Univerzita
Hradec Králové
Fakulta informatiky
a managementu

Bakalářská práce

Optimalizace kalorických příjmů a výdejů

Tereza Kvášová

Katedra informačních technologií
Vedoucí práce: Ing. Pavel Kříž, Ph.D.

30. dubna 2020

Poděkování

Poděkování za vedení práce patří Ing. Pavelu Křížovi, Ph.D. Za externí vedení práce a odbornou konzultaci bych chtěla poděkovat Ing. Vladimíru Blažkovi. Dále bych chtěla poděkovat hráčům kádru FC Hradec Králové a.s. za vstřícnou konzultaci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Hradci Králové dne 30. dubna 2020 .....

Univerzita Hradec Králové
Fakulta informatiky a managmentu

Odkaz na tuto práci

Kvášová, Tereza. *Optimalizace kalorických příjmů a výdejů.* Bakalářská práce. Hradec Králové: Univerzita Hradec Králové, Fakulta informatiky a managmentu, 2020.

Abstrakt

Cílem práce je navrhnout a implementovat aplikaci s kalorickými tabulkami a s energetickými výdaji pohybových aktivit. Cílem aplikace je poskytnutí uživateli informace o pohybové aktivitě, kterou musí vynaložit pro spálení kalorické hodnoty zkonzumované potraviny. Aplikace usnadní uživateli výpočet množství pohybu, navrhne mu vhodnou aktivitu a náročnost dané aktivity.

Klíčová slova: kalorie, kalorická tabulka, spalování kalorií, aplikace na hubnutí, zdravý životní styl

Abstract

The goal of this work is to design and implement application with caloric table and energetic expenditure of motion activities. The goal of application is to provide informations to the user about motion activity which he must expend to burn caloric value of consumed food. The application makes it easier for the user to calculate amount of movement, suggest him right sport activity and difficulty of concrete activity.

Keywords: calorie, caloric table, burn calories, weight loss application, healthy lifestyle

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Počítání kalorií se stalo fenoménem „moderní doby“. Lidé mají čím dál větší zájem o celkově zdravější životosprávu, kterou se snaží docílit nejen samotným počítáním kalorií, ale i zařazováním pravidelné pohybové aktivity do svého běžného režimu.

Součástí této doby je nutnost neustále někam spěchat, za něčím se hnát. To ovlivňuje životní styl nás všech. Místo procházky do práce či do školy raději zvolíme pohodlnější a rychlejší variantu. K přepravě použijeme auto nebo městskou hromadnou dopravu. Co se týká výživy, je pro nás jednodušší si jídlo koupit nebo objednat, protože jeho příprava zabere mnoho času. Návštěva fast foodu nám tak ušetří pár minut navíc.

Dříve nebylo potřeba stravu řešit. Jedním z důvodů bylo to, že lidé neměli možnost pestré stravy a tudíž ani přístup k tolika nezdravým potravinám. Většina lidí si také nemohla dovolit vlastnit dopravní prostředek, museli se dopravovat pěšky, případně na kole, z čehož plynul dostatek pohybu pro populaci. To se však dnešním způsobem života a technickými vymoženostmi, které máme dnes k dispozici, razantně změnilo.

Sedavé zaměstnání, nedostatek pohybu, špatná životospráva a máme tu problém. Všechna tato fakta vedou k tomu, že na světě stoupá výskyt obezity. To si ale dnes začíná mnoho lidí uvědomovat a není už nic neobvyklého navštěvovat výživového poradce, počítat si kalorie podle stanoveného denního příjmu nebo řešit příjem výživových hodnot v potravě.

Trend hlídaní si svého příjmu a výdeje ovlivňuje i vývoj aplikací v oblasti IT. K usnadnění počítání denního příjmu vzniká nepřeberné množství aplikací, které vše jednoduše spočítají za Vás.

Počítáním kalorií každý postupně získává přibližný přehled o tom, kolik a hlavně čeho, může za den sníst. Na trhu můžeme nalézt mnoho aplikací, do kterých se zapisují potraviny přijaté za den, a počítají, kolik kalorií je ještě možné přijmout. Z vlastní zkušenosti mohu říct, že je to přínosné, ale zároveň kvůli neustálému zapisování příjmu a výdeje velmi nepohodlné a únavné.

V důsledku tohoto poznatku vznikl nápad na vytvoření celé této aplikace, tedy uspokojit potřeby člověka, který není ochoten si neustále psát svůj denní příjem. Na rozdíl od ostatních, již existujících aplikací, nabídne tato aplikace lidem trochu jinou hodnotu, a to pohled na potravinu jako na pohybovou aktivitu, která je nutná ke spálení dané potraviny. Myšlenkou projektu je lidem pomoci kalorickou hodnotu lépe chápat, případně jí vůbec neřešit. Uživateli bude zobrazeno, jak náročné je potravinu spálit a podle toho bude moct zvážit vůbec její nákup.

Pro tyto účely vznikne jak webová stránka, tak i mobilní aplikace, kterou může uživatel využívat například v nákupním centru a rovnou zhodnotit, zda potravinu přidá do svého nákupního košíku, nebo je až příliš náročné ji vykompenzovat sportovní aktivitou, a proto si raději vybere nízkokalorickou alternativu.

Kapitola 1

Cíl práce

Cílem práce je navrhnout a implementovat aplikaci, která uživateli poskytne informaci o pohybové aktivitě, kterou je nutné vynaložit pro spálení kalorické hodnoty zkonzumované potraviny. Jedná se o informaci, kterou v mnoha kalorických kalkulačkách běžný uživatel postrádá.

Kapitola 2

Rešerše

2.1 Rešerše dokumentů zabývajících se kalorickými hodnotami

V dnešní době je moderní vysportovaná štíhlá postava a zdravý životní styl. V souvislosti s tímto trendem se mnoho lidí stále častěji uchyluje k různým dietám a omezením v příjmu potravy.

Nepochybnou součástí tohoto trendu se stává měření vlastního denního příjmu kalorií za den. Problémem však je, že mnoho z nás přesně ani neví, co ona sama kalorie je. Často nejsme schopni hodnotě kalorií zapsané u potraviny porozumět, a to i přesto, že najít si složení potravin a jejich kalorickou hodnotu již není v naší době žádný problém.

Tímto tématem se zabývá například článek Consumers May Not Use or Understand Calorie Labeling in Restaurants [4], kde se lze dočíst o tom, že široká veřejnost není v mnoha případech schopna efektivně využít informaci o kalorické hodnotě chodů v restauraci. V souvislosti s tím autor navrhuje, aby byl na potravinách místo přesného počtu kalorií zobrazen popisek málo, středně nebo hodně kalorické.

Tím, jak by mělo být zobrazeno množství kalorií na obalech potravin, se zabývá například i studie Public Health Nutrition [5]. V ní se autoři v souvislosti se vznikající pandemií obezity zabývají formátem informací o produktu. Výzkum na studii proběhl v Německu, Nizozemsku, Francii a Velké Británii. Jeho výsledkem byl celkem zajímavý poznatek, že respondenti jako nejoblíbenější zpracování zobrazení kalorií vybrali počet kalorií na porci nebo na 100 gramů. Složitější zobrazení znaků pro denní potřeby nebo cvičení nebylo v průzkumu preferováno. Ačkoli nebyly vybrány jiné znaky usnadňující chápání kalorie a účastníci s pojmem kalorií byli obeznámeni, nezdálo se, že by plně rozuměli, jak s touto informací naložit.

Možností, jak spotřebiteli přiblížit kalorii, je zobrazovat kalorické hodnoty potraviny v porovnání s celkovým kalorickým příjmem průměrného člověka.

Tato hodnota by lidem usnadnila výpočet denního maximálního množství potraviny, které mohou zkonzumovat, aniž by přijali nízké nebo naopak vysoké množství jedné ze složek v potravě viz [6].

Pokud je kalorická hodnota na obalech jídla znázorněna úrovní pohybové aktivity, kterou je nutné vynaložit pro spálení přijatých kalorií, potom lidé volí raději nízkokalorické svačiny, jak se píše v článku The influence of calorie and physical activity labelling on snack and beverage choices [7].
Oproti žádným informacím nebo jen zapsané kalorické hodnotě na obalu potraviny tak štítky s fyzickou aktivitou výrazně ovlivnily výběr občerstvení. Toto zjištění by mohlo pomoci ve zdravějším výběru výživy v populaci.

Publikace Journal of Physical Education, Recreation & Dance [8] se zabývá mimo jiné i nárustem obezity, která je zapříčiněna převážně špatnou volbou jídelníčku, tj. velkým množstvím přijatých kalorií, které tělo není schopno zpracovat. Při nadbytečném příjmu si tělo ukládá zbytky kalorií do tukových zásob, tím pádem dotyčný přibývá na váze. Logicky, pokud jedinec vydá více energie, než přijme, bude hubnout. U každého je samozřejmě tento jev individuální, ale jako obecné tvrzení je platný. Klíčem k hubnutí je redukovat množství přijímaných potravin a zvýšit kalorický výdej prostřednictvím pohybu. Propojení kalorií a pohybu autoři nazývají „calorie education“
(do češtiny přeloženo jako vzdělání v oblasti kalorií).

2.2 Rešerše existujících řešení

Aktuálně je na českém trhu velmi populární webová a mobilní aplikace **Kalorické tabulky**, která poskytuje uživateli přehled o jeho denních kalorických příjmech. Hodnota jídla je přepočítána na množství a přidána do jídelníčku uživatele. Ten si potom na základě těchto dat může zobrazit procentuální zastoupení jednotlivých složek v potravě, jako je např. množství bílkovin. Dále je v aplikaci možné kontrolovat váhu a zobrazovat si grafy příjmu/výdeje kalorií.

Co se týká hubnutí a zdravého jídelníčku, existuje např. aplikace **Body24**, která doporučuje jídelníček v podobě receptů. Aplikace **NutriAtlas** nabízí informace o potravinách a jejich složení, informuje uživatele o výskytu „éček“ a dalších složkách potravy.

Aplikace **Lose it!** funguje podobně jako zápisník kalorického příjmu potravin, kdy je potravinu možno naskenovat, vyhledat nebo vyfotit a přidat do jídelníčku. Zajímavou aplikací na světovém trhu je aplikace **Noom** která má poskytovat jejím uživatelům tipy a rady na hubnutí a lepší životní styl
na základě posudků od behaviorálních psychologů. Po vygenerování série otázek týkajících se životního stylu je uživateli zobrazen jeho rozbor kalorií.

Do aplikací vypočítávajících pohyb spadá i aplikace **Exercise Calorie**

Calculator, která generuje podle množství aktivity a jejího typu počet spálených kalorií. Dalšími aplikacemi jsou například: **My fitness pal**, **MyNetDiary**, **MyPlate** atd.

Pro zapisování sportovní aktivity existují speciální aplikace jako je např. **Nike Run Club** nebo **Runtactic**.

Většina z těchto aplikací umožňuje uživateli zaznamenávat své příjmy a výdeje kalorií. V těchto aplikacích je často připojená i záložka recepty, kde lze vyhledat zdravé alternativy jídel, které je možné připravit.

Je celkem běžné, že je možné na mobilním zařízení naskenovat kód a vyhledat potravinu, případně ji nalézt podle názvu. Dále jsou často potraviny doplněny o jejich složení a je upozorněno jak na jejich přínosné, tak nezdravé složky, např. éčka. Tyto hodnoty jsou doplněny o grafy.

V průzkumu nebyla nalezena aplikace obsahující informaci, která by usnadnila chápání kalorií. Uživatel se s kaloriemi seznamuje jen pomocí dlouhodobého zaznamenávání potravy do jídelníčku.

Již vytvořené aplikace sice umožňují zaznamenávání i pohybové aktivity, ale jejich výstupem bývá informace o množství shozených kalorií na základě dat o klientovi. V souvislosti s dlouhodobým přehledem o denním příjmu kalorií je možné tato čísla mezi sebou porovnat a pochopit, jak kalorická jednotlivá jídla jsou. Avšak zákazník dostává přehled nad svou potřebou denního příjmu a výdeje až po delší době. Je schopen svou přibližnou představu o množství kalorií v různých jídlech získat až po dlouhodobém zapisování potravin a aktivit do aplikace.

Proto vznikla myšlenka, že pro tyto účely lze využít propojení kalorií s aktivitou. Místo jednoduchého zobrazení kalorií uvidí uživatel nutnou délku trvání nějakého pohybu na spálení konkrétní potraviny. To mu umožní lepší představu o „závažnosti“ kalorické hodnoty jídla, které přijímá.

Aplikace, která touto prací vznikne, tak bude mít na kalorie nový pohled. Bude se dívat na potravinu z pohledu pohybové aktivity, která stojí za jejím spálením.

Vizí aplikace je uživateli zobrazit potravinu tak, aby si dokázal udělat představu, jak velká námaha bude stát za výdejem přijaté energie. Aplikace by tak mohla přinést pozitivní vliv na výběr zdravějších a méně kalorických položek.

Kapitola 3

Analýza a návrh

3.1 Kalorie

Podle studie The Americal Journal of Clinical Nutrition [9] je 1 kalorie definována jako jednotka energie ekvivalentní hodnotě 4,164 J, nebo také jako jednotka energie potřebné ke zvýšení teploty 1 g vody o 1 °C.

Na obalech potravin se setkáváme s jednotkou kilokalorie, značené zkratkou kcal, kterou v běžné řeči v kontextu s potravinami označujeme jako kalorii.

Z hlediska termodynamiky nám 1 g jednotlivých makroživin dává různé počty kalorií: 1 g tuku = 9 kcal, 1 g sacharidů nebo bílkovin = 4 kcal.

3.1.1 Kalorie a pohyb

Při vykonávání pohybu (jakékoli fyzické aktivity) tělo pracuje s energií.
Při namáhavějších aktivitách využije více energie než např. při chůzi.

Jak se v knize fyziologie zmiňuje J. Mourek [10], má každá organická látka přijímaná v naší potravě určitý energetický obsah, který tělo využije při fyzické aktivitě.

V tabulce 3.1 je zaznamenán energetický obsah některých vybraných potravin a současně pak i doba trvání tělesné aktivity jím krytá (chůze, plavání, „turistický“ běh) dle knihy o fyziologii od J. Mourka [10].

3.1.2 Energetický výdej

Pokud není naším účelem hubnout nebo nabírat kilogramy, je za normálních okolností množství energie přijaté v potravě potřeba vyrovnat s množstvím energie, které tělo za den vydá. Tedy dosáhnout určité bilance mezi těmito energiemi.

Je důležité si uvědomit, že tělo potřebuje energii na to, aby vůbec fungovalo samo o sobě. Každý organismus potřebuje přijmout dostatek energie na funkce základních životních orgánů.

Tabulka 3.1: Energetický obsah vybraných potravin

Potravina	Energie (kj)/(kcal)	Chůze (min)	Plavání (min)	Běh (min)
Vepřový plátek	1360 / 327	60	28	16
1 vejce	320 / 77	15	7	4
Pomeranč	290 / 70	13	6	5
Šunka (50 g)	700 / 168	32	15	9
Rybí filé (porce)	340 / 82	15	7	4
$\frac{1}{4}$ kuřete	980 / 235	45	21	12
Chléb s máslem	330 / 79	15	7	4
Pivo (0,5 l)	960 / 231	44	20	12

Pro výpočet energie, kterou náš organismus za den vydá, využijeme součtu energie potřebné na trávení (ta tvoří 10% energie ze snědených potravin za den), energie bazálního metabolismu (energie, kterou potřebuje tělo v klidovém režimu) a energie, kterou jedinec vydá při fyzické zátěži.

V případě, že máme více přijaté energie, budeme na váze přibývat. Pokud za den přijmeme menší množství, než je náš výdej, budeme na váze ubývat.

3.1.3 Bazální metabolismus

Pro zjištění bazálního metabolismu existuje několik vzorců. Pro naše účely si ukážeme tři vybrané vzorce. Nejznámější výpočet pro bazální metabolismus je beze sporu pomocí Harris-Benedictovy rovnice, dalšími rovnice, které si ukážeme jsou Faustova a Cunninghamova rovnice [11].

3.1.3.1 Harris-Benedictova rovnice

Harris-Benedictova rovnice je jednou z nejznámějších metod pro výpočet bazálního metabolismu. Je rozdělena na výpočet zvlášť pro může a zvlášť pro ženy. Výpočet pro muže je popsán vzorcem č. 3.1. Výpočet pro ženy potom vzorcem č. 3.2. Výsledkem je bazální metabolismus v kilokaloriích.

$$BMR = 66,5 + 13,75 * \textit{hmotnost (kg)} + 5,0 * \textit{výška (cm)} - 6,8 * \textit{věk (roky)}. \tag{3.1}$$

$$BMR = 665 + 9,6 * \textit{hmotnost (kg)} + 1,8 * \textit{výška (c)} - 4.7 * \textit{věk (roky)}. \tag{3.2}$$

3.1.3.2 Faustova rovnice

Výpočet pomocí Faustova vzorce není přesný a počítá pouze s váhou zkoumaného jedince.

Pro muže se používá vzorec č. 3.3. Pro ženy potom vzorec č. 3.4. Výsledkem je bazální metabolismus v kilokaloriích.

$$BMR = hmotnost\ (kg) * 24 \tag{3.3}$$

$$BMR = hmotnost\ (kg) * 23 \tag{3.4}$$

3.1.3.3 Cunninghamova rovnice

Výpočet Cunninghamovy rovnice (vzorec č. 3.5) nerozlišuje pohlaví, počítá s FFM a je vhodný pro sportovce. Výsledkem je bazální metabolismus v kilokaloriích.

$$BMR = 500 + 22 * FFM\ (kg) \tag{3.5}$$

V rovnici je $FFM = váha$ – $(procento\ tělesného\ tuku * váha)$

3.1.3.4 Zhodnocení výsledků

Pro porovnání výsledků rovnic si zvolíme fiktivní data pro muže. Váha muže bude 80 kg, výška 190 cm a věk 26 let. Procento tuků je 7%. Po dosazení do Harris-Benedictova vzorce $66,5 + 13,75 * 80 + 5 * 190 - 6,8 * 26$ získáme hodnotu 1939,7 kcal. Výsledek pro Faustovu rovnici $80 * 24$ vychází 1920 kcal a pro Cunninghamovu rovnici $500 + 22 * (80 - (80 * 0,07)$ je výsledek 2136,8 kcal.

Faustova rovnice je založena na hmotnosti, takže nezohledňuje ostatní faktory a je pro volbu výpočtu vhodná pouze v případě, že je známa pouze váha.

Pro fiktivní data byl zvolen sportovec, proto nám nejvyšší výsledek pro bazální metabolismus vyšel pro Cunninghamovu rovnici, kde se počítá s množstvím tuku (to je u sportovce minimální).

Nejpřesnější metodou je Harris-Benedictův vzorec. Zohledňuje více faktorů, které mohou výpočet upřesnit a k jeho výpočtu jsou nutné pouze základní údaje.

Tyto rovnice poslouží k výpočtu průměrného bazálního metabolismu běžného člověka a k vyhodnocené náročnosti aktivity.

3.2 Požadavky klienta

Stěžejním bodem analýzy a návrhu budoucího výrobku je názor zákazníka. Jeho mínění a postřehy tak mohou měnit celý směr vývoje díla. Zákazník bude také do značné míry ovlivňovat vzhled výrobku i jeho funkčnost.

Mezi počáteční požadavky na vývoj produktu spadá důraz na jednoduchost. Cílem odběratele je aplikace maximálně přizpůsobená svému účelu. To znamená, že nebude obsahovat žádné složité operace, aby s ní bylo snadné zacházet a v krátké časové době bude schopna vykonat potřebné úkony. V ideálním případě bude reagovat okamžitě na potřeby klienta.

Jedním z požadavků na aplikaci bude možnost připojení z mobilního telefonu. Z tohoto důvodu je dobré veškerou logiku přesunout na server, který bude dělat potřebné výpočty. Ušetří se tím práce na frontendové části. V opačném případě by bylo potřeba psát tyto výpočty pro mobilní a webovou verzi zvlášť. Pokud bude práce s daty probíhat na backendu, bude je možné využít pro jakékoli zařízení bez nutnosti opakovaného psaní stejného kódu.

Aplikace bude fungovat za přístupu internetu následovně. Spotřebitel si v nákupním centru naskenuje EAN kód potraviny, aplikace kód vyhledá v databázi a zobrazí uživateli základní informace o položce. Zároveň zobrazí i pohybovou aktivitu, kterou je třeba vynaložit pro spálení tohoto výrobku.

Důsledkem potřeby fungování projektu i na mobilním zařízení vznikne aplikace pro webové rozhraní a zároveň pro telefon. Uživatelské rozhraní (UI) je nezbytné navrhnout tak, aby bylo funkční pro požadované platformy. To představuje ve vývoji určité potřeby jako třeba zhotovení návrhů designu pro různá rozšíření obrazovek pro webové i mobilní displeje.

V souvislosti se zhotovením více návrhů pro větší množství rozlišení obrazovek se naskytne příhodné i testování funkčnosti a zobrazení na různých zařízeních. K těmto účelům budou sloužit jak vypůjčená či vlastněná reálná zařízení, tak i virtuální prostředí pro testování rozhraní různých velikostí na desktopu.

Testovacím prostředím je u webového zařízení prohlížeč Chrome s režimem pro simulaci různých zařízení ve vývojářských nástrojích. Tato funkce je nazvána Device Mode. V tomto módu lze měnit různé typy zařízení, tj. rozměry obrazovek nebo hustotu pixelů. Další užitečnou pomůckou je změna orientace zařízení nebo změna velikosti viewportu (oblast pro vykreslování prostoru stránky).

3.3 Volba technologie

Pro jakoukoli aplikaci je zásadním rozhodnutím volba technologie. Za tímto účelem vznikla jako součást této bakalářské práce analýza alternativ vhodných technologií pro vývoj díla. Výhodou předběžného zhodnocení možností pro tvorbu softwaru je například záruka, že nebude potřeba zdrojové kódy

v nejbližší době přepisovat.

Pro zvolení správné technologie je důležité zvážit jak její klady, tak možné zápory užití. Například Python je vhodnou volbou pro výpočty a statistiky, ale nemusí být vhodný pro mobilní aplikace, jelikož má větší nároky na paměť a je pomalý ve srovnání s jinými programovacími jazyky.

Dalším hlediskem, které do jisté míry ovlivňuje výběr, může být i požadavek od zákazníka nebo uživatele, který aplikaci využívá.

Sestavení jednotlivých bodů analýzy bylo zhotoveno na základě rozdělení požadavků dle článku 9 Steps: Choosing a tech stack for your web application [12], ze kterých autorka vybrala pár důležitých bodů k analýze. Ostatní body byly přidány na základě požadavků na aplikaci.

3.3.1 Rychlost, výkon a bezpečnost

Při dostatečném připojení k internetu bude aplikace fungovat svižně.

Po výkonnostní stránce bude veškeré dění přesměrováno na backendovou část, která bude zajišťovat vyhledávání v databázi potravin a vrácení výsledku s informacemi o položce. Součástí dat bude i údaj o množství a náročnosti pohybové aktivity. Frontendová část aplikace nebude výrazně zatížena a bude „pouze“ zobrazovat data uživateli.

Velmi důležitým faktorem je i bezpečnost aplikace. Aplikace disponuje citlivými údaji o uživatelem vyhledávaných potravinách. Pro zajištění bezpečnosti je využit protokol HTTPS.

HTTPS protokol je nadstavbou protokolu HTTP, kdy je využit navíc ještě protokol SSL nebo TLS. Tento protokol se využívá pro komunikaci serveru a webového prohlížeče. TSL a SSL protokol jsou kryptografické protokoly, které tuto komunikaci šifrují a zprostředkovávají autentizaci obou stran.

3.3.2 Open source technologie

Open source je otevřený zdrojový kód, kde spolupracují na vývoji dané technologie vývojáři po celém světe. Produkt je volně dostupný a lze ho bezplatně používat.

Dalšími výhodami open source softwaru je například to, že není potřeba vytvářet svůj vlastní software úplně od začátku. To programátorovi ušetří mnoho času. Vývoj vlastních frameworků by byl velmi zdlouhavý.

Oproti vytváření vlastního softwaru je využití open source prostředků bezpečnější z důvodu, že jsou ověřené a prozkoušené několika dalšími uživateli systému. Důsledkem toho je i fakt, že obsahují méně chyb.

Vady na projektu jsou nalezeny dříve a na jejich odladění spolupracuje zpravidla více programátorů, takže si mohou vzájemně kód opravující příčinu problému zkontrolovat a nevznikne tak spousta dalších chyb v důsledku opravy.

V případě psaní vlastního softwaru je nutné kód udržovat aktuální. To by znamenalo věnovat velké množství času úpravě kódu samotného softwaru pro vývoj.

Důležitým plusem je zajisté i volná dostupnost, přístup ke zdrojovým kódům a dokumentace takových projektů. Z těchto důvodů je open source technologie pro tento projekt jasnou volbou.

Nabízí se další otázka, zda vybraný framework používá jazyk, který bude vývojáři vyhovovat. To je dobré zvážit, protože v daném jazyce bude projekt psán. Vhodné je zvážit vlastní zkušenosti s určitými programovacími jazyky, případně zájem o novou zkušenost s novými jazyky a zvolit vhodnou alternativu. Součástí takové volby je i prostudování si dokumentace jednotlivých jazyků a výběr nejvhodnějšího způsobu dle různých parametrů jako je funkčnost s frameworkem, ve kterém plánuje programátor pracovat. Psaní projektu není jen jednodenní záležitostí, součástí je jeho údržba a oprava chyb, takže je autorka kódu zavázána k tomu, aby se ve vybraném softwaru a jazyce dokázal orientovat i později. V dnešním světe, kdy jdou technologie neuvěřitelnou rychlostí dopředu, je to někdy spletitý úkol.

Významným bodem k prodiskutování je i dokumentace vybraného frameworku. Pokud není produkt dostatečně zdokumentovaný a nelze v něm najít návod, jak začít nebo jak ho používat, bude se v něm obtížně vyvíjet. Hledání řešení problémů s vývojem v dané technologii může zabrat obrovské množství času. Příhodné může být to, že je technologie ověřená velkým množstvím zákazníků, kteří software využívají.

Pro volbu softwaru je podstatným kritériem i udržování frameworku. Pokud je práce nad frameworkem uzavřena, nebude v budoucnu opravován a je otázkou času, kdy bude potřeba aplikaci přepsat. Z toho vyplývá, že je vhodné kontrolovat, jestli se na výrobku neustále pracuje a podporuje aktuální technologie. To by mělo přispět k tomu, aby nebylo nezbytné v následujících letech celou aplikaci přepisovat.

3.4 Přístup k datům, využití paměti

V současné době se dostává do seznamu požadavků na většinu aplikací i její offline funkcionalita. Je potřeba si tedy položit otázku, zda je nutné, aby aplikace fungovala offline.

Motivací, aby byl výrobek v případě snížení dostupnosti internetu stále funkční a tzv. „nezamrzal“, je spokojenost klienta. Ten požaduje, aby aplikace byla schopná reagovat i při výpadku internetu. Nepostradatelnou součástí implementace tedy bude i potřeba zajistit zálohování dat na straně uživatele.

To je nezbytné, aby měl zákazník poslední synchronizovaná data v případě nedostatečného nebo žádného připojení k internetu k dispozici a mohl je využívat.

V našem případě bude nutností zůstat konzistentní i bez internetu, čehož se dosáhlo implementací NgRx storu na frontendové části, který zajistí uložení některých již vyhledaných produktů do zařízení, a bude tak možné data využívat i bez připojení k internetu. Pokud by nebyl dostatek potřebných dat k dispozici, zobrazí aplikace stránku s informací, že je potřeba se k internetu připojit. Tudíž se aplikace nikdy nedostane do stavu, kdy uživatel není schopen se v ní pohybovat.

Zajištění plné funkčnosti i při špatném nebo žádném připojení bylo testováno pomocí testovacích zařízení a nástroje v Chrome prohlížeči, který pomáhá simulovat situace s různými rychlostmi připojení a odlišnou odezvou. Za zmínku stojí i varianta využití vzdáleného testování na zařízeních, avšak ta bývá volně dostupná jen v omezené verzi a další funkcionality jsou zpoplatněny.

Vzhledem k tomu, že hlavní výpočty budou probíhat na backendové části aplikace, je funkcionalita bez připojení k internetu značně omezená. Aplikace je tedy schopna zobrazovat historii dat, ale není schopna vypočítat a předkládat uživateli data nová.

Aplikace bude v důsledku těchto poznatků ukládat některá data pro offline režim, i přesto nebude mít výrazné požadavky na využití paměti na zařízeních uživatele.

3.5 Nativní aplikace vs. PWA

Jak již bylo řečeno v požadavcích uživatele, bude k webové aplikaci vznikat i její mobilní verze. Pro tvorbu aplikace na mobilní zařízení jsou na zvážení dvě varianty. První možností je vytvořit nativní mobilní aplikaci, druhým způsobem je využít PWA.

Volba mezi nativní aplikací a PWA je dnes velmi diskutovaným tématem. Pro zvolení jedné z variant si v následujících sekcích rozebereme klady a zápory obou technologií. Text je inspirován článkem PWA vs Native App and How to Choose Between Them [13].

3.5.1 Nativní aplikace

Nativní aplikace je napsaná jazykem pro konkrétní platformu např. Swift pro iOS nebo Java, Kotlin pro Android tzn. je navržena a naprogramována pro konkrétní platformu. Z toho vyplývá, že je v případě vývoje nativní aplikace nevyhnutelné vyvíjet dva produkty k webové stránce navíc - jednu aplikaci pro iOS a další pro Android zařízení. Tudíž je nutné znát více programovacích jazyků, frameworků a také udržovat tyto aplikace neustále aktuální (tj. kontrola jejich publikace, oprava případných chyb nebo výtek uživatelů).

Nativní aplikace je potřebné publikovat, tzn. aplikace vytvořené pro iOS zařízení nahrát na App Store a Android aplikace např. na Windows Store nebo Google Play Store. V tomto procesu je každá aplikace zkontrolována App

Storem případně Google Play Storem a v případě, že je nezávadná, je přidána do obchodu. Po publikování je následně dohledatelná pouze v těchto obchodech. Rozšiřitelnost aplikace mezi uživatele je zde usnadněna pomocí ASO. ASO je proces, který zvyšuje postavení aplikace při vyhledávání na storu. Zahrnuje to například volbu správných popisků k aplikaci atd.

Bezpečnost v nativních aplikacích je zajištěna například více faktorovým ověřováním nebo certifikáty pro ještě bezpečnější komunikaci. Pro aplikaci je možné zabudovat TLS certifikát, který zajistí vysoký standard, co se týká bezpečnostní stránky. Kromě toho je potřeba, aby byla aplikace schválena v případě Androidu na Google Play, v případě Applu na Apps Storu, takže projde další kontrolou navíc.

Předností nativní aplikace je možnost komunikace s ostatními mobilními aplikacemi jako je například Facebook. Jde tak snadno uživatele přesměrovat do jiné aplikace. Nicméně tuto funkcionalitu ve své práci nebudu potřebovat.

V případě nativní aplikace je zajištěna podpora různých nástrojů v souvislosti s vývojem pro konkrétní platformu. V tomto projektu by mohlo být výhodné využití knihoven pro připojení kamery, zpracování EAN kódu atp. Kladem nativní aplikace je i to, že je psána v nativním jazyce tzn. je schopna efektivněji zatěžovat zařízení. Nicméně tato výhoda by zůstala také nevyužita, protože potřebná funkcionalita bude probíhat na serveru.

Z analýzy nativní aplikace vyplývá, že je výhodná pro aplikace,
které budou hojně využívat prostředky mobilního zařízení jako je například zvuk, světlo atp. Hodí se pro větší projekty, u kterých je zákazník ochoten si připlatit za vývoj dalších dvou produktů pro mobilní zařízení, která jsou udržována dvěma týmy vývojářů zaměřených na konkrétní platformu a zajistí tak jejich aktualizaci a opravy. Případně je jasně lepší, pokud je nutné využívat více paměti nebo jsou velmi vysoké požadavky na výkon.

3.5.2 PWA

V posledních letech začíná být velmi populárním typem produktu progresivní webová aplikace (PWA). PWA kombinuje funkci nativní aplikace s přístupem z webové stránky. Pro napsání takové aplikace je třeba znát JavaScript, CSS a HTML. To znamená, že k jejímu vývoji stačí specialista na webové stránky a znalost rozšiřitelnosti tohoto kódu o možnosti PWA.

Vytvoření takové aplikace je levnější, jelikož postačí jen jedna univerzální verze pro všechny platformy, tzn. že je kód jednotný pro všechny typy aplikací, není třeba ho přepisovat do několika různých jazyků a znát spousty softwarových nástrojů. Je tudíž i jednodušší na údržbu. Na správu aktualizací, opravu chyb a další úpravy je nezbytné opravovat jen jeden zdrojový kód.

Pro uživatele je potřebný jen webový prohlížeč a URL adresa produktu. To vše stačí pro přidání ikony na plochu a využívání aplikace z mobilního zařízení. Pro rozšíření aplikace mezi zákazníky je u PWA umožněno využívat SEO. Aplikaci lze snadno sdílet pomocí URL, tím je možné nasměrovat ostatní

na stejnou stránku, kterou uživatel sdílí. Oproti nativní aplikaci je PWA snadněji dostupná a nevyžaduje komplexní instalaci, tedy stahování z App Storu nebo Google Play, jak je tomu v případě aplikace napsané pro Android nebo iOS telefony.

Pro PWA je nutné využití HTTPS protokolu, díky kterému je aplikace bezpečnější. HTTPS protokol zajišťuje, že nedochází k narušení komunikace mezi klientem a serverem.

Jelikož PWA využívá service workers ke zpracovávání požadavků, je její načítání rychlé i v případě omezení internetu. To platí i v případě, kdy nejsou k dispozici nová data v důsledku výpadku internetu. Jinými slovy je PWA nezávislá na připojení.

Výhodou pro uživatele může být to, že PWA aplikace se nijak neliší od webové stránky. Prostředí aplikace je sjednoceno a uživatel se v ní jednoduše vyzná stejně jako tomu je na webové stránce, takže se nemusí učit pracovat s dalším uživatelským rozhraním. Je progresivní, takže pracuje na různých prohlížečích nebo operačních systémech a responzivní, jelikož se přizpůsobuje zařízení.

Co se úložiště týká, využívá PWA aplikace méně místa než aplikace nativní. Pokud tedy řeší zákazník málo místa na svém úložišti, s PWA aplikací mu tato starost většinou odpadá.

V dnešní době je dostatek místa na úložišti mobilního zařízení již častým standardem, avšak stále tímto aspektem zvyšuje PWA možný počet jejich uživatelů. Nemluvě o tom, že stažení PWA netrvá tolik, co stahování nativní aplikace. V našem případě by však byl rozdíl pravděpodobně zanedbatelný, protože se jedná o malou aplikaci.

Celkově ve světě stoupá na oblibě a pravděpodobně je PWA i hudbou budoucnosti. Je dost možné, že postupným vylepšováním této technologie bude brzy tak velkou konkurencí pro nativní aplikace, že je nahradí úplně. To je však otázkou, na kterou zatím nelze odpovědět. Některé informace čerpány z [14].

3.6 Frontendová část aplikace

Frontendovou část aplikace zastřešují technologie, které se podílejí na vývoji toho, co je uživatelem viditelné tj. uživatelské rozhraní (UI). Jedná se o jazyky, které běží na klientově straně, neboli v prohlížeči.

Na žebříček oblíbenosti se mezi frontendovými vývojáři v posledních letech dostali tři nejznámější frameworky - React.js, Angular a Vue.js. Za frontendovou část je zvolen framework Angular s programovacím jazykem TypeScript. Důvodem je předchozí zkušenost autorky s těmito technologiemi.

Pro naučení technologie bývá Angular řazen spíše k těm obtížnějším, protože pro jeho používání je podmínkou znalost TypeScriptu.
Po něm následuje React a nakonec je Vue, které je jednoduché na naučení.

TypeSctipt je typovaný jazyk, to má za důsledek to, že je možné odhalit mnoho chyb ještě před spuštěním aplikace a tím přidává Angularu na jeho hodnotě.

React se vyznačuje tím, že je rychlejší oproti Angularu, protože je v něm implementován Virtual DOM a mnoho rendrovacích optimalizací. React používá funkcionální programování. Angular je založený na OOP a to je autorce bližší.

Vue.js je založeno na architektuře MVVM. Vue.js je v porovnání s těmito velkými soupeři opravdu velikostně malé. Jde s ním vytvořit single-page aplikaci, ale zároveň i složité webové aplikace. Jde ho integrovat do již existujících systémů bez vedlejšího efektu na daný systém, pokud se nejedná o opravdu velký projekt. Vue.js má menší komunitu a menší podporu než jeho dva stoupenci.

Za zvážení stojí ale využití nástroje Ionic, který by umožnil napsat jeden jediný kód pro všechny platformy (web, Android, IOS).

3.6.1 Angular

Angular byl vyvinut Google týmem jako nadstavba původního Angular.js. Je to framework, který je založený na komponentách. Naproti Reactu nabízí např. 2-way data binding (všechny změny provedené v modelu se promítnou na view, tj. zobrazení v aplikaci a obráceně). Naučit se Angular je poměrně složité, protože má velmi rozsáhlou dokumentaci a velkou komunitu.

Dokumentace je často matoucí, obzvláště pro začátečníky, ale vzhledem k tomu, že autorka již nějaké zkušeností s touto technologií má, není dokumentace takovou překážkou.

Pokud je vyvíjen projekt v Angularu, musí vývojář znát programovací jazyk TypeScript. TypeScript se pro mnohé programátory může nalézat spíše v mínusech této technologie, protože je nutné se ho naučit, ale obecně je brán jako výhoda. Více si ho rozebereme v následujícím bodě.

3.6.2 TypeScript

TypeScript je open source, který byl navržen firmou Microsoft v roce 2012. Jazyk je nadstavbou JavaScriptu, kde oproti svému „rodiči“ dovoluje uvádět datové typy, tzn. rozšiřuje JavaScript o statické typování, používání rozhraní, tříd a další věci z OOP. Výsledkem je i lepší napovídání např. ve vývojovém prostředí WebStorm a automatická kontrola. Pro všechny tyto funkce navíc potřebuje TypeScript speciální kompilátor tzv. transpiler. Ten převádí kód TypeScript do klasické verze ES5.

TypeScript spadá pod typované jazyky, to znamená, že se musí deklarovat datový typ. Možnou nevýhodou je, že je v tomto případě objemnější a jeho vývoj trvá o něco déle. Nicméně velkým plusem je kontrola kompilátorem, zda datový typ sedí ve všech případech. TypeScript nedovoluje spuštění pro-

gramu a zkompilování, pokud program obsahuje chybu, což u JavaScriptu není. V programu založeném na JavaScriptu je chybu možné odhalit například až při běhu aplikace.

3.6.3 Ionic

Ionic framework je volně dostupným nástrojem, jehož prostřednictvím lze vytvořit aplikaci jak pro mobilní, tak i pro webové aplikace používající webové technologie (HTML, CSS, JavaScript). Zaměřuje se na frontendovou část aplikací a lze ho kombinovat s Angularem, případně Reactem či Vue. Konkrétně Angular tento nástroj podporuje od verze 6.0.0 a výš. Používá web API jako Custom Elementy a Shadow DOM, jak se lze dočíst v jeho dokumentaci [15].

Custom elementy jsou jedním z klíčových prvků standardu webové komponenty. [16]

Dalším důležitým aspektem webových komponent je tzv. zapouzdření. Zapouzdření je schopnost komponent udržovat strukturu, styl, chování a oddělení od jiných kódů na stránce, aby se různé části neshodovaly, tzn. udržovat kód pěkný a čistý. Klíčovým prvkem je rozhraní Shadow DOM API, které nabízí způsob, jak připojit k prvku skrytou oddělenou doménu. [17]

Pro frontendovou část byl zvolen framework Angular s Ionicem a jazykem TypeScript. Implementací pomocí Ionicu zajistí autorka vývoj jak webové, tak mobilní aplikace. Tyto aplikace budou mít stejný základ, ale nebude se jedná o PWA aplikaci. Důvodem je fakt, že je v plánu mobilní aplikaci publikovat na App Store a Google Play.

Tato varianta využívá stejný kód pro různé platformy. Tím snižuje pracovní zátěž a usnadňuje práci vývojáře. Je však stále potřeba počítat s tím, že v některých případech bude nutné aplikace odlišit. To znamená, že webová aplikace bude v některých částech využívat jiné komponenty. Například pro webový prohlížeč není vhodné používat tab komponenty použité v případě mobilního zařízení. Ty lze nahradit využitím menu.

3.7 Volba databázového systému

Databáze jako takové jsou systémy, které slouží k ukládání dat. „Databázové systémy jsou speciální programy, které se používají pro správu datových souborů. Soubory mohou být značně rozsáhlé a lze v nich vyhledávat, třídit, organizovat, měnit, mazat, zadávat, různým způsobem zobrazovat a podobně.“ [18] Za vznikem původních databází stály papírové kartotéky, ve kterých veškerou práci s daty odváděl její správce. Postupným vývojem a digitalizací se databáze proměnily v databázové systémy, jak je známé již dnes.

Součástí analýzy a návrhu jakéhokoli projektu by měla být i rešerše aktuálních technologií, které lze pro tvorbu aplikace využít. Pro výběr tech-

nologie je nutné stanovit si jasné požadavky na aplikaci a volbu těmto účelům přizpůsobit.

Vzhledem k tomu, že se autorka práce specializuje převážně na frontend vývoj, je nezbytné udělat podrobnější analýzu databázových systémů a porovnat jejich přínosy pro projekt. Argumentem pro zařazení rozboru vybraných systémů do bakalářské práce je odvrácení rizika přepisování již napsaného kódu do jiné technologie z důvodu, že databáze nebude projektu vyhovovat. Zmapováním možností získá programátor přehled o dostupných softwarech pro vývoj a je schopný zvolit typ, který by v budoucnu měl zajistit kompatibilitu s nároky projektu.

V rámci průzkumu byly vybrány některé doporučené databáze,
kterým je věnována větší pozornost. Všechny vybrané systémy byly zvoleny tak,
aby se jednalo o open source technologie a jsou založeny na bázi požadavků klienta. Pro celistvost je rozbor doplněn i o základní informace pro jednotlivé typy databází, které do výběru pro vlastní projekt nebudou s největší pravděpodobností použity.

Databáze lze dělit podle různých specifikací jako je například rozklad podle obsahu, formy nebo třeba architektury. V souvislosti s různým dělením databází a jejich porovnáváním je dobré zmínit tzv. CAP teorém. Tento teorém by mohl pomoci s výběrem budoucího systému, proto je do bakalářské práce zařazen jako součást zmapování existujících technologií.

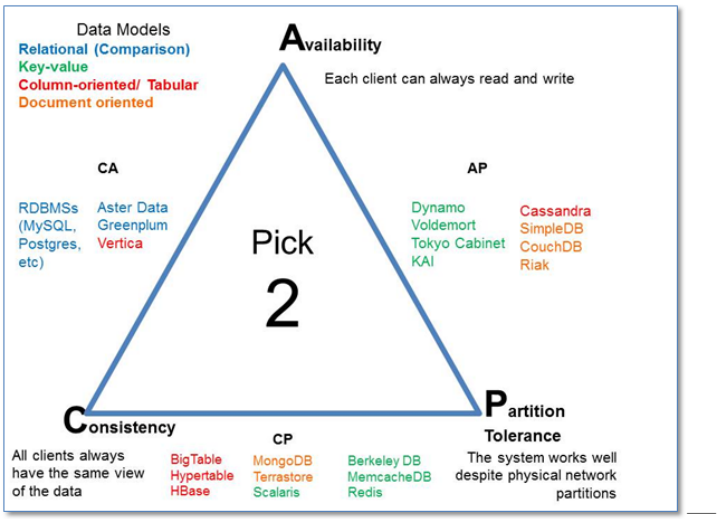
Ve svém zkoumání databáze rozděluji podle způsobu ukládání dat. Dělením podle dat dostaneme čtyři skupiny, a to databáze hierarchické, síťové, objektové a relační. Hierarchické a síťové databázové systémy nebudou pro projekt využity a nejsou součástí analýzy. Naopak byly do analýzy přidány NoSQL databáze, které by požadavkům projektu mohly vyhovovat.

3.7.1 CAP teorém (též Brewerův teorém)

CAP teorém vymezuje určitý omezený prostor, ve kterém jsou databázové systémy definované. Vytyčuje pro ně tři hlavní vlastnosti: konzistence
(**C**onsistency), dostupnost (**A**vailability), tolerance oddílů (**P**artitioning tolerance) a říká, že neexistuje žádný databázový systém, který by byl schopen v jednu chvíli ovládat všechny tyto vlastnosti najednou. Databáze je schopna splňovat maximálně dvě z nich.

Pro úplné chápání rozdělení databází podle CAP teorému si jednotlivé vlastnosti popíšeme.

Systém, který je konzistentní, poskytuje všem uzlům stejná data.
Pro představu, máme jednoho uživatele databáze v New Yorku, který využívá uzel A a druhého v Brně, který je napojen na uzel B. Oba si zažádají ve stejný čas o data pro stejnou knihu a oba uzly jim oběma vrátí stejné informace
o vybrané knize tzn. není možné, aby vracely každému něco jiného.



Obrázek 3.1: CAP teorém [1]

Dostupná databáze je taková databáze, která garantuje, že na každý dotaz dostane klient vždy odpověď. To znamená, že dostane odezvu s úspěchem nebo se vrátí informace o chybě.

Systém tolerující oddíly je schopen pokračovat v provozu, i pokud je ztracena zpráva nebo některá jeho část selže. Takže v případě, že mezi dvěma servery přestane fungovat spojení a nejsou tak schopny spolu komunikovat, bude systém nadále fungovat.

CAP teorém bývá graficky znázorněn pomocí trojúhelníku (jak je možné vidět na obrázku č. 3.1) nebo Vénnovým diagramem.

U teorému znázorněného trojúhelníkem je každá vlastnost součástí jednoho vrcholu. Hrany potom značí propojení dvou vlastností. Databáze jsou umisťovány kolem trojúhelníku dle kombinace znaků, které jsou jim charakteristické. Uvnitř trojúhelníku se nenachází žádná z nich, protože ani jedna nemá současně všechny tři vlastnosti.

Zakreslení pomocí Vénnových diagramů pro 3 množiny (konzistence, dostupnost, tolerance oddílů) je dalším možným grafickým zobrazením, kdy jsou databáze zakresleny do množin podle svých rysů a do průniku všech množin opět nespadá žádná z nich.

Jak již bylo zmíněno výše, z této teorie vyplývá, že každá databáze bude mít jen dvě tyto vlastnosti. Pro výběr databáze je dobré zvážit, které dvě vlastnosti jsou pro projekt nutnou součástí, a bez které je možné se obejít. Tím nám vzniká možnost výběru ze tří typů: konzistence-dostupnost, konzistence-tolerance a dostupnost-tolerance.

Mezi systémy s vlastnostmi konzistence a dostupnosti patří například

relační databáze (MySQL, POSTGRES, atd.). Tyto technologie zajišťují, že je každému uzlu vrácená ve stejný čas stejná (aktualizovaná) hodnota a že na každou odpověď bude vrácena nějaká odezva. Nejsou však schopny správné funkcionality za podmínky, že některá z částí systému přestane fungovat. Obvykle jsou vytvořeny na jednom serveru.

Varianta dostupnosti a tolerance oddílů je neustále dostupná a zároveň toleruje výpadky některých částí, ale nezajišťuje konzistenci dat. Pokud budou do systému zapisovat současně dva uzly, mohou vzniknou mezi daty konflikty a ty je potom třeba v databázi řešit. Systém vyznačující se těmito vlastnostmi je například CouchDB.

Kombinace konzistence a tolerance oddílů má za důsledek to, že systém bude odpověď blokovat, dokud nebudou data konzistentní a dokud je nepůjde odeslat klientovi. Z toho vyplývá, že bude služba v některých případech nedostupná. Tolerancí oddílů je zajištěno, že pokud některá část není funkční, je schopen systém přesto správně fungovat. Mezi technologie řadící se k těmto vlastnostem patří například MongoDB.

Pro funkcionalitu na více serverech je nutné, aby měla databáze vlastnost tolerance oddílů. Po zmapování databází pomocí CAP teorému vychází pro projekt jako nejvhodnější volba dvou vlastností konzistence-tolerance oddílů nebo dostupnost-tolerance oddílů.

3.7.2 Relační databáze

Relační databáze má definovanou strukturu relačním modelem, v němž jsou data logicky uspořádána do relací, tj. výsledků kartézského součinu nad množinami údajů. Relační databáze je založena na tabulkách, které obvykle chápeme tak, že uchovávají informace o relacích mezi jednotlivými záznamy.

Základním prvkem relačních databází jsou tedy relace (tabulky), které jsou naplněny záznamy (řádky).

Tyto systémy se začaly vyvíjet někdy v 70., 80. letech.

Zásadním rozdílem oproti objektové technologii je způsob vkládání dat do databáze. Pro vkládání do relační databáze se nejdříve vytváří tabulka a nastavuje se typ atributů, primárních klíčů apod. Poté je dovoleno do databáze vkládat jednotlivé řádky. To práci s databází trochu komplikuje oproti jednoduchému vkládání objektů v případě objektové technologie.

Další nevýhodou principu relační databáze pro projekt je nutnost vyhledávání primárního a cizího klíče při propojení tabulek. To vyhledávání záznamů značně zpomaluje. Pro urychlené hledání nejčastěji používaných spojení musí být použity indexy.

Naproti tomu výhodou relační databáze je její možnost upravovat konkrétní buňku tabulky. Relační databáze je výhodná z pohledu vytížení zdrojů počítače.

Na operace pravděpodobně nebude potřebovat zásadně vytěžovat systém. Avšak data se nám budou vracet pomaleji v případě, že se bude jednat o dost zanořený dotaz (propojování více tabulek).

Velkou část relačních databází lze horizontálně škálovat, to znamená, že ji lze rozdělit do více uzlů. Spojování velkých tabulek pro horizontálně škálované databáze může být pomalé, protože výsledné spojení lze uskutečnit jen na jednom uzlu. Navíc jsou často jednotlivé uzly přetížené velkým přenosem dat, a to systém zpomaluje.

Relační databáze podporují transakce a mají jejich vlastnosti. Je pro ně tedy charakteristická atomicita, konzistence, izolovanost a trvalost (ACID). Tyto vlastnosti jsou požadovány například pro bankovní převody. V důsledku důrazu na bezpečnost a celistvost dat je databáze většinou pomalejší.

Výběr konkrétních systémů je omezen jen na volně dostupné technologie, proto byly pro hlubší analýzu zvoleny databáze MySQL, MariaDB a POSTGRESQL.

3.7.2.1 MySQL

MySQL je relační databáze, která ke komunikaci využívá jazyk SQL. Tuto databázi lze využívat serverem běžícím na platformě Microsoft Windows Serveru i Linuxu a je v oblasti webových aplikací velmi rozšířená. S porovnáním možností cloudového úložiště databáze však v dnešní době již klesá na oblibě.

U MySQL databáze je výhodou větší kontrola a možnost zpracovávání. MySQL databáze nemá dostatečný výkon při extrémně velkém zatěžování webové aplikace. Aktuální požadavky vývoje „kostry“ aplikace nejsou tak výrazné, aby tuto technologii zavrhly, nicméně je v plánu software vyvíjet a tím umožnit klientovi s aplikací více interagovat. To způsobí, že nastanou vyšší nároky na běh aplikace a databáze by byla zahlcena více požadavky, které by nemusela být schopna v rychlém časovém sledu obsloužit. Vzhledem k tomu, že bude aplikace dále rozšiřována a požadavky na její správu a výkon budou stoupat, nebude MySQL pravděpodobně vhodnou volbou.

Dalším důležitým bodem, který vylučuje databázi z výběru (pravděpodobně i zbytek relačních databází), je fakt, že relační databáze mají danou pevnou strukturu, což pro projekt není úplně tak vhodné.

V důsledku vyhnutí se migracím je tato technologie zamítnuta.

3.7.2.2 MariaDB

MariaDB je tzv. nástupníkem MySQL. Rozdílem mezi nimi je to, že MariaDB má plnou GNU GPL licenci (obecnou veřejnou licenci GNU) a MySQL má duální licenci (komerční i volně dostupná verze). MariaDB podporuje více modulů a nabízí většinou lepší výkon.

Vzhledem k tomu, že se jedná o odkloněnou větev MySQL, lze mezi těmito databázemi migrovat. Aktuálně je však nutné při migraci již některé změny

kódu udělat, protože se databáze MariaDB začíná od MySQL postupem času lišit. Oproti MySQL na MariaDB vychází bezpečnostní vydání.

Z dlouhodobého hlediska je MariaDB citlivá na zvětšování svého objemu. V důsledku nabývání na velikost se zpomaluje její výkon. Jak již bylo zmíněno výše u MySQL databáze, jsou relační databáze nevhodné z více důvodů, i proto nebude MariaDB do projektu zvolena a není nutné ji hlouběji analyzovat.

3.7.2.3 POSTGRESQL

Lepší alternativou je databázový systém POSTGRESQL, který je také volně dostupný a s jeho pomocí je možné vyvíjet databázi na nepřeberném množství serverových jazyků jako je Python, Perl, Java atp. Disponuje vlastním procedurálním jazykem PgSQL a má řadu funkcí SQL jazyka.

Protože je zde využit vlastní jazyk, některé hostingové servery ho nemusí podporovat. Je také pomalejší při malém množství uživatelů, což by mohlo aplikaci činit z počátku pomalou.

Základní dokumentace POSTGRESQL není příliš popisná a pro její zlepšení je zde ještě dostatek prostoru. V souvislosti s těmito zjištěními a poznatky z předchozích kapitol o relačních databázích se došlo k závěru, že nebude tato technologie pro projekt vybrána.

3.7.3 Objektová databáze

V 90. letech se začaly objevovat první objektové databáze. V nich jsou informace reprezentovány ve formě objektů, stejně jak tomu je u OO programování.

Objekty jsou zapouzdřené a funguje mezi nimi dědičnost. Místo tabulek jsou zde uloženy přímo objekty včetně svých vlastností. Místo řádků se ukládají samotné instance objektů.

Původně se předpokládalo, že objektová databáze nahradí relační databázi. To se nicméně nestalo a na místo toho vznikly navíc objektově-relační databáze.

Objektová databáze se od relační databáze liší tím, že při vkládání objektů se objekt ukládá jako celek a není potřeba před jeho uložením databázi nijak připravovat. Pro získání dat z databáze je vyhledán nejvýše postavený objekt, ve kterém jsou vnořené vyhledávané záznamy.

V relačních databázích se dá pracovat s jednoduchými datovými typy a ty následně mapovat na objekty. U objektové databáze tyto úpravy nejsou nutné, protože výstupem je již objekt. Je možné využívat dědičnosti, zapouzdření a polymorfismu.

Nevýhodou objektové databáze je, že je vracen vždy celý objekt i s vnořenými objekty. V případě složitých struktur musí odpovídat celou strukturou. Tato záležitost je v databázi řešena pomocí nastavování hloubky zanoření.

Objekt je dohledatelný přímým přístupem pomocí OID, ten na logické úrovní ukazuje do virtuální paměti, proto se pro objekty nevytváří primární klíče.

U relačních databází je možnost úpravy pouze konkrétní položky. Pro editaci v relační databázi tak lze upravovat jen část, v objektových databázích je nutno editovat celý objekt.

Pro objektové databáze nepůjde analýza do větší hloubky, protože žádná z těchto databází nebude pro aplikaci zvolena. Je zde varianta kombinace relační a objektové databáze a ta by mohla být pro náš budoucí projekt vyhovující.

Objektově relační databáze jsou kombinací obou databází a neexistuje pro ně žádný standard, avšak je nutná podpora objektově orientovaného přístupu. Mnoho z těchto databází nabízí verzování objektů.

3.7.4 NoSQL databáze

V rámci analýzy jsem narazila také na NoSQL databáze, které jsou jinou alternativou pro tradiční relační databázi. Hodí se zejména pro práci s velkým objemem dat. Nabízí speciální práci se strukturou dat, kterou v projektu můžeme využít.

Naopak není vhodná, pokud potřebujeme transakce, pevnou strukturu dat nebo je potřeba tabulky propojovat.

NoSQL databáze lze dělit na dokumentované, databáze klíč-hodnota, sloupcové a grafové. Dokumentované databáze využívají místo řádku dokument. Má relační i objektové typování. Je vhodná pro zpracování reálně se měnících dokumentů. Databáze klíč-hodnota dává důraz na rychlost v práci s daty. Data jsou ukládána podle unikátního klíče. U sloupcové databáze jsou data uložena do sloupců, které lze libovolně přidávat. Hodí se pro datové sklady. Grafová databáze funguje na základě grafů tzn. obsahuje uzly a hrany a má určité vztahy mezi sousedními uzly.

Pro analýzu byly vybrány volně dostupné databáze CouchDB a MongoDB.

3.7.4.1 CouchDB

Jedná se o open source dokumentově orientovaný databázový systém. Používá kolekci JSON dokumentů, kdy kolekce těchto dokumentů nemá stejnou strukturu a na dotazy fungují pohledy. Nabízí ACID (je schopná pracovat s transakcemi), RESTful http API (viz. bod č. 3.8 str. 26), které je požadavkem projektu a možnost různých jazyků pro pohledové servery jako je třeba PHP, kterým bude psána část aplikace. Podporuje offline režim a synchronizaci oběma směry. Z toho vyplývá, že si replika uchovává své vlastní kopie a později se vzájemně synchronizují. Problémem jsou vzdálené konflikty.

3.7.4.2 MongoDB

Jedná se o NoSQL nebo také dokumentově orientovanou open source databázi, která ukládá data v JSON dokumentech. Je objektově orientovaná.
Pro MongoDB je tento formát označován jako BSON (binární JSON). Data v ní mají daný formát a objekty do sebe lze vnořovat. MongoDB může běžet na více serverech.

Which database is right for your business?		
	CouchDB	**MongoDB**
Speed	If read speed is critical to your database, MongoDB is faster than CouchDB.	MongoDB provides faster read speeds.
Mobile support	CouchDB can be run on Apple iOS and Android devices, offering support for mobile devices.	No mobile support provided.
Replication	Offers master-master and master-slave replication.	Offers master-slave replication.
Size	While your database can grow with CouchDB, MongoDB is better suited for rapid growth when the structure is not clearly defined from the beginning.	If you have a rapidly growing database, MongoDB is the better choice.
Syntax	Queries use map-reduce functions. While it may be an elegant solution, it can be more difficult for people with traditional SQL experience to learn.	For users with SQL knowledge, MongoDB is easier to learn as it is closer in syntax.
Analysis	If you need a database that runs on mobile, needs master-master replication, or single server durability, then CouchDB is a great choice.	If you need maximum throughput, or have a rapidly growing database, MongoDB is the way to go.

Obrázek 3.2: CouchDB vs. MongoDB [2]

V plánu projektu je využití technologie PWA, kdy je důraz na rychlost. Podle tabulky je v tomto lepší volba MongoDB. Je dost možné, že databáze bude růst exponenciálně rychle, tím pádem opět vyhovuje více MongoDB. MongoDB je zároveň lepší na pochopení se znalostí SQL, to je rovněž znak, který je pro autorku přínosem. Vybrána je proto MongoDB.

3.8 REST API

REST je datově orientované rozhraní, které určuje, jak se bude přistupovat k datům v distribuovaném prostředí. Ve spojení s JSON se stává velmi populárním API v oblasti webu. Pro webovou aplikaci je v dnešní době vhodné paralelní zpracování, které REST poskytuje. Tato technologie je totiž beze stavová. Využívá HTTP a ve většině případů se používá 5 metod: GET, POST, PUT, DELETE, PATCH.

HTTP GET je metoda pro získání dat. POST metoda je určená pro vytváření záznamů v systému. Pro metodu POST se využívá konkrétní URI pro konkrétní zdroj. Je učena k vytváření nové entity. PUT je podobná metodě POST, ale používá se v souvislosti s úpravou prvku. Při volání URI HTTP s metodou DELETE je smazán objekt z databáze.

Pro REST je charakteristické to, že se neposílá požadavek na jeden centrální zdroj, ale má těch bodů více. Ukážeme si na příkladu. Budu posílat např. GET /users - vrátí se mi seznam všech uživatelů a pokud budu posílat GET /users/1/activities - vrátí se mi aktivity pro uživatele s id 1.

Pro URL je vhodné používat následující schéma zdroj/id_zdroje/relace/id_relace to znamená, že bude vypadat např. DELETE /users/1/activities/2 - mažu pro uživatele s id 1 aktivitu s id 2.

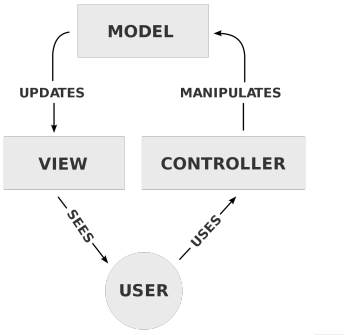
Vzhledem k tomu, že k technologii náleží určité standardy, je velmi jednoduché ji použít pro další uživatele nebo napojit na jiné API.

Pojem RESTful je potom používán pro technologii, která splňuje všechny body REST rozhraní.

3.9 Nette Framework

Nette Framework je český open source framework pro práci s webovými aplikacemi v PHP 5 a PHP 7. Skládá se ze samostatných a znovupoužitelných balíčků. Jeho autorem je David Grudl, dnes jeho rozvoj zajišťuje organizace Nette Foundation. Podporuje AJAX, KISS, DRY a MVC a eliminuje bezpečnostní rizika. Je používán významnými společnostmi jako jsou GE Money, ESET nebo třeba Slevomat. V České republice je jedním z nejpopulárnějších a nejpoužívanějších frameworků.

3.9.1 MVC



Obrázek 3.3: MVC schéma [3]

MVC neboli model-view-controler je architektura, která vznikla na základě potřeby oddělit aplikační logiku - model, kód obsluhy - controller od kódu pro zobrazení dat - view. Tímto schématem je zajištěno, že bude kód lépe testovatelný, čitelnější a pokud bude některou část nutno upravit, nebudou mít tyto změny v ostatních částech velký dopad (mnohdy nejsou ovlivněny vůbec).

3.9.1.1 Model

Model obsahuje data a celou business logiku aplikace. Spravuje si svůj vnitřní stav a o existenci view nebo kontroleru vůbec neví.

„Pojem Model představuje celou vrstvu, nikoliv samostatnou třídu.“ [19]

3.9.1.2 View

View, česky pohled, je uživatelským rozhranním (UI). Jeho úkolem je zobrazit data.

3.9.1.3 Controller

Controller, česky řadič, zajišťuje práci se vstupy od uživatele. Jinými slovy definuje chování aplikace. V Nette Fremeworku jsou jejich obdobou presentery.

3.9.2 PHP

PHP patří mezi dynamicky typované skriptovací programovací jazyky.
Je využíván pro vývoj webových stránek nebo konzolových či webových aplikací.

Tento jazyk není závislý na operačním systému, lze ho tedy mezi systémy většinou bez problému přenášet. Na projektu pracuje aktuálně jen jeden člověk, ale do budoucna se počítá, že se na něm bude podílet více pracovníků a ti mohou mít různé požadavky na využívaný operační systém, což s PHP jazykem nebude problém.

Pro webové stránky je využíván na straně serveru. Syntaxe je podobná několika jazykům (Perl, C, Pascal a Java). Vzhledem k předešlým zkušenostem s jazykem C a Java je PHP dobrou volbou i v tomto ohledu. Jeho učení bude velmi rychlé.

Jazyk je rozšířený díky jednoduchému použití a velkému množství funkcí. V programovacím jazyce PHP je napsaný např. Facebook.

Negativní ohlasy na PHP mohou být třeba z důvodu, že u podobných funkcí neřadí parametry stejně, což je v počáteční fázi učení velmi nepříjemné. Další nevýhodou je horší ladění chyb oproti staticky typovaným jazykům.

Mezi výhody používání PHP patří jeho specializovanost na webové stránky a možnost využití velkého spektra funkcí. To je pro aplikaci značnou výhodou, protože jde o vývoj webové aplikace.

Celkem dobře pochopitelná dokumentace, kterou PHP má, je pro začátečníka v backendové sekci velkým usnadněním a vzhledem k jeho oblibě bude možné případně dohledat potřebné rady a tipy v jeho komunitě.

PHP má obrovskou podporu na hostingových službách. Pro projekt
je důležité, že konkrétně v České republice je hosting opravdu jednoduché sehnat.

Kapitola 4

Realizace

4.1 Frontendová část

4.1.1 Webová a mobilní aplikace

Pro webovou část aplikace byl v konečném důsledku zvolen framework Ionic. Důvodem pro volbu Ionicu byla možnost zapisovat v něm stejný kód pro více aplikací. Tím se usnadnila práce na frontendové části aplikace. Zároveň implementace aplikace v poměrně nové technologii Ionic slouží jako průzkum této technologie a v závěru bude shrnuta zkušenost s tímto frameworkem z pohledu frontend developera pracujícího v Angularu.

V průběhu práce s Ionicem byla vizuální stránka webové aplikace v některých případech odlišena od aplikace pro mobilní zařízení.

Pro mobilní zařízení byly přidány taby, které se odkazují na skener a na vyhledávač. Na webové části tyto taby zobrazovány nejsou. Architektura webové aplikace je odlišná a je založena na vyhledávači, do kterého uživatel může zadat kód potraviny. Skener je zde také implementován, ale nepředpokládá se jeho velké využití.

Po vyhledání nebo naskenování kódu potraviny je zobrazena komponenta s názvem a kalorickou hodnotou potraviny. Dále v ní uživatel nalezne devět různých sportovních aktivit, u kterých je jednotlivě vyhodnocena doba, kterou je nutno provozovat danou sportovní aktivitu pro spálení vyhledané potraviny.

Sportovní aktivity byly pro aplikaci zvoleny tak, aby si každý uživatel mohl zvolit činnost sobě blízkou a vhodnou. Proto byly vybrány tyto různorodé aktivity: chůze, plavání, kolo, běh, volejbal, hokej, fotbal, golf, tenis. Sportovní úroveň vybraných aktivit je na amatérské úrovni.

Načítání kódu potravin je zpracováno zvlášť pro mobil a zvlášť pro počítač, protože knihovna frameworku Ionic pro čtení barkódu není určena pro webový prohlížeč. Původně byla implementována knihovna nxg-scanner (pro webový prohlížeč) [20] na obou platformách, ale knihovna frameworku Ionic pro mo-

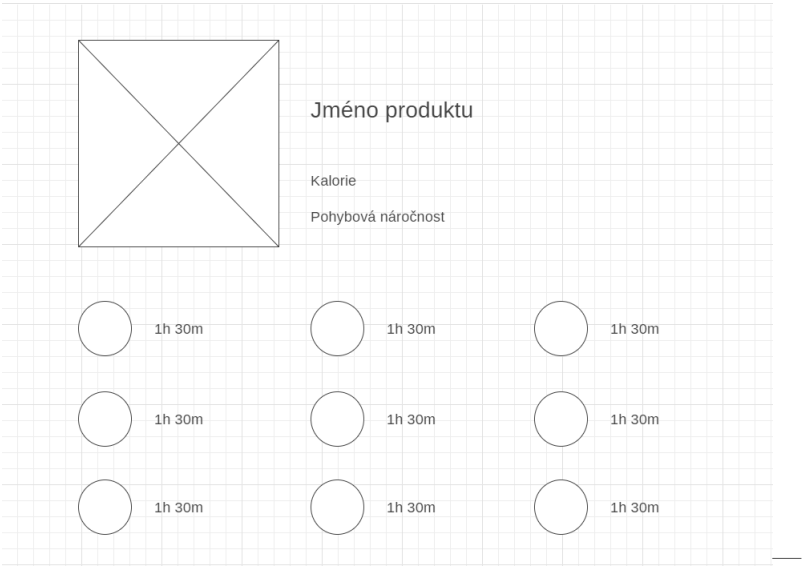
bilní zařízení, která se stará o zajištění naskenování kódu produktu, byla v mnohém přínosnější. To vedlo k implementaci různých knihoven na straně webu a na straně Android aplikace.

Celá frontendová část je otestována unit testy za pomocí Karmy a Protractoru. Pro testy jsou vytvořeny speciální kontexty, které usnadňují práci v testech a není tak třeba psát některý kód opakovaně při inicializaci testů.

4.1.2 Wireframe

Wireframe nebo také „drátěný model“ je model, pomocí kterého se při návrhu webové stránky určí rozmístění prvků aplikace.

Před implementační částí aplikace byly vytvořeny návrhy vzhledu aplikace. Konkrétně autorka zhotovila na obrázku č. 4.1 (verze pro web) a obr. č. 4.2 (pro menší rozlišení) detail komponenty pro zobrazování produktu s pohybovými aktivitami.

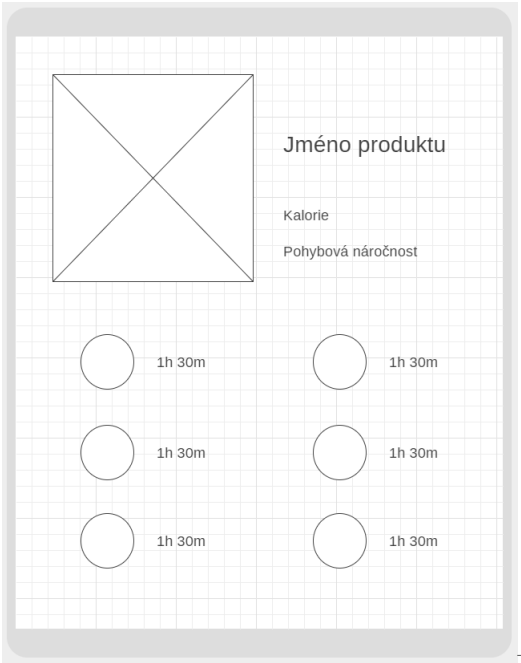


Obrázek 4.1: Wireframe detailu produktu pro web

4.1.3 Výsledná aplikace

Tento model byl (až na odchylku v umístění času pohybu u mobilního zařízení) dodržen. Vzhled aplikace po vyhledání potraviny je možné vidět na obrázku č. 4.3 nebo na obr. č. 4.4.

V horní levé části se nachází obrázek produktu, který je uživatelem vyhledáván. V pravé horní části se zobrazují základní data o produktu (jméno, kalorie a míra pohybové zátěže).



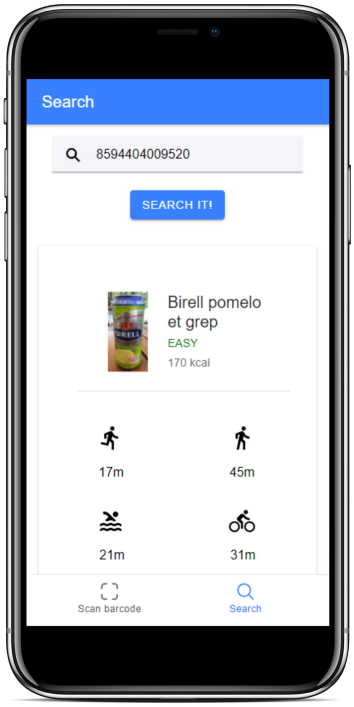
Obrázek 4.2: Wireframe detailu produktu pro nižší rozlišení

Ve spodní části jsou zobrazeny ikony pohybu (ve schématu je znázorňují kolečka) a jejich časový interval nutný ke spálení kalorií vybrané potraviny.

4.1.4 Struktura frontendové části aplikace

Aplikace se skládá z jednotlivých logicky členěných modulů a komponent.

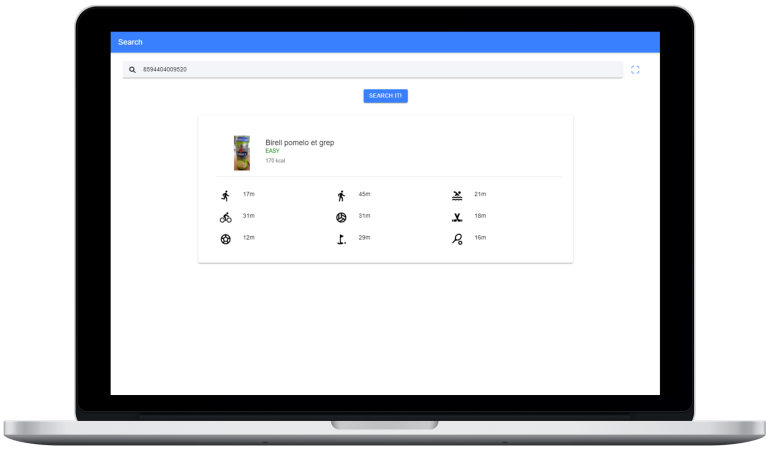
- android
  - Ionicem vygenerovaná složka pro mobilní zařízení Android
  - obsahuje kódy, které aplikaci umožní spustit i na mobilním zařízení
- resources
  - Ionicem vygenerovaná složka pro mobilní zařízení
  - obsahuje splash screen obrazovky a různá jejich rozlišení
  - obsahuje ikony aplikace pro mobilní zařízení
- src/app
  - kódy pro aplikaci
- src/app/app-routing



Obrázek 4.3: Aplikace na mobilním zařízení

  - hlavní routování aplikace, další routovací moduly jsou umístěny ve složkách u svých modulů

- src/app/global
  - obecné funkce, třídy atp. použité pro aplikaci na více místech
- src/app/models
  - modely přijatých nebo odeslaných HTTP requestů
  - modely vytvořené pro použití na frontendové aplikaci
- src/app/modules
  - logicky rozřazené moduly ostatních částí aplikace
- src/app/resources
  - HTTP metody pro komunikaci s backendovou aplikací (serverem)
- src/app/services



Obrázek 4.4: Aplikace na webu

  - servisy obsluhující HTTP metody

- src/app/store
  - hlavní store aplikace
  - aplikace používá knihovnu NgRx store pro ukládání dat do storu
  - každá složka store obsahuje data k obsluze storu - efects, actions, reducers a selectors viz. dokumentace této knihovny [21]

4.2 Backendová část

Backendová část byla implementována pomocí jazyka PHP, Nette.

S databází komunikuje prostřednictvím MongoDB klienta. S frontendovou částí aplikace komunikuje prostřednictvím HTTP dotazů, které jsou založeny na REST API. Data jsou posílána jako JSON. Dokumentaci REST API je možné dohledat na adrese `https://ca15.docs.apiary.io/#reference`.

Po vyhledání produktu je poslán z frontendu dotaz na informace o produktu a jeho pohybové náročnosti. To je na serveru zpracováno. Vznikne dotaz

na databázi, kde si vyžádá aktivity s hodnotou nutného energického výdeje a potravinu z databáze. Na základě těchto údajů jsou generovány pohybové náročnosti produktu a poslány na frontendovou aplikaci.

4.2.1 Struktura backendové části aplikace

Aplikace je logicky členěna do menších celků (složek). Pro pochopení tohoto členění poslouží následující seznam některých složek a popis, co v nich lze nalézt.

- app - kódy pro aplikaci
- app/ApiModule/V1Module/Presenters
  - k ukládání souborů s presentery
  - cesta je zvolena na základě API URL (.../api/v1/... - pro verzování serveru)
  - obsahuje i základní kostru pro presentery (jedná se o abstraktní třídu, ze které ostatní třídy presenterů dědí), kde jim nastavuje například CORS hlavičky
- app/config
  - obsahuje konfiguraci nutnou k fungování aplikace
  - jedná se o soubory typu neon, které jsou logicky členěné do celků patřícím určitým částem aplikace jako je např. soubor services.neon
  - hlavním souborem je config.neon, který ostatní soubory obsahuje (pomocí includes)
- app/Exceptions
  - slouží k vytvoření hlášek o chybách
- app/Models
  - složka s modely aplikace
  - obsahuje entity pro práci s daty
  - pro práci s entitami je zde základní model (abstraktní třída) obsahující metody pro usnadnění práce s entitami
- app/Router
  - obsahuje RouterFactory třídu, která se stará o routování aplikace
- app/Services
  - obsahuje servisy pro práci s databází

  - poskytuje v servise metody pro práci s daty databáze, jejich kontrolu atp.

- app/Utils
  - jedná se o složku s různými třídami a metodami využívanými opakovaně v aplikaci
  - například zde lze nalézt složku Http, kde jsou třídy pro práci s HTTP dotazy

4.3 Databáze

Po rozsáhlé analýze byla zvolena databáze MongoDB. Slouží aplikaci pro uchování dat o pohybových aktivitách. Aktuálně se v databázi nachází devět pohybových aktivit, které slouží k zobrazení pohybové náročnosti potraviny. Do budoucna je v plánu tyto aktivity rozšířit a umožnit uživateli volbu oblíbených aktivit, které mu budou následně zobrazovány.

V databázi jsou uloženy i produkty, jejich kódy a informace o jejich kalorické hodnotě. Ty slouží k výpočtu náročnosti a zobrazení informací o vyhledaném produktu.

4.3.1 Tabulky

Aplikace používá jako databázi MongoDB. Tu je možné dynamicky měnit a přizpůsobit postupně vývoji aplikace.

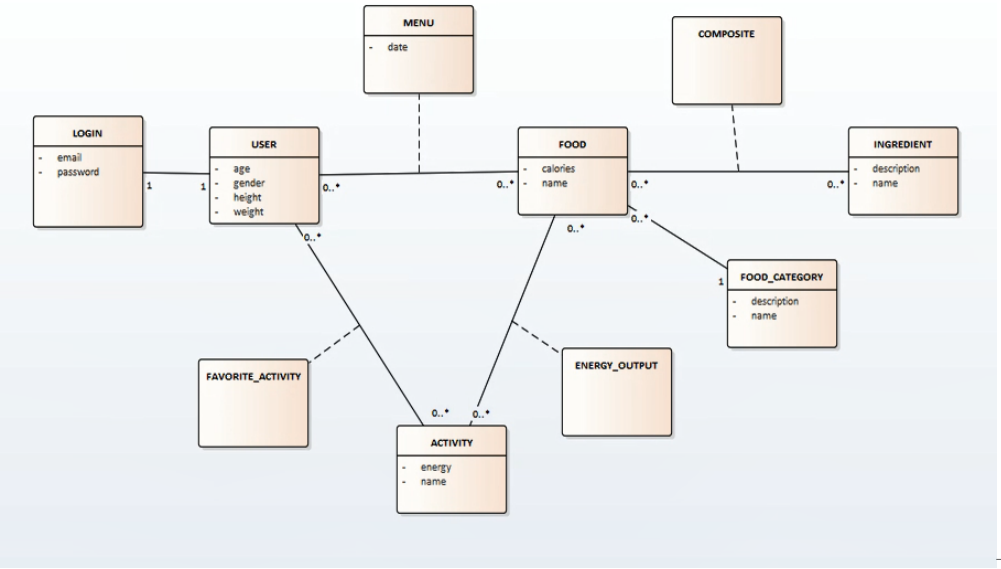
Vzhledem k velkým ambicím projektu a možnosti jeho dalšího vývoje autorka připravila návrh databáze viz. obrázek č. 4.5.

Schéma slouží jako milník do budoucna a v aktuální verzi aplikace je aplikována pouze jeho část.

Pro potraviny v databázi existuje tabulka items (v obrázku znázorněna jako food). V ní se nachází potraviny s jejich vlastnostmi jako jsou name a calories.

Do databáze byla vytvořena i tabulka pro aktivity tj. activities.
V ní se nachází devět vybraných aktivit s jejich kalorickou hodnotou výdejů.

Model, který je odesílán na frontend aplikace je generován na základě těchto dvou tabulek pomocí výpočtů s převodem na časovou náročnost dané aktivity. Obsahuje informaci o produktu jako je jeho název, kalorická hodnota, energetická náročnost (generuje se podle hodnoty kalorií produktu) a seznam aktivit. U každé aktivity je generována hodnota časové náročnosti podle kalorických hodnot produktu a aktivity.



Obrázek 4.5: Návrh databáze

4.4 Fyzická náročnost spálení produktu

Jako další kritérium při vyhodnocení kalorické hodnoty potraviny autorka zvolila tři stupně zátěže při provádění dané sportovní aktivity. Je to zátěž:

- mírná 0-300 Kcal
- střední 300-600 Kcal
- vysoká 600 Kcal a více

Pro příklad brusinková müsli tyčinka (30 g) má přibližnou hodnotu 100 kcal a aplikace vyhodnotí, že k jejímu spálení stačí přibližně půl hodina chůze, jedná se tedy o mírnou zátěž. Naproti tomu balíček chipsů má hodnotu asi 800 kcal, která se rovná asi 4 h běhu, což značí zátěž vysokou. Rozmezí hodnot bylo určeno z vlastní zkušenosti autorky a po konzultaci s profesionálními sportovci, konkrétně hráči týmu FC Hradec Králové a.s. Jak je výše zmíněno výpočty doby trvání daných činností jsou na úrovni amatérských sportovních aktivit, k čemuž bylo při konzultaci přihlédnuto.

4.5 Kanban

Kanban spolu se Scrumem jsou dvěma velmi mocnými nástroji projektového plánování. Umožňují efektivnější práci na projektu prostřednictvím rozvržení

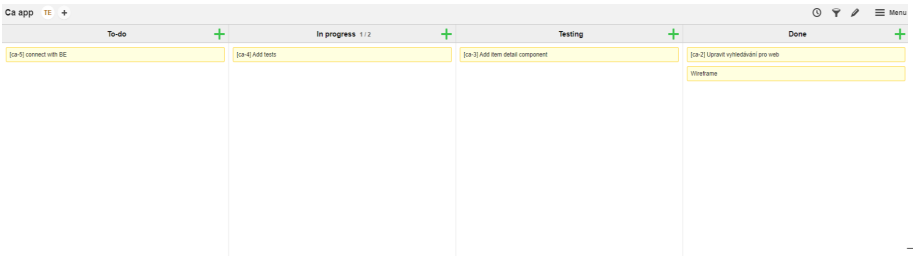
jednotlivých úkolů a časového plánování prací. Podle analýzy v článku [22] je Kanban o trochu lepší než Scrum a byl zvolen k vývoji této aplikace.

Kanban nepatří mezi agilní metodiky, ale často je s nimi spojován, protože se k nim hodí. Lze ho však aplikovat na jakoukoli jinou metodiku jako například vodopád.

Kanban by měl z procesu odstraňovat neefektivitu. Tento bod v průběhu práce splnil. Práce byla rozčleněna a zpracovávána postupně. V průběhu práce byla tato metodika na chvíli přerušena a výsledky opravdu naznačují to, že práce v tomto časovém úseku nebyla dostatečně efektivní. Po opětovném vrácení se k metodice byla práce na aplikaci efektivnější.

Díky této metodice byla práce rozdělena do několika úkolů, které autorka systematicky zpracovávala. To mělo podle metody kanban práci zkvalitnit a dle výsledků tomu tak opravdu bylo. Při správném rozdělení větších částí na menší části a soustředěním se jen na jednu konkrétní věc, byla práce kvalitnější a rychleji odvedená. Mimo jiné je z hlediska vývojáře nedocenitelné to, že se na jednom úkolu pracuje po kratší časový úsek a dříve se může práce označit jako hotová. To slouží k lepší psychice vývojáře. Z velkých úkolů, na kterých je nutné pracovat dlouhou dobu, vznikají většinou problémy a nejsou tak kvalitně odvedené.

Pro ukázku je do této kapitoly zařazen obrázek č. 4.6, kde je znázorněno workflow projektu. Je v něm zaznamenán proces na projektu v průběhu vývoje. Autorka zvolila pro rozepsání svého workflow na projektu čtyři sloupce a to: věci k implementování (to-do), úkoly právě zpracovávané (in progress), sloupec pro testování úkolů (testing), sloupec hotových úkolů (done).



Obrázek 4.6: Ukázka tabule s úkoly

Pracovní proces je následující. Ze sloupce to-do si autorka po zkonzultování se zadavatelem přesune aktuálně implementovanou část do in progress sloupce. Po jeho zpracování přesune úkol do sloupce testing a pokračuje na tomto úkolu. Tentokrát však aplikaci testuje, zda vše funguje a není někde chyba.

Dále je tedy možné, že se úkol může vrátit do sloupce in progres s popisem chyb, které byly nalezeny, nebo bude úkol označen za hotový. Testování při většině úkolů probíhalo tak, že si autorka kontrolovala aplikaci sama.

Při dokončování prací byla tato aplikace poskytnuta několika respondentům a ti zkoušeli aplikaci projít a následně dávali autorce zpětnou vazbu, zda ne-

byly nalezeny nějaké nedostatky.

Často bývá ještě sloupec pro kontrolu kódu (code review). Ten nebyl na projektu aplikován, protože autorka sama sobě sice může poskytovat kontrolu kódu, ale nedává smysl. Kontrola kódu probíhala průběžně a pro naše účely je zahrnuta v sekci testing.

4.6 Jednotkové testy

Jednotkové testy slouží k testování jednotlivých částí kódu, jednotek. Jedná se o automatizované testy, které si píše programátor sám. Je potřeba podotknout, že tyto testy jsou limitovány znalostí autora. Pokud napíše testy špatně, potom nemusí splňovat původní účel, a to najít v kódu případné chyby.

Testy se doporučuje psát z důvodu, který v počátcích jejich psaní není tak evidentní. Pro mnohé jsou testy zdlouhavé a nedávají jim smysl, ale pokud se na ně podíváme z dlouhodobého hlediska, tak co je hodina psaní testů oproti hledání zvláštní chyby za rok. Proto na frontendové aplikaci byly spolu s implementací částí psány průběžné i testy.

Pokud je v průběhu úpravy kódu s některými testy problém, projeví se tato chyba na výsledku testů.

Vzhledem k tomu, že se inicializační záležitosti jednotlivých testů opakují, připravila si autorka práce základní metody pro tvorbu jednotlivých testovacích kontextů. Ty lze nalézt ve složce src/app/global/test-contexts.

V této složce je soubor test-context.spec.ts, který obsahuje pomocné metody pro uvolnění paměti po testech, inicializace komponenty a konfigurace testovacího modulu.

```
1  export function releaseMemory<H>(fixture: ComponentFixture<H>) {
2      if (fixture) {
3          fixture.destroy();
4          fixture.nativeElement.remove();
5      }
6  }
7
8  export function initComponent<H, S>(
9      context: TestComponentContext<H> | TestContainerContext<H, S
           >,
10     componentType: Type<H>,
11     imports: any[] = [],
12     declarations: any[] = [],
13     providers: any[] = [],
14     entryComponents: any[] = []
15 ) {
16     try {
17         configureTestingModule(imports, declarations, providers,
               entryComponents);
18         context.fixture = TestBed.createComponent(componentType);
19         context.component = context.fixture.componentInstance;
20     } catch (e) {
```

```
21          console.error(e);
22      }
23  }
24
25  export function configureTestingModule(
26      imports: any[] = [],
27      declarations: any[] = [],
28      providers: any[] = [],
29      entryComponents: any[] = []
30  ) {
31      TestBed.configureTestingModule({
32          imports: imports,
33          declarations: declarations,
34          providers: providers
35      }).overrideModule(BrowserDynamicTestingModule, {
36          set: {
37              entryComponents: entryComponents
38          }
39      });
40  }
```

Tyto metody jsou využity při vytváření kontextu testovací komponenty umístěného v src/app/global/test-contexts/test-container-context.spec.ts. Zde se provádí inicializace komponenty. Konkrétně se jedná o inicializaci komponenty se stavem tj. obsahuje store a service, které je nutné také inicializovat.

```
1   export interface TestContainerContext<H, S> {
2       fixture: ComponentFixture<H>;
3       component: H;
4       store: MockStore<S>;
5       initialState: any;
6       data: any;
7
8       create(
9           hostType: Type<H>,
10          storeModuleTypes: StoreModuleType[],
11          imports?: any[],
12          declarations?: any[],
13          providers?: any[],
14          entryComponents?: any[]
15      ): void;
16  }
17
18  export function setupContainer(): void {
19      beforeEach(function <H, S>(this: TestContainerContext<H, S>)
            {
20          this.create = (
21              hostType: Type<H>,
22              storeModuleTypes: StoreModuleType[],
23              imports: any[] = [],
24              declarations: any[] = [],
25              providers: any[] = [],
26              entryComponents: any[] = []
27          ) => {
```

```
28              try {
29                  declarations.push(hostType);
30                  providers.push(provideMockStore({initialState:
                        getInitialState(storeModuleTypes)}));
31                  initComponent(this, hostType, imports,
                        declarations, providers, entryComponents);
32                  this.fixture.detectChanges();
33                  this.store = TestBed.get(Store);
34                  this.data = {};
35              } catch (e) {
36                  console.error(e);
37              }
38          };
39      });
40
41      afterEach(function <H, S>(this: TestContainerContext<H, S>) {
42          releaseMemory(this.fixture);
43      });
44  }
45
46  export function getInitialState(storeModuleTypes: StoreModuleType
        [], state = {}) {
47      storeModuleTypes.forEach(storeModuleType => {
48          switch (storeModuleType) {
49              case StoreModuleEnum.APPLICATION_MODULE:
50                  state = {
51                      ...state,
52                      [StoreModuleEnum.APPLICATION_MODULE]:
                            moduleInitialState
53                  };
54                  break;
55              default:
56                  return state;
57          }
58      });
59      return state;
60  }
```

Použití kódu v konkrétní testované komponentě je možné vidět níže. Je umístěno v přiloženém souboru v adresáři s cestou src/app/modules/home/containers/scan/scan.component.spec.ts.

```
describe('ScanComponent', () => {
    let componentContext: TestContainerContext<ScanComponent,
        NutritionItemState>;

    setupContainer();

    beforeEach(function(this: TestContainerContext<ScanComponent,
         NutritionItemState>) {
        this.create(
            ScanComponent,
            [StoreModuleEnum.APPLICATION_MODULE],
            [IonicModule.forRoot(), ZXingScannerModule,
                RouterModule.forRoot([]), ResourceModule.forChild
                (), CommonModule],
            [ScanComponent],
            [DeviceDetectorService, BarcodeScanner]
        );
        componentContext = this;
    });

    it('should create', () => {
        expect(componentContext.component).toBeTruthy();
    });

    it('should dispatch action - scanSuccessOnDesktop', () => {
        checkDispatchAction(
            componentContext.fixture,
            componentContext.store,
            'scanSuccessOnDesktop',
            loadNutritionItem({barcode: '0'}),
            '0'
        );
    });
});
```

V předchozí ukázce kódu je možné zaznamenat další pomocnou metodu checkDispatchAction ze složky src/app/global/test-contexts konkrétně ze souboru test-helper.spec.ts. Ta slouží k otestování úpravy stavu komponenty.

Pro servisy byly vytvořeny následující pomocné metody dohledatelné v souborech test-service-context.spec.ts a test-service.spec.ts v src/app/global/test-contexts.

```
export interface ServiceTestModel<Service, Resource> {
    serviceMethodName: string;
    resourceMethodName: Resource[keyof Resource] extends Function
         ? keyof Resource : never;
    dataForService: any[];
}

function checkServiceCall<Service, Resource>(
    context: TestServiceContext<Service, Resource>,
    serviceMethodName: string,
```

```
10      resourceMethodName: Resource[keyof Resource] extends Function
            ? keyof Resource : never,
11      dataForService: any[]
12  ) {
13      const spy = spyOn(context.resource, resourceMethodName);
14      context.service[serviceMethodName](...dataForService);
15      expect(spy).toHaveBeenCalled();
16  }
17
18  export function checkServiceCalls<Service, Resource>(
19      service: Type<Service>,
20      resource: Type<Resource>,
21      serviceTestModels: ServiceTestModel<Service, Resource>[]
22  ) {
23      describe('test service calls', () => {
24          let context: TestServiceContext<Service, Resource>;
25
26          beforeEach(function(this: TestServiceContext<Service,
                Resource>) {
27              this.create(service, resource);
28              context = this;
29          });
30
31          it('should be created', () => {
32              expect(context.service).toBeTruthy();
33          });
34
35          serviceTestModels.forEach(serviceTestModel => {
36              it(`should call ${serviceTestModel.resourceMethodName
                    } on resource`, () => {
37                  checkServiceCall<Service, Resource>(
38                      context,
39                      serviceTestModel.serviceMethodName,
40                      serviceTestModel.resourceMethodName,
41                      serviceTestModel.dataForService
42                  );
43              });
44          });
45      });
46  }
```

Na ukázce níže je použití těchto metod. Je vidět testování pro jednu GET metodu servisu. V případě potřeby přidání dalšího testu na jinou metodu servisu postačí pouze doplnit do pole objekt jako serviceMethodName: 'get', resourceMethodName: 'getItem', dataForService: [id: '0'] s jinými názvy metod a dat (odpovídajících testovaným metodám).

```
1  describe('FoodNutritionService', () => {
2      setupService();
3
4      checkServiceCalls<FoodNutritionService, FoodNutritionResource
           >(FoodNutritionService, FoodNutritionResource, [
```

```
5           {serviceMethodName: 'get', resourceMethodName: 'getItem',
                dataForService: [{id: '0'}]}
6       ]);
7   });
```

Psaním co nejvíce generických metod (konkrétně testů) je možné usnadnit si v budoucnu psaní celé aplikace, kde lze použít napsaného kódu na více částech. Možná se to teď zdá jako investice velkého množství času, ale do budoucna bude aplikace pokrytá testy a jejich rozšiřovaní nebude tak časově nákladné.

Shrnutí výsledků

Vyzkoušený framework Ionic je velmi dobrým nástrojem pro webové developery, kteří se nechtějí navíc učit specifické jazyky pro nativní aplikace. Framework poskytuje možnost vývoje s využitím frameworků jako je například Angular s jazykem TypeScript, takže není složitý na naučení.

Odlišnosti jsou v používání některých tagů v šablonách, které Ionic poskytuje. Ionic obsahuje mnoho knihoven, např. knihovnu pro skenování barkódu či QR kódů, které lze v aplikaci použít. Tyto knihovny však nejsou většinou podporovány pro webové prohlížeče a jsou optimalizovány pouze pro nativní aplikace. To znamená, že tento framework je vhodné využívat spíše za účel napsání aplikace na mobilní zařízení a pro webovou aplikaci využít jiných možností.

Jelikož se v mnohých případech jedná o knihovny pro mobilní zařízení a není je možné použít pro desktopové zařízení, bylo pro aplikaci nutné použít například knihovnu od Ionicu a zároveň jinou knihovnu pro desktopové zařízení.

Aplikace splnila aktuální požadavky, které na ní byly kladeny. Po naskenování nebo vyhledání barkódu vrací uživateli na výstupu časovou náročnost některých aktivit, takže si dokáže udělat přibližnou představu o tom, jak je potravina kaloricky výživná.

Vzhledem k požadavkům je aplikace velmi jednoduchá a snadná na ovládání či pochopení. Aktuálně nebyl na design aplikace kladen takový důraz, takže využívá především základní styly a již předdefinované komponenty frameworkem.

Aplikace splnila očekávání. Její základ je funkční a je také otestován pomocí jednotkových testů. Struktura je navržena tak, aby byla snadno rozšířitelná a bylo možné na aplikaci navazovat další funkcionality, jak má autorka v plánu.

Závěr

Aktuální verze aplikace však není konečná a bude se nadále vyvíjet. Z tohoto důvodu není aktuálně nikde zveřejněna a bude poskytnuta široké veřejnosti až po dopracování určitých změn. Například se v průběhu práce došlo k závěru, že bude vhodné aplikaci rozdělit na aplikaci pro web a pro mobilní zařízení. Tyto dvě aplikace budou značně odlišné a není důvod je nadále udržovat pohromadě.

Aplikace by měla cílovému uživateli sloužit jako hrubý přehled o tom, kolik kalorií jeho tělo z vybraných potravin příjme a kolik sil ho bude stát dané potraviny spálit. Každý uživatel si z nabídky může vybrat aktivitu jemu nejbližší. Aktivity jsou zvoleny od nejméně náročných pro méně zdatné sportovce – např. chůze, až po pohybově náročné aktivity jako jsou běh, fotbal atd.

Aplikace má seznamovat uživatele s tím, jak kalorické potraviny jsou a umožnit mu náhled na kalorie a potraviny skrze pohybovou aktivitu. Primárním účelem aplikace není sestavování jídelníčků, jejich generování nebo možnost zaznamenávání snědených potravin. Nicméně může svým uživatelům k jejich sestavení pomoci. Především k volbě méně kalorického jídelníčku. To však neznamená, že se bude jednat o zdravější potraviny, pouze budou kaloricky méně náročné.

Cílovou skupinou pro tuto aplikaci může být v podstatě kdokoliv, kdo má zájem se více věnovat svému jídelníčku, tělu a především zdraví. Aplikace může oslovit i uživatele, kteří jsou v tomto oboru znalí, jako např. výživoví poradci či profesionální sportovci, aplikace jim může být velice dobře nápomocná při vykonávání jejich činnosti.

Výsledná aplikace je základním kamenem pro rozsáhlý projekt, na kterém má autorka v plánu pokračovat i nadále. Jak již bylo v některých sekcích zmíněno, do aplikace bude přidána v budoucnu další funkcionalita jako je např. možnost uživatelského účtu a zpřesnění výpočtu délky pohybové aktivity. Další možností, jak aplikaci rozvinout, je přidání možnosti volby oblíbených aktivit atp. Vzhledem k dobře navrženému základu nebude problém na této aplikaci dále stavět.

Literatura

[1] What is CAP theorem? [cit. 2020-02-15]. Dostupné z: http://java-questions.com/cap-theorem.html

[2] Who Uses These Databases? 2017, [cit. 2020-02-15]. Dostupné z: https://blog.panoply.io/couchdb-vs-mongodb

[3] Model–view–controller. [cit. 2020-02-15]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[4] Krukowski, R. A.; Harvey-Berino, J.; Kolodinsky, J.; aj.: Consumers May Not Use or Understand Calorie Labeling in Restaurants. *Journal of the American Dietetic Association*, ročník 106, č. 6, 2006: s. 917 – 920, ISSN 0002-8223, doi:https://doi.org/10.1016/j.jada.2006.03.005. Dostupné z: http://www.sciencedirect.com/science/article/pii/S000282230600304X

[5] van Kleef, E.; van Trijp, H.; Paeps, F.; aj.: Consumer preferences for front-of-pack calories labelling. *Public Health Nutrition*, ročník 11, č. 2, 2008: str. 203–213, doi:10.1017/S1368980007000304.

[6] Levy, L.; Patterson, R. E.; Kristal, A. R.; aj.: How Well Do Consumers Understand Percentage Daily Value on Food Labels? *American Journal of Health Promotion*, ročník 14, č. 3, 2000: s. 157–160, doi:10.4278/0890-1171-14.3.157, [cit. 2019-11-19], https://doi.org/10.4278/0890-1171-14.3.157. Dostupné z: https://doi.org/10.4278/0890-1171-14.3.157

[7] Masic, U.; Christiansen, P.; Boyland, E.: The influence of calorie and physical activity labelling on snack and beverage choices. *Appetite*, ročník 112, 2017: s. 52 – 58, ISSN 0195-6663, doi:https://doi.org/10.1016/j.appet.2017.01.007, [cit. 2019-11-19]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0195666317300284

[8] Strand, B.; Roesler, K.: Calorie Education: A New Plan of Study in Physical Education. *Journal of Physical Education, Recreation & Dance*, ročník 70, č. 9, 1999: s. 46–52, doi:10.1080/07303084.1999.10605968, [cit. 2019-11-19], https://doi.org/10.1080/07303084.1999.10605968. Dostupné z: https://doi.org/10.1080/07303084.1999.10605968

[9] Buchholz, A. C.; Schoeller, D. A.: Is a calorie a calorie? *The American Journal of Clinical Nutrition*, ročník 79, č. 5, 05 2004: s. 899S–906S, ISSN 0002-9165, doi:10.1093/ajcn/79.5.899S, https://academic.oup.com/ajcn/article-pdf/79/5/899S/23713769/znu0050400s899.pdf. Dostupné z: https://doi.org/10.1093/ajcn/79.5.899S

[10] MOUREK, J.: *Fyziologie. Učebnice pro studenty zdravotnických oborů.* Praha: Grada, druhé vydání, 2012, ISBN 978-80-247-3918-2.

[11] Špručková, M.: *Klidový energetický výdej člověka.* Bakalářská práce, Masarykova Univerzita, 2013.

[12] 9 Steps: Choosing a tech stack for your web application [online]. [cit. 2019-11-23]. Dostupné z: https://medium.com/unicorn-supplies/9-steps-how-to-choose-a-technology-stack-for-your-web-application-a6e302398e55

[13] PWA vs Native App and How to Choose Between Them. listopad 2018, [cit. 2019-11-23]. Dostupné z: https://blog.magestore.com/pwa-vs-native-app/

[14] Hume, D.: *Progressive web apps.* Shelter Island, NY: Manning Publications, 2018, ISBN 9781617294587.

[15] Ionic Framework [online]. [cit. 2020-1-30]. Dostupné z: https://ionicframework.com/docs/

[16] Using custom elements [online]. [cit. 2020-1-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements

[17] Using shadow DOM [online]. [cit. 2020-1-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

[18] Databázové systémy [online]. [cit. 2020-01-30]. Dostupné z: http://bech.mzf.cz/accessok.pdf

[19] MVC aplikace & presentery — Nette 2.0 Docs. [online]. All rights reserved. [cit. 01.02.2020]. Dostupné z:. Dostupné z: https://doc.nette.org/cs/2.0/presenters

[20] Ngx-scanner [online]. [cit. 2020-4-10]. Dostupné z: https://github.com/zxing-js/ngx-scanner

[21] NgRx [online]. [cit. 2020-4-20]. Dostupné z: https://ngrx.io/

[22] Lei, H.; Ganjeizadeh, F.; Jayachandran, P. K.; aj.: A statistical analysis of the effects of Scrum and Kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, ročník 43, 2017: s. 59 – 67, ISSN 0736-5845, doi:https://doi.org/10.1016/j.rcim.2015.12.001, special Issue: Extended Papers Selected from FAIM 2014. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0736584515301599

Příloha **A**

Seznam použitých zkratek

ACID Atomic, Consistent, Isolated, Durable

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

ASO App Store Optimalization

CSS Cascading Style Sheets

DRY Don't Repeat Yourself

EAN European Article Number

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IT Informační technologie

J Joul

JSON JavaScript Object Notation

KISS Keep It Simple, Stupid

MVC Model-View-Controller

MVVM Model–view–viewmodel

NoSQL Not Only SQL database

OID Object Identifier

OO Object Oriented

OQL Object Query Language

PHP Hypertext Preprocessor

PWA Progressive Web Apps

REST Representational State Transfer

SEO Search Engine Optimization

SQL Structured Query Language

SSL Secure Sockets Layer

TSL Transport Layer Security

UI User Interface

URL Uniform Resource Locator

XML Extensible Markup Language

Příloha **B**

Obsah elektronické přílohy

- **backend** https://github.com/terez2/ca-database
- **frontend** https://github.com/terez2/ca

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Tereza Kvášová**
Osobní číslo:	**I1700610**
Adresa:	Krapkova 306/6, Hradec Králové – Malšovice, 50009 Hradec Králové 9, Česká republika
Téma práce:	Optimalizace kalorických příjmů a výdajů
Téma práce anglicky:	Optimization of caloric intake and expenditure
Vedoucí práce:	Ing. Pavel Kříž, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat aplikaci s kalorickými tabulkami a s energetickými výdaji pohybových aktivit. Cílem aplikace je poskytnutí uživateli informace o pohybové aktivitě, kterou musí vynaložit pro spálení kalorické hodnoty zkonzumované potraviny. Aplikace usnadní uživateli výpočet množství pohybu a navrhne mu vhodnou aktivitu. Pro tyto účely bude zhotovena webová aplikace. Bude kladen důraz na celkové testování aplikace. Pro řízení vývoje aplikace bude použita iterativní metodika s prvky agilního vývoje (např. využití kanban-boardu).
Osnova:

1. úvod
2. cíl práce
3. rešerš existujících řešení
4. analýza a návrh
5. realizace
6. literatura

Externí téma: Quanti (Vladimír Blažek)

Seznam doporučené literatury:

Hume: D.A. Building Progressive Web Apps: Bringing the Power of Native to the Browse

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: