Ekonomická fakulta
Faculty of Economics
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované matematiky a informatiky

Bakalářská práce

Vývoj webové aplikace pro správu kuchařských receptů

Vypracovala: Ing. Alena Pucová
Vedoucí práce: Mgr. Radim Remeš

České Budějovice 2020

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Ekonomická fakulta

Akademický rok: 2018/2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Bc. Alena PUCOVÁ
Osobní číslo:	E17790
Studijní program:	B6209 Systémové inženýrství a informatika
Studijní obor:	Ekonomická informatika
Téma práce:	Vývoj webové aplikace pro správu kuchařských receptů
Zadávající katedra:	Katedra aplikované matematiky a informatiky

Zásady pro vypracování

Cílem práce je vytvořit webovou aplikaci pro správu kuchařských receptů. Aplikace bude umožňovat evidovat recepty. V receptech bude možné vyhledávat podle různých parametrů (např. časová náročnost, obtížnost receptu). Pro registrované uživatele bude dostupná evidence vlastních surovin a personalizovaný seznam receptů. Na základě dostupných surovin uživatele bude aplikace navrhovat vhodné recepty a doporučovat doplnění surovin.

Metodický postup:

1. Studium odborné literatury.
2. Návrh a popis vývoje a implementace výsledné aplikace.
3. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
4. Závěr.

Rozsah pracovní zprávy:	40 – 50 stran
Rozsah grafických prací:	dle potřeby
Forma zpracování bakalářské práce:	tištěná

Seznam doporučené literatury:

1. Freeman, A. (2018). *Pro Angular 6.* New York, NY, USA: Apress.
2. Giamas, A. (2017). *Mastering MongoDB 3.x.* Birmingham, UK: Packt.
3. Kyrnin, J., Meloni, J. C. (2019). *Sams Teach Yourself HTML, CSS, and JavaScript All in One.* London, UK: Pearson Education.
4. Magolan, G., Bell, J., Guijarro, D., de Peretti, A. & Housley, P. (2018). *Nest.js: A Progressive Node.js Framework.* Santa Rosa, CA, USA: Bleeding Edge.
5. Oluyege, P. (2019). *MongoDB, Express, Angular, and Node.js Fundamentals.* Birmingham, UK: Packt.

Vedoucí bakalářské práce: **Mgr. Radim Remeš**
Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: 15. ledna 2019
Termín odevzdání bakalářské práce: 12. dubna 2020

V Českých Budějovicích dne 15. března 2019

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

L.S.

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13
370 05 České Budějovice

doc. RNDr. Jana Klicnarová, Ph.D.
vedoucí katedry

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracovala samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to – v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou – elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Datum Podpis studenta

Poděkování

Tímto bych ráda poděkovala svému vedoucímu bakalářské práce Mgr. Radimovi Remešovi za ochotu a trpělivost. Dále bych ráda poděkovala své rodině a blízkým za podporu, kterou mi nejen během studia projevují.

Obsah

1 Úvod

V dnešní uspěchané době je velmi časté plýtvání jídlem. Mnoho lidí si ani neuvědomuje, kolik jídla během měsíce vyhodí do koše. Organizace pro výživu a zemědělství zveřejnila studii, která tvrdí, že v Evropě připadá na jednoho člověka 96 až 115 kg vyhozeného jídla za rok. Pokud by tyto potraviny nebyly vyhozeny, dokázaly by nasytit až 3 miliardy lidí. (Organizace pro výživu a zemědělství) Tato skutečnost mě přivedla k nápadu vytvoření aplikace, která by pomohla tyto zbytky zpracovat.

První část práce je věnována teoretickému přehledu, kde je popsán princip webových aplikací, jejich vývoj a také nástroje, které se k vývoji využívají. Druhá část práce představuje již samotný vývoj aplikace Zbytky v Lednici.

1.1 Cíl

Cílem této bakalářské práce je vytvořit webovou aplikaci, která bude na základě zadaných surovin nabízet kuchařské recepty. Smyslem této aplikace je využití zbytků z lednice a zamezit tak plýtvání jídlem. Často se totiž stává, že má člověk pocit, že v lednici nemá žádné suroviny, ale to nemusí být vždy pravda. Nejen, že se recepty filtrují na základě surovin, která má zrovna člověk doma, ale je i možnost recepty ještě více specifikovat, například výběrem stylu stravování (Bez masa, Bez lepku, Low Carb), dále podle druhu pokrmu (Hlavní chod, Polévka, Snídaně, Svačina a Dezert), v neposlední řadě také podle náročnosti (Snadné, Středně obtížné, Náročné). Recepty lze také selektovat podle času, který je člověk ochoten vaření věnovat, zároveň lze nastavovat i počet porcí. Cílem je taktéž možnost registrace a následné označování receptů jako oblíbených nebo vytvoření a uložení seznamu běžných surovin u konkrétního uživatele. Odpadne tak nutnost tyto suroviny znovu vypisovat. Záměrem je i navrhnout doplnění poslední chybějící suroviny k receptu.

2 Literární rešerše

2.1 Úvod k webovým aplikacím

Webová aplikace je počítačový program, který využívá klient-server architekturu a provádí konkrétní funkce pomocí webového prohlížeče jako svého klienta. Aplikace může být velmi jednoduchá, jako například kontaktní formulář, zároveň ale může být rozsáhlého charakteru, jako například komplexní hra, kterou lze stáhnout do mobilního zařízení. Prostředí klient-server je prostředí, ve kterém více počítačů sdílí informace, jako například zadávání informací do databáze. Klient je aplikace používaná k zadávání informací a server je aplikace používaná k jejich ukládání. (Nations, c2019)

2.1.1 Princip webové aplikace

Webové aplikace mají několik společných principů. Základních principy podle (Gibb, c2019) jsou následující

1. uživatel spouští požadavek na webový server přes internet, a to buď prostřednictvím webového prohlížeče, nebo uživatelského rozhraní aplikace
2. webový server předá tuto žádost příslušnému webovému aplikačnímu serveru
3. webový aplikační server provede požadovanou úlohu – například dotazování databáze nebo zpracování dat – a poté vygeneruje požadované výsledky
4. webový aplikační server odešle výsledky na webový server s informacemi nebo zpracovanými daty
5. webový server odpoví zpět klientovi se získanými informacemi, které se poté
6. zobrazí na displeji uživatele

Webové aplikace postupně nahrazují desktopové aplikace, důvodů je hned několik (Nations, c2019):

- webové aplikace běží na více platformách bez ohledu na operační systém nebo zařízení
- všichni uživatelé mají přístup ke stejné verzi, což eliminuje jakékoliv problémy s kompatibilitou

- webové aplikace není potřeba stahovat a instalovat na pevný disk, čímž je eliminováno omezení místa na disku
- snižují náklady jak pro podnik, tak pro koncového uživatele. Podnik vyžaduje menší podporu, údržbu a na koncového uživatele jsou kladeny nižší nároky na počítač.

2.1.2 Postupný vývoj webových aplikací

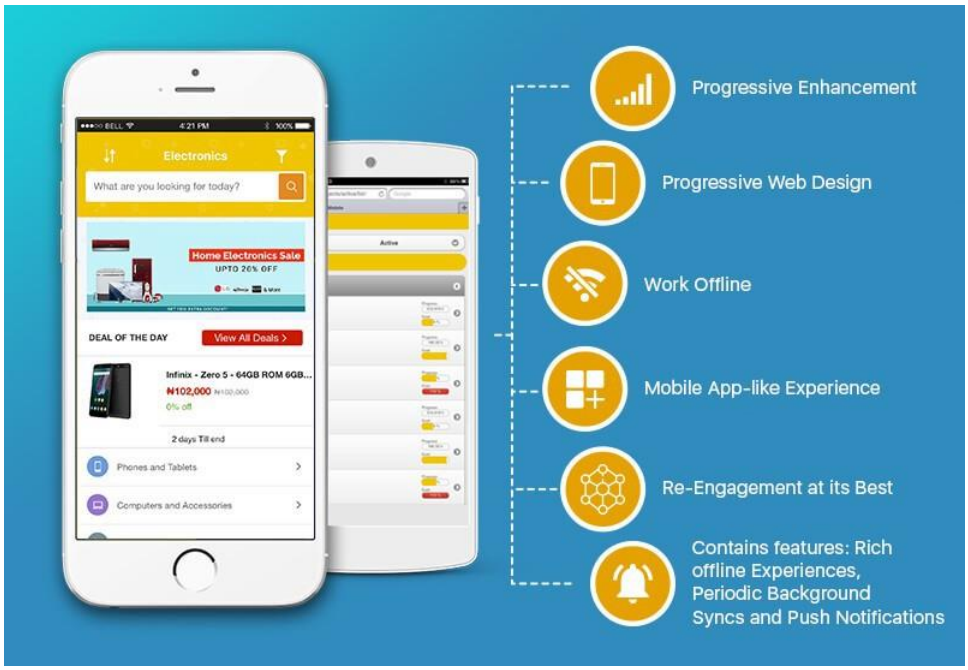
Webové aplikace rozhodně nevypadaly vždy tak, jak vypadají dnes. Na začátku 90.let byl internet plný textových dokumentů ve formě statických stránek HTML. Později bylo možné přidávat obrázky, videa a zvukové soubory, nicméně stále se jednalo o statické stránky. Velký krok k dynamičtějším stránkám pak nastal v roce 1995, při vzniku JavaScriptu na straně klienta. Díky JavaScriptu bylo možné obohatit webové stránky různými interaktivními prvky, včetně vektorové animace. Přechod ze statických na dynamické webové stránky se uskutečnil v roce 2005 zavedením Ajaxu, nového přístupu k responzivnímu designu webových stránek a vývoji webových asynchronních aplikací. Termín webová aplikace si stanovila svou pozici v historii webu. To však nestačilo, moderní technologie vyžadovaly novou úroveň webových aplikací, proto v roce 2015 Alex Russell a Frances Berman představili progresivní webovou aplikaci (PWA). (Uryutin, c2019)

Podle vývojářů Google jsou PWA (Bartík, c2019):

- progresivní – nejsou závislé na volbě prohlížeče či operačního systému
- responzivní – zobrazení stránky je optimalizováno na základě vybraného zařízení
- nezávislá na konektivitě – díky technologii service workers je možnost využívat aplikaci offline
- app-like – uživatel má pocit, jako by používal nativní mobilní aplikace
- zabezpečená – díky HTTPS je zabráněno odposlouchávání a změně obsahu dat
- dohledatelná – W3C manifesty jsou identifikované jako aplikace, tudíž je vyhledávače mohou naleznout
- instalovatelná – není nutné stahování z App Store nebo Google Play
- dostupné – jednoduchost ve sdílení pomocí URL

Vlastnosti progresivních webových aplikací shrnuje i Obrázek 1.

Obrázek 1 Vlastnosti PWA



Zdroj: https://medium.com/easyread/build-progressive-web-apps-6248a7152730

2.1.3 Proces vývoje webových aplikací

Při vývoji webových aplikací se postupuje podle určité osnovy, která může být následující

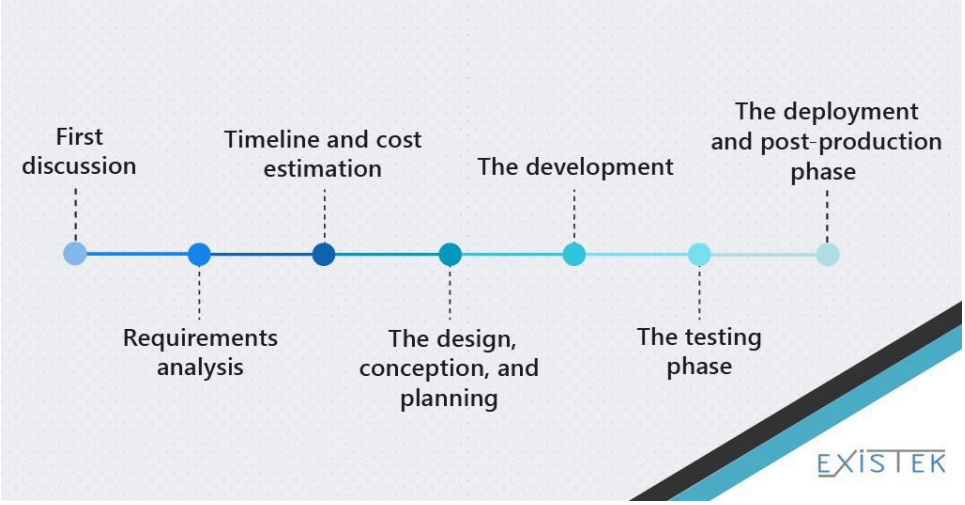
- Prvotní zamyšlení
    - Cílem prvního kroku je poskytnout přehled o problému, který chceme vyřešit nebo o našich potřebách. Pokud nejsou potřeby dobře pochopeny, povede to k selhání při vytváření webové aplikace. V této fázi bychom si měli zodpovědět následující otázky.
        - Jaké jsou základní potřeby aplikace, která má být vytvořena?
        - Jaké problémy by měla aplikace vyřešit?
        - Jak/kým bude aplikace používána?

- Analýza požadavků
    - Výsledkem předchozího kroku bylo identifikovat cíle implementace webové aplikace. Ty mohou být mnohdy složité, v tomto kroku proto dochází k rozdělení složitých cílů na zvládnutelnější úkoly, aby bylo možné je snadněji implementovat i testovat. Na konci této fáze by měly být splněny následující úkoly.
        - Přeformulování základních potřeb a cílů
        - Identifikace všech funkcí a modulů
        - Rozdělení každého cíle na jednotlivé úkoly
- Časový rozvrh a odhad nákladů
    - V tomto kroku by měla být u každého úkolu provedena analýza složitosti.
- Návrh, koncepce a plánování
    - Během tohoto kroku je cílem odpovědět na otázky zaměřující se na technický přístup, zodpovědnosti jednotlivých členů týmu (pokud je samozřejmě aplikace vyvíjena týmem), architektonickou volbu, určení závislosti jednotlivých úkolů a možností paralelního provedení.
- Vývoj
    - Samotný vývoj bývá nejznámějším krokem. Cílem této fáze je vytvořit aplikaci, která vyhovuje potřebám identifikovaných v předchozích krocích, potřeby se obecně vyvíjení a během implementační fáze se mohou objevit nové myšlenky.

- Testování
    - Během této fáze dochází k testování aplikace, ověřuje se sada funkcí, případné chyby jsou hlášeny a dochází k opravám.
- Nasazení
    - Jakmile byly opraveny nahlášené chyby, může být zahájena funkce nasazení, tzn. webová aplikace se spustí v reálném prostředí. Některé typy chování se mohou objevit až v tomto momentu, je důležité poskytnout čas na podporu a údržbu. Postprodukční fáze je tím nejlepším okamžikem, kdy lze hovořit o možném rozšíření. (Existek, c2012-2020)

Vývoj webových aplikací shrnuje následující Obrázek 2.

Obrázek 2 Vývoj webových aplikací

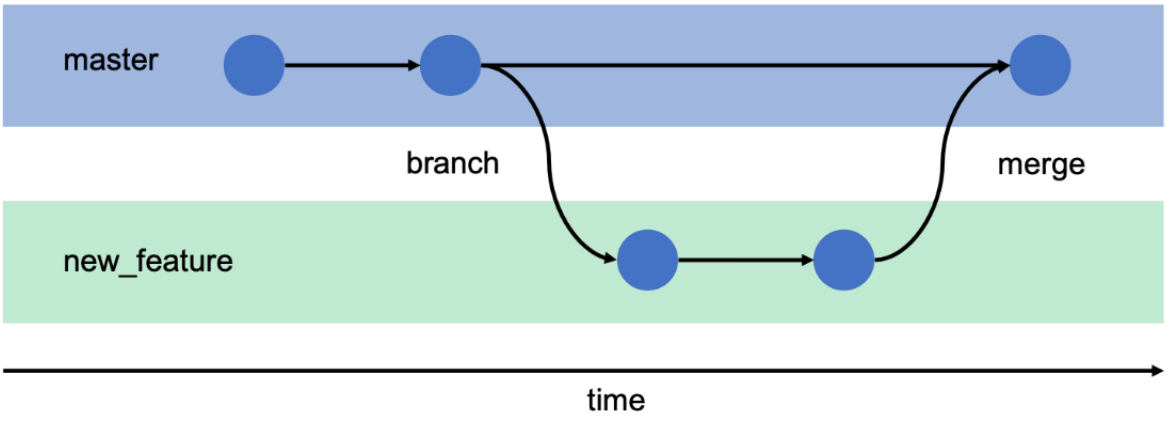


Zdroj: https://existek.com/blog/web-application-development-process-flow/

2.1.4 Verzovací systém

Při vývoji aplikací má verzovací systém nezastupitelnou roli. Jedním z nejznámějších verzovacích systémů je Git. Git slouží k průběžnému ukládání změn, přidávání funkcionalit a úprav kódu. Git umožňuje pracovat paralelně, jež je velkou výhodou právě pro týmovou práci vývojářů. Díky verzovacímu systému je snadné udržet si přehled nad tím, kdo a kdy jaké změny přidal, od jaké změny přestala aplikace dobře fungovat apod. V praxi funguje git na základě tzv. *větví*. Vývojář, který chce upravit kód, si vytvoří svou větev z hlavní větve (master nebo např. develop), upraví kód podle potřeby, následně vytvoří *commit* s popisem úprav a příkazem *push* nahraje své změny. Následně vytvoří *pull-request*, ve kterém se zobrazí změny, které provedl. Často je nutné, aby pull-request schválil kolega, který může potvrdit úpravy nebo může mít připomínky ke změnám. Poté, co dojde ke schválení změn, dojde k *mergi*. Tímto krokem se změny zahrnou do hlavní větve a jsou již součástí hlavního kódu. Nespornou výhodou verzovacího systému je to, že se jednotlivé změny mohou zpětně vracet, pokud se potvrdí, že nefungují tak jak mají apod. Systém větvení si lze představit jako následující schéma. (McCullough & Loeliger, 2012) (Obrázek 3)

Obrázek 3 Git – větvení



Zdroj: https://gitbookdown.site/branching-git-branch.html

2.1.5 MVC architektura

Při vývoji webové aplikace je nutné stavět na pevných základech, konstrukce aplikace by měla být kvalitní, aby byl podpořen možný budoucí vývoj, který může vyplývat například ze zvýšené poptávky či ze zvyšujících se požadavků na spolehlivost aplikace. Existuje několik typů architektur, každá má své opodstatnění a využívá se v jiných situacích.

Dvouvrstvá architektura

Tento typ architektury se využívá, pokud se neprezentují data uživateli, například u API serverů. Dvouvrstvá aplikace často komunikuje s další aplikací, od které přijímá požadavky a následně jí vrací data. Základním prvkem této architektury je Controller a Model. Controller představuje oddělený objekt, kde by měla být soustředěna veškerá komunikace s uživatelem. Model by pak měl obsahovat logickou část. (Čápka, c2020)

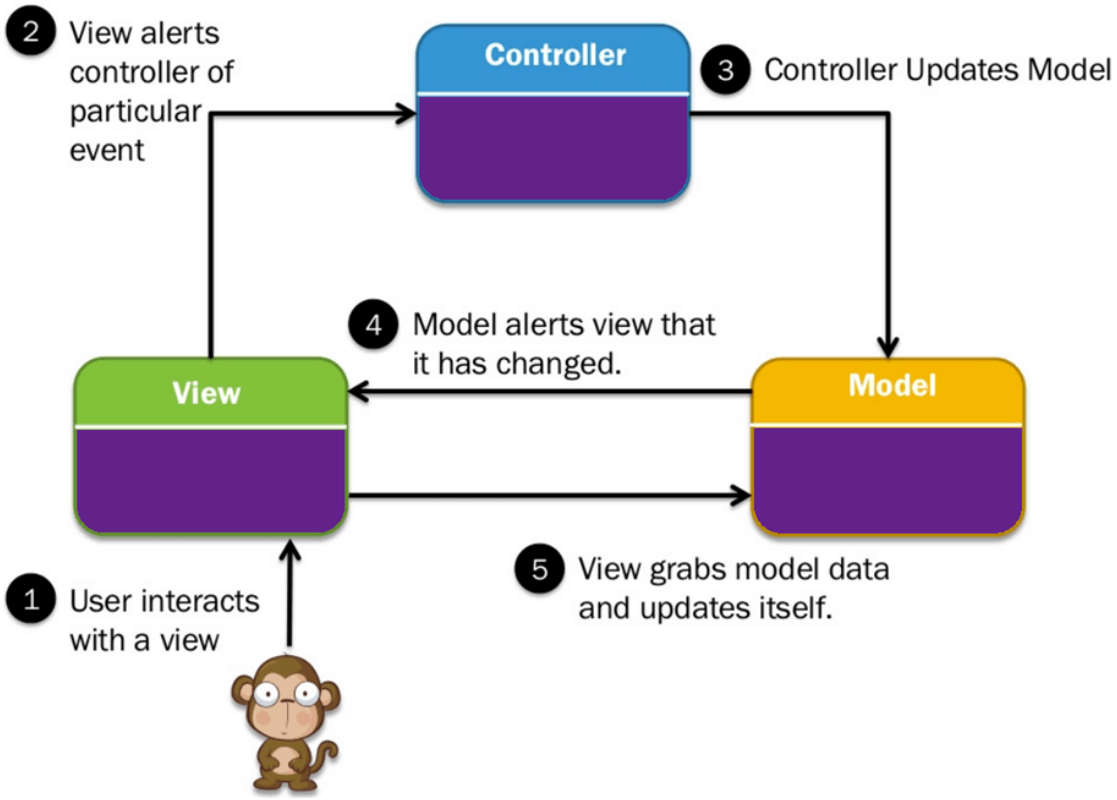
Třívrstvá architektura

Tato architektura doplňuje tu část, kterou dvouvrstvá architektura neposkytuje, a to zobrazení dat uživateli. Vrstvy aplikace jsou Model, View a Controller. Počáteční písmena vrstev pak tvoří nejznámější třívrstvou architekturu MVC.

- Model – představuje nejnižší úroveň, je zodpovědný za uchovávání a správu dat
- View – v českém překladu pohled, komponenta, která je zodpovědná za zobrazení dat uživateli
- Controller – česky řadič, komponenta, která řídí interakci mezi modelem a pohledem, zpracovává interakci uživatele

Model MVC je populární z důvodu izolace aplikační logiky od vrstvy uživatelského rozhraní. Obrázek 4 zobrazuje schéma MVC, jež popisuje celý průběh. Řadič přijímá žádost od uživatele, následně pracuje s modelem na přípravě veškerých dat potřebných pro pohled. Pohled poté použije data připravená řadičem k vygenerování reprezentativní odpovědi. (“Basic MVC Architecture”, c2020)

Obrázek 4 MVC architektura



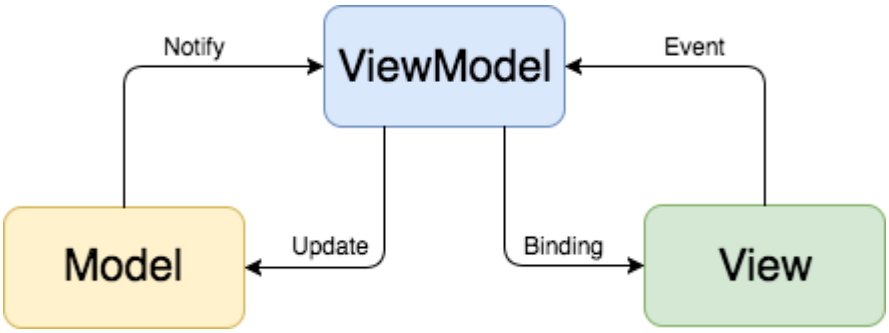
Zdroj: https://www.guru99.com/mvc-tutorial.html

Další třívrstvou architekturou je Model-View-ViewModel (MVVM), jehož prvky jsou následující:

- Model – obsahuje aplikační data. Jsou to obvykle struktury nebo jednoduché třídy, měly by obsahovat pouze vlastnost. Model není zodpovědný za získávání ani ukládání dat.
- View – zobrazuje vizuální a ovládací prvky na obrazovce
- ViewModel – transformuje informace o modelu na hodnoty, které lze zobrazit v pohledu. Většinou se jedná o třídy, tudíž je lze předávat jako reference. ViewModel se dokáže navázat na View a tudíž se změna ve View okamžitě projeví změnou ve ViewModelu. (Čápka, c2020)

Jednoduché znázornění MVVM architektury shrnuje Obrázek 5.

Obrázek 5 MVVM architektura



Zdroj: https://benoitpasquier.com/ios-swift-mvvm-pattern/

2.2 Vývojářské nástroje a ostatní technologie

2.2.1 Programovací jazyk

Javascript je objektově orientovaný programovací jazyk, představen již v roce 1995. Syntaxí se JavaScript podobá jazykům C, C++ a Java, nicméně JavaScript a Java nejsou nijak více propojeny, kromě podobnosti v názvu. JavaScript je v současnosti velmi populární, využívají ho společnosti jako Google nebo Yahoo k vývoji webových aplikací, nicméně v minulých letech byl často JavaScript považován za jazyk vhodný pro "domácí programování". Tohoto přívlastku se JavaScript však již zbavil a nyní stále nabírá na popularitě. Jednou z kritik Javascriptu je to, že se jedná o netypový jazyk, což může být, zejména pro zkušenější programátory, důvodem k nelibosti. Často se JavaScript využívá ve spojení s CSS a HTML k vytvoření interaktivních prvků na webových stránkách či aplikacích. (Sawyer McFarland, 2008)

TypeScript je programovací jazyk představen v roce 2012 jako nadstavba JavaScriptu. TypeScript je rozšířením zejména v rámci typových prvků, podporuje koncepty klasického objektového programování jako třídy, dědičnost a rozhraní. Samotný kód psaný v TypeScriptu se kompiluje do JavaScriptu, tudíž každý kód psaný v JavaScriptu je automaticky validním kódem v TypeScriptu. (Maharry, 2013)

2.2.2 Vývojové prostředí

Visual Studio Code, editor, který byl vyvinut Microsoftem jak pro Windows, tak pro Linux a macOS v roce 2015, nyní dostupný ve verzi 2019. Tento editor se stal velmi oblíbeným, zejména díky tomu, že může být používán v souvislosti s mnoha programovacími jazyky. Editor nabízí mnoho rozšíření, díky nimž se Visual Studio Code stalo mocným nástrojem. Na základě průzkumu Stack Overflow[1] z roku 2019 vyšlo najevo je VS Code je hodnocen jako nejvíce populární nástroj pro vývojáře.

1 Webová stránka pro dotazy a odpovědi pro profesionální a nadšené programátory

2.2.3 Databáze

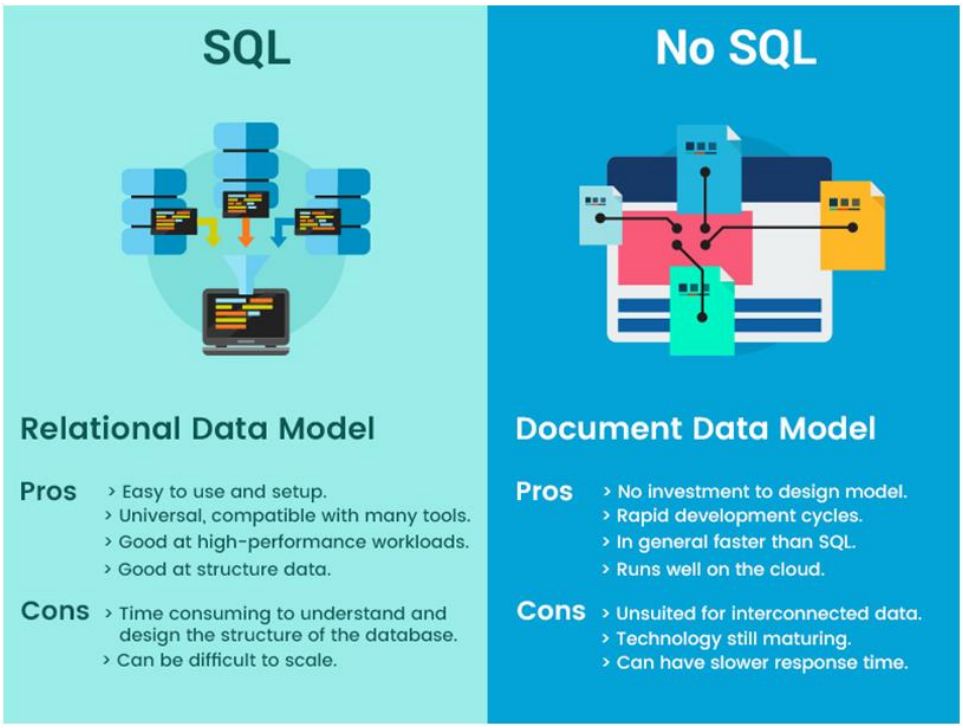
Pro vývoj webových aplikací je často potřebná přehledná databáze, která uchovává data. Existuje několik druhů databází, které lze rozdělit do dvou skupin, a to relační (SQL), a v druhém případě se jedná o NoSQL databáze. Z relačních databází je nejznámější MySQL či Oracle, v druhém případě Redis nebo MongoDB. (Rubino, c2017)

Hlavní rozdíly mezi SQL a NoSQL databází jsou následující:

- SQL používají strukturovaný dotazovací jazyk a mají předdefinované schéma. NoSQL mají dynamická schémata pro nestrukturovaná data.
- SQL jsou vertikálně škálovatelné, NoSQL jsou horizontálně škálovatelné.
- SQL jsou založeny na tabulkách, zatímco NoSQL představují dokumenty, klíče, hodnoty, grafy.
- SQL jsou lepší pro víceřádkové transakce, NoSQL jsou lepší pro nestrukturovaná data, jako jsou dokumenty nebo JSON. (MongoDB, c2020)

Další porovnání zobrazuje Obrázek 6.

Obrázek 6 SQL vs No SQL databáze



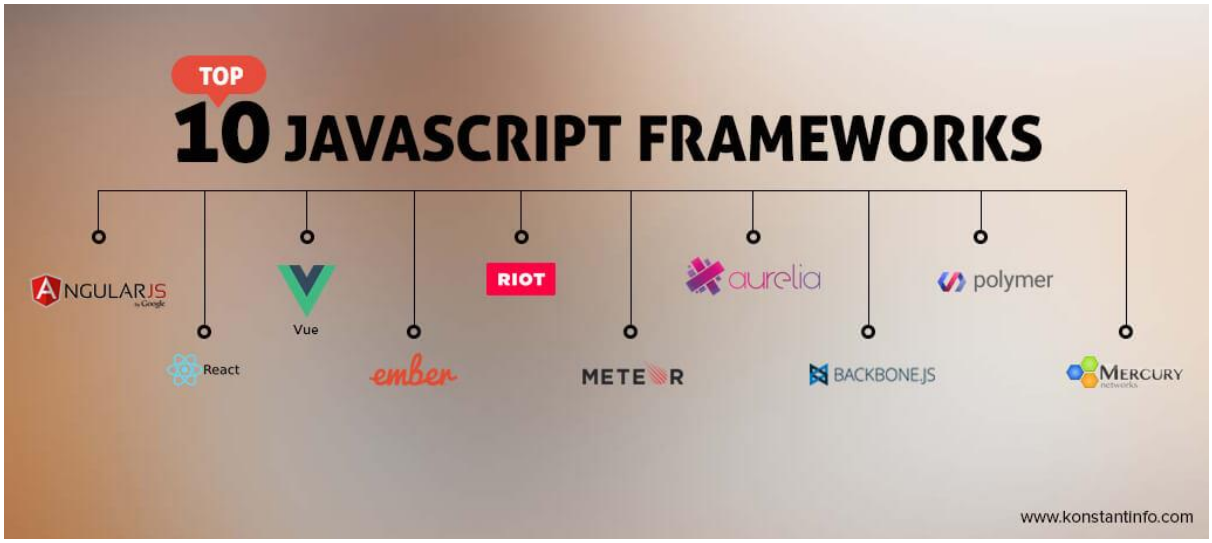
Zdroj: https://www.clariontech.com/blog/harvest-nosql-speed-with-the-combination-of-php

Zmínku si rozhodně zaslouží i HTML a CSS. HTML (ang. Hyper Text Markup Language) je značkovací jazyk, který je jedním z hlavních pilířů pro tvorbu webových stránek. Slouží k vytvoření struktury za pomocí množiny značek (tzv. Tagů). Samotné HTML nedává stránce finální vzhled, zde přichází na řadu CSS (Cascading Style Sheets). To dodává stránce pomocí selektorů možnost nastavit vzhled stránky. Může se jednat o styl textu (velikost, barvy, font), ohraničení elementů, jejich pozici na stránce, volbu pozadí, případně také změnu vzhledu při vyvolání určité situace, například najetí na element myší apod. (Duckett, 2011)

2.2.4 JavaScriptové frameworky a knihovny

JavaScript nabízí v současné době nejeden framework pro vývoj aplikací. Mezi nejznámější patří Angular, React a Vue, které jsou v práci dále popsány. Nelze říct, který je nejlepší nebo který je špatný, každý má své opodstatnění, příznivce i odpůrce. Další frameworky jsou například Ember, Meteor a několik dalších, které jsou zobrazeny na obrázku níže.

Obrázek 7 JS framework



Zdroj: https://www.konstantinfo.com/blog/top-10-javascript-frameworks-for-2016/

React

React byl poprvé představen v roce 2013 jako JavaScript knihovna a jeho hlavním benefitem je to, že je vyvíjen a spravován Facebookem, uznávanou společností, jež dodává vývojářům pocit stability, který mnoho frameworků v začátcích postrádá. Díky tomu je React velmi populárním a často zvoleným frameworkem pro vývoj single-page webových aplikací. Z pohledu architektury MVC, využívá React pouze View, proto jsou obvykle při vývoji React aplikací používány další knihovny jako Redux, React router a Axios.

Výhody

- Component-based – kód je rozdělen na komponenty, které dovolují znovupoužití kódu, kód je čistější a přehlednější
- Snadný přechod k vývoji mobilních aplikací díky React Native
- Podpora SEO
- Virtual DOM[2]
- JSX – zkrácený výraz pro Javascript XML, umožňuje vývojářům psát HTML kód v Reactu (Mahmood, c2019)

Nevýhody

- Časté aktualizace
- Nedokonalá databáze (Patel, c2013-2019)

Obrázek 8 Ukázka React komponenty

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Zdroj: https://reactjs.org/

Na obrázku č. 8 můžeme vidět příklad jednoduché komponenty. React komponenta implementuje metodu render (), která přijímá vstupní data a vrací, co se má zobrazit. K vstupním datům, která jsou předána do komponenty, lze přistupovat pomocí tzv. props. Na obrázku je zobrazeno zároveň i užití JSX. (“React – A JavaScript library for building user interfaces”, c2020)

[2] Programovací koncept, ve kterém je virtuální reprezentace UI uložena v paměti a synchronizována se skutečnou reprezentací

Vue

Vue.js byl poprvé představen v roce 2014, jedná se o progresivní framework pro tvorbu uživatelských rozhraní. Autorem je Evan You, který v té době pracoval pro společnost Google. Při hledání vhodného frameworku pro práci nenašel žádný, který by splňoval jeho požadavky. Angular byl v té době široce využíván, ale Evanovi se jevil až příliš robustní. Rozhodl se tedy vytvořit framework, který bude více flexibilní a vhodný i pro menší projekty. (Filipova, 2016)

Výhody a nevýhody jsou podle (AltexSoft, c2020) následující:

Výhody

- Virtual DOM
- Component-based
- Vhodný i pro začátečníky
- Malá velikost
- Čtivost kódu

Nevýhody

- Malé rozšíření/malá komunita
- Stále relativně mladá technologie, a tudíž nedostatek zkušených vývojářů
- Jazykové bariéra – Vue je velmi populární v Číně, tudíž hodně obsahu a diskusí je v čínštině

Angular

Ideologie skrývající se za Angularem je poskytnout framework, který snadno implementuje dobře strukturované a designové weby a webové aplikace používající MVC architekturu. Hlavním prvkem frameworku je rozdělení projektu na jednotlivé komponenty, které jsou znovupoužitelné a napomáhají k dobré čitelnosti kódu. Angular byl vyvinut Googlem, který do tohoto projektu velmi investuje, není tedy divu, že Angular je spojen s velkou podporou do budoucna. (Dayley, 2014)

Jednou z předností Angularu je CLI (command-line-interface). CLI usnadňuje práci vývojářům tím, že díky příkazům lze generovat nové komponenty se všemi náležitostmi, automaticky aktualizuje ostatní sobory v projektu, usnadňuje testování apod. Zejména na samém počátku projektu je CLI velmi užitečné, kdy je jedním příkazem založen projekt, který už je funkční a splňuje veškeré best practises. (Nayrolles, Gunasundaram, & Rao, 2017)

Výhody a nevýhody podle (Sakshi, c2008-2020):

Výhody

- Podpora stabilní společností Google
- Využití TypeScriptu
- Snadné testování
- Component-based
- CLI

Nevýhody

- Náročnost v pokročilém vývoji
- Mezery v dokumentaci

Přehledné porovnání zmíněných frameworků nabízí Tabulka 1

Tabulka 1 Porovnání JS frameworků

Parameter or Attribute	Angular	React	Vue
Backed by	Google	Facebook	Community
Prevalent Architecture	MVC	Flux	Flux
Architecture flexibility	No	Yes	Yes
CLI	angular-cli is available that is loved by Angular developers	No official CLI, but in general, https://github.com/facebook/create-react-app is used	Vue-cli is proficient and it enables you to set up your project using desired template such as webpack or browserify However, once set up is done, you don't have much to go far
Documentation	Simple and to the point Documentation is available	The official Documentation is somehow limited and may not be the best suited for a newbie.	Adequate to work with Documentation
Code Reusability	Yes	No. only CSS	Yes, CSS and HTML
What Developers love more about it	One of the oldest frameworks with Modularity	The Node Tree, Virtual DOM	The kind of amalgamation between Angular and ReactJS
Its well-known users	Of course, Google and Forbes, weather.com etc.	PayPal, Uber, Netflix etc.	Expedia, Alibaba, Nintendo etc.

Zdroj: https://medium.com/@AzilenTech/angularjs-v-s-vue-js-v-s-reactjs-choose-the-best-in-2018-66647bab1a65

3 Metodika

Pro vývoj aplikace jsem se rozhodla využít JavaScriptový framework Angular. Volba právě toho frameworku dala velký předpoklad pro výběr ostatních nástrojů, jimiž jsou Express, Node a MongoDB. Kombinace těchto nástrojů se označuje jako jeden balíček, aplikace se poté nazývá MEAN Stack application. Toto spojení získalo na popularitě zejména proto, že jak klientská, tak serverová část je psána v JavaScriptu nebo jako v tomto případě – TypeScriptu. Všechny výše zmíněné nástroje je nutné pouze nainstalovat několika příkazy, které lze snadno dohledat v jednotlivých dokumentacích.

Kód aplikace je psán v programu Visual Studio Code, který je popsán již dříve v literární rešerši. Při práci s databází MongoDB byl využit Robo 3T, open-source program, který zprostředkovává grafické uživatelské rozhraní pro jednotlivé kolekce. Díky tomuto programu je práce s databází více srozumitelná i pro laiky.

Pro pochopení některých pojmů byl vytvořen stručný slovník

Tabulka 2 Slovník pojmů

Callback	Callback představuje funkci, která se má provést až poté, co se dokončí jiná funkce
@Input	@Input () umožňuje vstup dat do podřízené komponenty z nadřazené komponenty
@Output	@Output () představuje cestu v podřízené komponentě, kterou mohou data cestovat z podřízené komponenty k nadřízené komponentě. Podřízená komponenta poté musí vyvolat událost, aby rodič věděl, že se něco změnilo.

Zdroj: Autor

4 Řešení a výsledky

4.1 Popis aplikace

Vyvíjená webová aplikace nabízí uživateli na základě surovin, které má uživatel doma, seznam receptů, jež je možné z daných surovin uvařit. Jedná se převážně o jednoduché recepty z několika ingrediencí. Uživatel má po příchodu na stránku možnost vybrat z nadefinovaného seznamu konkrétní suroviny, která má zrovna doma, případně zvolit další kritéria jako délku vaření, styl stravování (bez masa, bez lepku apod.), počet porcí, náročnost vaření, ale i typ výsledného pokrmu (polévka, hlavní jídlo apod.). Po potvrzení seznamu se uživateli vyhledá a zobrazí seznam receptů, ze kterých si může vybrat. Náhled obsahuje název, parametry a fotografii pokrmu. Po rozkliknutí se zobrazí detail se seznamem ingrediencí a postupem vaření. Uživatel má možnost vytvořit si vlastní profil, který mu umožní evidovat svůj seznam stálých surovin, které má ve většině případů doma, a dále také evidovat oblíbené recepty. Pokud po zadání surovin existují recepty, které vyžadují ještě jednu zbývající ingredienci, je na to uživatel upozorněn a je mu tento recept zobrazen, avšak oddělen od ostatních, pro které má uživatel všechny ingredience.

4.2 Instalace

K použití frameworku Angular je nutno nainstalovat Node.js. Jedná se o JavaScriptovou platformu na straně serveru využívající asynchronní I/O a model událostí umožňující rychlý vývoj škálovatelných aplikací běžící v reálném čase. Node.js hraje významnou roli v softwarovém vývoji a osvobozuje JavaScript od webového prohlížeče. (Herron, 2018)

Instalace je intuitivní a spolu s aktuálními verzemi Node.js je automaticky instalován i balíček npm. Npm usnadňuje programátorům publikování a sdílení zdrojového kódu knihoven Node.js a je navržen tak, aby zjednodušil instalaci, aktualizaci i odinstalování knihoven. Pro instalaci Node.js zadáme příkaz `install nodejs`, pro kontrolu průběhu instalace a zjištění instalované verze stačí zadat příkaz `npm -v`.

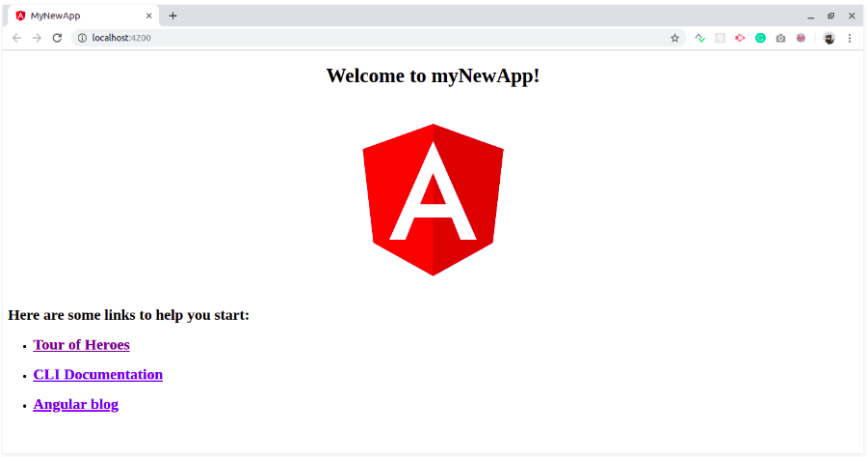
Angular dále nabízí nástroj Angular Material. Jedná se o set komponent, který usnadňuje práci vývojářům tím, že nabízí již hotové součásti, jako například tlačítka, formuláře, ikony apod. rovnou k použití. Instalaci stačí provést jednoduchým příkazem `ng add @angular/material` (“Angular Material UI component library”, c2019-2019)

Dále je potřeba nainstalovat již zmíněný TypeScript, jednoduchým příkazem `npm install -g typescript`

Vytvoření projektu

K vytvoření projektu stačí zadat příkaz `ng new myNewApp`, poté přejít do příslušného adresáře příkazem `cd myNewApp` a po příkazu `ng serve -o` je projekt plně funkční a připraven k vývoji. Náhled na počáteční stránku zobrazuje Obrázek 9. (Singh, c2020)

Obrázek 9 Náhled UI po vytvoření projektu



Zdroj: Autor

4.4 Frontend

Frontend představuje tu část, která je viditelná navenek. V této kapitole budou popsány jednotlivé modely, servisy a komponenty. Modely jsou graficky znázorněny v diagramu, který je k náhledu jako Příloha 2.

4.4.1 Popis modelů

Model Ingredient

Model Ingredient představuje jak surovinu zadanou uživatelem, tak ingredience jednotlivého receptu. Jednotlivá ingredience obsahuje následující vlastnosti

- name: string – uchovává název dané suroviny
- _id: string – vygenerovaná kombinace čísel a písmen pro identifikaci
- amount: number – zachycuje množství
- unit: string – představuje měrnou jednotku (Ks, G, Ml)

Model Recipe

Model Recipe představuje data jednotlivého receptu

- _id: string – identifikátor
- title: string – název
- ingredients: Ingredient [] - seznam potřebných ingrediencí k uvaření pokrmu
- progress: string – postup vaření
- time: number – orientační časová náročnost
- difficulty: Difficulty – představuje náročnost receptu (Easy, Medium, Difficult)
- foodStyle: FoodStyle [] - prezentuje styl stravování (NoMeat, LowCarb, GlutenFree, None)
- foodType: FoodType [] - představuje druh pokrmu (Breakfast, Snack, Soup, MainMeal, Dezert)

Model User

Model User slouží k uchování dat o každém uživateli, je tvořen základní charakteristikou uživatele jako jméno, email, heslo, zároveň ale uchovává odkaz na recepty (id), které uživatel označil jako oblíbené. Podobná logika se aplikuje i v případě oblíbených ingrediencí, tam je kromě id zaneseno i množství dané suroviny.

- _id: string – identifikátor
- firstName: string – jméno
- lastName: string – příjmení
- email: string – email, který slouží pro přihlášení do aplikace
- password: string – heslo pro přihlášení
- favouriteRecipes {_id: string} [] - uchovává identifikátor receptu pro vyhledávání v databázi
- favouriteIngredients {_id: string), amount: number} [] - uchovává identifikátor ingredience pro vyhledávání v databázi zároveň s množstvím dané suroviny

Model NameAndValue

Model NameAndValue slouží pro filtrování receptů podle specifikovaných kritérií. Každý element (v těchto případech checkbox) disponuje identifikátorem (id), názvem (name) a dále boolenovskou proměnnou (checked), která uchovává informaci, zda je daná volba zaškrtnutá.

- id: number
- name: string – název konkrétního elementu, např. bez masa, hlavní chod apod.
- checked: boolean

Model Criteria

Model Criteria představuje možnosti, které může uživatel specifikovat při vyhledávání receptů.

- time: number – maximální čas, který chce uživatel vařením strávit
- portions: number – počet porcí, které chce uživatel uvařit
- style: NameAndValue [] - styl stravování (Low Carb, Bez lepku, Bez masa)
- difficulty: NameAndValue [] - náročnost vaření (Snadné, Středně obtížné, Náročné)
- typeOfMeal: NameAndValue [] - druh pokrmu (Hlavní chod, Polévka, Snídaně, Svačina, Dezert)

Model SideBarRange

Model SideBarRange je využíván v případě 2 kritérií, a to počet porcí a časová náročnost. Jak už název modelu vypovídá, pro zvolení hodnoty tohoto kritéria je využit input typu range.

- title: string – název kritéria, např. časová náročnost
- min: number – určuje počáteční hodnotu inputu
- max: number – určuje maximální hodnotu inputu
- def: number – určuje hodnotu, která je nastavena defaultně
- path:string - uchovává cestu, která je využita pro zobrazení ikony kritéria
- value:number – zachycuje konkrétní hodnotu

Model SideBarCheckbox

Tento model je využíván pro zbývající kritéria, kdy je pro zvolení hodnoty využito inputu typu checkbox.

- title: string – název kritéria, např. styl stravování
- options: NameAndValue [] - hodnoty jednotlivých checkboxů
- path: string – uchovává cestu, která je využita pro zobrazení ikony kritéria

Model SuggestedRecipe

Tento model je využíván pro recepty, pro které nemá uživatel všechny potřebné suroviny, ale pouze 1 surovina mu chybí.

- ingredient: Ingredient – obsahuje údaje o chybějící ingredienci
- recipes: Recipe [] - obsahuje seznam receptů, které by bylo možné s chybějící ingrediencí uvařit

4.4.2 Popis servis

UserService

UserService je jedna z nejrozsáhlejších service v aplikaci. Obsahuje metody potřebné k registraci uživatele, přihlášení, validaci emailové adresy, ověření uživatele, získání dat již přihlášeného uživatele, dále k označení a získání jak oblíbených receptů, tak oblíbených surovin.

Metoda **validateRegister**, návratového typu boolean, přijímá 1 argument, a to uživatele datového typu User. V rámci této metody dojde ke zkontrolování dat, které uživatel uvedl při registraci. Metoda vrací true, pokud jsou všechny informace vyplněné a žádná z nich není undefined, jinak vrací false.

Další metodou je stručná metoda **validateEmail**, která přijímá jako argument uživatelův email. V tom kroku dojde ke zkontrolování, že emailová adresa obsahuje pouze povolené znaky a jsou splněny podmínky pro emailovou adresu. Metoda je návratového typu boolean.

Následuje metoda pro samotnou registraci, **registerUser**, která opět přijímá jako argument uživatele datového typu User. Tělo této metody odkazuje na backend aplikace, proces registrace je popsán v kapitole č. 4.5.2. Stejně je to i s metodou **authenticateUser**, která je také popsána dále v práci.

Další metody se zaměřují na oblíbené recepty a oblíbené suroviny. Metoda **setFavouriteRecipe** přijímá 2 argumenty, a to userID typu string, a recipeID, taktéž typu string. Tato metoda vrací Observable a je využívána v komponentě RecipeComponent, kdy se volá po kliknutí na ikonu srdce u jednotlivého receptu. Metoda odkazuje na backend aplikace, bude jí proto věnován prostor ještě dále v práci. S touto metodou se spojuje metoda **getFavouriteRecipes**, která přijímá jediný argument, a to userID. Tato metoda opět odkazuje na backend a stará se o to, aby byly zobrazeny oblíbené recepty přihlášeného uživatele.

Metoda **setFavouriteIngredients** přijímá argument userID typu string a dále favouriteIngredients, datového typu Ingredient []. Tato metoda se využívá na uživatelském profilu, kde uživatele zadává své oblíbené suroviny. Metoda odkazuje na backend, kde jsou do databáze uloženy konkrétnímu uživateli jeho oblíbené suroviny.

Poslední metodou v této service je metoda **getFavouriteIngredients** která se naopak stará o získání oblíbených ingrediencí z databáze. V této metodě je pouze provolán endpoint na backendu, který je popsán v backendové části práce.

IngredientsService

IngredientsService obsahuje pouze jednu metodu, a to **getIngredients**, která nepřijímá žádný argument a vrací Observable typu Ingredient []. Tělo metody obsahuje provolání na backend aplikace. (Ukázka kódu 1)

Ukázka kódu 1 Metoda getIngredients

```
getIngredients(): Observable<Ingredient[]> {
    return this.http.get<Ingredient[]>(`${this.uri}/ingredients`);
  }
```

Zdroj: Autor

RecipeService

RecipeService obsahuje metody, které slouží k získání všech receptů z databáze, dále k filtrování receptů podle zadaných surovin a kritérií a také k nalezení receptů, pro které nemá uživatel dostupné všechny suroviny.

Metoda **getRecipes** nepřijímá žádný argument a vrací výčet všech receptů. Tělo metody obsahuje volání na backend, metoda vrací Observable typu Recipe []. Náhled metody je zobrazen jako Ukázka kódu 2.

Ukázka kódu 2 Metoda getRecipes

```
getRecipes(): Observable<Recipe[]> {
    return this.http.get<Recipe[]>(`${this.uri}/recipes`);
  }
```

Zdroj: Autor

Metoda **getFilteredRecipes** přijímá 1 povinný argument a 2 nepovinné. Povinným argumentem je pole datového typu Recipe, které má být přefiltrováno, nepovinné argumenty jsou zadané ingredience datového typu Ingredient [] a zvolená kritéria datového typu Criteria. V úvodu metody je vytvořena proměnná filteredRecipes, která je ihned naplněna recepty, které byly metodě předány. Pokud jsou k dispozici zadané ingredience, začíná fáze filtrování receptů podle ingrediencí.

To probíhá tak, že se porovnává jednotlivá ingredience v receptu s jednotlivou ingrediencí zadanou uživatelem, včetně uvedeného množství. Po tomto kroku je zavolána metoda **filterByCriteria**, která přijímá 2 argumenty, a to zvolená kritéria a opět pole receptů. Ukázka kódu 3 znázorňuje, že pokud proběhlo filtrování podle ingrediencí, jsou druhým argumentem již takto přefiltrované recepty.

Ukázka kódu 3 Metoda getFilteredRecipes

```
getFilteredRecipes(
     recipes: Recipe[],
     ingredients?: Ingredient[],
     criteria?: Criteria): Recipe[]
{
   let filteredRecipes: Recipe[] = recipes;
   if (ingredients && ingredients.length > 0) {
     filteredRecipes = recipes.filter(recipe =>
       recipe.ingredients.every(
         recipeIngredient =>
          !!ingredients.find(ingredient =>
            recipeIngredient._id === ingredient._id &&
            recipeIngredient.amount * criteria.portions <= ingredient.amount)
       )
     );
   }
   filteredRecipes = this.filterByCriteria(criteria, filteredRecipes);
   return filteredRecipes;
 }
```

Zdroj: Autor

Metoda **getSuggestedRecipes** (Ukázka kódu 4) přijímá stejné argumenty jako metoda **getFilteredRecipes**, rozdílem je však to, že v tomto případě jsou zadané ingredience povinným argumentem. V úvodu metody je vytvořena proměnná suggestedRecipes, typu SuggestedRecipe [], s přiřazenou hodnotou prázdného pole. Následuje krok získání ingrediencí, které se nenacházejí v surovinách zadaných uživatelem, ale nacházejí se v receptu. Poté jsou vyselektovány recepty, kde chybí pouze 1 surovina. Tato surovina s odpovídajícím receptem je uložena do proměnné suggestedRecipes a následně ještě přefiltrována na základě kritérií, stejným způsobem jako v předcházející metodě.

Ukázka kódu 4 Metoda getSuggestedRecipes

```
getSuggestedRecipes(

       recipes: Recipe[],
       ingredients?: Ingredient[],
       criteria?: Criteria): SuggestedRecipe[]
{
   let suggestedRecipes: SuggestedRecipe[] = [];
   if (ingredients && ingredients.length > 0) {
     recipes.forEach(recipe => {
       const missingIngredients = recipe.ingredients.filter(
         recipeIngredient =>
         !ingredients.find(ingredient =>
           recipeIngredient._id === ingredient._id &&
           recipeIngredient.amount * criteria.portions <= ingredient.amount)
       );
       if (missingIngredients && missingIngredients.length === 1) {
         const missingIngredient = suggestedRecipes.find(
           suggested =>
           suggested.ingredient._id === missingIngredients[0]._id);
         if (missingIngredient) {
           missingIngredient.recipes.push(recipe);
         } else {
           suggestedRecipes.push({
             ingredient: missingIngredients[0],
             recipes: [recipe]
           });
         }
       }
     });
   }
   suggestedRecipes.forEach(suggested => this.filterByCriteria(
     criteria, suggested.recipes));
   return suggestedRecipes;
 }
```

Zdroj: Autor

4.4.3 Popis komponent

V této části jsou popsány jednotlivé komponenty aplikace, jejichž diagram je zobrazen jako Příloha 1.

Komponenta IndexComponent

Lze říct, že IndexComponent je hlavní, obalovou komponentou pro ostatní komponenty. Na začátku je vložena HeaderComponent, následuje FridgeItemsComponent, poté RecipeListComponent a nakonec SuggestedRecipesComponent. Hlavní funkcionalitou této komponenty je zobrazení všech receptů, ať už základních či těch, pro které potřebuje uživatel další suroviny. Pokaždé, když se změní seznam ingrediencí, zavolá se metoda **onIngredientChange**, kde se seznam ingrediencí aktualizuje a následně se zavolá metoda **onChange** (Ukázka kódu 6), kde se přefiltrují recepty. Podobně je to v případě, kdy se změní uvedená kritéria. Po změně kritérií se zavolá metoda **onCriteriaChanged**, uvnitř které se opět zavolá metoda **onChange** a recepty se přefiltrují podle aktuálních kritérií. Stejně tak je to i s recepty, pro které nemá uživatel všechny suroviny. S každou změnou ingredience nebo kritéria se recepty přefiltrují, aby byl výčet stále aktuální. (Ukázka kódu 5)

Ukázka kódu 5 Změna ingrediencí a kritérií

```
onIngredientChange(ingredients: Ingredient[]) {
    this.selectedIngredients = ingredients;
    this.onChange();
  }
  onCriteriaChanged(criteria: Criteria) {
    this.selectedCriteria = criteria;
    this.onChange();
  }
```

Zdroj: Autor

Ukázka kódu 6 Metoda onChange

```
onChange() {
   this.filteredRecipes = this.recipeService.getFilteredRecipes(
     this.allRecipes,
     this.selectedIngredients,
     this.selectedCriteria
   );
   this.suggestedRecipes = this.recipeService.getSuggestedRecipes(
     this.allRecipes,
     this.selectedIngredients,
     this.selectedCriteria
   );
 }
```

Zdroj: Autor

Komponenta HeaderComponent

Komponenta HeaderComponent je drobnou komponentou jejíž funkcionalita spočívá v zobrazení tlačítka pro přihlášení a registraci v případě nepřihlášeného uživatele a zobrazení tlačítka odhlášení a odkaz na profil v případě přihlášeného uživatele. První metodou v této komponentě je metoda **openLoginDialog**, které se zavolá pro kliknutí na tlačítko Přihlášení/Registrace. Metoda nepřijímá žádný argument, a kromě otevření dialogového okna nevykonává žádnou další funkci. Zbytek logiky je vykonán v rámci komponenty LoginComponent, která se otevře jako zmíněné dialogové okno.

Druhou metodou je **logoutUser** (Ukázka kódu 7), která se zavolá po kliknutí přihlášeného uživatele na tlačítko Odhlásit se. Metoda opět nepřijímá žádný argument a v jejím těle se odkazuje na metodu logout, která je vytvořena v rámci userService. Po stisknutí tlačítka se zobrazí informační hláška a uživatel je přesměrován na domovskou stránku.

Ukázka kódu 7 Metoda logoutUser

```
logoutUser() {
   this.userService.logout();
   this._snackBar.open("Odhlášení proběhlo úspěšně", "Skrýt", {
     duration: 3000
   });
   this.router.navigate(["/home"]);
   return false;
 }
```

Zdroj: Autor

Komponenta FridgeItemsComponent

Hlavním prvkem této komponenty je formulář pro zadávání surovin, včetně jejich gramáže a množství, dále pak samotný seznam zvolených surovin. Zobrazení této komponenty zachycuje Obrázek 10. V rámci této metody je vykreslena i níže popsaná komponenta SideBarComponent.

Obrázek 10 Ukázka komponenty FridgeItemsComponent

Zdroj: Autor

Metoda **addItem** v této komponentě nepřijímá žádný argument, volá se po stisknutí tlačítka *Přidat surovinu*. V úvodu metody dojde k ověření, že pole textového inputu nezůstalo prázdné, následně dojde k ověření, že zadané množství je vyjádřeno číslem. Na základě těchto dvou podmínek dojde k uložení suroviny do proměnné chosenIngredients a zároveň smazání suroviny z nabízených možností metodou **removeOption**. Po přidání suroviny se inputová pole vymažou, aby byla připravena pro zvolení další ingredience. Metoda je zobrazena jako Ukázka kódu 8.

Ukázka kódu 8 Metoda addItem

```
addItem() {

   if (this.item !== "") {
     this.chosenIngredient.amount = this.amount;

     if (this.isNumber(this.amount)) {
       this.chosenIngredients.push(this.chosenIngredient);
       this.removeOption(this.options, this.chosenIngredient._id);

       this.amount = null;
       this.item = "";
       this.chosenIngredient = null;

       this.ingredientsChanged.emit(this.chosenIngredients);
       this.upDateOptions();
     } else {
       console.log("not a number");
     }
   }
 }
```

Zdroj: Autor

Metoda removeOption přijímá 2 argumenty, a to argument options, datového typu Ingredient [], a argument id, datového typu string. V metodě se nachází for cyklus, který prochází pole options. Pokud se id jednotlivé option shoduje s id předaným jako argument, je tato option vymazána z pole options. (Ukázka kódu 9)

Ukázka kódu 9 Metoda removeOption

```
removeOption(options: Ingredient[], id: string) {
   for (let i = 0; i < options.length; i++) {
     if (options[i]._id === id) {
       options.splice(i--, 1);
     }
   }
 }
```

Zdroj: Autor

Další metodou, která je potřebná v této komponentě, je **deleteItem**, které se zavolá po stisknutí tlačítka, které je u každé suroviny v seznamu. Tato metoda přijímá 1 argument, a to item datového typu Ingredient. Nejdříve pomocí metody indexOf zjistíme pozici dané položky v seznamu ingrediencí a následně tuto položku odstraníme pomocí metody splice. Poslední řádek v této metodě vrátí vymazanou surovinu zpět do nabídky surovin. (Ukázka kódu 10)

Ukázka kódu 10 Metoda deleteItem

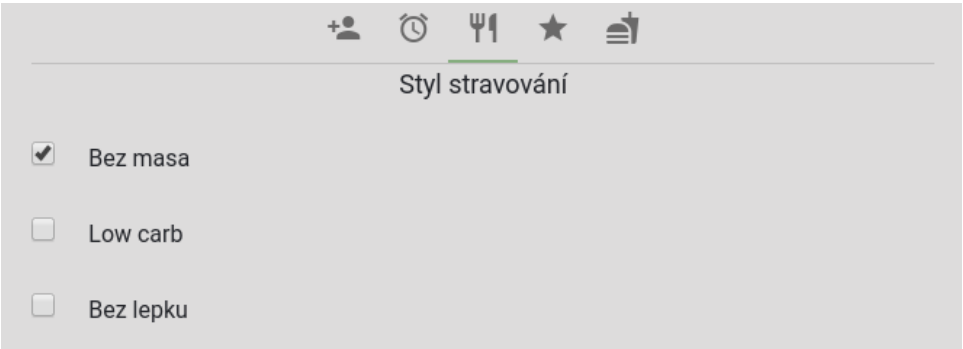
```
deleteItem(item: Ingredient) {

   const index: number = this.chosenIngredients.indexOf(item);
   if (index !== -1) {
     this.chosenIngredients.splice(index, 1);
     this.ingredientsChanged.emit(this.chosenIngredients);
     this.options.push(item);
     this.upDateOptions();
   }
 }
```

Zdroj: Autor

Komponenta SideBarComponent

Komponenta SideBarComponent slouží k volbě kritérií. Uživatel může zvolit styl stravování, druh pokrmu, časovou náročnost, počet porcí a obtížnost vaření. Časová náročnost a počet porcí jsou zachyceny jako input typu range, ostatní kritéria jako input typu checkbox. Komponenta je zachycena jako Obrázek 11.

Obrázek 11 Ukázka komponenty SideBarComponent



Zdroj: Autor

Metody v této komponentě jsou na podobném principu, jako příklad je uvedena metoda **onTypeOfMealChanged**. Komponenta obsahuje criteria, datového typu Criteria. V metodě onTypeOfMealChanged se nastaví zvolené typy jídel do proměnné, která je následně emitována rodičovské komponentě. (Ukázka kódu 11)

Ukázka kódu 11 Metoda onTypeOfMealChanged

```
onTypeOfMealChanged() {
   this.criteria.typeOfMeal = this.typeOfMeal.options;
   this.criteriaChanged.emit(this.criteria);
   this.saveCriteriaStorage();
 }
```

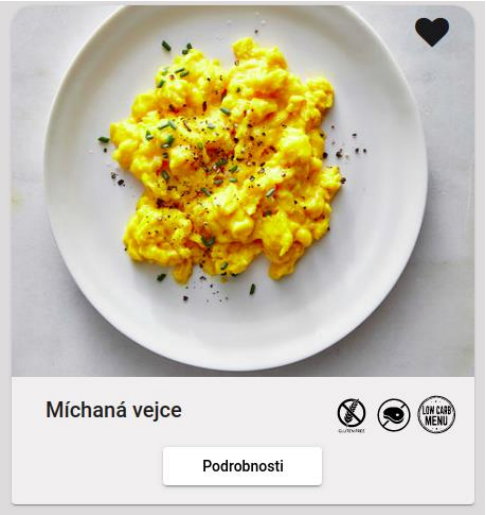
Zdroj: Autor

Další metody v této komponentě jsou **onPortionsChanged**, **onStyleChanged**, **onDifficultyChanged** a **onTimeChanged**, všechny jsou na stejném principu jako **onTypeOfMealChanged**, uloží se vybraná kritéria receptů do proměnné komponenty a její hodnota je následně předána rodičovské komponentě.

Komponenta RecipeComponent

Jednotlivý recept (Obrázek 12) obsahuje fotku receptu, název spolu s ikonami stylu stravování a odkaz na detail receptu. Ikona srdce se zobrazuje pouze přihlášeným uživatelům, po kliknutí zčervená a daný recept se přidá mezi oblíbené recepty v uživatelském profilu.

Obrázek 12 Ukázka komponenty RecipeComponent



Zdroj: Autor

Tato komponenta obsahuje metodu **getFoodStyle**, která jako argument přijímá proměnou foodstyle, datového typu FoodStyle. Návratovou hodnotou je string, který představuje cestu ke správné ikoně stylu stravování. Náhled metody spolu s cestami znázorňuje Ukázka kódu 12.

Ukázka kódu 12 Metoda getFoodStyle a cesty k ikonám

```
getFoodStyle(foodStyle: FoodStyle): string {

   if (foodStyle.toString() === "Bez lepku") {
     return this.noGluten;
   } else if (foodStyle.toString() === "Bez masa") {
     return this.noMeat;
   } else if (foodStyle.toString() === "Low carb") {
     return this.lowCarb;
   }
 }
 noGluten: string = "./assets/img/wheat.svg";
 noMeat: string = "./assets/img/nomeat.svg";
 lowCarb: string = "./assets/img/lc.svg";
```

Zdroj: Autor

Pokud uživatel označuje či odznačuje recept jako oblíbený, klikne na ikonu srdce, která je u každého receptu. V tento moment se zavolá metoda **setFavouriteRecipe**, která nepřijímá žádný argument. V metodě se nejprve zneguje dosavadní hodnota oblíbenosti a následně se emituje změna s daty id receptu a stavu oblíbenosti. (Ukázka kódu 13)

Ukázka kódu 13 Metoda setFavouriteRecipe

```
setFavouriteRecipe(event: any): void {

   this.isFavourite = !this.isFavourite;
   this.favouriteChange.emit({
     id: this.recipe._id, isFavourite: this.isFavourite
   });
}
```

Zdroj: Autor

Komponenta RecipeListComponent

Komponenta RecipeListComponent obsahuje výpis všech receptů. Ukázka kódu 14 znázorňuje, že recepty se vypisují v rámci cyklu, kdy se každý recept zobrazí jako komponenta RecipeComponent.

Metoda **isFavourite** (Zdroj: Autor

Ukázka kódu 16) přijímá 1 argument datového typu Recipe, návratová hodnota metody je boolean. V případě, že je recept oblíbený, vrátí metoda true, jinak vrací false.

Metoda **setFavourite** (Zdroj: Autor

Ukázka kódu 15) se volá při emitování změny stavu oblíbenosti některého z receptů. Jako argument přijímá objekt, který obsahuje id změněného receptu a proměnnou isFavourite, která označuje, zda je recept oblíbený, či nikoliv. Daný recept se přidá či odebere ze seznamu oblíbených a následně se aktuální data uloží do databáze skrz UserService a metody setFavouriteRecipe.

Ukázka kódu 14 HTML kód komponenty RecipeListComponent

```
<div class="recipe-list">
       <div *ngFor="let recipe of recipes" class="recipe">
           <app-recipe
               [recipe]="recipe"
               [isFavourite]="isFavourite(recipe)"
               (favouriteChange)="setFavourite($event)">
           </app-recipe>
       </div>
 </div>
```

Zdroj: Autor

Ukázka kódu 15 Metoda setFavourite

```
setFavourite(value: { id: string, isFavourite: boolean }) {
   if (value.isFavourite) {
     this.userService
       .setFavouriteRecipe(this.user._id, value.id)
       .subscribe();
     this.user.favouriteRecipes.push({ _id: value.id });
     this._snackBar.open("Recept byl uložen do oblíbených", "Skrýt", {
       duration: 3000
     });
   } else {
     this.user.favouriteRecipes = this.user.favouriteRecipes.filter(
     recipe => recipe._id !== value.id);
     this.userService
       .setFavouriteRecipe(this.user._id, value.id)
       .subscribe();
     this._snackBar.open("Recept byl smazán z oblíbených", "Skrýt", {
       duration: 3000
     });
   }
 }
```

Zdroj: Autor

Ukázka kódu 16 Metoda isFavourite

```
isFavourite(recipe: Recipe): boolean {
   if (this.favouriteRecipes) {
     return this.favouriteRecipes.includes(recipe._id);
   }
   return false;
 }
```

Zdroj: Autor

Komponenta RecipeDetailComponent

RecipeDetailComponent je komponentou, která zobrazuje konkrétní recept se všemi potřebnými surovinami, včetně gramáže. Zobrazuje také počet porcí, postup vaření a detailní fotografii pokrmu. Na tuto komponentu je uživatel přesměrován po kliknutí na tlačítko *Podrobnosti* v komponentě RecipeComponent. Tato komponenta implementuje metodu NgOnInit, která se zavolá po vytvoření celé komponenty. V rámci této metody se využívá recipeService a metoda **getRecipeById** k získání konkrétního receptu. Získaná data se uloží do proměnné recipe, typu Recipe. Nejen, že se zde získá konkrétní recept, ale získá se zde i počet porcí, které uživatel zvolil.

Metoda **getFullAmount** přijímá 1 argument a to amount, typu number, návratová hodnota je taktéž number. Každý recept má uvedeny ingredince pro 1 porci, proto se v této metodě přepočítá potřebné množství surovin, v závislosti na počtu porcí. (Ukázka kódu 17)

Ukázka kódu 17 Metoda getFullAmount

```
getFullAmount(amount: number): number {
   return this.portion * amount;
}
```

Zdroj: Autor

Komponentu RecipeDetailComponent v grafické podobě zobrazuje Obrázek 13.

Obrázek 13 Ukázka komponenty RecipeDetailComponent

Zdroj: Autor

Komponenta RegisterComponent

Komponenta RegisterComponent slouží pro registraci uživatele, obsahuje proměnné firstName, lastName, email a password, všechny zmíněné proměnné jsou datového typu string. Tato komponenta obsahuje pouze jednu metodu, a to **createUser**. V HTML je u každého textového inputu využit NgModel, díky kterému můžeme k hodnotám z inputu přistupovat snadno v rámci konkrétních metod. Prvním krokem metody je vytvoření konstanty user, která je naplněna daty, které zadal uživatel.

Druhým krokem je set podmínek, kde jsou využity metody **validateRegister**, **validateEmail** a **registerUser**, které jsou vytvořeny a popsány v rámci userService. Pokud nejsou podmínky splněny, je na to uživatel vždy upozorněn hláškou. Jako příklad je uvedena část kódu s validací emailu. (Ukázka kódu 18)

Ukázka kódu 18 Podmínka pro validaci emailu

```
if (!this.userService.validateEmail(user.email)) {
     this._snackBar.open("Použijte validní email", "Skrýt", {
       duration: 3000
     });
     return false;
 }
```

Zdroj: Autor

Náhled na komponentu z pohledu UI je zobrazen jako Obrázek 14.

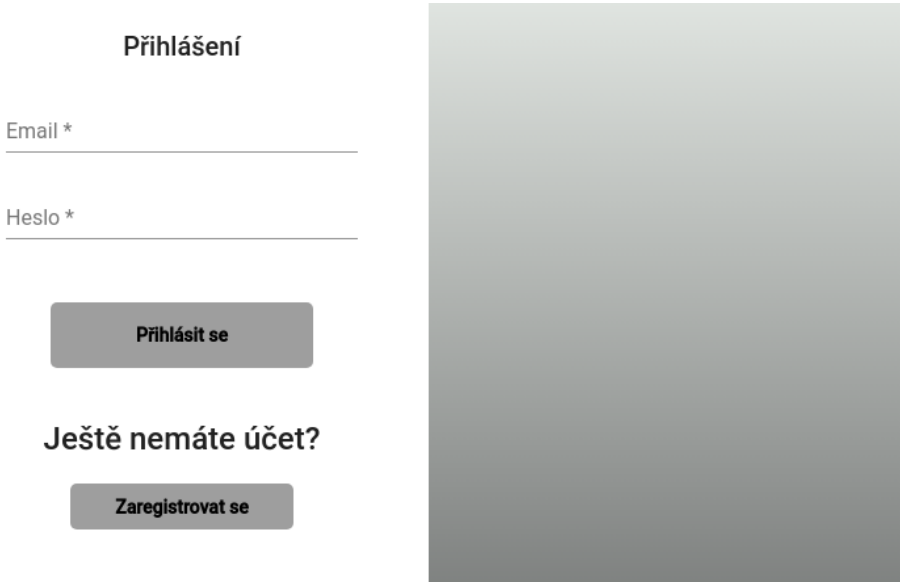
Obrázek 14 Ukázka komponenty RegisterComponent

Zdroj: Autor

Komponenta LoginComponent

Login komponenta slouží pro přihlášení uživatele. Uživatel se přihlašuje svým emailem a heslem, které je v databázi uložené šifrované. Grafické zobrazení se velmi podobá RegisterComponent, liší se jen v počtu inputů. Pokud nemá uživatel vytvořen účet, může se jedním proklikem přesměrovat na registraci. (Obrázek 15)

Obrázek 15 Ukázka komponenty LoginComponent



Zdroj: Autor

Jedinou metodou v této komponentě je metoda **loginUser**. Metoda nepřijímá žádný argument, prvním krokem v metodě je vytvoření konstanty user, kde se přiřadí hodnoty do proměnné email a password. Stejně jako při registraci, i zde se využívá NgModel, díky kterému získáváme data přímo z textových polí.

Následuje zavolání metody **authenticateUser** z userService. Tato metoda přijímá jako argument právě vytvořeného uživatele a provede autentizaci. Konkrétní proces autentizace je popsán v kapitole č. 4.5.2. Pokud proběhne autentizace úspěšně, uloží se uživatelova data do localStorage, aby nedošlo k odhlášení, když se okno s aplikací zavře. Následně je uživatel přesměrován do svého profilu. (Ukázka kódu 19)

Ukázka kódu 19 Metoda loginUser

```
loginUser() {

   const user = {
     email: this.email,
     password: this.password
   };
   this.userService.authenticateUser(user).subscribe(data => {
     if (data.success) {
       this.userService.storeUserData(data.token, data.user);
       this._snackBar.open("Přihlášení proběhlo úspěšně", "Skrýt", {
         duration: 3000
       });
       this.router.navigate(["profile"]);
       this.dialogRef.close();
     } else {
       this._snackBar.open("Přihlášení se nezdařilo", "Skrýt", {
         duration: 3000
       });
     }
   });
 }
```

Zdroj: Autor

Komponenta ProfileComponent

Komponenta ProfileComponent představuje uživatelský profil. V rámci uživatelského profilu se zobrazuje uživatelské jméno a email, který slouží pro samotné přihlášení. Dále profil obsahuje oblíbené recepty uživatele. Pro tento výpis je opět využita RecipeListComponent. Uživatel zde najde i seznam svých oblíbených ingrediencí, který zobrazuje komponenta FridgeItemsComponent. Ingredience lze jednoduše upravovat, přidávat i mazat. Uživatelský profil je vyobrazen jako Příloha 6.

Komponenta SuggestedRecipeComponent

Díky této komponentě lze uživateli doporučit další recepty, které by ho mohly zajímat, ale pro které nemá všechny suroviny, konkrétně 1 mu chybí. Tato surovina je zobrazena nad konkrétním receptem. Výpis receptů je velice podobný komponentě RecipeListComponent, opět probíhá v rámci for cyklu. (Ukázka kódu 20)

Ukázka kódu 20 HTML kód komponenty SuggestedRecipeComponent

```
<div class="suggested-recipe-list">
       <div *ngFor="let suggestedRecipe of suggestedRecipes" class="recipe">
           <h3><i class="fa fa-times" aria-hidden="true"></i>
               {{suggestedRecipe.ingredient.name}}
           </h3>
           <app-recipe-list
               [recipes]="suggestedRecipe.recipes">
           </app-recipe-list>
       </div>
</div>
```

Zdroj: Autor

Tato komponenta neobsahuje žádné metody, funkcionalita doporučených receptů je provedena v rámci IndexComponent. V této komponentě SuggestedRecipeComponent je pouze vytvořen input, do kterého přicházejí data právě z komponenty IndexComponent. (Ukázka kódu 21)

Ukázka kódu 21 Input v rámci komponenty SuggestedRecipeComponent

```
export class SuggestedRecipeComponent {

  @Input() suggestedRecipes: SuggestedRecipe[] = [];
}
```

Zdroj: Autor

Náhled na tuto komponentu je zobrazen jako Příloha 8.

4.5 Backend

Proto, aby mohla aplikace spolupracovat s databází, bylo nutné vytvořit také serverovou část.

4.5.1 Popis modelů

V rámci backendu byly vytvořeny modely User, Recipe a Ingredient, které se shodují s modely na frontendu. V této části jsou ale ve složkách s modely vytvořeny metody, které jsou potřeba pro práci s databází.

User

První metodou je **getUserById**, která, jak už název napovídá, slouží k vyhledání uživatele v databázi podle konkrétního id. Argumentem této metody je id uživatele a callback, který představuje funkci, která se zavolá v momentě, kdy dojde k nalezení či nenalezení uživatele v databázi.

Druhou metodou je **getUserByUsername**, která přijímá jako argument email uživatele a opět callback. Tato metoda bude později použita pro autentizaci uživatele.

Dále se v tomto modelu objevuje metoda **addUser**, která přijímá argument newUser a používá se při registraci uživatele. V rámci této metody dojde k zašifrování hesla a následně k uložení uživatele do databáze.

Metoda **comparePassword** se využívá při autentizaci uživatele, porovnává se zde heslo, které uživatel zadal v přihlašovacím formuláři s heslem, pod kterým se registroval. Na základě této metody je poté uživateli povolen či zamítnut přístup do uživatelského profilu.

Poslední metodou v rámci tohoto modelu je **saveUser**, která je využívána nejen v případě označování oblíbených receptů a surovin. Po každé z těchto akcí je nutné uložit uživatele s novými informacemi, například s nově označeným oblíbeným receptem.

Recipe

V modelu Recipe se nachází pouze jedna funkce, a to **getRecipeById**. Tato funkce je totožná s getUserById, jediným rozdílem je to, že nyní vyhledáváme konkrétní recept, nikoli uživatele.

Ingredient

Model Ingredient je taktéž velmi stručný, opět obsahuje pouze 1 metodu, a to pro vyhledání Ingredience podle id (**getIngredientById**).

4.5.2 Endpoints

V této sekci budou představeny jednotlivé endpointy, které bylo nutno vytvořit kvůli komunikaci frontendu s backendem.

Endpoint pro registraci uživatele

První endpoint slouží pro registraci uživatele. Je přístupný metodou **POST** na Url adrese */users/register*. V rámci tohoto endpointu se nejprve vytvoří nový objekt datového typu User, který je následně metodou addUser přidán do databáze s tokenem, který slouží pro autentizaci a jehož platnost je 1 týden. V momentě, kdy se uživatel neodhlásí a znovu navštíví web později, dojde k automatické autentizaci. Pro vytvoření tokenu je využit balíček jsonwebtoken, zkráceně jwt. (Ukázka kódu 22)

Ukázka kódu 22 Endpoint pro registraci uživatele

```
router.route("/users/register").post((req, res) => {
  let newUser = new User({
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    email: req.body.email,
    password: req.body.password
  });

  User.addUser(newUser, (err, user) => {
    if (err) {
      res.json({ success: false, message: "Failed to register user" });
    } else {
      const token = jwt.sign({ data: user }, config.secret, {
        expiresIn: 604800 //1 week
      });

      res.json({
        success: true,
        token: "JWT " + token,
        message: "User has been registered",
        user: {
          id: user._id,
          firstName: user.firstName,
          lastName: user.lastName,
          email: user.email
        }
      });
    }
  });
});
```

Zdroj: Autor

Endpoint pro autentizaci

Druhý endpoint je dostupný na Url adrese */users/authenticate* metodou **POST**. (Ukázka kódu 23) V rámci tohoto endpointu jsou získány přihlašovací údaje uživatele, a to email a heslo. Následně je vyhledán uživatel podle uživatelského jména, konkrétně emailu. Pokud je uživatel nalezen, provede se porovnání hesel, po úspěšném porovnání je uživatel přihlášen do systému a je vygenerován token stejným způsobem jako při registraci.

Ukázka kódu 23 Endpoint pro autentizaci uživatele

```
router.route("/users/authenticate").post((req, res, next) => {

 const email = req.body.email;
 const password = req.body.password;

 User.getUserByUsername(email, (err, user) => {
   if (err) throw err;
   if (!user) {
     return res.json({ success: false, message: "User not found" });
   }

   User.comparePassword(password, user.password, (err, isMatch) => {
     if (err) throw err;
     if (isMatch) {
       const token = jwt.sign({ data: user }, config.secret, {
         expiresIn: 604800 //1 week
       });

       res.json({
         success: true,
         token: "JWT " + token,
         user: {
           id: user._id,
           firstName: user.firstName,
           lastName: user.lastName,
           email: user.email
         }
       });
     } else {
       return res.json({ success: false, message: "Wrong password" });
     }
   });
 });
});
```

Zdroj: Autor

Endpoint pro vyhledání oblíbených receptů

Endpoint pro vyhledání oblíbených receptů konkrétního uživatele je dostupný na Url adrese */user/:userID/favouriteRecipes* metodou **GET** .(Ukázka kódu 24) V tomto případě se nejdříve uloží získané id uživatele, následně je podle tohoto id uživatel vyhledán a vrácená data představují seznam identifikátorů oblíbených receptů ve formě datového typu string[].

Ukázka kódu 24 Endpoint pro vyhledání oblíbených receptů

```
router.route("/user/:userID/favouriteRecipes").get((req, res) => {

 const userID = req.params.userID;

 User.getUserById(userID, (err, user) => {
   if (err) throw err;
   if (!user) {
     return res.json({ success: false, message: "User not found" });
   } else {
     return res.json(user.favouriteRecipes);
   }
 });
});
```

Zdroj: Autor

Endpoint pro získání oblíbeného receptu

Další endpoint je dostupný metodou **GET** na Url adrese */user/:userID/favouriteRecipes/:recipeID*. V tomto bloku kódu se nejdříve získá id receptu a id uživatele. Následně se vyhledá uživatel na základě id a pokud dosud uživatel žádné oblíbené recepty nemá, dojde k vytvoření pole s tímto jedním receptem. V opačném případě se zkontroluje, že recept s tímto id ještě není zařazen mezi oblíbené a poté se přidá mezi oblíbené recepty. Pokud není zařazen, znamená to, že recept je do seznamu oblíbených nově přidán. Nakonec je uživatel uložen s obohacenými údaji. (Ukázka kódu 25)

Ukázka kódu 25 Endpoint pro získání oblíbeného receptu

```
router.route("/user/:userID/favouriteRecipes/:recipeID").get((req, res) => {

 const userID = req.params.userID;
 const recipeID = req.params.recipeID;

 User.getUserById(userID, (err, user) => {
   if (err) throw err;
   if (!user) {
     return res.json({ success: false, message: "User not found" });
   }
   if (user.favouriteRecipes) {
     if (user.favouriteRecipes.find(e => e.id === recipeID)) {
       user.favouriteRecipes = user.favouriteRecipes.filter(
         recipe => recipe._id !== recipeID
         );
     } else {
       user.favouriteRecipes.push(recipeID);
     }
   } else {
     user.favouriteRecipes = [recipeID];
   }
   User.saveUser(user, (err, user) => {
     if (err) {
       res.json({ success: false, message: "Failed to save" });
     } else {
       res.json({ success: true, message: "User has been saved", user });
     }
   });
 });
});
```

Zdroj: Autor

Endpoint pro nastavení oblíbených surovin

Pro nastavení oblíbených ingrediencí uživatele je určen endpoint s Url adresou */user/:userID/setFavouriteIngredients*, dostupný metodou **POST**. (Ukázka kódu 26) Nejprve jsou id uživatele a seznam ingrediencí uloženy do proměnných, následně je vyhledán uživatel podle id. Pokud bylo hledání úspěšné, jako oblíbené ingredience se nastaví získané pole identifikátorů ingrediencí. Následně se upravená data uloží do databáze.

Ukázka kódu 26 Endpoint pro nastavení oblíbených surovin

```
router.route("/user/:userID/setFavouriteIngredients").post((req, res) => {
 const userID = req.params.userID;
 const ingredients = req.body;

 User.getUserById(userID, (err, user) => {
   if (err) throw err;
   if (!user) {
     return res.json({ success: false, message: "User not found" });
   }
   user.favouriteIngredients = ingredients;

   User.saveUser(user, (err, user) => {
     if (err) {
       res.json({ success: false, message: "Failed to save" });
     } else {
       res.json({ success: true, message: "User has been saved", user });
     }
   });
 });
});
```

Zdroj: Autor

Endpoint pro získání oblíbených ingrediencí

Následující endpoint */user/:userID/favouriteIngredients* je dostupný metodou **GET**. Nejprve je vyhledán uživatel na základě předaného id, pokud má uživatel oblíbené ingredience, jsou u něj v databázi uložena pouze jejich id a množství (položka amount, datového typu number), proto je nutné nejdříve načíst všechny ingredience z databáze a obohatit uživatelovo oblíbené ingredience o zbylé informace (name, datového typu string a unit, datového typu string).

Ukázka kódu 27 Endpoint pro získání oblíbených ingrediencí

```
router.route("/user/:userID/favouriteIngredients").get((req, res) => {
 const userID = req.params.userID;

 User.getUserById(userID, (err, user) => {
   if (err) throw err;
   if (!user) {
     return res.json({ success: false, message: "User not found" });
   } else {
     Ingredient.find((err, ingredients) => {
       if (err) console.log(err);
       else {
         user.favouriteIngredients.forEach(favIng => {
           const richIngredient = ingredients.find(
             item => item._id.toString() === favIng._id.toString()
           );
           if (richIngredient) {
             favIng.name = richIngredient.name;
             favIng.unit = richIngredient.unit;
           }
         });
         res.json(user.favouriteIngredients);
       }
     });
   }
 });
});
```

Zdroj: Autor

Endpoint pro získání receptů

Další endpoint, dostupný metodou **GET** na Url adrese */recipes*, slouží pro získání veškerých receptů. Jako první krok jsou načteny všechny recepty, které mají v databázi uložené ingredience pouze jako identifikátor a potřebné množství na 1 porci. Proto je nutné načíst všechny ingredience a recepty opět obohatit o položky name a unit, podobně jako bylo popsáno jako v endpointu pro získání oblíbených ingrediencí.

Ukázka kódu 28 Endpoint pro získání receptů

```
router.route("/recipes").get((req, res) => {

 Recipe.find((err, recipes) => {
   if (err) console.log(err);
   else {
     Ingredient.find((err, ingredients) => {
       if (err) console.log(err);
       else {
         recipes.forEach(recipe => {
           recipe.ingredients.forEach(ingredient => {
             const richIngredient = ingredients.find(
               item => item._id.toString() === ingredient._id.toString()
             );
             if (richIngredient) {
               ingredient.name = richIngredient.name;
               ingredient.unit = richIngredient.unit;
             }
           });
         });
         res.json(recipes);
       }
     });
   }
 });
});
```

Zdroj: Autor

Endpoint pro vyhledání konkrétního receptu

Poslední endpoint s Url adresou */recipe/:id*, dostupný metodou **GET**, slouží pro vyhledání konkrétního receptu podle id. Jako první krok je vyhledán konkrétní recept podle identifikátoru, následně jsou načteny všechny ingredience. Stejně jako v předchozích endpointech zde dojde k obohacení ingrediencí receptu o položky name a unit.

Ukázka kódu 29 Endpoint pro vyhledání konkrétního receptu

```
router.route("/recipe/:id").get((req, res, next) => {

 const id = req.params.id;

 Recipe.getRecipeById(id, (err, recipe) => {
   if (err) throw err;
   if (!recipe) {
     return res.json({ success: false, message: "Recipe not found" });
   } else {
     Ingredient.find((err, ingredients) => {
       if (err) console.log(err);
       else {
         recipe.ingredients.forEach(ingredient => {
           const richIngredient = ingredients.find(
             item => item._id.toString() === ingredient._id.toString()
             );
           if (richIngredient) {
             ingredient.name = richIngredient.name;
             ingredient.unit = richIngredient.unit;
           }
         });
         return res.json(recipe);
       }
     });
   }
 });
});
```

Zdroj: Autor

4.6 Databázový model

Vzhledem k tomu, že MongoDB je nerelační databází, nemají jednotlivé kolekce jasně definovanou strukturu. Databáze obsahuje celkem 3 kolekce a to Users, Recipes a Ingredients. Obsah těchto kolekcí se shoduje s modely již dříve popsanými v práci, v kapitole 4.4.1.

5 Závěr

Praktickým výstupem této práce je webová aplikace Zbytky v lednici. Ta slouží pro vyhledávání receptů podle surovin, které má člověk doma, typicky se jedná o zbytky, které už by nemusely být využity a skončily by v koši. Recepty se dají kromě ingrediencí selektovat také podle několika kritérií. Uživatel má možnost registrace a následně může recepty spravovat ve svém uživatelském profilu. Tam může taktéž spravovat své oblíbené ingredience.

Aplikace byla vyvinuta pomocí frameworku Angular, se kterým se pojí další technologie jako Express, Node a MongoDB. Při práci s Angularem jsem pocítila velký benefit z pohledu CLI, které poskytuje velkou pomoc zejména v začátcích projektu, ale i při samotném vývoji. Díky jednomu příkazu dojde k vytvoření nové komponenty, která splňuje potřebné náležitosti a je automaticky importována na potřebná místa v kódu.

Při vývoji jsem se také poprvé setkala s NoSQL databází a musím říct, že jsem byla překvapena, jak intuitivní je tvorba kolekcí, především díky zapisování dat ve formátu JSON.

Není tajemstvím, že pokud se chce člověk naučit něco nového, zejména v oblasti programování platí to, že by měl člověk začít svůj vlastní projekt. Toto mohu jen potvrdit. S Angularem jsem nikdy předtím nepracovala, díky bakalářské práci jsem ale získala přehled, který rozhodně uplatním v budoucnu.

I. Summary and keywords

The first part of this thesis describes web applications in general, their development and use. This part also focuses on programming tools, such as programming language JavaScript and TypeScript. Big part of the theoretical part is dedicated to JavaScript frameworks, such as Angular, React and Vue.

The main goal of this bachelor thesis was to develop web application, which could be used to reduce food waste. The user has the option to select ingredients he/she has at home, together with the amount of ingredient. The user has also the option to adjust his selection by entering criteria such as the number of servings or the time he wants to spend with cooking. Furthermore, the user selects the criteria of eating style (no meat, low carb, gluten free), type of dish (main course, soup, snack, breakfast, dessert) and difficulty (easy, medium, demanding). After this, the list of recipes is presented to the user.

For the development, Angular was used, together with Express, Node and MongoDB.

Keywords: web application, Angular, JS framework, food waste

II. Seznam použitých zdrojů

AltexSoft. (c2020). The Pros and Cons of Vue.js | AltexSoft [Online]. Retrieved January 24, 2020, from https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

Angular Material UI component library [Online]. (c2019-2019). Retrieved January 24, 2020, from https://material.angular.io/guide/getting-started

Bartík, M. (c2019). Proč a jak psát progresivní webové aplikace | Ackee blog [Online]. Retrieved January 08, 2020, from https://www.ackee.cz/blog/proc-a-jak-psat-progresivni-webove-aplikace/

Basic MVC Architecture [Online]. (c2020). Retrieved January 24, 2020, from https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.html

Čápka, D. (c2020). Lekce 2 - Monolitická a dvouvrstvá architektura. Itnetwork.cz - Ajťácká Sociální Síť A Materiálová Základna Pro C#, Java, Php, Html, Css, Javascript A Další. Retrieved August 31, 2020, from https://www.itnetwork.cz/navrh/architektury-a-dependency-injection/monoliticka-a-douvrstva-architektura

Dayley, B. (2014). *Learning AngularJS*. Addison-Wesley Professional.

Duckett, J. (2011). *HTML and CSS: design and build websites*. Indianapolis, Indiana: John Wiley.

Existek. (c2012-2020). *Web Application Development Process Flow*. Offshore Software Development Company. Retrieved June 18, 2020, from https://existek.com/blog/web-application-development-process-flow/

Filipova, O. (2016). *Learning Vue.js 2*. Packt Publishing.

Gibb, R. (c2019). What is a Web Application? | How a Web Application Works [Online]. Retrieved January 07, 2020, from https://blog.stackpath.com/web-application/

Google. (c2010-2020). *Angular*. Angular. Retrieved June 19, 2020, from https://angular.io/guide/file-structure

Herron, D. (2018). Node.js Web Development: Server-side development with Node 10 made easy (4 ed.). Packt Publishing.

Maharry, D. (2013). *TypeScript revealed*. California: Apress.

Mahmood, H. (c2019). Advantages of Developing Modern Web apps with React.js [Online]. Retrieved January 24, 2020, from https://medium.com/@hamzamahmood/advantages-of-developing-modern-web-apps-with-react-js-8504c571db71

McCullough, M., & Loeliger, J. (2012). Version Control with Git: Powerful tools and techniques for collaborative software development (2nd ed.). O'Reilly Media.

MongoDB. (c2020). NoSQL vs SQL Databases. The Most Popular Database For Modern Apps. Retrieved August 31, 2020, from https://www.mongodb.com/nosql-explained/nosql-vs-sql

Nations, D.Moreau, E. (Ed.). (c2019). What Exactly Is a Web Application? [Online]. Retrieved January 07, 2020, from https://www.lifewire.com/what-is-a-web-application-3486637

Nayrolles, M., Gunasundaram, R., & Rao, S. (2017). *Expert Angular*. Packt Publishing.

Organizace pro výživu a zemědělství. O Zachraň jídlo. Zachraň Jídlo. Retrieved September 04, 2020, from https://zachranjidlo.cz/kolik-se-plytva/

Patel, R. (c2013-2019). Pros and Cons of ReactJS Web Application Development [Online]. Retrieved January 24, 2020, from https://aglowiditsolutions.com/blog/pros-and-cons-of-reactjs/

React – A JavaScript library for building user interfaces [Online]. (c2020). Retrieved January 24, 2020, from https://reactjs.org/

Rubino, G. (c2017). Choosing a Database for Your Web Application. Scalable. Powerful. Global. | Future Hosting. Retrieved August 31, 2020, from https://www.futurehosting.com/blog/choosing-a-database-for-your-web-application/

Sawyer McFarland, D. (2008). JavaScript: The Missing Manual: The Missing Manual. O'Reilly Media.

Silhavy a kol., R. (2013). Vybrané aspekty návrhu webových informačních systémů. Scientific Press by Silhavy.

Singh, P. (c2020). Angular CLI | Angular Project Setup - GeeksforGeeks [Online]. Retrieved January 24, 2020, from https://www.geeksforgeeks.org/angular-cli-angular-project-setup/

Uryutin, O. (c2019). A brief history of web app - Oleg Uryutin - Medium [Online]. Retrieved January 08, 2020, from https://medium.com/@aplextor/a-brief-history-of-web-app-50d188f30d

III. Seznam obrázků

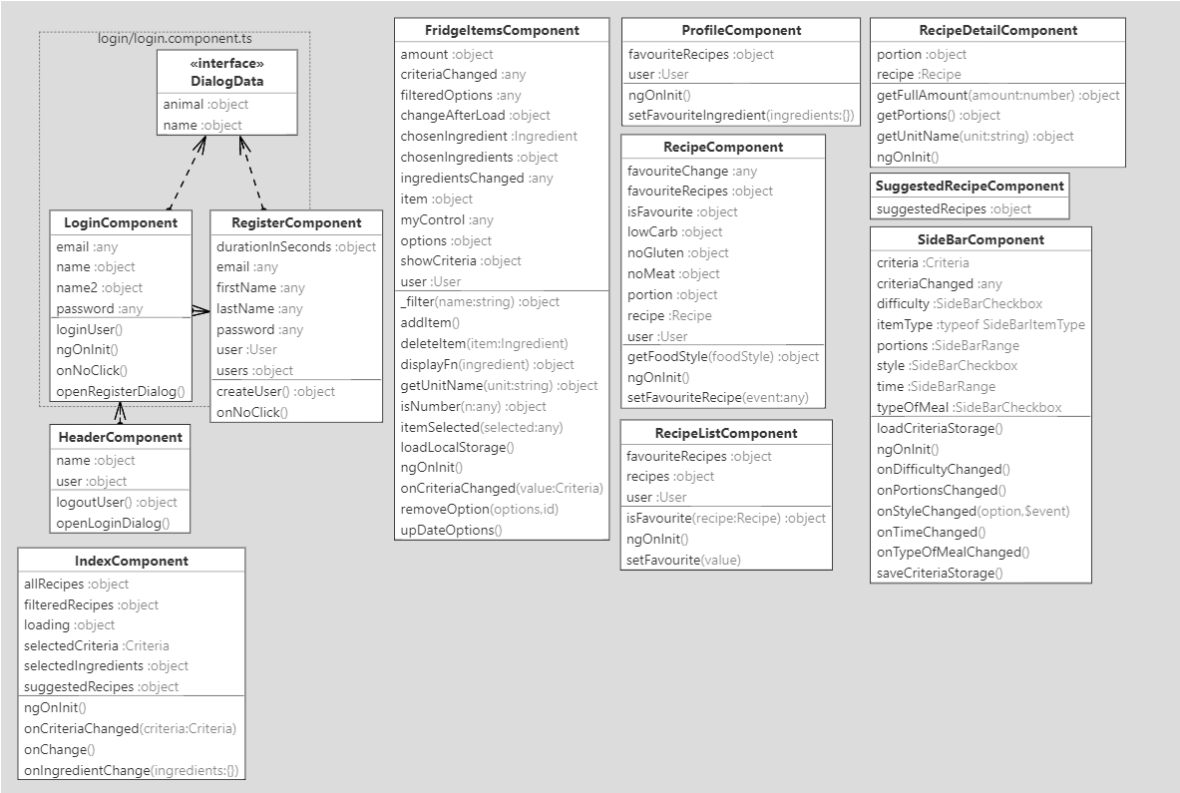
IV. Seznam tabulek

V. Seznam ukázek kódů

VI. Seznam příloh

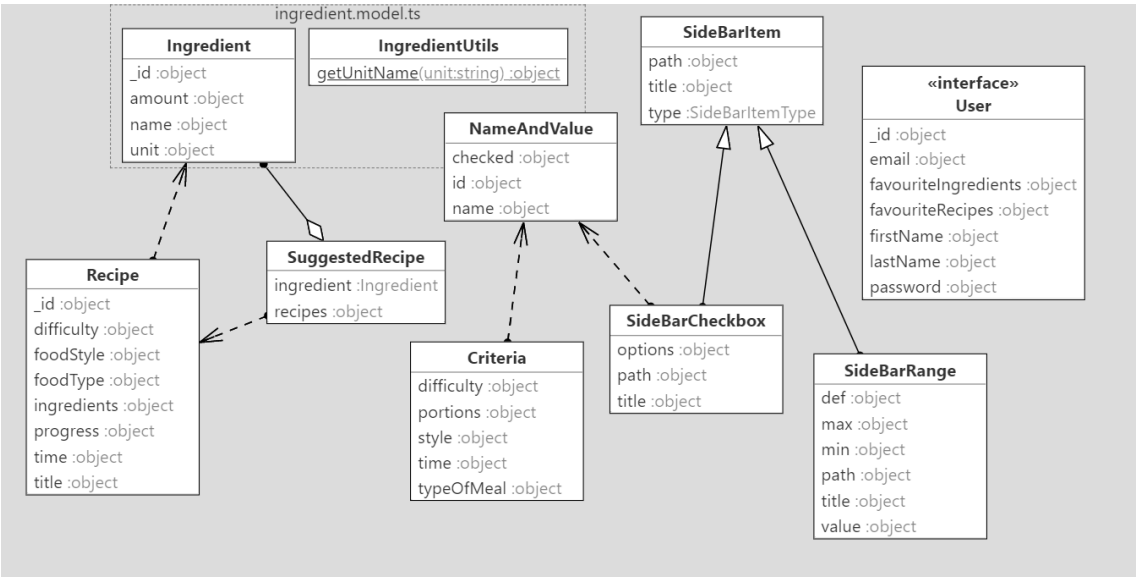
VII. Přílohy

Příloha 1 Diagram komponent



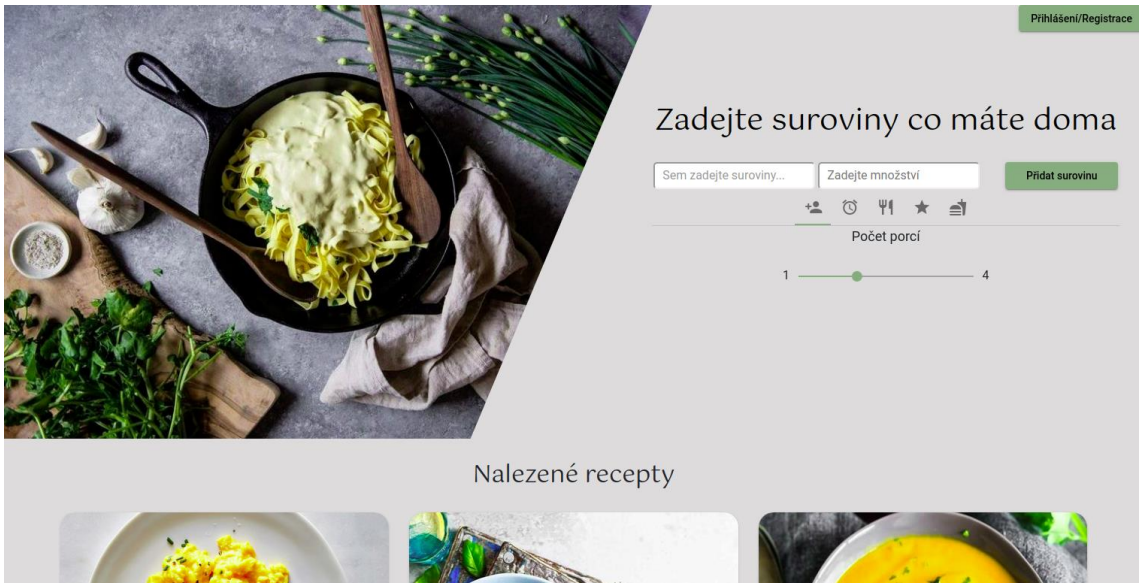
Zdroj: Autor

Příloha 2 Digram modelů



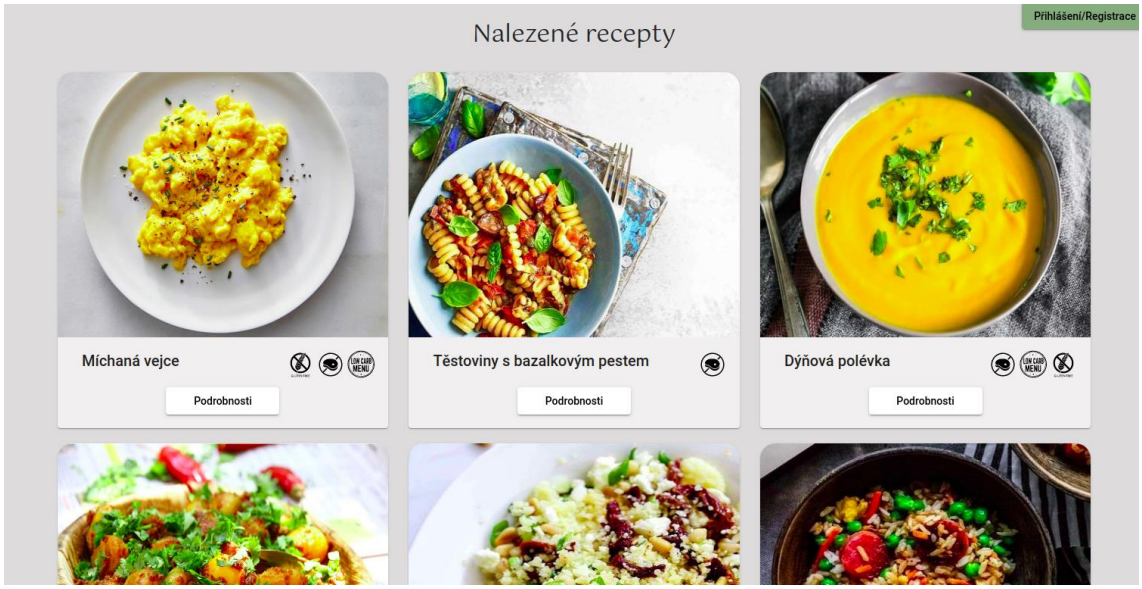
Zdroj: Autor

Příloha 3 Úvodní obrazovka aplikace



Zdroj: Autor

Příloha 4 Výčet všech receptů



Zdroj: Autor

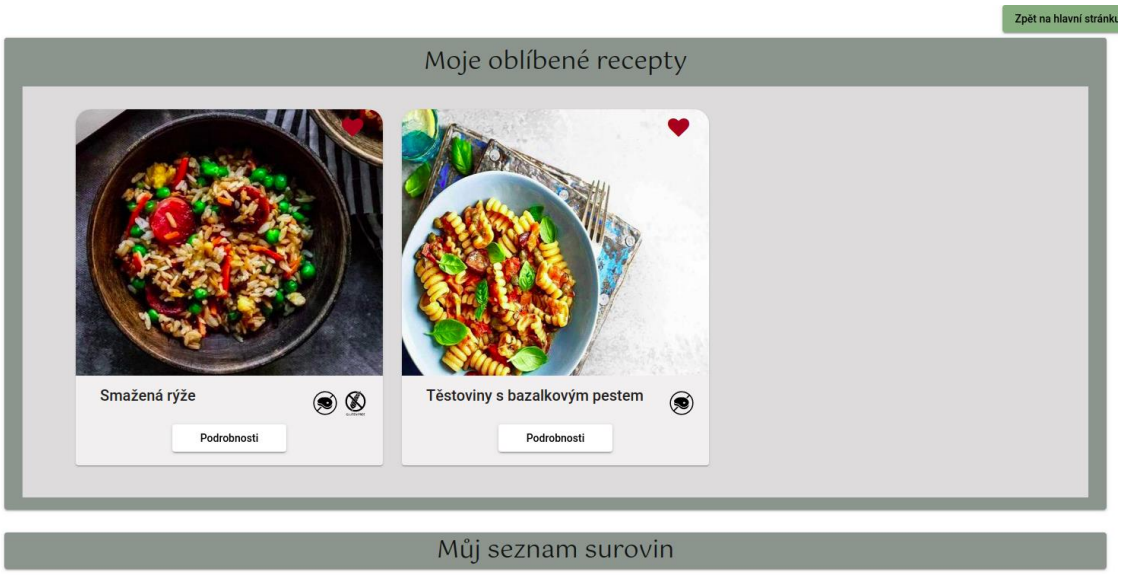
Příloha 5 Detail receptu

Zdroj: Autor

Příloha 6 Uživatelský profil

Zdroj: Autor

Příloha 7 Oblíbené recepty v uživatelském profilu



Zdroj: Autor

Příloha 8 Navrhovaný recept s nekompletními surovinami

Zdroj: Autor