Wichterlovo gymnázium, Ostrava-Poruba, příspěvková organizace



Marek Tyl

Systém e-shopu – webová aplikace

*Maturitní práce*

Ostrava

Vedoucí práce: Ing. David Hrbáč

Čestné prohlášení

Prohlašuji, že jsem tuto práci vypracoval/a samostatně s použitím literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací.

V Ostravě, dne 22.4.2022

Podpis: ………………………………………….

Obsah

[Úvod 4](#_Toc101473526)

[1 Použité Jazyky 5](#_Toc101473527)

[1.1 HTML 5](#_Toc101473528)

[1.2 CSS 6](#_Toc101473529)

[1.2.1 Bootstrap 6](#_Toc101473530)

[1.3 C# 7](#_Toc101473531)

[1.3.1 Blazor 7](#_Toc101473532)

[2 Použité programy 9](#_Toc101473533)

[2.1 Visual Studio 9](#_Toc101473534)

[2.2 SQL Server a SQL Server Managment Studio 9](#_Toc101473535)

[3 Uživatelské rozhraní 10](#_Toc101473536)

[3.1 Produkty 10](#_Toc101473537)

[3.2 Vyhledávání 11](#_Toc101473538)

[3.3 Košík 11](#_Toc101473539)

[4 Funkce 12](#_Toc101473540)

[4.1 Databáze 12](#_Toc101473541)

[4.2 Vyhledávání 16](#_Toc101473542)

[4.3 Zobrazení produktů 17](#_Toc101473543)

[4.4 Košík 18](#_Toc101473544)

[Závěr 20](#_Toc101473545)

[Použitá Literatura 21](#_Toc101473546)

[Seznam Obrázků 22](#_Toc101473547)

# Úvod

Jako téma maturitní práci do informatiky jsem si vybral webovou aplikaci e-shopu. Hlavním cílem webové aplikace je umožnit návštěvníkovi prohlédnout si dostupné produkty a následně si je objednat.

Kód celé práce je dostupný na:

GitHub - tylmarek1/TM\_Eshop\_v1: Maturita - Eshop - v1. GitHub: Where the world builds software · GitHub [online]. Copyright © 2022 GitHub, Inc. [cit. 21.04.2022]. Dostupné z: <https://github.com/tylmarek1/TM_Eshop_v1>

Návod na spuštění aplikace je dostupný na:

Návod k TM Eshopu. Návod k TM Eshopu [online]. Dostupné z: <http://tylmarek.euweb.cz/>

# Použité Jazyky

## HTML

Hypertext Markup Language neboli zkráceně HTML je značkovací jazyk používaný hlavně ve webových stránkách. Jak už lze odvodit z názvu, od normálního textu se liší tím, že je provázaný hypertextovými odkazy. HTML by se dalo považovat za základní stavební kostku naprosté většiny webových stránek.

V HTML se používají různé předdefinované tagy, které mají různé významy. Tento způsob byl převzat z SGML (Standard Generalized Markup Language). Tagy jsou většinou ve dvojicích, jeden otevírající a jeden uzavírající, mezi nimi je pak text, který vkládáme na stránky. Uzavírající tag se liší pouze lomítkem před názvem. Tagy taky můžou být vnořené, například v tagu <p></p>, který představuje odstavec, může být vnořen tag <i></i>, který změní styl textu na kurzívu.

V rámci HTML jsou i pro koncového uživatele „neviditelné“ tagy, které obsahují informace pro prohlížeč.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <!DOCTYPE html>  <html lang="cs">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  </head>  <body>  <nav>  <li>  <ul>Tohle je list.</ul>  <ul>Tohle je list.</ul>  <ul>Tohle je list.</ul>  </li>  </nav>  <div>  <h1>Tohle je nadpis.</h1>  <p>Tohle je odstavec.</p>  <p>V tomhle textu jsou <i>tahle slova napsaná kurzívou</i>, a tahle už zase ne.</p>  <p>V tomhle textu jsou <b>tahle slova napsaná tučně</b>, a tahle už zase ne.</p>  <p>V tomhle textu jsou <a href="cesta">tahle slova hyperlinkovým odkazem</a>, a tahle už zase ne.</p>  <img src="" alt="">  </div></body></html> |

## CSS

CSS neboli Kaskádové styly je jazyk používaný k úpravě vzhledu HTML stránek. Původně se vzhled stránky upravoval pomocí jazyka HTML, pak se ale z důvodu přehlednosti vyvinul jazyk CSS, aby HTML jazyk popisoval pouze obsah a strukturu a jazyk CSS popisoval vzhled stránky.

V CSS souboru pak můžeme navazovat na jména tagu, třídy tagu, nebo identifikátoru tagu. Lze také použít „interaktivní“ element jako například :hover, který se spustí až když nad objektem projedeme kurzorem.

V CSS také můžeme používat několik různých jednotek velikosti, podle toho, které se zrovna danou věc hodí. Patří mezi ně: px, em, rem, vh, vw, % a další. Jednotka px už se v dnešní době tak moc nepoužívá, protože je statická a uživateli se tak stránka může špatně zobrazit.

|  |  |  |
| --- | --- | --- |
| 1  2  3 | <div></div>  <div class="class1"></div>  <div id="id1"></div> | |
|  |  | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | | div {  **display**: flex;  flex-**direction**: column;  **justify**-**content**: space-around;  align-items: **center**;  }  **.class1** {  **width**: **500px**;  **border**: **solid** red **2px**;  **padding**: **10px**;  }  **#id1** {  **max-height**: **30**vh;  }  div**:hover** {  **outline**: **solid** green **1**rem;  transform: rotate(**90**);  } | |

### Bootstrap

Bootstrap je předepsaná knihovna v jazyce LESS, kterou můžeme připojit k našemu projektu a pak ji pomocí specifických tříd používat v HTML souboru pro úpravu vzhledu. Bootstrap byl vyvinut pro podporu konzistence vzhledu stránek. Před přestavením Bootstrapu se pro design webu využívalo mnoho různých knihoven, což vedlo k jeho namáhavé údržbě. Proto se vývojáři Bootstrapu rozhodli vytvořit univerzální nástroj pro sjednocení kódu na svých rozhraních. Jak můžeme vidět v ukázkovém kódu, při využívání Bootstrapu upravujeme HTML jazyk pomocí předdefinovaných tříd.



## C#

C# je vysokoúrovňový programovací jazyk. Je odvozen od jazyků C++ a Java a byl vyvinut společností Microsoft. Jazyk běží ve vývojovém prostředí .NET. Jazyk má několik základních vlastností:

* Jazyk je objektově orientovaný.
* Obsahuje jenom jednoduchou dědičnost, ale násobnou implementaci rozhraní.
* Obsahuje události.
* Obsahuje garbage collector, takže správa paměti je automatická.
* Je bezpečnější než C++.

V následné ukázce kódu vidíme nejzákladnější funkci Hello World zapsanou v jazyce C#.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **using** **System**;  **namespace** **HelloWorld**  {  **class** **Program**  {  **static** **void** **Main**(**string**[] args)  {  Console.WriteLine("Hello World!");  }  }  } |

### Blazor

Blazor je framework pro C#, který se používá na vytváření webových stránek. Jsou dva hlavní typy Blazoru: Blazor Server a Blazor WebAssembly. Blazor Server, podobně jako PHP, běží na serveru a klientovi se potom posílá pouze výsledek. Na rozdíl od toho Blazor WebAssembly funguje podobně jako JavaScript, kde se klientovi posílá celá aplikace, která pak běží u klienta. V mém projektu jsem použil ASP.NET core hosted Blazor WebAssembly, která má i serverovou část. V následné ukázce vidíme, jak jednoduše lze HTML a C# provázat.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <p>count: @count</p>  <button class="btn btn-primary" @onclick="PlusCount">Plus jeden.<button>  @code {  **private** **int** count = **0**;  **private** **void** **PlusCount**()  {  count++;  }  } |

# Použité programy

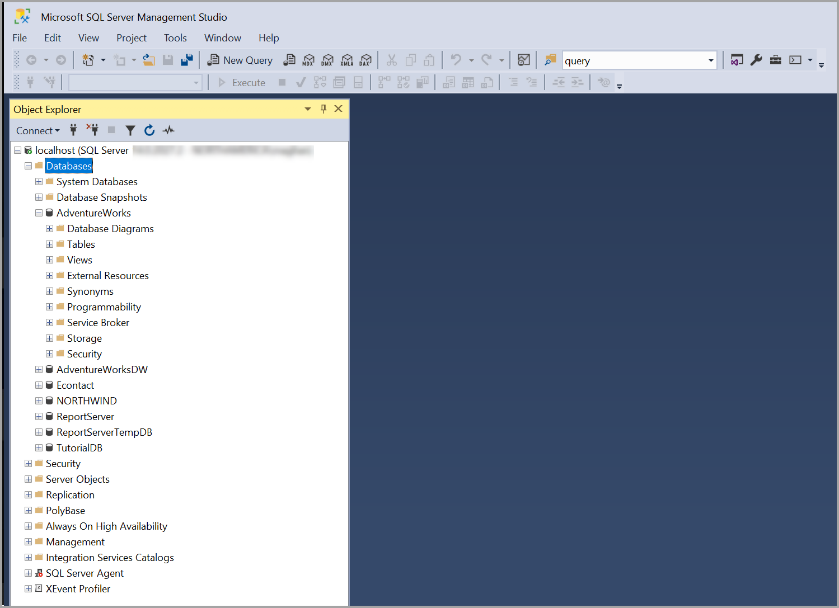
## Visual Studio

A screenshot of a computer

Description automatically generated with medium confidenceVisual Studio je vývojové prostředí vydané firmou Microsoft. Program poskytuje širokou škálu možností pro vývoj aplikací v platformě .NET, ale zároveň i v jiných platformách. Aplikace má spoustu chytrých vymožeností jako predikce kódu, nebo zvýrazňování chyb a nabídku možných řešení.

Obrázek 1: Visual Studio

## SQL Server a SQL Server Managment Studio

V mém projektu jsem používal SQL Server na zprostředkování databáze pro můj web a SQL Server Managment Studio na úpravu databáze.

Obrázek 2: SQL Server Managment Studio

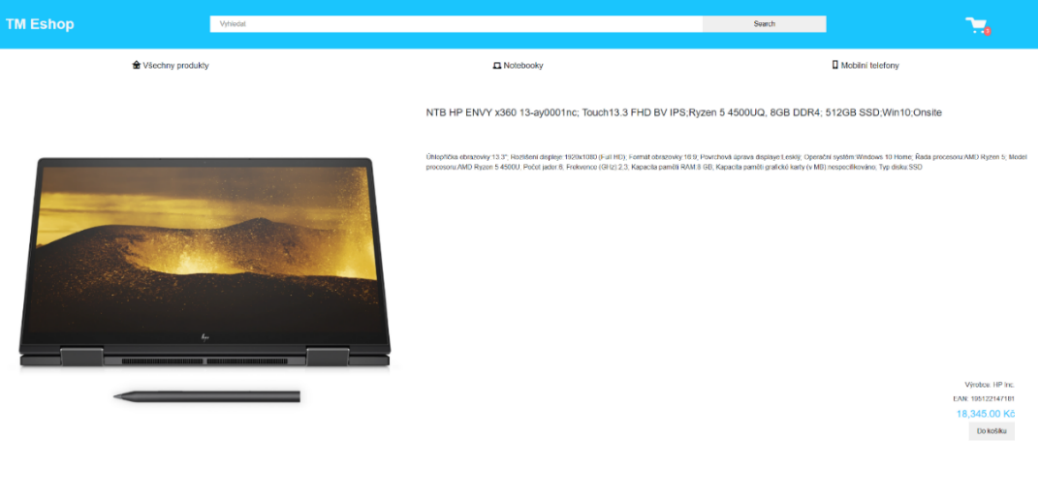
# Uživatelské rozhraní

## Produkty

Graphical user interface

Description automatically generatedKaždý zákazník má možnost zobrazit si seznam produktů. Ať už si chce zobrazit všechny produkty najednou, nebo po kategoriích. V seznamu produktů jsou produkty seřazeny v blocích v mřížce, která má čtyři sloupce. U každého produktu vidí název produktu, obrázek a cenu. Každému produktu se pro lepší orientaci trochu změní pozadí, když po něm zákazník přejíždí kurzorem.

Obrázek 3: Seznam produktů

Poté, co zákazník na produkt klikne, dostane se na stránku s detaily daného produktu, kde si daný produkt může přidat do košíku.

Obrázek 4: Detail produktu

## Vyhledávání

Zákazník má vždy možnost použít v záhlaví vyhledávací lištu, aby si našel produkt podle jeho jména. Po zadání klíčového slova se zákazníkovi zobrazí seznam produktů, které mají shodu klíčového slova a názvu produktu.

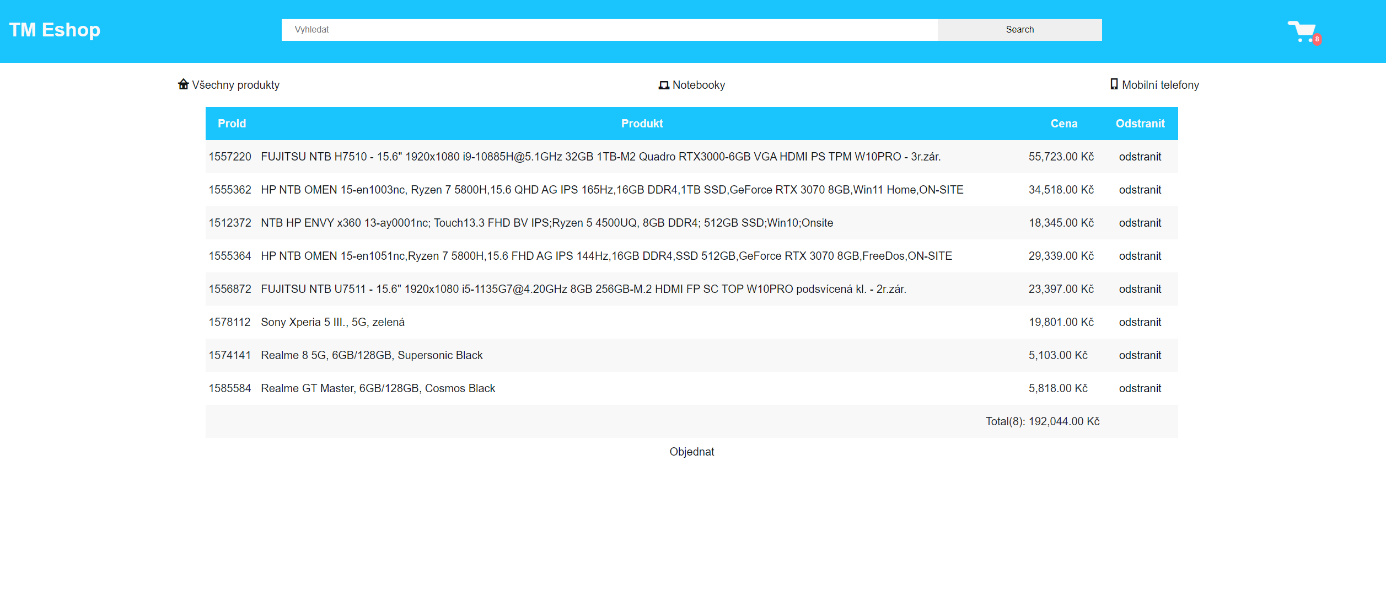
Obrázek 5: Vyhledávací lišta

## Košík

Pokud si zákazník na stránce s detailem produktu přidá produkt do košíku, produkt se vloží do košíku a aktualizuje se indikátor u ikony košíku, aby zákazník viděl, kolik má produktů v košíku.

Obrázek 6: Ikona košíku

Po kliknutí na ikonu se zákazník dostane do košíku, kde uvidí tabulku produktů, které jsou v košíku. Produkty, které nechce může zákazník z košíku odstranit pomocí tlačítka odstranit. A pokud zákazníkovi všechno vyhovuje, může kliknout na tlačítko objednat, pak už stačí pouze zadat údaje na které to chceme poslat a objednávka se odešle. Košík se vyprázdní a zákazník může nakupovat dále.



Obrázek 7: Košík

# Funkce

## Databáze

V mém projektu používám databázi k uložení informací o produktech, kategoriích a objednávkách. Abychom dostali data z databáze na webovou stránku je potřeba hned několik souborů a operací. Na první pohled to vypadá jako spousta náhodných souborů, ale jakmile člověk pochopí, k čemu se který soubor používá, stává se z toho lehká skládanka. Ten postup pak lze s malými obměnami opakovat a vytváření aplikace se stává vcelku jednoduché.

První věc, kterou potřebujeme udělat, je zapsat si připojení do databáze v souboru appsettings.json.

|  |  |
| --- | --- |
| 1  2  3 | "ConnectionStrings": {  "DefaultConnection": "server=localhost\\sqlexpress;database=EshopDB\_v1;trusted\_connection=true;"  } |

Tohle je připojení do databáze, která nemá žádné přihlašovací údaje. Pokud bychom používali databázi, která vyžaduje přihlášení, musíme přidat další atributy.

Další věc, kterou potřebujeme vytvořit, je model dat, které budeme používat. U dat jdou nastavit různé parametry. Jdou nastavit různé typy, ať už stringy, čísla, desetinná čísla, čas a další. Taky si jde do modelu nastavit parametry jako: povinné, emailová adresa, maximální délku, nebo minimální délku. Datové modely vytváříme ve sdílené části projektu, aby byly přístupné jak v serverové, tak v klientové části projektu.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **namespace** **TM\_Eshop\_v1.Shared**{  **public** **class** **Buyer**{  [Key]  **public** **int** BuyerId { **get**; **set**; }  [Required]  **public** **string** Name { **get**; **set**; } = **string**.Empty;  [Required, EmailAddress]  **public** **string** Mail { **get**; **set**; } = **string**.Empty;  [Required]  **public** **string** AdressCity { **get**; **set**; } = **string**.Empty;  [Required]  **public** **string** AdressStreet { **get**; **set**; } = **string**.Empty;}} |

U modelu dat lze vytvořit i relace mezi jednotlivými modely, podle kterých se potom na sebe mohou navazovat, když voláme jednotlivé modely dat ve funkcích.

Poté pomocí Entity Frameworku vytvoříme příkazem PM> dotnet ef migrations add DbUpdate1 migraci, která podle datových modelů automaticky vytvoří zápis do databáze, následně použijeme příkaz PM> dotnet ef database update vytvoříme v databázi příslušné tabulky.

Dále je potřeba vytvořit datový kontext, abychom věděli, co z databáze nahrát. V mém projektu mám v databázi zapsané tři různé tabulky, a to jsou tabulka produktů, tabulka kategorií a tabulka objednávek.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **namespace** **TM\_Eshop\_v1.Server.Data.DataContext**  {  **public** **class** **DataContext** : DbContext  {  **public** **DataContext**(DbContextOptions<DataContext> options) : **base**(options) { }  **public** DbSet<Product>? Products { **get**; **set**; }  **public** DbSet<Category>? Categories { **get**; **set**; }  **public** DbSet<Order>? Orders { **get**; **set**; }  }  } |

Když máme nastavený model i připojení, můžeme vytvořit serverový servis, ve kterém bude logika zpracování dat. Každý servis má dva soubory, rozhraní a soubor s logikou. Nejdříve si v rozhraní vypíšeme funkce, které bude servis potřebovat a pak v druhém souboru definuje, co dané funkce dělají. Obvykle se rozhraní pojmenuje stejně jako soubor s logikou a přidáme před název „I“, aby program rozpoznal, že je to rozhraní. „I“ pochází z anglického interface. Například: ProductService.cs a IProductService.cs. Použití rozhraní zvyšuje bezpečnost kódu a zároveň umožní vícenásobné dědictví, které jinak není možné v C#. V příkladu uvedeném níže je funkce na získání jednoho produktu z databáze podle jeho identifikátoru.

Nejdříve si zavedeme neznámou datového kontextu a poté vztahujeme z databáze podle datového modelu a přidáváme filtr na identifikátor, následně už stačí jen vzít v potaz možnost prázdné odpovědi a máme funkci hotovou.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **namespace** **TM\_Eshop\_v1.Server.Services.ProductService**  {  **public** **class** **ProductService** : IProductServiceInterface  {  **private** **readonly** DataContext \_context;  **public** **ProductService**(DataContext context)  {  \_context = context;  }  **public** **async** Task<ServiceResponce<Product>> GetProductAsync(**int** proId)  {  **var** responce = **new** ServiceResponce<Product>();  **var** product = **await** \_context.Products.FindAsync(proId);  **if** (product == **null**)  {  responce.Success = **false**;  responce.Message = "Wrong path or maybe error.";  }  **else**  {  responce.Data = product;  }  **return** responce;  } |

Posledním krokem na serverové části je vytvoření kontroléru, který vytvoří, nebo dostává Http žádost. Například, pokud získáváme data, používáme metodu [HttpGet] a pokud ukládáme data, používáme metodu [HttpPost]. Existují dva způsoby, jak vytvářet kontrolery. Jedním způsobem je do kontroléru zahrnout i logiku a už není potřeba žádného servisu. Tento způsob, ale není doporučovaný, protože se kontrolér nestává znovupoužitelný a my bychom jich museli vytvářet zbytečně pořád více a zpomalovali bychom naši aplikaci. Proto je lepší všechnu logiku vytvořit v servisu a pak postavit kontrolér, který pouze vytváří http žádosti.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **namespace** **TM\_Eshop\_v1.Server.Controllers**  {  [Route("api/[controller]")]  [ApiController]  **public** **class** **ProductController** : ControllerBase  {  **private** **readonly** IProductServiceInterface \_productService;  **public** **ProductController**(IProductServiceInterface productService)  {  \_productService = productService;  }  [HttpGet("getallproducts")]  **public** **async** Task<ActionResult<ServiceResponce<List<Product>>>> GetProducts()  {  **var** result = **await** \_productService.GetProductsAsync();  **return** **Ok**(result);  }  [HttpGet("{proId}")]  **public** **async** Task<ActionResult<ServiceResponce<Product>>> GetProduct(**int** proId)  {  **var** result = **await** \_productService.GetProductAsync(proId);  **return** **Ok**(result);  }  ... |

Tímhle krokem jsme dokončili serverovou část, všechno máme připravené a můžeme se přesunout na klientovou část.

## Vyhledávání

Pro vyhledávání si vytvoříme dva soubory, klientový servis, který se bude dovolávat HTTP žádosti ze serverového kontroléru, a komponent, který se bude zobrazovat na webu.

Jako první vytvoříme servis, který je zase stvořený ze dvou souborů, a to z rozhraní a soubor s logikou. V mém případě dostávám HTTP žádost s listem produktů, které budou vyfiltrované pomocí klíčového slova, až tuto funkci v zavoláme v komponentu.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **namespace** **TM\_Eshop\_v1.Client.Services.ProductService**{  **public** **class** **ProductService** : IProductServiceInterface  {  **private** **readonly** HttpClient \_http;  **public** **ProductService**(HttpClient http) {\_http = http;}  **public** List<Product> Products { **get**; **set**; } = **new** List<Product>();  **public** **string** Message { **get**; **set**; } = "Produkty se načítají...";  **public** **event** Action ProductChanged;  **public** **async** Task **Search**(**string** searchString) {  **var** result = **await** \_http.GetFromJsonAsync<ServiceResponce<List<Product>>>  ($"api/product/search/{searchString}");  **if** (result != **null** && result.Data != **null**)  Products = result.Data;  **if** (Products.Count == **0**)  Message = "Tady nic není...";  ProductChanged?.Invoke();} |

Dále pak vytvoříme komponent Search.razor, ve kterém bude HTML část a C# část. V HTML části vytvoříme <input> tag do kterého bude uživatel zapisovat klíčové slovo, které bude chtít vyhledat a tlačítko, jehož kliknutím potvrdí svůj input a spustí danou funkci. Ještě to můžeme obalit do blokového elementu <div>, aby se nám stránka lépe graficky upravovala. V C# části potom zavedeme funkci, která se spouští tlačítkem. Jediné, co funkce musí dělat, je zavolat si příslušný servis a dosadit do něho klíčové slovíčko.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @inject NavigationManager NavigationManager  @inject IProductServiceInterface ProductService  <div class="searchdiv">  <div class="searchinputdiv">  <input type="search" id="searchbarid" placeholder="Vyhledat" @bind-value="searchString" @bind-value:event="oninput" @onkeyup="Keyboard" @ref="userSearch" />  </div>  <div class="buttondiv">  <button @onclick="SearchPro">Search</button>  </div>  </div> |
| 12  13  14  15  16  17  18  19 | @code {  **private** **string** searchString = **string**.Empty;  **protected** ElementReference userSearch;  **public** **void** **SearchPro**(){  **if** (searchString == **string**.Empty) {  NavigationManager.NavigateTo($"products");}  **else**{ NavigationManager.NavigateTo($"products/search/{searchString}"); }}} |

Pak už jenom jen vložíme <Search /> do našeho kódu a máme hotovo.

## Zobrazení produktů

U zobrazení produktů postupujeme podobně jako u vyhledávání. V tomto případě však buď žádáme o všechny produkty, v případě, že zobrazujeme seznam všech produktů, anebo je filtrujeme podle kategorie nebo identifikátoru, abych dostali buď produkty z jedné kategorie, nebo pouze jeden produkt.

Jako první si zase vytvoříme servis.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **namespace** **TM\_Eshop\_v1.Client.Services.ProductService** {  **public** **class** **ProductService** : IProductServiceInterface {  **private** **readonly** HttpClient \_http;  **public** **ProductService**(HttpClient http){ \_http = http;}  **public** List<Product> Products { **get**; **set**; } = **new** List<Product>();  **public** **string** Message { **get**; **set**; } = "Produkty se načítají...";  **public** **event** Action ProductChanged;  **public** **async** Task<ServiceResponce<Product>> GetProduct(**int** proId) {  **var** result = **await** \_http.GetFromJsonAsync<ServiceResponce<Product>>  ($"api/product/{proId}");  **return** result;}  **public** **async** Task **GetProducts**(**string?** catUrl = **null**) {  **var** results = catUrl == **null**?  **await** \_http.GetFromJsonAsync<ServiceResponce<List<Product>>>  ("api/product/getallproducts"):  **await** \_http.GetFromJsonAsync<ServiceResponce<List<Product>>>  ($"api/product/category/{catUrl}");  **if** (results != **null** && results.Data != **null**)  Products = results.Data;  ProductChanged.Invoke();} |

Potom to můžeme hezky pomocí foreach() cyklu vyobrazit v komponentu ProductList.razor. A poté vložit na webovky jako <ProductList />.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @foreach (var product in ProductService.Products)  {  <a href="product/@product.ProId">  <h4>@product.Name</h4>  <img src="@product.ImageUrl">  <p>@product.Price Kč</p>  </a>  } |

## Košík

S košíkem začneme pracovat v momentu, kdy se zákazník rozhodne něco přidat do košíku. V rámci košíku jsou tři hlavní operace. Přidání produktu do košíku, zobrazení produktu v košíku a vytvoření objednávky.

Když přidáváme produkty do košíku, vezmeme si identifikátor produktu a zapíšeme ho do localStorage prohlížeče. To je úložiště v prohlížeči, které zůstane zachováno i po restartu prohlížeče, zákazník proto bude mít produkty v prohlížeči i když další den znovu navštíví e-shop.

Toho dosáhneme pomocí funkce v servisu.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **public** **async** Task **AddToCart**(CartItem cartItem)  {  **var** cart = **await** \_localStorage.GetItemAsync<List<CartItem>>("cart");  **if** (cart == **null**) { cart = **new** List<CartItem>(); }  cart.Add(cartItem);  **await** \_localStorage.SetItemAsync("cart", cart);  OnChange.Invoke();  } |

Poté na zobrazení produktů v košíku budeme potřebovat dvě funkce. Jednu, která získá list identifikátorů z localStorage a druhou, která vytáhne z databáze produkty se shodujícími se identifikátory a všechny je pomocí funkce foreach() zobrazí v tabulce. Pokud budeme chtít produkty z košíku vymazat, budeme potřebovat ještě třetí funkci, která z localStorage vymaže určitý identifikátor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **public** **async** Task<List<CartResponce>> GetCartProducts()  {  **var** cartItems = **await** \_localStorage.GetItemAsync<List<CartItem>>("cart");  **var** responce = **await** \_http.PostAsJsonAsync("api/cart/products", cartItems);  **var** cartProducts = **await** responce.Content.ReadFromJsonAsync<ServiceResponce<List<CartResponce>>>();  **return** cartProducts.Data;  } |

Na vytvoření objednávky, budeme potřebovat získat data o zákazníkovi z formuláře, identifikátory produktů z localStorage a aktuální čas, který získáme pomocí jednoduché funkce. Poté to všechno odeslat do databáze. A nakonec vymazat localStorage.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **public** **async** Task **CreateOrder**(Order order)  {  **var** result = **await** \_http.PostAsJsonAsync("api/order/createorder", order);  **var** responce = **await** result.Content.ReadFromJsonAsync<List<Order>>();  Orders = responce;  **await** \_localStorage.ClearAsync();  \_navigationManager.NavigateTo("", forceLoad: **true**);  } |

# Závěr

Maturitní práce se mi povedla. Na stánkách funguje prohlížení produktů, vyhledávání produktů, i funkce košíku a objednávek. Do budoucnosti by aplikaci šlo vylepšit přidáním uživatelů a plateb, nebo taky tmavé verze stránky pro pohodlnější sledování během nočních hodin. Stránky jsou kompatibilní se všemi nejpoužívanějšími prohlížeči a zároveň mají jednoduchý a přehledný vzhled, aby měl zákazník příjemný zážitek. Během vytváření projektu jsem si velmi prohloubil své znalosti C# a určitě s ním budu chtít programovat i v budoucnosti.

# Použitá Literatura

1. Hypertext Markup Language. *Wikipedie* [online]. [cit. 2022-04-21]. Dostupné z: <https://cs.wikipedia.org/wiki/Hypertext_Markup_Language>
2. Kaskádové styly. *Wikipedie* [online]. [cit. 2022-04-21]. Dostupné z: <https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly>
3. Bootstrap. *Wikipedie* [online]. [cit. 2022-04-21]. Dostupné z: <https://cs.wikipedia.org/wiki/Bootstrap>
4. C Sharp. *Wikipedie* [online]. [cit. 2022-04-21]. Dostupné z: <https://cs.wikipedia.org/wiki/C_Sharp>
5. C Sharp. *VSB* [online]. [cit. 2022-04-21]. Dostupné z: <http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf>
6. Visual Studio. *Wikipedia* [online]. [cit. 2022-04-21]. Dostupné z: <https://cs.wikipedia.org/wiki/Microsoft_Visual_Studio>
7. CHAN, Jamie. C#: Learn C# in One Day and Learn It Well. C# for Beginners with Hands-on Project. Amazon [ebook]. Kindle Store: LCF Publishing, 2015, 20.10.2015 [cit. 2022-04-21].
8. ENGSTRÖM, Jimmy. Web Development with Blazor: A hands-on guide for .NET developers to build interactive UIs with C#. Amazon [kniha]. India: Packt Publishing Limited, 2021, 18.06.2021 [cit. 2022-04-21].

# Seznam Obrázků

[Obrázek 1: Visual Studio 9](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460081)

[Obrázek 2: SQL Server Managment Studio 9](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460082)

[Obrázek 3: Seznam produktů 10](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460083)

[Obrázek 4: Detail produktu 10](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460084)

[Obrázek 5: Vyhledávací lišta 11](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460085)

[Obrázek 6: Ikona košíku 11](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460086)

[Obrázek 7: Košík 11](https://d.docs.live.net/56c7ebb3e7e95439/C%5eN/_zavprace/DokumentaceTyl.docx#_Toc101460087)