FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Receptář
Student:	Jakub Rathouský
Vedoucí:	Ing. Adam Vesecký
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je návrh a implementace prostředí pro správu kuchařských receptů (webová a mobilní aplikace), dále pak analýza existujících řešení a možností importu dat z nich (receptů, dat z kalorických tabulek atd.).

Požadavky na prostředí:
- webová aplikace: backend a frontend
- prototyp mobilní aplikace, která bude využívat API backendu

Požadavky na funkce aplikace:
- webová aplikace: vytváření, upravování, zobrazování, hodnocení, komentování a sdílení receptů mezi uživateli aplikace
- mobilní aplikace: zobrazování receptů

Požadavky na technologické prvky:
- backend bude implementován v technologii NodeJS, frontend v ReactJS
- mobilní aplikace bude nativní (bez použití webview) - Android, iOS nebo multiplatformní (Flutter)

Obecné požadavky:
- všechny části budou obsahovat automatizované testy
- podrobná návrhová dokumentace pomocí SI metodik, která umožní jednoduché rozšiřování do budoucna

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 2. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Receptář

Jakub Rathouský

Katedra softwarového inženýrství
Vedoucí práce: Adam Vesecký

14. května 2019

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 14. května 2019 .....

Odkaz na tuto práci

Rathouský, Jakub. *Receptář*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019. Dostupný také z WWW: ⟨`https://bi-bap-recipe-book.herokuapp.com/`⟩.

Abstrakt

Hlavním cílem této bakalářské práce je vytvořit aplikaci na správu receptů, stejně tak jako zpracování dat z jiných webových portálů. Implementace bude zahrnovat webovou aplikaci, tvořenou z frontendu a backendu, a mobilní aplikaci.

Tématem analýzy budou existující receptáře, po kterých bude následovat návrh vlastního řešení, a dále popis použitých technologií při implementaci.

Výsledná aplikace bude obsahovat funkce jako správu receptů, vyhledávání mezi nimi nebo externími recepty, a možnost přístupu k nim jak z webové aplikace, tak i z mobilní.

Přínosem této aplikace je umožnit jejím uživatelům vyhledávat a zobrazovat si recepty z více zdrojů na jedné stránce, stejně tak jako ukládat jejich vlastní recepty a sdílet je mezi uživateli.

Klíčová slova receptář, webová aplikace, mobilní aplikace, multiplatformní aplikace, data mining, React, NodeJs, Flutter

Abstract

The main goal of this thesis is to create an application for managing recipes, as well as fetching data from other web portals. The implementation will include a web application, consisting of a frontend and a backend, and a mobile application.

Existing solutions will be subject to analysis, followed by a proposal of an own solution, as well as the description of technologies chosen for the implementation. Ultimately, the solution will be properly tested.

The solution will contain features such as the management of recipes, searching among stored recipes or external sources and accessing them from both the web application and mobile application.

The benefit of this application is to allow its users to search and view recipes from multiple sources on a single page, as well as to store their own recipes and share them with others.

Keywords recipe book, web application, mobile application, multiplatform aplication, data mining, React, NodeJs, Flutter

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Gastronomie zasahuje do života každého z nás, ať už přímo či nepřímo. Stále se objevují nové pokrmy a variace postupů již existujících receptů, mnoho z nich se zveřejní v tiskopisu, ale jsou i takové, které zůstanou zapsané pouze v našich poznámkách, které se obtížně udržují nebo sdílí mezi lidmi. Také sledovat více webových stránek a zkoumat jednotlivé postupy pokrmů je velmi obtížné kvůli jejich velkému počtu.

Před několika lety se nedaly recepty získat jinak než formou knih nebo vlastnoručně sepsaných poznámek. Toto se změnilo vznikem mnoha blogů na internetu. Už samotné blogy usnadnily možnost uschování a sdílení receptů jiným způsobem, než bylo doposud možné. Z blogů po krátké době vznikly profesionální informačních systémy, které se začaly tomuto tématu věnovat více do hloubky. Nejen, že umožnily uživatelům své recepty sdílet veřejně mezi lidmi, ale také je hodnotit a komentovat. Tím přidaly možnost rychlé orientace při vyhledávání určitého receptu. Protože lidé používají více telefon než počítač[1], vzniklo i několik mobilních aplikací, které mají zpříjemnit lidem vyhledávání receptů na telefonu.

Výsledek bude významný jak pro autory, kteří chtějí mít své recepty lehce a rychle k dispozici, či nabídnout veřejnosti jejich vymyšlené recepty, postupy a úpravy receptů již známých, tak i ostatním umožní sledovat více zdrojů pouze na jedné stránce, což pro ně bude pohodlnější.

Pro toto téma jsem se rozhodl proto, že jsem chtěl vytvořit aplikaci, jež by dovolila ukládat soukromé recepty, sdílet je mezi lidmi a prohlížet recepty z jiných zdrojů.

Práce se zabývá analýzou existujících řešení, návrhem a implementací webové a mobilní aplikace.

Kapitola 1

Cíl práce

Cílem této bakalářské práce je analýza, návrh a implementace informačního systému pro správu kuchařských receptů, kde si jednotliví uživatelé mohou recepty vytvářet nebo sdílet mezi sebou. Dále také implementace mobilní aplikace pro zobrazení receptů, ke kterým má uživatel přístup.

Prvním z úkolů analýzy je prozkoumat a analyzovat všechny uživatelské požadavky, dále provést analýzu vybraných existujících receptářů po funkční stránce a v neposlední řadě odhalit a popsat uživatelské případy užití.

Náplní praktické části je návrh informačního systému, který splňuje definované uživatelské požadavky v analýze. Dále je zaměřen na architekturu aplikace. Jednotlivé části architektury se popíší, vysvětlí se jejich funkce v aplikaci a provede se návrh databázového modelu. Na návrh následně navazuje kapitola implementace, která detailně popisuje části aplikace a jejich funkce.

Dalším krokem je samotná implementace a pokrytí automatizovanými testy.

Systém bude implementován jako webová aplikace, jejíž frontend bude naprogramován v ReactJs frameworku a backend server bude naprogramován v NodeJs frameworku. Mobilní aplikace bude implementována ve frameworku Flutter.

Kapitola 2

Analytická část práce

2.1 Analýza uživatelských požadavků

Při každém vytváření systému je potřeba znát požadavky na software neboli co má aplikace dělat a splňovat.

Jednotlivé požadavky jsou rozděleny do dvou skupin, první skupina zastupuje požadavky funkční, neboli požadavky upřesňující chování a funkce aplikace, a druhá skupina jsou požadavky nefunkční, což jsou požadavky upřesňující technologie a architekturu, které aplikace musí použít.

Pro větší přehlednost je zavedena zkratka F pro funkční požadavky a N pro nefunkční, následovanou číslem požadavku, například tedy N3 a F5.

2.1.1 Funkční požadavky - webová aplikace

V této sekci jsou vypsány všechny požadavky na aplikaci, které má obsahovat a být schopna vykonávat.

2.1.1.1 P1 - Vytvoření receptu

Uživatel při vytváření nového receptu vyplní, jak dlouho se recept připravuje, počet porcí, jaká je obtížnost přípravy, vyplní ingredience spolu s množstvím a měrnými jednotkami a postup přípravy v jednotlivých krocích. Také bude mít možnost si vybrat, jestli je recept viditelný pro všechny návštěvníky aplikace a nebo jen pro autora samotného.

2.1.1.2 P2 - Úprava receptu

Přihlašený uživatel bude mít možnost editovat své recepty. Editovatelná budou všechna pole.

2.1.1.3 P3 - Zobrazení receptů

Návštěvníci a uživatelé aplikace si budou moci zobrazit detail zobrazených receptů.

2.1.1.4 P4 - Hodnocení receptu

Přihlášený uživatel bude mít možnost ohodnotit recept. Pokud ho již jednou hodnotil, jeho hodnocení se změní na nové.

2.1.1.5 P5 - Okomentování receptu

Pro možnost poděkování autorovi, vznesení dotazu nebo případného nápadu, jak recept zlepšit, bude u každého receptu možnost připsat komentář. Tuto funkci bude moci využít pouze přihlášený uživatel.

2.1.1.6 P6 - Sdílení receptu mezi uživateli

Autorovi bude umožněno přidávat i odebírat oprávnění k zobrazení receptu vybraným uživatelům aplikace.

2.1.1.7 P7 - Registrace

Aplikace bude obsahovat možnost registrace. Pro registraci se použije kombinace uživatelského jména a hesla.

2.1.1.8 P8 - Přihlášení

Registrovaný uživatel se bude moci přihlásit.

2.1.1.9 P9 - Vyhledat recept v aplikaci

Je potřeba, aby bylo možné recepty vyhledat, případně vyfiltrovat a následně zobrazit. Nepřihlášený uživatel může prohlížet pouze recepty označené jako veřejné, zatímco přihlášený uživatel uvidí recepty své a navíc ještě ty, pro které mu bylo přiděleno oprávnění pro zobrazení.

2.1.1.10 P10 - Vyhledat recept v externích receptářích

Všichni návštěvníci budou moci přejít na stránku s výběrem cílového jazyka receptu, upřesnit jednu z podporovaných stránek, na které chtějí hledat a zobrazit si vyhledané výsledky.

2.1.1.11 P11 - Odhlášení

Přihlášený uživatel bude mít možnost se ze stránky odhlásit.

2.1.2 Funkční požadavky - mobilní aplikace

2.1.2.1 P12 - Přihlášení

Uživatel bude mít možnost se přihlásit a nebo pokračovat bez přihlášení. Nepřihlášený uživatel může prohlížet pouze recepty označené jako veřejné, zatímco přihlášený uživatel uvidí recepty své a navíc ještě ty, pro které mu bylo přiděleno oprávnění pro zobrazení.

2.1.2.2 P13 - Vyhledání receptů v mobilní aplikace

Nepřihlášený uživatel může prohlížet pouze recepty označené jako veřejné, zatímco přihlášený uživatel uvidí recepty své a navíc ještě ty, pro které mu bylo přiděleno oprávnění pro zobrazení.

2.1.2.3 P14 - Zobrazení receptu

Zobrazené recepty v aplikaci budou mít detailní náhled.

2.1.2.4 P15 - Odhlášení

Přihlášenému uživateli bude umožněno se odhlásit.

2.1.3 Nefunkční požadavky

2.1.3.1 N1 - Rozdělení architektury na frontend a backend

Aplikace bude vhodně rozdělena na frontend a backend. Backend bude poskytovat API, se kterým bude frontend komunikovat pro práci s daty aplikace.

2.1.3.2 N2 - Prototyp mobilní aplikace komunikující se serverem

Vytvoří se mobilní aplikace, která využije stejné API pro získaní dat, jako webová aplikace.

2.1.3.3 N3 - Technologie implementace

Pro implementaci frontendu se použije knihovna ReactJS a backend bude implementován v technologii NodeJS.

2.1.3.4 N4 - Technologie mobilní aplikace

Pro zajištění multiplatformní podpory se aplikace implementuje ve Flutter frameworku.

2.2 Analýza existujících řešení

V této kapitole, ve které se budeme zaměřovat na analýzu současných řešení, věnuji nejvíce pozornosti aplikacím, jež umožňují uživatelům ukládat a vyhledávat recepty, a dále jejich mobilní aplikaci.

Rozhodl jsem se podrobit analýze nejen tuzemské aplikace, ale také zahraniční weby a aplikace.

České webové stránky jsem hledal přes vyhledávač *Google*, do kterého jsem zadal výrazy: „*recepty*“ a „*recepty online*“. Z těchto výsledků jsem vybral stránky *Recepty.cz*, *TopRecepty.cz* a *Vareni.cz*.

Pro anglické webové stránky jsem použil stejný webový vyhledávač, do kterého jsem tentokrát napsal: „*recipes*“ a „*recipes online*“. Tím jsem získal stránky *AllRecipes.com* a *Epicurious.com*.

Výše uvedené stránky byly analyzovány podle následujích vlastností:

- Vyhledání receptu

  V této sekci budou popsané možnosti vyhledávání receptů a použití filtrů pro získání relevantnějších výsledků.

- Přidání receptu

  Přidání receptu je jedna z klíčových vlastností pro uživatele.

- Možnosti uživatele po přihlášení

  Tato část se bude zabývat doplňujícími funkcemi uživatele, které získá po jeho přihlášení.

- Mobilní zobrazení

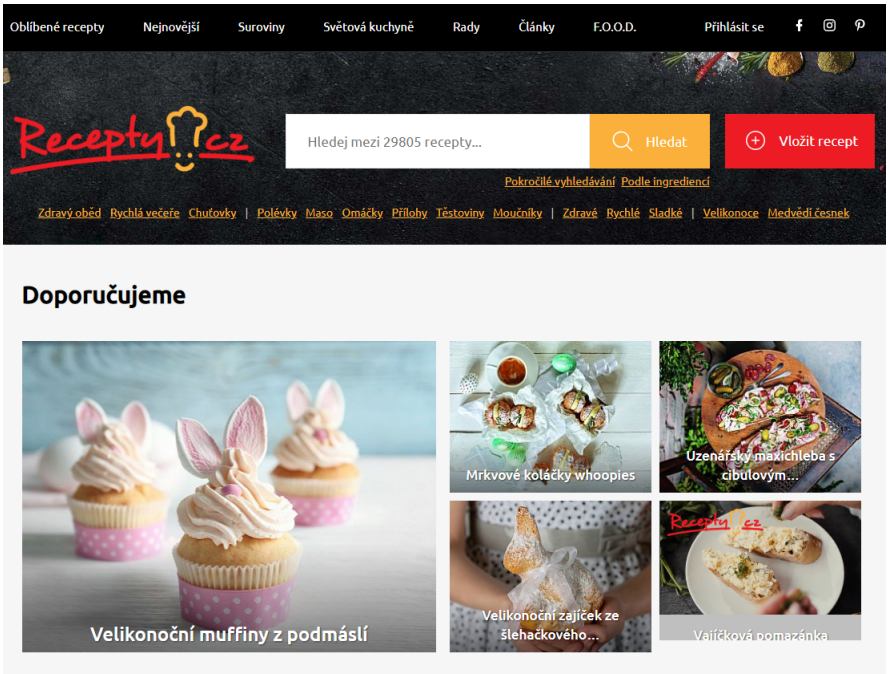
  Mít možnost pracovat s daty aplikace na telefonním zařízení také patří mezi klíčové vlastnosti, proto je potřeba ji podrobit analýze.

2.2.1 Recepty.cz

Webová aplikace *Recepty.cz* je první aplikace, která se objeví při hledání receptů ve vyhledávání. Má elegatní design doprovázený pestrými kombinacemi světlých barev a obrázků ingrediencí. Na hlavní stránce je vidět několik karet s doporučenými pokrmy, sekce s několika kategoriemi receptů a na závěr sekce se zajímavostmi z oblasti gastronomie. [2]

2.2.1.1 Vyhledání receptu

Možností vyhledávání je na této stránce mnoho. Již na úvodní stránce si lze vybrat hned z několika způsobu hledání. Nabízí fulltextové vyhledávání, tedy zda-li recept obsahuje část hledaného textu, dále možnost přechodu na některou z bližších kategorií receptů nebo si otevřít pokročilé vyhledávání

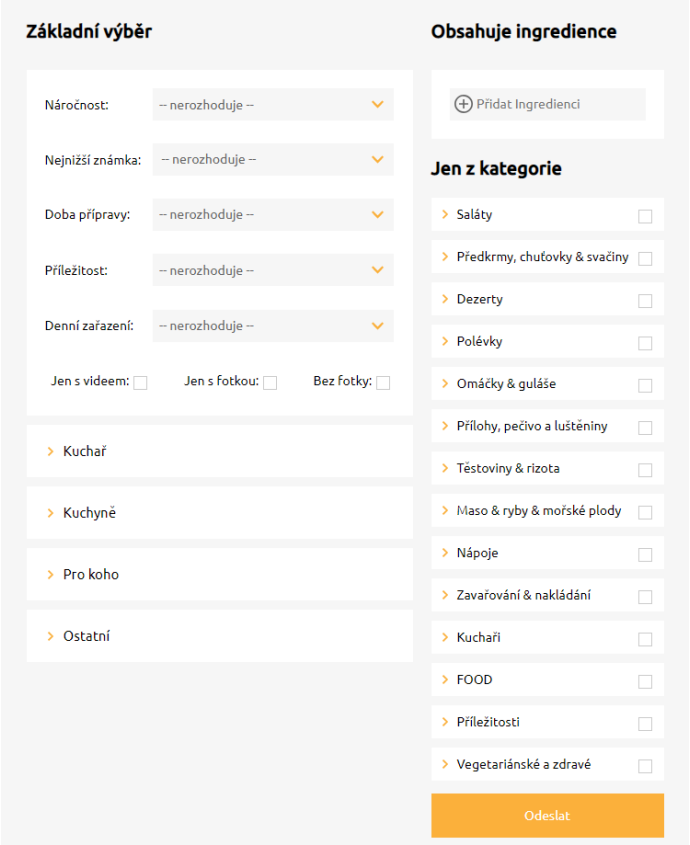


Obrázek 2.1: Úvodní obrazovka

či hledání podle ingrediencí. Horní menu obsahuje odkaz na nejnovější recepty a na oblíbené recepty uživatele. Pokročilé vyhledávání nabízí výběr kategorií a velkou kombinaci různých kritérií pro specifikování hledaného pokrmu, jako například volbu náročnosti, nejnižšího hodnocení od uživatele nebo doby přípravy. [3] Obsahuje také možnost specifikování ingrediencí, které funguje stejně jako samotné hledání podle ingrediencí zmíněné výše [4]. Výsledky hledání se dají seřadit podle hodnocení, datumu přidání a abecedy. Pokud uživatel chce upřesnit filtr, musí se vrátit na formulář s filtry, kde svou volbu upraví a znovu odešle. Všechny stránky doprovází pravý panel s popiskem „Poslední sledované recepty“, kde se neobjevují poslední recepty zobrazené uživatelem, ale recepty, které si momentálně prohlíží nejvíce lidí.

2.2.1.2 Přidání receptu

Hned vedle fulltextového vyhledávání je tlačítko pro přidání receptu. Po stisknutí dojde k přesměrování na stránku s formulářem [5]. Pro odeslání formuláře musí být vyplněné hodnoty: název, čas přípravy, náročnost, počet porcí, fotografie, alespoň jedna ingredience, postup přípravy. Ač se zdá, že recept může odeslat kdokoliv, tedy i nepřihlášený uživatel, tak po odeslání formuláře je nepřihlášený uživatel přesměrován na úvodní stránku bez jakékoliv reakce. Přihlášený uživatel je po odeslání přesměrován na stránku náhledu detailu receptu. Recept musí projít procesem schválení receptu, než je veřejně vidi-

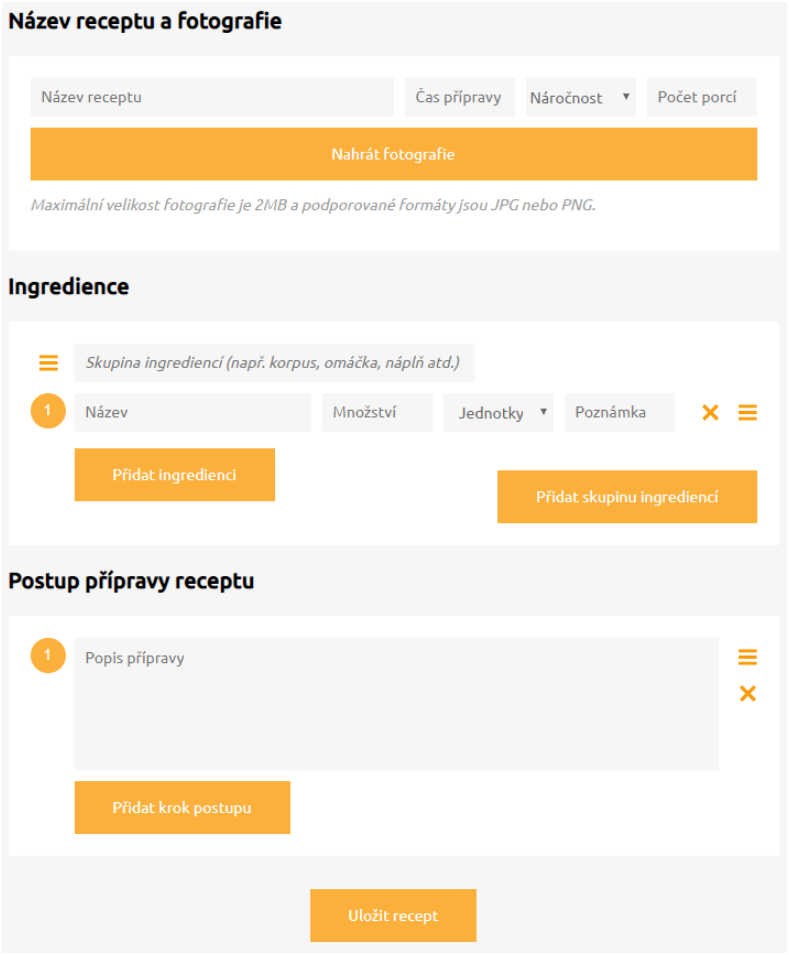


Obrázek 2.2: Filtr výsledků

telný. Není možnost přidání soukromého receptu. Po přidání receptu se recept automaticky nezařadí do uživatelovy kuchařky [6], kam se přiřadí pouze uživatelskou akcí na receptu.

2.2.1.3 Možnosti uživatele po přihlášení

Po přihlášení má uživatel možnost vykonat několik nových akcí nad receptem, může ho komentovat, připsat si poznámku, přidat do kuchařky [6] nebo vložit do svého nákupního košíku [7]. Zobrazí se mu sekce pod jeho jménem, kde si může otevřít svou kuchařku, oblíbené recepty, vzkazy nebo diskuze. V kuchařce lze vytvářet složky, do kterých jdou zařazovat jednotlivé recepty. Dále obsahuje možnost vyhledávání, odebrání receptu z kuchařky a vytisknutí zvolených receptů včetně komentářů, galerie, hlavní fotografie, ingrediencí a postupu přípravy.



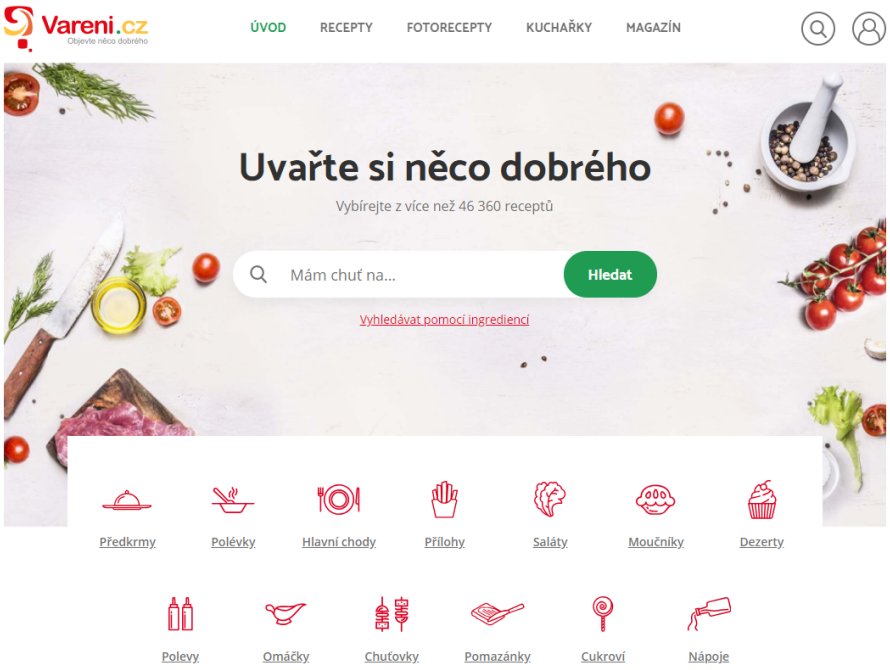
Obrázek 2.3: Formulář pro přidání receptu

2.2.1.4 Mobilní zobrazení

Aplikace nemá vlastní mobilní aplikaci, pouze je možné si aplikaci zobrazit v prohlížeči v responzivním módu pro telefon. V tomto zobrazení stránka neobsahuje pokročilý filtr a hlavní menu je transformované do hamburger menu.

2.2.2 Vareni.cz

Aplikace, která při použití Google vyhledávače zaujme druhé místo mezi výsledky. Na rozdíl od *Recepty.cz* sází na jednoduchý design a nekombinuje mnoho barev.[8]



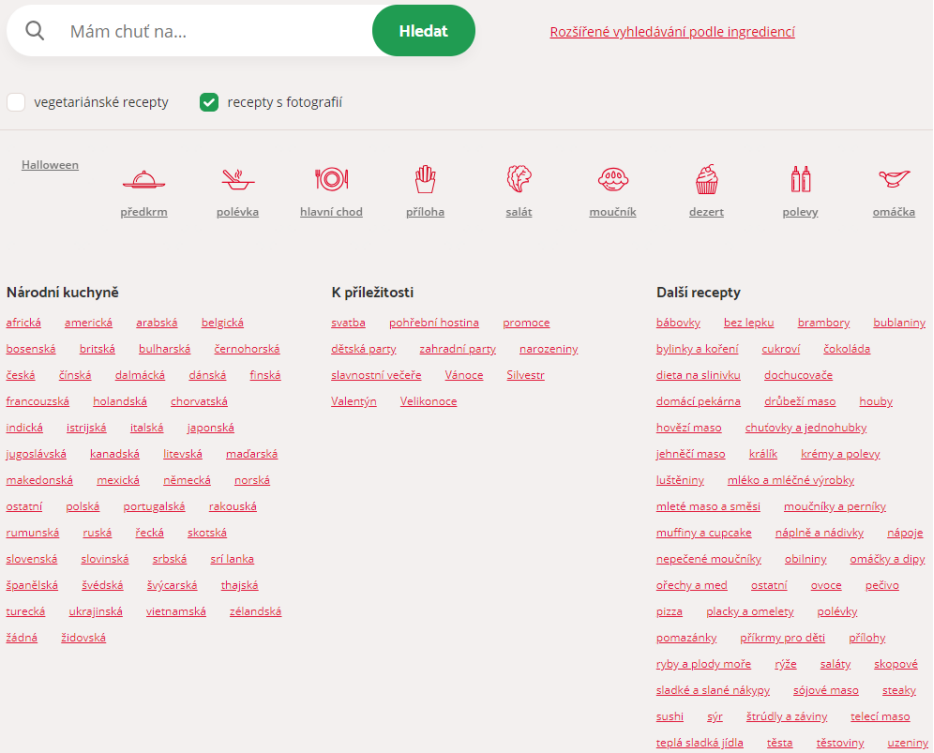
Obrázek 2.4: Úvodní obrazovka

2.2.2.1 Vyhledání receptu

Na hlavní stránce je fulltextové vyhledávání, které hledá daný text v názvu receptu, dále výčet kategorií a možnost přejít na vyhledávání podle ingrediencí. Všechny možnosti přesměrují uživatele na stejnou stránku s výsledky, kde svůj filtr může dospecifikovat [9]. Liší se to pouze v předem předvyplněných hodnotách jednotlivých filtrů. Tato stránka navíc obsahuje možnost filtrování podle typu kuchyně, což je výčet 3 hlavních kategorií: „Národní kuchyně“, „K příležitosti“ a „Další recepty“. Každá z těchto kategorií má poté několik možností, které se dají navolit. Velký rozdíl oproti *Recepty.cz* je v tom, že se recepty načítají ve chvíli úpravy filtru a tak vidí ihned relevatní výsledky bez nutnosti přecházení mezi stránkami. Výsledky hledání se dají seřadit podle hodnocení, doby přípravy a podle počtu uvaření uživateli. Výsledky se také dají přepínat mezi náhledem a více detailním zobrazením, které navíc obsahuje dobu přípravy, kategorii pokrmu, typ kuchyně a kolik uživatelů recept vařilo.

2.2.2.2 Přidání receptu

Možnost přidání receptu je na stránce s recepty. Tato funkce je povolena pouze přihlášeným uživatelům. Nepřihlášený uživatel je přesměrován na stránku s přihlášením. Proces vytváření se nepodařilo nasimulovat a analyzovat, protože



Obrázek 2.5: Filtr výsledků

v čase vypracování této práce nefungovala možnost přihlášení se, při odeslání přihlašovacího formuláře se objevila serverová chyba.

2.2.2.3 Možnosti uživatele po přihlášení

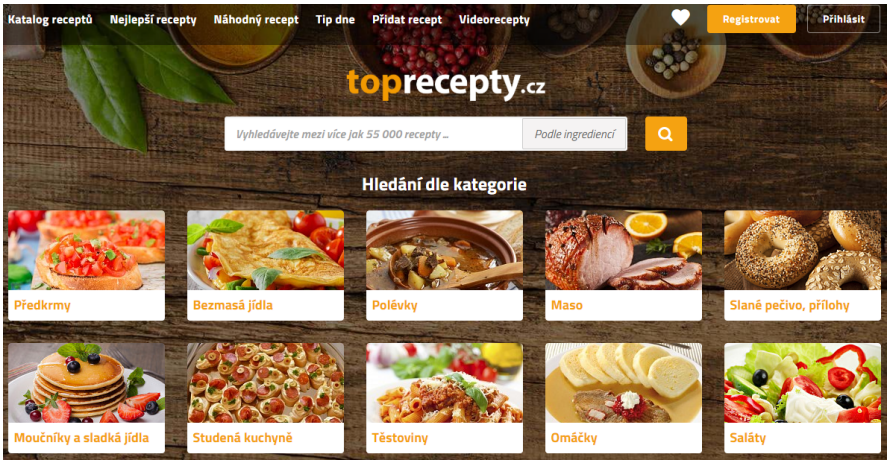
Kromě zmíněného přidávání vlastního receptu může uživatel ještě přidat vlastní fotografie k cizím receptům, přihlásit se k odběru novinek, sledovat aktivity zpřátelených uživatelů, ukládat recepty do vlastní kuchařy. Více funkcí se nepodařilo nalézt z důvodu výše zmíněné chyby.

2.2.2.4 Mobilní zobrazení

Vareni.cz také nemají vlastní mobilní aplikaci, stejně jako *Recepty.cz* upravili zobrazení obsahu stránky pro mobilní zařízení. Vzhled vypadá téměř stejně jako při zobrazení na desktopovém zařízení, je zde pouze několik rozdílů. V horním menu, které se převedlo do hamburger menu, jsou navíc dvě možnosti a to přidat recept a přidat kuchařku, stejně jako na počítači, uživatel musí být přihlášený. Stránka s výslednými recepty se liší pouze v tom, že není kategorie „Další recepty“ a recepty jsou pouze v detailním náhledu.

2.2.3 TopRecepty.cz

TopRecepty.cz jsou výjímečné tím, že na rozdíl od *Recepty.cz* a *Vareni.cz* mají uživatelé možnost ke svým receptům přidávat nejen fotografie, ale i videa. Také obsahuje funkci na zobrazení náhodného seznamu receptů. [10]



Obrázek 2.6: Úvodní obrazovka

2.2.3.1 Vyhledání receptu

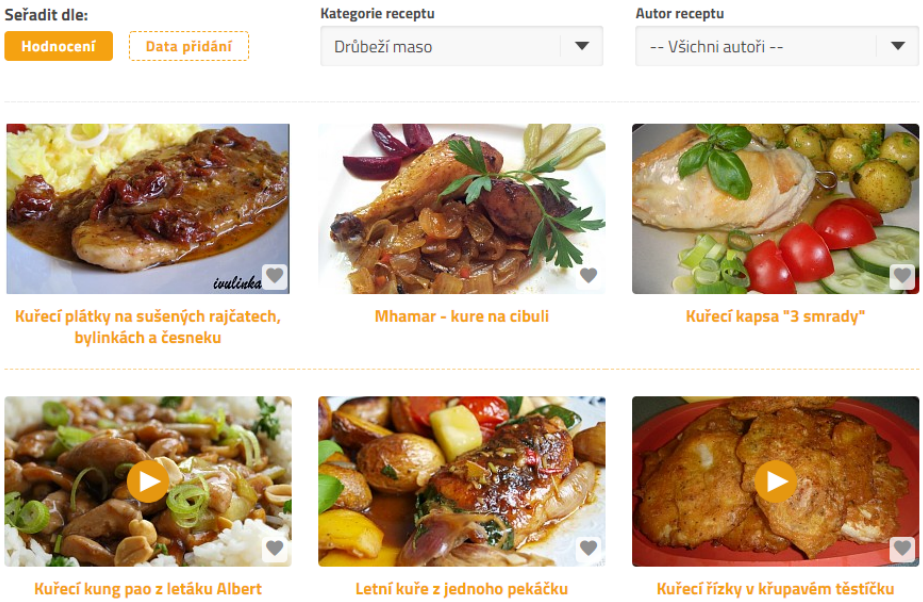
Kromě již zmíněného náhodného vyhledávání nabízí aplikace uživateli množinu kategorií, vyhledávací pole a výběr obsahu ingrediencí v receptu jako možnost vyhledávání. Po přechodu na seznam receptů je možné přidat filtr podle kategorie a podle autora, dané výsledky se dají řadit podle hodnocení a data přidání. Pokud uživatel přejde na katalog receptů ručně přes menu, tak se zmíněný filtr neobjeví, místo něj je pouze možnost vybrat si kategorii pokrmu z detailního výběru a v řazení je navíc možnost seřazení podle doporučení.

2.2.3.2 Přidání receptu

Přidat recept je umožněno jen přihlášeným uživatelům, nepřihlášený uživatel je přesměrován na stránku s přihlášením. Je nutné vyplnit název, kategorii receptu, suroviny a alespoň jeden krok přípravy [11]. Přidané recepty se dají lehce nalézt pod přidanými recepty uživatele.

2.2.3.3 Možnosti uživatele po přihlášení

Kromě prohlížení je většina akcí na stránce povolena pouze pro přihlášené uživatele. Stejně jako přidání receptu, tak i jeho okomentování, ohodnocení, přidání vlastní fotografie a přidání do seznamu oblíbených receptů mohou pouze přihlášení uživatelé. U náhledu receptu navíc přibude možnost připsání



Obrázek 2.7: Filtr výsledků

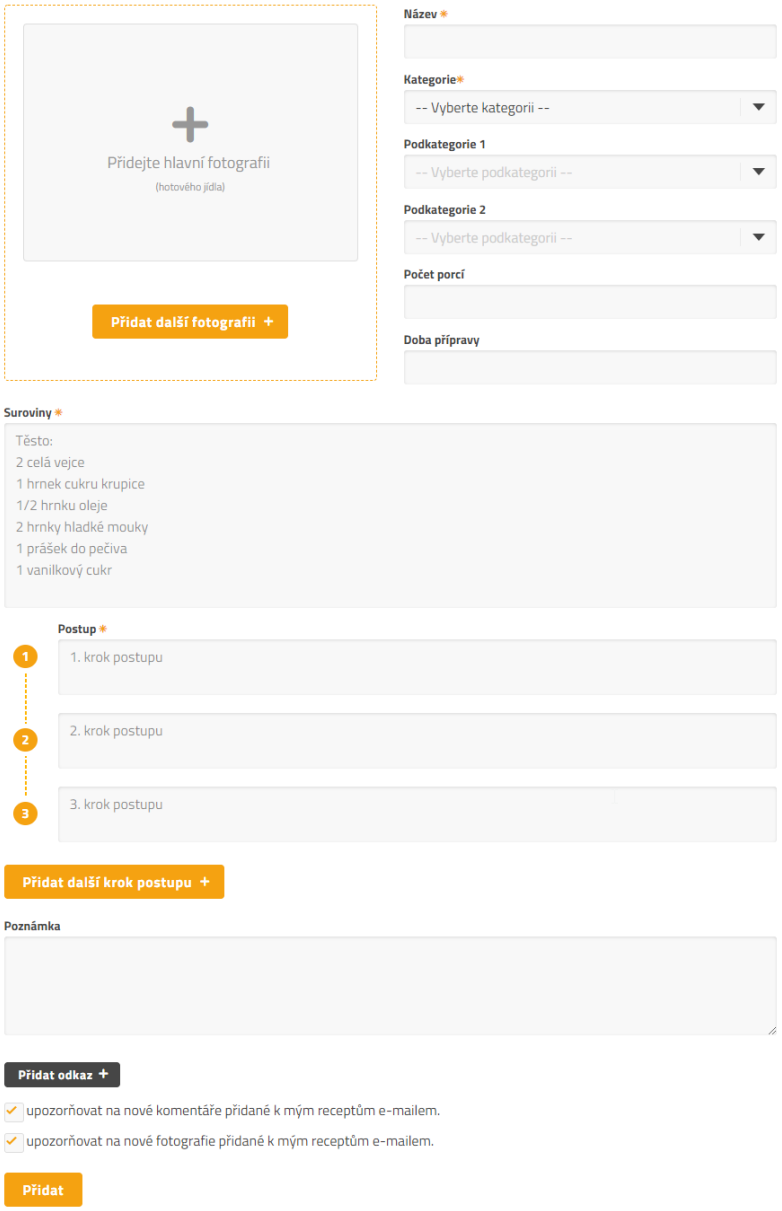
poznámky. V hlavním menu stránky přibyla sekce diskuzí [12], kde si uživatel může zobrazit nové komentáře, příspěvky nebo se také například účastnit skupinové konverzace mezi uživateli.

2.2.3.4 Mobilní zobrazení

Stejně jako předešlé tuzemské aplikace, ani *TopRecepty.cz* nemají vlastní mobilní aplikaci. Na druhou stranu mají svůj web plně responzivní pro mobilní náhled a navíc je v obou náhledech, tedy jak na mobilu tak na počítači, konzistentní.

2.2.4 AllRecipes.com

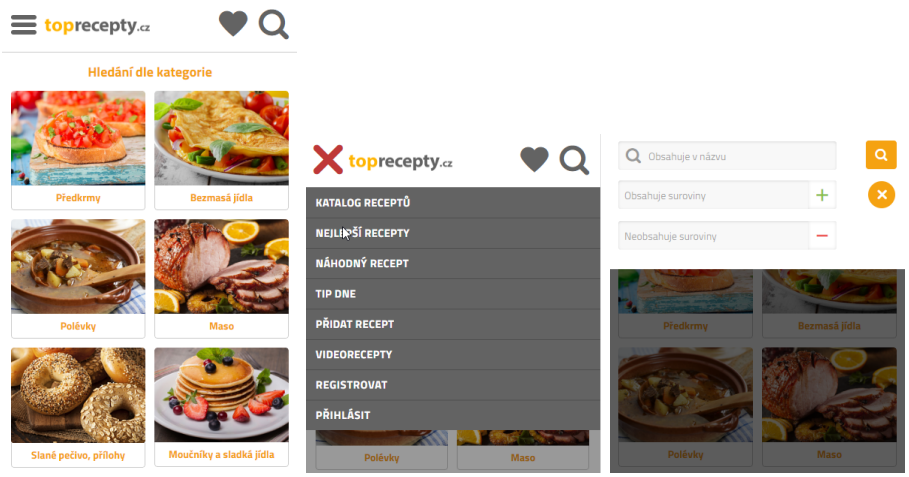
Při vyhledávání receptů v anglickém jazyce se tato aplikace objeví na prvním místě. Stejně jako u stránky *TopRecepty.cz* i zde u receptu lze nalézt videa i fotografie. Aplikace není zaměřena pouze na sdílení receptů, nabízí navíc ještě funkci „Cooking school“ [13], která má zkušební verzi zdarma, ale jinak je placená, kde se uživatelé mohou naučit technikám, tipům a informacím o ingrediencích, jež jim pomohou při vaření. Dále nabízejí svůj magazín a obchod, ale ani jedno není pro Českou republiku podporováno. Aplikace explicitně zmiňuje, že z důvodu vzniku GDPR není umožněno vytváření účtů pro uživatele z Evropy. [14]



Obrázek 2.8: Formulář pro přidání receptu

2.2.4.1 Vyhledání receptu

Vyhledávání je téměř totožné s tím, které se nachází na *TopRecepty.cz*. V hlavičce stránky je vyhledávací pole s možností specifikování obsahu ingrediencí v pokrmu. Díky svému umístění je možné tohoto vyhledávání využít kdekoliv. Na rozdíl od zmíněné české aplikace, *AllRecipes.com* mají možnost vyhledávání podle kategorie kdekoliv, nejen na hlavní stránce, protože je vy-

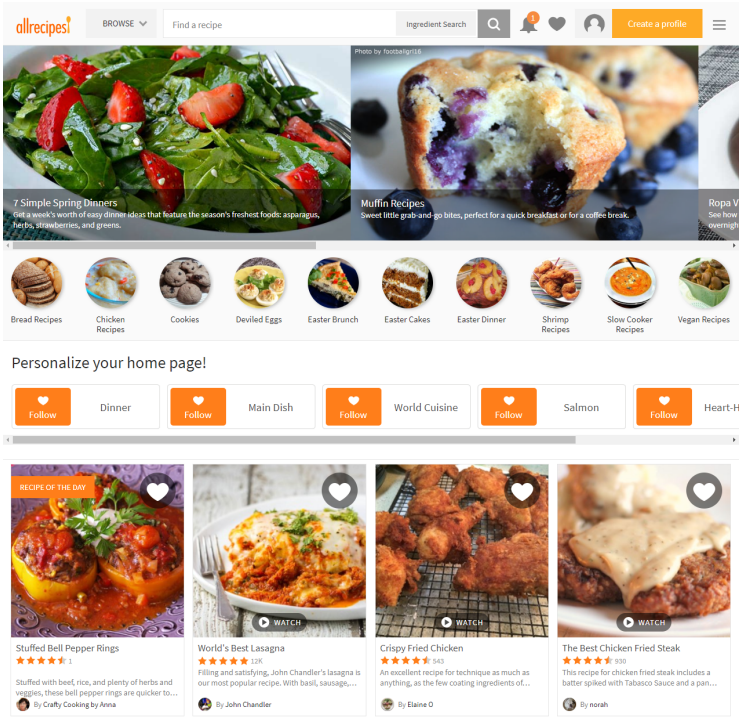


Obrázek 2.9: Ukázka úvodní obrazovky, menu a vyhledávání

hledávání umístěno hned vedle vyhledávácího pole v hlavním menu stránky. Vyhledané výsledky jdou seřadit podle nejlepší shody vyhledávání, data přidání a podle toho jak je recept populární mezi uživateli. Dále aplikace nabízí hledání receptu podle postupného průchodu přes kategorie, čímž si uživatel může postupně vybírat, o co konkrétně má zájem. Průchod funguje tak, že je uživateli nejdříve nabídnuta základní sada a po vybrání nějaké z kategorií je uživatel přesměrován na seznam receptů, které ji splňují a navíc se zobrazí další již více specifická nabídka určena tím, co si uživatel vybral.

2.2.4.2 Přidání receptu

Proces přidání receptu na této stránce nebyl nasimulován, protože je přidávání umožněno pouze přihlášeným uživatelům. Celý postup je ale samotnou aplikací popsán v detailních krocích v sekci „Learn more“ [15], která obsahuje návody při práci s aplikací, pravidel stránky, informace a podobně. Z návodu se lze dočíst, že vytvořený recept je automaticky přidán do uživatelské sekce osobních receptů a navíc, že si uživatel může vybrat, kdo daný recept uvidí. Na výběr jsou tři možnosti; první možnost je viditelnost pro všechny návštěvníky stránky, nezáleží na přihlášení, další možnost je takzvaný „Kitchen Approved Recipe“, což umožní týmu editorů na stránce přidat k receptu důvěryhodnost, přidat nutriční údaje, jak se škáluje množství ingrediencí při změně počtu porcí, a další nástroje. Velkou změnou oproti předchozím aplikacím je poslední možnost, a to přidat recept jako privátní a tím zajistit, že recept nebude nikým jiným vidět. Touto možností je uživatelům umožněno používat aplikaci jako svůj vlastní receptář bez toho, aby do něj mohl kdokoliv zasahovat. [16]



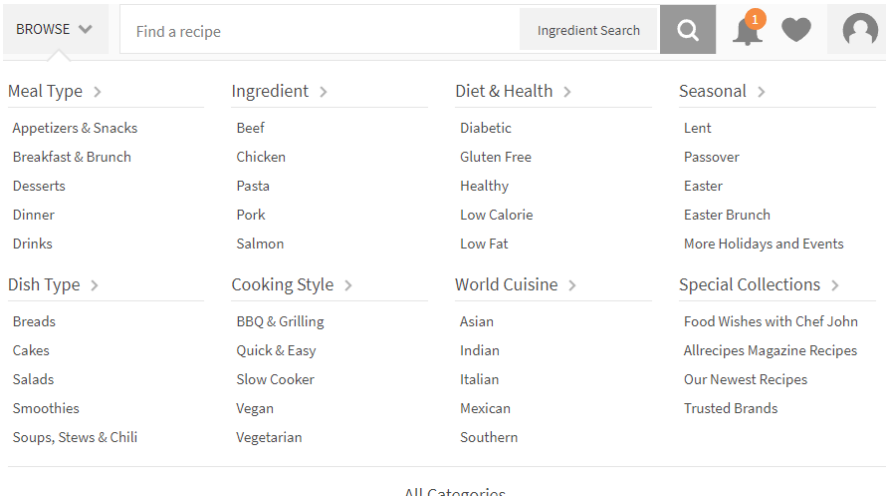
Obrázek 2.10: Úvodní obrazovka

2.2.4.3 Možnosti uživatele po přihlášení

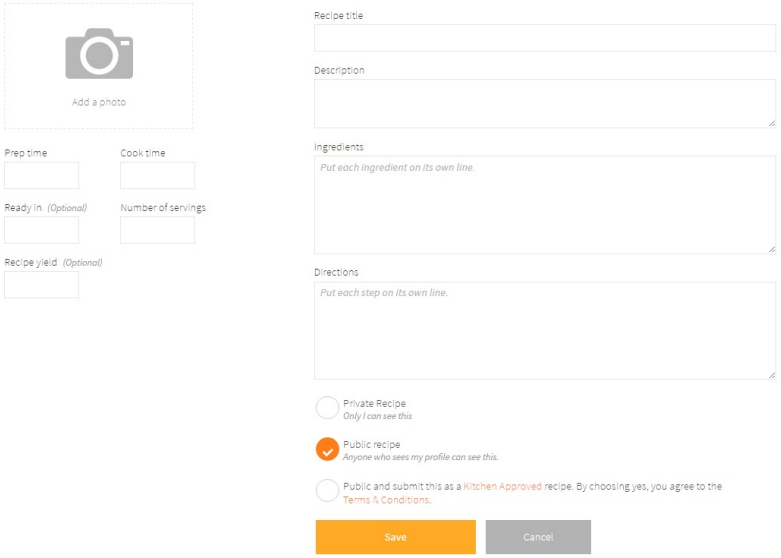
Funkce získané přihlášením jsou například přidání uživatelů do svého listu přátel, možnost přidání receptu do svého listu oblíbených, hodnotit, komentovat, přidat fotografii svého receptu nebo sdělit, že jste daný recept uvařili.

2.2.4.4 Mobilní zobrazení

Jako první aplikace z již zmíněných má svou mobilní aplikaci, která se jmenuje „*Allrecipes Dinner Spinner*“ [17]. Na Google Play má hodnocení 4.5 hvězdy, dostala 67 tisíc hodnocení a byla stažena již více jak 5 miliony uživateli [18]. Při otevření aplikace si uživatel musí zvolit několik kategorií, podle kterých se mu poté generuje obsah na hlavní stránce. Aplikace se skládá z několika částí: vyhledávání receptu, kde základní obsah je tvořen předvybranými kategoriemi, které se dají změnit pod symbolem nastavení na stejné obrazovce a obsah se dá filtrovat podle textu a navíc kromě ingrediencí i podle doby přípavy a diety, jako například pokrmy bezlepkové, vegetariánské nebo bez obsahu mléka. Další součástí aplikace je „Dinner Spinner“, což je funkce náhodného filtru pokrmu. Uživateli je vygenerovaná náhodná kombinace typu pokrmu, jedné ingredience a doby přípravy s možností přejití na nalezené pokrmy. Pokud uživatel není spokojený s výsledkem, může buď to zkusit nové gene-

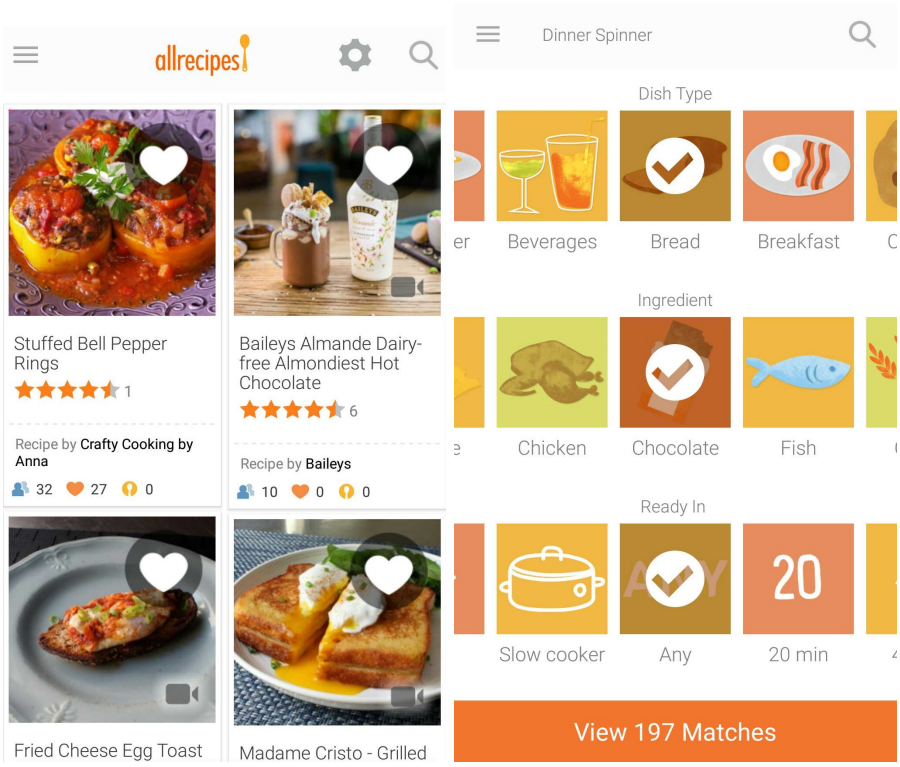


Obrázek 2.11: Filtr výsledků



Obrázek 2.12: Formulář pro založení receptu

rování, a nebo si nějakou ze zmíněných tří částí manuálně změnit. Oblíbené recepty a nákupní seznam jsou zbývající části aplikace. Jak již podle názvu lze poznat, oblíbené recepty obsahují seznam uživatelem zařazených receptů. Ty se zde dají seřadit podle data přidání, hodnocení, názvu a nebo rozřadit do předpřipravených nebo uživatelsky definovaných kolekcí. Do nákupního seznamu se dají ručně zapsat potřebné ingredience pro jednotlivá jídla, nebo při náhledu receptu zvolit možnost „Add all ingredients to list“ a tím se seznam ingrediencí na daném receptu přidá do nákupního seznamu.



Obrázek 2.13: Home screen a Dinner Spinner

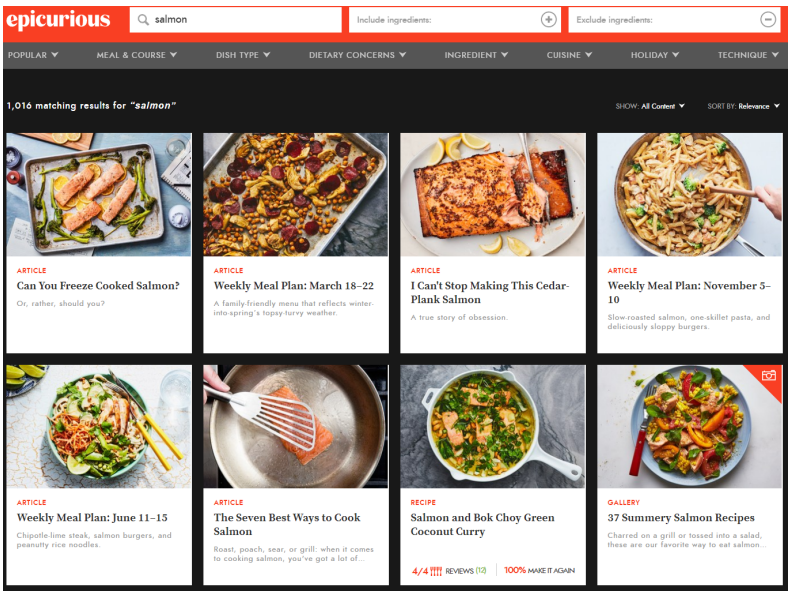
2.2.5 Epicurious.com

Aplikace vznikla v roce 1995 a za svou dlouhou existenci získala mnoho ocenění. Je vhodnou volbou pro každého člověka, jenž se zajímá o gastronomii, protože zde lze dohledat nejen recepty ale i mnoho článků o jídle s ruznými tipy napříč gastronomií. Oproti předchozím receptům nabízí unikátní funkci, a to porovnání až tří receptů vedle sebe. Uživatel si tak může dát více receptů vedle sebe a vidět rozdíly mezi nimi. [19]

2.2.5.1 Vyhledání receptu

Vyhledávání je zabudované do hlavního menu, tedy je možné se k němu dostat z jakékoliv stránky. Na hlavní stránce je vyhledávací pole, které expanduje po jeho otevření. Po zadání a odeslání hledaného textu se výsledky spolu s rozšiřujícími filtry ukáží na nové samostatné stránce. Filtry jsou rozdělené do kategorií a každá z nich má možnost vybrat více položek. Na samém začátku je dále filtr podle obsahu ingrediencí v pokrmu. Seznam výsledků se mění okamžitě při jakékoliv změně v nastavení vyhledávání. Uživatel si může zvolit, o jaký typ výsledku má zájem, zda-li hledá recepty, článek, menu, atd. a dané výsledky navíc seřadit podle data přidání, hodnocení, počtu hodno-

cení, relevantnosti a počtu procent uživatelů, kteří by daný recept chtěli znovu uvařit.



Obrázek 2.14: Přehled filtrů pro vyhledání receptu

2.2.5.2 Přidání receptu

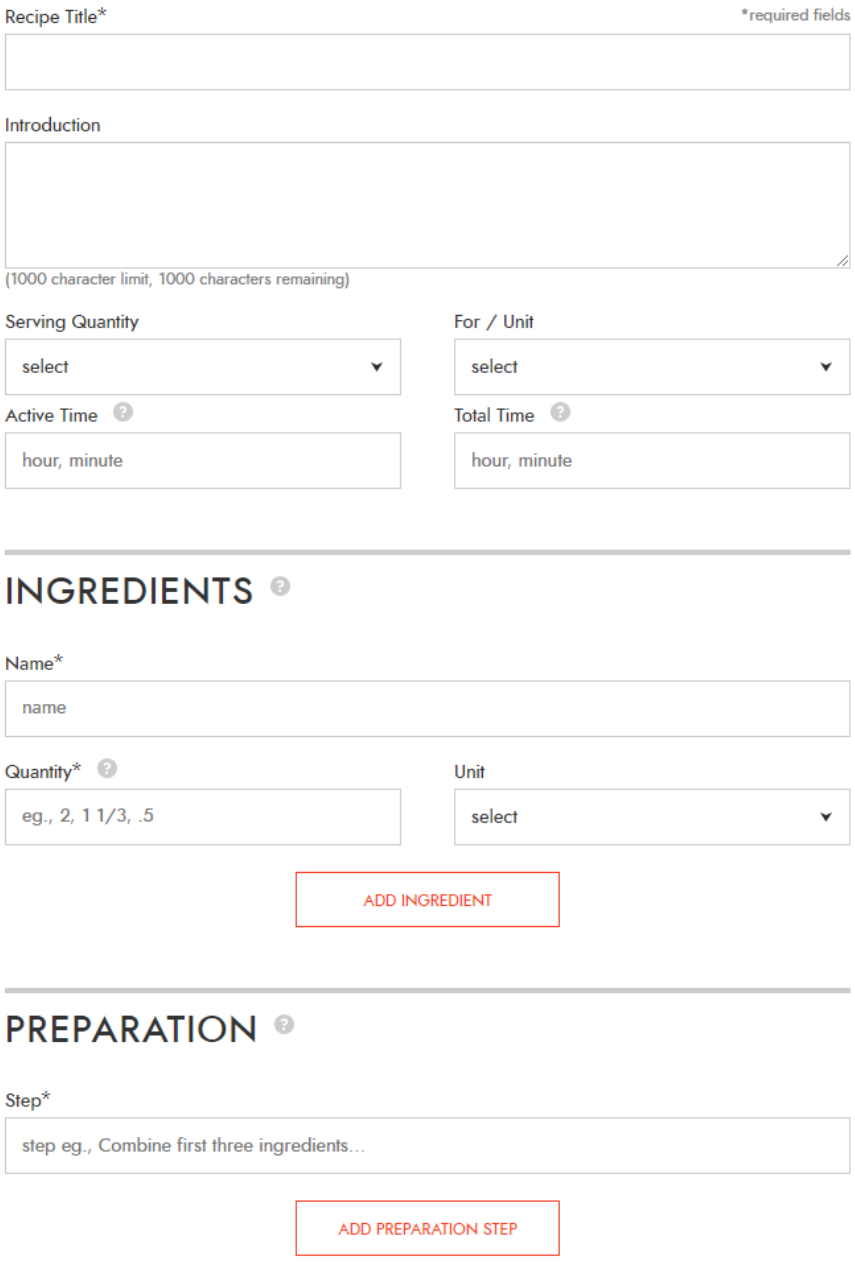
I *Epicurious.com* nabízí uživatelům možnost přidání vlastních receptů, ale jen pokud se přihlásí. Stejně jako *AllRecipes.com* nabízí možnost privátních receptů. Stránka dokonce vyzývá uživatele, aby recepty, které si odněkud převzali, nastavili jako privátní a nešířili je dál pod svým jménem. [20]

2.2.5.3 Možnosti uživatele po přihlášení

Stejně jako v přechozích aplikacích, *Epicurious.com* také nabízí přidání receptu do seznamu oblíbených, poznámky a komentování jako funkce navíc pro přihlášené uživatele. Nabízí ale navíc sestavení si vlastního menu z receptů [20], které jde přidat buď přímo na stránce s menu a nebo na náhledu receptu přes možnost přidání do menu. Tato menu jdou dále sdílet s uživateli.

2.2.5.4 Mobilní zobrazení

Aplikace je nejen plně responzivní a konzistetní při používání telefonního zařízení, ale také má svou mobilní aplikaci. Pro operační systém Android bohužel byla aplikace zrušena.



Obrázek 2.15: Formulář na přidání receptu

Obrázek 2.16: Formulář na přidání receptu

2.2.6 Shrnutí

Většina aplikací poskytuje možnost přidávání receptů pouze přihlášeným uživatelům, stejně tak i komentování a hodnocení receptů. Toho se bude držet i naše aplikace. Co se bude lišit nejvíce, je podpora mobilního zobrazení stránky. Kromě jedné ze zmiňovaných stránek, žádná nemá vlastní mobilní aplikaci. Také možnost přidávání soukromých receptů na většině zmíněných stránek chybí, což neumožňuje používání aplikace jako soukromého receptáře. Právě poslední dvě funkce poskytují velkou konkurenční výhodu pro naší aplikaci, protože bude splňovat obě zmíněné vlastnosti.

2.3 Případy užití

V případě, že nezáleží na přihlášení, bude návštěvník stránky označován jako uživatel, v opačném případě bude přihlášený nebo nepřihlášený.

2.3.1 Webová aplikace

2.3.1.1 UCW1 - Registrace

V hlavním menu nepřihlášený uživatel klikne na tlačítko registrace. Objeví se registrační formulář, do kterého vyplní zvolené jméno a heslo i s potvrzením hesla. Pokud hesla nesouhlasí, aplikace uživatele upozorní zvýrazněním polí. Jakmile se vyplní jméno i shodující heslo, umožní se kliknutí na registrační tlačítko. Backend zkontroluje, zda-li není jméno již použito jiným uživatelem.

Pokud je jméno již použito, uživatel je informován o tomto faktu s výzvou vybrání si jiného uživatelského jména.

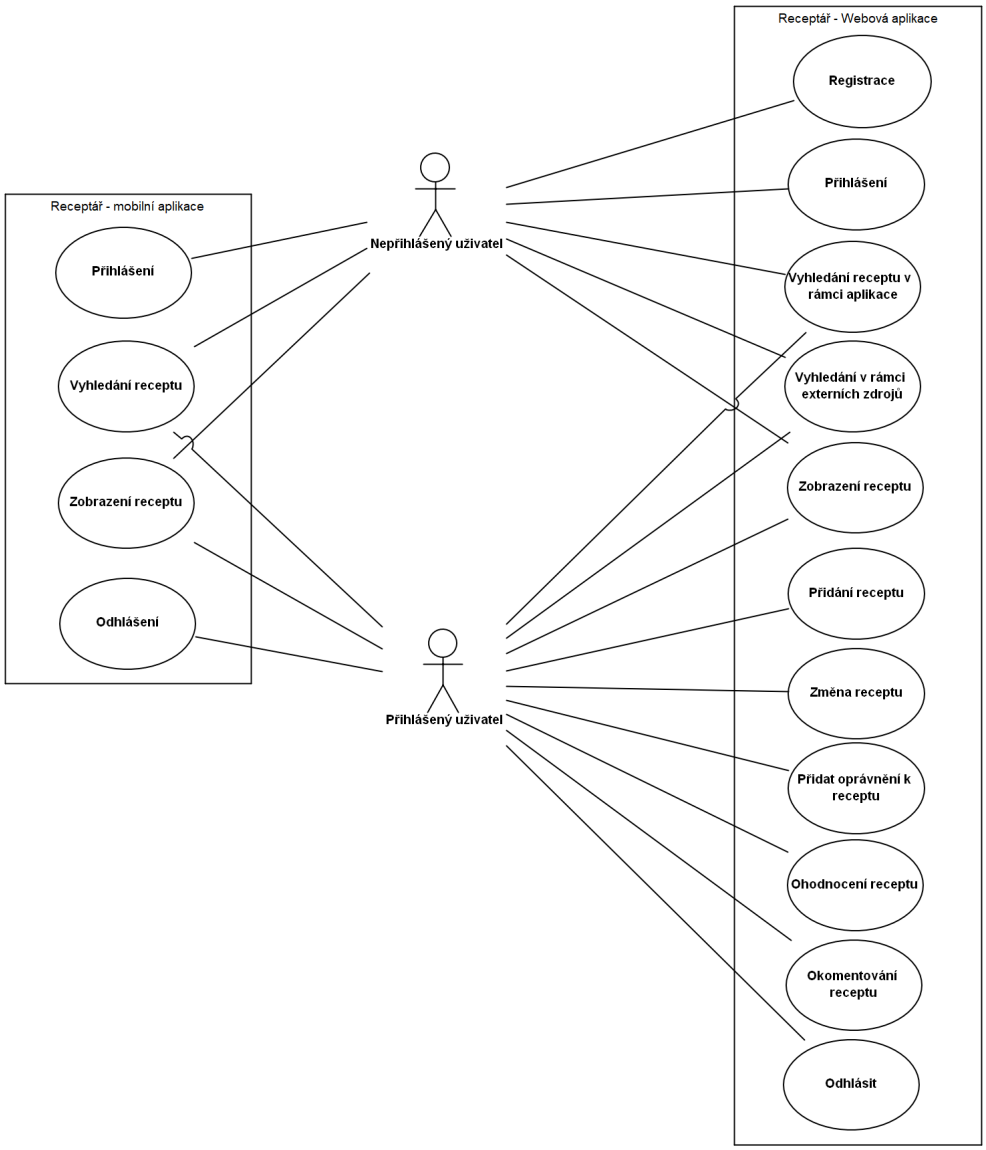
Uživatelský učet se pro danou kombinaci jména a hesla vytvoří tehdy, pokud jméno ještě není použité jiným uživatelem. V případě, že je jméno volné, se vytvoří účet pro danou kombinaci jména a hesla. Po úspěšném vytvoření účtu je uživatel rovnou přihlášen.

2.3.1.2 UCW2 - Přihlášení

Nepřihlášený uživatel v hlavním menu zvolí možnost přihlásit se, čímž se zobrazí přihlašovací formulář. Po otevření vyplní uživatel své jméno, hesla a klikne na přihlášení se. Údaje se odešlou na server, kde proběhne kontrola existence účtu a zda-li souhlasí uživatelské heslo.

Pokud účet neexistuje a nebo bylo zadáno špatné heslo, uživatel je informován o špatné kombinaci jména a heslo.

Jinak proběhne přihlášení uživatele do aplikace.



Obrázek 2.17: Diagram případu užití

2.3.1.3 UCW3 - Vyhledání receptu v rámci aplikace

Uživatel napíše text do vyhledávacího pole, které mu zobrazí nalezené výsledky. Nebo přejde na hlavní stránku, kde se mu zobrazí všechny dostupné recepty, zvolí si filtr, aby specifikoval, co hledá, a výsledky se mu zobrazují během editace filtru.

Výsledky se liší pro přihlášené a nepřihlášené uživatele. Přihlášený uživatel vidí navíc recepty, které vytvořil, a nebo na ně dostal od vlastníka oprávnění.

Pro zobrazení pouze vlastních receptů, přihlášený uživatel zvolí v hlavním

menu možnost přejití na domovskou stránku, kde se mu zobrazí všechny recepty jím vytvořené. Tyto recepty si vyfiltruje podle potřeby.

2.3.1.4 UCW4 - Vyhledání v rámci externích zdrojů

Uživatel zvolí v nabídce možnost pro přejití na stránku s možností výběru jazyka a stránky, na které se bude hledaný výraz vyhledávat. Uživatel zvolí jazyk, stránku, hledaný výraz a stiskne tlačítko pro odeslání požadavku na server. Server si stáhne z konkrétního zdroje HTML kód, ze kterého získá data, ta převede na použitelný formát, který používá aplikace a vrátí uživateli výsledek. Navrácený výsledek se zobrazí uživateli.

2.3.1.5 UCW5 - Zobrazení receptu

Uživatel si vyhledá recept, který ho zajímá, ať už přes globální vyhledávací pole, hlavní stránku, externí zdroje a v případě přihlášeného uživatele v jeho receptech a po kliknutí na něj je uživatel přesměrován na stránku s detailem daného receptu, který si zvolil.

2.3.1.6 UCW6 - Přidání receptu

Přihlášený uživatel přejde na kartu s jeho recepty a zvolí možnost přidání receptu. To ho přesměruje na stránku s formulářem pro vyplnění informací ohledně receptu. Uživatel vyplní název, počet porcí, obtížnost, přidá fotografii, sepíše jednotlivé kroky přípravy a vyplní část ohledně ingrediencí, která se skládá z výběru ingredience, množsví, jednotek a případné poznámky. Také zvolí, jestli je recept veřejný či privátní. Při požadavku na odeslání formuláře se zkontroluje, zda-li jsou pole *jméno*, *počet porcí*, *obtížnost* vyplněné, jestli byl přidán aspoň jeden krok přípravy a jedna ingredience. Požadavek se odešle na server, kde se data zvalidují, založí se recept a vrátí se data pro daný recept. Zakladatel receptu je poté přesměrován na stránku s detailem, kde se mu recept zobrazí.

2.3.1.7 UCW7 - Změna receptu

Přihlášený uživatel přejde na detail svého receptu, kde zvolí možnost editace, čímž se dostane na formulář, který je předvyplněný daty receptu. Poté upraví vybraná pole a recept uloží. Při požadavku na uložení se zkontrolují povinná pole a při jejich správném vyplnění se odešlou data na server, kde se provede jejich validace a následné uložení. Aktuální upravená data se odešlou zpátky, daný uživatel je přesměrován na detail daného receptu, kde jsou zobrazeny data včetně provedených úprav.

2.3.1.8 UCW8 - Přidat oprávnění k receptu

Přihlášený uživatel přejde na kartu svých receptů, zvolí recept, který chce s někým sdílet a zobrazí si ho. Na stránce s detailem zvolí možnost správy oprávnění. Objeví se modální okno, které obsahuje seznam se jmény uživatelů, kteří mají již přístup k receptu, seznam jmen všech registrovaných uživatelů aplikace a možnost pro filtrování jmen registrovaných uživatelů.

Tento uživatel zde buď přidělí a nebo odebere oprávnění.

2.3.1.9 UCW9 - Ohodnocení receptu

Přihlášený uživatel si zobrazí recept, jenž chce ohodnotit, a zvolí hodnocení od 1 do 5.

Pokud daný uživatel již tento recept v minulosti hodnotil, jeho hodnocení se změní na novou hodnotu a celkové hodnocení se změní. V opačném případě, tedy v případě, že hodnotí recept poprvé, se jeho hodnocení uloží, promítne do celkového hodnocení a zobrazí se aktualizovaný počet hodnocení.

2.3.1.10 UCW10 - Okomentování receptu

Přihlášený uživatel si zobrazí recept, jenž chce okomentovat, na konci stránky vyplní do pole obsah komentáře a zvolí možnost pro přidání komentáře. Tento komentář se uloží a zobrazí na konci seznamu komentářů s aktuálním datem a jménem autora komentáře.

2.3.1.11 UCW11 - Odhlásit

Přihlášený uživatel v horním menu zvolí pod svým jménem možnost odhlášení se. Dojde k odhlášení daného uživatele a případně přesměrování na hlavní stránku, pokud byl na stránce určené pouze přihlášenému uživateli.

2.3.2 Mobilní aplikace

2.3.2.1 UCM1 - Přihlášení

Nepřihlášený uživatel přejde na obrazovku přihlášení, kam se dostane buď prvotním spuštěním aplikace a nebo navrácením se na první obrazovku přes zpětné tlačítko. Zde vyplní své přihlašovací údaje a odešle žádost o přihlášení. Server zvaliduje přihlašovací údaje a v případě úspěchu vrátí uživatelská data aplikaci. Uživateli se zobrazí, že je přihlášen.

2.3.2.2 UCM2 - Vyhledání receptu

Uživatel přejde na obrazovku s recepty, kde se mu zobrazí viditelné recepty. Upraví hledaný název receptu a dobu přípravy, čímž se mu recepty vyfiltrují.

Výsledky se liší pro přihlášené a nepřihlášené uživatele. Přihlášený uživatel vidí navíc recepty, které vytvořil a nebo na ně dostal od vlastníka oprávnění.

Přihlášený uživatel vidí navíc možnost navigace na záložku se svými recepty, kde se mu zobrazí pouze recepty jím vytvořené.

2.3.2.3 UCM3 - Zobrazení receptu

Uživatel si vyhledá recept, který ho zajímá, na záložce všech receptů nebo na záložce vlastních receptů v případě přihlášeného uživatele. Po stisknutí daného receptu se zobrazí jeho detail.

2.3.2.4 UCM4 - Odhlášení

Přihlášený uživatel se vrátí zpět na první obrazovku, kde bude možnost se odhlásit. Po jeho zvolení dojde k odhlášení daného uživatele.

2.3.3 Pokrytí požadavků případy užití

Následující tabulka zobrazuje, jak jednotlivé případy užití odpovídají zadaným uživatelským funkčním požadavkům.

Tabulka 2.1: Pokrytí požadavků případy užití ve webové aplikaci

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
UCW1							X				
UCW2								X			
UCW3									X		
UCW4										X	
UCW5			X						X	X	
UCW6	X							X			
UCW7	X	X						X			
UCW8	X		X			X		X			
UCW9			X	X				X			
UCW10			X		X			X	X		
UCW11								X			X

Tabulka 2.2: Pokrytí požadavků případy užití v mobilní aplikaci

	P12	P13	P14	P15
UCM1	X			
UCM2		X		
UCM3			X	
UCM4	X			X

2.3.4 Analýza technologií

Zvolí se technologie takové, aby neporušily zadané nefunkční požadavky, a ty se rozšíří podle dané potřeby. Dle zadání je nutné, aby frontend webové aplikace využil ReactJS framework a backend byl postaven na NodeJS. Pro mobilní aplikaci bude použit framework Flutter, protože nabzí multiplatformní podporu. Navíc je vývoj v něm rychlý díky nízké komplexitě a množství předpřipravených widgetů.

Následující sekce se zabývají zmíněnými technologiemi.

2.3.4.1 NodeJs

Společnost Node.js Foundation vznikla v roce 2009. Již od počátku byl Node.js jeden z nejrychleji rostoucích open-source projektů. [21] Běží na V8 enginu [22], což je open-source engine vytvořený společností Google [23]. V8 engine je napsaný v javascriptu a c++. Funguje na bázi přímé kompilace, a případně optimalizace javascript kódu do nativního strojového kódu před jeho spuštěním. [23]

Node.js byl k roku 2017 stažen více jak 25milionkrát, využívají ho například stránky jako Netflix, PayPal a nebo Airbnb. Kromě aplikační platformy se začíná používat i při práci s daty, modernizaci aplikací a IoT řešení. [21]

Použít Node.js framework je vhodné pro rychlé vytvoření snadno rozšířitelné aplikace.

2.3.4.2 ReactJS

React je javascriptová knihovna pro vytváření uživatelských rozhraní. Na své stránce nabízejí ukázky a návody nejen jak začít React používat, ale i jednoduché návody na pochopení jak React funguje. [24]

Funguje na bázi komponent, které se dělí mezi funkcionální a stavové. Díky tomu, že se jedná o javascriptovou knihovnu, je možné lehce manipulovat s velkými objemy dat bez nutnosti si je ukládat do DOMu [24].

Rozdíl mezi funkcionální a stavovou komponentou je ten, že stavová zapouzdřuje svůj stav, do kterého si může ukládat data a pracovat s nimi, čímž je schopna vytvářet komplexní samostatnou komponentu. Funkcionální komponenty nemají svůj stav a proto se používají jen pro vykreslování komponent, které nepotřebují reagovat na interakci uživatele, a nebo pro zapouzdření jiných komponent.

Komponenty jsou schopny si v rámci hiearchie posílat data a funkce, čímž se složitý datový model dá lehce rozdělit na elementární komponenty, kde každá komponenta je zodpovědná pouze za jednu věc, a pomocí funkcí dávat komponentám nad sebou v hiearchii informaci o změně. Při změně stavu komponenty dojde k překreslení celé části hiearchie pod touto komponentou, čímž se zajistí, že celá komponenta, tedy i její děti, reflektuje změnu dat a je schopna se měnit a přizpůsobovat momentálnímu stavu dat.

2.3.4.3 Flutter

Flutter je framework vytvořený společností Google. Umožňuje vytvořit nativní multiplatformní aplikace, tedy aplikace, které fungují pro iOS i Android zároveň, bez nutnosti vytvoření více projektů. [25]

Pro rychlý vývoj nabízí hned několik funkcí. Jedna z nich je „Hot Reload" [25], což je funkce, která zajišťuje překreslení aplikace při provedení změny v kódu[26]. Další velkou výhodou je velká databáze předem vytvořených widgetů [27], neboli komponent s předdefinovanými vlastnostmi, které se navíc chovají stejně na všech mobilních systémech.

Stejně jako React využívá hiearchie stavových a funkcionálních komponent, kde každá komponenta má vlastní renderer. Takže také dokáže rychle a šetrně obnovovat pouze ty části, které jsou potřeba kvůli změně dat bez nutnosti obnovení celé obrazovky.

Programovacím jazykem, který se používá ve frameworku Flutter, je Dart [28]. Jedná se o programovací jazyk, který je také vyvinut firmou Google a používá se nejen pro tvorbu mobilních aplikací, ale i pro webové stránky, serverové aplikace a nebo okenní aplikace [29].

2.3.5 Možnosti webhostingu

Poněvadž všechna funkcionalita obou částí webové aplikace, tedy backend, je napsána či přeložena do javascriptu, jenž podporuje téměř každý hosting, není těžké najít takový, kam by se dala aplikace nasadit.

Za zamyšlení tedy stojí hlavně volba mezi placeným nebo neplaceným hostingem. Protože neplacené hostingy nabízejí funkce, které pro náše potřeby stačí, budeme hledat tuto variantu.

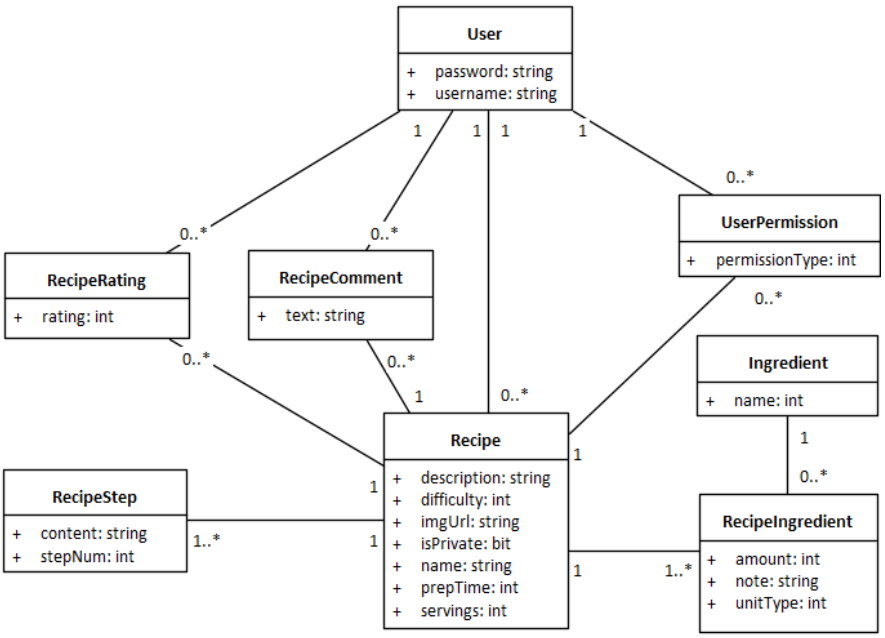
Mezi hostingy s neplacenou verzí patří například i *Heroku* [30], které nabízí bezplatné nahrání a provoz aplikace po dobu čerpání měsíčního kreditu, který se každý měsíc obnovuje. Po vyčerpání kreditu je stránka suspendována do konce období. Dál nabízí bezplatný databázový hosting pro Postgres, což se nám vyplatí z důvodu, že naše aplikace bude používat PostgreSQL. Také podporuje SSH službu, která umožňuje podporu automatickému nasazení naší aplikace. [31]

Mezi další varianty patří napríklad hosting od *Amazonu*, který je primárně placený, ale nabízí i zkušební bezplatnou verzi na vyzkoušení [32], nebo *Firebase* hosting [33], který patří *Googlu*. Oba hostingy nabízejí velké množství funkcí a služeb, některé ve zkušební verzi a jiné jen placené, které umožňují například analyzovat data, frekvenci dotazů nebo nastavovat pravidelné úlohy.

2.3.6 Doménový model

Na následujícím obrázku 2.18 lze vidět základní návrh tříd aplikace, který popisuje jejich vzájemné vztahy. Lze si všimnout, že hlavními objekty domény

je recept a uživatel, protože pouze přihlášený uživatel může zakládat recepty.



Obrázek 2.18: Doménový model

Kapitola 3

Návrh

Díky analýze v předchozí kapitole jsme získali funkcionality, které se musí nyní navrhnout a nadefinovat. Tím se bude zabývat tato kapitola.

Mezi důležité části, které se v této kapitole budou řešit, patří například databázový model, diagram nasazení a rozhodnutí ohledně architektury aplikace.

3.1 Architektura

Rozhodnutí ohledně toho, jaká architektura se pro aplikaci použije, je velmi důležité kvůli správě a rozšíření v budoucnu. Vybraná architektura musí odpovídat nefunkčním požadavkům. V našem případě bylo zadáno, že webová aplikace musí být rozdělena na backend a frontend, neboli klient a server, takže tato část je za nás rozhodnuta. Zbývá určit jednotlivé architektury pro každou z těchto dvou částí a vybrat způsob, jak je spolu propojit kvůli komunikaci.

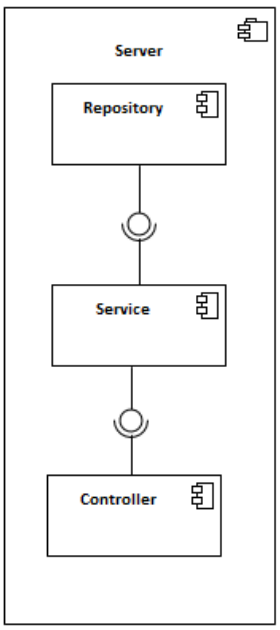
Nejdříve si určíme typ komunikace mezi klientem a serverem. Když se podíváme do nefunkčních požadavků, všimneme si požadavku, že mobilní aplikace má používat API serveru pro komunikaci. Takže jako vhodné propojení připadá v úvahu právě zmíněné API, protože se nebude muset implementovat více způsobů komunikace a zároveň tím, že bude zvolen pouze jeden typ se bude lehce udržovat konzistentnost dat při použití webové aplikace i mobilní.

Api je rozhraní, které server vystaví a klienti ho mohou využívat pro různé druhy požadavků. V našem případě využijeme takzvaného REST API, které podporuje přijímání mnoha druhů požadavků. Použijeme sadu operací označené jako CRUD [34] (create, read, update, delete) operace, které odpovídají požadavkům post, get, put a delete. Komunikace mezi klienty, jak mobilní aplikace, tak i webová aplikace, a serverem bude probíhat pomocí REST, data se budou odesílat ve formátu JSON.

3.2 Server

Serverová část bude mít na starost přijímání i odesílání dat, práci s nimi a ukládání či načítání dat z databáze. Pokud každou z těchto funkcí rozdělíme na separátní nezávislé celky, vzniknou nám tři části. První část starající se o přijímání požadavku od klientů a následné odeslání dat jako odpověď. Druhá část, která obdrží data a bude s nimi dále pracovat, a třetí část, která komunikuje s databází. Tím, že jsme si aplikaci rozdělili do jednotlivých částí, nám to umožní v budoucnu jednotlivé části měnit nebo nahrazovat jinými nezávisle bez dopadu na aplikaci.

Toto rozdělení odpovídá třívrstvé architektuře, která se skládá z prezentační, aplikační a datové vrstvy. Struktura je zobrazena na diagramu na obrázku 3.1, zobrazený *Repository* odpovídá datové vrstvě, *Service* odpovídá aplikační vrstvě a *Controller* odpovídá vrstvě prezentační.



Obrázek 3.1: Diagram komponent serveru

3.2.1 Datová vrstva

Datová vrstva se stará o veškerou komunikaci s PostgreSQL databází. Jedná se jak o získávání dat, tak i o úpravu, vkládání či mazání.

Tato vrstva vrací typ objektu zvaný jako *Entity*, je to objekt který odpovídá struktuře tabulky v databázi.

3.2.2 Aplikačí vrstva

Aplikační vrstva typicky příjme požadavek, případně i s daty, od prezentační vrstvy, a provede nad nimi nějaké logické operace. Může při tom využít i datové vrstvy, pokud ji k dané operaci potřebuje. Stará se tedy o veškerou logiku, která se v aplikaci nachází a slouží k propojení zbylých dvou vrstev. Třídy, které se o logiku zde starají, se nazývají *Service*.

Zmíněné servisy vracejí takzvané DTO (data transfer object), což jsou objekty vzniklé buď převedením z entit a nebo v průběhu použitých operací. [35]

3.2.3 Prezentační vrstva

Prezentační vrstva má na starosti komunikaci s klienty a se svou aplikační vrstvou. Příjme požadavek, zkontroluje, jestli datově odpovídá dané operaci, když odpovídá, zavolá metodu z příslušné servisy, která daný požadavek zpracovává, a čeká na výsledek, který následně upraví do předpokládaného tvaru a odešle zpět klientovi. V opačném případě požadavek zamítne.

3.3 Klient

Klient má na starosti interakci s uživatelem a případně reagovat na jeho akce. *React* [24] samotný je vhodný pro menší projekty, kde se neudržuje mnoho dat napříč celou aplikací, protože se pak musí vytvořit rodičovská stavová komponenta se stavem odpovídající všem datům v aplikaci a pak postupně data posílat přes všechny komponenty v hiearchii mezi ním a komponentou, která s daty pracuje.

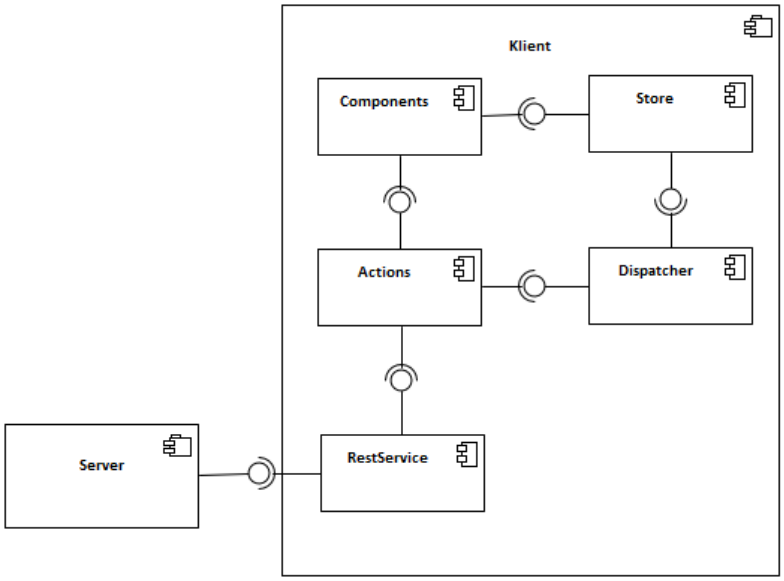
Pro složitější aplikace se tedy používají rozšíření ve formě knihoven. Jednou z nich je knihovna *Redux* [36], která poskytuje celé aplikaci datové úložiště, do kterého mají přístup všechny komponenty. Dále knihovna *React-Redux* [37] sloužící k propojení obou zmíněných knihoven.

Aplikace naprogramované v Reactu patří do kategorie jednostránkových webových aplikací, protože obvykle nedojde k přenačtení celé stránky, pouze částí, které se změní. Mnoho knihoven se dá rozšířit na typovou variantu s použitím *Typescriptu* [38], čímž se aplikaci přidá typová kontrola a kontrola stavu jednotlivých objektů. Počáteční časová investice do nastavení *Typescriptu* [38] se rychle vrátí při práci na aplikaci.

React má podrobnou dokumentaci na svých webových stránkách.

3.3.1 Popis architektury

Na následujícím obrázku 3.2 je znázorněna komunikace komponent v klientské aplikaci a komunikace se serverem.



Obrázek 3.2: Diagram komponent klienta

3.3.1.1 RestService

Služba, která zajišťuje komunikaci se serverem. Tato komunikace je zajištěna pomocí knihovny *axios* [39].

3.3.1.2 Actions

Jedná se o akce, které komunikují s centrálním úložištěm aplikace pomocí `dispatch`. Než tyto akce modifikují úložiště, mohou vykonat téměř jakoukoliv činnost. V našem případě ke komunikaci se serverem a získání dat.

3.3.1.3 Dispatcher

Na základě zavolaných akcí mění centrální úložiště aplikace.

3.3.2 Rozdělení grafického rozhraní

V této sekci je výčet stránek, které webová aplikace obsahuje.

3.3.2.1 Hlavní strana

Uživatel zde vidí všechny recepty, ke kterým má přístup. Recepty může filtrovat pomocí obtížnosti, doby trvání a jména. Kliknutím na konkrétní recept se zobrazí jeho detail.

3.3.2.2 Externí recepty

Zde je možné zvolit jazyk a příslušnou externí stránku, na které se má recept vyhledat. Kliknutím na konkrétní recept se zobrazí jeho detail.

3.3.2.3 Osobní receptář

Přihlášený uživatel zde vidí všechny recepty, které vytvořil, a může filtrovat pomocí obtížnosti, doby trvání a jména. Dále uvidí možnost na vytvoření nového receptu, který ho přesune na formulář. Kliknutím na konkrétní recept se zobrazí jeho detail.

3.3.2.4 Formulář na vytváření/editování receptu

Formulář, ve kterém uživatel může vyplnit/změnit data a následně zvolit možnost uložení, které ho přesune na detail receptu.

3.3.2.5 Detail receptu

Obsahuje detailní výpis informací zvoleného receptu. Pokud je uživatel vlastník, vidí zde vypsané uživatele, kteří mají přidané oprávnění k zobrazení receptu, a možnost na správu oprávnění.

3.3.2.6 Správa oprávnění

Uživatel v něm může přidat nebo odebrat oprávnění jinému uživateli.

3.3.2.7 Přihlášení

Zde uživatel může vyplnit své jméno a heslo a přihlásit se.

3.3.2.8 Registrace

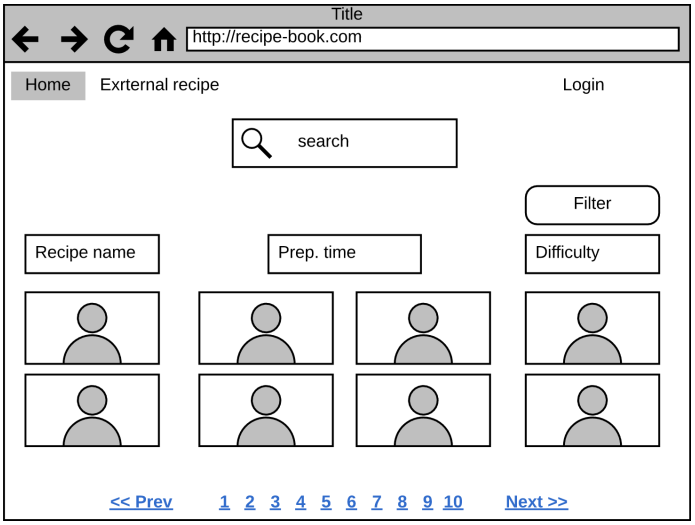
Zde uživatel může vyplnit své jméno, heslo a potvrzení hesla a registrovat se.

3.3.3 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní vychází z analýzy, která poukazuje na důležité části aplikace. Mezi tyto části patří například zobrazení seznamu receptů a zobrazení detailu receptu.

3.3.3.1 Seznam receptů

Na obrázku 3.3 je návrh zobrazení seznamu receptů. Obsahuje zobrazený seznam, ukázku možných filtrů, podle kterých lze seznam filtrovat, a stránkování výsledků.



Obrázek 3.3: Návrh zobrazení seznamu receptů

3.3.3.2 Detail receptu

Na obrázku 3.4 je zobrazen návrh rozložení komponent na detailu receptu, při prohlížení jeho vlastníkem.

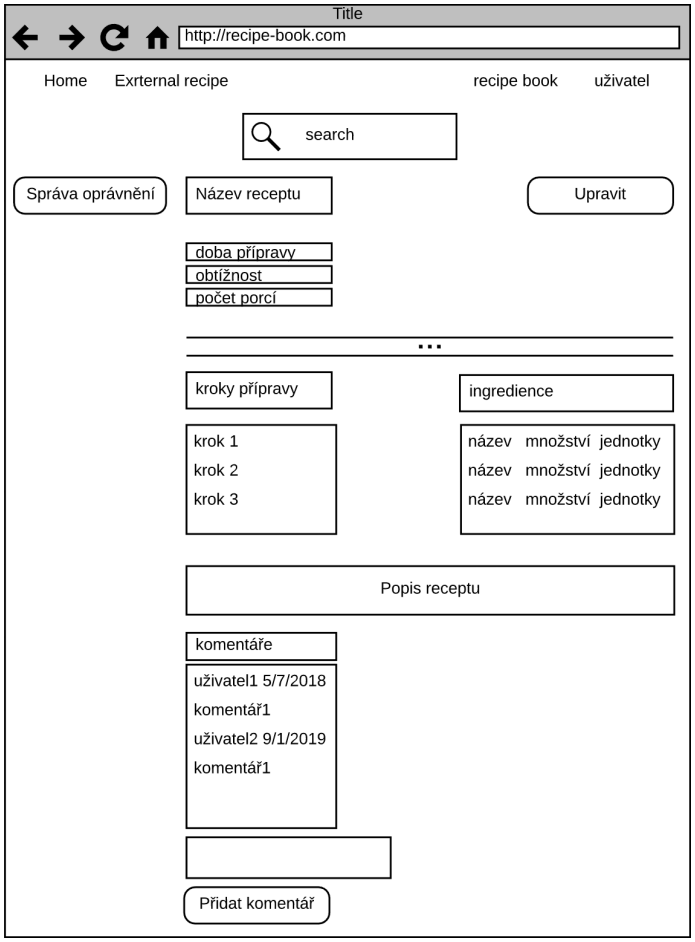
Na levé straně od názvu je tlačítko na správu oprávnění a na pravé straně tlačítko, které uživatele přesune na možnost editace receptu. Hned pod názvem lze vidět výpis doby přípravy, obtížnosti a počtu porcí, po kterém následuje výpis všech kroků přípravy a ingrediencí.

Popis receptu je až na samém konci výpisu detailu receptu, protože se nepovažuje za klíčovou informaci definující recept. Na konci obrazovky je pak vidět možnost přidávání komentářů k tomuto receptu. U komentáře je zobrazen název uživatele, datum přidání a samotný obsah komentáře.

3.4 Mobilní aplikace

Na rozdíl od klientské aplikace, má mobilní aplikace na starosti pouze zobrazení dat.

Protože mobilní aplikace nemá aplikační vrstvu s logikou, jedná se tedy o získávání a následné zobrazování dat, komunikuje prezenční vrstva přímo s datovou. Je potřeba se zamyslet nad opětovným získáváním dat ze serveru. Aby se minimalizovala doba odezvy a počet requestů na server, je zavedena vrstva navíc, viditelná na obrázku 3.5, která překrývá komunikaci s repozitářem. Tato vrstva má na starosti cachování, které je implementované způsobem, že všechny požadavky na repozitář jdou přes cachovanou vrstvu a ta přidává logiku kolem získávání dat. Nejdříve zkontroluje, zda-li dotaz na tyto data nebyl již vykonán, když byl, tak z cache data získá. V opačném



Obrázek 3.4: Návrh zobrazení detailu receptu

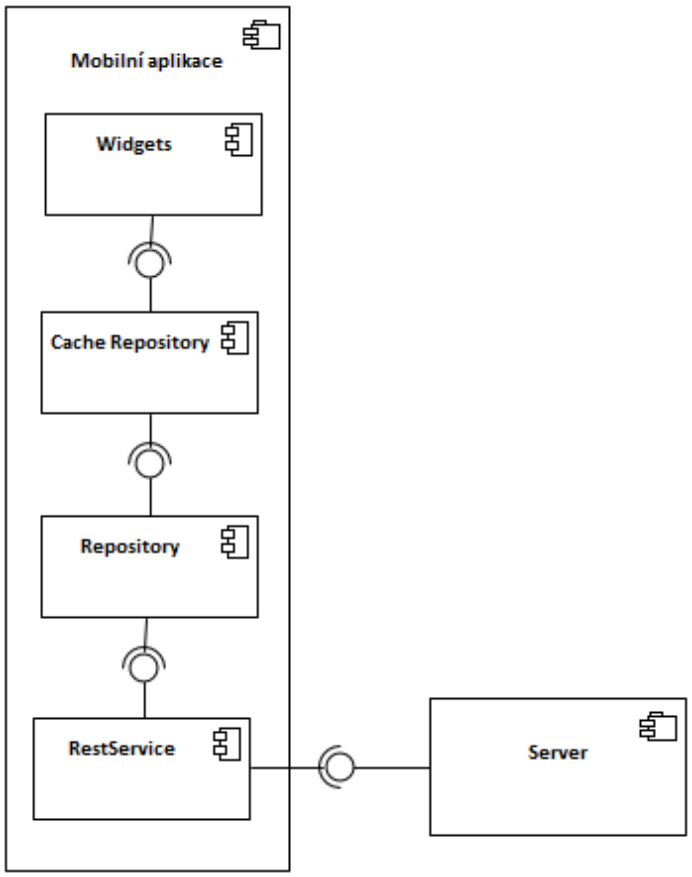
případě zavolá repozitář s požadavkem na data, které následně uloží do cache, a vrátí je jako odpověď.

3.4.1 Rozdělení grafického rozhraní

V této sekci je výčet stránek, které mobilní aplikace obsahuje.

3.4.1.1 Hlavní strana

Uživatel zde vidí dvě záložky, mezi kterými může přepnout, první záložka obsahuje všechny recepty, ke kterým má přístup. V druhé záložce, pokud je uživatel přihlášen, uvidí všechny recepty, které vytvořil, jinak je informován o nutnosti přihlášení se. Recepty se mohou filtrovat pomocí obtížnosti, doby trvání a jména. Kliknutím na konkrétní recept se zobrazí jeho detail.



Obrázek 3.5: Diagram komponent mobilní aplikace

3.4.1.2 Detail receptu

Obsahuje detailní výpis informací zvoleného receptu.

3.4.1.3 Přihlášení

Zde uživatel může vyplnit své jméno a heslo a přihlásit se nebo přejít do aplikace bez přihlášení se.

3.4.2 Návrh uživatelského rozhraní

V následujících sekcích je ukázka návrhu pro přihlášení se a zobrazení vyhledaných receptů.

3.4.2.1 Přihlášení

Na obrázku 3.6 vidíme ukázku obrazovky na přihlášení uživatele. Protože podle analýzy má mít uživatel možnost používat aplikaci i bez přihlášení se,

proto je zde vidět i tlačítko na pokračování do aplikace bez přihlášení.

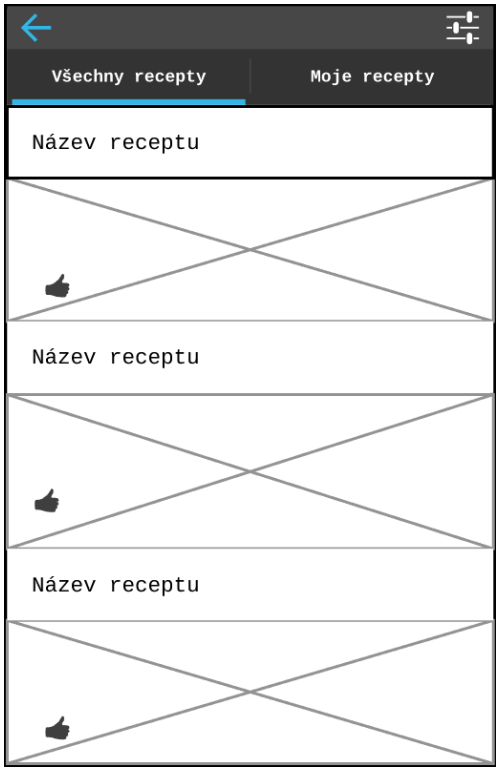
Obrázek 3.6: Návrh přihlašovací obrazovky v mobilní aplikaci

3.4.2.2 Seznam receptů

Obrázek 3.7 vyobrazuje návrh zobrazení seznamu receptů. Nad seznamem je vidět možnost přepnutí záložek mezi všemi dostupnými recepty a recepty vytvořených uživatelem. Nad záložkami je možnost navrácení se na obrazovku přihlášení, pomocí tlačítka zpět, a možnost nastavení, kde se dá nastavit filtrování zobrazených receptů. U samotných receptů je vidět jejich název, informace, které blíže specifikují recept, a případně obrázek, pokud byl uživatelem zadán.

3.5 Databázový model

Databázový model lze vidět na obrázku 3.8. Liší se od doménového modelu hlavně tím, že každá tabulka obsahuje informaci o svém vytvoření, změně nebo vymazání. Informace o vymazání je pro nás důležitá, protože jsme se rozhodli pro *soft delete* [40] variantu, která se od *hard delete* varianty liší tím,



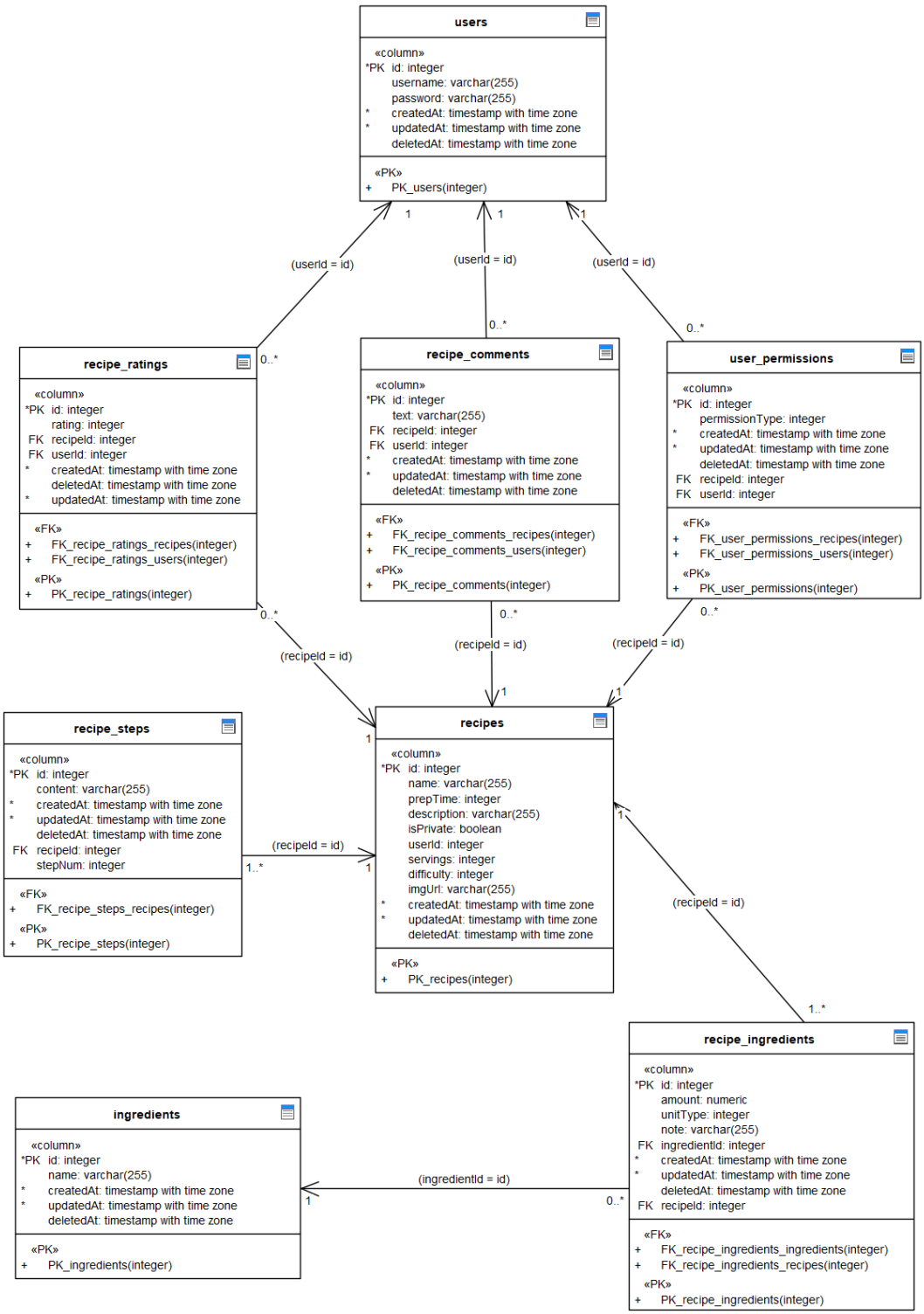
Obrázek 3.7: Návrh zobrazení seznamu receptů v mobilní aplikaci

že svá data zásadně nemaže trvale z databáze, ale pouze jim nastaví příznak o vymazání.

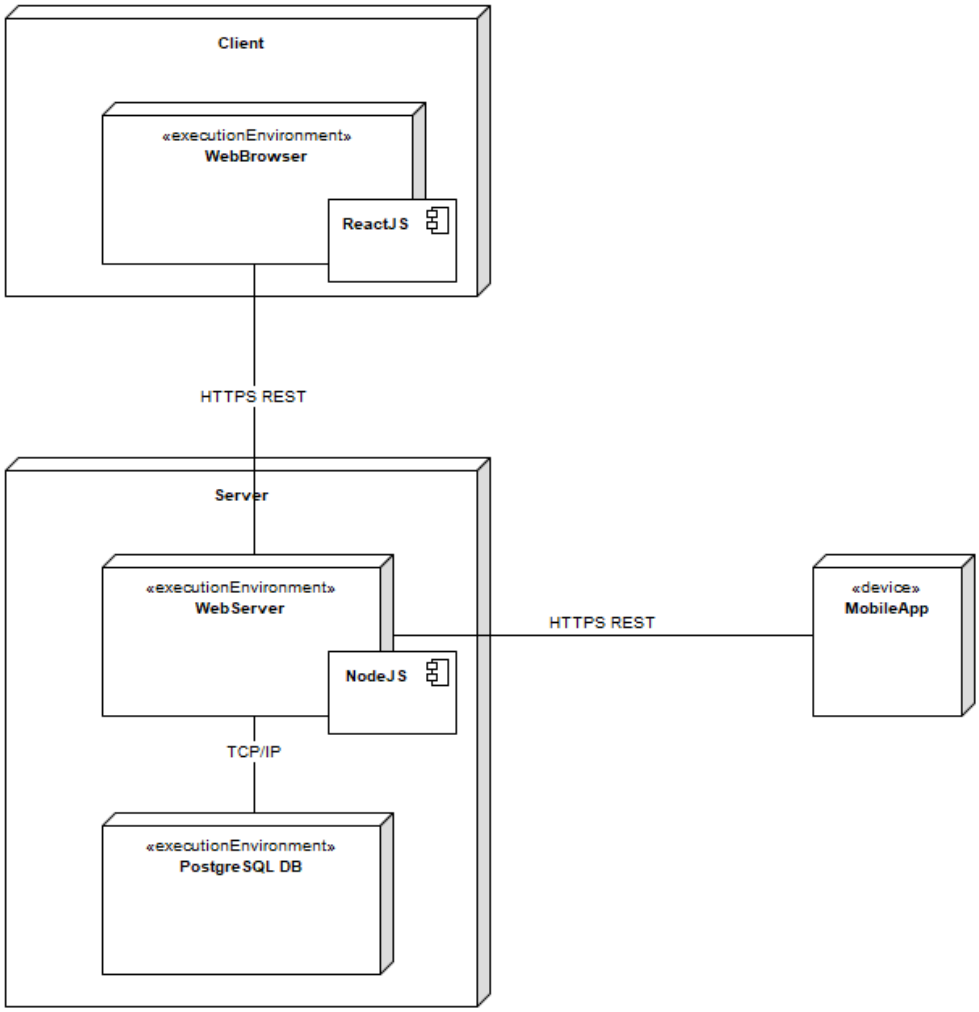
Soft delete funguje na bázi přidání sloupce, který indikuje jeho aktivní stav, do každé tabulky. Tento sloupec se většinou pojmenovává jako *deletedAt* a je typu *date*, *datetime* nebo *timestamp*. Při vymazání daného záznamu se nezavolá příkaz *delete*, kterým by se data vymazaly, ale *update* a nastaví se aktuální datum do sloupce *deletedAt*. Při získávání dat se filtrují data s vyplněným sloupcem *deletedAt*.[40]

3.6 Nasazení

Diagram 3.9 zachycuje komunikaci a nasazení jednolivých částí aplikace.



Obrázek 3.8: Databázový model



Obrázek 3.9: Diagram nasazení

Kapitola 4

Implementace

V této kapitole je rozebrána implementační část všech částí této práce. Začne se serverovou částí, protože ji dále využívají obě klientské aplikace, pak webovou aplikací a na závěr mobilní aplikací.

Pro vývoj všech zmíněných částí byl použit editor *Visual Studio Code*, který je zadarmo. K zálohování a verzování práce byl použit školní GitLab repozitář.

4.1 Server

Prostředí *NodeJS* obsahuje již všechny potřebné prostředky pro spuštění programů napsaných v Javascriptu.

Aplikace se dá spustit více způsoby, jeden způsob je zavolat v terminálu příkaz *node* následován cestou ke kořenovému souboru, při práci s Node.js je konvence ho pojmenovat „App.js“. Daný příkaz řekne prostředí, že si přejeme spustit daný soubor, takže se vezme daný soubor spolu se všemi referencemi a interpretuje jej do strojového kódu, který se nemusí interpretovat a proto se může ihned spustit. Druhý způsob je zavolat taktéž z terminálu příkaz *npm run* s názevem scriptu. Daný příkaz prozkoumá soubor *package.json* v sekci skriptů a vybere ten, který odpovídá názvem. Tyto scripty umožňují předdefinovat sekvenci příkazů potřebných k vykonání akce. V našem případě se použila druhá varianta, protože bylo potřeba nadefinovat jak scripty pro spuštění samotné aplikace v developerské i produkční verzi, tak i pro spuštění testů nebo sestavení projektu pro produkční nasazení.

4.1.1 REST rozhraní

Tabulka 4.1 zobrazuje detailní popis REST rozhraní.

Tabulka 4.1: Serverové REST rozhraní

URI	Metoda	Popis
/api/comments/:id	GET	Vrací komentáře patřící receptu
/api/comment	POST	Vytvoří nový komentář
/api/external/recipes	GET	Vrátí recepty z vybrané externí stránky
/api/external/recipe	GET	Vrátí detail receptu z externí stránky
/api/ingredients/:name	GET	Vrátí ingredience odpovídající výrazu
/api/seedIngredient	POST	Vytvoří ingredience
/api/rating/:id	GET	Vrátí hodnocení patřící receptu
/api/rating	POST	Vytvoří nové hodnocení
/api/recipes	GET	Vrátí vyfiltrované přístupné recepty
/api/recipes/:id	GET	Vrátí detail receptu
/api/recipePermissions/:id	GET	Vrátí oprávnění patřící receptu
/api/recipe	PUT	Upraví existující recept
/api/recipe	POST	Založí nový recept
/api/addPermission	POST	Vytvoří nové oprávnění
/api/removePermission/:id	DELETE	Vymaže existující oprávnění
/api/userRecipes	GET	Vrátí vyfiltrované recepty uživatele
/api/recipe/:id	DELETE	Vymaže recept

4.1.2 Datová vrstva

Pro práci s databází se používá Sequelize ORM [41], které má MIT licenci. Sequelize má rozsáhlou dokumentaci. Právě kvůli zmíněné dokumentaci a častému spojování s Node.js v článcích jsem se rozhodl pro toto ORM.

4.1.2.1 Definice databáze

Začít používat *Sequelize* [41] není nijak těžké při použití čistého javascriptu, stačí přidat konfiguraci připojení, nadefinovat entity a přidat ji do schématu pro vytvoření databázové struktury. Problémy nastávají po přidání typescriptu. I přes to, že Sequelize obsahuje i svou typovanou verzi knihovny, musí se některé části dopsat ručně a ne vždy to je intuitivní.

Pro jednotlivé entity se musí nadefinovat název tabulky, jednotlivé sloupce i s jejich typy. Každá entita obsahuje již předem definované sloupce, které se dají přepsat. Například čas vytvoření a poslední změny je dodán ke každé entitě. Dále se dají upravit různá nastavení ohledně soft delete, kde v tabulce přibude sloupec s časem vymazání, nebo například asociace.

Právě asociace jsou velkou výhodou kvůli možnosti definování propojení jednotlivých tabulek mezi sebou. Dají se poté použít jak pro složitější dotazy na databázi, když chceme získat data a k tomu i informace navázaných objektů, nebo pro filtrování, nebo pro předem definované operace frameworkem *Sequelize* [41].

4.1.2.2 Operace nad databází

Sequelize poskytuje množinu operací nad databází, jako například `insert`, `update` nebo `delete`. Těchto operací se da využít tím způsobem, že se vezme model, který odpovídá tabulce v databázi, a zavolá se na něm příslušná operace. Ukázka použití operace můžeme vidět na obrázku 4.1, všimneme si, že `RecipeIngredient` je v tomto případě model, který odpovídá tabulce *recipe_ingredients*, a používá se z něj metoda `destroy`, která odpovídá *SQL* operaci `delete`.

Každá tato metoda vrací *BlueBird promise*. *Promise* je asynchroní objekt, který slibuje, že vrátí nějakou hodnotu jakmile skončí výpočet uvnitř něj. Tento výsledek navrátí buď pomocí metody `resolve`, pokud operace skončila úspěšně, a nebo `reject`, pokud došlo k nějaké chybě. Pokud potřebujeme použít hodnotu, kterou *promise* vrací, může se použít kombinace, kde se metoda označí jako `async` a poté se v ní dá využít příkazu `await`, který říká, že chceme počkat na výsledek dané operace. Další možností, jak hodnoty získat, je využití metod `then` a `catch`. Jedná se o metody, které poskytuje *promise*, a odpovídají uspěšnému a neuspěšnému stavu. Každá z metod přijímá callback funkci, která se zavolá v daném případě.

Jak si lze dále všimnout, tak výsledná *promise*, kterou dotaz vrací, se konvertuje do *promise* jiného typu. Rozdíl mezi nimi je ten, že výsledná *promise*, kterou vrátí metoda `deleteRalation`, je ta, která je nativně poskytnutá prostředím. Tedy všude jinde v aplikaci se používá tento typ *promise* a proto se zde kvůli konzistenci na tento typ převádí.

Protože by bylo těžké uhlídat všechny výskyty volání operací nad modely, jsou jednotlivé modely uzavřeny v repozitářích, které centralizují jejich použití. Tím se zajisťuje, že všechny operace nad danou tabulkou jsou na jednom místě.

4.1.2.3 Vytvoření databáze

Při spuštění aplikace se provede případné vytvoření databázového schématu. Dá se nastavit, zda-li se tabulky vytvoří vždy a nebo jen, když se v databázi nenachází.

Při spuštění aplikace se provádí spojení s databází, kontrola a porovnání aktuálních tabulek s těmi, které se zrovna nachází v databázi. Pokud v definici přibyla tabulka, která se v databázi nenachází, je vytvořena. Pomocí příznaků u modelu se dá definovat kompletní přemazání aktuální struktury

```
deleteRelation (id: number) : Promise<number> {
        return new Promise<number>(resolve => {
                return Connection._db.RecipeIngredient.destroy({
                        where: {
                                id: id
                        }
                }).then(result => resolve(result));
        });
};
```

Obrázek 4.1: Ukázka zavolání Sequelize operace nad modelem

tabulky strukturou v modelu. Jakmile se zkontrolují tabulky databáze, dojde k nastavení asociací mezi modely.

4.1.3 Aplikační vrstva

Jak již bylo řečeno, aplikační vrstva obsahuje třídy, které pracují nad daty získanými z jednotlivých repozitářů. Kromě těchto se zde nachází i třídy pro získávání dat z externích zdrojů. Protože o práci a komunikaci mezi vrstvami jsme již mluvili, tato sekce je zaměřena na popis funkce získání dat z externích zdrojů.

4.1.3.1 Rozšířitelnost

Aby bylo rozšíření externích zdrojů snadné, jsou jednotlivé zdroje rozděleny podle jazyka a později ještě podle stránky.

Pro přidání dalšího zdroje stačí vytvořit třídu, která se postará o získání seznamu receptů a dále konkrétního receptu, tu nadefinovat ve třídě, která se stará o tuto jazykovou variantu a přidat do číselníku, který obsahuje definice pro jednotlivé jazyky. Tento číselník se poté používá nejen v aplikační vrstvě pro vybrání správné třídy pro získání dat, ale i na klientovi, kde se objeví v seznamu tato možnost pro vybrání.

4.1.3.2 Získání dat

Pro každý zdroj tedy existuje jedna třída, která má dvě metody. Jednu pro získání listu receptů a druhou pro získání konkrétního receptu, když chce uživatel vidět detail zvoleného receptu. Obě metody posílají dotaz na konkrétní adresu specifikovanou stránkou a akcí, ze které následně těží data. Poněvadž se návrh formuláře ingrediencí liší od různých implementací na externích stránkách, musel se pro ně vytvořit nový všeobecný model.

Těžba dat je implementována pomocí knihovny *cherio*, která implementuje jádro jQuery pro využití na serveru. Tato knihovna si nahraje HTML

strukturu do paměti a vytvoří z něj DOM model. Pomocí tohoto modelu se pak hledají konkrétní elementy, ve kterých jsou uložena hledaná data.

K převádění nalezených elementů nad textem se používá knihovna *he*. Jedná se o robustní kodér a dekodér HTML kódu. Tato knihovna vezme string a příkazem `encode` nahradí všechny znaky příslušnými HTML značkami. Naopak pomocí `decode` dekóduje HTML do podoby stringu. Výsledkem této metody je tedy platný HTML kód.

4.1.4 Prezentační vrstva

Prezentační vrstva je realizována *controllery*, jejichž akce se zapisují pod jednotlivé adresy *routeru*.

Express.js [42] je webový server, který obsahuje svůj *router*, do kterého lze nadefinovat jednotlivé routovací adresy. Při startu aplikace se vezme tento router a pošle se objektu, který shromažďuje všechny objekty s jednotlivých námi definovanými routami. Ten pošle do každého tohoto objektu referenci na router a objekt do něj zapíše všechny definované adresy, typ dotazu a referenci na metodu controlleru, která se má zavolat při dotazu na danou adresu.

Při zavolání metody *controlleru* se z dotazu získají potřebná data, případně se provede jejich validace a zavolá se příslušná metoda na aplikační vrstvě, která dotaz dále zpracuje.

Data, která jsou vrácena metodou servisu zpět, jsou namapována na společně definovaný typ pro komunikaci mezi serverem a klienty, a následně odeslána jako odpověď. Pokud během zpracovávání požadavku v servisu došlo k chybě, uživateli je navrácen obsah chyby spolu s příznakem, že dotaz neproběhl úspěšně.

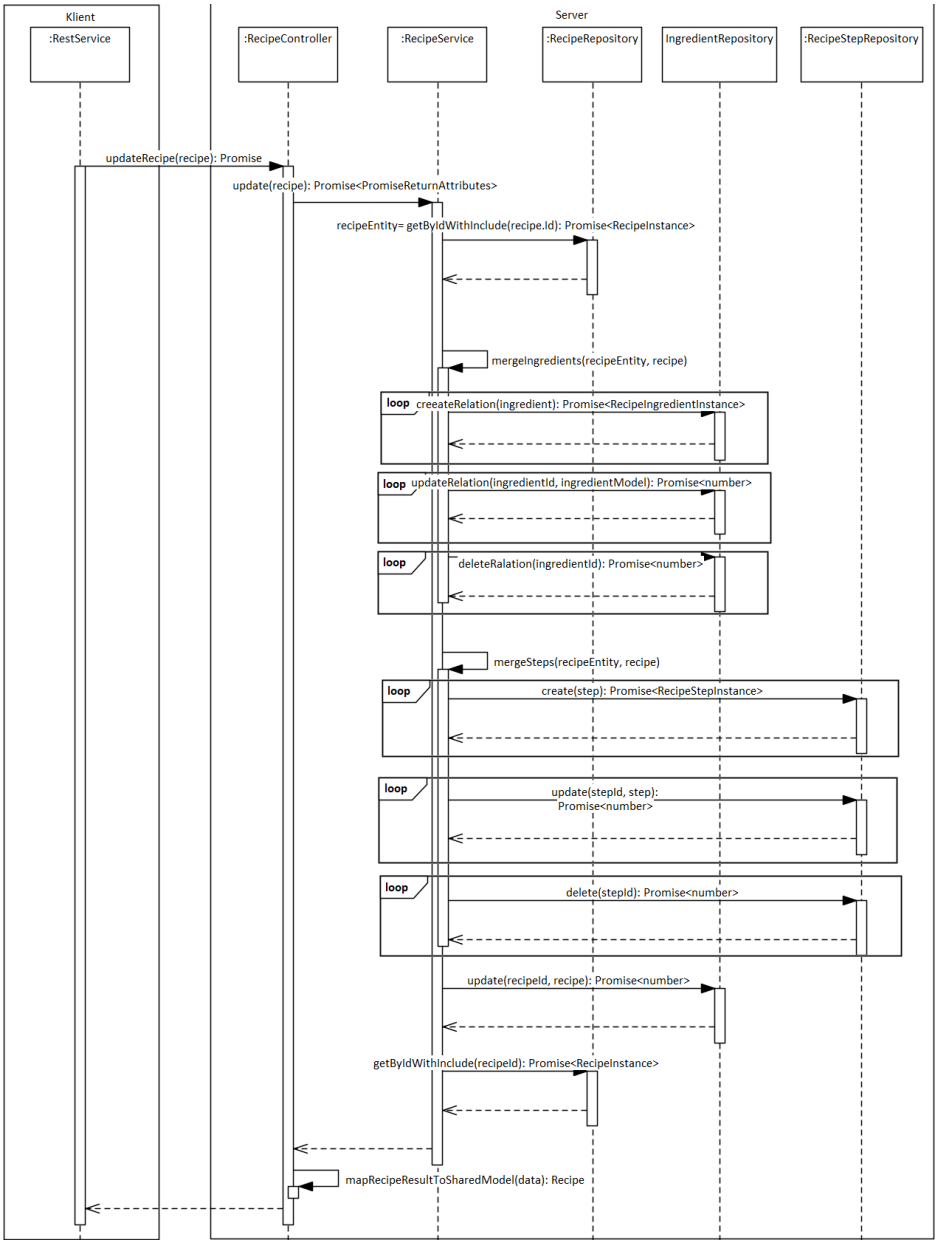
4.1.5 Realizace uložení změny receptu

Proces ukládání změny receptu je zaznamenán sekvenčním diagramem na obrázku 4.2. Klient odešle na server požadavek s daty nového receptu.

Tento požadavek odchytí *RecipeController*, který z požadavku získá data a předá je *RecipeService* pomocí metody `updateRecipe`. Po předání dat čeká na splnění *promise*, kterou metoda `udateRecipe` vrací.

Tato metoda nejdříve vyhledá recept, který má upravit, se všemi navázanými daty v databázi. Poté provede porovnání existujících kroků přípravy s těmi, které obsahuje model odeslaný klientem. Porovnání provede pomocí čísla jednotlivých kroků, které určuje jejich pořadí. Ty, které se shodují aktualizuje, ty, které jsou navíc vytvoří a zbytek vymaže. To samé se provede i s ingrediencemi, kde se místo čísla kroku použije identifikátor vazby. Nakonec se aktualizuje samotná entita receptu a z databáze se získá aktuální obraz receptu, který metoda vrátí do controlleru.

Controller, který čeká na odpověď servisu převezme recepty a převede ho na společný model a odešle jako odpověď klientovi.



Obrázek 4.2: Sekvenční diagram změny receptu

4.2 Klient

Na rozdíl od serverové aplikace se klient spouští v prohlížeči, kde je nativní podpora javascriptu, ve kterém je klientská část napsána.

V projektu se vytvořil indexový html soubor, kam se vložily reference na styly, meta údaje a jiné informace potřebné na stránce. Poté se v tomto

souboru vytvořil HTML *div* element, kterému se přes id přiřadil libovolný identifikátor. Příklad je znázorněn na obrázku 4.3. Tento identifikátor slouží pouze pro to, aby ho šlo na stránce nalézt a mohl se do něj vložit obsah naší aplikace. Dále se vytvořila komponenta, která slouží pro zapouzdření aplikace, a pomocí metody z knihovny React-DOM, která slouží pro propojení React aplikace se stránkou, se propojil document s naší aplikací. A to tak, že metoda přijímá dva parametry, jeden parametr odpovídá elementu dokumentu, kam se má obsah vložit, a druhý udává komponentu, co má do elementu vložit.

```
<div id="root"></div>
```

Obrázek 4.3: Definice kořenového *div* elementu

Jak lze vidět na obrázku 4.4, v našem případě je App, definovaná ve vlasním souboru, vrcholovou komponentou a identifikátor našeho *divu* v dokumentu je *root*.

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import App from './components/App';

ReactDOM.render(
        <App />,
        document.getElementById("root")
);
```

Obrázek 4.4: Ukázka propojení dokumentu s React aplikací

Většiny funkcí lze dosáhnout samotnou prací s jednotlivými komponentmi, které jsme vytvořili. *React* díky své velké popularitě se dá v repozitáři najít mnoho komponent vytvořených komunitou a jejich použitím získat velké množství funkcí za relativně krátkou dobu.

Mezi jednu takovou sadu komponent patří i framework *Semantic-UI* [43], který se použil pro řešení vzhledu uživatelského rozhraní. Pro nereactové aplikace nabízí *Semantic-UI* knihovnu se styly a sadu javascriptových knihoven pro různé funkcionality jako například identifikátor progresu, podpora tabů nebo hodnocení. *Semantic-UI* [43] obsahuje podrobnou dokumentaci pro oba typy zmíněných projektů, včetně ukázek, možnosti si kód vyzkoušet a popisu všech typů atributů na jednotlivých komponentách.

4.2.1 Lokalizace

Pro lokalizování textů na straně klienta se použila knihovna *react-redux-i18n* [44]. Knihovna funguje na bázi, že si zaregistruje posluchače akcí přes „redux reducer“ a následně přes něj uloží do úložiště odpovídající znakovou sadu, ze které má brát překlady. Jak lze vidět v 4.5, tak nejdříve přes metodu

`loadTranslations` načteme všechny překlady do paměti (translationsObject obsahuje importované všechny podporované překladové sady). Následně se zavolá `setLocale`, která nastaví aktuální platnou jazykovou sadu. Poněvadž klient podporuje pouze anglickou verzi, tudíž nejsou překlady pro jiné jazyky, byl jazyk aplikace nastaven na angličtinu defaultně.

```
import { loadTranslations, setLocale } from 'react-redux-i18n';
import translationsObject from './../i18n';

store.dispatch(loadTranslations(translationsObject));
store.dispatch(setLocale('en'));
```

Obrázek 4.5: Ukázka nastavení překladů a jazyka

Následné použití v komponentách je takové, že se načetla metoda *i18n* z knihovny *react-redux-i18n* a přes tuto metodu se specifikoval text, který na daném místě měl být.

Pro přehlednost umístění překladu byla definována hierarchie překladu, která odpovídá struktuře projektu. Jak je například vidět na 4.6, hledá se text pro soubor „menuUser“ s klíčovým slovem „LogOut“. Tato struktura odpovídá struktuře objektu starajícího se o překlad, jak lze vidět na 4.7.

```
I18n.t("menuUser.LogOut")
```

Obrázek 4.6: Ukázka použití překladu

```
menuUser: {
        LogOut: "Log Out"
},
```

Obrázek 4.7: Ukázka definice překladu

4.3 Mobilní aplikace

Vyvíjet aplikace na telefonní zařízení trochu komplikuje samotné zkoušení a testování aplikace, protože je potřeba mít prostředí odpovídající telefonnímu zařízení. Toho lze dosáhnout buď propojením fyzického telefonu s vývojovým prostředím, které posílá build aplikace k instalaci, a nebo emulováním tohoto prostředí. Byla zvolena možnost emulace z důvodu lepšího debugování a možnosti vyzkoušení aplikace na více zařízeních.

Projekt obsahuje soubor „main.dart“, ve kterém je definovaná počáteční komponenta 4.9 a její zavolání při spuštění aplikace 4.8. Tato komponenta specifikuje lokalizaci aplikace, podporované jazykové sady, název aplikace a první komponentu, neboli komponentu nejnižší úrovně na stacku. Přecházení mezi

obrazovkami aplikace funguje způsobem, že se na sebe vrství tak, jak se postupně otevírají, a tlačítkem zpět se odebere obrazovka, která je na stacku nejvýše. Tento postup odpovídá metodě LIFO [45].

```
void main() => runApp(new StateWidget(child: new MyApp()));
```

Obrázek 4.8: Ukázka zavolání první komponenty po spuštění aplikace

```
class MyApp extends StatelessWidget {
        @override
        Widget build(BuildContext context) {
                return MaterialApp(
                localizationsDelegates: [
                        AppLocalizationsDelegate(),
                        GlobalMaterialLocalizations.delegate,
                        GlobalWidgetsLocalizations.delegate
                ],
                supportedLocales: [Locale("en")],
                debugShowCheckedModeBanner: false,
                title: 'Recipe book',
                theme: buildTheme(),
                home: LoginScreen(),
                );
        }
}
```

Obrázek 4.9: Ukázka vrcholové komponenty

4.3.1 Lokalizace

Lokalizace je realizována pomocí knihovny *l10n* [46]. Vytvořil se soubor, který si tuto knihovnu importoval, v souboru se nastavila sada podporovaných jazyků. Z importované knihovny se použila metoda „message“, ukázka kódu 4.10, která přijímá dva parametry. První parametr udává výchozí text, pokud se nenajde překlad, druhý parametr udává klíč překladu. Pomocí zmíněné metody se nadefinovaly všechny texty, které se měly lokalizovat.

```
String get proceedToApp => Intl.message(
        "Proceed to Recipe book",
        name: "proceedToApp"
);
```

Obrázek 4.10: Ukázka definování překladu

Pomocí příkazu 4.11 se vytvořila šablona lokalizačního souboru, který stačilo rozkopírovat pro jednotlivé překlady a přeložit. Z důvodu podpory pouze anglické verze aplikace, se vytvořila pouze jedna kopie a to pro anglickou verzi, která již obsahovala výchozí překlady předvyplněné. Nad touto vytvořenou kopií se spustil příkaz 4.12, který na základě vstupního překladového souboru vygeneroval soubor, který kopii s překlady importoval a umožnil k ní přístup.

```
flutter pub pub run intl_translation:extract_to_arb
 --output-dir=lib/i18n lib/i18n/app_localizations.dart
```

Obrázek 4.11: Ukázka vygenerování překladové šablony

```
flutter pub pub run intl_translation:generate_from_arb
         --output-dir=lib/i18n \
         --no-use-deferred-loading lib/i18n/app_localizations.dart
          lib/i18n/intl_en.arb
```

Obrázek 4.12: Ukázka vygenerování souboru zpřístupňující daný překlad

4.3.2 Vyhledávání receptů

Jednotlivé vyhledávání receptů i jejich detailů probíhá přímo z komponenty přes cachovaný repozitář. Tento repozitář si ukládá vyhledané detaily receptů v paměti pro menší zátěž sítě a rychlejší zobrazení receptů, které uživatel již zobrazil. Cachování výsledků pro jednotlivé kombinace filtrů se neimplementovalo pro složitost, ale připravila se podpora pro budoucí případnou implementaci. Protože bylo potřeba uložit a mít přístup ke stavu přihlášeného uživatele z více míst, vytvořilo se centrální stavové úložiště, které se při zapnutí aplikace inicializuje a v průběhu aplikace je přístupné. Toto úložiště funguje podobně jako již zmíněné centrální úložiště na webovém klientovi. Pomocí kontextu, přesněji příkazem 4.13, lze toto úložiště získat v libovolné komponentě a tak mít přístup k datům uživatele.

```
var appState = StateWidget.of(context).state;
```

Obrázek 4.13: Ukázka přístupu k úložišti aplikace

Kapitola 5

Testování

5.1 Server

Protože serverová část obsahuje nejvíce logiky a pokrývá všechny požadavky, je kladen největší důraz na její otestování.

Testují se všechny repozitáře a servisy kromě těch, které se starají o externí recepty. Repozitáře jsou testovány pomocí mockovaných unit testů, zatímco servisy jsou pokryty integračními testy, které zkoumají funkčnost a komunikaci s databází.

Testy se nachází ve složce „test“ v kořenovém adresáři. V této složce se nachází podsložky pro testy a definice mockovaných modelů a repozitářů. Každý soubor s testy je pojmenován podle souboru, který testuje, a obsahuje všechny otestované metody daného testovaného souboru.

Na otestování serverové části se použily následující knihovny: *ts-mocha* [47], *chai* [48] a *Sequelize-mock* [49].

Testy běží nad předpřipravenou sadou dat, jednotlivé testy si případně data vytváří a následně zase vymazávají. Výsledek testu je zobrazen v konzoli shrnutím počtu úspěšných a neúspěšných testů. Pokud nějaký test selhal, je pro něj vypsána chybová hláška buď s neočekávanou chybou, nebo s informací, na kterém assertu došlo k chybě.

5.1.1 TS-Mocha

Jedná se o knihovnu, jež obaluje testy napsané v typescriptu a spouští testování pomocí knihovny *Mocha*. Samotné integrování typescriptu a knihovny *Mocha* je komplikované a tato knihovna umožňuje jejich spolupráci bez nutnosti úpravy typescript kódu. [47]

Testy jsou spouštěné v prostředí *ts-node*, které umožňuje importování typescript souborů.

Testování se spouští pomocí skriptu 5.1, který funguje tak, že projde rekurzivně všechny složky ve složce „test“ a spustí všechny testy, které najde

v souborech s příponou „ts“.

```
ts-mocha test/**/*.ts
```

Obrázek 5.1: Ukázka spuštění testů

Pomocí metody „describe“ lze sady testů uskupit do logických celků a následně v každém z celků nadefinovat testy, které se této části týkají. Na ukázce testu 5.2 lze vidět, že testy se týkají „Comment repository“, přesněji její metody „create“. Prvek „it“ popisuje, co by měl daný test udělat. Test vytvoří nový komentář k existujícímu receptu a poté kontroluje, zda-li data vytvořeného komentáře souhlasí s těmi, co se měly uložit.

```
describe("Comment repository", function() {
        context("#create", function() {
        it("should create new comment", async function() {
                var comment = "new comment";
                var newComment = await commentRepository
                        .create(
                                userData[1].id,
                                recipeData[0].id,
                                comment
                        );
                expect(newComment.text)
                        .to.be.equal(comment);
                expect(newComment.id)
                        .to.not.be.null;
                expect(newComment.recipeId)
                        .to.be.equal(recipeData[0].id);
                expect(newComment.userId)
                        .to.be.equal(userData[1].id);
        });
        });
})
```

Obrázek 5.2: Ukázka testu

5.1.2 Chai

Jedná se o assertovací knihovnu pro *nodejs* a prohlížeč. Lze ji použít při jakémkoliv zvoleném testovacím frameworku. Poskytuje lehce čitelný styl psaní testů. V ukázce testu 5.2 se jedná o příkaz „expect“. Tento příkaz přijímá parametr a dále ho testuje pomocí řetězení jednotlivých assertů.

5.1.3 Sequelize-mock

Sequelize-mock je mockovací knihovna, která simuluje funkci *Sequelize* ORM.

Byla potřeba pro každý model nadefinovat vlastnosti, včetně toho, co má vracet při daných dotazech. Chování při dotazování lze specifikovat využitím property „queryInterface“ a její metody „useHandler“ 5.3. Tato metoda poskytuje parametr pro callback, který dostane do parametru aktuální *query* příkaz, jako například „findAll“, „create“ nebo „update“, a v „*queryOptions*“ obsahuje případné parametry.

```
Model.$queryInterface.$useHandler(
        function(query, queryOptions, done) {
        }
)
```

Obrázek 5.3: Ukázka specifikace chování Sequelize-mock modelu

Po zmíněném nastavení chování jednotlivých modelů se musely upravit jednotlivé repozitáře, právě kvůli používání těchto modelů. Vznikly tak tedy mockované repozitáře, které při testování nahrazují reálné repozitáře, které komunikují s databází. Tím se simuluje samotná práce nad databází.

5.2 Klient

5.2.1 Unit testy

Pro napsání unit testů se použila knihovna *Jest* [50]. Struktura testu je vidět na ukázce 5.4. Lze si všimnout, že syntaxe je velmi podobná knihovně *TS-Mocha*, nadefinuje se kontext, který zapodřuje více testů, příkazem *describe*. V rámci tohoto kontextu se vypíší testy, které s ním souvisí. Jednotlivé testy se definují pomocí příkazu *test*. Příkaz na porovnání očekávaného výsledku a aktuálního se používá příkaz *expect*, který se nachazí také v knihovně *Jest*.

Na ukázce 5.4 lze tedy vidět zadefinování kontextu, který se nazývá „External recipe reducer valid state“, jde tedy o sadů testů, které kontrolují funkčnost akce týkající se získání externího receptu. Konkrétní test v tomto případě dostane jako data svůj aktuální stav a případná data a kontroluje, zda *reducer* vrací správný stav.

5.2.2 Testování komponent

Pro kontrolu komponent se může použít například knihovna *react-test-renderer* [51] a nebo knihovna *Enzyme* [52].

```
describe("External recipe reducer returns valid state", () => {
test('Should set recipe to store', () => {
        let state = { recipe: null }
        state = externalReducer(
                state,
                {
                        type: ACTION_TYPES
                                        .ExternalRecipeActions
                                        .DisplayExternalRecipe,
                        payload: { data }
                }
        )

        expect(state.recipe.name).toEqual(data.name);
        expect(state.recipe.description).toEqual(data.description);
        expect(state.recipe.difficulty).toEqual(data.difficulty);
        expect(state.recipe.prepTime).toEqual(data.prepTime);
})})
```

Obrázek 5.4: Ukázka unit testu pomocí knihovny *Jest*

5.2.2.1 Snapshot testy

Pro získání snapshotu komponent se může použít například knihovna *react-test-renderer* [51]. Pomocí této knihovny se nahraje struktura komponenty do paměti a následně uloží ve formě snapshotu do souboru. Testy pak probíhají tak, že se porovná nově vygenerovaný snapshot s tím, který je uložený. Při spuštění testů se tak porovnává, zda-li nedošlo v komponentě ke změně, knihovna případně ohlásí, že snapshoty nesouhlasí.

```
it('Empty login form renders correctly', () => {
        const callBack = (username: string, password: string) => {
                return new Promise<any>(resolve => resolve());
        }
        const tree = renderer.create(
                        <LoginForm
                                fixed={true}
                                LogIn={callBack}
                        />
                ).toJSON()
        expect(tree).toMatchSnapshot()
})
```

Obrázek 5.5: Ukázka snapshot testu v Reactu

5.2.2.2 Testy vykreslení komponent

Další použitou knihovnou je *Enzyme* [52]. Jedná se o javascript knihovnu, která se používá pro kontrolu výstupu, průchod komponentou nebo například manipulaci s komponentou samotnou. Knihovna si pomocí příkazu `shallow` nebo `mount` nahraje strukturu do paměti a umožní nad ní vykonávat zmíněné operace.

Rozdíl mezi `shallow` a `mount` je ten, že první varianta nahrává pouze danou komponentu, kterou příjme v parametru, a nezanořuje se do dalších komponent v této struktuře použitých.

Ukázka testování komponenty, která se stará o vykreslení detailu receptu, je znázorněna na obrázku 5.6. V této ukázce lze vidět, že aby komponenta mohla být otestována, musela být zanořena do komponenty *Provider*.

Důvod je takový, že komponenta *RecipeDetail* využívá úložiště aplikace, které poskytuje právě komponenta *Provider*. Recept, který se má zobrazit, se právě z tohoto místa získává, proto se data do úložiště musela vložit před tím, než se vložilo do *Provider*. Toto úložiště je mockované pomocí knihovny *redux-mock-store* [53], která poskytuje funkci `buildStore`, která příjímá v parametru objekt, který má reprezentovat úložiště. Výsledek této funkce se předá *Providerovi*, jak už je vidět na ukázce.

Jakmile je komponenta nahrána v paměti, kontroluje se, zda-li obsahuje vyrenderovanou komponentu, která odpovídá té, co dostane v parametru. Kontrola správného vyrenderování se dá provést například i pomocí metody `find`, která najde element podle selektoru, a pak kontrolovat její obsah. Problém v tomto případě je ten, že pro vykreslování se používá zmíněný framework *SemanticUI*, který může v budoucnu změnit popis svých tříd a identifikátorů, takže při hledání přes selektor by se mohlo stát, že nebude v budoucnu fungovat i při správném vykreslení komponent.

5.2.3 Porovnání doby běhu testů

Při běhu všech skupin testů, kde v každé skupině je jeden test, nejdéle trvá snapshot test, který trvá téměř dvě třetiny celkového času. Nejrychleji proběhl unit test, který testuje fungování *reduceru*.

5.2.4 Porovnání obtížnosti nastavení testů

Oproti serverovým testům se testy na klientovi nastavovaly značně hůř. Mezitím, co serverové testy pouze simulují běh aplikace, tak klientská část si musí komponenty vyrenderovat do paměti a obalit je ve wrapperu, aby s nimi mohla pracovat.

Na zprovoznění unit testů v klientské aplikaci není potřeba žádné speciální nastavení a nastavení pro snapshoty a testy komponent se shoduje, protože je oboje testováno stejnou knihovnou.

```
it("renders whole", () => {
        const result = enzyme.mount(
                <Provider store={store}>
                        <RecipeDetail />
                </Provider>
        );

        expect(
                result.contains(
                        <Header as="h1">
                                {recipe.name}
                        </Header>
                )
        ).toEqual(true);
});
```

Obrázek 5.6: Ukázka testování pomoci *enzyme*

Aby se komponenty napsané v typescriptu vyrenderovaly správně, musí se nastavit konfigurační soubor knihovny *Jest*, protože ta nativně hledá pouze javascriptové soubory. Nastavení knihovny se dá vepsat přímo do souboru konfigurace aplikace a nebo do vlastního konfiguračního souboru knihovny. Na obrázku 5.7 je ukázka nastavení. Klíč „testPathIgnorePatterns“ definuje složky, ve kterých se nemají hledat testy a druhý klíč specifikuje v jak pojmenovaných souborech se testy mají hledat.

```
"testRegex": "(/__tests__/.*|(\\.|/)(test|spec))\\.(ts|tsx)\$",
"testPathIgnorePatterns": [
        "/lib/",
        "/node_modules/"
]
```

Obrázek 5.7: Ukázka nastavení knihovny Jest

5.3 Mobilní aplikace

Flutter nativně obsahuje nástroje pro unit testování, testování widgetů i integračních testů.

V novém souboru se vytvořila „main“ metoda, do které se přidávají buď metody „test“ 5.9, pro unit testy, nebo „testWidgets“ 5.8, pro testy widgetů.

Testování widgetu 5.8 funguje tak, že přes tester se vytvoří komponenta, kterou chceme testovat. Tester nám také umožňuje interagovat s widgety a testovat prostředí. Pomocí třídy „Finder“ se dají vyhledávat komponenty

ve stromové struktuře a vracet komponenty, které splňují vyhledávací pravidlo. Widgety se dají vyhledat pomocí jejich typu, definovaného klíče, ikonou, co komponenta má, a mnoha dalšími způsoby.

```
testWidgets('Popis testu', (WidgetTester tester) async {
        await tester.pumpWidget(new MyWidget());
        await tester.tap(find.text('Save'));
        expect(find.text('Success'), findsOneWidget);
});
```

Obrázek 5.8: Ukázka testování widgetu

```
test('Parse recipe from map', () {
        var result = Recipe.fromMap(json.decode(jsonRecipe));

        expect(result.name, data.name);
        expect(result.description, data.description);
        expect(result.difficulty, data.difficulty);
        expect(result.imagePreviewUrl, data.imagePreviewUrl);
        expect(result.servings, data.servings);
        expect(result.prepTime, data.prepTime);
        expect(result.ingredients.length, data.ingredients.length);
        expect(result.recipeSteps.length, data.recipeSteps.length);
});
```

Obrázek 5.9: Ukázka unit testu ve *Flutter*

Jak v unit testech, tak i ve widget testech se používá příkaz „expect“ pro kontrolu hodnot. Pro widget testy jsou přidány ještě konstanty jako například „findsOneWidget“ nebo „findsNothing“, které pomáhají testovat počet nalezených výsledků přes „Finder“.

Závěr

Cílem práce bylo analyzovat, navrhnout a implementovat aplikaci na správu receptů a jejich vyhledávání.

V analytické části byly popsány uživatelské požadavky. Poté se vyhledala existující řešení receptářů a ze seznamu výsledků se vybralo několik konkrétních receptářů. Každý tento receptář byl následně podroben analýze podle konkrétních bodů. Na konci této kapitoly se provedlo prozkoumání jednotlivých uživatelských případů užití a na konci této sekce se přidala tabulka pokrytí požadavků, kterou následovala analýza technologií.

Na základě analýzy se provedl návrh, který se zaměřil na popis architektury a jednotlivých částí aplikace. Databázovým diagramem se popsala struktura databáze a jednotlivých tabulek. Sekci zakončil diagram nasazení, popisující rozdělení jednotlivých částí aplikace a jejich komunikaci.

Následovala implementace, která využila informací z analýzy i návrhu. Popsaly se zde jednotlivé části aplikace a metody použité během implementace. Pro zhotovení implementace byla vytvořena webová i mobilní aplikace, které splňovaly všechny požadavky z analytické části.

Nakonec byla celá aplikace otestována pomocí automatizovaných testů, kterým se věnuje poslední kapitola.

Aplikaci lze rozšířit v mnoha směrech. Například lze přidat více zdrojů pro získávání receptů, přidat funkci na kopírování receptů, připisování poznámek přihlášených uživatelů nebo přidat možnost organizování receptů do skupin.

Všechny cíle, které měla tato bakalářská práce splňovat, byly splněny.

Bibliografie

1. ENGE, Eric. *Mobile vs Desktop Traffic in 2019* [online] [cit. 2019-04-18]. Dostupné z: `https://www.stonetemple.com/mobile-vs-desktop-usage-study/`.
2. SA, Gemius. *Recepty.cz* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/`.
3. SA, Gemius. *Pokročilé vyhledávání receptů* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/vyhledavani/pokrocile`.
4. SA, Gemius. *Vyhledávání podle ingrediencí* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/vyhledavani/ingredience`.
5. SA, Gemius. *Vložení nového receptu* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/recept/vlozit`.
6. SA, Gemius. *Vlastní kuchařka na webu* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/moje-kucharka`.
7. SA, Gemius. *Nákupní seznam* [online] [cit. 2019-03-26]. Dostupné z: `https://www.recepty.cz/nakupni-seznam`.
8. VARENI.CZ. *Vareni.cz* [online] [cit. 2019-03-26]. Dostupné z: `https://www.vareni.cz/`.
9. VARENI.CZ. *Katalog receptů* [online] [cit. 2019-03-26]. Dostupné z: `https://recepty.vareni.cz/vyhledavani/?fotka=on&order=relevance`.
10. S.R.O., Toprecepty.cz. *toprecepty.cz* [online] [cit. 2019-03-26]. Dostupné z: `https://www.toprecepty.cz`.
11. S.R.O., Toprecepty.cz. *Přidat recept* [online] [cit. 2019-03-26]. Dostupné z: `https://www.toprecepty.cz/pridat_recept.php`.
12. S.R.O., Toprecepty.cz. *Diskuze* [online] [cit. 2019-03-26]. Dostupné z: `https://www.toprecepty.cz/diskuze.php`.
13. ALLRECIPES.COM. *Be a Better Cook* [online] [cit. 2019-03-27]. Dostupné z: `https://cookingschool.allrecipes.com/p/overview/`.

14. ALLRECIPES.COM. *Allrecipes* [online] [cit. 2019-03-27]. Dostupné z: www.allrecipes.com.

15. ALLRECIPES.COM. *Site Map* [online] [cit. 2019-03-27]. Dostupné z: http://dish.allrecipes.com/faq-sitemap/.

16. ALLRECIPES.COM. *Submit your Recipes!* [online] [cit. 2019-03-27]. Dostupné z: http://dish.allrecipes.com/customer-service/submit-your-recipes/.

17. ALLRECIPES.COM, Inc. *Allrecipes Dinner Spinner* [online] [cit. 2019-03-29]. Dostupné z: https://play.google.com/store/apps/details?id=com.allrecipes.spinner.free.

18. ALLRECIPES.COM, Inc. *Allrecipes Dinner Spinner* [online] [cit. 2019-03-27]. Dostupné z: https://play.google.com/store/apps/details?id=com.allrecipes.spinner.free&hl=en.

19. NAST, Condé. *epicurious* [online] [cit. 2019-03-28]. Dostupné z: https://www.epicurious.com.

20. NAST, Condé. *ADD YOUR OWN RECIPE* [online] [cit. 2019-03-28]. Dostupné z: https://www.epicurious.com/user/addrecipe.

21. FOUNDATION, Node.js. *About The Node.js Foundation* [online] [cit. 2019-04-10]. Dostupné z: https://foundation.nodejs.org/about.

22. FOUNDATION, Node.js. *NodeJs* [online] [cit. 2019-04-10]. Dostupné z: https://nodejs.org.

23. KADISHAY, Yotam. *JavaScript V8 Engine Explained* [online] [cit. 2019-04-10]. Dostupné z: https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef.

24. INC., Facebook. *React* [online] [cit. 2019-04-10]. Dostupné z: https://reactjs.org/.

25. GOOGLE. *About* [online] [cit. 2019-04-10]. Dostupné z: https://flutter.dev/.

26. GOOGLE. *Hot reload* [online] [cit. 2019-04-10]. Dostupné z: https://flutter.dev/docs/development/tools/hot-reload.

27. GOOGLE. *Widget catalog* [online] [cit. 2019-04-10]. Dostupné z: https://flutter.dev/docs/development/ui/widgets.

28. LELER, Wim. *Why Flutter Uses Dart* [online] [cit. 2019-04-10]. Dostupné z: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf.

29. GOOGLE. *Who Uses Dart* [online] [cit. 2019-04-19]. Dostupné z: https://www.dartlang.org/community/who-uses-dart.

30. HEROKU. *Heroku* [online] [cit. 2019-04-11]. Dostupné z: https://www.heroku.com.

31. HEROKU. *Simple, flexible pricing* [online] [cit. 2019-04-11]. Dostupné z: https://www.heroku.com/pricing.

32. AMAZON. *AWS Free Tier* [online] [cit. 2019-04-11]. Dostupné z: https://aws.amazon.com/free/?nc2=h_ql_pr.

33. GOOGLE. *Pricing Plans* [online] [cit. 2019-04-11]. Dostupné z: https://firebase.google.com/pricing/.

34. CODECADEMY. *What is CRUD?* [online] [cit. 2019-04-19]. Dostupné z: https://www.codecademy.com/articles/what-is-crud.

35. YAKUSHIN, Mikhail. *Entity and DTO: What's the difference?* [online] [cit. 2019-04-19]. Dostupné z: https://www.driver733.com/2018/10/08/entity-and-dto.html.

36. REDUX DOKUMENTACE, Dan Abramov a autoři. *Redux* [online] [cit. 2019-04-20]. Dostupné z: https://redux.js.org/.

37. REDUX DOKUMENTACE, Dan Abramov a autoři. *React-Redux* [online] [cit. 2019-04-20]. Dostupné z: https://react-redux.js.org/.

38. MICROSOFT. *Typescript* [online] [cit. 2019-04-20]. Dostupné z: https://www.typescriptlang.org/.

39. *axios* [knihovna]. 2019. Verze 0.18.0. Dostupné také z: https://www.npmjs.com/package/axios.

40. CFWHEELS. *Soft Delete* [online] [cit. 2019-04-20]. Dostupné z: https://guides.cfwheels.org/docs/soft-delete.

41. DEPOLD, Sascha. *Sequelize* [online] [cit. 2019-04-11]. Dostupné z: http://docs.sequelizejs.com.

42. FOUNDATION, Node.js. *Fast, unopinionated, minimalist web framework for Node.js* [online] [cit. 2019-04-20]. Dostupné z: https://expressjs.com/.

43. ORGANIZATION, Semantic. *User Interface is the language of the web* [online] [cit. 2019-04-20]. Dostupné z: https://semantic-ui.com/.

44. AVOTINS, Artis. *react-redux-i18n* [knihovna]. 2019. Verze 1.9.3. Dostupné také z: https://www.npmjs.com/package/react-redux-i18n.

45. HOPE, Computer. *LIFO* [online] [cit. 2019-04-25]. Dostupné z: https://www.computerhope.com/jargon/l/lifo.htm.

46. MITTERER, Mike. *l10n* [knihovna]. 2019. Verze 0.2.4. Dostupné také z: https://www.npmjs.com/package/l10n.

47. TS-MOCHA. *Piotrek Witek* [knihovna]. 2019. Verze 6.0.0. Dostupné také z: https://www.npmjs.com/package/ts-mocha.

48. *Chai Assertion Library* [knihovna]. 2019. Verze 4.2.0. Dostupné také z: https://www.npmjs.com/package/chai.

49. BLINKUX. *Sequelize Mock* [knihovna]. 2019. Verze 0.10.2. Dostupné také z: `https://www.npmjs.com/package/sequelize-mock`.

50. INC., Facebook. *Jest* [knihovna]. 2019. Verze 24.8.0. Dostupné také z: `https://www.npmjs.com/package/jest`.

51. INC., Facebook. *react-test-renderer* [knihovna]. 2019. Verze 16.8.6. Dostupné také z: `https://www.npmjs.com/package/react-test-renderer`.

52. *Enzyme* [knihovna]. 2019. Verze 3.9.0. Dostupné také z: `https://www.npmjs.com/package/enzyme`.

53. *redux-mock-store* [knihovna]. 2019. Verze 1.5.3. Dostupné také z: `https://www.npmjs.com/package/redux-mock-store`.

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

REST Representational State Transfer

SSH Secure Shell

ORM Object Relational Mapping

JSON JavaScript Object Notation

DOM Document Object Model

HTML Hypertext Markup Language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `install.txt` ..... stručný popis, jak spustit server a klienta
- `impl` ..... zdrojové kódy webové a mobilní aplikace
- `img` ..... obrázky použité v písemné práci
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF