Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Distanční výukové materiály pro výuku webových technologií

Diplomová práce

Autor: Bc. Jan Dušek

Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové Duben 2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 27.4.2023

Poděkování:

Děkuji vedoucí diplomové práce Mgr. Daniele Ponce, Ph.D. za metodické vedení práce a za poskytnuté cenné rady.

Anotace

Diplomová práce se zabývá vývojem webové aplikace, jež je v prvé řadě určena jako výukový materiál pro studenty předmětu Technologie pro publikování na Webu a především na vývoj klientské části. Nejprve jsou vysvětleny základní principy a pojmy v tomto odvětví, které jsou nezbytné pro správné porozumění fungování webové aplikace. Následně jsou představeny technologie, jež byly použity pro vývoj a jejich alternativy. Dále je představeno ukázkové zadání projektu, které má za úkol simulovat požadavky na projekt od eventuálního zákazníka. Následuje analýza a možný návrh projektu, přičemž vývoj je rozdělen do pěti částí, které na sobě nemusí být nutně závislé, avšak v každé další části existuje jistý progres části předchozí. V závěru jsou detailněji popsány vybrané úseky kódu, kterým v přecházejících kapitolách nebyla věnována pozornost, avšak jsou podstatné pro správný chod projektu.

Annotation

Title: Distance learning materials for web technologies learning

The diploma thesis focuses on the development of a web application, which is primarily intended as a teaching material for students of the subject Technology for the publishing of the Web and especially for the development of the customer section. Firstly, the basic principles and terms in the respective industry are explained, that are necessary for the correct and common understanding of how the web application works. Secondly, the technologies as well as their alternatives that were used for the development are introduced. Furthermore, a sample project assignment is presented, which has the purpose of simulating project requirements from a potential customer. In addition, the analysis and the project proposal follow, while the development is divided into five parts which do not have to be dependent on each other but in each subsequent part there is a certain progress of the previous part reflected. In the conclusion, the selected sections of the code are described more in the detail as in the previous chapters they are not part of the main focus, however, they are crucial for the proper running of the project.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Vývoj webových aplikací je stále populárnější díky rozvoji technologických nástrojů a velmi dostupnému internetovému připojení. Vzhledem k neustálému růstu velkých technologických společností a díky jejich inovativním řešením se způsob vývoje softwaru neustále mění. Velký rozvoj zaznamenává jazyk JavaScript, ať už v podobě knihoven, frameworků nebo nadstaveb jako je například TypeScript od firmy Microsoft.

Vývoj webových aplikací se nesoustředí pouze na vylepšování vizuální složky, ale také na vylepšování výkonu, rychlosti stránek a bezpečnosti. S neustálým vývojem JavaScriptu souvisí i narůstající popularita jednostránkových aplikací (SPA). Tento přístup umožňuje vytvářet aplikace, které se v prohlížeči načtou jednou a pak se interaktivně mění bez nutnosti znovu načítání celé stránky znovu. SPA aplikace jsou často vyvíjeny pomocí frameworků React, Angular nebo Vue, napsaných právě v jazyce JavaScript. Velká obliba tohoto jazyka mezi vývojáři, je důvodem proč se tato práce zabývá jeho používáním, a protože představuje perspektivní nástroj pro studenty informatiky.

Této práci je kladen cíl představit některé moderní nástroje pro vývoj webových aplikací se zaměřením na frontend. Výsledkem bude funkční aplikace skládající se z frontendu, backendu i databáze, při jejímž vývoji budou demonstrovány jak základní, tak pokročilé postupy při implementaci. K dispozici bude několik samostatných verzí jedné aplikace, které slouží jako edukační materiály pro studenty webových technologií. Hlavní technologie, jež byly zvoleny pro zpracování praktického projektu, jsou JavaScriptové knihovny ReactJS a Redux-Toolkit, CSS preprocesor LESS a v neposlední řadě Python s frameworkem Flask.

2 Cíl práce

Cílem diplomové práce je vytvořit webovou aplikaci za účelem edukace studentů předmětu Technologie pro publikování na Webu. Nejprve je nutné provést seznámení s problematikou, poté jsou definovány požadavky na projekt, které simulují potencionální poptávku. Aplikace bude rozvětvena do několika vývojových částí, které studenta postupně vzdělávají. Student je posléze schopen nabyté znalosti aplikovat v praxi. Diplomová práce má za cíl především představit vývoj klientské části webové aplikace za použití moderních technologií, jedná se o JavaScriptové knihovny ReactJS a Redux-Toolkit. Aplikace bude disponovat i funkčním serverem napsaným v jazyce Python s pomocí frameworku Flask.

3 Vývoj webových aplikací

V současné době se webové aplikace těší značné popularitě, jedním z důvodů, proč tomu tak je, může být fakt, že uživatel služby není nucen ke stažení a následné instalaci softwaru do vlastního zařízení, kterým je typicky telefon nebo stolní počítač, jak tomu je u tradičních aplikací. Dalším plusem je přístupnost, jelikož obecně je webová aplikace dostupná na jakémkoliv zařízení disponující internetovým připojením a prohlížečem. Po tvorbě nových webových aplikací je velká poptávka, a proto i velké firmy vyvíjejí nemalé úsilí pro vytváření nástrojů ulehčující dosažení cíle, jež je vytvořit rychlou, responzivní, bezpečnou a v neposlední řadě uživatelsky přívětivou aplikaci.

Přední technologické firmy mnohdy využívají své zdroje pro vývoj proprietárního a komerčního softwaru, ovšem i firmy jako IBM, Docker nebo Intel, nabízejí některé své produkty jako otevřené (z anglického Open-source Software), tedy volně dostupné a zdarma. Díky tomu, že programátoři vnímají oporu velké firmy, tak si tyto vývojářské nástroje získávají oblibu mimo jiné i v odvětví vývoje webových aplikací. Příklady firem, které poskytnuly svoji technologii veřejnosti, jež se posléze začala používat po celém světe, jsou kupříkladu Meta a její JavaScriptová knihovna React, Microsoft se svým jazykem TypeScript nebo také internetový prohlížeč Google Chrome od firmy Google a mnoho dalších. [25]

Programátor má k dispozici velké množství technologií vhodných pro vývoj. Pokud se webová aplikace rozdělí do tří zjednodušených celků:

1. frontend neboli klientská část, jakožto grafické rozhraní aplikace přístupné uživatelům,
2. backend neboli serverová část, obsahující logiku aplikace, neumožňující zásah běžného uživatele napřímo,
3. a databázi, která slouží k perzistentnímu uložení dat,

pak se nabízejí pro každý výše zmíněný celek jednotky či desítky technologických nástrojů, knihoven, frameworků, a především možností jakým způsobem celek implementovat.

Tato práce poskytuje teoretickou oporu, a zároveň demonstruje její aplikování v praxi díky osvědčeným postupům běžných při vývoji webových aplikací, tyto poznatky jsou promítnuty do praktického projektu, o kterém diplomová práce pojednává. Jako jeden z počátečních kroků vývoje aplikací je výběr technologií. Ideální je situace, kdy zhotovitel zná co nejpřesnější zadání na základě něhož se snaží zvolit ty nejvhodnější nástroje. Z tohoto důvodu a za účelem vytvoření reálné představy o praktickém projektu je další odstavec věnován zjednodušené interpretaci zadání.

Zadavatel požaduje vytvoření moderních webových stránek pro restaurační zařízení. Kromě standardních nároků, jimiž jsou například přehlednost, jednoduchost a bezpečnost, mezi hlavní funkcionality patří možnost tvorby objednávek pouze pro přihlášené uživatele, zároveň je nutné implementovat správu objednávek a registrovaných uživatelů pro zaměstnance restaurace.

Z předcházejícího, i když zjednodušeného zadání je možné postupně začít navrhovat webovou aplikaci. Následující kapitoly popisují, jaké technologie a postupy byly zvoleny včetně s jejich možnými, taktéž frekventovanými alternativami.

3.1 Typy webových aplikací

Webové aplikace se dají rozdělovat dle několika hledisek, nicméně v této kapitole se rozebírají typy podle způsobu renderování obsahu. Renderování je proces převádějící napsaný kód webové stránky do podoby takové, s kterou posléze uživatel může interagovat. Když návštěvník odešle požadavek serveru na získání webové stránky, je mu vrácen HTML kód (z anglického HyperText Markup Language, HTML), který je prohlížečem převzat a převeden do vizuální podoby. Otázkou zůstává, jakým způsobem a kde byl HTML kód vygenerován. Zde existují dva hlavní přístupy, a to buď generování obsahu v serverové nebo klientské části, případně jsou k dispozici i některá modifikovaná řešení. [27]

Pro praktický projekt, kde je vytvářena webová aplikace pro restaurační zařízení byl zvolen přístup renderování obsahu v klientské části, navíc s využitím jednostránkové aplikace (z anglického Single Page Application, SPA) o které je

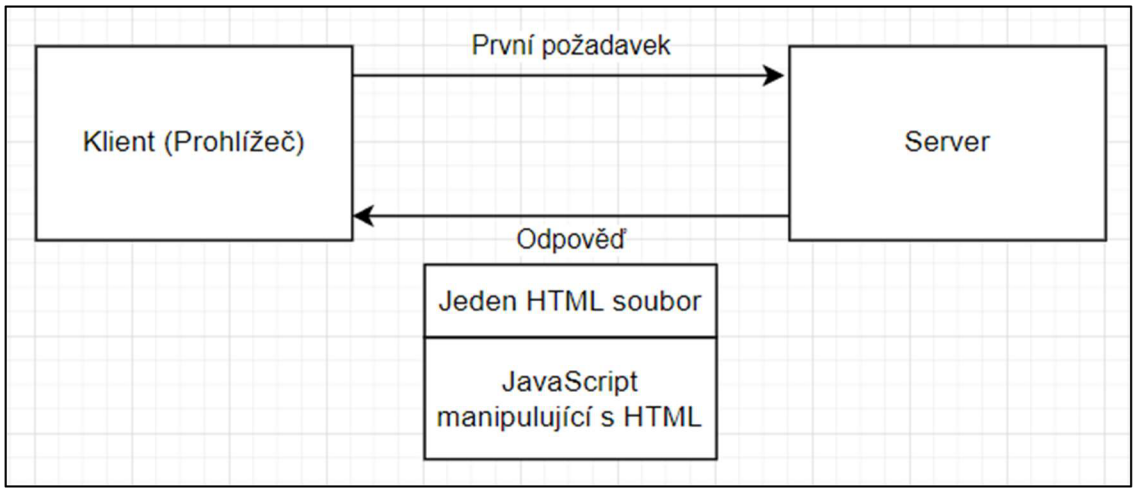
podrobněji psáno později v práci. Projekt si potřebuje ukládat uživatelská data, vkládat jejich nejaktuálnější podobu do stránky a zobrazovat je uživateli vždy, když o ně bude požádáno. Po načtení musí web působit svižně a nenutit uživatele zdlouhavě vyčkávat při načítání dalšího kroku v procesu vytváření objednávky, což by mohlo negativně ovlivnit její dokončení.

3.1.1 Stránky generované u klienta

Generování obsahu na straně klienta (z anglického Client Side Rendering, CSR), pomocí webového prohlížeče, vytváří obsah, který se plně odvíjí od dodaného JavaScriptového kódu. Ten totiž dynamicky vytváří webovou stránku ihned po jeho stažení. [9] Prohlížeči je tak umožněno, aby při vykreslování stránky odvedl co nejvíce práce a zmírnil požadavky na výkon serveru. "Klientem" je aktuální prohlížeč, který uživatel používá k prohlížení stránky. Načítání je časově náročnější a přináší tedy zpoždění, protože provedení JavaScriptového kódu, jenž je u vykreslování na klientské straně obsáhlejší, než tomu tak je u jiných přístupů, trvá nějaký čas. Proto se většinou na tomto druhu stránek běžně objevuje animace načítání pro uvědomění klienta, že se čeká na vytvoření obsahu. Ovšem po zobrazení je vše plně interaktivní, funkční a relativně rychlé. [27]

Jednostránková aplikace

V rámci přístupu popsaném v předešlém odstavci je častokrát zmiňován pojem jednostránková aplikace, jedná se o označení takové webové aplikace, jenž využívá přístup vykreslovaní na straně klienta a zároveň pracuje právě s jedním HTML souborem, jehož obsah je podle potřeby skrýván či překreslován. Jinými slovy místo toho, aby každý požadavek vracel jinou HTML stránku, je webová stránka dynamicky vykreslována přímo v prohlížeči. [5] Na Obr. 1 je vyobrazené schéma, jak SPA funguje.



Obr. 1 Schéma SPA. Zdroj: [35]

Je důležité mít na paměti, že ač je snaha přesunout co nejvíce výpočetních operací do klientské části, tak server je stále potřebný. Kupříkladu není možné, respektive je silně nedoporučené z důvodu porušení základních bezpečnostních pravidel, aby JavaScript komunikoval napřímo s databází, jelikož by byly odhaleny přístupové údaje – prováděný kód v prohlížeči nelze skrýt. Častokrát je nutné zaktualizovat zobrazovaná data, což obnáší vytvořit požadavek na pozadí a s překreslením vyčkat, než přijde odpověď s novými daty ze serveru. Tato akce většinou nenarušuje pozitivní dojem z aplikace, neboť v rámci SPA nedochází k načítání nové stránky a zároveň dotazovaná data na pozadí nejsou většinou objemná jako prvotní požadavek na zobrazení. [35]

Vzhledem k tomu, že dynamické aktualizace celé HTML stránky prostřednictvím JavaScriptu vyžadují velké množství instrukcí, navíc je pravděpodobné, že k tomu bude docházet frekventovaně, obvykle znamená, že je používán některý framework případně knihovna. Možné technologie jsou rozebrány v Kapitole 3.2.

Progresivní webová aplikace

Stejně jako SPA je progresivní webová aplikace (z anglického Progressive Web App, PWA) nejčastěji spojována s přístupem generování u klienta, ale existují PWA aplikace se serverovým renderováním nebo o něco častějším hybridním řešením. Jedná se o typ webové aplikace, ke které lze přistupovat prostřednictvím prohlížeče jako k běžným internetovým stránkám, a navíc přebírá některé vlastnosti nativních aplikací kupříkladu funkčnost v offline režimu nebo lepší uzpůsobení vzhledem ke

klientskému operačnímu systému. PWA není vyrobena s využitím jedné technologie, ale s podporou mnoha nástrojů i několika specifickými návrhovými vzory. Nemusí být při prvním pohledu zřejmé, zda se jedná o PWA nebo ne, aplikace může být považována za PWA v případě, že splňuje určité požadavky, jako jsou offline režim a možnost instalace do zařízení. Výčet typických vlastností pro web identifikovatelný jako progresivní aplikace je následující:

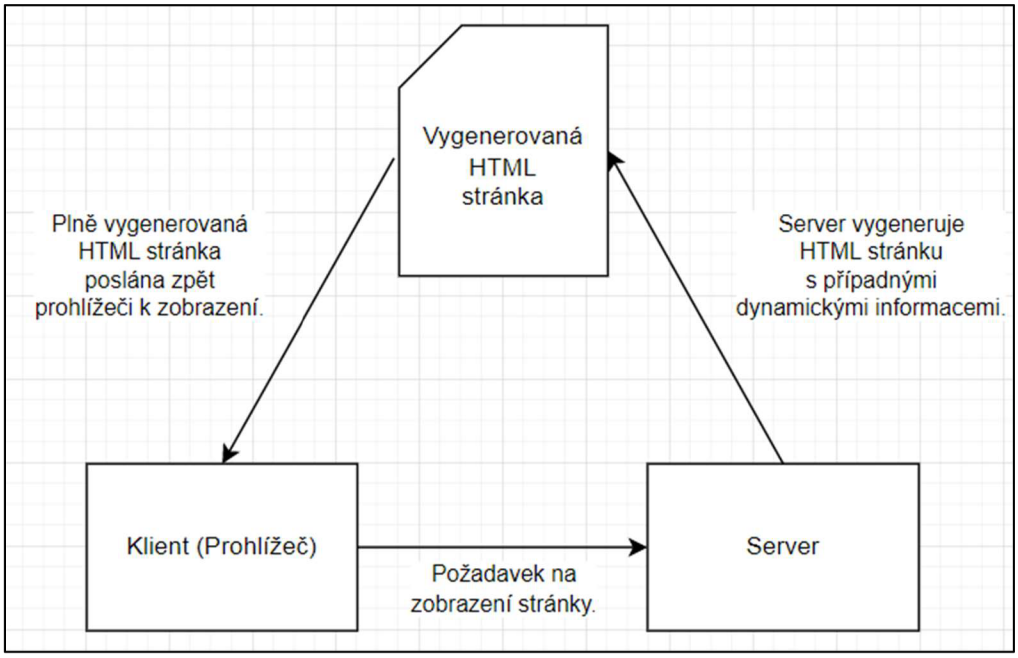
1. vyhledatelná, aby bylo možné stránku najít skrz vyhledávače,
2. instalovatelná, takže je přístupná z domovské stránky zařízení,
3. propojitelná, umožňuje sdílení aplikace pomocí URL odkazu,
4. není plně závislá na internetovém připojení,
5. postupně se rozšiřující, tedy je použitelná se starým prohlížečem, ale plně funkční s nejnovějšími verzemi prohlížeče,
6. a další. [19]

3.1.2 Stránky generované na serveru

Na principu generování obsahu v serverové části (z anglického Server Side Rendering, SSR) fungovaly webové stránky v minulosti. I přesto, že současný fenomén vykreslování v klientské části, v kombinaci s nástroji umožňujícími implementaci SPA, se odvíjí přesně opačným směrem, tak se znovu objevují tendence vrátit co nejvíc režie a práce zpět na server. Pakliže uživatel skrze prohlížeč požádá o webovou stránku, je požadavek převzat serverem a následně poslána odpověď s již kompletně vykreslenou neboli sestavenou HTML stránkou obsahující všechna data. Dynamické informace jsou vkládány serverem rovnou do výsledného HTML. Při opětovné návštěvě je server nucen provádět všechno sestavování znovu, to má za následek, že pomalé internetové připojení, nevhodné umístění serveru či velké množství uživatelů ve stejný čas, může negativně ovlivnit uživatelský dojem z těchto webových stránek. [27]

Velkou výhodou generování v serverové části je absence objemného inicializačního požadavku oproti renderování v klientské části. Protože, čím více funkcionalit a stránek webová aplikace (SPA) nabízí, tím více JavaScriptového kódu je potřeba, což vyústí v delší dobu stahování. Toto počáteční zbrzdění je daň za

zajištění pozdějšího procházení webové aplikace mnohem rychleji. Řadě vývojářům tento kompromis nevyhovuje, díky čemuž je možné se s opačným přístupem, tedy serverovým generováním, v dnešní době setkávat stále častěji. [9] Schéma ukazující průběh renderování webové stránky pomocí serveru je vyobrazeno na následujícím Obr. 2.



Obr. 2 Schéma SSR. Zdroj: [9]

3.1.3 Generované statické stránky

Statická stránka je vykreslena prohlížečem přesně v takovém stavu, v jakém byla vygenerována. Server nevyrábí HTML kód při každém dotazu, nýbrž je kód vytvořen jednou v serverové části, uložen a nezměněn posílán klientovi při požadavku. Obsah není ovlivněn uživatelskými daty. Statické stránky jsou vhodným řešení pro případy kdy se zobrazovaný obsah nikdy nemění, popřípadě velmi zřídka. Princip statických stránek, tedy připravených vygenerovaných dat, je stále používaný, protože před příchodem JavaScriptových frameworků byly všechny stránky statické. Oproti minulosti, v dnešní době, existují lepší, efektivnější způsoby, jak tento typ stránek generovat.

Přístup Generování Statických Stránek (z anglického Static Site Generation, SSG) z velké části zahrnuje automatizaci procesu vytváření webových stránek. JavaScriptové frameworky jako Nuxt.js, Next.js a další, poskytují šablonovací

systémy pro vytváření mnoha stránek najednou pomocí jedné šablony, což programátorovi značně ušetří čas. Na rozdíl od generování v serverové či klientské části je HTML kód sestaven v rámci spuštění aplikace – předtím, než se uživatel pokusí vstoupit. Hlavní nevýhodou je povinnost nechávat generovat jednotlivé stránky pro všechny přístupné URL adresy na webu. [9]

Jak již bylo zmíněno výše, praktický projekt této práce využívá přístup generování u klienta a zároveň implementuje SPA. Protože je zřejmé, že webová aplikace musí pracovat s dynamickými daty, pak se použití generování statických stránek nejeví jako ideální. Možná alternativa, generování na serveru je realizovatelná, ale vzhledem k vlastnostem tohoto přístupu byla dána přednost SPA. Následující Tab. 1 shrnuje a porovnává specifika generování v serverové a klientské části.

Tab. 1 Porovnání SSR a CSR. Zdroj: [37]

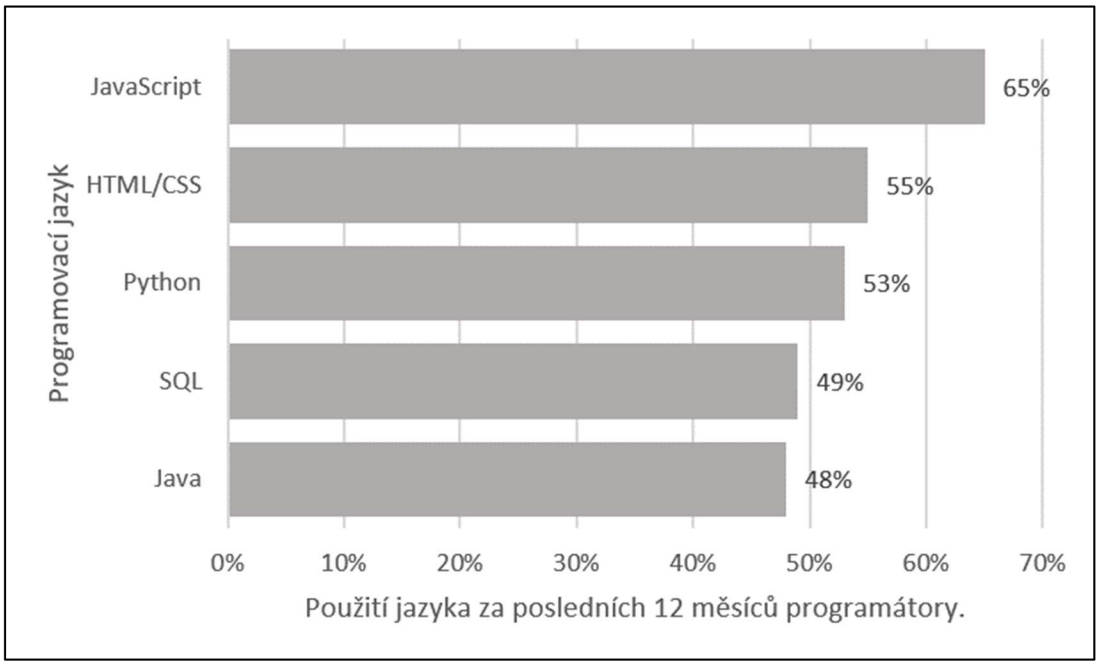
	Generování na serveru	**Generování u klienta**
Účel	Webové stránky poskytující především statický, ale i dynamický obsah.	Ideální pro webové aplikace.
Rychlost	Nutnost načítat celé stránky, z toho plyne konstantní rychlost i při prvotním načtení.	Po pomalejším načtení stránky jsou následující změny v zobrazení téměř instantní.
Závislosti	Povinnost používat JavaScript odpadá.	Nutné používat JavaScript, volitelně s knihovnami třetích stran.
Vyhledávače	Stránka je zcela vytvořena předtím, než je doručena uživateli čili webové vyhledávače jednoduše rozpoznají obsah.	Generování dynamické stránky až v prohlížeči neposkytuje vyhledávačům přímočaré informace.
Interakce	Základní obecné interakce se stránkou.	Díky JavaScriptu a možnosti přizpůsobit obsah konkrétnímu uživateli jsou k dispozici bohaté interakce.

Vytížení serveru	Zde je větší vytížení serveru, především v případech, kdy server musí obsloužit mnoho požadavků na stránku.	Nejsou zde tak velké nároky na výkon, protože stránku vykresluje prohlížeč.

3.2 Technologie pro vývoj webových aplikací

K dispozici je spoustu technologií, které lze zvolit, resp. využít při tvorbě webové aplikace. Jedná se o obor, který je ve vývojářské komunitě nejpopulárnější, až 75 % programátorů je zapojeno do webového vývoje. [15] Je nutné zvolit několik technologií při plánování aplikace, jelikož existují rozdílné programovací jazyky, frameworky, knihovny a databázové systémy s konkrétním určením buď pro frontend, backend či ukládání dat. Adekvátní výběr nástrojů může výrazně ovlivnit výslednou kvalitu a efektivitu aplikace, proto je na místě brát v úvahu specifické požadavky projektu, resp. jeho zadavatele, tzn. bezpečnost, škálovatelnost, rychlost, hlavní funkcionality stránky a další faktory.

Dle průzkumů za rok 2022 jsou nejpopulárnějšími programovacími jazyky JavaScript a Python, následováni Javou. Konkrétně Python je druhý nejpopulárnější programovací jazyk, přičemž stále jeho oblíbenost sílí, jako primární jazyk překonal Javu a snižuje náskok JavaScriptu. Dalšími nejpoužívanějšími jazyky jsou HTML/CSS nebo SQL. [15]



Obr. 3 Nejpopulárnější jazyky za rok 2022. Zdroj: [15]

Vzhledem k předešlým výsledkům průzkumu a za účelem využití moderních neboli populárních řešení během tvorby projektu této práce, byl zvolen pro zhotovení frontendu Javascript s jeho knihovnou ReactJS a pro stylizaci CSS preprocesor Less. Dále bude pro backend využíván jazyk Python včetně mikro webového frameworku Flask. Pro persistentní ukládání dat byl vybrán relační databázový systém SQLite. Díky této sestavě tak bude možné setkat se v rámci tohoto projektu se čtyřmi nejpopulárnějšími jazyky za rok 2022. V následujících podkapitolách jsou právě tyto technologie podrobněji popsány, taktéž jsou uvedeny jejich možné alternativy pro porovnání.

3.2.1 Frontend

Mezi nejdůležitější frontend technologie, které se nacházejí na drtivé většině webů je trojice HTML, CSS a JavaScript. Jejich hlavním cílem je vytvářet vizuální a interaktivní uživatelské rozhraní. Zmíněné technologie procházejí nekončícím vývojem a stále se objevují nové nástroje. Správné použití frontend technologií je klíčové pro vývoj kvalitních webových aplikací, které uživateli nabízejí intuitivní a příjemný dojem z používání stránek.

Less je zpětně kompatibilní rozšíření jazyka CSS, umožňuje vývojářům psát kód kaskádových stylů s pokročilými funkcemi a posléze je přeložit do obyčejného

CSS. [17] Alternativou může být nástroj SASS, který je taktéž CSS preprocesor. Při porovnání je možné si povšimnout, že LESS kód je lehčí na čtení, umí lépe lokalizovat případné chyby, zaměřuje se na odstranění duplicit kódu a zvládá čas kompilace je relativně rychlý. Zatímco SASS vyniká spíše ve výkonu, dokáže definovat vlastní funkce a obecně lze říci, že má větší komunitu. Oba preprocesory využívají specifickou vlastní syntaxi a není možné je využívat zároveň. [26]

Klíčovým nástrojem pro vývoj frontend části pro programátory je nejčastěji JavaScript potažmo jeho frameworky či knihovny. Frameworky poskytují řadu užitečných funkcí a nástrojů, které umožňují psát kód rychleji a efektivněji, díky čemuž snižují náklady na vývoj. V následujících třech podkapitolách se práce zaměřuje na jedny z nejpopulárnějších JavaScriptových frameworků pro frontend vývoj, představuje jejich vlastnosti, přínosy a rozdíly mezi nimi. Vybrány jsou ReactJS, jež je zvolen i jako knihovna pro praktický projekt této práce, vzhledem k jeho flexibilitě a rychlosti vývoje, dále se porovnává AngularJS a na závěr Vue.js.

ReactJS

Oficiální dokumentace uvádí, že knihovna ReactJS umožňuje vytvářet uživatelské rozhraní skládajících se z částí nazývaných komponenty. Je možné vyrobit komponenty jako *Náhled*, tlačítko *To se mi líbí* nebo *Video* a podobně, posléze je možné s nimi pracovat napříč celým webem a získat tak sjednocenou podobu prvků. [30] Tato knihovna tedy slouží k práci s vytvořenými znovupoužitelnými komponentami. ReactJS umožňuje práci s komplexními aplikacemi, které pracují s dynamickými daty, zobrazuje uživateli aktuální stav aplikace bez vyústění k celkovému znovu načtení stránky. Používá se pro implementaci pohledové (V) vrstvy v rámci architektury Model-View-Controller (MVC). [3]

Typické vlastnosti knihovny ReactJS jsou následující:

1. Znovupoužitelnost kódu, díky komponentám.
2. Relativně snadné používání a poměrně strmou učící křivku.
3. Míra přizpůsobitelnosti je hlavní oblastí, kde se pojem knihovna a framework rozcházejí. ReactJS je knihovna, a právě v této oblasti má oproti frameworkům výhodu, jelikož je velice přizpůsobitelný. Zatímco

například AngularJS, jež je framework, neumožňuje příliš úprav, vývojář tedy nemá plnou kontrolu nad kódem a nemůže začlenit či měnit libovolné části přímo ve frameworku.

4. ReactJS používá virtuální DOM (z anglického Document Object Model, DOM), který vykresluje HTML uzly podle potřeby, čímž se snaží maximalizovat efektivitu.
5. Nabízí automatické metody pro snížení zatížení uživatelských zdrojů.
6. Je optimalizovaný pro vyhledávače (SEO) – díky čitelnému a jednoduchému kódu, je doba načítání webu téměř zanedbatelná. Navíc tato vlastnost je extrémně důležitá i z uživatelského hlediska, dle průzkumů odejde až 38 % návštěvníků webu, pokud je doba načítání delší než 5 sekund.

ReactJS funguje tak, že efektivně vykresluje obsah do DOM. Jeho podstatou jsou komponenty a budování uživatelského rozhraní z jednotlivých částí. Kromě toho poskytuje nástroje a funkce potřebné k určení toho, co se má vykreslovat, jakým způsobem a za jakých podmínek. Negativem může být, že knihovna ReactJS neobsahuje HTTP klienta, ani vlastní *Router,* který má za úkol vykreslovat různé komponenty podle URL adresy. Navíc postrádá další unikátní funkce a je bezesporu menší neboli štíhlejší než AngularJS. [43]

ReactJS kombinuje šablony uživatelského rozhraní a logiku JavaScriptu, což je vzhledem k ostatním originální koncept, který dosud žádný framework neimplementoval. Výsledek tohoto konceptu se nazývá JSX. ReactJS používá komponenty, které obsahují HTML značky i logiku JavaScriptu v jednom souboru a k tomu je navíc vyžadována pouze znalost jazyka JavaScript. Příklad tohoto chování je demonstrován ukázkou kódu viz. Obr. 4 níže.

```
import React from 'react'

export function UserList(props) {
  function userSelectHandler(userId) {
    props.onSelectUser(props.users.find(u => u.id === userId))
  }

  return (
    <ul>
      {props.users.map(user => (
        <li key={user.id} onClick={userSelectHandler.bind(null, user.id)}>
          {user.name}
        </li>
      ))}
    </ul>
  )
}
```

Obr. 4 Ukázka kódu ReactJS. Zdroj: [43]

ReactJS obvykle používá jazyk JavaScript (ale je ho možné používat i s TypeScriptem), a používá jeho speciální funkci, která se nazývá JSX, což není jeho přirozenou součástí. Projekty v ReactJS jsou na to připraveny, a tato syntaxe JavaScriptu, připomínající HTML, je při vývoji podporována a očekávána. Stejně jako TypeScript, je JSX při sestavování aplikace na pozadí zkompilováno do běžného, pro prohlížeč přijatelného jazyka JavaScript. [43]

AngularJS

Dle oficiálního zdroje, AngularJS je framework pro návrh aplikací a vývojová platforma pro vytváření efektivních a sofistikovaných SPA. [1] Jedná se o strukturální framework pro dynamické webové aplikace původně vyvinutý společností Google. Umožňuje používat jazyk HTML jako šablonovací jazyk a rozšiřuje jeho syntaxi, což umožňuje jasně a stručně definovat komponenty aplikace. AngularJS eliminuje velkou část repetitivního kódu, kterou by jinak programátor musel psát. Vše se navíc odehrává v prohlížeči, takže je vhodnou volbou pro jakoukoli serverovou technologii. [11]

AngularJS je z porovnávaných tří frameworků nejobsáhlejší. Často je označován jako vyvíjecí platforma než framework. AngularJS totiž nabízí mj. následující funkce či vlastnosti:

1. Ovládání uživatelského rozhraní.
2. Reagovat na vstupy uživatele.
3. Ověřování, uživatelem vložených, informací skrze formuláře.
4. Obsahuje vestavěný *Router* a správu stavu aplikace.
5. Umožňuje odesílat AJAX požadavky v rámci frameworku.
6. Zajištění testování typu end-to-end (metodologie, která ověřuje funkčnost softwaru).
7. Podpora správy více aplikací a jejich propojení.

Jednou z nejoceňovanějších vlastností na AngularJS jsou jeho neustálé aktualizace. Společnost vydává aktualizaci každých šest měsíců a nové verze jsou vytvářeny na základě těch starších. Kupříkladu aktualizace Angular 11 odstraňuje všechny chyby předchozí verze. Je doporučeno aktualizace sledovat, protože větší aktualizace může mít dopad na současný kód. Google však po vydání aktualizace čeká dalších šest měsíců, než stáhne nástroje z podporované předchozí verze, díky tomu je vývojářům poskytnut celý rok na změnu kódu, pokud to bude potřeba. Nicméně AngularJS kromě výše zmíněných bodů poskytuje další sadu nástrojů. Poskytuje podporu manipulace s DOM, a kromě toho obsahuje oficiální příkazový řádek, který pomáhá vytvářet a spravovat projekty, udržovat je aktuální, přidávat závislosti, a také provádět produkční nasazení. [43]

Projekty s AngularJS používají TypeScript, tedy nadmnožinu jazyka JavaScript. Protože TypeScript nelze spustit v prohlížeči, projekty s AngularJS jsou dodávány s nástroji, které interně kompilují kód TypeScriptu do jazyka JavaScript, který posléze už lze použít v prohlížečích. Protože je TypeScript staticky typovaný jazyk, je třeba uvést typ proměnné, což je přívětivější pro vývojáře se zkušenostmi s objektově orientovaným programováním. Takto by vypadala typická ukázka kódu s využitím AngularJS:

```
import { Component, Input, Output, EventEmitter } from '@angular/core'

@Component({
  selector: 'app-user-list',
  template: `
    <ul>
      <li *ngFor="let user of users" (click)="onSelectUser(user.id)">
        {{ user.name }}
      </li>
    </ul>
  `,
})
export class UserListComponent {
  @Output() selectUser = new EventEmitter<{ id: string; name: string }>()
  @Inout() users: { id: string; name: string }[]

  onSelectUser(id: string) {
    this.selectUser.emit(this.users.find(u => u.id === id))
  }
}
```

Obr. 5 Ukázka kódu AngularJS. Zdroj: [43]

Je vidět, že AngularJS používá funkci TypeScriptu zvanou dekorátory (@Component) k připojení dalších dat a logiky ke standardním třídám (UserListComponent). Tímto způsobem může být psán kód a framework se na pozadí postará o manipulaci s DOM. Vývojář tak nikdy nemusí psát žádný kód, který by přímo vytvářel nebo odebíral prvky v rámci DOM. Místo toho vytváří komponenty pomocí výše uvedené syntaxe a nechá zbytek zařídit AngularJS. Je umožněno definovat vstupy a výstupy komponent a také spravovat některé stavy specifické pro komponentu nebo pro celou aplikaci. [43]

Vue.js

Dokumentace Vue (vyslovuje se /vju:/) popisuje svůj produkt jako JavaScriptový framework pro vytváření uživatelských rozhraní. Je postaven na standardních jazycích jako HTML, CSS nebo prostém JavaScriptu a poskytuje deklarativní a na komponentách založený programovací model, který pomáhá efektivně vyvíjet ať už jednoduchá, nebo složitá uživatelská rozhraní. [42]

Správa dat je ve Vue.js realizována prostřednictvím Model-View-ViewModel (MVVM). Na rozdíl od jiných velkých frameworků je Vue.js postaven na myšlence komponentového a datově řízeného (z anglického data-driven) vývoje. V rámci MVVM jsou data a pohledy (Views) od sebe odděleny, nemůže mezi nimi probíhat

napřímo žádná komunikace. A proto je nutné využít View-Model vrstvu jako posluchače jak pohledové vrstvy (View), tak modelové (Model), aby mohly být sledovány provedené akce obou stran a včas provést odpovídající navazující nebo nějakým jiným způsobem provázanou operaci. Při aktualizaci dat se zároveň aktualizují i odpovídající uzly DOM. Naopak pokud se změní zobrazení v DOM, View-Model zavolá příslušnou aplikační logiku, aby aktualizovala data v Modelu. [18]

Vue.js je systematický framework, který se nabízenými funkcemi a velikostí nachází mezi knihovnou ReactJS a AngularJS frameworkem. Není tak komplexní jako Angular, ale obsahuje více funkcí než ReactJS. Charakteristickými vlastnostmi jsou kupříkladu:

1. Vue.js má vestavěnou správu stavů aplikace a je také vybaven vlastním *Routerem*. Neobsahuje však HTTP klienta, ani validaci HTML formulářů.
2. Simulace prokázaly, že překonává AngularJS a ReactJS při manipulaci s řádky a sloupci tabulek. Je to proto, že Vue.js na rozdíl od AngularJS používá virtuální DOM, který je jednoduší a přímočařejší než ten, který používá ReactJS.
3. Odladění programu probíhá paralelně s procesem programování. Uživatelské rozhraní lze vizualizovat, právě když je psán kód, což vývojářům pomáhá snadno ladit jejich kód.
4. Jedná se o framework s velikostí pouhých 21 KB, takže jeho stažení a spuštění nezabere téměř žádný čas. I kód ve Vue.js je krátký a přímočarý, tím je umožněn časově úspornější vývoj aplikací a šablon.

Základem Vue.js je vytváření uživatelských rozhraní integrací znovupoužitelných komponent, podobně jako je tomu u zbylých dvou porovnávaných frameworků. [43]

Vue.js používá HTML, JavaScript a CSS samostatně. Protože využívá přístup založený na šablonách (z anglického template-based approach), v praxi je pak možné se setkat s následujícím kódem:

```
<template>
  <ul>
    <li v-for="user in users" :key="user.id" @click="selectUser(user.id)">
      {{ user.name }}
    </li>
  </ul>
</template>
<script>
  export default {
    props: ['users'],
    methods: {
      selectUser: function(userId) {
        this.$emit(
          'selectUser',
          this.users.find(u => u.id === userId)
        )
      },
    },
  }
</script>
```

Obr. 6 Ukazka kódu Vue.js. Zdroj: [43]

Vue.js využívá prostý JavaScript, volitelně také TypeScript, a obvykle i jedno souborové komponenty (z anglického Single File Components). Stejně jako AngularJS odděluje HTML šablonu a JavaScript logiku. Obdobně jako ReactJS, tak Vue.js také podporuje JSX, ale ve většině projektů se používá kód na Obr. 6 výše, pravděpodobně se jedná o snáze uchopitelný kód, protože nedochází k míchání JavaScriptu s HTML. Psaní jednotkových testů pro izolované komponenty je relativně jednoduché pomocí jednosouborových komponent. [43]

Shrnutí

Předchozí kapitoly zkoumaly vhodnost frontend technologií pro tvorbu webových aplikací. Byly popsány frameworky AngularJS, Vue.js a ReactJS knihovna, včetně porovnání z různých hledisek mj. manipulace s DOM, funkcionality a syntaxe kódu. Lze konstatovat, že všechny tři mají své přednosti – AngularJS je robustní a časem odladěný, ReactJS je flexibilní a rychlý, Vue.js je jednoduchý a vysoce výkonný. Vývoj všech tří frameworků je velice aktivní, společnosti udržují aktuální verze a zároveň vydávají stále nové. Tyto tři frameworky lze použít pro vývoj webových aplikací v závislosti na zadavatelových požadavcích a cílech. [43]

3.2.2 Backend

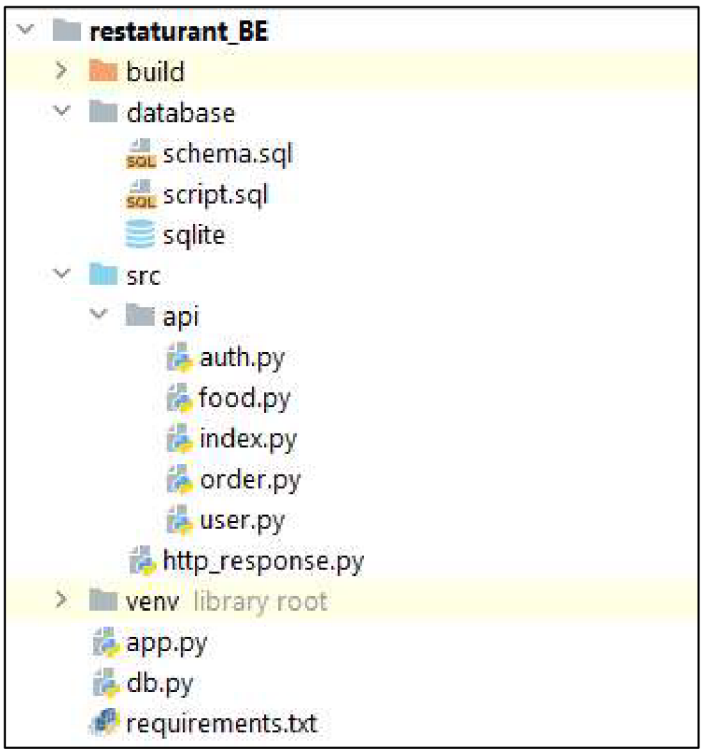
Backend technologie slouží pro programování serverového softwaru, tento program běží na serveru, obsluhuje příchozí požadavky a je napojen na databázi. Mezi nejběžnější jazyky využívajících se pro backend patří například Python, Java nebo JavaScript, tyto jazyky nejčastěji využívají nějaký framework, kterým může být Django, Spring nebo Flask. Backend technologie umožňují vývojářům vytvářet robustní a bezpečné webové aplikace, které mohou být nasazeny na širokou škálu platforem a zařízení, jsou tedy důležitým prvkem pro vývoj moderních webových aplikací a systémů.

Vzhledem k tomu, že v praktickém projektu je knihovna ReactJS zodpovědná za fungování aplikace v klientské části, je možné zvolit celou řadu technologií pro obsluhu části serverové, jelikož tyto dvě části na sobě nejsou závislé z hlediska integrace kódu, stačí pouze zajistit komunikaci mezi nimi. Pro vytvoření REST API, serverové logiky, middlewaru a dalšího, je pro webovou aplikaci, jež představuje praktický projekt, zvolen jazyk Python s mikro frameworkem Flask z důvodu flexibility, rozšiřitelnosti, velké komunitě, a především jeho snadného používání. Podrobnější popis je v nadcházející kapitole, kromě toho je popsána i častá alternativa pro Flask framework a to sice, taktéž na Pythonu založený framework Django.

Flask

Flask je mikro framework, jelikož obsahuje to nejnutnější pro vývoj backendu webových aplikací. Je navržen tak, aby začátek nového projektu byl rychlý a snadný a aby bylo možné jej v budoucnu rozšířit na komplexní aplikace. Původně byl určen pro zjednodušení práce se systémy Werkzeug (komplexní knihovna pro webové aplikace) a Jinja (bohatý šablonovací jazyk) a stal se jedním z nejoblíbenějších frameworků pro webové aplikace v jazyce Python. Flask nabízí návrhy, ale nevynucuje žádné závislosti ani rozvržení projektu. Je na vývojáři, aby si vybral nástroje a knihovny, které chce používat. Komunita poskytuje mnoho rozšířeních, která usnadňují přidávání nových funkcí. [24]

Přestože Flask striktně nestanovuje strukturu, podle které je vyžadováno se řídit, aplikace by měly mít jeden konkrétní způsob. Adresáře mají stromovou strukturu, kde se nacházejí všechny případné pohledy, šablony či obslužný kód k příchozím požadavkům. Instance aplikace je vytvořena při jejím startování, což je samotným frameworkem velmi doporučováno. [10] Na obrázku níže je znázorněn příklad adresářové struktury aplikace, v závislosti na použité frontend technologii se častokrát objevují podsložky s názvem „static“ a „templates“, právě pro soubory týkající se uživatelského rozhraní.



Obr. 7 Struktura Flask aplikace. Zdroj: [autor]

Flask zpracovává kontext požadavku a automaticky vkládá do obslužné funkce jeho objekt, který je zaveden v momentě, kdy server zaznamená nový příchozí požadavek. Podobně jako kontext celé aplikace, jedná se o globální objekt, který je k dispozici po celou dobu životnosti aplikace.

Primárním rozhraním pro aplikaci, skrze které je možné komunikovat zvenčí, je mapování URL adres na konkrétní obslužné funkce volitelně přijímající dodatečné informace v rámci dotazované adresy nebo těla požadavku. Ve Flasku je pohled definován dekorátorem „route“, přidruženou URL cestu lze zadat přímo v dekorátoru zadáním příslušného řetězce. Flask poskytuje taktéž dekorátory

„before_request“ a „after_request“, které provádějí kód před nebo po každém či pouze vybraném požadavku. Přidání cesty v aplikaci Flask je přímočaré viz. Obr. 8 představující endpoint pro odhlášení uživatele.

```
@bp.route('/logout')
def logout():
    session.clear()
    return json_response("ok")
```

Obr. 8 Flask endpoint. Zdroj: [autor]

„Blueprint“ je objekt obstarávající rozdělení aplikace do modulů, nejčastěji se využívá pro členění URL endpointů pomocí prefixu, musí však být připojen k aplikaci. Ve Flasku může mít jedna aplikace více takových objektů, pomáhají třídit kód podle účelu specifického pro danou URL adresu, nebo podle metody HTTP požadavku koncového bodu.

Vzhledem k tomu, že Flask je mikro framework, je povinností vývojářů myslet na bezpečnost aplikace. Existuje mnoho rozšíření, který tyto problémy řeší. Například před útoky CSRF (z anglického Cross-Site Request Forgery, CSRF) chrání rozšíření „Flask-WTF“ pro formuláře.

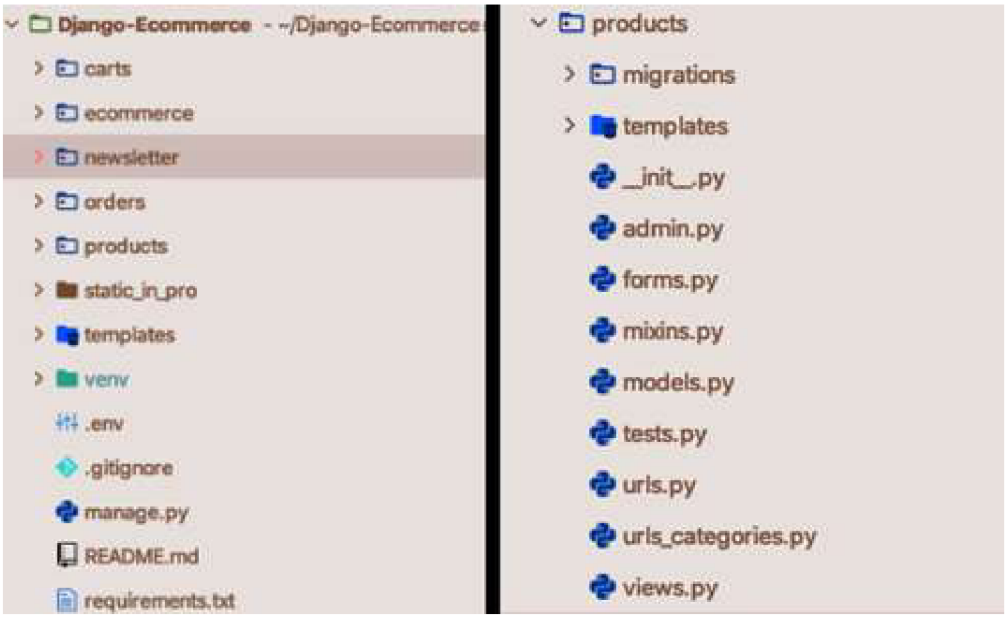
S pomocí rozsáhlé dokumentace široké komunity jsou učení a začátky poměrně rychlé. Většina obecných otázek je pokryta a demonstrována v dokumentaci, proto potřeba hledat pomoc při jakékoli základní implementaci či konfiguraci je zcela minimální. Definování API rozhraní aplikace Flask je velmi přímočaré, intuitivní a srozumitelné, na rozdíl od jiných rozhraní, která v sobě mají skrytou logiku. Komunita hraje zásadní roli v aplikacích s otevřeným zdrojovým kódem, Flask není výjimkou a je mezi vývojáři Pythonu na populárním serveru StackOverflow dobře znám. V současné době je na této platformě několik desítek tisíc dotazů s označením Flask včetně odpovědí. Při takovém počtu veřejně diskutovaných problémů lze nalézt většinu otázek týkajících se použití tohoto frameworku. [10]

Django

Django je vysokoúrovňový webový framework v jazyce Python, který podporuje rychlý vývoj a pragmatický design. Při vývoji byla hlavní motivace vyřešit většinu

problémů spojených s vývojem webu, takže je možné využít mnoho připravených funkcionalit. Je zdarma a s volně dostupným zdrojovým kódem. [6]

V rámci hlavní aplikace lze zavést více menších aplikací, které spolu budou komunikovat pomocí jednoduchých API rozhraních. Tyto aplikace je třeba přidat do seznamu, který se nachází v souboru s nastavením. Django má svá vlastní pravidla a způsob implementace funkcionalit, dodržuje také architekturu MVC. Pokud se programátoři řídí sadami pravidel frameworku Django, pak je možné ušetřit při vývoji čas. Jak ukazuje Obr. 9 níže, organizace různých částí aplikací je zabalena v samostatném adresáři. V rámci těchto adresářů jsou k vidění názvy "admin.py", "forms.py", "models.py", "views.py" a "urls.py" což jsou standardní názvy souborů. Většina implementace související s databází se nachází v souborech „models.py“ a „views.py“, které obsahují funkce obsluhující endpointy. Soubor „urls.py“ obsahuje logiku mapování URL adres na příslušné funkce.



Obr. 9 Organizace souborů v Django. Zdroj: [10]

Django zpracovává požadavky jinak než Flask. V Djangu je třeba objekt požadavku explicitně poskytnout obsluhující funkci nebo třídě. Objekt obsahuje všechny potřebné informace od aktuálního stavu celé aplikace až po aktuální relaci uživatele. Když klient požádá aplikaci o informace, vytvoří se objekt „HttpRequest“ a zavolá se příslušná funkce. Pokud je třeba objekt požadavku upravit nebo k němu

přistupovat, je třeba jej explicitně poskytnout funkci. V Djangu je obsluha požadavku řešena prostřednictvím URL dispečeru.

Bezpečnost je velmi důležitým aspektem každé webové aplikace. Především když jde o správu a údržbu informací koncového uživatele. Zavedení preventivních opatření a používání osvědčených postupů je však něco, za co by měl ručit backend, tedy framework webové aplikace. Vzhledem k tomu, že Django je open-source projekt, jsou všechny bezpečnostní aspekty frameworku promyšlené. Zahrnuje řešení pro následující bezpečnostní hrozby:

1. CSRF útok,
2. Cross-site scripting (XSS),
3. SQL injection,
4. Clickjacking,
5. Ověřování hlaviček požadavků,
6. SSL/HTTPS.

Většina potřebných informací je obsažena v dokumentaci. Vývojáři si při standardních požadavcích na aplikaci vystačí s pouhou dokumentací. Samotný framework nabízí širokou škálu funkcí. Proto je logické, že i dokumentace je rozsáhlá. Vývojáři, kteří mají zkušenosti s programovacím jazykem Python, se pravděpodobně naučí framework rychleji než vývojáři pocházející z jiných jazyků. Má velkou komunitu a vývojář může velmi rychle získat pomoc od ostatních. V době psaní této práce se na serveru StackOverflow, objevilo již přes 250 tisíc otázek s označením Django. [10]

Shrnutí

Oba frameworky mají své výhody i nevýhody, rozhodně se oba zdají býti vhodnou možností pro využití při vývoji webové aplikace. Na základě informací v předešlých kapitolách je zřejmé, že Django je obtížnější na naučení, ale může být vhodnějším kandidátem pro rozsáhlé projekty. Flask se nejlépe hodí pro tvorbu projektů menšího rozsahu, ale neomezuje se pouze na ně. Flask se lze naučit a nakonfigurovat rychle, ale pokud jde o správu a údržbu, vyžaduje více práce než první jmenovaný. [10]

3.2.3 Databáze

Databáze jsou nezbytnou součástí webových aplikací. Často je nutné ukládat, zpracovávat a poskytovat data, což vyžaduje vhodný systém pro jejich správu. Existují různé typy databázových systémů, které mohou být použity v rámci webové aplikace, jimiž jsou především relační či NoSQL databáze.

Databáze je rozsáhlá kolekce strukturovaných dat, ke které lze přistupovat a vyhledávat v ní konkrétní informace. Relační databáze jsou známé svou strukturou a mezi programátory mají dlouholetou tradici. Jedná se o úložiště dat, které udržuje vztah mezi dvěma nebo více entitami neboli tabulkami. Obecně lze říci, že relační databáze je struktura pro ukládání dat, kde každý záznam, skládající se z hodnot atributů, představuje jeden řádek v tabulce. Mezi hlavní výhody relačních databází patří:

1. Proces dotazování na data využívá jazyk SQL. Tento jazyk je mezi vývojáři dobře znám, oproti ostatním řešením, kde je nutné naučit se syntaxi dané technologie poskytující ukládání dat.
2. Správně implementovaná relační databáze zajistí, že všechna data zůstanou neporušená ve správném formátu a konzistentní v čase.
3. Jsou navrženy tak, aby byly snadno pochopitelné a použitelné. Vztahy mezi všemi tabulkami a ostatními prvky jsou jasně definovány, takže je zřejmé, jak spolu fungují.

Relační databáze mají i své nevýhody, zejména:

1. Nejsou vhodná pro analýzu dat v reálném čase, protože neukládají data tak, aby bylo možné se na ně rychle znovu dotázat.
2. Relační databáze mají fixní schéma. Strukturu databáze nelze měnit za běhu aplikace, to může být časově náročnější a náchylnější k chybám. [28]

Na druhou stranu non-relační databáze, známy pod názvem NoSQL nebo taktéž objektově orientované databáze, se od relačních databází velmi liší. Jsou to databáze, ve kterých nejsou striktně rozlišovány relace, řádky ani sloupce. Objekty uložené v těchto databázích nemusí mít strukturalizovaný tvar. Non-relační databáze

pomáhají vylepšit rychlost, výkon, a především poskytují mnohem flexibilnější používání než relační databáze. Shrnutí jejich pozitivních vlastností je:

1. Výhodou NoSQL databází je jejich rychlost, například prohledávaní každého záznamu, aniž by se musely nejprve dotazovat databáze.
2. Protože nejsou omezeny fixním schématem, lze je škálovat snadněji než relační databáze. To je užitečné zejména při provádění složitých výpočtů nad velkým množstvím dat.
3. Data lze ukládat v jakémkoli formátu využívaném danou aplikací.

Mezi hlavní nevýhody NoSQL databází patří:

1. Žádná standardizace. Každá firma určuje vlastní způsob používání databáze a její funkce, což ztěžuje implementaci.
2. Non-relační databáze nemá pevnou strukturu, což je uvedeno i jako výhoda, ale toto může ztížit údržbu webové aplikace z důvodu neočekávaných vstupů a chování. [28]

Pro praktický projekt této práce byl zvolen přístup ukládání dat v relační databázi. Konkrétně se jedná o databázový systém SQLite, protože je vhodný pro menší projekty nevyžadující složité dotazy, podporuje standardní SQL syntaxi, a navíc je v Pythonu, tedy v programovacím jazyku backendu aplikace, snadno použitelný, jelikož je zabudován v rámci Pythonovské knihovny „sqlite3“. V případě nutnosti nebo zájmu používat NoSQL databázi se jako první nabízí systém MongoDB. Dle statistik serveru StackOverflow za rok 2022, je MongoDB nejpoužívanější NoSQL databáze a zároveň čtvrtá nejpopulárnější ze všech. [39] Právě tyto nástroje pro uchovávání informací jsou předmětem následujících kapitol, kde jsou popsány jejich hlavní vlastnosti, výhody a nevýhody.

SQLite

Oficiální zdroj definuje SQLite jako knihovnu v jazyce C, která implementuje malý, rychlý, samostatný, vysoce spolehlivý a plnohodnotný SQL databázový engine. SQLite je nejpoužívanější databázový engine na světě, je zabudován ve všech mobilních telefonech a většině počítačů a je součástí mnoha dalších aplikací. [38]

Systém SQLite nepotřebuje velkou podporu od operačního systému, je tedy velmi přenosný. Navíc je umožněno importovat a exportovat databázi mezi 64bitovými a 32bitovými počítači či operačními systémy. Nevyžaduje ke svému fungování tradiční architekturu klient-server – je „bez serverová“. SQLite je tzv. databáze v paměti (z anglického in memory), nejčastěji se tedy integruje do aplikace, která k ní přistupuje napřímo. Aplikace ukládá a čte data přímo ze souboru, jež je uložen na disku, jenž je identifikovatelný pomocí jedné z přípon .sqlite3, .sqlite nebo .db. Protože je SQLite „bez serverová“, po vývojáři softwaru není vyžadována žádná instalace ani konfigurace před jejím použitím. Kromě toho se nevytváří žádné konfigurační soubory a pro připojení aplikace k databázi není nutné spouštět, ani zastavovat, žádný databázový server. Všechny transakce se řídí ACID (z anglického atomic, consistent, isolated, and durable, ACID), tedy zachovávají atomicitu, konzistenci, izolovanost a trvalost, SQLite transakce tedy proběhnou buď celé anebo vůbec. [36]

MongoDB

Dokumentace tohoto produktu uvádí, že MongoDB je univerzální databáze dokumentů. Základní koncepcí MongoDB je flexibilní NoSQL schéma, ukládající data jako BSON dokumenty, tedy JSON objekty v binární podobě, seskupených do kolekcí a databází. MongoDB funguje i na malých zařízeních, ale její určení je především jako backend databáze pro webové aplikace. [21]

MongoDB je ideální pro práci s hierarchicky strukturalizovanými informacemi. Data jsou ukládána s využitím JSON formátu, z čehož plyne v tomto ohledu lepší správa, než je tomu u relačních databází. MongoDB je dobře uzpůsobena pro vertikální i horizontální škálovatelnost, zatímco většina databází založených na SQL je omezena na vertikální škálování například prostřednictvím navýšení paměti RAM. Nicméně databáze MongoDB lze škálovat přidáním dalších serverů a jejich paralelním provozem. Pokud je k dispozici dostatek prostředků, pak může její výkon rapidně vzrůst, protože ukládá velkou část svých dat do paměti RAM počítače. V důsledku toho umožňuje rychlejší přístup k datům. Syntaxe dotazů JSON může být jednodušší na pochopení než SQL, navíc je mnohem expresivnější a srozumitelnější

pro uživatele, kteří mají zkušenosti s jazykem JavaScript. MongoDB umožňuje vytvářet dotazy, které získají informace o aktuálním stavu databáze. [36]

3.3 Asynchronní komunikace v rámci webové aplikace

Ve webové aplikaci, přičemž praktický projekt této práce není výjimkou, probíhá neustálá komunikace za účelem výměny dat mezi uživatelským rozhraním (frontend) a serverem aplikace (backend). V tradičním přístupu se při každé akci vyvolané uživatelem odesílá požadavek na server, jenž následně posílá zpět odpověď s novými daty a celá stránka se v prohlížeči znovu načítá s již aktualizovanými informacemi. Tento přístup má několik nevýhod, které v některých případech zhoršují zážitek uživatele. Při synchronní komunikaci všechny procesy čekají, dokud nejsou data k dispozici, aby mohly pokračovat v činnosti na stránce, tím je zaručeno, že žádný proces neproběhne dříve, než bude komunikace ukončena, což je v prostředí vývoje webových stránek spíše nevýhodou, protože by v témže čase mohly být vykonávány práce, které s přenášenými daty nesouvisí.

Eliminaci tohoto problému řeší přístup asynchronní komunikace, jedná se o metodu výměny zpráv mezi dvěma nebo více stranami, kupříkladu frontend a backend. Každá strana přijímá, odesílá a zpracovává zprávy kdykoli je to možné, což nemusí nutně znamenat bezprostředně po jejich obdržení, protože příjemce zprávy může momentálně vykonávat činnosti s vyšší prioritou. Kromě toho lze zprávy posílat bez čekání na potvrzení s tím, že pokud se vyskytne problém, příjemce si vyžádá novou odpověď nebo situaci vyřeší vlastním způsobem. Analogií k této problematice je použití e-mailové komunikace, odesílatel odešle e-mail a příjemce si zprávu přečte a odpoví na ni, až to bude vhodné, než aby tak učinil ihned. Obě strany mohou nadále odesílat a přijímat zprávy, místo čekání na odpověď.

Pro asynchronní komunikaci ve webových stránkách může být poslán požadavek na informace od serveru a během čekání na odpověď pokračovat v dalších činnostech. V tomto případě, kde příjemce zprávy je frontend, jsou data obsažená v odpovědi zpracována, jakmile to je možné, následně program aktualizuje uživatelské rozhraní, bez nutnosti znovu načítání celé stránky, taktéž bez nutnosti předchozího blokování probíhajícího kódu. Pro implementaci této komunikace se

využívá AJAX (z anglického Asynchronous JavaScript and XML, AJAX), který doslovně znamená „asynchronní JavaScript a XML“, i přesto, že se v současnosti používá jakožto formát pro přenášená data více JSON než XML, tak se toto označení, tedy AJAX, stále používá. [20]

Protože technologie AJAX je zásadní pro tvorbu nejen praktického projektu, ale obecně pro vytváření jakékoliv SPA, je věnována kapitola 3.3.3 níže, právě charakteristice této technologie. Před samotným popisem AJAX je vysvětlen pojem REST architektura společně s pojmem HTTP požadavek, které jsou pro správné pochopení a využívání technologie AJAX nezbytné.

3.3.1 REST API

Termín API je soubor definic a protokolů, někdy je popisován jako úmluva mezi poskytovatelem informací (serverem) a uživatelem informacích (klientem). Stanovuje, jak má vypadat požadavek a jakou podobu bude mít odpověď na něj. Například návrh API pro meteorologickou službu by mohl specifikovat, že uživatel služby zadá poštovní směrovací číslo a poskytovatel služby zašle zpět odpověď s dvěma části. První znamená nejvyšší teplota během dne a druhá je nejnižší teplota. [31]

REST je architektura, nikoli protokol nebo standard. Vývojáři API mohou implementovat REST různými způsoby. Když je proveden požadavek klienta na REST API, přenáší se požadovaný zdroj (z anglického resource), který je dosažitelný na právě jednom z mnoha URL endpointů, žadateli. Tato informace je prostřednictvím protokolu HTTP, doručena v jednom z několika formátů, nejčastěji JSON. Volání na REST API jsou tedy HTTP požadavky. Aby bylo rozhraní API považováno za REST API, musí splňovat kromě jiných i tato kritéria:

1. Architektura klient-server skládající se z klientů, serverů a zdrojů, přičemž požadavky jsou prováděny prostřednictvím protokolu HTTP.
2. Bezstavová komunikace mezi klientem a serverem, což znamená, že mezi požadavky na získání informací se neukládají žádné informace o klientovi a každý požadavek je samostatný a nezávislý na ostatních.

3. Data mohou být ukládána do mezipaměti, což zefektivňuje interakci mezi klientem a serverem. [31]

3.3.2 HTTP Požadavek

HTTP požadavek je vytvářen klientem a posílán na server. Cílem požadavku je získat přístup ke zdroji na serveru. HTTP odpověď poskytuje server klientovi. Cílem odpovědi je poskytnout klientovi požadovaný zdroj nebo informovat klienta, že jím požadovaná akce byla provedena, případně informovat klienta, že při zpracování jeho požadavku došlo k chybě. V HTTP odpovědi, která je zaslána klientovi, se nachází stavový kód, dále jen „status“, což je třímístné číslo, doprovázeno vysvětlující frází, která shrnuje význam kódu. Ze statusu odpovědi je možné určit výsledek požadavku. Specifikace HTTP protokolu klasifikuje statusy dle jejich hodnoty následovně:

- 100-199 jsou pouze informativní odpovědi,
- úspěšné požadavky mají hodnotu 200-299,
- pokud je výsledkem přesměrování status má hodnotu 300-399,
- v případě chyby ze strany klienta bude hodnota 400-499,
- při chybě na straně serveru se vrátí odpověď se statusem 500-599,
- další kódy, které nespadají do žádného z předešlých rozsahů HTTP protokol nedefinuje. [13]

Na ukázku jsou v následující Tab. 2 vypsány některé status kódy, které za jistých podmínek je možné získat v HTTP odpovědi praktického projektu Restaurace. Pro správnost nejsou v aplikaci statusy psány ručně, ale jsou vyhledávány podle jména v Python knihovně „http“. Kupříkladu získání hodnoty statusu se jménem „OK“, lze v Pythonu provést příkazem http.HTTPStatus.OK.value, podobným způsobem je možné zjistit frázi i dlouhý popis.

Tab. 2 Status kódy. Zdroj: [autor]

Kód	Jméno	Význam
200	OK	Požadavek s tímto kódem značí, že proběhl úspěšně, ale nespecifikuje bližší informace.
201	CREATED	Tento požadavek byl úspěšný, a navíc proběhlo vytvoření nového zdroje.
400	BAD_REQUEST	Odpověď má status 400 protože požadavek nemohl být serverem zpracován, kvůli chybě v požadavku.
401	UNAUTHORIZED	Server odmítl obsloužit požadavek, protože uživatel nebyl autorizován.

Požadavkům i odpovědím lze nastavovat spoustu parametrů, jedním z nejdůležitějších u požadavku je tzv. HTTP metoda, která určuje, jestli požadavek chce zdroje z příslušné URL adresy například získat nebo smazat. Základní operace pro práci s daty jsou nazývány zkratkou CRUD z anglického Create, Read, Update and Delete, přeloženo jako Vytvoření, Čtení, Upravení a Smazání. V architektuře REST, kde se využívá HTTP protokol, mají CRUD operace následující charakteristiku [34]:

Tab. 3 CRUD operace. Zdroj: [34]

Operace	HTTP metoda	Určení
Vytvořit (Create)	POST	Při metodě POST klient odešle data serveru, který vytvoří prostředek s dodanými daty.
Číst (Read)	GET	Požadavek GET žádá REST API o načtení zdroje, například záznam v databázi nebo obsah souboru, a jeho odeslání klientovi.

Upravit (Update)	PUT	Klient odešle požadavek PUT na REST API, aby aktualizoval existující data v databázi.
Smazat (DELETE)	DELETE	Klient odešle požadavek DELETE, aby odstranil existující zdroj.

3.3.3 AJAX

Asynchronní komunikace se často využívá i v praktického projektu, kde je frontend vytvářen s technologií ReactJS. Protože tato knihovna neobsahuje vlastního HTTP klienta, je nutné, aby byla komunikace se serverem obsluhována „obyčejným“ JavaScriptem, který následně bude uvědomovat knihovnu ReactJS, aby s daty nadále pracovala. Provést AJAX požadavek je dovoleno s využitím více syntaxí, v rámci projektu Restaurace je vždy využíván postup s klíčovými slovy „async“ a „await“. Pro zprostředkování HTTP klienta byla vybrána knihovna Fetch z následujících důvodů:

1. Knihovna Fetch je součástí standardu JavaScript, tím pádem je součástí programového vybavení prohlížeče.
2. Jeho moderní syntaxe je přizpůsobena pro práci s asynchronním kódem.
3. Je založena na principu HTTP požadavek a HTTP odpověď, přičemž dokáže pracovat s daty v často používaných formátech XML, JSON, TEXT nebo HTML, aniž by zapříčinila kompletní znovu načtení stránky.
4. Při volání funkce pro vytvoření požadavku je možné vložit jako parametr objekt s vlastnostmi, které se mají pro volání nastavit, může jimi být zvolená HTTP Metoda, objekt s přidruženými daty nebo požadované HTTP Hlavičky a další.

Ukázka kódu jednoduchého AJAX požadavku:

```
1.   async function loadDoc() {
2.   let url = 'https://api.github.com/users';
3.   try{
4.   const response = await fetch(url);
5.   const json = await response.json();
6.   document.getElementById('demo').innerHTML = (JSON.stringify(json));
7.   }catch(err)
8.   { console.log('fetch failed - async syntax', err);
}}
```

Obr. 10 Ukázka AJAX požadavku. Zdroj: [40]

Kód výše na Obr. 10 využívá právě syntaxi async/await. Mnoho vývojářů dává přednost tomuto způsobu, protože je velmi podobný syntaxi v ostatních jazycích. Kdybychom odstranili slovo await z předcházejícího kódu, na obrazovce se objeví prázdná závorka {}. Funkce by stále byla označena slovem async, ale neprováděla by žádný asynchronní příkaz definovaný slovem await, z toho důvodu by funkce nebyla pozastavena v těchto místech a nečekala by na odpověď. Tím je demonstrováno, že pokud odpověď není načtena a klíčové slovo await chybí, nebude se na odpověď čekat a JavaScript pokračuje dál. V opačném případě, který je vidět na Obr. 10, tedy kdy je await správně doplněn, JavaScript pozastaví provádění funkce a věnuje se ostatnímu kódu, který může provádět během čekání na odpověď. Když dorazí odpověď vloží ji do konstanty „response“ z které jsou následně přečteny JSON data. Téměř celé tělo funkce je navíc obaleno kódem try/catch pro případ, že ze serveru přijde negativní odpověď, status takové odpovědi má většinou hodnotu větší nebo rovnu 400. [40]

3.4 Bezpečnost webových stránek

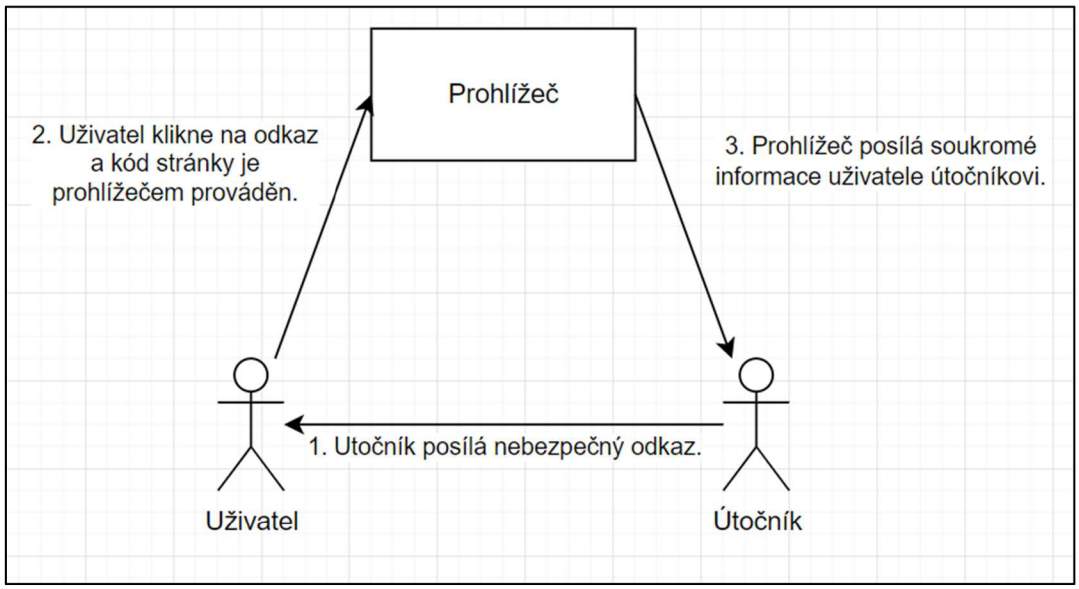
Jelikož bezpečnost webových stránek je velice široké téma, není v možnostech této diplomové práce obsáhnout celou tuto problematiku. V této kapitole budou rozebrány známé útoky a zranitelnosti nejen webových stránek založených na knihovně ReactJS.

V dnešním světě je třeba mít na paměti rizika spojená s jakoukoli technologií, která je součástí webové aplikace. Ačkoli má ReactJS menší počet zranitelností než

jiné frameworky, stále není zcela bezpečný. Vzhledem k tomu, že ReactJS je kompatibilní s jinými komponentami s otevřeným kódem a nemá velmi přísné nastavení zabezpečení, stává se zranitelným i z tohohle hlediska. Navíc aplikace častokrát sdílejí obrovské množství osobních údajů, a to zvyšuje pravděpodobnost jejich vyzrazení. [41]

3.4.1 Zranitelnosti webových stránek

Útoky „Cross-Site Scripting“, dále jen XSS, jsou jedny z nejčastějších zranitelností na internetu a objevily se jako přímá reakce na rostoucí množství interakcí uživatelů v dnešních webových aplikacích. Podstatou útoku XSS je využití skutečnosti, že webové aplikace spouštějí skripty v prohlížečích uživatelů. Jakýkoli typ dynamicky vytvořeného skriptu, který je spuštěn, ohrožuje webovou aplikaci, pokud může být spouštěný skript jakýmkoli způsobem upraven - zejména koncovým uživatelem. Posloupnost kroků útoku je zobrazena na Obr. 11.



Obr. 11 XSS útok. Zdroj: [41]

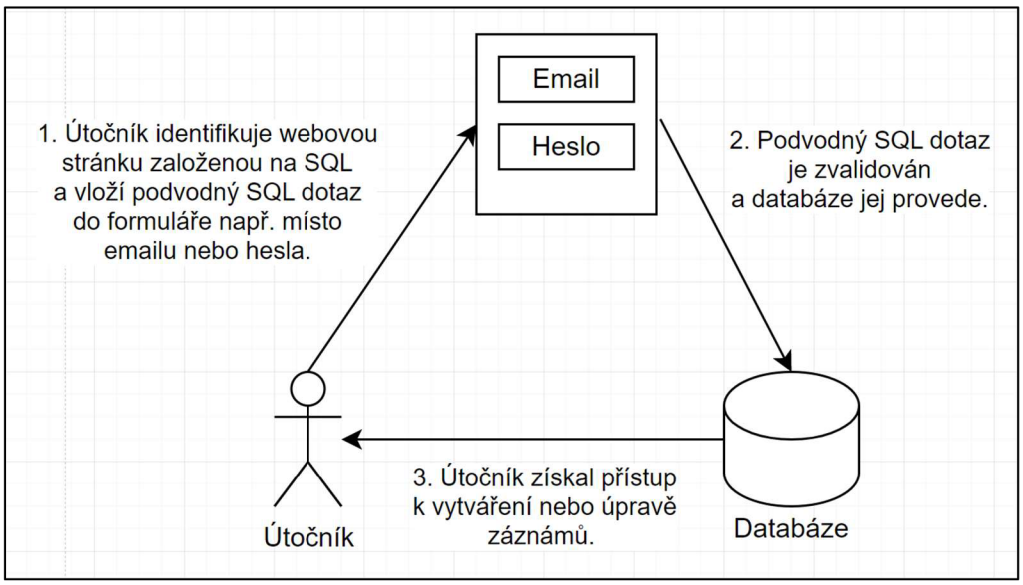
Útoky XSS se dělí do několika kategorií, z nichž hlavní tři jsou:

1. Uložené, kód je před spuštěním uložen v databázi.
2. Reflektované, kód není uložen v databázi, ale existuje přímo na serveru.
3. DOM, kód je uložen i spuštěn v prohlížeči.

Kromě toho existují ještě další varianty, ale tyto tři zahrnují typy XSS, na které si většina moderních webových aplikací musí dávat pozor. Tyto tři typy útoků XSS byly označeny organizací Open Web Application Security Project (OWASP) za nejčastější druhy útoků XSS na webu. [12]

Dalším častým problémem v aplikacích React.js je nedostatečná nebo špatná autorizace. To může vést k tomu, že útočníci prolomí přihlašovací údaje uživatelů útoky hrubou silou. S nefunkční autorizací mají spojitost různá rizika, například odhalení identifikátorů relací v adresách URL, snadné a předvídatelné přihlašovací údaje, nešifrovaná komunikace a další faktory související s relacemi. [41]

Následný popisovaný útok proti webové aplikaci je SQL injection. SQL injection je typ útoku, který se zaměřuje na databáze SQL a umožňuje útočníkovi buď poskytnout vlastní parametry existujícímu dotazu SQL, nebo použít svůj vlastní dotaz. SQL injection je nejběžnější formou injekce kódu, ale ne jedinou. Průběh útoku je zobrazen na Obr. 12. [12]

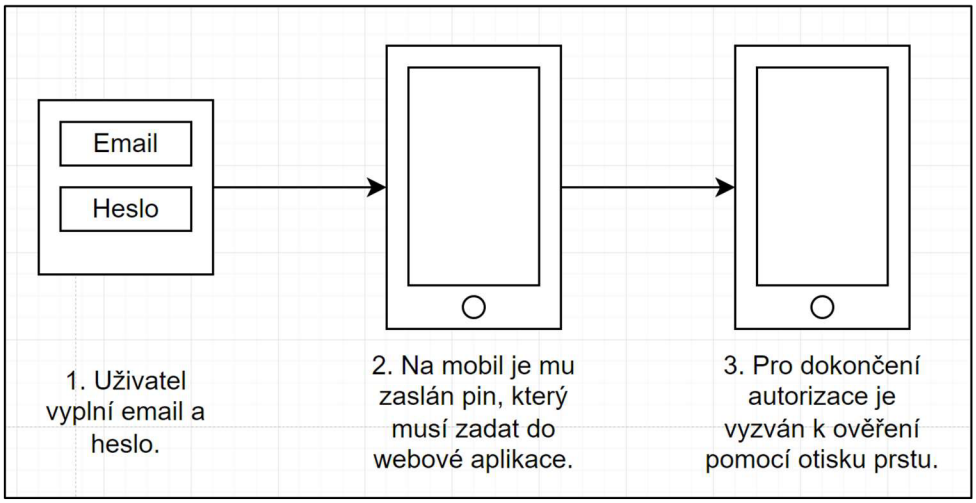


Obr. 12 Útok SQL Injection. Zdroj: [23]

V aplikacích ReactJS existuje velmi specifická zranitelnost známá jako Zip Slip, která spočívá ve zneužití funkce umožňující nahrávání souborů zip na server. Útočník může nahrané soubory rozbalit mimo přidělený adresář, pokud nástroj použitý k rozbalení souboru zip není bezpečný, a pak získá přístup k souborům umístěným na serveru.

3.4.2 Zabezpečení ReactJS aplikace

Základní, ale důležitou zásadou pro zabezpečení aplikace je zajistit, aby spojení mezi serverem a klientem bylo bezpečné. Pokud je to možné, účinnou technikou je použití více faktorového ověřování. Tato metoda ověřování zajišťuje, že uživatel získá přístup k důležitým částem webové aplikace až po zadání dvou nebo více ověřovacích údajů pro autorizaci. Možná implementace tří fázového ověření je na Obr. 13.



Obr. 13 Tří fázové ověření uživatele. Zdroj: [41]

Každá webová aplikace založená na knihovně ReactJS potřebuje vykreslovat HTML kód, proto je nutné eliminovat v něm zranitelnosti. K tomu slouží tři osvědčené postupy:

1. Nastavit HTML elementům atribut „disabled“. Pokud prvek má tento atribut, pak je neměnný. Tomuto prvku není možné nastavit fokus nebo s ním odeslat formulář. Doporučený postup je povolit prvek pouze tehdy, když jsou data uvnitř formuláře validována.
2. Používat tzv. escapování textu. ReactJS používá syntaxi JSX, která umožňuje psát HTML a má vestavěnou funkci automatického escapování, která je použita k zabezpečení webové aplikace. Pokud se útočník pokusí vložit škodlivý kód, analyzátor JSX tento neplatný vstup odhalí, data tedy budou escapována a útok bude neutralizován.
3. Použití atributu „dangerouslySetInnerHTML“. Pokud webová aplikace potřebuje vykreslovat dynamický HTML kód, jenž je standardně

prováděno pomocí nastavení 'innerHTML', činí aplikaci zranitelnou vůči škodlivým datům. ReactJS obsahuje atribut, který na potenciální zranitelnost může upozornit, nazvaný „dangerouslySetInnerHTML“. Ten funguje jako varování, takže je možné zkontrolovat a ujistit se, že data zadaná do HTML prvku skutečně pocházejí z důvěryhodného zdroje. Taktéž je možné využít knihovny třetích stran. [41]

Při používání HTML elementu „a“ je třeba dávat pozor na útočníky, kteří se snaží vkládat URL odkaz začínající s předponou JavaScript. Vždy je třeba ověřovat, že URL adresa obsahuje protokol HTTP nebo HTTPS.

V ReactJS aplikaci je třeba používat princip nejmenšího oprávnění. To znamená, že každý uživatel musí mít povolen přístup pouze k těm informacím, které jsou nezbytně nutné. Při připojování k databázi prostřednictvím webové aplikace je nebezpečné povolit komukoli aktualizovat, vkládat nebo mazat data, proto je důležité přiřadit uživatelům správné role v databázi.

Závěrem k bezpečnosti, každé produkční použití webové aplikace musí zajistit, aby všechna data odesílaná přes síť byla šifrována. Tím se sníží riziko útoku typu „man-in-the-middle“, který by mohl uživatelům ukrást přihlašovací údaje. Secure Sockets Layer (SSL) a Transport Layer Security (TLS) jsou dva hlavní kryptografické protokoly, které se dnes používají k zabezpečení dat při jejich posílání po síti. Všechny moderní prohlížeče šifrovanou komunikaci vyžadují, sice i přesto mohou povolit komunikaci s cílovým serverem, ale výrazně to sníží důvěru uživatele k webové aplikaci. [12]

4 Analýza a návrh projektu

Tato kapitola se zabývá analýzou a návrhem praktického projektu této práce, dále označovaného jako projekt Restaurace. Jak již bylo v přechozích textech psáno, nejvíce v kapitole „3.2 Technologie pro vývoj webových aplikací“, tento projekt bude zhotoven především s využitím technologií ReactJS a Python Flask. Probírána bude především, ale ne zcela, problematika vývoje frontendu, nikoli backendu, jelikož to je hlavní cíl této práce.

V nadcházejících podkapitolách bude popsáno několik verzí neboli vývojových fází aplikace, které se postupně stávají více komplexní, vedoucí k finální podobě webu. Nejprve proběhne seznámení s potřebnými diagramy, jimiž jsou například entitně vztahový diagram (z anglického Entity-relationship diagram, ER Diagram) vyobrazující vztahy mezi tabulkami v databázi, digram všech možných pohledů na stránce či drátový model uživatelského rozhraní.

Dále bude probírána raná fáze projektu, kde proběhne seznámení s uspořádáním složek, souborů a prvními komponentami. Dalším krokem bude nastínit proces přihlašování, následovaný vytvořením několika dynamických stránek projektu. Jako poslední budou vysvětleny výhody knihovny Redux a její často používané nadstavby Redux Toolkit. Všechny zdrojové kódy týkající se frontend vývoje jednotlivých fází jsou k dispozici online na odkazech uvedených v přílohách 8.1 - 8.5. Zdrojový kód pro backend zůstává neměnný po celou dobu vývoje, taktéž přístupný v gitovém repozitáři na URL adrese vložené jako příloha 8.6.

4.1 Požadavky a návrh projektu

Hypotetický klient, restaurační zařízení zaměřené na prodej především pizzy a burgeru, vyžaduje takovou webovou aplikaci, která zaměstnancům umožní správu objednávek a pro zákazníky zlepší proces objednávání a umožní správu jejich údajů. Webová stránka musí disponovat moderním a jednoduchým designem a konkrétně těmito funkcemi:

1) Objednávkový systém:
    a) Zákazník může vytvářet objednávku pouze v otevíracích hodinách.

b) Objednávce je možné nastavit adresu doručení pouze po Praze.
c) Při vytvoření objednávky je nutné, aby systém automaticky určil její čas zhotovení na základě předpokládané vytíženosti restaurace.
d) Nabídka je k dispozici všem a vždy, ale objednat mohou pouze přihlášení uživatelé.
e) Zákazník má přístup k historii svých objednávek a k údajům zadaných při registraci

2) Správa systému zaměstnanci:
a) Zaměstnanci po přihlášení do aplikace, svými privilegovanými účty, vidí kompletní historii všech objednávek a mohou měnit jejich stav.
b) V případě nutnosti mohou zaměstnanci smazat libovolné uživatelské účty, zároveň tedy mají přístup k seznamu všech zákazníků.

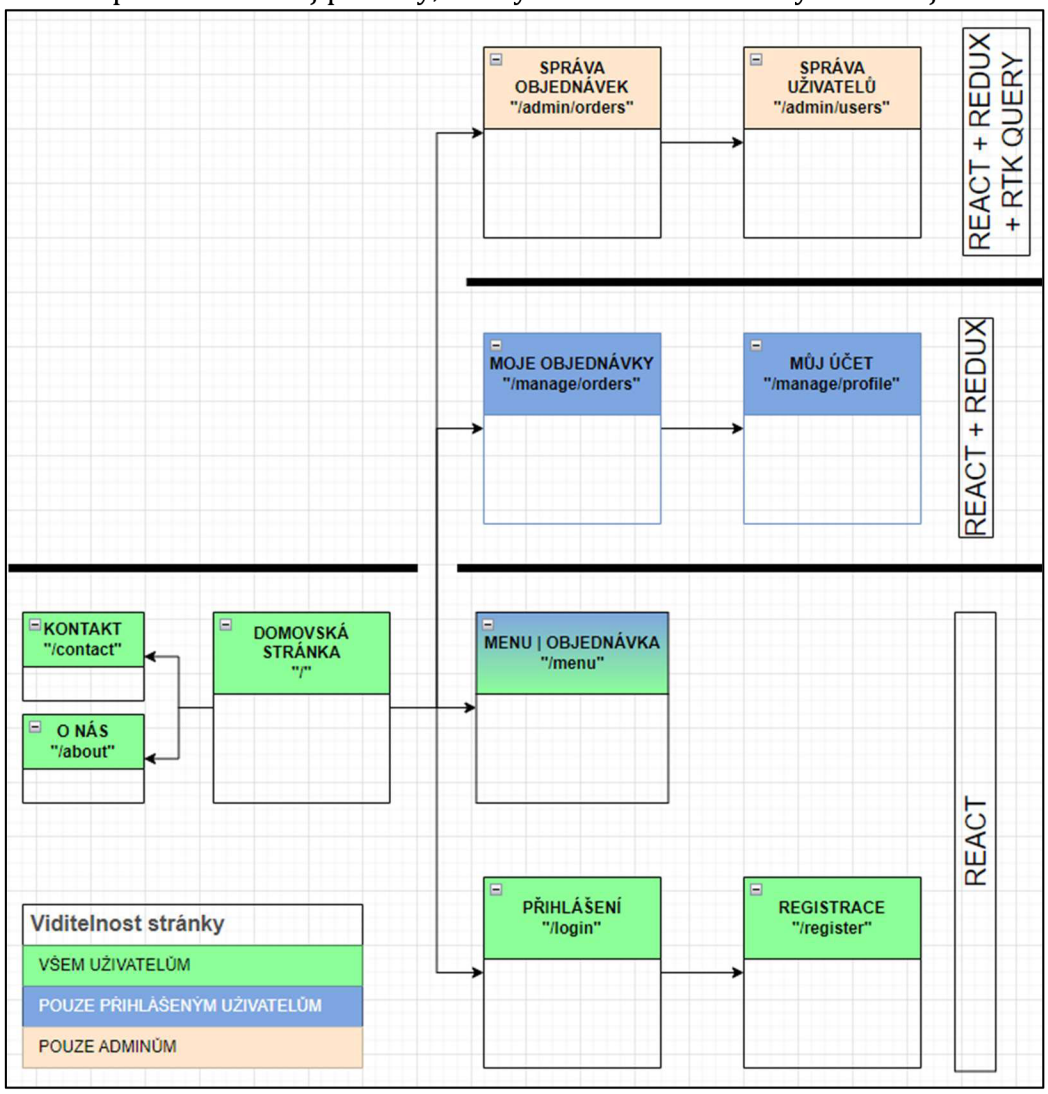
4.1.1 Pohledy v aplikaci

Na základě těchto požadavků je nejprve vypracován návrh všech pohledů neboli stránek ve webové aplikaci. Jedná se o Obr. 14 níže, díky němuž lze lépe pochopit pohyb na stránce a jejich provázanost. Každý jednotlivý obdélník s nadpisem na barevném pozadí představuje jeden pohled, respektive stránku, která bude později implementována. Pod nadpisem je v uvozovkách napsána relativní URL adresa, na které bude stránka ve webové aplikaci přístupná. Vazby mezi obdélníky v podobě šipek napovídají, jakým způsobem se uživatel může pohybovat v aplikaci a jak spolu stránky z hlediska jejich obsahu souvisí. Nadpis, shrnuje účel stránky, zatímco barva pozadí označuje, pro jakého uživatele bude stránka přístupná.

- Zelenou barvou jsou označeny stránky přístupné všem, tedy zaměstnancům a přihlášeným i nepřihlášeným uživatelům. Patří mezi ně stránky s názvy „domovská stránka“, „kontakt“, „o nás“, „přihlášení“ a „registrace“.
- Obsah na stránkách s modrým nadpisem, tedy „moje objednávky“ a „můj účet“ se zobrazí pouze přihlášeným uživatelům, kde budou moci listovat svými objednávkami a spravovat své údaje.

- Poslední barvou na obrázku je oranžová, na těchto stránkách zaměstnanci spravují objednávky a uživatele. Jedná se o stránky „správa objednávek“ a „správa uživatelů“.

Pohled s názvem „menu | objednávka“ má modro-zelenou barvu z důvodu, že je přístupný všem, ale přihlášeným uživatelům bude zobrazován odlišný obsah než nepřihlášeným. Na této stránce nepřihlášení uživatelé uvidí pouze nabídku restaurace, zatímco přihlášení uživatelé uvidí taktéž nabídku, ale navíc i svůj košík s možností přidávat do něj položky, a díky tomu následovně vytvořit objednávku.



Obr. 14 Pohledy ve webové aplikaci. Zdroj: [autor]

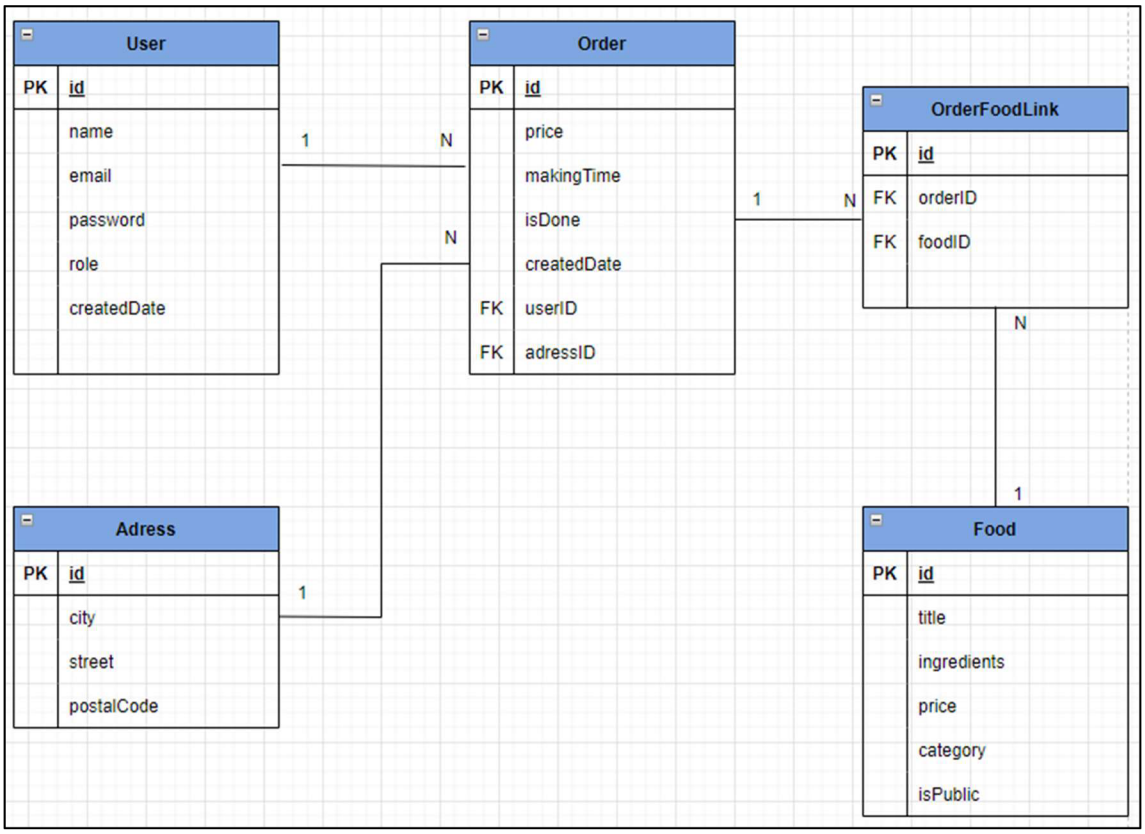
Posledním rozdělením, které lze na Obr. 14 zpozorovat je horizontální rozdělení v podobě černých úseček dělící obrázek na tři pomyslné bloky. Tyto bloky říkají, jaký se využije přístup pro vytvoření stránek spadajících do jednoho ze tří bloků, taktéž

sdělují, v jakém pořadí, tedy zezdola nahoru, budou pohledy vyvíjeny. Nejprve dostanou přednost stránky, kde se budou využívat možnosti knihovny ReactJS bez využití velkých knihoven třetích stran. Poté, kombinací ReactJS a Redux knihovny bude vyvíjen prostřední blok. Jako poslední budou vyráběny stránky přístupné pouze zaměstnancům, a to jak s předešlou kombinací technologií, tak navíc s nástrojem Redux ToolKit Query, dále jen RTK Query.

4.1.2 Databázové schéma

Díky předešlému obrázku jsou k dispozici, alespoň teoretické informace o propojenosti pohledů, jejich určení a dalších několik informací. Ještě před samotným začátkem analýzy či popisu implementace je výhodné se seznámit se strukturou databáze. K tomu slouží Obr. 15, na němž je vidět entitně vztahový diagram databáze. Projekt využívá SQLite jakožto relační databázový systém. Diagram obsahuje tabulky, mezi které patří User, Order, Address, OrderFoodLink a Food.

Tabulka User ukládá informace o všech uživatelích aplikace, uchovává jejich jméno, email, heslo a také roli, která určuje, zda se jedná o běžného uživatele nebo zaměstnance. Order udržuje informace o vytvořených objednávkách, tím se má na mysli cena, výrobní čas, datum vytvoření, uživatel, jenž objednávku vytvořil, nebo status, jestli je hotová, probíhající či zrušená. Tabulka Address slouží k zaznamenání adresy pro doručení objednávky. Informace o jídlech nabízených v rámci aplikace jsou v tabulce Food. Tabulka OrderFoodLink spojuje tabulky Order a Food, a slouží tak k ukládání informací o jídlech obsažených v jednotlivých objednávkách. Třeba poznamenat, že každá tabulka v této databázi bude mít primární klíč "id". Primární klíč slouží k jednoznačné identifikaci řádků v tabulce a zajišťuje, že každý řádek bude mít unikátní identifikátor. Tento způsob identifikace řádků umožňuje snadno vyhledávat, upravovat nebo mazat určité záznamy v databázi, a také propojovat řádky z různých tabulek pomocí cizích klíčů. Kupříkladu tabulka Order obsahuje cizí klíč „userID“ odkazující na „id“ uživatele, kterému objednávka patří, nadále obsahuje klíč „addressID“, což odkazuje na „id“ adresy, která byla použita pro doručení objednávky.



Obr. 15 ER Diagram databáze. Zdroj: [autor]

4.2 Základní pilíře aplikace

Při vytváření uživatelského rozhraní pomocí ReactJS se nejprve rozhraní rozdělí na části nazývané komponenty. Po dokončení jejich implementace následuje jejich propojení, typicky použitím jedné komponenty v rámci jiné a posíláním dat mezi nimi. Tyto samostatné komponenty obsahují logiku, stav a specifický vzhled pro danou část aplikace, častokrát bývají znovupoužitelné a usnadňují tím tvorbu a údržbu kódu. V této kapitole je popis prvních kroků projektu Restaurace, včetně vytvoření domovské stránky, navíc jsou vysvětleny základní principy vytváření webových aplikací s knihovnou ReactJS. Zdrojové kódy pro tuto část aplikace jsou k dispozici na odkazu gitového repozitáře v příloze 8.1.

4.2.1 JSX

JSX, zkratka pro JavaScript XML, je velmi užitečný nástroj pro vývojáře v ReactJS. Jedná se o rozšíření jazyka JavaScript, které poskytuje způsob, jak strukturovat vykreslování komponent pomocí syntaxe podobné HTML. JSX dává možnost

zjednodušeně vytvářet prvky HTML a umisťovat je do DOM bez nutnosti použití dalších JavaScriptových metod, jako jsou například createElement() nebo appendChild(). [2] Další využívanou vlastností JSX je schopnost poskytovat řadu atributů, které jsou navrženy tak, aby zrcadlily atributy poskytované jazykem HTML, navíc lze komponentě předat i vlastní atributy, a následně jsou všechny atributy dohromady komponentou přijaty jako tzv. „props“ proměnné, které si lze představit jako vstupní parametry funkce.

Na Obr. 16 je ukázkový příklad, kdy komponenta, tedy funkce, vrací hlavní kořenový element „div“ obsahující další vnořené komponenty, taktéž vytvořené syntaxí JSX. Platí, že názvy JSX komponent symbolizující HTML prvek začínají malým písmenem, kdežto všechny ostatní mají na začátku velké písmeno. Na obrázku níže jsou tedy k vidění celkem čtyři komponenty, ta hlavní s nastaveným názvem třídy na „app“, obsahuje kromě dvou běžných HTML prvků i vlastní komponentu s názvem „VlastniKomponenta“ přijímající jednu „props“ proměnnou se jménem „vlastniParametr“ a hodnotou „123“. Tato komponenta by byla uložena v samostatném JavaScript souboru s názvem „VlastniKomponenta.js“, kde by přijímala a zpracovávala jak vstupní parametry, tak veškerý obsah mezi otevírací a uzavírací značkou, v tomto případě řetězec „Hello World“.

```
return (
    <div className="app">
        <button onClick={handleClick}>Add food</button>
        <div className="food-list">{renderedFood}</div>
        <VlastniKomponenta vlastniParametr={123}>Hello World</VlastniKomponenta>
    </div>
)
```

Obr. 16 Kód s JSX. Zdroj: [autor]

ReactJS je založen na komponentách, přičemž existují dva druhy, prvním z nich jsou komponenty vytvořené pomocí třídy (z anglického Class Components) nebo pomocí funkce (z anglického Function Components). Třída udržuje svůj vlastní stav a definuje metody pro manipulaci s ním. Ve verzi React 16.8 byl představen princip tzv. „hooks“, který umožňuje manipulaci se stavem a kontextem i ve funkčních komponentách, a protože funkční komponenty v kombinaci s „hooks“ jsou jednodušší na pochopení i definici, je tento způsob pro vývoj v ReactJS preferován, stejně tomu je i v projektu Restaurace. Pro zjednodušení, lze o jakémkoliv „hooku“

uvažovat jako o obyčejné JavaScript funkci, které dokáže číst nebo měnit stavové proměnné aplikace. Vytvářet vlastní „hooky“ je možné, ale ReactJS sám o sobě nabízí mnoho zabudovaných ve své knihovně, které usnadní práci programátorovi. Zásadní a nejpoužívanější „hooky“ jsou „useState()“, „useEffect()“ nebo „useContext()“. Každý „hook“ by měl začínat slovem „use“, to platí jak u zabudovaných, tak i u vlastních. [4]

4.2.2 Virtuální DOM a useState

Stavová proměnná (z anglického State Variable) uchovává aktuální stav dané komponenty. Tento stav se může během běhu aplikace měnit, například v reakci na uživatelské interakce nebo změny vstupních dat. Když se stavová proměnná změní, automaticky dojde k znovu vykreslení komponenty. Díky „useState“, což je zabudovaný „hook“ v ReactJS, je možné stavové proměnné vytvářet a měnit. Při volání funkce „useState“ je vrácena proměnná, jejíž hodnota reprezentuje daný stav, taktéž je vrácena funkce, kterou lze později v komponentě použít na změnu, tudíž aktualizaci stavu. [4] Ukázka je na Obr. 17, kde je ve funkční komponentě vytvořena stavová proměnná s počáteční hodnotou 0, ke které je přistupováno skrze konstantu „cislo“. Pokud uživatel klikne na tlačítko, které tato komponenta vrací, bude v obsloužení této události hodnota stavové proměnné zvýšena o jedničku.

```
import {useState} from "react";

function VlastniKomponenta({ vlastniParametr }) {
    const [cislo, setCislo] = useState()


    const handleClick = (event) => {
        setCislo( value: cislo + 1)
    }

    return (
        <div>
            <span>Aktuální hodnota: {cislo}</span>
            <button onClick={handleClick}>Přičíst jedničku</button>
        </div>
    )

}
```

Obr. 17 Použití stavové proměnné. Zdroj: [autor]

ReactJS pro zvýšení efektivity překreslování komponent využívá svůj virtuální DOM, což je zjednodušeně abstraktní reprezentace skutečného DOM. Virtuální DOM je udržován v paměti a je aktualizován pouze tehdy, když se změní stavová proměnná komponenty. Pokud dojde ke změně, ReactJS porovná novou verzi s předchozí verzí, aby určil, které komponenty se musí aktualizovat, a následně provede pouze nejmenší možný počet DOM manipulací potřebných k aktualizaci uživatelského rozhraní. [29]

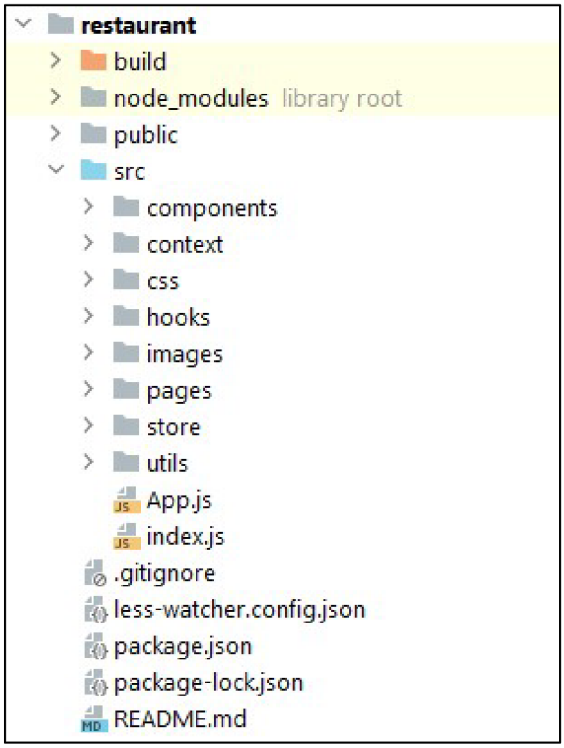
4.2.3 Struktura složek a komponent

Adresářová struktura projektu se dá vytvářet několika způsoby. Jedním z přístupů, jak rozdělovat soubory do adresářů je podle jejich typu, např. všechny JavaScriptové soubory s definicí znovupoužitelných komponent jsou vloženy do složky s názvem Komponenty. Dalším způsobem, jak členit soubory je podle jejich účelu, kupříkladu do složky Košík vložit všechny soubory týkající se košíku. V projektu Restaurace je zvolen způsob rozdělování souborů podle jejich typu, jak je možné vidět na Obr. 18.

Hlavní kořenový adresář v projektu obsahuje složku „build“, která je automaticky vytvářena v případě, že se spustí příkaz pro vygenerování kódu pro produkční použití, tuto složku je pak možné nahrát na server a její obsah poskytovat uživatelům. V „node_modules“ jsou nainstalovány všechny závislosti projektu a složka „public“ obsahuje mj. hlavní, v tomto projektu i jediný, HTML soubor. Nejdůležitější složka obsahující všechny JavaScriptové kódy, v níž se odehrává většina vývoje, nese název „src“ a obsahuje následující složky a soubory:

1. Složka obsahující soubory s komponentami, které jsou znovupoužitelné a využívají se v různých částech aplikace je pojmenovaná „components“. Tyto komponenty mohou být například tlačítka, formuláře nebo seznamy.
2. Kontext neboli globální stavové proměnné přístupné napříč celou aplikací mají obsluhující soubory v adresáři „context“. Tyto souboru pro dosažení své funkcionality používají „hook“, kterým je „useContext“.

3. CSS soubory jsou uloženy ve stejnojmenné složce. V tomto případě je obsah tvořen především soubory s příponou .less, ze kterých je později vygenerován jeden velký CSS soubor.
4. Vlastní vytvořené „hooky“ jsou uloženy ve složce „hooks“
5. Soubory s obrázky, které jsou použity v aplikaci, patří do složky „images“.
6. Složka „pages“ obsahuje soubory s komponentami, které slouží jako jednotlivé oddělené stránky aplikace. Tyto komponenty nejsou považovány jako znovupoužitelné.
7. V pozdějších fázích vývoje bude využívána knihovna Redux, jejíž kód náleží adresáři „store“. Použití této knihovny je vysvětleno později.
8. Pomocné funkce a nástroje napsané v obyčejném JavaScriptu nemanipulující se stavovými proměnnými jsou ve složce „utils“.
9. App.js soubor obsahuje hlavní komponentu aplikace, která je zodpovědná za renderování všech dalších komponent.
10. Soubor index.js obsahuje kód pro inicializaci aplikace a renderování hlavní komponenty v HTML dokumentu.



Obr. 18 Struktura souborů v projektu. Zdroj: [autor]

4.2.4 Vytvoření stránky v aplikaci

Při vývoji SPA aplikace v ReactJS je vhodné nejprve pracovat se statickým nebo alespoň s minimálně se měnícím obsahem. To znamená, že první vytvářené komponenty a stránky jsou ty s pevně daným obsahem, který se nemění na základě uživatelské interakce. Tento přístup umožňující rychle vytvořit základní strukturu a vizuální podobu stránky se zaměřuje na správné rozvržení a uspořádání jednotlivých prvků, a vytvoření prvních stylů pomocí CSS, případně Less.

Počátek vývoje s ReactJS je v souborech App.js a index.js, tyto jména se staly konvencí, kterou by měl dodržovat každý ReactJS projekt. V souboru index.js je definován kořenový element, do kterého bude aplikace vykreslovat, bývá jím zpravidla HTML „div“ element mívající atribut id s hodnotou „root“, o vytvoření hlavního HTML souboru je psáno v této práci později. App.js je soubor definující hlavní komponentu aplikace, do níž se postupně přidávají další komponenty a prvky. Obsah těchto dvou souborů je ukázán na Obr. 19, kód v nich zůstává téměř neměnný od začátku až do konce vývoje aplikace, jejich rozšíření nastane v momentě, kdy se přidává do aplikace kontext (např. knihovna Redux) nebo se implementuje logika směrování.

```
// ====================
// obsah souboru index.js
import App from './App';

const el = document.getElementById( elementId: 'root');
const root = ReactDOM.createRoot(el);

root.render(
    <App />
);

// ====================
// obsah souboru App.js
function App() {
    return (
        <HomePage />
    );
}

export default App;
```

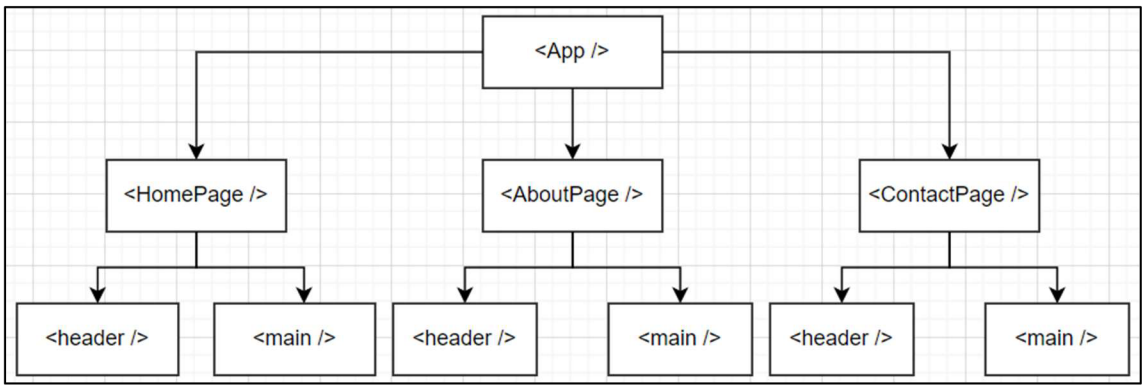
Obr. 19 Počátek projektu. Zdroj: [autor]

Jak plyne z textu výše, první vyvíjené stránky budou ty s neměnným obsahem a kde není potřeba interagovat s uživatelem. Díky předešlému představení pohledů aplikace (kapitola 4.1.1, na Obr. 14), lze předpokládat, že stránky splňující tyto požadavky jsou Domovská stránka, Kontakt a O nás.

V souboru HomePage.js je vytvořena komponenta se shodným názvem reprezentující Domovskou stránku, která obsahuje HTML elementy „header“ a „main“, stejně tak je budou obsahovat i ostatní stránky. Díky tomu je možné vytvářet kaskádové styly mnohem jednodušeji, protože stránka se tím pádem dělí vždy na dvě části. V souboru App.js je komponenta Domovská stránka naimportována a použita jako vnořená, jak si lze povšimnout na Obr. 19. Poté se postupně vytváří další potřebné komponenty a umísťují se do příslušných složek.

Na Obr. 19 je do hlavní komponenty „App“ přidána pouze HomePage, po otevření aplikace v prohlížeči, tak bude k vidění pouze její obsah. Stránky pro Kontakt a O nás jsou vytvořeny obdobně jako Domovská stránka, ovšem do hlavní komponenty budou přidány v momentě, kdy aplikace bude schopna zobrazovat jenom některé komponenty v závislosti na zadané URL adrese. Všechny tři vytvořené stránky

dodržují stejnou strukturu, z čehož plyne, že strom komponent aplikace vypadá jako na Obr. 20.



Obr. 20 Strom komponent. Zdroj: [autor]

4.3 Směrovač a přihlašovací proces

Webová aplikace Restaurace má v současném stavu vytvořené tři stránky, následujícím řešeným problémem je, že aplikace zatím nedokáže měnit zobrazovaný obsah. Bezprostředně poté musí být přidaná stránka, která s podporou backendu umožní přihlášení a registraci uživatele do aplikace. Oba tyto problémy jsou v této kapitole rozebrány a je k nim poskytnuta jak příslušná teorie, tak i ukázky kódu. Kompletně vypracovaný zdrojový kód k této fázi projektu je dostupný na URL adrese v příloze 8.2.

V aplikaci jsou tedy vytvořeny tři stránky, ale zatím webová aplikace neumí měnit svůj obsah na základě hodnoty v URL. Pro tento účel je nutné implementovat tzv. směrovač (z anglického Router), ten má za úkol zobrazovat žádané komponenty podle cesty URL bez nutnosti načítání celé stránky znovu. Tento způsob navigace zvyšuje uživatelskou přívětivost a zároveň se jedná o jednu z nejdůležitějších vlastností SPA. Veškerý obsah je stažen při prvním požadavku uživatele a je zobrazena komponenta vázaná na dotazovanou URL, všechny ostatní směrovač skryje. Při pohybu na stránce, který programově vyvolá změnu URL, tudíž adresa není zadána uživatelem ručně do adresního řádku, ale je změněna samotnou aplikací, směrovač zachytí změnu URL a aktualizuje zobrazovaný obsah. Všechny dynamické informace potřebné v zobrazené komponentě jsou staženy ze serveru pomocí AJAX požadavků na pozadí, tudíž bez kompletního znovu načtení.

Tento způsob změny obsahu je možný od příchodu standardu HTML5, jehož součástí je tzv. History API, díky němuž mohou programátoři manipulovat s historií URL adres prohlížeče. Často používané metody, které toto rozhraní nabízí, jsou „history.back()“ a „history.forward()“, jež simulují tlačítka dopředu a zpět v prohlížeči. Metoda „history.pushState()“ vkládá nový záznam URL do historie prohlížeče, aniž by zapříčinila znovu načtení stránky. [22]

Ač je možné vytvořit vlastní směrovač, existuje mnoho knihoven třetích stran, které pro směrování v aplikaci nabízejí vlastní produkt. V projektu Restaurace je použita knihovna React Router Dom, jejíž použití je na následujícím Obr. 21.

```
function App() {

    return (
        <>
            <BrowserRouter>
                <Routes>
                    <Route path="/" element={<Layout />}>
                        <Route index element={<HomePage />} />
                        <Route path="contact" element={<ContactPage />} />
                        <Route path="about" element={<AboutPage />} />

                        <Route path="*" element={<ErrorPage />} />
                    </Route>
                </Routes>
            </BrowserRouter>
        </>
    );
}

export default App;
```

Obr. 21 Směrovač aplikace. Zdroj: [autor]

Ve výše uvedeném kódu, kde je upravena komponenta App, se používají komponenty BrowserRouter, Routes a Route z knihovny React Router Dom. Pro fungování směrovače je nutné veškerou logiku směrování provádět uvnitř komponenty BrowserRouter. Určení, jaká komponenta se bude zobrazovat na konkrétní URL adrese probíhá vytvořením komponenty Route s dvěma atributy. Prvním z nich je „path“, jehož hodnota určuje, s jakou URL adresou bude komponenta svázána. Atribut „element“ definuje, jaká komponenta má být zobrazena.

4.3.1 Přihlášení a registrace

Díky funkčnímu směrovači může být každá nová stránka jednoduše zanesena do aplikace pomocí přidání komponenty Route. V této fázi vývoje následuje vytvoření přihlašovacího a registračního systému. Registrace do aplikace umožňuje uživatelům vytvořit si vlastní profil a může zobrazovat dynamická data spjatá s konkrétním uživatelem.

Vytváření funkčního přihlašovacího systému spočívá především v práci na backendu, protože pouze ten může ověřovat správnost zadaných údajů, neboť má na rozdíl od frontendu přístup do databáze. Existuje mnoho nástrojů, jak provádět autentizaci, Python Flask nabízí řešení v podobě tzv. session cookie, jež je využito i projektem Restaurace. Implementace může být provedena více způsoby, jeden z nich je použit v praktickém projektu a obnáší následující kroky:

1. Nejprve je vytvořena stránka obsahující formulář pro přihlášení uživatele. To je docíleno pomocí HTML formuláře s polem pro email a heslo uživatele.
2. Po vyplnění potřebných informací uživatelem jsou data převedena do formátu JSON a odeslány jako AJAX požadavek s metodou POST.
3. Backend přijme data a ověří, zda se uživatel s daným emailem nachází v databázi, a následně ověří shodu obou hesel.
4. V případě úspěchu v bodě 3, je id uživatele uloženo do cookie s názvem „session“ a posláno zpět klientovi, prohlížeč ji při získání odpovědi ze serveru automaticky ukládá do klientského zařízení. Všechny informace v session cookie jsou Flaskem šifrovány tajným klíčem, proto i kdyby někdo zjistil id jiného uživatele, není schopen ho manuálně podvodně uložit do session cookie, jelikož nezná klíč, kterým jsou data šifrována.
5. Poté, co je na frontendu přijatá kladná odpověď, je uživatel přesměrován z přihlašovací stránky na stránku hlavní.
6. Při každém požadavku, který uživatel pošle na server, je session cookie odeslána taktéž. Server před provedením jakékoliv akce nejprve rozšifruje id v session cookie a pokusí se takového uživatele nalézt v databázi, a poté, je-li úspěšný, má server jistotu, že požadavek provádí přihlášený uživatel.

Následně zkontroluje, zda na provedení příchozího požadavku má uživatel právo a případně mu vyhoví.

7. Pokud se uživatel rozhodne odhlásit, jsou ze session cookie odstraněny všechny informace. Toho lze docílit pomocí metody session.clear().

V momentě, kdy je přihlašovací proces naimplementován, zbývá vytvořit obdobnou stránku s registračním formulářem. Po odeslání registračních dat jsou tato data na serveru zkontrolována, uložena do databáze jako nový uživatel, a následně je nově vytvořený uživatel přihlášen již existujícím procesem. Při zdárném uložení a přihlášení zákazníka je poslána odpověď obsahující novou session cookie zpět klientovi, jenž provede přesměrování na hlavní stránku aplikace.

4.4 Objednávkový systém

Web Restaurace umožňuje sice uživatele přihlásit, ale při vytváření objednávkového systému, což je další úkol, je žádoucí udržovat informace o přihlášeném uživateli v objektu dostupném napříč celou aplikací. V této kapitole jsou představeny problémy spojené s přístupností stavových proměnných a jejich řešení v podobě definování kontextu aplikace. Všechen zdrojový kód týkající se této fáze je v gitovém repozitáři, jehož URL adresa je přiložena jako příloha 8.3.

Stavové proměnné jsou k dispozici pouze v rámci té komponenty, v níž jsou definovány, pokud k nim chtějí přistupovat přímý potomci, musí být tyto proměnné předány, jako „props“ parametry, rodičovskou komponentou. Není způsob, jak by mohl rodič přečíst nebo měnit stavovou proměnnou, definovanou v jeho potomkovi nebo dokonce v jiné nesouvisející komponentě.

Existuje-li množina komponent, které mají společné, že pracují se stejnou stavovou proměnnou, nabízí se způsob, vždy definovat proměnnou v prvním společném předkovi všech komponent v množině a přes potomky ji postupně rozdistribuovat do všech ostatních. Toto řešení by fungovalo, ale kód by se stal, ve většině případů, méně přehledný, kdyby stavová proměnná musela pouze procházet například skrze čtyři komponenty, než by se dostala k tomu pravému potomkovi, proto se tento způsob využívá především v případech, kdy je stavová proměnná dopravována přímému potomkovi. Ovšem tento problém je nutné vyřešit, protože

v aplikaci pravděpodobně existuje více proměnných, které je potřeba definovat globálně, aby k nim, v případě potřeby, měly přístup všechny komponenty. Řešením je přidat, pomocí ReactJS, kontext aplikace.

Kontext aplikace

Pokud je uživatel přihlášen a jsou od serveru získána jeho data, je žádoucí, aby byly uloženy globálně, a tak bude objekt s uživatelskými daty sdílen napříč celou aplikací se všemi komponentami. Kontext ke sdílení je umístěn do vlastního souboru ve složce „context“. Kód nutný k vytvoření kontextu je zachycen na Obr. 22.

```
import { createContext, useState } from 'react';
const UserContext = createContext();

function UserProvider({ children }) {
  const [user, setUser] = useState( initialState: {});

  return (
    <UserContext.Provider value={{ user, setUser }}>
      {children}
    </UserContext.Provider>
  );
}

export { UserProvider };
export default UserContext;
```

Obr. 22 Vytvoření kontextu. Zdroj: [autor]

Nejdříve je vytvořen, pomocí funkce createContext z knihovny ReactJS, objekt v proměnné UserContext. Tento objekt je klíčem k sdílení aktuálních dat o uživateli. [16] Kód na Obr. 22 sice kontext definuje, ale pro přístup k jeho hodnotě se musí provést dva další kroky:

1. Obalit hlavní komponentu App, nově vytvořenou komponentou UserProvider sloužící jako poskytovatel hodnoty.
2. Komponenta, v níž je vyžadován kontext, jej musí importovat a vložit jako parametr do zabudovaného „hooku“ useContext.

Oba tyto kroky jsou znázorněny na Obr. 23. Díky tomu, je zpřístupněna stavová proměnná „user“ a metoda pro její změnu „setUser“, jako kontext celé aplikace. Její

hodnota se v aplikaci nastavuje, jakmile jsou stažena uživatelská data. To se nejčastěji provádí ihned po prvním načtení stránky, pomocí AJAX požadavku.

```
// ====================
// 1. změna v souboru index.js
root.render(
    <UserProvider>
        <App />
    </UserProvider>
);

// ====================
// 2. změna v souboru HomePage.js
import { useContext } from 'react';
import UserContext from './context/user';

function HomePage() {
    const { user, setUser } = useContext(UserContext)
    console.log(user)

    // zbytek těla komponenty
```

Obr. 23 Použití kontextu. Zdroj [autor]

V úvodu kapitoly 4.1 byly popsány podmínky dané zadavatelem, vzhledem ke znalostem získaných z předešlých kapitol, zejména ohledně stavových proměnných a kontextu udržujícího uživatelská data, je možné vytvořit stránku s jídelním lístkem, včetně objednávkového formuláře. Zdrojový kód je k dispozici v příloze 8.3.

4.5 Redux a přihlášení uživatelé

Stránka s menu či objednávkovým systémem restaurace představovala poslední část při vytváření obsahu dostupného všem, tedy přihlášeným, nepřihlášeným nebo správcům aplikace. Následuje vývoj sekce určené ryze přihlášeným uživatelům, kde budou moci vidět historii svých objednávek a hlavně jejich stav. Navíc v této sekci bude připraven formulář pro případnou změnu uložených dat, například jméno, emailovou adresu nebo heslo. V této kapitole je představen populární nástroj Redux, včetně používaných principů při práci s ním a způsobu propojení s ReactJS aplikací. Stejně jako u předchozích fází, i k této je dostupný zdrojový kód online na adrese uvedené v příloze 8.4.

4.5.1 Redux

Na rozdíl od předešlých implementovaných částí aplikace je tato poprvé vyvíjena s využitím nástroje Redux s knihovnou Redux Toolkit. ReactJS a Redux jsou velmi dobrou kombinací pro správu všech stavových proměnných v aplikaci, zejména při vytváření velkých projektů. Ovšem, složitý proces konfigurace Reduxu se stal pro mnoho vývojářů velkým problémem, proto vznikla sada nástrojů, pro efektivní a jednoduchou práci s Redux knihovnou, nazvaná Redux-Toolkit, jejíž použití je silně doporučované při rozhodnutí využít knihovnu Redux. [7]

Díky Reduxu, má aplikace jediné úložiště pro všechna data nebo stavové proměnné, ke kterým se komponenty potřebují dostat, což má za následek, že není nutné posílat informace skrze „props“ proměnné přes několik komponent. Nutno zmínit, že Redux není náhradou za používání „hooků“ useState nebo useContext, jež byly vysvětleny v předešlých kapitolách, stále je v určitých případech vhodné, aby byly užívány současně s knihovnou Redux. Tato knihovna umožňuje pomocí kódu přistupovat k úložišti, které je globálně přístupné jakékoli komponentě. Úložiště zařizuje, aby data byla reaktivní pro ostatní komponenty, podobně jako tomu je u běžné stavové proměnné. Za předpokladu, že došlo ke změnám dat v úložišti, pak každá komponenta, která tato data používá, znovu vykreslí ovlivněnou část DOM objektu, tím pádem se změny projeví v uživatelském rozhraní bez ohledu na to, jak hluboko je úroveň komponenty. Pro hlubší porozumění správy dat v Reduxu, před jeho použitím v kódu, následuje vysvětlení základních pojmů, týkajících se jeho fungování:

1. Dispečer (z anglického Dispatcher) je ten, kdo začíná akce pro změnu dat v úložišti. Funkce „dispatcher“ odesílá, respektive zprostředkovává akci reduktoru.
2. Reduktor (z anglického Reducer) má výlučné právo upravovat data uvnitř úložiště. Většinou je vytvořeno více reduktorů, kde každý je zodpovědný za změny týkající se konkrétní části úložiště.
3. Pojmem úložiště (z anglického store) je myšlen globální objekt uchovávající stavové proměnné na jednom místě, a zároveň jejich hodnoty udržuje synchronizované s komponentami.

4. Selektory jsou funkce, jež uvnitř komponenty získají požadovanou část dat z úložiště. Uvnitř komponenty není totiž dovoleno importovat celé úložiště.

Redux Toolkit je jednotný způsob používání Reduxu, protože složitost samotného Reduxu měla za následek, že vzniklo spoustu možných vlastních cest, jak s ním pracovat. Před vytvořením sady nástrojů Redux Toolkit, neexistoval žádný standardní návod pro implementaci nebo konfiguraci Reduxu v ReactJS aplikaci. Redux Toolkit, již obsahuje předinstalovaný Redux nebo například knihovnu "Immer", sloužící pro jednoduší zacházení se stavovými proměnnými. [7]

4.5.2 Vývoj s knihovnami ReactJS a Redux

Propojení ReactJS a Redux v projektu je usnadněno použitím knihovny „React-Redux", která zařizuje napojení projektu na Redux úložiště jednoduchou cestou. Konkrétně je z této knihovny importována komponenta s názvem „Provider" a obalí se s ní vše uvnitř funkce root.render(). Jak je možné vidět na Obr. 24 níže, v projektu Restaurace jsou obaleny stávající komponenty App i UserProvider poskytující kontext uživatele. Komponentě Provider je navíc předán jako „props" parametr importované Redux úložiště. Těmito kroky je zajištěno, že ReactJS a Redux úložiště jsou propojené a je umožněna komunikace mezi nimi.

```
// ====================
// Změna v hlavním souboru index.js
import { Provider } from "react-redux";
import { store } from "./store";

root.render(
    <Provider store={store}>
        <UserProvider>
            <App />
        </UserProvider>
    </Provider>
);
```

Obr. 24 Propojení ReactJS s Redux. Zdroj: [autor]

Při vytváření úložiště je důležité správně definovat reduktory, kteří budou s daty pracovat, jak už bylo psáno výše, těchto reduktorů může být více, a tak je jejich

definice rozmístěna do samostatných souborů. Z těchto souborů jsou poté importovány do hlavního Redux úložiště, odkud jsou poté přístupné aplikaci. Reduktory se vytvářejí v rámci tzv. dílů (z anglického slices), každý díl obsahuje jméno, sloužící pro identifikaci, počáteční hodnotu části úložiště, kterou díl má na starosti a v neposlední řadě i všechny potřebné reduktory, což jsou funkce, manipulující s daty příslušících danému dílu.

V případě projektu Restaurace bude Redux úložiště tvořeno dvěma díly, první pracuje s daty týkající se stránky pro uživatelskou historii objednávek, druhý obstarává režii stránky pro změnu uživatelských dat. Konkrétní implementace a konfigurace Redux dílů je rozebrána v kapitole 5.3.

4.6 Automatizace správcovské části s Redux Toolkit Query

Přichází na řadu ještě více využít potenciál knihovny Redux Toolkit, při vytváření sekce pro zaměstnance, kteří jsou v roli správců objednávek a uživatelů. První stránka musí nabízet přehled všech objednávek, jejich obsah, adresu doručení, celkovou cenu, informace o průběhu objednávky a také důležitá data o uživateli. Nejen, že zaměstnanec vidí status objednávky je nutné, aby ho dokázal i měnit. Další možnost pro zaměstnance je otevřít nebo uzavřít restauraci, podle toho je zákazníkům buď povoleno či zakázáno vytvářet objednávky. Druhá stránka nacházející se v této sekci slouží pro vypsání všech uživatelů webové aplikace s možností jednotlivé uživatele odstranit, v tom případě budou odstraněny z databáze všechny záznamy s ním spojené.

Z požadavků na tuto sekci je zřejmé, že zde půjde především o komunikaci mezi frontendem a backendem pomocí několika AJAX požadavků a následném zpracování dat pomocí nástroje Redux Toolkit, jelikož stejně jako předchozí sekce i tato bude využívat Redux. Rozdíl je, ale ve způsobu získávání dat ze serveru, do této chvíle byly všechny data přenášeny pomocí manuálně vyvolaných AJAX požadavků, nicméně tato sekce bude využívat kromě klasického Redux Toolkit i jeho doplněk Redux Toolkit Query, dále jen RTK Query. Jedná se o finální fázi vývoje projektu, jejíž zdrojový kód je dostupný na odkazu v příloze 8.5.

4.6.1 Redux Toolkit Query

Dle dokumentace je RTK Query velmi efektivní nástroj pro načítání a ukládání dat ze serveru do mezipaměti. Je navržen tak, aby zjednodušil a zautomatizoval běžné případy načítání dat ve webové aplikaci, čímž eliminuje nutnost ručně psát vlastní logiku. [32] RTK Query se inspiruje u jiných nástrojů, které jsou vytvořeny za účelem načítání dat, jeho charakteristika je následující:

1. Logika načítání dat a ukládání do mezipaměti se definuje s jistou podobností jako Redux díly, jež byly vysvětleny v předešlé kapitole.
2. URL adresy endpointů serveru jsou definovány předem, včetně procesu generování případných URL parametrů, těla požadavku či následné zpracování odpovědi ze serveru.
3. RTK Query může také generovat ReactJS „hooky“, které zapouzdřují celý proces načítání dat, díky tomu je v komponentách umožněno pracovat s informacemi o konkrétním stavu procesu získávání dat nebo s daty samotnými. [32]

V předchozí fázi byly pomocí funkce createSlice() vytvářeny jednotlivé Redux díly obstarávající stavové proměnné v úložišti, v této fázi za účelem integrace RTK Query do projektu, je použita funkce createApi() pro vyrábění tzv. RTK Query Api. Stále platí, že všechny reduktory ať už vytvořené pomocí createSlice() nebo pomocí createApi() musí být importovány do úložiště. Protože je z předešlých fází Redux úložiště nakonfigurované a propojené s ReactJS projektu Restaurace, pak není nutné tento proces provádět znovu za účelem integrace RTK Query. Definice Redux úložiště v projektu Restaurace s již importovanými všemi díly i RTK Query Api je znázorněna na .

```
const store = configureStore( options: {
  reducer: {
    userOrders: userOrdersReducer,
    userData: userDataReducer,
    [adminOrdersApi.reducerPath]: adminOrdersApi.reducer,
    [adminUsersApi.reducerPath]: adminUsersApi.reducer,
    [adminRestaurantApi.reducerPath]: adminRestaurantApi.reducer
  },
  middleware: (getDefaultMiddleware : … ) => {
    return getDefaultMiddleware() …
        .concat(adminOrdersApi.middleware) …
        .concat(adminUsersApi.middleware) …
        .concat(adminRestaurantApi.middleware);
  }
});
```

Obr. 25 Konfigurace Redux úložiště. Zdroj: [autor]

Na obrázku výše je možné si povšimnout dvou Redux dílů „userOrdersReducer“ a „userDataReducer“, které byly definovány v předešlé fázi vývoje. V této fázi přibyly celkem tři importy RTK Query Api do Redux úložiště, které se zabývají stahováním dat přístupných pro správce webové aplikace, konkrétně o všechny informace a operace spojené s objednávkami, uživateli i provozu restaurace. Je důležité upozornit, že všech pět importovaných objektů dodává do úložiště své reduktory, protože jenom s jejich pomocí, lze měnit uchovávaný obsah. Položka „middleware“, říká, že definované RTK Query Api nevyžadují v průběhu dotazu provádět speciální akce a je tedy použita výchozí hodnota. Podrobné definice jednotlivých RTK Query Api jsou dále rozvedeny v kapitole 5.4.

5 Implementace projektu

Předešlá kapitola 4 pokrývá svým obsahem návrh aplikace Restaurace a z velké části i její implementaci, nicméně v této kapitole je detailně popsán implementační proces některých vybraných úseků kódu, kterým nebyla zatím věnována velká pozornost, ale i přesto se jedná o důležité části pro správný chod aplikace či pro vytvoření kvalitního uživatelského zážitku. Pro každou následující podkapitolu platí, že zdrojový kód pro frontendové řešení daného problému je zahrnuto ve finální verzi projektu k nalezení na odkazu v příloze 8.5. Zdrojový kód backendu je přístupný v příloze 8.6.

5.1 useEffect

Jeden z nejčastěji používaných „hooků“ o kterém se zatím v předešlých kapitolách nemluvilo je useEffect. V této podkapitole je vysvětleno, co je useEffect a jak usnadní vývoj webové stránky.

Pokud se změní stavová proměnná je ovlivněná komponenta znovu vykreslena, jinými slovy, je vždy proveden kód uvnitř funkce definující komponentu. Existují činnosti, které programátor nechce provádět při každém znovu vykreslení, zvlášť když ReactJS vykresluje komponenty velmi často. Takové činnosti jsou nejčastěji dotazování se na data od serveru, které jsou potřeba získat pouze jednou, a to při prvním zobrazení komponenty. Konkrétně, když zákazník klikne na odkaz v navigaci vedoucí na stránku s menu, tak se získávají od serveru potřebná data co nejdříve a pouze při prvním načtení komponenty, nikoliv pokaždé když proběhne na stránce s menu jakákoliv nesouvisející změna. [8] Reálný případ použití funkce useEffect z projektu Restaurace je na následujícím Obr. 26.

```
function MenuPage() {

    const [food, setFood] = useState( initialState: []);

    useEffect( effect: () => {
        async function fetchFood() {
            const response = await makeRequest( url: "/food/")
            const foodWithCount = await response.data.map((food) => {
                if (!food.count) food.count = 0;
                return food
            })

            setFood(foodWithCount)
        }
        fetchFood()
    }, deps: [])

    // pokračování funkce
```

Obr. 26 Ukázka použití useEffect. Zdroj: [autor]

Předešlý kód se týká získání jídelního lístku ze serveru při načtení stránky, protože parametrem pro useEffect musí být synchronní funkce, je nutné případný asynchronní kód vložit do vnořené funkce uvnitř useEffect a tu i uvnitř zavolat. Uvnitř asynchronní funkce je pomocí AJAX požadavku získán jídelní lístek, který je uložen do stavové proměnné „food“ pomocí funkce „setFood“.

5.2 Navigace

V kapitole 4.3 je již popsáno proč je důležité využívat směrovač a měnit URL adresu programově. Tradiční řešení pomocí HTML elementů „a“, nebo pomocí JavaScriptu změnou globální proměnné „window.location.href“, není přípustné, jelikož oba tyto způsoby mají za následek kompletní znovu načtení stránky. V aplikaci Restaurace je mnohokrát využíváno přesměrování uživatele pomocí JavaScriptu, nikoliv však změnou proměnné „window“, ale s využitím „hooků“ či komponent dodané knihovnou React Router Dom, která byla zároveň použita pro výrobu směrovače.

Pro korektní přesměrování uživatele v souladu s myšlenkou SPA, jsou z knihovny React Router Dom využívány především komponenta „Link“ a „hook“ useNavigate. Oba způsoby využívá v projektu kupříkladu komponenta „Navigation“ obsahující odkazy, skrze které je možné se prokliknout na požadovanou stránku. Navigace

navíc obsahuje i logo restaurace fungující taktéž jako odkaz směřující na domovskou stránku.

```
const handleClick = () => {
  if (pathname !== "/") {
    navigate("/")
  }
}

return (
  <nav>
    <img id="main-logo" src={Logo} alt="Pizza Python Logo" onClick={handleClick} />
    <div id="nav-links">
      { renderedLinks }
      <FaFacebookSquare />
      <FaInstagramSquare />
    </div>
  </nav>
);
}
```

Obr. 27 Přesměrování uživatele. Zdroj: [autor]

Obr. 27 předkládá jedno z možných řešení, jak požadované funkcionality implementovat. V elementu „nav“ je umístěn obrázek s logem („<img>“), který má vytvořeného posluchače na událost kliknutí, jakmile se tak stane, posluchač přesměruje uživatele funkcí „navigate“ získané z „hooku“ useNavigate na URL adresu domovské obrazovky.

Proměnná „renderedLinks“, obsahuje pole komponent „Link“, které je vytvořeno na začátku funkce, důležitá informace je, že každá „Link“ komponenta musí přijímat „props“ parametr s názvem „to“, jehož hodnotou se nastavuje cílová destinace. Komponenta „Link“ je ekvivalent HTML elementu „a“ s tím rozdílem, že nezapříčiní kompletní znovu načtení.

5.3 Redux díly

Co je Redux úložiště a jak probíhá jeho propojení s ReactJS projektem bylo popsáno v kapitole 4.5.2. Zároveň bylo nastíněno, co jsou tzv. Redux díly. Příklad konkrétní definice Redux dílu a jeho konfigurace je na Obr. 28. Díl se vytváří pomocí funkce createSlice, která přijímá parametr, kterým je relativně obsáhlý objekt obsahující jméno dílu („name“), počáteční hodnotu objektu stavových proměnných („initialState“) a hlavně reduktory („reducers“). Obr. 28 obsahuje definici dílu

starajícího se o všechny stavové proměnné na stránce pro změnu uživatelských dat. Reduktor „setUserData“, stejně jako každý jiný, přijímá dva parametry:

1. Parametr „state“, což je automaticky vložený objekt reduktorem, představuje objekt všech stavových proměnných, které tento Redux díl spravuje. Počáteční hodnota tohoto objektu je nastavena v poli „initialState“.
2. Parametr „action“ reprezentuje tzv. objekt akce (z anglického action object). [7]

```
const userDataSlice = createSlice( options: {
  name: 'userData',
  initialState: {
    nameInput: "",
    emailInput: "",
    passwordInput: "",
    passwordRepeatInput: ""
  },
  reducers: {
    setUserData(state, action) {
      // action.payload === User Object
      state.nameInput = action.payload.name
      state.emailInput = action.payload.email
    },
    setFormValue(state, action) {
      // action.payload === ["nameInput", "newValue"]
      state[action.payload[0]] = action.payload[1]
    }
  },
});

export const { setUserData, setFormValue } = userDataSlice.actions;
export const userDataReducer = userDataSlice.reducer;
```

Obr. 28 Uživatelská data v Redux úložišti. Zdroj: [autor]

Funkci reduktoru „setUserData“, prostřednictvím dispečera, je možné zavolat z jakékoliv komponenty příkazem „dispatch(setUserData(vlastniPromenna))“, dispečer poté najde požadovanou funkci reduktoru, vloží do parametru „state“ objekt se stavovými proměnnými a do parametru „action“ pod klíč „payload“ vloží obsah hodnoty „vlastniPromenna“, reprezentující data vložená při volání reduktoru.

5.4 Redux Toolkit Query Api

Proč se používá RTK Query bylo nastíněno v kapitole 4.6.1, zde bude podrobněji vysvětleno, jak se vytváří RTK Query Api, které komponentám dovoluje jednoduše přistupovat k datům získaných ze serveru. RTK Query Api se definují obdobně jako Redux díly, rovněž v samostatných souborech odkud se poté importují do Redux úložiště. Funkce vytvářející tyto Api se jmenuje createApi a přijímá obsáhlý objekt s konfigurací, kde je například definováno jméno reduktoru, URL adresa serveru a hlavně tzv. endpointy. Je dobré zmínit, že i když se uvnitř konfigurace nedefinují reduktory manuálně, jak tomu je u Redux dílů, tak jsou automaticky vytvořeny za programátora, aby bylo možné je poté předat Redux úložišti.

Endpoint je objekt, který mj. definuje, jak má vypadat požadavek na server. Příklad, kde se požadavkem mění stav otevření restaurace je na Obr. 29.

```
restaurantOpened: builder.mutation( definition: {
  query: (opened : QueryArg ) => {
    return {
      url: `/restaurant-opened`,
      method: 'PUT',
      body: {
        opened: opened,
      },
    };
  }})
```

Obr. 29 RTK Query Api endpoint. Zdroj: [autor]

Endpointy se dají rozdělit na dva typy:

1. Typ dotaz (z anglického queries) je nejčastější případ použití RTK Query. Operaci dotazování dat se doporučuje používat pouze na požadavky získávající data.
2. Cokoli, co mění data na serveru nebo případně smaže obsah v mezipaměti, je bráno jako typ mutace. [32]

5.4.1 Použití RTK Query

RTK Query Api pracující s daty ze serveru ohledně toho, jestli je restaurace zavřená nebo otevřená, má v sobě definované dva endpointy, jeden je typu dotaz s názvem „fetchRestaurantOpened“ a druhý je typ mutace pojmenovaný

„restaurantOpened“. Pro použití v komponentách je nutné znát správný název automaticky vytvořených „hooků“ pro jednotlivé endpointy. Endpoint s názvem „fetchRestaurantOpened“ je dostupný pomocí „useFetchRestaurantOpenedQuery“, zatímco endpoint „restaurantOpened“ má vytvořen „hook“ se jménem „useRestaurantOpenedMutation“. Použití těchto nových „hooků“ je na Obr. 30.

```
const { data: restaurantOpened } = useFetchRestaurantOpenedQuery()
const [ setRestaurantOpened ] = useRestaurantOpenedMutation()
```

Obr. 30 Vygenerované „hooky“ endpointů. Zdroj: [autor]

Kdekoliv bude zavolána funkce „useFetchRestaurantOpenedQuery“, tak budou automaticky stažena data ze serveru. Funkce „useRestaurantOpenedMutation“ vrací funkci po jejímž zavolání kdekoliv v kódu se spustí proces mutace dat na serveru.

5.4.2 Tag systém

Po získání dat ze serveru pomocí RTK Query je odpověď uložena do mezipaměti, to má za následek, že pokud v libovolné komponentě bude ten samý endpoint volán znovu, RTK Query mu ihned vrátí data, jež má uložené v mezipaměti, protože je považuje za aktuální. Existují situace, kdy je nutné po mutaci dat na serveru označit data uložená v mezipaměti jako neaktuální a při první příležitosti stáhnout nová aktuální data ze serveru. Tento problém řeší tzv. tag systém. [33]

Tagy jsou pouze názvem, kterým lze označit přijatá data. Uvnitř endpointu lze definovat, jak se případná data označí. Například v situaci, kde je stažen seznam všech uživatelů, bude seznam označen řetězcem „Uživatelé“. V případě, že jeden ze zaměstnanců ve webové aplikaci smaže libovolného uživatele je provedena mutace dat a zároveň je tag „Uživatelé“ znehodnocen, tím dojde ke znovu stažení seznamu uživatelů, aktualizace stavové proměnné v Redux úložišti, a nakonec samotné znovu vykreslení komponenty s novými daty.

5.5 Přihlášení s OAuth

V případě webové aplikace, kde je nutnost pracovat s přihlášenými uživateli, existuje tradiční řešení, kterým je zajistit vlastní přihlašovací systém včetně databáze. To ovšem může být náročné jak časově pro vývojáře, tak finančně pro majitele služby. V dnešní době je stále více populární přihlašovat se do aplikace

pomocí OAuth neboli třetí strany, tou bývá často Google, GitHub anebo Seznam. Majiteli podniku odpadne starost s vytvářením přihlašovacího systému a uživatel není nucen k zapamatování si dalších přihlašovacích údajů. [44]

V projektu Restaurace je zpřístupněno OAuth přihlášení pomocí Google. Nejprve je nutné od Googlu získat ID klienta pro projekt, vytvořením OAuth aplikace na webových stránkách https://console.cloud.google.com/. Pro integraci OAuth přihlášení přes Google v ReactJS slouží knihovna @react-oauth/google. Importování z této knihovny komponentu „GoogleOAuthProvider“ a obalení s ní kód v souboru index.js je první krok, zobrazený na Obr. 31.

```
// =================
// změna v hlavním souboru index.js
root.render(
    <GoogleOAuthProvider clientId="266707545030-0gbhh4c061lum5goqboj********.apps.googleusercontent.com">
        <Provider store={store}>
            <UserProvider>
                <App />
            </UserProvider>
        </Provider>
    </GoogleOAuthProvider>
);
```

Obr. 31 Google OAuth komponenta. Zdroj: [autor]

Integrace přihlašování Google s aplikací umožňuje přístup k profilu uživatele, který obsahuje jeho jméno, e-mailovou adresu, obrázek, přístupový kód a podobně. Tyto informace jsou použity k vytvoření účtu zákazníka. Pomocí importovaného „hooku“ z knihovny @react-oauth/google je definováno jaké funkce se zavolají v případě úspěchu či neúspěchu. Tento „hook“ vrací funkci, která je volána v posluchači na tlačítku pro přihlášení pomocí Google, jak je možné vidět na Obr. 32 níže. Poté co uživatel klikne na tlačítko je otevřeno známé okno pro přihlášení od Googlu, v případě úspěchu je získán přístupový kód a ten se použije v navazujícím požadavku pro získání dat o uživateli ze serverů Googlu. [14]

```
const googleLogin = useGoogleLogin( options: {
    onSuccess: handleGoogleSuccess,
    onError: handleGoogleError
});

return (
    <>
        <Header title="PŘIHLÁŠENÍ"></Header>
        <main className="login">
            <LoginForm />
            <div id="google-oauth">
                <button onClick={() => googleLogin()} >PŘIHLÁSIT POMOCÍ GOOGLE <FcGoogle/></button>
            </div>
            <Link to="/register">NEMÁTE ÚČET?</Link>
        </main>
    </>
)
```

Obr. 32 Google OAuth „hook“. Zdroj: [autor]

Pokud data o uživateli byla v pořádku získána a nenastal žádný problém, pak v projektu Restaurace je prováděn následující postup:

1. Jméno, email a Google ID je z odpovědi převzato a posláno na backend aplikace jako pokus o přihlášení.
2. Backend nejprve prohledá databázi, zda v ní existuje uživatel se stejným emailem a Google ID. Pokud byl takový uživatel nalezen, pak je přihlášen a frontendu je umožněno získat o něm data z databáze s ním spojená.
3. Pokud přechozí krok nebyl úspěšný, musí být vytvořen nový uživatel. Registrace Google uživatele probíhá stejně jako běžná, s tím rozdílem, že není nastaveno žádné heslo, ale je uloženo Google ID, jelikož to je neměnný identifikující údaj uživatele.
4. Uživatelé, kteří byli vytvořeni pomocí Google OAuth, se nemohou přihlašovat běžným formulářem, ale jsou vyzváni k použití přihlášení přes Google.

6 Závěr

Po vysvětlení teoretických základů pro vývoj webových aplikací následovalo představení zadaní praktického projektu této práce. Projekt zpracovával fiktivní požadavek pro vytvoření uživatelsky přívětivé webové aplikace restauračního zařízení, která by měla integrovaný objednávkový systém a umožňovala zaměstnancům jej spravovat. Implementace této aplikace je rozdělena do pěti fází a slouží jako výukové materiály pro studenty předmětu Technologie publikování na Webu. Zdrojové kódy jednotlivých vývojových fází jsou dostupné v online gitovém repositáři, jehož URL adresa je vždy uvedena v příslušné kapitole. Hlavní použitou technologií pro zhotovení klientské části webových stránek byla knihovna ReactJS. Studentům byl navíc představen nástroj Redux, který se se zmíněnou knihovnou často kombinuje.

Tato diplomová práce může sloužit jako edukační materiál nejen studentům zmíněného předmětu, ale obecně jakémukoliv studentovi informatiky zabývajícím se vývojem klientské části moderních webových stránek. Tato práce ve svých materiálech poskytuje plně funkční backend i relační databázi, aby se student mohl zaměřit především na frontend část a zároveň, aby se seznámil s tokem informací v rámci webové aplikace.

7 Seznam použité literatury

[1] Angular. *Angular - Introduction to the Angular docs* [online]. [vid. 2023-04-09]. Dostupné z: https://angular.io/docs

[2] ARANCIO, Stephen. What is JSX? *Medium* [online]. 5. říjen 2021 [vid. 2023-04-20]. Dostupné z: https://medium.com/@sjarancio/what-is-jsx-e3dda0af3490

[3] BHALLA, Archana, Shivangi GARG a Priyangi SINGH. PRESENT DAY WEB-DEVELOPMENT USING REACTJS. 2020, **07**(05).

[4] BUGL, Daniel. *Learn React Hooks: build and refactor modern React.js applications using Hooks*. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1-83864-051-4.

[5] DIGITAL, Wolfpack. SPA/CSR, SSR, SSG - Demystifying Rendering Modes. *https://wolfpack-digital.com* [online]. 23. září 2022 [vid. 2023-04-04]. Dostupné z: https://www.wolfpack-digital.com/blogposts/spa-csr-ssr-ssg-demystifying-rendering-modes

[6] Django Project. *Django* [online]. [vid. 2023-04-11]. Dostupné z: https://www.djangoproject.com/

[7] DULDULAO, Devlin Basilan a Ruby Jane Leyva CABAGNOT. *Practical enterprise React: become an effective react developer in your team*. United States: Apress, 2021. ISBN 978-1-4842-6975-6.

[8] ELROM, Elad. *React and Libraries: Your Complete Guide to the React Ecosystem* [online]. Berkeley, CA: Apress, 2021 [vid. 2023-04-26]. ISBN 978-1-4842-6695-3. Dostupné z: doi:10.1007/978-1-4842-6696-0

[9] FAVOUR C., Felix. How Web Pages Get Rendered on the Browser – Different Methods Explained. *freeCodeCamp.org* [online]. 26. říjen 2022 [vid. 2023-04-04]. Dostupné z: https://www.freecodecamp.org/news/web-page-rendering-on-the-browser-different-methods/

[10] GHIMIRE, Devndra. *Comparative study on Python web frameworks: Flask and Django* [online]. Helsinky, Finsko, 2020. Bachelor's Thesis. Metropolia University of Applied Sciences. Dostupné z: https://urn.fi/URN:NBN:fi:amk-2020052513398

[11] GOYAL, Shubham a Deepanshu BATRA. Angular JS. *International Research Journal of Modernization in Engineering Technology and Science* [online]. 2022, **4**(6). ISSN 25825208. Dostupné z: https://www.irjmets.com/uploadedfiles/paper/issue_6_june_2022/26286/final/fin_irjmets1655469952.pdf

[12] HOFFMAN, Andrew. *Web application security: exploitation and countermeasures for modern web applications*. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020. ISBN 978-1-4920-5311-8.

[13] IBM. *IBM Documentation* [online]. 13. říjen 2021 [vid. 2023-04-15]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests

[14] INNOCENT, Chimezie. The guide to adding Google login to your React app. *LogRocket Blog* [online]. 9. srpen 2022 [vid. 2023-04-25]. Dostupné z: http://blog.logrocket.com/guide-adding-google-login-react-app/

[15] JetBrains. *The State of Developer Ecosystem in 2022 Infographic* [online]. červen 2022 [vid. 2023-04-08]. Dostupné z: https://www.jetbrains.com/lp/devecosystem-2022

[16] LARSEN, John R. *React Hooks in action: with suspense and concurrent mode*. Shelter Island: Manning, 2021. ISBN 978-1-61729-763-2.

[17] Less CSS. *Getting started | Less.js* [online]. [vid. 2023-04-09]. Dostupné z: https://lesscss.org/

[18] LI, Nian a Bo ZHANG. The Research on Single Page Application Front-end development Based on Vue. *Journal of Physics: Conference Series* [online]. 2021, **1883**(1), 012030. ISSN 1742-6588, 1742-6596. Dostupné z: doi:10.1088/1742-6596/1883/1/012030

[19] MDN Web Docs. *Introduction to progressive web apps - Progressive web apps (PWAs) | MDN* [online]. 5. duben 2023 [vid. 2023-04-08]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Introduction

[20] MDN Web Docs. *Asynchronous - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [online]. 21. únor 2023 [vid. 2023-04-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Asynchronous

[21] MongoDB. *SQLite Vs MongoDB* [online]. [vid. 2023-04-13]. Dostupné z: https://www.mongodb.com/compare/sqlite-vs-mongodb

[22] NYAMADOR, Desmond. Using React with the History API. *PluralSight* [online]. 17. září 2020 [vid. 2023-04-21]. Dostupné z: https://www.pluralsight.com/utilities/promo-only?noLaunch=true

[23] OZA, Shyam. SQL Injection Attacks — Web-based App Security, Part 4. *Spanning* [online]. 18. červenec 2019 [vid. 2023-04-27]. Dostupné z: https://spanning.com/blog/sql-injection-attacks-web-based-application-security-part-4/

[24] PalletsProjects. *Flask* [online]. [vid. 2023-04-10]. Dostupné z: https://palletsprojects.com/p/flask/

[25] PARTIDA, Devin. Top Open Source Companies 2023. *Datamation* [online]. 20. březen 2023 [vid. 2023-04-03]. Dostupné z: https://www.datamation.com/open-source/35-top-open-source-companies/

[26] PEDAMKAR, Priya. SASS vs LESS | Top 6 Most Useful Differences To Learn. *EDUCBA* [online]. 9. listopad 2018 [vid. 2023-04-09]. Dostupné z: https://www.educba.com/sass-vs-less/

[27] POSTMA, Brittney. The Acronyms of Rendering on the Web. *Netlify* [online]. 31. květen 2022 [vid. 2023-04-04]. Dostupné z: https://www.netlify.com/blog/the-acronyms-of-rendering/

[28] Ramotion. *The Role of Database in Web Application Development* [online]. 9. srpen 2022 [vid. 2023-04-13]. Dostupné z: https://www.ramotion.com/blog/database-in-web-app-development/

[29] RAVICHANDRAN, Adhithi. React Virtual DOM Explained in Simple English. *Medium* [online]. 31. březen 2023 [vid. 2023-04-20]. Dostupné z: https://adhithiravi.medium.com/react-virtual-dom-explained-in-simple-english-fc2d0b277bc5

[30] React. *React* [online]. [vid. 2023-04-09]. Dostupné z: https://react.dev/

[31] Red Hat. *What is a REST API?* [online]. 8. květen 2020 [vid. 2023-04-15]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[32] Redux Toolkit. *RTK Query Overview | Redux Toolkit* [online]. 24. červen 2022 [vid. 2023-04-25]. Dostupné z: https://redux-toolkit.js.org/rtk-query/overview

[33] Redux Toolkit. *Automated Re-fetching | Redux Toolkit* [online]. 18. duben 2023 [vid. 2023-04-27]. Dostupné z: https://redux-toolkit.js.org/rtk-query/usage/automated-refetching

[34] SHARIF, Arfan. CRUD vs REST: Similarities & Differences Explained - CrowdStrike. *Crowdstrike* [online]. 21. prosinec 2022 [vid. 2023-04-16]. Dostupné z: https://www.crowdstrike.com/cybersecurity-101/observability/crud-vs-rest/

[35] SCHWARZMÜLLER, Maximilian. *Dynamic vs SPA vs Static Websites* [online]. 15. duben 2019 [vid. 2023-04-04]. Dostupné z: https://academind.com/tutorials/dynamic-vs-static-vs-spa

[36] SIBISI, Mduduzi. SQLite vs MongoDB - What's the Difference? (Pros and Cons). *Cloud Infrastructure Services* [online]. 2. srpen 2022 [vid. 2023-04-13]. Dostupné z: https://cloudinfrastructureservices.co.uk/sqlite-vs-mongodb-whats-the-difference/

[37] SÖHLEMANN, Claudia. *SSR or CSR - what is better for Progressive Web App?* [online]. 3. duben 2023 [vid. 2023-04-06]. Dostupné z: https://kruschecompany.com/ssr-or-csr-for-progressive-web-app/

[38] SQLite. *SQLite Home Page* [online]. 22. březen 2023 [vid. 2023-04-13]. Dostupné z: https://sqlite.org/index.html

[39] StackOverflow. *Stack Overflow Developer Survey 2022* [online]. květen 2022 [vid. 2023-04-13]. Dostupné z: https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022

[40] SUD, Kanika. *Practical hapi: build your own hapi apps and learn from industry case studies*. Berkeley, CA: Apress, 2020. ISBN 978-1-4842-5805-7.

[41] SWADIA, Shachee. How to Secure Your React.js Application. *freeCodeCamp.org* [online]. 28. říjen 2021 [vid. 2023-04-27]. Dostupné z: https://www.freecodecamp.org/news/best-practices-for-security-of-your-react-js-application/

[42] Vue.js. *Introduction | Vue.js* [online]. [vid. 2023-04-10]. Dostupné z: https://vuejs.org/guide/introduction.html#what-is-vue

[43] VYAS, Rishi. Comparative Analysis on Front-End Frameworks for Web Applications. *International Journal for Research in Applied Science and Engineering Technology* [online]. 2022, **10**(7), 298–307. ISSN 23219653. Dostupné z: doi:10.22214/ijraset.2022.45260

[44] ŽÁRA, Ondřej. Přihlášení pomocí třetí strany: ověření identity s OAuth 2.0. *Root.cz* [online]. [vid. 2023-04-27]. Dostupné z: https://www.root.cz/clanky/prihlaseni-pomoci-treti-strany-overeni-identity-s-oauth-2-0/

8 Přílohy

8.1 Příloha 1 – Zdrojový kód fáze Základní pilíře aplikace

https://github.com/dusekjan/restaurace_FE/tree/1_Zakladni_pilire_aplikace

8.2 Příloha 2 – Zdrojový kód fáze Směrovač a přihlašovací proces

https://github.com/dusekjan/restaurace_FE/tree/2_Smerovac_a_prihlasovaci_proces

8.3 Příloha 3 – Zdrojový kód fáze Objednávkový systém

https://github.com/dusekjan/restaurace_FE/tree/3_Objednavkovy_system

8.4 Příloha 4 – Zdrojový kód fáze Redux a přihlášení uživatelé

https://github.com/dusekjan/restaurace_FE/tree/4_Redux_a_prihlaseni_uzivatele

8.5 Příloha 5 – Zdrojový kód finální fáze projektu Restaurace

https://github.com/dusekjan/restaurace_FE

8.6 Příloha 6 – Zdrojový kód backendu webové aplikace

https://github.com/dusekjan/restaurace_BE

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Jan Dušek**
Studium:	I2100056
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Distanční výukové materiály pro výuku webových technologií**
Název diplomové práce AJ:	Distance learning materials for web technologies learning

Cíl, metody, literatura, předpoklady:

Cíl: Vytvořit startovací aplikaci, která bude sloužit studentům předmětu Technologie pro publikování na Webu.

Metodika: Diplomová práce bude popisovat a pracovat s projektem studenta, na kterém budou demonstrovány postupy při vzniku webové aplikace.

Výsledek(přínos): Během vytváření projektu bude brán zřetel na přehledné rozdělení vývoje do předem daných kroků, celý projekt pak lze převzít a použít jako distanční výukové materiály.

Osnova:

1. Úvod
2. Cíl a Metodika
3. Vývoj webových aplikaci
4. Návrh projektu
5. Implementace projektu
6. Shrnutí a Závěr

1. Comparative Analysis on Front-End Frameworks for Web Applications. *IJRASET Publication* [online]. 2022. ISSN 2321-9653. Dostupné z: doi:https://doi.org/10.22214/ijraset.2022.45260
2. GHIMIRE, Devndra. *Comparative study on Python web frameworks: Flask and Django* [online]. 2020. https://urn.fi/URN:NBN:fi:amk-2020052513398. Dostupné z: https://www.theseus.fi/handle/10024/339796
3. RÍOS, Jimmy Molina a Nieves Pedreira SOUTO. *Comparison of development methodologies in Web applications* [online]. 2019. Dostupné z: doi:https://doi.org/10.1016/j.infsof.2019.106238
4. HOFFMAN, Andrew. *Web Application Security*. Sebastopol, CA 95472: O'Reilly Media, 2020. ISBN 9781492053118.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D.
Datum zadání závěrečné práce:	26.1.2021