Ukázková aplikace ASP.NET Blazor a relační databáze

Jakub Plesník

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub Plesník**
Osobní číslo:	**A19090**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Ukázková aplikace ASP.NET Blazor a relační databáze**
Téma práce anglicky:	**ASP.NET Blazor Sample Application and Relational Database**

Zásady pro vypracování

1. Vypracujte stručný rozbor technologií, které budou použity k návrhu.
2. Proveďte rozbor a analýzu požadavků na zvolené řešení.
3. Vypracujte návrh databáze pro toto řešení.
4. Zpracujte Blazor aplikaci na základě výsledků analýzy.
5. Věnujte pozornost zabezpečení aplikace.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SAINTY, Chris. Blazor in Action. Manning Publications, 2022. ISBN 9781617298646.
2. JOHNSON, Glenn. Programming in HTML5 with JavaScript and CSS3: training guide. Redmond, Wash.: Microsoft, 2013. ISBN 978-0735674387.
3. UNHELKAR, Bhuvan. Software engineering with uml. Auerbach Publications, 2017.
4. LETT, Jacob. Bootstrap 4 Quick Start: A Beginners Guide to Building Responsive Layouts with Bootstrap 4. Bootstrap Creative, 2018.
5. JAKOBUS, Benjamin. Mastering Bootstrap 4: Master the latest version of Bootstrap 4 to build highly customized responsive web apps. Packt Publishing Ltd, 2018.
6. BEN-GAN, Itzik; DAVIDSON, Louis; VARGA, Stacia. MCSA SQL Server 2016 Database Development Exam Ref 2-pack: Exam Refs 70-761 and 70-762. Microsoft Press, 2017.

Vedoucí bakalářské práce: **doc. Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 5. 5. 2023 Jakub Plesník, v.r.

ABSTRAKT

Bakalářská práce se zabývá návrhem a vývojem ukázkové aplikace pomocí webového frameworku Blazor a vybraném systému relační databáze. Cílem práce je analýza požadavků, návrh aplikace a její následná implementace. Teoretická část se zabývá rozborem technologií použitých k návrhu a vývoji aplikace. V praktické části je představena analýza požadavků, návrh databáze a jsou zde popsány hlavní prvky ukázkové aplikace včetně její demonstrace ze strany uživatele. Aplikace byla vyvinuta s modelem hostování Blazor WebAssembly a systémem relační databáze SQL Server.

Klíčová slova: Blazor, .NET, relační databáze, webová aplikace

ABSTRACT

The bachelor's thesis deals with designing and developing a sample application using the Blazor web framework and a selected relational database system. The thesis' aims are to analyze requirements and design and create a sample application. The theoretical part analyses the technologies used to design and develop the application. In the practical part, the analysis of requirements and database design are presented, and the main elements of the sample application, including its demonstration by the user, are described. The application was developed using the Blazor WebAssembly hosting model and the SQL Server relational database system.

Keywords: Blazor, .NET, relational database, web application

Tímto bych rád poděkoval vedoucímu práce doc. Ing. Petrovi Šilhavému, Ph.D. za jeho odborné vedení, ochotu a trpělivost při řešení problémů a směřování mé bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Platforma .NET již spoustu let hraje centrální roli ve vývoji software a pohání nespočet webových, desktopových a mobilních aplikací. Toto tvrzení je možné potvrdit počtem open-source projektů, řešených problémů na komunitních fórech, či postavením programovacího jazyka C# v roce 2023 mezi pěti nejpopulárnějšími v rámci TIOBE indexu [1].

Platforma .NET je multiplatformní a umožňuje vývoj robustních webových aplikací pomocí frameworku ASP.NET Core. Nový webový framework Blazor nyní však umožňuje vývoj klientských aplikací čistě pomocí jazyka C# bez použití jazyka JavaScript pro komunikaci s aplikační logikou. To představuje určité výhody pro firmy s vývojáři se zkušenostmi primárně v jazyce C# a prostředí .NET.

Tato práce si dává za cíl představit vývoj webové aplikace pomocí frameworku Blazor a zaměřuje se na představení možností vývoje, jednotlivých technologií a samotný návrh a demonstraci ukázkové aplikace.

Teoretická část se zabývá popisem platformy .NET včetně jejich verzí a frameworků, aktuálních možností vývoje webových aplikací pomocí ASP.NET Core a představuje framework Blazor. Dále jsou představeny technologie pro zabezpečení a komunikaci s databází, relační databáze, jazyk SQL a framework pro vytváření responzivního uživatelského rozhraní Bootstrap.

Praktická část se zaměřuje na návrh a implementaci klíčových částí vyvíjené aplikace. Je provedena analýza požadavků, představeny případy užití včetně jejich scénářů a návrh databáze pomocí datových modelů. Následně je popsáno rozvržení projektu a klíčové části hotové aplikace dle výsledné analýzy a návrhu. Nakonec je demonstrováno rozhraní a funkcionalita aplikace ze strany uživatele.

I. TEORETICKÁ ČÁST

1 .NET

.NET je bezplatný softwarový framework vyvíjený společností Microsoft původně pro operační systém Microsoft Windows, který byl následně přepsán a změněn na open-source s multiplatformní podporou pod názvem .NET Core. Podporuje všechny hlavní platformy včetně desktopových a mobilních operačních systémů. Platforma podporuje jazyky C#, F# a Visual Basic [2].

.NET Core je nástupcem původního .NET Frameworku a lze jej použít k implementaci různých druhů desktopových, serverových, mobilních, IoT či cloudových aplikací [3].

Hlavní cíl a důvod vývoje nové codebase pro .NET Core byla multiplatformnost a otevřenost kódu (open-source) jenž byl rychlý, volně propojený a často aktualizovaný – na rozdíl od archaického .NET Frameworku, který byl limitován nutností podporovat starší verze [4]. Core navíc umožnil nativní vývoj pro zařízení se systémem založeným na UNIX a využívá méně prostředků než jeho předchůdce [5].

Následující verze postupně implementovaly více funkcí z rozhraní .NET API a v roce 2020 byla vydána verze .NET 5.0, jež odstranila název „Core“ a stala se hlavní implementací .NET platformy. Nejaktuálnější stabilní verze platformy .NET k datu vystavení této bakalářské práci je .NET 7.0.

1.1 .NET Standard

Pro účely podpory různých prostředí a vývojových sad byla definována sada standardů s názvem .NET Standard. Jedná se o specifikaci různých API, které platforma .NET poskytuje.

První verze .NET Standard byla představena společně s vydáním .NET Core a konečnou se stala verze .NET Standard 2.1 při vydání .NET Core 3.0 [6].

S vydáním .NET 5.0 nakonec zmizela potřeba pro další vývoj .NET Standard, jelikož .NET 5.0 vše spojil v jednu codebase, která podporuje všechny hlavní platformy [7].

1.2 ASP.NET Core

ASP.NET Core je bezplatný open source framework pro vývoj webových aplikací. Jedná se o nástupce a kompletní přepsání původního frameworku s názvem ASP.NET, který měl jako cílovou platformu pouze operační systém Microsoft Windows. ASP.NET Core podporuje

operační systémy Windows, Linux a macOS a má prvotřídní podporu ve Visual Studio IDE [8].

Pro vývoj webových aplikací ASP.NET Core podporuje a nabízí následující technologie:

- *ASP.NET Core Razor Pages* pro psaní server-side kódu do HTML stránek; součástí jsou taktéž tzv. *Razor Class Libraries* (RCLs), jež umožňují zabalit a distribuovat komponenty uživatelského rozhraní.
- *ASP.NET Core MVC* pro vývoj komplexních webových aplikací pomocí návrhového vzoru Model-View-Controller (MVC).
- *Blazor*, pro vytváření interaktivního webového uživatelského rozhraní pomocí jazyka C# a platformy .NET narozdíl od tradičního JavaScriptu. Blazor je schopný klientský kód provádět na serveru protokolem WebSockets nebo přímo v prohlížeči pomocí WebAssembly [8].

Vývoj klientské části webové aplikace je možný namísto Razor Pages či Blazoru i pomocí JavaScript frameworků pro vývoj jednostránkových aplikací (SPA) jako jsou např. Angular či React. Pro vývoj pomocí těchto frameworků existují šablony, které mohou být použity pro rychlé sestavení základu aplikace [9].

ASP.NET Core má také řadu knihoven pro základní účely webového vývoje, mezi ně patří např. inicializace a použití HTTP klientů, volání webových služeb pomocí SOAP, validace a serializace dat, zabezpečení pomocí ASP.NET Core Identity a další.

1.2.1 Razor

Razor je syntaxe umožňující vyvíjet dynamické webové stránky použitím jazyka C# uvnitř HTML. Tato syntaxe je psána do šablon, jež jsou odlišovány od klasických HTML souborů souborovou příponou *.cshtml* [10]. Typická šablona s Razor syntaxí obsahuje dva typy obsahu:

- Klientský obsah – HTML značky, CSS stylování, případně JavaScript pro práci s klientem a čistým textem;
- Server-side kód – C# kód označen pomocí symbolu @, případně definován blokem označeným závorkami @{ ... }

V případě, že šablona obsahuje kód ze strany serveru, server jej nejprve spustí před samotným odesláním požadavku k zobrazení dané stránky. Server-side kód je tímto způsobem schopen provádět složitější operace na straně serveru přímo v klientském obsahu [11].

Hlavním účelem Razoru je přidání schopnosti server-side kódu vytvářet dynamický webový obsah za běhu při samotném renderování webové stránky v prohlížeči.

V rámci Blazor frameworku je Razor syntaxe používána v Razor komponentách, které mají souborovou příponu *.razor*.

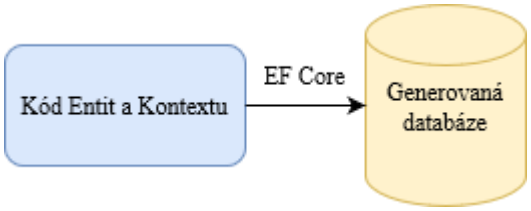
1.2.2 Entity Framework Core

Entity Framework Core (EF Core) je knihovna, která umožňuje .NET aplikacím komunikovat s relační databází, vytvářet instance databází pomocí migrací či mapovat doménový model na databázové objekty (tabulky) [12]. Jedná se o moderní verzi a nástupce původního Entity Frameworku, který měl podporu pouze pro platformu .NET Framework a operační systém Windows.

EF Core je založen na tzv. objektově-relačním mapování (ORM), jež umožňuje pracovat s datovou vrstvou v rámci objektového programování. .NET objekty, které jsou definovány v programovacím jazyce aplikace jako entity, jsou pomocí EF Core namapovány na data v relační databázi, a umožňují tak jednoduchý přístup ke správě dat v relační databázi pomocí objektového programování [12]. EF Core nabízí dva přístupy vývoje – code-first a database-first.

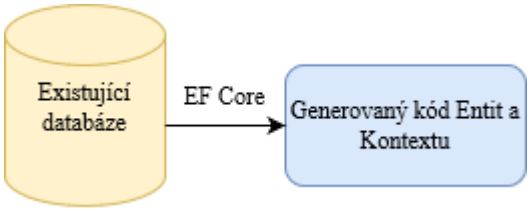
U přístupu „code-first“ EF Core vytváří databázi dle implementace a konfigurace doménových tříd (entit) a databázového kontextu v kódu aplikace [13]. Konfigurace primárních a

cizích klíčů, indexace, datových typů apod. je možná v databázovém kontextu pomocí Fluent API [16] nebo pomocí C# atributů přímo v souborech jednotlivých entit. Poté je možné generovat migrační soubor, pomocí kterého po provedení příkazu k aktualizaci databáze se vytvoří či upraví instance databáze. Tento přístup je EF Core více podporován a je vhodnější při vývoji aplikace, která není závislá na již existující databázi [14].



Obrázek 1. Code-first přístup

Přístup „database-first“ se využívá primárně při vývoji nové aplikace pro již existující vytvořenou databázi. Veškeré změny v této databázi se poté musí odrážet v kódu jednotlivých entit a business logice. EF Core umožňuje vytvářet a upravovat C# kód databázových entit a databázového kontextu dle návrhu existující databáze, do vygenerovaného kódu poté již však není vhodné moc zasahovat [14]. Microsoft tento postup označuje za reverzní inženýrství a je možné jej provést příkazem *Scaffold-DbContext* s uvedeným řetězcem pro navázání spojení s databází [15]. Výsledkem je vygenerovaný kód tříd entit a databázového kontextu.



Obrázek 2. Database-first přístup

1.3 Zabezpečení

Zabezpečení je kritickou částí vývoje webových aplikací a mělo by být středem pozornosti již od prvních fází vývojového procesu. Zabezpečení je možné pochopit jako ochranu dat aplikace před neoprávněnými a zlomyslnými přístupy. Těmi je možné se vyvarovat autentizací uživatele, omezením přístupových práv k jednotlivým zdrojům a ochranou dat uložených na serveru a přenášených po síti [17].

ASP.NET Core disponuje širokou škálou vestavěných nástrojů a externích knihoven pro správné zabezpečení aplikace a zamezením různých druhů napadení.

1.3.1 Cross-Site Scripting

Cross-Site Scripting (XSS) je způsob útoku na webovou aplikaci vložením škodlivého skriptu do pole vstupů/formuláře webové stránky, případně do hlavičky nebo URL řetězce HTTP požadavku s úmyslem ukrást důvěrné informace, jako jsou přihlašovací údaje či jiné (např. cookies či data ve webovém úložišti local storage nebo session storage) [40].

Pro ochranu aplikace před XSS útoky je vhodné použít regulární výrazy k validaci vstupu jak na straně severu, tak na straně klienta. V databázi by poté měly být uloženy pouze validovaná data. Technologie Razor automaticky kóduje veškeré vstupy, část s útočným skriptem tedy nikdy není provedena. Kódování pro danou část kódu je možné vypnout použitím Razor syntaxe @Html.Raw() [17].

1.3.2 Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) je útok, který vynutí koncového uživatele na aktuálně autentizované webové aplikaci provést nežádoucí operace. Tomuto typu útoku je většinou dosaženo sociálním inženýrstvím. Typickým příkladem je poslání škodlivého, na venek nevinného, URL odkazu prostřednictvím e-mailu či chatu danému uživateli. Tento odkaz poté přiměje uživatele provést útočníkovi zvolenou akci [41].

Pokud je obětí běžný uživatel, úspěšný CSRF útok může uživatele donutit provést požadavky pro převod finančních prostředků, změnu e-mailové adresy apod. V případě, že se obětí stane administrativní účet, CSRF může ohrozit celou webovou aplikaci [41]

K ochraně uživatele a aplikace před CSRF útoky je možné v rámci HTTP požadavků vždy odeslat požadavek s tzv. anti-forgery tokenem (ověřovací token požadavku). Anti-forgery tokeny je možné v ASP.NET Core aplikaci používat přidáním atributu

ValidateAntiForgeryToken k akci daného controlleru. Server uživateli odešle token a poté, co uživatel zadá požadavek, token odešle zpět na server k ověření. Tokeny je možné ukládat v hlavičce požadavku nebo v cookies [40].

1.3.3 SQL Injection

SQL Injection je nebezpečný způsob útoku, při kterém neoprávněný uživatel vloží škodlivé SQL dotazy, které jsou poté spuštěny nad databází aplikace a umožní útočníkovi získat přístup k důvěrným informacím v databázi [40].

Tento útok je možné zamezit dodržováním zásad a technologií, které se nespoléhají na přímé dotazování databáze pomocí SQL dotazů. Příkladem je použití frameworku Entity Framework Core nebo jiného ORM, použití parametrizovaných dotazů, ověřování vstupů na straně serveru a klienta nebo použití procedur uložených v databázovém systému [17]. Citlivá data jako hesla by nicméně nikdy neměla být v databázi uložena jako prostý text [40].

1.3.4 JSON Web Token

JSON Web Token (JWT) je navržený webový standard (RFC 7519), který umožňuje bezpečný, kompaktní a samostatný způsob přenosu informací mezi odesílatelem a příjemcem prostřednictvím URL adresy, parametru nebo HTTP hlavičky [18].

Informace mezi odesílatelem a příjemcem jsou v JSON formátu, který je digitálně podepsán šifrovacím algoritmem pro ověření pravosti. JWT se obvykle používá k implementaci autentizace a autorizace webových aplikací [18].

Obrázek 3. Logo JWT

JWT je strukturován do tří částí zakódovaných base64url algoritmem, který narozdíl od standardního base64 šifrování nepoužívá znaky se speciální funkcí v URL adresách. Tyto části jsou definovány jako Header (hlavička), Payload (přenášené informace) a Signature (podpis).

Header sekce obsahuje informace o typu tokenu a algoritmu použitého k zašifrování. Typicky je používán šifrovací algoritmus HS256 (HMAC s SHA-256), který používá hashovací funkci s jednotným klíčem pro obě strany (symetrické šifrování), nebo RS256 (RSA s SHA-256), který k podpisu využívá kombinaci veřejného a soukromého klíče (asymetrické šifrování) [20].

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Obrázek 4. Hlavičková část (JSON) se specifikovaným HS256 algoritmem

Payload reprezentuje přenášená data, které systém vyžaduje k autentizaci a správě přihlášení uživatele. Typicky obsahuje nějaký subjekt tokenu, identifikátor uživatele (jméno), deklarace identity (claims), povolená oprávnění apod.

```
{
  "sub": "1234567890",
  "name": "Jakub Plesník",
  "admin": true
}
```

Obrázek 5. Přenášená data (JSON)

Do sekce s podpisem (Signature) se nakonec zašifrují informace z hlavičky, přenášených dat a klíčem (secret). Tato sekce slouží k potvrzení, že během přenosu informací nebyly data nijak zfalšovány. V případě, že JWT byl zašifrován soukromým klíčem umožní také jednoznačnou identifikaci odesílatele požadavku [19].

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  symetricky-klic
)
```

Obrázek 6. Podpis (pseudokód)

Výsledným tokenem je textový řetězec, který má jednotlivé části oddělené tečkou (‘.’) a je jednoduché jej posílat v HTTP požadavku v hlavičce nebo přímo v URL.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpha
3ViIFBsZXNuw61rIiwiYWRtaW4iOnRydWV9.xCy
3dL54nP6E4dnEArBFE3Z3-
wKdrPapvk_pNaWqjUk
```

Obrázek 7.Výsledný JWT token

ASP.NET Core má pro autentizaci pomocí JWT prvotřídní podporu díky balíčku *Microsoft.AspNetCore.Authentication.JwtBearer*. Autentizaci je jednoduché implementovat a nastavit v souboru Program.cs ASP.NET Core webové aplikace.

1.3.5 ASP.NET Core Identity

ASP.NET Core nabízí také velmi výkonnou knihovnu ASP.NET Core Identity k autentizaci a správě uživatelů. Tato knihovna umožňuje pomocí Entity Framework Core a Identity modelu vygenerovat vhodné tabulky uživatelů, rolí, deklarací identit a dalších pomocných tabulek. Předem definovaný Identity model je možné upravit a po aplikaci vygenerované migrace použitím code-first přístupu EF Core vytvořit patřičné tabulky [21].

2 BLAZOR

Blazor je open-source webový framework pro vývoj moderních jednostránkových klientských aplikací (Single Page Applications), který je založený na platformě .NET a programovacím jazyce C#. Název Blazor vznikl mutací termínů Browser a Razor (viz kapitola 1.2.1). Jak již z této kombinace termínů vyplývá, Blazor umožňuje prezentovat Razor stránky přímo v prohlížeči na straně klienta [22].

Obrázek 8.Blazor logo

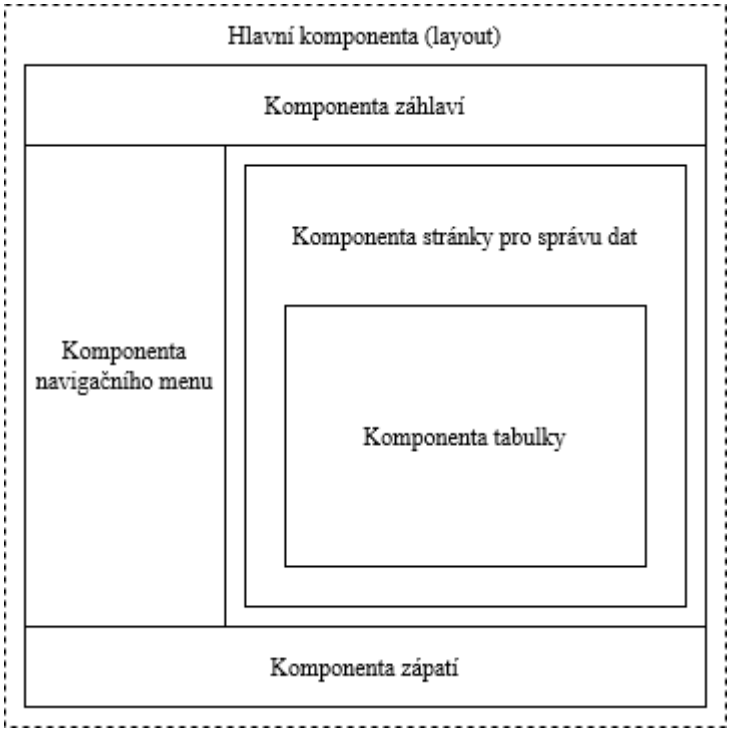
Použití platformy .NET a jazyka C# v klientské části aplikace přináší určité výhody. Jedna z nejatraktivnějších výhod je psaní klientského kódu v jazyce C# namísto jazyka JavaScript. Jelikož C# je hluboce zasazen do ekosystému platformy .NET, Blazor umožňuje využít její výkonnost, možnosti zabezpečení a širokou škálu již existujících knihoven. Mezi další výhodu patří sdílení serverového a klientského kódu v aplikační logice [24].

Blazor je možné považovat za alternativu k JavaScript frameworkům jako jsou např. Angular, React či Vue.js a přebírá některé z jejich konceptů. Mezi hlavní patří rozdělení uživatelského rozhraní do komponent [23].

2.1 Komponenty

Komponenty jsou základní stavební kameny Blazor aplikace a jsou velmi užitečné k rozdělení uživatelského rozhraní do logických částí [23]. Mezi hlavní výhody uživatelského rozhraní rozděleného do komponent je znuvupoužitelnost kódu a jeho udržitelnost v rámci celé aplikace – při změně kódu komponenty se změny projeví ve všech instancích dané komponenty [23]. Jednotlivé komponenty je také možné vnořovat do dalších komponent [25].

Komponenta může také definovat vlastní API a přijímat či vracet data a eventy (akce) [23]. Příkladem může být např. komponenta, která zobrazuje data ve formátu tabulky. Tento typ komponenty by měl mít jednotné stylování a funkcionalitu v rámci celé aplikace, pouze přijímat jiný typ dat.



Obrázek 9. Příklad rozdělení uživatelského rozhraní do komponent

Komponenty v Blazor aplikacích se vytvářejí pomocí Razor komponent, jež využívají Razor syntaxi (viz kapitola 1.2.1). Razor komponenty jsou implementovány v souborech s příponou *.razor* [25].

2.2 Modely hostování

Uživatelské rozhraní Blazoru lze hostovat různými způsoby. Komponenty je možné spustit na straně klienta v prohlížeči pomocí WebAssembly (model Blazor WebAssembly) nebo na straně ASP.NET Core serveru (model Blazor Server). Blazor Hybrid také nově umožňuje hostovat Razor komponenty v nativní desktopové či mobilní aplikaci vykreslením webového obsahu do WebView objektu. Razor komponenty lze použít v libovolném hostovacím modelu bez vyžadování dalších úprav [26].

V rámci této práce bude primárně popsán hostovací model Blazor WebAssembly, který byl použit k vývoji ukázkové aplikace. Nastíněny budou ovšem i základní koncepty modelů Blazor Server a Blazor Hybrid.

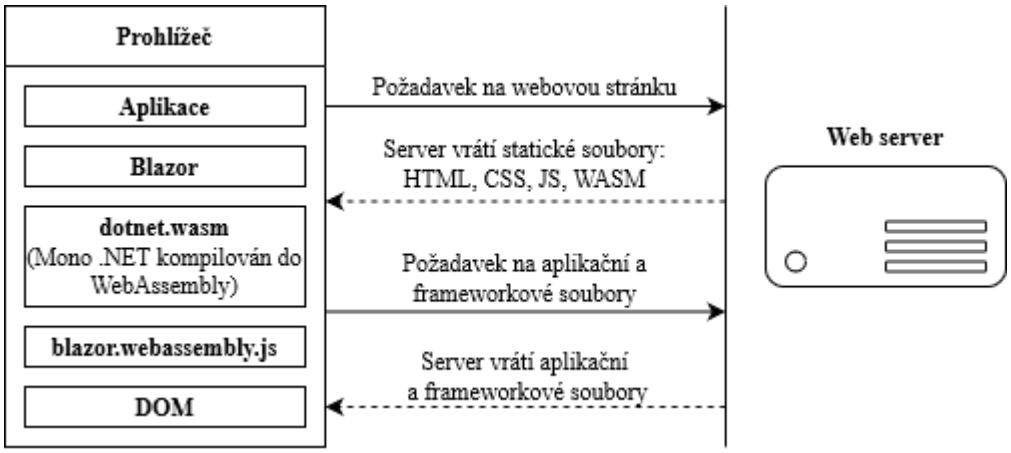
2.2.1 Blazor WebAssembly

Blazor WebAssembly aplikace běží na straně klienta v prohlížeči pomocí WebAssembly a slouží jako alternativa k JavaScript SPA frameworkům [23].

WebAssembly (WASM) je nízkoúrovňový programovací jazyk podobný Assembly, který lze spustit v moderních webových prohlížečích s téměř nativním výkonem po kompilaci z vyššího programovacího jazyka (např. jazyk C#). WASM je navržen tak, aby mohl být použit v jedné aplikaci společně s jazykem JavaScript [27].

Blazor WebAssembly aplikace může být nasazena bez back-endové ASP.NET Core aplikace (standalone) nebo na back-endovém serveru (hosted). Standalone nasazení umožňuje vývoj progresivních webových aplikací (PWA), které mohou fungovat částečně offline např. v aplikaci mimo prohlížeč [26].

K pochopení struktury a inicializace Blazor WebAssembly aplikace hostované na serveru s ASP.NET Core slouží následující obrázek.



Obrázek 10. Inicializace Blazor WebAssembly aplikace, interakce mezi klientem a serverem [23]

Aplikace Blazor, její dependence a modul .NET runtime jsou staženy do prohlížeče a je spuštěna přímo ve vlákně uživatelského rozhraní prohlížeče. Aktualizace uživatelského rozhraní a zpracování událostí poté probíhají v rámci stejného procesu. Prostředky aplikace jsou nasazeny ve formátu statických souborů na webový server [26].

Proces inicializace začíná požadavkem ze strany prohlížeče a jeho odesláním na webový server. Server poté vrátí sadu souborů vyžadovaných k načtení aplikace. Do této sady souborů patří hlavní stránka aplikace (obvykle nazvaná *index.html*), statické soubory jako

obrázky, CSS a JavaScript, včetně speciálního JavaScript souboru *blazor.webassembly.js* [23]. Tento soubor zajišťuje pro aplikaci následující:

- Načte a inicializuje Blazor aplikaci v prohlížeči;
- Poskytuje schopnost manipulovat s Document Object Modelem (DOM), umožňující Blazoru aktualizovat uživatelské rozhraní;
- Poskytuje API pro JavaScript interoperabilitu (vyvolávání JS funkcí z .NET metod a vyvolávání .NET metod z JS funkcí) [28]

Jakmile prohlížeč získá tyto soubory ze serveru, může zkonstruovat DOM webové stránky. Následně je spuštěn soubor *blazor.webassembly.js*, který vyvolá požadavek ke stažení souboru s názvem *blazor.boot.json*, dle kterého se poté stáhnou frameworkové a aplikační soubory vyžadované ke spuštění aplikace. Jeden ze souborů který je stažen, se nazývá *dotnet.wasm*. Ten obsahuje kompletní .NET runtime kompilovaný do WebAssembly [23]. Ve verzi .NET 6 byla přidána ahead-of-time (AOT) kompilace, která umožňuje kompilovat celou aplikaci do WebAssembly, což má za následek zlepšení výkonu běhu aplikace na úkor větší velikosti statických souborů [26].

Po stažení prostředků je aplikace spuštěna pomocí .NET runtime ve WebAssembly, kdy je nejprve načten Blazor framework a následně samotná aplikace [23].

Mezi hlavní výhody Blazor WebAssembly aplikace je oproštění prostředků ze strany serveru (i když na úkor zvyšení požadavků na straně klienta) a statická funkcionalita, např. ve formě progresivní webové aplikace, která je schopná fungovat i bez serveru. Nevýhody jsou ovšem pomalejší start aplikace z důvodu velikosti inicializačních souborů a neumožnění indexace obsahu webovými vyhledávači [22].

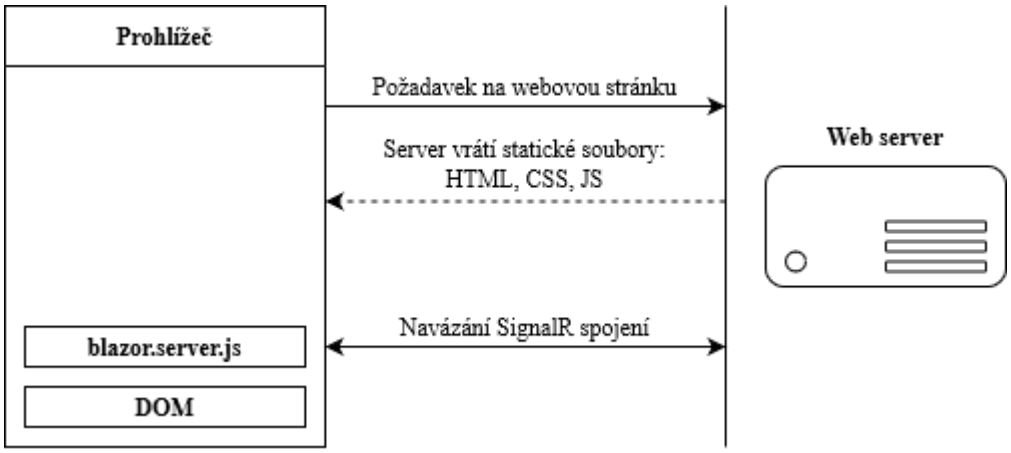
2.2.2 Blazor Server

S modelem Blazor Server se aplikace spouští na straně serveru v rámci ASP.NET Core aplikace. Aktualizace uživatelského rozhraní a veškeré interakce jsou zpracovány skrz protokol WebSockets pomocí frameworku SignalR [26].

Každé připojení klienta je označováno jako okruh (circuit). Tyto okruhy nejsou vázané na konkrétní síťové připojení a při dočasném výpadku sítě mohou tolerovat opětovné pokusy připojení k serveru [26].

Klientská část aplikace má za úkol primárně uchovávat a obnovovat stav aplikace dle potřeby. Samotné SignalR připojení na server je navázáno pomocí skriptu *blazor.server.js* [26].

Blazor Server má oproti Blazor WebAssembly výhodu v tom, že veškerý HTML obsah aplikace je vykreslen ještě předtím než je poslán do prohlížeče klienta. To umožňuje kratší inicializaci aplikace a indexaci webovými vyhledávači [22]. Naopak jelikož každá instance aplikace v prohlížečích klientů je brána jako samostatný okruh (i např. nové záložky s aplikací ve stejném prohlížeči), samotná komunikace a připojení může mít vyšší odezvu než předchozí řešení a serverová část aplikace má vyšší využití paměti a procesorové zatížení [26].



Obrázek 11. Inicializace Blazor Server aplikace, interakce mezi klientem a serverem [23]

2.2.3 Blazor Hybrid

Blazor Hybrid je nejnovější model hostování frameworku Blazor. Blazor Hybrid aplikace jsou nativní, ale k zobrazování obsahu využívají webové technologie. Razor komponenty jsou spuštěny přímo v aplikaci bez použití WebAssembly. Vykreslení HTML a CSS ve WebView je umožněno pomocí lokálního interop kanálu [26].

Pro vývoj Blazor Hybrid aplikací je možné použít framework .NET MAUI, WPF nebo WinForms. Ovládací prvky BlazorWebView umožňují přidávat Razor komponenty do aplikací založených na těchto frameworcích. Vývoj v .NET MAUI umožňuje pohodlný vývoj multiplatformních aplikací pro mobilní a desktopová zařízení, zatímco integrace s WPF a Windows Forms nabízí možnost modernizovat již existující aplikace založených na těchto frameworcích [26].

Razor komponenty vyvinuté pro Blazor WebAssembly nebo Blazor Server aplikaci je možné použít také v rámci Blazor Hybrid aplikace. Díky tomu, že Blazor Hybrid aplikace jsou

nativní, umožňují navíc používat funkce .NET API které nejsou dostupné pro čistě webové aplikace.

3 RELAČNÍ DATABÁZE

Relační databáze je typ databáze, která ukládá a organizuje data do tabulek a soustředí se na vztah (relaci) mezi uloženými datovými prvky. Tabulky obsahují data uspořádaná do definovaných sloupců (atributy entit) a vytvořených řádků (entity). Proces strukturování dat tímto způsobem se označuje jako normalizace databáze [29]. Vytvoření vztahů mezi tabulkami je možné pomocí primárních a cizích klíčů. Vazby pomocí těchto identifikátorů se při návrhu nejčastěji ilustrují pomocí různých typů datových modelů.

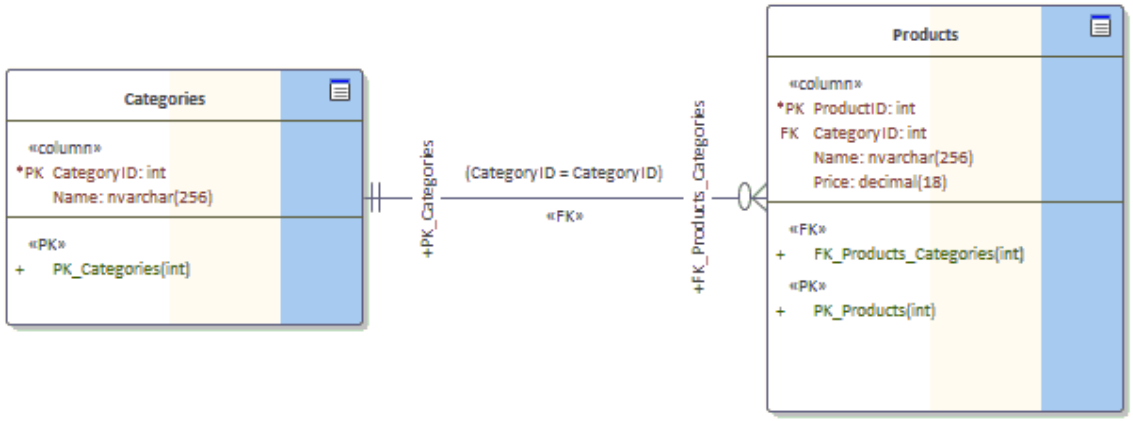
Ke koncepci datového modelu relační databáze je možné použít různé nástroje. V rámci této práce je k návrhu použit program Enterprise Architect, který poskytuje nástroje pro modelování informačních systémů, jejich architektur, softwarových aplikací a databází [30].

Počet vztahů instancí, kterých se daná entita může účastnit, se nazývá kardinalita. Kardinalita je kategorizována dle třech základních typů vazeb:

- *One-to-one* (1:1) vazba – umožňuje vytvořit vazbu mezi jedním řádkem jedné tabulky a jedním řádkem druhé tabulky. Typickým příkladem může být tabulka osoby a tabulka s kontaktními detaily na danou osobu;
- *One-to-many* (1:N) vazba – nejběžnější typ vazby. Vytváří vazbu mezi jedním řádkem jedné tabulky a více řádky druhé tabulky;
- *Many-to-Many* (M:N) vazba – vytváří vazbu mezi více řádky jedné tabulky a více řádky druhé tabulky, tzn. více instancí jedné entity může mít vztah s více instancemi druhé entity. Příkladem mohou být studenti, kteří jsou přihlášeni do předmětů. Student může být přihlášen do více předmětů a předměty mohou mít přihlášeny více studentů. Tento typ vztahu je možné realizovat pomocí vazebné tabulky, která obsahuje referenční identifikátory pro obě entity;

Je nutné podotknout, že při procesu návrhu datového modelu a volbě kardinalit je také vhodné uvádět volitelnost daných vazeb. Obě strany vztahu mezi dvěma entitami mohou být nezbytné či volitelné [31].

Na následujícím obrázku je zobrazen elementární příklad jednoduchého fyzického datového modelu reprezentujícího one-to-many (1:N) vazbu mezi tabulkou kategorií a produktů. Vazba je vytvořena pomocí cizího klíče CategoryID v tabulce Products, který je vázán na primární klíč CategoryID tabulky Categories.



Obrázek 12. Fyzický datový model reprezentující vazbu 1:N mezi kategorií a produktem

Vazba ze strany produktu je označena jako volitelná (šipka s nulou), ze strany kategorie jako nezbytná (dvě čáry). Ve výsledku tato vazba znamená, že jednotlivé záznamy v tabulce kategorií mohou být vázány na žádné či více produktů.

Relační databáze mají úzkou souvislost se standardizovaným programovacím jazykem SQL (Structured Query Language).

3.1 Integrita dat

Pro udržování integrity relační databáze by se také měly dodržovat určitá integritní pravidla. Integritou databáze je myšleno zajištění toho, že data jsou konzistentní, přesná a vždy dostupná [32]. Integrita dat se rozděluje na dva typy – fyzickou a logickou.

Fyzická integrita se zabývá o úplnost a přesnost dat. Když jsou data uložena, přesunuta či je s nimi manipulováno, mohou ztratit svoji integritu. Mezi běžné faktory patří např. výpadky proudu, přírodní pohromy, napadení systému, nebo opotřebení úložiště [33].

Logická integrita zajišťuje, že data v kontextu relační databáze jsou vždy správná a neměnná. Mezi faktory ovlivňující logickou integritu patří hlavně lidské chyby a hacking. Častá kontrola chyb a používání validačních metod pomáhá zachovat logickou integritu dat [33]. Je rozdělena do čtyř kategorií:

- *Entitní integrita* je založena na konceptu primárních klíčů či hodnot které identifikují daná data. Účelem je zajistit, aby data nebyla zaznamenána vícekrát (tj. každý záznam dat byl jedinečný) a tabulka neobsahovala identifikační pole s hodnotou NULL [34];

- *Referenční integrita* zajišťuje, že data jsou uložena a používána jednotně. Cizí klíče, jež referují na hodnotu primárního klíče jiné tabulky by měly vždy souviset nebo obsahovat hodnotu NULL (vazba neexistuje nebo je neznámá) [33]. Referenční integrita zaručuje, že prostřednictvím pravidel o používání cizích klíčů dojde pouze k požadovaným změnám, přidáním nebo odebráním dat. Tato pravidla mohou zahrnovat podmínky, které odstraňují duplicitní záznamy či zakazují nevhodné záznamy [34];
- *Doménová integrita* zahrnuje pravidla a další procesy omezující formát, typ a objem dat zaznamenaných v databázi. Zajišťuje tedy, že každý sloupec v relační databázi je v definované doméně. Doménou je v tomto kontextu myšlena sada přijatelných hodnot, které může daný sloupec tabulky obsahovat [33];
- *Uživatelem definovaná integrita* poskytuje pravidla a omezení pro přizpůsobení specifickým požadavkům uživatele. Obvykle se používá v případě, že entitní, referenční a doménová integrita nestačí k udržení integrity dat [33]. Příkladem může být např. definované pravidlo, že maximální počet položek v nákupním košíku je třicet – v tomto případě musí být zamezeno mít více než třicet vazeb položek na danou objednávku.

K dosažení vysoké integrity dat a zamezení risku jejího narušení je vhodné např. omezovat přístup k datům, používat systém detekce a logování chyb a při manipulaci s daty vstupní hodnoty validovat. Pravidelné zálohování a inspekce dat může také pomoci udržet vysokou integritu [33].

3.2 SQL

SQL (Structured Query Language) je zkratka pro strukturovaný dotazovací jazyk, který je dle Amerického národního normalizačního institutu (ANSI) standardním jazykem pro práci se systémy relačních databází. Příkazy SQL se používají k vytváření dotazů, jako jsou úpravy dat nebo načítání dat z databáze [35].

Ačkoli většina databázových systémů používá SQL, většina z nich má také svá vlastní další proprietární rozšíření, které se obvykle používají pouze v jejich systému. Standardní příkazy jako SELECT, INSERT, UPDATE, DELETE, CREATE a DROP však lze použít k provedení téměř všeho, co je potřeba s běžnou databází [35].

Mezi běžné systémy pro správu relačních databází patří např. Microsoft SQL Server, MySQL, PostgreSQL či Oracle Database.

3.3 Microsoft SQL Server

Microsoft SQL Server je relační databázový a analytický systém vyvíjen společností Microsoft. Podobně jako u jiných databázových systémů, SQL Server je postaven na jazyce SQL. SQL Server používá své vlastní proprietární rozšíření jazyka SQL nazvané Transact-SQL (T-SQL) [36].

T-SQL rozšiřuje funkcionalitu a možnosti práce s daty a databázovými objekty. Umožňuje např. deklaraci proměnných, smyčky a podmínky, což z něj činí výkonným nástrojem k programování a správě databáze. T-SQL má také vestavěné funkce k provádění matematických operací, zpracování řetězců či výpočty dat zaměřující se na datum a čas. Umožňuje vytvářet a ukládat procedury, což jsou objekty databáze obsahující T-SQL dotazy které lze opakovaně provádět s různými vstupy [37].

Microsoft SQL Server fungoval exkluzivně v prostředí Windows více než 20 let. V roce 2016 ji Microsoft zpřístupnil pro operační systém Linux vydáním verze SQL Server 2017, od které je SQL Server spustitelný na obou operačních systémech [36].

4 BOOTSTRAP

Bootstrap je bezplatný CSS framework, který umožňuje rychlý vývoj responzivních webových rozhraní s konzistentním stylováním. Byl vytvořen dvěma vývojáři ve společnosti Twitter Inc. v roce 2010 – Markem Ottem a Jacobem Thorntonem. Následující rok byl vydán jako open-source projekt a od té doby je jedním z nejpoužívanějších frameworků ve vývoji uživatelských rozhraní. Snadné použití společně s kompatibilitou pro různé prohlížeče a zařízení z něj činí velmi dobrý stavební kámen pro jakoukoli moderní webovou aplikaci [38].

Bootstrap je strukturován jako knihovna obsahující HTML, CSS a JS soubory. Tato kapitola předpokládá, že čtenář byl již seznámen a má alespoň základní znalost těchto jazyků.

Obrázek 13. Logo Bootstrap 5.2

4.1 Rozložení obsahu

Pravděpodobně nejdůležitější a nejpoužívanější funkcí frameworku Bootstrap jsou možnosti rozložení a uspořádání obsahu uživatelského rozhraní.

Bootstrap disponuje grid (mřížkovým) systémem, který pomocí CSS tříd a @media pravidel umožňuje definovat horizontální a vertikální umístění jednotlivých HTML prvků s ohledem na různé velikosti obrazovky. Grid systém je konceptuálně velmi jednoduchý a k jejímu používání stačí znalost několika CSS tříd. Mezi tyto třídy patří primárně *row* a *col*, pomocí kterých je možné prvky uspořádat do řádků a sloupců [38].

K použití grid systému je vhodné jednotlivé prvky vnořit do tzv. *containeru*. Ten slouží k dynamickému nastavení prvků uvnitř – např. nastavení šířky vnějších okrajů (padding) [39].

Od verze 4.0 Bootstrap obsahuje také podporu pro systém rozložení CSS3 zvaný flexbox. Flexbox umožňuje snadnější umístění prvků pro zohlednění různých velikostí obrazovky a dynamického zarovnání prvků [38].

4.2 Stylování obsahu

Jak již bylo zmíněno, jeden z cílů frameworku Bootstrap je dosáhnout konzistentního stylování napříč celé webové aplikace. Tzn. webové aplikace založené na frameworku Bootstrap by se měly chovat a vypadat stejně bez ohledu na to, který prohlížeč je uživatelem používán.

Jelikož prohlížeče nastavují výchozí stylování jednotlivých prvků různě, Bootstrap je přepisuje svými vlastními styly a umožňuje tak konzistenci napříč prohlížeči. Přepisování je zprostředkováno pomocí sady Bootstrap CSS pravidel kolektivně nazývaných *Reboot* [38].

Bootstrap poskytuje sadu základních stylů, které umožňují vylepšit celkový vzhled a dojem nejběžněji používaných prvků k vytvoření uživatelského rozhraní.

Obrázek 14. Výchozí stylování tlačítek a barev v Bootstrap 5.2

4.3 Komponenty

Bootstrap nabízí sadu základních komponent, které umožňují rychle vytvořit jakýkoliv základní typ uživatelského rozhraní. V rámci verze 5.2 Bootstrap nabízí pro příklad následující komponenty:

- Modal, což je vyskakovací obrazovka např. k potvrzení dané akce;
- Ukazatele průběhu ve formě pruhu nebo točícího se elementu;
- Karty k rozložení a stylování různých typů obsahu;
- Dropdown, který slouží ke kontextuálnímu překrytí obsahu k zobrazení např. seznamu odkazů;
- Navigační lišta aplikace;
- Tlačítka pro stránkování obsahu.

Výše uvedené komponenty jsou pouze jedny z několika dostupných v Bootstrap 5.2, jejich funkcionalita ovšem výrazně rozšiřuje možnosti vývoje a jsou vždy konzistentní s celkovým vzhledem uživatelského rozhraní [38].

II. PRAKTICKÁ ČÁST

5 TÉMA UKÁZKOVÉ APLIKACE

K odpovídající demonstraci funkčnosti a výkonu technologií ASP.NET Blazor a relačních databází byl vybrán vývoj aplikace na téma elektronického obchodování.

E-shopy jsou dnes velmi populární a často vytvářené webové aplikace. Často se také jedná o velmi rozmanité a dynamické aplikace řešící různé funkce a procesy, kterými se vhodně testují možnosti vybraných technologií. Mezi řešené funkce a procesy jsou např. zákaznické účty, objednávky, platby, správa produktů, kategorií a další.

Jsou také vhodným příkladem k vysvětlení použití relačních databází, protože zpravidla obsahují mnoho tabulek, které jsou mezi sebou propojeny. Studenti mohou tak lépe porozumět tomu, jak funguje práce s databázemi v reálném prostředí a jakým způsobem mohou efektivně modelovat data.

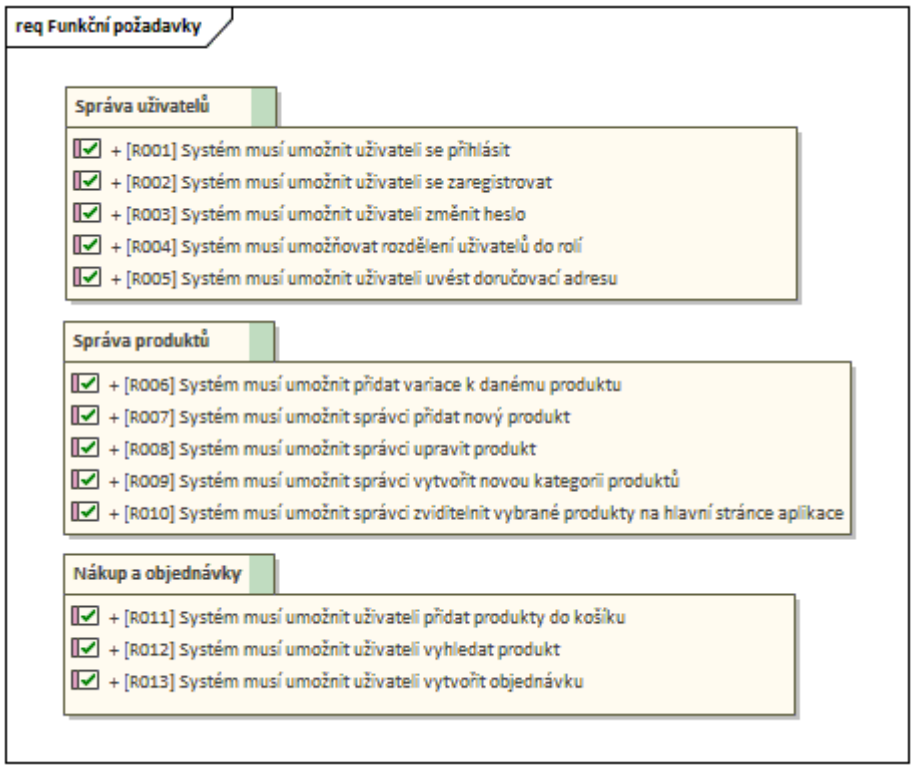
Celkově je vytvoření ukázkové aplikace ve stylu e-shopu užitečným způsobem, jak demonstrovat funkce a výkon Blazor aplikace s relační databází a současně tak poskytnout studentům praktické zkušenosti s tvorbou jednostránkových webových aplikací (SPA).

6 NÁVRH UKÁZKOVÉ APLIKACE

Nejprve byla provedena analýza požadavků. Požadavky byly konceptualizovány dle existujících řešení na veřejném webu. Jako hlavní požadavky byla zvolena implementace funkce košíku, vytvoření objednávky a implementace administrativní části aplikace přístupnou uživateli s manažerskou rolí. Jednotlivé požadavky byly rozděleny na funkční a nefunkční.

6.1 Funkční požadavky

Funkční požadavky byly rozděleny do třech hlavních kategorií. Každá z nich se zaměřuje na určitou funkcionalitu systému aplikace.



Obrázek 15. Funkční požadavky

Správa uživatelů popisuje požadavky soustředící se na správu uživatelských účtů a jejich detailů.

- [R001] Uživateli musí být umožněno se přihlásit pomocí e-mailu a hesla;
- [R002] Uživateli musí být umožněno se zaregistrovat pomocí e-mailu a hesla. V registračním formuláři musí být heslo zopakováno;
- [R003] Po přihlášení uživatele musí být umožněno uživateli změnit heslo. Formulář by měl být ve stejném formátu jako registrační formulář;

- [R004] Systém musí umožnit rozdělit uživatele do rolí. Pro základní funkcionalitu by měly být realizovány role správce a běžného uživatele;
- [R005] Po přihlášení uživatele musí být umožněno uvést doručovací adresu.

Správa produktů obsahuje požadavky řešící správu produktů, jejich kategorií a variací.

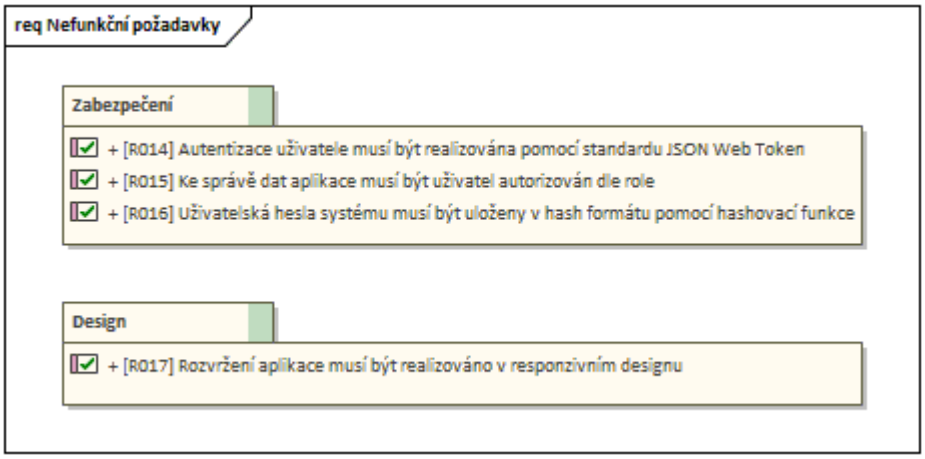
- [R006] Uživateli v roli správce musí být umožněno přidat variace k danému produktu. Typickým příkladem může být např. nápoj v různých objemech;
- [R007] Uživateli v roli správce musí být umožněno přidat nový produkt;
- [R008] Uživateli v roli správce musí být umožněno upravit daný produkt. Úpravou je myšleno také smazání daného produktu. Po smazání daného produktu nesmí být daný záznam odstraněn z databáze, ale pouze označen jako odstraněný;
- [R009] Uživatel v roli správce musí být schopen přidat novou kategorii produktů. Příkladem kategorie produktů mohou být např. nápoje, do kterých jsou zařazeny pouze nápoje;
- [R010] Uživatel v roli správce musí být schopen zviditelnit dané produkty na hlavní stránce aplikace.

Nákup a objednávky popisuje požadavky soustředící se na nákup a vytvoření objednávky.

- [R011] Musí být zaveden systém košíku. Nepřihlášený či přihlášený uživatel bude schopen přidat libovolné množství daného produktu do košíku a daný košík poté zobrazit před evaluací objednávky;
- [R012] Systém musí umožnit jakémukoliv uživateli (i nepřihlášenému) vyhledat produkty pomocí vyhledávacího formuláře;
- [R013] Přihlášený uživatel musí být schopen po přidání produktu do košíku vytvořit objednávku.

6.2 Nefunkční požadavky

Nefunkční požadavky byly navrženy tak, aby bylo umožněno dostatečného zabezpečení navrhované aplikace a rozvržení bylo responzivní pro všechny typy zařízení.

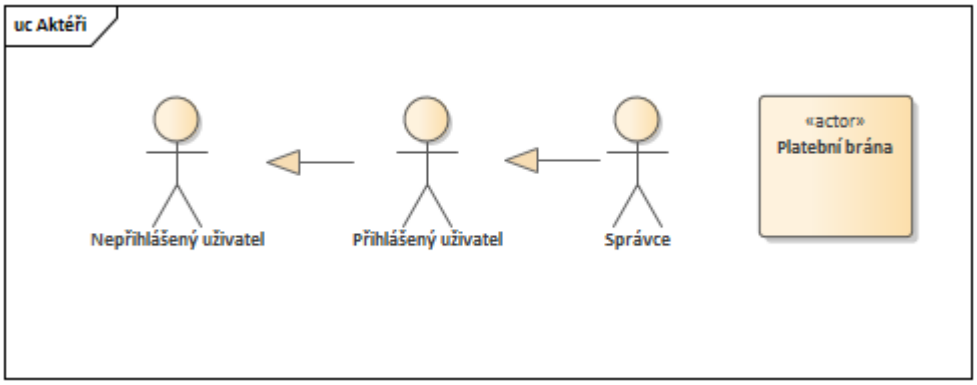


Obrázek 16. Funkční požadavky

- [R014] Autentizace uživatele musí být realizována pomocí standardu JSON Web Token. Základní koncepty JWT byly představeny v teoretické části v kapitole 1.3.4;
- [R015] Pro přístup do sekcí administrace dat aplikace (správa produktů, objednávek apod.) musí být uživatel autorizován, tj. mít roli správce;
- [R016] Uživatelská hesla musí být v databázovém systému uložena v hash formátu pomocí hashovací funkce;
- [R017] Design aplikace musí být responzivní, tzn. stylování uživatelského rozhraní musí být funkční pro desktopová i mobilní zařízení.

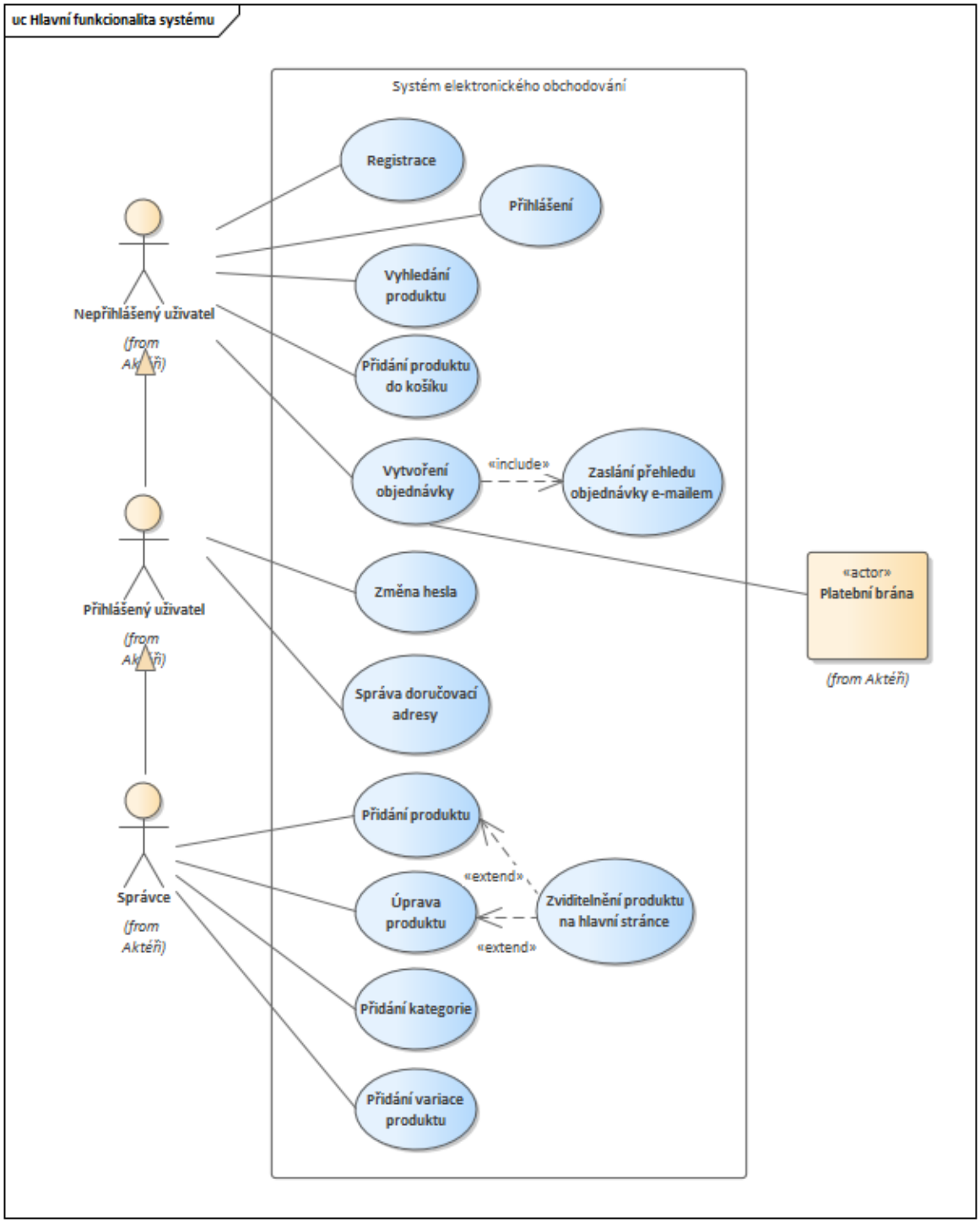
6.3 Případy užití

V této kapitole jsou nastíněny případy užití navržené dle předchozí analýzy požadavků a scénáře klíčových případů užití. Na následujícím diagramu jsou představeni aktéři systému, kteří jsou poté vyobrazení v diagramu případů užití.



Obrázek 17. Aktéři systému

Následující diagram případů užití zobrazuje chování systému ze strany jednotlivých aktérů:



Obrázek 18. Diagram případů užití pro hlavní funkcionalitu systému

6.3.1 Scénáře případů užití

Tato kapitola popisuje klíčové scénáře případů užití systému navrhované aplikace. Jsou představeny scénáře pro registraci uživatele, přidání nového produktu ze strany správce a vytvoření nové objednávky v případě registrovaného i neregistrovaného uživatele.

Tabulka 1. Scénář registrace

Název	Registrace	
Popis	Zachycení procesu registrace nového uživatele do systému.	
Aktéři	Nepřihlášený uživatel	
Vstupní podmínky	• Aktér není Přihlášený uživatel • E-mail registrovaného uživatele není používán jiným uživatelským účtem	
Výstupní podmínky	• Nový uživatel je úspěšně zaveden do systému • Vytvořený uživatel je automaticky přihlášen	
Hlavní scénář		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko "Registrovat" v navigační liště
2	Systém	Systém zobrazí stránku s registračním formulářem
3	Aktér	Aktér vyplní registrační formulář
4	Aktér	Aktér klikne na tlačítko "Registrovat"
5	Systém	Systém provede validaci zadaných dat
6	Systém	Systém zavede nového uživatele a přihlásí jej
7	Systém	Systém přesměruje aktéra na hlavní stránku
Alternativní scénáře		
–		
Výjimky		
5a Neúspěšná validace zadaných dat		

Tabulka 2. Výjimka registrace – Neúspěšná validace zadaných dat

Název	Registrace – Neúspěšná validace zadaných dat	
Popis	Nepřihlášený uživatel zadal údaje, které nevyhovují pravidlům definovaných systémem, případně zadal údaje již existují v rámci jiného uživatelského účtu	
Scénář výjimky		
Krok	**Aktér/Systém**	**Popis**
1	Systém	Systém zobrazí hlášky o nevalidním vyplnění údajů, popisující porušená pravidla a validní formát. Případ použití Registrace pokračuje od kroku 3 hlavního scénáře – uživatel musí zadat validní data.

Tabulka 3. Scénář vytvoření produktu

Název	Vytvoření produktu	
Popis	Zachycení procesu vytvoření nového produktu	
Aktéři	Správce	
Vstupní podmínky	• Aktér je Správce (přihlášený do systému)	
Výstupní podmínky	• Nový produkt je úspěšně zaveden do systému • Správce je přesměrován na správu produktů	
Hlavní scénář		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko svého přihlášení v navigační liště
2	Systém	Systém zobrazí navigační menu s odkazy do jednotlivých částí aplikace
3	Aktér	Aktér klikne na tlačítko "Produkty" v navigačním menu
4	Systém	Systém zobrazí stránku se seznamem všech produktů
5	Aktér	Aktér klikne na tlačítko pro přidání produktu
6	Systém	Systém zobrazí stránku s formulářem pro přidání nového produktu
7	Aktér	Aktér vyplní povinné, případně nepovinné, údaje pro přidání nového produktu
8	Aktér	Aktér klikne na tlačítko "Přidat"
9	Systém	Systém provede validaci zadaných dat
10	Systém	Systém přesměruje aktéra na správu produktů
Alternativní scénáře		
7a Přidání varianty produktu		
Výjimky		
9a Neúspěšná validace zadaných dat		

Tabulka 4. Alternativní scénář přidání produktu – Přidání varianty

Název	Vytvoření produktu – Přidání varianty	
Popis	V rámci vyplnění údajů je správci umožněno přidat variantu vytvářeného produktu	
Alternativní scénář		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Aktér klikne na tlačítko přidání varianty
2	Aktér	Aktér vyplní údaje varianty produktu
3	Systém	Systém přidá variantu do seznamu variant přidávaného produktu

Tabulka 5. Výjimka přidání produktu – Neúspěšná validace zadaných dat

Název	Vytvoření produktu – Neúspěšná validace zadaných dat	
Popis	Při přidání produktu správce vyplnil údaje, které nevyhovují pravidlům definovaných systémem	
Scénář výjimky		
Krok	**Aktér/Systém**	**Popis**
1	Systém	Systém zobrazí hlášky o nevalidním vyplnění údajů, popisující porušená pravidla a validní formát. Případ použití Přidání produktu pokračuje od kroku 7 hlavního scénáře – uživatel musí zadat validní data.

Tabulka 6. Scénář vytvoření objednávky

Název	Vytvoření objednávky	
Popis	Zachycení procesu vytvoření nové objednávky, provedení transakce a zaslání e-mailu s potvrzením objednávky	
Aktéři	Nepřihlášený uživatel, Přihlášený uživatel, Platební brána	
Vstupní podmínky	• Přihlášený/Nepřihlášený uživatel má alespoň jeden produkt v košíku	
Výstupní podmínky	• Úspěšné vytvoření nové objednávky • Uživateli je odeslán e-mail s potvrzením objednávky	
Hlavní scénář		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Nepřihlášený či přihlášený uživatel klikne na tlačítko košíku v navigační liště
2	Systém	Systém zobrazí stránku s obsahem košíku rozpracované objednávky
3	Aktér	Nepřihlášený či přihlášený uživatel klikne na tlačítko „Pokračovat“ k zobrazení stránky pro provedení platby.
4	Systém	Systém zobrazí stránku s platební bránou a přehledem objednávky
5	Aktér	Platební brána poskytne rozhraní k provedení platby
6	Systém	Systém předvyplní dostupné údaje do rozhraní platební brány
7	Aktér	Platební brána zobrazí možnosti platby
8	Aktér	Nepřihlášený či přihlášený uživatel vybere možnost platby
9	Aktér	Nepřihlášený či přihlášený uživatel doplní chybějící údaje
10	Aktér	Nepřihlášený či přihlášený uživatel provede platbu
11	Systém	Systém změní stav objednávky na úspěšnou transakci
12	Systém	Systém zobrazí hlášku a odešle uživateli potvrzovací e-mail o úspěšné transakci

Alternativní scénáře
9a Uvedení údajů u přihlášeného uživatele
Výjimky
10a Platba nebyla provedena úspěšně

Tabulka 7. Alternativní scénář vytvoření objednávky – Uvedení údajů u přihlášeného uživatele

Název	Vytvoření objednávky – Uvedení údajů u přihlášeného uživatele	
Popis	U přihlášeného uživatele se údaje předvyplní a neumožní možnost zadání e-mailu odlišného od přihlašovacího e-mailu	
Alternativní scénář		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Platební brána přihlášenému uživateli schová vstup pro zadání e-mailu a předvyplní údaje uvedené na svém profilu

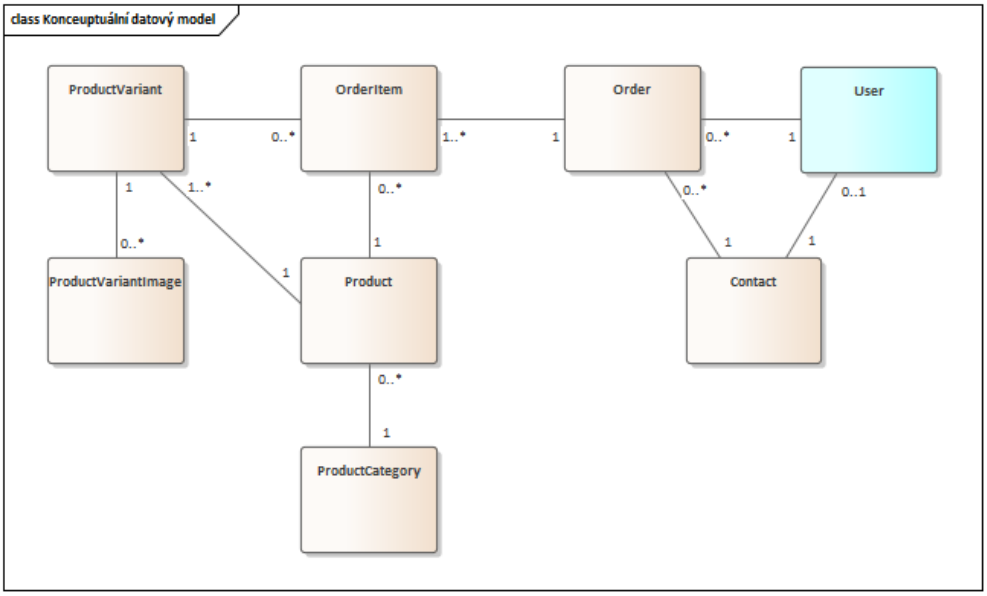
Tabulka 8. Výjimka vytvoření objednávky – Platba nebyla provedena úspěšně

Název	Vytvoření objednávky – Platba nebyla provedena úspěšně	
Popis	Při provedení platby ze strany platební brány nastala chyba	
Scénář výjimky		
Krok	**Aktér/Systém**	**Popis**
1	Aktér	Platební brána zobrazí hlášku o neúspěšné platbě
2	Systém	Systém změní stav objednávky na neúspěšnou platbu a umožní uživateli platbu zopakovat od 2. kroku hlavního scénáře

6.4 Návrh databáze

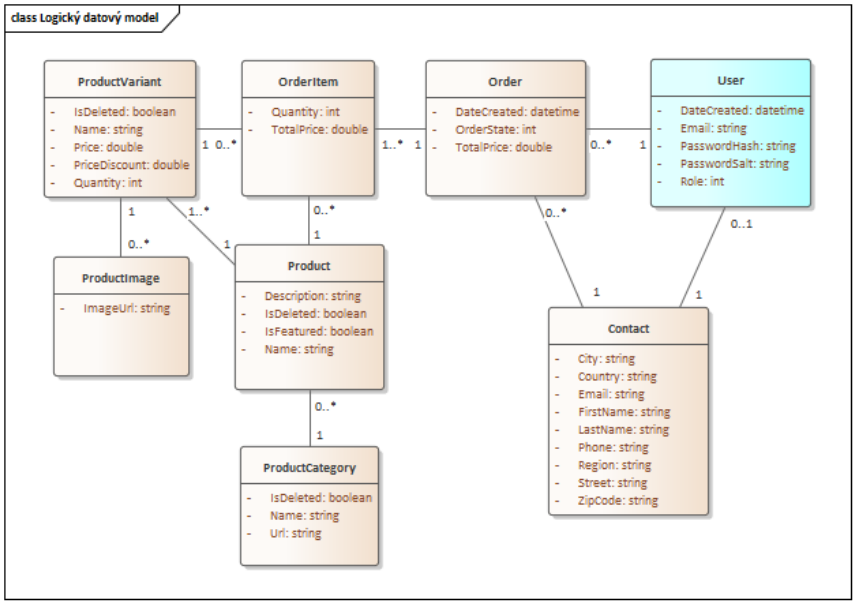
Návrh databáze byl proveden vytvořením datového modelu ve třech úrovních abstrakce:

1. *Konceptuální datový model*, u kterého byly definovány jednotlivé entity a vazby mezi nimi;



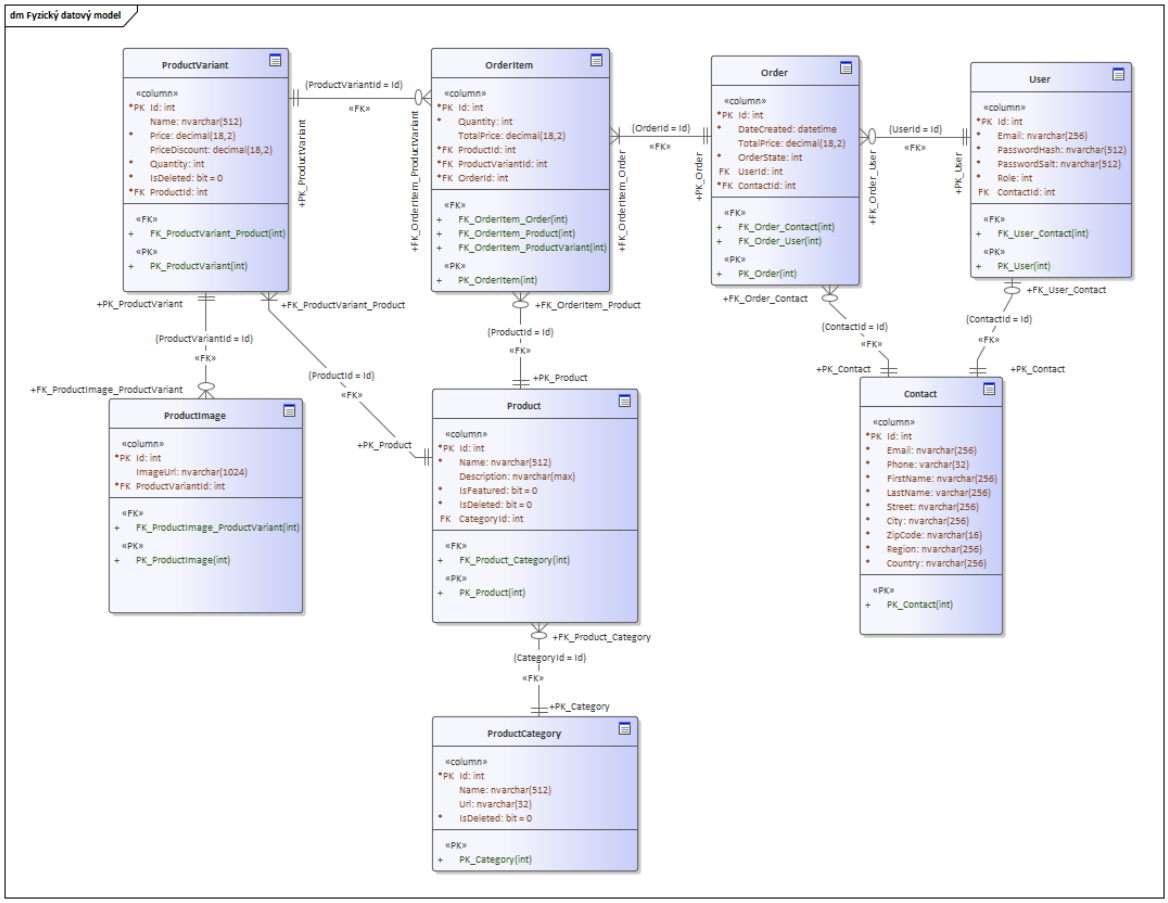
Obrázek 19. Konceptuální datový model

2. *Logický datový model*, u kterého byly přidány hlavní atributy jednotlivých entit definovaných v konceptuálním modelu. Každý z atributů má také uveden abstraktní datový typ (nespecifický pro daný databázový systém);



Obrázek 20. Logický datový model

3. *Fyzický datový model* jednoznačně definoval vazby pomocí atributů ve formě primárních a cizích klíčů. Datové typy atributů jednotlivých entit byly specifikovány dle databázového systému Microsoft SQL Server.



Obrázek 21. Fyzický datový model

7 POPIS UKÁZKOVÉ APLIKACE

Tato kapitola se zabývá popisem vyvinuté ukázkové aplikace. Jsou představeny použité technologie, způsob vytvoření projektu a jeho rozvržení. Nakonec jsou detailně popsány implementace klíčových částí aplikace.

7.1 Použité technologie a služby

Vyvíjená aplikace byla postavena na následujících technologiích:

- .NET 7.0 (runtime 7.0.3, SDK 7.0.201),
- ASP.NET Core 7.0.3,
- Blazor WebAssembly,
- Entity Framework Core 7.0.3,
- SQL Server 2019,
- Bootstrap 5.2.3,
- Bootstrap Icons 1.10.0

Aplikace byla vyvíjena v IDE Visual Studio 2022 Community. K zahájení vývoje byla použita šablona „Blazor WebAssembly Empty“ s volbou „ASP.NET Core Hosted,“ která inicializovala projekt ASP.NET Core serveru a knihovnu „Shared“ pro sdílený kód. Serverový projekt umožnil hostování klientského Blazor WebAssembly projektu.

V rámci serverové části aplikace byly použity následující balíčky:

- MailKit 3.6.0 (knihovna implementující e-mailové klienty k odesílání e-mailových zpráv),
- Swashbuckle 6.5.0 (OpenAPI specifikace k dokumentaci a testování REST API endpointů serveru)

Na straně klienta byl použit balíček Blazored.LocalStorage 4.3.0 ke zjednodušené práci s local storage uživatelského prohlížeče.

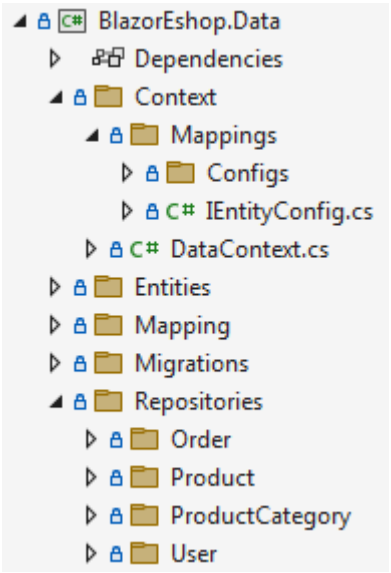
K implementaci platební brány byla využita služba PayPal Developer. Tato služba byla vybrána především z důvodu možnosti bezplatného použití a snadné integrace do webových aplikací. PayPal API poskytuje připravené JavaScript funkce, lokalizaci pro český jazyk a vytvoření testovacích platebních údajů.

7.2 Rozvržení projektu

K lepší organizaci a znuvupoužitelnosti kódu byl projekt vyvíjené aplikace rozdělen do podprojektů a knihoven, které jsou popsány v následujících kapitolách.

7.2.1 Data

Tato knihovna umožňuje logické oddělení datové vrstvy od zbytku aplikační logiky (tzv. vzor „repository pattern“). Obsahuje třídy nastavující entity a ORM dle Entity Framework Core. Dále obsahuje rozhraní a třídy pod složkou „Repositories,“ jež implementují dotazy na data pomocí dotazovacích metod. Knihovna také udržuje EF Core migrace.

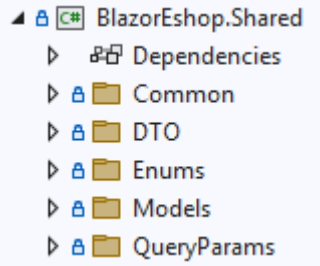


Obrázek 22. Knihovna „Data“

7.2.2 Shared

V „Shared“ knihovně je implementován kód používaný zároveň v serverové a klientské části aplikace. Jsou zde definovány modely a data transfer objekty (DTO), které slouží jako prostředník v procesu komunikace mezi serverem a klientem. Dále jsou zde definovány číselníky rolí, stavů objednávek a dalších konstant k jednotné implementaci pro všechny části aplikace.

Zde je vhodné podotknout jednu z výhod vývoje Blazor aplikací. Používání jednotných rozhrání, tříd a číselníků umožňuje udržování konzistentních referencí v kódu a minimalizuje konflikty napříč verzemi v případě vývoje ve větších týmech.

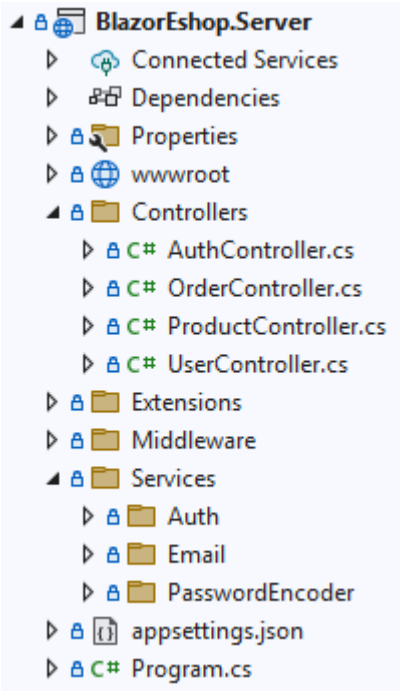


Obrázek 23. Knihovna „Shared“

7.2.3 Server

Tento projekt implementuje ASP.NET Core server aplikace. Jsou zde prováděny server-side operace, jako je hostování statických souborů, endpointů API a další aplikační logiky jako je např. autentizace nebo odesílání e-mailů.

K implementaci REST API jsou použity Web API controllery, které definují routing, dotazování na data skrz knihovnu „Data,“ jejich mapování na DTO a patřičné odeslání odpovědi na HTTP požadavek dle implementovaných podmínek. Je zde také implementován middleware pro zpracování výjimek napříč celého serveru (exception handler).

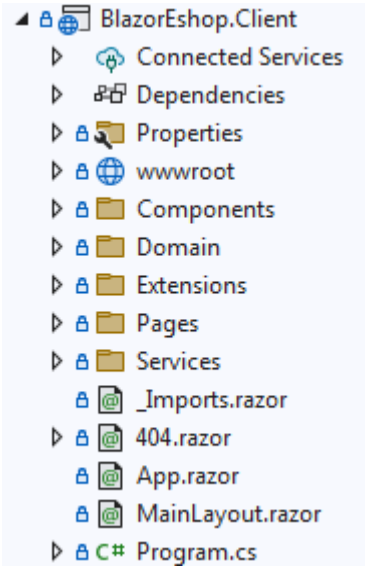


Obrázek 24. Projekt serveru

7.2.4 Client

Tento projekt obsahuje Blazor WebAssembly klienta, který implementuje logiku a vzhled uživatelského rozhraní aplikace. Obsahuje stránky a jednotlivé komponenty aplikace, jako je např. generická tabulka, kterou je možné použít pro různé modely napříč více komponent.

Dále implementuje rozhraní a třídy k dotazování a zpracování odpovědi serveru a spravuje autentizaci na straně uživatele (lokální uložení JWT).



Obrázek 25. Projekt klienta

7.3 Klíčové části aplikace

V následujících kapitolách jsou popsány klíčové části aplikace. Je zde představeno připojení k databázi, popsána definice příkladové entity a její mapování v databázovém kontextu. Dále jsou popsány metodiky zabezpečení aplikace na straně serveru a klienta.

Na straně klienta je představena funkcionalita routování stránek v rámci Blazor aplikace, implementace vlastní generické Razor komponenty a využití JavaScript interoperability k vytvoření nové objednávky z dat úspěšné platby pomocí služby PayPal.

7.3.1 Spojení a správa relační databáze

Databáze byla vytvořena a spravována použitím Entity Framework Core code-first přístupem. Nejprve byly definovány entity dle navrženého fyzického modelu, které sloužily k mapování na databázové tabulky.

Pro každou entitu byla implementována dle definovaného rozhraní IEntityConfig konfigurační třída, která nastavuje pomocí Fluent API vazby s dalšími entitami a chování při smazání záznamu. Jako příklad je níže uveden kód entity User (obrázek 26). Entita User má one-to-one vazbu na entitu Contact, která obsahuje kontaktní data uživatele. Definováno je také chování při smazání (DeleteBehavior), které je nastaveno na „Cascade," to umožňuje smazání záznamů i z vázané tabulky v případě smazání záznamu této entity.

```
public class UserConfig : IEntityConfig
{
    public void Configuration(ModelBuilder model-
Builder)
    {
        var entity = modelBuilder.Entity<User>();

        entity.HasKey(e => e.Id);

        entity.HasOne(e => e.Contact)
            .WithOne(e => e.User)
            .HasForeignKey<User>(e => e.ContactId)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Obrázek 26. Konfigurace entity User

Ve třídě databázového kontextu jsou následně veškeré konfigurace entit aplikovány v implementaci metody OnModelCreating zděděné DbContext třídy.

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    foreach (var config in _entityConfig)
    {
        config.Configuration(modelBuilder);
    }

    SeedData(modelBuilder);

    base.OnModelCreating(modelBuilder);
}
```

Obrázek 27. Konfigurace entit v metodě OnModelCreating

Databázový kontext je zaregistrován do aplikace pomocí dependency injection v souboru Program.cs serverového projektu.

```
builder.Services.AddDbContext<DataContext>(dbOb =>
    dbOb.UseSqlServer(
        builder.Configuration.GetConnection-
String("BlazorEshopConnection"),
        sqlOb =>
        {
            sqlOb.EnableRetryOnFailure(
                maxRetryCount: 5,
                maxRetryDelay: TimeSpan.FromSeconds(3),
                errorNumbersToAdd: null);
        }
    )
);
```

Obrázek 28. Přidání a konfigurace databázového kontextu

ConnectionString je řetězec, který se používá pro připojení k databázi. V ASP.NET Core aplikaci je možné tento řetězec definovat v souboru appsettings.json. Případně je možné upřesnit konfiguraci pro různá vývojová prostředí (např. vývojové, testovací nebo produkční) vytvořením separátních konfigurační souborů ve formátu appsettings.{název_prostředí}.json.

Pro vytvoření migrací databáze byl použit příkaz „dotnet ef migrations add <název_migrace>". Vytvořené migrace definovaly změny, které je potřeba provést v databázi. Následně byl použit příkaz „dotnet ef database update" k provedení změn definovaných v poslední migraci.

7.3.2 Zabezpečení citlivých dat

V předchozí kapitole bylo představeno upřesnění konfigurací v ASP.NET Core aplikaci pomocí konfiguračního souboru appsettings.json. Další místo, kde je možné konfigurace definovat je v user secrets daného projektu. User secrets umožňují vytvořit na lokálním zařízení konfigurační JSON soubor, který posléze není součástí verzovacího systému projektu a zamezí tak nechtěnému šíření citlivých informací, jako jsou řetězce pro připojení k databázi, údaje pro připojení k e-mailovému serveru nebo heslo pro certifikát podepisovacího klíče.

Obsah tohoto souboru je poté používán jako konfigurační soubor spouštěného projektu. Přístup k tomu souboru je umožněn pravým kliknutím na daný projekt a výběrem volby „Manage user secrets…" v anglické verzi Visual Studio IDE.

7.3.3 Zamezení SQL injection

K zamezení SQL injection byly veškeré dotazy na databázi složeny pomocí query builder metod, ze kterých Entity Framework Core vytváří parametrizované dotazy. Následující metoda slouží k získání seznamu produktů dle vstupních parametrů:

```
public async Task<IEnumerable<Entities.Product>> GetProd-
uctsQuery(ProductQueryParams queryParams)
{
	var query = _context.Products
		.Include(p => p.ProductCategory)
		.Include(p => p.ProductVariants)
			.ThenInclude(v => v.ProductImages)
		.Where(p => !p.IsDeleted)
		.Where(p => string.IsNullOrEmpty(queryParams.CategoryUrl) ||
p.ProductCategory.Url.ToLower() == queryParams.CategoryUrl.ToLower())
		.Where(p => string.IsNullOrEmpty(queryParams.Query) ||
EF.Functions.Like(p.Name, $"%{queryParams.Query}%"))
		.AsQueryable();

	if (queryParams.Count != null)
	{
		query = query.Take(queryParams.Count.Value);
	}

	return await query
		.AsSplitQuery()
		.OrderByDescending(p => p.Id)
		.ToListAsync();
}
```

Obrázek 29. Metoda pro získání seznamu produktů dle vstupních parametrů

Query builder metody Include() a ThenInclude() vytvářejí dotazy na vázané tabulky pomocí klauzule JOIN. Z důvodu definovaných vazeb v entitách při dotazování dojde k vytvoření dvojitého JOINu, z toho důvodu je zde použita metoda AsSplitQuery(), aby se dotazy rozdělily na více dotazů a zamezilo se odesílání velkého množství dat na klienta.

7.3.4 Autentizace uživatele

Ke správě uživatelských účtů se autor rozhodl namísto použití API knihovny ASP.NET Core Identity implementovat vlastní uživatelský model (viz obrázek 30), službu pro hashování hesel a autentizační službu. Jedním z důvodů byla snaha o srozumitelnější kód a jednodušší kódovou bázi v rámci celé aplikace. ASP.NET Core Identity obsahuje mnoho funkcí, které není třeba využít v každém projektu, navíc může být složitější k porozumění a nastavení pro začátečníky. Použitím vlastního řešení autor chtěl pomoci studentům snadněji porozumět jak se autentizace a autorizace v aplikaci provádí na nižší úrovni abstrakce.

```
public class User
{
    [Column(TypeName = "int")]
    public int Id { get; set; }

    [Column(TypeName = "nvarchar(256)")]
    public string Email { get; set; }

    [Column(TypeName = "nvarchar(512)")]
    public string PasswordHash { get; set; }

    [Column(TypeName = "nvarchar(512)")]
    public string PasswordSalt { get; set; }

    [Column(TypeName = "int")]
    public UserRole Role { get; set; }

    [Column(TypeName = "int")]
    public int? ContactId { get; set; }

    public Contact? Contact { get; set; }
}
```

Obrázek 30. Entita uživatele

Role uživatele je určena pomocí číselníku UserRole, dle kterého je poté v databázi uložena hodnota s datovým typem int.

```
public enum UserRole
{
    Customer = 0,
    Manager = 1
}
```

Obrázek 31. Role uživatele

Pro hashování hesel byl využit symetrický šifrovací algoritmus HS256 (sdílený kód HMAC s hashovací funkcí SHA256) implementovaný v .NET namespace Systém.Security.Cryptography. Na následujícím obrázku je zobrazen kód třídy implementující metody k hashování hesel pomocí algoritmu HS256.

```
public class HS256PasswordEncoder : IPasswordEncoder
{
	public byte[] GenerateSalt(int byteLength)
	{
		return RandomNumberGenerator.GetBytes(byteLength);
	}

	public byte[] GenerateSaltedPasswordHash(string password,
byte[] salt)
	{
		using var hs256 = new HMACSHA256(salt);
		return hs256.ComputeHash(System.Text.Encoding
			.UTF8.GetBytes(password));
	}

	public bool IsPasswordHashValid(string password, byte[]
passwordHash,
		byte[] salt)
	{
		var generatedHash = GenerateSaltedPasswordHash(pass-
word, salt);
		return generatedHash.SequenceEqual(passwordHash);
	}
}
```

Obrázek 32. Třída k hashování hesel algoritmem HS256

Pro každé heslo se náhodně vygeneruje sůl (náhodná hodnota o stanovené délce, která se přidá k heslu před hashováním), což zajišťuje, že dvě stejná hesla se nebudou mapovat na stejný hash. Poté se k vytvoření hashe použije HMAC-SHA256 se solí a heslem jako klíčem. Takto vytvořený hash se uloží do databáze v base64 formátu. Vygenerovaná sůl je rovněž uložena do databáze pro použití při porovnání hesel u přihlašování nebo změně hesla.

V případě potřeby použití jiného algoritmu (např. HMAC-SHA512) byla třída implementovaná dle vytvořeného rozhraní IPasswordEncoder. Použitý algoritmus je tímto způsobem možné specifikovat registrací třídy pomocí dependency injection v Program.cs serverového projektu:

```
builder.Services.AddScoped<IPasswordEncoder, HS256PasswordEncoder>();
```

Zaregistrovaná třída je poté použita v rámci třídy AuthService, která implementuje metodu pro generování JWT tokenu (viz obrázek 33), registraci, přihlášení a změnu hesla.

```
public string GenerateToken(User user)
{
	var tokenHandler = new JwtSecurityTokenHandler();
	var secretKey = new X509SecurityKey(
		new X509Certificate2(
			_configuration.GetSection("AppSettings:Auth:CertFile-
Name").Value,
			_configuration.GetSection("AppSettings:Auth:CertPass-
word").Value
		)
	);
    var expirationDate = DateTime.Now.AddHours(Convert.ToInt32(_configura-
tion.GetSection("AppSettings:Auth:ExpiresInHours").Value));

	var tokenDescriptor = new SecurityTokenDescriptor
	{
		Subject = new ClaimsIdentity(new List<Claim>
		{
			new Claim(ClaimTypes.NameIdentifier,
user.Id.ToString()),
			new Claim(ClaimTypes.Name, user.Email),
			new Claim(ClaimTypes.Email, user.Email),
			new Claim(ClaimTypes.Role, user.Role.ToString()),
			new Claim(ClaimTypes.Expiration, expira-
tionDate.ToString())
		}),
		Expires = expirationDate,
		SigningCredentials = new SigningCredentials(secretKey, Secu-
rityAlgorithms.RsaSha256)
	};

	var token = tokenHandler.CreateToken(tokenDescriptor);
	return tokenHandler.WriteToken(token);
}
```

Obrázek 33. Metoda pro generování JWT

Metoda pro generování tokenu obsahuje v payloadu identifikační claims definující data o uživateli. Dle konfiguračního souboru je také použit self-signed X.509 certifikát k podepsání tokenu a nastavení jeho expirační doby. Po úspěšné validaci hesel je token uživateli poslán v odpovědi HTTP požadavku přihlášení.

X.509 certifikát byl podepsán samotným autorem lokálně v PowerShell terminálu dle následujících příkazů:

```
> $cert = New-SelfSignedCertificate -DnsName "blazoreshopwasm" -CertStoreLo-
cation "Cert:\LocalMachine\My"
> $pwd = ConvertTo-SecureString -String 'ukazkove_heslo' -Force -AsPlainText
> $path = 'Cert:\LocalMachine\My\' + $cert.Thumbprint
> Export-PfxCertificate -Cert $path -FilePath 'C:\certs\blazoreshop-
wasm_cert.pfx' -Password $pwd
```

Obrázek 34. Vytvoření self-signed certifikátu v PowerShell terminálu

V Program.cs serverového projektu bylo posléze nastaveno autentizační schéma, které bylo specifikováno pro standard JWT. Jako podepisovací klíč byl nastaven self-signed X.509 certifikát:

```
builder.Services.AddAuthentication(JwtBearerDefaults.Authentica-
tionScheme)
    .AddJwtBearer(o =>
    {
        o.TokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = false,
            ValidateAudience = false,
            IssuerSigningKey = new X509SecurityKey(
                new X509Certificate2(
                    builder.Configuration.GetSection("AppSet-
tings:Auth:CertFileName").Value,
                    builder.Configuration.GetSection("AppSet-
tings:Auth:CertPassword").Value
                )
            )
        };
    });
```

Obrázek 35. Konfigurace autentizace

7.3.5 Autorizace požadavků

Určité endpointy implementované v API controllerech bylo potřeba autorizovat pro přihlášené uživatele nebo uživatele s rolí manažera. K tomu byla v souboru Program.cs serverového projektu použita metoda AddAuthorization() s příslušným nastavením:

```
builder.Services.AddAuthorization(o =>
{
      o.AddPolicy(Constants.PolicyManagerOnly, p => p.Requir-
eRole(UserRole.Manager.ToString()));
});
```

Obrázek 36. Konfigurace autorizace

Jednotlivým endpointům či celým controllerům byl následně přidán atribut Authorize, který zajistil middleware pro kontrolu přihlášeného uživatele a jeho práv pro dané metody.

Na straně klienta byly implementovány metody pro uložení tokenu po přihlášení uživatele. Tento token je po přihlášení uchován v local storage prohlížeče, kdy při každém požadavku pro autorizaci je token odeslán v hlavičce požadavku. Identita přihlášeného uživatele je rovněž udržována na straně klienta v rámci vlastní implementace AuthenticationStateProvideru.

Vlastní AuthenticationStateProvider implementuje metodu GetAuthenticationStateAsync(). Tato metoda vrací stav identity přihlášeného uživatele dle uloženého JWT v local storage.

```
public override async Task<AuthenticationState> GetAuthenticationState-
Async()
{
	string authToken = await _localStorageService.GetItemAsStrin-
gAsync("authToken");
	var identity = new ClaimsIdentity();
	_http.DefaultRequestHeaders.Authorization = null;
	if (!string.IsNullOrEmpty(authToken))
	{
		try
		{
			var claims = ParseClaimsFromJwt(authToken);
			identity = new ClaimsIdentity(claims, "jwt");
			var expiry = identity.Claims.SingleOrDefault(c => c.Type
== ClaimTypes.Expiration)?.Value;
			var expiryDate = DateTime.ParseExact(expiry?.ToString(),
Constants.DateTimeFormat, CultureInfo.InvariantCulture);
			if (expiry != null && DateTime.Now > expiryDate)
			{
				await _localStorageService.Remove-
ItemAsync("authToken");
				identity = new ClaimsIdentity();
			}
			_http.DefaultRequestHeaders.Authorization =
				new AuthenticationHeaderValue("Bearer", authTo-
ken.Replace("\"", ""));
		}
		catch (Exception e)
		{
			await _localStorageService.RemoveItemAsync("authToken");
			identity = new ClaimsIdentity();
		}
	}
	if (identity.Claims.Any())
	{
		var roleClaim = identity.Claims.FirstOrDefault(c => c.Type ==
"role");
		var fixedRoleClaim = new Claim(identity.RoleClaimType,
roleClaim?.Value);
		identity.AddClaim(fixedRoleClaim);
	}
	var user = new ClaimsPrincipal(identity);
	var state = new AuthenticationState(user);
	NotifyAuthenticationStateChanged(Task.FromResult(state));
	return state;
}
```

Obrázek 37. Implementace metody pro správu identity uživatele na straně klienta

7.3.6 Page routing a autorizace komponent

K implementaci přesměrování v jednostránkové aplikaci (SPA) klientský Blazor WebAssembly projekt implementuje soubor App.razor. Tento soubor slouží jako kořenová komponenta celé aplikace a umožňuje řídit které komponenty se zobrazují v určitých situacích – např. v případě, že stránka na dané adrese neexistuje nebo uživatel k ní nemá přístup.

```
<CascadingAuthenticationState>
	<Router AppAssembly="@typeof(App).Assembly">
		<Found Context="routeData">
			<AuthorizeRouteView RouteData="@routeData" Default-
Layout="@typeof(MainLayout)">
				<NotAuthorized>
					<PageTitle>401</PageTitle>
					<_401></_401>
				</NotAuthorized>
			</AuthorizeRouteView>
			<FocusOnNavigate RouteData="@routeData" Selec-
tor="h1" />
		</Found>
		<NotFound>
			<PageTitle>404</PageTitle>
			<LayoutView Layout="@typeof(MainLayout)">
				<_404></_404>
			</LayoutView>
		</NotFound>
	</Router>
</CascadingAuthenticationState>
```

Obrázek 38. Implementace komponenty App.razor

Nastavení autorizace je možné v rámci jednotlivých komponent použitím atributu Authorize:

```
@attribute [Authorize(Policy = Constants.PolicyManagerOnly)]
```

Obrázek 39. Použití atributu Authorize v Razor komponentě

7.3.7 Ukázka generické komponenty

Jako ukázka generické Razor komponenty byla implementována tabulka, kterou je možné použít pro seznamy objektů různých typů. Implementace byla rozdělena na dvě komponenty. V následující komponentě *CustomGridColumn* má komponenta vstupní parametry k zobrazení dat ve sloupcích tabulky:

```
@typeparam TItem

@if (Item == null)
{
	<th>@Label</th>
}
else if (ChildContent == null)
{
	var property = typeof(TItem).GetProperty(Name);
	<td class="text-truncate align-middle">@property.Get-
Value(Item).ToString()</td>
}
else
{
	<td class="text-truncate align-middle">@ChildContent</td>
}

@code {
	[Parameter]
	public string Name { get; set; }

	[Parameter]
	public string Label { get; set; }

	[CascadingParameter]
	public TItem Item { get; set; }

	[Parameter]
	public RenderFragment? ChildContent { get; set; }
}
```

Obrázek 40. Implementace komponenty CustomGridColumn

Komponenta zobrazuje obsah dle vstupních parametrů. V případě, že není poskytnut vstupní parametr *Item* s objektem generického typu, je jako hlavička tabulky nastavena hodnota parametru Label. Parametr *ChildContent* je typu *RenderFragment*, který umožňuje vykreslit část HTML mezi HTML tagy této komponenty.

Druhou částí tabulky je komponenta *CustomGrid*, která přijímá jako vstupní parametry seznam dat a *RenderFragmenty* jednotlivých sloupců:

```
@attribute [CascadingTypeParameter(nameof(TItem))]
@typeparam TItem

@if (Items == null)
{
	<LoadingSpinner></LoadingSpinner>
}
else
{
	<div class="table-responsive">
		<table class="table table-hover">
			<thead>
				<tr>
					@Columns(default(TItem))
				</tr>
			</thead>
			<tbody>
				@foreach (var item in Items)
				{
					<CascadingValue Value="item">
						<tr>@Columns(item)</tr>
					</CascadingValue>
				}
			</tbody>
		</table>
	</div>
}

@code {
	[Parameter]
	public IList<TItem> Items { get; set; }

	[Parameter]
	public RenderFragment<TItem>? Columns { get; set; }
}
```

Obrázek 41. Implementace komponenty CustomGrid

V této komponentě je možné vidět využití komponenty *CascadingValue* a atributu *CascadingTypeParameter*. Jedná se o tzv. „kaskádové“ hodnoty a parametry, které umožňují řídit tok dat z nadřazených Razor komponent do libovolného počtu sestupných komponent. Použitím typového parametru *TItem* je umožněno vytvořit tabulku pro objekt libovolného datového typu.

Příklad použití těchto komponent je možné pozorovat např. v komponentě stránky pro správu kategorií produktů:

```
<CustomGrid Items="CategoryList" TItem="ProductCategoryDTO">
	<Columns>
		<CustomGridColumn Name="Name" Label="Název"></CustomGridColumn>
		<CustomGridColumn Name="Url" Label="Url">
			<span>@(context.Url)</span>
		</CustomGridColumn>
		<CustomGridColumn Name="Icon" Label="Ikona">
			<span><i class="@(context.Icon)"></i></span>
		</CustomGridColumn>
		<CustomGridColumn>
			<div class="d-flex justify-content-end">
				<button class="btn btn-outline-primary btn-sm me-
1" @onclick="(e) => NavigateToCategoryCreateEdit(context.Id)"><i class="bi-
pencil-fill"></i></button>
				<button class="btn btn-outline-danger btn-sm" @on-
click="(e) => SelectCategoryToDelete(context.Id)" data-bs-toggle="modal"
data-bs-target="#confirmModal"><i class=" bi-trash-fill"></i></button>
			</div>
		</CustomGridColumn>
	</Columns>
</CustomGrid>
```

Obrázek 42. Ukázka použití komponent CustomGrid a CustomGridColumn

V tomto případě je jako typový parametr *TItem* uveden typ třídy *ProductCategoryDTO*. Pro úpravu stylování mají některé sloupce, jako např. sloupec k zobrazení ikony kategorie, mezi tagy komponenty *CustomGridColumn* specifikovaný obsah pro zobrazení ikony (využití *RenderFragment* parametru).

7.3.8 Demonstrace JavaScript interoperability

Jelikož platební brána PayPal vyžaduje JavaScript k zadání vstupu, odeslání požadavku a vrácení odpovědi o platbě, bylo potřeba použít JavaScript interoperabilitu ke zpracování odpovědi a vytvoření objednávky.

V komponentě stránky pro platbu bylo pomocí dependency injection nejprve přidáno rozhraní *IJSRuntime*, které definuje metody pro JavaScript interoperabilitu.

Následně byl do C# kódu komponenty přidán field *dotNetHelper*, který je nastaven na referenci třídy komponenty v metodě, která je zavolána po prvním vykreslení komponenty:

```
private DotNetObjectReference<Checkout>? dotNetHelper;

protected override async Task OnAfterRenderAsync(bool firstRender)
{
	if (firstRender)
	{
		dotNetHelper = DotNetObjectReference.Create(this);
		await Js.InvokeVoidAsync("CheckoutHelper.setDotNetHelper",
dotNetHelper);
		await Js.InvokeVoidAsync("initPayPalButton");
	}
}
```

Obrázek 43. Nastavení JavaScript interoperability pro komponentu Checkout

Tato metoda poté volá pomocí rozhraní *IJSRuntime* JavaScript funkce k získání reference třídy komponenty a k předvyplnění dat pro rozhraní PayPal, které jsou implementovány v separátním JavaScript souboru order.js. Tento soubor implementuje třídu *CheckoutHelper*, která implementuje funkce k získání reference komponenty a volání metody *CreateOrder*.

```
class CheckoutHelper {
    static dotNetHelper;

    static setDotNetHelper(value) {
        CheckoutHelper.dotNetHelper = value;
    }

    static async createOrder(response) {
        await CheckoutHelper.dotNetHelper.invokeMethodAsync('CreateOr-
der', response);
    }
}
window.CheckoutHelper = CheckoutHelper;
```

Obrázek 44. Implementace JavaScript třídy CheckoutHelper

Metoda *CreateOrder* je implementovaná v samotné Razor komponentě. Aby ji bylo možné volat pomocí JavaScript interoperability, musí jí dekorovat atribut *JSInvokable*. Metoda přijímá data ze služby PayPal a umožňuje vytvoření objednávky.

```
[JSInvokable]
public async Task CreateOrder(string value)
{
	var result = JsonConvert.DeserializeObject<dynamic>(value)!;
	if (result != null)
	{
		var createdOrderResponse = await OrderService.CreateOr-
der(LoadedCart, result, userInfo);
		if (createdOrderResponse.Success == true)
		{
			await CartService.ClearCart();
			NavigationManager.NavigateTo("summary");
		}
		else
		{
			NavigationManager.NavigateTo("cart");
		}
	}
	else
	{
		NavigationManager.NavigateTo("cart");
	}
}
```

Obrázek 45. Implementace metody CreateOrder

8 DEMONSTRACE UKÁZKOVÉ APLIKACE

V této kapitole je demonstrováno rozhraní a funkcionalita aplikace ze strany uživatele. Kapitola je rozdělena do podkapitol, ve kterých jsou popsány a vyobrazeny klíčové prvky hlavních funkcí aplikace.

8.1 Navigační lišta

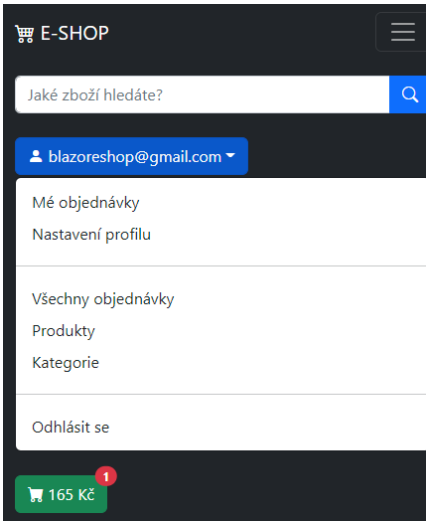
Aplikace má na každé stránce komponentu navigační lišty, ve které je zobrazen název aplikace, pole pro vyhledávání produktů, tlačítka pro registraci, přihlášení a košík.

Obrázek 46. Navigační lišta nepřihlášeného uživatele

Po přihlášení do aplikace jsou tlačítka pro registraci a přihlášení zaměněny tlačítkem pro přístup k uživatelskému menu.

Obrázek 47. Navigační lišta přihlášeného uživatele

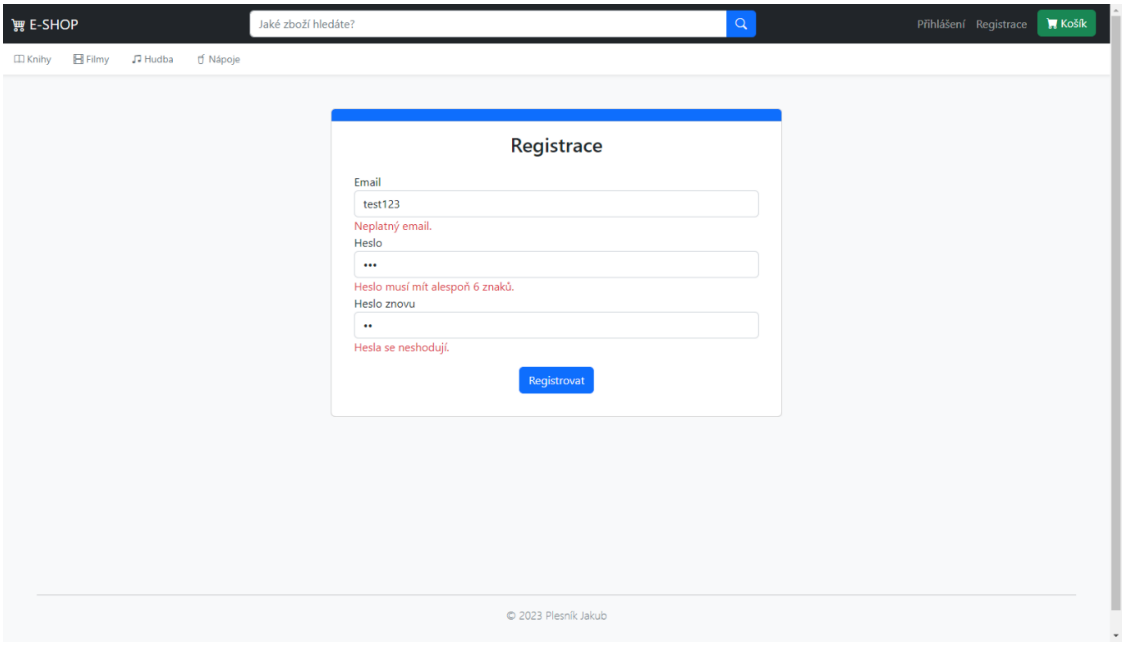
Navigační lišta má také responzivní stylování. Na zařízeních s užší šířkou obrazovky je lišta prezentována dle obrázku níže. Na obrázku je také vyobrazeno rozbalené uživatelské menu uživatele s rolí manažera.



Obrázek 48. Navigační lišta nepřihlášeného uživatele

8.2 Registrace a přihlášení

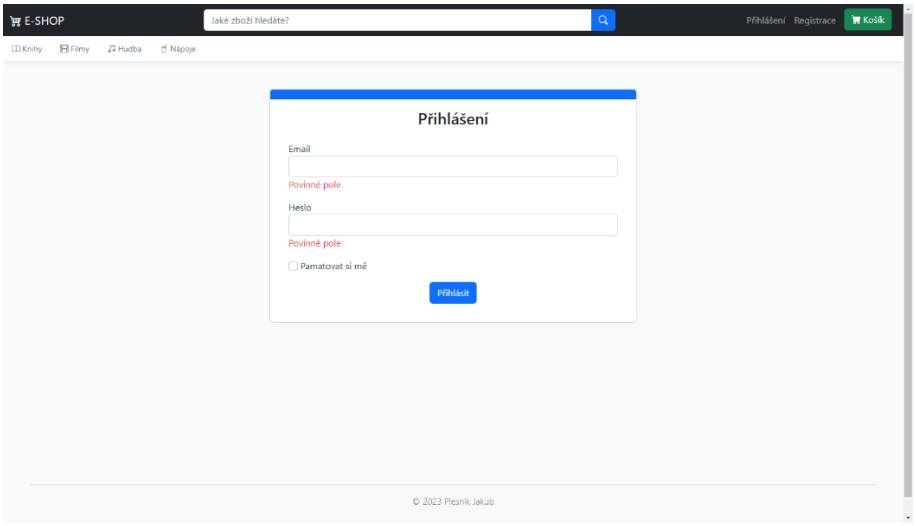
Na registrační stránce je nepřihlášenému uživateli umožněno založit nový uživatelský účet. Registrační formulář obsahuje pole pro zadání e-mailu, hesla a opakovaného hesla. Všechny pole jsou povinné a jsou validovány dle nastavených validačních atributů hned při vyplnění. Při potvrzení registrace jsou provedeny další validace k potvrzení stejnosti hesla a duplicity e-mailu.



Obrázek 49. Stránka pro registraci

Po úspěšné registraci je uživatel přesměrován na přihlašovací stránku a je mu odeslán potvrzovací e-mail.

Stránka pro přihlášení uživatele obsahuje formulář v podobném stylu jako registrační stránka. Zde je ovšem potřeba zadat pouze pole pro e-mail a heslo. Pro zjednodušení opakovaného přihlášení je uživatelskému prohlížeči umožněno pamatovat si přihlašovací e-mail.

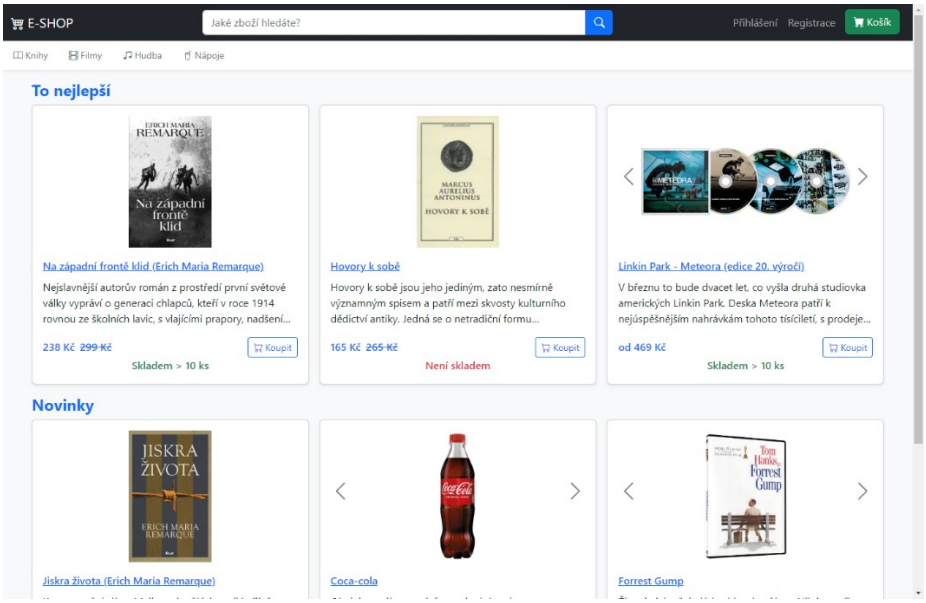


Obrázek 50. Stránka pro přihlášení

8.3 Hlavní stránka s produkty

Po načtení aplikace je uživatel přivítán hlavní domovskou stránkou. Tato stránka zobrazuje manažerem zviditelněné produkty a ostatní produkty seřazené od nejnovějších.

Tato stránka také slouží k zobrazování filtrovaných produktů pomocí vyhledávacího pole nebo výběru kategorie v navigační liště.



Obrázek 51. Hlavní stránka s produkty

Po úspěšném přihlášení uživatele je uživatel přesměrován na poslední navštívenou stránku před přihlášením.

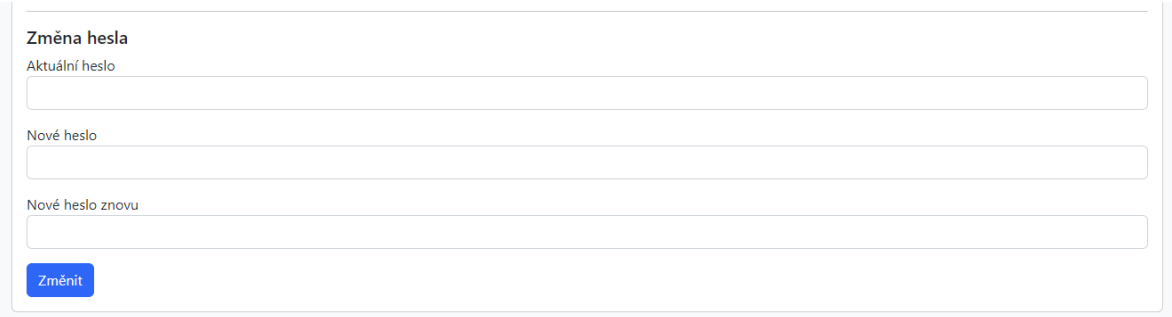
8.4 Nastavení profilu uživatele

Tato stránka je přístupná přihlášenému uživateli nezávisle na jeho roli. Stránka je rozdělena na sekci pro správu doručovací adresy a sekci umožňující změnu hesla uživatelského účtu.

Přihlášenému uživateli je umožněno uložit doručovací adresu. Tato adresa je při vytváření objednávky předvyplněna v rámci kroku platby objednávky.

Obrázek 52. Sekce pro správu doručovací adresy

Uživateli je také umožněno změnit své heslo. K úspěšné změně hesla musí přihlášený uživatel zadat své dosavadní heslo a opakovaně uvést nově používané.



Obrázek 53. Sekce pro změnu hesla

8.5 Výběr produktu

Hlavní stránka s produkty (kapitola 8.3) zobrazuje komponenty dostupných produktů, které je možné dále filtrovat pomocí vyhledávacího pole nebo výběru kategorie v navigační liště. V případě, že daný produkt má více variant, systém vždy zobrazuje nejlevnější produkt.

Obrázek 54. Komponenta karty produktu

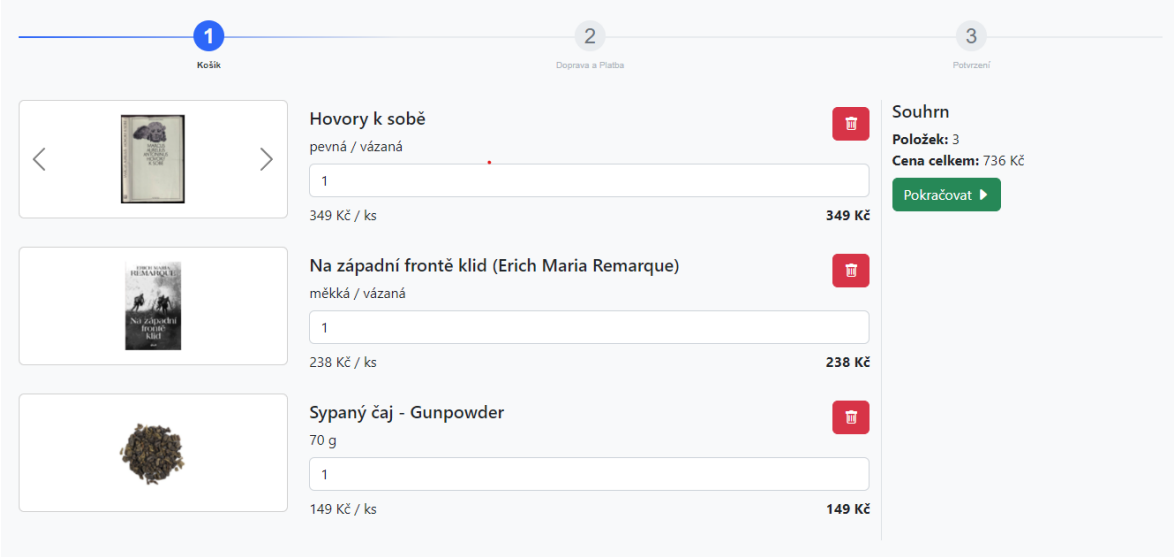
Komponenta umožňuje zobrazovat obrázky všech variant pomocí šipek. Počet produktů na skladě je rovněž spočítán součtem zásob všech variant. Zásoba produktů je stylována dle výšky hodnoty. Kliknutím na obrázek, název produktu nebo tlačítko „Koupit“ je uživatel přesměrován na stránku produktu.

Obrázek 55. Stránka produktu

Stránka produktu zobrazuje veškeré detaily a umožňuje vybírat mezi jednotlivými variantami. Po výběru varianty a počtu přidaných produktů je stisknutím tlačítka „Do košíku“ daná varianta produktu přidána do košíku. Výběr počtu produktů je omezen dle počtu kusů skladem.

8.6 Vytvoření objednávky

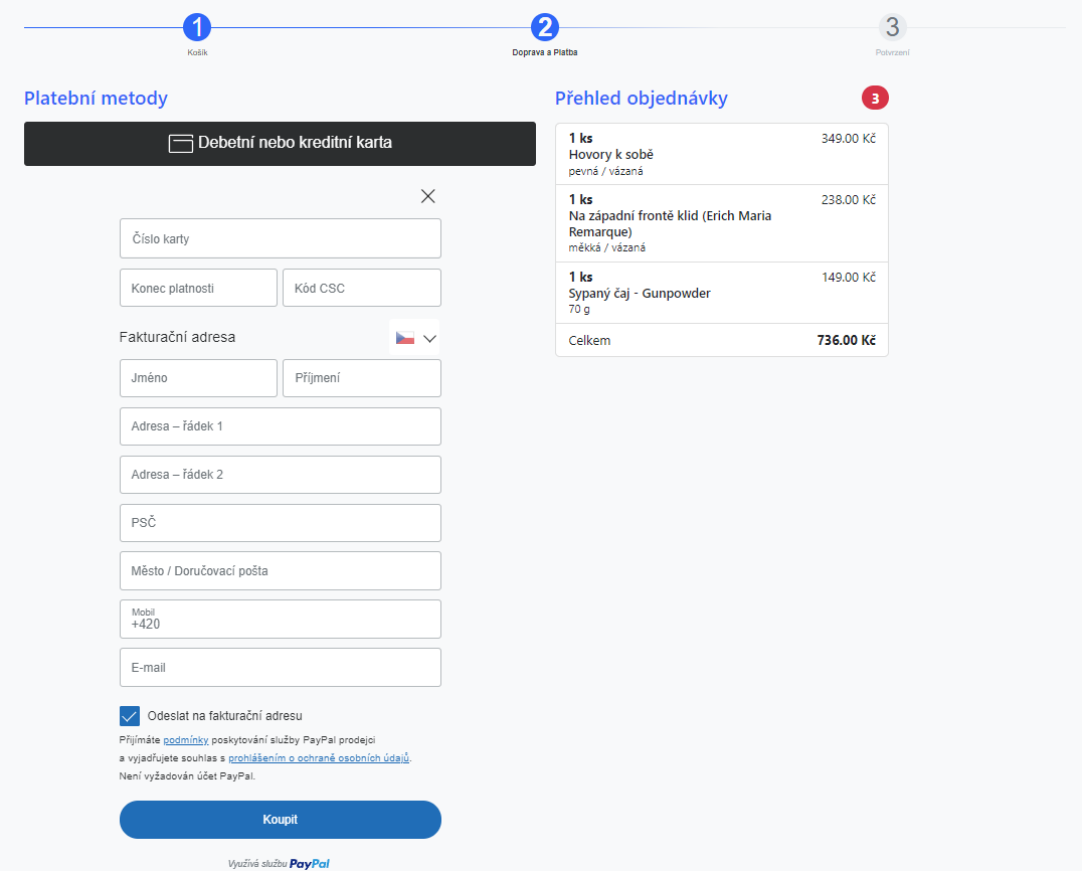
Celkový proces vytvoření objednávky je rozdělen na tři kroky, jejichž stav je možné sledovat v horní části každého kroku. Stisknutím tlačítka košíku v navigační liště je uživatel přesměrován na stránku s vybranými produkty.



Obrázek 56. Stránka košíku

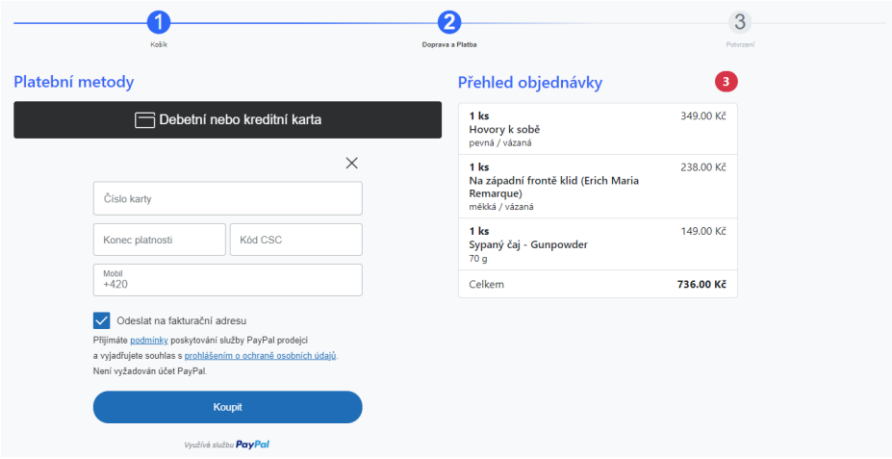
Stránka košíku obsahuje seznam vybraných produktů, které je možné odstranit nebo přidat jejich počet. Po shrnutí objednávky a stisknutí tlačítka pokračovat je uživatel přesměrován na druhý krok vytvoření objednávky.

Na stránce dopravy a platby si uživatel může vybrat mezi platební metodou PayPal a „Debetní nebo kreditní karta“. V případě, že se jedná o nepřihlášeného uživatele nebo přihlášeného uživatele s neuloženou doručovací adresou je nutné v platební bráně vyplnit veškerá pole.



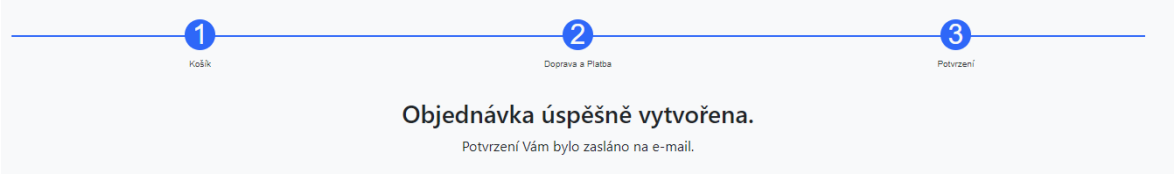
Obrázek 57. Stránka dopravy a platby nepřihlášeného uživatele nebo přihlášeného uživatele s neuloženou adresou

V případě, že na tuto stránku přistupuje přihlášený uživatel s uloženou doručovací adresou, jsou údaje v platební bráně týkající se adresy předvyplněny a uživatel je požádán pouze o vyplnění platebních údajů.



Obrázek 58. Stránka dopravy a platby přihlášeného uživatele s uloženou adresou

Po úspěšném provedení transakce pomocí platební brány je uživatel přesměrován na poslední krok. Na tomto kroku je uživatel informován o úspěšném vytvoření objednávky a zaslání potvrzení na jeho e-mail.

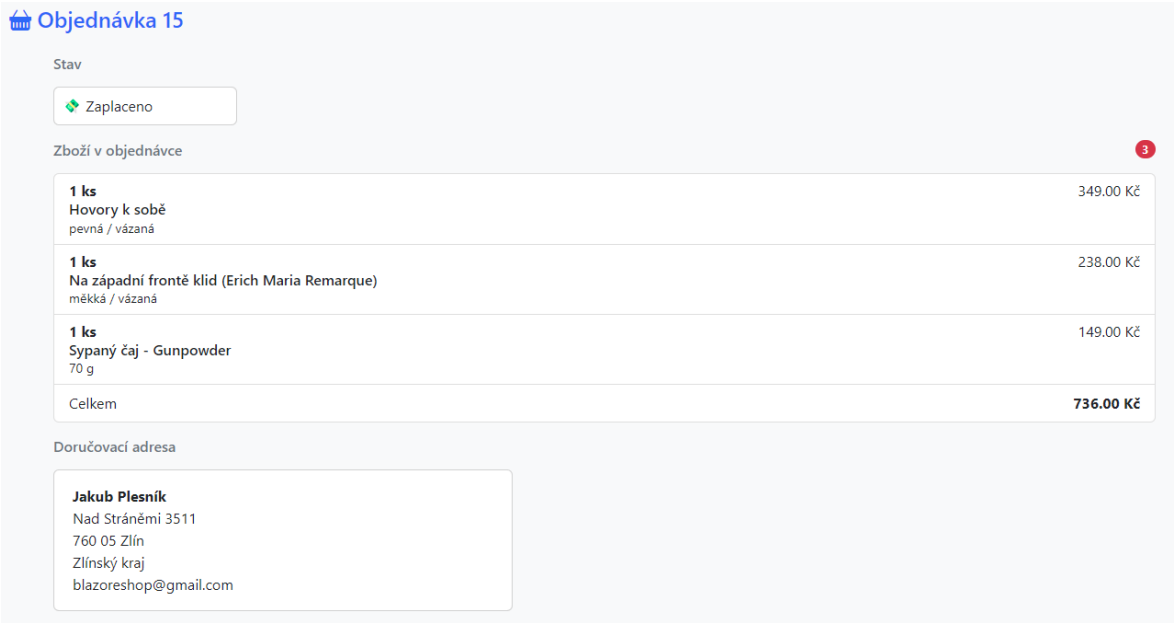


Obrázek 59. Informace o úspěšném vytvoření objednávky

Přihlášenému uživateli je umožněno zhlédnout vytvořené objednávky v sekci „Mé objednávky“ v uživatelském menu.

Obrázek 60. Objednávky přihlášeného uživatele

Kliknutím na tlačítko lupy u některé z objednávek v seznamu je uživatel přesměrován na stránku s detaily dané objednávky. Tato stránka obsahuje stav, zakoupené zboží a doručovací adresu objednávky.

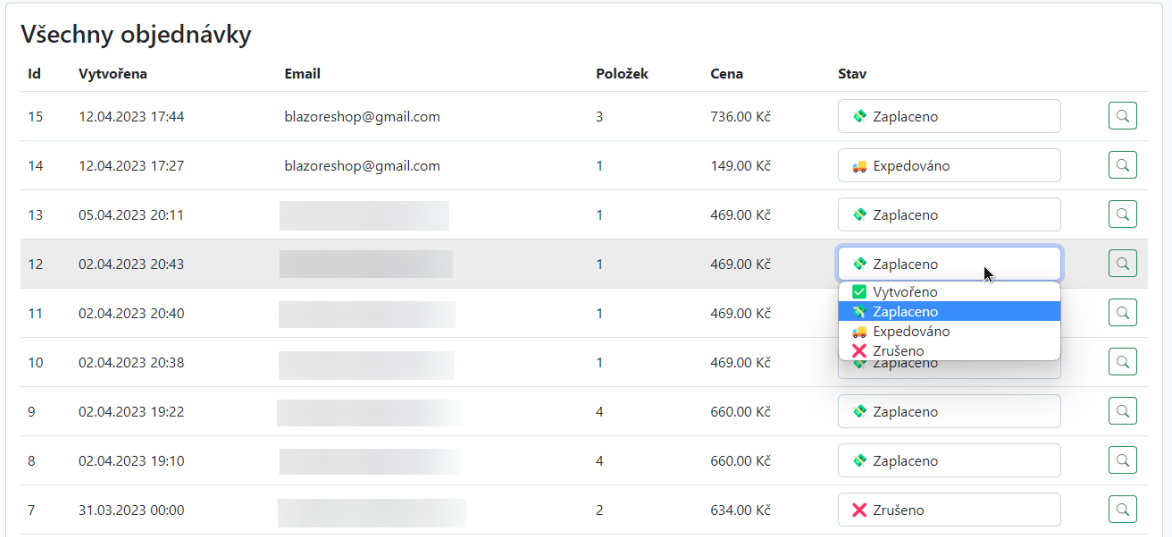


Obrázek 61. Detail objednávky

8.7 Administrace objednávek, produktů a kategorií

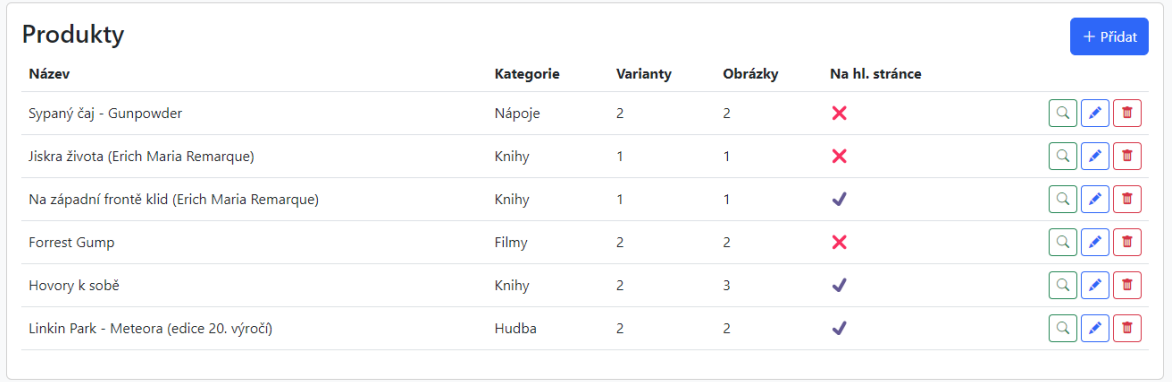
Přihlášený uživatel s rolí manažera má přístup ke stránkám sloužícím k administrativě objednávek, produktů a jejich kategorií. Odkazy na tyto stránky mohou být nalezeny v uživatelském menu.

Na stránce „Všechny objednávky“ je možné změnit stav objednávek všech uživatelů nebo přejít na detail dané objednávky.



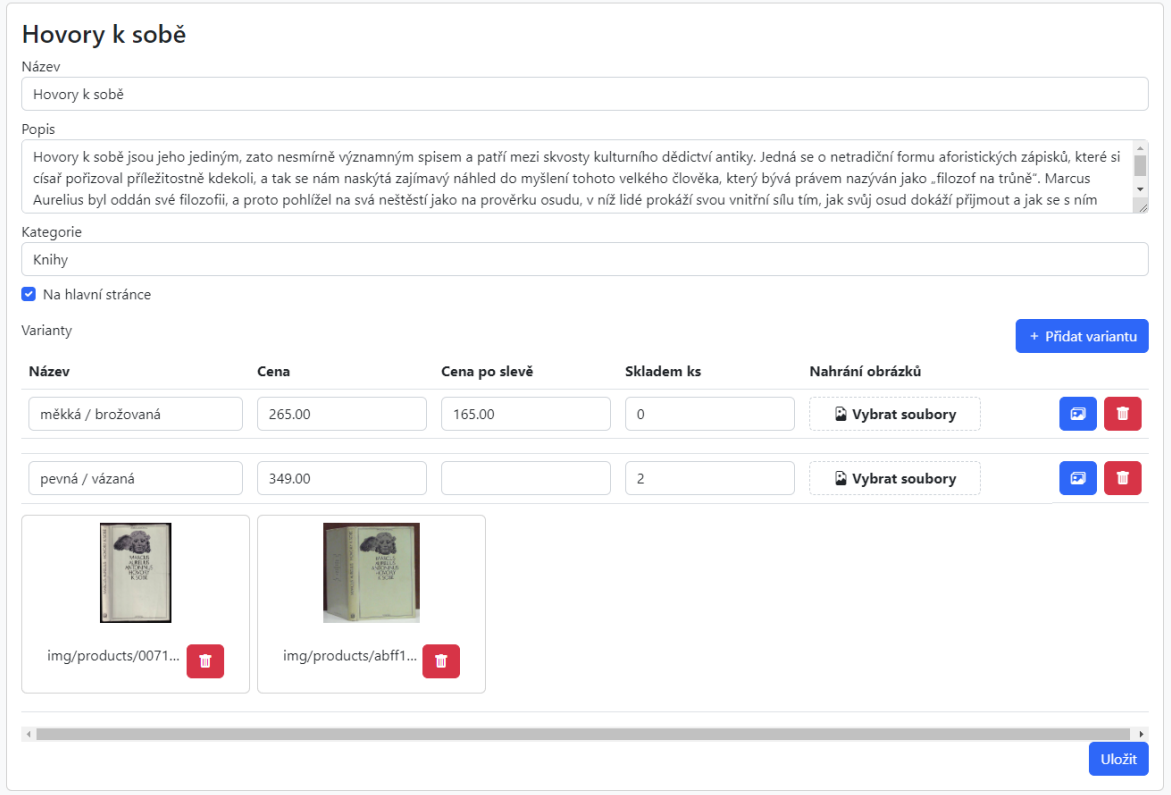
Obrázek 62. Stránka s objednávkami všech uživatelů

Stránka „Produkty“ obsahuje seznam všech produktů v databázi. U každého produktu je možné přejít na detail daného produktu, stránku editace nebo je možné přímo produkt odstranit. Před samotným odstraněním produktu je manažerovi zobrazeno potvrzovací okno.



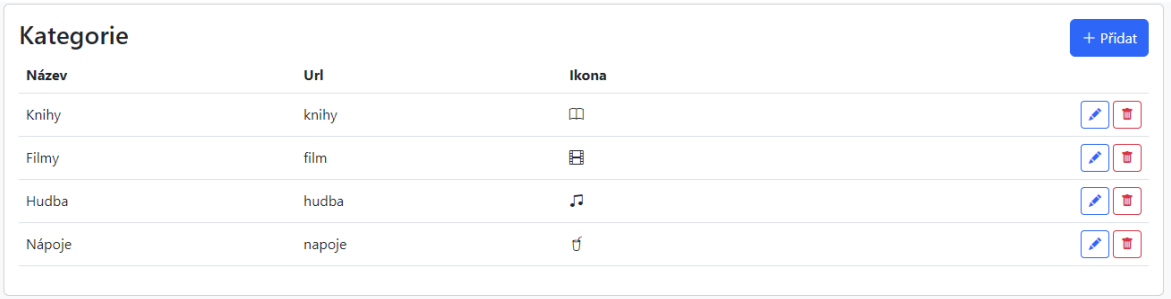
Obrázek 63. Stránka správy produktů

Stránka editace produktu umožňuje upravovat daný produkt. V rámci routingu aplikace se jedná o stejnou stránku jako u vytvoření produktu - při vytvoření produktu se pouze mění titulek stránky na „Nový produkt“ a tlačítko „Uložit“ se mění na „Vytvořit“. Na této stránce je možné upravovat parametry daného produktu a dynamicky přidávat či odstraňovat jeho varianty včetně jejich obrázků. Sloupec „Nahrání obrázků“ obsahuje tlačítko pro nahrávání souborů s obrázky, které je možné zobrazit modrým tlačítkem s ikonou obrázku a následně odstranit.



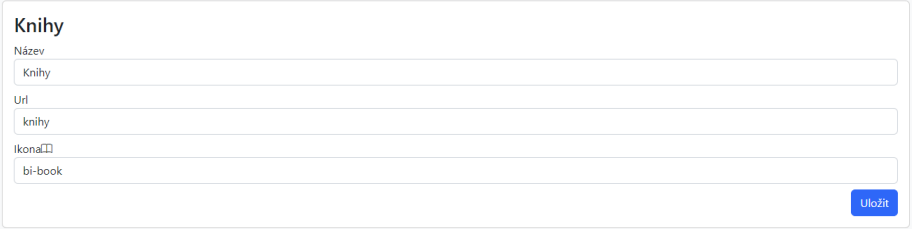
Obrázek 64. Stránka editace nebo přidání produktu

Na stránce „Kategorie“ je zobrazen seznam všech produktových kategorií. Je možné přidat novou kategorii a upravit nebo odstranit již existující kategorie.



Obrázek 65. Stránka správy kategorií

Kliknutím na tlačítko s ikonou tužky je manažer přesměrován na stránku editace dané kategorie. Parametr ikony kategorie umožňuje uvést kód ikony dle knihovny Bootstrap Icons.



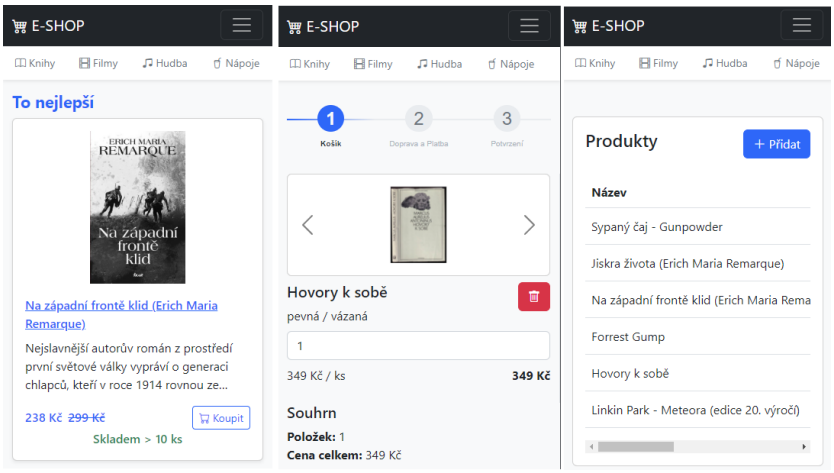
Obrázek 66. Stránka editace nebo přidání kategorie produktu

8.8 Speciální stránky a responzivní stylování

Speciální stránky, které slouží k informování uživatele o chybějících stránkách nebo omezeném přístupu k obsahu, jsou běžné komponenty webových aplikací. Stránka 404 se zobrazí, když uživatel pokusí navštívit neexistující stránku, zatímco stránka 401 se objeví, když uživatel nemá oprávnění k zobrazení požadované stránky. Tyto stránky obsahují chybové hlášky a odkazy zpět na hlavní stránku.

Obrázek 67. Speciální stránky

Na obrázku níže je poukázáno na responzivní chování uživatelského rozhraní. Na menších zařízeních je zachována veškerá funkcionalita hotové aplikace.



Obrázek 68. Ukázka responzivního designu aplikace

ZÁVĚR

Výstupem bakalářské práce je ukázková aplikace z oblasti elektronického obchodnictví vyvinutá pomocí ASP.NET Core Blazor WebAssembly a relační databáze SQL Server. Téma elektronického obchodnictví autor vybral především z důvodu rozmanitosti řešených procesů v těchto typech aplikace, což umožňovalo adekvátní představení možností webového frameworku Blazor a vhodného modelování dat v relační databázi.

V teoretické části byla popsána platforma .NET a její různé verze. Dále byla popsána platforma pro vývoj webových aplikací ASP.NET Core a možnosti zabezpečení aplikací vyvíjených na této platformě. Následně byl popsán webový framework Blazor a jeho modely hostování. Primárně byl popsán model hostování Blazor WebAssembly, který byl použit v praktické části k vývoji ukázkové aplikace. Ke konci teoretické části byly popsány relační databáze včetně možností udržení integrity dat, jazyk SQL a použití frameworku Bootstrap k vytváření responzivního uživatelského rozhraní.

V praktické části byl vytvořen návrh ukázkové aplikace. Autor nejprve provedl analýzu požadavků, dle kterých následně navrhl případy užití a popsal jejich scénáře. Dále provedl návrh databáze postupným snižováním abstrakce navrhovaných modelů. V další kapitole byly popsány klíčové části aplikace ukázkami kódu – je zde popsáno spojení s relační databází, zabezpečení aplikace, autentizace uživatele, autorizace požadavků a ze strany Blazor klienta ukázka generické komponenty ve formě tabulky a JavaScript interoperability ke komunikaci se službou platební brány PayPal. Poslední kapitola praktické části demonstruje chování aplikace ze strany uživatele – především hlavní funkce a responzivitu uživatelského rozhraní.

Vývoj aplikace v Blazor WebAssembly umožňoval pohodlný vývoj debugováním klientského kódu a napovídáním našeptávačem IntelliSense v robustním IDE Visual Studio. Sdílením kódu v serverové a klientské části autor nemusel navíc udržovat modely ve více projektech. Vývojem aplikace na téma elektronického obchodnictví autor přišel na hlavní výhody a nevýhody vývoje pomocí technologie Blazor WebAssembly.

Klientská část pro běžného uživatele, která slouží k výběru produktů a vytváření objednávek, je pro vystavení na veřejný web nevhodná, a to především z důvodu neumožnění indexace pro internetové vyhledávače a vyšších požadavků pro klientské prohlížeče. Délka prvotního načtení aplikace by mohla odradit potenciální zákazníky a snížit tak tržby obchodu používající webovou aplikaci.

Technologie Blazor ukázala své výhody především v rámci administrativní části aplikace. Pohodlný vývoj dynamických komponent a krátká odezva u interakce s uživatelským rozhraním autorovi naznačila především vhodnost technologie pro vývoj interních firemních aplikací nebo komplexních online kalkulátorů. Pro týmy s vývojáři s hlubokými zkušenostmi s vývojem .NET aplikací má Blazor velký potenciál a v budoucnu by mohl konkurovat např. technologii Angular a umožnit tak vývoj webových aplikací čistě v prostředí .NET.

SEZNAM POUŽITÉ LITERATURY

[1] TIOBE Index for April 2023. *TIOBE* [online]. [cit. 2023-04-24]. Dostupné z: https://www.tiobe.com/tiobe-index/

[2] What Is Microsoft .NET Core? A Complete Guide. *Brainspire* [online]. [cit. 2023-04-25]. Dostupné z: https://www.brainspire.com/blog/what-is-microsoft-dot-net-core-a-complete-guide

[3] What is .NET? Introduction and overview. Microsoft learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction

[4] How Open Source changed .NET forever. Medium [online]. [cit. 2023-04-25]. Dostupné z: https://abstarreveld.medium.com/how-open-source-changed-net-forever-f0e730dbc6f5https://abstarreveld.medium.com/how-open-source-changed-net-forever-f0e730dbc6f5

[5] .NET Core vs .NET Framework. InterviewBit [online]. [cit. 2023-04-25]. Dostupné z: https://www.interviewbit.com/blog/net-core-vs-net-framework/

[6] .NET Standard. Microsoft learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-2-1

[7] The future of .NET Standard. .NET Blog [online]. [cit. 2023-04-25]. Dostupné z: https://devblogs.microsoft.com/dotnet/the-future-of-net-standard/

[8] J. PRICE, Mark. *C# 10 and .NET 6 – Modern Cross-Platform Development*. 6. Packt Publishing, 2021. ISBN 9781801077361.

[9] Overview of ASP.NET Core. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0

[10] Razor syntax reference for ASP.NET Core. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-7.0

[11] Introduction to ASP.NET Web Programming Using the Razor Syntax (C#). Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c

[12] SMITH, Jon. Entity Framework Core in Action. 2. Manning Publications, 2021. ISBN 9781617298363.https://www.learnentityframeworkcore.com/

[13] Entity Framework Core. Entity Framework Tutorials [online]. [cit. 2023-04-25]. Dostupné z: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[14] Entity Framework Code First vs Database First vs Model First Approach. Pro Code Guide [online]. [cit. 2023-04-25]. Dostupné z: https://procodeguide.com/csharp/entity-framework-code-first/https://procodeguide.com/csharp/entity-framework-code-first/

[15] Scaffolding (Reverse Engineering). Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding/?tabs=dotnet-core-clihttps://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding/?tabs=dotnet-core-cli

[16] Fluent API - Configuring and Mapping Properties and Types. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/fluent/types-and-properties

[17] Best practices to secure ASP.NET Core web applications. FiduciaSoft [online]. [cit. 2023-04-25]. Dostupné z: https://fiduciasoft.com/blog/best-practices-to-secure-asp-net-core-web-applications/

[18] Implementing JWT Authentication in ASP.NET Core 5. CODE Online [online]. [cit. 2023-04-25]. Dostupné z: https://www.codemag.com/Article/2105051/Implementing-JWT-Authentication-in-ASP.NET-Core-5

[19] Introduction to JSON Web Tokens. JSON Web Tokens [online]. [cit. 2023-04-25]. Dostupné z: https://jwt.io/introduction

[20] RS256 vs HS256: What's The Difference?. Auth0 [online]. [cit. 2023-04-25]. Dostupné z: https://auth0.com/blog/rs256-vs-hs256-whats-the-difference/

[21] Introduction to Identity on ASP.NET Core. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio

[22] What is Blazor?. Blazor University [online]. [cit. 2023-04-25]. Dostupné z: https://blazor-university.com/overview/what-is-blazor/

[23] SAINTY, Chris. *Blazor in Action*. Manning Publications, 2022. ISBN 9781617298646.

[24] ASP.NET Core Blazor. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0

[25] ASP.NET Core Razor components. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/components/?view=aspnetcore-7.0

[26] ASP.NET Core Blazor hosting models. Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0

[27] WebAssembly. MDN Web Docks [online]. [cit. 2023-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly

[28] ASP.NET Core Blazor JavaScript interoperability (JS interop). Microsoft Learn [online]. [cit. 2023-04-25]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/javascript-interoperability/?view=aspnetcore-7.0

[29] What is a relational database?. Cloud Computing Services | Microsoft Azure [online]. [cit. 2023-04-25]. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-relational-database

[30] What is data modeling?. IBM [online]. [cit. 2023-04-25]. Dostupné z: https://www.ibm.com/topics/data-modeling

[31] What Is Cardinality in Data Modeling? The Theory and Practice of Database Cardinality. Vertabelo Database Modeler [online]. [cit. 2023-04-25]. Dostupné z: https://vertabelo.com/blog/cardinality-in-data-modeling/

[32] A Relational Database Overview. Oracle Help Center [online]. [cit. 2023-04-25]. Dostupné z: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

[33] What is data integrity?. LinkedIn [online]. [cit. 2023-04-25]. Dostupné z: https://www.linkedin.com/pulse/what-data-integrity-ankur-gupta/

[34] What is Data Integrity in a Database? Why Do You Need It?. Astera Software [online]. [cit. 2023-04-25]. Dostupné z: https://www.astera.com/type/blog/data-integrity-in-a-database/

[35] BEN-GAN, Itzik, Louis DAVIDSON a Stacia VARGA. *MCSA SQL Server 2016 Database Development Exam Ref 2-Pack: Exam Refs 70-761 And 70-762*. Boston, United States: Microsoft Press, U.S., 2017. ISBN 9781509304332.

[36] What is SQL Server. SQL Server Tutorial [online]. [cit. 2023-04-25]. Dostupné z: https://www.sqlservertutorial.net/getting-started/what-is-sql-server/

[37] Introduction to Transact-SQL. TSQL Tutorial [online]. [cit. 2023-04-25]. Dostupné z: https://www.tsql.info/https://www.tsql.info/

[38] JAKOBUS, Benjamin a Jason MARAH. Mastering Bootstrap 4: Master the latest version of Bootstrap 4 to build highly customized responsive web apps. 2. 2018: Packt Publishing. ISBN 9781788834902.

[39] Containers. Bootstrap · The most popular HTML, CSS, and JS library in the world. [online]. [cit. 2023-04-25]. Dostupné z: https://getbootstrap.com/docs/5.2/layout/containers/

[40] 10 Best Practices to Secure ASP.NET Core MVC Web Applications. Syncfusion [online]. [cit. 2023-04-25]. Dostupné z: https://www.syncfusion.com/blogs/post/10-practices-secure-asp-net-core-mvc-app.aspx

[41] Cross Site Request Forgery (CSRF). OWASP Foundation [online]. [cit. 2023-04-25]. Dostupné z: https://owasp.org/www-community/attacks/csrf

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SQL	Structured Query Language
IoT	Internet of Things
API	Application Programming Interface
IDE	Integrated Development Environment
RCL	Razor Class Library
ASP	Active Server Pages
MVC	Model-View-Controller
SPA	Single Page Application
HTTP	Hypertext Transfer Protocol
SOAP	Simple Object Access Protocol
ORM	Object Relational Mapping
URL	Uniform Resource Locator
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
JWT	JSON Web Token
RFC	Request for Comments
JSON	JavaScript Object Notation
HMAC	Hash-based Message Authentication Code
SHA	Secure Hash Algorithm
RSA	Rivest–Shamir–Adleman
WASM	WebAssembly
PWA	Progressive Web Application
HTML	HyperText Markup Language
CSS	Cascading Style Sheets

DOM	Document Object Model
JS	JavaScript
AOT	Ahead-of-Time
MAUI	Multi-platform App UI
WPF	Windows Presentation Foundation
T-SQL	Transact-Structured Query Language
ANSI	American National Standards Institute
UI	User Interface

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Příloha P I: CD s kompletní dokumentací práce, zdrojovým kódem a Enterprise Architect projektem