ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Výuka s použitím flash cards

Palina Nalbandian
Softwarové inzenýrství a technologie

Srpen 2022
Vedoucí práce: Ing. Božena Mannová, Ph.D

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nalbandian** Jméno: **Palina** Osobní číslo: **474532**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Výuka s použitím flash cards

Název bakalářské práce anglicky:

Flash cards in education

Pokyny pro vypracování:

Cílem práce je analýza, návrh, implementace a testování webové aplikace pro vytváření flash karet - virtuálních karet, které pomáhají lidem při studiu. Aplikace bude sloužit jako pomůcka pro výuku a přípravu ke zkouškám. Kartička (flash card) má na jedné straně pojem a na obrácené straně definici. Aplikace umožní vytvářet sety kartiček a systematizovat výuku.. Aplikaci navrhněte a implementujte s použitím vhodných prostředků SE. Aplikaci otestujte pomocí heuristické evaluace a UAT testů. Zhodnoťte své řešení a navrhněte případná vylepšení.

Seznam doporučené literatury:

[1] Roger S. Pressmann Bruce Maxim: Software Engineering: A Practitioner's Approach ,
ISBN-10: 9780078022128
[2] https://www.ankiapp.com/
[3] https://collegeinfogeek.com/flashcard-apps

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Božena Mannová, Ph.D. kabinet výuky informatiky FEL

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

Datum zadání bakalářské práce: **02.07.2021** Termín odevzdání bakalářské práce: **15.08.2022**

Platnost zadání bakalářské práce: **19.02.2023**

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

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studentky

Poděkování / Prohlášení

Chtěla bych poděkovat své vedoucí inženýrce Boženě Mannové za její rady a pomoc při psaní závěrečné práce. Dále bych chtěla poděkovat svým blízkým, že mě nenechali vzdát a vždy mě podporovali na této nelehké cestě. Děkuji také svému pracovnímu týmu, který mi vždy vyšel vstříc, když bylo potřeba věnovat více času na napsání závěrečné práce.

Prohlašuji, že jsem tuto práci vypracovala samostatně pouze s použitím literatury, kterou deklarují v kapitole Literatura.

V Praze dne 15. 08. 2022

Abstrakt / Abstract

Cílem této práce je implementace webové aplikace pro flash karty. Tato aplikace slouží jako pomocník pro zapamatování různých druhů učiva (cizí slova, termíny, vzorce). Implementace spočívá ve sběru a analýze požadavků na aplikaci, návrhu uživatelského rozhraní, implementaci serverové a klientské části aplikace a finální testování.

Klíčová slova: bakalářská práce, flash karty, java, react..

The purpose of this project is the implementation of a web application for flash cards. This application serves as assistant to people in memorizing various types of subject matter (foreign words, terms, formulas). The implementation consists of the collection and analysis of requirements for the application, the design of the user interface, the implementation of the server and client parts of the application, and final testing and deployment.

Keywords: bachelor, flashcards, java, react.

Obsah /

Kapitola 1
Úvod

1.1 Téma

Tématem této bakalářské práce je kompletní webová aplikace, která bude simulovat a nahrazovat ruční psaní kartiček pro zapamatování různých druhů materiálů. I přesto, že byla aplikace původně koncipována jako pomocník speciálně pro stu- denty, poslouží i dalším sociálním skupinám.

1.2 Předmluva

Toto téma jsem si vybrala z několika důvodů. Téměř během celého studia jsem se připravovala na různé zkoušky a testy vytvořením několika „karet“, které mi pomohly vizualizovat materiál a lépe si ho zapamatovat a vypracovat. Nejprve jsem si tyto karty vyráběla sama – na papíře. Tato metoda se mi opravdu líbila, ale bralo to hodně času. Pak jsem objevila některé online služby pro vytváření takových karet, ale nemohla jsem najít nějaké, které by byly úplně zdarma nebo ty, které by nabídly celou řadu funkcionalit za nízkou cenu.

Proto jsem se rozhodla, vytvořit takovou webovou aplikaci pro vytváření flash karet, sama. Kromě toho bych velmi ráda využila znalosti technologií, kterým jsme se během studia učili a které jsou potřebné pro realizaci tohoto projektu. V následujících kapitolách podrobněji popíšu, jaký bude účel této webové aplikace. V kapitole “Rešerše” popíšu problém, který mě podnítil k zahájení vytvořené této webové aplikace.

Poté se v kapitole “Analýza” podíváme na další řešení a vysvětlím, proč jsem se rozhodla pro vlastní řešení. Podotknu také finanční stránky projektu a rizik, která mohou vzniknout jak během implementace, tak po i ní. Seznámím vás se všemi požadavky na aplikaci, které vznikly při analýze tohoto projektu. Poté specifikuji technologickou stránku projektu. Uvedu, jaké technologie použiji k implementaci, k čemu budou potřebné, za co budou odpovědné a jak jsou propojeny.

V kapitole “Návrh” bude vybrána a vysvětlena architektura aplikace. Ukážu i některé high-fidelity prototypy, které budou sloužit jako základ pro vytvoření grafického rozhraní aplikace.

Kapitola “Implementace” vás provede všemi fázemi vývoje projektu od vytvoření databáze, až po návrh uživatelského rozhraní. Jak byla aplikace nasazena na cloudovou platformu, bude popsáno v kapitole “Na- sazení”.

Kapitola “Testování” popisuje, jak byla aplikace testována – jak na straně backendu, tak na straně uživatelského rozhraní.

Kapitola 2
Rešerše

2.1 Seznámení s problémem

V rámci přípravy na zkoušky a testy procházejí dnes studenti a školáci spoustu materiálů, ukládají si do paměti desítky stránek poznámek v naději, že si zapamatují a pochopí někdy velmi složitý a obtížně zapamatovatelný obsah.

Vědci rozlišují u lidí motorickou, verbálně-logickou, emocionální a obraznou paměť [1]. U každého z nás převládá jeden nebo více těchto typů – díky jejich rozvoji se stáváme schopnějšími zapamatovat si někdy velmi složité a obsažné pojmy. Různé metody memorování materiálů jsou vhodné pro různé lidi – někdo vypráví nahlas to, co si zapamatoval, někdo si píše poznámky a ty si zapamatuje, ale nepochybně jedním z nejúčinnějších způsobů, které používali i naši rodiče, je sestavování karet, kde na jedné straně bude otázka/pojem a na jiné straně – odpověď na otázku nebo definice pojmu.

Kartičky jsou účinné, protože vás nutí vytahovat informace z paměti (místo toho, abyste je jen četli nebo rozpoznávali), a to vám pomáhá rychleji si vytvářet vzpomínky a podávat mnohem lepší výsledky v testech. Jsem přesvědčena, že nejen školáci a studenti čelí problému zapamatování si nového učiva. Se stejným problémem se čas od času setkají i dospělí, ať už je čeká rekvalifikace, další vzdělávání či se potřebují naučit cizí jazyk.

Shrneme-li to tedy, můžeme dojít k závěru, že aplikace pro vytváření karet pro zapamatování informací, jejich uchování v paměti a následné opakování či použití je rozhodně relevantní právě v tomto moderním světě. Lidé si stále více zvykají konzumovat velmi rychle informace z internetu jedním kliknutím, mají menší snahu najít potřebné informace v knihách a knihovnách, což nepochybně vede k problémům s pamětí.

Aplikace flash card se snaží tento problém vyřešit – pomáhá lidem zlepšovat jejich paměť tím nejjednodušším, ale nejvýkonnějším způsobem – vytvářením flash kartiček.

Kapitola 3
Analýza

Tato kapitola poskytne kompletní analýzu tohoto projektu.

Budou shromážděny a systematizovány funkční a nefunkční požadavky na aplikaci, analyzována budou stávající řešení uvedeného problému a navrženo bude vlastní řešení.

Kromě toho budou v této kapitole rozebrány technologie, které lze při realizaci projektu použít, bude učiněn a určen konečný výběr.

V rámci analýzy byla zvažována i finanční stránka projektu a rizika, která mohou nastat při jeho realizaci a po nasazení.

Na závěr budou shrnuty výsledky provedené analýzy.

3.1 Sběr požadavků

Za hlavní zdroj funkčních požadavků na projekt považuji sebe a své osobní zkušenosti s přípravou na zkoušky a memorováním látky, navíc jsem mezi svými přáteli a známými provedla malou anketu na téma, co by na takové práci nejvíce ocenili, jaké funkcionality by rádi viděli.

Výsledky mě trochu překvapily, protože většina respondentů oceňuje jednoduchost uživatelského rozhraní a intuitivnost používání služby.

Na druhém místě z hlediska důležitosti pro uživatele byla zaznamenána rozmanitost funkcionalit, které aplikace nabízí – významná část respondentů by ocenila možnost zadávat vzorce a vkládat obrázky do karet.

Na základě průzkumu byly sestaveny hlavní funkční a nefunkční požadavky na výslednou aplikaci, které jsou uvedeny níže.

3.1.1 Funkční požadavky

Tato podkapitola představuje seznam všech funkčních požadavků na projekt identifikovaných analýzou.

Funkční požadavky zachycují systém zamýšleného chování, jež může být vyjádřeno jako služby, úkoly nebo funkce, které má systém vykonávat [2].

Přehled požadavků je reprezentován jako seznam, kde každá položka má svůj identifikátor ve formátu FRQ + unikátní číslo.

- FRQ01 – Systém bude umožňovat založit účet.
- FRQ02 – Systém bude umožňovat se přihlásit.
- FRQ03 – Systém bude umožňovat změnit a uložit údaje profilu.
- FRQ04 – Systém bude umožňovat uživateli smazat svůj profil.
- FRQ05 – Systém bude umožňovat vytvořit sadu karet.
- FRQ06 – Systém bude umožňovat měnit údaje card boxu.

- FRQ07 – Systém bude umožňovat vytvořit kartu v nějakém card boxu.
- FRQ08 – Systém bude umožňovat upravovat údaje karty.
- FRQ09 – Systém bude umožňovat smazat card box.
- FRQ10 – Systém bude umožňovat smazat kartu.
- FRQ11 – Systém bude umožňovat uživateli vyhledat si libovolný card box, který je v public stavu.
- FRQ12 – Systém bude umožňovat uživateli si otevřít Card Box.
- FRQ13 – Systém bude umožňovat nechat komentář k Card Boxu.
- FRQ14 – Systém bude umožňovat uživateli přidat card box do oblíbených.
- FRQ15 – Systém bude umožňovat uživateli nechat zpětnou vazbu a poslat e-mail.

3.1.2 Nefunkční požadavky

Nefunkční požadavky jsou požadavky, které definují vlastnosti, které musí systém vykazovat, nebo omezení, která musí splňovat a která nesouvisejí s chováním systému. Například výkon, udržovatelnost, rozšiřitelnost, spolehlivost, provozní faktory [3].

- NFRQ01: Intuitivní rozhraní – rozhraní aplikace uživatel používá intuitivně, bez speciálních pokynů.
- NFRQ02: Kompatibilita – aplikace je dostupná uživatelům z různých zařízení a prohlížečů.
- NFRQ03: Rychlost – aplikace funguje rychle, reaguje na interakci ze strany uživatele rychlou odezvou.

3.2 Analýza hotových řešení

Po důkladné analýze trhu a aplikací, které fungují stejně jako pomocník při zapamatování materiálů, byly nalezeny dvě hlavní, nejoblíbenější alternativy mezi uživateli – Brainscape a Quizlet.

Obě možnosti mají sadu bezplatných funkcí, ale také poskytují celou řadu doplňujících funkčností na základě měsíčních plateb.

Dále popíšu každou variantu a zvážím její klady a zápory.

3.2.1 Brainscape

„Brainscape"[1] je webová aplikace, která nejvíce odpovídá požadavkům pro tento projekt. Jediným problémem je, že většina funkcí je k dispozici pouze v placené verzi. Bezplatná verze má omezený počet sad karet, nelze přidávat obrázky, karty není možné obrátit a v bezplatné verzi jsou všechny uživatelské karty veřejné.

Aplikace má tolik funkcí a možností, že se v ní uživatel nejprve začne ztrácet. Je docela těžké se orientovat ve smyslu určitých funkcí.

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

Problém je také v tom, že aplikace je dostupná pouze v angličtině. V kapitole 3.1 jsem uváděla, že aplikace by měla být snadno použitelná pro lidi všech věkových skupin, neměla by uživatele, který poprvé otevře web, vyděsit, jeho oči by neměly běhat divoce – měl by okamžitě vědět, kam kliknout, aby vytvořil box a kartičku. Systém by měl být intuitivně ovladatelný. Myslím si, že Brainscape tyto požadavky nesplňuje na 100 procent.

Můžeme se však inspirovat širokou nabídkou funkčností, které tato aplikace poskytuje pro další vývoj naší webové aplikace.

Klady:

- široká škála funkcí
- uspokojí většinu požadavků
- mobilní aplikace

Zápory:

- plná sada služeb pouze v placené verzi
- UI jenom v angličtině
- interface není uživatelsky přívětivý

3.2.2 Quizlet

Aplikace „Quizlet"[2] rovněž značně vyhovuje požadavkům na náš projekt. Poskytuje uživatelům možnost vytvářet karty, testy a mnoho dalšího.

Aplikace má také velmi „user-friendly" rozhraní, je velmi snadné ji pochopit a může ji používat osoba s absolutně jakoukoli zkušeností s používáním podobných webových aplikací. Má intuitivní navigaci a snadno se ovládá uživatelem jakékoliv věkové skupiny.

Navíc aplikace existuje ve webové i mobilní verzi a poskytuje uživatelům různé metody pro míchání a zobrazování karet.

Klady:

- User-friendly" interface
- základní funkce jsou zdarma
- mobilní aplikace

Zápory:

- plná sada služeb pouze v placené verzi

3.3 Vlastní řešení

Nyní se dostáváme k mému vlastnímu řešení, jehož demo název je „Flip Card". Řešením problému je návrh aplikace, která řeší dvě běžné nevýhody hotových řešení popsaných výše – nedostatek plné řady funkcí zdarma a uživatelský přívětivý interface.

Výše uvedená dvě řešení vyžadují měsíční poplatky pro uživatele, kteří chtějí aplikaci plně využívat. Naše aplikace nebude takový poplatek vyžadovat – bude zcela zdarma.

Aplikace bude muset být přístupná uživatelům všech věkových skupin, při používání této aplikace by se nikdo neměl setkat se žádnými problémy.

[2] `https://quizlet.com/`

Aplikace umožní uživatelům vytvářet karty na jakékoliv téma s jakýmkoliv obsahem, který přispěje k lepšímu pochopení studovaného materiálu. Aplikace je zamýšlena tak, aby uživatel mohl sadu karet zpřístupnit pouze pro sebe, nebo pro všechny ostatní uživatele.

3.4 Use Case model

Případ užití je metodologie používaná v systémové analýze k identifikaci, objasnění a uspořádání systémových požadavků. Je tvořen souborem možných sekvencí interakcí mezi systémy a uživateli v konkrétním prostředí a souvisejících s konkrétním cílem. Metoda vytvoří dokument, který popisuje všechny kroky provedené uživatelem k dokončení aktivity [4].

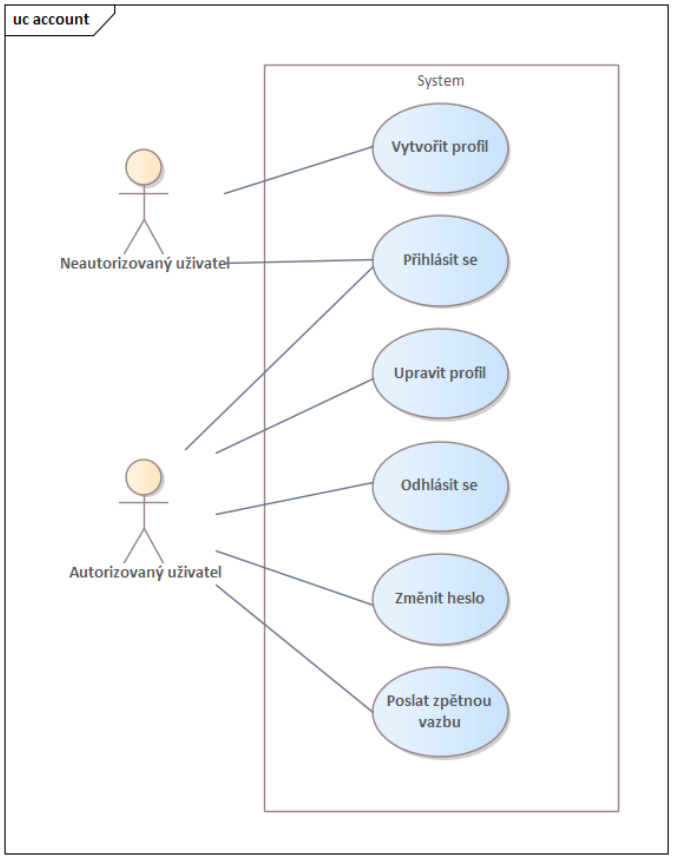
Případy užití vyplývají z definovaných požadavků, určí všechny základní aktivity uživatelů v aplikaci. Projekt bude implementovat dva typy uživatelů – autorizované a neautorizované.

Seznam aktérů a jejich popis jsou uvedeny níže.

- Neautorizovaní uživatelé – buď si budou moci založit účet, pokud jej ještě nemají, nebo se přihlásit, pokud už účet založený mají.
- Autorizovaní uživatelé – mají přístup ke všem funkcím aplikace.

Pro srozumitelnost a lepší pochopení procesů v systému je níže uvedeno několik diagramů případů užití, které popisují hlavní funkcionální části aplikace.

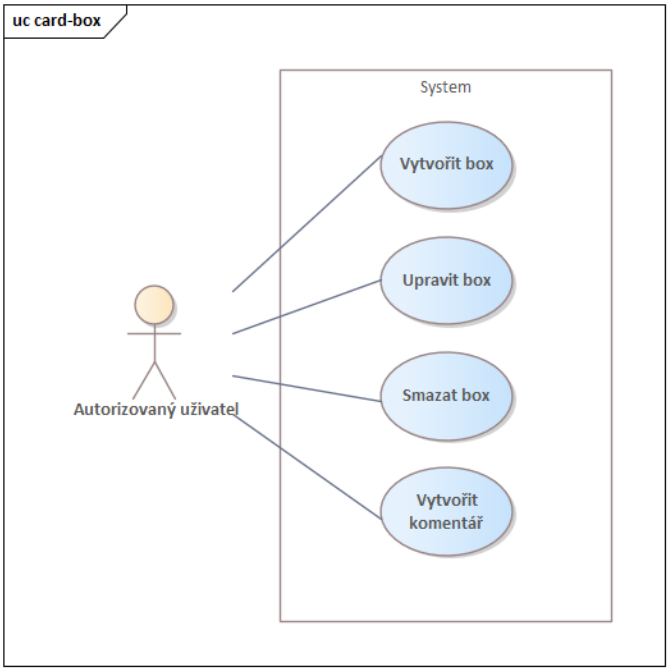
- Account. Use case “Account” zobrazuje funkce související s osobním profilem uživatele.



Obrázek 3.1. Use Case “Account”

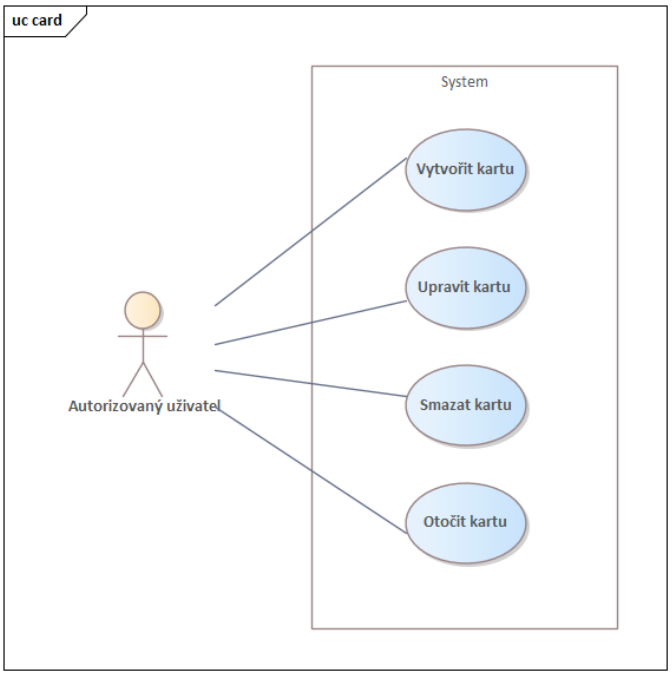
Ukazuje, že neautorizovaný uživatel si může vytvořit profil, nebo se přihlásit do existujícího. Zároveň může autorizovaný uživatel spravovat svá profilová data a odesílat e-maily se zpětnou vazbou.

- Card Box. Use case "CardBox" představuje veškerou funkcionalitu, kterou má autorizovaný uživatel ve vztahu ke kard boxu.



Obrázek 3.2. Use Case "Card Box"

- Card. Use case "Card" představuje všechny funkce, které má autorizovaný uživatel ve vztahu ke kartám.



Obrázek 3.3. Use Case "Card"

3.5 Technologie

Tato kapitola představuje analýzu všech technologií použitých při realizaci projektu.

Je uvedeno srovnání mezi vývojáři nejoblíbenějších technologií, jejich silných a slabých stránek a důvod výběru konkrétní technologie.

V budoucí kapitole "Implementace" bude popsáno, jak byla vybraná technologie implementována v rámci projektu.

3.5.1 Databáze

Databáze (DB) je organizovaná struktura určená k ukládání, úpravě a zpracování souvisejících informací, a to většinou velkých objemů. Databáze jsou aktivně využívány pro dynamické stránky se značným množstvím dat – často se jedná o internetové obchody, portály, firemní stránky. [5]

Tato část poskytuje definici, srovnání a seznam výhod a nevýhod dvou typů databází – relační a nerelační.

- **Relační databáze (SQL)**

  Relační databáze (SQL) je databáze, kde jsou data uložena ve formátu tabulek, jsou přísně strukturované a vzájemně propojené. Tabulka má řádky a sloupce, přičemž každý řádek představuje jeden záznam a sloupec představuje pole s přiřazeným datovým typem. V každé buňce jsou informace zapsány podle šablony. Relační databáze je ideální pro práci se strukturovanými daty, jejichž struktura nepodléhá častým změnám. [6]

  Zvláštnosti:

  - Hlavním rysem je spolehlivost a neměnnost dat, nízké riziko ztráty informací. Při aktualizaci dat je zaručena jejich integrita, nahrazují se v jedné tabulce.
  - Relační databáze na rozdíl od nerelačních vyhovují ACID – to jsou požadavky na transakční systémy. Jejich dodržování zaručuje bezpečnost dat a předvídatelnost databáze. [7]

  Při práci s takovými DBMS je třeba vzít v úvahu, že případné změny objektů se musí promítnout do struktury tabulek, fyzická datová struktura neodpovídá objektovému modelu aplikace.

- *MySQL* je jednou z nejpopulárnějších open source relačních databází, jež je vhodná pro malé a střední projekty, které potřebují levný a spolehlivý datový nástroj. [8]

  Klady:

  - Podporuje mnoho typů tabulek.
  - Existuje obrovské množství pluginů a rozšíření, usnadňujících práci se systémem.
  - Vyznačuje se jednoduchou instalací.
  - Může být integrován s jinými DBMS.

  Zápory:

  - Ne všechny úkoly se provádějí automaticky, pokud něco, co potřebujete, není součástí funkčnosti, budete muset trávit čas vylepšováním.
  - Neexistuje žádná vestavěná podpora OLAP.

- *PostgreSQL* je druhá nejoblíbenější open source SQL databáze. [9]

  Klady:
  - Umožňuje pracovat se strukturovanými daty, podporuje JSON, což dává určitou flexibilitu ve schématu dat.
  - Je vhodná, pokud je důležitá bezpečnost dat, očekává se jejich složitá struktura.
  - Má mnoho vestavěných funkcí a doplňků, včetně škálování a sdílení tabulek.
  - Je stabilní, je téměř nemožné ji zakázat nebo něco rozbít v tabulkách.

  Zápory:
  - Složitost konfigurace vyžaduje určité zkušenosti uživatelů.
  - Rychlost práce může klesnout během dávkových operací nebo během požadavků na čtení.

- **Nerelační databáze (NoSQL)** – ukládá data bez jasných vazeb na sebe a jasné struktury. Místo strukturovaných tabulek je v databázi mnoho heterogenních dokumentů, včetně obrázků, videí, a dokonce i příspěvků na sociálních sítích. Na rozdíl od relačních databází nepodporují databáze NoSQL dotazy SQL.

  Zvláštnosti:
  - Na rozdíl od relačních databází je v nerelačních databázích datové schéma dynamické a může se kdykoli změnit.
  - Obtížnější je přístup k datům, tedy najít něco potřebného uvnitř databáze – s tabulkou je to snadné, stačí znát souřadnice buňky. Ale takové DBMS se liší výkonem a rychlostí.
  - Fyzické objekty v NoSQL lze většinou uložit přímo v podobě, v jaké s nimi pak aplikace pracuje.

  NoSQL databáze jsou vhodné pro ukládání velkého množství nestrukturovaných informací a také pro rychlý vývoj a testování hypotéz. Můžete do nich ukládat data libovolného typu a v průběhu práce přidávat nová.
  Příkladem takové databáze je MongoDB.

- *MongoDB* je open source databáze typu dokumentu.

  Klady:
  - Dokáže pracovat se strukturovanými i nestrukturovanými daty. Při práci s relačními datovými modely však mohou nastat problémy s výkonem.
  - Vhodné pro projekty, které pracují s heterogenními daty, jež je obtížné klasifikovat, nebo pokud se v budoucnu očekává významná změna ve struktuře dat.
  - Tato databáze se dobře horizontálně škáluje bez ztráty rychlosti, snadno se používá, dobře funguje a je vhodná pro velké objemy dat.
  - Snadno se instaluje a vyzkouší, mnoho nastavení.

Zápory:

- Nepoužívá SQL jako dotazovací jazyk, existují nástroje pro překlad SQL dotazů, ale vyžadují konfiguraci.
- Nechybí ani datová konektivita.
- MongoDB je obtížné udržovat, protože vyžaduje zkušenosti s NoSQL. [10]

3.5.2 Server

V této kapitole budou porovnané hlavní programovací jazyky pro serverovou stranu aplikace, budou zanalyzované silné a slabé stránky a bude proveden výběr nejvhodnější varianty pro projekt.

Java

Java je příkladem programovacího jazyka pro vývoj webových aplikací na straně serveru. Objektově orientovaný programovací jazyk je široce používán pro vývoj webových aplikací na podnikové úrovni spolu s vývojem aplikací pro Android, desktopových aplikací, vědeckých aplikací atd.

Hlavní výhodou použití Javy je, že funguje na principu Write Once Run Anywhere, tj. kompilovaný kód Java lze spustit na jakékoli platformě s podporou Java bez nutnosti rekompilace. Přesněji řečeno je kód Java nejprve zkompilován do bajtového kódu, který je nezávislý na počítači, a poté je tento bajtový kód spuštěn na JVM bez ohledu na základní architekturu.

Java navíc podporuje multithreading, který umožňuje běh dvou, nebo více vláken současně, aby se maximalizovalo využití procesoru.

Java frameworky pro vývoj webu na straně serveru: Spring, Struts, Grails.
Oblíbené webové stránky využívající Javu: LinkedIn, IRCTC, Yahoo atd.

Omezení Java:

- Kód Java může být složitý zejména ve scénářích, kde je aplikace velká.
- Java aplikace vyžadují ke spuštění více paměti.

PHP

PHP (nebo by se dalo říci hypertextový preprocesor) je veteránem ve světě webového vývoje. Tento open source jazyk na straně serveru byl vytvořen v roce 1994 a používá se speciálně pro vývoj webových aplikací. Vzhledem k tomu, že se jedná o interpretovaný jazyk, také nevyžaduje kompilátor a může běžet na téměř všech hlavních operačních systémech jako Windows, Linux, macOS, Unix atd.

Umožňují snadné učení, kompatibilitu napříč platformami, funkce OOP, podporu různých standardních databází jako MySQL, SQLite atd., obrovskou podporu komunity a mnoho dalších. Jinak je PHP jako skriptovací jazyk na straně serveru velmi bezpečný, protože PHP má mnoho hashovacích funkcí pro šifrování uživatelských dat. [11]

PHP frameworky pro vývoj webu na straně serveru: Laravel, CodeIgniter, Symfony atd.
Oblíbené webové stránky využívající PHP: WordPress, MailChimp, Flickr atd.

Python

Přestože je Python mezi lidmi poměrně známý svou kompatibilitou s pokročilými technologiemi, jako jsou Machine Learning, Internet of Things (IoT), Data Science atd., řekneme, že tento obohacující programovací jazyk je široce používaný a velmi vhodný také pro serverový webový rozvoj.

I jeden z předních IT gigantů v současnosti Google hodně spoléhá na Python, jenž je jedním ze tří hlavních jazyků, které používá (další dva jsou Java a C++).

Jednou z hlavních výhod používání Pythonu pro vývoj webu je obrovská sbírka standardních knihoven, díky nimž je práce vývojářů relativně snadná a efektivní. Další vynikající a jedinečné vlastnosti Pythonu jsou lepší čitelnost kódu, snadnější integrace s jinými jazyky, podpora programování GUI, přenositelnost. [12]

Python frameworky pro vývoj webu na straně serveru: Django, Flask, Pyramid atd.
Oblíbené webové stránky používající Python: Spotify, Pinterest, Instacart atd.

Omezení Pythonu:

- Spouštění kódu v Pythonu je ve srovnání s jinými programovacími jazyky pomalejší, proto může velký Python program běžet pomalu.
- Kódy Pythonu napsané v jedné verzi neběží hladce na jiných verzích. Pravděpodobně narazíte na chybu, zejména při používání knihoven.

3.5.3 API

Dvě samostatné aplikace potřebují ke vzájemné komunikaci prostředníka. Proto vývojáři často staví mosty – rozhraní pro programování aplikací neboli API, aby umožnily jednomu systému přístup k informacím nebo funkcím z jiného.

Pro usnadnění rychlé a rozsáhlé integrace aplikací jsou API implementován pomocí protokolů a/nebo specifikací, které definují sémantiku a syntaxi předávaných zpráv. Tyto specifikace tvoří architekturu API [13]

Dnes existují tři kategorie API protokolů nebo architektur: REST, SOAP a RPC. V této kapitole se budeme zabývat pouze prvními dvěma.

REST API

Representational State Transfer (REST) je architektonický styl, který definuje sadu omezení, která se mají použít pro vytváření webových služeb. REST API je způsob, jak přistupovat k webovým službám jednoduchým a flexibilním způsobem bez jakéhokoli zpracování.

REST není tak striktně definován jako SOAP. RESTful architektura musí splňovat šest architektonických omezení (Architektura klient-server, stateless, cache-able, uiniform interface, code on demand, layered system [14]):

Klady:

- Jediné rozhraní, které umožňuje konzistentní interakci se serverem bez ohledu na typ zařízení nebo aplikace.
- Bez stavu: nezbytný stav zpracování požadavku je obsažen v požadavku samotném, aniž by server ukládal jakákoli data související s relací.

- Ukládání do mezipaměti.
- Architektura klient-server.
- Schopnost serveru poskytovat spustitelný kód na klientovi.

Zápory:

- Nedostatek stavu: většina webových aplikací vyžaduje stavové mechanismy.
- Nedostatek zabezpečení: REST nevynucuje zabezpečení, jako je SOAP.

SOAP API

SOAP – je vysoce standardizovaný webový komunikační protokol založený na formátu XML. [15]

Klady:

- Jazyková a platformová nezávislost. Vestavěná funkce pro vytváření webových služeb umožňuje SOAP zpracovávat zprávy a provádět odpovědi nezávisle na jazyku a platformě.
- Konektivita s různými transportními protokoly. SOAP je flexibilní z hlediska přenosových protokolů a přizpůsobuje se více než jednomu scénáři.
- Vestavěné zpracování chyb. Specifikace SOAP API vám umožňuje vrátit zprávu XML Retry s kódem chyby a vysvětlením.
- Řada bezpečnostních rozšíření. Díky integraci s protokoly WS-Security splňuje kvalita transakcí SOAP podnikové standardy.

Zápory:

- Pouze XML. Zprávy SOAP obsahují mnoho metadat a podporují pouze podrobné struktury XML pro požadavky a odpovědi.
- Tíha. Vzhledem k velké velikosti souborů XML vyžadují služby SOAP velkou šířku pásma.
- Vysoce specializované znalosti. Vytváření serverů SOAP API vyžaduje hluboké pochopení všech zahrnutých protokolů a jejich přísných pravidel.
- Zdlouhavá aktualizace zpráv.

3.5.4 Klientská část

Pro implementaci klientské části webové aplikace bude použit jeden z frameworků Java Script.

V této podkapitole budou rozebrány dva hlavní frameworky, jejich klady a zápory a na základě toho bude vybrán ten nejvhodnější pro účely tohoto projektu. [16]

React.js

React.js je javascriptová knihovna vyvinutá Facebookem v roce 2013, je skvělá pro tvorbu moderních jednostránek aplikace libovolné velikosti a měřítka. [17]

Klady:

- Snadno se učí díky jednoduchému designu, použití JSX popř. TSX (syntaxe podobná HTML) pro šablony a velmi podrobné Dokumentace.
- velmi rychlé díky implementaci React Virtual DOM a různé optimalizace vykreslování.
- React implementuje koncepty funkcionálního programování (dále FP), vytváří snadno testovatelný a opakovaně použitelný kód.

Zápory:

- Velký výběr nástrojů je matoucí.
- Zvládnutí všech nuancí trvá dlouho.

Angular.js

Angular je framework JavaScript, který implementuje vzor MVVM. (dále Model-View-ViewModel), založený v roce 2009, je skvělý k vytváření interaktivních webových aplikací. [18]

Klady:

- Velké množství různých funkcí.
- Funkce jsou na sobě závislé.
- Minimální riziko chyby.

Zápory:

- Různé struktury (injekce, komponenty, trubky, moduly atd.) komplikuje proces učení oproti tomu v React, který má pouze "Component".
- Chyby během migrace mezi verzemi.

3.5.5 Zvolení řešení

Na základě poskytnuté analýzy možných řešení a osobních zkušeností se došlo k závěru, že implementace serverové části aplikace bude postavena na bázi Java frameworku – spring boot.

Pro sestavení databáze byla zvolena databáze typu SQL – PostgreSql.

Klientská strana bude implementována pomocí javascriptové knihovny React.

Pro komunikaci mezi serverovou a klientskou částí byl zvolen přístup REST API.

3.6 Finanční přínos

V této kapitole se budeme zabývat finanční stránkou projektu.

Většina podobných aplikací vydělává peníze tím, že nabízí předplatné uživatelům výměnou za plnou sadu uživatelských funkcí a offline přístup z aplikace.

Hlavním požadavkem uživatelů je, aby taková aplikace byla zdarma. Jsem připravena implementovat aplikaci, která bude tento požadavek splňovat. V budoucnu budu samozřejmě muset přijít na způsob, jak zpeněžit svůj projekt, protože udržování aplikace v průběhu času bude vyžadovat investice (pro platbu za servery atd.).

Za jedno z řešení problému financování takové aplikace považuji použití reklamních bannerů v naší aplikaci. To je docela efektivní způsob, mnoho webových aplikací používá tento typ dosažení příjmů. Studium je poměrně široké téma, do něhož lze integrovat obrovské množství vzdělávacích služeb, například těch, které poskytují vzdělávací kurzy. Aplikace samozřejmě nebude mít každý

den tisíce uživatelů, ale dokonce i jedna bannerová reklama bude schopna vyřešit problém počáteční nedostatečné investice.

3.7 Rizika

V následujících dvou kapitolách budu analyzovat všechna možná rizika, která mohou nastat během implementace tohoto projektu a po ní. Udělám seznam rizik, s jakou pravděpodobností k němu dojde, co se stane, pokud k němu dojde, a co je třeba udělat, aby se tomuto riziku zabránilo.

3.7.1 Rizika během implementace

Riziko	Pravděpodobnost výskytu	Dopad	Protiopatření
Nedostatečná znalost použitých technologii	Nizká	Neschopnost plně implementovat funkčnosti systému	Dobrý time management a kvalitní samostudium spring, react
Nedostatek času na realizaci projektu	Nizká	Může se prodloužit doba implementace aplikace	Kvalitní time-management

Tabulka 3.1. Rizika během implementace.

3.7.2 Rizika po nasazení

Riziko	Pravděpodobnost výskytu	Dopad	Protiopatření
Systém nebude využivan	Nizká	Ztráta financí kvůli nedosažené kvalitě produktu	Navrhnout takový systém, který by byl opravdu užitečný.
Nedostatek funkčnosti aplikaci	Střední	Systém nebude využivan	Kvalitní sběr požadavků

Tabulka 3.2. Rizika po nasazení

3.8 Shrnutí kapitoly

V této kapitole „Analýza“ byla provedena kompletní a podrobná analýza funkčních a nefunkčních požadavků výsledného systému na základě existujících řešení a sběru vlastních požadavků.

Poté byly postaveny hlavní UML diagramy, které budou základem pro realizaci samotného projektu.

V dalším kroku byla provedena analýza všech možných potenciálních technologií využitelných v rámci tohoto projektu a byla provedena optimální volba.

Kromě toho byla provedena finanční analýza projektu z hlediska jeho komerčního potenciálu do budoucna. Byly uvedeny hlavní rizikové faktory, se kterými se lze při realizaci projektu setkat, a jak tato rizika minimalizovat.

Kapitola 4
Návrh

Tato kapitola se bude zabývat návrhem webové aplikace, jejímž cílem je popsat řešení, které musí splňovat funkční a nefunkční požadavky a také omezení prostředí, ve kterém bude fungovat.

Dříve shromážděné požadavky jsou zpřesňovány a vylepšeny tak, aby vyhovovaly možným technologickým omezením.

4.1 Systémová Architektura

Během analýzy procesu návrhu webové aplikace se ukázalo, že architektura webu by měla být rozdělena do tří hlavních komponent: klient, server a databáze.

Byl také vyvinut datový model – databáze, se kterou bude serverová část aplikace komunikovat. Komunikace mezi klientskou a serverovou částí bude probíhat pomocí Rest API.

4.2 Fyzická Architektura

Architektura webové aplikace v podstatě představuje vztahy a interakce mezi komponentami, jako jsou uživatelská rozhraní, monitory zpracování transakcí, databáze a další. Hlavním cílem je ujistit se, že všechny prvky spolu správně spolupracují.
V rámci implementace této webové aplikace byla zvolena monolitická architektura. Toto řešení se ukázalo jako optimálnější než použití architektury mikroslužeb. [19]

Níže jsou uvedeny hlavní důvody pro výběr monolitické architektury:

- Snadný vývoj a snadné spouštění programu.
- Jediná technologie: projekt nevyžaduje složitou sadu technologií a často je možné vyvinout známou sadu technologií.
- Nízkonákladový – jeden programátor může dokončit celý proces realizace projekt.

4.3 Serverová Čast

Serverová část aplikace je vyvíjena pomocí Spring Framework, který se ve skutečnosti skládá z mnoha modulů odpovědných za různé součásti webu. [20]

Moduly jsou na sobě nezávislé, což nám umožňuje snadno je přidávat, nebo odebírat během vývojového procesu. Jako webový server, na kterém bude spuštěna serverová aplikace, bude provádět nejpopulárnější aplikační server Apache Tomcat. [21]

```
flipcard-backend-spring
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com.projects.flashcards
│   │   │       ├── controller
│   │   │       ├── enums
│   │   │       ├── exception
│   │   │       ├── model
│   │   │       ├── repository
│   │   │       ├── security
│   │   │       ├── service
│   │   │       └── FLashcardsApplication.java
│   │   └── resources
│   │       └── application.properties
│   └── test
│       └── java
│           └── com.project.flashcards
├── .mvn
├── target
└── pom.xml
```

Obrázek 4.1. Struktura serverové časti

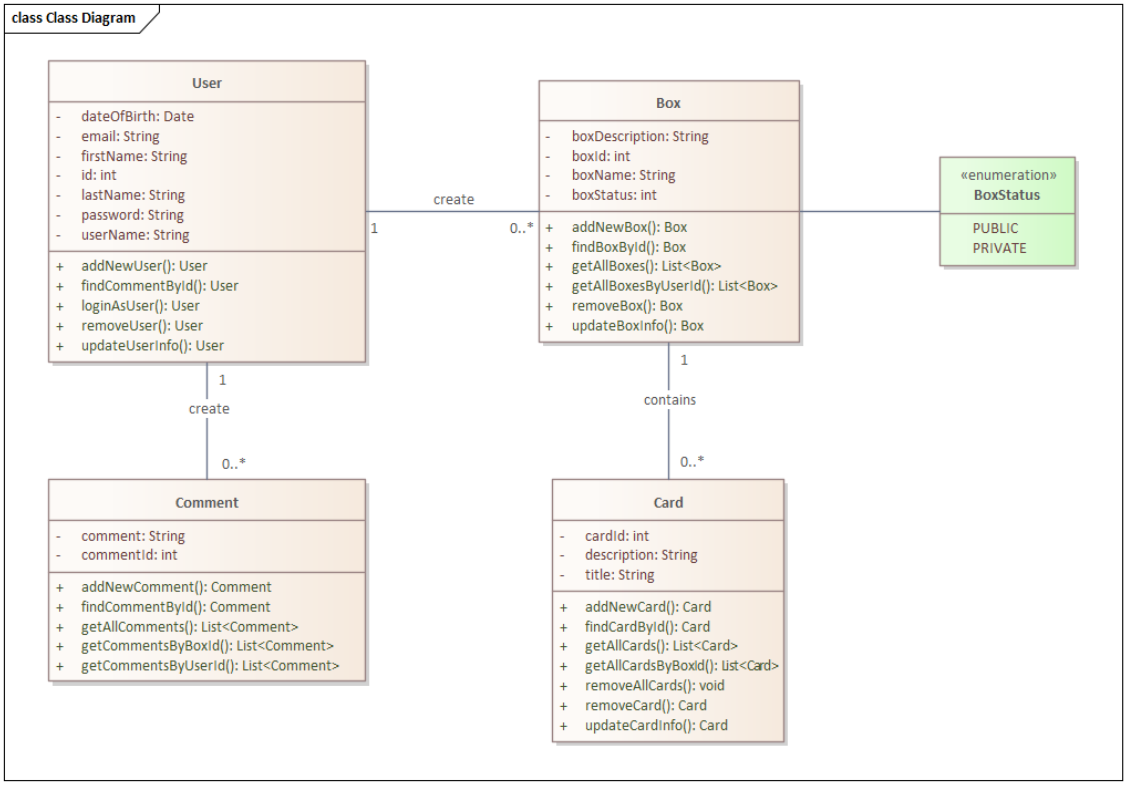
4.3.1 Class diagram

Diagram tříd je statický diagram. Představuje statický pohled na aplikaci. Diagram tříd se nepoužívá pouze pro vizualizaci, popis a dokumentaci různých aspektů systému, ale také pro konstrukci spustitelného kódu softwarové aplikace.

Diagram tříd popisuje atributy a operace třídy a také omezení kladená na systém. Diagramy tříd jsou široce používány při modelování objektově orientovaných systémů, protože jsou jedinými diagramy UML, které lze přímo mapovat pomocí objektově orientovaných jazyků.

Účel diagramů tříd:

- Zobrazuje statickou strukturu klasifikátorů v systému.
- Diagram poskytuje základní zápis pro další vývojové diagramy předepsané UML.
- Užitečné pro vývojáře a další členy týmu.
- Obchodní analytici mohou používat diagramy tříd k modelování systémů z obchodní perspektivy.



Obrázek 4.2. Class diagram

4.4 Klientská Část

Klientská část implementuje uživatelské rozhraní, klade požadavky na server a zpracovává z něj odpovědi. Na základě analýzy v předchozí kapitole byla pro implementaci na straně klienta vybrána javascriptová knihovna React.

```
flipcard-react
├── node_modules
├── public
│   └── index.html
└── src
    ├── assets
    │   ├── css
    │   ├── fonts
    │   ├── img
    │   └── js
    ├── components
    │   ├── boxes
    │   ├── contact
    │   ├── footer
    │   ├── home
    │   ├── navbar
    │   ├── public
    │   └── routes
    ├── const
    ├── App.jsx
    └── index.js
package.json
package=lock.json
README.md
```

Obrázek 4.3. Struktura klientské časti

4.4.1 Návrh UI

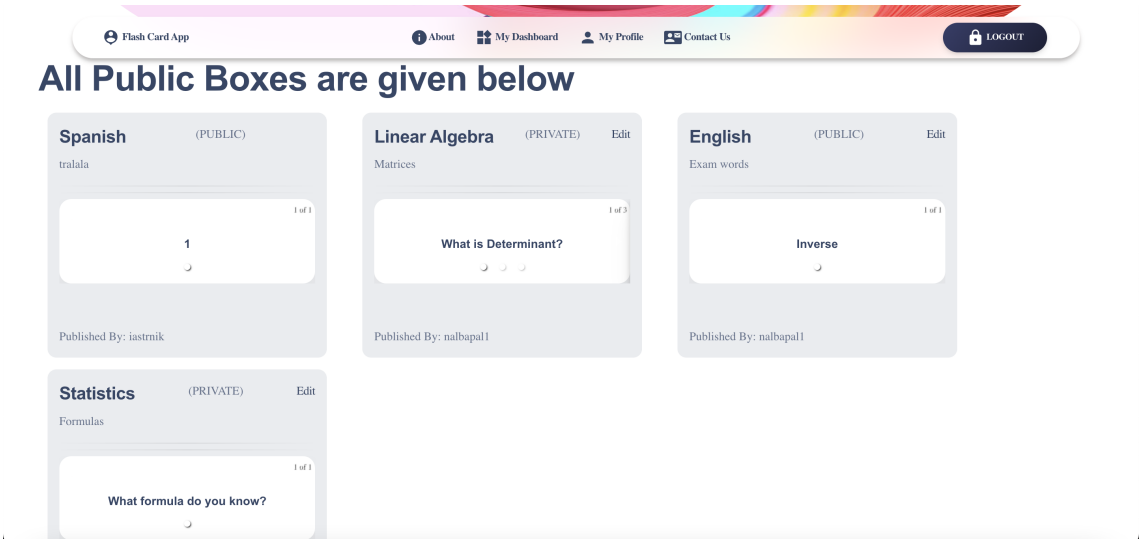
Tato část představuje návrh aplikace.

Jako základ pro to byl použit low-fidelity prototyp [22], který zobrazuje hlavní případy užití našeho projektu, což byla sada wireframů vizualizujících budoucí aplikaci a poskytující jasnou představu o její struktuře.

Slouží pouze k vizualizaci vzhledu a funkčnosti webové aplikaci a usnadňují pochopení jeho podstaty, což stačilo k realizaci finálního návrhu. Příklady low-fidelity prototypu najdete v přílohách práce - název souboru *wireframes-flashcards.pdf*. Pak na základě low-fidelity prototypu byl navržen finální návrh aplikace:

- Hlavní stránka

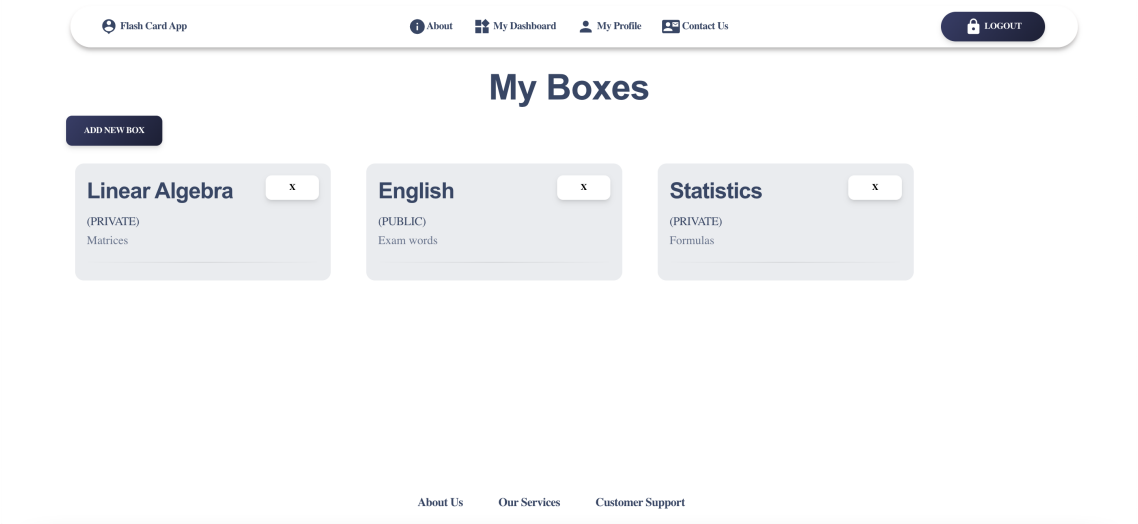
  Na hlavní stránce aplikace je seznam všech boxů uživatele i všech veřejných boxů.



Obrázek 4.4. Main Page

- Card Box

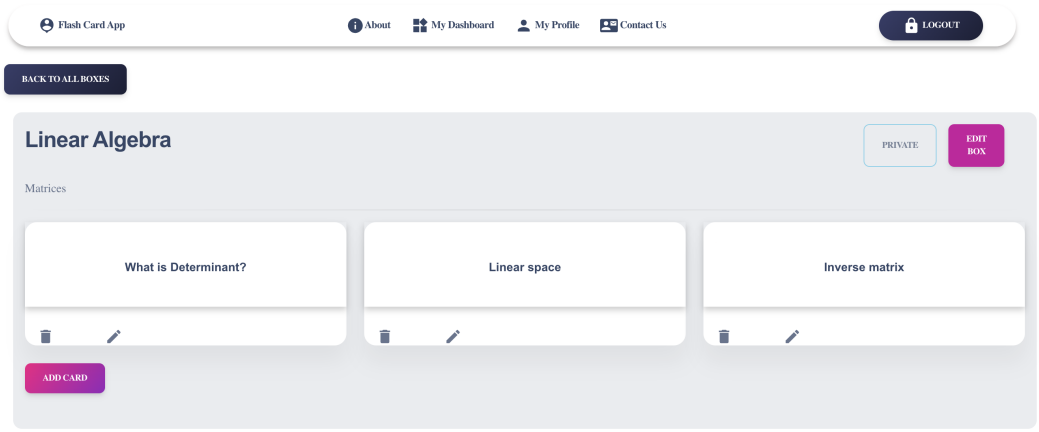
  Stránka Card Box představuje seznam všech veřejných a soukromých boxů. Pomocí tlačítka "X" může uživatel box smazat.



Obrázek 4.5. Card Box Page

- Cards

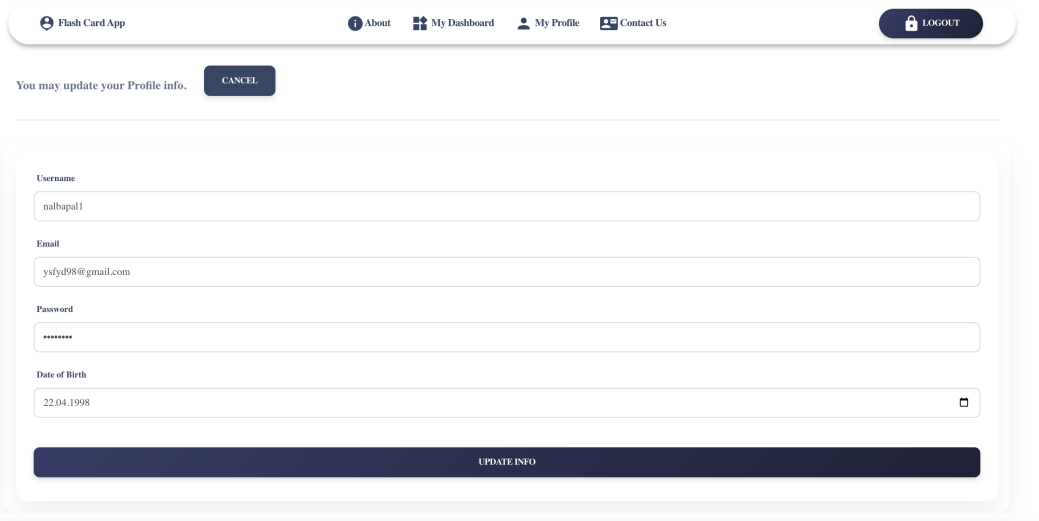
  Na stránce Cards se zobrazí všechny karty ve vybraném boxu. Na této stránce může uživatel upravit box, stejně jako upravit data kartiček nebo je smazat.



Obrázek 4.6. Cards

- Profile

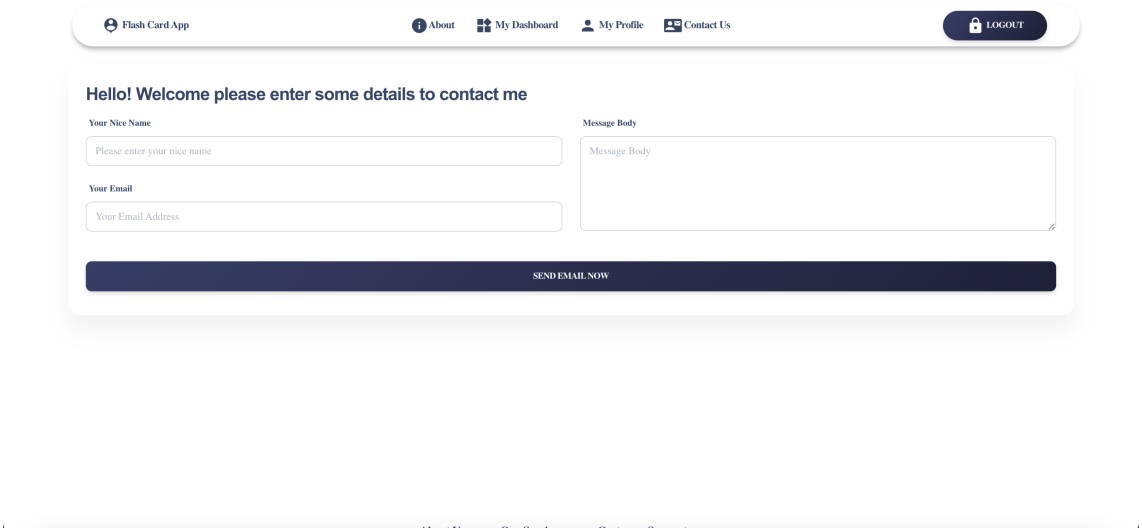
Na stránce profilu může uživatel aktualizovat své údaje – změnit username, email, heslo nebo své datum narození.



Obrázek 4.7. Profile Page

- Odeslání zpětné vázby

Stránka zpětné vazby od uživatele žádá, aby poslal e-mail tvůrci aplikace, a to pro sběr zpětné vazby a budoucímu vylepšení webové aplikace.



Obrázek 4.8. Email Sending Page

4.5 Shrnutí Kapitoly

V rámci této kapitoly byla poskytnuta řešení pro návrh a implementaci webové aplikace. Pro vizuální znázornění práce aplikace byly předloženy některé diagramy, které zjednodušují pochopení a budoucí implementaci projektu.

V další kapitole bude představena samotná realizace projektu.

Kapitola 5
Implementace

V této kapitole popíšu technickou specifikaci tohoto projektu, jaké technologie budou použity k implementaci webové aplikace, jak spolu souvisejí a jaký mají význam.

5.1 Vývojové prostředí

IDE neboli integrované vývojové prostředí spojuje všechny nástroje, které potřebujete k psaní, ladění a testování kódu. IDE umožňuje vývojářům pracovat v jediném prostředí a zlepšovat pracovní postup při programování. Každé IDE má různé zdroje, ale všechny zahrnují textový editor, nástroje pro sestavení a ladicí program. [23]

Při výběru Java IDE je důležité nejprve určit, co od něj potřebujeme, abyste se ujistili, že má všechny správné nástroje pro naše potřeby.

- **Intellij IDEA**

  IntelliJ je jedno z nejlepších IDE pro vývoj v Javě. Jedná se o funkční a ergonomické IDE s funkcemi pro zvýšení produktivity, aniž by zahlcovalo uživatelské rozhraní. Zahrnuje sadu nástrojů, které usnadňují programování, jako je inteligentní dokončování, refactoring. [24]

  Klady:
  - Silné možnosti nastavení.
  - Poskytuje podporu pro programovací jazyky založené na JVM, jako je Kotlin.
  - Podporuje různé programovací jazyky.
  - Vestavěná podpora správy verzí.
  - Silná podpora pluginů a integrací.
  - Výkonný kompilátor

  Použitá verze: 2020.3.2

- **VSCode**

  IDE společnosti Microsoft, Visual Studio je k dispozici pouze pro operační systémy Windows a macOS. Podporuje Python, PHP, JavaScript, HTML, CSS a mnoho dalších jazyků. [25]. Visual Studio má všechny výhody IDE, včetně vzdáleného ladění.

  Kromě toho platforma obsahuje:
  - Přidání inteligentního kódu IntelliSense pro urychlení procesu psaní programů.
  - Nástroje pro spolupráci: řízení přístupu a přizpůsobitelné možnosti editoru vám umožní psát kód v jediném stylu.
  - Integrace s Git.
  - Snadné nasazení díky integrované integraci s Azure.

Použita verze: 1.67.1

- **PgAdmin4**

  PgAdmin4 je open source aplikace pro správu a vývoj databáze PostgreSQL, která umožňuje provádět mnoho úkolů od monitorování a údržby až po provádění SQL dotazů. [26]

  Použitá verze: 3.5

5.2 Backend

Tato kapitola popisuje strukturu serverové části aplikace, vysvětluje význam jednotlivých částí a to, jak spolu komunikují.

5.2.1 Struktura projektu

- **FlashcardsAppliccation.java**

  Tato třída má hlavní metodu, která je vstupním bodem do aplikace. Anotace *@SpringBootApplication* určuje, že aplikace poběží jako aplikace Spring Boot. Také říká Springu, že pro vše, co máme v souboru závislostí *pom.xml*, musíme zahrnout automatickou konfiguraci.

- **Model**

  Balíček *Model* zahrnuje v sobě veškeré entity, které jsou používány v projektu. Každá entita obvykle představuje tabulku v relační databázi a každá instance entity odpovídá řádku v této tabulce. Model funguje jako kontejner, který obsahuje data aplikace. Zde mohou být data v jakékoli formě, jako jsou objekty, řetězce, informace z databáze atd.

  Anotace *@Entity* určuje, že třída je entita a je mapována na databázovou tabulku.

  Anotace *@Table* určuje název databázové tabulky, která má být použita pro mapování. Na obrázku 5.1 je vidět příklad entity v projektu Flash cards a to, jak se mapuje na databázi pomocí anotace *@Table*.

```java
@AllArgsConstructor
@NoArgsConstructor
@Data
@ToString
@Entity
@Table(name="box")
public class Box {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int boxId;
    private String boxName;
    private String boxDescription;
    private BoxStatus boxStatus;
    @Transient
    private String username;
    @Transient
    private int userId;
    @ManyToOne(cascade = CascadeType.ALL)
    @JsonIgnore
    @JoinColumn(name = "fk_user")
    private User user;

    @OneToMany(mappedBy = "box", cascade = CascadeType.ALL)
    private List<Card> cards;
```

Obrázek 5.1. Realizace entity “Box”.

Repository

Spring boot framework nám poskytuje úložiště, které je odpovědné za provádění různých operací s objektem. Abychom vytvořili jakýkoli repozitář tříd v spring boot, musíme použít anotaci *@Repository* na této třídě, tuto anotaci poskytuje samotný spring framework.

- BoxRepository.java
- CardRepository.java
- CommentRepository.java
- UserRepository.java

Service

V aplikaci je business logika umístěna ve vrstvě *service*, takže k označení, že třída patří do této vrstvy, používáme anotaci *@Service*.

Controller

Ve Spring Boot je třída Controller zodpovědná za zpracování příchozích požadavků REST API, přípravu modelu a vrácení pohledu k vykreslení jako odpověď.

Na obrázku 5.2 na příkladu třídy *CardController.java* je vidět, jak jsou realizovány controllery v projektu "Flash cards".

Třídy Controller v Spring jsou anotovány anotací *@Controller* nebo *@RestController*. @RestController v Spring je v podstatě jen kombinace *@Controller* a *@ResponseBody*.

Tato anotace byla přidána do Spring 4.0, aby se odstranila redundance deklarace *@ResponseBody* ve vašem ovladači.

```
@RestController
@RequestMapping("/card")
public class CardController {

    @Autowired
    private CardService cardService;

    /**
     * Returns all cards
     * @return cards
     */
    @GetMapping("getAll")
    public ResponseEntity<?> findAll() { return ResponseEntity.ok(cardService.getAllCards()); }

    /**
     * Gets all cards by Box Id
     * @param id
     * @return
     * @throws NotFoundException
     * @throws EmptyException
     */
    @GetMapping("getByBoxId/{id}")
    public ResponseEntity<?> findByBoxId(@PathVariable( name= "id") int id) throws NotFoundException, EmptyException {
        return ResponseEntity.ok(cardService.getAllCardsByBoxId(id));
    }
```

Obrázek 5.2. Realizace controlleru "CardController"

Enum

Java enum je sada pojmenovaných konstant, která pomáhá při definování vašich vlastních datových typů, v našem případě to jsou stavy Boxu – *Public* nebo *Private*. V balíčku *Enum* jsou představeny Enum třídy jako BoxStatus.

```
public enum BoxStatus {
    PUBLIC,
    PRIVATE
}
```

Obrázek 5.3. Realizace třídy typu Enum “BoxStatus”

- **Exception**

  Balíček exception obsahuje všechny zachycené výjimky, které program zpracovává.
  - EmptyException.java
  - InvalidCredentials.java
  - NotFoundException.java
  - DuplicateException.java

5.3 Frontend

Front-end aplikace bude implementována pomocí javascriptové knihovny React. Celé uživatelské rozhraní bude psáno v React.js pomocí CSS frameworku Bootstrap.

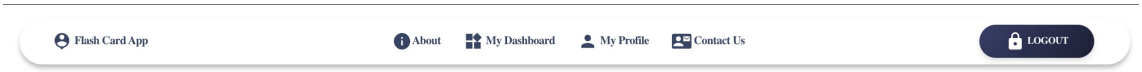
Frameworky jako Bootstrap abstraktní vytvářejí komponenty do bodu, kdy nutí vývojáře používat pouze dostupné šablony. Totéž platí i pro jiné frameworky uživatelského rozhraní.

V rámci front-end implementace aplikace byly definovány následující komponenty: Navbar, Footer, Header, Boxes, Login, Register, Comment, Home.

K vytváření komponent používáme deklarativní kód, což je způsob, jakým zobrazujeme informace. Komponenty jsou v podstatě opakovaně použitelná uživatelská rozhraní, která umožňují rozdělit aplikaci do samostatných bloků, které jednají nezávisle na sobě. Komponenty přijímají libovolný vstup s daty (prop) a vracejí prvek React, aby deklarovaly, co se má objevit na obrazovce. Mohou interagovat s dalšími komponentami prostřednictvím rekvizit a vytvořit tak komplexní uživatelské rozhraní.

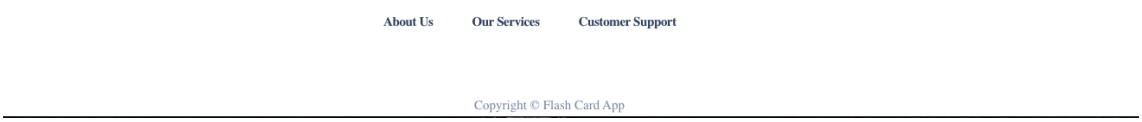
- Navbar

  Tato komponenta implementuje navigační panel aplikace – část, která se nachází v horní části stránky a zůstává nezměněna.



Obrázek 5.4. Navigační panel

- Footer

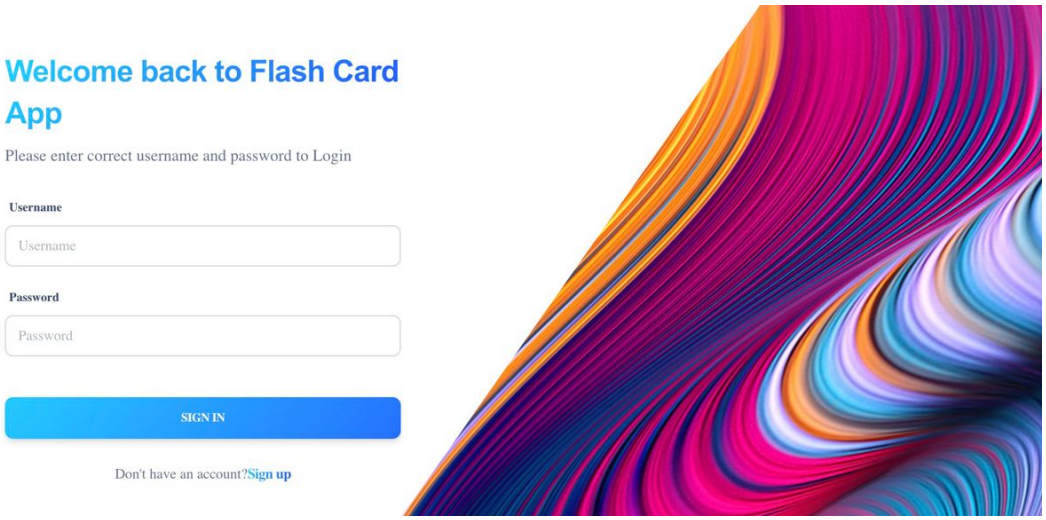


Obrázek 5.5. Footer

- Boxes

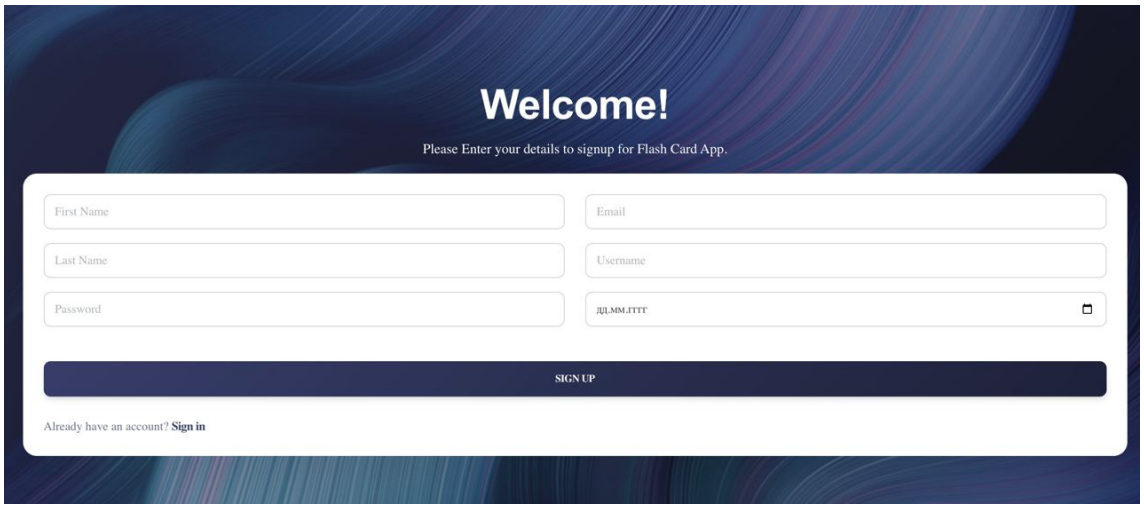
Obrázek 5.6. Boxes

- Login



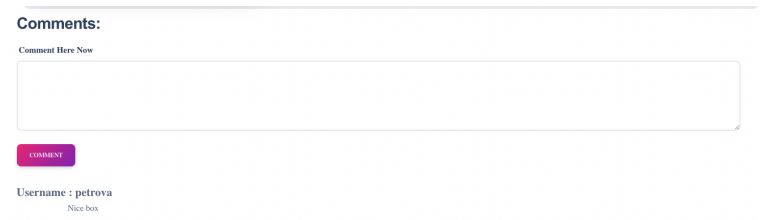
Obrázek 5.7. SignIn formulář

- Register



Obrázek 5.8. Registrační formulář

- Comment

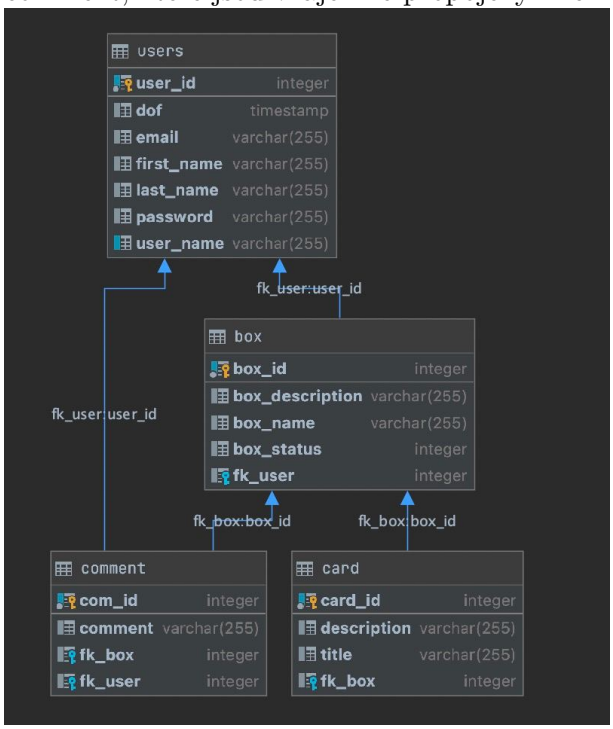


Obrázek 5.9. Comment

5.4 Databáze

Jak již bylo zmíněno, aplikace využívá PostgreSql databázi. Ke spojení serverové části s backendem dochází v souboru *application.properties*. V této fázi projektu se používá databáze na lokálním serveru. Pro správu databáze byl použit pgAdmin4.

Obrázek 5.5 ukazuje schéma databáze projektu. Projekt se skládá ze 4 tabulek: users, box, card, comment, které jsou vzájemně propojeny klíči.



Obrázek 5.10. Databáze projektu

5.5 Shrnutí kapitoly

V rámci kapitoly “Implementace” byly uvedeny všechny fáze realizace projektu.

Bylo představeno databázové schéma, které sloužilo jako základ pro tvorbu budoucí aplikace. Byla představena struktura serverové části aplikace, byly popsány její části, a jak spolu komunikují. Poté byla představena implementace front-endové části aplikace, byly uvedeny hlavní komponenty, které sloužily jako základ projektu.

V další kapitole bude popsáno, jak byla aplikace testována jak ve fázi vývoje, tak ve fázi user acceptance.

Kapitola 6
Testování

Tato kapitola popisuje, jak byl projekt testován. Budou popsány provedené unit testy, představeny hlavní testovací případy, které sloužily jako základ pro ruční testování aplikace. Na konci kapitoly budou shrnuty výsledky testování a vyvozeny odpovídající závěry.

6.1 Unit testování

Unit testing je proces vývoje softwaru, ve kterém jsou nejmenší testovatelné části aplikace nazývané *units* individuálně a nezávisle kontrolovány, zda správně fungují. Tuto metodologii testování provádějí během procesu vývoje vývojáři softwaru a někdy zaměstnanci QA.

Hlavním cílem testování jednotek je izolovat psaný kód k testování a určit, zda funguje tak, jak bylo zamýšleno. Unit testování je důležitým krokem v procesu vývoje, protože pokud je provedeno správně, může pomoci k odhalení raných chyb v kódu, jež může být v pozdějších fázích testování obtížnější.

6.2 Manuální testování

Ruční testování je typ testování softwaru, ve kterém jsou testovací případy prováděny ručně testerem bez použití jakýchkoli automatizovaných nástrojů. Účelem ručního testování je identifikovat chyby, problémy a závady v softwarové aplikaci.

Manuální testování softwaru je nejprimitivnější technika ze všech typů testování a pomáhá najít kritické chyby softwarové aplikace[27]. Každá nová aplikace musí být ručně otestována, než bude možné její testování automatizovat. Manuální testování softwaru vyžaduje více úsilí, ale je nezbytné pro kontrolu proveditelnosti automatizace.

Klíčovým konceptem manuálního testování je zajistit, aby aplikace byla bez chyb a fungovala v souladu se specifikovanými funkčními požadavky. Test case nebo případy jsou navrženy během testovací fáze a měly by mít 100% pokrytí. Zajišťuje také, že nahlášené bugy jsou opraveny vývojáři a testeři opravené bugy znovu testují. Toto testování v podstatě kontroluje kvalitu systému a dodává zákazníkovi produkt bez chyb[28].

6.2.1 Testovácí analýza

- Sběr požadavků:
  Požadavky na projekt, které budou sloužit jako základ pro manuální testování, jsou podrobně popsány v kapitole Sběr požadavků 3.1
- Nastavení testovacího prostředí:
  Program bude testován na lokálním serveru pomocí lokální databáze. K testování poslouží webové prohlížeče Mozilla a Chrome. Výsledky testu budou zaneseny do tabulky v kapitole Výsledky testů.
- Vytváření testovacích případů
- Provedení testu a hlášení bugů

6.2.2 Testovací scénáře

V rámci analýzy testu byly vytvořeny následující testovací případy:

- TC01 – Založit účet
  - Otevřít hlavní stránku projektu localhost:3000.
  - Stisknout tlačítko “Sign Up”.
  - Vyplnit formulář.
  - Stisknout tlačítko “Sign Up”.
  - Uživatel přesměrován do přihlašovací stránky.

- TC02 – Přihlášení/Odhlášení
  - Otevřít hlavní stránku projektu localhost:3000.
  - Uvést údaje pro přihlášení.
  - Stisknout tlačítko “Sign In” → Uživatel je přesměrován na hlavní stránku uživatelského portálu.
  - Stisknout “LogOut” → Uživatel je odhlášen z účtu. Zobrazí se přihlašovací stránka.

- TC03 – Vytvořit box s kartičkami

  Precondition: Uživatel je přihlášený
  - Přejít na stránku “My Dashboard”.
  - Stisknout tlačítko “Add New Box”.
  - Vyplnit formulář.
  - Stisknout tlačítko “Save” a zavřít formulář.
  - Zvolit card box.
  - Stisknout tlačítko “Add Card”.
  - Vyplnit, uložit a zavřít formulář.
  - Otočit kartičku jejím stisknutím.

- TC04– Upravit box a kartičku

  Precondition: Uživatel je přihlášený. Uživatel má vytvořeny card box a kartičku.
  - Přejít na “My dashboard”.
  - Přejít na stránku libovolného boxu.
  - Stisknout “Edit Box” → Změnit údaje → Uložit → Zavřít
  - Údaje boxu jsou změněny.
  - Stisknout ikonku propisky v dolní části kartičky → Změnit údaje → Uložit → Zavřít.

- TC05– Smazat box a kartičku

  Precondition: Uživatel je přihlášený. Uživatel má vytvořeny card box a kartičku.
  - Přejít na “My dashboard”.
  - Přejít na stránku libovolného boxu.
  - V dolní části kartičky stisknout ikonku košík.
  - Kartička se smazala – nezobrazuje se v uživatelském portálu.
  - Přejít na “My Dashboard” → Stisknout tlačítko “X” v horní části boxu → Card box je smazán a nezobrazuje se v uživatelském portále.

6.2.3 Výsledky testování

Na základě manuálního testování v prohlížečích “Google Chrome” a “Mozilla Firefox” byly učiněny následující závěry:

Test Case ID	Result
TC01	PASS
TC02	PASS
TC03	PASS
TC04	PASS
TC05	PASS
TC06	PASS

Tabulka 6.1. Výsledky manuálního testování

Výsledky testů naznačují, že hlavní funkce, které byly identifikovány v rámci analýzy, byly implementovány a fungují bez chyb.

Kapitola 7
Závěr

V kapitole "Závěr" budou shrnuty výsledky provedené práce a budou identifikována možná zlepšení a změny.

7.1 Zhodnocení

Účelem této práce bylo analyzovat, navrhnout, implementovat a otestovat webovou aplikaci "Flash Cards", která má studentům pomoci při osvojování a učení se nové látky.

V rámci práce byla provedena analýza existujících webových aplikací, které řeší stejný problém jako aplikace "Flash Cards". Na základě této analýzy a také na základě komunikace s potenciálními uživateli aplikace byly identifikovány a formulovány hlavní požadavky na projekt.

Ty sloužily jako základ pro vytvoření datové struktury projektu a na základě těchto požadavků a některých schémat byl vytvořen návrh budoucího projektu, který sloužil jako základ pro budoucí implementaci serverové a klientské části.

Po realizaci projektu byla ukončena testovací fáze, v jejímž rámci byla provedena testovací analýza, vytvořeny byly hlavní testovací scénáře a byl učiněn závěr.

Úkol považuji za splněný. Projekt byl realizován od začátku do konce, byly splněny všechny požadavky definované na začátku, projekt byl otestován a připraven k použití.

7.2 Možnosti dalšího vývoje

V této kapitole bych chtěla identifikovat případné možnosti rozšíření stávající aplikace, přidání funkcionality a provedení úprav, které nebyly původně implementovány z důvodu časové tísně.

- FRQ101 – Systém bude umožňovat uživateli smazat svůj profil.
- FRQ102 – Systém bude umožňovat uživateli vyhledat si libovolný card box, který je v public stavu podle kategorií z FRQ108.
- FRQ103 – Systém bude umožňovat uživateli přidat card box do oblíbených.
- FRQ104 – Systém bude umožňovat uživateli přidávat do kartičky obrázky, tabulky atd.
- FRQ105 – Systém bude umožňovat uživateli spouštět režim hry, v jehož rámcích budou kartičky promíchány.
- FRQ106 – Systém bude umožňovat uživateli nechat pozitivní, nebo negativní výsledek z kartičky.
- FRQ107 – Systém bude na základě výsledků z FRQ106 vytvářet statistiku card boxu.
- FRQ108 – Systém bude umožňovat uživateli zařadit card box do kategorií.

Literatura

[1] Rasmusson D. X. *Effects of three types of memory training in normal elderly.* 1999.
https://www.tandfonline.com/doi/abs/10.1076/anec.6.1.56.790.

[2] Malan R. *Functional requirements and use cases.* 2001.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.436.4773&rep=rep1&type=pdf.

[3] Chung L. *Non-functional requirements in software engineering.* 2012..
https://edisciplinas.usp.br/pluginfile.php/5281085/mod_resource/content/1/On_Non-Functional_Requirements_in_Software_Enginee.pdf.

[4] Miller G. Armour F. *Advanced use case modeling: software systems. – Pearson Education.* 2000.
https://edisciplinas.usp.br/pluginfile.php/5281085/mod_resource/content/1/On_Non-Functional_Requirements_in_Software_Enginee.pdf.

[5] Widom J. Garcia-Molina H., Ullman J. D. *Database system implementation. – Upper Saddle River : Prentice Hall.* 2000.
https://www.csd.uoc.gr/~hy460/pdf/000.pdf.

[6] Harrington J. L. *Relational database design and implementation.* 2016.
http://thuvien.thanglong.edu.vn:8080/dspace/bitstream/TLU-123456789/2788/1/TVS.001800_NV.0006182_Jan.

[7] Kanellakis P. C. *Elements of relational database theory. Formal models and semantics.* 1990.
ftp://128.148.32.111/pub/techreports/89/cs89-39.pdf.

[8] Rawat B. *MySQL Database Management System (DBMS) On FTP Site LAPAN Bandung.* 2021.
https://iiast-journal.org/ijcitsm/index.php/IJCITSM/article/view/47.

[9] Conrad A. *Database of the year: Postgres.* 2021.
https://ieeexplore.ieee.org/iel7/52/9520220/09520330.pdf.

[10] Agapin L. I. Boicea A., Radulescu F. *MongoDB vs Oracle–database comparison.* 2012.
https://productzense.com/wp-content/uploads/2017/11/Mongo-DB-vs-Oracle.pdf.

[11] Thomson L. Welling L. *PHP and MySQL Web development.* 2003.
https://www.acs.ase.ro/Media/Default/documents/java/ClaudiuVinte/books/ArnoldGoslingHolmes06.pdf.

[12] Drake Jr F. L. Van Rossum G. *Python tutorial.* 1995.
https://www.academia.edu/download/62442353/tutorial20200322-66446-ykbseo.pdf.

[13] Effah J. Ofoeda J., Boateng R. *Application programming interface (API) research: A review of the past to inform the future*. 2019.
https://dl.acm.org/doi/abs/10.4018/IJEIS.2019070105.

[14] Surwase V. *REST API modeling languages-a developer's perspective*. 2016.
https://www.academia.edu/download/47318910/IJSTEV2I10199.pdf.

[15] Belqasmi F. *SOAP-based vs. RESTful web services: a case study for multimedia conferencing*. 2012.
https://spectrum.library.concordia.ca/id/eprint/980040/1/PersonalCopy-SOAPvsREST.pdf.

[16] Chengalur-Smith I. S. Duchessi P. *Client/server benefits, problems, best practices*. 1998.
https://dl.acm.org/doi/pdf/10.1145/274946.274961.

[17] Fedosejev A. *React. js essentials*. 2015.
https://strukturnifondovi.hr/wp-content/uploads/2017/03/9781783551620-REACTJS_ESSENTIALS.pdf.

[18] Singh R. K. Kumar A. *Comparative analysis of angularjs and reactjs*. 2016.
https://www.academia.edu/download/54960538/148051944230.1245.pdf.

[19] De Lauretis L. *From monolithic architecture to microservices architecture*. 2019.
https://ieeexplore.ieee.org/abstract/document/8990350/.

[20] Chitnis K. Mane D., Ojha N. *The spring framework: An open source java platform for developing robust java applications*. 2013.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.683.8397&rep=rep1&type=pdf.

[21] Goodwill J. Vukotic A. *Apache tomcat 7*. 2011.
https://www.doc-developpement-durable.org/file/Projets-informatiques/cours-&-manuels-informatiques/Apache/Apache%20Tomcat%207.pdf.

[22] Giller V. Sefelin R., Tscheligi M. *Paper prototyping-what is it good for? A comparison of paper-and computer-based low-fidelity prototyping*. 2003.
https://www.sunyoungkim.org/class/old/hci_sp18/readings/lofi_hifi3.pdf.

[23] Beller M. *Developer testing in the ide: Patterns, beliefs, and behavior*. 2017.
https://pure.tudelft.nl/ws/files/38319277/TSE2776152.pdf.

[24] Böck H. *IntelliJ IDEA and the NetBeans Platform. The Definitive Guide to NetBeans™ Platform 7*. 2012.
https://link.springer.com/chapter/10.1007/978-1-4302-4102-7_40.

[25] Habib K. Khan O. M. A. *Developing Multi-Platform Apps with Visual Studio Code: Get up and running with VS Code by building multi-platform, cloud-native, and microservices-based apps*. 2020.
https://books.google.com/books?hl=ru&lr=&id=vkD-DwAAQBAJ&oi=fnd&pg=PP1&dq=VScode+javascript&ots=GINqrbO-Lt&sig=vunsYNQYDX-3mMyg5NWxQAJcZJw.

[26] Robinson C. *Basic introduction into pgAdmin III and SQL queries*. 2011.
https://www.researchgate.net/profile/Christina-Robinson-7/publication/265745588_Basic_introduction_into_pgAdmin_III_and_SQL_queries/links/5980e01b4585150575ba1846/Basic-introduction-into-pgAdmin-III-and-SQL-queries.pdf.

[27] Olan M. *Unit testing: test early, test often*. 2003. http://www.sast.se/q-moten/2007/stockholm/q1/2007_q1_runeson_artikel.pdf.

[28] Lassenius C. Itkonen J., Mantyla M. V. *How do testers do it? An exploratory study on manual testing practices*. 2009. http://lib.tkk.fi/Diss/2011/isbn9789526043395/article4.pdf.

Příloha A
Seznám zkratek

Tato příloha představí všechny zkratky použité v práci.

FRQ	Functional requirements
NFRQ	Non-Functional requirements
DB	Database
Sql	Structured Query Language
NoSql	Non-Structured Query Language
DBMS	Database Management System
CMS	Content Management System
OLAP	Online Analytical Processing
JSON	JavaScript Object Notation
Php	Hypertext Preprocessor
API	Application Programming Interface
REST	Representational state transfer
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
JSX	JavaScript XML
DOM	Document Object Model
FP	Functional Programming
MVVM	Model - View - ViewModel
UML	Unified Modeling Language
PDF	Portable Document Format
IDE	Integrated Development Environment
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JPA	Java Persistence API
UAT	User Acceptance Testing