

Gymnázium, Praha 6, Arabská 14

Programování

MATURITNÍ PRÁCE



David Čech

květen 2020

Gymnázium, Praha 6, Arabská 14

MATURITNÍ PRÁCE

Předmět: Programování

Téma: Stravovací deníček

Autor: David Čech

Třída: 4.E

Školní rok: 2019/2020

Třídní učitelka: Mgr. Jana Urzová, Ph.D.

Konzultant: Mgr. Jan Lána

Prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne

.....
David Čech

Anotace

Ve své maturitní práci jsem se zabýval vytvořením webové aplikace, která by sloužila jako stravovací deníček. Má maturitní práce má dvě části – backend napsaný v aplikačním rámci Express pro softwarový systém Node.js a frontend napsaný v JavaScriptových knihovnách React a Redux. Frontend vytváří uživatelské rozhraní v internetovém prohlížeči a backend obstarává zpracovávání a ukládání dat přijatých od frontendu do databáze. Jako databázi aplikace používá MongoDB uloženou na cloudu webové stránky MongoDB Atlas. Výsledkem mé maturitní práce je fungující webová aplikace, která umožňuje uživateli spravovat jídla, jejich výživové hodnoty a vytvářet tzv. deníčkové záznamy, které obsahují údaje o aktivitách a stravě uživatele v daný den. Tyto deníčkové záznamy si může uživatel zpětně prohlížet a porovnávat své výsledky se svým nutričním cílem.

Abstract

My project consisted of creating a web application which would serve as a diet diary. It has two parts – backend which is written in a framework Express for JavaScript runtime environment Node.js, and frontend written JavaScript libraries React, and Redux. Frontend creates user interface in web browser, and backend takes care of processing, and saving data it receives from frontend into the database. The application uses MongoDB as its database which is located in a cloud storage of the website MongoDB Atlas. The result of my work is a working web application that allows the user to manage foods, their nutritional values, and create so-called diary entries which contain information about the user's activities, and nutrition throughout the day. These diary entries can also be retrospectively viewed by the user, and the user can compare their progress with their nutritional goal.

Zadání

Jako maturitní projekt bych chtěl vypracovat webovou aplikaci (frontend aplikace React-Redux, backend v Node.js - Express a databázi MongoDB). Cílem by bylo vytvoření "Stravovacího deníčku"; aplikace ke spravování receptů a jejich kalorických hodnot, uživatel by měl také být schopen podívat se v retrospektivě na to, co jedl, a zhodnotit tak svou stravu (nejspíše ve formě grafu). Také bych chtěl, aby bylo součástí spočítání doporučených kalorických hodnot podle informací o uživateli.

Obsah

1. Úvod.....	8
2. Technologie použité k vývoji projektu.....	9
2.1 Vývojové nástroje.....	9
2.1.1 IntelliJ IDEA	9
2.1.2 Git a GitHub	9
2.2 Frontend.....	9
2.2.1 React.....	9
2.2.2 Redux	10
2.3 Backend	11
2.3.1 Express	11
2.3.2 MongoDB Atlas a Mongoose.....	12
3. Instalace.....	13
3.1 Backend	13
3.2 Frontend.....	13
4. Backend.....	14
4.1 Modely.....	15
4.1.1 Uživatel	15
4.1.2 Jídla	16
4.1.3 Deníčkové záznamy	16
4.2 Endpointy.....	18
4.2.1 Uživatelské endpointy	18
4.2.2 Jídelní endpointy	19
4.2.3 Endpointy pro deníčkové záznamy	20
5. Frontend	22
5.1 Reducers	23
5.1.1 DiaryEntryReducer.....	23
5.1.2 FoodReducer	23
5.1.3 AuthReducer.....	24
5.2 Action creators.....	24
5.2.1 DiaryEntryActionCreator	24
5.2.2 FoodActionCreator	25
5.2.3 AuthActionCreator	25

5.3	Komponenty	26
5.3.1	Navigation a UserInformation.....	26
5.3.2	Login	27
5.3.3	Register.....	27
5.3.4	SearchFood.....	28
5.3.5	FoodDetails	28
5.3.6	CreateIngredient	29
5.3.7	CreateMeal	30
5.3.8	CreateDiaryEntry	31
5.3.9	ViewDiaryEntries.....	32
5.3.10	UserInformation	34
6.	Závěr.....	36
7.	Použitá literatura	37
8.	Seznam obrázků	38

1. ÚVOD

Ve své maturitní práci jsem se zabýval vytvořením webové aplikace, která by sloužila jako stravovací deníček. Účelem stravovacího deníčku je umožnit uživateli zaznamenávat data o jeho aktivitách a stravě, načtež si tato data může uživatel zpětně prohlížet, a hodnotit tak své výkony. Zpětné prohlížení zápisů je realizováno pomocí grafů, ve kterých je promítnutý jak nutriční cíl, tak výživové údaje zkonsumovaného jídla. Uživatel si může taktéž prostřednictvím aplikace prohlíže nebo vytvářet vlastní jídla s nutričními hodnotami a nutriční cíl, který sestává z doporučeného denního příjmu energie a makronutrientů. Nutriční cíl je vypočítáván z informací o uživateli pomocí rovnice Mifflin St. Jeor.

Po technické stránce má aplikace dvě části – serverové rozhraní backend a uživatelské rozhraní frontend. Backend zajišťuje komunikaci s databází a vyřizování požadavků, jež obdrží od frontendu (od uživatele). Frontend se stará o přenos informací mezi uživatelem a backendem.

Motivací pro vytvoření mého projektu bylo vytvoření aplikace, která by intuitivně propojovala vypočítání nutričního cíle, počítání výživových údajů zkonsumovaného jídla a databázi, ve které by byla uložena samotná jídla. Uživatel by díky tomuto propojení nebyl nucen hledat prostřednictvím různých zdrojů informace o těchto třech položkách a následně je manuálně spojovat, nýbrž měl by k nim všem přístup na jednom místě. Takové aplikace již existují (př.: KalorickéTabulky, MyFitnessPal), avšak jsou často placené, nebo příliš komplikované, čemuž jsem se chtěl u své aplikace vyhnout.

2. TECHNOLOGIE POUŽITÉ K VÝVOJI PROJEKTU

Tato podkapitola popisuje technologie, které byly použity při vývoji jak frontendu, tak backendu. Bude se také povrchně věnovat struktuře celé aplikace.

2.1 Vývojové nástroje

V této podkapitole jsou popsány nástroje, byly použity při vývoji obou částí mé maturitní práce, ale na chodu výsledného produktu se již nepodílejí.

2.1.1 IntelliJ IDEA

První technologií, která stojí za zmínku je vývojové prostředí IntelliJ IDEA, ve kterém byly napsány obě části mé maturitní práce. Největší jeho předností je podpora aplikačních rámců (frameworků), v nichž je má webová aplikace napsána. Pro frontend to byly aplikační rámce React a Redux a pro backend Express a Mongoose. Dále je v něm zabudovaný Git, což je systém pro spravování verzí projektu (Version Control System – VCS). V neposlední řadě kontroluje IntelliJ IDEA syntaxi kódu, upozorňuje na chyby, umožňuje snadný pohyb mezi soubory, jež tvoří projekt, ukazuje nabídku doplnění názvů proměnných apod.

2.1.2 Git a GitHub

Pro spravování verzí projektu a k zálohování jsem využíval Git a GitHub. Git slouží, mimo jiné, ke sdílení projektu na několika zařízeních a k ukládání změn. GitHub je webová služba, která poskytuje bezplatný webhosting pro ukládání opensource projektů. Obě části stravovacího deníčku mají na GitHubu vlastní repositář; pro frontend je to repositář na url <https://github.com/DavidCech/dietdiary-frontend> a pro backend je to repositář na url <https://github.com/DavidCech/dietdiary-backend>.

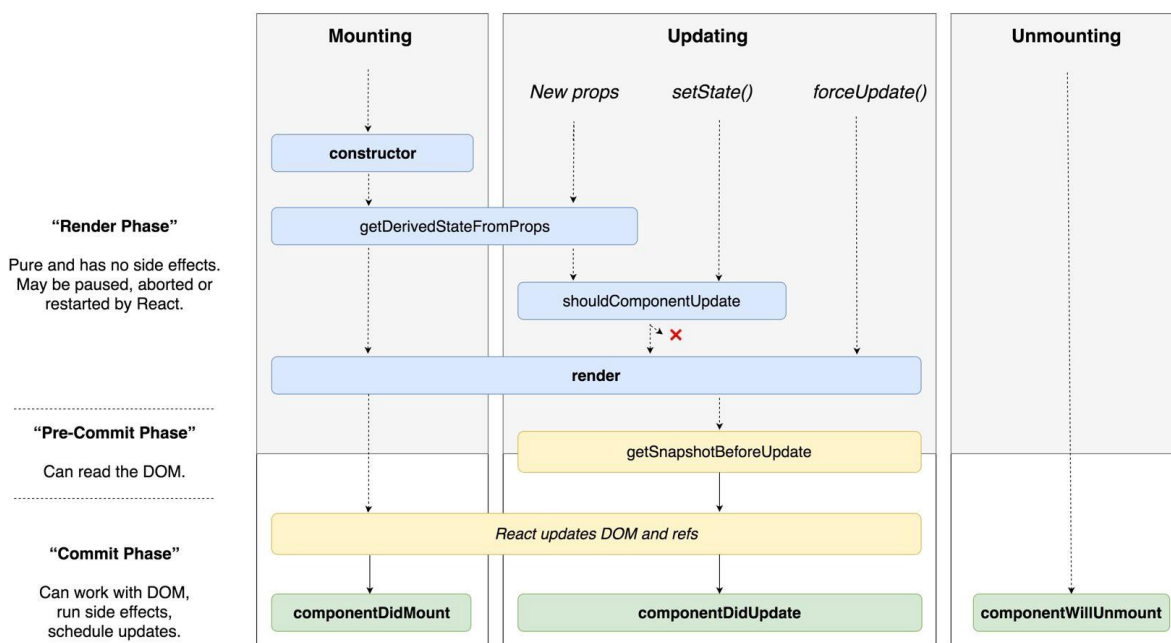
2.2 Frontend

Tato podkapitola charakterizuje technologie, které byly použity pouze při vývoji frontendu. Také blíže specifikuje, jak na sebe dané technologie navazují a fungují spolu, čímž již povrchně popisuje vnitřní strukturu frontendu.

2.2.1 React

React je aplikační rámec pro tvorbu uživatelského rozhraní. Jeho hlavní výhodou je kombinace programovacích jazyků JavaScript a HTML přímo v kódu – tzv. jsx. Základním stavebním blokem programů napsaných v Reactu jsou komponenty (Components), které mají své životní cykly (viz obr. 1) a funkci render, která musí vracet prosté HTML nebo jsx a která určuje, co se uživateli zobrazí v prohlížeči. Životní cykly mají 3 hlavní fáze: Mount, Update a Unmount. Mount je první fáze životního cyklu a dochází k ní, když se komponenta poprvé vykresluje na webové stránce. K Update fázi dochází pokaždé, když se změní data, která komponenta dostává

od svých rodičovských komponent (tzv. props), nebo data v samotné komponentě (např.: vyplnění formuláře, kliknutí na tlačítko). A k poslední fázi Unmount dochází, když už není třeba komponentu na webové stránce vykreslovat. Tyto životní cykly spouštějí nesmírně užitečné funkce, které může vývojář ve svém programu dále využívat. Kromě pohodlnosti psaní aplikace z hlediska programátora, je React optimální pro práci s rychle měnícími se daty, která je schopen velice rychle zpracovávat. Jedinou jeho nevýhodou je to, že je mezi jeho komponentami možná pouze jednostranná komunikace – rodičovské komponenty nejsou schopny přijímat data od svých potomků. Tento problém řeší další framework s názvem Redux. Podrobný pohled na strukturu Reactu a na to, jak React funguje, lze nalézt v citacích (Mawer, 2018).



Obr. 1: Životní cyklus komponenty v Reactu¹

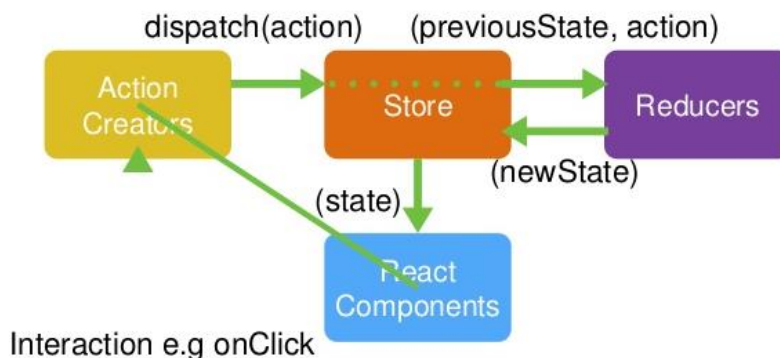
2.2.2 Redux

Redux je další JavaScript framework, který vytváří datovou strukturu, jež se nazývá úložiště (Store). V případě React-Redux aplikací se tato datová struktura nachází v hierarchii téměř nad všemi ostatními komponentami, což znamená, že skoro všechny komponenty mohou od této datové struktury dědit data. Její druhou vlastností je, že do ní mohou všechny komponenty data zapisovat pomocí metody `dispatch`, která jako parametr dostává objekty, jež se nazývají akce (actions). Tyto akce obsahují data, která se mají zapsat do úložiště. K úložišti se připojují komponenty prostřednictvím funkce `connect`, jež také zajišťuje, že komponenty obdrží v podobě props jak funkce zapisující data do úložiště, tak data z úložiště samotná. Jednotlivé položky v úložišti a jejich přepisování akcemi je definováno v souborech, které se nazývají

¹ Obr. 1: Životní cyklus komponenty v Reactu, převzato z webové stránky *blogDidMount – 2018 Guide to React Component Lifecycle Methods*. Dostupné z url: https://medium.com/@_ChrisBradshaw/blogdidmount-2018-guide-to-react-component-lifecycle-methods-1614e0bbe80a

reducers. Všechny funkce, v nichž se volá funkce `dispatch` s akcemi, by se měly nacházet v souborech se jmény `action creators`. Více o Reduxu naleznete v citacích (Engineering Team, 2017).

Redux Flow



Obr. 2: Oběh dat v React-Redux aplikaci²

2.3 Backend

Tato podkapitola se zabývá technologiemi, které byly použity pouze při vývoji backendu. Zároveň také pojednává o použité databázi a knihovně použité ke komunikaci mezi backendem a databází.

2.3.1 Express

Express je JavaScriptový framework, který zjednodušuje psaní serverového rozhraní aplikace v softwarovém systému Node.js. V mé aplikaci ho využívám pro vytvoření endpointů (tj. url, na která jsou posílány požadavky z frontendu). Tyto požadavky jsou dále pomocí Expressu zpracovávány. Zpracování požadavku může vypadat různě, avšak většinou zahrnuje čtení, zapisování, mazání nebo upravování záznamů v databázi. Poté, co se požadavek zpracuje a vyřídí, odešle backend frontendu odpověď, která opět může mít mnoho podob – nejčastěji to jsou buď data ve formátu JSON (JavaScript Object Notation – JavaScriptový objektový zápis), nebo stavové kódy (200 – OK, 400 – Bad Request, 500 – Internal Server Error apod.). Odpověď od backendu přijímá frontend a dále tato přijatá data zpracovává. Díky Expressu je backend kompaktní, jelikož každému endpointu je přiřazena právě jedna funkce, v níž dochází k vyřizování požadavků. Více o Expressu se můžete dočíst v citacích (MDN contributors, 2020).

² Obr. 2: Oběh dat v React-Redux aplikaci, převzato z videa *React Workshops 01: A Step-by-Step Walkthrough Of Redux Data Flow (part1)*. Dostupné z url: <https://www.youtube.com/watch?v=hiaqh162zZs>.

2.3.2 MongoDB Atlas a Mongoose

MongoDB Atlas je webová služba, která poskytuje webhosting pro MongoDB databáze. Pokud se jedná o málo frekventované databáze, je tato služba bezplatná. Hlavní předností MongoDB databází je to, že jsou v nich dokumenty zapsané ve JSONu, tudíž je není nutné převádět do jiného formátu při komunikaci mezi backendem a databází. Mongoose je knihovna pro Node.js, která slouží k připojování se k MongoDB databázím a následné komunikaci mezi Node.js aplikací a MongoDB databází. Komunikace probíhá prostřednictvím tzv. dotazů (queries), které mohou být jednoduché i velmi komplikované. Dále Mongoose umožňuje vytvoření schémat pro jednotlivé typy dokumentů v databázi, v nichž je definováno, které atributy dokumenty musejí mít, které atributy jsou dobrovolné, hodnoty kterých atributů musejí být unikátní, jejich výchozí hodnoty atd. Detailnější pohled na MongoDB a Mongoose se nachází v citacích (Karnik, 2018).

3. INSTALACE

Tato kapitola popisuje, jak nainstalovat mou webovou aplikaci. K instalaci je potřeba mít nainstalovaný Node Package Manager, který zajišťuje instalaci a spravování modulů potřebných k fungování aplikace. Dále v této kapitole budou napsány příkazy pro instalaci aplikace pomocí Gitu, avšak lze použít i jiný Version Control System a příkazy analogické těm, které používá Git.

3.1 Backend

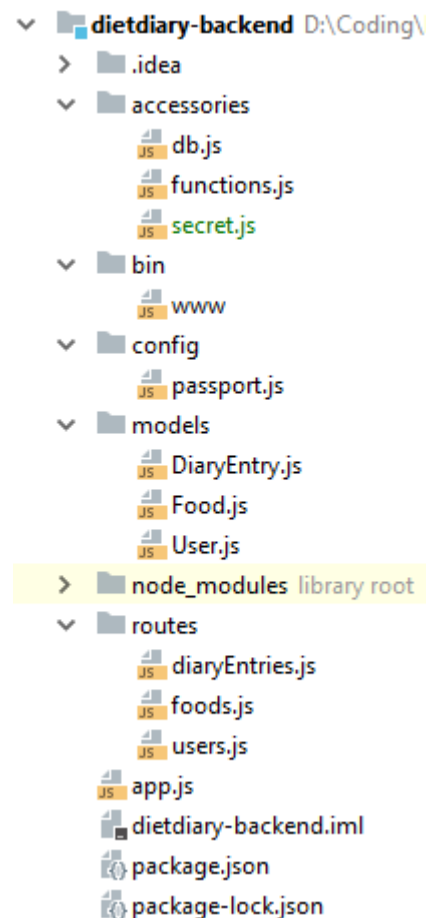
1. Otevřeme si příkazovou řádku s Gitem a zadáme do ní příkaz **git clone https://github.com/DavidCech/dietdiary-backend.git**, který nám vytvoří v pracovním adresáři složku *dietdiary-backend* se soubory z GitHub repositáře, v němž je backend uložen.
2. Pomocí příkazů **cd dietdiary-backend** a **npm install** zavoláme Node Package Manager a nainstalujeme všechny moduly zapsané v souboru *package.json*.
3. Pro spuštění aplikace poté vždy používáme příkaz **npm start**, který za pomoci nainstalovaných modulů spustí backend.

3.2 Frontend

1. Analogicky k instalaci backendu si otevřeme příkazovou řádku s Gitem a zadáme do ní příkaz **git clone https://github.com/DavidCech/dietdiary-frontend.git**, který nám vytvoří v pracovním adresáři složku *dietdiary-frontend* se soubory z GitHub repositáře, v němž je frontend uložen.
2. Pomocí příkazů **cd dietdiary-frontend** a **npm install** zavoláme Node Package Manager a nainstalujeme všechny moduly zapsané v souboru *package.json*.
3. Pro spuštění aplikace poté vždy používáme příkaz **npm start**, který za pomoci nainstalovaných modulů spustí frontend.

4. BACKEND

Jak bylo již několikrát zmíněno backend je serverové rozhraní, jež vyřizuje požadavky, které dostává od frontendu. Také zařizuje obousměrnou komunikaci s databází v cloudovém úložišti MongoDB Atlas. Základním souborem, který propojuje všechny části backendu, je *app.js*. V něm se specifikuje užití knihovny Express a ostatních přídatných součástí aplikace jako např.: middleware Cookie Parser. Také zčásti definuje url, na kterých se budou nacházet endpointy (zčásti jsou url endpointů definovány ještě v souborech, v nichž se nachází samotné vyřizování požadavků). V mé aplikaci jsou 3 pomyslné kmeny endpointů, které se dále větví a které jsou definované právě v souboru *app.js*. Těmito kmeny jsou **/foods**, **/users** a **/diaryEntries** před lomítky těchto částí url by byla ještě adresa, na níž běží samotný backend, avšak ta se může měnit podle zařízení, na kterém je backend spuštěn, proto ji sem uvádět nebudu. Každý z těchto kmenů dále větví jeden ze souborů uložených ve složce *routes*. Pro **/foods** je to *foods.js*, pro **/users** je to *users.js* a pro **/diaryEntries** *diaryEntries.js*. Dále je v *app.js* také definový error handler. Nad *app.js* stojí pouze serverový soubor *www* ve složce *bin*. Ten udává, na jaké url a na jakém portu poběží backend, a spouští na něm server s Expressovou aplikací. Kromě toho jsem přidal do tohoto souboru také funkci, která se připojuje k databázi při spuštění aplikace. Tato funkce se nachází v souboru *db.js* ve složce *accessories* a využívá funkci *mongoose.connect* k tomu, aby se připojila k databázi na url, které je definováno v souboru *secret.js*. Kromě url, na němž běží databáze jsou v souboru *secret.js* také uloženy přihlašovací údaje na připojení k databázi a klíč, který je užíván k autentizaci uživatelů (více v kapitole 3.2.1 Uživatelské endpointy). Soubor *secret.js* je lokální, což znamená, že nedochází k jeho sdílení na githubu kvůli tomu, aby nikdo nemohl ohržet tato data tím, že se na githubu podívá na kód aplikace. Dále jsou součástí backendu soubory *package.json*, *package-lock.json* a všechny soubory ve složce *node_modules*, které jsou vytvořené Node Package Managerem a které obstarávají instalaci, aktualizaci a používání modulů aplikace. To jsou všechny soubory, které mají na starosti spuštění, chod a nastavení backendu. Ostatní soubory zajišťují samotnou funkcionalitu aplikace. Soubory ve složce *models* definují, jak mají vypadat dokumenty uložené v databázi, a soubory ve složce *routes* definují endpointy, na nichž se vyřizují uživatelské požadavky, a samotné vyřizování těchto požadavků. Tyto soubory jsou podrobně popsány v následujících kapitolách.



Obr. 3: Struktura backendu

4.1 Modely

Tato podkapitola je zasvěcena popsání tří modelů, na nichž je celý můj projekt postaven, a jejich atributů. Pojmeme modely se v mé aplikaci rozumí schémata knihovny mongoose, která specifikují strukturu dokumentů v databázi MongoDB. Všechny tyto modely jsou uloženy ve složce *models*.

4.1.1 Uživatel

V souboru *User.js* (viz obr. 4) se nachází schéma, které diktuje podobu objektů, jež reprezentují v databázi uživatele. Každý uživatel musí mít čtyři atributy, přičemž tři z těchto atributů jsou typu String a jeden je typu Object. Prvním z nich je email, který musí být pro daného uživatele unikátní, protože se pomocí něho přihlašuje do frontendu. Stejně tak musí být unikátní i druhý atribut, uživatelské jméno, jenž slouží k identifikaci jednotlivých uživatelů v uživatelském rozhraní. Poslední atribut typu String, heslo, unikátní být nemusí, avšak je speciální v tom, že se do databáze nesmí uložit jako prostý text. Kdyby bylo heslo uloženo do databáze jako prostý text, hrozilo by odcizení účtu uživatele, proto se nejprve šifruje. Šifrování probíhá pomocí knihovny Bcrypt, která heslo promění skrze hashovací funkci na hash, což je v našem případě dlouhý řetězec, jenž nemá s původním heslem nic společného. Tento řetězec ještě obohatí tzv. sůl (salt), která přidává další znaky do hashe a znemožňuje útoky pomocí duhové tabulky nebo hrubé síly. Work factor soli je v tomto případě 10. Více o hashovací funkci bcrypt naleznete v citacích (Arias, 2018). K hashování dochází před každým uložením dokumentu do databáze, tudíž se nemůže stát, že by v databázi bylo uloženo heslo jako prostý text. Hashování je jednostranné, proto se musí heslo hashovat i při přihlašování, při němž musí dojít k porovnání hashe zadaného hesla se zahashovaným heslem uloženým v databázi. Jak funkce na hashování před ukládáním dokumentu, tak funkce na porovnávání hesel jsou také uloženy v tomto souboru. Poslední atribut typu Object, intakeGoal, reprezentuje nutriční cíl každého uživatele tzn. kolik kilokalorií a gramů makronutrientů by měl každý den uživatel zkonzumovat. Ve výchozím stavu je to prázdný objekt a uživatel si ho může nastavit pomocí uživatelského prostředí.

```
let userSchema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
  username: {
    type: String,
    required: true,
    unique: true,
  },
  intakeGoal: {
    kcal: Number,
    protein: Number,
    carbs: Number,
    fat: Number,
    fibre: Number,
    required: false,
    default: {},
  }
});
```

Obr. 4: Schéma pro dokument uživatele

4.1.2 Jídla

Obdobně jako s uživateli je v souboru *Food.js* (viz obr. 5) uloženo schéma, jenž popisuje podobu objektů, které v databázi představují jídla. Toto schéma má 4 povinné položky: název a uživatelské jméno autora, jež jsou obě typu řetězec, poté nutriční hodnotu, která se dále dělí na 5 podpoložek – čísla určující počet kilokalorií, bílkovin, sacharidů, tuků a vlákniny; a id autora. Navíc se skládá ještě ze dvou nepovinných položek: pole ingrediencí a řetězce popisu, které když uživatel nezadá, tak nabydou výchozích hodnot (pole bude prázdné a řetězec bude „Bez popisu“). Toto schéma definuje jak ingredience, tak pokrmy, jelikož jediný rozdíl mezi těmito dvěma typy dokumentů je, že pokrmy mají ingredience, kdežto ingredience se z dalších ingrediencí neskládají.

4.1.3 Deníčkové záznamy

Podobně jako v předešlých případech předepisuje soubor *DiaryEntry.js* (obrázky 6 a 7) podobu dokumentů, jež reprezentují deníčkové záznamy v databázi. Stejně jako u schématu jídel jsou součástí deníčkových záznamů dva atributy popisující autora zápisu, jimiž jsou uživatelské jméno autora a id autora. Dalšími povinnými položkami jsou datum, pro které byl zápis vytvořen, shrnutí nutričních hodnot pro daný den a pole chodů, jež uživatel v tento den zkonzumoval. Jelikož jsou součástí deníčkových zápisů, mají chody vlastní schéma, přestože se do databáze chody samy o sobě neukládají. Každý chod má vlastní nutriční hodnoty, strukturou shodné s těmi u jídel, a řetězec název, který nabývá pouze několika předepsaných hodnot (snídaně, dopolední svačina, oběd, odpolední svačina, večeře a ostatní), a nakonec složení, jež popisuje z jakých jídel se chod skládal. Atributy chodů mají výchozí hodnoty, protože uživatel není povinen stravovat se šestkrát denně. Tyto výchozí hodnoty jsou následující: složení je prázdný řetězec, položky nutričních hodnot nabývají hodnoty 0 a název je jeden z 6 výše zmíněných chodů. Poslední položka schématu popisující deníčkové zápisy je vyhrazena aktivitám a je nepovinná. Má dva vlastní atributy – číslo spálených kilokalorií a řetězec popis. Výchozími hodnotami těchto atributů je 0 a „Žádné aktivity“ respektive. Stejně jako u jídel není potřeba žádné šifrování.

```
let foodSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  nutritionVal: {
    kcal: Number,
    protein: Number,
    carbs: Number,
    fat: Number,
    fibre: Number,
  },
  ingredients: {
    type: Array,
    required: false,
    default: [],
  },
  desc: {
    type: String,
    required: false,
    default: "Bez popisu"
  },
  author: {
    type: Schema.ObjectId,
    ref: 'User',
    required: true
  },
  authorUsername: {
    type: String,
    required: true
  }
});
```

Obr. 5: Schéma pro dokument jídla


```

let diaryEntrySchema = new Schema({
  date: {
    type: Date,
    required: true
  },
  meals: [mealSchema],
  nutritionSummary: {
    kcal: Number,
    protein: Number,
    carbs: Number,
    fat: Number,
    fibre: Number
  },
  activities: {
    kcal: {type: Number, required: false, default: 0},
    description: {type: String, required: false, default: "Žádné activity"}
  },
  author: {
    type: Schema.ObjectId,
    ref: 'User',
    index: true,
    required: true
  },
  authorUsername: {
    type: String,
    required: true
  }
});

```

Obr. 6: Schéma pro deníkové zápisy

```

let mealSchema = new Schema({
  kcal: {type: Number, default: 0},
  protein: {type: Number, default: 0},
  carbs: {type: Number, default: 0},
  fat: {type: Number, default: 0},
  fibre: {type: Number, default: 0},
  name: {type: String},
  contents: {type: Array},
});

```

Obr. 7: Schéma pro chody

4.2 Endpointy

K popsání endpointů backendu mé aplikace slouží tato podkapitola. Endpointy jsou definovány pomocí rozcestníků (routers) v souboru *app.js* a samotnými metodami v těchto rozcestnících. Má aplikace má tři rozcestníky, přičemž všechny jsou uloženy ve složce *routes*. Jedná se o soubory: *foods.js*; *users.js* a *diaryEntries.js*. Každý endpoint je spjat s funkcí, která je volána, když na tento endpoint přijde požadavek. Za účelem zjednodušení popisu backendu odkazují v dokumentaci na endpoint i na funkci, jež je s ním spjata, pouze jako na endpoint.

4.2.1 Uživatelské endpointy

Uživatelské endpointy jsou čtyři a jsou klíčové pro spravování informací o uživateli. Endpoint na url **/users/login** slouží pro přihlašování uživatelů, endpoint na url **/users/register** zajišťuje registraci nových uživatelů, endpoint na url **/users/intake** má na starosti výpočet nutričního cíle pro uživatele a poslední endpoint na url **/users/userInformation** získává z databáze uživatelské informace (uživatelské jméno, email a nutriční cíl). Všechny tyto endpointy operují s uživatelskými dokumenty v databázi a jejich schématy.

Endpoint pro registraci používá metodu post a dostává v těle požadavku (request body) email, uživatelské jméno, heslo a heslo pro kontrolu. Nejprve zkontroluje, zdali v databázi již neexistuje uživatel s daným emailem nebo uživatelským jménem. Pokud existuje, pak odešle stavový kód 400 jako opověď. V případě, že se takový uživatel v databázi nenachází, zkontroluje, jestli se obě přijatá hesla shodují. Když se neshodují, odešle odpověď v podobě stavového kódu 401. V opačném případě vytvoří objekt podle schématu uživatele a přiřadí mu příslušné uživatelské údaje. Tento objekt je následně po šifrování hesla uložen do databáze, načež se odešle odpověď v podobě stavového kódu 200. Poslední možností je, že dojde při zapisování do databáze k chybě. Za takovýchto okolností se jako odpověď odešle stavový kód 500.

Stejně jako endpoint pro registraci používá endpoint pro přihlašování metodu post a dostává přihlašovací údaje v těle požadavku. Tentokrát ovšem stačí pouze email a heslo. Email se použije pro vyhledání objektu uživatele v databázi, přičemž pokud dojde při vyhledávání k chybě, odešle endpoint odpověď v podobě stavového kódu 500, a pokud žádný uživatel s daným emailem v databázi není, odešle se stavový kód 400. Stejně tak se odešle stavový kód 400, neshoduje-li se heslo uložené v databázi s heslem přijatým v požadavku. Pouze když se hesla shodují, dochází k autentizaci. Autentizace spočívá ve vytvoření unikátního hashe pro uživatele, který se nazývá authentication token. K autentizaci používá aplikace knihovnu passport-jwt. Tento hash poté uživatel používá, aby se prokázal serveru např.: při vytváření nebo při upravování dokumentů v databázi. Email, uživatelské jméno a authentication token jsou součástí JSON objektu, jenž je odeslán jako odpověď na správný požadavek. Dodatečné informace o knihovně passport-jwt a authentication tokenech můžete nalézt v citacích (Vanyan, 2017).

I endpoint pro vytváření nutričního cíle uživatele používá metodu `post` a očekává informace v těle požadavku. Tělo požadavku musí obsahovat pět atributů: věk, váhu, výšku, pohlaví a míru aktivity uživatele. Navíc se musí uživatel prokázat `backednu` skrze `authentication token`, který obsahuje hlavička požadavku. Data obdržená v požadavku poté použije endpoint k tomu, aby vytvořil nutriční cíl uživateli a uložil ho do databáze. Na výpočet kilokalorií slouží rovnice Mifflin-St Jeor. Kilokalorie jsou nutné na výpočet příjmu makronutrientů. Každý makronutrient by měl tvořit určitý díl denního příjmu kilokalorií. O poměru, v němž by se makronutrienty měly podílet na příjmu kalorií a rovnici Mifflin-St Jeor se můžete dozvědět více v citacích (Kubala, 2018). Gram sacharidů a bílkovin má čtyři kilokalorie, gram tuků má osm kilokalorií a gram vlákniny nemá kilokalorie žádné (doporučený příjem vlákniny nezáleží na příjmu kilokalorií). Z informací o tom, kolik kilokalorií mají gramy jednotlivých makronutrientů, a velikosti dílů, jež by jednotlivé makronutrienty měly zaujímat na příjmu celkových kilokalorií, se následně vypočítá nutriční cíl a uloží se do databáze. Pokud endpoint v požadavku neobdrží všechna potřebná data, je odpovědí stavový kód 400. Dojde-li k chybě při ukládání nutričního cíle do databáze, je odpovědí stavový kód 500, a proběhne-li všechno v pořádku, je odpovědí stavový kód 200.

Poslední endpoint slouží pro získávání uživatelských informací z databáze. Používá metodu `get` a vyžaduje pouze, aby v hlavičce požadavku byl `authentication token`. `Authentication token` poté používá k získání uživatelských informací daného uživatele z databáze, které následně odesílá jako JSON v odpovědi.

4.2.2 Jídelní endpointy

Jídelní endpointy jsou dohromady tři a zodpovídají za čtení, mazání a ukládání objektů reprezentující jídla, proto pracují se schématy, jež diktují podobu těchto objektů. Endpoint zodpovědný za čtení jídel se nachází na url `/foods/`, endpoint zodpovědný za mazání jídel na url `/foods/delete` a endpoint, který má na starost vytváření jídel, na url `/foods/create`.

Jediný z těchto endpointů, který nevyžaduje autentizaci, je endpoint určený pro čtení jídel. Používá metodu `get` a parametry přijímá ve formě tzv. dotazu (query). Tento dotaz je součástí url tohoto endpointu a obsahuje dva parametry – `name` (název pokrmu) a `skipNumber` (číslo dokumentů, které se mají při čtení přeskočit). Příklad požadavku s dotazem vypadá takto: `/foods/?name=mrkev&skipNumber=0`. V tomto příkladu je název pokrmu mrkev a číslo dokumentů, které se mají přeskočit 0. Čtení stovek jídel z databáze se stejným jménem najednou by bylo velmi časově i paměťově náročné, proto se čtení dělí na více požadavků (jeden požadavek čte v současnosti 10 záznamů). Číslo `skipNumber` je z tohoto důvodu pro čtení potřebné a udává, od kolikátého dokumentu se mají jídla číst, přičemž pro 0 by to bylo prvních 10 záznamů, pro 10 by to byly záznamy s čísly 11-20 atd. Název pokrmu může být prostý řetězec nebo regulární výraz. Pokud dojde k chybě při čtení dokumentů odešle se jako odpověď stavový kód 500, v opačném případě je to objekt JSON s polem přečtených jídel a booleanem, který říká, jestli jsou po těchto přečtených jídlech ještě nějaká jídla uložena v databázi, nebo jestli tato jídla byly poslední položky v ní uložené.

Endpoint pro mazání jídel využívá metodu `delete`. Součástí požadavku musí být tělo, v němž je uloženo id jídla, které má tento endpoint z databáze smazat, a hlavička, v níž je uložený authentication token. Pomocí funkce `passport`, která je uložena v souboru `passport.js`, získá tento endpoint uživatele, jemuž odpovídá hash authentication token. Pokud endpoint neobdrží potřebné informace v požadavku, odešle stavový kód 400 jako odpověď, v opačném případě vyhledá pomocí těchto informací příslušnou položku a smaže ji z databáze. Při mazání může dojít k chybě, která se projeví stavovým kódem 500 v odpovědi. Jídlo s příslušným id a autorem se v databázi také nemusí nacházet, což způsobí, že se v odpovědi objeví stavový kód 400. Pokud vše proběhne v pořádku, sestává odpověď ze stavového kódu 200.

Poslední endpoint pro vytváření jídel využívá metodu `post` a stejně jako endpoint na mazání jídel vyžaduje, aby součástí požadavku byl authentication token v hlavičce. V těle požadavku se musí nacházet atributy `name` (název jídla) a `nutritionalVal` (jeho nutriční hodnoty), nebo `ingredients` (jeho ingredience). V případě, že se v požadavku vyskytuje atribut `ingredients`, dopočítává si položku `nutritionalVal` endpoint sám z těchto ingrediencí. Dále může požadavek obsahovat atribut `desc`, který reprezentuje popis jídla, ale ten je nepovinný. Pomocí schématu pro jídlo z těchto parametrů a uživatele, kterého získá endpoint funkcí `passport` a authentication tokenu, vytváří objekt, který je následně uložen do databáze. Když dojde při ukládání objektu k chybě, odešle se jako odpověď stavový kód 500, a když uložení proběhne úspěšně odešle se stavový kód 200.

4.2.3 Endpointy pro deníčkové záznamy

Stejně jako jídelní endpointy jsou i endpointy pro deníčkové záznamy 3: **`diaryEntries/get`**; **`diaryEntries/create`** a **`diaryEntries/delete`**. Zastávají i ty samé funkce jako jídelní endpointy, tedy čtení, vytváření a mazání dokumentů respektive, ale na rozdíl od jídelních endpointů pracují s objekty, které reprezentují deníčkové záznamy a které jsou definovány schématem pro deníčkové záznamy.

Endpoint pro čtení deníčkových záznamů používá metodu `get` a dostává jediný parametr `date` skrze dotaz. Zároveň vyžaduje authentication token v hlavičce požadavku, jelikož více uživatelů může mít deníčkové záznamy pro ty samé dny a endpoint musí poznat, záznamy kterého uživatele má přečíst. Tento parametr může mít dvě podoby – může to být jedno datum, nebo pole dat o dvou položkách. Když je `date` jedno datum, tak endpoint přečte položku pro dané datum a uživatele, kterého získá z authentication tokenu. Pokud se jedná o pole, pak první datum určuje začátek intervalu dnů, pro které chce uživatel deníčkové záznamy zobrazit, a druhé datum tento interval uzavírá. Pro ilustraci pro parametr `date=[22. 2. 2020; 28. 2. 2020]` by přečetl endpoint záznamy pro dny 22. 2. 2020, 28. 2. 2020 a všechny dny mezi těmito dvěma daty. Ve skutečnosti je zápis dat trochu komplikovanější – pro příklad udávám požadavek, který má jako parametr jedno datum, **`/diaryEntries/get/?date=%222020-03-11T23:00:00.000Z%22`**. Tyto přečtené záznamy endpoint dále zpracovává a pomocí jejich položky `nutritionSummary` dopočítá, kolik gramů makronutrientů a kolik kilokalorií zbývá do nutričního cíle, který si uživatel může nastavit. Pokud vše proběhne v pořádku, je odpovědí pole objektů ve formátu JSON, v němž je pro každý

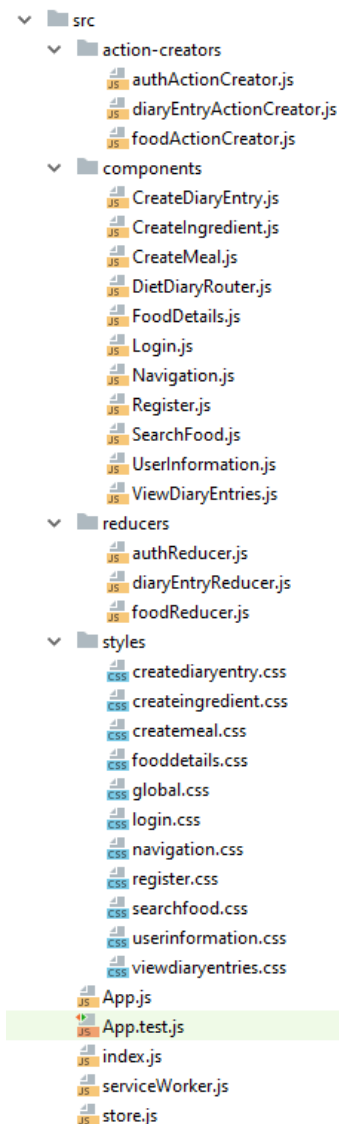
den z intervalu právě jeden objekt s atributy, které popisují zkonsumované nutriční hodnoty v daný den, kolik jich zbývá do nutričního cíle, a datum tohoto dne. Když je vstupním parametrem jedno datum, dojde k vyhledání zápisu pomocí id uživatele obdrženého z authentication tokenu a vstupního parametru. Tento jediný zápis je také obohacen nutričním cílem, avšak nijak dále se již nezpracovává a je odeslán jako odpověď ve formátu JSON. Dojde-li k chybě během vyhledávání dokumentů, odešle se jako odpověď na požadavek stavový kód 500.

Endpoint na mazání deníčkových záznamů používá metodu delete, přičemž vyžaduje v těle požadavku id deníčkového záznamu, jenž má být odstraněn z databáze, a v hlavičce authentication token. Skrz funkci passport, kterému předá endpoint authentication token, dostane objekt uživatele, jehož id použije společně s id záznamu k vyhledání požadovaného deníčkového záznamu a jeho následnému smazání. Proběhne-li vše v pořádku odešle se stavový kód 200, chybí-li v požadavku potřebná data nebo nenachází-li se v databázi dokument, jenž by odpovídal datům v požadavku, odešle stavový kód 400 nebo dojde-li k chybě během mazání deníčkového záznamu je odpovědí stavový kód 500.

Poslední endpoint používá metodu post a slouží k ukládání nových deníčkových záznamů. Stejně jako předchozí endpointy vyžaduje, aby byl v hlavičce požadavku přítomen authentication token. Kromě toho musí být přítomny v těle požadavku atributy určující datum deníčkového záznamu a stravu uživatele v daný den (pole objektů, v němž každý objekt má právě dva atributy – počet gramů zkonsumovaného jídla a objekt představující toto jídlo). Dále může uživatel v požadavku také zaslat záznam o svých aktivitách (objekt o právě dvou attributech – počtu splánených kalorií a popisu této aktivity), tato položka je však nepovinná. Z těchto vstupních dat poté dopočítá endpoint atribut nutritionSummary, který představuje přehled zkonsumovaných kilokalorií a maktrnutrientů v daný den. Dále je součástí deníčkového zápisu nutriční cíl, který endpoint získává skrze authentication token. Nemá-li uživatel nutriční cíl nastavený, jedná se o 0 kilokalorií a 0 gramů každého makronutrientu. V opačném případě se použije vypočítaný nutriční cíl. Následně vytvoří endpoint pomocí obdržených dat v požadavku a dat, která si sám vypočítal, deníčkový zápis. Pokud vše proběhne úspěšně odpovědí je patřičný deníčkový zápis jako JSON objekt, dojde-li k chybě během vytváření zápisu je odpovědí stavový kód 500 a je-li chyba ve vstupních datech je odpovědí stavový kód 403.

5. FRONTEND

Frontend mé aplikace slouží jako uživatelské rozhraní pro zasílání požadavků backendu a zobrazování dat, jež od backendu přijme, přičemž má standardní strukturu React-Redux aplikace. Komponentou, která propojuje všechny ostatní soubory, je v souboru *App.js*. V ní se nachází komponenta *Provider*, která zajišťuje, že všichni její potomci budou mít přístup k reduxovému úložišti, které dostává tato komponenta jako props. Dále se v ní nachází klíčová komponenta *BrowserRouter* z knihovny *react-router-dom*, která společně s komponentou *DietDiaryRouter* ve složce *components* určuje, na jakých url se mají vykreslovat jaké komponenty. Nad *App.js* stojí pouze *index.js*, který tuto reactovou komponentu renderuje do prohlížeče. Reduxové úložiště je inicializováno v souboru *store.js*, kde dochází k sjednocení reducerů ze složky *reducers* a vytvoření patřičných atributů úložiště. S reducery komunikují pouze funkce ze souborů, jež se nazývají action creators a jež se nachází ve složce *action-creators*. V těchto funkcích je totiž volána reduxová funkce *dispatch* s parametry, které se nazývají akce. Akce obsahují data, která mají být uložena do reduxového úložiště, a data, která blíže specifikují, do jakých atributů úložiště mají být tato data zapsána. Funkce ze souborů action creators volá uživatel pomocí uživatelského rozhraní, které vytváří samotné reactové komponenty, jež jsou uloženy ve složce *components*. Na komponenty navazují kaskádové styly, což jsou soubory, které mají na starosti vizuální stránku uživatelského rozhraní vytvářeného komponentami. Tyto soubory se nacházejí ve složce *styles* a většina komponent má právě jeden vlastní soubor s kaskádovými styly, jehož název je shodný s názvem komponenty. Výjimkou je komponenta *DietDiaryRouter*, která pouze specifikuje, na jakých url se mají vykreslovat jaké komponenty, a která sama o sobě žádnou grafickou podobu nemá. Druhou výjimkou je soubor *global.css*, v němž jsou definovány proměnné (barvy) a jiné kaskádové styly (např.: font), které platí pro všechny komponenty.



Obr. 8: Struktura frontendu

5.1 Reducers

V mé aplikaci rozlišuji 3 reducery, přičemž každý zachází s jedním typem objektů, které jsou definovány pomocí modelů v backendu. Reducer v souboru *foodReducer.js* zachází s objekty, které představují jídla, reducer v souboru *authReducer.js* zachází s objekty, které představují uživatele, a reducer v souboru *diaryEntryReducer.js* zachází s objekty, které představují deníčkové zápisy. Každý reducer (viz obr. 9) se skládá z proměnné *initialState* a funkce, která obrátí jako parametr *initialState* a akci. V této funkci je poté mnoho případů (cases), jež blíže specifikují, co se má do atributů v úložišti zapsat. Která case se spustí určuje typ akce (*action.type*), jenž se zadává akci při jejím dispatchování. Typů akcí je mnoho, proto zde nebudou všechny podrobně vypsané, dojde pouze k nastínění obsahu atributů úložiště, které akce mění.

5.1.1 DiaryEntryReducer

DiaryEntryReducer má v proměnné *initialState* 3 položky: *deleteMessage*, *searchedDiaryEntries* a *createMessage* (viz obr. 9). Do atributu *deleteMessage* se ukládá zpráva o tom, zdali bylo smazání záznamu v deníčku úspěšné. Podobné je to s položkou *createMessage*, do které se ukládá zpráva o tom, jestli v pořádku proběhlo vytvoření nového záznamu v deníčku. Tato zpráva se z úložiště následně může také mazat. Do atributu *searchedDiaryEntries* se vkládají deníčkové záznamy přijaté od backendu, když si o ně uživatel zažádá.

```
const initialState = {
  searchedDiaryEntries: null,
  deleteMessage: "",
  createMessage: "",
};

export default function (state = initialState, action) {
  switch (action.type) {
    case 'SEARCHED_DIARYENTRY_TO_STATE':
      return {
        ...state,
        searchedDiaryEntries: action.payload,
      };
    case 'SEARCHED_DIARYENTRY_CLEANUP':
      return {
        ...state,
        searchedDiaryEntries: null,
      };
    case 'DELETE':
      return {
        ...state,
        searchedDiaryEntries: null,
        deleteMessage: action.payload,
      };
    case 'CREATE_DIARYENTRY_TO_STATE':
      return {
        ...state,
        createMessage: action.payload.message,
      };
    case 'MESSAGE_CLEANUP':
      return {
        ...state,
        createMessage: "",
      };
    default:
      return state
  }
}
```

Obr. 9: *diaryEntryReducer*

5.1.2 FoodReducer

InitialState *foodReduceru* se skládá z 5 atributů (viz obr. 10). Pole *foods* uchovává všechna jídla přijatá od backendu při vyhledávání jídel. S přijatými jídly souvisí také atribut *isEmpty*, který vypovídá o tom, je-li pole přijatých objektů prázdné, a atribut *last*, jenž udává, zdali jsou v databázi ještě další dokumenty, nebo zdali toto byly položky s daným jménem poslední. *SearchedFood* je položka, do níž se ukládá jídlo, které si chce uživatel prohlédnout detailně. Poslední atribut *deleteMessage* představuje zprávu, jež sděluje uživateli, jestli bylo odstranění jídla z databáze úspěšné.

```
const initialState = {
  foods: [],
  last: false,
  isEmpty: false,
  searchedFood: null,
  deleteMessage: "",
};
```

Obr. 10: *foodReducer initialState*

5.1.3 AuthReducer

InitialState authReduceru je tvořen 8 atributy. Atributy username a email uchovávají uživatelské jméno a email přihlášeného uživatele respektive. Boolean loggedIn indikuje, je-li uživatel přihlášen. Podobnou funkci má i boolean registered, který uchovává informaci o tom, zdali byl uživatel zaregistrován, či nikoliv. Do položek loginMessage, registerMessage a goalMessage jsou ukládány zprávy o tom, jestli bylo přihlášení, registrace nebo vytvoření nutričního cíle uživatele úspěšné. Do poslední položky userGoal se ukládá nutriční cíl uživatele, když si o jeho přečtení uživatel zažádá.

```
const initialState = {
  loggedIn: false,
  username: "",
  email: "",
  userGoal: null,
  registerMessage: "",
  loginMessage: "",
  goalMessage: "",
  registered: false
};
```

Obr. 11: authReducer initialState

5.2 Action creators

Action creators jsou soubory plné funkcí, z nichž každá volá reduxovou funkci dispatch, která posílá akce do reducerů. Pro každý reducer existuje v mém projektu jeden action creator: *authActionCreator.js* dispatchuje akce do *authReduceru*, *foodActionCreator.js* do *foodReduceru* a *diaryEntryActionCreator.js* do *diaryEntryReduceru*. V action creators také dochází ke komunikaci mezi frontendem a backendem prostřednictvím funkce fetch, do níž se, mimo jiné, zadávají url jednotlivých backendových endpointů. Některé funkce v action createorech jsou velmi komplexní, proto zde nebudou zevrubně popsány, nýbrž dojde pouze k přiblížení problematiky, kterou tyto funkce řeší.

5.2.1 DiaryEntryActionCreator

V souboru *diaryEntryActionCreator.js*, který se nachází ve složce *action-creators*, je 5 funkcí. Funkce *getDiaryEntries* komunikuje s backendovým endpointem **diaryEntries/get/**, kterému posílá prostřednictvím query svůj vlastní parametr *date* a od kterého dostává v odpovědi pole deníčkových zápisů. Toto pole poté posílá prostřednictvím akce do *diaryEntryReduceru*, který ho ukládá do atributu *searchedDiaryEntries*. Funkce *createDiaryEntry* obdobně komunikuje s endpointem **diaryEntries/create/**, jemuž zasílá v těle požadavku objekt, reprezentující nový zápis do deníčku, který má být uložen do databáze. Od tohoto endpointu dostává jako odpověď stavový kód, podle něhož vytváří zprávu, kterou posílá skrze akci do *diaryEntryReduceru*. Tam je poté uložena do položky *createMessage*. Poslední funkce komunikující s backendem se nazývá *deleteDiaryEntry*. Tato funkce posílá požadavky na endpoint **diaryEntries/delete/**, kterému zasílá v těle požadavku id deníčkového záznamu, který má být vymazán, přičemž stejně jako dvě předešlé funkce odesílá backendu i authentication token v hlavičce požadavku. Jako odpověď dostává stavový kód, podle něhož vytváří zprávu, která je skrze akci uložena do atributu *deleteMessage* v *diaryEntryReduceru*. Funkce *messageCleanUp* prostřednictvím akce maže položku *createMessage* v *diaryEntryReduceru* a funkce *diaryEntryCleanUp* obdobně nastavuje položku *searchedDiaryEntries* na hodnotu null.

5.2.2 FoodActionCreator

FoodActionCreator uložený v souboru *foodActionCreator.js* má 6 funkcí, z nichž 3 komunikují s backendovými endpointy. První z těchto funkcí je funkce *getFoods*, která dostává jako parametr název jídla a číslo, které udává kolik dokumentů má být při čtení z databáze přeskočeno. Oba tyto parametry předává ve formě query backendovému endpointu na url **/foods/**. Odpovědí je pole jídel, které dispatchuje v jako akci do *foodReduceru*, v němž je toto pole uloženo do atributu s názvem *foods*. Další funkcí komunikující s backendem je funkce *createFood*, která vyžaduje jídlo jako parametr. Tento parametr poté posílá v těle požadavku společně s authentication tokenem v hlavičce požadavku backendovému endpointu **/foods/create**. Jako odpověď dostává JSON uloženého jídla, nebo stavový kód s chybou. Poslední funkcí ve *foodActionCreatoru*, jež komunikuje s backendem je funkce *deleteFood*. Tato funkce jako parametr dostává jídlo, jež má být smazáno z databáze, načež ho společně s authentication tokenem zasílá backendovému endpointu **/foods/delete**. Od backendu obdrží ve formě odpovědi stavový kód, který je posléze základem pro tvorbu zprávy, jež je uložena prostřednictvím akce do atributu *deleteMessage* ve *foodReduceru*. Funkce *totalCleanUp* a *searchedFoodCleanUp* nastavují skrze akci určité atributy objektu *initialState* na jejich výchozí hodnoty. V případě funkce *searchedFoodCleanUp* je to položka *searchedFood* a v případě funkce *totalCleanUp* to jsou položky: *last*, *isEmpty*, *searchedFood* a *foods*. Poslední funkce se nazývá *searchedFoodToState* a jako parametr dostává jídlo, které prostřednictvím akce ukládá do atributu úložiště *searchedFood*.

5.2.3 AuthActionCreator

AuthActionCreator zodpovídá za dispatchování akcí, jež souvisejí s manipulací s uživatelskými informacemi, přihlašováním, odhlašováním a registrováním uživatele. Má sedm funkcí, z nichž čtyři komunikují s backendovými endpointy. Těmito funkcemi jsou *login*, *register*, *getUserInformation* a *createUserGoal*. *login* dostává jako parametr přihlašovací údaje, které posílá v requestu na endpoint **users/login**. Jako odpověď obdrží email, uživatelské jméno a authentication token v JSON objektu, přičemž email a uživatelské jméno uloží jak skrze akci do atributů *email* a *username* v *authReduceru*, tak do *localStorage*. V *authReduceru* dojde také touto akcí ke změně hodnoty položky *loggedIn* na *true*. Authentication token je uložen pouze do *localStorage*, odkud je získáván pokaždé, když je vyslán požadavek, jenž ho potřebuje. Funkce *login* může však obržet také stavový kód ohlašující chybu. Stane-li se tato možnost, dojde k vytvoření zprávy a jejímu uložení skrze akci do atributu *loginMessage* reduxového úložiště. *Register* také dostává přihlašovací údaje jako parametry a stejně jako *login* je posílá ve formě požadavku na backendový endpoint. V tomto případě je to ovšem endpoint **users/register**. Jako odpověď posléze dostává stavový kód, který je využit k vygenerování patřičné zprávy pro uživatele a booleanu, jenž určuje, zdali byl uživatel zaregistrován. Obě tyto proměnné jsou uloženy prostřednictvím akce do *authReduceru* – boolean do atributu *registered* a zpráva do atributu *registerMessage*. Funkce *getUserInformation* používá authentication token uložený v *localStorage* k tomu, aby poslala požadavek na endpoint **users/userInformation**. V odpovědi získává objekt s uživatelským jménem, emailem a nutričním cílem, jehož položky ukládá do atributů reduxového úložiště *username*, *email* a *userGoal* respektive. Poslední funkce

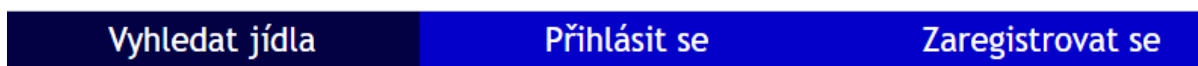
komunikující s backendem se nazývá `createUserGoal`. Tato funkce dostává v parametru údaje o uživateli – výšku, váhu, věk, pohlaví a míru aktivity, které následně posílá v těle požadavku společně s authentication tokenem na endpoint `/users/intake`. Odpovědí je stavový kód, podle kterého je vytvořena a dispatchnuta akce s patřičnými parametry. Tato akce mění v `authReduceru` položku `goalMessage`, jež informuje uživatele o tom, zdali bylo vytvoření nutričního cíle úspěšné. Zbývající tři funkce: `logout`, `loginCleanUp` a `registerCleanUp` slouží k tomu, aby nastavovaly různé atributy `authReduceru` skrze akce na jejich výchozí hodnoty. V případě funkce `loginCleanUp` to je to proměnná `loginMessage`, v případě funkce `registerCleanUp` to jsou proměnné `registerMessage` a `registered` a v případě funkce `logout` to jsou proměnné `loggedIn`, `username` a `email`. Kromě toho funkce `logout` také maže authentication header, `username` a `email` z `localStorage`, čímž odhlašuje uživatele z aplikace.

5.3 Komponenty

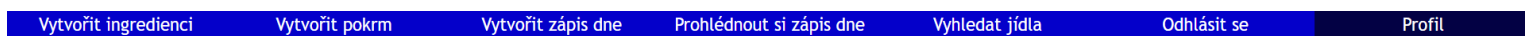
Všechny komponenty se nacházejí ve složce `components` a podílejí se na vykreslování vlastního html kódu uživatelského rozhraní. Jedinou výjimkou v tomto ohledu je komponenta `DietDiaryRouter`, která pouze určuje, na jakých url se mají renderovat jaké komponenty. U každé komponenty v této podkapitole bude zmíněno na jakém url se renderuje, přičemž se z důvodu vyhnutí se repetici v této kapitole neobjeví detailní pohled na komponentu `DietDiaryRouter`.

5.3.1 Navigation a UserInformation

Komponenta `Navigation` (viz obr. 12 a 13) slouží k pohybu mezi jednotlivými url pomocí grafického uživatelského rozhraní. Uživateli se zobrazuje jako pruh v horní části obrazovky a skládá se buď ze 3, nebo 7 položek podle toho, jestli je uživatel přihlášen, nebo ne. Když není uživatel přihlášen, má na výběr položky Přihlásit se, Zaregistrovat se nebo Vyhledat jídla. Je-li přihlášen může vytvořit ingredienci, pokrm, zápis do deníčku, prohlédnout si deníčkové zápisy, prohlédnout si svůj profil, vyhledat jídla nebo odhlásit se. Všechny tyto položky jsou odkazy na url s patřičnými reactovými komponentami, přičemž je položka odkazující na aktuální url podkreslena jinou barvou než ostatní položky. Komponenta `Navigation` se renderuje na všech url frontendu.



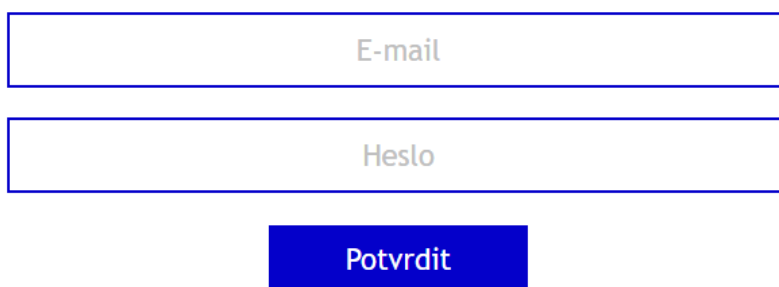
Obr. 12: Navigation, není-li uživatel přihlášen



Obr. 13: Navigation, je-li uživatel přihlášen

5.3.2 Login

Komponenta *Login* má na starosti nejen přihlašování uživatelů do aplikace, ale i jejich odhlašování. Když je uživatel nepřihlášený má podobu formuláře s tlačítkem Potvrdit a dvěma vstupními poli (viz obr. 14), přičemž jedno slouží na zadání emailu a druhé na zadání hesla. Po kliknutí na tlačítko Potvrdit zkontroluje komponenta, zadal-li uživatel zadal informace do obou polí, načež buď uživateli oznámí chybu ve vstupu, nebo zavolá funkci `logIn` se zadanými přihlašovacími údaji z *authActionCreatoru*. Podle odpovědi, kterou obdrží tato funkce od backendu, tato komponenta buď ukáže uživateli chybovou hlášku, nebo ho přihlásí a promění se na tlačítko Odhlásit se v navigaci. Komponenta *Login* se vykresluje na url **/login**.

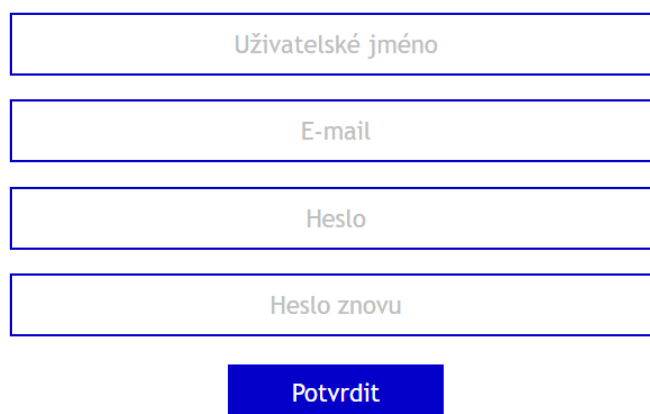


The diagram shows a login form with two input fields and a button. The first input field is labeled 'E-mail' and the second is labeled 'Heslo'. Below these fields is a blue button labeled 'Potvrdit'.

Obr. 14: Komponenta Login

5.3.3 Register

Register je komponenta, která obstarává registrování uživatelů do aplikace. Stejně jako komponenta *Login* je viditelná pouze, když není uživatel přihlášen, a sestává z formuláře, do něhož zadává uživatel své přihlašovací údaje, a tlačítka (viz obr. 15). Po kliknutí na tlačítko Potvrdit obdobně jako komponenta *Login* i komponenta *Register* zkontroluje, zdali uživatel zadal informace do všech polí. Kromě toho také kontroluje, jestli se shodují hesla a jestli má zadaná emailová adresa standardní emailovou formu. Podle výsledku této kontroly komponenta buď vykreslí uživateli chybovou hlášku, nebo zavolá funkci `register` z *authActionCreatoru*. Po přijetí odpovědi z backendu je uživateli sděleno, byl-li úspěšně zaregistrován, nebo vyskytla-li se při registrování nějaká chyba. Komponenta *Register* se vykresluje na url **/register**.



The diagram shows a register form with four input fields and a button. The fields are labeled 'Uživatelské jméno', 'E-mail', 'Heslo', and 'Heslo znovu'. Below these fields is a blue button labeled 'Potvrdit'.

Obr. 15: Komponenta Register

5.3.4 SearchFood

Komponenta *SearchFood* umožňuje uživateli vyhledávat v databázi jídla s daným jménem. Samotná se nachází na url končící /, tudíž je komponentou, která se uživateli vyrenderuje jako první. Kromě toho je však také součástí komponent *CreateMeal* a *CreateDiaryEntry*. Zároveň je to jediná komponenta, jež ukazuje záznamy z databáze a nevyžaduje k tomu přihlášení. Nejprve má podobu vstupního pole, do kterého uživatel může zadat buď název chodu, nebo regulární výraz. Po napsání výrazu do tohoto vstupního pole se zavolá funkce *getFoods* z *foodActionCreatoru* s tímto výrazem jako parametrem. Pokud se z backendu vrátí v podobě odpovědi pole jídel, vykreslí se názvy těchto jídel (viz obr. 16), na které když uživatel klikne, přesune se patřičné jídlo do *foodReduceru* pomocí funkce *searchedFoodToState* a vykreslí se místo komponenty *SearchFood* komponenta *FoodDetails*. Uživatel může zažádat o další nebo předchozí výsledky hledání pomocí tlačítek, která se nacházejí pod vyhledanými jídly.

mrkev
Mrkev special
Mrkev slozena
Mrkev Bezchybna
Mrkev uživatelská
Mrkev s popisem
Mrkev hola
Mrkev plná
Mrkev jednotna
Mrkev dvojita
Mrkev s krátkým popisem

Další výsledky

Obr. 16: Komponenta SearchFood

5.3.5 FoodDetails

K vykreslování detailních informací o vyhledávaných jídlech slouží komponenta *FoodDetails*. Informace o vyhledaném jídle získává z atributu *searchedFood*, který čte z *foodReduceru*. U ingrediencí ukazuje jejich název, nutriční hodnoty a popis, je-li u nich nějaký. To samé platí i pro pokrmy, avšak u těch vykresluje navíc ještě seznam jejich ingrediencí (viz obr. 17). Renderuje se vždy, když uživatel klikne na výsledek vyhledávání v rámci komponenty *SearchFood*, a na stejném url jako komponenta *SearchFood*. Její podoba se dynamicky mění podle údajů ukazovaného jídla. Pokud je přihlášený uživatel autorem hledaného jídla, má možnost své jídlo z databáze odstranit (zavolání funkce *deleteFood* z *foodActionCreatoru*) pomocí ikony popelnice, která se vykresluje v pravém horním rohu komponenty.

Mrkev plná				
Ingredience:				
Mrkev syrova 150 gramů Mrkev smazena 150 gramů Mrkev pečená 150 gramů Mrkev oloupaná 150 gramů				
Kalorické údaje na 100 gramů				
Kilokalorie	Bílkoviny	Sacharidy	Vláknina	Tuky
1769 kcal	57 gramů	51 gramů	111 gramů	32 gramů
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam erat volutpat. Nulla quis diam. Nunc tincidunt ante vitae massa. Nulla est. Duis bibendum, lectus ut viverra rhoncus, dolor nunc faucibus libero, eget facilisis enim ipsum id lacus. Aenean id metus id velit ullamcorper pulvinar. Aliquam ornare visi eu metus. Nulla quis diam. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. Aenean placerat. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat.				

Zpět

Obr. 17: Komponenta FoodDetails

5.3.6 CreateIngredient

První komponenta, která umožňuje uživateli ukládat nová jídla do databáze, se nazývá *CreateIngredient* a vykresluje se na url **/createIngredient**. Je designovaná pro vytváření objektů reprezentujících ingredience pomocí jednoduchého formuláře, který má 7 polí (viz obr. 18). Vstupem prvního pole je název ingredience typu řetězec, do druhého, třetího, čtvrtého, pátého a šestého pole zadává uživatel číslo odpovídající dané nutriční hodnotě na 100 gramů ingredience. Poslední dobrovolné pole je zasvěceno popisu ingredience. Po kliknutí na tlačítko Potvrdit komponenta zkontroluje, zdali uživatel vyplnil všechna povinná pole a zdali kontrolní součet všech makronutrientů nepřesahuje 100 gramů, protože není možné, aby makronutrienty 100 gramů ingredience vážily více než 100 gramů (toto neplatí u některých testovacích položek, které se mohou objevovat v této dokumentaci na obrázcích). Pokud je vše v pořádku, zavolá se funkce *createFood* z *foodActionCreatoru*. Po obdržení odpovědi se zobrazí uživateli buď chybová hláška, nebo zpráva informující ho o úspěšném vytvoření.

Jméno ingredience

Nutriční hodnoty na 100 gramů:

Kilokalorie

Sacharidy

Tuky

Bílkoviny

Vláknina

Nepovinné pole

Popis

Potvrdit

Obr. 18: Komponenta CreateIngredient

5.3.7 CreateMeal

Druhá komponenta, jež slouží k vytváření nových jídel v databázi, je *CreateMeal*. Narozdíl od komponenty *CreateIngredient* se renderuje na url **/createMeal** a neslouží k vytváření jídel reprezentujících ingredience, ale celých pokrmů. Pokrmům uživatel nezadáva žádné nutriční hodnoty, místo nich zadává pouze jejich složení, ze kterého se nutriční hodnoty dopočítávají v backendu. Skládat se pokrm může jak z ingrediencí, tak z jiných pokrmů. Proces vytváření pokrmu má 3 kroky. V prvním z nich uživatel zadá do vstupního pole název pokrmu a má možnost také zadat jeho popis. Druhý krok spočívá v zadávání složení pokrmu. To probíhá pomocí komponent *SearchFood* a *FoodDetails* – uživatel nejdříve ingredienci nebo pokrm vyhledá v komponentě *SearchFood* a v okně s komponentou *FoodDetails* zadá, kolik gramů daného jídla pokrm obsahuje, načež svůj vstup potvrdí tlačítkem Přidat ingredienci. Poslední třetí krok je přehled ingrediencí daného pokrmu (viz obr. 19) a zavolání funkce *createFood* z *foodActionCreatoru* pomocí tlačítka Potvrdit. V přehledu má uživatel ještě možnost odebrat nějaké ingredience, které přidal do pokrmu omylem pomocí tlačítek s ikonou popelnice. Mezi jednotlivými kroky se uživatel pohybuje pomocí tlačítek Další a Předchozí krok v horní části obrazovky. Navíc je součástí komponenty text, jenž uživatele informuje o tom, ve kterém kroku se zrovna nachází. Po obdržení odpovědi od backendu sdělí komponenta uživateli, byl-li jeho požadavek úspěšně vyřízen.



Obr. 19: Přehled ingrediencí pokrmu komponenty CreateMeal

5.3.8 CreateDiaryEntry

K vytváření zápisů do deníčku slouží komponenta *CreateDiaryEntry*, která se nachází na url `/createDiaryEntry`. Stejně jako *CreateMeal* má několik kroků. V prvním z nich uživatel zadává datum, pro které chce zápis vytvořit, k čemuž slouží veřejně dostupná knihovna *react-calendar*³. Ve druhém přiřazuje jednotlivým chodům (snídaně, dopolední svačina, oběd, odpolední svačina, večeře a ostatní) pokrmu nebo ingredience, jež jedl. Vyhledávání jídel probíhá prostřednictvím komponent *SearchFood* a *FoodDetails* a přepínání mezi jednotlivými chody je uskutečňováno pomocí tlačítek Další a Předchozí chod. Ve třetím kroku může uživatel zadat počet spálených kilokalorií aktivitami a přidat k těmto aktivitám popis; tento bod je ovšem nepovinný. Čtvrtý krok je opět zkontrolování zadaných údajů a případná možnost z jednotlivých chodů některé položky odebrat pomocí ikon popelnic přítomných u každé položky (viz obrázek 20). Mezi kroky se stejně jako u komponenty *CreateMeal* uživatel pohybuje pomocí tlačítek Předchozí a Další krok. Také je součástí komponenty *CreateDiaryEntry* text sdělující uživateli, ve kterém kroku se v současnosti nachází. Kvůli tomu, že je vytváření záznamu do deníčku komplikované, a kvůli tomu, že by uživateli tento úkon napoprvé mohl dělat potíže, je komponenta doplněná o nápovědu v prvním kroku. Po kliknutí na tlačítko Potvrdit dojde ke zkontrolování vstupu a zavolání funkce *createDiaryEntry* z *diaryEntryActionCreatoru*. Když dostane frontend od backendu odpověď na požadavek, vygeneruje se z této odpovědi zpráva pro uživatele a vykreslí se.

³ Knihovna *react-calendar* je veřejně dostupná z url <https://www.npmjs.com/package/react-calendar>.

Krok 4/4 - Přehled a potvrzení

Snídaně	Dopolední svačina	Oběd	Odpolední svačina	Večeře	Ostatní
Mrkev hola 153 gramů 🍷	Přidejte chod	Mrkev plná 365 gramů 🍷	Přidejte chod	Mrkev slozena 145 gramů 🍷	Přidejte chod
				Mrkev dvojita 133 gramů 🍷	

Potvrdit

Obr. 20: Přehled denních chodů komponenty CreateDiaryEntry

5.3.9 ViewDiaryEntries

Další komponenta mé aplikace s názvem *ViewDiaryEntries* poskytuje uživateli grafický nástroj k prohlížení deníčkových záznamů. Jediný vstup, který se od uživatele v této komponentě očekává, je datum nebo rozmezí dat, pro něž chce ukázat záznamy z databáze. Zadáání tohoto vstupu je realizováno prostřednictvím knihovny *react-calendar* (viz obr. 21). Tyto vstupní informace se odešlou pomocí funkce *getDiaryEntries* z *diaryEntryActionCreatoru* backendu, od kterého frontend jako odpověď přijímá buď jeden deníčkový záznam, nebo pole deníčkových záznamů. Komponenta *ViewDiaryEntries* poté promítne data z přijatých záznamů do grafu, který vykresluje veřejně dostupná knihovna *Recharts*⁴. Jedná-li se o jeden záznam a jsou-li dostupná podrobná data, vyrenderuje komponenta také ikonu popelnice, jež slouží k zavolání funkce *deleteDiaryEntry* a smazání daného záznamu v deníčku, tabulku s aktivitami a tabulku s nutričními hodnotami a složením jednotlivých chodů (viz obr. 22). *ViewDiaryEntries* se vykresluje na url **/viewDiaryEntry**.

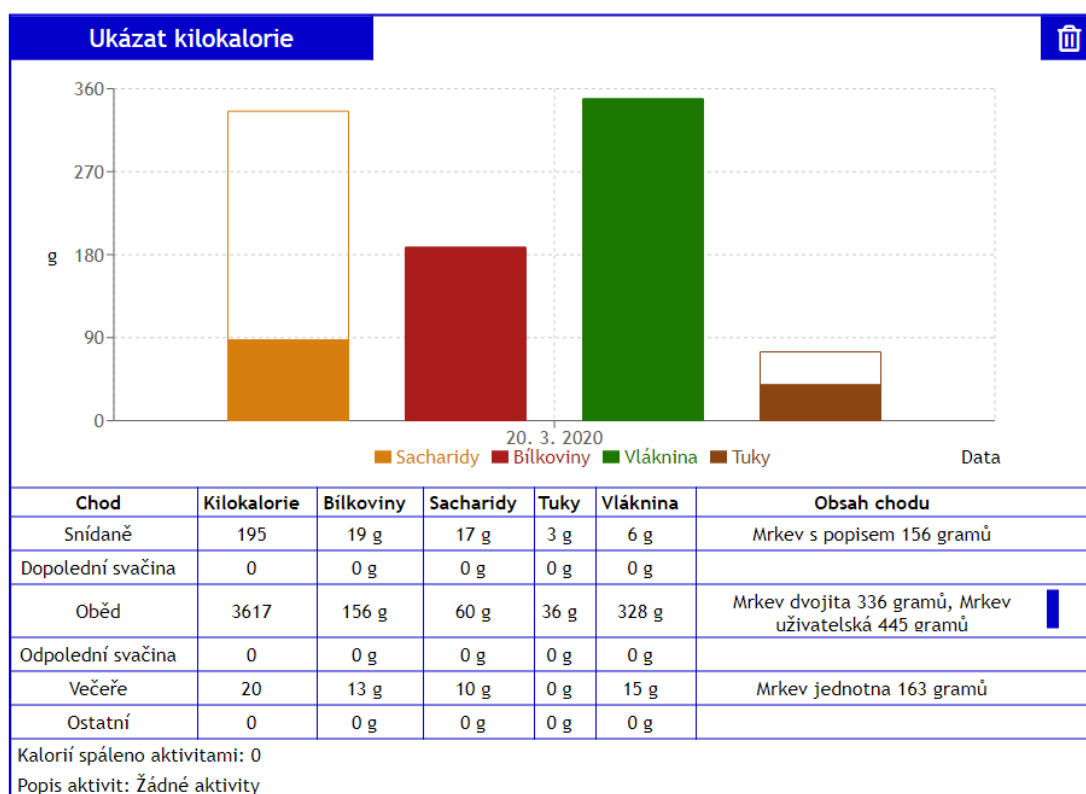
⁴ Knihovna *Recharts* je veřejně dostupná z url: <http://recharts.org/en-US/>.

Vybrat více než jeden den

březen 2020						
PO	ÚT	ST	ČT	PÁ	SO	NE
24	25	26	27	28	29	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Ukázat záznamy

Obr. 21: Veřejně dostupná komponenta ReactCalendar



Zpět

Obr. 22: Přehled o příjmu nutričních hodnot a o složení jednotlivých chodů

5.3.10 UserInformation

Poslední komponenta mého projektu se nachází na url **/userInformation** a má dvě scény. Jedna scéna slouží k zobrazování uživatelských informací (viz obr. 23) a druhá slouží k vytváření nutričního cíle (viz obr. 24). Mezi uživatelské informace patří: emailová adresa, uživatelské jméno a nutriční cíl. Uživatelské jméno a emailovou adresu zadává uživatel při registraci a nutriční cíl si uživatel může vytvořit v této komponentě tím, že vyplní formulář o pěti položkách – věk, výška, váha, pohlaví a míra aktivity. Tyto údaje slouží pro vypočítání doporučeného denního příjmu kilokalorií a gramů sacharidů, bílkovin, vlákniny a tuků na straně backendu. Kromě toho je součástí formuláře také vysvětlivka, jež uživatele informuje o tom, co si má představit pod jednotlivými pojmy, které popisují míru aktivit. Pomocí tlačítka Potvrdit uživatel volá funkci `handleSubmit`, která kontroluje, zda uživatel zadal všechna potřebná data. Pokud jsou vstupní informace nedostatečné, napíše komponenta uživateli chybu, v opačném případě zavolá funkci `createUserGoal` z *authActionCreatoru*. Uživatelské informace získává komponenta skrze funkci `getUserInformation` v *authActionCreatoru*. Tuto funkci volá komponenta pokaždé, když se vykresluje.

Přihlášen jako maturant2020

Email: maturant2020@gmail.com

Váš nutriční cíl:

Kilokalorie: 2684.69

Bílkoviny: 167.79 gramů

Sacharidy: 335.59 gramů

Tuky: 74.57 gramů

Vláknina: 34.27 gramů

Změnit cíl

Obr. 23: Přehled uživatelských informací

Zpět

Výška

Váha

Věk

Pohlaví

Míra aktivity

Vysvětlivka: Výška se zadává v centimetrech, váha v kilogramech a věk v letech. Sedavý způsob života nezahrnuje skoro žádnou fyzickou aktivitu, lehká aktivita zahrnuje lehkou fyzickou zátěž maximálně třikrát týdně, střední aktivita zahrnuje střední fyzickou zátěž většinu dní v týdnu, vysoká aktivita počítá s velkou fyzickou zátěží každý den a extrémní aktivita značí velkou fyzickou zátěž více než jednou denně. K výpočtu kalorického cíle je využívána tzv. Rovnice Mifflin-St Jeor.

Obr. 24: Formulář na vytvoření uživatelského cíle

6. ZÁVĚR

Cílem mé maturitní práce bylo vytvoření plnohodnotné webové aplikace, která by sloužila jako stravovací deníček. Uživatel mé aplikace měl být schopen vytvořit si nutriční cíl, vlastní jídla a deníčkové záznamy, v nichž by byly přehledně uloženy informace o uživatelově stravě a aktivitách v daný den. Má aplikace měla být co nejjednodušší a ulehčovat uživateli zaznamenávání informací o příjmu energie a makronutrientů v průběhu dne. Tohoto cíle jsem, dle mého uvážení, dosáhl, přestože jsem zjistil, že mnohé se současnými technologiemi zjednodušit příliš nelze – příkladem je zadávání gramů zkonzumovaného jídla nebo spálených kilokalorií aktivitami. Tato data musí vždy zadat uživatel a je velmi těžké vědět, kolik gramů jednotlivých ingrediencí pokrm obsahuje nebo kolik kilokalorií člověk spálil při aktivitě. Existují sice zařízení, která by uživateli mohla v zjišťování těchto informací pomoci (fitness náramky, chytré váhy aj.), avšak stále musí uživatel strávit nějaký čas získáváním a zadáváním těchto informací.

Z mého pohledu existují dva doplňky, kterými by se dala má aplikace poměrně snadno vylepšit, a jeden doplněk, jehož implementace je téměř neproveditelná. Prvním proveditelným doplňkem je adaptace na mobilní zařízení a druhým je čtečka čárových kódů. Aby aplikace mohla fungovat na mobilních zařízeních, je potřeba změnit zejména její grafické uživatelské rozhraní a způsob pohybu mezi jednotlivými stránkami, přičemž funkcionality může zůstat v zásadě stejná. Zakomponovat čtečku čárových kódů do mé aplikace by bylo těžší, ovšem výhodou je, že již existují knihovny, které by značnou část práce dělaly za mne. Téměř neproveditelným doplňkem je potom komunikace s fitness náramky a chytrými vahami. Problém s komunikací spočívá v tom, že každý výrobce má vlastní komunikaci mezi svými zařízeními a svými aplikacemi. Má aplikace by tudíž musela obsahovat hned několik verzí komunikace, aby byla schopna komunikovat se zařízeními od různých výrobců, a umět rozpoznávat zařízení, která by se k ní připojila, což je velmi komplexní problém.

7. POUŽITÁ LITERATURA

Arias, Dan. 2018. Hashing in Action: Understanding bcrypt. *auth0.com*. [Online] 31. května 2018. [Citace: 22. února 2020.] <https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>.

Engineering Team. 2017. A Beginners Guide to Redux. *gistia.com*. [Online] 4. září 2017. [Citace: 18. února 2020.] <https://www.gistia.com/beginners-guide-redux/>.

Karnik, Nick. 2018. Introduction to Mongoose for MongoDB. *freecodecamp.org*. [Online] 11. února 2018. [Citace: února. 19 2020.] <https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/>.

Kubala, Jillian. 2018. How to Count Macros: A Step-by-Step Guide. *healthline.com*. [Online] 14. října 2018. [Citace: 28. března 2020.] <https://www.healthline.com/nutrition/how-to-count-macros>.

Mawer, Daine. 2018. An Introduction to React. *medium.com*. [Online] 11. října 2018. [Citace: 17. února 2020.] <https://medium.com/@dainemawer/an-introduction-to-react-578be2aa0a6f>.

MDN contributors. 2020. Express/Node introduction. *developer.mozilla.org*. [Online] 5. ledna 2020. [Citace: 18. února 2020.] <https://www.gistia.com/beginners-guide-redux/>.

Vanyan, Arpy. 2017. Learn using JWT with Passport authentication. *medium.com*. [Online] 28. prosince 2017. [Citace: 24. února 2020.] <https://medium.com/front-end-weekly/learn-using-jwt-with-passport-authentication-9761539c4314>.

8. SEZNAM OBRÁZKŮ

Obr. 1: Životní cyklus komponenty v Reactu ¹	10
Obr. 2: Oběh dat v React-Redux aplikaci ²	11
Obr. 3: Struktura backendu	14
Obr. 4: Schéma pro dokument uživatele	15
Obr. 5: Schéma pro dokument jídla	16
Obr. 7: Schéma pro deníčkové zápisy	17
Obr. 6: Schéma pro chody	17
Obr. 8: Struktura frontendu	22
Obr. 9: diaryEntryReducer	23
Obr. 10: foodReducer initialState	23
Obr. 11: authReducer initialState	24
Obr. 12: Navigation, není-li uživatel přihlášen	26
Obr. 13: Navigation, je-li uživatel přihlášen	26
Obr. 14: Komponenta Login	27
Obr. 15: Komponenta Register	27
Obr. 16: Komponenta SearchFood	28
Obr. 17: Komponenta FoodDetails	29
Obr. 18: Komponenta CreateIngredient	30
Obr. 19: Přehled ingrediencí pokrmu komponenty CreateMeal	31
Obr. 20: Přehled denních chodů komponenty CreateDiaryEntry	32
Obr. 21: Veřejně dostupná komponenta ReactCalendar	33
Obr. 22: Přehled o příjmu nutričních hodnot a o složení jednotlivých chodů	33
Obr. 23: Přehled uživatelských informací	34
Obr. 24: Formulář na vytvoření uživatelského cíle	35