FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Webový objednávkový systém pro restaurace
Student:	Manh Tú Do
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webový systém pro restaurace, a to konkrétně aplikaci pro realizaci objednávek hostů. Zaměřte se především na samotné objednávání jednotlivých nabídek z menu restaurace a tvorbu samotného menu. Aplikace bude realizována v technologii ASP.NET. V rámci práce se zaměřte i na možnosti tvorby webových aplikací ve výše zmíněné technologii a jejich provoz.

Postupujte dle následujících kroků:

1) Analyzujte potřeby cílových uživatelů
2) Specifikujte funkční a nefunkční požadavky aplikace
3) Analyzujte vhodné technologie a navrhněte systém
4) Implementujte prototyp
5) Otestujte implementovaný prototyp vhodnými testy
6) Na základě testování navrhněte a případně realizujte úpravy v aplikaci
7) Připravte řešení pro produkční nasazení

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 19. prosince 2021 v Praze.

Bakalářská práce

WEBOVÝ OBJEDNÁVKOVÝ SYSTÉM PRO RESTAURACE

Manh Tú Do

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Filip Glazar
9. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu Ing. Filipovi Glazarovi za jeho trpělivost, ochotu v konzultování a vedení mé práce. Děkuji své rodině, která mě za celou dobu studia podporovala a bez ní by tato práce nebyla možná.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2022 .....

Abstrakt

Práce se zabývá vytvořením webové aplikace, která bude sloužit jako objednávkový systém pro restaurace. Náplní práce je analýza existujících objednávkových systému a analýza technologií pro vývoj webových aplikací. Na základě analýzy byl proveden návrh funkčních požadavků a pro vývoj aplikace byly zvoleny Full Stack .NET technologie, zejména ASP.NET Core a front endový framework Blazor, který má nahradit již starší JavaScript na front end vývoji. Dále byla provedena analýza dvou rozdílných hostitelských modelů Blazor Server pro správu restaurační části a Blazor WebAssembly pro klientskou část a bylo naimplementováno netradiční hybridní řešení pro bohatší ukázku obou technologií a srovnání jejich výhod a nevýhod.

Klíčová slova webová aplikace, objednávkový systém, restaurace, Blazor, ASP.NET

Abstract

The thesis deals with creating a web application, which serves as a ordering system for restaurants. Contents of this work consists of analysis of existing ordering systems and analysis of technologies for web development. Based on these analyses functional requirements were designed and for the development itself Full Stack .NET technologies were chosen, particularly ASP.NET Core and front end framework Blazor, which should replace well-known JavaScript on the front end development. Furthermore analysis was done on two different hosting models Blazor Server for managing the restaurant and Blazor WebAssembly for client side part and a hybrid solution was implemented for richer illustration of both technologies and comparison of their pros and cons.

Keywords web application, ordering system, restaurant, Blazor, ASP.NET

Seznam zkratek

SPA	Single-Page Application
CRUD	Create Read Update Delete
HTTP	HyperText Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
XML	Extensible Markup Language
PHP	Hypertext Preprocessor
DOM	Document Object Model
AJAX	Asynchronous JavaScript and XML
JSX	JavaScript Syntax Extension
SEO	Search Engine Optimalization
MEAN	MongoDB, Express.js, Angular, Node.js
API	Application Programming Interface
W3C	World Wide Web Consortium

Úvod

Webové aplikace jsou dnes velmi oblíbeným a používaným softwarem, protože se v moderní době dají spustit na drtivé většině zařízení s internetovým připojením. Jednou z hlavních výhod této aplikace je, že pro aktualizaci a správu webové aplikace není nutno stahovat a instalovat software. V současnosti můžeme nalézt mnoho druhů webových aplikací – nejvíce populární jsou např. e-shopy, diskuzní fóra, online aukce, web blogy, nebo i objednávkový systém, který cílem této práce.

Hlavní motivací této aplikace je zpřehlednit a zjednodušit práci pro restaurace, a to zejména s přijímáním objednávek, komunikace s kuchyní, a zároveň na druhé straně umožnit klientům si přehledně na svém prohlížeči vytvořit objednávku podle svých představ. Další mojí motivací, je naučit se vyvíjet v nové technologii a také si do portfolia připsat nový projekt. Dle mého názoru by dnešní vývojář, ať už webový nebo softwarový, měl mít schopnost se naučit a adaptovat se v moderních technologiích, které každým rokem přibývají.

V této práci se chci věnovat analýze a aplikaci nové technologii Blazor od Microsoftu, která vývojářům poskytuje možnost Full Stackového pojetí vývoje webových aplikací. Jako front endový framework Blazor nahrazuje již dlouho známé JavaScriptové frameworky Angular, React a Vue. Blazor řeší určité problémy JavaScriptu a vývojáři dovolí psát místo toho čistý C# kód. Existují dvě varianty Blazoru – první se nazývá Blazor Server, která představuje unikátní možnosti práce s webovou aplikací pomocí obousměrného websocketového spojení, a druhé řešení je Blazor WebAssembly, které poskytuje vývoj SPA od Microsoftu a umožňuje pouštět C# kód přímo ve webovém prohlížeči. Provedeme průzkum těchto dvou hostitelských modelů, ukážeme si jejich výhody, nevýhody a jejich aplikace na odlišné situace, a pokusíme se vytvořit novou netradiční hybridní aplikaci zkombinováním těchto dvou hostitelských modelů.

Následně bych chtěl systém nasadit v reálném provozu, zejména u nás v restauraci, nebo u potenciálních zákazníků, kteří by měli zájem o objednávkový systém.

Cíle

Výsledkem této práce by měl být naimplementovaný prototyp webového objednávkového systému pro restaurace, který bych chtěl následně nasadit do reálného provozu. Aplikace by měla plnit všechny funkční i nefunkční požadavky na základě analýzy cílových uživatelů. V teoretické části práce provedu analýzu technologií pro vývoj webových aplikací a zvolení vhodné technologie. V praktické části budu implementovat kód, který se následně pokusím vysvětlit a ukázat, jak funguje zvolená technologie a krátké ukázky práce.

Kapitola 1
Analýza

V této kapitole uděláme analýzu potenciálních cílových uživatelů, kteří budou tuto aplikaci používat. Rozvedeme si funkční a nefunkční požadavky a poté následuje analýza použité technologie.

1.1 Analýza současných systémů

Při navštívení dnešních restaurací jsem z vlastního pozorování zaznamenal, že některé restaurace stále používají papírky na zapisování objednávek, stejně jako v našem rodinném podniku. Jednoduché řešení, které je funkční v malých podnicích, ale dle mého názoru neefektivní, protože papírky se musí kupovat, zápis může trvat déle, rukopis může být nečitelný a komunikace objednávek s kuchyní zabírá nějaký čas. Pro větší podniky je rozumnější si pořídit objednávkové a pokladní systémy, které usnadňují provoz restaurace. Mezi hlavními funkcemi těchto systémů by měly být

- objednávkový web
- tvorba objednávek
- vyřizování objednávek
- mobilní číšník
- tvorba menu
- bezkontaktní menu
- správa uživatelů

1.1.1 Přehled existujících objednávkových systémů

Na trhu se dnes nabízí mnoho objednávkových a pokladních systémů, které mají usnadnit provoz restaurace a plní většinu vyjmenovaných funkcí v minulé části. Dále vyberu pár objednávkových a pokladních systémů na základě vlastního pozorování v restauracích a vyhledávaní na Google. Systémy popíšu a srovnám jejich výhody a nevýhody.

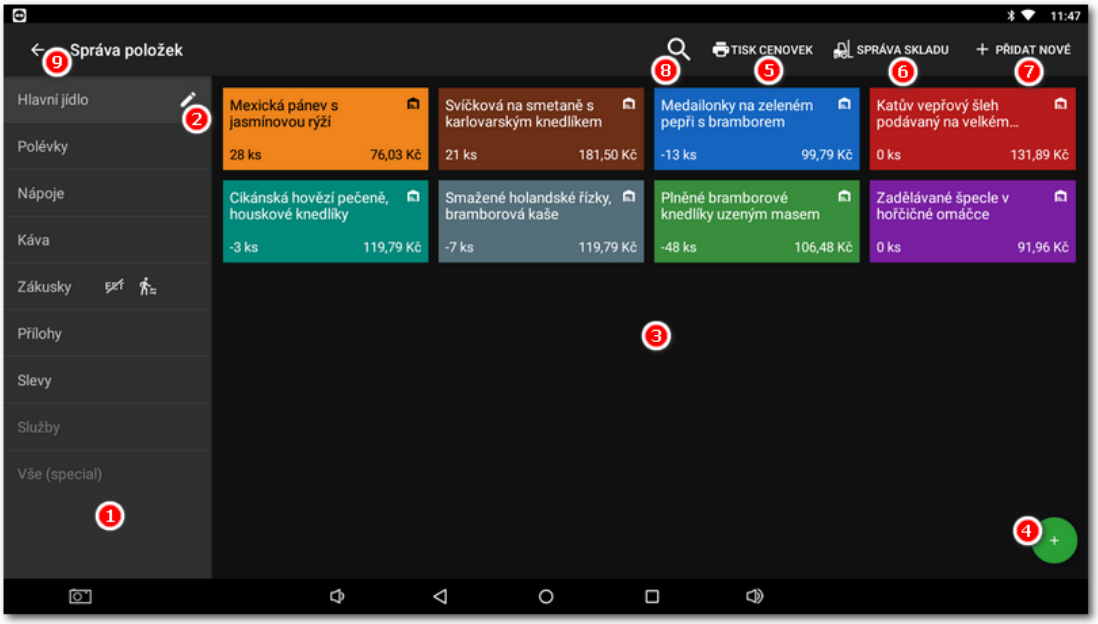
1.1.1.1 Dotykačka a Vyzvednisi.cz

Dotykačka [3] je pokladní a platební systém, který se provozuje v odvětví gastronomie, obchodů, služeb nebo ubytování. Nás bude zajímat nejvíce odvětví gastronomie, které nabízí hromadu funkcí, ačkoliv některé funkce jsou přístupné, až po zakoupení plné licence. Dotykačka nabízí právě 3 licence. Nejlevnější licence SNADNO činí 360 Kč měsíčně, licence NAPLNO činí 639 Kč měsíčně a licence NEOMEZENĚ stojí 1 017 Kč měsíčně. Mezi nejvíce praktické funkce zde patří platební služby, skladový systém, vzdálená správa a mobilní číšník. Pokladní systém je navázaný na objednávkový systém Vyzvednisi.cz, který je od stejné společnosti.

Mezi výhody tohoto systému určitě patří non-stop technická podpora, která je poskytovaná na webové stránce a je přístupná pro všechny druhy licencí. Dále podporuje platební služby společně s vlastní pokladnou, která tiskne účtenky. Nejvíce používanou funkcí je mapa stolů, která zpřehledňuje objednávky a účty zákazníků. Kompletní správa rezervací, online objednávek, docházky zaměstnanců a přehled inventury a tržeb je na jednom místě bez nutnosti přepínání mezi různými systémy.

Nevýhodou tohoto systému je, že pro plné využití všech funkcí je nutné zakoupit plnou licenci. Například funkce mobilního číšníka je přístupná v licenci NAPLNO, ale je omezena jen na jednoho číšníka. Další nevýhodou je instalace aplikace, která běží jen na operačním systému Android a to jen v určitých verzí. Hardware také musí splňovat určité minimální požadavky. Dotykačka nenabízí vlastní plnohodnotný objednávkový web na míru, ale pouze registrace podniku na jejich webu Vyzvednisi.cz.

Obrázek 1.1 Ukázka aplikace Dotykačky - správa položek



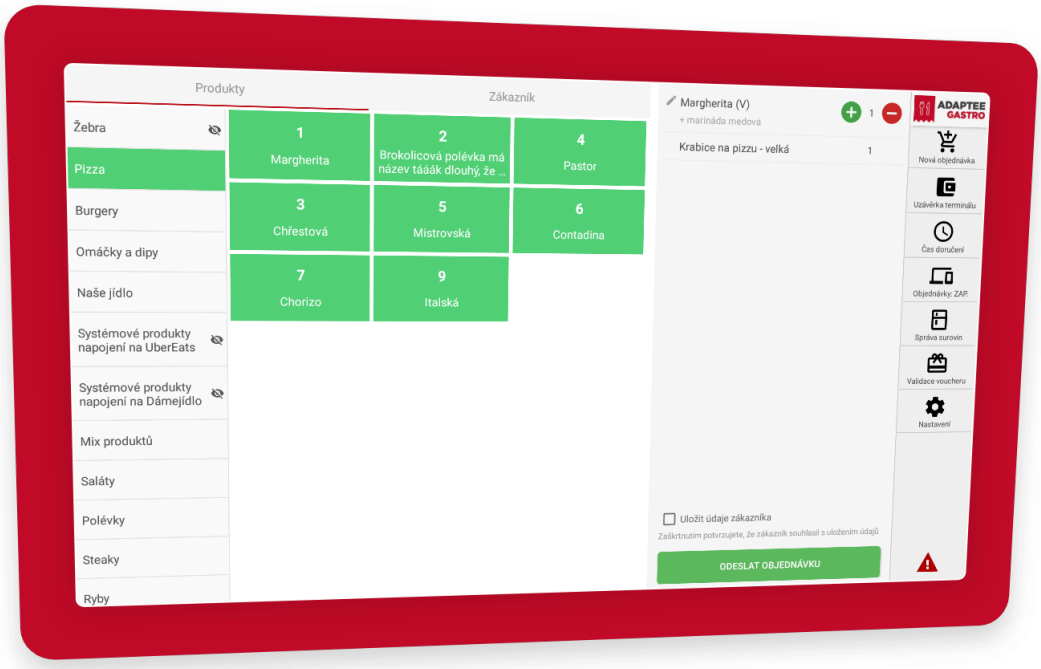
1.1.1.2 Adaptee Gastro

Adaptee Gastro [4] je online objednávkový systém pro restaurace. Mezi hlavními funkcemi tohoto systému patří objednávkový web, terminál na vyřizovaní objednávek, online platba, sledování objednávek, integrace s pokladnami, integrace s Dáme jídlo a Wolt, přehledné statistiky a marketingové akce.

Výhodou systému Adapte Gastro bych zvolil jejich objednávkový web. Jedná se o plnohodnotný web udělaný na míru pro jednotlivé podniky a je hostovaný na vlastní doméně. Na webu se nabízí tvorba objednávek, které se dají přijímat až sedm dní dopředu, uživatelské účty a jejich správa, kde mohou zákazníci sbírat kredity a přehledná administrace webu.

Nevýhodou Adaptee Gastro je jejich cena. Nabízí zde jen jednu licenci, která stojí 2 500 Kč měsíčně. V porovnání s Dotykačkou jde o víc než dvojnásobek ceny a nenabízí se zde tolik funkcí, jako u Dotykačky. Není tu žádná zmínka o mobilním číšníkovi. Aplikace Terminál Adaptee Gastro na vyřizování objednávek v restauraci je dostupná pouze pro operační systém Android.

Obrázek 1.2 Ukázka aplikace Terminál Adaptee Gastro - tvorba objednávky



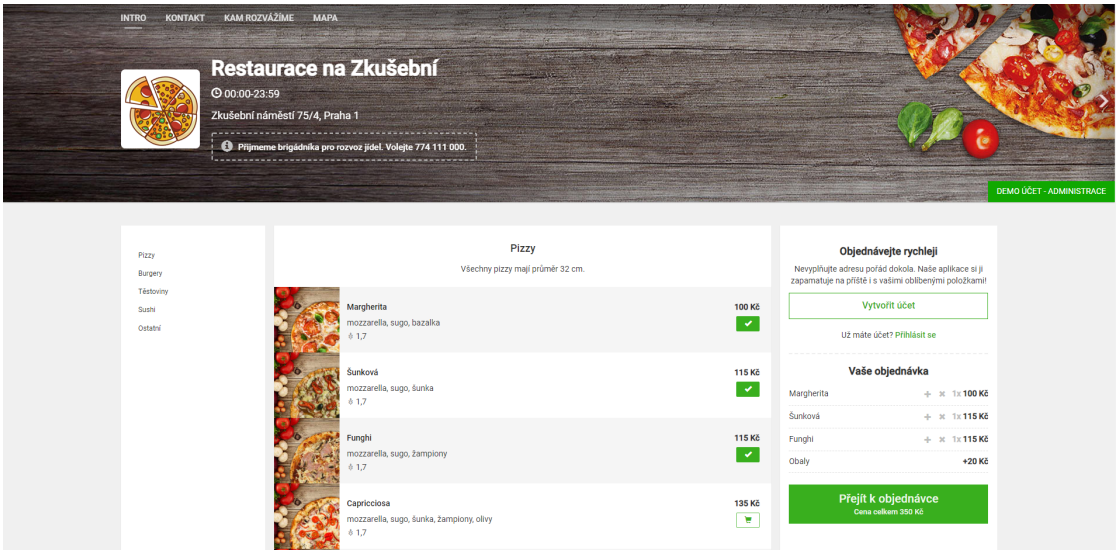
1.1.1.3 Mealgo

Mealgo [5] je objednávkový systém pro restaurace s podobnými funkcemi jako Dotykačka a Adaptee Gastro. Systém nabízí prostředí pro online objednávky a prostředí pro administraci, kde se spravují sortimenty, zobrazuje se přehled objednávek, přehled tržeb a nastavení provozovny. Mealgo nabízí právě 3 licence, přičemž jedna z nich je zdarma. Funkce této licence jsou zde velmi limitované a prostředí pro online objednávky je hostované na jejich doméně s reklamami. Další licence OPTIMAL je placená a stojí 600 Kč měsíčně. Zde je například objednávkový web hostovaný na vlastní doméně a bez reklam, nabízí se také technická podpora a online platba kartou. Poslední licencí je FULL a cena činí 4 % z měsíčního obratu. Tato licence obsahuje všechny funkce nabízené Mealgo.

Výhodou tohoto systému je licence, která je zdarma. Ačkoliv je zde omezena spousta funkcí, tak pro malé podniky by mohl být tento systém úplně dostačující. Cena druhé licence je také přívětivá, ale za stejnou cenu nabízí Dotykačka více funkcí.

Nevýhodou Mealgo je chybějící aplikace pro obsluhu. Není zde žádná zmínka o mobilním číšníkovi, ani o centralizované pokladně, kde by obsluha mohla vytvářet objednávky. Píší pouze, že při nové objednávce dostane restaurace upozornění na objednávku, předpokládám tedy, že upozornění přijde na stránku administrace.

Obrázek 1.3 Ukázka Mealgo zákaznického prostředí - tvorba objednávky



1.2 Analýza funkčních a nefunkčních požadavků aplikace

Cílovými uživateli jsou restaurace a jejich zákazníci. Nejdůležitější motivací této práce je zjednodušit a zpřehlednit práci zaměstnancům i zákazníkům restaurace. Restaurace by měla moci vytvořit si své vlastní menu z nabízených produktů a s příslušnými kategoriemi pro větší přehlednost a filtrování.

Obsluha restaurace by měla chodit ke stolům se zařízením s webovým prohlížečem, např. tablet a jednoduchým naklikáním vytvořit objednávku. V objednávce se objeví informace o vybraných jídlech a ke kterému stolu patří. Tato objednávka se objeví na zařízení v kuchyni, kde objednávku příjme a po dokončení označí jako dokončenou. Druhou možností je objednávka s sebou. Zákazníci po registraci a přihlášení si mohou vytvořit objednávku na jejich mobilním zařízení nebo počítači. Objednávku kuchyni příjme a notifikuje zákazníka o zpracování.

Tímto způsobem se zbavíme zápisu na papír, jednoho centralizovaného počítače a pro restaurace je zajištěna historizace a logování objednávek zákazníků. Díky těmto datům lze provézt marketingovou analýzu, která umožní dělat speciální akce pro zákazníky. Nevýhodou je prvotní investiční vklad na nákup zařízení.

1.2.1 Registrace a přihlašování uživatelů

Aby mohl uživatel aplikaci plně využívat, musí se registrovat a přihlásit se. Každý uživatel bude mít svojí roli a autorizaci. Příkladné role jsou admin (provozovatel restaurace), pracovník (obsluha, kuchyně) a uživatel (zákazník). Přihlašovací stránka bude klasický formulář s e-mailem

a heslem. Registrační stránka bude obsahovat formulář s uživatelským jménem, e-mailem, telefonním číslem a heslem. Na jiné stránce se správou uživatelského účtu lze údaje měnit, nebo přidat nové údaje jako doručovací a fakturační adresy.

1.2.2 Správa produktů a kategorií

Provozovatel restaurace bude moci vytvářet menu dle svých potřeb. Menu se skládá z kategorií, které budou mít vlastní stránku správy s běžnými operacemi CRUD – vytvořit, číst, update a delete. Na stránce budou vypsané vytvořené jednotlivé kategorie a k nim dvě tlačítka pro úpravu a mazání. Pro vytvoření nové kategorie slouží jedno tlačítko pro přidání, které spustí modální okno s formulářem. Ve formuláři se vyplní jméno a popis. Kategorie jsou poté zvoleny ve formuláři vytvoření produktů.

Obdobně produkty budou mít podobnou stránku jako kategorie s CRUD operacemi. Avšak se bude lišit modálním oknem s formulářem pro vytvoření produktu. Formulář obsahuje prvky jméno, popis, cena bez DPH, kategorie a sazba DPH produktu. Všechny prvky jsou před odesláním formuláře validovány v prohlížeči. Modální okno s formulářem na úpravu produktu bude při otevření obsahovat již známé data.

1.2.3 Tvorba objednávek

Obsluha bude moct vytvořit objednávku z nabízených produktů v menu. Na stránce bude zobrazen košík s vybranými produkty s počtem a cenou. Na konci košíku bude napsaná celková cena objednávky. Zde je nutno vyplnit i údaj o stole, ke kterému objednávka patří. Po vybrání produktů uživatel stiskne tlačítko pro vytvoření objednávky, která spustí modální okno s potvrzením. Pokud je objednávka úspěšně vytvořena, košík se vyprázdní a stránka je hned připravena na tvorbu nové objednávky. Pod košíkem bude zobrazen seznam produktů s filtry jmen a kategorií. Produkty budou mít napsané jména a ceny, tlačítka plus a mínus pro přidání a odebrání z košíku, které automaticky upravují i celkovou cenu.

Zákazník vytváří objednávky na jiné podobné stránce, ale bude mít navíc další prvky na vyplnění – jméno a příjmení zákazníka, telefonní číslo a e-mail zákazníka, vyzvednutí na místě či doručovací adresa, fakturační adresa, čas vyzvednutí objednávky. Většina těchto údajů budou předvyplněny z přihlášeného účtu.

1.2.4 Správa objednávek

Obsluha a kuchyň budou mít přístup na stránku se správou objednávek. Stránka bude obsahovat seznam nových objednávek s příslušnými produkty. Objednávky budou rozděleny na objednávky v restauraci, objednávky s sebou a objednávky s doručením. Budou seřazeny podle času vytvoření a každá objednávka bude mít tlačítko pro změnu stavu objednávky. Až kuchyň objednávku dokončí, stiskne tlačítko, které změní stav objednávky na připraveno a odebere objednávku ze seznamu nových objednávek. Na jiné podobné stránce se bude moct obsluha podívat na připravené objednávky, které také mají tlačítko pro změnu stavu. Po zaplacení objednávky obsluha odklikne, že je zaplacena a změní její stav. Zákazníci si budou moct zobrazit své předešlé a aktuální objednávky.

1.3 Analýza použitých technologií

Pro vývoj webových aplikací je důležité porozumět základům, co to vůbec web je a na jakých principech funguje. V následujících sekcích jsou popsány hlavní pojmy nezbytné k vývoji webové aplikace.

1.3.1 Webová stránka

Webová stránka je soubor uložený na internetu na webovém serveru. Tyto soubory jsou napsány ve značkovacím jazyce HTML, pro grafickou úpravu stránek se používají kaskádové styly (CSS) a pro dynamické změny se používá skriptovací jazyk JavaScript, který pak prohlížeč vyrendruje. Stránky jsou uživateli zobrazeny ve webovém prohlížeči na základě architektury klient-server dříve pomocí protokolu HTTP, ačkoliv byl brzo nahrazen šifrovanou verzí HTTPS. Klient pomocí prohlížeče posílá požadavek na server přes URL a ten mu vrací odpověď – stránku. Dříve byly stránky statické, jejich obsah se dynamicky neměnil, avšak dnes tomu tak není. Většina stránek je dynamických – jsou vytvářeny v daném časovém okamžiku a obsah se mění na základě obsahu připojené databáze. [6]

1.3.2 Architektura klient-server

Klient-server je síťová architektura, která odděluje klienta a server, kteří spolu komunikují přes počítačovou síť. Pod pojmem komunikaci rozumíme předání informací a zdrojů. Opakem této architektury je peer-to-peer, kde spolu komunikují rovnou přímo jednotliví klienti. V architektuře klient-server se jedná o vztah mezi dvěma programy, kde klient posílá požadavek serveru, který požadavek zpracuje a vrací odpověď. V tomhle modelu běží aplikace typu E-mail, Web, přístup k databázi atd... V našem případě tento model používáme pomocí webového prohlížeče, který se chová jako klient. Webové prohlížeče posílají požadavky přes internetový protokol HTTP a HTTPS, servery tyto požadavky můžou akceptovat, zpracovat a vrátit klientovi požadované informace. [7]

1.3.3 Protokol HTTP

HTTP – HyperText Transfer Protocol je internetový protokol určený pro komunikaci webových prohlížečů s webovými servery. Slouží pro přenos hypertextových dokumentů ve formátu HTML, XML a jiné. HTTP je jeden z nejvíce používaných protokolů společně s elektronickou poštou. Aby protokol plně fungoval, používá další aplikace např. URL, který jednoznačně určí umístění zdroje na internetu. Bohužel je tento protokol nešifrovaný a může zde dojít ke ztrátě integrity dat, proto byl v roce 1994 vyvinut šifrovaná verze HTTPS. [8]

První a nejstarší verzí protokolu je verze 0.9. Používala pouze metodu GET s jediným parametrem – název požadovaného souboru. Server vrátil jako odpověď dokument bez hlaviček. Kvůli svým omezením byl rychle nahrazen verzí HTTP/1.0. Zde se objevil stavový řádek v odpovědi, HTTP hlavičky v požadavku i odpovědi a nové metody POST a HEAD. Dnes je nejvíce používaná verze HTTP/1.1, která přidává nové metody, umožňuje přenos více souborů v jednom spojení a udržuje TCP spojení (tzv. keep-alive connection). [8]

```
Request ::= RequestLine (Header CRLF)* CRLF [body]
RequestLine ::= Method RequestUri HTTPVersion CRLF
Header ::= Host | Accept | ...
Method ::= GET | HEAD | ...
```

Výpis kódu 1 Ukázka 1: Formát požadavku

```
GET /profile/140f269cb240 HTTP/1.1
Host: usermap.cvut.cz
Accept: text/html
```

Výpis kódu 2 Ukázka 2: Příklad požadavku

```
Response ::= StatusLine (Header CRLF)* CRLF [body]
StatusLine ::= HTTPVersion Status CRLF
Status ::= StatusCode ReasonPhrase
Header ::= Content-Type | ...
```

Výpis kódu 3 Ukázka 3: Formát odpovědi

```
HTTP/1.1 200 OK
Date: Sun, 5 Jul 2020 16:46:06 GMT
Content-Length: 39
Content-Type: text/html
<html>
<body>Ahoj svete!</body>
</html>
```

Výpis kódu 4 Ukázka 4: Ukázka odpovědi [1]

1.3.3.1 Některé dotazovací metody

GET požadavek na uvedené zdroje, výchozí metoda při zobrazení stránek

HEAD podobná metodě GET, ale pouze poskytuje metadata o požadovaném zdroji

POST metoda posílá uživatelská data na server např. při vyplnění formulářů

PUT nahrání dat na server, objekt je jméno vytvářeného souboru

DELETE smaže uvedený objekt ze serveru [8]

1.3.4 HTML

HTML, zkráceně HyperText Markup Language je značkovací jazyk pro tvorbu webových stránek, které jsou strukturovány hypertextovými odkazy. Jazyk je charakterizován množinou značek (tzv. tagů) a jejich vlastností (tzv. atributů). Mezi značky se uzavírá části dokumentu a tím se určuje sémantika textu. Tagy se uzavírají závorky <a>. Část dokumentu tvořená z otevírací značkou, obsahem a koncovou značkou tvoří prvek (tzv. element) dokumentu. Existují párové značky, kde koncová značka má po závorce lomítko nebo nepárové značky, které nemají v sobě obsah, ale spíš mohou něco vykreslit. Ukážeme si zde příklad důležité párové značky <a> a její vlastnosti href, která určuje URL při kliknutí na tento odkaz. [9]

```
<a href="http://example.com">text odkazu</a>
```

Výpis kódu 5 Ukázka elementu s atributy

1.3.4.1 Struktura dokumentu

Deklarace typu dokumentu značka `<!DOCTYPE html>`, prohlížeč pozná, že to je HTML dokument

Kořenový element prvek html, značky `<html>` a `</html>`

Hlavička dokumentu prvek head, značky `<head>` a `</head>` >

Tělo dokumentu prvek body, značky `<body>` a `</body>` [9]

1.3.5 CSS

Kaskádové styly, anglicky Cascading Style Sheets je jazyk pro popis způsobů renderování elementů v dokumentech napsaných v jazycích HTML, XHTML a XML. Hlavním cílem jazyka je umožnit návrhářům oddělit vzhled dokumentu od jeho struktury a významu. Jazyk byl navržen standardizační organizací W3C, autorem prvotního návrhu byl Hakon Wium Lie.

1.3.5.1 Syntax

„Definice kaskádových stylů sestává z pravidel. Každé pravidlo obsahuje selektor a blok deklarací. Každý blok deklarací pak obsahuje deklarace oddělené středníky ; a každá deklarace sestává z identifikátoru vlastnosti, následuje dvojtečka : a hodnota vlastnosti. Nepovinně ještě může následovat označení !important, které zvýší sílu deklarace.“ [2]

```
body {
  background-color: white;
  color: black;
  padding: 10px !important;
}
```

Výpis kódu 6 Příklad pravidla:

Celý blok nazýváme pravidlem, body je selektor, celá část v závorkách je blok deklarací, jednotlivé řádky jsou deklarace oddělené středníkem, background-color je identifikátor vlastnosti a white je její hodnotou. Toto pravidlo nastavuje barvu pozadí na bílou.

1.3.5.2 Připojení kaskádových stylů do HTML stránky

Existují různé způsoby, jak aplikovat styly na elementy např. in-line, zápis do elementu style a nebo nejvíce používaný je připojení externího souboru pomocí elementu link.

```
<p style="color: red; text-decoration: underline">Text.</p>
```

Výpis kódu 7 Příklad in-line

```
<style type="text/css">
#hlavicka{
  width: 200px;
  height: 450px;
}
</style>
```

Výpis kódu 8 Příklad zápisu do elementu style

```
<head>
  <link rel='stylesheet' href='styly.css' type='text/css'>
</head>
```

Výpis kódu 9 Příklad připojení pomocí externího souboru [2]

1.3.6 JavaScript

JavaScript je multiplatformní, objektově orientovaný, událostmi řízený skriptovací jazyk, jehož autorem je Brendan Eich ze společnosti Netscape. Syntaxí spadá do rodiny jazyků jako je C, C++, Java, ale liší se od nich sémantikou. V roce 1997 byl tento jazyk standardizován asociací ECMA (European Computer Manufacturers Association) a tato standardizovaná verze byla pojmenována ECMAScript, a z ní byly odvozeny i další implementace, jako je ActionScript.

Ačkoliv se jazyk může používat na straně serveru, ve většině případů se používá hlavně na straně klienta ve webovém prohlížeči vložený do HTML kódu stránky, kde je poté interpretován. Pomocí JavaScriptu lze ovládat interaktivní prvky grafického uživatelského rozhraní (tlačítka, textové pole) nebo tvoří animace a efekty obrázků. Program v JavaScriptu se tedy spouští až po stažení souboru z Internetu, na rozdíl od ostatních interpretovaných jazyků, jako je PHP, který se spouští ještě před stažením z Internetu. JavaScript spuštěný v prohlížeči například nemůže pracovat se soubory a nedojde k ohrožení soukromí uživatele. [10]

1.3.7 Webová aplikace

Webová aplikace je software, který je poskytován uživateli webovým serverem a je spuštěn webovým prohlížečem. Prohlížeč je nazýván tenkým klientem, protože sám o sobě logiku aplikace nezná a je poskytovaná serverem. Největší výhodou webové aplikace je univerzální klient, tedy dá se spustit na mnoho zařízeních a nezáleží na jakých operačních systémech běží. Aktualizace a správa systému bez nutnosti šíření a instalace softwaru je hlavním důvodem výběru. Dříve aplikace typu klient-server měla klientský program, kde bylo uživatelské rozhraní a musel být nainstalován na každém počítači. Každá aktualizace vyžadovala aktualizaci na každé pracovní stanici a to zvyšovalo náklady na podporu a snížení efektivnosti zaměstnanců.

Oproti tomu webové aplikace dynamicky generují sérii webových stránek, který je podporován běžnými prohlížeči. Webové stránky jdou podány jako statické stránky, ale sled takových stránek vyvolává pocit interaktivity a díky skriptovacímu jazyku JavaScript lze přidat dynamičnost i pro jednotlivé prvky. Ačkoliv je prohlížeč univerzálním klientem, nepodporuje standardní technologie pro aplikace jako jsou například vykreslení na obrazovku nebo obecné techniky jako drag-and-drop. Při tvoření webové aplikace je snaha o interaktivity bez znovunačtení stránek pomocí skriptovacích jazyků. Dnes existují nové technologie, které umožňují spolupráci skriptů na klientské straně se serverem, např. AJAX nebo Blazor. Webové aplikace jsou většinou strukturovány jako třívrstvé architektury. Webový prohlížeč slouží jako prezentační vrstva, nástroje

pro generování stránek a logika aplikace je aplikační vrstva a třetí vrstva je datová, která je databáze připojená k aplikaci. [11]

1.3.8 Single-page application

Single-page application je webová aplikace, která interaguje s uživatelem a jeho vstupem pře-kreslením současné webové stránky novými daty z webového serveru, místo výchozího přístupu, kde prohlížeč znovu načítá novou stránku. Cílem SPA jsou rychlejší přechody, aby aplikace měla nativní dojem. V SPA nikdy nedochází k znovu načtení nové stránky, ale místo toho všechen HTML, CSS a JavaScript kód je načten prohlížečem prvotním spuštěním stránky a další zdroje jsou načteny dynamicky dle potřeb aplikace a uživatele. Pro vývoj SPA existují mnoho JavaScript frameworků a knihoven, mezi nejvýznamnější patří AngularJS, EmberJS, React a Vue.js. [12]

1.4 Analýza technologií pro vývoj webových aplikací

Pro popis a srovnání konkurenčních technologií pro vývoj webových aplikací vyberu právě ty, které jsou nejvíce populární podle vyhledávání na Google a ty, se kterými jsem už pracoval. Některé frameworky jsou rozdělené na back endové, front endové, nebo některé mohou být obojí, tzn. full-stackové.

1.4.1 Symfony

Symfony je framework pro vývoj webových aplikací v programovacím jazyce PHP. Framework vychází z návrhového vzoru MVC (Model View Controller). Symfony také obsahuje znovu použitelné komponenty a knihovny. Cílem frameworku je usnadnit vývoj a údržbu webové aplikace a nahradit opakující se kódování. Součástí Symfony jsou existující PHP open-source projekty, které Symfony značně používá. Mezi tyto projekty patří Propel nebo Doctrine, které se starají o ORM, šablonovací engine Twig nebo PHPUnit pro unit testing. [13] Mezi nejznámější webové stránky, které používají Symfony patří Spotify, Vogue, Dailymotion a více.

Výhody Symfony

- znovu použitelné komponenty a knihovny ušetří čas
- flexibilita, mnoho pluginů a možnost vše konfigurovat
- komerčně podporovaný framework, velká komunita

Nevýhody Symfony

- nízká výkonnost, dlouhé načítání
- build a launch trvá dlouho
- je těžší se ho naučit oproti jiným frameworkům [14]

1.4.2 Django

Django je framework pro vývoj webových aplikací v programovacím jazyce Python. Framework vychází z návrhového vzoru MTV (Model Template View). Cílem frameworku je tvorba komplexních webových aplikací napojené na databáze. Django klade důraz na opětovné použití komponentů, méně kódu, rychlý vývoj a princip don't repeat yourself. Mezi nejznámější webové stránky, které používají Django patří Instagram, Mozilla, Disqus a více.

Mezi hlavní funkce frameworku Django patří ORM, které se stará o mapování data modelů na relační databázi, systém pro zpracování HTTP požadavků a šablonovací systém. Obsahuje také odlehčený a samostatný webový server, serializační a validační systémy, cachovací systém a více. [15]

Výhody Django

- vývoj v jazyce Python
- rychlý vývoj
- škálovatelnost
- dobrá bezpečnost v autentizačních a autorizačních systémech

Nevýhody Django

- monolitický, obsahuje předem dané soubory a proměnné
- nehodí se na malé projekty
- není možné zpracovat více požadavků najednou [16]

1.4.3 React

React (také známý jako React.js nebo ReactJS) je front endová JavaScriptová knihovna pro tvorbu uživatelského rozhraní založené na UI komponentách psaných v JSX. Je vyvíjen hlavně společností Meta (původně Facebook) a komunitou jednotlivých vývojářů a jiných společností. React byl využit jako základ pro tvorbu Single-page nebo mobilních aplikací. Nicméně React se pouze stará o stav aplikace a vykreslení tohoto stavu do DOM, takže při psaní React aplikací je nutné použít dodatečné knihovny navíc, například pro routing. [17]

Výhody React

- virtuální DOM, umožní provádět změny v malé komponentě, bez zásahu do celé aplikace
- znovu použitelné komponenty
- React Native, vývoj mobilních aplikací
- jednoduchý se naučit oproti jiným JavaScriptovým frameworkům
- velká komunita vývojářů

Nevýhody React

- nedostatek vlastní dokumentace
- rychlý vývoj Reactu, vývojáři se musí učit nové věci
- pouze front-endový framework [18]

1.4.4 Angular

Angular je framework pro vývoj front-endu webových aplikací psaný v jazyce TypeScript. Je vyvíjen hlavně společností Google a komunitou jednotlivých vývojářů a jiných firem. Angular je následníkem původního frameworku AngularJS. Angular společně s ostatními frameworky MongoDB, Express.js a Node.js tvoří MEAN stack, kde Angular je používán na vývoj front endu. [19]

Výhody Angular

- psaní v jazyce TypeScript
- obsahuje v sobě mnoho feature - routing, dependency injection, state, reactivity
- cross-platform - desktopové nativní aplikace nebo mobilní aplikace
- flexibilní šablonovací systém

Nevýhody Angular

- limitované SEO
- komplexní, nevhodný pro malé aplikace
- pouze front endový framework [20]

1.5 Analýza vybrané technologie pro vývoj webové aplikace

Dnes existují mnoho technologií a frameworků, které umožňují vývojářům psát webové aplikace a výběr vhodné technologie není lehký. Pro moji práci jsem zvolil technologii ASP.NET Core a její front endový framework Blazor, který představím a zdůvodním jeho volbu v následující sekci 1.5.6.

1.5.1 .NET

„.NET je bezplatná open source vývojová platforma pro vytváření mnoha druhů aplikací, jako jsou webové aplikace, webová rozhraní API a mikroslužby, mobilní aplikace. S .NET, váš kód a projektové soubory vypadají podobně, bez ohledu na typu aplikace, který vyvíjíte. Máte přístup k stejnému modulu runtime, rozhraní API a jazyku.“ [21]

Framework, který nás bude zajímat nejvíce je ASP.NET Core rozšiřující ASP.NET a běžící na .NET. ASP.NET Core je open source cross-platform framework sloužící k budování webových aplikací a služeb za pomocí .NET knihoven v programovacím jazyku C#. Na rozdíl od svého předchůdce ASP.NET přešel novému designu architektury, který míří na štíhlou a modulárnější architekturu. Mezi nabízející nástroje patří:

- unifikované prostředí pro vývoj UI a web API
- stránkový šablonový syntax Razor a Razor Pages
- open source a cross-platform - vývoj na Windows, Linux a macOS
- integrace moderních front endových frameworků jako je Blazor
- vestavěný návrhový vzor dependency injection
- nástroje na jednoduchý vývoj web aplikace [22]

1.5.2 Vývojové prostředí Visual Studio

Oficiálním a podporovaným vývojovým prostředím pro .NET aplikace je Microsoft Visual Studio. Vyvíjejí se v něm programy, webové stránky, webové aplikace, webové servisy a mobilní aplikace. Visual Studio je podporován editorem kódu IntelliSense, který našeptává a doplňuje kód společně s refaktoringem. Obsahuje také integrovaný debugger na úrovni kódu i na úrovni stroje. Další podporované nástroje zahrnují designer formulářů, designer webů, tříd a databázových schémat. Podporuje mnoho programovacích jazyků, mezi vestavěné patří C, C++, C#, ale další jazyky jako F#, Python, Ruby mohou být přidány a nainstalovány zvlášť pomocí jazykových služeb. Pro vývoj webových stránek je podporováno XML/XSLT, HTML/XHTML, JavaScript a CSS. [23]

Kdybych měl porovnat Visual Studio s jinými vývojovými prostředími, umístil bych ho spíše na lepší půlce. Jako student FITu, kde se učí jazyky C a C++ jsem používal především Visual Studio Code, který je velmi minimalistický a je tam potřeba vše pro vývoj v C/C++. Ačkoliv se zde dá nastavit debugger, nepodařilo se mi to zprovoznit. Oproti tomu integrovaný debugger ve Visual Studio dobře funguje a pomocí breakpointů, krokování a přístup do paměti je skvělým nástrojem pro hledání bugů. V jiných předmětech jsem si osahal i vývojové prostředí IntelliJ IDEA, které podporuje jazyky Java, Python a PHP. Toto vývojové prostředí také obsahuje vše potřebné a i další nástroje, které ulehčují vývoj, ale jedinou nevýhodou je, že je potřeba licence. To u Visual Studio není potřeba, existuje zde Pro verze, která je placená a Community verze, která je přístupná všem.

1.5.3 Balíčkovací systém NuGet

Základním nástrojem pro jakékoliv moderní vývojové prostředí je balíčkovací systém, pomocí kterého si vývojáři sdílejí kód a knihovny. Obvykle tento kód je zkompilován do balíčků společně s dalšími potřebnými věci, které tyto balíčky využívají.

Pro .NET je oficiálním a podporovaným systémem NuGet, který definuje, jak se balíčky vytváří, hostují, používají a poskytuje nástroje pro tyto operace. Jednoduše, NuGet balíček je prostý ZIP soubor s příponou .nupkg, který obsahuje zkompilovaný kód v DLL, soubory spojené s tímto kódem a další informace typu verze balíčku. Vývojáři, kteří balíčky vytváří poté publikují veřejnému nebo privátnímu hostiteli. Ostatní si mohou z těchto hostitelů balíčky přidat do svých projektů a použít funkcionalitu kódu balíčku. Výchozím a veřejným hostitelem je nuget.org, kde se nachází spousta užitečných balíčků, ale existují i privátní hostitele, kde se vytvářejí balíčky například pro nějakou firmu, organizaci nebo skupinu lidí. [24]

Ve Visual Studio existuje pěkné uživatelské rozhraní pro instalaci balíčků. Jeho jednoduchý design umožní vyhledávání a přidání balíčků do projektu pomocí klikáním myší. Vyskytuje se tam i záložka nainstalovaných balíčků a vedle ní záložka pro aktualizace nových verzí. Seznam nainstalovaných balíčků se dá také najít v projektovém souboru.

1.5.4 Entity Framework Core

Entity Framework Core (EF Core) je odlehčená, rozšiřitelná, open source a cross-platform verze Entity Framework technologie. Tyto technologie slouží jako objektově relační mapovač (ORM), tudíž umožňují vývojářům práce s databází pomocí objektů v objektově orientovaném jazyku. Díky němu je eliminována hromada kódu pro přístup k datům v databázi.

V Entity Framework Core, přístup k datům je provedeno přes model. Model se skládá ze tříd entit a kontextová třída, která reprezentuje relaci s databází. Přes tento kontext se právě dotazují a ukládají data. Entity Framework podporuje obvyklé způsoby pro vývoj modelu např. generování modelu z existující databáze, napsat kód tak, aby odpovídal databázi a pro vytvoření databáze z modelu lze použít Entity Framework Migrace. U dotazování na data se instance tříd

entit získávají pomocí Language Integrated Query (LINQ). Přes tyto instance tříd se dají data vytvářet, mazat a upravovat. [25]

1.5.5 Entity Framework Migrations

Při vývoji jakékoliv aplikace se datové modely mění i v průběhu, něco jsme zapomněli přidat, nebo zákazník chtěl doplnit doplňující informace. Schéma databáze tedy musí být v synchronizaci s aplikací. Právě díky těmto migracím je synchronizace možná, migrace inkrementálně aktualizují schéma databáze podle aktuálního datového modelu a zároveň se uchovávají již existující data v databázi.

Migrace fungují tak, že když je datový model upraven, vývojář použije EF Core nástroje pro vygenerování nové migrace, která popisuje, jaké změny se ve schématu mají provést. Generování probíhá tak, že porovná aktuální model se snímkem starého modelu a určí rozdíly. Tyto vygenerované soubory se potom ukládají do projektové složky, aby např. při týmové práci si databázi aktualizovali i ostatní vývojáři. Poté stačí migrace aplikovat na databázi, např. pomocí EF Core nástroje. EF Core také má záznamy o všech migracích ve speciální tabulce historie, pomocí kterého pozná, jaké migrace už byly provedeny. [26]

1.5.6 Blazor

Blazor je framework pro tvorbu interaktivního webového uživatelského rozhraní na straně klienta pomocí .NET. Jelikož jsem studentem oboru Webového inženýrství, v předmětu BI-TWA jsme si vyzkoušeli práci s JavaScriptem, který mě moc neoslovil. V Blazoru se uživatelské rozhraní na straně klienta a logika na serveru píše ve společném jazyce C#. To ulehčuje práci už jen v tom, že máme na front endu i na back endu stejné definice objektů, takže není třeba přepínat mezi dvěma jazyky, jako u vývoj pomocí JavaScriptu. Jazyk C# je na rozdíl od JavaScriptu silně typovaný, podporuje programovací koncepty jako zapouzdření, dědičnost a polymorfismus, protože je objektově orientovaný. Pro vývoj lze využít i široký výběr existujícího ekosystému .NET knihoven. Z těchto důvodů jsem zvolil Blazor na vývoj aplikace. [27]

1.5.7 Blazor Server

Blazor server je druh hostitelského modelu, který funguje na principu WebSocketu. Jelikož veškerá logika aplikace je na straně serveru, jedná se o architekturu thin client - thick server. Uživatelské rozhraní a její aktualizace zajišťuje knihovna SignalR, která dokáže v real time dostat kód ze serveru ke klientovi. Odesílání událostí uživatelského rozhraní se posílají z klienta na server, tam se aktualizují změny a vyrendruje se znova komponenta, která je poslána zpět ze serveru.

Stav na serveru spojený s každým připojeným klientem se nazývá okruh. Tento okruh umí tolerovat dočasné síťové přerušení a pokouší se o znovu navázání spojení s klientem, pokud je připojení ztracené. V každém prohlížeči je požadován odlišný okruh a samostatné instance stavu komponent spravované serverem. Aplikace rozezná zavření záložky nebo přesměrování na externí URL jako řádné ukončení. V tomhle případě jsou okruh a přidružené prostředky okamžitě uvolněny. V opačném případě jde o odpojení bez odkladu a server ukládá odpojené okruhy na nějaký daný interval, ve kterém se klient může znovu připojit. Na počátku se na klientským prohlížeči se spustí Blazor script (blazor.server.js), který vytvoří SignalR spojení se serverem. Klientská aplikace je zodpovědná za zachování a obnovení stavu aplikace podle potřeby. [28]

Tabulka 1.1 Výhody a nevýhody Blazor Server hostitelského modelu

Výhody	Nevýhody
velikost souborů obsluhována klientům je výrazně menší než Blazor WebAssembly aplikace	možnost vyskytnutí vysoké latence, protože každá interakce uživatele zahrnuje síťový hop
aplikace plně využívá možnosti serveru, včetně použití rozhraní ASP.NET Core API	není offline podpora, pokud připojení selže, aplikace neběží
aplikace je spuštěna na ASP.NET Core, tudíž jsou funkční debuggovací nástroje	škálování aplikace s mnoha uživateli požaduje náročné serverové zdroje, které mohou být drahé
podpora tenkých klientů, např. funguje na prohlížečích, které nepodporují WebAssembly	ASP.NET Core server je nutný, aby běžel server, scénáře bezserverového nasazení nejsou možné
základ kódu aplikace, včetně kódu v komponentách není veřejný klientům	

1.5.8 Blazor WebAssembly

Blazor WebAssembly je nový SPA framework od Microsoftu pro vývoj aplikace pomocí .NET. Na rozdíl od tradičního přístupu, kde logika je na serveru, SPA vykonává logiku uživatelského rozhraní rovnou v prohlížeči přes komunikaci s web serverem použitím web API. Pouštění .NET kódu je možné díky webovému standardu WebAssembly. Když je aplikace zbuilděná a běží v prohlížeči, tak zdrojové kódy v C# a Razor soubory jsou zkompilovány do .NET assembly (sestavení), které jsou stažené prohlížečem společně s runtime .NET. Blazor WebAssembly spustí modul runtime, který následně načte stažené assembly soubory. Blazor WebAssembly používá interoperabilitu JavaScriptu ke zpracování voláni uživatelského rozhraní a manipulace s modelem DOM.

Důležitou součástí této aplikace je její back end, díky němu získáme plnohodnotné prostředí pro vývoj full-stack webu s .NET. Je umožněno sdílení kódu mezi klientskou a serverovou částí a integrace MVC s Razor Pages. Komunikace s back end serverem je zajištěna pomocí různých architektur a protokolů, jako jsou web API nebo SignalR. Stejně jako u Server modelu, na počátku je spuštěn skript blazor.webassembly.js, který stáhne .NET runtime, aplikaci a její závislosti a posléze inicializuje modul runtime pro spuštění aplikace. [28]

Tabulka 1.2 Výhody a nevýhody Blazor WebAssembly hostitelského modelu

Výhody	Nevýhody
podpora offline módu, pokud spadne server, aplikace stále běží	velikost souboru obsluhována klientům je značně větší než Blazor Server aplikace a načtení může trvat déle
zatížení není na serveru, ale na klientovi	je nutný dostatečně dobrý klientský hardware a software (podpora WebAssembly)
lze nasadit aplikaci bez serveru, např. poskytování aplikace z Content Delivery Network (CDN)	aplikace je omezená na vymoženosti prohlížeče

1.5.9 Blazor komponenta

Aplikace je založena na Blazor komponentách. Její formálním názvem je Razor komponenta, ačkoliv jí neformálně nazýváme Blazor komponentou, protože má příponu .razor a chceme ji

odlišit od Razor stránky. Komponenta je základním prvkem uživatelského rozhraní, jako například stránka, dialog nebo formulář. Její funkcí je flexibilní renderování uživatelského rozhraní, zpracování událostí uživatele, dají se vnořovat a opakovaně používat a lze je sdílet a distribuovat jako knihovny tříd nebo NuGet balíčky.

Komponenta je obvykle psaná v souboru s příponou .razor ve formě HTML stránky s Razor markup syntaxí. Tato syntaxe umožní kombinaci C# kódu s prvky HTML a zároveň vývoj ve Visual Studio je podporován IntelliSense. HTML prvky určují uživatelské rozhraní, jak se kde a co nachází a Razor markup určuje chování těchto prvků a zpracovává události. Komponenty se rendrují do prohlížečové reprezentace modelu Document Object Model, který se nazývá strom vykreslování, jehož úkolem je aktualizovat uživatelské rozhraní flexibilním a efektivním způsobem. [29]

Kapitola 2

Návrh systému

V této kapitole se budu věnovat návrhu systému aplikace, kde s použitím předešle zmíněných technologií se pokusíme o novou a efektivní aplikaci.

2.1 Motivace hybridního hostitelského modelu

Hlavní motivací kombinaci hostitelského modelu je využít výhody a zároveň překrýt nevýhody hostitelských modelů Blazor Server a Blazor WebAssembly. V našem konkrétním případě je pro správu systému a práce obsluhy a kuchyně výhodnější Blazor Server model, protože je v restauraci limitovaný počet zaměstnanců a je spíše nepravděpodobné, že zde dojde k zatížení serveru. Naopak počet zákazníků může být potenciálně mnoho a proto je zde použit Blazor WebAssembly, kde uživatelského rozhraní a komponenty jsou vykresleny prohlížečem a kromě požadavků na web API se tak zbytečně nezahlcuje server.

2.2 Původní návrh

Mojí původní snahou bylo vytvořit aplikaci, která pomocí URL rozezná, kterých z hostitelských modelů použít – tedy pustit na klientovi Blazor Server nebo Blazor WebAssembly. Rozdělil jsem aplikaci z pohledu uživatele, serverová část běží na správu samotné aplikace, např. stránky na správu produktů, správa kategorií, správa objednávek a vytvoření objednávky obsluhou. Na druhé straně se pustí klientům Blazor WebAssembly, kde si mohou zobrazit produkty, vytvořit objednávku s sebou nebo rozvoz a správa uživatelského účtu pomocí web API komunikace s back end serverem. Třetí část je jenom pomocná knihovna, kde se ukládají Data Transfer Objects (DTO).

Tento návrh fungoval dobře, ale bohužel jsem tu narazil na problém. Jelikož hlavní náplní této práce není řešit identitu, autentizaci a autorizaci, použil jsem Microsoft knihovnu ASP.NET Core Identity UI, která vygeneruje nový datový model uživatele, potřebné stránky a uživatelské rozhraní, aby se mohl uživatel registrovat, přihlásit se a spravovat svůj účet. Tato knihovna od Microsoftu je plně funkčním řešením, která vývojářům ulehčí značnou práci, ale funguje jen v případě, že používáte jeden nebo druhý hostitelský model zvlášť. Problém je, že při generování IdentityServer pomocí knihovny ASP.Net Core Identity se vytvoří konfigurace, která registruje profil pro klientskou část. Protože se zde používá knihovna, která je nadstavbou původního IdentityServer, nepodařilo se mi překopat konfiguraci tak, abych zaregistroval i serverovou část.

```
RestaurantAppSolution
├── RestaurantApp.Client
│   ├── Dependencies
│   ├── Properties
│   ├── wwwroot
│   ├── Pages
│   ├── Shared
│   ├── _Imports.razor
│   ├── App.razor
│   └── Program.cs
└── RestaurantApp.Server
    ├── Dependencies
    ├── Properties
    ├── wwwroot
    ├── Areas
    ├── Controllers
    ├── Data
    ├── DTO
    ├── Hubs
    ├── Migrations
    ├── Pages
    ├── Repositories
    ├── Services
    ├── Shared
    ├── _Imports.razor
    ├── App.razor
    ├── appsettings.json
    └── Program.cs
```

Výpis kódu 10 Původní návrh projektové struktury

2.3 Nový návrh

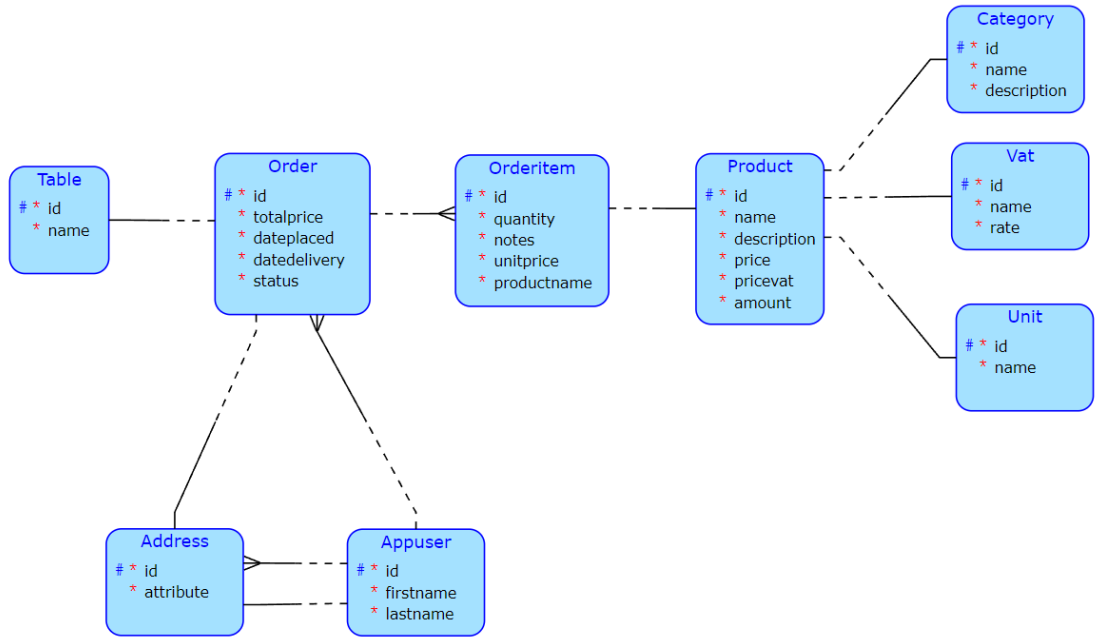
Pro vyřešení problému jsem se rozhodl napsat si vlastní IdentityServer, který je oddělený od serverové části i klientské části a poslouchá na jiném portu. Rozdělil jsem tedy projekt na více vrstev, serverová část, klientská WebAssembly část, IdentityServer část, doménová část a část pomocné sdílené knihovny. V doménové části se vyskytují třídy entit, třída kontextu a migrace, pomocná sdílená knihovna Shared obsahuje v sobě Data Transfer Objects. Obě tyto části jsou referencované jak v serverové části, tak klientské. Spustitelné budou právě 3 aplikace, serverová část pro správu, klientská část pro klienty a IdentityServer část pro autentizaci a autorizaci uživatelů. Každá z nich bude poslouchat na jiném portu, serverová část na 5001, klientská část na 5004 a IdentityServer na 5003.

```
RestaurantAppSolution
├── RestaurantApp.Client
├── RestaurantApp.Domain
├── RestaurantApp.Ids
├── RestaurantApp.Server
└── RestaurantApp.Shared
```

Výpis kódu 11 Nový návrh projektové struktury

2.4 Návrh databázového modelu

Obrázek 2.1 Entity-relationship model



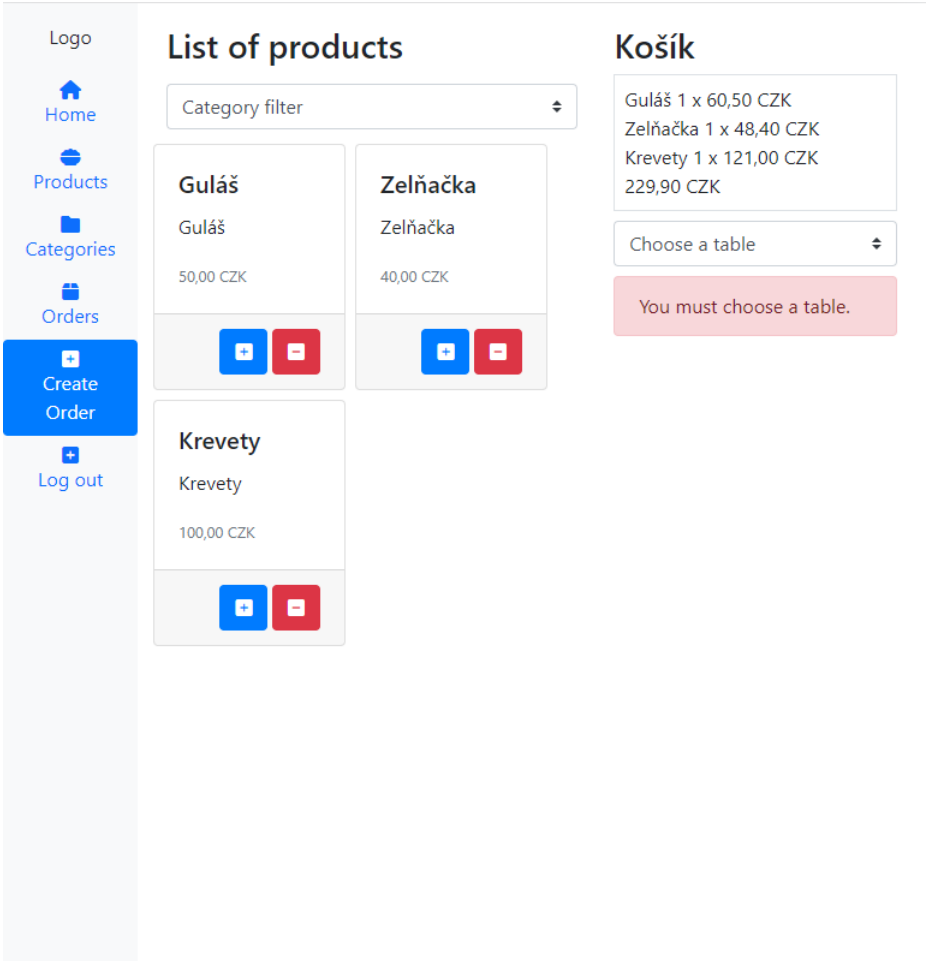
Na obrázku 2.1 je popsaná databáze pomocí entity-relationship diagramu, která v sobě ukládá informace o produktech, objednávek a uživatelích. Každá entita má atribut primárního klíče id. Relace jsou zvoleny tak, aby co nejblíže odpovídaly scénáři v reálném světě. Entita produkt je v relaci 1:1 s entitami kategorie, daně a jednotky. Entita OrderItem spojuje entity produkt a objednávka, každý řádek v tabulce drží informaci o zvoleném produktu, počtu, ceně a ke které objednávce patří. V objednávce se dále objevuje celková cena objednávky, datum a čas zadání a doručení, stav objednávky, kdo objednávku zadal a k jakému stolu nebo adrese objednávka patří. Entita uživatele má v sobě jen doplňující atributy křestního jména a příjmení, neboť jsem zvolil použít knihovnu ASP.Net Core Identity a třída uživatele dědí třídu IdentityUser, která v sobě nese ty důležité hlavní atributy, např. username, email, telefonní číslo, heslo atd.

2.5 Návrh uživatelského rozhraní

Pro vzhled stránky jsem si vybral velmi známou CSS knihovnu Bootstrap. Mnozí argumentují, že při použití této knihovny všechny stránky vypadají podobně. Jelikož nejsem webový designer a účelem této práce není grafika, volba byla jasná. Knihovna Bootstrap mi pomůže ve stylování uživatelského rozhraní tak, aby informace byly přehledné a ovládání intuitivní.

Na obrázku 2.2 je načrtnutý návrh prototypu, kde jsem se zaměřil na rozložení prvků. Pomocí Bootstrapu se dá lehce navrhnout lepší a hezčí design. Poněvadž aplikace bude hlavně využívat obsluha a kuchyň, předpokládá se, že primárním zařízením bude tablet. Tablety většinou operují ve vertikálním módu a proto jsem pro navigační menu zvolil sidebar, kam se potenciálně může vejít více prvků. Důležité je, aby uživatel poznal na které stránce se nachází, proto aktuální položka má vždy atribut active. Barevné rozložení tlačítek a alertů jsem nechal výchozí nastavení pro Bootstrap.

Obrázek 2.2 Příklad rozložení UI na stránce pro vytvoření objednávky



Kapitola 3
Implementace

3.1 Serverová část

Serverová část slouží jako back end pro komunikaci dat s klientskou částí aplikace. Stránky se správou a administrací jsou vytvořené pro zaměstnance restaurace a aplikace běží na hostitelském modelu Blazor Server. V této části je možné přihlásit se, admin má zde přístup ke komponentám pro správu produktů a kategorií, obsluha a kuchyň mají přístup ke komponentám pro tvorbu nové objednávky a správu objednávek. V této části je také napsaný web API, které zpřístupňuje zdroje z databáze klientské části.

3.1.1 Struktura

```
RestaurantAppSolution
├── RestaurantApp.Client
├── RestaurantApp.Domain
├── RestaurantApp.Ids
├── RestaurantApp.Server
│   ├── Dependencies
│   ├── Properties
│   ├── wwwroot
│   ├── Controllers
│   │   └── Api
│   ├── Hubs
│   ├── Migrations
│   ├── Pages
│   ├── Repositories
│   ├── Services
│   ├── Shared
│   ├── _Imports.razor
│   ├── App.razor
│   ├── appsettings.json
│   └── Program.cs
└── RestaurantApp.Shared
```

Výpis kódu 12 Struktura serverové části

3.1.2 API Kontrolery

Ve složce Controllers/Api najdeme soubory CategoryController.cs, OrderController.cs, ProductController.cs, TableController.cs, UserController.cs. Jsou rozdělené podle zdrojů, se kterými pracují. Třída kontrolerů dědí ze třídy Microsoft.AspNetCore.Mvc.Controller a pomocí dependency injection se přes konstruktor do kontrolerů vkládají příslušné servisy.

Ve výpisu kódu 13 OrderController.cs vidíme, že routování, autentizace a určení metody požadavku jsou řízeny pomocí atributů. Například [Route(”api/[Controller]”)] říká, že každý požadavek, který v sobě obsahuje URL /api/{název kontroleru z .cs souboru} směruje na funkce téhle třídy. V konstruktoru jsou předány rozhraní servisů pro dependency injection. Atributy [HttpPost] a [HttpGet] určí metodu HTTP požadavku. Pokud chceme více specifikovat URL požadavku, který chceme poslouchat, tak přidáme [HttpGet(”current”)], potom v tomhle případě to poslouchá na /api/order/current. Atribut [Authorize] autorizuje příchozí požadavek, v tomhle případě jsem u všech kontrolerů specifikoval, podle kterého schéma se autorizovat. V konfiguračním souboru Program.cs jsem defaultně zvolil schéma autentizace přes cookies pro serverovou část, ale pro web API endpointy používám autentizaci přes Bearer token, který je doporučeným schématem. Token nese v sobě informaci o přihlášeném uživateli a atribut [Authorize] rovnou vrací uživatele na serveru. Jelikož třída dědí ze třídy Controller, jsou tu implementované metody pro returny, například Ok a Created, které rovnou transformují objekt do JSON formátu.

```
namespace RestaurantApp.Server.Controllers.Api
{
    [Route("api/[Controller]")]
    public class OrderController : Controller
    {
        private readonly IOrderService orderService;
        ...

        public OrderController(IOrderService orderService,
            ...)
        {
            this.orderService = orderService;
            ...
        }

        [Authorize(AuthenticationSchemes = JwtBearerDefaults
        .AuthenticationScheme)]
        [HttpGet("current")]
        public async Task<IActionResult> GetOrdersCurrentUser()
        {
            List<OrderDto> result = new List<OrderDto>();

            ...

            return Ok(result);
        }
    }
}
```

Výpis kódu 13 OrderController.cs

3.1.3 Hubs

Ve složce Hubs najdeme soubor OrdersHub.cs. Třída dědí ze třídy Microsoft.AspNetCore.SignalR a slouží k odeslání signálů registrovaným klientům. Tato knihovna funguje na principu návrhového vzoru Observer. Knihovna SignalR je zde použita, protože při vytvoření objednávky obsluhou nebo klientem je nutné aktualizovat stránku s objednávkami v kuchyni bez obnovení stránky. V komponentě s objednávkami se ustanoví připojení k hubu, které poslouchá příchozí zprávy.

```
namespace BlazorServerSignalRApp.Hubs
{
    public class OrdersHub : Hub
    {
        public async Task SendMessage(string message)
        {
            await Clients.All.SendAsync("ReceiveMessage", message);
        }
    }
}
```

Výpis kódu 14 SignalR Hub OrdersHub.cs

```
hubConnection = new HubConnectionBuilder()
            .WithUrl(navigationManager.ToAbsoluteUri("/ordershub"))
            .Build();

hubConnection.On<string>("ReceiveMessage", async x =>
{
    orders = (await orderService.GetPlacedOrders()).ToList();
    await InvokeAsync(StateHasChanged);
});

        await hubConnection.StartAsync();
```

Výpis kódu 15 Připojení k hubu

3.1.4 Pages

Ve složce Pages najdeme vygenerované soubory _Host.cshtml a _Layout.cshtml, které jsou potřebné pro spuštění aplikace. Tyto soubory s příponou .cshtml nazýváme Razor stránkami a obsahují Razor syntaxi.

Soubory a nové stránky s příponou .razor, které jsme přidali nazýváme Razor komponentou nebo neformálně Blazor komponentou. Na ukázce souboru 16 v horní části vidíme direktivy, které začínají @ a jsou následovány klíčovými slovy. Nejvíc časté jsou @page, která vykreslí šablonu dané URL, poté @using pro namespaces a @inject pro použití dependency injection na servisy aplikace. Pro základní programování jsou direktivy @if a @foreach.

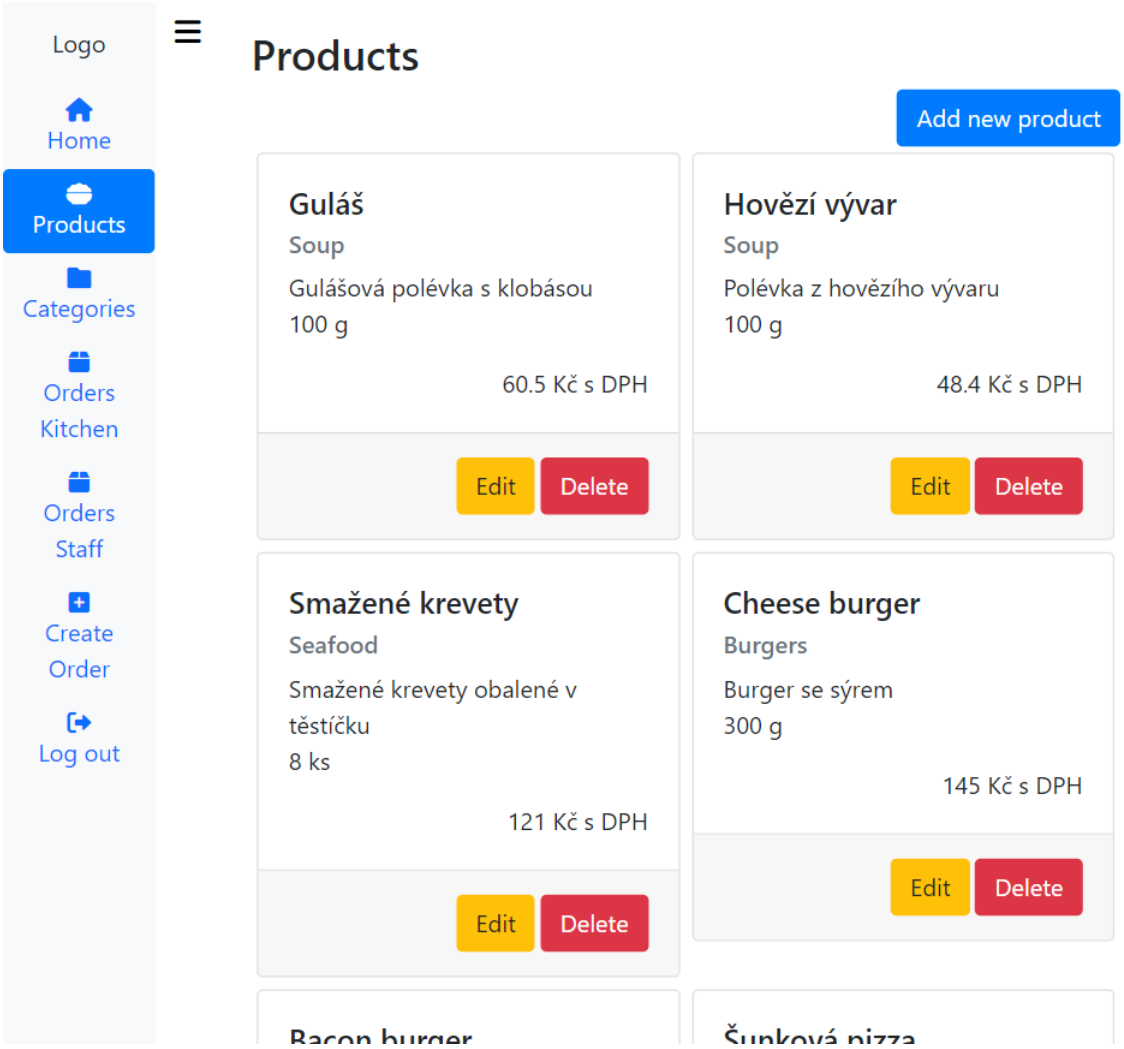
```
@page "/Products"
@using RestaurantApp.Domain.Models
@inject IProductService productService
...

    @if (products is not null)
    {
        @foreach (var item in products)
        {
            ...
        }
    }

@code {
    private IEnumerable<Product> products;
    ...

    protected override async Task OnInitializedAsync()
    {
        products = await productService.GetProducts();
        ...
    }
```

Výpis kódu 16 ManageProducts.razor

3.1.5 Správa produktů

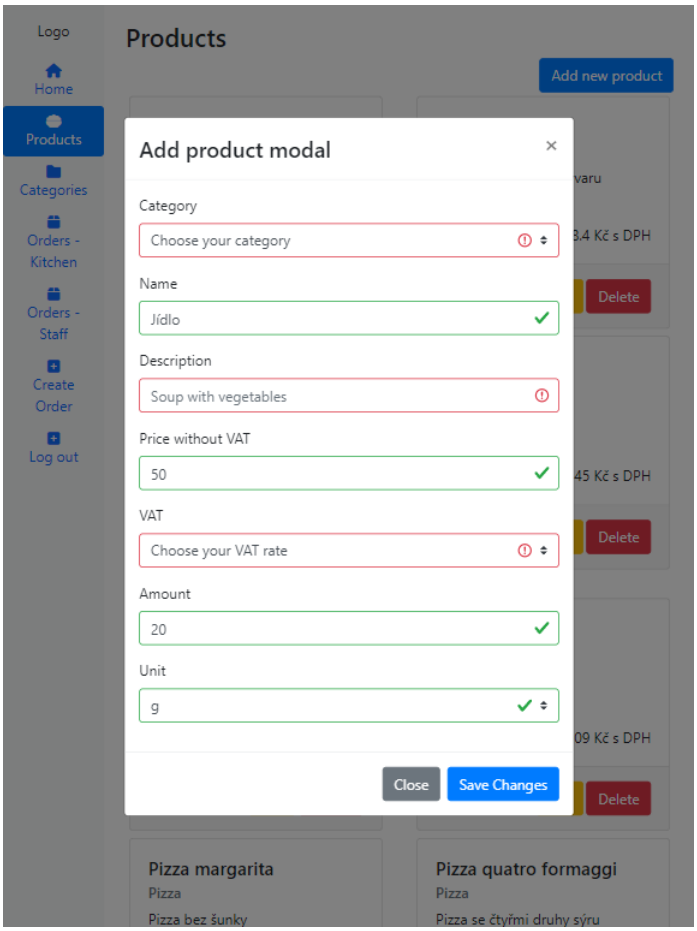
Komponenta pro správu produktů vykreslí stránku pro správu produktů. V komponentě je definován List produktů, který je v přepsané metodě OnInitializedAsync() načten metodou z productService. Pro výpis seznamu do HTML kódu je List proiterován foreach loopem a produkty jsou seskupené do jednotlivých bootsrapových karet, kde jsou vypsané informace společně s tlačítky pro úpravu a mazání. Nad seznamem je jedno tlačítko pro vytvoření nového produktu.

Obrázek 3.1 Ukázka stránky pro správu produktů



Do komponenty je přidán modální okno z knihovny Blazorise [30] s formulářem na vytvoření nového produktu, který se spustí po kliknutí na tlačítko. Zvolil jsem modální okno místo formuláře na jiné stránce, protože po odeslání formuláře se hned zobrazí nový produkt v celkovém seznamu. Formulář v okně je validován na klientovi. Kontroluje, jestli jsou vybrané nutné položky ze select listu, a že jsou vyplněné pole.

Obrázek 3.2 Ukázka modálního okna pro vytvoření nového produktu



3.1.6 Správa kategorií

Komponenta je napsaná stejně jako stránka správu produktů, akorát je zde místo produktu kategorie. Je opět definován List kategorií, který je naplněn při inicializaci stránky. Kategorie jsou poté vypsány do seznamu karet, které obsahují tlačítka pro úpravu a mazání. Nad seznamem se nachází tlačítko pro tvorbu nové kategorie.

Po stisknutí tlačítka se spustí modální okno s formulářem pro tvorbu nové kategorie. Formulář je také validován na klientovi.

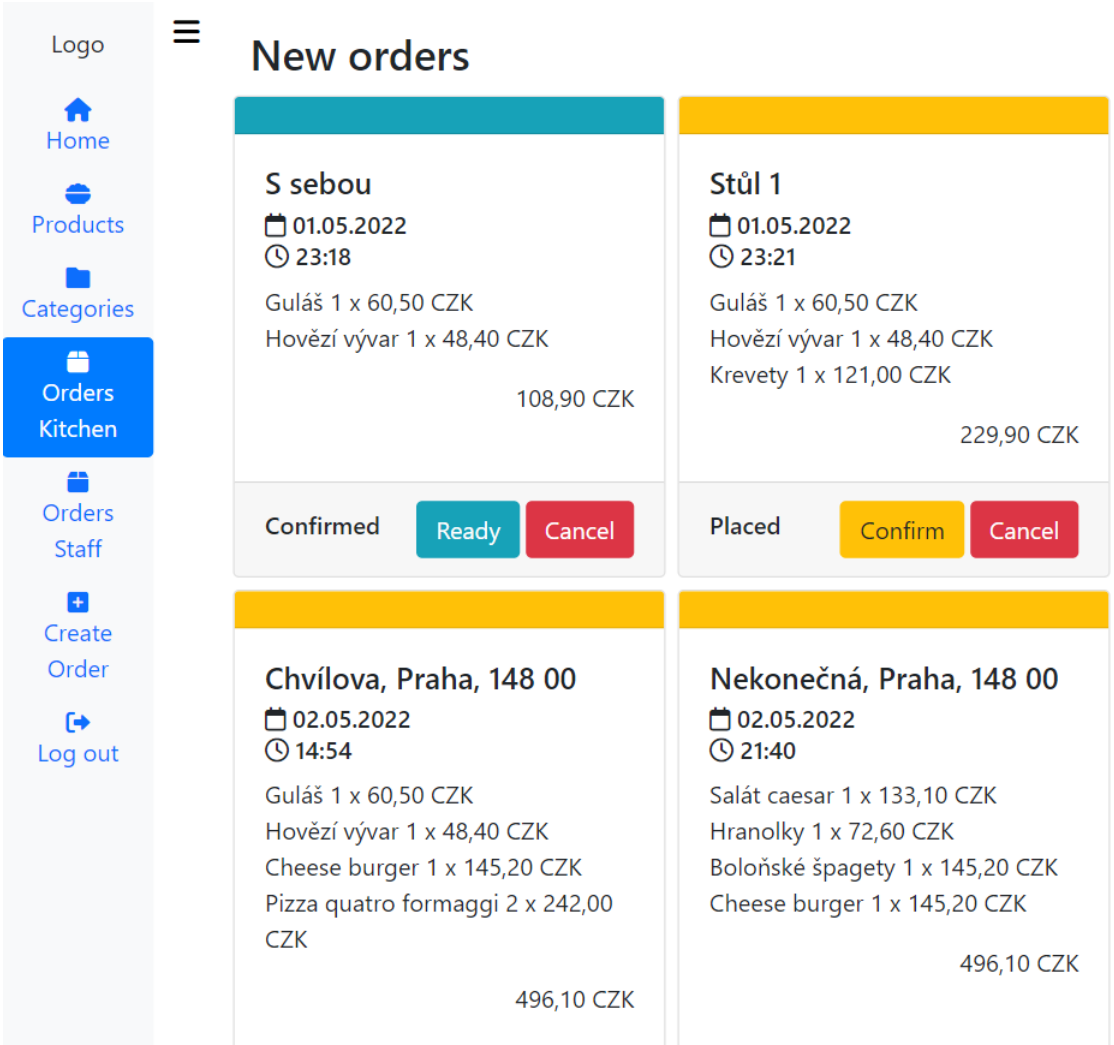
3.1.7 Příjem nových objednávek

Tato stránka slouží pro práci s novými příchozími objednávkami. Stránka bude hlavně používána kuchyní, která objednávky přijímá, potvrdí je a po zpracování je označí jako připravené, aby obsluha objednávku předala dál.

V komponentě je definován List objednávek, který je opět naplněn metodou z orderService při inicializaci stránky. V metodě servisy určím, aby vracela právě jen objednávky, které mají stav Placed a Confirmed. List objednávek proiteruji foreach loopem, kde objednávky seskupím do karet. Pro přehlednost obarvíme header část podle stavu. V těle karty vypíšu informace o stolu nebo doručovací adrese, čas vytvoření objednávky a samotné produkty s cenami. Ve footeru karty jsou tlačítka pro změny stavu nebo zrušení objednávky. Obě metody napojené na tlačítka znovu volají metodu pro naplnění Listu a metodu StateHasChanged, která celou komponentu znovu překreslí.

Důležité je, aby stránka se aktualizovala pokaždé, když přijde nová objednávka bez nutnosti na obnovení stránky. Této funkčnosti dosáhnu díky knihovny SignalR. Do komponenty zaregistruji připojení na Hub, který poslouchá signály. Pokud přijde nová objednávka, zavolá se znovu metoda pro naplnění Listu.

Obrázek 3.3 Ukázka stránky pro nové objednávky



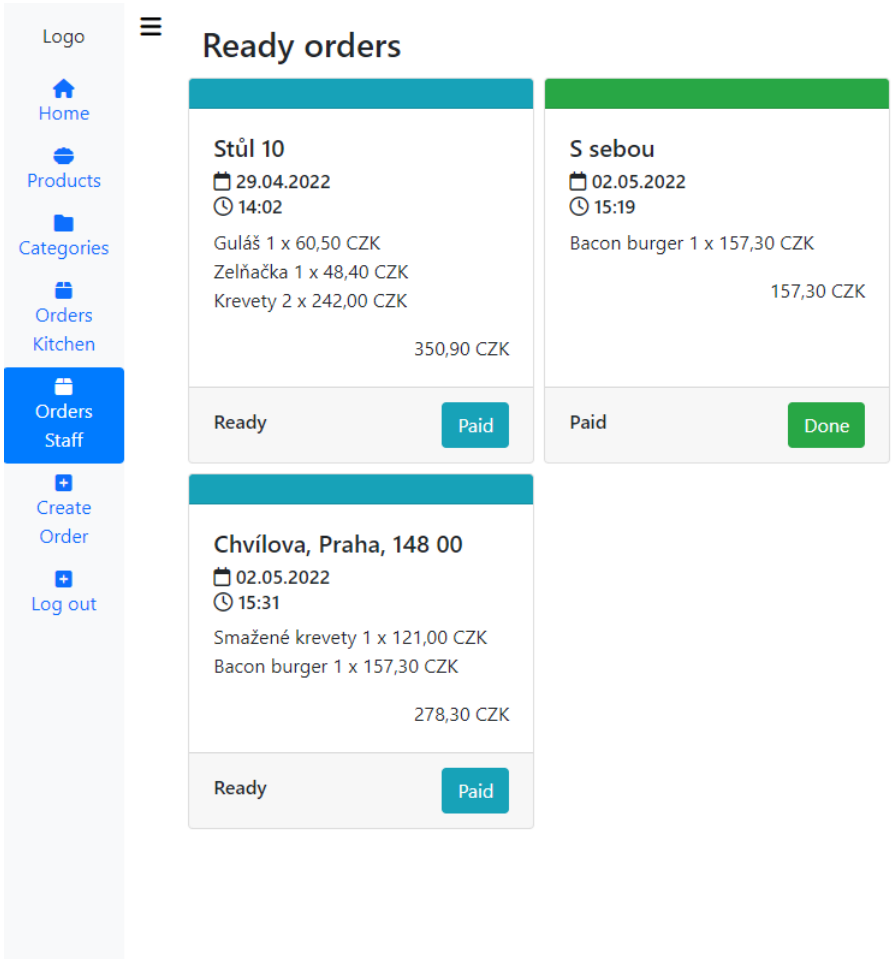
3.1.8 Správa připravených objednávek

Stránka je určena pro obsluhu restaurace, která hotové objednávky odnáší z kuchyně. Objednávky na této stránce jsou ve stavu připravené nebo zaplacené. Po zaplacení už zbývá jen odkliknout na změnu konečného stavu hotovo.

Opět je tu definován List objednávek, který je naplněn metodou z orderService při inicializaci stránky. Metoda pouze vrací takové objednávky, které mají stav Ready a Paid. List objednávek se proiteruje a vypíše se seznam objednávek uvnitř karet, které jsou barevně odlišné v headeru karty. Vypsané informace objednávky se neliší od komponenty pro příjem nových objednávek. Tlačítka na změny stavů volají metodu v orderService, která aktualizuje objednávku. Poté se zavolá metoda, která vrací objednávky a metodu StateHasChanged() pro překreslení.

I zde je zaregistrováno připojení na Hub, který poslouchá signály. Signál je odeslán, právě tehdy když kuchyň uvaří jídlo a označí objednávku jako připravenou. Znovu zde voláme metody pro načtení objednávek a překreslení.

Obrázek 3.4 Ukázka stránky pro správu produktů



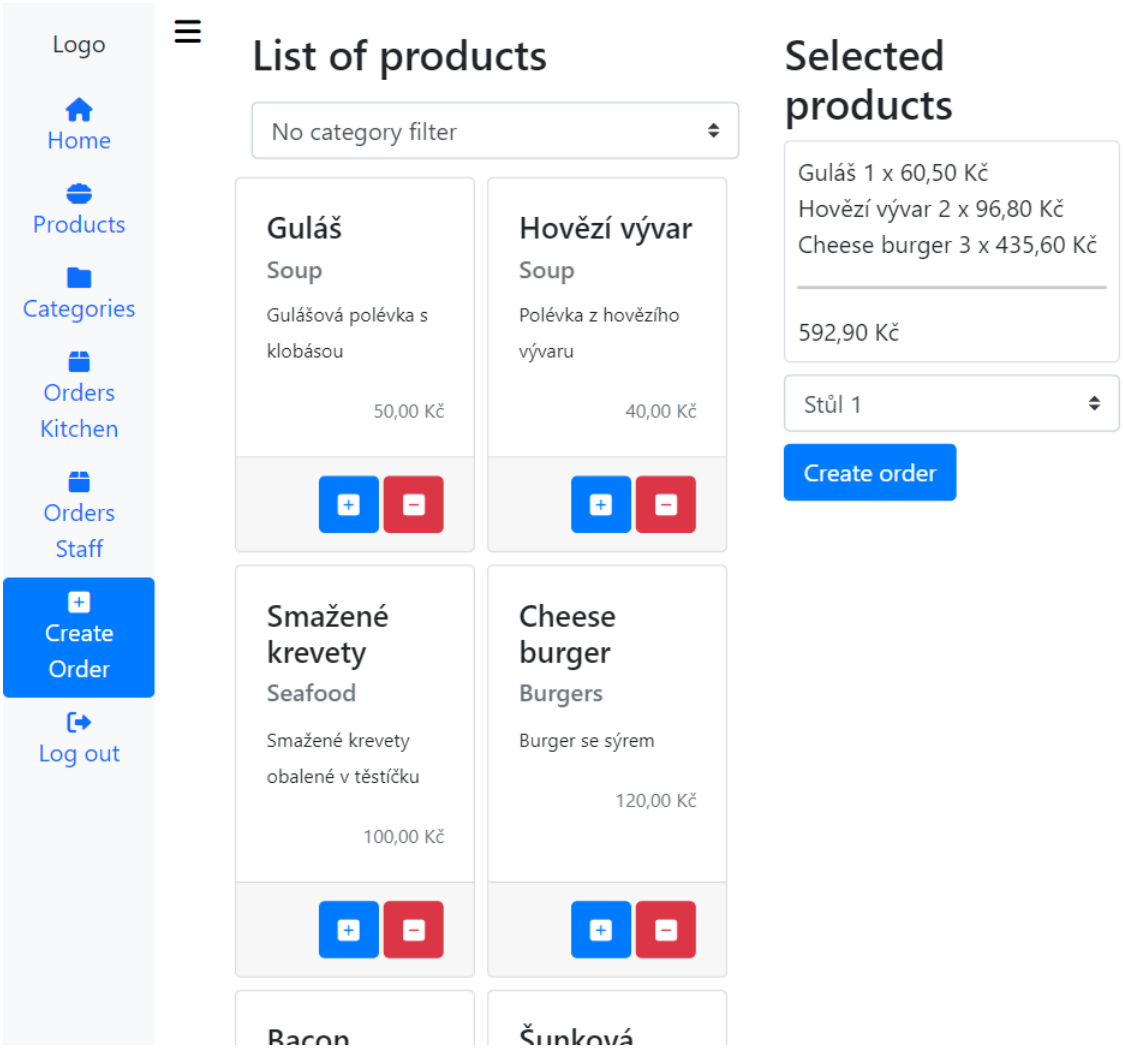
3.1.9 Tvorba nové objednávky

Stránka slouží pro tvorbu objednávek v restauraci. Skládá se ze dvou částí – ze seznamu produktů a košíku. Obsluha příjme objednávku od zákazníků a rovnou nakliká do košíku vybrané produkty.

Po vybrání, ke kterému stolu objednávka patří se objeví tlačítko, které objednávku vytvoří a pošle notifikaci do komponenty pro příjem nových objednávek. Po vytvoření se košík vyprázdní, aby obsluha mohla rovnou vytvořit novou objednávku. Validace je pouze na straně klienta, kontroluje se, jestli není košík prázdný nebo jestli není vybraný stůl.

V komponentě je definován List produktů a List jednotlivých položek orderItems. List produktů se při inicializaci stránky načte z productService a ve foreach loopu se vypíše do seznamu karet. Karty také obsahují tlačítka, které přidávají nebo ubírají věci z košíku. Při přidání se vytvoří nový objekt orderItems se správným produktem a přidá se do Listu orderItems. Tento List je vypsán právě v košíku, společně s počtem kusů a cenou. Při odebrání se z Listu položky mažou a zde je kontrola, jestli položka vůbec v Listu existuje. Na konci seznamu je přidána celková cena, která se postupně přepočítává při přidání a odebírání položek.

Obrázek 3.5 Ukázka stránky pro tvorbu objednávek



3.1.10 Repositories

Ve složce Repositories najdeme rozhraní a samotné implementace repositories jednotlivých zdrojů. Tyto repositories slouží ke práci s daty z databáze, nejběžnější jsou operace CRUD. Ve zdrojo-

vém kódu 17 v konstruktoru injektujeme kontextovou třídu databáze, která v sobě uchovává DbSety zdrojů. K datům se dotazujeme přes kontextovou třídu a DbSet, kde použijeme .NET integrovaný jazyk pro dotazovaní LINQ.

```
namespace RestaurantApp.Server.Repositories
{
    public class ProductRepository : IProductRepository
    {
        private readonly AppDbContext appDbContext;

        public ProductRepository(AppDbContext appDbContext)
        {
            this.appDbContext = appDbContext;
        }

        public async Task<IEnumerable<Product>> GetProducts()
        {
            return await appDbContext.Products.ToListAsync();
        }

        public async Task<Product> GetProduct(int productId)
        {
            return await appDbContext.Products
            .FirstOrDefaultAsync(x => x.Id == productId);
        }

        ...
    }
}
```

Výpis kódu 17 Repository ProductRepository.cs

3.1.11 Services

Ve složce Services najdeme rozhraní a jednotlivé implementace servis aplikace. Narozdíl od repositories, které drží informace o zdrojích a jsou zde jen základní CRUD operace, v servisách je napsaná business logika, která pracuje s daty. V kódu 18 v konstruktoru injektujeme appUserRepository, kde voláme definované CRUD operace. V případě téhle servisy jsou tu navíc metody AddDeliveryAddress a AddBillingAddress, které přidají nové adresy do databáze, pokud ještě v databázi neexistují a propojí se správnou relací.

```
namespace RestaurantApp.Server.Services
{
    public class AppUserService : IAppUserService
    {
        private IAppUserRepository appUserRepository;
        public AppUserService(IAppUserRepository appUserRepository)
        {
            this.appUserRepository = appUserRepository;
        }

        ...

        public async Task AddDeliveryAddress(AppUser appUser, Address address)
        {
            if (appUser.DeliveryAddress is null)
                appUser.DeliveryAddress = new List<Address>();

            appUser.DeliveryAddress.Add(address);
            await appUserRepository.UpdateAppUser(appUser);
        }
    }
}
```

Výpis kódu 18 Service ProductService.cs

3.1.12 Shared

V této složce se nacházejí komponenty, které jsou sdílené v této části aplikace. Komponenty slouží jako layout šablony. Hlavní komponentou je MainLayout.razor, ve kterým je definován sidebar a vykresluje na každé stránce. Tato komponenta je stylována vlastními CSS pravidly v souboru MainLayout.razor.css. Další komponentou je LoginDisplay.razor, která vykresluje odkazy na Login nebo Logout stránku na základě toho, jestli je uživatel přihlášen.

3.1.13 App

Soubor App.razor je kořenovou komponentou aplikace, která nastavuje routing na klientské straně pomocí Router komponenty. Router komponenta zachycuje navigace prohlížeče a následně vykreslí stránku dle zadané URL. [31]

```
<CascadingAuthenticationState>
    <Router AppAssembly="@typeof(App).Assembly">
        <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData"
            DefaultLayout="@typeof(MainLayout)" />
            <FocusOnNavigate RouteData="@routeData" Selector="h1" />
        </Found>
        <NotFound>
            <PageTitle>Not found</PageTitle>
            <LayoutView Layout="@typeof(MainLayout)">
                <p role="alert">Sorry, there is nothing at this address.</p>
            </LayoutView>
        </NotFound>
    </Router>
</CascadingAuthenticationState>
```

Výpis kódu 19 App.razor

3.1.14 appsettings.json

Soubor appsettings.json je konfigurační a obsahuje nastavení pro aplikaci. Týká se spíše prostředí, jak se má aplikace pustit, connection string pro databázi a nastavení pro IdentityServer.

3.1.15 Program.cs

Soubor Program.cs je vstupním bodem aplikace, který nastavuje ASP.NET Core hostitele a obsahuje logiku spuštění aplikace, a to zejména registrací servis (a repositories) a konfigurace pipeliny pro zpracování požadavků. Servisy se registrují pro použití dependency injection. Následně jsou volány metodou AddServerSideBlazor a jednotlivé servisy jsou přidány do servisních kontejnerů používány komponentami. Metoda MapBlazorHub nastavuje endpoint pro připojení v reálném čase s prohlížečem. Připojení je vytvořeno pomocí knihovny SignalR, framework, který přidá webové real-time funkcionality. Metoda MapFallBackToPage("_Host") nastavuje kořenovou stránku aplikace Pages/_Host.cshtml a umožňuje navigaci v aplikaci. [31]

3.2 Klientská část

Klientská část aplikace je určena pro zákazníky restaurace. Aplikace běží na hostitelském modelu Blazor WebAssembly. V této části je možná registrace, přihlášení a odhlášení uživatele. Zákazník si zde může prohlédnout vytvořené menu a její produkty. Hlavní náplní je samozřejmě vytvoření nové objednávky a přístup ke svým starým a aktuálním objednávkám.

3.2.1 Struktura

```
RestaurantAppSolution
├── RestaurantApp.Client
│   ├── Dependencies
│   ├── Properties
│   ├── wwwroot
│   │   ├── css
│   │   ├── appsettings.json
│   │   ├── index.html
│   │   └── ...
│   ├── Pages
│   │   ├── MyOrders.razor
│   │   ├── CreateOrderClient.razor
│   │   ├── FinishCreateOrderClient.razor
│   │   └── ...
│   ├── Services
│   ├── Shared
│   │   ├── MainLayout.razor
│   │   └── LoginDisplay.razor
│   ├── _Imports.razor
│   ├── App.razor
│   └── Program.cs
├── RestaurantApp.Domain
├── RestaurantApp.Ids
├── RestaurantApp.Server
└── RestaurantApp.Shared
```

Výpis kódu 20 Struktura klientské části

3.2.2 Web Root

Složka Web Root obsahuje veřejné statické soubory aplikace, včetně appsettings.json, který obsahuje konfiguraci a nastavení prostředí aplikace. Webová stránka index.html je kořenovou stránkou aplikace implementovaná jako HTML stránka. Pokaždé, když je nějaká stránka aplikace požadována, stránka index.html je vrácena a vykreslena. Na této stránce je specifikováno, kde kořenová komponenta App.razor je vykreslená, a to uvnitř elementu div s atributem id a hodnotou app – `<div id="app"> Loading... </div>`. [31]

3.2.3 Pages

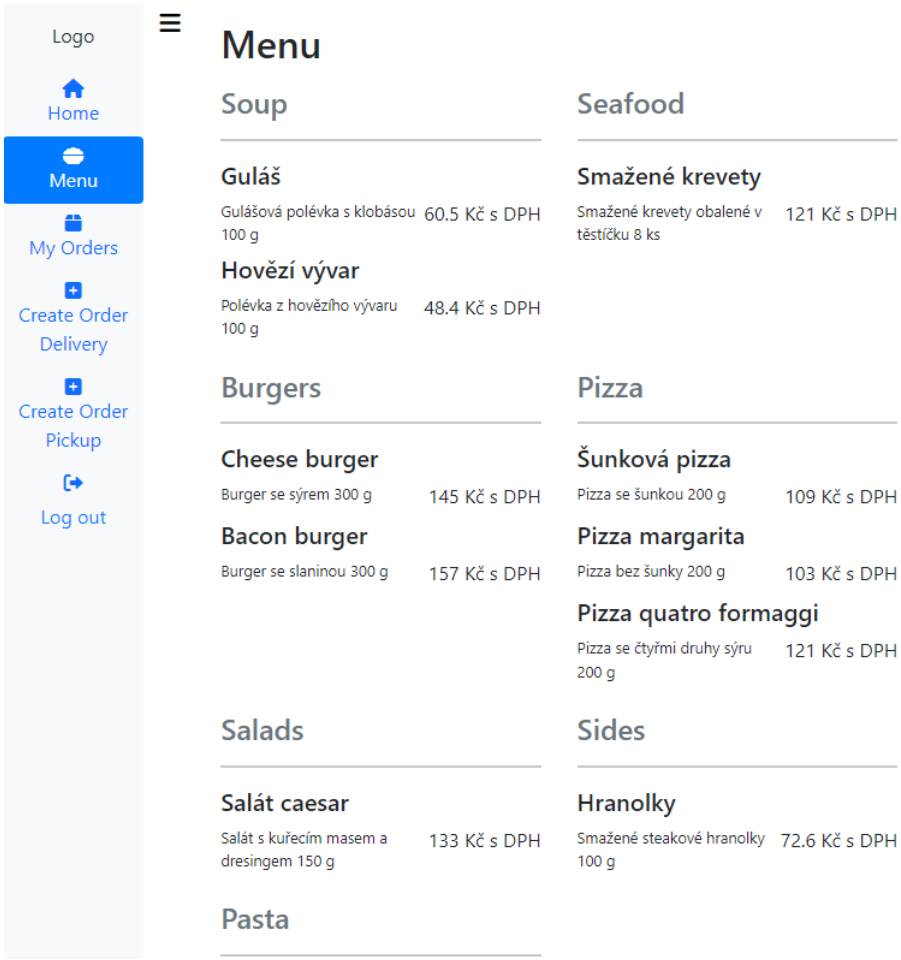
V této složce najdeme Blazor komponenty, které tvoří stránky aplikace. Komponenty v klientské části se liší od serverové části tím, že zde nemají přístup k aplikačním servisám, takže komunikace dat probíhá přes web API. Posíláme HTTP požadavky pomocí knihovny System.Net.Http a třídy HttpClient, který je nakonfigurovaný v souboru Program.cs. Odpověď požadavku vrací objekty v JSON formátu a metoda GetFromJsonAsync umí transformovat JSON odpověď do objektů. HTTP požadavky posíláme na naší serverovou část na URL /api/resource.

3.2.4 Bezkontaktní jídelní menu

Stránka slouží jako výpis produktů v menu. Vzhled stránky je inspirován menu, které jsou podané v restauracích. Nejdříve je menu rozdělené do kategorií a poté pod každou kategorii je výpis produktů. Je důležité zde i vypsat popis a gramáž jídla, samozřejmě se nesmí zapomenout na cenu. Stránka je přístupná pro zákazníky restaurace.

V komponentě je definován List kategorií a List produktů. Tyto listy jsou naplněny voláním HTTP požadavků na API kontrolery v serverové části, které v odpovědi vrací požadované objekty. Výpis je proveden iterováním Listu kategorií a pro každou kategorii iterovat List produktů, kde se vypíše produkt se shodnou kategorií.

Obrázek 3.6 Ukázka stránky menu

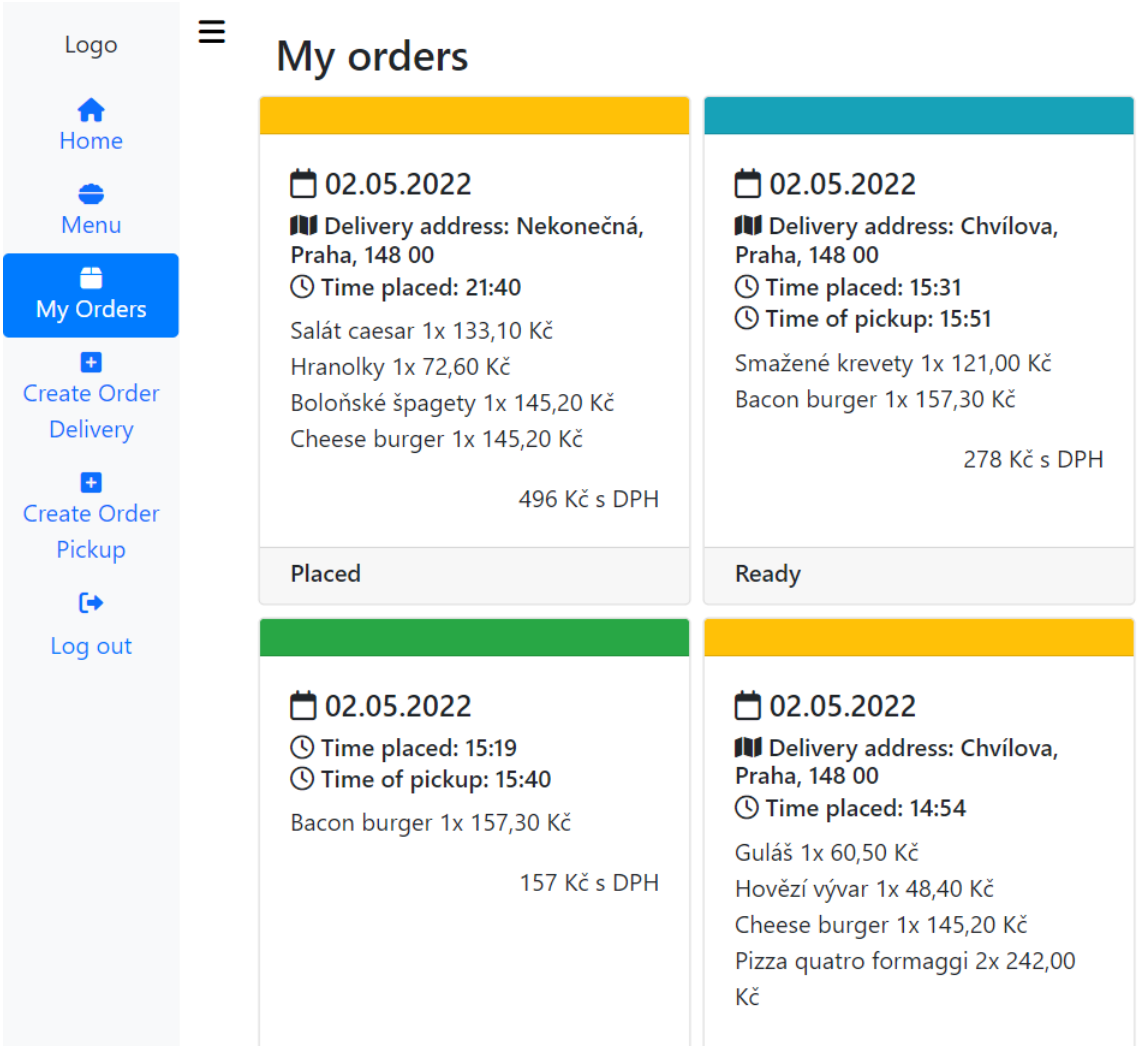


3.2.5 Moje objednávky

Stránka obsahuje objednávky přihlášeného uživatele. Objednávky jsou opět barevně rozlišitelné podle jejich stavu. V každé objednávce jsou obsaženy informace o datu, času vytvoření objednávky, adresy doručení, čas doručení a samotné objednané položky s celkovou cenou. Stránka je přístupná pouze přihlášeným uživatelům.

V komponentě je definován List objednávek, který je naplněn voláním HTTP požadavku na API OrderController. V požadavku je v hlavičce poslán JWT token, pomocí kterého se na serveru provede autentizace a v odpovědi vrací objednávky přihlášeného uživatele. Výpis je proveden iterací Listu objednávek.

Obrázek 3.7 Ukázka stránky moje objednávky

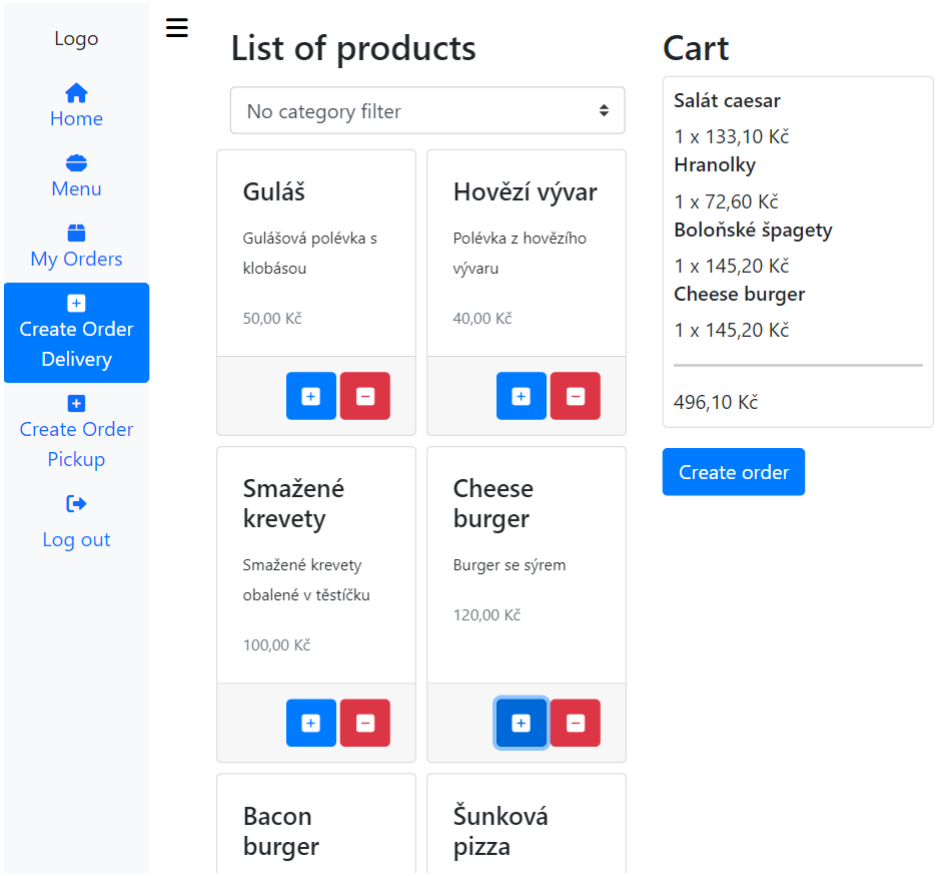


3.2.6 Tvorba objednávky s doručením

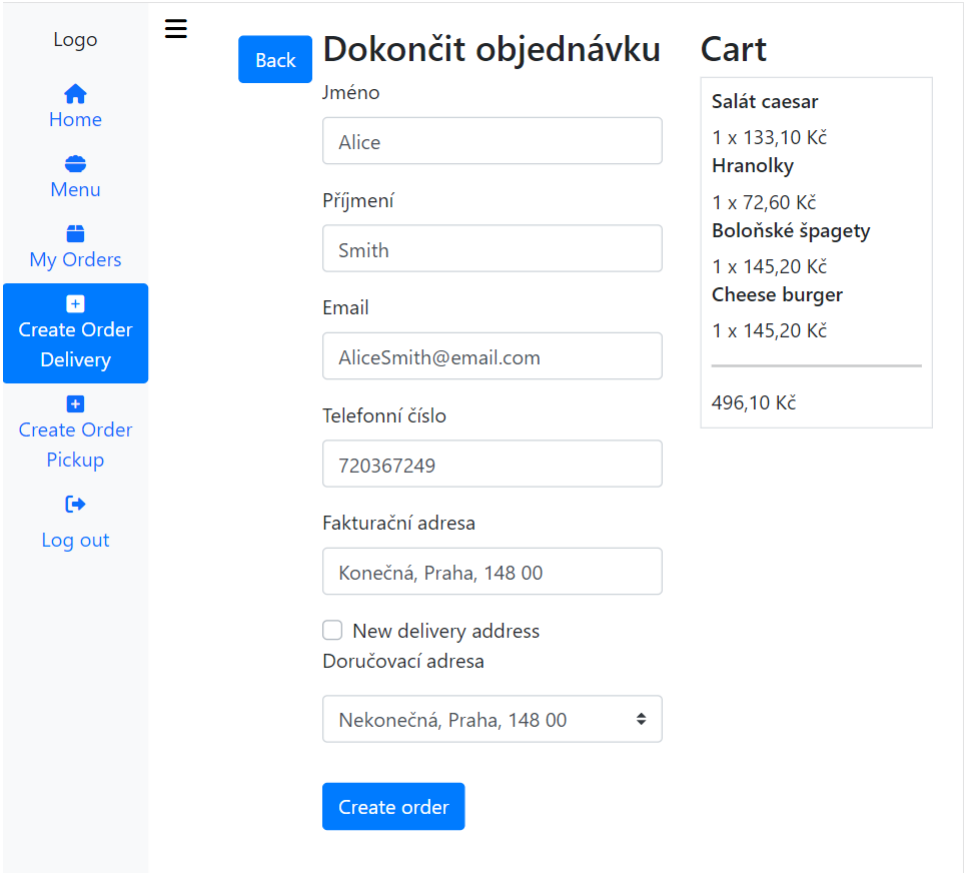
Stránka slouží pro tvorbu nových objednávek s doručením pro přihlášené uživatele. Je zde použit stejné uživatelské rozhraní, jako v komponentě v serverové části. Po vybrání jídel a stisknutí tlačítka pro tvorbu nové objednávky se vykreslí další komponenta, kde se vyplňují informace o uživateli. Tyto informace jsou předem vyplněné a dají se změnit.

V rodičovské komponentě je nastavený bool, který ovládá vykreslení vnořené potomkové komponenty. Při stisknutí tlačítka se tento bool nastaví na true a vykreslí se vnořená komponenta. Uvnitř je definován objekt AppUsera, který se načte z HTTP požadavku na API UserController. Informace o uživateli uložené v proměnných jsou poté předány bindingem jednotlivým prvkům formuláře. Po validaci na klientské části se posílá HTTP Post požadavek s uživatelem, adresami a samotnou objednávkou. Server tento požadavek zpracuje a posílá SignalR notifikaci všem posluchačům.

Obrázek 3.8 Ukázka stránky tvorba objednávky s doručením



Obrázek 3.9 Ukázka stránky dokončení tvorby objednávky s doručením



3.2.7 App v klientské části

App.razor je kořenovou komponentou klientské aplikace, která nastartuje routing v prohlížeči u klienta pomocí Router komponenty. Tato Router komponenta zachycuje navigace prohlížeče a vykresluje stránky dle zadané URL. [31]

3.2.8 Program.cs

Program.cs je vstupním bodem klientské části aplikace, který nastavuje WebAssembly a App.razor komponentu, která je v souboru wwwroot/index.html do kolekce kořenových komponentů `builder.RootComponents.Add<App>("#app")`. Dále se tu registrují různé servisy pro dependency injection, podobně jako v serverové části [31]. Příkladnou servisou, kterou jsem přidal je HttpClient, který jsem musel překonfigurovat. Cílem je, aby v každém poslaném HTTP požadavku pomocí tohoto HttpClientu se posílal Bearer token pro autentizaci a jelikož mám klientskou část oddělenou od serverové části, integrovaný HttpMessageHandler nepřikládá Bearer token do headeru požadavků posílané do jiné URL. Z toho důvodu jsem si napsal vlastní AuthorizationMessageHandler třídu, do které jsem přidal URL serverové části.

```
var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

// Blazorise
builder.Services
    .AddBlazorise(options =>
    {
        options.Immediate = true;
    })
    .AddBootstrapProviders()
    .AddFontAwesomeIcons();

builder.Services.AddScoped<CustomAuthorizationMessageHandler>();

builder.Services.AddHttpClient("WebAPI",
client => client.BaseAddress = new Uri("https://localhost:5001"))
    .AddHttpMessageHandler<CustomAuthorizationMessageHandler>();

// Supply HttpClient instances that include access tokens
when making requests to the server project
builder.Services.AddTransient(sp =>
   sp.GetRequiredService<IHttpClientFactory>().CreateClient("WebAPI"));

builder.Services.AddOidcAuthentication(options =>
{
    ...
});

await builder.Build().RunAsync();
```

Výpis kódu 21 Klientská část Program.cs

3.3 IdentityServer

IdentityServer je autentizační server, který implementuje OpenID Connect (OIDC) a OAuth 2.0 standardy pro mojí ASP.NET Core aplikaci. Poskytuje společný způsob, jak autentizovat i autorizovat žádosti všech aplikací, ať to je webová, nativní, mobilní aplikace nebo API endpointy. Také se tu nabízí možnost implementace Single Sign-On (SSO) pro více aplikací, ale ten v mé webové aplikaci není používán. [32] Nejběžnější formou autentizace je přes login formulář, který po úspěšném vyhodnocení identity uživatele mu poskytne identity token. Tento token má formát JSON Web Tokenu (JWT), který je šifrovaný a obsahuje informace o uživateli, serveru, způsobu přihlášení a více.

Od vydání .NET 6 se přešlo z původního IdentityServer4 na Duende IdentityServer, který má nový duální licencování. Pro účely vývoje, testování a učení je software zdarma v nekomerčním scénáři a také je zdarma, pokud v komerčním nasazení je roční zisk menší než jeden milion dolarů. V opačném případě je nutné zaplatit poplatek za licenci.

3.3.1 Struktura

```
RestaurantAppSolution
├── RestaurantApp.Client
├── RestaurantApp.Domain
├── RestaurantApp.Ids
│   ├── Dependencies
│   ├── Properties
│   ├── wwwroot
│   ├── keys
│   ├── Pages
│   ├── appsettings.json
│   ├── Config.cs
│   └── Program.cs
├── RestaurantApp.Server
└── RestaurantApp.Shared
```

Výpis kódu 22 Struktura klientské části

3.3.2 Keys

Obsahem složky keys jsou šifrovací klíče pro šifrování identity tokenů, JWT tokenů, logout tokenů atd... Výchozí nastavení generuje RSA klíče pro šifru RS256. Klíče jsou automaticky vyměněny každých 90 dní, výměny jsou hlášeny 14 dní dopředu a klíče jsou uchované dalších 14 dní. [33]

3.3.3 Pages

Ve složce Pages jsou převzaté stránky z GitHub repositáře pro Duende dokumentaci pro integraci IdentityServeru s ASP.NET Core Identity [34]. Stránky slouží hlavně pro Login a Logout a používají se zde ASP.NET Core Identity šablony. Tyto stránky jen vzhledově upravím a přidělám stránku pro registraci.

3.3.4 Config.cs

V souboru Config.cs se definují data, které jsou potřebné ke spuštění IdentityServeru a jsou poté využity v souboru Program.cs. Na prvním místě se zde definují Identity Resources, které jsou jen pojmenované skupiny claimů o uživatelích. Následně tu jsou API Scopes, které definují, jestli má uživatel přístup k určitým zdrojům. API Resources umožní lepší organizaci, seskupení, izolaci nebo průnik nastavení scopů. Nejdůležitější je definovat klienty. Klientem je každá aplikace, která posílá požadavky o tokeny z IdentityServeru.

3.3.5 Program.cs

I zde je soubor Program.cs vstupním bodem IdentityServeru. Je zde nutné přidat servisy do dependency injection systému zavoláním metody AddIdentityServer. Poté musíme přidat Duende IdentityServer middleware zavoláním UseIdentityServer.

3.4 Doménová část

Doménová vrstva reprezentuje databázi a její entity. Obsahuje v sobě kontextovou třídu databáze AppDbContext.cs, vygenerované migrace a třídy modelů entit. Tato vrstva je referencovaná serverovou částí a Identity serverem.

3.4.1 Struktura

```
RestaurantAppSolution
├── RestaurantApp.Client
├── RestaurantApp.Domain
│   ├── Dependencies
│   ├── Data
│   │   └── AppDbContext.cs
│   ├── Migrations
│   └── Models
│       ├── Address.cs
│       ├── AppUser.cs
│       ├── Category.cs
│       ├── Order.cs
│       ├── OrderItem.cs
│       ├── Product.cs
│       ├── Table.cs
│       ├── Unit.cs
│       └── VAT.cs
├── RestaurantApp.Ids
├── RestaurantApp.Server
└── RestaurantApp.Shared
```

Výpis kódu 23 Struktura klientské části

3.4.2 Data/AppDbContext.cs

Kontextová třída AppDbContext.cs dědí ze třídy IdentityDbContext<AppUser>, která je součástí knihovny Identity. Ve třídě jsou inicializovány DbSety entit zdrojů, které obsahují řádky tabulky databáze. V této třídě můžeme přepsat metodu OnModelCreating a přidat sem počáteční data. Metody kontrolují, jestli takový řádek už existuje a jestli ne, tak se vytvoří.

```
namespace RestaurantApp.Domain.Data
{
    public class AppDbContext : IdentityDbContext<AppUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
        {
        }

        public DbSet<Product> Products { get; set; }
        public DbSet<Order> Orders { get; set; }
        ...

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            builder.Entity<VAT>().HasData(
                new VAT { Id = 1, Name = "Normal", Rate = 21 });
                ...

            builder.Entity<Unit>().HasData(
                new VAT { Id = 1, Name = "ks" });
                ...

            builder.Entity<Table>().HasData(
                new VAT { Id = 1, Name = "Stůl 1" });
                ...
        }
    }
}
```

Výpis kódu 24 AppDbContext.cs

3.4.3 Models

Ve složce Models se objevují třídy entit. Ve výpisu kódu 25 vidíme modelování entity databáze, které se řídí podle konvencí Entity Framework Core. EF Core umí například rozeznat primární klíče, které ve svém názvu končí Id nebo i relace mezi entitami. [35]

```
using System.ComponentModel.DataAnnotations;

namespace RestaurantApp.Domain.Models
{
    public enum Status
    {
        Placed,
        Finished,
        Confirmed,
        Canceled,
        Paid,
        Done
    }
    public class Order
    {
        public int Id { get; set; }
        public decimal TotalPrice { get; set; }
        public DateTime DatePlaced { get; set; }
        public DateTime DateDelivery { get; set; }
        public ICollection<OrderItem>? OrderItems { get; set; }
        public Status Status { get; set; }
        public Address? BillingAddress { get; set; }
        public Address? DeliveryAddress { get; set; }
        public Table? Table { get; set; }
        public AppUser? AppUser { get; set; }
    }
}
```

Výpis kódu 25 Models/Order.cs

3.5 Pomocná knihovna Shared

Tato knihovna je pouze pomocná a slouží hlavně pro držení Data Transfer Objects (DTO). Knihovna je referencovaná serverovou a klientskou částí, kde při komunikaci přes web API dochází ke konverzi objektů do JSON formátu. Abychom se vyvarovali serializaci složitých objektů, cyklům a přenosu citlivých nebo nechtěných dat, serializujeme místo toho tyto Data Transfer Objects a posíláme jejich JSON formu v požadavcích a odpovědích.

3.5.1 Struktura

```
RestaurantAppSolution
├── RestaurantApp.Client
├── RestaurantApp.Domain
├── RestaurantApp.Ids
├── RestaurantApp.Server
└── RestaurantApp.Shared
    ├── Dependencies
    └── Data
        ├── AddressDto
        ├── CategoryDto
        ├── OrderDto
        └── ...
```

Výpis kódu 26 Struktura Shared knihovny

Kapitola 4
Testování

4.1 Manuální testování aplikace

Při psaní a vývoji jsem aplikaci manuálně postupně testoval. Testování probíhalo na navrženém uživatelském rozhraní i na funkčnosti kódu aplikace. Na konzultacích jsem dával vedoucímu otestovat průběžné implementace prototypu, například správu produktů a kategorií. Tam se zjistilo, že je nutné při mazání jednotlivých položek s vazbami na jiné entity dát pozor, aby objekt neměl referenci na null objekt. Tato situace je možná, poněvadž navržená databáze obsahuje entity s nullable vlastnostmi. Výchozí nastavení Entity Framework Core zajišťuje, že při mazání položek dochází ke kaskádovému odstranění. Problém byl ale ve výpisu v jednotlivých komponentách, ve kterých jsem nejprve nekontroloval, jestli jsou objekty null. Úprava tedy byla jednoduše vyřešena pomocí ifů. Důležité také je, aby se kontrolovaly všechny objekty inicializované v metodě OnInitializedAsync(), která se volá, až po vykreslení stránky a metoda znovu stránku překreslí.

Díky Chrome DevTools bylo možné otestovat responzivitu uživatelského rozhraní na simulacích různých zařízení. Nejvíce mě zajímalo, jak vypadá aplikace na tabletovém zařízení, jelikož tablety jsou preferované v práci v restauracích. Tam se naštěstí vešel navigační sidebar a nezabíral tolik místa, aby narušoval jiné prvky v aplikaci. Avšak problémovým zařízením byl mobilní telefon. Na obrazovku se bohužel nevejde sidebar a naopak ho škálovat by vedlo k miniaturní navigaci. Řešením je tedy nastavit jiná CSS pravidla pro vykreslení sidebaru pro určité prahy šířky obrazovky. Pro mobilní telefony je navigace umístěná nahoře a vykreslená jako seznam, který je skrývatelný tlačítkem. Výpisy karet a seznamů jsou řešeny pomocí Bootstrapové knihovny a jejích responzivních tříd rows a columns.

Při testování formulářů jsem nejprve zadával předpokládané vstupy, které procházely. Poté jsem se snažil cíleně aplikaci shodit neplatnými nebo chybějícími vstupy. Některé vstupy byly vyřešeny pomocí validací z knihovny Blazorise. Avšak například u formuláře pro tvorbu objednávek nesmí být košík s vybranými produkty prázdný při odeslání formuláře. Řešením bylo skrytí nebo disable tlačítka, pokud byl seznam vybraných produktů prázdný.

4.2 Programové testy

Tato sekce pojednává o podrobení kódu ze serverové části webové aplikace unit testy. Jsou psané především k testování jednotky kódu. V této práci byly použity několik frameworků, jejichž souhrnný název je xUnit psaný v jazyce C#. Význam testů jednotlivých částí aplikace spočívá v ověření, zda kód funguje správně i po jeho rozšíření a úpravě.

K testování byl použit známý testovací vzor AAA. Každý test je rozdělen do tří částí – Arrange, Act a Assert. V části arrange se připravuje prostředí pro test, zde se vytvářejí ob-

jekty, inicializují se servisy a nastaveny předpoklady. V části act je zavolána testovaná metoda. V poslední části assert se pouze testuje, jestli byly splněny předpoklady.

Napsané jednotkové testy ověřovaly funkčnost metod implementované v repositories a kontrolerech objednávek. V repositories jsou testované metody pro vrácení správných objednávek podle stavu, podle uživatele a metoda pro změnu stavu objednávky. Testy používají in-memory databázi, do které jsem vytvořil dummy objekty. V testu ověřující metodu pro vrácení objednávek podle přihlášeného uživatele v API kontroleru se nekontroluje autentizace, autorizace uživatele, ani volání funkcí ze servisy, ale jestli kontroler vrátí očekávané serializované objekty.

Kapitola 5

Konfigurace k nasazení aplikace

5.1 Možnosti provozu

Aplikace je složená ze tří menších aplikací a jedné databáze – IdentityServer, serverová část, klientská část a MSSQL databáze. Existují dva standardní scénáře při nasazování webových aplikací. První způsob, jak hostovat výslednou aplikaci je mít vše nasazené na jednom serveru. A to buď na vlastním hardwaru nebo na virtuálním serveru. V takovém případě by mohly aplikace být hostovány uvnitř Microsoft IIS a databáze by běžela na MSSQL serveru.

Druhou variantou je pořídit si hosting webových aplikací u poskytovatele. Znamenalo by to, že každé podaplikace by běžely na vlastním web hostingu. Databáze také potřebuje vlastní hosting, nicméně většina poskytovatelů nabízí hosting databáze ve stejném balíčku. Nesmíme opomenout, že podaplikace pro zákazníky může být provozována na CDN nebo jakémkoliv static file hosting poskytovateli, například GitHub Pages nebo Google Cloud Storage, protože se jedná o Blazor WebAssembly aplikaci, tedy se jedná o předem zkompilovaná statické soubory. Pro web hosting podaplikaci serverové části pro správu restaurace je vřele doporučené, aby podporoval komunikaci přes WebSocket, jinak by aplikace používala jiné způsoby komunikace SignalR, které nejsou tak rychlé a stabilní.

5.2 Konfigurace souborů

Pro všechny podaplikace se musí nastavit konfigurační soubor appsettings.json. V něm je důležité nastavit správný connection string pro připojení k databázi a toto se týká podaplikací IdentityServer a Server. Další, co musí být nastavené je URL jednotlivých podaplikací, aby spolu mohly komunikovat, zejména klientská část mohla posílat HTTP požadavky na server API a aby se mohly obě podaplikace autentizovat na IdentityServeru.

Kapitola 6

Závěr

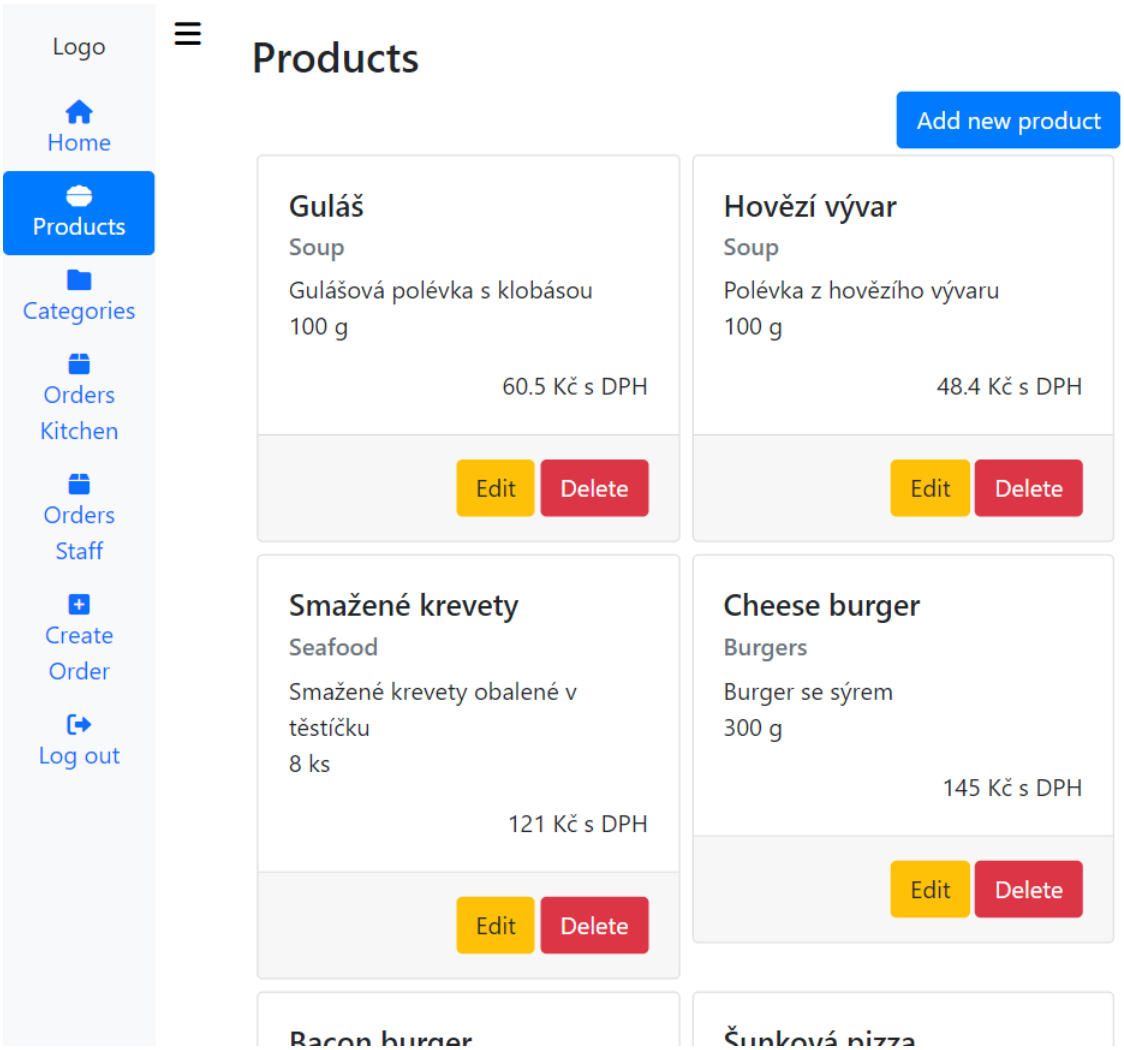
Cílem této práce byl návrh a implementace webového objednávkového systému pro restaurace. Aplikace má pomoci zaměstnancům restaurace se zadáváním a příjímáním objednávek v kuchyni. Po provedení analýzy současných systémů v restauracích a analýzy technologií jsem navrhl vybral vhodné technologie a navrhl funkční požadavky. Posléze jsem naimplementoval prototyp, který byl postupně testován a podle nich provedl korektní úpravy. Zbývalo už jen připravit řešení pro produkční nasazení.

Při analýze systémů v restauracích, jsem si vzpomněl na časy, kdy jsem pracoval v gastronomii jako obsluha. Při přijímání objednávek od hostů a následně předání objednávky kuchyni se člověk dokáže nadřít, a proto mě napadlo, jak tenhle postup zefektivnit a zmodernizovat. Při analýze, návrhu a implementace aplikace jsem se rozhodl pro pokus vytvoření unikátního řešení s hybridním hostitelským modelem, i přes to, že je to nad rámec této práce. To samozřejmě znamenalo víc práce, poněvadž toto řešení je netradiční. Při vývoji jsem narazil na mnoho problémů a bugů, které se mi podařilo vyřešit díky čitelné dokumentaci od Microsoftu. Budoucí práce na projektu zahrnuje přidání a rozšíření funkcionalit prototypu, který následně plánuji otestovat a nasadit u nás v restauraci. Jako student oboru webového inženýrství jsem ve vývoji aplikace využil znalosti, které mi nabyly během studia z předmětů, jako jsou Databázové systémy, Programování v C#, Tvorba webových aplikací.

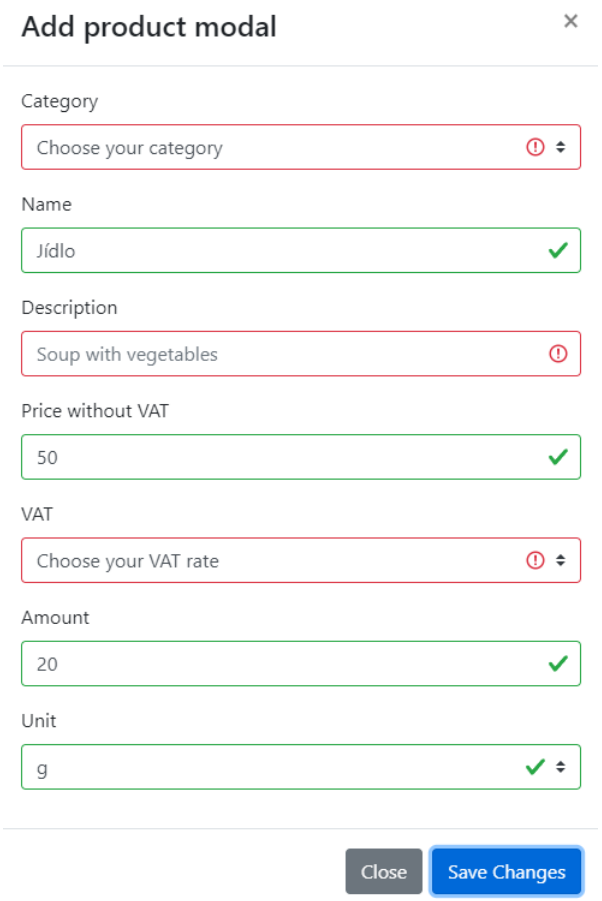
Dodatek A

Ukázky aplikace

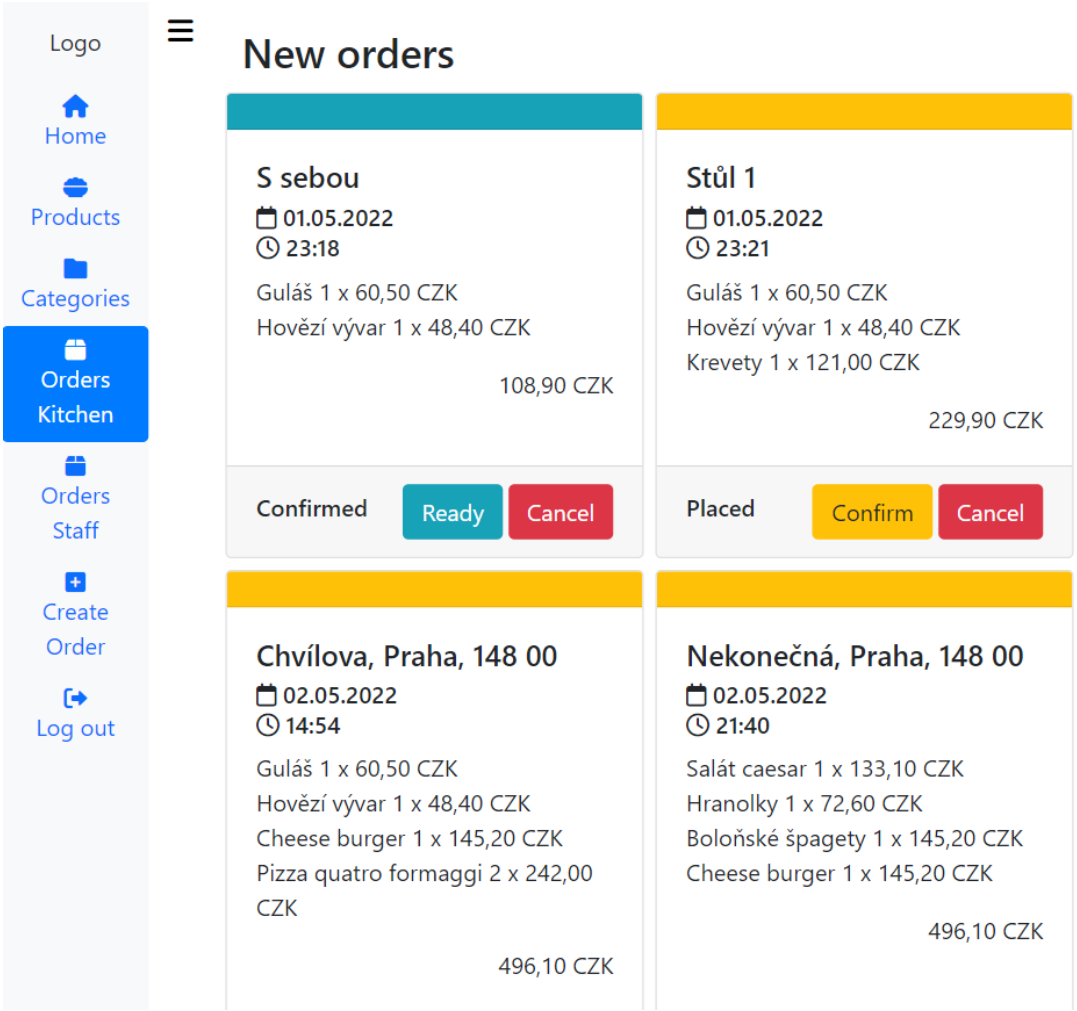
Obrázek A.1 Správa produktů



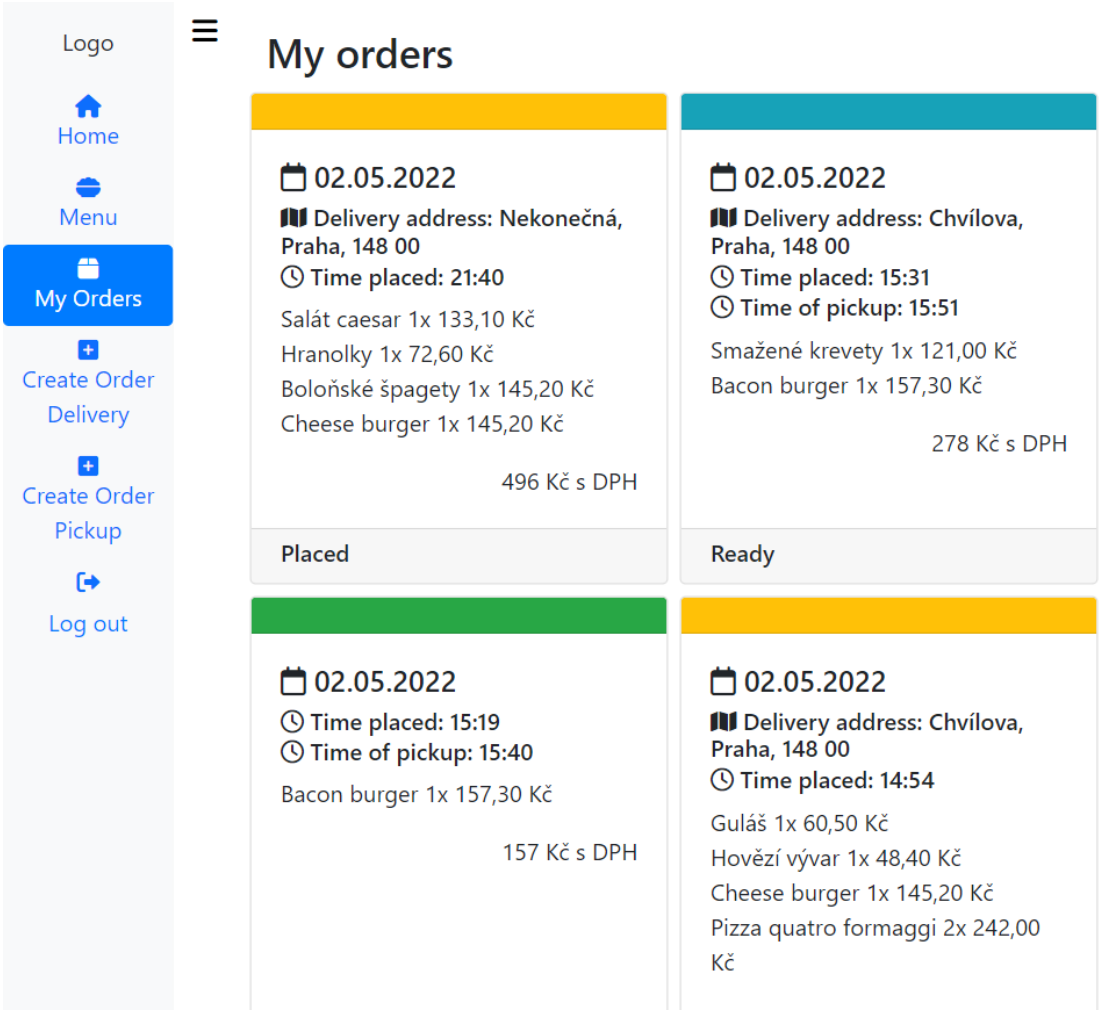
Obrázek A.2 Modal pro tvorbu nového produktu



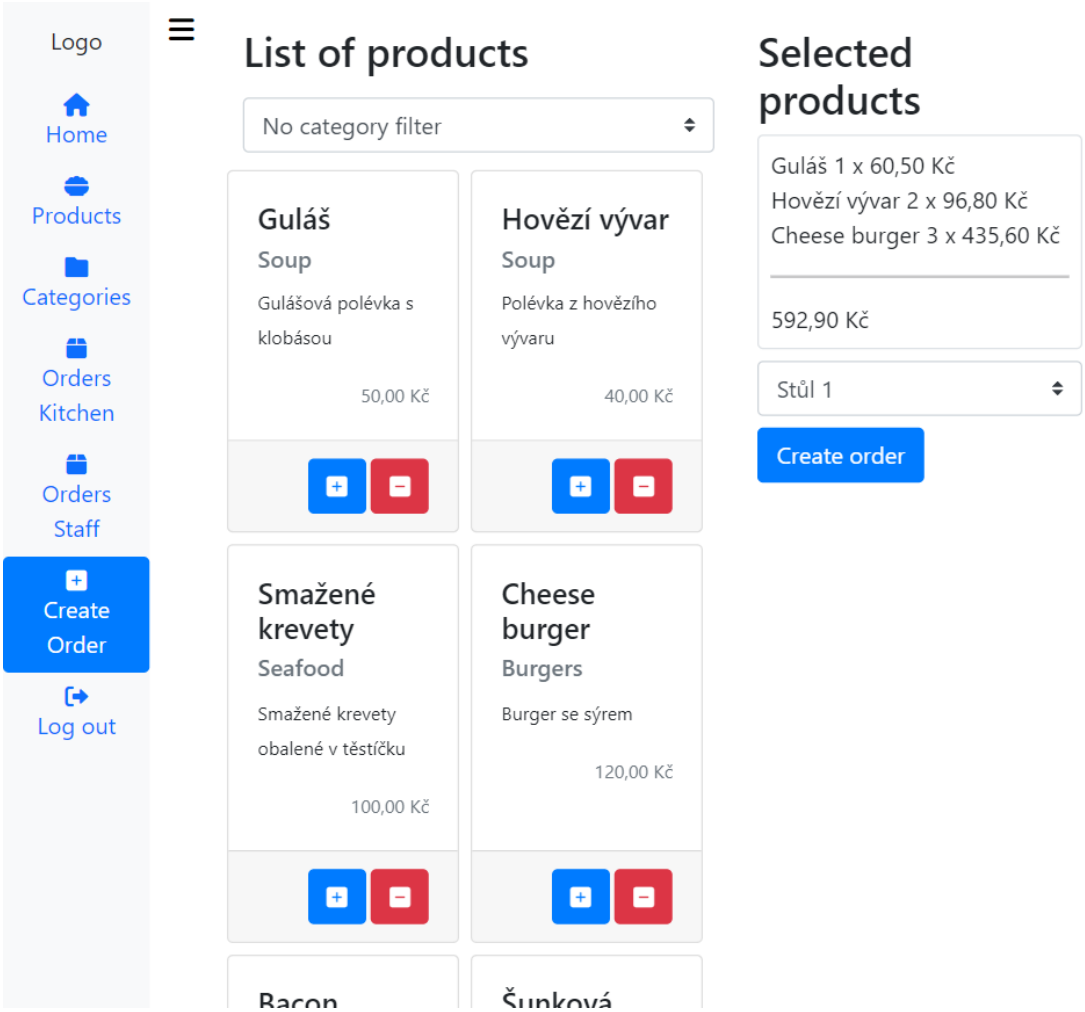
Obrázek A.3 Správa objednávek



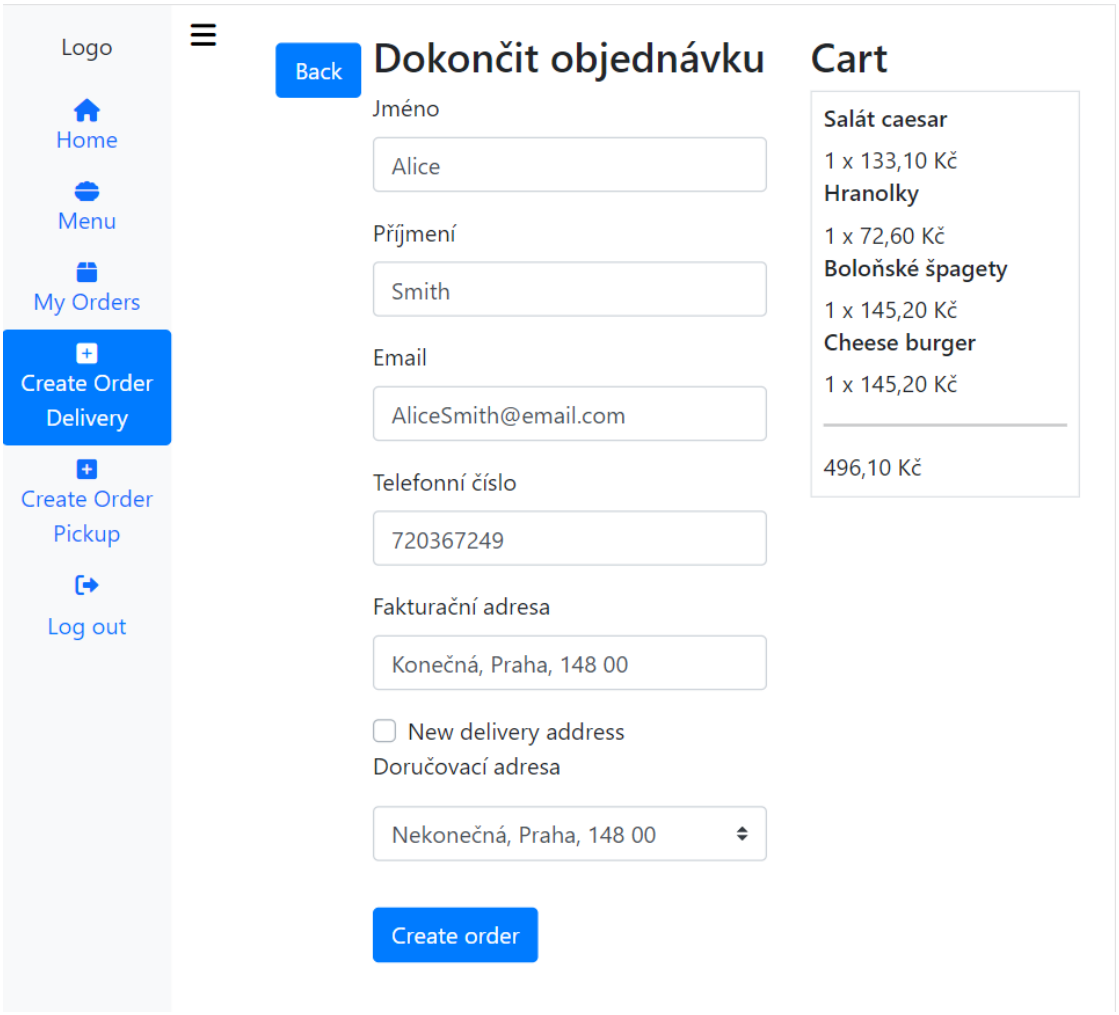
Obrázek A.4 Objednávky uživatele



Obrázek A.5 Tvorba nové objednávky



Obrázek A.6 Dokončení tvorby nové objednávky



Bibliografie

1. BERNHAUER, David. *HTTP požadavek a odpověď* [online]. 2021-12 [cit. 2022-04-03]. Dostupné z: https://courses.fit.cvut.cz/BI-TWA/media/topics/t02-http.pdf.
2. *Kaskádové styly* [online]. Wikimedia Foundation, 2022-03 [cit. 2022-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly.
3. *Dotykačka* [online]. 2022-04 [cit. 2022-04-30]. Dostupné z: https://dotykacka.cz/pokladni-system-pro-gastro.
4. *Adaptee Gastro* [online] [cit. 2022-04-30]. Dostupné z: https://www.adapteegastro.cz/.
5. *Mealgo* [online] [cit. 2022-04-30]. Dostupné z: https://mealgo.cz/.
6. *Webová Stránka* [online]. Wikimedia Foundation, 2020-12 [cit. 2022-04-03]. Dostupné z: https://cs.wikipedia.org/wiki/Webov%C3%A1_str%C3%A1nka.
7. *Klient-Server* [online]. Wikimedia Foundation, 2021-06 [cit. 2022-04-03]. Dostupné z: https://cs.wikipedia.org/wiki/Klient-server.
8. *Hypertext transfer protocol* [online]. Wikimedia Foundation, 2022-02 [cit. 2022-04-03]. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
9. *Hypertext markup language* [online]. Wikimedia Foundation, 2020-11 [cit. 2022-04-03]. Dostupné z: https://cs.wikipedia.org/wiki/Hypertext_Markup_Language.
10. *JavaScript* [online]. Wikimedia Foundation, 2022-03 [cit. 2022-04-05]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript.
11. *Webová aplikace* [online]. Wikimedia Foundation, 2019-06 [cit. 2022-04-05]. Dostupné z: https://cs.wikipedia.org/wiki/Webov%C3%A1_aplikace.
12. *Single-Page Application* [online]. Wikimedia Foundation, 2022-05 [cit. 2022-04-05]. Dostupné z: https://en.wikipedia.org/wiki/Single-page_application.
13. *Symfony* [online]. Wikimedia Foundation, 2022-02 [cit. 2022-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Symfony.
14. D., Prachi. *The good and the bad of Symfony Web Development* [online]. Cisin, 2022-04 [cit. 2022-05-01]. Dostupné z: https://www.developers.dev/tech-talk/technology/the-good-and-the-bad-of-symfony-web-development.html.
15. *Django (web framework)* [online]. Wikimedia Foundation, 2022-04 [cit. 2022-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Django_(web_framework).
16. *Let's Understand The Pros And Cons Of Using Django* [online] [cit. 2022-05-01]. Dostupné z: https://www.benchmarkit.solutions/lets-understand-the-pros-and-cons-of-using-django/.

17. *React (JavaScript library)* [online]. Wikimedia Foundation, 2022-04 [cit. 2022-05-01]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library).
18. BARANOWSKI, Wojciech; WCISŁO, Mateusz; SUSIDKO, Tatiana. *REACT JS: Advantages and disadvantages* [online]. 2022-04 [cit. 2022-05-01]. Dostupné z: https://massivepixel.io/blog/react-advantages-disadvantages/.
19. *Angular (web framework)* [online]. Wikimedia Foundation, 2022-04 [cit. 2022-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Angular_(web_framework).
20. SIDDHARTH. *Angular: Pros and cons* [online]. DEV Community, 2021-08 [cit. 2022-05-01]. Dostupné z: https://dev.to/siddharthshyniben/angular-pros-and-cons-m9l.
21. WARREN, Genevieve. *.Net (A .NET core) – úvod a přehled* [online]. 2022-04 [cit. 2022-04-03]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/introduction.
22. ANDERSON, Rick; ROTH, Daniel; LUTTIN, Shaun. *Overview of ASP.NET core* [online]. 2022-04 [cit. 2022-04-03]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0.
23. *Microsoft Visual studio* [online]. Wikimedia Foundation, 2022-04 [cit. 2022-04-15]. Dostupné z: https://cs.wikipedia.org/wiki/Microsoft_Visual_Studio.
24. DOUGLAS, Jon. *What is Nuget and what does it do?* [Online]. Microsoft, 2022-02 [cit. 2022-04-16]. Dostupné z: https://docs.microsoft.com/en-us/nuget/what-is-nuget.
25. VICKERS, Arthur. *Přehled entity Framework Core – EF Core* [online]. Microsoft, 2022-04 [cit. 2022-04-16]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/core/.
26. LAMBSON, Brice. *Přehled migrace – EF core* [online]. Microsoft, 2022-04 [cit. 2022-04-17]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli.
27. LATHAM, Luke. *ASP.NET Core Blazor* [online]. Microsoft, 2022-04 [cit. 2022-04-22]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/blazor/?view=aspnetcore-6.0.
28. LATHAM, Luke. *Blazor modely hostování ASP.NET Core* [online]. Microsoft, 2022-04 [cit. 2022-04-22]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/blazor/hosting-models?view=aspnetcore-6.0.
29. LATHAM, Luke. *ASP.NET Core Razor Komponenty* [online]. ASP.NET Core Razor komponenty, 2022-04 [cit. 2022-04-22]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/blazor/components/?view=aspnetcore-6.0.
30. *Modal component* [online]. Blazorise [cit. 2022-04-23]. Dostupné z: https://blazorise.com/docs/components/modal.
31. LATHAM, Luke. *ASP.NET Core Blazor project structure* [online]. Microsoft, 2022-04 [cit. 2022-04-23]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/project-structure?view=aspnetcore-6.0.
32. ANIL, Nish. *IdentityServer for cloud native apps* [online]. Microsoft, 2022-04 [cit. 2022-04-26]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/identity-server.
33. *Key Management* [online]. Duende [cit. 2022-04-28]. Dostupné z: https://docs.duendesoftware.com/identityserver/v6/fundamentals/keys/.
34. *Duende Software Documentation* [online]. Duende [cit. 2022-04-28]. Dostupné z: https://docs.duendesoftware.com/identityserver/v6/quickstarts/5_aspnetid/.
35. VICKERS, Arthur. *Code first conventions - EF6* [online]. Microsoft, 2020-10 [cit. 2022-04-24]. Dostupné z: https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/conventions/built-in.

Obsah přiloženého média

- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF