Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

# DIPLOMOVÁ PRÁCE

Praktické porovnání frameworku Laravel a ASP.NET Core architektury

2022

Bc. Vladimír Vykoupil

Vedoucí práce:
RNDr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

**Bibliografické údaje**

| | |
|---|---|
| Autor: | Bc. Vladimír Vykoupil |
| Název práce: | Praktické porovnání frameworku Laravel a ASP.NET Core architektury |
| Typ práce: | diplomová práce |
| Pracoviště: | Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci |
| Rok obhajoby: | 2022 |
| Studijní obor: | Aplikovaná informatika, prezenční forma |
| Vedoucí práce: | RNDr. Martin Trnečka, Ph.D. |
| Počet stran: | 43 |
| Přílohy: | 1 CD/DVD |
| Jazyk práce: | český |

**Bibliograhic info**

| | |
|---|---|
| Author: | Bc. Vladimír Vykoupil |
| Title: | Practical comparison of the Laravel framework and ASP.NET Core architecture |
| Thesis type: | master thesis |
| Department: | Department of Computer Science, Faculty of Science, Palacký University Olomouc |
| Year of defense: | 2022 |
| Study field: | Computer Science, full-time form |
| Supervisor: | RNDr. Martin Trnečka, Ph.D. |
| Page count: | 43 |
| Supplements: | 1 CD/DVD |
| Thesis language: | Czech |

## Anotace

*Cílem této diplomové práce je porovnat framework Laravel a ASP.NET Core architekturu při implementaci plnohodnotného internetového obchodu. Porovnání bude založeno zejména na mých dojmech a na tom, jak se v daných technologiích pracuje.*

## Synopsis

*The aim of this thesis is to compare the Laravel framework and ASP.NET Core architecture in the implementation of a full-fledged online store. The comparison will be based mainly on my impressions and how the technologies work.*

**Klíčová slova:** závěrečná práce; Laravel; ASP.NET Core; framework; porovnání

**Keywords:** thesis; Laravel; ASP.NET Core; framework; comparison

Tímto bych chtěl poděkovat vedoucímu bakalárské práce panu RNDr. Martinu Trnečkovi, Ph.D. za odborné vedení při vypracovávání této práce. Chtěl bych také poděkovat mé rodině za podporu, projevenou během studia.

*Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.*

datum odevzdání práce podpis autora

# Obsah

# Seznam obrázků



# Seznam tabulek



# Seznam zdrojových kódů

# 1 Úvod

V této diplomové práci se budu zabývat porovnáním dvou technologií pro tvorbu webových stránek. Budu srovnávat PHP framework Laravel s frameworkem ASP.NET Core postaveným na platformě .NET. Aby bylo možné provést objektivní srovnání, budu vytvářet pomocí každé z technologií internetový obchod zaměřený na elektroniku. Funkcionalita a vzhled webových stránek bude totožná.

## 1.1 Předpoklady

Text této práce je zejména určen čtenáři, kterého zajímá vývoj webových technologií a nemá s danými frameworky zkušenosti. Tato práce může být také určena pro vývojáře, který se rozhoduje, jakou technologii zvolí pro implementaci webu. Předpokládají se základní znalosti objektového programování a dotazovacího jazyka SQL. Vhodná je také znalost jazyka PHP a C#.

## 1.2 Důvod výběru tématu

Důvodem výběru tématu práce bylo, že jsem si chtěl rozšířit obzory v tvorbě webových stránek. Předtím jsem vytvořil stránky pomocí čistého PHP a zajímalo mě, jak se ulehčí práce při použití frameworku. Framework Laravel jsem zvolil díky jeho velké popularitě. Jako srovnání jsem zvolil ASP.NET Core, kvůli tvorbě v jazyku C#.

## 1.3 Očekávaný výstup

Jako výsledek práce předpokládám, že se mi podaří zhodnotit v jakých aspektech je jedna technologie lepší než druhá. Za sekundární výstup práce lze považovat souhrn informací pro začátek vývoje v daných technologiích.

# 2 Použité technologie

V této kapitole popíši použité technologie, které jsem použil při vypracování této diplomové práce.

## 2.1 Webové technologie

V této sekci jsou rozebrány použité webové technologie.

### 2.1.1 HTML (Hypertext Markup Language)

Značkovací jazyk HTML se používá pro tvorbu webových stránek. Díky tomuto jazyku se vytváří základní obsahová kostra webové stránky.[1] HTML dává význam jednotlivým částem stránky pomocí HTML značek, které se nazývají tagy.[2] Vytvořili jej Tim Berners-Lee a Robert Cailliau v roce 1990. Poslední specifikací HTML je HTML Living Standard, který je neustále upravován. Poslední úprava proběhla v roce 2021.

### 2.1.2 CSS (Cascading Style Sheets)

CSS neboli kaskádové styly je technologií umožňující vizualizovat webovou stránku. Díky kaskádovým stylům dostává HTML kód stránky svůj vzhled.[3] Jazyk byl vytvořen organizací W3C v roce 1996. Vývoj tohoto jazyka už nadále neprobíhá pomocí verzí, ale pomocí zveřejnění kolekce modulů. Vše co přišlo po verzi 2.1, je označováno jako CSS 3.[4]

### 2.1.3 JavaScript

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk. Vytvořil jej Brendan Eich v roce 1995. Z pohledu syntaxe patří do rodiny programovacích jazyků C, ale z pohledu sémantiky je velmi odlišný.[5] Tento jazyk se zejména používá na straně klienta ke zlepšení uživatelského prožitku při návštěvě webové stránky.[6] JavaScript byl standartizován v roce 1997 asociací ECMA a hojně se využívá dodnes.

### 2.1.4 jQuery

jQuery je knihovna pro JavaScript, která ulehčuje interakci mezi HTML a JavaScriptem.[7] Vznikla v roce 2006 díky Johnu Resigovi. Jedná se o svobodný software pod licencí MIT. Mezi největší výhody této knihovny patří metoda `ajax()`, kterou jsem využil k posílání asynchroních požadavků na server.

### 2.1.5 MVC architektura

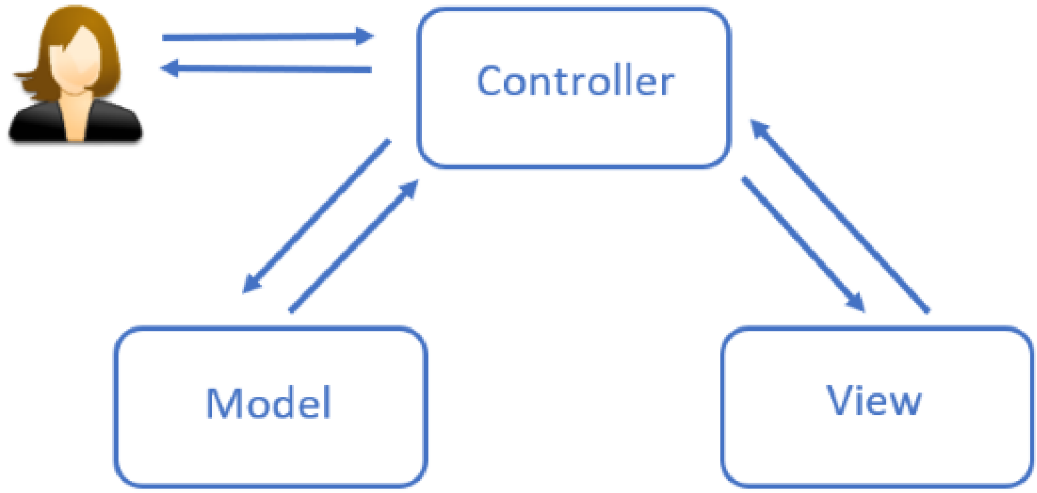
Architektura MVC (model–view–controller) je populární návrhový vzor, používaný zejména při vytváření webových stránek. Hlavní myšlenkou je oddělení řídící

logiky od uživatelského rozhraní a datového modelu. Jedná se o řešení problému tzv. špagetového kódu, kde je v jednom souboru logika aplikace a zároveň části vykreslování výstupu.[8] Díky tomu je kód přehledný a je snazší jej upravovat. Struktura MVC se dělí na:

**Model** Do této vrstvy patří zejména data a k nim vázané výpočty. Mohou zde být zastoupeny i pravidla pro validaci dat z formulářů. Model není závislý na ostatních vrstvách.

**View (pohled/zobrazení)** Úkolem této vrstvy je zobrazení dat zpracovaných modelem a uživatelského rozhraní. View se tedy stará o zobrazení výstupu uživateli a nezajímá jej, odkud data přišla.

**Controller (řadič)** Jedná se o řídící vrstvu, která reaguje na akce provedené uživatelem. Přijme data od uživatele, nechá je zpracovat modelem a poté pověří pohled zobrazením výsledné stránky.




Obrázek 1: MVC architektura

## 2.2 Programovací jazyky

Zde jsou popsány programovací jazyky, ve kterých se pracuje v porovnávaných frameworcích. Laravel je založen na PHP a v ASP.NET Core se pracuje v C#.

### 2.2.1 PHP

Programovací jazyk PHP (Hypertext Preprocessor) je jedním z nejrozšířenějších programovacích jazyků používaných ke tvorbě webových stránek. Při vytváření

webů se používá na straně serveru. Jedná se o dynamicky typovaný skriptovací jazyk, jehož největší výhodou je platformová nezávislost.[10] Jazyk PHP vznikl v roce 1995 a poslední stabilní verzí je verze 8.1 z listopadu 2021.

### 2.2.2 C#

Jazyk C# je vvysokoúrovňový objektově orientovaný staticky typovaný programovací jazyk vytvořený firmou Microsoft. Syntaxe C# je založena na jazycích Java a C++. Tento programovací jazyk vznikl v roce 2000 a poslední představenou verzí byla verze 10 v roce 2021.[11]

## 2.3 Frameworky

V této sekci naleznete informace o použitých frameworcích. Framework můžeme chápat jako abstrakci, poskytující základní funkcionalitu, která se mění uživatelsky psaným kódem. Framework je univerzálním a opakovaně použitelným softwarovým prostředím a obsahuje další podpůrné programy a knihovny, které do sebe zapadají.[12] Cílem je řešení typických problémů, aby se vývojář mohl zabývat hlavně svým úkolem.[19]

### 2.3.1 Bootstrap

Bootstrap je jednoduchý CSS framework, používaný pro rychlou tvorbu responzivního rozhraní webových stránek. Je velmi oblíbený zejména díky Grid systému, který umožňuje rychlé pozicování prvků na stránce.[13] Bootstrap se původně jmenoval Twitter Blueprint a byl zamýšlen pro udržení konzistence vzhledu Twitteru, ale nakonec byl vydán jako open source projekt a využívá se dodnes. Nejnovější verzí Boostrapu je verze 5, která byla vydána 5. května 2021.

### 2.3.2 Laravel

Laravel je jedním z nejrozšířenějších PHP frameworků, který ulehčuje tvorbu webových aplikací. Je prezentován jako webový framework s elegantní a výstižnou syntaxí.[14] V roce 2012 jej vytvořil Taylor Otwell. Jedná se o open source projekt, který je poskytován zdarma pod licencí MIT. Je zde využita architektura Model-View-Controller (MVC), která zabezpečuje větší pořádek ve zdrojových souborech. Laravel může být využit jak na malé webové aplikace, tak i na velké projekty.[16]

### 2.3.3 ASP.NET Core

Jedná se o modulární framework postavený na platformě .NET, který je využíván k tvorbě webových aplikací. Byl představen v roce 2016 jako přepracování ASP.NET, který bylo možno provozovat jen pod Windows.[17] ASP.NET Core je jednoduše řečeno sada knihoven pro tworbu webových alikací v programovacím jazyce C#, které obsahují hotová řešení velkého počtu základních věcí. Tento

framework lze použít jak na malé projekty, tak na obrovské webové aplikace.[19] Oproti PHP frameworkům není moc rozšířený, ale je oblíben v korporátní sféře. Od svých předchůdců se liší tím, že se na server nahrává i samotný framework, díky čemuž můžeme svou aplikaci nahrát i tam, kde není prostředí Windows.

## 2.4 Použité programy

V této sekci kapitoly jsou popsány vývojová prostředí a programy, které jsem použil.

### 2.4.1 PhpStorm

PhpStorm je multiplatformní vývojové prostředí, vytvářeno společností JetBrains, které je naprogramováno v jazyce Java. Je využíváno zejména k práci v jazyce PHP a jeho frameworcích, ale nabízí podporu i dalších jazyků vázaných na tvorbu webových stránek. Přestože společnost JetBrains sídlí v České republice, je tento produkt pouze v anglickém jazyce. Nejedná se o bezplatný software, ale jsou dostupné bezplatné licence na určitou dobu pro studenty, učitele a instituce.[20]

### 2.4.2 Visual Studio

Jedná se o vývojové prostředí vyvíjené Microsoftem. Lze použít pro vytvoření různých druhů aplikací implementovaných v různých programovacích jazycích. Nejčastěji se v tomto prostředí pracuje s jazyky C/C++, VB.NET a C#.
Visual Studio nabízí mnoho vestavěných nástrojů pro pohodlný vývoj aplikací v podporovaných jazycích.[21]

### 2.4.3 XAMPP

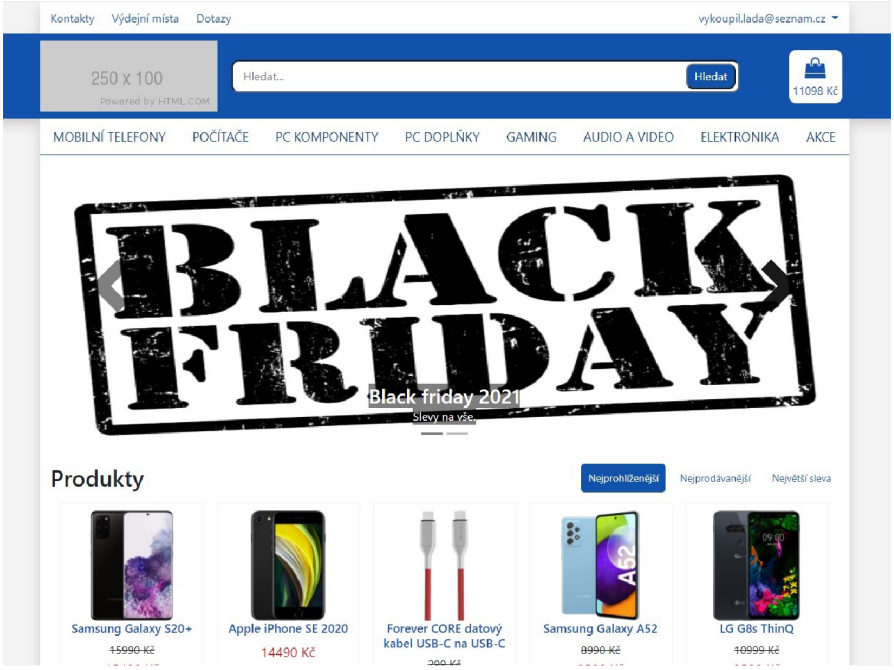
Pod názvem XAMPP je označován multiplatformní softwarový balíček, který obsahuje webový server Apache, databáze MariaDB, mail server a programovací jazyky PHP a Perl. Písmena v názvu XAMPP označují charakteristiku balíčku (Cross-Platform – X, Apache – A, MariaDB – M, PHP – P, Perl – P). Nejčastěji se využívá ke spuštění webů offline na lokálním serveru při vývoji.[22]

# 3 Tvorba internetového obchodu

Jako podklad pro srovnání dvou zmíněných frameworků mi sloužila implementace internetového obchodu zaměřeného na elektroniku. Webové stránky vznikly ve dvou verzích podle použitého frameworku.

## 3.1 Postup tvorby

Jako první věc bylo třeba si rozmyslet vzhled a základní funkcionalitu webu. Poté přišla na řadu tvorba šablony webu pomocí technoligií HTML, CSS a JavaScript. Tvorbu uživatelského rozhraní jsem pojal jako desktop first design (design zaměřen hlavně na počítač a poté přizpůsoben pro mobilní zařízení). Použil jsem také knihovnu Bootstrap, aby se lépe pracovalo na responzivitě webu. Při tvorbě vzhledu stránek jsem si už rozmýšlel podobu databáze, kterou jsem průběžně tvořil pomocí programu phpMyAdmin z balíčku XAMPP. Jakmile jsem měl hrubou podobu stránek, tak jsem se vrhl na tvorbu ve frameworku Laravel. V průběhu tvorby jsem ještě mírně zasahoval do vzhledu stránek. Rozhodl jsem se použít knihovnu jQuery pro zajištění lepšího pocitu při procházení stránek. Při tvorbě v Laravelu rovněž vznikla podoba administrativní části webu. Jakmile jsem měl vytvořen web v Laravelu, tak přišla řada na tvorbu v ASP.NET Core. Ta už probíhala rychleji, protože jsem se mohl odpíchnout od druhé implementace. V této implementaci jsem se zaměřil na to, aby bylo možno použít stejnou databázi pro obě verze. Po dokončení obou implementací jsem zkoumal, jestli mají stejnou funkcionalitu a opravoval chyby.




Obrázek 2: Úvodní stránka

## 3.2 Vzhled webu

Vzhled webu jsem směřoval do kombinace modré a bílé barvy. Chtěl jsem docílit jednoduššího designu. V administrativní části obchodu jsem vycházel ze základního vzhledu knihovny Bootstrap.

## 3.3 Popis funkčnosti

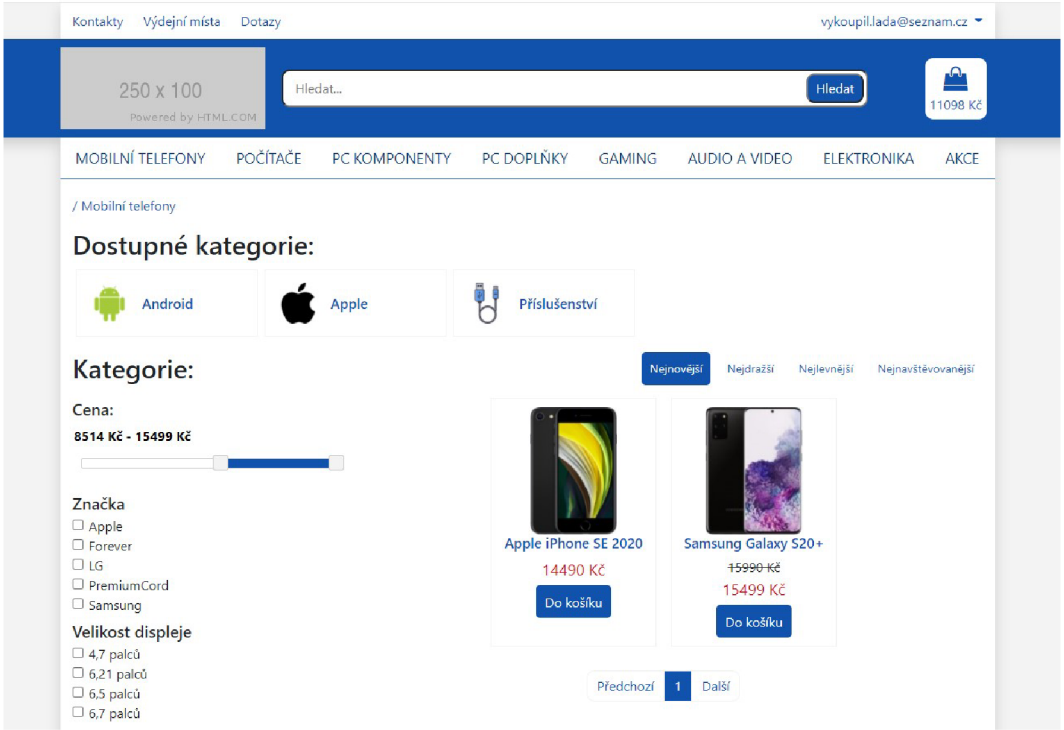
V této sekci popíšu hlavní stránky webu a jejich funkčnost. Text obohatím o obrázky vzhledu stránek.

### 3.3.1 Úvodní stránka

Na úvodní stránce se na viditelném místě zobrazuje filtrování produktů dle jejich oblíbenosti nebo slevy. Dále se zde nachází novinky a články ohledně sdělení o obchodu. V horní části je přítomno vyhledávání produktů podle názvu nebo zvolením podle kategorie.

### 3.3.2 Stránka kategorie produktu

Zde se zobrazují položky konkrétní kategorie a podkategorie. Na levé straně stránky se nachází filtrování, podle kterého jsou zobrazeny produkty. Filtrování je umožněno podle ceny produktu a podle parametrů, které udává daná kategorie.



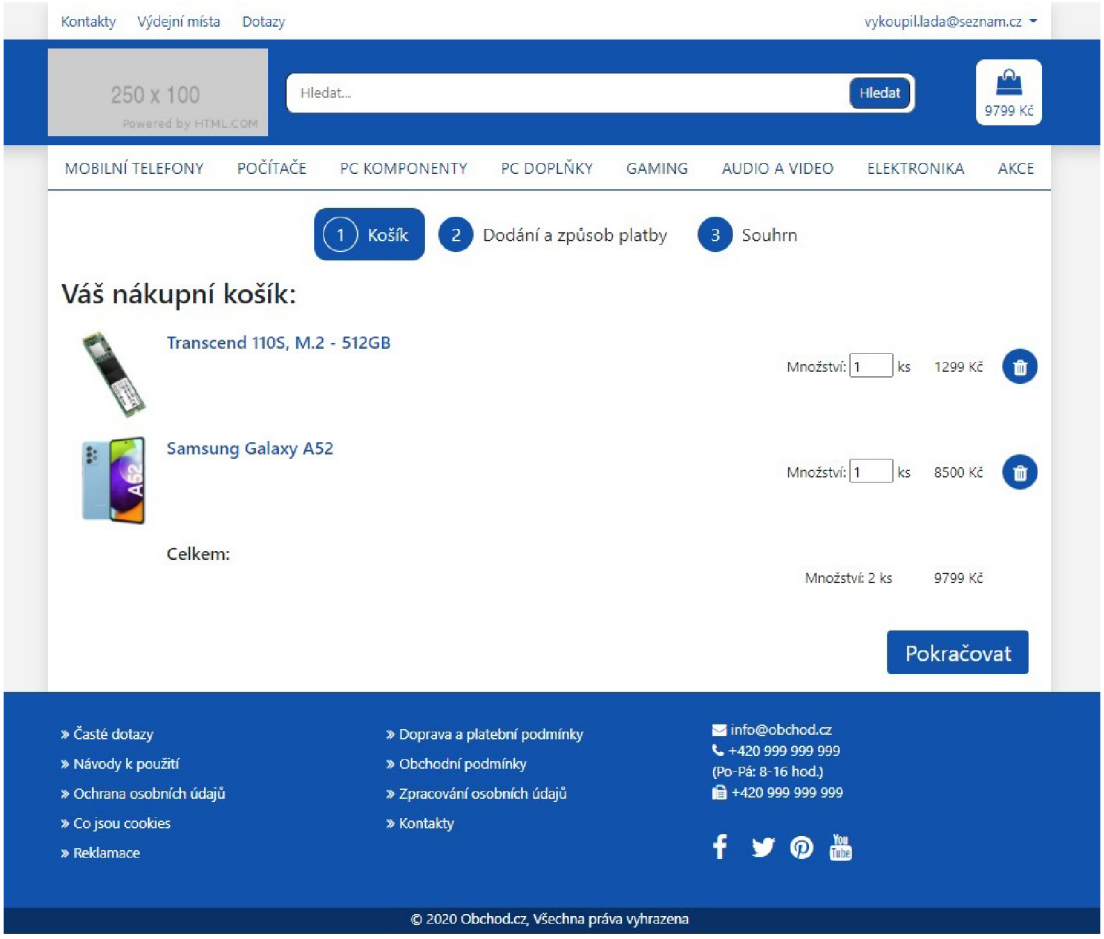

Obrázek 3: Kategorie produktu

### 3.3.3 Stránka produktu

Zde jsou zobrazeny všechny informace o daném produktu v přehledné formě. Zboží můžeme přidat do košíku. Dále je možné produkt ohodnotit, popřípadě napsat recenzi.

### 3.3.4 Košík a dokončení objednávky

Na stránce košíku nalezneme námi vybrané produkty, které chceme zakoupit. Můžeme zde upravit množství nebo je odebrat z košíku. V případě nepřihlášeného uživatele je zde výzva k přihlášení. Po přihlášení se uživatel postupně dostane přes vyplnění adresy a volbu dopravy až k platbě.




Obrázek 4: Stránka košíku

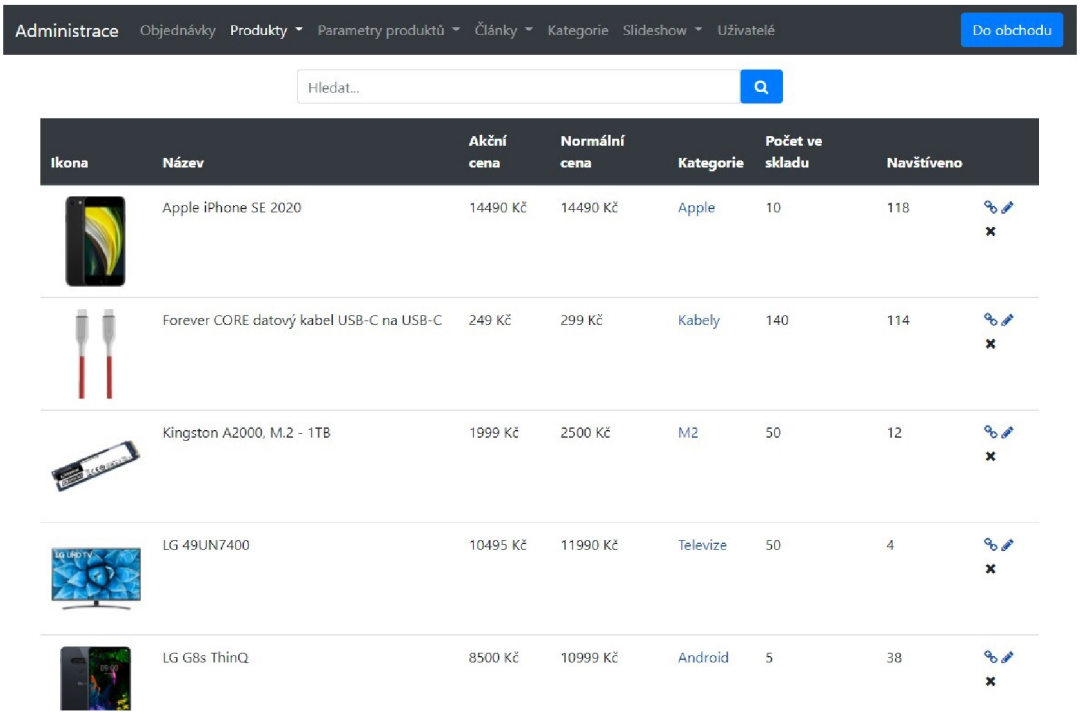
### 3.3.5 Uživatelský prostor

Do uživatelského prostoru se lze dostat kliknutím na jméno uživatele v pravém horním rohu. Jako první je na výběr stránka s názvem Moje objednávky, kde uži-

vatel nalezne seznam všech uskutečněných objednávek. Další na výběr je stránka Moje údaje s nastavením adresy pro rychlé vyplňování při objednávce. Uživatel s administrátorskými právy může zvolit ještě přístup do administrativní části webu.

### 3.3.6 Administrativní část

Zde může přihlášený uživatel s admistrativními právy přidávat nové produkty, kategorie a články. Je možnost také hledat v objednávkách a prohlížet uživatele, kterým je možné upravit práva. Součástí této části webu je také nastavení slideshow z úvodní stránky.




Obrázek 5: Seznam produktů v administraci

## 3.4 Návrh databáze

Databáze obsahuje jednotlivé uživatele s informacemi a přihlašovacími údaji s heslem v zašifrované podobě. Tyto záznamy obsahuje zejména tabulka `users` s doplňkovými informacemi o adresách uživatelů v tabulce `addresses`.

V tabulce `categories` jsou uloženy kategorie, kde pomocí indentifikátorů ve sloupci `id_superior` vzniká struktura podkategorií.

Produkty jsou v databázi zastoupeny ve 4 tabulkách. Hlavní informace o konkrétním zboží nalezneme v tabulce `products`. Informace o obrázkové galerii konkrétního produktu jsou k dispozici v tabulce `product_images`. V tabulce

`products_parameters` se nachází klíčové informace o daném produktu, uložené jako parametr–hodnota. Tabulka `product_parameters_parameters` obsahuje konkrétní parametry, ze kterých se dá vybírat při přidávání produktu do databáze.

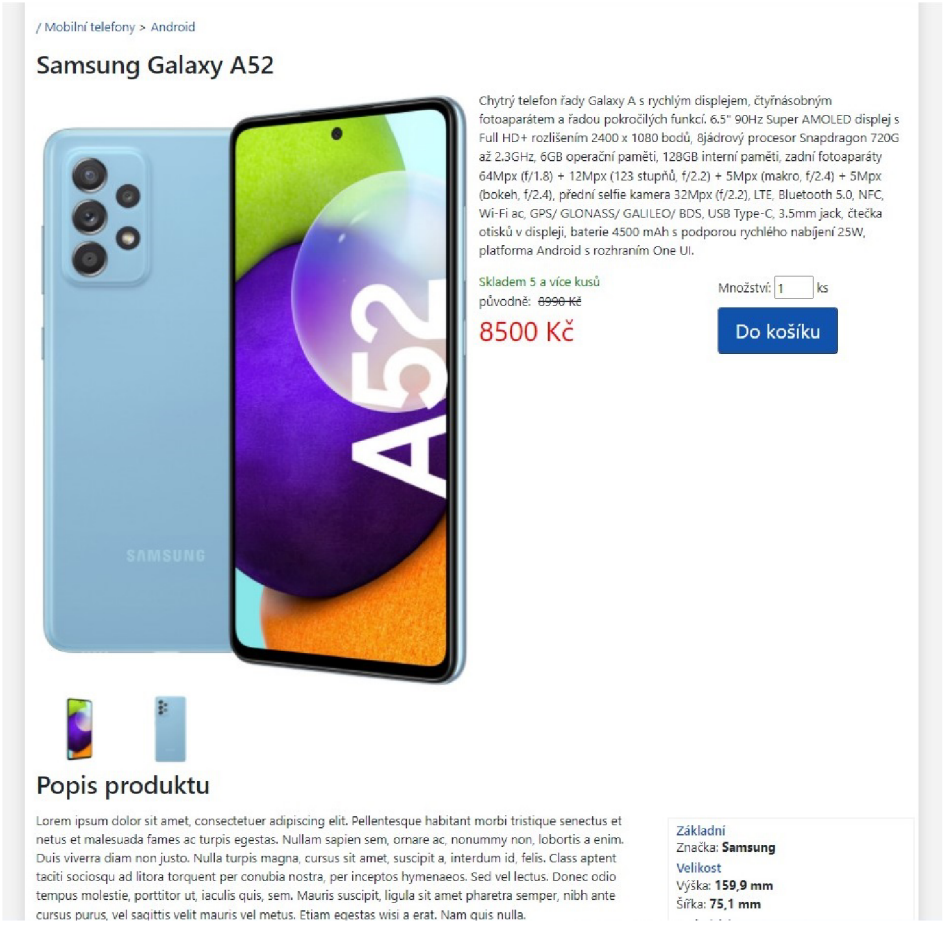
K informacím o objednávkách se dostaneme v tabulce `orders`. Jsou zde obsaženy i detaily o platbě a dopravě. Jednotlivé produkty patřící k určité objednávce jsou uloženy v tabulce `order_items`, obsahující i informaci o ceně z času zakoupení produktu.

Uživatelské hodnocení s textem recenze lze nalézt v tabulce jménem `reviews`. Další části recenze, což jsou kladné a záporné odrážky, obsahuje tabulka `reviews_pros_cons`.

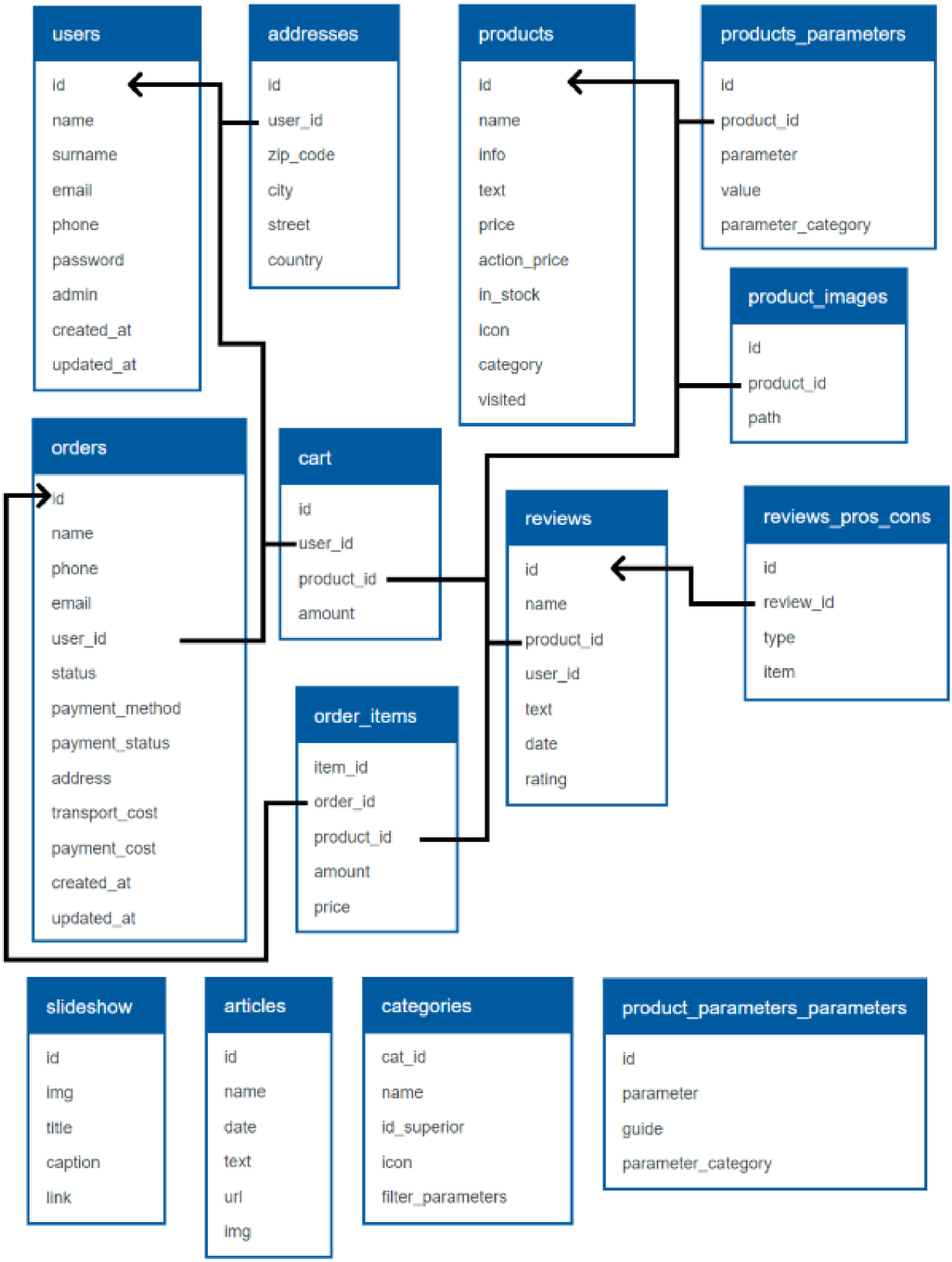
Články s akcemi a sděleními obchodu se nacházejí v tabulce `articles`, která obsahuje i cestu k titulnímu obrázku k článku.

Košík je v databázi zastoupen tabulkou `cart`, kde jsou uloženy košíky veškerých uživatelů. Co patří kterému uživateli, je rozděleno pomocí id uživatele.

Slideshow z titulní stránky má data uložena v tabulce se jménem `slideshow`. Můžeme zde nalézt nadpis, popisek obrázek a také odkaz.




Obrázek 6: Detaily produktu




Obrázek 7: Schéma databáze

# 4 Porovnání frameworků při tvorbě webů

V této kapitole rozeberu části porovnávaných frameworků při tvorbě webů. Ke každé popsané části připojím mé dojmy při vytváření webových stránek.

## 4.1 Založení projektu

V této sekci popíšu vytvoření projektu pomocí porovnávaných technologií. Budu popisovat zejména mnou použité postupy, ale to neznamená, že neexistují jiné.

### 4.1.1 Založení projektu - Laravel

Pro zprovoznění Laravelu je třeba mít PHP alespoň ve verzi 7.3, které jsem nainstaloval díky balíčku XAMPP. Pro samotnou instalaci Laravelu jsem použil nástroj Composer. Po nainstalování nástroje stačí v příkazovém řádku ve vybrané složce zadat příkaz,

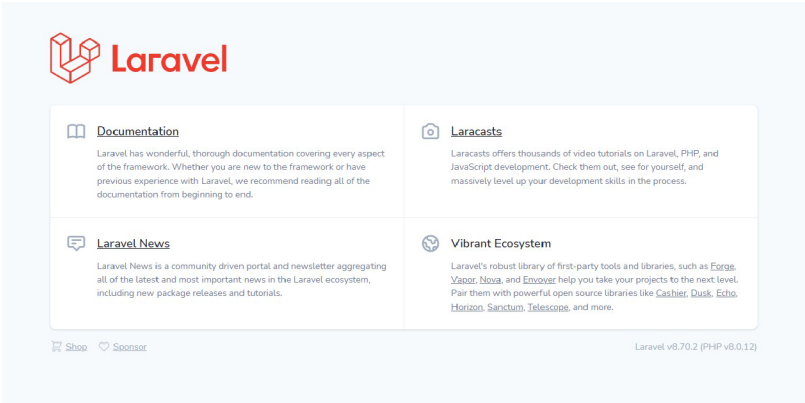
```
composer create-project --prefer-dist laravel/laravel nazev
```

jenž vytvoří složku s projektem, ve kterém budeme pracovat. Alternativní postupy můžeme najít na oficiálních stránkách Laravelu.

Pro vyzkoušení, jestli vše funguje, můžeme v příkazové řádce ve složce projektu zadat příkaz,

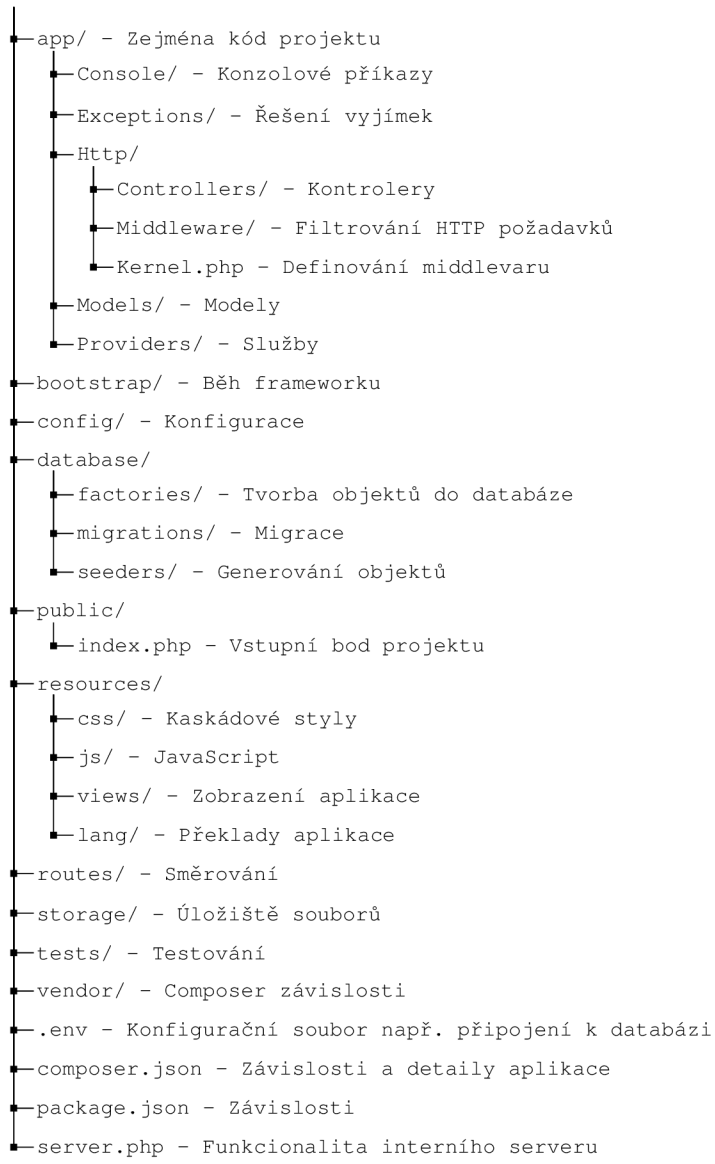
```
php artisan serve
```

který pomocí vestavěného webového serveru projekt spustí na localhostu. Po zadání příslušné adresy do internetového prohlížeče bychom měli vidět základní stránku Laravelu. Dále už jen zbývá zvolit si vývojové prostředí. Já jsem si pro práci s frameworkem zvolil PhpStorm, který obsahuje všechny potřebné nástroje.




Obrázek 8: Základní stránka Laravel projektu

### 4.1.2 Struktura projektu - Laravel



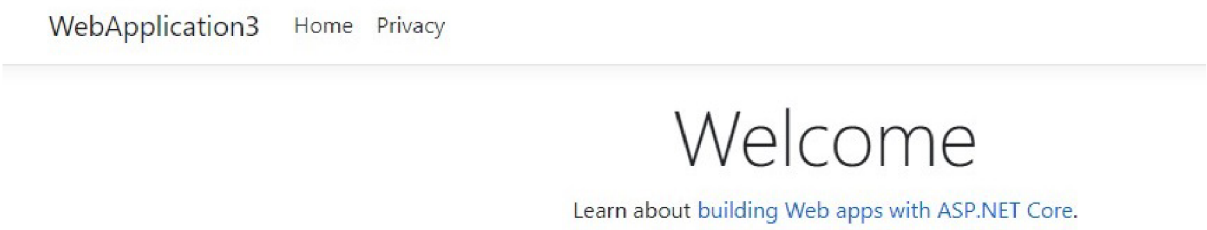

### 4.1.3 Založení projektu - ASP.NET Core

Nejjednodušší možností, jak vytvořit a editovat ASP.NET Core projekt, je v prostředí Visual Studio. Zvolíme vytvořit nový projekt a vybereme jazyk C#. Poté zvolíme položku ASP.NET Core (Model-View-Controller). V případě, že tato volba chybí, je třeba doinstalovat potřebná rozšíření v instaleru Visual Studia. V dalším menu zadáme název projektu a zvolíme umístění. Po kliknutí na tlačítko `dále`, zvolíme verzi a klikneme na `vytvořit`.

Pro vyzkoušení funkčnosti klikneme ve Visual Studiu na tlačítko s ikonou play a spustíme projekt nebo v příkazovém řádku ve složce projektu napíšeme

```
dotnet run
```

Poté bychom měli vidět základní stránku projektu.




Obrázek 9: Základní stránka ASP.NET Core projektu

### 4.1.4 Struktura projektu - ASP.NET Core

```
├─bin/ - Binární soubory generované při kompilaci
├─Controllers/ - Kontrolery
├─Migrations/ - Migrace do databáze
├─Models/ - Modely
├─obj/ - Dočasné soubory a objekty
├─Properties/
│  └─launchSettings.json - Informace ke spuštění aplikace
├─Views/ - Zobrazení
├─wwwroot/
│  ├─css/ - Kaskádové styly
│  ├─js/ - JavaScript
│  └─lib/ - Knihovny
├─appsettings.json - Konfigurace aplikace
├─Program.cs - Vstupní bod aplikace
├─Startup.cs - Konfigurace kanálu požadavků a middlewaru
└─nazev.csproj - Nastavení projektu
```

**Mé dojmy**

Vytvoření nového projektu mi přišlo snazší v ASP.NET Core. Možná to bylo díky tomu, že jsem pracoval na systému Windows a projekt vytvářel pomocí Visual Studia. Na jiném systému by nejspíš bylo snazší vytvoření Laravel projektu. Adresářová struktura projektu mi přišla rovněž přehlednější u ASP.NET Core než u Laravelu, protože obsahovala méně složek s více intuitivním umístěním zdrojových souborů.

## 4.2 Vytváření zobrazení

V této sekci předvedu, jak je řešeno vytváření zobrazení u jednotlivých frameworků. Zobrazení (View) slouží k převedení dat do podoby, která je prezentována uživateli. Díky zobrazením můžeme odlišit kód logiky stránky v kontroleru od vzhledu a tudíž je kód přehlednější.

### 4.2.1 Blade - Šablonovací systém v Laravelu

Blade je jednoduchý, ale výkonný šablonovací systém, který je součástí Laravelu. Na rozdíl od ostatních systémů umožňuje používat v šablonách klasický PHP kód. Zobrazení se zkompilují do čistého PHP a jsou uloženy do mezipaměti do té doby, než se změní. Díky tomuto ubývá režie při provozu aplikace. Blade šablony používají koncovku `.blade.php` a jsou uloženy v adresáři `resources/views`.

Syntaxe v Bladu využívá u názvů funkcí znak @. Blokové funkce jsou ve tvaru `@nazev .... @endnazev`. K proměnným se přistupuje pomocí složených závorek `{{$nazev}}`. Veškerý obsah v závorkách prochází z důvodu ochrany funkcí, která konvertuje speciální HTML znaky a zabraňuje útokům. Tuto ochranu lze obejít použitím syntaxe `{!! $nazev !!}` v případě, když potřebujeme tvořit HTML z proměnné. Výhodou Bladu je, že automaticky vytváří v iteracích proměnnou `$loop`, která obsahuje užitečné informace o aktualní iteraci cyklu. Díky ní se dostaneme k aktuálnímu indexu, počtu iterací, rodičovské proměnné `$loop` (pokud jsou cykly vnořené) nebo k dalším věcem. Komentáře se dají tvořit použitím `{{- komentar -}}`.

```
1  <article class="article">
2          <h2>{{ $article['name'] }}</h2>
3          <span>{{date_format(date_create( $article['date'] ), 'j. n.
               Y')}}</span>
4          <img src="{{ $article['img'] }}">
5          <div>{!! $article['text'] !!}</div>
6  </article>
7
8  {{--  Ukazka cyklu s promennou loop --}}
9  <h1>Seznam uzivatelu:</h1>
10 <ul>
11 @foreach ( $users as $user )
12      <li>{{ $user->name }}</li>
13     @if ($loop->last)
14           </ul><p>Bylo zobrazeno {{ $loop->count }} uzivatelu a
       posledni uzivatel ma id {{ $user->id }}.</p>
15     @endif
16 @endforeach
```

Zdrojový kód 1: Ukázky syntaxe v Bladu

Předávání dat pohledu probíhá v kontroleru nejčastěji s příkazem `return` při vracení zobrazení. V šabloně jsou data reprezentována jako PHP proměnné.

```
...
//prvni zpusob
return view( 'product', ["products"=>$products ,
"pagesCount"=>$pagesCount, 'page'=>$req->page ]);
}

...
//druhy zpusob
return view('orders')->with( 'orders', $orders )->with('page',1);
}
```

Zdrojový kód 2: Předání dat šabloně Laravel

### 4.2.2 Razor - Šablonovací systém v ASP.NET Core

Razor je programovací syntaxe ASP.NET, používaná k tvorbě dynamických webových stránek. Šablony používají koncovku `.cshtml` a jsou uloženy ve složce `Views`.

V ASP.NET Core se zobrazení rozdělují do složek podle názvů kontrolerů. Díky tomu je v zobrazeních větší přehled a můžeme je z kontroleru zavolat bez udání názvu (šablona se zvolí automaticky ze složky odpovídající názvu kontroleru a má jméno podle metody). Sdílené šablony jsou uloženy ve složce `Views/Shared`. Do souboru s názvem `_ViewImports.cshtml` se zapisují použité direktivy, abychom je nemuseli psát do každého zobrazení ve složce zvlášť. V souboru `_ViewStart.cshtml` se většinou určuje šablona layoutu pro dané stránky.

Syntaxe Razoru také používá symbol @. Narozdíl od Bladu slouží @ k rozlišení jak funkcí, tak i proměnných. Blokové funkce se píší ve tvaru `@nazev{...}`. Složitější výrazy k vyhodnocení lze psát ve tvaru `@(...)` a C# kód ve tvaru `@{...}`. Když potřebujeme vypsat HTML kód z proměnné, použijeme funkci `@Html.Raw(promena)`. Komentáře se píší pomocí vzorce `@* komentar *@`.

```
<article class="article">
  <h2>@Model.Name</h2>
  <span>@Article.ArticleDateFormat(Model.Date.ToString())</span>
  <img src="@Model.Img">
  <div>@Html.Raw(Model.Text);</div>
</article>

@* Ukazka cyklu *@
@foreach (User user in ViewBag.Users){
  <tr>
    <td>@(user.Name + " " + user.Surname)</td>
    <td>@user.Email</td>
    <td>@user.Phone</td>
    <td>@user.Admin</td>
    <td>@user.Created_at.ToString()</td>
    <td><a href="/admin/edituser/@user.Id"><i class="fa fa-pencil"
        ></i></a>
```

```
17      </td>
18    </tr>
19  }
```

Zdrojový kód 3: Ukázka syntaxe v Razoru

Do šablony lze předávat data pomocí objektu ViewBag, slovníku ViewData a silně typovaného modelu. Použití je k vidění níže.

```
...
//ViewBag a ViewData
ViewBag.nazev1 = data1;
ViewData["nazev2"] = data2;
...
//model
return View("about", address);
...
//v pripade ze se zobrazeni jmenuje stejne jako metoda
return View(address);
}
```

Zdrojový kód 4: Předání dat šabloně ASP.NET Core

```
// model musime na zacatku definovat
@model WebApplication1.Models.address
//pouziti modelu
<address>
    @Model.Street<br>
    @Model.City, @Model.State @Model.PostalCode<br>
</address>

//ViewBag a ViewData
@ViewBag.nazev1;
@ViewData["nazev2"];
```

Zdrojový kód 5: Přistoupení k datům v šabloně ASP.NET Core

**Mé dojmy**

Z mého pohledu se v obou zmíněných systémech pracuje po pochopení syntaxe intuitivně, ale v Bladu se mi pracovalo o něco lépe. Přišlo mi, že je tam lépe vyřešeno rozlišení mezi HTML a kódem jazyka. Další výhodou byla možnost použití automaticky vytvářených proměnných v cyklech.

## 4.3 Práce s databází

V této části textu popíšu, jak se pracuje s databázemi ve zmíněných frameworcích. Nejprve probereme připojení k databázi a poté práci s ní.

### 4.3.1 Připojení k databázi v Laravelu

V Laravelu je zvykem použití databáze MySql. Typ databáze, ke které se připojujeme, se nastavuje v umístění `config/database.php` (v základu je to nastaveno na MySql). Nastavení připojení se děje v souboru `.env`. Nalezneme v něm řádky začínající písmeny DB a nastavíme hodnoty. Pro použití databáze stačí v daném zdrojovém souboru zadat, že používáme databázi:
`use Illuminate\Support\Facades\DB;`. Pro přístup k tabulce použijeme objekt `DB:table('tabulka')` nebo lze tabulku přiřadit k modelu díky proměnné `$table`, reprezentující název tabulky, a přistupovat k modelu.

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel-shop
DB_USERNAME=root
DB_PASSWORD=
...
```

Zdrojový kód 6: Připojení k databázi Laravel

### 4.3.2 Použití databáze - Laravel

Rozhraní databázových dotazů v Laravelu je velmi pohodlné. Dotaz začíná databázovým objektem DB, kterému se upřesňuje tabulka výrazem `DB::table("nazev")`, nebo modelem. Příkazy se navazují na sebe pomocí šipky ->.

Při výběru dat z databáze nejčastěji použijeme funkci `select()`, ve které zvolíme potřebné sloupce z tabulky, a funkci `where()` filtrující data podle zadané podmínky. Funkci `where()` je možno volat jen se dvěma parametry(sloupec, hodnota), což je bráno jako operace =, nebo se mezi tyto dva parametry napíše jiná operace(např '>'). Dotaz bývá ukončen funkcí `get()`, jež získá výsledky z databáze. Dalšími užitečnými ukončovacími funkcemi jsou funkce `first()`, která vrací první výsledek, a funkce `value()`, vracející jednu hodnotu zvoleného sloupce. Ještě se hodí znát funkci `pluck()`, jež vrátí kolekci dat jen z jednoho určeného sloupce. Seřazení dat se řeší funkcí `orderBy()`, mající dva parametry: sloupec, podle kterého se řadí a směr řazení udávaný hodnotou `'asc'` nebo `'desc'`.

Vkládání prvků do databáze probíhá pomocí funkce `insert()`, která má jako parametr pole hodnot nebo pole polí hodnot. Hodnoty jsou udány ve tvaru `sloupec=>hodnota` a jsou odděleny čárkami. Úprava záznamu v databázi

probíhá metodou `update()` se stejným tvarem parametru jako `insert()`, které předchází metoda `where()`. Jako kombinace obou postupů lze použít `updateOrInsert()`, která data buď upraví nebo přidá. Další možností, jak upravit prvek v databázi, je získat záznam do proměnné pomocí modelu, upravit jej a poté zavolat funkci `save()` pro uložení (`$promenna->save()`). Metoda `save()` funguje i pro přidání záznamu. Do proměnné se přiřadí nová instance modelu (`$promenna = new nazev_modelu`), poté se upraví a zavolá se funkce `save()` pro uložení.

Mazání záznamů se provádí pomocí metody `delete()`. Použijeme příkazy, jako bychom chtěli filtrovat z databáze, ale místo ukončovací funkce napíšeme `delete()`. Pomocí funkce `destroy()` lze smazat data modelu pomocí primárního klíče (`Model::destroy(id)`).

Spojování tabulek se provádí pomocí metody `join()`, která má parametry: název tabulky, sloupec první tabulky, operace ('=') a sloupec druhé tabulky.

```
...
DB::table('product_parameters')->where( 'product_id', $item->id )->
    where( 'parameter',$key )->whereIn( 'value', $params )->first();
...
$user = new User;
$user->name = $req->name;
$user->surname = $req->surname;
$user->email = $req->email;
$user->phone = $req->phone;
$user->password = Hash::make($req->password1);
$user->admin = 0;
$user->save(); //ulozeni do db
...
DB::table('orders')->where('id', $req->id )->update(array('status'=>
    $req->status , 'payment_status'=>$req->payment_status ));
...
DB::table('addresses')->insert($values);
...
$items[$i] = OrderItem::where('order_id', $orders[$i]->id )
->join('products', 'product_id','=','products.id')
->select('order_items.*', 'products.name', 'products.icon')->get();
...
```

Zdrojový kód 7: Ukázka práce s databází v Laravelu

### 4.3.3 Připojení k databázi v ASP.NET Core

ASP.NET Core používá k připojení k databázím Entity Framework Core. Obyčejně se pracuje s databázemi Microsoft SqlServer, ale v mém případě jsem zvolil pracovat s databází MySql kvůli zachování stejné databáze v obou implementacích obchodu. Ve Visual Studiu bylo třeba doinstalovat NuGet balíček `MySql.EntityFrameworkCore` a balíček `Microsoft.Entity.FrameworkCore.Tools` (při použití SqlServer databází je třeba doinstalovat balíček

`Microsoft.EntityFrameworkCore.SqlServer`). Dále je třeba vytvořit připojovací string. Většinou se píše do souboru `appsettings.json`.

```
...
"ConnectionStrings": {
    "Default": "server=localhost;user=root;password=;database=
       laravel-shop;port=3306;SslMode=none;convert zero datetime=
       True"
  },
...
```

Zdrojový kód 8: Připojovací string ASP.NET Core

Pro používání databáze je potřeba vytvořit třídu, která dědí od třídy `DbContext` z balíčku `EntityFrameworkCore`. Vytvoříme konstruktor a dáme mu jako parametr objekt `DBContextOptions`, který používá naši nově vytvořenou třídu. V konstruktoru předáme `DBContextOptions` rodičovské třídě pomocí klíčového slova `base`. Dále už jen zbývá přidat vlastnosti typu `DbSet` využívající odpovídající model pro každou tabulku v databázi. Tuto vytvořenou třídu je třeba registrovat v souboru `Startup.cs` v metodě `ConfigureServices()` pomocí `AddDbContextPool()` nebo `AddDbContext()` (AddDbContext vytváří pokaždé novou instanci). Při registraci použijeme připojovací string, který jsme si vytvořili dříve. Pro použití databáze v kontroleru přidáme do konstruktoru jako parametr naši databázovou třídu a v těle metody přiřadíme do proměnné.

```
1  //registrace v metode Configure services v souboru Startup.cs.
2  public void ConfigureServices(IServiceCollection services)
3  {
4    services.AddDbContextPool<DatabaseContext>(options => options.
       UseMySQL(Configuration.GetConnectionString("Default")));
5    ...
6  }
7
8
9  //databazova trida
10 using ASPNetCore_shop.Models;
11 using Microsoft.EntityFrameworkCore;
12
13 namespace ASPNetCore_shop.Data
14 {
15     public class DatabaseContext : DbContext
16     {
17         public DatabaseContext(DbContextOptions<DatabaseContext>
               options) : base(options)
18         {}
19
20         public DbSet<User> Users { get; set; }
21         public DbSet<Product> Products { get; set; }
22         public DbSet<Category> Categories { get; set; }
23         public DbSet<Review> Reviews { get; set; }
24         public DbSet<ReviewProsCons> Reviews_pros_cons { get; set; }
```

```
25          public DbSet<Slideshow> Slideshow { get; set; }
26          public DbSet<Article> Articles { get; set; }
27          public DbSet<ProductImage> Product_images { get; set; }
28          public DbSet<ProductParameter> Product_parameters {get;set;}
29          public DbSet<Order> Orders { get; set; }
30          public DbSet<OrderItem> Order_items { get; set; }
31          public DbSet<Address> Addresses { get; set; }
32          public DbSet<CartItem> Cart { get; set; }
33          public DbSet<ProductParameterParameter>
                Product_parameters_parameters { get; set; }
34      }
35  }
```

Zdrojový kód 9: Třída pro přístup do databáze ASP.NET Core

### 4.3.4 Použití databáze - ASP.NET Core

Entity framework využívá dvě možnosti přístupu k databázi. Jednodušší dotazy začínají objektem databázového kontextu, ze kterého pomocí tečky přistoupíme k objektu `DBSet`, odpovídajícího chtěné tabulce, za který se postupně píšou příkazy oddělené tečkami. Pro složitější dotazy, kde je například třeba spojovat tabulky, můžeme použít syntaxi LINQ (Language-Integrated Query), která má blízko klasickým SQL dotazům. Používají se klíčová slova `from, in, where, select` a jiné. Mně se osvědčilo tyto dva přístupy kombinovat.

Při klasickém dotazu na databázi, nejčastěji používáme funkci `Where()`, které jako parametr předáme lambda výraz. Pokud potřebujeme vrátit data v jiném objektu, než je model databáze, použijeme funkci `Select()`, které předáme v lambda výrazu přižazení proměnných do slotů objektu. Seřazení výsledků se provádí příkazem `OrderBy()` nebo `OrderByDescending()` podle směru seřazení. Jako parametr zadáváme lambda výraz s prvkem, podle kterého se řadí. Dotaz většinou ukončujeme funkcí `ToList()`, která vytvoří seznam objektů. Když potřebujeme jen jeden objekt, tak dotaz ukončíme funkcí `First()`, která vyvolá výjimku, když není co vrátit, nebo `FirstOrDefault()`, která může vrátit `null`. Když se dotazujeme pomocí LINQ, tak je většinou dotaz ve tvaru: `from nazevPromenne in tabulkaNeboVyraz where podminka orderby vyraz select objekt`.

Vkládat do databáze můžeme pomocí funkce `Add()` nebo `AddRange()` pro více záznamů. Upravovat záznamy můžeme analogicky metodami `Update()` a `UpdateRange()`. Upravit záznam se dá také tak, že objektu po získání z databáze upravíme nějakou vlastnost. Ať už přidáváme nebo měníme data jakkoli, musíme vždy potvrdit změny zavoláním metody `SaveChanges()` z databázového kontextu.

Mazání záznamů z databáze se provádí metodami `Remove()` a `RemoveRange()`. Poté musíme stejně jako u úprav potvrdit změny metodou `SaveContext()`.

Spojovat tabulky lze pomocí `join` v syntaxi LINQ ve tvaru `... join promena in pripojovanaTabulka on sloupecPredchoziTab`

equals sloupecPripojovanaTab ....

```
...
CartItem item = _context.Cart.Where(item=>item.Id==id).Where(item =>
    item.User_id == userId).FirstOrDefault();
...
List<List<OrderItemJoinProduct>> items = new List<List<
    OrderItemJoinProduct>>();
for (int i = 0; i < orders.Count; i++)
{
  items.Add((from it in _context.Order_items.Where(item => item.
      Order_id == orders[i].Id)
    join p in _context.Products on it.Product_id equals p.Id
    select new OrderItemJoinProduct{
        Name = p.Name,
        Icon = p.Icon,
        Amount = it.Amount,
        Price = it.Price,
        Item_id = it.Item_id,
        Order_id = it.Order_id,
        Product_id = p.Id
    }).ToList());
}
...
int sum = (from i in _context.Cart.Where(item => item.User_id ==
    userId)
    join u in _context.Products on i.Product_id equals u.Id
    select u.Action_price * i.Amount).Sum();
...
List<CategoryWithShift> categoriesZeroLevel = _context.Categories.
    Where(c => c.Id_superior == 0).Select(c => new CategoryWithShift
    {
       Name = c.Name,
       Cat_id = c.Cat_id,
       Filter_parameters = c.Filter_parameters,
       Icon = c.Icon,
       Id_superior = c.Id_superior
  }).ToList();
...
context.Cart.Add(new CartItem { Product_id = product.Id, Amount =
    cart[product.Id], User_id = userId});
context.SaveChanges();
...
Slideshow slideshow = _context.Slideshow.Where(slide => slide.Id ==
    id).FirstOrDefault();
if (slideshow != null)
{
  _context.Slideshow.Remove(slideshow);
  _context.SaveChanges();
...
```

Zdrojový kód 10: Ukázka práce s databází v ASP.NET Core

**Mé dojmy**

Práce s databází mi přišla lepší v ASP.NET Core díky pevně daným typům dat. Staticky typované datové typy jsou zároveň hlavní výhodou a nevýhodou. Výhodou je, že je třeba přiřadit data do odpovídajícího slotu objektu, tím se zmenšuje pravděpodobnost, že se člověk splete nebo se zamění sloupce se stejným jménem. Právě díky tomuto jsem přišel na chybu v implementaci v Laravelu, kde se mi zaměnil sloupec ceny z tabulky produktů, vázaných k objednávkám, za sloupec ceny z tabulky produktů při dotazu z databáze. Nevýhodou je, že na každý složitější dotaz s propojováním tabulek je třeba zakládat nová třída.

## 4.4 Směrování (Routing)

V této části se bude řešit směrování. Úkolem směrování je podle URL adresy poznat, co uživatel hledá, a pověřit příslušný kontroler, aby obsloužil požadavek.

### 4.4.1 Směrování Laravel

V Laravelu se routování naší aplikace řeší v souboru `web.php` ve složce `routes`. Nejčastěji využívané metody jsou `get()` a `post()`, ale existují i jiné. Například když potřebujeme nadefinovat cestu, která se použije při metodě `post` i metodě `get`, tak použijeme funkci `match()`. Důležité je vědět, že metoda `post` a další, potřebují při zpracování formuláře mít definovaný CSFR token kvůli ochraně. Ten se nadefinuje pomocí `@csfr` v šabloně v HTML tagu `form`. Parametr se v routování definuje pomocí složených závorek, když je volitelný, tak se za jeho název napíše otazník (`.../{param?} ...`). U parametrů může být omezen formát pomocí metody `where()` s regulárním výrazem.

```
1  //priklad metody match
2  Route::match(['get', 'post'], '/', function () {
3      //
4  });
5
6  //priklad where
7  Route::get('/user/{name}', function ($name) {
8      //
9  })->where('name', '[A-Za-z]+');
10 ...
11
12 Route::get('/', [IndexController::class, 'index']);
13
14 Route::post('/index_update', [IndexController::class, 'indexUpdate'
       ]);
15
16 Route::get('/basket', [BasketController::class, 'cartList']);
17
18 Route::post("add_to_cart", [BasketController::class, "addToCart"]);
```

```
19
20 Route::get("/updatecart/{id}/{value}", [BasketController::class, '
       updateCart']);
21
22 Route::get("/removecart/{id}", [BasketController::class, 'removeCart
       ']);
23
24 Route::post("/addtobasket", [BasketController::class, 'addToBasket'
       ]);
25
26 Route::get('/search', [CategoryController::class, 'search']);
27
28 Route::get('/product/{id}', [ProductController::class, 'detail']);
29
30 Route::get('/login', function () {
31     return view('login');
32 });
33 ...
```

Zdrojový kód 11: Ukázka routování v Laravelu

### 4.4.2 Směrování ASP.NET Core

V ASP.NET Core lze směrovat dvěma způsoby. První způsob je konvenční směrování, které se nastavuje v souboru `Startup.cs` v metodě `Configure()`. V této metodě se definuje použitý middleware, což si můžeme představit jako filtry, přes které postupně prochází požadavek, než se najde jeden vhodný, který jej zpracuje. V základu bývá routování nastaveno takto: `{controller}/{action}/{id?}`. Při spuštění se v základu nasměruje aplikace na `HomeController` a použije se metoda `Index()`. Jako ekvivalent k základnímu routování lze použít řádek `app.UseMvcWithDefaultRoute();`. Vlastní směrování lze nadefinovat pomocí metody `app.useMVC()` v parametru `routes` nebo pomocí `app.UseEndpoints()`.

```
1  public void Configure(IApplicationBuilder app, IWebHostEnvironment
       env)
2  {
3    if (env.IsDevelopment())
4    {
5      app.UseDeveloperExceptionPage();
6    }
7    app.UseHttpsRedirection();
8    app.UseStaticFiles();
9    app.UseRouting();
10   app.UseAuthorization();
11   app.UseSession();
12   app.UseEndpoints(endpoints =>
13   {
14     endpoints.MapControllerRoute(
15       name: "default",
16       pattern: "{controller=Home}/{action=Index}/{id?}");
```

```
17    });
18 }
```

Zdrojový kód 12: Ukázka metody Configure v souboru Startup.cs v ASP.NET Core

Druhý způsob je atributové směrování. Spočívá v přidání atributu route před metodu. Atribut se píše do hranatých závorek takto: `[Route("adresa")]`. Díky atributům lze specifikovat, v jakém případě se metoda použije. Můžeme přidat atribut `[HttpPost]` nebo `[HttpGet]`, podle toho v jakém případě chceme metodu použít. Atribut lze napsat i před třídu kontroleru. Díky tomu se cesty v atributech u metod spojují s cestou v atributu kontroleru. Atributové směrování přepisuje konvenční směrování.

```
1  [HttpGet]
2  [Route("/Article/{url}")]
3  public IActionResult Article(string url)
4  {
5     Article article = _context.Articles.Where(article => article.Url
       == url).FirstOrDefault();
6     if (article != null)
7     {
8        return View(article);
9     }
10    else
11    {
12       return Redirect("/not_found");
13    }
14 }
```

Zdrojový kód 13: Ukázka Atributového routování v ASP.NET Core

**Mé dojmy**

Podle mě je směrování lépe uděláno v ASP.NET Core, protože si člověk může vybrat. Dále se mi líbí, že u atributového směrování je vidět URL hned u metody a nemusí ji hledat v jiném souboru. Šetří to čas.

## 4.5 Middleware

V této sekci si ukážeme, jak je ve frameworcích vyřešeno použití middlewaru. Middleware poskytuje mechanismus pro filtrování požadavků, které vstupují do naší aplikace.

### 4.5.1 Middleware v Laravelu

Všechno co se týká middlewaru v Laravelu, můžeme nalézt v adresáři `app/Http/Middleware`. Nový middleware lze vytvořit konzolovým příkazem:

```
php artisan make:middleware nazev
```

Ve vytvořené funkci `handle($request, $next)` definujeme funkčnost našeho middlewaru. Pracujeme s tokenem, ke kterému se ve funkci dostaneme pomocí parametru `$request->input('token')`. Na konci metody bychom vždy měli volat metodu, která přišla v parametru `$next`, aby mohl pokračovat ve zpracování další middleware v řadě. Aby byl náš middleware funkční, je třeba jej registrovat v souboru `app/Http/Kernel.php`. V tomto souboru registrujeme náš middleware v zastoupených proměnných podle toho, jak jej plánujeme využít. Pokud chceme middleware využít při každém HTTP requestu, tak jej registrujeme v proměnné `$middleware`. Pro použití middleware vyvoláním metody `middleware()` při routování hledáme proměnnou `$routeMiddleware`. Pro použití více middlewarů pomocí jednoho klíče slouží proměnná `$middlewareGroups`. Middleware může mít také dodatečné parametry, které se definují po parametru `$next`.

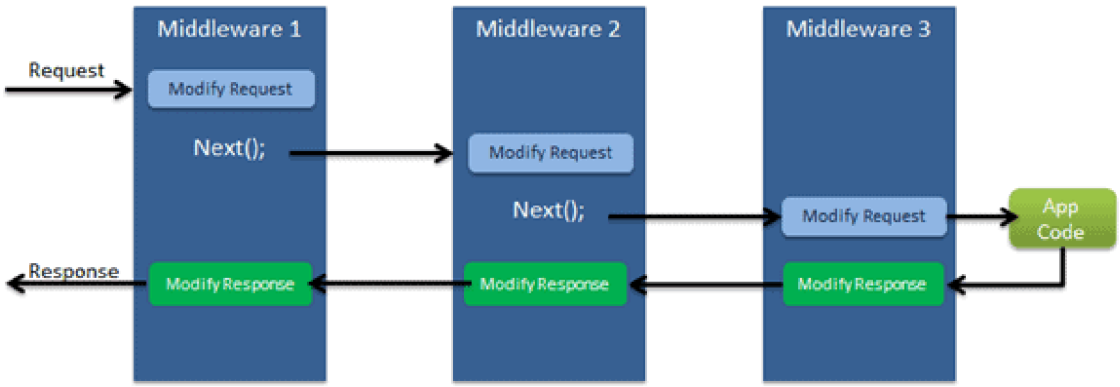
```
...
//priklad funkce handle ve tride middlewaru
public function handle($request, Closure $next, $param1 )
{
  if ($request->input('token') !== 'muj-token') {
    // telo metody
    return redirect('home');
  }
  return $next($request) ;
}

//volani middlewaru s parametrem pri smerovani
Route::put('/post/{id}', function ($id ) {
    //telo metody
})->middleware('nazev_middlewaru:param1');

//pouziti vice middlewaru pri smerovani
Route::get('/', function () {
    //telo metody
})->middleware(['middleware1', 'middleware2']);
```

Zdrojový kód 14: Ukázky middleware v Laravelu

### 4.5.2 Middleware v ASP.NET Core

V ASP.NET Core se definuje použití middlewaru v metodě `Configure()` v souboru `Startup.cs`. Každý middleware přidává nebo upravuje požadavek http a volitelně předává řízení další komponentě middlewaru. Díky volitelnému předávání řízení záleží na pořadí, ve kterém middlewary zapíšeme, protože některé middlewary jsou terminální, zpracují svůj požadavek a nevolají další middleware. Při každém požadavku na server jsou vykonávány middlewary v zapsaném pořadí. V ASP.NET Core existuje mnoho předpřipravených middlewarů, které jsou

připraveny k použití. Například můžu zmínit middleware `UseDeveloperExceptionPage()`, zachycující výjimky a generující chybové hlášení, který by se měl používat jen při vývoji, protože by útočník mohl zjistit části kódu naší stránky. Další příklady middlewarů můžete naleznout v zdrojovém kódu u konvenčního směrování, které se také nastavuje pomocí middlewaru.




Obrázek 10: Middleware pipeline v ASP.NET Core

Vlastní middleware můžeme vytvořit tak, že založíme třídu, která bude mít v konstruktoru parametr `RequestDelegate`. Třída musí také obsahovat metodu `Invoke()` nebo `InvokeAsync()`. Tyto metody musí mít první parametr typu `HttpContext` a návratový typ `Task`. Pokud nechceme aby náš middleware byl terminální, tak musíme použít na konci metody delegáta na další middleware (ukázka v kódu níže). Middleware přidáme do metody `Configure()` pomocí `app.UseMiddleware<TridaMiddlewaru>();`

```
1  //priklad tridy middlewaru
2  public class CustomMiddleware
3  {
4      private readonly RequestDelegate _next;
5
6      public CustomMiddleware(RequestDelegate next)
7      {
8          _next = next;
9      }
10
11
12
13     public async Task Invoke(HttpContext context,  IMyScopedService
           svc)
14     {
15         svc.MyProperty = 1000;
16         await _next(context);
17     }
18 }
19
```

```
20 //pouziti middlewaru
21 public class Startup
22 {
23     public void Configure(IApplicationBuilder app)
24     {
25         ...
26         app.UseMiddleware<CustomMidleware>();
27         ...
28     }
29 }
```

Zdrojový kód 15: Ukázky middleware vASP.NET Core

**Mé dojmy**

V Laravelu u mé implementace webu teoreticky nebylo třeba do middlewarů zabíhat. V ASP.NET Core to už byla nutnost, jelikož je na nich postavena velká funkcionalita. Bylo pro mě trochu těžší proniknout do middlewaru v ASP.NET Core kvůli tomu, že záleželo na pořadí.

## 4.6 HTTP Session

Zde si rozebereme, jak je řešena session v porovnávaných frameworcích. Jedná se o možnost, jak uložit informace v rámci více požadavků na stránku. Tyto informace jsou uloženy v trvalém úložiti na straně serveru.

### 4.6.1 Session v Laravelu

Session můžeme nastavit v umístění `config/session.php`. Lze tam například nastavit doba, po kterou budou data uložena, nebo zapomenutí dat při zavření prohlížeče. Data se ukládají do session pomocí dvojice klíč a hodnota. Existují dva způsoby práce s úložištěm session. První možností je přístup k session pomocí objektu třídy `Request`, který lze použít jako `$request->session()`. Druhou možností je globální funkce `session()`.

Data můžeme uložit pomocí funkce `put()`, která má parametry klíč a hodnota. K uloženým datům se dostaneme pomocí funkce `get()` s parametrem klíč. Lze získat i všechna data naráz pomocí funkce `all()`. Mazání dat z úložiště je zajištěno pomocí metody `forget()`, která má jako parametr klíč nebo pole klíčů. Všechna data z úložiště lze smazat pomocí `flush()`. Existuje také metoda `pull()`, která získá data a smaže je z úložiště.

Velmi užitečné jsou funkce `has()` a `exist()`, které můžeme použít ke zjištění, zda položka existuje v úložišti. Rozdíl mezi těmito dvěma funkcemi je, že `has()` navíc rozlišuje, jestli položka není `null`.

```
1 //ulozeni dat
2 $request -> session() -> put('key', 'value');
3 session(['key' => 'value']);
```

```
...
//ziskani dat
$value =  $request -> session() -> get('key');
$data =  $request -> session() -> all();
...
//mazani dat
$request -> session() -> forget('name');
$request -> session() -> forget(['address', 'status']);
$request -> session() -> flush();
...
//zkouska, jestli session obsahuje prvek
if ($request -> session() -> has('users')) {
    //kod
}

if ($request -> session() -> exists('users')) {
    //kod
}
```

Zdrojový kód 16: Práce se session v Laravelu

### 4.6.2 Session v ASP.NET Core

Abychom mohli používat session v ASP.NET Core, musíme do metody `ConfigureServices()` v souboru `Startup.cs` dopsat řádek `services.AddDistributedMemoryCache();`, který slouží k definici záložního úložiště, a řádek `services.AddSession();`. V metodě `AddSession();` je možné provést nastavení. Dále je třeba použít middleware `useSession()`, což uděláme tak, že dopíšeme `app.UseSession();` do metody `Configure()`. Je doporučeno volat `UseSession()` mezi voláním middlewarů `UseRouting()` a `UseEndpoints()`. K session přistoupíme pomocí objektu `HttpContext.Session`.

V ASP.NET Core je práce se session oproti Laravelu trochu omezená. Můžeme ukládat jen data v typech `int`, `string` a `byte[]`. Pro uložení komplikovanějších dat lze třeba použít serializaci do stringu JSON. Data vložíme do úložiště pomocí metod `SetInt32()`, `SetString()` a `Set()`, které mají parametry klíč a hodnota. Získání dat probíhá obdobně pomocí metod `GetInt32()`, `GetString()` a `Get()` s parametrem klíč. Mazání položek probíhá pomocí metody `Remove()` s parametrem klíč. Všechna data lze odstranit z úložiště metodou `Clear()`.

```
//nastaveni v souboru Startup.cs
public void ConfigureServices(IServiceCollection services)
{
  services.AddDistributedMemoryCache();
  services.AddSession();
          //dalsi kod
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment
    env)
{
  app.UseRouting();
  app.UseSession();
  //dalsi middleware
  app.UseEndpoints(endpoints =>
  {
    endpoints.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
  });
}

//ulozeni dat
HttpContext.Session.SetString("car", "Audi");
HttpContext.Session.SetInt32("age", 10);
...
//ziskani dat
string car = HttpContext.Session.GetString("car");
int age = HttpContext.Session.GetString("age")
...
//mazani dat
HttpContext.Session.Remove("car");
HttpContext.Session.Clear();
...
```

Zdrojový kód 17: Práce se session v ASP.NET Core

**Mé dojmy**

Práce s úložištěm session byla podle mého pohodlnější v Laravelu. Důvodem bylo, že jsem do úložiště mohl ukládat pole a pole polí. V ASP.NET Core bylo nutné serializovat data do JSON stringu pro splnění stejného účelu.

# 5 Shrnutí

V této kapitole si shrneme veškeré klady a zápory, o kterých jsem se doposud zmínil. Ve prospěch Laravelu nahrává poměrně snadné vytvoření nového projektu. Velkou výhodou je také šablonovací systém Blade, který má velmi zdařilé rozlišení mezi kódem jazyka a HTML. Hodně dobře vymyšlená je také práce s databází. V neposlední řadě Laravel disponuje vynikající dokumentací, ve které lze nalézt vše celkem rychle.

U laravelu jsem nezpozoroval žádnou klíčovější nevýhodu, snad jen dynamické typování, ale to může být pro někoho výhoda.

Výhodou ASP.NET Core je možnost práce ve Visual Studiu, což je velmi zdařilé vývojové prostředí. K tomu se váže snadné založení projektu a velká přehlednost kódu s vhodným zvýrazněním klíčových slov. Další výhodou jsou staticky typované datové typy, díky nimž člověk přesně ví, co se nachází v dané proměnné. Práce s databází je taktéž velmi zdařilá.

Jednou z nevýhod ASP.NET Core je dle mého celkem obtížné proniknutí do systému práce v tomto frameworku. Další nevýhodou je nepřehledná dokumentace, která mi vůbec nevyhovovala.

Hlavní stránky frameworků jsou porovnány níže v tabulce.

| **Porovnávaná oblast** | **Laravel** | **ASP.NET Core** |
|---|---|---|
| Proniknutí do systému | rychlejší | pomalejší |
| Dokumentace | lepší | horší |
| Licence | MIT | MIT |
| Programovací jazyk | PHP | C# |
| Typovanost jazyka | dynamicky typovaný | staticky typovaný |
| Podpora databází | velká | široká podpora (NuGet balíčky) |
| Práce s databází | výborná | výborná |
| Šablonovací systém | Blade | Razor |
| Tvorba šablon | lepší | horší |
| Routování | horší | lepší |
| Rozšiřitelnost | Composer | NuGet balíčky |
| Zdrojové soubory | méně přehledné | přehlednější |

Tabulka 1: Závěrečné srovnání

# Závěr

Cílem této práce bylo představit a porovnat frameworky Laravel a ASP.NET Core při tvorbě webových stránek. Byla to pro mě výzva, protože jsem neměl zkušenosti ani s jedním z nich. Pochopení principů práce v obou zmíněných technologiích mi zabralo mnoho času, ale nakonec jsem do nich pronikl. Nejtěžší bylo naučit se základní mechanismy frameworků, poté už práce šla od ruky.

V textu práce jsme rozebrali klíčové pojmy a technologie potřebné k porozumění tématu. V další části byly popsány webové stránky, které jsem vytvořil pomocí zmíněných frameworků. Další a nejpíš nejdůležitější kapitolou byla ukázka, jak ve zmíněných technologiích probíhá tvorba webu. Z této kapitoly lze také vyčíst porovnání syntaxe a názvů metod daných frameworků. Poslední částí textu bylo shrnutí, ve kterém jsme si rozebrali hlavní klady a zápory jednotlivých technologií.

Po dokončení práce zjišťuji, že zde není jasný vítěz. Obě dvě technologie jsou velmi dobře použitelné a dají se pomocí nich vytvořit velké věci. Komfort práce v nich záleží na stylu práce a návycích uživatele. Po desítkách hodin práce ve frameworcích neupřednostňuji ani jednu z technologií. Zjistil jsem, že jsem díky této diplomové práci získal mnoho zkušeností do budoucnosti, které využiji v následné praxi.

# Conclusions

The aim of this work was to present and compare the Laravel and ASP.NET Core frameworks when creating websites. It was a challenge for me because I had no experience with any of them. It took me a long time to understand the principles of working in both of these technologies, but in the end I got into them. The hardest part was learning the basic mechanisms of the frameworks, then the work went hand in hand.

In the text of the thesis we discussed the key concepts and technologies needed to understand the topic. The next section describes the websites that I created using the mentioned frameworks. The next and most important chapter was a demonstration of how the web is created in the mentioned technologies. This chapter also provides a comparison of syntax and method names of given frameworks. The last part of the text was a summary in which we discussed the main pros and cons of each technology.

After finishing the work, I find that there is no clear winner. Both technologies are very useful and can be used to create great things. The comfort of work in them depends on the style of work and habits of the user. After tens of hours of working in the frameworks, I do not prefer any of the technologies. I found out that thanks to this diploma thesis I gained a lot of experience for the future, which I will use in my subsequent practice.

# A První příloha

Implementaci vytvořenou pomocí frameworku Laravel lze nalézt nahranou na bezplatném hostingu na odkaze https://laravel-shop.trialhosting.cz/. Pro přístup do administrativní části je připraven uživatelský účet `test@test.cz` s heslem `12345`.

# B Obsah přiloženého CD/DVD

**`ASPNetCore-shop/`**
: Složka projektu vytvořeného pomocí ASP.NET Core.

**`laravel-shop/`**
: Složka projektu vytvořeného pomocí frameworku Laravel.

**`readme.txt`**
: Informace ke zprovoznění.

**`laravel-shop.sql`**
: Soubor s databází.

# Literatura


[1] HTML: HyperText Markup Language [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[2] ŠTRÁFELDA, Jan. HTML [online]. Dostupné z: https://www.strafelda.cz/html

[3] CSS: Cascading Style Sheets [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[4] The Evolution of CSS in 3 Decades [online]. Dostupné z https://ageek.dev/css-evolution

[5] JavaScript [online]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript

[6] ŠTRÁFELDA, Jan. JavaScript [online]. Dostupné z: https://www.strafelda.cz/javascript

[7] jQuery [online]. Dostupné z: https://jquery.com/

[8] KOĎOUSKOUVÁ, Barbora. Architektura MVC: Definice, Struktura, Frameworky [online]. Dostupné z: https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky

[9] BERNARD, Borek. Úvod do architektury MVC [online]. Dostupné z: https://zdrojak.cz/clanky/uvod-do-architektury-mvc/

[10] ŠTRÁFELDA, Jan. PHP [online]. Dostupné z: https://www.strafelda.cz/php

[11] C Sharp (programming language) [online]. Dostupné z: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[12] Software framework [online]. Dostupné z: https://en.wikipedia.org/wiki/Software_framework

[13] Build fast, responsive sites with Bootstrap [online]. Dostupné z: https://getbootstrap.com/

[14] Oficiální stránky frameworku Laravel [online]. Dostupné z: https://laravel.com/

[15] Laravel neoficiálně a česky [online]. Dostupné z: https://laravel.blog.cz/

[16] LUPČÍK, Jan. Úvod do Laravel frameworku pro PHP [online]. Dostupné z: https://www.itnetwork.cz/php/laravel/uvod-do-laravel-frameworku-pro-php

[17] What is ASP.NET Core? [online]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core

[18] Oficiální dokumentace ASP.NET Core [online]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-6.0

[19] ČÁPKA, David. Úvod do Laravel frameworku pro PHP [online]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net-core/zaklady/tutorial-uvod-do-asp-dot-net

[20] PHPStorm: The Lightning-Smart PHP IDE [online]. https://www.jetbrains.com/phpstorm/

[21] Visual Studio [online]. Dostupné z: https://visualstudio.microsoft.com/cs/

[22] XAMPP Apache + MariaDB + PHP + Perl [online]. Dostupné z: https://www.apachefriends.org/index.html