikace na jednotlivé komponenty.

3.1 Forma aplikace

Jedním z hlavních požadavků je dostupnost aplikace z různých operačních systémů. Pro docílení tohoto požadavku existují dva přístupy: desktopová multiplatformní aplikace nebo webová aplikace. Dále proberu výhody a nevýhody obou těchto způsobů.

3.1.1 Desktopová multiplatformní aplikace

Desktopová aplikace se instaluje přímo na počítač uživatele, což znamená, že má své systémové požadavky. [8] To omezuje počet potenciálních uživatelů. Různé operační systémy mohou působit určité komplikace a přidávat práci navíc. Například i v programovacím jazyku Java, ve kterém není nutností vytvářet zvláštní aplikaci pro každý operační systém, je potřeba mít různé buildy. Také si každou aktualizaci uživatel musí instalovat ručně. [9] Na druhou stranu se v desktopové aplikaci dá větší část funkcionality udělat lokálně, díky tomu může mít uživatel přístup ke svým záznamům i bez připojení k internetu. [10] Navíc komunikace probíhající přes internet za použití webových prohlížečů s sebou může nést bezpečnostní rizika. [8]

Výhody:

- Dostupnost bez síťového připojení.
- Nižší bezpečnostní rizika.

Nevýhody:

- Různé operační systémy komplikují vývoj aplikace.
- Vetší systémové požadavky.

3.1.2 Webová aplikace

Webová aplikace se otevírá v prohlížeči, což má jak pozitivní, tak i negativní důsledky. Mezi pozitivní patří to, že aplikace je přístupná z jakéhokoliv operačního systému (jako bonus je dostupná i z mobilního zařízení). [8] Není tudíž potřeba vytvářet zvláštní buildy pro různé operační systémy. Díky tomu je aktualizace aplikace lehčí jak pro vývojáře, tak i pro uživatele. Vývojáři nemají práci navíc a uživatelé nemusí ručně instalovat nové verze. [9] Systémové požadavky jsou omezené pouze na prohlížeč. Nevýhodou je, že se není možné k aplikaci dostat bez síťového připojení. [10]

Výhody:

- Stejná aplikace pro každý operační systém (včetně mobilních zařízení).
- Uživatel nemusí provádět aktualizace.
- Systémové požadavky pouze pro prohlížeč.

Nevýhody:

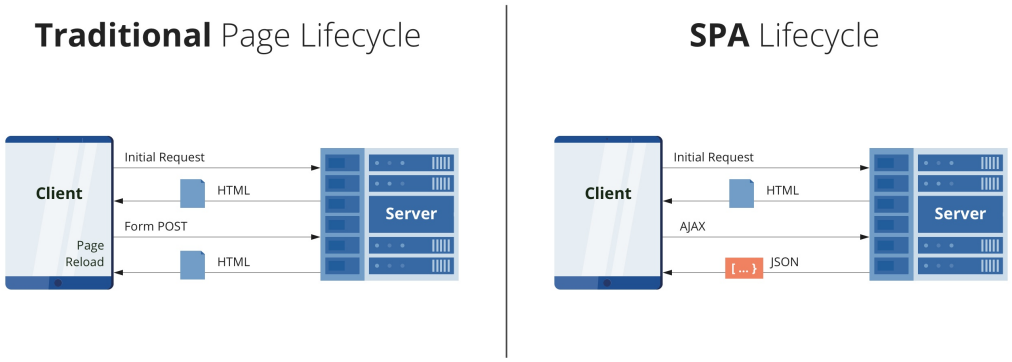
- Aplikace není dostupná bez připojení k internetu.

Zvolená forma aplikace

Na základě výše uvedených kladů a záporů jsem si zvolila variantu webové aplikace.

3.2 Typ webové aplikace

Při vytváření webové aplikace existují dva přístupy postupu. Tradiční *Multi-Page* aplikace (MPA) a více moderní způsob – *Single-Page* aplikace (SPA). Jejich rozdíl zjednodušeně znázorňuje obrázek 3.1.



Obrázek 3.1 Rozdíl mezi MPA a SPA [11]

MPA posílá požadavek na server pokaždé, když uživatel provede změnu. Server poté zpracovává požadavek a vrací HTML kód aplikace. Hlavním charakteristickým rysem MPA je nutnost aktualizovat celou webovou stránku při každé malé změně dat.

Název SPA odkazuje na to, že se aplikace formálně skládá pouze z jedné stránky, do které se dynamicky přidávají prvky. Díky tomu není nutné pokaždé aktualizovat celou stránku. Řeší se to tak, že po inicializaci již server není zodpovědný za správu HTML prvků. Požadavky na server se obvykle odesílají pomocí technologii AJAX a server vrací pouze nutné změny a ne celou stránku, a to ve formátu JSON.

Typickou architekturou aplikace je její rozdělení do tří vrstev:

- prezentační,
- aplikační,
- datová.

Prezentační vrstva je frontendovou částí aplikace. Je zodpovědná pouze za vzhled aplikace. Neobsahuje žádnou logiku zpracování dat a nemá ani přístup k databázi.

V aplikační vrstvě je realizována celá logika aplikace, všemožné výpočty a zpracování dat přístupných z databáze. Jedná se o backendovou část aplikace.

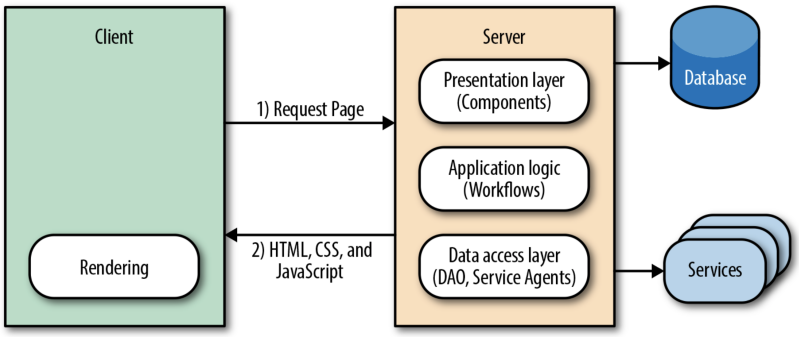
Databáze se nachází na nejnižší vrstvě aplikace – datové. Zajišťuje práce s daty, tedy jejich ukládaní, přístup k nim atd.

MPA a SPA mají různé přístupy k nastíněnému rozdělení architektury, tyto rozdíly popíši v následující sekci.

3.2.1 Multi-Page aplikace

MPA je založena na technologii renderingu ze strany serveru (*server-side*). Zde je potřeba upřesnit, že se v kontextu formy aplikace pod pojmem rendering rozumí sestavení HTML značek, a ne jejich parsování v prohlížeči.

Na obrázku 3.2 je zobrazen princip fungování MPA. Při renderingu ze strany serveru se prezentační a aplikační vrstva nachází na serveru. Server má přístup k databázi, zpracovává data a vrací straně klienta hotovou stránku složenou z HTML prvků, CSS stylů a JavaScript zdrojových kódů, které určují chování komponent. Prohlížeči tj. klientovi zbývá pouze vykreslit z toho webovou stránku.



Obrázek 3.2 Princip fungování MPA [12]

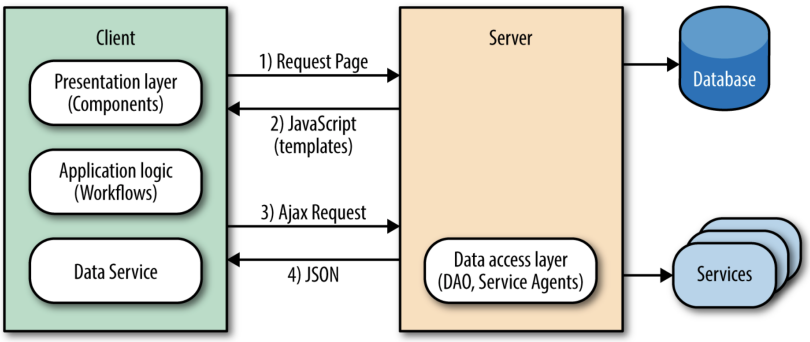
Dle [11] se MPA více hodí při vytváření větších a komplexnějších aplikací, které spravují velké množství dat. Výhodou je snadná rozšiřitelnost aplikace. Díky tomu, že se aplikace dělí na stránky, lze přidat další stránku s novou funkcionalitou, aniž by bylo nutné zásadně měnit

architekturu celé aplikace. Na druhou stranu, jelikož má MPA obvykle větší funkcionalitu, je její celkový vývoj a podpora složitější a potřebuje více zdrojů a času.

Jak jsem již uvedla dříve, MPA neukládá kód na straně uživatele, a proto musí při každé změně čekat na odpověď serveru, který posílá HTML kód. Poté MPA zaktualizuje obsah celé stránky. Trvalé odesílaní požadavků zatěžuje server, který musí nejprve úkol zpracovat a poté odeslat odpověď. Z toho vyplývá hlavní nevýhoda MPA – nižší rychlost.

3.2.2 Single-Page aplikace

SPA je na rozdíl od MPA založena na technologii renderingu ze strany klienta (*client-side*). Její princip je znázorněn na obrázku 3.3. Hlavní odlišností od renderingu ze strany serveru je to, že server není zodpovědný za žádnou logiku aplikace nebo zpracovávaní dat. Server pouze poskytuje data získaná z datové vrstvy a veškeré jejich zpracování se provádí na straně klienta. Klient obsahuje jak prezentační, tak i aplikační vrstvu aplikace. Hlavní myšlenka je v tom, že díky takovému přístupu odpadává potřeba posílat mezi serverem a klientem celou webovou stránku a provádět tak překreslování celé stránky. Namísto toho klient přijímá pouze data nutná pro změnu ve formátu JSON a dle nich provádí překreslení jen příslušných částí stránky.



Obrázek 3.3 Princip fungování SPA [12]

SPA má několik podstatných výhod oproti MPA [11]. Hlavní je rychlost načítaní webové stránky. Celá stránka se načítá pouze jednou při její inicializaci. Poté už se aktualizují jen části, které jsou měněny. Kód aplikace se ukládá na straně klienta, a proto je server zodpovědný pouze za práci s databází.

SPA se používá pro návrh aplikací s menší funkcionalitou, a proto je snáze udržitelná a nepotřebuje od vývojářů tolik času a finančních zdrojů jako MPA.

Díky zpracovávání kódu na straně uživatele a komunikaci se serverem pouze v případě nutnosti aktualizace dat, umí SPA dobře pracovat s cache. Důsledkem je možnost převedení velké části funkcionality do offline režimu.

Jako bonus je backend SPA použitelný jak pro webovou, tak i pro mobilní aplikaci. Díky tomu je SPA snáze rozšiřitelná do mobilní verze aplikace.

Zvolený přístup

Cílem mé aplikace je správa menších poznámek. K tomu není potřeba ukládat velké množství dat, ani v budoucnu přidávat zásadně odlišnou funkcionalitu. Kvůli tomu jsem se rozhodla, že se pro ni více hodí SPA – modernější a rychlejší varianta.

3.3 Technologie

V této sekci se budu věnovat výběru vhodných technologií pro frontendovou i backendovou část aplikace.

3.3.1 Základní technologie

Existují dvě nejpopulárnější technologie pro vytváření SPA: knihovna React [13] a framework Angular [14].

React (nebo ReactJS) je knihovna vyvíjená společností Facebook a zveřejněná v roce 2013. Efektivně pracuje s interaktivními prvky a má dobrou dokumentaci, díky které je začít programovat v Reactu snadné i pro začátečníky. React využívá metodu virtuálního DOM. Při každé změně se vytváří nový virtuální DOM, který se porovnává s předchozím a ve skutečném DOM se poté mění jen nové, odlišné prvky [15]. Díky tomu jsou aktualizace webové stránky rychlejší.

Na rozdíl od knihovny React má framework Angular rozsáhlejší funkcionalitu. Je starší (první verze byla vydána v roce 2010) a vyvíjen společností Google. Má přesně stanovenou strukturu návrhu aplikace. Frontendová část aplikace se ukládá do HTML souboru a chování prvků je definováno v JavaScript souborech.

Obě technologie se průběžné vyvíjí, mají dobrý výkon a poskytují pro moje účely postačující funkcionalitu. Jsou také podporované velkou komunitou vývojářů. Nakonec jsem si zvolila React, protože dává vývojáři více volnosti z pohledu následujícího návrhu aplikace.

3.3.2 Firebase

K ulehčení práce s databází jsem se rozhodla využít vývojovou platformu Firebase [16], která je podporována společností Google (první vydaní v roce 2016). Firebase má rozsáhlou funkcionalitu a poskytuje služby k ulehčení vývoje aplikace. Z nich jsem nejvíce použila *Realtime Database*, *Firebase Authentication* a *Firebase Hosting*.

Mezi požadavky na funkcionalitu mojí aplikaci jsou také registrace a přihlášení uživatelů. Vývoj vlastního přihlašovacího systému, který navíc bezpečně ukládá údaje, je časově náročný a pracný proces, proto pro autentizaci a správu uživatelů Firebase nabízí svou službu Firebase Authentication. [17] Vestavěný autentizační systém platformy Firebase podporuje různé metody přihlášení. Mezi nimi jsou přihlášení přes:

- poštovní schránku a heslo,
- telefonní číslo,
- dočasný anonymní účet,
- účty Google, Facebook, Twitter, GitHub atd.

Firebase poskytuje cloudovou databázi Realtime Database, která je založena na konceptu NoSQL. Tato databáze ukládá data ve formátu JSON a synchronizuje je v reálném čase napříč všemi připojenými klienty. [18] Bezpečnostní pravidla databáze určují, kdo je oprávněn číst a zapisovat data, a chrání tak své uživatele před neoprávněným přístupem.

Webhosting neboli zkráceně hosting je pronájem prostoru pro webovou stránku. Tuto službu poskytuje i Firebase, který zajišťuje bezpečné připojení přes HTTPS protokol z nejbližšího serveru globální sítě CDN. Nasazení se provádí snadno přes terminál. [19]

3.3.3 Bootstrap

Pro navržení vzhledu aplikace jsem zvolila CSS framework Bootstrap. CSS framework je sada stylů, které se dají aplikovat na HTML značky pro získání určitého vzhledu elementů. [20] Při manuální opravě vzhledu uživatelského rozhraní se HTML prvkům přidávají atributy *class* a na jejich základě se poté určují CSS styly. Bootstrap obaluje obyčejné HTML prvky a přidává jim vlastní styly. Díky tomuto frameworku může vývojář použít hotové nadesignované komponenty a nevěnovat tolik času psaní CSS stylů.

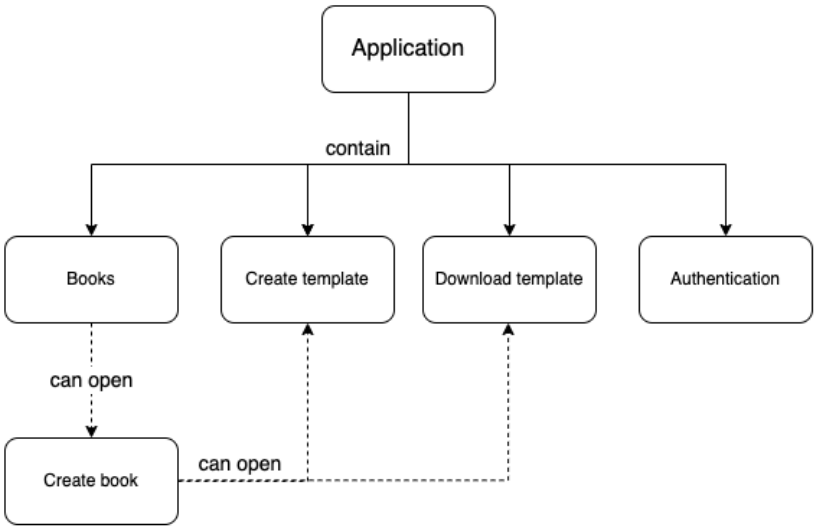
3.3.4 Programovací jazyk

Aplikaci jsem se rozhodla implementovat v programovacím jazyce JavaScript z důvodu svých předchozích zkušeností s ním. Alternativou by mohl být jazyk TypeScript, který navíc umí například kontrolu typů při kompilaci.

3.4 Základní návrh architektury aplikace

Stránky

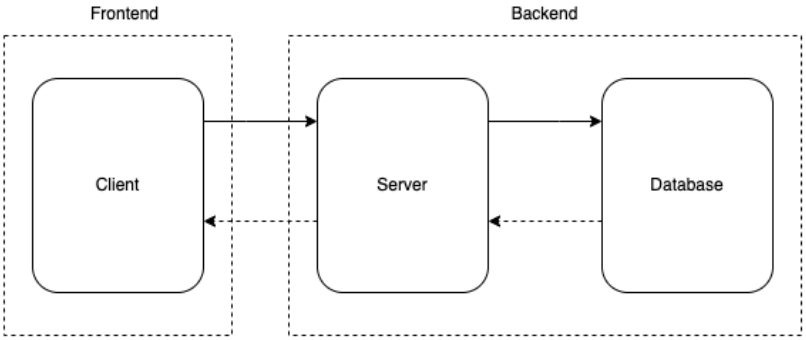
Jádrem aplikace jsou poznámkové bloky vytvořené uživatelem (tj. tematické knihy – *books*) a šablony (*templates*). Na nich je postavena celá aplikace. Následně lze knihy vytvářet a prohlížet. Šablony lze buď vytvářet vlastní, nebo stáhnout veřejné. Další nezbytnou součástí aplikace je autentizace uživatelů. Obrázek 3.4 obsahuje základní rozdělení aplikace na webové stránky. Každá stránka obsahuje pouze logiku zobrazení aplikace, jinými slovy pouze frontendovou část aplikace.



Obrázek 3.4 Rozdělení aplikace na stránky

Rozdělení architektury a práce s databází

Při vytváření webové aplikace vypadá běžný přístup následovně:



Obrázek 3.5 Typický návrh webové aplikace

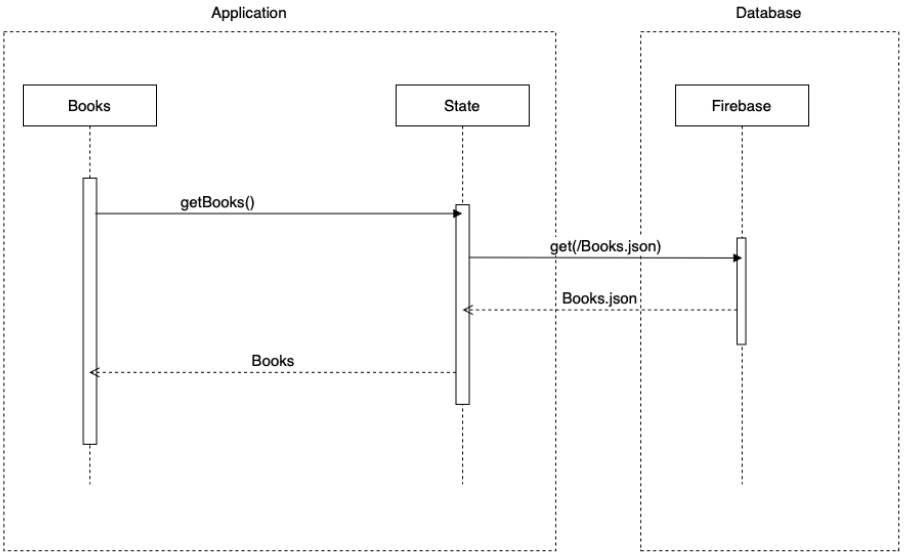
Návrh je přesně rozdělen na tři části: klient, server a databáze.

Databáze neobsahuje žádnou logiku a je jen úložištěm dat. Klientem se obvykle rozumí webový prohlížeč, který pracuje pouze s frontendovou částí aplikace.

Server přijímá požadavky od klienta, sbírá příslušná data z databáze, zpracovává je a vrací je v nutné podobě klientovi. Server je zodpovědný za celou logiku aplikace.

Výše popsaný přístup upravím takovým způsobem, aby více vyhovoval typu aplikace a zvoleným technologiím.

Firebase není jen databází, ale vývojovou platformou s rozsáhlou funkcionalitou. Neobsahuje pouze data, ale i velkou část serverové logiky, proto není třeba vyvíjet samostatný server. V mé aplikaci bude jak frontendová, tak i backendová část umístěná na straně klienta, odkud bude komunikovat napřímo s Firebase. Obrázek 3.6 znázorňuje základní myšlenku rozdělení aplikace.



Obrázek 3.6 Základní rozdělení aplikace na komponenty

Pro názornost popíši funkcionalitu třídy *Books*, která bude vypisovat seznam poznámkových bloků uživatele. *Books* v sobě neobsahuje žádná data. K tomu aby věděla, jaké elementy má

zobrazit, se zeptá třídy *State*. *State* posílá požadavek na Firebase a následně zpracovává a ukládá příslušná data. Díky tomu má třída *Books* přístup k datům a může uživateli zobrazit seznam knih.

Vytváření nové knihy

Více dopodrobna jsou získaní dat aplikací a komunikace s databází znázorněné na obrázku 3.7, na kterém jsem znázornila proces vytváření nové knihy.

Pro vytvoření nové knihy musí uživatel otevřít stránku *„Books“*, která obsahuje seznam všech jeho knih, a stisknout tlačítko *„Create new Book“*. Při otevírání stránky se aplikace zeptá databáze, jaké knihy patří uživateli, Firebase odešle odpověď ve formátu JSON, aplikace ji zpracuje a vykreslí seznam knih. Uživatel stiskne tlačítko a otevře se stránka *„Create book“* se seznamem šablon a možností zadat název nové knihy. Stejně jako v případě seznamu knih se aplikace zeptá databáze na šablony uživatele a vykreslí je.

Poté nastává jedna ze tří možnosti:

- uživatel si přeje vytvořit knihu podle jedné ze svých šablon,
- uživatel si chce stáhnout novou šablonu ze seznamu veřejných,
- uživatel si chce vytvořit novou vlastní šablonu.

V případě, že uživateli vyhovuje jedna z jeho šablon, pokračuje ve vytváření knihy na stejné stránce. Zvolí šablonu ze seznamu, napíše název knihy a stiskne tlačítko *„Create new book“*. Aplikace odešle novou knihu databázi a v případě, že ukládání proběhlo úspěšně, zobrazí potvrzovací zprávu. V tento okamžik je kniha úspěšně vytvořena a až uživatel znovu otevře stránku *„Books“*, zobrazí se v seznamu i nově vytvořená kniha.

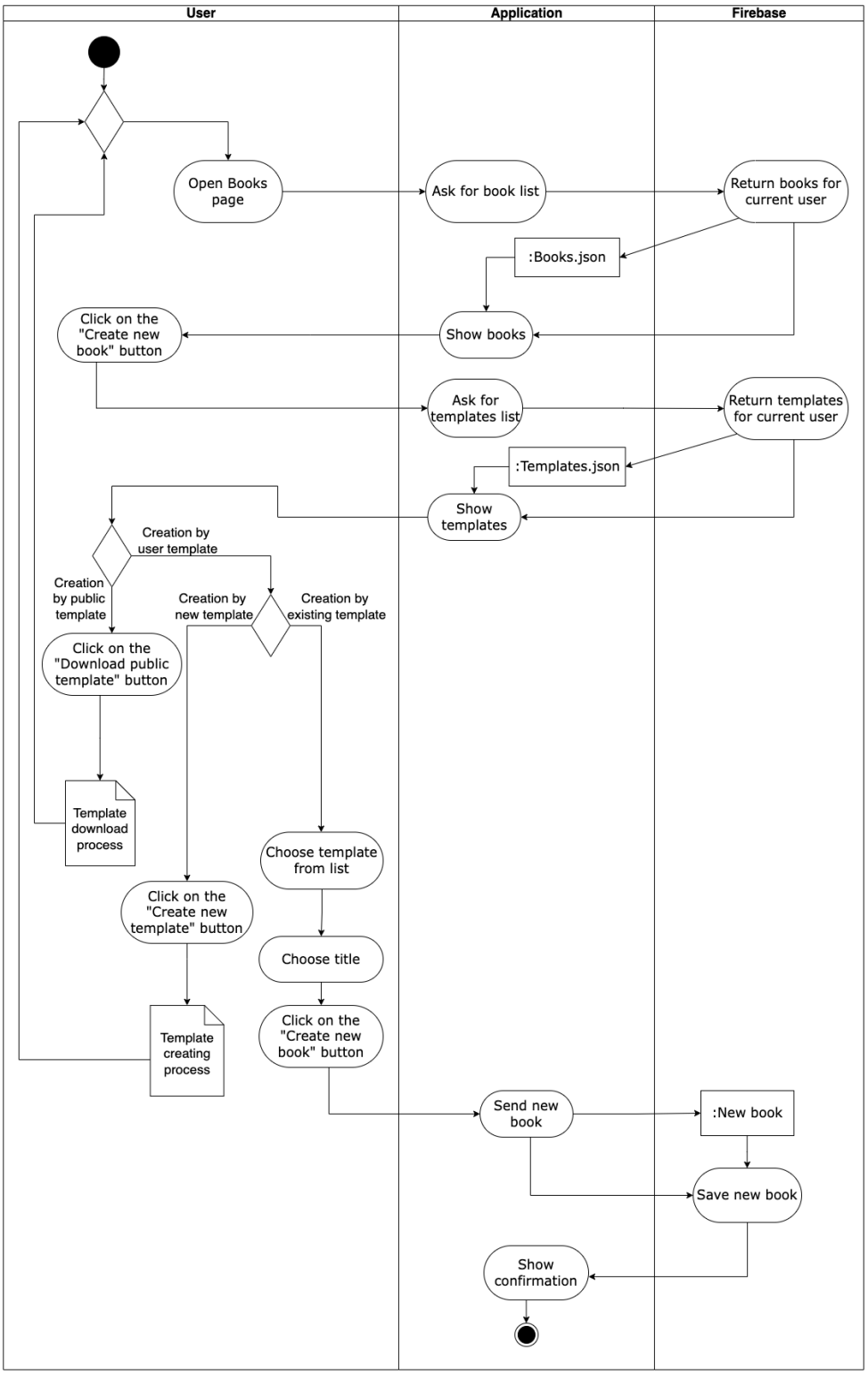
Pokud si uživatel chce stáhnout novou šablonu ze seznamu veřejných šablon, stiskne tlačítko *„Download public template“*, které ho přesměruje na stránku *„Download template“*.

Podobná situace nastane, pokud si uživatel chce vytvořit šablonu vlastní, s tím rozdílem, že stiskne tlačítko *„Create new template“* a dostane se na stránku *„Create template“*.

Obrázek 3.7 nezahrnuje proces stažení ani vytváření šablon, ale stačí pro základní pochopení získaní dat a komunikace s databází.

Po stažení nebo vytvoření nové šablony musí uživatel zopakovat proces vytváření od začátku tj. otevřít stránku *„Books“*, stisknout tlačítko *„Create new book“*, zvolit šablonu a název a stisknout tlačítko *„Create new book“*.

Tohle je pouze zjednodušený obrázek, vice dopodrobna některé části tohoto procesu popíši v kapitole 4.

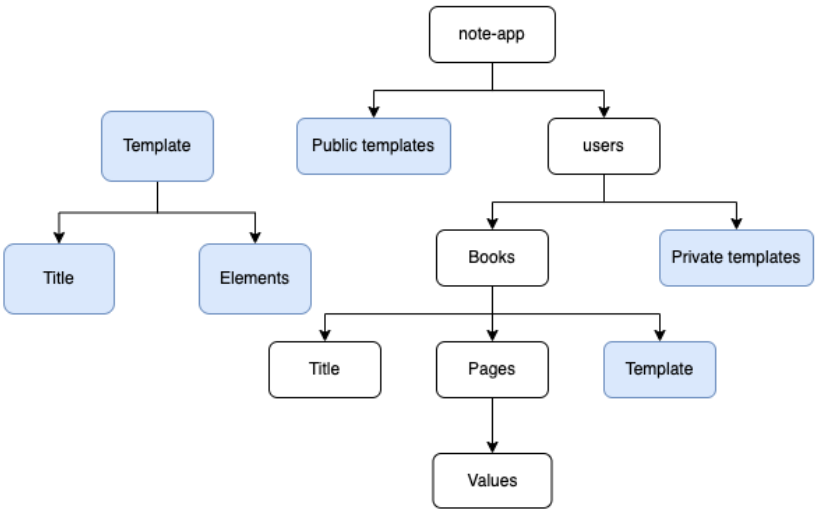


Obrázek 3.7 Proces vytváření nové knihy

Kapitola 4
Implementace

4.1 Databáze

Databáze mé aplikace v Firebase má následující rozhraní:



Obrázek 4.1 Rozhraní databáze

Na první úrovni databáze obsahuje veřejné šablony a uživatele. Šablona se skládá z názvu a elementů. Každý uživatel má vytvořené knihy a seznam jeho soukromých šablon. Kniha se vytváří podle šablony, má název a obsahuje stránky. Stránka se zobrazuje podle elementů šablony a do databáze se ukládají pouze hodnoty interaktivních elementů, jako jsou například *input*, *textarea* a *checkbox*.

Během vývoje jsem narazila na to, že Firebase neukládá do databáze prázdné objekty. Narazila jsem na to, když jsem při vytváření nové knihy chtěla do atributu *pages* uložit prázdné pole, aby ho pak uživatel rozšiřoval přidáním stránek do knihy. To však působilo všemožné komplikace, kdy například:

- nebylo jednoznačné, co se má vykreslovat při otevírání prázdné knihy,
- při vytváření první stránky ještě neexistoval atribut *pages*,

- vypočítání počtu stránek knihy z délky seznamu *pages* vyhazovalo výjimku, jelikož seznam *pages* ještě neexistoval.

Jedním z možných řešení bylo ošetřit všechny problematické situace, to by s sebou ovšem přineslo velké větvení kódu, proto jsem se nakonec rozhodla vytvářet knihy s jednou prázdnou stránkou a tím se všechny problémy vyřešily.

Pro práci s databází jsem použila knihovnu Axios [21], která umí transformovat JSON formát. Používala jsem tzv. CRUD operace, tj. metody *get*, *post*, *put* a *delete*.

4.2 Šablony

Hlavním odlišením mé bakalářské práce od obyčejných poznámkových bloků, které má každé chytré zařízení, jsou šablony. Aplikace má několik předpřipravených šablon (cestovaní, vaření, filmy, sport atd.), které si uživatel může stáhnout do seznamu svých soukromých šablon. Dále si uživatel může vytvořit vlastní šablonu, přičemž ji bude mít možnost zveřejnit. Kniha se vždy vytváří podle šablony a podle ní se také zobrazuje.

Vytváření šablon je realizováno přes dynamické přidávaní prvků do HTML. Do výpisu kódu 4.1 jsem přidala funkci *addPageElement*, která se používá všude, kde se zobrazují šablony.

Výpis kódu 4.1 Dynamické přidávání prvků do HTML [1]

```
export const addPageElement = (parentId, elementTag, elementId, html) =>
{
  let parentElement = document.getElementById(parentId);
  let elementToAdd = document.createElement(elementTag);
  elementToAdd.setAttribute("id", elementId);
  elementToAdd.innerHTML = html;
  parentElement.appendChild(elementToAdd);
  return elementToAdd;
};
```

Nový prvek se přidává do elementu s ID *parentId*, tag a text se určují podle parametrů *elementTag* a *html*, ID nového prvku se rovná *elementId*.

Šablony se vytváří na stránce *„Create template“*, která se skládá z několika částí:

- nabídka elementů, které uživatel může přidat do nové šablony,
- ukázka vzhledu nové šablony,
- seznam přidaných elementů,
- pojmenování, zveřejnění a uložení šablony.

Nabídka elementů obsahuje záhlaví, inputy a větší textové bloky. V budoucnu by se nabídka mohla rozšířit například i o obrázky nebo seznamy. Po přidání se elementy zobrazí uprostřed v aktuálním přehledu šablony. V seznamu přidaných elementů lze smazat jeden nebo všechny prvky. Po zadání názvu si uživatel může rozhodnout, jestli chce šablonu zveřejnit, nebo nechat soukromou. Po stisknutí tlačítka *„Save template“* se šablona uloží do databáze a přidá se do seznamu šablon aktuálního uživatele.

Na stránce *„Download template“* a při vytváření knihy se šablony zobrazují stejným způsobem jako přehled v *TemplateCreator.js*. V komponentě *Book* jsem při vykreslení šablony editovatelným prvkům (*input, textarea*) navíc přidala funkci, která průběžně ukládá změny do databáze.

4.3 Hooky

Ze začátku jsem aplikaci začala implementovat pomocí tzv. React tříd (*class*). Časem jsem ale zjistila, že se poslední dobou více využívá kombinace funkcionálního přístupu a moderní technologie React Hooků [22]. Kvůli tomu jsem všechny komponenty změnila na funkcionální a přidala hooky.

Hook je funkce, která vývojáři dovoluje pracovat se stavem a životním cyklem elementů přes funkcionální komponenty. Hooky si lze napsat sám jako obyčejné funkce, ovšem React pro lidi, kteří využívají jejich knihovnu, připravil několik vestavěných hooků. [23] Během celého vývoje jsem opakovaně používala některé z nich, a proto bych je tu chtěla alespoň krátce popsat.

Hook stavu: useState

Hook *useState* uchovává stav lokálních proměnných komponent mezi rendery. Přijímá inicializační hodnotu a vrací aktuální stav a funkci, přes kterou lze proměnnou měnit [23]. Po každé změně React provádí rerendering příslušné komponenty. *UseState* je základní hook, díky kterému lze ukládat a měnit informaci mezi překresleními komponent.

Tento hook jsem využila skoro v každé své komponentě, jelikož je základem práce se stavy. Ukázku jeho využití jsem přidala do výpisu kódu 4.2.

Výpis kódu 4.2 Příklad využití hooku *useState* v *BookCreator.js*

```
export const BookCreator = () => {
    const [bookTitle, setBookTitle] = useState("");
    const createBook = () => {
        addBook(bookTitle, templates[selectedTemplate])
        //..
    };
    return (
        <div>
            //..
            <Form saveValue={setBookTitle} placeholder="Write␣title"/>
            <h2>{bookTitle}</h2>
            //..
        </div>
    );
};
```

V tomto příkladu se jedná o komponentu *BookCreator*, která je zodpovědná za přidání nové knihy. Přes hook *useState* se vytváří proměnná *bookTitle* a metoda *setBookTitle*. *BookTitle* uchovává název knihy, který se vypisuje v záhlaví a mění se v komponentě *Form* pomocí *setBookTitle*. Při každé změně se provede překreslení stránky a zobrazí se aktuální název. Dále se název odesílá jako parametr do metody *addBook*, která přidává novou knihu do databáze.

Hook efektu: useEffect

Hook efektu *UseEffect* umožňuje provádět vedlejší efekty ve funkčních komponentách. [24] Pokud *useEffect* nemá druhý parametr nebo je druhým parametrem prázdné pole, změny se provádí před každým překreslením stránky. Jinak se provádí pouze v případě, že se stav druhého parametru změnil.

Nejčastěji jsem tento hook používala pro zajištění aktuálnosti seznamu knih nebo šablon. Kupříkladu komponenta *TemplateLoader*, určená pro stahování veřejných šablon, v sobě má hook *useEffect*, který obsahuje metodu aktualizace šablon – *fetchPublicTemplates* (viz výpis kódu 4.3). To zaručuje průběžné objevování změn na stránce. Jedná se například o přidání a mazání šablon.

Výpis kódu 4.3 Příklad využití hooku *useEffect* v *TemplateLoader.js*

```
export const TemplateLoader = () => {
    useEffect(() => {
    fetchPublicTemplates();
  }, []);
  //..
}
```

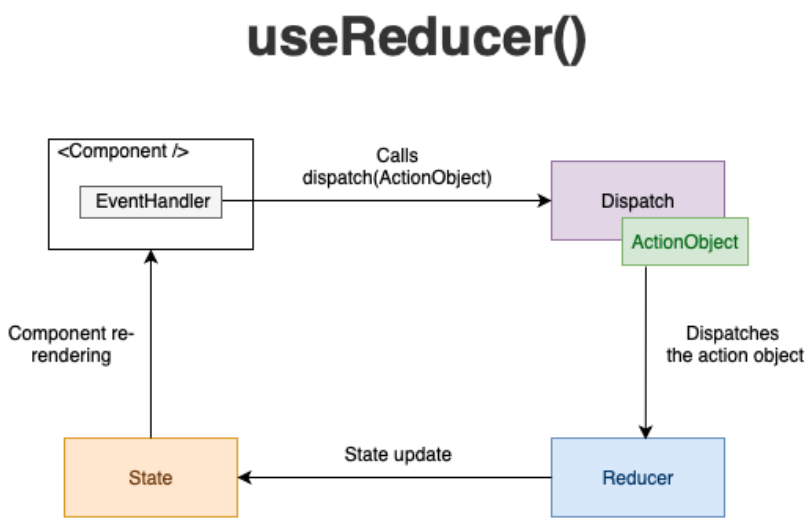
Hook useReducer

Hook *useReducer* je komplexnější a těžší na vysvětlení, ovšem na principu jeho fungování se zakládá logika celé aplikace. Tento hook je alternativou výše popsaného *useState*, ale hodí se více při složitější stavové logice, kdy je například další stav závislý na předchozím. [2]

Výpis kódu 4.4 Příklad využití hooku *useReducer* [2]

```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

Jako parametry *useReducer* přijímá *reducer* – funkce, která na základě typu akce mění aktuální stav na nový – a inicializační stav. *UseReducer* vrací pár – aktuální stav (*state*) a metodu *dispatch*, která umožňuje měnit stav proměnné *state*. Proces aktualizace stavu je znázorněn na obrázku 4.2.



Obrázek 4.2 Princip fungování hooku *useReducer* [25]

Když komponenta potřebuje aktualizaci, vyvolá se metoda *dispatch* s příslušným typem akce. Dále *dispatch* zavolá funkci *reducer*, která byla předána jako první parametr do hooku *useReducer*. Na základě typu akce *reducer* aktualizuje stav a na konci se provede překreslení komponenty.

Na hooku *useReducer* je postavena velká část logiky mé aplikace. Tomu, jak je zařazen do architektury mé implementace, jsem věnovala celou sekci 4.4.

4.4 Kombinace FirebaseState a firebaseReducer

Jádrem architektury mé aplikace je kombinace komponenty *FirebaseState* a funkce *firebaseReducer*. Frontendové komponenty nemají přímý přístup do databáze, proto k tomu používají třídu *FirebaseState*, přes kterou je realizována komunikace s databází. Další funkcí *FirebaseState* je ukládání dat aktuálního uživatele, aby k nim měly přístup ostatní komponenty.

Výpis kódu 4.5 obsahuje inicializaci komponenty *FirebaseState*. V proměnné *state*, kterou vrací *useReducer*, se ukládají všechny položky obsažené v *initialState*:

- *currentUser* uchovává aktuálního uživatele,
- *books* je seznam knih příslušících uživateli,
- *templates* je seznam soukromých šablon uživatele,
- *publicTemplates* je seznam všech veřejných šablon,
- dle proměnné *loading* se zobrazuje nebo skrývá ikona načítání.

Výpis kódu 4.5 *FirebaseState.js*

```
export const FirebaseState = ({ children }) => {

  const initialState = {
    currentUser: null,

    books: [],
    templates: [],

    publicTemplates: [],
    loading: false,
  };
  const [state, dispatch] = useReducer(firebaseReducer, initialState);
  //..
}
```

Metoda *dispatch* bude vyvolávat funkci *firebaseReducer* z výpisu kódu 4.6. Na základě typu akce *firebaseReducer* vrací nový stav, kterým metoda *dispatch* vymění proměnnou *state*.

Typy akce jsou uložené v souboru *types.js* a zpravidla odpovídají seznamu akcí, které uživatel může v aplikaci provádět (vytvoření a mazání knihy, stahovaní šablony atd.). Vyskytují se mezi nimi i typy, se kterými uživatel neinteraguje, jako je zobrazení a skrytí ikony načítání nebo upozornění. Několik příkladů typů akce jsem uvedla ve výpisu kódu 4.7.

Výpis kódu 4.6 *firebaseReducer.js*

```
export const firebaseReducer = (state, action) => {
  const handle = handlers[action.type] || handlers.DEFAULT;
  return handle(state, action);
};
```

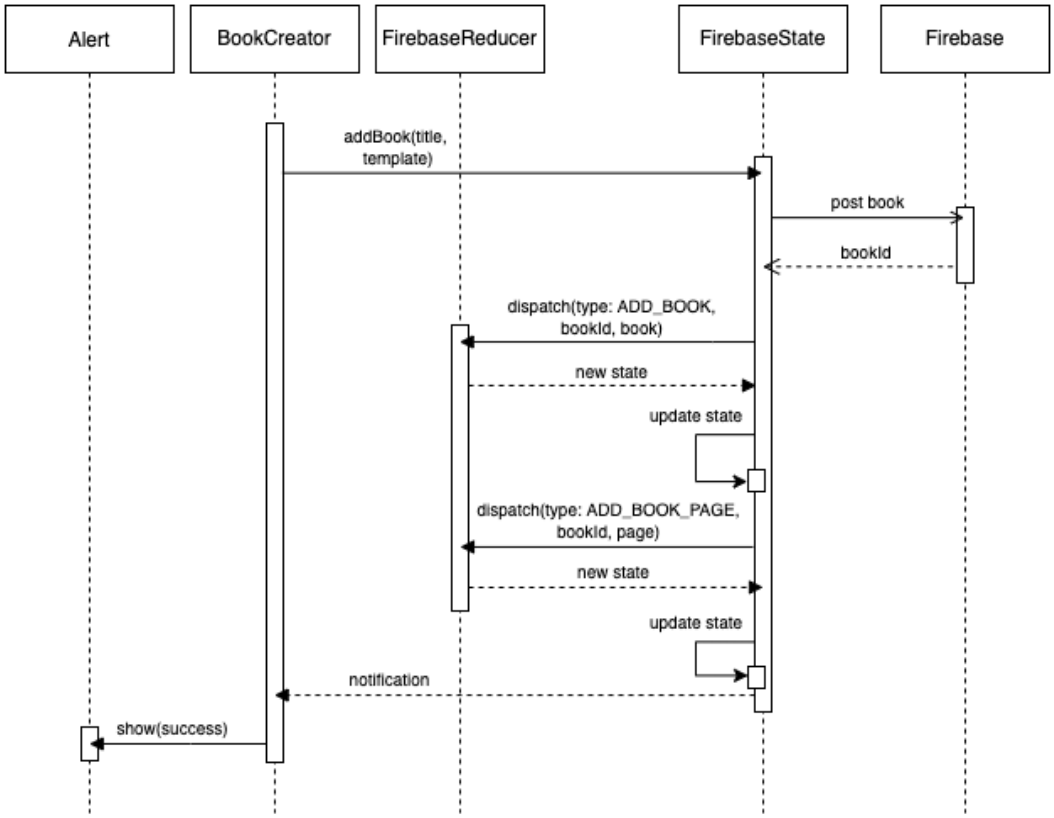
Výpis kódu 4.7 *types.js*

```
export const ADD_BOOK = "ADD_BOOK";
export const FETCH_BOOKS = "FETCH_BOOKS";
export const REMOVE_BOOK = "REMOVE_BOOK";

export const SHOW_LOADER = "SHOW_LOADER";
export const CLOSE_LOADER = "CLOSE_LOADER";
/..
```

Vytváření knihy

Na příkladu vytváření knihy (viz obrázek 4.3) popiši proces spolupráce komponent mé aplikace.



Obrázek 4.3 Proces vytváření nové knihy

Knihy se vytváří na stránce *„Create book“* z obrázku 3.4. Tuto stránku reprezentuje komponenta *BookCreator*, která po vyplnění názvu a zvolení šablony vyvolá metodu *addBook* komponenty *FirebaseState*. Nejdříve se provede ukládání do databáze. Přes funkci *axios.post FirebaseState* pošle informaci o nové knize a uvnitř odpovědi dostane ID vygenerované databází. Dále se zavolá *dispatch* s akcí typu *„ADD_BOOK“*, získaným ID a samotnou knihou. Podle typu akce *firebaseReducer* vrací novy stav, ve kterém se objeví vytvořená kniha. V tomto případě bude změna stavu vypadat následovně:

Výpis kódu 4.8 *firebaseReducer.js*

```
[ADD_BOOK]: (state, { payload }) => {
    const newBooks = state.books;
    newBooks[payload.bookId] = payload.book;
    return {
      ...state,
      books: newBooks,
    };
  },
```

Metoda *dispatch* provede výměnu předchozího stavu na nový, tj. aktualizuje proměnnou *state* z výpisu kódu 4.5.

Jelikož se kniha vytváří s prázdnou stránkou, zavolá se metoda *addBookPage*. Proběhne obdobný proces, který obsahuje aktualizaci stavu na základě návratové hodnoty funkce *firebaseReducer*. Po obdržení upozornění, že vše proběhlo bez chyb, vyvolá *BookCreator* vyskakovací okno, které uživateli sdělí, že byla kniha úspěšně vytvořena.

4.5 Služby Firebase

Ve velké míře se Firebase stará o autentizaci. Nově registrované uživatele ukládá do databáze, provádí šifrování hesel, a to tak, že ani vývojář k nim nemá přístup. Firebase také spravuje přihlašování uživatelů a kontrolu tokenů.

Autentizační stránka se skládá z registrace a přihlašování. Její rozhraní jsem vytvořila na základě příkladu z oficiálních stránek frameworku Bootstrap [26], které obsahují velké množství předpřipravených vzorů.

Nasazení aplikace do reálného provozu nebylo cílem mé bakalářské práce, ale pro ulehčení testování jsem zveřejnila aplikaci na hostingu, který poskytuje Firebase. Aplikace momentálně běží na adrese `all-your-notes-app.web.app`.

4.6 Stránkování

Přechod mezi stránkami se provádí pomocí komponent *Switch*, *Route* a *Link* z balíčku *react-router-dom*. V komponentě *Link* je uložena adresa, kam bude uživatel přesměrován po kliknutí na odkaz. Kupříkladu *Link* z výpisu kódu 4.9 odkazuje na adresu */template_creator*, tj. na stránky vytváření šablony.

Výpis kódu 4.9 Příklad využití komponenty *Link*

```
<Link to={{ pathname: `/template_creator` }}> {/*...*/} </Link>
```

Switch hraje roli rozcestníku, který porovnává adresu s atributem *path* u všech svých *Route*. Při shodě se zobrazí komponenta, přidaná do atributu *component*. Pro výše uvedený případ se vykreslí komponenta *TemplateCreator*.

Výpis kódu 4.10 Příklad využití komponenty *Switch*

```
<Switch>
    <Route path={"/template_creator"} component={TemplateCreator} />
    <Route path={"/books"} component={BookNavigation} />
    <Route path={"/book_creator"} component={BookCreator} />
    <Route path={"/authentication"} component={Authentication} />
    <Route path={"/template_loader"} component={TemplateLoader} />
</Switch>
```

Z toho plyne, že se při změně adresy ve skutečnosti neodesílá žádný požadavek na server, ale pouze se mění mezi sebou komponenty. Díky tomu je aplikace chráněna před bezpečnostními riziky. Uživatel se tudíž nemá možnost například zadat do adresy jiné UID a dostat k cizím datum.

Kapitola 5
Testování

Po dokončení implementace jsem provedla uživatelské testování, kterého se zúčastnilo pět lidí. Aplikace byla otestována na operačních systémech Windows, Linux a MacOS, a to v prohlížečích Google Chrome a Safari.

Typickým testovacím scénářem bylo:

- zaregistrovat se do aplikace,
- přihlásit se do aplikace,
- vytvořit novou šablonu,
- vytvořit novou knihu podle soukromé šablony,
- stáhnout šablonu,
- vytvořit novou knihu podle stažené šablony,
- otevřít knihu,
- vyplnit stránku knihy,
- vytvořit novou stránku a vyplnit ji.

Aplikace spuštěná na různých operačních systémech a v různých prohlížečích měla pouze drobná stylistická odlišení. Žádný závažný rozdíl z pohledu funkcionality nebyl odhalen.

Testováním jsem zjistila, že funkcionality aplikace jsou snadno pochopitelné. Při používání aplikace, si nejvíce testeři stěžovali na to, že se názvy šablon a knih ukládají jen po stisknutí klávesy „Enter“. To bylo častým zdrojem chyb a testeři poměrně často vytvářeli knihy bez názvu. Pro vyřešení tohoto problému jsem u názvu změnila ukládání vstupu na průběžné a přidala zákaz vytváření knih a šablon bez vyplněného nazvu.

5.1 Frontend

Přestože jsem pro návrh uživatelského rozhraní používala framework Bootstrap, ukázalo se, že frontend je slabá stránka mé aplikace. Čtyři z pěti testerů naznačili, že nepovažují rozhraní za uživatelsky přívětivé. Zmiňovali především to, že je v některých případech matoucí, ale i to, že celkově aplikace vypadá podprůměrně.

Kvůli tomu jsem upravila vzhled celé aplikace. Po testování jsem během opravy provedla větší množství změn, část z nich je v následujícím seznamu.

- Přidala jsem blokování editovatelných prvků šablony (input, textarea) při zobrazování jejího náhledu na stránkách vytváření knihy a stahovaní šablony.
- Vzhled tlačítek jsem odlišila podle jejich účelu na modrou, bílou a šedou barvu.
- Přidala jsem tlačítka „Back to your books“ na stránky prohlížení a vytváření knih.
- Změnila jsem stránku autentizace. Po úpravě neautorizovaný uživatel vidí pouze formu registrace a přihlášení. Autorizovaný uživatel vidí tlačítko odhlášení a uvítací stránku, která nabízí přechod do seznamu knih.
- U vyskakovacího upozornění jsem upravila pozici a dobu zobrazení. V dřívější verzi aplikace se upozornění umísťovalo hned vedle horního menu a proto si ho uživatel nevšiml (pokud byla stránka obsažnější), dokud se manuálně neposunul na začátek stránky. Dříve se navíc upozornění nezavíralo automaticky a zobrazovalo se i po přesunutí na jiné stránky, pokud jej uživatel neodstranil ručně. Po opravě se nyní fixně zobrazuje na horní části obrazovky nad ostatními komponentami a po deseti vteřinách samo zmizí.

Po opakovaném testování se testeři shodli na tom, že po opravě vypadá aplikace o dost lépe a že je intuitivnější.

Další připomínky vzešlé z testování, které se do aplikace nepřidaly, a svoje nápady jsem zmínila v možnostech dalšího rozšíření.

Závěr

Naplnění cílů

Cílem mé bakalářské práce bylo vytvořit aplikaci pro správu poznámek různých typů. Nedílnou součástí aplikace byly šablony na různá témata. Na začátku práce jsem prozkoumala již existující podobné aplikace, označila funkční a nefunkční požadavky a navrhla řešení. Tím jsem splnila cíle rešeršní části práce. Následně jsem aplikaci implementovala. Výsledkem je webová aplikace napsaná v programovacím jazyce JavaScript pomocí knihovny React. Pro přívětivé uživatelské rozhraní byl použit framework Bootstrap. Aplikace podporuje uživatelské účty a vytváření neomezeného množství zápisníků, a to buď podle předpřipravených, vytvořených nebo stáhnutých šablon. Uživatelé mají možnost své vlastní šablony sdílet ostatním uživatelům. Tímto způsobem jsem splnila požadavky z kapitoly 2. Na konci jsem také provedla otestování výsledné aplikace.

Možnosti dalšího rozšíření

Do budoucna bych chtěla navrhnout několik směrů rozšíření aplikace.

- Hodnocení veřejně sdílených šablon s možností řadit dle něj šablony v jejich seznamu. To by uživatelům pomohlo stahovat nejužitečnější šablony, aniž by je museli dlouho hledat.
- Přidání dalších prvků do sekce vytváření vlastních šablon. K přidání navrhuji například místa na mapě, obrázky, události.
- Lokalizace aplikace do dalších jazyků.
- Možnost přidání stylů prvkům šablony dle preferencí uživatele.

Zhodnocení výsledků a využitelnost v praxi

Výsledek mé práce je dobře využitelný pro uživatele, kteří si neradi instalují různé aplikace pro různé typy poznámek, ale zároveň chtějí své poznámky oddělovat dle témat (např. recepty, filmy, sport). Vytváření nových šablon a zápisníků je snadné a rychlé, a díky Firebase hostingu je aplikace nasazena do reálného provozu.

Příloha A

Instalační příručka

Nasazená verze

Aplikace je dostupná na adrese `all-your-notes-app.web.app`, odkud může být přímo spuštěna a používána.

Spuštění ze zdrojových kódů

Pro spuštění aplikace ze zdrojových kódů je třeba mít nainstalovaný Node.js spolu se správcem balíčků npm. Aplikaci pak lze spustit podle následujících kroků:

- V terminálu přejděte do kořenového adresáře zdrojového kódu.
- Zadáním příkazu npm install nainstalujte požadované balíčky.
- Zadejte příkaz npm start, který zkompiluje a spustí aplikaci na lokální adrese `http://localhost:3000`.

Bibliografie

1. POHATH, E. Create elements dynamically within React component. In: *Stack Overflow* [online]. 2017 [cit. 2021-10-22]. Dostupné z: `https://stackoverflow.com/questions/43873145/create-elements-dynamically-within-react-component`.
2. Hooks API Reference. In: *React Docs* [online]. Meta Platforms, Inc., 2021 [cit. 2021-12-28]. Dostupné z: `https://reactjs.org/docs/hooks-reference.html`.
3. MAYNE, P.; TEAM DAY ONE. *Day One* [soft.]. Bloom build, Inc., 2021. Ver. 6.13 [cit. 2021-11-30]. Dostupné z: `https://dayoneapp.com`.
4. PARTL, T. *Diarium* [soft.]. Timo P, 2021. Ver. 2.6.6 [cit. 2021-11-30]. Dostupné z: `https://timopartl.com/?app=Diarium`.
5. *Grid Diary* [soft.]. Xiamen Sumi Network Technology Co., Ltd., 2021. Ver. 2.2.10 [cit. 2021-11-30]. Dostupné z: `https://griddiaryapp.com`.
6. *Evernote* [soft.]. Evernote, 2021. Ver. 10.25.6 [cit. 2021-11-30]. Dostupné z: `https://evernote.com`.
7. *Notion* [soft.]. Notion Labs, Inc., 2021. Ver. 2.0.19 [cit. 2021-12-14]. Dostupné z: `https://www.notion.so/`.
8. Desktop App vs Web App: Comparative Analysis. In: *Digital Skynet Corp. blog* [online]. Digital Skynet CORP, 2020 [cit. 2021-12-03]. Dostupné z: `https://digitalskynet.com/blog/Desktop-App-vs-Web-App-Comparative-Analysis`.
9. MAL, N. Web App vs. Desktop App: What Is the Difference? In: *Qulix Systems* [online]. Qulix Systems, 2021 [cit. 2021-12-03]. Dostupné z: `https://www.qulix.com/about/web-app-vs-desktop-app/`.
10. Why you should choose a web application over a desktop one. In: *w3 lab digital agency* [online]. 2020 [cit. 2021-12-03]. Dostupné z: `https://w3-lab.com/choose-web-application-over-desktop/`.
11. VALUY, S. A Comparison of Single-Page and Multi-Page Applications. In: *DZone Web Dev* [online]. DZone, Inc., 2020 [cit. 2021-12-16]. Dostupné z: `https://dzone.com/articles/the-comparison-of-single-page-and-multi-page-appli`.
12. STRIMPEL, J.; NAJIM, M. *Building Isomorphic JavaScript Apps: From Concept to Implementation to Real-World Solutions*. Sebastopol, CA 95472: O'Reilly Media, Inc., 2016. ISBN 978-1-491-93293-3.
13. *React* [soft.]. Meta Platforms, Inc., 2021 [cit. 2021-12-27]. Dostupné z: `https://reactjs.org`.
14. *Angular* [soft.]. Google, 2021 [cit. 2021-12-27]. Dostupné z: `https://angular.io`.

15. Angular Vs React: Difference Between Angular and React. In: *InterviewBit* [online]. 2021 [cit. 2021-12-20]. Dostupné z: `https://www.interviewbit.com/blog/angular-vs-react/`.

16. *Firebase* [soft.]. Google Developers, 2020. Ver. 9.6.1 [cit. 2021-12-14]. Dostupné z: `https://firebase.google.com`.

17. MORONEY, L. *The Definitive Guide to Firebase*. Seattle, Washington, USA: Apress, Berkeley, CA, 2017. ISBN 978-1-4842-2942-2.

18. TANNA, M.; SINGH, H. *Serverless Web Applications with React and Firebase*. Birmingham: Packt Publishing Ltd., 2018. ISBN 978-1-78847-860-1.

19. YAHIAOUI, H. *Firebase Cookbook*. Birmingham, UK: Packt Publishing, 2017. ISBN 978-1-78829-239-9.

20. COPELAND, D. *Rails, Angular, Postgres, and Bootstrap: Powerful, Effective, Efficient, Full-Stack Web Development*. Pragmatic Bookshelf, 2017. ISBN 978-1-68050-444-6.

21. *Axios* [soft.]. Matt Zabriskie, 2014. Ver. 0.23.0 [cit. 2021-12-28]. Dostupné z: `https://axios-http.com`.

22. Introducing Hooks. In: *React Docs* [online]. Meta Platforms, Inc., 2021 [cit. 2021-12-27]. Dostupné z: `https://reactjs.org/docs/hooks-intro.html`.

23. Hooks at a Glance. In: *React Docs* [online]. Meta Platforms, Inc., 2021 [cit. 2021-12-27]. Dostupné z: `https://reactjs.org/docs/hooks-overview.html`.

24. Using the Effect Hook. In: *React Docs* [online]. Meta Platforms, Inc., 2021 [cit. 2021-12-28]. Dostupné z: `https://reactjs.org/docs/hooks-effect.html`.

25. PAVLUTIN, D. An Easy Guide to React useReducer() Hook. In: *Dmitri Pavlutin* [online]. 2021 [cit. 2021-12-28]. Dostupné z: `https://dmitripavlutin.com/react-usereducer/`.

26. BOOTSTRAP TEAM. Examples. In: *Bootstrap* [online]. 2021 [cit. 2021-11-30]. Dostupné z: `https://getbootstrap.com/docs/5.1/examples/sign-in/`.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF