Zhodnocení technologií .NET a Docker pro tvorbu webových aplikací malých a středních firem

Jaroslav Potočiar

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jaroslav Potočiar**
Osobní číslo:	**A20373**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Zhodnocení technologií .NET a Docker pro tvorbu webových aplikací malých a středních firem**
Téma práce anglicky:	**Evaluation of .NET and Docker Technologies for the Development of Web Applications for Small and Medium-Sized Enterprises**

Zásady pro vypracování

1. Popište současný stav technologií pro tvorbu webových aplikací.
2. Zaměřte se na technologie .NET a Docker.
3. Navrhněte vybranou webovou aplikaci pomocí technologie .NET.
4. Pro správu a přenositelnost aplikace použijte technologii Docker.
5. Zhodnoťte dosažené výsledky a možnosti dalšího rozvoje aplikace.

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

Seznam doporučené literatury:

1. HALL, Gary McLean. Adaptive Code via C#. [Online]. Redmond: Microsoft Press, 2014. ISBN 978-0-7356-8448-6.
2. POULTON, Nigel. Docker Deep Dive: 2023 Edition. UK: Nielson Book Services, 2023. ISBN 978-1916585256
3. DENNIS, Andrew K.; SCHWARTZ, Michael; BULLINGTON-MCGUIRE, Richard. Docker for Developers. UK: Packt Publishing Ltd. 2020. ISBN 9781789539486.
4. PRICE, Mark J. C# 11 and .NET 7. UK: Packt Publishing Ltd, 2022. ISBN 978-1803237800.
5. FREEMAN, Adam. Pro ASP.NET Core 7. South Carolina: Manning, 2023. ISBN 978-1633437821.

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

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

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

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

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

Ve Zlíně dne 5. ledna 2024

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 …….

podpis studenta

ABSTRAKT

Tato práce je zaměřena na tvorbu webové aplikace pomocí technologie Docker a .NET. Aplikace bude vedena jakožto eshop a bude naprogramována ve vývojovém prostředí Visual Studio 2022, jazykem C#. Uživatelské rozhraní aplikace bude vytvořeno kombinací technologii Vue.js, TypeScript, Bootstrap. Nejprve jsou představeny jednotlivé technologie, následně popis programovacích jazyku, které jsou použity pro uskutečnění webové aplikace. Výsledkem práce je eshop se zakladními funkcemi umožňující uživatelům zakoupení nabízených produktů nebo správu jednotlivých produktů. Popis postupu k vytvoření této webové aplikace je uveden v praktické části.

Klíčová slova: .NET, C#, webová aplikace, eshop, Docker, Visual Studio 2022, Vue.js

ABSTRACT

This thesis focuses on creating a web application using Docker and .NET technology. The application will be run as an e-shop and will be programmed in Visual Studio 2022 development environment, using C# language. The user interface of the application will be created using a combination of Vue.js, TypeScript, Bootstrap technologies. First, the individual technologies are introduced, followed by a description of the programming languages that are used to realize the web application. The result of the work is an e-shop with basic functions allowing users to purchase the offered products or manage individual products. A description of the procedure to create this web application is given in the practical part.

Keywords: .NET, C#, web application, eshop, Docker, Visual Studio 2022, Vue.js

Poděkování:

Tímto bych rád poděkoval vedoucím me bakalářské práce panu Ing. et Ing. Erik Král, Ph.D. Za jeho odbornou pomoc při začátcích této práce.

Velký dík patří panu doc. Ing. Radek Šilhavý, Ph.D. za jeho odborné rady, vedení a ochotu, která byla potřebná k vypracování a dokončení této práce.

V poslední řadě bych chtěl poděkovat firmě Edhouse a jejim zaměstnancům za možnost u nich pracovat a tím si vydělat na možnost studia, také za jejich odborné rady a praxi, která mi napomohla v této práci.

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

OBSAH

OBSAH **1-14-7**

ÚVOD **8**

I. TEORETICKÁ ČÁST **9**

1 TECHNOLOGIE TVORBY WEBOVÝCH APLIKACÍ **10**

1.1 TRENDY VE WEBOVÉM VÝVOJI 10

1.1.1 Progressive Web Apps (PWAs) *10*

1.1.2 Single Page Applications (SPAs) *11*

1.2 STANDARDY V OBLASTI WEBOVÉHO VÝVOJE 11

1.2.1 HTML5 ..... *12*

1.2.2 CSS3 ..... *13*

1.2.3 ECMAScript 6 (JavaScript) *14*

1.3 NÁSTROJE VE WEBOVÉM VÝVOJI 15

1.3.1 Version control systémy *15*

1.3.2 Automatizace testování a nasazování *16*

1.3.2.1 Jenkins: 17

1.3.2.2 Travis CI: 17

1.4 MOŽNOSTI A OMEZENÍ TECHNOLOGIÍ S OHLEDEM NA MSF 18

2 TECHNOLOGIE .NET A DOCKER VE VÝVOJI WEBOVÝCH APLIKACÍ PRO MALÉ A STŘEDNÍ FIRMY **19**

2.1 ÚVOD DO TECHNOLOGIE .NET 19

2.2 VÝHODY TECHNOLOGIE .NET 20

2.2.1 Výkon a Univerzalita *20*

2.2.2 Škálovatelnost a Bezpečnost *20*

2.3 ÚVOD DO TECHNOLOGIE DOCKER 20

2.4 VÝHODY TECHNOLOGIE DOCKER 21

2.4.1 Přenositelnost *21*

2.4.2 Rychlost *21*

2.4.3 Izolace *21*

2.4.4 Škálovatelnost *21*

2.4.5 Široká komunita a ekosystém *21*

2.4.6 Efektivní využití zdrojů *21*

2.4.7 Snadná správa a automatizace *22*

2.5 SYNERGICKÉ EFEKTY TECHNOLOGIÍ .NET A DOCKER 22

3 TVORBA WEBOVÉHO ROZHRANÍ WEBOVÝCH APLIKACÍ **23**

3.1 MODERNÍ FRAMEWORKY A KNIHOVNY 23

3.1.1 Vue.js ... *23*
3.2 Responsivní design ... 25
3.3 Stavový management ... 26
3.4 Bezpečnost na straně klienta ... 27
II. PRAKTICKÁ ČÁST ... 29
4 VYTVOŘENÍ API A PODPŮRNÝCH PROJEKTŮ ... 30
4.1 Tvorba API ... 31
4.1.1 Nakonfigurovaní Program.cs ... 31
4.1.2 Nastavení appsettings a launchSettings ... 33
4.1.3 Dependency injection ... 35
4.1.4 Příprava kontrolerů ... 36
4.1.5 Příprava modelů ... 38
4.1.6 Příprava profile ... 39
4.2 Přidání podprojektů ... 40
4.2.1 Domain ... 40
4.2.2 Application ... 41
4.2.3 Infrastructure ... 42
4.3 Seedování dat a testování ... 42
4.3.1 Seed data ... 43
4.3.2 Testování ... 44
5 TECHNOLOGIE DOCKER ... 45
5.1 Dockerfile ... 45
5.2 Konfigurace docker-compose ... 48
6 TVORBA UŽIVATELSKÉHO ROZHRANÍ ... 50
6.1 Uskupení UI.EshopApp ... 51
6.2 Uskupení UI.Common ... 52
6.3 Konečné zobrazení uživatelského rozhraní ... 53
ZÁVĚR ... 54
SEZNAM POUŽITÉ LITERATURY ... 56
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK ... 58
SEZNAM OBRÁZKŮ ... 59
SEZNAM PŘÍLOH ... 60

ÚVOD

Rychlý technologický pokrok v oblasti informačních technologií otevírá nové možnosti pro vývoj webových aplikací, které mohou efektivně podporovat potřeby malých a středních firem (MSF). Moje bakalářská práce se zaměřuje na zhodnocení a porovnání aktuálních technologií používaných při tvorbě webových aplikací a následné návrhy a implementace webové aplikace pro MSF s využitím technologií .NET a Docker.

První část práce se věnuje popisu současného stavu technologií pro tvorbu webových aplikací. Analyzuji trendy, standardy a nástroje, které jsou aktuálně využívány v oblasti webového vývoje. Zohledním jaké možnosti a omezení tyto technologie přinášejí, zejména s ohledem na specifické požadavky MSF.

Druhá část se detailně zaměřuje na technologie .NET a Docker. Prozkoumal jsem výhody, možnosti a přínosy, které tyto platformy přinášejí při vývoji webových aplikací. Důkladně analyzuji jejich schopnosti a jak mohou efektivně spolupracovat při tvorbě a správě aplikací pro MSF.

V třetí části se ponořím do detailu tvorby webového rozhraní webových aplikaci. Poukážu na jednotlivé technologie a odvětví, ve kterých se musí programátor pohybovat. Také vyzdvihnu určité technologie a zmíním problémové části.

Čtvrtá část práce obsahuje návrh webové aplikace s využitím technologie .NET. Identifikuji klíčové požadavky aplikace a navrhnul jsem odpovídající architekturu, která bude schopna efektivně podporovat potřeby MSF.

Pátá část se zabývá problematikou správy a přenositelnosti aplikace. Využiju technologii Docker k vytvoření kontejnerizovaného prostředí pro moji webovou aplikaci, což umožní snadnou správu, škálovatelnost a přenositelnost napříč různými prostředími.

Předposlední neboli šestá část obsahuje návrh projektu pro uživatelské rozhraní s využitím technologií Vue.js, TypeScript a Bootstrap. Objasním infrastruktůru tohoto projektu a také zde přidám projekt Common, který se bude využívát pro sdilení services a modelů pro případný další projekt v budoucnu.

V poslední části práce provedu zhodnocení dosažených výsledků a představím možnosti dalšího rozvoje aplikace. Diskutuji o přínosech a výzvách při využívání technologií .NET a Docker a navrhuju doporučení pro MSF, které chtějí efektivně využít moderní webové technologie pro svůj podnikatelský růst.

I.TEORETICKÁ ČÁST

1 TECHNOLOGIE TVORBY WEBOVÝCH APLIKACÍ

Pro tvorbu webových apikací se využivají různé technologie, které jsou schopné zpracovávat naprogramované kódy v různých jazycích. Technologie se nikdy nevyužívá pouze jedna k uskutečnění celkové webové aplikace. Webová aplikace se často skladají z nejméně 4 – 5 ruzných technologii a jazyků. Každá technologie má nějaký význam a část o kterou se stará. Například nějaká obyčejná webová aplikace, která se používá pouze interně v nějaké firmě. V tuto chvíli aplikace nemusí mít nějaké složité zabezpečení, ale může se mnohem více zabývat svojí rychlosti a jednoduchostí. Technologie pro webovou aplikaci tohoto kalibru by mohli být: ASPNET Core, Databáze, C#, JavaScript, Vue.js.

1.1 Trendy ve webovém vývoji

S rychlým technologickým pokrokem v oblasti informačních technologií dochází k neustálým změnám a inovacím ve webovém vývoji. Jedním z hlavních trendů je přechod k moderním architekturám, jako jsou například mikroslužby, které umožňují agilní a škálovatelný vývoj. Dalším významným směrem je rostoucí důraz na optimalizaci výkonu a uživatelského zážitku prostřednictvím technologií jako Progressive Web Apps (PWAs) a Single Page Applications (SPAs). Důležité je, že se tyto koncepty vzájemně nevylučují a často se kombinují, aby vytvořili vysoce výkonné a uživatelsky přívětivé webové aplikace.

1.1.1 Progressive Web Apps (PWAs)

PWA jsou webové aplikace, které využívají moderní webové technologie k tomu, aby uživatelům poskytly zážitek podobný nativní aplikaci.

Klíčové vlastnosti:

Progresivní vylepšení: PWA jsou navrženy tak, aby fungovaly na jakémkoli zařízení a postupně se vylepšovaly a využívaly nejnovější webové možnosti. [11]

Responzivní design: PWA jsou vytvořeny podle zásad responzivního designu, který zajišťuje bezproblémové používání na různých zařízeních a při různých velikostech obrazovky. [11]

Funkčnost offline: PWA mohou fungovat offline nebo v oblastech se špatným internetovým připojením díky servisním pracovníkům, kteří ukládají základní zdroje do mezipaměti. [11]

Interakce podobné aplikacím: PWA často poskytují plynulé uživatelské prostředí podobné aplikacím, včetně gest, plynulých přechodů a pohlcujících interakcí.

Možnost instalace: Uživatelé si mohou PWA přidat na domovskou obrazovku svého zařízení, takže jsou dostupné jako nativní aplikace bez nutnosti používat obchod s aplikacemi. [11]

Souhrnně lze říci, že PWA se zaměřují na poskytování uceleného a progresivního webového prostředí s funkcemi, jako je offline funkčnost, responzivita a možnost instalace

1.1.2 Single Page Applications (SPAs)

SPA jsou webové aplikace, které načítají jednu stránku HTML a dynamicky aktualizují obsah podle toho, jak uživatel s aplikací pracuje, aniž by bylo nutné načítat celou stránku.

Klíčové vlastnosti:

Asynchronní načítání: SPA používají AJAX (asynchronní JavaScript a XML) k načítání dat na pozadí a dynamické aktualizaci obsahu bez nutnosti načítat celou stránku. [12]

Plynulé přechody: SPA umožňují plynulé přechody mezi jednotlivými zobrazeními, čímž vytvářejí plynulejší uživatelský zážitek. [12]

Rychlejší výkon: Protože SPA načítají pouze nezbytné zdroje a aktualizují obsah podle potřeby, mohou ve srovnání s tradičními vícestránkovými aplikacemi nabídnout vyšší vnímaný výkon. [12]

Směrování na straně klienta: SPA často používají směrování na straně klienta pro správu navigace v aplikaci, což umožňuje plynulejší a citlivější ovládání. [12]

Společné rámce: K vytváření SPA se běžně používají populární frameworky JavaScriptu, jako jsou React, Angular a Vue.js.

SPA se soustředí na vytváření plynulého a responzivního uživatelského rozhraní pomocí dynamické aktualizace obsahu na jedné stránce.

1.2 Standardy v oblasti webového vývoje

Při tvorbě webových aplikací je klíčové dodržovat standardy, které zajišťují interoperabilitu a dlouhodobou udržitelnost projektů. Standardy, jako je HTML5, CSS3 a ECMAScript 6 (JavaScript), hrají klíčovou roli ve zlepšování kompatibility, bezpečnosti a výkonu webových aplikací.

1.2.1 HTML5

HTML5 (Hypertext Markup Language 5) je verze standardu HTML, což je značkovací jazyk používaný ke strukturování obsahu na webu. Jazyk HTML5 byl vyvinut s cílem vylepšit jej o podporu nejnovějších multimediálních, grafických a interaktivních funkcí, díky čemuž je vhodnější pro moderní vývoj webových stránek.

Sémantika: HTML5 zavádí nové sémantické prvky, například <header>, <footer>, <nav>, <article>, <section>, <figure> a <figcaption>. Tyto prvky pomáhají strukturovat webovou stránku smysluplnějším způsobem a usnadňují prohlížečům pochopení obsahu. [13]

Podpora multimédií: HTML5 obsahuje nativní podporu prvků pro zvuk a video (<audio> a <video>), čímž odpadá nutnost používat zásuvné moduly třetích stran, jako je například Flash. To umožňuje vkládat multimediální obsah přímo do webových stránek. [13]

Plátno: Element <canvas> poskytuje kreslicí plochu pro vytváření grafiky a animací pomocí JavaScriptu. Je obzvláště užitečný při vytváření dynamického a interaktivního obsahu, jako jsou hry nebo vizualizace dat.

Geolocation API: HTML5 obsahuje rozhraní Geolocation API, které umožňuje webovým aplikacím přistupovat k zeměpisné poloze uživatele. Tato funkce se běžně používá v polohových službách a mapových aplikacích. [13]

Místní úložiště: HTML5 zavedlo rozhraní API localStorage a sessionStorage, která webovým aplikacím umožňují ukládat data lokálně v zařízení uživatele. To je užitečné pro vytváření offline webových aplikací a zlepšení výkonu díky snížení potřeby neustálého načítání dat ze serveru. [13]

Web Workers: HTML5 umožňuje používat webové pracovníky, což jsou skripty na pozadí, které běží nezávisle na hlavním vlákně prohlížeče. Tato funkce umožňuje provádět paralelní zpracování, čímž se zvyšuje výkon webových aplikací. [13]

WebSockets: HTML5 podporuje WebSockets, které poskytují obousměrný komunikační kanál mezi klientem a serverem. To umožňuje přenos dat v reálném čase, takže je vhodný pro aplikace, které vyžadují okamžité aktualizace, jako jsou chatovací aplikace a online hry.

Responzivní webový design: HTML5 obsahuje funkce podporující responzivní webový design, které umožňují vytvářet webové stránky přizpůsobené různým velikostem obrazovek a zařízení. To má zásadní význam pro poskytování konzistentního uživatelského prostředí na různých platformách. [13]

Vylepšení formulářů: HTML5 zavádí nové typy vstupů a atributy, jako například <input type="date">, <input type="email"> a <input type="url">, které usnadňují vytváření a ověřování vstupů formulářů. [13]

Celkově hraje jazyk HTML5 zásadní roli při utváření moderního webu, protože poskytuje rozšířené možnosti a nástroje pro vytváření bohatých, interaktivních a multimediálních webových stránek.

1.2.2 CSS3

CSS3 neboli kaskádové styly 3 je vývoj jazyka kaskádových stylů, který se používá k popisu prezentace dokumentu napsaného v jazyce HTML nebo XML. CSS je základní technologií při vývoji webových stránek, která umožňuje stylovat a rozvrhovat webové stránky.

Zde jsou některé klíčové funkce a možnosti zavedené v CSS3:

Selektory: CSS3 zavádí nové a výkonnější selektory, které umožňují cílit na prvky s větší specifičností. Patří sem selektory atributů, pseudotřídy a pseudoelementy.[15]

Vylepšení box modelu: CSS3 poskytuje další vlastnosti pro lepší kontrolu nad box modelem, například vlastnost box-sizing, která umožňuje určit, zda mají být výplň a ohraničení prvku zahrnuty do jeho celkové šířky a výšky. [15]

Flexbox: Flexbox je model rozvržení, který umožňuje navrhovat složitá rozvržení a rozdělení prostoru v kontejneru, i když velikost prvků není známa nebo je dynamická. Zjednodušuje zarovnání a rozložení prostoru responzivním způsobem. [14]

Rozložení mřížky: Rozložení mřížky CSS je další výkonný systém rozvržení, který umožňuje navrhovat dvourozměrná rozvržení s řádky a sloupci. Poskytuje přesnou kontrolu nad umístěním a velikostí položek v rozvržení. [14]

Přechody a animace: CSS3 zavádí vlastnost přechodů, která umožňuje plynulé přechody mezi různými stavy. Pravidlo @keyframes umožňuje vytvářet animace a poskytuje větší kontrolu nad procesem animace. [15]

Transformace: CSS3 zavádí vlastnost transform, která umožňuje transformaci prvků z hlediska translace, rotace, měřítka a zkosení. To je užitečné zejména při vytváření interaktivních a dynamických uživatelských rozhraní. [15]

Mediální dotazy: Media queries v CSS3 umožňují použít různé styly na základě vlastností zařízení, jako je velikost obrazovky, rozlišení nebo orientace. To je nezbytné pro vytváření responzivních návrhů, které se přizpůsobí různým zařízením. [14]

Vlastní písma: CSS3 podporuje pravidlo @font-face, které umožňuje používat na webových stránkách vlastní písma. To zvyšuje typografickou flexibilitu a možnosti návrhu. [14]

Více pozadí: CSS3 umožňuje používat více obrázků na pozadí prvku s možností řídit jejich umístění, velikost a vrstvení. [14]

Přechody a stíny: CSS3 zavádí vlastnosti pro vytváření gradientních pozadí a stínů v rámečcích, což poskytuje větší kontrolu nad vizuálním vzhledem prvků. [14]

Tyto vlastnosti společně rozšiřují možnosti jazyka CSS a činí jej výkonnějším a flexibilnějším pro moderní vývoj webových stránek, zejména pokud jde o responzivní design a dynamická uživatelská rozhraní.

1.2.3 ECMAScript 6 (JavaScript)

ECMAScript 6, známý také jako ES6 nebo ECMAScript 2015, je významnou aktualizací programovacího jazyka JavaScript. V červnu 2015 byl oficiálně standardizován organizací ECMA International. ECMAScript je standard, na kterém je založen JavaScript, a každá nová verze přináší do jazyka vylepšení a nové funkce.

Zde jsou některé klíčové funkce zavedené v ECMAScriptu 6:

deklarace let a const: Let umožňuje deklarovat blokově omezené proměnné a nahrazuje tak tradiční deklaraci var.[16]

Funkce const slouží k deklaraci proměnných s konstantními hodnotami, čímž se zabrání jejich opětovnému přiřazení.

Šípkové funkce: Šipkové funkce poskytují stručnější syntaxi pro zápis funkčních výrazů, díky čemuž je kód čitelnější. [16]

Šablonové literály: Šablonové literály umožňují vkládání výrazů uvnitř řetězcových literálů pomocí zpětných znaků (``).[16]

Destrukturalizace přiřazení: Destrukturování umožňuje extrahovat hodnoty z polí nebo objektů do proměnných. [16]

Operátory Spread a Rest: Operátor spread (...) slouží k rozprostření prvků pole nebo objektu. [16]

Operátor rest umožňuje shromažďovat zbývající parametry do pole.

Třídy: ECMAScript 6 zavádí stručnější syntaxi pro definici tříd. [16]

Přísliby: Přísliby poskytují strukturovanější způsob zpracování asynchronních operací ve srovnání s zpětnými voláními. [16]

Moduly: ECMAScript 6 zavádí nativní podporu modulů, která umožňuje uspořádat kód do samostatných souborů a snadno opakovaně používat funkce. [16]

Tyto a další funkce zvyšují čitelnost, udržovatelnost a celkovou výkonnost kódu JavaScriptu. Mějte na paměti, že verze ECMAScriptu se stále vyvíjejí a v dalších verzích přibývají nové funkce.

1.3 Nástroje ve webovém vývoji

Široká paleta nástrojů podporuje při tvorbě moderních webových aplikací. Version control systémy, jako Git, zajišťují efektivní správu kódu a spolupráci v týmu. Nástroje pro automatizaci testování a nasazování, například Jenkins a Travis CI, jsou klíčové pro zajištění kvality a bezproblémového nasazení aplikací.

1.3.1 Version control systémy

Version controll systems (VCS) jsou nástroje, které pomáhají spravovat a sledovat změny zdrojového kódu projektu, dokumentů a dalších souborů v průběhu času. Jsou klíčové pro společný vývoj softwaru, protože umožňují, aby na projektu pracovalo více vývojářů současně bez konfliktů. Git je jedním z nejpoužívanějších systémů pro správu verzí.

Zde je přehled systému Git:

Distribuovaná správa verzí: Git je distribuovaný systém správy verzí, což znamená, že každý vývojář má na svém lokálním počítači kompletní kopii celého úložiště. To umožňuje práci offline a zlepšuje spolupráci.

Úložiště: Úložiště Git je kolekce souborů a historie jejich revizí. Obsahuje celou historii projektu a metadata. Může existovat centrální úložiště (na serveru) a více místních úložišť (na počítačích vývojářů).

Commit: Commit je snímek změn provedených v úložišti. Vývojáři vytvářejí commity, aby zaznamenali změny a poskytli popis toho, co bylo provedeno. Commity jsou identifikovány jedinečným hashem.

Větev (Branch): Systém Git umožňuje vývojářům vytvářet větve, což jsou samostatné vývojové linie. Každá větev může mít vlastní sadu změn a může být sloučena s jinými větvemi. To je užitečné zejména při vývoji funkcí a opravách chyb.

Merge: Merge je proces slučování změn z jedné větve do druhé. Systém Git automaticky řeší mnoho konfliktů při slučování, ale vývojáři mohou být nuceni konflikty řešit ručně, pokud se změny překrývají.

Klonování (Clone): Klonování úložiště vytvoří kopii celého projektu včetně jeho historie. To je užitečné pro vytvoření místní kopie vzdáleného úložiště.

Pull: Stáhne změny ze vzdáleného úložiště a začlení je do místního úložiště.

Push: Odešle zapsané změny z místního úložiště do vzdáleného úložiště.

Vzdálené úložiště (Remote): Vzdálený repozitář je odkaz na jinou kopii repozitáře. Vývojáři mohou odesílat změny do vzdálených úložišť nebo z nich stahovat změny.

Oblast staging: Před odevzdáním změn je vývojáři umístí do oblasti staging. To umožňuje selektivní odevzdávání a pomáhá logicky uspořádat změny.

.gitignore: Vývojáři mohou pomocí souboru .gitignore zadat soubory nebo vzory souborů, které má systém Git ignorovat. To je užitečné pro vyloučení generovaných souborů, dočasných souborů nebo jiných souborů, které by neměly být verzovány.

Systém Git poskytuje robustní a efektivní mechanismus pro správu verzí a je široce používán při vývoji open source i proprietárního softwaru. Díky decentralizované povaze systému Git je univerzální a přizpůsobitelný různým pracovním postupům.

1.3.2 Automatizace testování a nasazování

Nástroje pro automatizaci testování jsou softwarové aplikace, které usnadňují provádění testovacích případů a porovnávání skutečných výsledků s očekávanými. Používají se k automatizaci opakujících se, ale nezbytných testovacích úloh v průběhu celého životního cyklu vývoje.

Nástroje pro nasazení automatizují proces distribuce verzí softwaru do různých prostředí, takže je efektivní, opakovatelný a méně náchylný k chybám.

Nástroje pro automatizaci testování a nasazení, jako jsou Jenkins a Travis CI, hrají klíčovou roli v životním cyklu vývoje softwaru, protože automatizují různé úkoly, zvyšují efektivitu a zajišťují spolehlivost vydání softwaru.

1.3.2.1 Jenkins:

Jenkins je automatizační server s otevřeným zdrojovým kódem, který se široce používá k vytváření, testování a nasazování softwarových projektů. Podporuje automatizaci různých úloh v procesu vývoje softwaru, včetně sestavování kódu, spouštění testů a nasazování aplikací.

Kontinuální integrace (CI): Jenkins umožňuje kontinuální integraci automatickým sestavováním a testováním změn kódu, kdykoli vývojáři odevzdají změny do systémů pro správu verzí, jako je Git.[22]

Rozšiřitelnost: Jenkins je schopen pracovat s různými typy aplikací, např: Jenkins je vysoce rozšiřitelný díky velkému množství zásuvných modulů, které uživatelům umožňují integraci s různými nástroji, systémy pro správu verzí a nástroji pro sestavování. [22]

Distribuované sestavení: Jenkins dokáže distribuovat úlohy sestavení a testování na více strojů, čímž zvyšuje efektivitu a zkracuje dobu sestavení. [22]

Pipeline podpora: Jenkins podporuje definici komplexních sestavovacích a nasazovacích potrubí prostřednictvím souboru Jenkinsfile, což umožňuje modelovat celý proces dodávání softwaru. [22]

1.3.2.2 Travis CI:

Travis CI je cloudová distribuovaná služba kontinuální integrace, která automatizuje testování a nasazování softwarových projektů umístěných v repozitářích GitHub.

Integrace s GitHubem: Travis CI se bezproblémově integruje se službou GitHub a automaticky spouští sestavení a testy při každém odeslání kódu nebo požadavku na stažení.[23]

Podpora jazyků: Travis CI podporuje širokou škálu programovacích jazyků a poskytuje předinstalovaná prostředí pro populární frameworky, takže je univerzální pro různá vývojová prostředí.[23]

Maticová sestavení: Umožňuje konfiguraci maticových sestavení pro testování softwaru ve více verzích programovacích jazyků, závislostí nebo operačních systémů.[23]

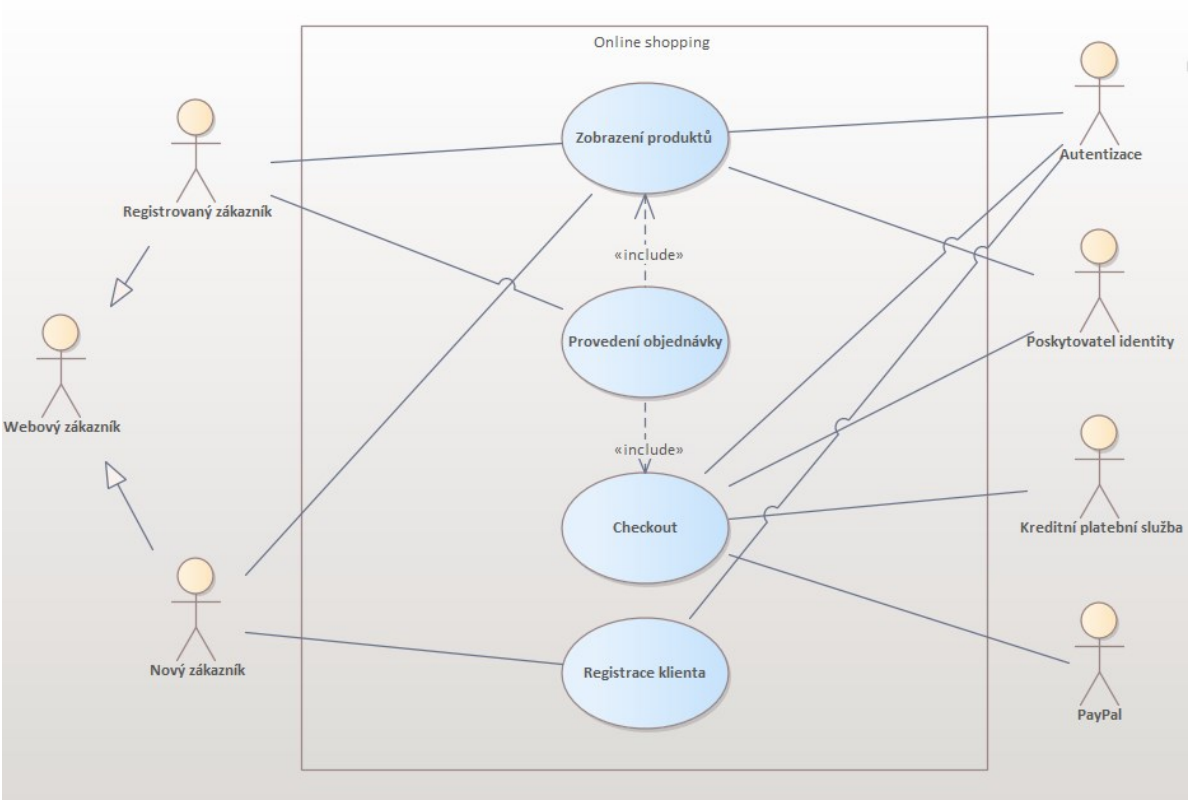
Nasazení: Travis CI podporuje automatizované nasazení na různé hostingové služby a cloudové platformy, což z něj činí komplexní řešení pro celý proces CI/CD.[24]

Fáze sestavení: Podobně jako pipelines Jenkins umožňuje Travis CI definovat fáze sestavení a modelovat tak komplexní pracovní postupy sestavení a nasazení.[24]

1.4 Možnosti a omezení technologií s ohledem na MSF

Při výběru technologií pro tvorbu webových aplikací pro malé a střední firmy je nutné brát v úvahu specifické potřeby a omezení tohoto segmentu. Zjistil jsem, že MSF často preferují technologie, které jsou nákladově efektivní, jednoduše udržovatelné a rychle škálovatelné. Omezení výpočetního výkonu a finančních prostředků jsou častými faktory, které ovlivňují rozhodování při výběru technologického stacku.

V následující části této práce se zaměřím na návrh webové aplikace pro MSF, přičemž využijeme technologie .NET a Docker. Tyto technologie byly vybrány s ohledem na jejich schopnost efektivně odpovídat potřebám MSF a zároveň poskytovat robustní a udržitelné řešení ve webovém vývoji. Samozřejmě že na samotný návrh webové aplikace pomocí diagramů je použito návrhové prostředí Enterprise Architect.



Obrázek 1 Use Case Diagram

2 TECHNOLOGIE .NET A DOCKER VE VÝVOJI WEBOVÝCH APLIKACÍ PRO MALÉ A STŘEDNÍ FIRMY

2.1 Úvod do technologie .NET

.NET je softwarový rámec vyvinutý společností Microsoft, který poskytuje prostředky pro vývoj, běh a správu aplikací. Jedná se o robustní a univerzální platformu, která podporuje různé programovací jazyky, jako jsou C#, VB.NET, F# a další. Hlavním cílem .NET je usnadnit vývoj spolehlivých a výkonných aplikací pro různé platformy, včetně Windows, Linux a macOS.

V jádru .NET stojí Common Language Runtime (CLR), který slouží jako běhové prostředí pro aplikace napsané v jazyce .NET. CLR zajišťuje správu paměti, bezpečnost, ovládání výjimek a další klíčové aspekty běhu aplikace. Díky této abstrakci může stejný kód běžet na různých operačních systémech, což usnadňuje multiplatformní vývoj.[4][6]

.NET nabízí také bohatou sadu knihoven a frameworků, které usnadňují vývoj různých typů aplikací. Například ASP.NET umožňuje vývoj webových aplikací, WPF a WinForms jsou určeny pro vývoj desktopových aplikací, a Xamarin umožňuje vytvářet mobilní aplikace pro iOS a Android. Tato rozmanitost umožňuje vývojářům vybírat si technologie podle konkrétních požadavků jejich projektů.[1]

Dalším klíčovým prvkem .NET je jeho otevřenost a podpora pro open-source komunitu. Vývojáři mohou přispívat k rozvoji různých částí .NET, což vede k rychlejšímu inovacím a vylepšením. .NET Core, odlehčená a multiplatformní verze .NET, byla také otevřena jako open-source projekt.[10]

Obrázek 2. Microsoft .NET Framework [6]

2.2 Výhody technologie .NET

2.2.1 Výkon a Univerzalita

Technologie .NET se vyznačuje vysokým výkonem a schopností být nasazena na různé platformy. Tato univerzalita umožňuje vytvářet aplikace, které jsou snadno přenosné a spustitelné na různých zařízeních a operačních systémech.[6]

2.2.2 Škálovatelnost a Bezpečnost

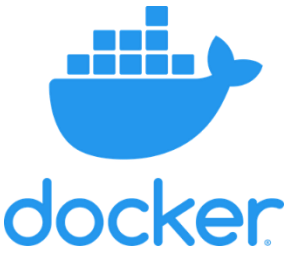
Díky vestavěným nástrojům a bezpečnostním prvkům poskytuje .NET robustní prostředí pro škálovatelný vývoj webových aplikací. Integrovaná podpora pro správu uživatelů, autorizaci a šifrování dat zajišťuje vysokou úroveň bezpečnosti pro aplikace MSF.[6]

2.3 Úvod do technologie Docker

Docker je revoluční technologie v oblasti kontejnerizace, která změnila způsob, jakým vývojáři, systémoví inženýři a operátoři nasazují, spravují a spouštějí aplikace. Tato open-source platforma umožňuje balení aplikací a všech jejich závislostí do jednotných kontejnerů, které jsou nezávislé na prostředí, ve kterém běží. To poskytuje konzistentní a spolehlivé prostředí pro vývoj, testování a produkční nasazení.

Kontejnerizace, kterou Docker přináší, umožňuje izolaci aplikací od operačního systému a hardware, což eliminuje problémy spojené s rozdílnými konfiguracemi a prostředími mezi vývojovými, testovacími a produkčními fázemi životního cyklu aplikace. Díky tomu můžete snadno sdílet a distribuovat kontejnery, aniž byste se museli starat o konflikty mezi verzemi knihoven, operačními systémy nebo dalšími závislostmi.[7]

Docker se skládá ze dvou hlavních komponent: Docker Engine a Docker Hub. Docker Engine je jádro samotné platformy, zatímco Docker Hub je cloudová služba, která umožňuje sdílet a spravovat kontejnery. Docker Hub poskytuje centrální úložiště pro obrazy kontejnerů, které lze snadno stáhnout a spustit na různých systémech.[2][3]



Obrázek 3. Docker[7]

2.4 Výhody technologie Docker

2.4.1 Přenositelnost

Docker kontejnery jsou nezávislé na prostředí, což znamená, že umožňuje spouštět stejný kontejner na libovolném zařízení nebo v libovolném prostředí, které podporuje Docker. To zajišťuje konzistentní chování aplikací od vývoje až po produkční prostředí.[3]

2.4.2 Rychlost

Docker umožňuje rychlou tvorbu a spouštění kontejnerů, což výrazně zrychluje vývoj a nasazování aplikací. Kontejnery mohou být spuštěny během několika vteřin, což je výrazně rychlejší než tradiční virtualizace. [3]

2.4.3 Izolace

Kontejnery poskytují izolaci aplikací a závislostí. Každý kontejner sdílí jádro operačního systému se svým hostitelským systémem, ale má vlastní souborový systém a oddělený proces. To eliminuje konflikty mezi aplikacemi a zajišťuje, že jedna aplikace nemá vliv na ostatní. [3][7]

2.4.4 Škálovatelnost

Docker umožňuje snadné škálování aplikací díky konceptu kontejnerů. Umožňuje jednoduše spouštět více instancí stejného kontejneru (horizontální škálování) nebo přidávat zdroje k jedné instanci (vertikální škálování), podle potřeby. To usnadňuje reagování na změny v zátěži nebo požadavcích aplikace. [3]

2.4.5 Široká komunita a ekosystém

Docker má obrovskou komunitu uživatelů a aktivní ekosystém. Existuje mnoho předdefinovaných obrazů kontejnerů dostupných na Docker Hub, což usnadňuje integraci a nasazení aplikací. Tato obrovská knihovna obrazů obsahuje různé operační systémy, databázové systémy, serverové aplikace a mnoho dalšího. [3]

2.4.6 Efektivní využití zdrojů

Docker umožňuje efektivní využití zdrojů, protože kontejnery sdílí jádro operačního systému a potřebné knihovny s hostitelským systémem. To znamená, že umožňuje spustit více kontejnerů na jednom fyzickém stroji bez zbytečné režie spojené s virtualizací. [3]

2.4.7 Snadná správa a automatizace

Docker poskytuje nástroje pro snadnou správu a automatizaci kontejnerů. Pomocí Docker Compose mohu definovat a spravovat celé aplikace v jednom konfiguračním souboru, což usnadňuje vývoj a nasazení. [3][7]

2.5 Synergické Efekty technologií .NET a Docker

V této části se podrobně analyzují synergické efekty spojené s kombinací technologií .NET a Docker při vývoji webových aplikací pro MSF. Zkoumám, jak tyto platformy spolupracují na dosažení optimálních výsledků v oblasti efektivity, škálovatelnosti a bezpečnosti.

Izolace a konzistence prostředí: Docker umožňuje vytvoření kontejnerů, které obsahují všechny závislosti a konfigurace potřebné pro spuštění aplikace. S použitím Dockeru mohu zajistit, že aplikace bude spouštěna v konzistentním prostředí, což snižuje riziko chyb spojených s rozdílnými konfiguracemi prostředí mezi vývojovými, testovacími a produkčními prostředími.

Snadnější nasazení a škálování: Docker umožňuje jednoduché nasazení aplikací do kontejnerů, které lze snadno spouštět a zastavovat. S kombinací .NET a Docker mohu snadno nasadit svou webovou aplikaci a škálovat ji podle potřeby, což umožňuje rychle reagovat na změny v zátěži.

Průhledné vývojové prostředí: Díky Dockeru můžu mít vývojové prostředí, které je podobné produkčnímu prostředí, což může minimalizovat problémy spojené s odlišnými prostředími vývoje a produkce.

Snadnější správa závislostí: Pomocí Dockeru můžu snadno spravovat závislosti aplikace a zajistit, že každý člen týmu bude pracovat se stejnými verzemi knihoven a frameworků.

Flexibilita a modularita: Díky Dockeru můžu snadno integrovat další služby a mikroslužby do webové aplikace. Umožňuje vytvářet různé kontejnery pro různé části aplikace a snadno je propojovat.

Snadnější migrace: Použití Dockeru může zjednodušit proces migrace aplikací mezi různými prostředími nebo mezi různými verzemi .NET frameworku.

3 TVORBA WEBOVÉHO ROZHRANÍ WEBOVÝCH APLIKACÍ

V dnešní éře digitální transformace a rozmachu internetu se vytváření webových aplikací stává klíčovým prvkem pro úspěšný rozvoj a konkurenceschopnost podniků. Tvorba webového rozhraní, jako klíčového spojení mezi uživateli a aplikací, vyžaduje nejen estetický design, ale také efektivní technologický základ. V této oblasti vývoje je nezbytné sledovat a využívat moderní frameworky, knihovny a postupy, které umožňují vytvářet robustní, responzivní a bezpečné webové aplikace. Zde se zaměřuji na důležité aspekty tvorby webových rozhraní, od moderních frameworků a knihoven, přes responsivní design až po optimalizaci výkonu a zabezpečení na straně klienta.

3.1 Moderní frameworky a knihovny

Moderními frameworky a knihovnami v oblasti programování uživatelského rozhraní se rozumí, takové technologie jako jsou React.js, Vue.js, Angular a mnoho dalších. Tyto technologie rozšiřují možnosti v oblasti UI. Každá z těchto technologii má své výhody i nevýhody.

Celkovou největší nevýhodou všech UI technologii je jejich růst a dokola opakujicí se cyklus rychlého posunu nebo změny. To bohužel každému projektu přidává nestálost a nutí to vývojáře furt aktualizovat a udržovat svoji webovou aplikaci v chodu s dobou. Pokud se tak neučinní, webová aplikace je vystavována velmi velkým rizikům.

3.1.1 Vue.js

Vue.js je progresivní framework JavaScriptu, který se používá k vytváření uživatelských rozhraní. Je od základu navržen tak, aby byl postupně osvojitelný, což znamená, že mohu Vue.js postupně integrovat do stávajících projektů podle potřeby. Vue.js je často oceňován pro svou jednoduchost a flexibilitu.

Deklarativní vykreslování: Vue.js používá deklarativní přístup k definici uživatelského rozhraní. Popíši požadovaný konečný výsledek a Vue.js se postará o aktualizaci DOM tak, aby odpovídal tomuto stavu.[17]

Architektura založená na komponentách: Vue.js je postaven na komponentové architektuře. Komponenty jsou opakovaně použitelné a zapouzdřené stavebními bloky, které lze skládat a vytvářet jak komplexní aplikace. [17]

Direktivy: Vue.js poskytuje sadu vestavěných direktiv, které lze použít k provádění úloh v DOM. Například direktiva **v-if** slouží k podmíněnému vykreslování a direktiva **v-for** k vykreslování seznamů. [17]

Reaktivní vazba dat: Vue.js používá systém reaktivního vázání dat. Když se podkladová data změní, uživatelské rozhraní se automaticky aktualizuje tak, aby tyto změny odráželo. [17]

Vypočtené vlastnosti: Vue.js umožňuje definovat vypočtené vlastnosti, které jsou odvozeny z podkladových dat, ale jsou uloženy v mezipaměti a znovu vyhodnoceny pouze v případě potřeby. To pomáhá optimalizovat výkon. [17]

Směrovač Vue: Vue Router je oficiální směrovač pro Vue.js. Umožňuje vytvářet jednostránkové aplikace s navigací na straně klienta. [17]

Vuex: Vuex je knihovna pro správu stavů pro Vue.js. Pomáhá spravovat stav aplikace v centralizovaném úložišti. [17]

Vue CLI: Vue CLI (Command Line Interface) je sada nástrojů pro rychlý vývoj Vue.js. Zahrnuje projektové lešení, nástroje pro sestavení a systém zásuvných modulů. Zjednodušuje proces nastavení a správy projektů Vue.js. [17]

Vue DevTools: Rozšíření prohlížeče, které umožňují kontrolovat a ladit aplikace Vue.js v prohlížeči. Poskytuje přehled o hierarchii komponent, stavu a událostech. [17]

Vue.js si získalo oblibu díky své jednoduchosti, snadné integraci a živé komunitě. Často se volí jak pro malé projekty, tak pro větší a složitější aplikace. Oceňují jeho přehlednou dokumentaci a možnost postupného osvojování do stávajících projektů.

Obrázek 4. Vue.js[17]

3.2 Responsivní design

Responzivní design označuje přístup k návrhu a tvorbě uživatelského rozhraní, které se přizpůsobuje a optimálně zobrazuje obsah na různých zařízeních a při různých velikostech obrazovky. Cílem je zajistit bezproblémový a konzistentní uživatelský zážitek bez ohledu na to, zda je aplikace přístupná na stolním počítači, notebooku, tabletu nebo chytrém telefonu.

Mezi klíčové principy a techniky patří:

Fluidní rozvržení mřížky: Místo pevných rozvržení založených na pixelech používá responzivní design relativní jednotky, jako jsou procenta pro šířku a výšku.[21]

Mřížkové systémy jsou navrženy tak, aby byly plynulé a umožňovaly dynamické přizpůsobení obsahu v závislosti na velikosti obrazovky.

Mediální dotazy: Dotazy na média CSS3 umožňují aplikaci použít různé styly na základě vlastností, jako je šířka a výška obrazovky, orientace zařízení a rozlišení.[14]

Mohu definovat specifická pravidla CSS pro různé body zlomu, čímž zajistím, že se design vhodně přizpůsobí.

Flexibilní obrázky a média: Obrázky a mediální prvky jsou nastaveny tak, aby se škálovaly podle velikosti zobrazovací plochy pomocí vlastností CSS, jako je max-width: 100 %.[15]

Responzivní typografie: Velikost písma je definována pomocí relativních jednotek (např. em, rem), aby se zajistilo, že se vhodně škáluje na různých zařízeních. [21]

Progresivní vylepšení: Začíná se základním funkčním rozvržením a postupně vylepšují design a funkce podle toho, jak se zvětšuje velikost zobrazovacího panelu. [21]

Tím je zajištěno, že i uživatelé s menšími obrazovkami mají přístup k základnímu obsahu a funkcím.

Design zaměřený na mobilní zařízení: Důraz se klade na návrh nejprve pro mobilní zařízení a poté na postupné vylepšování pro větší obrazovky. [21]

Začleněním těchto zásad a technik mohu vytvářet frontendy webových aplikací, které poskytují bezproblémové a příjemné uživatelské prostředí na různých zařízeních a obrazovkách různých velikostí.

3.3 Stavový management

Správa stavů ve frontendu webové aplikace obvykle znamená manipulaci s různými stavy nebo podmínkami, kterými může uživatelské rozhraní projít během svého životního cyklu. To zahrnuje správu prezentace dat, obsluhu interakcí s uživatelem a reakce na různé události.

Stav místní součásti: Mnoho webových frameworků a knihoven, například React a Vue.js, umožňuje komponentám udržovat lokální stav. Tento stav může představovat různé stavy nebo režimy, ve kterých se komponenta může nacházet.[19]

Správa globálního stavu: U složitějších aplikací se správa globálního stavu stává klíčovou. Nástroje jako Redux nebo Vuex pomáhají centralizovat a spravovat celkový stav aplikace. [19]

Podmíněné vykreslování: V závislosti na podmínkách nebo interakcích uživatele může být nutné dynamicky zobrazovat nebo skrývat různé komponenty nebo prvky. [19]

Podmíněné třídy/styl: Použití specifických stylů nebo tříd na základě určitých podmínek pro změnu vzhledu prvků. [19]

Zpracování chyb: Implementace ověřování na straně klienta pro uživatelské vstupy, aby bylo možné zabránit chybám před odesláním dat na server. [19]

Zobrazení smysluplných chybových zpráv uživatelům v případě problémů, které jim poradí, jak problém vyřešit.

Asynchronní operace: Správa stavů během asynchronních operací, jako jsou volání API, které indikují načítání, úspěch nebo chybové stavy. [19]

Načítání spinnerů/indikátorů: Zobrazení spinnerů nebo indikátorů načítání, které informují uživatele o probíhajících procesech. [19]

Ukazatele nástrojů a upozornění: Poskytování informativních tooltipů nebo výstrah, které uživatele navedou ke konkrétním akcím nebo podmínkám. [19]

Upozornění toastem: Zobrazení neinvazivních oznámení o důležitých událostech, zprávách o úspěchu nebo upozorněních na chyby. [19]

Validace v reálném čase: Ověření vstupu uživatele v reálném čase a poskytnutí zpětné vazby před odesláním formuláře. [19]

Zpracování odeslání formuláře: Správa stavu uživatelského rozhraní během odesílání formuláře, indikace stavu načítání nebo úspěšného odeslání. [19]

Strážci tras: Implementuování route guards pro správu přístupových podmínek pro různé části aplikace. [19]

Podmíněná navigace: Povolení nebo zakazání možnosti navigace na základě určitých podmínek. [19]

Uživatelské role a oprávnění: Správa prvků uživatelského rozhraní na základě role a oprávnění uživatele. [19]

Stavy ověřování: Správa změn uživatelského rozhraní na základě toho, zda je uživatel přihlášen, nebo ne. [19]

Efektivní správa podmínek ve frontendu přispívá k pružnější odezvě, uživatelsky přívětivější a robustnější webové aplikaci. Zahrnuje pečlivé zvážení různých stavů, interakcí s uživatelem a potenciálních scénářů, které mohou ovlivnit uživatelské prostředí.

3.4 Bezpečnost na straně klienta

Zabezpečení na straně klienta ve frontendových webových aplikacích se týká opatření a postupů implementovaných k zabezpečení uživatelského rozhraní a funkcí, které jsou spuštěny ve webovém prohlížeči uživatele.

Ověřování vstupů: Validace a úprava uživatelských vstupů na straně klienta, aby se zabránilo útokům typu Cross-Site Scripting (XSS).[18]

Ochrana proti Cross-Site Scripting (XSS): Duležité je použití hlavičky zásad zabezpečení obsahu (CSP) ke snížení rizika útoků XSS tím, které skripty je povoleno spouštět. Musím zabránit spuštění škodlivých skriptů. [18]

Ochrana proti zfalšování požadavku na jinou stránku (CSRF): Implementováním tokenů proti CSRF, kvůli ověření, že požadavky pocházejí od legitimního uživatele a nebyly podvrženy škodlivým útočníkem. [18]

Zabezpečená komunikace (HTTPS): Zajistění, aby veškerá komunikace mezi klientem a serverem byla šifrována pomocí protokolu HTTPS, který chrání data během přenosu. [18]

Potřebné je vyvarování se používání smíšeného obsahu (kombinace zabezpečeného a nezabezpečeného obsahu), abyste zabránili potenciálním bezpečnostním zranitelnostem.

Zabezpečení obsahu: Tady musím být opatrný při používání knihoven a obsahu třetích stran. Načítejte pouze zdroje z důvěryhodných zdrojů, aby bylo předejito možným bezpečnostním rizikům. [18]

Nutnost pravidelné aktualizace a oprava knihoven třetích stran, kvůli odstranění případné bezpečnostní chyby.

Bezpečnostní funkce prohlížeče: Bezpečnostní funkce moderních prohlížečů, jako jsou soubory cookie SameSite, abych zabránil úniku informací mezi jednotlivými uživateli. [18]

Bezpečnostní hlavičky: Nastavení bezpečnostní hlavičky, včetně zásad zabezpečení obsahu (CSP), Strict-Transport-Security (HSTS) a X-Content-Type-Options, aby bylo zvýšeno zabezpečení webové aplikace. [18]

Úvahy o mobilních zařízeních: Musím brát v potaz i bezpečnostní opatření pro mobilní zařízení, jako je bezpečné ukládání citlivých dat, ochrana proti manipulaci s kódem a bezpečné zpracování offline dat. [18]

Řešením těchto aspektů zabezpečení na straně klienta mohu u své webové aplikace výrazně zvýšit celkové zabezpečení a ochránit jak data uživatelů, tak integritu samotné aplikace.

II. PRAKTICKÁ ČÁST

4 VYTVOŘENÍ API A PODPŮRNÝCH PROJEKTŮ

Celkové vytvoření webová aplikace začíná se zaměřením na vytvoření API a doprovodných projektů. Proces vytváření robustního a škálovatelného API jsem uskutečnil ve vývojovém prostředí Visual Studio 2022.

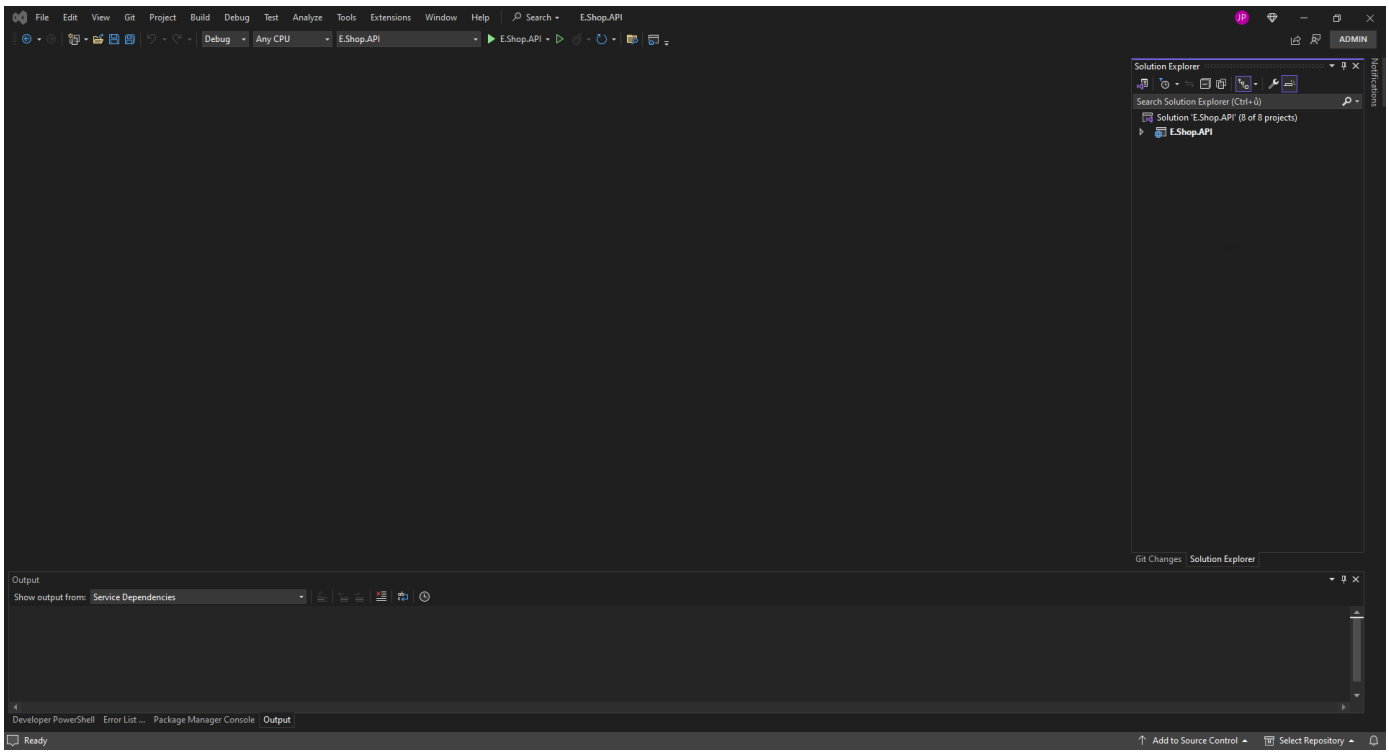
Práce na vytvoření API a souvisejících projektů, jako jsou Application, Infrastructure, Domain a Unit testy, představuje klíčový krok ve vývoji moderního webového eshopu. Cílem této práce je poskytnout kompletní pohled na proces vytvoření těchto komponent, aby bylo možné posoudít jak efektivní je nasadit a spravovat eshop s důrazem na kvalitu, výkon a udržitelnost pro MSF.

Mnou použitý postup je pouze jeden z mnoha standardů, ale prográmator by vždy měl přijít s originalitou a vyjimečností svého kódu. Proto jsem se rozhodl použit určité technologie určitým způsobem, tak aby byli velmi efektivními a doneslo nám to originalitu a možnost porovnání s ostatními postupy a technologiemi.

Věřím, že tato práce poskytne pevný základ pro úspěšné vytvoření API a doprovodných projektů pro webový eshop.

4.1 Tvorba API

V rychlém světě vývoje softwaru je vytváření robustních a škálovatelných rozhraní API klíčovou dovedností. Visual Studio 2022 poskytuje nástroje a funkce, které zefektivňují proces vývoje API. V této kapitole se ponořím do procesu vytváření rozhraní API .NET pomocí aplikace Visual Studio 2022 krok za krokem a cestou prozkoumám osvědčené postupy, základní koncepty a praktické příklady.

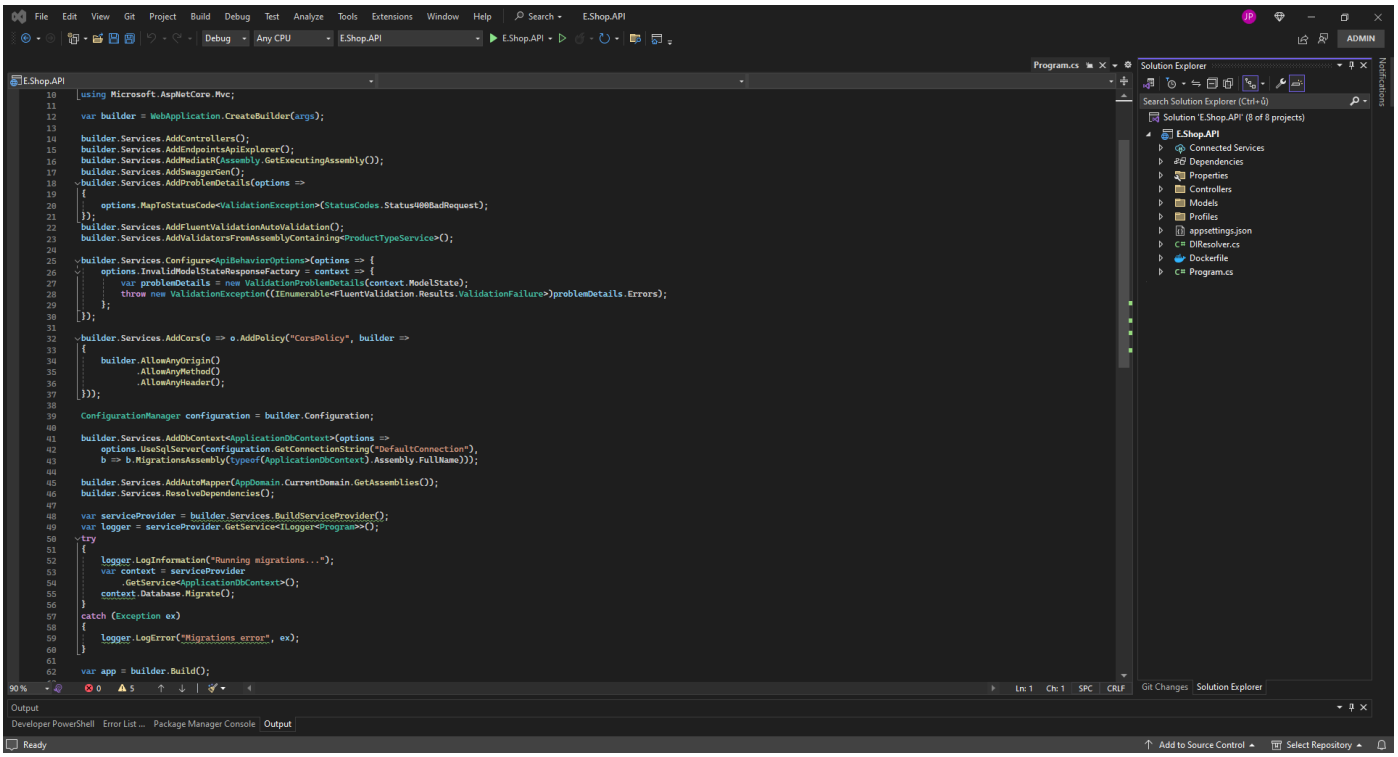


Obrázek 5 Přidání API

4.1.1 Nakonfigurovaní Program.cs

Program.cs nastavuje základní aplikaci ASP.NET Core pomocí vzoru builder.

Tento soubor program.cs nastavuje aplikaci ASP.NET Core s potřebnými službami, middlewarem a konfiguracemi pro zpracování požadavků a odpovědí HTTP, včetně ověřování a zpracování chyb.



Obrázek 6 Konfigurace Program.cs

Importy: Importuje potřebné jmenné prostory, jako jsou Microsoft.EntityFrameworkCore, MediatR, FluentValidation atd.

Nastavení sestavovače: V rámci sestavení sestavovače se vytvoří vzory typu C a C: Vytvoří sestavovač webové aplikace.

Konfigurace služby:

- Registruje různé služby, jako jsou kontroléry, MediatR pro zpracování příkazů a dotazů, Swagger pro dokumentaci API, FluentValidation pro ověřování vstupů, middleware ProblemDetails pro standardizované chybové odpovědi, zásady CORS, AutoMapper pro mapování objektů a nastavuje kontext databáze a migrace.
- Konfiguruje také vlastní továrnu na odpovědi na neplatný stav modelu pro zpracování chyb validace.

Migrace: Spouští migrace databáze, pokud jsou zjištěny nějaké čekající migrace.

Nastavení aplikace:

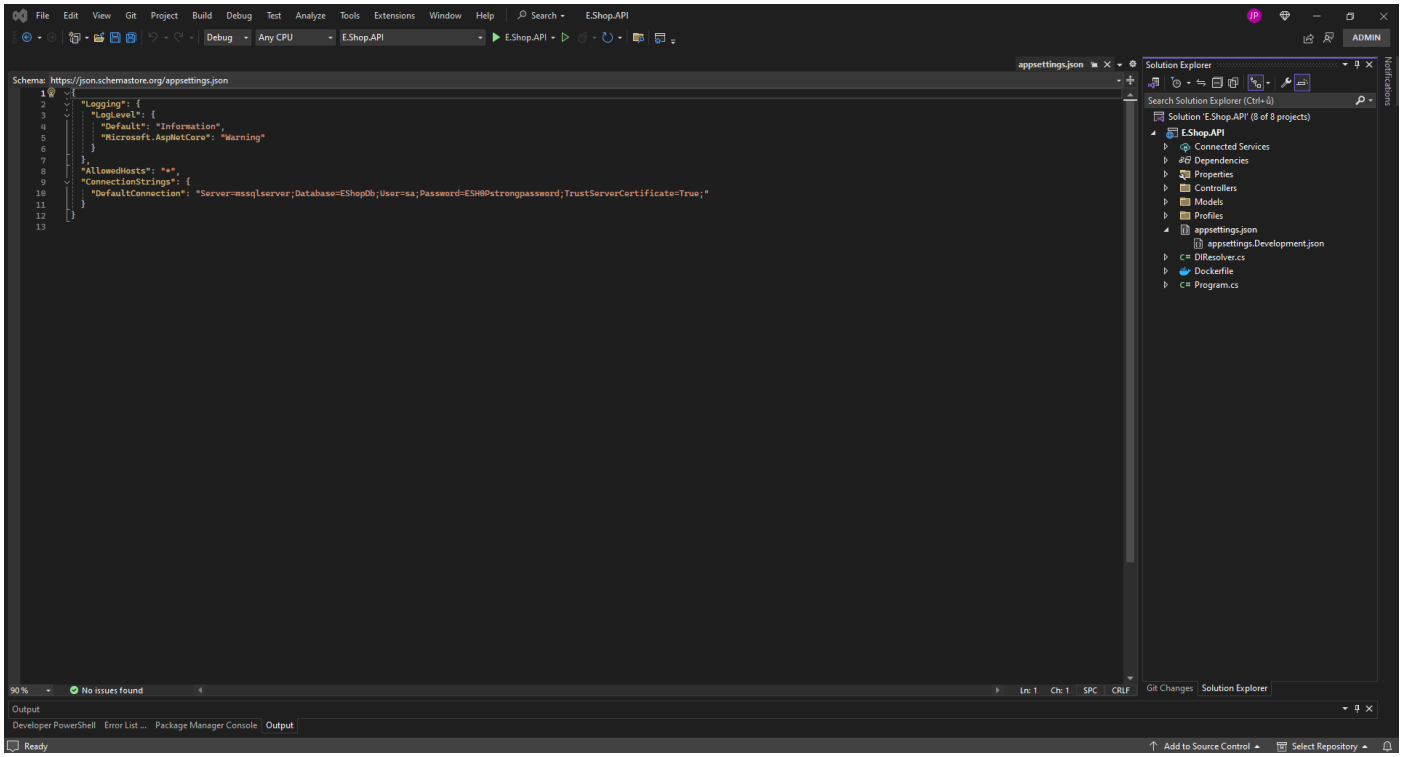
- Konfiguruje middleware jako ProblemDetails, Swagger UI, CORS, přesměrování HTTPS a autorizaci.
- Mapuje řadiče.
- Spustí aplikaci.

Protokolování: Zaznamenává informace a chyby při nastavení aplikace.

4.1.2 Nastavení appsettings a launchSettings

Každá aplikaci ASP.NET Core musí obsahovat konfigurační soubor appsettings.json.

Soubor appsettings.json konfiguruje úrovně protokolování, povolené hostitele a řetězec připojení pro vaši aplikaci ASP.NET Core.



Obrázek 7 Appsettings.json

Logging Configuration:

- V této části se konfiguruje protokolování pro moji aplikaci.
- **LogLevel.Default** nastavuje výchozí úroveň protokolu na "Information", což znamená, že se budou zaznamenávat zprávy se závažností "Information", "Warning", "Error" a "Critical".
- **LogLevel.Microsoft.AspNetCore** nastavuje úroveň protokolu speciálně pro kategorii Microsoft.AspNetCore na "Warning", což znamená, že pro tuto kategorii budou protokolovány pouze zprávy se závažností "Warning", "Error" a "Critical".

AllowedHosts:

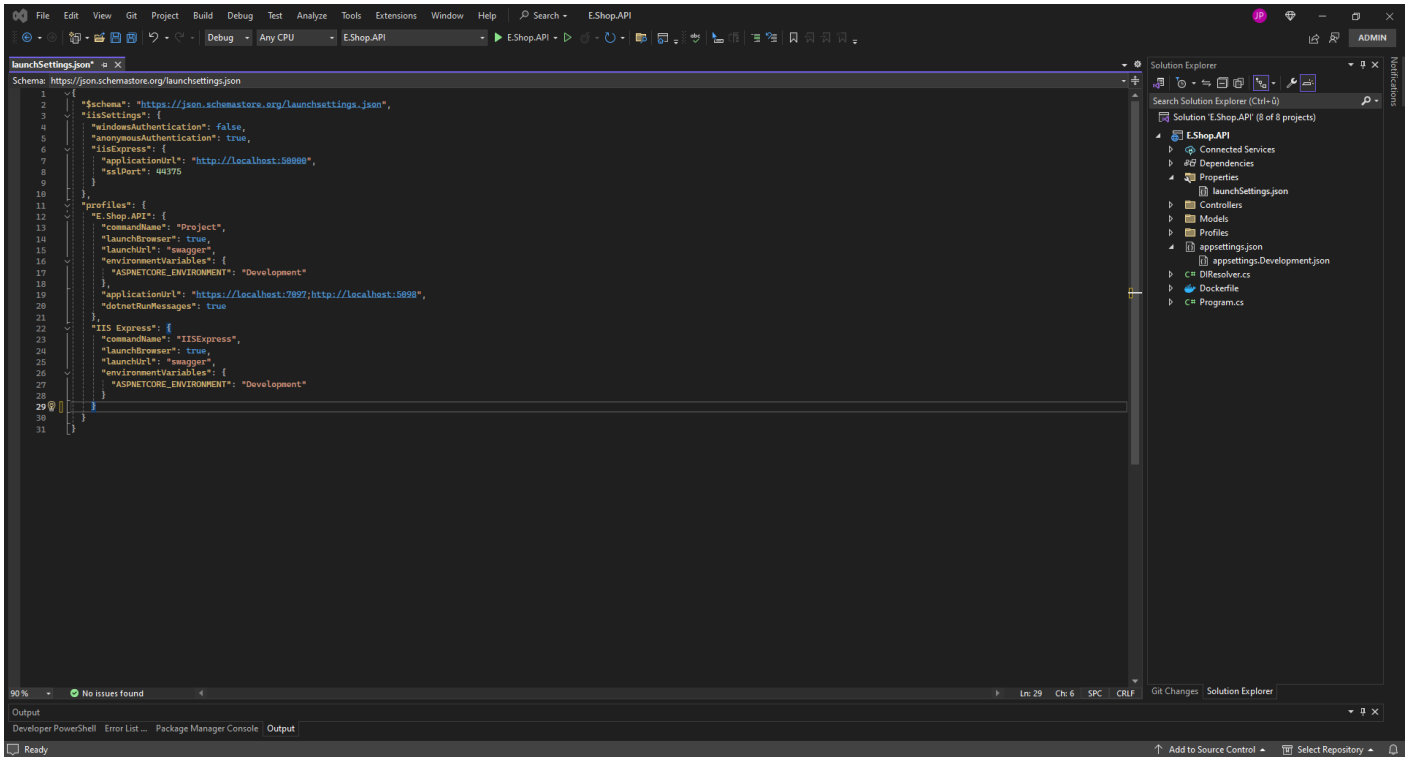
- Tato položka určuje, kteří hostitelé mají povolen přístup k aplikaci.

- Hodnota "*" znamená, že jsou povoleni všichni hostitelé. V produkčním prostředí musím s tímto nastavením být opatrný.

ConnectionStrings:

- Tato část definuje připojovací řetězec pro připojení aplikace k databázi SQL Server.
- **"DefaultConnection"** je název připojovacího řetězce.
- Samotný připojovací řetězec určuje podrobnosti, jako je název serveru (Server), název databáze (**Database**), uživatelské jméno (**User**), heslo (**Password**) a další možnosti (v tomto případě **TrustServerCertificate**).
- **Poznámka:** Ukládání hesel do konfiguračních souborů se z bezpečnostních důvodů obecně nedoporučuje. Zvažte použití jiných metod, jako jsou proměnné prostředí nebo Azure Key Vault, pro bezpečné ukládání citlivých informací.

Soubor **launchSettings.json** konfiguruje způsob spouštění aplikace ASP.NET Core v různých prostředích.



Obrázek 8 launchSettings.json

$schema: Tento parametr určuje schéma JSON, které se má použít pro ověřování.

iisSettings: Konfigurační nastavení pro Internetovou informační službu (IIS) Express, což je odlehčená, samostatná verze IIS optimalizovaná pro vývojáře.

- **windowsAuthentication:** Určuje, zda je povoleno ověřování systému Windows.
- **anonymousAuthentication:** Určuje, zda je povoleno anonymní ověřování.
- **iisExpress:** Konfigurace specifická pro službu IIS Express, včetně adres URL, které má používat (applicationUrl a sslPort).

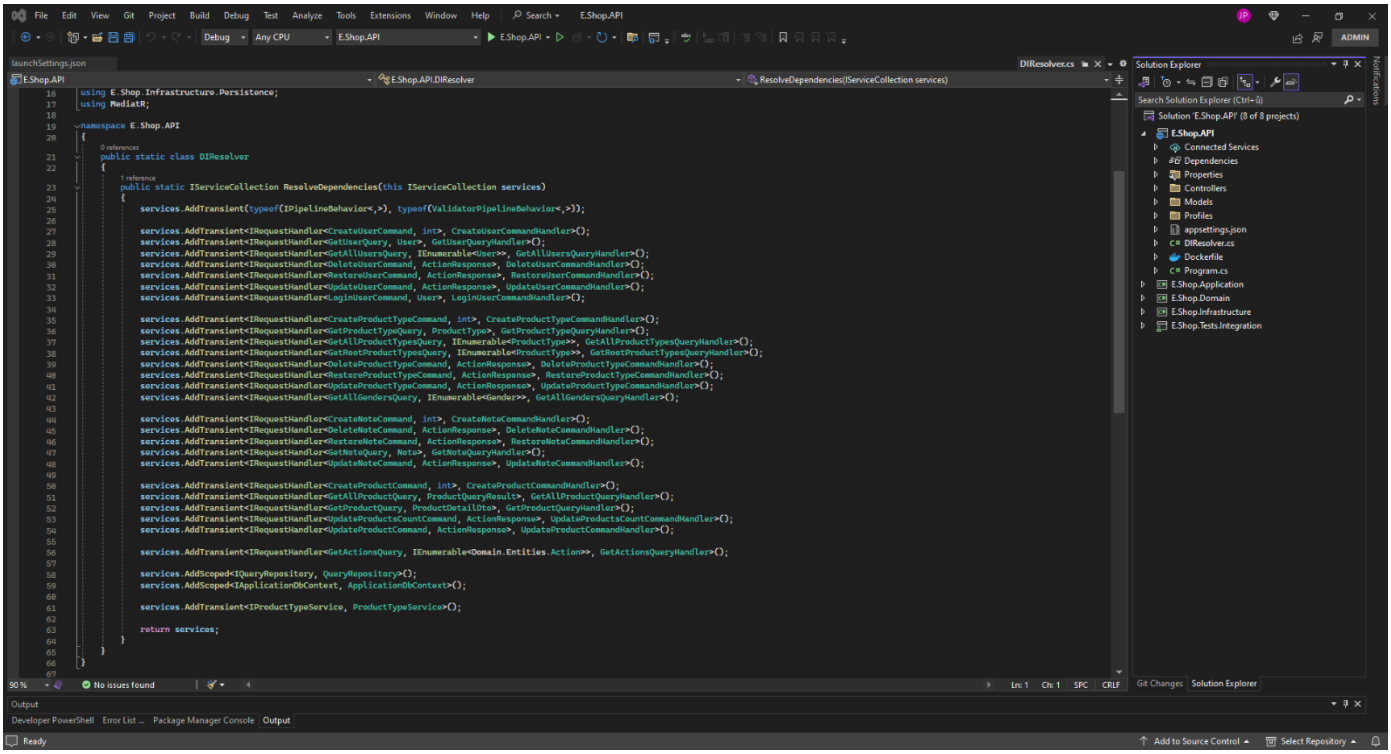
profiles: Definuje různé profily pro spouštění aplikace.

- **E.Shop.API:** Profil pro spuštění vašeho projektu API. Ke spuštění projektu používá příkaz dotnet run.
- **commandName:** Určuje typ příkazu pro spuštění projektu.
- **launchBrowser:** Určuje, zda se má při spuštění aplikace spustit výchozí prohlížeč.
- **launchUrl:** Adresa URL, která se má otevřít v prohlížeči při spuštění, v tomto případě je to "swagger", což obvykle odkazuje na uživatelské rozhraní Swagger pro dokumentaci API.
- **applicationUrl:** Určuje adresy URL, na kterých má aplikace naslouchat, a to jak HTTP, tak HTTPS.
- **dotnetRunMessages:** Povoluje podrobné zprávy během procesu dotnet run.
- **IIS Express:** Profil pro spuštění aplikace pod službou IIS Express.
- **commandName:** Určuje použití IISExpress.
- **launchBrowser:** Určuje, zda se má při spuštění aplikace spustit výchozí prohlížeč.
- **launchUrl:** Adresa URL, která se má otevřít v prohlížeči při spuštění, v tomto případě je to opět "swagger".
- **environmentVariables:** Jedná se o nastavení hodnoty "Development" do proměnné prostředí **ASPNETCORE_ENVIRONMENT**.

4.1.3 Dependency injection

Definuje statickou třídu s názvem DIResolver v rámci jmenného prostoru E.Shop.API.

Třída obsahuje metodu s názvem ResolveDependencies, která rozšiřuje IServiceCollection. Tato metoda je zodpovědná za registraci závislostí v rámci kontejneru DI (dependency injection) aplikace.



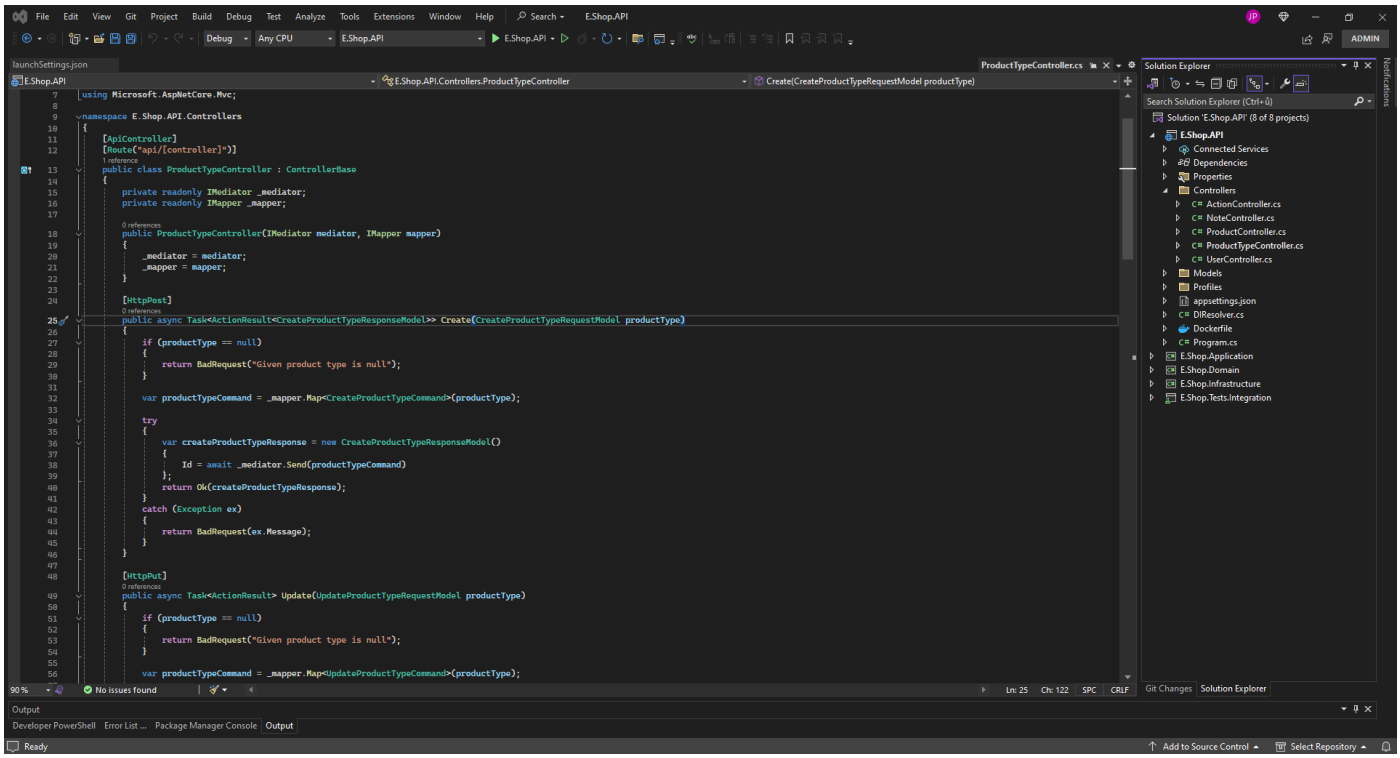
Obrázek 9 Dependency injection resolver

Registruje různé služby a jejich odpovídající obsluhy pomocí metody AddTransient, která označuje, že při každém požadavku na službu bude vytvořena její nová instance. Tyto služby se zřejmě týkají správy uživatelů, správy produktů, správy poznámek a dalších.

- Pro implementaci vzoru mediátor se používá knihovna MediatR, jak naznačuje použití IRequestHandler.
- Validátory jsou aplikovány na potrubí pomocí ValidatorPipelineBehavior.
- IQueryRepository a IApplicationDbContext jsou služby s rozsahem, což znamená, že se vytvářejí jednou pro každý požadavek.
- Kromě toho jsou některé DTO, entity a odpovědi na akce odkazovány ze jmenného prostoru E.Shop.Domain.
- Nakonec je registrována služba IProductTypeService.

4.1.4 Příprava kontrolerů

Kontrolery jsou zodpovědní za zpracování HTTP požadavků souvisejících s typy produktů v aplikaci elektronického obchodu.



Obrázek 10 Vzhled kontroleru

Routing: Kontrolér zpracovává požadavky s prefixem /api/ProductType. Například požadavek POST na /api/ProductType bude zpracován metodou Create a požadavek GET na /api/ProductType/{id} bude zpracován metodou Get.

Dependency Injection:

- Konstruktor řadiče přebírá dvě závislosti: **IMediator** a **IMapper**.
- **IMediator** pochází z knihovny **MediatR**, běžně používaný pro zpracování příkazů a dotazů.
- **IMapper** je z knihovny **AutoMapper**, která se používá pro mapování objektů na objekty.

Action Methods: Každá veřejná metoda v kontroléru představuje akci, kterou lze provést na typech produktů. Tyto metody jsou ozdobeny atributy určujícími metodu HTTP, na kterou reagují (HttpPost, HttpPut, HttpGet, HttpDelete), a v některých případech dalšími parametry trasy.

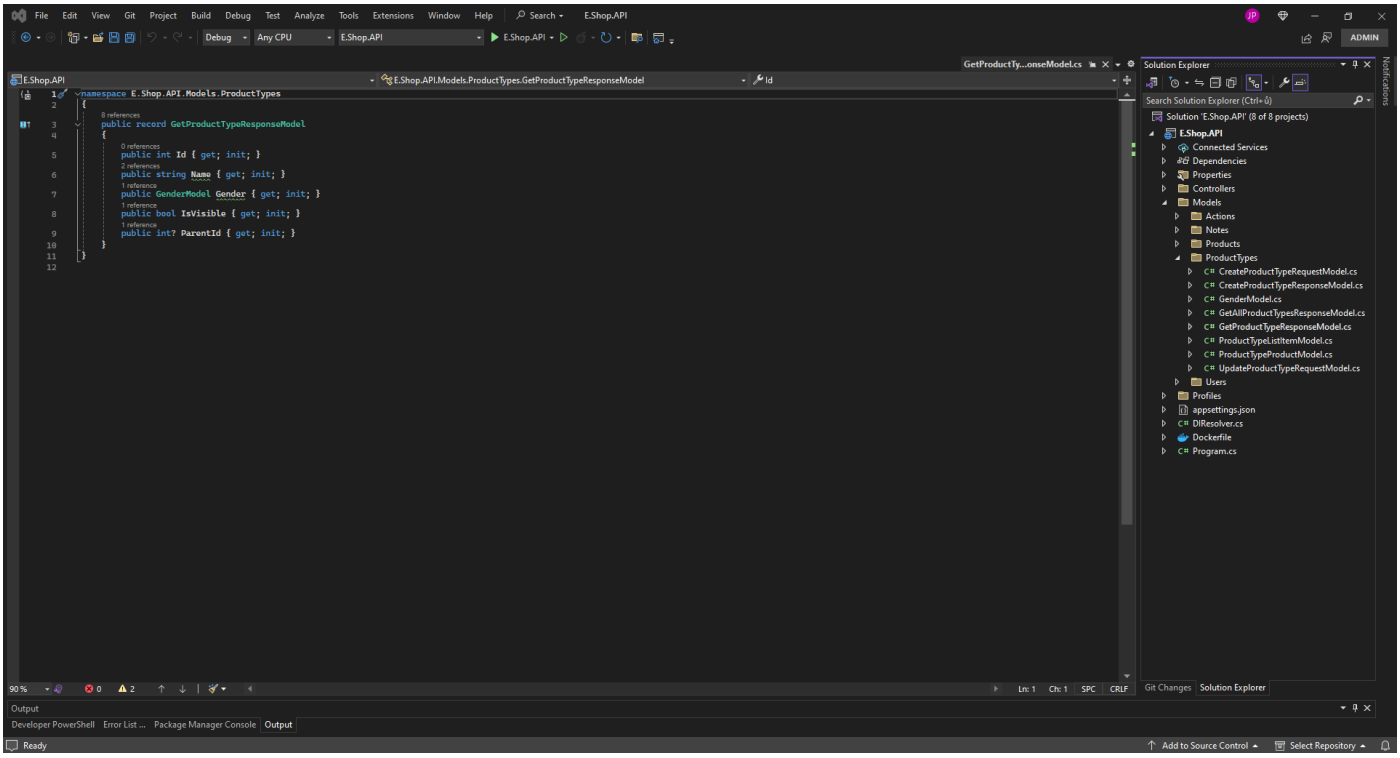
Request and Response Models: Metody Action přijímají vstupní parametry představující data ke zpracování a vracejí ActionResult nebo Task<ActionResult> představující odpověď HTTP. Často přijímají nebo vracejí objekty vlastního modelu.

Zpracování chyb: Implementováno pomocí bloků try-catch, kde jsou zachyceny výjimky vyhozené během zpracování požadavku a vráceny příslušné odpovědi HTTP. Například pokud není nalezen typ produktu (EntityNotFoundException), je vrácena odpověď 404 Not Found.

Mediator Pattern: Kontrolér deleguje zpracování příkazů a dotazů na pipeline MediatR odesláním instancí objektů příkazů nebo dotazů. To podporuje oddělení zájmů a umožňuje lepší organizaci aplikační logiky.

4.1.5 Příprava modelů

Model odpovědi pro načítání typů produktů z rozhraní API e-shopu. Zapouzdřuje informace, jako je ID, název, pohlaví, viditelnost a ID rodiče pro každý typ výrobku.



Obrázek 11 Vzhled modelu

Veřejná třída GetProductTypeResponseModel: Poskytuje stručnou syntaxi pro deklarování neměnných datových typů.

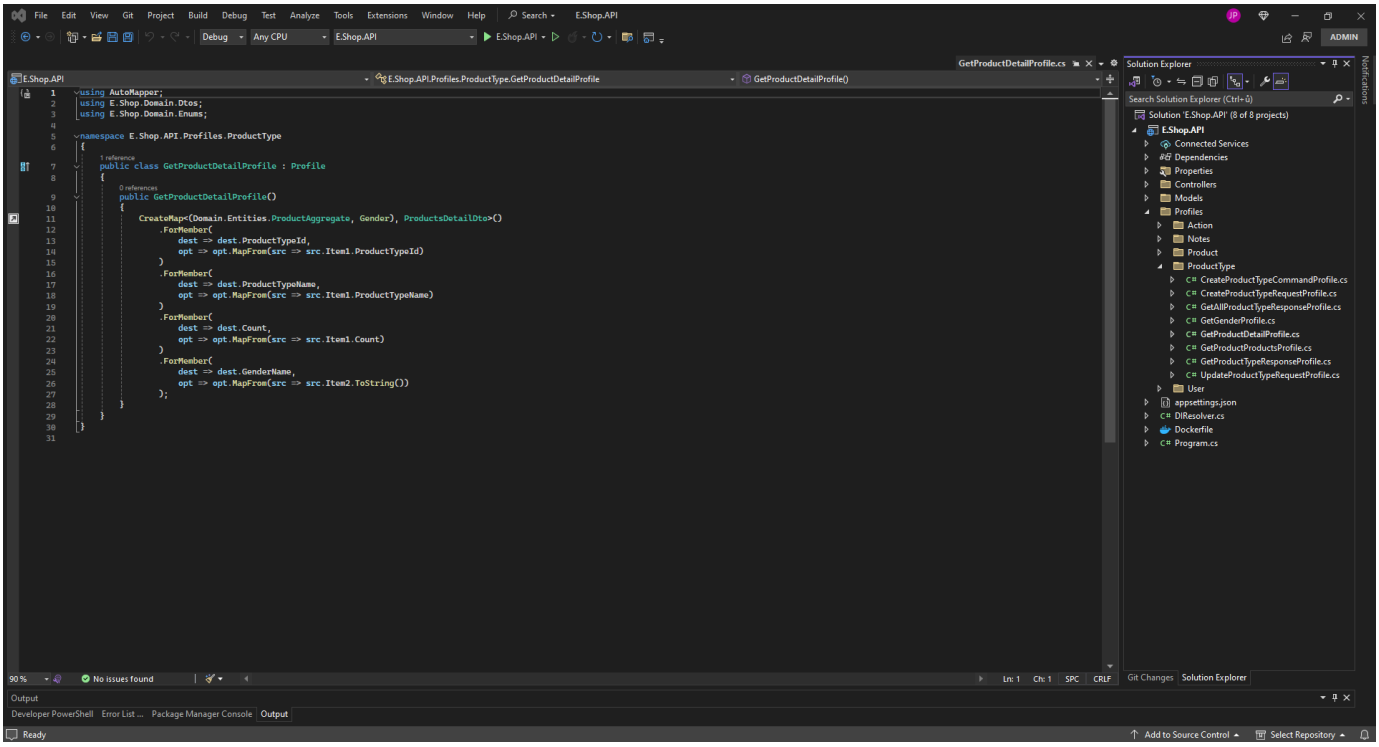
Uvnitř záznamu je několik vlastností:

- **public int Id { get; init; }:** Představuje ID typu produktu. Přístupový příkaz init umožňuje nastavit hodnotu během inicializace objektu, ale ne po ní.
- **public string Name { get; init; }:** Představuje název typu produktu.

- **public GenderModel Gender { get; init; }:** Představuje pohlaví spojené s typem produktu. Odkazuje na jinou třídu nebo strukturu s názvem GenderModel.
- **public bool IsVisible { get; init; }:** Tato vlastnost udává, zda je typ výrobku viditelný.
- **public int? ParentId { get; init; }:** Tato vlastnost představuje ID nadřazeného typu produktu. Symbol **?** označuje, že vlastnost může být nulová.

4.1.6 Příprava profile

Profil mapování pro nástroj AutoMapper. AutoMapper je knihovna v prostředí .NET, která zjednodušuje mapování objektů z jednoho typu na druhý.



Obrázek 12 Vzhled profilu

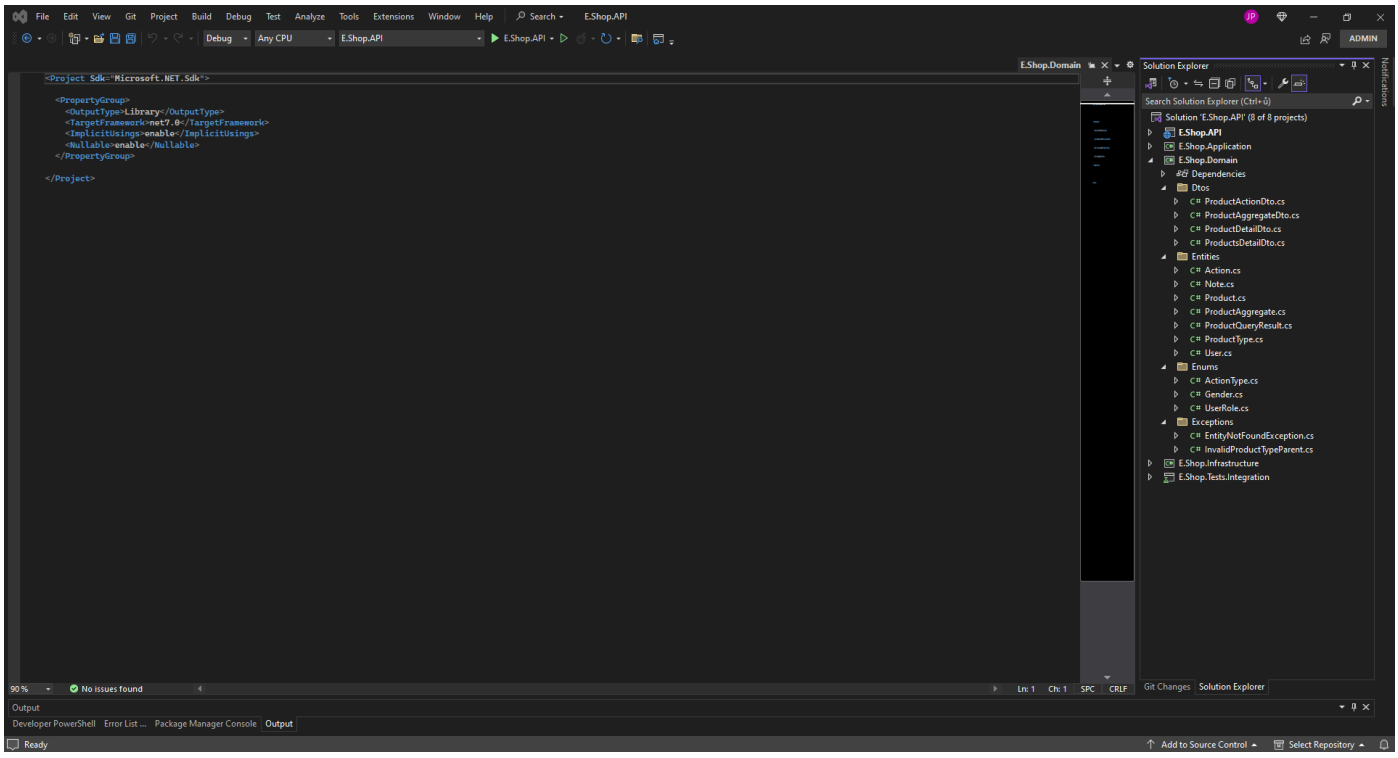
- Třída dědí od třídy **Profile** aplikace AutoMapper, což znamená, že se jedná o profil používaný pro mapování objektů.
- Uvnitř konstruktoru je volání CreateMap, které nastaví mapování mezi dvěma typy.
- Zdrojovým typem mapování je tuple, kde ProductAggregate je entita představující nějaký druh produktu a Gender je enum.
- Cílovým typem mapování je ProductsDetailDto
- Po metodě CreateMap následuje několik volání ForMember, z nichž každé určuje, jak namapovat konkrétní člen zdrojového typu na člen cílového typu.

4.2 Přidání podprojektů

Rozšíření základu projektu rozhraní .NET API o další podprojekty zlepšuje organizaci kódu, podporuje oddělení problémů a usnadňuje škálovatelnost a udržovatelnost. Tato kapitola se zabývá procesem vytváření a integrace těchto podprojektů do řešení API pomocí aplikace Visual Studio 2022.

4.2.1 Domain

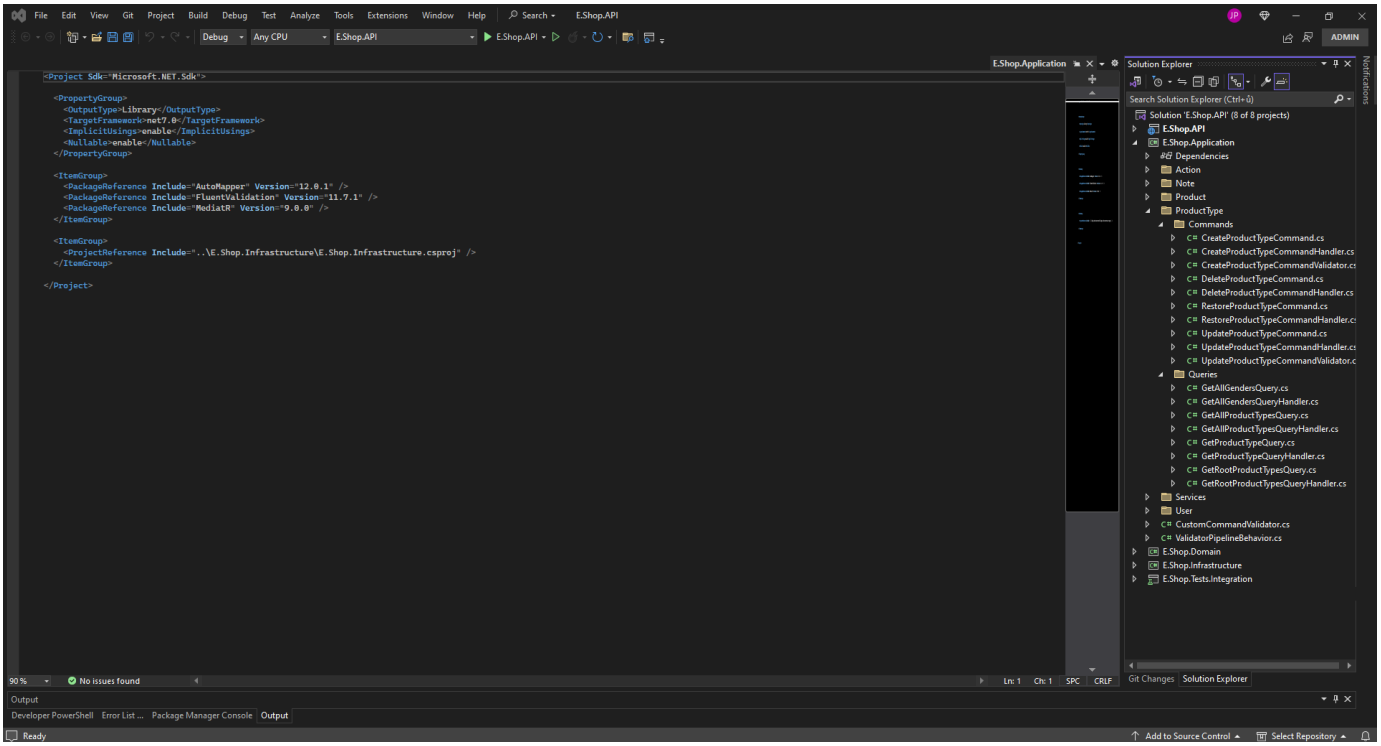
S jasnou představou o struktuře dílčích projektů začnu vytvořením projektu Domain v rámci řešení Visual Studio. Vytvořím základ modelu domény, definuju entity, hodnotové objekty a služby domény, které zapouzdřují business logiku aplikace. Dodržováním zásad návrhu řízeného doménou se zajistí, že projekt Domain zůstane soustředěný, soudržný a snadno rozšiřitelný v průběhu vývoje aplikace.



Obrázek 13 Infrastruktůra a nastavení Domain projektu

4.2.2 Application

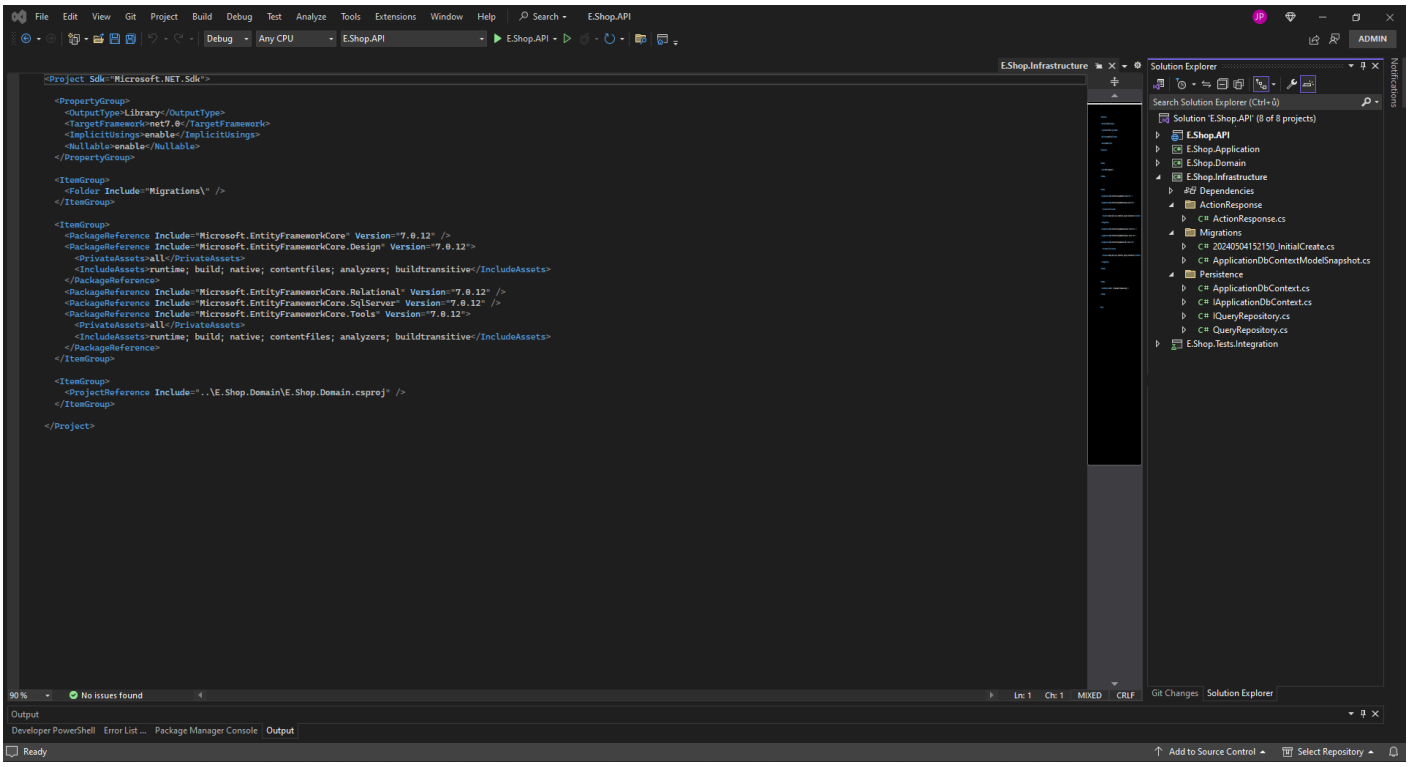
Na základě modelu domény vytvořeného v projektu Domain se zaměřím na projekt Application. Zde definuji případy užití, aplikační služby a logiku specifickou pro aplikaci, která organizuje interakce mezi entitami domény a externími klienty. Prostřednictvím praktických příkladů a návrhových vzorů, jako je vzor příkazu a vzor prostředníka, se dá naučit, jak oddělit obchodní logiku od zájmů infrastruktury, což podpoří udržovatelnost a testovatelnost celé aplikace.



Obrázek 14 Infrastruktůra a nastavení Application projektu

4.2.3 Infrastructure

Po vytvoření základní doménové a aplikační vrstvy se zaměřím na projekt Infrastruktura, který slouží jako rozhraní mezi aplikací a externími závislostmi. Prozkoumám, jak konfigurovat a integrovat součásti infrastruktury. Visual Studio poskytuje výkonné nástroje a rámce pro zefektivnění integrace a podporu konzistence celé aplikace.



Obrázek 15 Rozdělení a obsah projektu Infrastructure

4.3 Seedování dat a testování

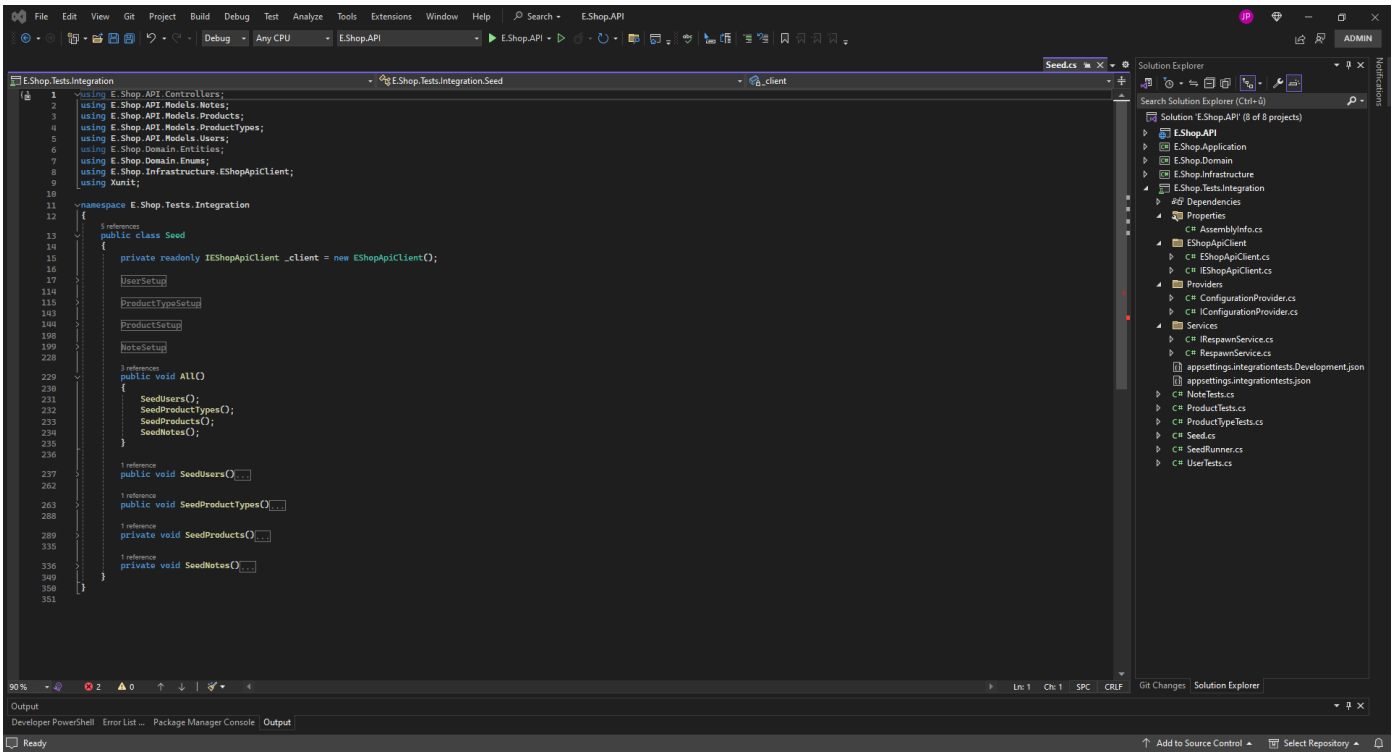
Každy kdo něco vytváří musí nějak ověřit zda je to plně funkční. V programátorském světě se tomuto ověřování říká testování. Na základě testu se lze naleznout konkretní chyby nebo problémy v logice projektu. Take je velmi důležité si otestovat svůj projekt a ověřit si tak jeho plnou funkcionalitu před nasazením do produkce. Testování má několik různých forem a druhu, já jsem si pro svůj projekt vybral integrační testy.

Programátor musí mít také validní data na kterých může provádět testy. Tyto data si musím nějak vytvořit nebo natáhnout. Je mnoho různých zpusobu jak si tzv. naseedovat, na kterých se poté mohou provádět testy. V dnešní době se programátor musí zamyslet nad tím že data nebude seedovat jen jednou, neboť skoro vždy nejde nic podle plánu a nebo se bude muset

nějaká část dat měnit, ať už kvůli jednoduchému zapomenutí nebo kvůli zákazníkovy. Já osobně jsem si vybral možnost seedování dat pomocí testu, neboli jsem vytvořil seedovací testy.

4.3.1 Seed data

Tento soubor slouží k nasazení počátečních dat do systému pro integrační testování a zajišťuje, že se systém při práci s různými entitami, jako jsou uživatelé, produkty a poznámky, chová podle očekávání. Je to vlastně integrační testovací třída pro vkládání dat do aplikace elektronického obchodu.



Obrázek 16 Provedení seedovaní dat

Inicializace polí: Nastavuje vzorová data pro uživatele, typy produktů, produkty a poznámky. Například pole _user, _productType, _product a _notes jsou inicializována vzorovými daty.

Metody nastavení: Existují metody nastavení jako SeedUsers, SeedProductTypes, SeedProducts a SeedNotes, které odpovídají za vytvoření vzorových dat pro uživatele, typy produktů, produkty a poznámky.

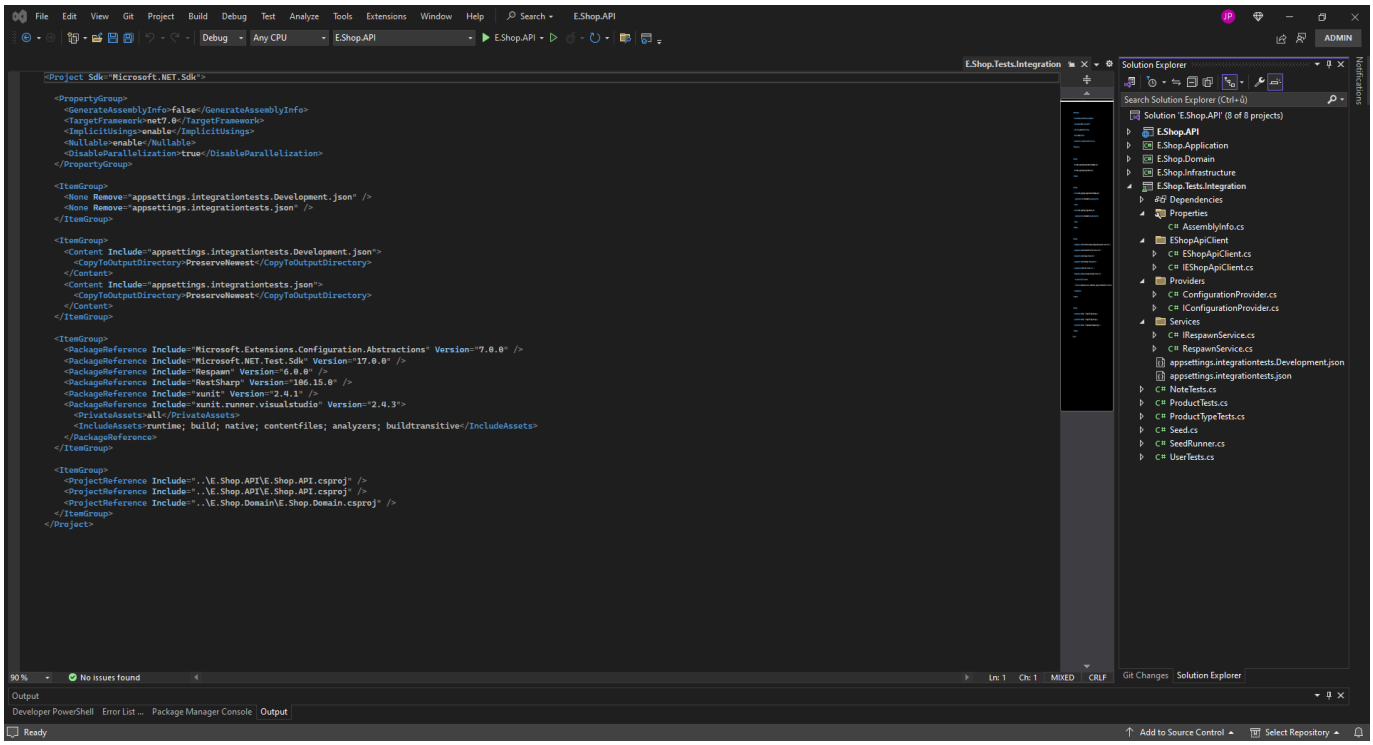
Metody testování integrace: Třída Seed obsahuje metody jako All, která organizuje nasazení všech typů dat, a jednotlivé metody pro nasazení uživatelů, typů produktů, produktů a poznámek.

Assertions (Tvrzení): V průběhu metod seeding jsou prováděna tvrzení, která zajišťují správné vložení dat do systému.

4.3.2 Testování

Integrační testy jsou typem testování softwaru, při kterém se jednotlivé moduly nebo komponenty softwaru kombinují a testují jako skupina. Účelem integračních testů je zajistit, aby interakce mezi moduly fungovaly podle očekávání a aby byly v rámci většího systému bezproblémově integrovány.

Testováním jsem ověřil všechny základní funkce (vytvoření, updatovaní, deletovaní, restorování), které musí být plně použitelné. Ověření funkcionality jsem neprovedl pouze na jedné entitě, duležité je ověřit všechnu základní funkcionalitu u každé entity. Entity se od sebe liší jak už v datech tak i v logice jaké se s nimi pracuje.



Obrázek 17 Rozdělení a nastavení Intergration Tests rojektu

5 TECHNOLOGIE DOCKER

V dnešní době, kdy se moderní softwarové projekty stávají stále složitějšími a dynamickými, je klíčové hledat efektivní a spolehlivé způsoby, jak spravovat jejich nasazení a běh. Technologie kontejnerů se staly základním stavebním kamenem pro tyto účely, a v jejich čele stojí Docker.

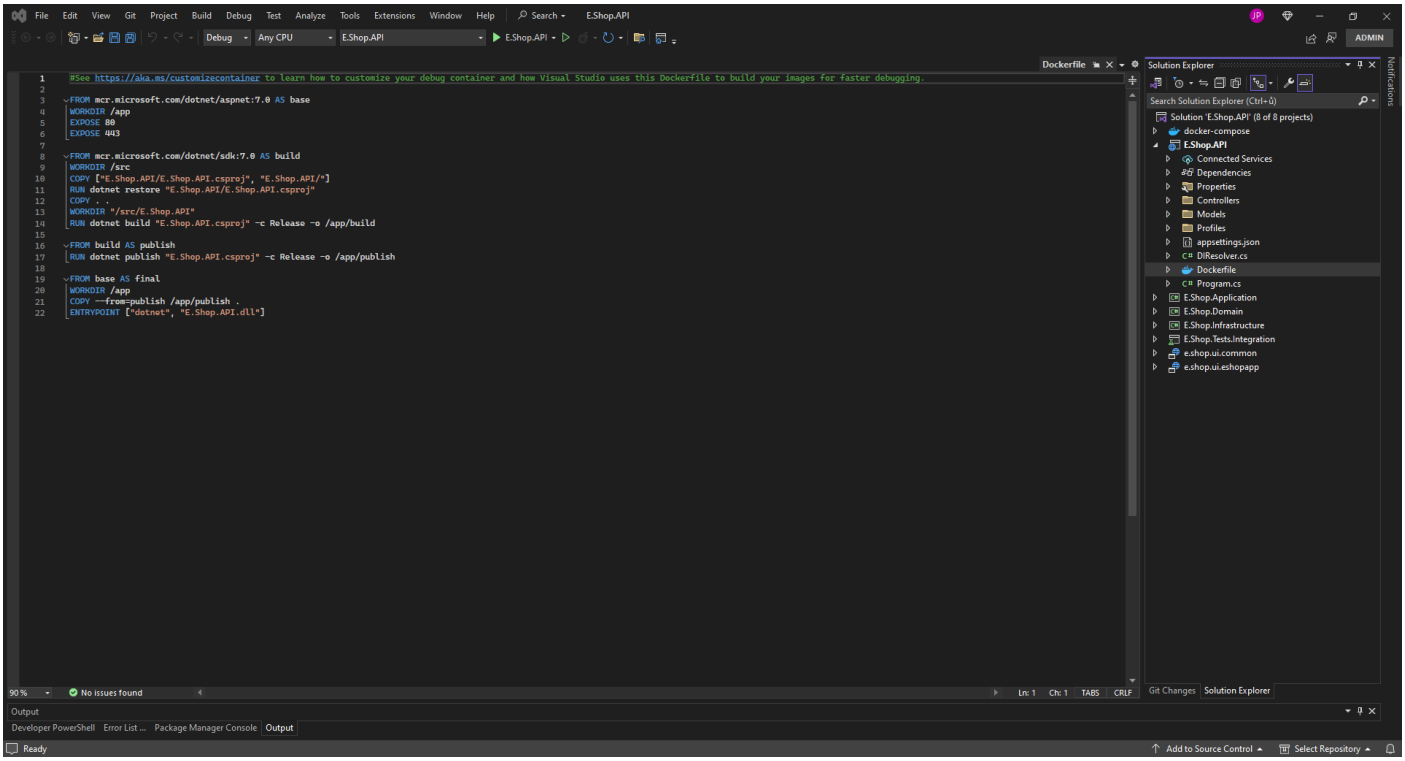
V této kapitole se zaměřím na technologii Docker, která se stala nepostradatelnou součástí moderního vývoje softwaru. Začnu zkoumáním Dockerfile, klíčového nástroje pro definici a vytváření kontejnerů. Prozkoumám jeho strukturu, syntaxi a základní příkazy, které umožňují vytvořit reprodukovatelné a snadno spravovatelné kontejnery.

Dále se podívám na Docker-compose, nástroj, který umožňuje definovat a spouštět více kontejnerů jako součást jediné aplikace. Ukážu, jak vytvořit konfiguraci pomocí Docker-compose a jak jednoduše propojit jednotlivé kontejnery a nastavit jejich konfiguraci.

Tato kapitola poskytne pevný základ při kontejnerizovaní mojí aplikace.

5.1 Dockerfile

Docker slouží k sestavení obrazu Docker pro aplikaci .NET, konkrétně webové rozhraní API ASP.NET.



Obrázek 18 Nazorná ukázka Dockerfile

Dockerfile rozděluje proces sestavení do několika fází, což pomáhá optimalizovat ukládání do mezipaměti a zmenšit konečnou velikost obrazu. Nejprve sestaví aplikaci, poté ji zveřejní a nakonec vytvoří odlehčený obraz obsahující pouze nezbytné součásti pro běh aplikace.

Základní obraz:

- FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
- Tento řádek určuje základní obraz pro proces sestavení. Jako základna se používá obraz běhového prostředí ASP.NET pro .NET 7.0.

Nastavení pracovního adresáře:

- WORKDIR /app
- Nastavuje pracovní adresář uvnitř kontejneru na /app.

Vystaví porty:

- EXPOSE 80 a EXPOSE 443
- Vystaví porty 80 a 443, aby umožnil přístup k aplikaci zvenčí.

Fáze sestavení:

- FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
- Tato fáze používá k sestavení aplikace obraz sady .NET SDK pro .NET 7.0.

Kopírování zdrojového kódu:

- COPY ["E.Shop.API/E.Shop.API.csproj", "E.Shop.API/"]
- Zkopíruje soubor projektu do pracovního adresáře kontejneru.

Obnovení závislostí:

- RUN dotnet restore "E.Shop.API/E.Shop.API.csproj“
- Obnoví balíčky NuGet uvedené v souboru projektu.

Kopírování zdrojového kódu:

- Zkopíruje celý zdrojový kód do pracovního adresáře kontejneru.

Nastavení pracovního adresáře:

- WORKDIR "/src/E.Shop.API"
- Nastaví pracovní adresář na adresář obsahující projekt API.

Sestavení aplikace:

- RUN dotnet build "E.Shop.API.csproj" -c Release -o /app/build
- Sestaví aplikaci .NET v konfiguraci Release a vypíše artefakty sestavení do adresáře /app/build.

Fáze publikování:

- FROM build AS publish
- Tato fáze definuje novou fázi sestavení s názvem publish, která se používá k publikování aplikace.

Zveřejnění aplikace:

- RUN dotnet publish "E.Shop.API.csproj" -c Release -o /app/publish
- Zveřejní aplikaci .NET v konfiguraci Release a vypíše zveřejněné artefakty do adresáře /app/publish.

Závěrečná fáze:

- FROM base AS final
- Tato etapa využívá dříve definovanou základní etapu.

Nastavení pracovního adresáře:

- Nastaví pracovní adresář na /app.

Kopírování publikovaných artefaktů:

- COPY --from=publish /app/publish
- Zkopíruje publikované artefakty z fáze publish do aktuálního adresáře v konečné fázi.

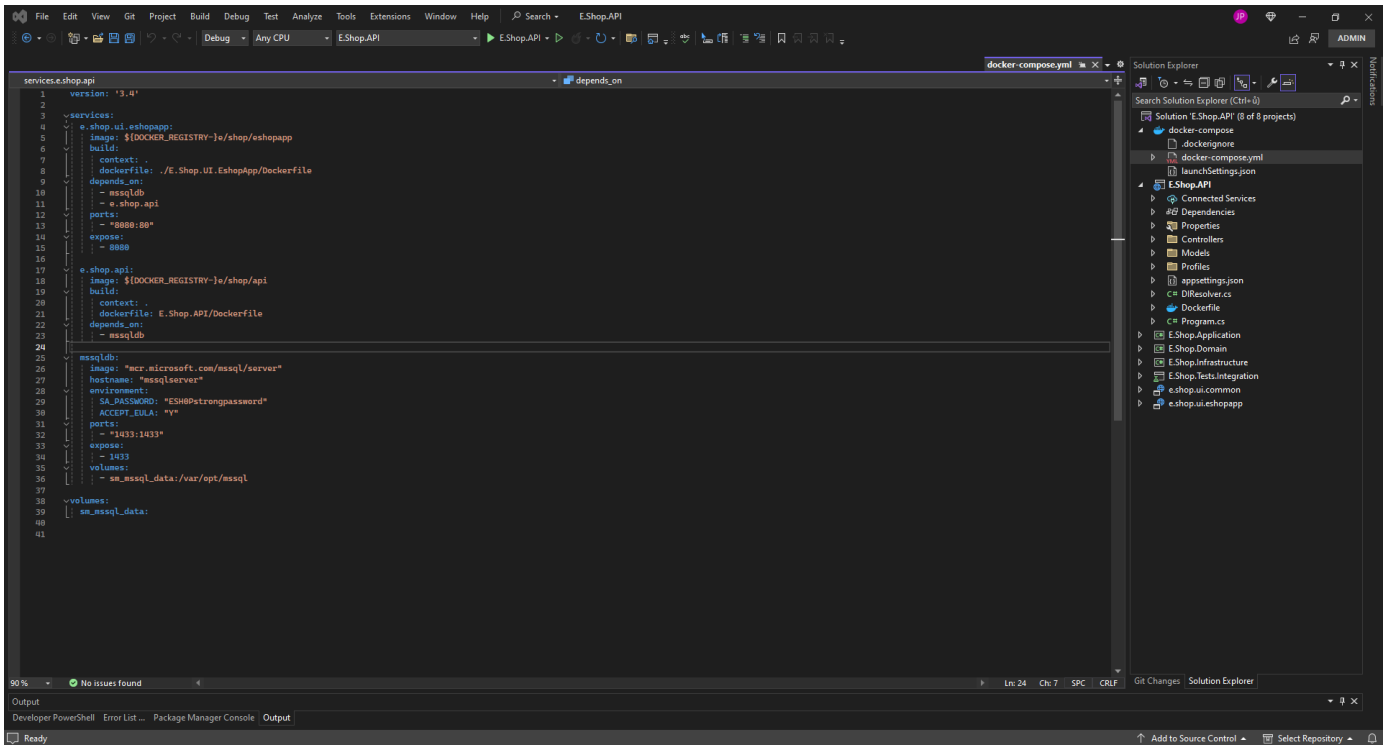
Nastavení vstupního bodu:

- ENTRYPOINT ["dotnet", "E.Shop.API.dll"]
- Určuje příkaz, který se spustí při spuštění kontejneru. V tomto případě spustí aplikaci .NET.

5.2 Konfigurace docker-compose

Docker Compose, který se používá k definování a spouštění aplikací Docker s více kontejnery.

Docker Compose nastavuje prostředí pro provoz aplikace e-shopu, která se skládá z komponenty uživatelského rozhraní, komponenty API a databáze Microsoft SQL Server. Kontejnery jsou orchestrovány tak, aby spolupracovaly, a mají správně nakonfigurované závislosti.



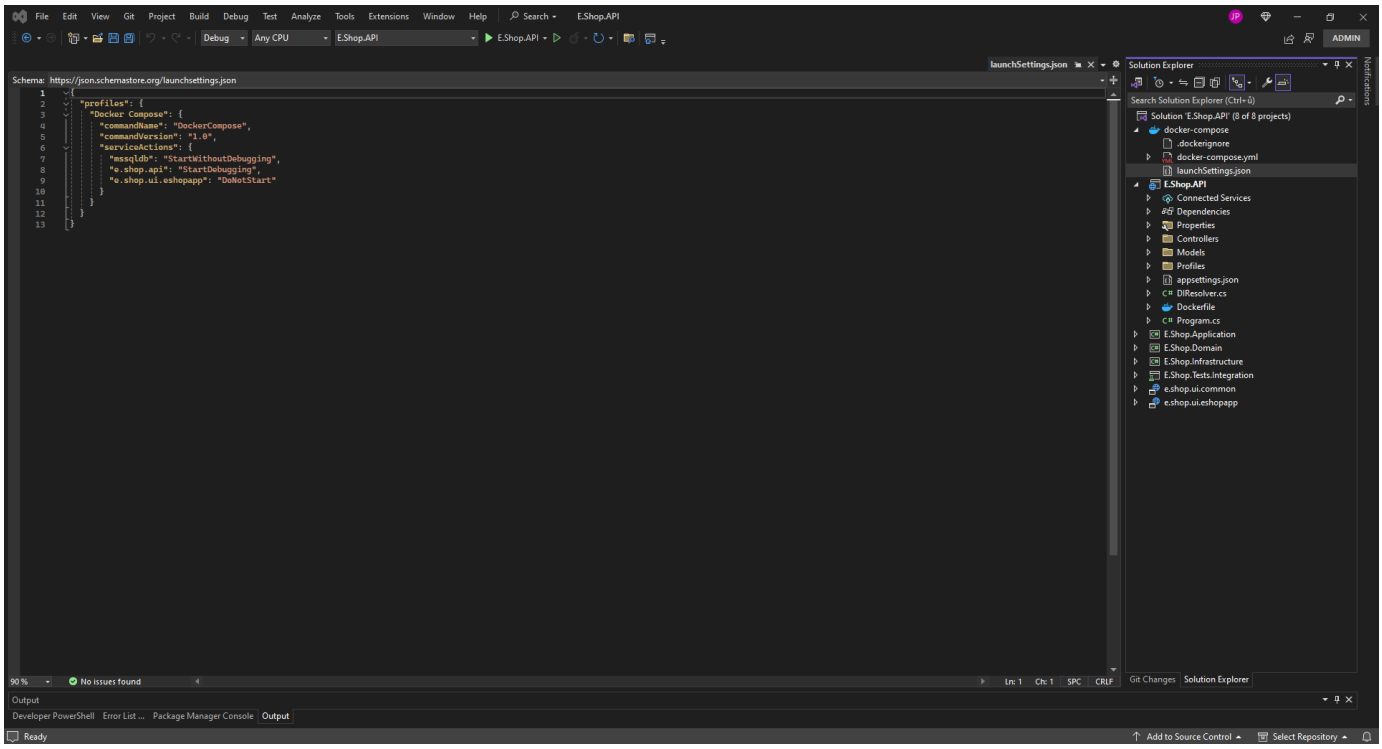
Obrázek 19 Rozdělení a nastavení docker-compose

Verze: To označuje použitou verzi syntaxe souboru Docker Compose. V tomto případě je to verze 3.4.

Služby: Tato část definuje jednotlivé služby.

Svazky: Tato část definuje pojmenované svazky, které mohou služby používat. V tomto případě je zde jeden svazek s názvem sm_mssql_data, který je připojen pro uchovávání dat databáze MSSQL.

Tento JSON definuje profily pro správu služeb Docker Compose ve vývojovém prostředí. Tato struktura JSON je součástí konfiguračního systému pro zefektivnění správy služeb Docker Compose, který určuje různé chování každé služby na základě požadavků vývoje.



Obrázek 20 docker-compose launchSetings.json

Profily: Tato část zahrnuje konfigurace nebo nastavení pro správu služeb Docker Compose.

Docker Compose: Určuje název profilu, který je pravděpodobně přizpůsoben pro použití Docker Compose.

commandName a commandVersion: Tato pole označují název a verzi používaného příkazu nebo nástroje.

serviceActions: Zde jsou definovány konkrétní akce pro jednotlivé služby v rámci nastavení Docker Compose.

- **mssqldb:** Tato služba má příkaz "StartWithoutDebugging", což znamená, že by měla být spuštěna bez připojení ladicího programu.
- **e.shop.api:** Tato služba má příkaz "StartDebugging", což znamená, že bude spuštěna s povoleným laděním.
- **e.shop.ui.eshopapp:** Tato služba má pokyn "DoNotStart", což naznačuje, že by neměla být automaticky spuštěna.

6 TVORBA UŽIVATELSKÉHO ROZHRANÍ

V dnešní digitální éře, kde uživatelská zkušenost hraje klíčovou roli v úspěchu softwarových aplikací, je nezbytné věnovat zvláštní pozornost tvorbě uživatelského rozhraní. V této kapitole se zaměřím na proces vytváření uživatelského rozhraní pro pomocí podprojektu UI.EshopApp, který slouží jako kritický bod kontaktu mezi uživatelem a aplikací.

Při tvorbě uživatelského rozhraní jsem se rozhodl využít moderní technologie a nástroje, které umožňují efektivní vývoj a zajišťují elegantní a responzivní design. Hlavními nástroji, které budu používat, jsou Bootstrap, TypeScript a Vue.js.

Bootstrap poskytuje širokou škálu komponent a stylů, které usnadňují vytváření konzistentního a atraktivního uživatelského rozhraní. S využitím jeho flexibilního grid systému a komponent jako tlačítka, formuláře a navigační panely budu schopny rychle a snadno vytvořit moderní vzhled pro naši aplikaci.

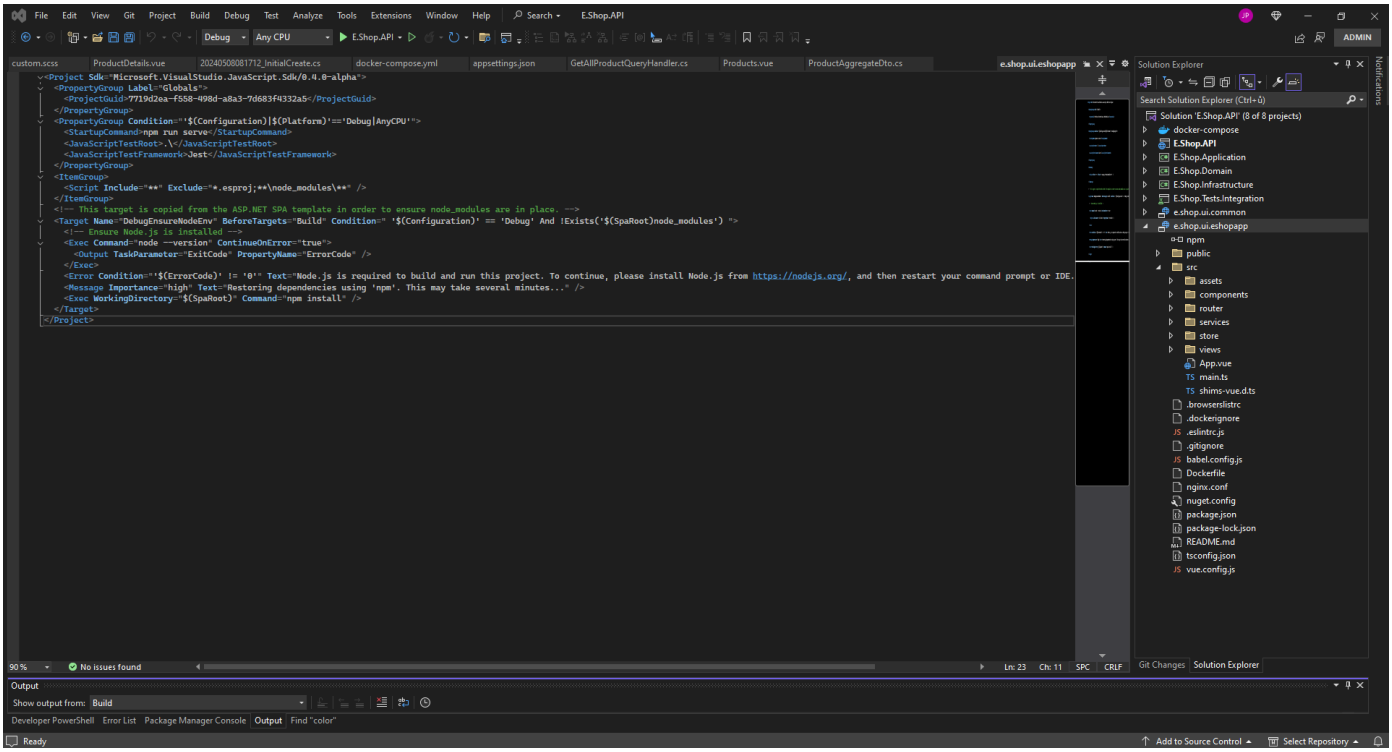
Typescript zase umožní psát čistý a dobře strukturovaný kód s lepší typovou kontrolou a možností refaktorizace. Tím zajistím robustnost a snadnější správu kódu během celého vývoje.

Framework Vue.js, s jeho deklarativním přístupem k psaní komponent a efektivním systémem reaktivních dat, poskytne ideální prostředí pro vytváření interaktivních a dynamických uživatelských rozhraní. Jeho jednoduchá syntaxe a široká komunitní podpora zajišťují efektivní a plynulý vývoj.

Během této kapitoly detailně popíši proces vytváření uživatelského rozhraní pomocí těchto technologií. Bude zde zobrazen způsob jak využít jejich sílu k vytvoření moderní a funkční aplikace. S představením těchto klíčových nástrojů budou prozkoumany další aspekty vývoje uživatelských rozhraní a vytvářet aplikace, které osloví a pohltí uživatele.

6.1 Uskupení UI.EshopApp

Projekt EshopApp nám slouží jakožto projekt pro uživatelské rozhraní (UI).



Obrázek 21 Infrastruktůra a nastavení EshopApp

Assets: V této složce se nachazí veškeré obrázky, ikony a také se zde nachází public styles.

Components: Ve složce komponenty se nachazí veškeré komponenty našeho UI (modals, spinner, atd.).

Router: V této složce najdeme soubor, který se nám stará o routering, neboli navigaci v aplikaci.

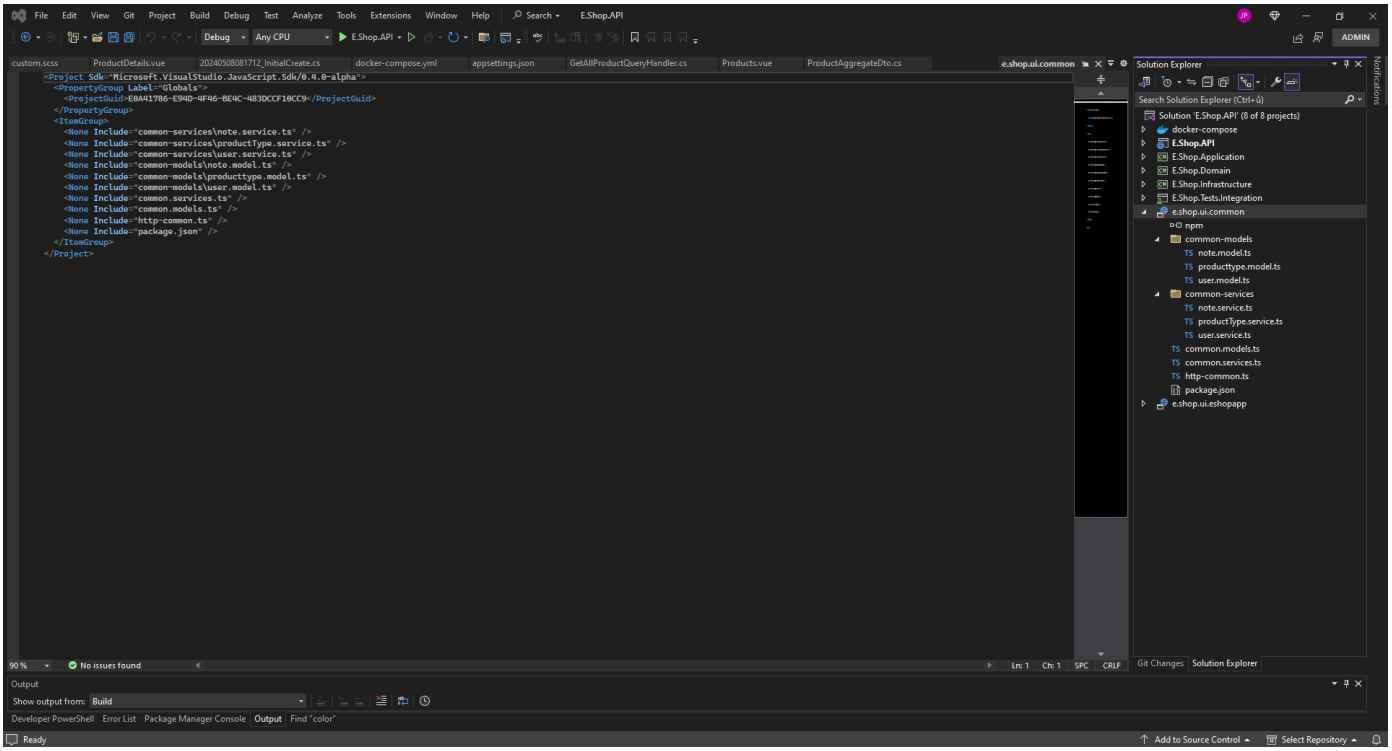
Services: Zde se ukrývaji metody na zpracování ruzných požadavků jako jsou například: getProductAsync, addProductAsync atd.

Views: Náhledy jednotlivých karet jsou k nalezení v této složce.

App.vue: Tento soubor slouží jako základní kámen. Zde se definuje prvotní zobrazení a také navbar.

6.2 Uskupení UI.Common

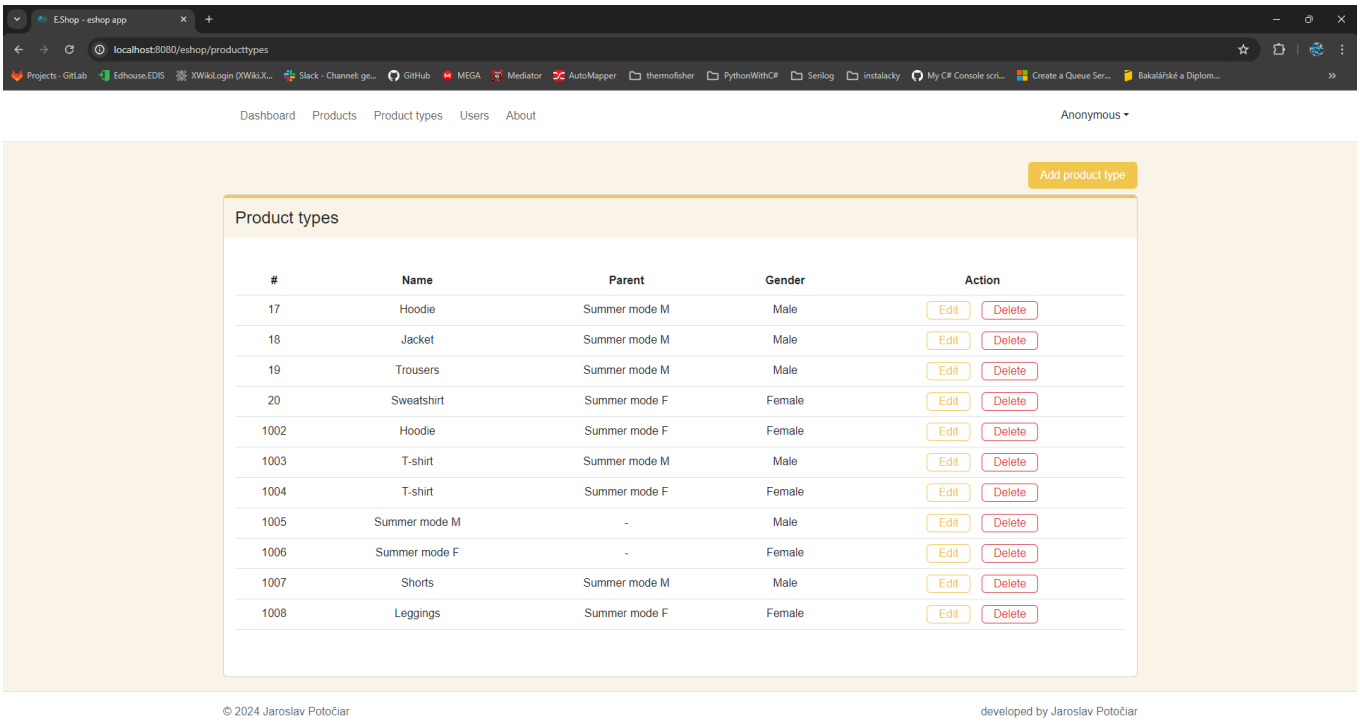
Poslední z představovaných projektu se zaměřuje na sdilení určitých modelů a servisek. Zdá se mi jako velmi užitečné, myslet do budoucna a vytvořit si malý základ k jednoduchosti sdílení určítých části kódu (možnost jednoduššího vytvoření SkladApp).



Obrázek 22 Infrastruktůra a nastavení Common

6.3 Konečné zobrazení uživatelského rozhraní

Pro ukázku konečného vzhledu uživatelského rozhraní jsem si vybral kartu Product types. Na této kartě lze najít název, rodiče, pohlaví a možné akce s jednotlivými typy produktů.



Obrázek 23 Uživatelské rozhraní vzhled

ZÁVĚR

Tato bakalářská práce byla vypracována pro komplexní analýzu různých technologií, které jsou klíčové pro vývoj moderní webové aplikace, se zaměřením zejména na vytvoření platformy internetového obchodu (e-shopu). Mezi podrobně zkoumané technologie patří .NET, Docker, TypeScript, Bootstrap a Vue.js. Každá z těchto technologií nabízí jedinečné výhody a funkce, které přispívají k celkové efektivitě, škálovatelnosti a uživatelskému komfortu webové aplikace.

Jako robustní framework pro vytváření škálovatelných a vysoce výkonných webových aplikací se ukazuje .NET. Jeho všestrannost, rozsáhlá podpora knihoven a kompatibilita s více jazyky z něj činí preferovanou volbu pro vývojáře, kteří hledají spolehlivost a efektivitu svých projektů.

Docker přináší revoluci v procesu nasazení tím, že poskytuje řešení kontejnerizace, které zajišťuje konzistenci v různých prostředích. Jeho odlehčená povaha, přenositelnost a škálovatelnost z něj činí nepostradatelný nástroj pro vývoj moderního softwaru, který vývojářům umožňuje zefektivnit proces nasazení a zvýšit produktivitu.

TypeScript díky statickému typování a pokročilým nástrojům zvyšuje kvalitu vývoje tím, že zachycuje chyby již v rané fázi vývoje a umožňuje lepší organizaci a udržovatelnost kódu. Jeho bezproblémová integrace s jazykem JavaScript a široké přijetí v komunitě vývojářů z něj činí vynikající volbu pro vytváření komplexních webových aplikací.

Bootstrap zjednodušuje proces vytváření responzivních a vizuálně atraktivních uživatelských rozhraní. Jeho rozsáhlá sbírka předpřipravených komponent a systém mřížek umožňuje vývojářům rychle vytvářet prototypy a přizpůsobovat rozvržení webu a zároveň zajišťuje kompatibilitu napříč prohlížeči a mobilní odezvu.

Vue.js, známý svou jednoduchostí, flexibilitou a reaktivitou, se ukazuje jako výkonný front-endový framework pro vytváření dynamických a interaktivních uživatelských rozhraní. Jeho architektura založená na komponentách, rozsáhlá dokumentace a živý ekosystém zásuvných modulů a knihoven z něj činí ideální volbu pro vývoj moderních jednostránkových aplikací.

V praktické části této bakalářské práce jsou poznatky získané z teoretické analýzy použity při vývoji webové aplikace (e-shopu), která využívá silné stránky těchto technologií. Integrací prostředí .NET pro backendovou logiku, Dockeru pro kontejnerizaci, TypeScriptu pro vylepšený vývojový workflow, Bootstrapu pro responzivní design a Vue.js pro

dynamické interakce s uživateli je výsledný e-shop příkladem synergie dosažené strategickou kombinací těchto technologií.

Celkově tato práce podtrhuje důležitost výběru správné sady technologií na základě konkrétních požadavků a cílů webové aplikace. Využitím možností technologií .NET, Docker, TypeScript, Bootstrap a Vue.js mohou vývojáři vytvářet moderní, efektivní a škálovatelné webové aplikace, které splňují vyvíjející se potřeby uživatelů v dnešním digitálním prostředí.

SEZNAM POUŽITÉ LITERATURY

[1] HALL, Gary McLean. Adaptive Code via C#. [Online]. Redmond: Microsoft Press, 2014. ISBN 978-0-7356-8448-6.

[2] POULTON, Nigel. Docker Deep Dive: 2023 Edition. UK: Nielson Book Services, 2023. ISBN 978-1916585256

[3] DENNIS, Andrew K.; SCHWARTZ, Michael; BULLINGTON-MCGUIRE, Richard. Docker for Developers. UK: Packt Publishing Ltd. 2020. ISBN 9781789539486

[4] PRICE, Mark J. C# 11 and .NET 7. UK: Packt Publishing Ltd, 2022. ISBN 978-1803237800.

[5] FREEMAN, Adam. Pro ASP.NET Core 7. South Carolina: Manning, 2023. ISBN 978-1633437821.

[6] Milanović, Milan. "A Brief Walk Through .NET Ecosystem." [online]. [cit. 2024-01-23]. Milanović.org, Dostupné z: https://milan.milanovic.org/post/a-brief-walk-through-net-ecosystem/.

[7] Ganvir, Anshul. "Introduction to Docker." [online]. [cit. 2024-01-21]. Medium, Dostupné z: https://medium.com/@anshulganvir/introduction-to-docker-337b9d09a079.

[8] "Docker Compose overview." [online]. [cit. 2024-02-02]. Docker Documentation, Dostupné z: https://docs.docker.com/compose/intro/features-uses/.

[9] "Docker multi-platform builds." [online]. [cit. 2024-01-19]. Docker Documentation, Dostupné z: https://docs.docker.com/build/building/multi-platform/.

[10] "ASP.NET Core Web API." [online]. [cit. 2024-03-24]. Microsoft Learn, Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website.

[11] "Progressive Web Apps." [online]. [cit. 2024-02-11]. Mozilla Developer Network, Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[12] "Jednostránková webová aplikace (SPA)." [online]. [cit. 2024-01-17]. RascasOne, Dostupné z: https://www.rascasone.com/cs/blog/jednostrankova-webova-aplikace-spa.

[13] "HTML 5." [online]. [cit. 2024-01-26]. Jak psát web, Dostupné z: https://www.jakpsatweb.cz/html/html-5.html.

[14] "Difference between CSS and CSS3." [online]. [cit. 2024-02-22]. GeeksforGeeks, Dostupné z: https://www.geeksforgeeks.org/difference-between-css-and-css3/.

[15] "What's New in CSS 3." [online]. [cit. 2024-01-21]. Medium, Beginners Guide to Mobile Web Development, Dostupné z: https://medium.com/beginners-guide-to-mobile-web-development/whats-new-in-css-3-dcd7fa6122e1.

[16] "ECMAScript® 2015 Language Specification." [online]. [cit. 2024-01-21]. ECMA International, Dostupné z: https://262.ecma-international.org/6.0/.

[17] "Introduction." [online]. [cit. 2024-03-14]. Vue.js Documentation, Dostupné z: https://vuejs.org/guide/introduction.html.

[18] "Úvod do bezpečnosti webových aplikací." [online]. [cit. 2024-05-01]. ITnetwork.cz, Dostupné z: https://www.itnetwork.cz/php/bezpecnost/uvod-do-bezpecnosti-webovych-aplikaci.

[19] "Použití systému stavů." [online]. [cit. 2024-04-09]. Microsoft Docs, Dostupné z: https://learn.microsoft.com/cs-cz/mem/configmgr/core/servers/manage/use-status-system.

[20] "Šablona pro monitorování dostupnosti webových aplikací." [online]. [cit. 2024-04-24]. Microsoft Docs, Dostupné z: https://learn.microsoft.com/cs-cz/system-center/scom/web-application-availability-monitoring-template?view=sc-om-2022.

[21] "Mobilní aplikace vs. responzivní a mobilní web." [online]. [cit. 2024-02-14]. GoodRequest, Dostupné z: https://www.goodrequest.com/cs/blog/mobilni-aplikace-vs-responzivni-a-mobilni-web.

[22] "Jenkins Documentation." [online]. [cit. 2024-04-25]. Jenkins, Dostupné z: https://www.jenkins.io/doc/.

[23] "Travis CI Enterprise." [online]. [cit. 2024-04-25]. Travis CI, Dostupné z: https://www.travis-ci.com/product-enterprise/.

[24] "Travis CI Cloud." [online]. [cit. 2024-02-23]. Travis CI, Dostupné z: https://www.travis-ci.com/product-cloud/.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MSF	Malé a Střední Firmy
PWA	Progressive Web Apps
SPA	Single Page Applications
AJAX	Asynchronní JavaScript a XML
HTML	Hypertext Markup Language
API	Application Programming Interface
CSS	Cascading Style Sheets
XML	Extensible Markup Language
ECMAScript	European Computer Manufacturers Association Script
VCS	Version controll systems
CI	Continuous Integration
CI/CD	Continuous Integration and Continuous Delivery/Continuous Deployment
CLR	Common Language Runtime
WPF	Windows Presentation Foundation
UI	User Interface
DOM	Document Object Model
XSS	Cross-site scripting
CSRF	Cross-Site Request Forgery
CSP	Content Security Policy
HTTPS	Hypertext Transfer Protocol Secure
HSTS	HTTP Strict Transport Security

SEZNAM OBRÁZKŮ

Obrázek 1 Use Case Diagram 18
Obrázek 2. Microsoft .NET Framework [6] 19
Obrázek 3. Docker[7] 20
Obrázek 4. Vue.js[17] 24
Obrázek 5 Přidání API 31
Obrázek 6 Konfigurace Program.cs 32
Obrázek 7 Appsettings.json 33
Obrázek 8 launchSettings.json 34
Obrázek 9 Dependency injection resolver 36
Obrázek 10 Vzhled kontroleru 37
Obrázek 11 Vzhled modelu 38
Obrázek 12 Vzhled profilu 39
Obrázek 13 Infrastruktůra a nastavení Domain projektu 40
Obrázek 14 Infrastruktůra a nastavení Application projektu 41
Obrázek 15 Rozdělení a obsah projektu Infrastructure 42
Obrázek 16 Provedení seedovaní dat 43
Obrázek 17 Rozdělení a nastavení Intergration Tests rojektu 44
Obrázek 18 Nazorná ukázka Dockerfile 45
Obrázek 19 Rozdělení a nastavení docker-compose 48
Obrázek 20 docker-compose launchSetings.json 49
Obrázek 21 Infrastruktůra a nastavení EshopApp 51
Obrázek 22 Infrastruktůra a nastavení Common 52
Obrázek 23 Uživatelské rozhraní vzhled 53

SEZNAM PŘÍLOH

Příloha P I: CD s webovou aplikací a elektronickou verzí bakalářské práce