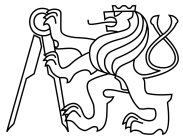
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro sledování nutričních hodnot
Student:	Thanh Cong Nguyen
Vedoucí:	Ing. Josef Gattermayer
Studijní program:	Informatika
Studijní obor:	Web a multimédia
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je vyvinout prototyp moderní webové aplikace, která umožní lepší životosprávu. Práce je zaměřena především na vývoj pomocí nejnovějších technologií.

Pokyny:
- Seznamte se s problematikou sledování, počítání přijatých kalorií/živin.
- Proveďte analýzu webových aplikací, které zprostředkovávají podobnou službu.
- Navrhněte vlastní webovou aplikaci zabývající se tímto tématem. Zohledněte výstupy analýzy.
- Aplikaci realizujte pomocí technologií React, Node.js a MongoDB.
- Vytvořenou webovou aplikaci otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 1. prosince 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Webová aplikace pro sledování nutričních hodnot

Thanh Cong Nguyen

Vedoucí práce: Ing. Josef Gattermayer

16. května 2016

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Josefu Gattermayerovi za odborný dohled a cenné rady při zpracování této práce.

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 Praze dne 16. května 2016 .....

Odkaz na tuto práci

Nguyen, Thanh Cong. *Webová aplikace pro sledování nutričních hodnot.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací moderní webové aplikace pro sledování nutričních hodnot. Součástí je i návrh a implementace REST API a databáze. Vyvíjí se v nejnovějších technologiích včetně React, Node.js a MongoDB.

Klíčová slova sledování nutriční hodnoty, počítání kalorií, React, Node.js, MongoDB, moderní webové aplikace, single page aplikace

Abstract

This bachelor thesis focuses on the analysis, design and implementation of modern web application for tracking nutrition values. Thesis also contais the design and implementation of REST API and database. Web application is developed by the newest technologies including React, Node.js and MongoDB.

Keywords tracking nutrition values, calorie counting, React, Node.js, MongoDB, modern web applications, single page applications

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódů

Úvod

Tělo pro svoji správnou funkčnost vyžaduje potřebné živiny. Nadbytek či naopak nedostatek může mít vliv na správnou funkčnost těla. Přijímáním potravin se správnými živinami udržuje svaly, kosti, orgány a další části těla v dobrém stavu a zmenšuje riziko různých onemocnění. Cílem této bakalářské práce je vyvinutí prototypu moderní webové aplikace, která poskytuje přehled o nutričních hodnotách přijatých potravinách.

Text práce obsahuje vymezení problematiky sledování nutričních hodnot (kalorií a živin). Je provedena analýza existujících webových aplikací podobného zaměření. Následně je navržena a implementována kompletní webová aplikace včetně REST API a databáze. K vývoji této moderní webové aplikace jsou použity nejnovější technologie včetně React, Node.js a MongoDB.

Kapitola 1

Teorie

1.1 Sledování nutričních hodnot

1.1.1 Termíny

1.1.1.1 Nutriční hodnota

Nutrična hodnota potraviny (poživatiny a pochutiny) definuje složení potravin. Důležitými nutričními hodnotami jsou údaje o energii (kalorie) a nutrienty (živiny).

Kalorie *Kalorie* (značka *cal*) je jednotka energie. Z hlediska výživy se obvykle používá její násobek *kilokalorie* (značka *kcal*) a určuje příjem a výdej energie z lidského těla. Příjem energetické hodnoty z potravin a výdej z fyzické aktivity [1].

Kilokalorie se běžně označuje jako kalorie, i když to není technicky správně. Jako další jednotkou energie se používá *joule* (značka *J*).

Přepočty: 1 kcal = 1000 cal = 4184 J = 4,184 kJ

Nutrienty *Nutrienty* se dělí na makronutrienty a mikronutrienty. *Makronutrienty* jsou živiny, které tělo vyžaduje nejvíce. *Mikronutrienty* tělo oproti makronutrientům vyžaduje méně. Těchto nutrientů je velké množství a některé se dělí dále na další typy. Mezi ty nejvýznamnější makronutrienty patří bílkoviny, sacharidy a tuky. Nejvýznamnější mikronutrienty jsou vitamíny a minerály [2].

1.1.2 Sledování kalorií

Kalorie jsou důležité pro výživu především kvůli výraznému vlivu na tělesnou hmotnost. Ta se zvýší ve chvíli, kdy tělo přijme více energie, než jí vydá během každodenní aktivity. Naopak energetický výdej převyšující energetický příjem bude mít za následek úbytek tělesné hmotnosti.

Kalorie jsou obsažené v potravinách v jejich makronutrientech a v alkoholu.

- Bílkoviny - 4 kcal/g
- Sacharidy - 4 kcal/g
- Tuky - 9 kcal/g
- Alkohol - 7 kcal/g

Níže uvedené rovnice a informace se mohou podle jednotlivce lišit. Na přibírání a hubnutí může mít vliv více faktorů jako nutrienty, načasování příjmu potravin, odpočinek apod [3]. Před použitím je doporučena konzultace s odborným lékařem.

1.1.2.1 Celkový energetický výdej

Celkový energetický výdej (zkratka *TEE*) lze spočíst pomocí rovnice 1.1 [4].

$$TEE = RMR \cdot PAL \tag{1.1}$$

kde:

TEE = celkový energetický výdej v kcal/den
RMR = množství energie vydané v klidovém stavu v kcal/den
PAL = koeficient úrovně fyzické aktivity

Klidová míra metabolizmu (zkratka *RMR*) určuje množství, které tělo vydá v klidovém stavu. *RMR* lze spočíst pomocí prediktivních rovnicí. Podle studie [5] American Dietetic Association z roku 2005 je tou nejpřesnější rovnicí *Mifflin-St Jeor* 1.2.

$$RMR = 10m + 6,25h - 5a + s \tag{1.2}$$

kde:

RMR = množství energie vydané v klidovém stavu v kcal/den
m = tělesná váha v kg
h = tělesná výška v cm
a = věk v letech
s = +5 pro muže, -161 pro ženy

Tabulka 1.1: Úroveň fyzické aktivity

Název	**Popis**	**Koeficient**
Sedavá	Skoro žádné pravidelné cvičení.	1,2
Mírná	Intenzivní cvičení po dobu 20 minut jeden až tři dny týdně (běhání, jízda na kole, basketbal, plavání apod.) nebo zaměstnání při kterém se často chodí po dlouhou dobu.	1,375
Střední	Intenzivní cvičení 30 až 60 minut tři až čtyři dny týdně (běhání, jízda na kole, basketbal, plavání apod.).	1,55
Těžká	Intenzivní cvičení po dobu 60 minut a více pět až sedm dní v týdnu nebo namáhavé zaměstnání (práce na stavbě, farmaření apod.).	1,7
Extrémní	Sportovci s několika tréninky denně nebo velmi náročné zaměstnání (práce v dolech, dlouhé hodiny u montážní linky apod.).	1,9

Úroveň fyzické aktivity lze vyjádřit dle tabulky 1.1.

1.1.2.2 Proč sledovat kalorie

Kalorie ovlivňují naší váhu. Obecně platí, že když člověk chce zhubnout, tak začne jíst méně a naopak. Hranice, podle které přijímat více či méně potravin je výše popsána jako celkový energetický výdej. To je důvod proč sledovat nebo-li počítat příjem kalorií.

Dle [6] je doporučeno přibírat nebo hubnout 0,45kg týdně, což odpovídá přijímáním přibližně 500 kcal navíc, respektive 500 kcal méně, než je celkový energetický výdej.

1.1.3 Sledování nutričních hodnot

Tělo pro svoji správnou funkčnost vyžaduje potřebné živiny. Nadbytek či naopak nedostatek může mít vliv na správnou funkčnost těla. Přijímáním potravin se správnými živinami udržuje svaly, kosti, orgány a další části těla v dobrém stavu a zmenšuje riziko různých onemocnění.

Tabulka 1.2: Doporučený denní příjem makronutrientů

Makronutrient	**Procentuální podíl z přijaté energie**		
	Děti, 1-3 let	**Děti, 4-18 let**	**Dospělí**
Tuk	30-40	25-35	20-35
omega-6 nenasycené mastné kyseliny	5-10	5-10	5-10
omega-3 nenasycené mastné kyseliny	0,6-1,2	0,6-1,2	0,6-1,2
Sacharid	45-65	45-65	45-65
Bílkovina	5-20	10-30	10-35

Doporučený denní příjem makronutrientů [7] je vypsán v tabulce 1.2. Doporučený denní příjem mikronutrientů[1]: vitamínů (vitamín A, vitamín C, vitamín D, vitamín E, vitamín K, vitamín B_1 (thiamin), vitamín B_2 (riboflavin), vitamín B_3 (niacin), vitamín B_6, Vitamín B_{11} (kyselina listová), vitamín B_{12}, Vitamín B_5 (kyselina pantothenová), vitamín H a cholin) a minerálů (vápník, chrom, měď, fluór, jód, železu, hořčík, mangan, molybden, fosfor, selen, zinek, draslík, sodík a chlorid) zde není kvůli obsáhlosti vypsán.

1.2 Webová aplikace

Webová aplikace je softwarová aplikace přístupná z webového prohlíže [8].

Rozdíl mezi webovou aplikací a webovou stránkou Webová aplikace se liší od tradiční webové stránky tím, že dovolí uživateli plnit různé úlohy jako např. nakupování produktů, odesílání e-mailů, vyhledávání článků atd. Webové stránky jsou obsahově orientované a jsou vytvářeny pro konzumaci statického obsahu.

1.2.1 Moderní webová aplikace

Moderní webová aplikace klade důraz na interakci s uživatelem, zprostředkování co nejmenší odezvy na akce uživatele. Celkově klade důraz, aby byl uživateli poskytnut zážitek jako při používání nativní aplikace.

[1] `https://fnic.nal.usda.gov/sites/fnic.nal.usda.gov/files/uploads/recommended_intakes_individuals.pdf`

1.2.2 Single page aplikace

Single page aplikace (dále jen *SPA*) je webová aplikace spuštěná ve webovém prohlížeči, která se při používání nemusí znovu načítat [9]. Taková webová aplikace potom celkově působí více jako nativní aplikace [10].

SPA si většinu zdrojů (HTML, CSS, JS) načítá pouze jednou. Po první načtení se již nemusí aktualizovat a běží v prohlížeči uživatele. Požadavky na data posílá jako asynchronní požadavky. Tím je SPA velmi rychlé. Zároveň je snížena zátěž na server.

V dnešní době se SPA vyvíjí zejména v moderních javascriptových frameworcích nebo knihovnách, díky kterým se velká část aplikace vykresluje přímo ve webovém prohlížeči. Mezi ty nejpopulárnější podle uživatelů Githubu [11] patři AngularJS [12], React [13], Backbone [14] a Ember.js [15].

To, že je SPA generované kompletně JavaScriptem je také nevýhodou. Mohou totiž nastat problémy se SEO a indexací vyhledávači. Některé vyhledávače (např. Google [16]) indexování javascriptového obsahu částečně podporují, jiné bohužel ne (např. Seznam [17]). Tento problém lze vyřešit tak, že se obsah SPA vykreslí první ze serveru. Takové řešení ale bývá mnohdy komplikované.

1.2.3 Vývoj moderních webových javascriptových aplikací

Vývoj komplexních aplikací v JavaScriptu je komplikovanější oproti případům, kdy se JavaScript používá např. pouze pro zlepšení interaktivity částí webových aplikací vykreslené ze serveru. Takové aplikace jsou komplikovanější hlavně z důvodu obsáhlosti - mají více kódu, více závislostí, používá se více knihoven atd.

V následujících podsekcích jsou v krátkosti popsány důležité body, které sice nejsou pro vývoj moderních webových aplikací v JavaScriptu nutné, ale velmi doporučené [18][19]. Mohou usnadnit nejen vývoj, ale i následnou správu a případné rozšiřování aplikace.

1.2.3.1 ECMAScript 6

ECMAScript 6 (dálen jen ES6) nebo také znám jako ECMAScript 2015 je poslední verze standardu ECMAScript, která přináší mnoho nových rozšíření od předchozí verze. Tato rozšíření jsou nyní postupně implementovány do javascriptových enginů včetně těch, které pohání webové prohlížeče. Plná podpora ještě ale není, proto je nutné při používání ES6 JavaScriptu kód před spuštěním v prohlížeči přeložit do předchozí ES5 verze.

Výběr několika novinek v ES6:

Proměnné v rámci bloku Deklarace proměnných, které jsou dostupné jen v rámci bloku pomocí `const` a `let`. `const` proměnné jsou jen pro čtení.

Arrow funkce Arrow funkce mají odlišnou syntaxi oproti běžným funkcím a má přístup k `this` a `arguments` z okolního kontextu.

```
function Car() { //běžný zápis funkce
  this.speed = 0;

  setInterval(() => { // zápis arrow funkce
    this.speed += 5; // má přístup k~this
  }, 1000);
}
```

Třídy ES6 umožňuje jednodušší práci s třídami.

```
class Car {
  constructor(name) {
    super(name);
    this.name = name;
  }
}
const myVehicle = new Car('Tank');
```

Moduly Možnost exportovat části kódu v jednom souboru a zpřístupňovat je v druhém souboru.

```
// prvni-soubor.js
export const port = 3000;
// druhy-soubor.js
import {port} from './prvni-soubor.js'
console.log(port);
```

Promises Promises řeší, co se má stát, až se dokončí daná akce (např. asynchronní požadavek).

```
const wait1000 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000)
}).then(()=> {
  console.log('Pockal jsem 1000ms')
})
```

Kompletní seznam ES6 novinek je možné nastudovat v knize Exploring ES6 [20].

1.2.3.2 JavaScript na serveru

JavaScript na serveru, zejména Node.js [21], je vhodný pro moderní webové javascriptové aplikace nejen kvůli tomu, že je dobře rozšiřitelný a velmi výkonný [22], ale zejména kvůli tomu, že je jak serverová tak klientská část v JavaScriptu. Díky tomu je více kódu, které mohou obě části mezi sebou sdílet. Tím je jednodušší implementace tzv. *univerzální javascriptové aplikace* [23], která umožňuje vykreslování klientské části kódu ze serveru při prvním načtení aplikace.

1.2.3.3 Správa balíčků

Existují nástroje pro správu javascriptových balíčků, na kterých je aplikace závislá. Díky nim není nutné jednotlivé balíčky třetích stran vyhledávat a stahovat. Populární nástroje pro správu balíčku jsou npm [24] a Bower [25].

1.2.3.4 Správa závislostí a sestavování kódu

Při manuálním vkládání závislostí a zdrojových kódů aplikace v podobě `<script>` tagů do aplikace, může nastat několik problémů [26]:

- Konflikty v globálním jmenném prostoru.
- Špatné pořadí `<script>` tagů. Na pořadí záleží.
- Ve velkých projektech mohou být seznamy závislostí obsáhlé a obtížné pro správu.

CommonJS [27], RequireJS [28] nebo ES6 moduly umožňují závislosti exportované jako moduly importovat. Když jsou závislosti potřeba, importují se v jednotlivých souborech a tím nebudou zasahovat do globálního jmenného prostoru.

Výsledný soubor (případně soubory) je potom sestaven pomocí nástrojů jako Webpack, Gulp, Grunt. Tyto nástroje umožňují krom javascriptu sestavovat i jiné části potřebné ve webové aplikaci (např. kaskádové styly).

1.2.3.5 Hot reloading

V dnešní době je pomocí nejnovějších technologií možné vyvíjet klientské javascriptové aplikace s pomocí tzv. *hot reloading*. Hot reloading při změně kódu aplikace načte změněný kód a zobrazí změněnou aplikaci bez aktualizace stránky a tím velmi urychlí vývoj.

Kapitola 2

Analýza

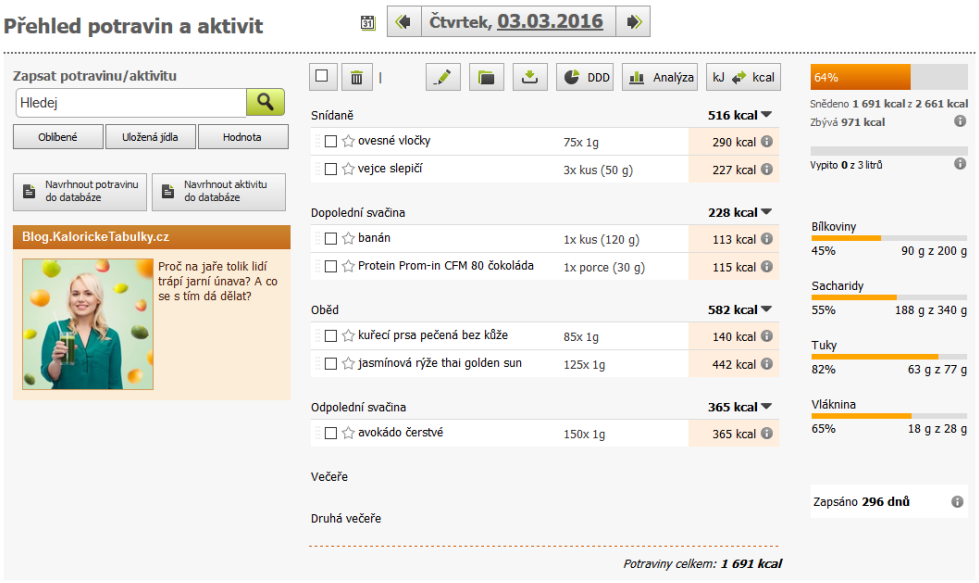
V této kapitole jsou určeny základní požadavky pro použití problematiky popsané v kapitole 1.1 v rámci webové aplikace. Podle těchto základních požadavků jsou vyhledány a analyzovány existující webové aplikace. Po analýze je vypsán kompletní seznam funkčních a nefunkčních požadavků. Součástí kapitoly je i seznam základních technologií, které budou použity k vytvoření webové aplikace.

2.1 Základní požadavky

- Možnost zapisování si přijatých potravin
- Zobrazování sečtených nutričních hodnot přijatých potravin v rámci jednoho dne
- Zobrazování doporučených nutričních hodnot v rámci jednoho dne

2.2 Existující webové aplikace

Existuje mnoho webových aplikací, které umožňují sledovat nutriční hodnoty a splňují výše uvedené požadavky 2.1. Vybrány byly tři podle výsledků vyhledávání klíčových slov problematiky a srovnávacího článku [29]. Jaké technologie aplikace používají, je zjištěno pomocí doplňku do prohlížeče - Wappalyzer [30]. Pokud aplikace obsahují mnoho dalších funkcí, analyzovány jsou pouze části, které se zabývají problematikou této práce. V případě existence placené verze aplikace je analyzována pouze její neplacená verze.



Obrázek 2.1: Screenshot z KalorickéTabulky.cz

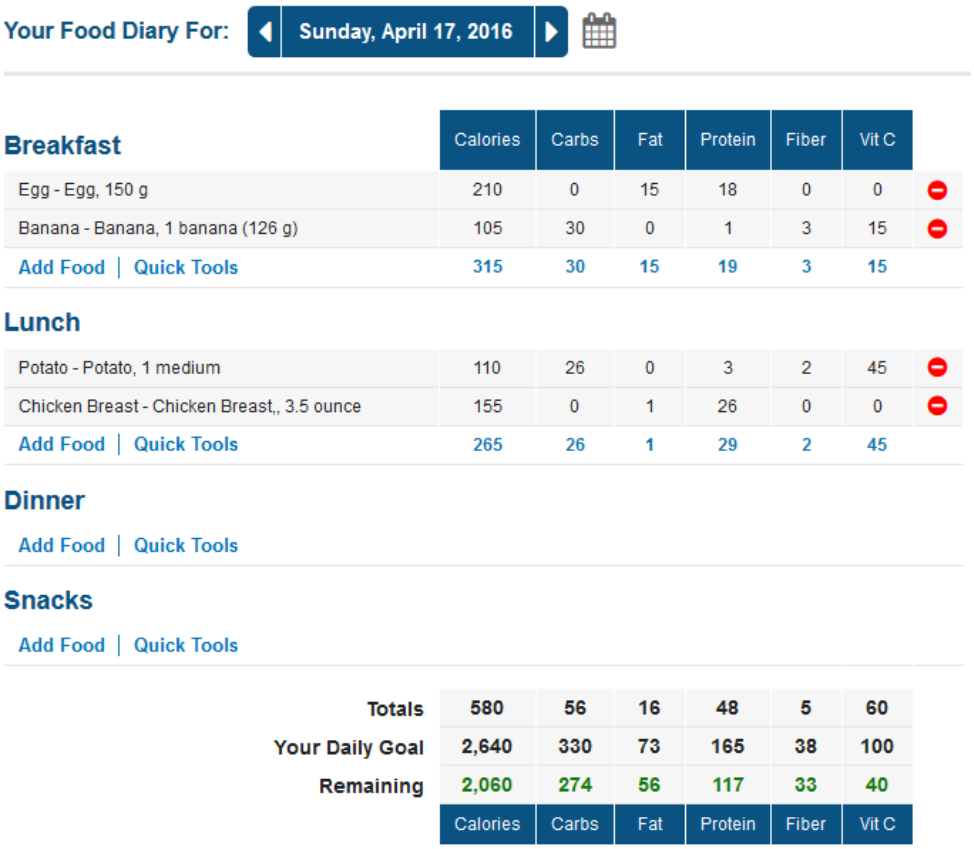
2.2.1 KalorickéTabulky.cz

KalorickéTabulky.cz[2] je česká webová aplikace pro zapisování si svého jídelníčku. Její hlavní část, Jídelníček (viz obr. 2.1), obsahuje chytře rozvržené důležité prvky zobrazené v rámci jednoho dne. Umožňuje rychlé vyhledání potraviny, zapsání do jídelníčku a následné zobrazení nutričních hodnot. V základním zobrazení jsou vidět sečtené hodnoty energetických hodnot a makronutrientů ze zapsaných potravin. Sečtené hodnoty se zobrazují společně s cílovými hodnotami. Zapsané potraviny lze zpětně editovat.

V nastavení lze vyplnit údaje o uživateli, podle kterých se vypočítají doporučené hodnoty, které se potom v jídelníčku zobrazují jako cíle. Doporučené hodnoty u makronutrientů lze přepsat vlastními. Aplikace z nutričních hodnot sleduje pouze kalorie, makronutrienty a několik mikronutrientů.

Aplikace běží na webovém serveru Nginx. Backend je v PHP. Na frontendu se v některých částech používá JavaScriptový framework Prototype, díky kterému jsou některé prvky v klientské části dynamické a není nutné při každé menší akci jako vyhledávání potraviny obnovovat aplikaci. Při větších akcích nebo přechodech mezi sekcemi je ale obnovení nutné.

[2] `http://www.kaloricketabulky.cz`



Obrázek 2.2: Screenshot z MyFitnessPal

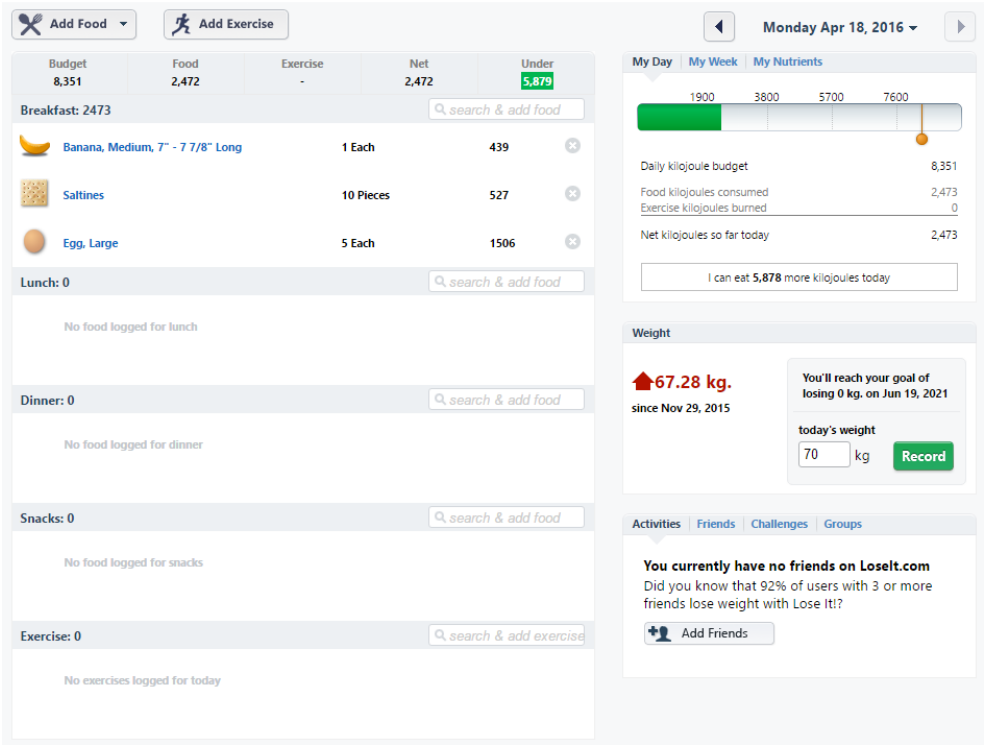
2.2.2 MyFitnessPal

MyFitnessPal[3] je obsáhlá komunitní webová aplikace pro sportovce a fitness nadšence. Aplikace také obsahuje sekci pro zapisování přijatých potravin a sledování jejich nutričních hodnot. Hlavním zobrazením sekce je jeden den a je k dispozici navigace mezi dny (viz obr. 2.2). Nutriční hodnoty z potravin jsou zobrazeny jak pro jednotlivé potraviny, tak i sečtené a doporučené hodnoty.

V nastavení je možné vyplnit údaje, které se použijí ke spočtení doporučených nutričních hodnot. Doporučené hodnoty lze přepsat vlastními. Je možné si vybrat, které nutriční hodnoty jsou sledovány. Počet sledovaných nutričních hodnot je ale limitován pouze na šest.

MyFitnessPal funguje na webovém serveru Nginx. Backend je v Ruby (framework Ruby on Rails). Na frontendu je JavaScriptová knihovna jQuery. Ja-

[3] `http://www.myfitnesspal.com`



Obrázek 2.3: Screenshot z Lose It!

vaScriptových prvků je na stránce minimum a většina akcí vyžaduje kompletní přechody mezi stránkami.

2.2.3 Lose It!

Lose It![4] je komunitní webová aplikace zabývající se fitness a výživou s hlavním zaměřením na hubnutí. Hlavní sekce je místo na zapisování potravin a aktivit (viz obr. 2.2). V základním zobrazení jsou krom potravin vidět informace o váze, přijatých kalorií a doporučené hodnoty v rámci dne. Lze přepnout zobrazení přijatých kalorií za týden. Informace o přijatých nutrientech lze také zobrazit přepnutím.

V nastavení lze vyplnit údaje potřebné k spočtení doporučené hodnoty kalorického příjmu. Doporučené hodnoty pro nutrienty se nezobrazují. Je možné si vybrat, které nutrienty jsou v hlavním zobrazení vidět. Na výběr jsou ale pouze makronutrienty a několik mikronutrientů.

Lose It! funguje na webovém serveru Apache. Backend je naprogramován v Javě. Na frontendu je nasazen Google Web Toolkit, díky kterému se obsah

[4] `http://www.loseit.com`

stránky vykresluje přes JavaScript a celkově webová aplikace funguje jako SPA.

2.2.4 Shrnutí

Z analýzy lze o existujících řešení usoudit, že mají sice kompletní systém zapisování potravin, ale část s počítáním a doporučováním nutričních hodnot bývá nekompletní. Aplikace sčítají nutriční hodnoty přijatých potravin a spočítají doporučené hodnoty. Pracují ale pouze se základními nutričními hodnotami - makronutrienty a několika mikronutrienty.

První dvě analyzované webové aplikace jsou implementovány jako běžné webové aplikace vykreslované ze serveru, kde při každé větší akci je nutná aktualizace. Lose It! je implementováno jako SPA, které není univerzální. Což ale z hlediska SEO není problém, protože je webová aplikace přístupná pouze pro přihlášené uživatele.

2.3 Požadavky

Požadavky na webovou aplikaci této práce vyplývají ze zadání, vymezení problematiky (viz kapitola 1.1) a analýzy existujících řešení (viz kapitola 2.2).

2.3.1 Funkční požadavky

1. Potraviny
   Potraviny je sekce, kde uživatel může spravovat potraviny a jejich nutriční hodnoty.

   1.1. Možnost vyhledávat a zobrazit existující potraviny a jejich nutričních hodnot dle výběru uživatele.

   1.2. Možnost upravit název a nutriční hodnoty existujících potravin.

   1.3. Možnost vytvořit novou potravinu.

2. Jídelníček
   Jídelníček je hlavní sekce aplikace, kde uživatel může spravovat potraviny, které přijal nebo se chystá přijmout a zobrazit jejich informace v rámci jednoho dne.

   2.1. Možnost vyhledávání, přidávání potravin a jejich hmotnosti do jídelníčku v rámci jednoho dne.

   2.2. Zobrazení přidaných potravin v rámci jednoho dne.

   2.3. Možnost úpravy hmotnosti přidaných potravin.

   2.4. Možnost smazání přidaných potravin.

   2.5. Možnost navigace mezi dny a zobrazení jídelníčku z jiného dne.

2.6. Zobrazení sečtených nutričních hodnot potravin v rámci jednoho dne. Zobrazují se nutriční hodnoty, které si uživatel vybral v sekci Předvolby (viz funkční požadavek 3).

2.7. Zobrazení doporučené hodnoty k zobrazeným součtům nutričních hodnot.

3. Předvolby
V sekci předvolby si uživatel vyplní údaje, které jsou potřebné k spočtení doporučených nutričních hodnot. Také si může nastavit nutriční hodnoty, které chce sledovat a případně přepsat jejich doporučené hodnoty.

3.1. Možnost vyplnění údajů potřebných pro spočtení celkového energetického výdeje dle kapitoly 1.1.2.1.

3.2. Možnost přidání nutriční hodnoty ke sledování. Nutriční hodnoty se vybírají ze seznamu dostupných hodnot dle kapitoly 1.1.3.

3.3. Spočtení doporučených nutričních hodnot.

3.4. Možnost přepsání doporučených hodnot vlastními hodnotami.

3.5. Při vyplňování nových předvoleb, musí být starší předvolby stále uloženy a zobrazovat se k patřičnému dni v jídelníčku.

4. Statistiky
V sekci statistiky si uživatel může zobrazit vizualizaci dat z přijatých potravin a doporučených hodnot.

4.1. Možnost zobrazení grafu kilokalorického příjmu rozděleného do jednotlivých makronutrientů a doporučené hodnoty kilokalorií. Na ose X jsou jednotlivé dny a na ose Y kilokalorické hodnoty. Graf se skládá ze dvou částí:

- skládaný sloupcový graf s daty z přijatých makronutrientů přepočítaných do kilokalorií
- spojnicový graf s daty z doporučených hodnot kilokalorií

4.2. Možnost zobrazení grafu každé nutriční hodnoty z přijatých potravin a její doporučené hodnoty. Na ose X jsou jednotlivé dny a na ose Y hodnoty podle typu nutriční hodnoty. Graf se skládá ze dvou částí:

- sloupcový graf s daty z přijaté nutriční hodnoty
- spojnicový graf s daty z doporučených hodnot

5. Možnost registrace a přihlašování uživatelů.

2.3.2 Nefunkční požadavky

1. Použití Node.js.
2. Použití React.
3. Použití MongoDB.
4. Vyvinout aplikaci jako univerzální SPA.

2.3.3 Uživatelské role

Webová aplikace rozlišuje dvě uživatelské role:

- Nepřihlášený uživatel - Má přístup pouze k sekci potraviny bez možnosti vytvoření nové potraviny a upravení existustujících potravin. Má přístup k přihlášení a registraci.
- Přihlášený uživatel - Nemá přístup k přihlášení a registraci. Ke všem ostatním funkcím (viz kapitola 2.3.1) přístup má.

2.4 Použité technologie

2.4.1 Node.js

Node.js je běhové prostředí pro vývoj vysoce škálovatelných serverových a internetových aplikací, především webových serverů. Základem Node.js je javascriptový engine V8 z Google Chrome, díky kterému je vykonávaný kód velmi rychlý. Node.js má událostmi řízenou architekturu a asynchronní I/O pro optimální propustnost a škálovatelnost webových aplikací s mnoha I/O operacemi. Existuje mnoho aplikací do Node.js, které se vyvíjí zejména v JavaScriptu [31].

2.4.2 npm

npm je balíčkovací systém pro Node.js, který umožňuje přes příkazový řádek spravovat závislosti aplikace [32]. Umožňuje také instalovat Node.js aplikace, které jsou dostupné v npm repozitáři[5].

2.4.3 Express

Express [33] je minimalistický a flexibilní Node.js pro tvorbu webových aplikací. Používá se hlavně jako webový server. Požadavky, které Express přijímá prochází tzv. *middleware* funkcemi, které rozhodují co se s požadavkem děje a co se vrátí na výstup. Můžeme říci, že Express aplikace je řada po sobě jdoucích middleware funkcí.

[5] `https://www.npmjs.com/`

2.4.4 MongoDB

MongoDB [34] je NoSQL dokumentová databáze. Data jsou v databázi uložena jako dokumenty a jsou shlukovány do kolekcí. Data dokumentů se skládají z dvojic klíč-hodnota a zapisují se stejně jako při zápisu JSON. MongoDB ukládá JSON dokumenty v binární reprezentaci BSON. BSON reprezentace rozšiřuje JSON o datové typy jako např. int, long.

Díky BSON formátu je možné modelovat bohatě strukturované dokumenty 1. Dokumenty mohou obsahovat pole, objekty nebo i struktury z jiných dokumentů. Vztahy mezi dokumenty lze modelovat vložením dat jednoho dokumentu do druhého nebo referencí.

```
{
   name:"Pink Floyd",
   genres:[
      "progressive rock",
      "psychedelic rock",
      "art rock"
   ],
   albums:[
      {
         name:"The Dark Side of the Moon",
         year:1973
      },
      {
         name:"Wish You Were Here",
         year:1975
      }
   ]
}
```

Ukázka kódu 1: Ukázka struktury MongoDB dokumentu

MongoDB má bohatý dotazovací jazyk. Příklad dotazů v ukázce 2.

```
db.bands.update({name: "Pink Floyd"}, {$push: {genres: "pop"}})
db.bands.find({genres: "progressive rock"})
```

Ukázka kódu 2: Ukázky dotazovacího jazyka MongoDB

MongoDB umožňuje i indexaci dat dokumentů pro zlepšení výkonu při dotazování. Pole `_id` v dokumentech je automaticky indexováno.

2.4.5 React

React je javascriptová knihovna pro tvorbu uživatelského rozhrání.

Hlavním stavebním kamenem Reactu jsou interaktivní, reusable komponenty s vlastní logikou a stavem. React aplikace se skládají ze stromu komponent, které si postupně předávají data (tzv. *props*), s kterými dále pracují a případně posílají na výstup přes `render` metodu. Komponenty mohou volat metody jiných komponent, které jsou zpřístupněny přes props.

React využívá koncept virtuálního DOMu. React si vytvoří v paměti vlastní virtuální DOM. Toto umožňuje psát jednodušší a přehlednější kód, který by normálně překresloval celý DOM. Při změně v komponentách React projde virtuální DOM, porovná rozdíly a podle nich aktualizuje skutečný DOM v prohlížeči. Virtuální DOM lze také vykreslit na straně serveru a zajistit tím univerzální webovou aplikaci.

Části Reactu komponent se mohou psát v JSX syntaxi, která je podobná XML. JSX umožňuje psát HTML tagy a React komponenty jako HTML tagy, které se transformují do javascriptových funkcí.

```
class Lorem extends React.Component {
  render() {
    return <p>Lorem ipsum</p>;
  }
}

class Hello extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello World</h1>
        <Lorem />
      </div>
    );
  }
}
```

Ukázka kódu 3: React komponenty s JSX syntaxí

2.4.6 Babel

Babel je kompilátor pro přeložení ES6, ES7 a JSX JavaScriptu do ES5 JavaScriptu. Výsledný JavaScript kód je podporován běžnými prohlížeči.

JavaScript kód v ES6:

```
[1,2,3].map(n => n + 1);
```

JavaScript kód přeložený do ES5:

```
[1,2,3].map(function(n) {
  return n + 1;
});
```

2.4.7 Webpack

Webpack je nástroj, který na základě závislostí z několika vstupních souborů vygeneruje statické výstupní soubory, které mohou být načteny prohlížeči. Pracuje s JavaScript soubory, kaskádovými styly, obrázky a mnoha dalšími typy souborů [35].

Webpack projde vstupní soubory, jejich závislosti a předá je loaderům, které je zpracují (např. babel-loader [36] přeloží vstupní JavaScript ES6 kód do ES5 kódu). Výstup z loaderů se předá do pluginů a ty na něm dělají finální úpravy (např. UglifyJsPlugin [37] minimalizuje JavaScript kód).

Webpack umožňuje sledovat vstupní soubory, jejich změny a automaticky aktualizovat výstupní soubory. Pomocí Hot Module Replacement [38] je možné implementovat i hot reloading.

2.4.8 JSON Web Tokens

JSON Web Tokens (JWT) [39] je otevřený standard, který definuje způsob, jakým bezpečně přenášet informace mezi stranami v rámci webové aplikace. JWT mají malou velikost, a proto je možné zasílat v rámci URL, POST nebo hlavičce požadavku. Také v sobě mohou obsahovat data (např. informace o uživateli). Díky těmto vlastnostem se JWT hodí zejména pro ověřování přihlášeného uživatele.

2.4.9 REST

Representational State Transfer je styl architektury pro návrh distribuovaného prostředí [40]. REST umožňuje přistupovat a manipulovat s daty na určitém místě pomocí HTTP metod:

- GET - získání
- POST - vytvoření

- PUT - úprava
- DELETE - smazání

REST API zpřístupní komunikaci mezi klientem a serverem pomocí HTTP metod.

Kapitola 3

Návrh

3.1 Architektura webové aplikace

3.1.1 Serverová část

Na serverové části poběží Express webový server, který bude obsluhovat požadavky klientské části.

Serverová část bude mít s klientskou částí sdílený kód React aplikace. Při prvním načtení lze ze serverové části vykreslit základní zobrazení React aplikace.

Na serverové části bude i databázový server MongoDB. Webový server bude napojen na databázový server a umožňovat práci s daty pro React aplikaci v klientské i serverové části přes REST API.

Při vývoji poběží na serverové části webpack dev server, který se bude starat o sestavování výstupních souborů.

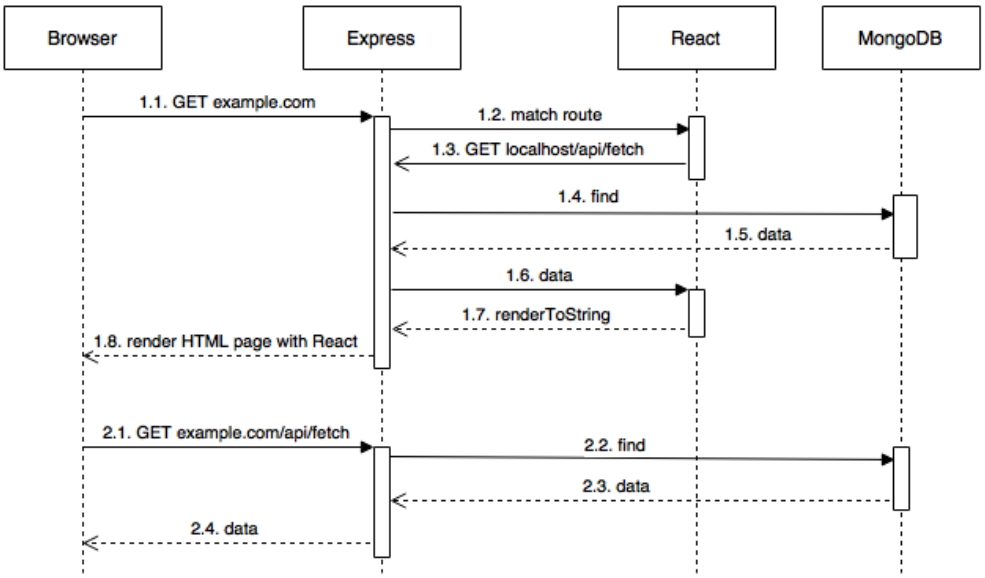
3.1.2 Klientská část

Na klientské části (ve webovém prohlížeči uživatele) poběží React aplikace načtená z JavaScript souboru poskytnutým ze serveru při prvním načtení. Po prvním načtení může klientská část fungovat sama o sobě, až na případy, kdy bude potřebovat pracovat s daty z databáze.

3.1.3 Komunikace mezi serverovou a klientskou částí

Klientská část komunikuje se serverovou částí v těchto dvou případech:

- Požadavek na první načtení webové aplikace
- Požadavek na zobrazení, vytvoření, úpravu nebo smazání dat v databázi



Obrázek 3.1: Komunikace mezi serverovou a klientskou částí

Na sekvenčním diagramu (viz. obrázek 3.1) je ukázka komunikace mezi serverovou a klientskou částí. V ukázce serverová část funguje na doméně `example.com` a jsou znázorněny dva případy komunikace:

1. Požadavek na první načtení webové aplikace.
    1.1. Prohlížeč zašle požadavek na načtení adresy `example.com`.
    1.2. Express webový server spojí cestu s částí React aplikace, která vyžaduje načtení dat z databáze.
    1.3. React komponenta zašle GET požadavek na `localhost/api/fetch` v rámci serveru.
    1.4. REST API na webovém serveru zpracuje požadavek a odešle požadavek na nalezení dat v databázi.
    1.5. Databáze nalezne a vrátí data.
    1.6. Data jsou předány do React aplikace.
    1.7. React aplikace vykreslí výsledek jako HTML řetězec.
    1.8. Webový server vrátí prohlížeči HTML stránku. Součástí je i nalinkovaný JavaScript soubor pro načtení React aplikace v prohlížeči.
2. V prohlížeči je již načtená React aplikace, která potřebuje data z databáze (požadavek na vytvoření, úpravu a smazání dat probíhá stejným způsobem).

2.1. React aplikace v prohlížeči zasílá požadavek `GET` na `example.com-/api/fetch`.

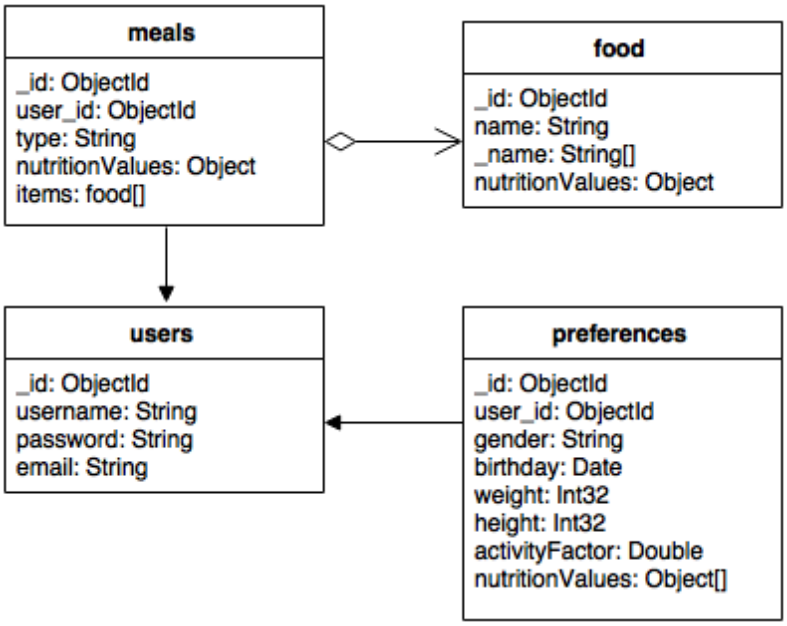
2.2. REST API na webovém serveru zpracuje požadavek a odešle požadavek na nalezení dat v databázi.

2.3. Databáze nalezne a vrátí data.

2.4. REST API vrátí React aplikaci v prohlížeči data.

3.2 Datový model

V této sekci je navržena struktura dat, s nimiž webová aplikace pracuje. Data jsou ukládána v dokumentech MongoDB databáze. Při návrhu schématu databáze je nutno brát v potaz možnosti bohaté struktury MongoDB dokumentů a možnosti řešení vztahů mezi dokumenty. Diagram z obrázku 3.2 znázorňuje, které kolekce v databázi budou, strukturu jejich dokumentů a vztahy mezi nimi.



Obrázek 3.2: Datový model

3.2.1 Dokumenty kolekce food

Kolekce `food` obsahuje dokumenty znázorňující jednotlivé potraviny. Potraviny se budou ve webové aplikaci hodně vyhledávat, proto je nutné vyřešit indexování jejich názvu. Je nutno brát v potaz vyhledávání jak s diakritikou,

tak bez diakritiky. Jestli potravina obsahuje v názvu více slov, musí se myslet i na vyhledávání mezi jednotlivými slovy. Dokumenty obsahují tato data:

- `_id` (typ `ObjectId`) je id dokumentu.
- `name` (typ `String`) je název jídla.
- `_name` (typ `Array`) je upravený název pro vyhledávání. Prvky `_name` jsou řetězce jednotlivých slov z názvu bez diakritiky. MongoDB umožňuje vyhledávání v dokumentech na základě regulárních výrazů. Nejlépe ale mohou dotazy s regulární výrazem využívat index v případě, pokud to je tzv. *prefixový výraz*[6] [41]. `_name` je tedy typu pole, aby mohl být prefixový výraz aplikován na každé slovo z názvu. Příklad vyhledávacího dotazu je vidět v ukázce kodu 5.
- `nutritionValues` (typ `Object`) je objekt s atributy znázorňující jednotlivé nutriční hodnoty potraviny.

```
{
    _id:ObjectId("5707dff0fd6537275fa4707c"),
    name:"salám jemný z~kuřecího masa",
    _name:[
        "salam",
        "jemny",
        "z",
        "kureciho",
        "masa"
    ],
    nutritionValues: {
        proteins:201600,
        carbs:15500,
        fats:175900,
        kcal:1850000
    }
}
```

Ukázka kódu 4: Ukázka struktury dokumentu kolekce food

[6] Prefixový výraz je regulární výraz, který začíná symbolem `^`, po kterým následují jednoduché symboly (např. `^abc`). Při vyhledávacím dotazu s regulárním výrazem `^abc` na políčko dokumentu, by se dokument nalezl, pokud by obsah jeho políčka začínal řetězcem *abc*.

```
db.food.find({_name: {$all: [/^salam/, /^kureci/]}})
```

Ukázka kódu 5: Ukázka vyhledávacího dotazu na dokument kolekce food

3.2.2 Dokumenty kolekce users

Kolekce `users` obsahuje dokumenty znázorňující zaregistrované uživatele. Dokumenty obsahují základní data o uživateli:

- `_id` (typ `ObjectId`) je id dokumentu.
- `password` (typ `String`) je zašifrovaný otisk hesla.
- `email` (typ `String`) je e-mail.

3.2.3 Dokumenty kolekce meals

Dokumenty kolekce `meals` znázorňují jídla zapsaná v uživatelově jídelníčku podle typu. Dokumenty obsahují tato data:

- `_id` (typ `ObjectId`) slouží k identifikaci a zároveň jako držitel data vytvoření. Je automaticky indexované a lze tedy podle něho rychle vyhledávat dokumenty z konkrétních dnů[7].
- `user_id` (typ `ObjectId`) je reference na uživatele, který tento dokument vytvořil.
- `type` (typ `String`) je typ jídla.
- `items` (typ `Array`) je pole objektů s potravinami z kolekce `food` přijaté v rámci jídla, které znázorňuje tento dokument. Vztah s kolekcí `food` je řešen vložením dokumentů z `food` do tohoto pole. Objekty tohoto pole navíc obsahují informace o počtu a hmotnosti přijaté potraviny.
- `nutritionValues` (typ `Object`) je objekt s vlastnostmi znázorňují sečtené nutriční hodnoty potravin z pole `items`.

3.2.4 Dokumenty kolekce preferences

Dokumenty kolekce `preferences` znázorňují předvolby - informace o nutričních hodnotách, které uživatel chce sledovat. Dále jsou v nich obsaženy údaje o uživatelovi, které jsou potřeba ke spočtení doporučených hodnot. Dokumenty obsahují tato data:

[7] První čtyři bajty ObjectId obsahují záznam o čase [42].

- `_id` (typ `ObjectId`) slouží k identifikaci a zároveň jako držitel data vytvoření (viz Id u dokumentů kolekce `meals` v sekci 3.2.3).
- `user_id` (typ `ObjectId`) je reference na uživatele, který tento dokument vytvořil.
- Údaje o uživatelově osobě - Několik políček jako např. *váha* `weight` (typ `Number`), *datum narození* `birthday` (typ `Date`).
- `nutritionValues` (typ `Array`) je pole objektů nutričních hodnot, které uživatel chce sledovat.

3.3 REST API

React aplikace přistupuje k dokumentům z databáze přes REST API. REST API je součástí webového serveru.

3.3.1 Ověření uživatele

Některé zdroje definované REST API jsou k dispozici pouze pro přihlášené uživatele, proto je nutné, aby se uživatel při každém takovém požadavku identifikoval.

3.3.2 Stavové kódy a formáty odpovědí

React aplikace pozná úspěšnost požadavku podle stavového kódu odpovědi. Využívané stavové kódy jsou následující:

- 200 OK
- 201 Created
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

Při úspěšném požadavku je vrácená odpověď ve formátu JSON s požadovaným objektem. Při neúspěšném požadavku je vrácená odpověď jako řetězec s chybovou zprávou.

3.3.3 Zdroje

3.3.3.1 S ověřením

K následujícím zdrojům je potřeba ověření uživatele.

- `/api/food`
  - `POST` - Vytvoří nový dokument v kolekci `food` s daty zaslanými v těle HTTP požadavku.
- `/api/food/{id}`, kde `{id}` je ObjectId, podle kterého se najde konkrétní dokument.
  - `PUT` - Upraví dokument kolekce `food`. V dokumentu se upraví data, která jsou zaslána v těle HTTP požadavku.
- `/api/food-list/{name}`, kde {name} je název potraviny bez diakritika, podle kterého se naleznou dokumenty.
  - `GET` - Nalezne dokumenty kolekce `food` podle názvu v parametru.
- `/api/meals`
  - `POST` - Vytvoří nový dokument v kolekci `meals` s daty zaslanými v těle HTTP požadavku.
- `/api/meals?from={from}&to={to}`, kde `{from}` a `{to}`, kde jsou ObjectId.
  - `GET` - Nalezne dokumenty kolekce `meals` v časovém rozmezí daným parametry a patřící ověřenému uživateli.
- `/api/meals/{id}`, kde `{id}` je ObjectId, podle kterého se najde konkrétní dokument.
  - `PUT` - Upraví dokument kolekce `meals` patřící ověřenému uživateli. V dokumentu upraví pole `items`: přidá novou potravinu/edituje/smaže existující. Podle změn se také přepočítají sečtené nutriční hodnoty v `nutritionValues`. Pole se upravuje na základě dat zaslanými v HTTP požadavku.
- `/api/preferences`
  - `GET` - Nalezne posledně vytvořený dokument kolekce. `preferences` patřící ověřenému uživateli.
  - `POST` - Vytvoří nový dokument v kolekci `preferences` s daty zaslanými v těle HTTP požadavku.

- `/api/preferences?from={from}&to={to}`, kde `{from}` a `{to}`, kde jsou ObjectId.
  - `GET` - Nalezne dokumenty kolekce `preferences` v časovém rozmezí daným parametry a patřící ověřenému uživateli.
- `/api/preferences/{id}`, kde `{id}` je ObjectId, podle kterého se najde konkrétní dokument.
  - `PUT` - Upraví dokument kolekce `preferences` patřící ověřenému uživateli. V dokumentu se upraví data, která jsou zaslána v těle HTTP požadavku.
- `/api/user`
  - `GET` - Nalezne dokument ověřeného uživatele z kolekce `users`.

3.3.3.2 Bez ověření

Následující zdroje fungují bez ověření uživatele.

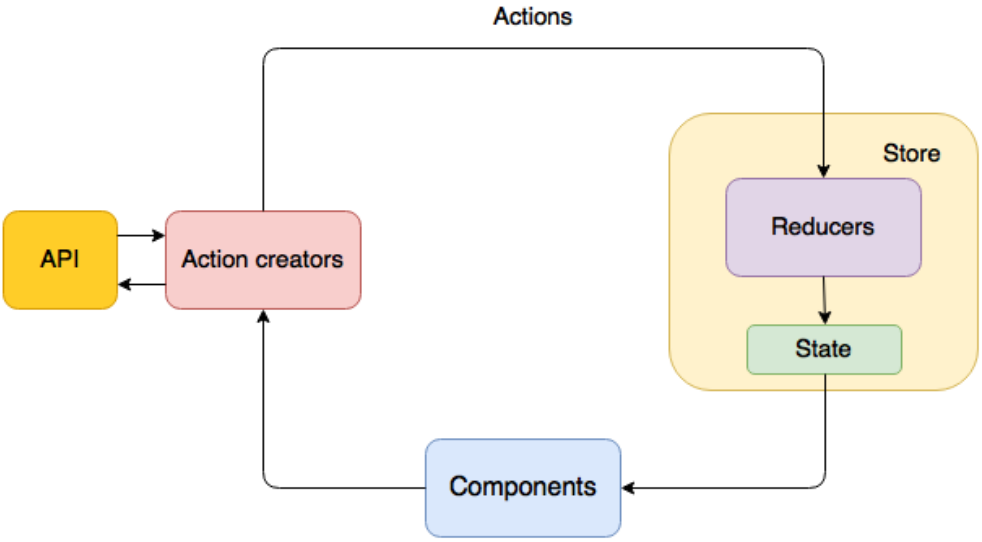
- `/api/food/{id}`, kde `{id}` je ObjectId, podle kterého se najde konkrétní dokument.
  - `GET` - Nalezne dokument podle id.
- `/api/food-list/{name}`, kde `{name}` je název potraviny bez diakritiky.
  - `GET` - Nalezne dokumenenty potraviny z kolekce `food` podle názvu.
- `/api/signup`
  - `POST` - Vytvoří nový dokument v kolekci `users` s daty zaslanými v těle HTTP požadavku.
- `/api/signin`
  - `POST` - Ověří správnout přihlašovacích údajů zaslaných v HTTP požadavku, nalezne podle nich dokument z kolekce `users` a vytvoří nový JWT token.

3.4 React aplikace

3.4.1 Architektura

React aplikace používá návrhový vzor Flux [43], konkrétně jeho implementaci Redux [44], kde funguje jednosměrný tok dat.

Redux drží stav celé aplikace jako strom objektů v jednom místě (v tzv. *store*). Stav v store je jen pro čtení a nedá se přímo měnit. Pro nový stav



Obrázek 3.3: Architektura React aplikace

se musí odeslat tzv. *akce* (objekt popisující, co se stalo). Akce se předává do čistých funkcí[8] tzv. *reducer* funkce, které vezmou předchozí stav, akci a podle nich vrátí nový transformovaný stav. Díky tomu, že reducer funkce nemění předchozí stav, je možné se k němu vrátit nebo ho efektivně porovnávat s novým.

React komponenty, které potřebují data ze stavu store se musí ke store *připojit* a naslouchat pro nový stav. Pokaždé když bude mít store nový stav, připojená komponenta ho získá jako součást props. Komponenty si potom mohou tento stav (neplést s vnitřním stavem komponenty) předávat dále přes props. Není tedy nutné, aby byla každá komponenta připojena ke store.

Požadavek na nový stav začíná v komponentě. Připojená komponenta má k dispozici tzv. *dispatch* funkci, která umožňuje odeslat akci směrem k reducer funkcím. Pokud komponenta potřebuje vytvořit nový stav s daty z databáze, musí se před odesláním akce dotázat na REST API. Kvůli tomu se používají tzv. *action creator* funkce, které se odešlou uvnitř dispatch funkce. Action creator funkce odešlou asynchronní požadavek na REST API, získají data a odešlou akci s těmito daty. Akce následně putuje do reducer funkcí, kde lze vytvořit nový stav na základě dat z databáze. V action creator funkcích se mohou krom požadavků na REST API dělat i vedlejší efekty, které nejsou možné v reducer funkcích.

[8]Čistá funkce je taková funkce, která vrací vždy stejný výsledek podle jejích argumentů. Funkce nemění proměnné mimo její rámec a nedělá žádné vedlejší efekty. [45]

3.4.2 Sekce a jejich cesty

Sekce webové aplikace dostupné pro uživatele, jsou výsledkem vykreslování komponent React aplikace. Sekce jsou dostupné na různých cestách a k těmto cestám jsou mapovány konkrétní React komponenty (viz tabulka 3.1).

Tabulka 3.1: Sekce webové aplikace - jejich cesty a komponenty

Cesta v URL	Název sekce	Komponenta
/	Jídelníček	Diary
/jidelnicek/{datum}	Jídelníček	Diary
/potraviny	Potraviny	FoodList
/potravina/{id}	Detail potraviny	FoodDetail
/potravina/{id}/editace	Úprava potraviny	FoodDetailEdit
/pridat-potravinu	Přidat potravinu	FoodDetailEdit
/statistika	Statistika	Statistics
/predvolby	Předvolby	Preferences
/prihlaseni	Přihlášení	SignInForm
/registrace	Registrace	SignUpForm

`{datum}` je parametr pro datum ve formátu `YYYY-MM-DD` (např. 2016-01-30)
`{id}` je parametr pro ObjectId potraviny

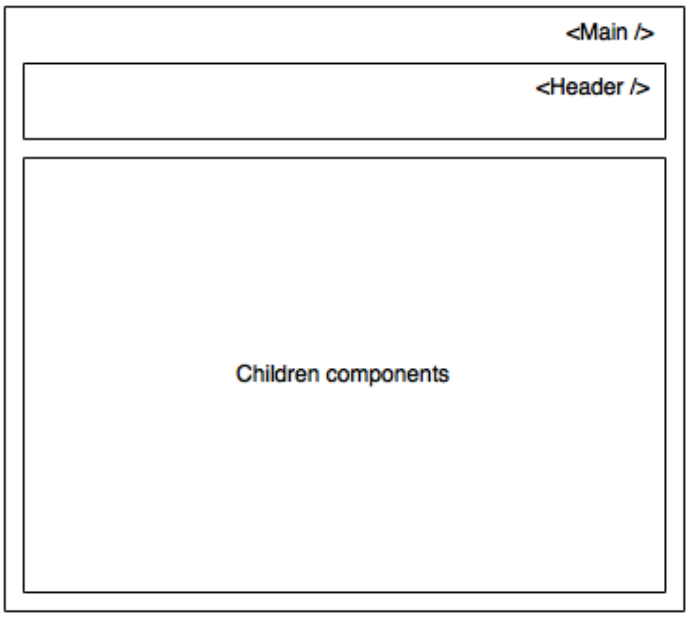
3.4.3 Komponenty

React aplikace se skládá ze stromu komponent. Každá komponenta má na starosti vykreslování části aplikace a její interakci s uživatelem. V následujících podkapitolách jsou navrženy jednotlivé sekce jako komponenty a základní prvky těchto sekcí také jako komponenty.

Ke komponentám, které se skládají z více komponent jsou přiloženy obrázky. Tyto obrázky znázorňují rozmístění komponent a slouží rovnou jako wireframy. V obrázcích jsou komponenty vyznačeny JSX syntaxí (např. `<Header />`).

3.4.3.1 Hlavní komponenta

`Main` je hlavní komponenta, která obaluje všechny sekce a jejich komponenty. Obsahuje nahoře `Header` pro navigaci mezi jednotlivými sekcemi (viz obr. 3.4). Všechny ostatní sekce jsou vnořené v `Main` a vykreslují se pod `Header`.

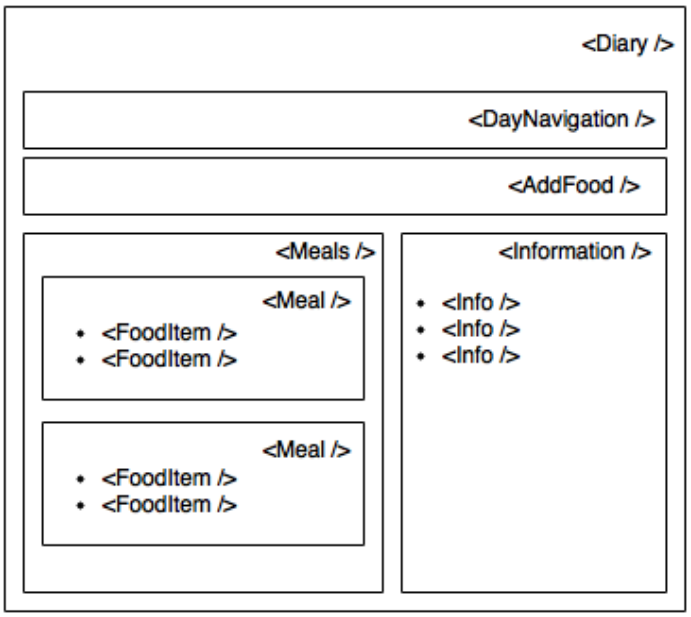


Obrázek 3.4: Hlavní komponenta

3.4.3.2 Jídelníček

`Diary` je komponenta sekce jídelníček. Obsahuje tyto komponenty (viz obr. 3.5):

- `DayNavigation` obsahuje navigaci mezi dny v jídelníčku.
- `AddFood` obsahuje formulář pro přidání potraviny do jídelníčku.
- `Meals`
  - Seznam komponent `Meal` dle typů jídel.
    - ⋆ Seznam komponent `FoodItem`. Každá komponenta znázorňuje potravinu, která je obsažená v jídle (v rodičovské komponentě `Meal`).
- `Information`
  - Seznam komponent `Info` dle typů nutričních hodnot. Každá komponenta znázorňuje součet nutriční hodnoty obsažené v potravinách v jídlech právě zobrazeného dne jídelníčku. Dále zobrazuje doporučenou hodnotu nutriční hodnoty.



Obrázek 3.5: Komponenta sekce jídelníček

3.4.3.3 Potraviny

`FoodList` je komponenta sekce potraviny. Obsahuje tyto komponenty (viz obr. 3.6):

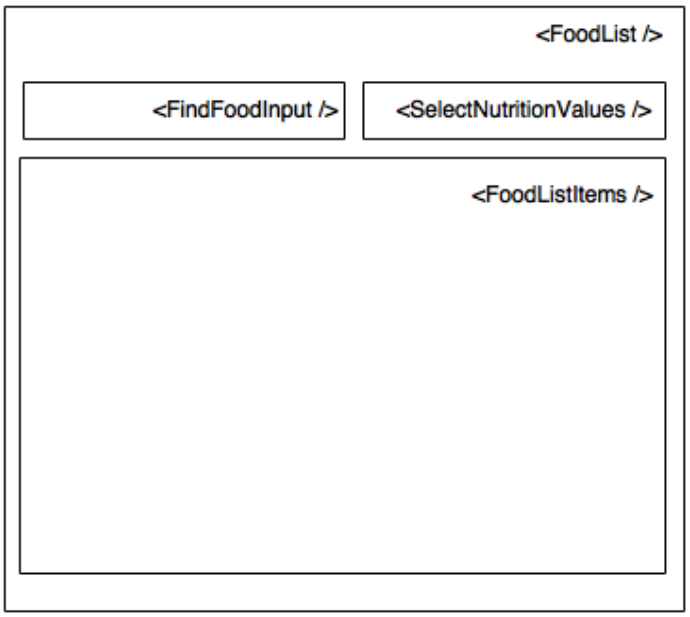
- `FindFoodInput` obsahuje pole pro nalezení potravin podle názvu.
- `SelectNutritionValues` obsahuje pole pro výběr nutričních hodnot k zobrazení.
- `FoodListItems` obsahuje seznam nalezených potravin s vybranými nutričními hodnotami.

3.4.3.4 Detail potraviny

`FoodDetail` je komponenta sekce detail potraviny. Obsahuje informace o nutričních hodnotách konkrétní potraviny.

3.4.3.5 Upravit nebo přidat potravinu

`FoodDetailEdit` je komponenta sekce upravit potravinu a přidat potravinu. Obsahuje komponentu `FoodDetailForm` s formulářem pro editaci dat potraviny.



Obrázek 3.6: Komponenta sekce potraviny

3.4.3.6 Statistika

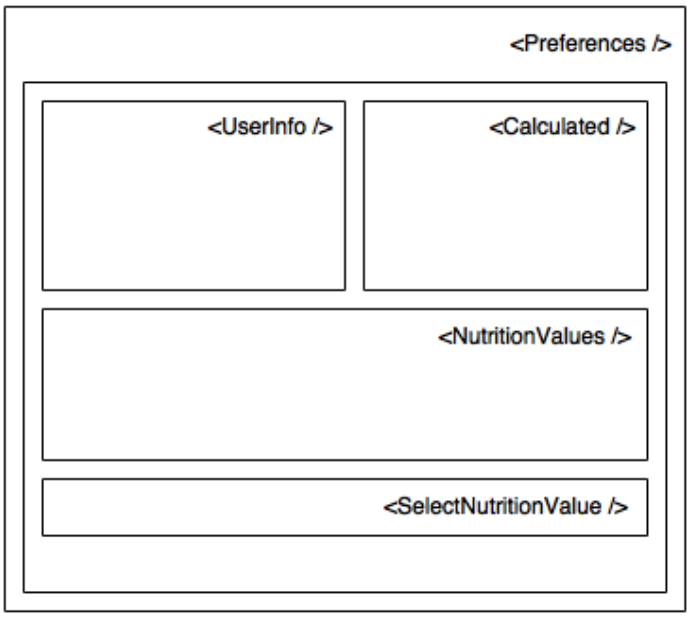
`Statistics` je komponenta sekce statistiky. Obsahuje tyto komponenty (viz obr. 3.7):

- `MainGraph` nebo `NVGraph`. `MainGraph` je pro zobrazení grafu přijatých kilokalorií rozdělených do makronutrientů a doporučených hodnot kilokalorií. `NVGraph` je pro zobrazení grafu vybrané přijaté nutriční hodnoty a jejích doporučených hodnot.

3.4.3.7 Předvolby

`Preferences` je komponenta sekce předvolby. Obsahuje tyto komponenty:

- `PreferencesForm` obsahuje formulář pro vyplnění údajů o uživateli a nastavení doporučených nutričních hodnot.
  - `UserInfo` obsahuje pole formuláře pro vyplnění údajů o uživateli.
  - `Calculated` obsahuje informace o spočtených hodnotách.
  - `NutritionValues` obsahuje seznam nutričních hodnot, které chce uživatel sledovat a jejich nastavení.
  - `SelectNutritionValue` obsahuje pole pro vybrání a přidání nutriční hodnoty ke sledování.



Obrázek 3.7: Komponenta sekce předvolby

3.4.3.8 Přihlášení

`SignInForm` je komponenta sekce přihlášení. Komponenta obsahuje formulář pro přihlášení uživatele do aplikace.

3.4.3.9 Registrace

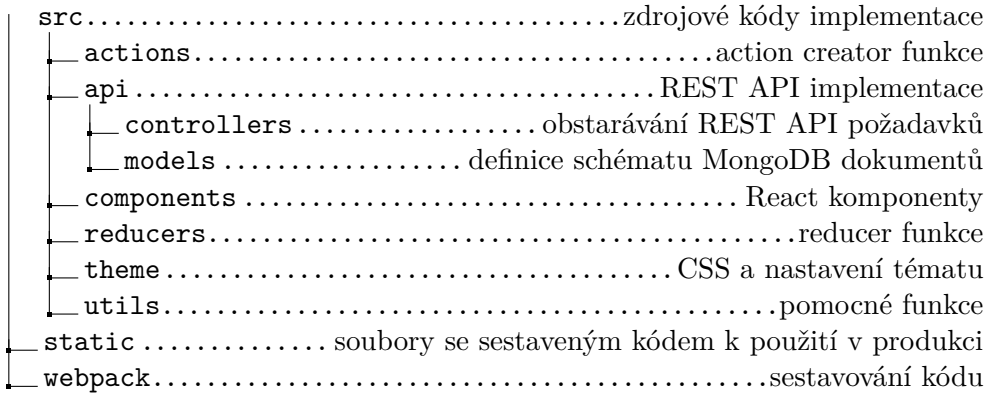
`SignUpForm` je komponenta sekce registrace. Komponenta obsahuje formulář pro registrace nového uživatele.

Kapitola 4

Implementace

4.1 Adresářová struktura

Jednotlivé části webové aplikace se pro přehlednost shlukují do adresářů. Všechny důležité funkce a objekty se exportují jako ES6 moduly. Exportované moduly lze naimportovat a použít v rámci celé webové aplikace. Na obrázku 4.1 je znázorněná adresářová struktura s krátkými popisky jednotlivých adresářů.



Obrázek 4.1: Adresářová struktura implementace

4.2 Webový server

Webový server je implementován v Node.js frameworku, Expressu. Přijímá požadavky a ty prochází middleware funkcemi. Implementovány jsou čtyři typy middleware funkcí.

Application-level middleware Přiřadí middleware funkci přímo k instanci webového serveru. V této aplikaci se používá např. na vykreslení React aplikace při prvním načtení nebo na kontrolu JWT tokenu uživatele.

V ukázce kódu 6 je middleware funkce, která vezme token z cookies z objektu požadavku - `req` a zkontroluje jeho platnost. Pokud je neplatný, smaže token z cookies. Na konci se zavolá funkce `next` a tím se zavolá další middleware funkce v řadě. Další middleware funkce dostane už upravený `req` objekt, který obsahuje token, který je v pořádku nebo neobsahuje žádný token.

```
app.use((req, res, next) => {
  const token = req.cookies.token;
  if(token) {
    try {
      jwt.verify(token, config.secret);
    } catch(err) {
      res.clearCookie('token', { path: '/' });
    }
  }
  return next();
});
```

Ukázka kódu 6: Middleware pro kontrolu JWT tokenu uživatele

Router-level middleware Tyto middleware funkce se přiřazují k instanci routeru `express.Router()`. Implementována je instance routeru pro REST API (viz kapitola 4.2.1), která je na cestě `/api`. Middleware funkce přiřazené k této instanci se potom vztahují pouze na požadavky vedoucí na cesty `/api/*`.

Third-party middleware V této aplikaci se používá několik middleware funkcí třetích stran, které jsou dostupné z npm. Např. `bodyParser` [46] a `cookieParser` [47] rozparsují tělo respektive cookies z požadavku do javascriptového objektu a zpřístupní ho dalším middleware funkcím v `req` objektu.

Error-handling middleware Middleware funkce pro zpracování chyb je definovaná s čtyřmi parametry (ostatní middleware funkce mají maximálně jen tři parametry). Pokud se z jiných middleware funkcí zavolá `next()` s jakýmkoli argumentem, přeskočí se všechny ostatní middleware funkce a zavolá se middleware funkce pro zpracování chyb.

4.2.1 REST API

REST API je implementováno na instanci routeru definované na cestě `/api`. Na další cesty tohoto routeru a jejich HTTP požadavky(viz návrh REST API v kapitole 3.3.3) se přiřazují middleware funkce. Při příchodu požadavku k těmto middleware funkcím již požadavek prošel všemi důležitými middleware funkcemi, které patřičně zpracovaly jeho data. V `req` objektu těchto middleware funkcí jsou již připravena data ve správném formátu. Tyto middleware funkce potom provádějí operace s MongoDB databází pomocí *mongoose* [48].

V ukázce kódu 7 je ve dvou souborech znázorněno načtení potraviny podle jejího `id`. V `/src/api/index.js` je přiřazení middleware funkce `getById` k cestě `/api/food/{id}`. V `/src/api/food.js` je `getById` middleware funkce, která nalezne dokument z kolekce `food` podle `id` z parametru a vrátí ho jako JSON objekt. `Food` v prvním řádku je schéma dokumentu kolekce `food`.

Dokumenty všech kolekcí mají definované své schéma (viz kapitola 3.2), s kterými mongoose pracuje. Zápis do databáze se validuje podle těchto schémat.

```
// /src/api/food.js:
import Food from '../models/food'

export function getById(req, res, next) {
  Food.findOne({
    _id: req.params.id
  },
  (err, food) => {
    if (err) return next(err);
    res.json(food ? food : {});
  });
};
// /src/api/index.js:
import express from 'express'
import * as FoodController from './controllers/food'

const router = express.Router();
router.route('/food/:id')
  .get(FoodController.getById)

export default router
```

Ukázka kódu 7: Ukázka implementace REST API

4.3 Vývojový server

Ještě předtím než se začne implementovat React aplikace je nutné zprovoznit vývojový server. Webpack dev server běží také na Express serveru, ale naslouchá na jiném portu. V ukázce kódu 8 je ukázka jeho konfigurace. Webpack dev server vezme vstupní soubor `/src/main.js`, kde začíná React aplikace, projde všechny naimportované moduly a předá je loaderům, které kód přetransformují a uloží do výstupního souboru `/dist/bundle.js`, odkud ho může prohlížeč přečíst. Při každé změně zdrojového kódu se výstupní soubor aktualizuje a pomocí *react-hot-loader* [49] se načte ve webovém prohlížeči bez aktualizace stránky.

```
{
  entry: [
    'react-hot-loader/patch',
    'webpack-dev-server/client?http://' + config.host + ':' + port,
    'webpack/hot/only-dev-server',
    './src/main',
  ],
  output: {
    path: path.join(__dirname, '..', 'dist'),
    filename: 'bundle.js',
    publicPath: 'http://' + config.host + ':' + port + '/dist/'
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin(),
  ],
  module: {
    loaders: [
      {
        test: /\.jsx?/,
        loader: 'babel',
        include: path.join(__dirname, '..', 'src'),
      }
    ]
  },
}
```

Ukázka kódu 8: Ukázka konfigurace Webpacku při vývoji

4.4 React aplikace

Podle návrhu architektury 3.4.1 je třeba v React aplikaci implementovat čtyři části:

- Action creator funkce
- Reducer funkce
- Routování (cesty)
- Komponenty

4.4.1 Action creator funkce

4.4.1.1 Synchronní

Synchronní action creator funkce ihned po zavolání změní stav React aplikace.

- `clearFoodList` Smaže seznam načtených potravin a odešle akci typu `CLEAR_FOOD_LIST`.
- `changeFoodListFilters` Změní filtry v seznamu potravin a odešle akci typu `CHANGE_FOOD_LIST_FILTERS`.
- `clearFoodDetail` Smaže detail načtené potraviny a odešle akci typu `CLEAR_FOOD_DETAIL`.
- `logout` Smaže v cookies JWT token přihlášeného uživatele a odešle akci typu `LOGOUT_SUCCESS`.

4.4.1.2 Asynchronní

Asynchronní action creator funkce odesílají asynchronní požadavky na REST API, kde jsou důležité dva momenty:

1. Odeslání požadavku
2. Získání odpovědi nebo dosažení časového limitu požadavku

V obou momentech je v některých případech třeba změnit stav React aplikace tím. V asynchronních action creator funkcích se odesílají tyto akce:

- Akce, která informuje reducer funkce o začátku požadavku a tím v komponentách zobrazí informaci o načítání. Akce jsou typu `X_REQUEST`.
- Akce, která informuje reducer funkce o úspěšném přijetí dat a tím v komponentách skryje informaci o načítání a zobrazí přijatá data. Akce jsou typu `X_SUCCESS`.

- Akce, která informuje reducer funkce o neúspěšném přijetí dat a tím v komponentách skryje informaci o načítání a zobrazí chybovou zprávu. Akce jsou typu `X_FAILURE`.

Seznam action creator funkcí v React aplikaci:

- `fetchFoodList` Odešle `GET` požadavek na `/api/food-list/{name}` pro nalezení potravin podle názvu. Odesílají se tyto akce:
  - `FETCH_FOOD_LIST_REQUEST`
  - `FETCH_FOOD_LIST_SUCCESS`
  - `FETCH_FOOD_LIST_FAILURE`
- `fetchFoodDetail` Odešle `GET` požadavek na `/api/food-list/{id}` pro nalezení potravin podle `id`. Odesílají se tyto akce:
  - `FETCH_FOOD_DETAIL_REQUEST`
  - `FETCH_FOOD_DETAIL_SUCCESS`
  - `FETCH_FOOD_DETAIL_FAILURE`
- `updateFood` Odešle `POST` požadavek na `/api/food` nebo `PUT` požadavek na `/api/food/{id}` s daty pro vytvoření nebo upravení upravení potraviny.
  - `UPDATE_FOOD_REQUEST`
  - `UPDATE_FOOD_SUCCESS`
  - `UPDATE_FOOD_FAILURE`
- `fetchUser` - Odešle `GET` požadavek na `/api/user` pro načtení přihlášeného uživatele. Odesílají se tyto akce:
  - `FETCH_USER_SUCCESS`
  - `FETCH_USER_FAILURE`
- `signin` - Odešle `POST` požadavek na `/api/signin` s daty pro přihlášení uživatele. Odesílají se tyto akce:
  - `SIGNIN_SUCCESS`
- `signup` - Odešle `POST` požadavek na `/api/signup` s daty pro vytvoření nového uživatele. Odesílají se tyto akce:
  - `SIGNUP_SUCCESS`
- `fetchMeals` - Odešle `GET` požadavek na `/api/meals?from={from}&to={to}` pro načtení jídel z období `{from}` do `{to}` přihlášeného uživatele. Odesílají se tyto akce:

  - FETCH_MEALS_REQUEST
  - FETCH_MEALS_SUCCESS
  - FETCH_MEALS_FAILURE

- addMeal - Odešle POST požadavek na /api/meals s daty pro vytvoření nového jídla přihlášeného uživatele. Odesílají se tyto akce:
  - ADD_MEAL_REQUEST
  - ADD_MEAL_SUCCESS
  - ADD_MEAL_FAILURE

- updateMeal - Odešle PUT požadavek na /api/meals/{id} s daty pro přidání potraviny do konkrétní jídla dle {id} přihlášeného uživatele. Odesílají se tyto akce:
  - UPDATE_MEAL_REQUEST
  - UPDATE_MEAL_SUCCESS
  - UPDATE_MEAL_FAILURE

- updateMealFood - Odešle PUT požadavek na /api/meals/{id} s daty pro upravení potraviny v konkrétním jídle dle {id} přihlášeného uživatele. Odesílají se tyto akce:
  - UPDATE_MEAL_FOOD_REQUEST
  - UPDATE_MEAL_FOOD_SUCCESS
  - UPDATE_MEAL_FOOD_FAILURE

- removeMealFood - Odešle PUT požadavek na /api/meals/{id} s daty pro smazání potraviny v konkrétním jídle dle {id} přihlášeného uživatele. Odesílají se tyto akce:
  - REMOVE_MEAL_FOOD_REQUEST
  - REMOVE_MEAL_FOOD_SUCCESS
  - REMOVE_MEAL_FOOD_FAILURE

- fetchPreferences - Odešle GET požadavek na /api/preferences pro načtení poslední předvolby přihlášeného uživatele. Odesílají se tyto akce:
  - FETCH_PREFERENCES_SUCCESS
  - FETCH_PREFERENCES_FAILURE

- updatePreferences - Odešle POST požadavek na /api/preferences nebo PUT požadavek na /api/preferences/{id} s daty pro vytvoření nebo upravení upravení předvoleb. Odesílají se tyto akce:

  - `UPDATE_PREFERENCES_SUCCESS`

- `fetchMealsForStats` - Odešle `GET` požadavek na `/api/meals?from={from}&to={to}` pro načtení jídel z období `{from}` do `{to}` přihlášeného uživatele. Odesílají se tyto akce:

  - `FETCH_MEALS_FOR_STATS_REQUEST`
  - `FETCH_MEALS_FOR_STATS_FAILURE`

- `fetchPreferencesForStats` - Odešle `GET` požadavek na `/api/preferences?from={from}&to={to}` pro načtení předvoleb z období `{from}` do `{to}` přihlášeného uživatele. Odesílají se tyto akce:

  - `FETCH_PREFERENCES_FOR_STATS_REQUEST`
  - `FETCH_PREFERENCES_FOR_STATS_SUCCESS`
  - `FETCH_PREFERENCES_FOR_STATS_FAILURE`

Asynchronní požadavky Pro asynchronní požadavky na REST API se používá FETCH API, které je dostupné v moderních prohlížečích [50]. Pro podporu ve starších prohlížečích a pro použití na serverové straně se používá isomorphic-fetch [51].

V první části ukázky kódu 9 je vidět, jak asynchronní async creator funkce odesílá fetch požadavek a s užitím promises zpracovává odpověď. Při zpracování odpovědi kontroluje status a podle toho dispatchuje (odesílá) akce směrem k reducer funkcím. Asynchronních action creator funkcí je mnoho a každá musí takto zpracovávat odpověď. Tímto vzniká velké množství duplicitního kódu.

V Redux aplikaci je možné vytvářet middleware funkce, které podobně jako u Expressu zpracovávají a posílají dál určitý objekt. V tomhle případě se jedná o akce. S implementovanou middleware funkcí se v action creator funkci posílá speciální typ akce, kterou middleware odchytí a zavolá požadavek včetně zpracování odpovědi. Taková action creator funkce je vidět v druhé části ukázky kódu 9.

```
// bez použití middleware funkce:
function fetchFoodDetail({ id }) => {
  return (dispatch) => {
    dispatch({ type: FETCH_FOOD_DETAIL_REQUEST })
    return fetch('/api/food/' + id)
      .then((response) => {
        const json = response.json();
        if (response.status >= 200
            && response.status < 300) return json;
        else return json.then(Promise.reject.bind(Promise));
      })
      .then(
        (result) => dispatch({
          type: FETCH_FOOD_DETAIL_SUCCESS
          result
        }),
        (error) => {
          dispatch({
            type: FETCH_FOOD_DETAIL_FAILURE
            error
          });
          return Promise.reject(error);
        }
      )
  }
}
// s~použitím middleware funkce:
function fetchFoodDetail({ id }) {
  return (dispatch) => {
    return dispatch({
      types: [
        FETCH_FOOD_DETAIL_REQUEST,
        FETCH_FOOD_DETAIL_SUCCESS,
        FETCH_FOOD_DETAIL_FAILURE
      ],
      promise: fetch('/api/food/' + id)
    });
  }
}
```

Ukázka kódu 9: Ukázka implementace action creator funkce

4.4.2 Reducer funkce

V React aplikaci je několik reducer funkcí a každá z nich má na starosti vytváření části stavu.

- `food` reducer vytváří část stavu, kde jsou uloženy právě načtené potraviny v seznamu potravin, filtry, informaci o načítání a detail právě zobrazené potraviny. Zpracovává tyto akce:
  - `FETCH_FOOD_LIST_REQUEST`
  - `FETCH_FOOD_LIST_SUCCESS`
  - `FETCH_FOOD_LIST_FAILURE`
  - `CLEAR_FOOD_LIST`
  - `CHANGE_FOOD_LIST_FILTERS`
  - `FETCH_FOOD_DETAIL_SUCCESS`
  - `UPDATE_FOOD_SUCCESS`
  - `CLEAR_FOOD_DETAIL`
- `auth` reducer vytváří část stavu, kde jsou uloženy data přihlášeného uživatele. Zpracovává tyto akce:
  - `FETCH_USER_SUCCESS`
  - `FETCH_USER_FAILURE`
  - `SIGNIN_SUCCESS`
  - `LOGOUT_SUCCESS`
- `meals` reducer vytváří část stavu, kde jsou uložené jídla právě zobrazeného dne v jídelníčku a informaci o načítání jídel. Zpracovává tyto akce:
  - `FETCH_MEALS_REQUEST`
  - `FETCH_MEALS_SUCCESS`
  - `FETCH_MEALS_FAILURE`
  - `ADD_MEAL_REQUEST`
  - `ADD_MEAL_SUCCESS`
  - `ADD_MEAL_FAILURE`
  - `UPDATE_MEAL_REQUEST`
  - `UPDATE_MEAL_SUCCESS`
  - `UPDATE_MEAL_FAILURE`

  - `UPDATE_MEAL_FOOD_REQUEST`
  - `UPDATE_MEAL_FOOD_SUCCESS`
  - `UPDATE_MEAL_FOOD_FAILURE`
  - `REMOVE_MEAL_FOOD_REQUEST`
  - `REMOVE_MEAL_FOOD_SUCCESS`
  - `REMOVE_MEAL_FOOD_FAILURE`

- `nutritionValues` reducer vytváří část stavu, kde jsou uloženy nutriční hodnoty jídel právě zobrazeného dne v jídelníčku a jejich doporučené hodnoty z předvoleb. Zpracovává tyto akce:
  - `FETCH_PREFERENCES_SUCCESS`
  - `ADD_MEAL_SUCCESS`
  - `UPDATE_MEAL_SUCCESS`
  - `UPDATE_MEAL_FOOD_SUCCESS`
  - `REMOVE_MEAL_FOOD_SUCCESS`

- `preferences` reducer vytváří část stavu, kde jsou uloženy právě zobrazené předvolby. Zpracovává tyto akce:
  - `FETCH_PREFERENCES_SUCCESS`
  - `FETCH_PREFERENCES_FAILURE`
  - `UPDATE_PREFERENCES_SUCCESS`

- `statistics` reducer vytváří část stavu, kde jsou uloženy nutriční hodnoty jídel a jejich doporučené hodnoty z předvoleb v rozmezí vybraných dnů. Zpracovává tyto akce:
  - `FETCH_MEALS_FOR_STATS_REQUEST`
  - `FETCH_MEALS_FOR_STATS_FAILURE`
  - `FETCH_PREFERENCES_FOR_STATS_REQUEST`
  - `FETCH_PREFERENCES_FOR_STATS_SUCCESS`
  - `FETCH_PREFERENCES_FOR_STATS_FAILURE`

V ukázce kódu 10 je vidět ukázka reducer funkce food, která zpracovává tři typy akcí a podle nich změní vrátí nový stav na základě existujícího stavu a dat z akce. Stav je implementován jako immutable struktura dat pomocí knihovny Immutable.js [52], díky které se při každé úpravě struktury vytváří vždy nová struktura.

```
const InitialState = Immutable.Record({
  loading: false,
  list: Immutable.List(),
});
const initialState = new InitialState;
function food(state = initialState, action) {
  switch (action.type) {
    case FETCH_FOOD_LIST_REQUEST:
      return state.set('loading', true);
    case FETCH_FOOD_LIST_FAILURE:
      return state.set('loading', false);
    case FETCH_FOOD_LIST_SUCCESS:
      return state.merge({
        loading: false,
        list: Immutable.fromJS(action.result)
      });
    default:
      return state
  }
}
```

Ukázka kódu 10: Ukázka implementace reducer funkce

```
<Route path="/" component={Main}>
  <IndexRoute
    component={Diary}
    onEnter={requireAuth} />
  <Route path="/jidelnicek/:date"
    component={Diary}
    onEnter={requireAuth} />
  <Route path="/potraviny"
    component={FoodList} />
  <Route path="/potravina/:id"
    component={FoodDetail} />
  <Route path="/potravina/:id/editace"
    component={FoodDetailEdit}
    onEnter={requireAuth} />
  ...
</Route>
```

Ukázka kódu 11: Ukázka implementace routování v React aplikaci

4.4.3 Routování

Routování v React aplikaci je vyřešeno pomocí knihovny React Router [53], která umožňuje mapovat cesty v URL na React komponenty.

V ukázce kódu 11 je vidět definice cest jako `Router` s přiřazenými komponentami. Na základní `Route /` je komponenta `Main`, která obsahuje potomky. Pokaždé když se v aplikaci načte cesta, která je obalena `Main`, tak se vykreslí `Main` a uvnitř konkrétní potomek (komponenta). V routování se také řeší kontrola ověření uživatele v rámci React aplikace. Při vstupu na `Route`, která má `onEnter{requireAuth}` se ověřuje token uživatele.

Odkaz k navigaci se v komponentách vytváří tímto zápisem:

```
<Link to="/predvolby">Předvolby</Link>
```

4.4.4 Komponenty

Strom React komponent je realizován podle návrhu 3.4.3. Komponenty se v této aplikaci dělí na dva typy:

- *Chytré* komponenty
  - Starají se o vykreslování aplikace a vytváření jejího nového stavu.
  - Jsou připojené ke store. Naslouchají ke store a při novém stavu z něj tento stav přijímají.
  - Mají přístup k dispatch funkci, pomocí které mohou odesílat action creator funkce pro vytvoření nového stavu.
  - Mohou v sobě obsahovat potomky - chytré i hloupé komponenty.
  - Stav ze store přenášejí potomkům přes props.
- *Hloupé* komponenty
  - Starají se jen o vykreslování aplikace.
  - Přijímají data nebo callback funkce pouze z props rodičovských komponent.
  - Mohou v sobě obsahovat potomky - chytré i hloupé komponenty.
  - Mohou obsahovat vlastní stav.

Dále jsou popsány jen chytré komponenty. Hloupé komponenty jen přijímají data a vykreslují je.

4.4.4.1 Chytré komponenty

- `Main` - Získává část stavu store vytvářeným reducer funkcí `auth`. Dispatchuje action creator funkce `fetchUser` a `logout`.

- `Diary` - Získává část stavu store vytvářeným reducer funkcemi `meals` a `preferences`. Dispatchuje action creator funkce `fetchMeals` a `fetchPreferences`.
- `AddFood` - Dispatchuje action creator funkce `addMeal` a `updateMeal`.
- `FoodList` - Získává část stavu store vytvářeným reducer funkcí `food`. Dispatchuje action creator funkce `fetchFoodList`, `changeFoodListFilters` a `clearFoodList`.
- `FoodDetail` - Získává část stavu store vytvářeným reducer funkcí `food`. Dispatchuje action creator funkci `fetchFoodDetail`.
- `FoodDetailEdit` komponenta získává část stavu store vytvářeným reducer funkcí `food`. Dispatchuje action creator funkce `fetchFoodDetail`, `updateFood` a `clearFoodDetail`.
- `Statistics` - Získává část stavu vytvářeným reducer funkcí `statistics`. Dispatchuje action creator funkce `fetchMealsForStats` a `fetchPreferencesForStats`.
- `Preferences` - Získává část stavu vytvářeným reducer funkcí `preferences`. Dispatchuje action creator funkce `fetchPreferences` a `updatePreferences`.
- `SignInForm` - Dispatchuje action creator funkci `signin`.
- `SignUpForm` - Dispatchuje action creator funkci `signup`.

Komponenty vyššího řádu Chytré komponenty musí mít implementováno připojení ke store a načítání nového stavu. Chytrých komponent je mnoho a každá by měla duplicitní kód. Proto se používají tzv. *komponenty vyššího řádku*, což jsou funkce, které vezmou komponentu a vrátí novou komponentu s přidanou funkcionalitou. K připojení komponenty ke store se používá `connect` z React Redux [54] a zapisuje se jako ES7 dekorátor.

Další použitou komponentou vyššího řádu je `reduxForm` z redux-form [55], podle které se v aplikaci implementují komponenty s formulářem.

4.4.5 Vzhled

Pro vzhled výsledné aplikace jsou použity CSS frameworku Bootstrap [56]. S React Bootstrap [57] lze psát HTML tagy s Bootstrap třídami jako React komponenty.

4.4.6 Počítání doporučených hodnot

Doporučené hodnoty se počítají v rámci React aplikace. Hodnoty se počítají podle údajů z předvoleb. Údaje se doplní do vzorce z kapitoly 1.1.2.1, z kterého se vypočítají doporučené hodnoty pro kilokalorický příjem a makronutrienty. Doporučené hodnoty mikronutrientů jsou závislé pouze na pohlaví a věku. Ty jsou uloženy společně s jednotkami v pevně dané struktuře dat.

Doporučené hodnoty se počítají v komponentě `Preferences` a jejích potomcích.

4.5 Vykreslování React aplikace na straně serveru

Vykreslování React aplikace při prvním načtení ze serverové strany je implementováno v middleware funkci v části webového serveru. Uvnitř této funkce je funkce `match`, která podle cesty požadavku nalezne příslušnou React komponentu. Pokud komponenta pro své vykreslení potřebuje data z databáze, musí mít definovanou statickou metodu, která dispatchuje asynchronní action creater funkci. I přesto, že se dispatchuje asynchronní funkce, musí se počkat, dokud se nenačtou data do stavu a až poté se vykreslí komponentu. Komponenta se vykreslí na výstup jako čistý HTML řetězec pomocí `ReactDOM.renderToString`. Součástí výstupu je i nalinkovaný JavaScript soubor se sestaveným kódem React aplikace. Po tomto prvním načtení se v prohlížeči rovnou načte React aplikace z nalinkovaného souboru a může dále fungovat jen v prohlížeči. Na webový server se bude dotazovat pouze v případě, kdy bude volat asynchronní action creator funkce s požadavkem na REST API.

4.6 Ověření uživatele

Ověření uživatele funguje na základě JSON Web Tokens. Po úspěšném přihlášení se na serverové straně vygeneruje JSON Web Token (dále jen token), který v sobě obsahuje informace o uživateli. Token je poté uložen v cookies prohlížeče uživatele. Při každém načtení webové aplikace se ověřuje platnost tokenu. Ověřovány jsou i REST API zdroje. Požadavky na REST API, které vyžadují ověření, musí token obsahovat v hlavičce pod `Authorization: Bearer`.

Kapitola 5

Testování

5.1 Nasazení

Pro testování REST API a uživatelské testování byla webová aplikace nasazena na server Heroku[9]. Bylo ale nutné webovou aplikaci přepnout do produkční režimu, kde neběží webpack dev server a neprobíhají validace, které zpomalují načítání aplikace [58]. Také bylo nutné sestavit produkční verzi souboru s kódem pro spuštění React aplikace v prohlížeči. Pro tento účel byla vytvořena produkční verze konfigurace Webpacku, která oproti vývojové verzi neobsahuje webpack dev server a hot reloading. Navíc obsahuje pluginy pro optimalizaci a minimalizaci kódu.

Databáze byla nahrána na mLab[10]. Kolekce s potravinami byla naplněna daty z Musculus.cz [59] s písemným souhlasem autora [60]. Data obsahují téměř 1200 potravin, jejich energetické hodnoty a makronutrienty. Pro účely testování byly všechny potraviny doplněny o důležité vitamíny a minerály náhodných hodnot.

Prototyp webové aplikace v době psaní této práce funguje na `https://caloric.herokuapp.com/`. Dovoluji si upozornit, že výše zmíněné služby jsou využívány v neplacené verzi, které mají omezený výkon. Webová aplikace tedy nemusí dosahovat takové odezvy jako při nasazení na běžných placených serverech.

5.2 Jednotkové testy

Jednotkové testy ověřují funkčnost a korektnost jednotlivých částí aplikace. V React aplikaci byly napsány jednotkové testy pro problematickou část, kde se vytváří stav celé React aplikace - reducer funkce. Pro tyto testy byly použity nástroje Mocha [61], Chai [62] a Chai Immutable [63].

[9] https://www.heroku.com/
[10] https://mlab.com/

5.3 Testování REST API

Pro testování REST API byl použit nástroj SoapUI [64]. Testovaly se všechny nadefinované cesty 3.3.3, na které se posílaly požadavky a kontrovaly se statusy a těla odpovědí. Všechny cesty vrátily správné odpovědi a celý REST API test proběhl v pořádku.

5.4 Uživatelské testování

Pro uživatelské testování byla vybrána malá skupinka jedinců. Část z nich již měla zkušenosti s užíváním podobných webových aplikací. Uživatelům bylo na začátku jen řečeno, k čemu webová aplikace slouží. Následně jim byl dán seznam úkolů pro testování.

5.4.1 Úkoly pro testování

1. Zaregistrujte se.
2. Přihlašte se.
3. Vyplňte v předvolbách, které nutriční hodnoty chcete sledovat.
4. Zapište si do jídelníčku potraviny, které jste včera a dnes zkonzumovali (množství a váhu stačí přibližně).
5. Zobrazte si statistiky jedné vámi vybrané sledované nutriční hodnoty.
6. Nalezněte v sekci potraviny brambory a zobrazte si jejich sacharidy.
7. Vytvořte vlastní potravinu (nutriční hodnoty nemusí být reálné).

5.4.2 Shrnutí uživatelského testování

Provedení úkolů proběhlo bez komplikací. Všichni uživatelé splnili všech sedm zadaných úkolů. Uživatelé byly poté dotázáni, jestli při testování narazili na problémy nebo jestli jim něco ve webové aplikaci nechybí.

Byl nalezen jeden problém. Při zapisování jídelníčku za včerejší den, nebyly zobrazeny doporučené hodnoty a nešlo je ani nastavit.

5.4.3 Úprava webové aplikace na základě testování

Dodatečně byla implementována možnost vytváření a upravování předvoleb ze starších dnů. Úprava se týkala pouze React aplikace, kde byla přidána kontrola na existenci předvoleb ze starších dnů a definice cesty pro úpravu starších předvoleb. Odkaz na úpravu předvoleb byl přidán do bloku nutričních hodnot v jídelníčku.

Závěr

Cílem této práce bylo vyvinout prototyp moderní webové aplikace pro sledování nutričních hodnot.

Nejdříve bylo nutné se s touto problematikou seznámit, definovat pojmy a potřebné vzorce. Následně byla provedena analýza webových aplikací, které se touto problematikou zabývají. Při analýze byly zjištěny existující postupy a zároveň chybějící funkcionalita.

Na základě výsledků analýzy byla navržena a implementována univerzální Single page aplikace včetně REST API a databáze. Při vývoji byly použity nejnovější postupy a technologie včetně zadaných React, Node.js a MongoDB.

Webová aplikace byla na konci otestována třemi typy testů.

Všechny body zadání práce byly splněny.

Výsledná webová aplikace umožňuje uživatelům lepší životosprávu tím, že jim poskytuje komplexní přehled o přijatých potravinách.

Další vývoj

Při vývoji této webové aplikace bylo dbáno na možnosti budoucí rozšířitelnosti. Na dalším vývoji hodlám pokračovat. Prototyp mám v plánu dotáhnout do verze připravené k nasazení v produkci. V následujících bodech uvádím možnosti, kterými lze webovou aplikaci a celkově tento projekt dále rozšířit.

Detailnější jídelníček Zobrazovat detaily zapsaných potravin v jídelníčku. Zobrazovat přijaté a doporučené nutriční hodnoty v rámci týdne.

Správa potravin Přidáním nové potraviny je potravina dostupná pouze uživateli, který ji přidal. Pro zpřístupnění potraviny ostatní uživatele musí projít ověřovacím procesem.

Automatické ukládání a doporučování často zapisovaných potravin.

Správa účtu Detailnější správa uživatelského účtu. Možnost přihlašování a registrace přes služby třetích stran (např. Facebook, Google apod.).

Nativní mobilní aplikace S React Native [65] lze využít velké části existujícího React kódu a implementovat nativní mobilní aplikaci.

Literatura

[1] Nordqvist, C.: What Are Calories? How Many Do We Need? [online], 02 2016, [přístup 2016-02-18]. Dostupné z: `http://www.medicalnewstoday.com/articles/263028.php`

[2] Osborne, D.; Voogt, P. i.; aj.: *The analysis of nutrients in foods*. Academic Press Inc.(London) Ltd., 24/28 Oval Road, London NW1 7DX., 1978.

[3] Owen, W.: Does Calorie Counting Work? [online], 2015, [přístup 2016-03-18]. Dostupné z: `http://travelstrong.net/calorie-counting`

[4] David McAuley, P.: Determination of the Resting Metabolic Rate. [online], 2016, [přístup 2016-03-20]. Dostupné z: `http://www.globalrph.com/resting_metabolic_rate.htm`

[5] Frankenfield D., C. C., Roth-Yousey L.: Comparison of predictive equations for resting metabolic rate in healthy nonobese and obese adults: a systematic review. *Journal of the American Dietetic Association*, ročník 105, č. 5, 2005: s. 775–789, [přístup 2016-02-18].

[6] Crome, G.: How Many Calories Do We Really Need? [online], 10 2014, [přístup 2016-03-18]. Dostupné z: `http://www.acefitness.org/acefit/expert-insight-article/60/3772/how-many-calories-do-we-really-need`

[7] A Report of the Panel on Macronutrients, S. o. U. R. L. o. N.; Interpretation; Uses of Dietary Reference Intakes, F., Standing Committee on the Scientific Evaluation of Dietary Reference Intakes; aj.: *Dietary Reference Intakes for Energy, Carbohydrate, Fiber, Fat, Fatty Acids, Cholesterol, Protein, and Amino Acids (Macronutrients)*. National Academies Press, 2005, ISBN 978-0-309-08525-0.

[8] Vora, P.: *Web application design patterns*. Morgan Kaufmann, 2009, [přístup 2016-03-20].

[9] Mikowski, M. S.; Powell, J. C.: Single Page Web Applications. *B and W*, 2013, [přístup 2016-03-20].

[10] Takada, M.: Modern web applications: an overview. [online], 2015, [přístup 2016-03-20]. Dostupné z: `http://singlepageappbook.com/goal.html`

[11] GitHub Inc.: Front-end JavaScript frameworks. [online], 2016, [přístup 2016-03-28]. Dostupné z: `https://github.com/showcases/front-end-javascript-frameworks`

[12] Google Inc.: AngularJS. [online], [přístup 2016-03-28]. Dostupné z: `https://angularjs.org/`

[13] Facebook Inc.: React. [software], [přístup 2016-03-28]. Dostupné z: `https://facebook.github.io/react/`

[14] Ashkenas, J.: Backbone. [software], [přístup 2016-03-28]. Dostupné z: `http://backbonejs.org/`

[15] Tilde Inc.: Ember.js. [software], [přístup 2016-03-28]. Dostupné z: `http://emberjs.com/`

[16] Kazushi Nagayama: Deprecating our AJAX crawling scheme. [online], 2015, [přístup 2016-03-28]. Dostupné z: `https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html`

[17] Bohumil Jahoda: Indexování JavaScriptu. [online], 2016, [přístup 2016-03-28]. Dostupné z: `http://jecas.cz/seo-javascript`

[18] Brown, E.: *Modern JavaScript*. O'Reilly, 2015, [přístup 2016-03-25]. Dostupné z: `http://www.oreilly.com/web-platform/free/files/modern-javascript.pdf`

[19] Francois Ward: State of the Art JavaScript in 2016. [online], 02 2016, [přístup 2016-03-28]. Dostupné z: `https://medium.com/javascript-and-opinions/state-of-the-art-javascript-in-2016-ab67fc68eb0b`

[20] Rauschmayer, A.: Exploring ES6. [online], 2016, [přístup 2016-03-22]. Dostupné z: `http://exploringjs.com/es6/`

[21] Node.js Foundation: Node.js. [software], [přístup 2016-03-28]. Dostupné z: `https://nodejs.org/`

[22] Sanchez, R.: Comparing Node.js vs PHP Performance. [online], 10 2015, [přístup 2016-03-22]. Dostupné z: `http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/`

[23] Azat Mardan: Why Everyone is Talking About Isomorphic / Universal JavaScript and Why it Matters. [online], 01 2016, [přístup 2016-03-28]. Dostupné z: http://www.capitalone.io/blog/why-is-everyone-talking-about-isomorphic-javascript/

[24] npm Inc.: npm. [software], [přístup 2016-03-28]. Dostupné z: https://www.npmjs.com/

[25] Twitter Inc.: Bower. [software], [přístup 2016-03-28]. Dostupné z: http://bower.io/

[26] Koppers, T.: Motivation. [online], 2016, [přístup 2016-03-22]. Dostupné z: https://webpack.github.io/docs/motivation.html

[27] Dangoor, K.: CommonJS. [software], [přístup 2016-03-28]. Dostupné z: http://www.commonjs.org/

[28] Burke, J.: RequireJS. [software], [přístup 2016-03-28]. Dostupné z: http://www.requirejs.org/

[29] Adda Bjarnadottir, M.: The 5 Best Calorie Counter Websites and Apps. [online], 03 2016, [přístup 2016-03-20]. Dostupné z: https://authoritynutrition.com/5-best-calorie-counters/

[30] Alias, E.: Wappalyzer. [online], 2015, [přístup 2016-03-20]. Dostupné z: https://github.com/AliasIO/Wappalyzer

[31] Node.js Foundation: About Node.js®. [online], 2016, [přístup 2016-04-01]. Dostupné z: https://nodejs.org/en/about/

[32] npm, Inc.: What is npm? [online], 2016, [přístup 2016-04-01]. Dostupné z: https://docs.npmjs.com/getting-started/what-is-npm

[33] StrongLoop, I.: Express. [software], 2016, [přístup 2016-04-20]. Dostupné z: http://expressjs.com/

[34] MongoDB Inc.: MongoDB. [software], [přístup 2016-04-10]. Dostupné z: https://www.mongodb.org/

[35] Koppers, T.: Motivation. [online], 2016, [přístup 2016-03-29]. Dostupné z: https://webpack.github.io/docs/list-of-loaders.html

[36] Couto, L.: babel-loader. [software], 2016, [přístup 2016-04-01]. Dostupné z: https://github.com/babel/babel-loader

[37] Koppers, T.: list of plugins. [online], 2016, [přístup 2016-04-01]. Dostupné z: https://webpack.github.io/docs/list-of-plugins.html

[38] Koppers, T.: list of plugins. [online], 2016, [přístup 2016-04-01]. Dostupné z: http://webpack.github.io/docs/hot-module-replacement-with-webpack.html

[39] Auth0® Inc.: JSON Web Tokens. [online], 2016, [přístup 2016-04-20]. Dostupné z: https://jwt.io/

[40] Masse, M.: *REST API design rulebook*. "O'Reilly Media, Inc.", 2011.

[41] MongoDB Inc.: $regex. [online], 2016, [přístup 2016-04-15]. Dostupné z: https://docs.mongodb.org/manual/reference/operator/query/regex/

[42] MongoDB Inc.: $regex. [online], 2016, [přístup 2016-04-15]. Dostupné z: https://docs.mongodb.com/manual/reference/method/ObjectId/

[43] Facebook Inc.: Flux. [online], 2016, [přístup 2016-04-30]. Dostupné z: https://facebook.github.io/flux/

[44] Abramov, D.: Redux. [software], 2016, [přístup 2016-04-30]. Dostupné z: https://github.com/reactjs/redux

[45] Arne Brasseur: Functional Programming: Pure Functions. [online], 09 2016, [přístup 2016-04-20]. Dostupné z: http://www.sitepoint.com/functional-programming-pure-functions/

[46] Jonathan Ong, D. C. W.: body-parser. [software], 2016, [přístup 2016-04-20]. Dostupné z: https://github.com/expressjs/body-parser

[47] TJ Holowaychuk, D. C. W.: cookie-parser. [software], 2016, [přístup 2016-04-20]. Dostupné z: https://github.com/expressjs/cookie-parser

[48] LearnBoost: mongoose. [software], [přístup 2016-04-20]. Dostupné z: http://mongoosejs.com/

[49] Abramov, D.: cookie-parser. [software], 2016, [přístup 2016-04-30]. Dostupné z: https://github.com/gaearon/react-hot-loader

[50] Network, M. D.; individual contributorv: Fetch API. [online], 2016, [přístup 2016-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[51] Andrews, M.: isomorphic-fetch. [software], 2016, [přístup 2016-05-05]. Dostupné z: https://github.com/matthew-andrews/isomorphic-fetch

[52] Facebook Inc.: Immutable.js. [software], 2016, [přístup 2016-05-05]. Dostupné z: https://facebook.github.io/immutable-js/

[53] Ryan Florence, M. J.: React Router. [software], 2016, [přístup 2016-05-05]. Dostupné z: https://github.com/reactjs/react-router

[54] Abramov, D.: React Redux. [software], 2016, [přístup 2016-05-10]. Dostupné z: https://github.com/reactjs/react-redux

[55] Rasmussen, E.: React Redux. [software], 2016, [přístup 2016-05-10]. Dostupné z: https://github.com/erikras/redux-form

[56] Twitter, Inc.: Bootstrap. [software], 2016, [přístup 2016-05-10]. Dostupné z: http://getbootstrap.com/

[57] Stephen J. Collings, P. V., Matthew Honnibal: React-Bootstrap. [software], 2016, [přístup 2016-05-10]. Dostupné z: http://react-bootstrap.github.io/

[58] Facebook Inc.: Reusable Components. [online], 2016, [přístup 2016-05-12]. Dostupné z: https://facebook.github.io/react/docs/reusable-components.html

[59] Diviš, L.: Energetické tabulky potravin. [online], 2003, [cit. 2016-05-12]. Dostupné z: http://www.musculus.cz/kulturistika/programy/energeticke-tabulky-potravin

[60] Diviš, L.: [e-mailová komunikace], 2016, [cit. 2016-05-12].

[61] TJ Holowaychuk: Mocha. [software], 2016, [přístup 2016-05-12]. Dostupné z: http://mochajs.org/

[62] Jake Luer: Chai. [software], 2016, [přístup 2016-05-12]. Dostupné z: https://github.com/chaijs/chai

[63] Jérémie Astori: Chai Immutable. [software], 2016, [přístup 2016-05-12]. Dostupné z: https://github.com/astorije/chai-immutable

[64] Software, S.: SoapUI. [software], 2016, [přístup 2016-05-12]. Dostupné z: https://www.soapui.org/

[65] Facebook Inc.: React Native. [software], 2016, [přístup 2016-05-12]. Dostupné z: https://facebook.github.io/react-native/e

Příloha **A**

Seznam použitých zkratek

ES ECMAScript

TEE Total energy expenditure

RMR Resting metabolic rate

npm node package manager

NoSQL Not only Structured Query Language

BSON Binary JavaScript Object Notation

JSON JavaScript Object Notation

DOM Document Object Model

XML Extensible Markup Language

HTML HyperText Markup Language

CSS Cascading Style Sheets

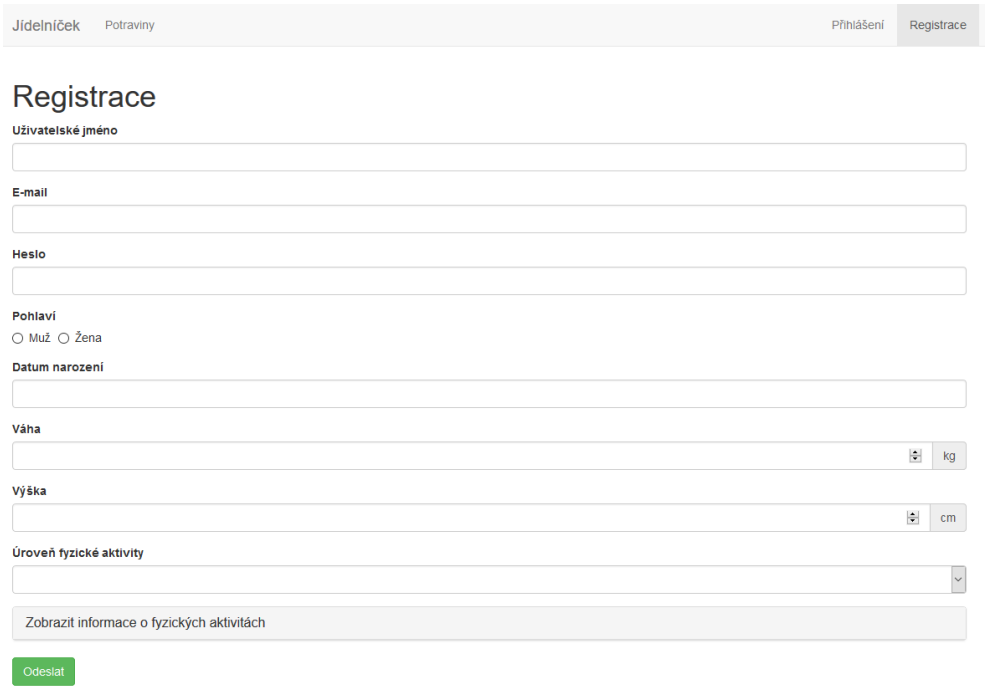
REST Representational State Transfer

API Application Programming Interface

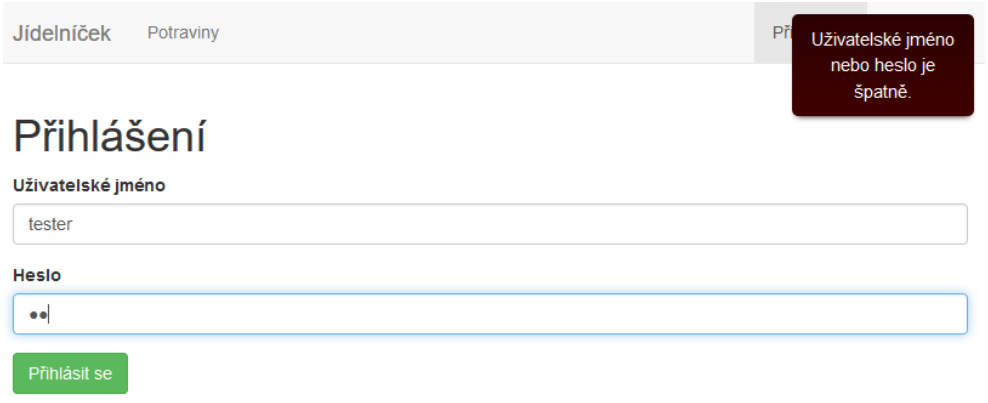
JWT JSON Web Tokens

Příloha B

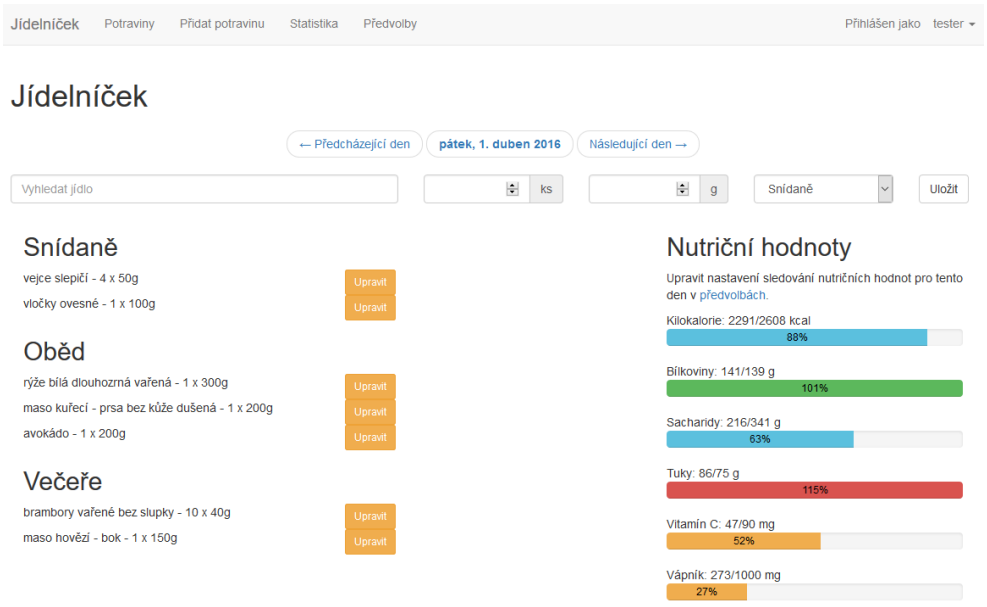
Ukázka výsledné aplikace



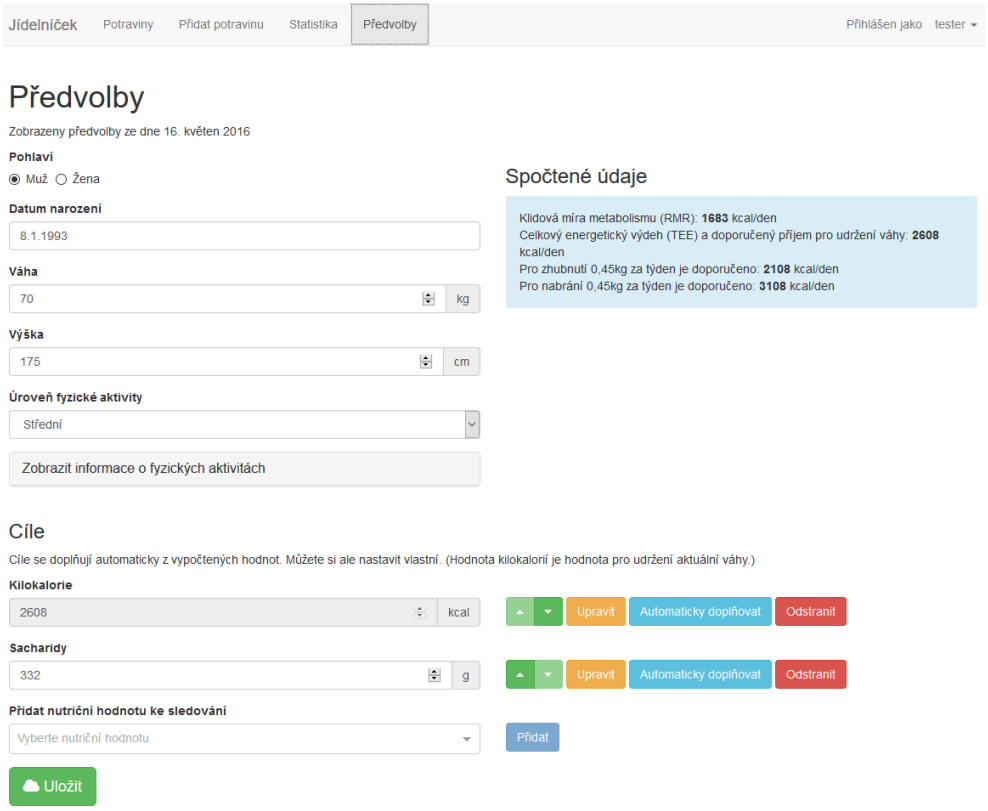
Obrázek B.1: Ukázka registrace



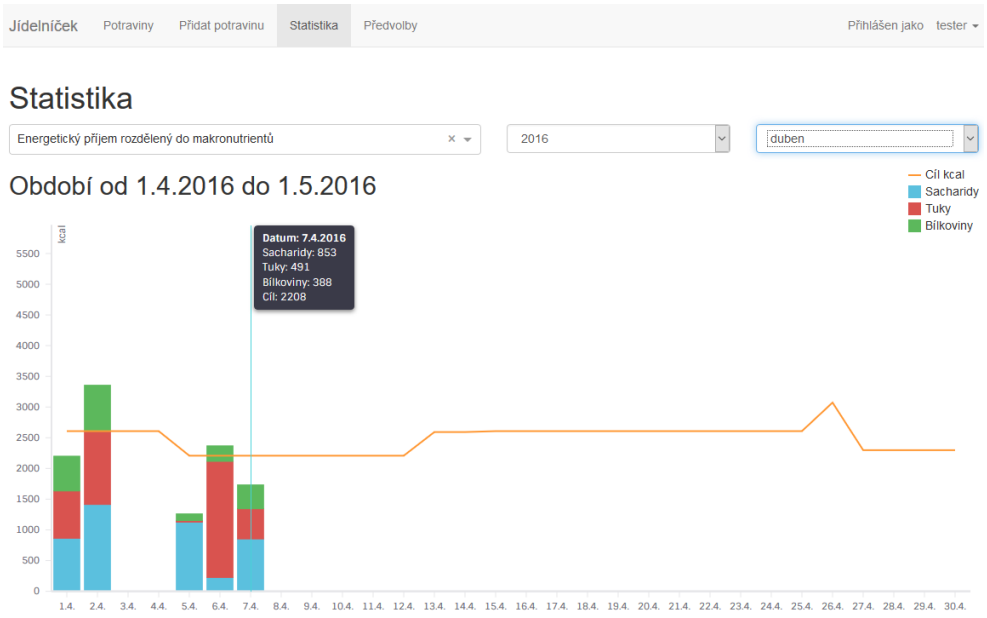
Obrázek B.2: Ukázka přihlášení



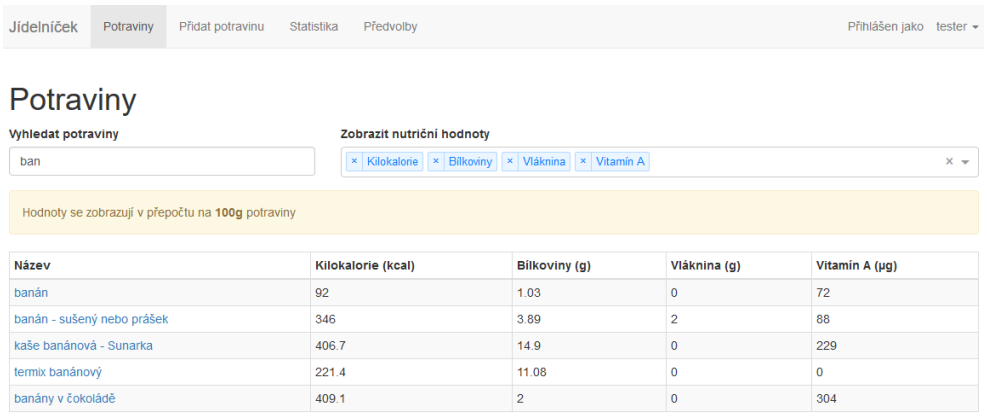
Obrázek B.3: Ukázka jídelníčku



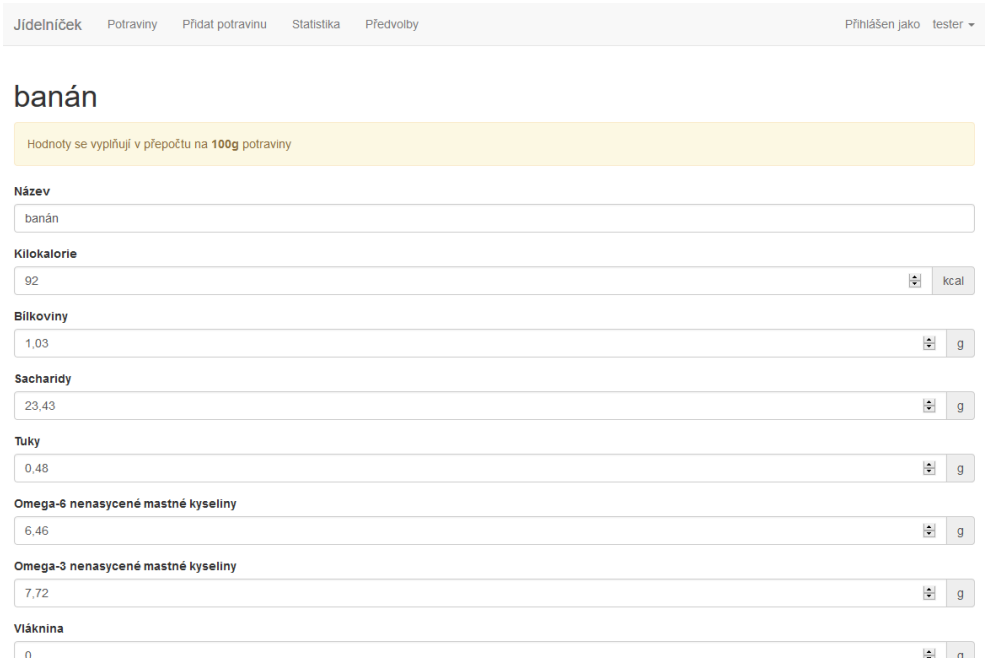
Obrázek B.4: Ukázka předvoleb



Obrázek B.5: Ukázka statistik



Obrázek B.6: Ukázka seznamu potravin



Obrázek B.7: Ukázka úpravy potraviny

Příloha C

Obsah přiloženého CD

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