# Anotace

KytaryEshop je jednoduchá webová aplikace implementující (některé funkcionality spojené s) webovým obchodem s kytarama. Aplikace má implementovány všechny základní funkce, které klient od aplikace potřebuje-bezpečné přihlášení, registraci, odhlášení, přidání/odebrání produktů do/z košíku, zobrazení obsahu košíku, potvrzení objednávky a zobrazení historie objednávek přihlášeného uživatele.

# Data aplikace

Naše aplikace ukládá data ve dvou databázích na lokálním SQL serveru. Jedná se o databáze **DatabazeKytary** a **KytaryEshop**:

**DatabazeKytary**: Ukládá všechna data spojená s eshopem, která se přímo netýkají uživatelského profilu. Řetězec spojení je obsažen v App.Config souboru v našem řešení. Samotná databáze pak obsahuje 4 tabulky.

1. Artikl : ukládá data ohledně prodejních artiklů. Obsahuje sloupce *IdArtikl*, *CenaKus*, *Znacka*, *NazevProdukt*, *KusuNaSklade* a *TypArtiklu*
   1. Typ Artiklu je implementován jako integer, kde 0-zastupuje elektrické kytary, 1-baskytary a 2-akustické kytary. Omezení na tento rozsah hodnot je dáno v definici tabulky.
2. Objednavka: ukládá základní informace spojené s proběhlými objednávkami. Obsahuje sloupce *Datum*, *IdObjednavka* a *IdUzivatel*.   
   Prodané artikly jsou posléze obsaženy v tabulce *ObjednavkaArtikl*.
3. ObjednavkaArtikl: K objednávce samozřejmě přísluší položky košíku, které si uživatel koupil. Tabulky *Artikl* a *Objednávka* jsou však ve vztahu m:n. Neboť každá objednávka může obsahovat více (typů) artiklu a každý (typ) artiklu může patřit do více objednávek. Protože relační databáze nabývají tvaru dvourozměrné tabulky, tak je třeba vztah mezi nimi rozbít na dva typy vztahů 1:n a 1:m s pomocnou tabulkou, která je propojuje dohromady. V našem případě se tedy jedná o tabulku *ObjednavkaArtikl*, která obsahuje data vázající se k položce objednávky. Obsahuje tedy sloupce *idArtikl*, *idObjednavka* (cizí klíče- spojující příslušné tabulky), *id* (primární klíč položky) a *PocetKusu*.
4. PolozkaKosiku: Tabulka položka košíku průběžně ukládá položky, které si uživatel přidal do košíku, ale ještě je nepředal k expedaci. Obsahuje 4 sloupce *Id* (což je přimární klíč), *IdArtikl*, *IdUzivatel* (což jsou cizí klíče označující uživatele a artikl v košíku) a *PocetKusu*.

**KytaryEshop**: Databáze KytaryEshop je vytvořena a udržována pomocí Identity Core frameworku Microsoftu. V dokumentaci ji tedy netřeba rozvádět.

# Implementace podrobně

Aplikace je rozdělena do dvou vrstev: Vrstva datového přístupu a ASP.NET MVC vrstva, implementující samotnou webovou aplikaci.

## Vrstva datového přístupu

V našem případě je datová vrstva (nacházející se ve složce backend) rozdělená do tří částí, které jsou uloženy v oddělených složkách:

1. **BModels** složka obsahuje modely záznamů z databáze **DatabazeKytary** naší aplikace. Jinými slovy daná složka obsahuje určité třídy. Ty nesou shodné jméno jako databáze, které modelují se sufixem BModel. (tzn. máme například UzivatelBModel). A jejich vlastnosti jsou pojmenovány a mají stejný typ jako sloupce tabulky, kterou modelují.
2. **Business logika** implementuje logiku přechodu mezi databází a programem (tzn. mezi tabulkou a našimi Modely). Jsou to tedy třídy údržby, skrze jejichž metody posléze manipulujeme s daty naší aplikace.

**Poznámka** Protože řetězec spojení potřebují všechny třídy naší Business logiky, tak manipulace s nimi (v našem případě pouze navrácení a ukončení nového spojení) je obsaženo ve vlastní třídě Data Access.

1. **Uložené procedury**: Všechny požadované procedury Business logiky jdou v zásadě rozbít na konečné množství jednoduchých SQL (pod)dotazů. Z důvodu robustnosti vzhledem bezpečnosti (např. obrana proti SQL injection útokům), optimalizaci z hlediska času na procesoru a z hlediska dlouhověkosti kódu, jsem tedy považoval za moudré uložit tyto dotazy do Programmability složky na lokálním SQL serveru (viz databáze DatabazeKytary).

**Poznámka(K Business logice)** Pro přístup k datům je v .NET frameworku vyhrazena architektura ADO.NET. S ní je ovšem ten problém, že u DQL dotazů dává výstup tvaru DataTable. Ten je pak potřeba skrze jednotlivé sloupce mapovat do našeho modelu, se kterým už jsme posléze schopni nějak programačně manipulovat. Jedná se o tzv. ORM (Objektově relační mapování). To však vyžaduje poměrně velké množství vcelku repetitivního kódu. Přesně k tomuto účelu tedy existují nástroje, které zmíněné mapování dělají za nás. V této aplikaci používáme přesně jednu takovou knihovnu s názvem Dapper. S jeho pomocí stačí pojmenovat sloupce tabulky stejně jako vstupní položky konstruktoru daného modelu (a zachovat typy). Generické funkce Dapperu se o nezbytné mapování pak starají za nás.

BModelsimplementuje objektovou reprezentaci pro tabulky databáze DatabazeKytary.

**Poznámka** Názvy vlastností v příslušných modelech a jejich datové typy odpovídají názvům sloupců, jež modelují. Dané třídy kromě potřebných vlastností a nezbytného konstruktoru neobsahují nic.

**Modely:**

*public class ArtiklBModel:* Model položky v tabulce dbo.Artikl.

*public class ObjednavkaBModel:* Model položky v tabulce dbo.Objednavka.

*public class PolozkaKosikuBModel:* Model položky v tabulce dbo.PolozkaKosiku.

*public class PolozkaObjednavkyBModel:* Model položky v tabulce dbo.ObjednavkaArtikl.

*public enum TypyArtiklu:* Typy artiklu, na základě kterých klasifikujeme položky v databázi. (jedná se o ElektrickeKytary, Baskytary, AkustickeKytary a Prislusenstvi)

BusinessLogikaBusiness logika pro každou tabulku DatabazeKytary, vyjma spojující tabulky ObjednavkaArtikl, implementuje statickou třídu přístupu k datům.

Business logika tedy obsahuje právě tři třídy:  
 ArtiklProcesor, ObjednávkaProcesor, PolozkaKosikuProcesor.

ArtiklProcesor*:* Třída datového přístupu k tabulce dbo.Artikl, jež obsahuje data vztahující se k prodejním artiklů našeho eshopu.

private static int ZmenPocetKusuO(int IdArtikl, int OKolikKusu)   
Obecně: Změní počet kusů na skladě o zadanou hodnotu.

Paramter IdArtikl: Identifikátor zboží, u kterého chceme změnit počet uskladněných kusů.

Paramter OKolikKusu: Počet, o který chceme změnit množství artiklu na skladě.

Návratová hodnota: Počet záznamů v tabulce, u kterých došlo ke změně.

public static int ProdejArtikl(int idArtikl, int pocet)

Obecně: Sníží počet kusů na skladě o zadanou hodnotu.

Parametr idArtikl: Identifikátor zboží, u kterého chceme změnit počet uskladněných kusů.

Parametr pocet: Počet, o který chceme snížit množství artiklu na skladě.

Návratová hodnota: Počet záznamů tabulky, u kterých došlo ke změně.

public static int ProdejArtikl(int idArtikl, int pocet, SqlConnection spojeni, SqlTransaction transakce

Obecně: Sníží počet kusů na skladě o zadanou hodnotu v rámci probíhající transakce.

Parametr idArtikl: Identifikátor zboží, u kterého chceme změnit počet uskladněných kusů.

Parametr pocet: Počet, o který chceme snížit množství artiklu na skladě.

Parametr spojeni: Spojení, na kterém probíhá aktuální transakce

Parametr transakce: Transakce, v rámci které má být metoda vykonána.

Návratová hodnota: Počet záznamů tabulky, u kterých došlo ke změně.

private static int ZmenPocetKusuO(int IdArtikl, int OKolikKusu, SqlConnection spojeni, SqlTransaction transakce)

Obecně: Sníží počet kusů na skladě o zadanou hodnotu v rámci probíhající transakce.

Parametr idArtikl: Identifikátor zboží, u kterého chceme změnit počet uskladněných kusů.

Parametr OKolikKusu: Počet, o který chceme změnit množství artiklu na skladě.

Parametr spojeni: Spojení, na kterém probíhá aktuální transakce.

Parametr transakce: Transakce, v rámci které má být metoda vykonána.

Návratová hodnota: Počet položek tabulky, u kterých došlo ke změně.

public static ArtiklBModel NactiArtikl(int idArtikl)

Obecně: Načte artikl se specifikovaným identifikátorem. Pokud daný artikl není na skladě, tak vrací null.

Parametr idArtikl: Identifikátor artiklu, který chceme načíst.

Návratová hodnota: Načtený artikl (může být null).

public static List<ArtiklBModel> NactiArtikly(int offset, int pocet, TypyArtiklu typ)

Obecně: Načte specifikovaný počet artiklů určitého typu z databáze s tím, že přeskočí zadaný počet položek.

Parametr offset: Počet artiklů, které má metoda při načítání přeskočit.

Parametr pocet: Počet artiklů, které má metoda načíst.

Parametr typ: Typ artiklu, který má metoda načíst.

Návratová hodnota: Seznam artiklů na skladě s příslušnými indexy.

public static List<ArtiklBModel> NactiArtikly(int offset, int počet)

Obecně: Načte specifikovaný počet artiklů z databáze s tím, že přeskočí zadaný počet položek.

Parametr offset: Počet artiklů, které má metoda při načítání přeskočit.

Parametr pocet: Počet artiklů, které má metoda načíst.

Návratová hodnota: Seznam artiklů na skladě s příslušnými indexy.

public static int PocetArtiklu(TypyArtiklu typ)

Obecně: Zjistí, počet různých artiklů (ve smyslu druhu položky - př. Gibson Les Paul) na skladě od určitého typy artiklu (např. baskytara).

Parametr typ: Typ artiklu, jehož počet chceme načíst.

Návratová hodnota: Počet artiklů zadaného typu.

public static int PocetArtiklu()

Obecně: Zjistí, počet různých artiklů (ve smyslu druhu položky - př. Gibson Les Paul) na skladě přes všechny typy artiklu (např. baskytara).

Návratová hodnota: Počet různých artiklů na skladě.

public static int CetnostArtiklu(int id)

Obecně: Zjistí počet produktů na skladě od artiklu se zadaným identifikátorem.

Parametr id: Id artiklu, jehož četnost chceme zjistit

Návratová hodnota: Četnost artiklu se zadaným identifikátorem.

public static int CenaArtiklu(int id)

Obecně: Zjistí prodejní cenu artiklu se zadaným identifikátorem.

Parametr id: Id artiklu, jehož četnost chceme zjistit

Návratová hodnota: Cena artiklu se zadaným identifikátorem.

#### ObjednavkaProcesor

public static bool PridejObjednavku(DateTime datum, string IdUzivatel, List<PolozkaKosikuBModel> polozkyKosiku)

Obecně: Zařadí objednávku a její položky do databáze. V případě úspěšného vyřízení objednávky vrátí true, v opačném případě false.

Parametr datum: Datum, ve kterém došlo k vytvoření objednávky.

Parametr IdUzivatel: Identifikátor uživatele, jež přidal objednávku.

Parametr polozkyKosiku: Seznam položek, které si uživatel objednal.

Návratová hodnota: Informace o tom, jestli vytvoření záznamu o objednávce proběhlo v pořádku.

**Poznámka(K implementaci)** Pro zavedení objednávky do databáze je třeba nejprve vytvořit záznam o objednávce v tabulce Objednavka, pak implementovat snížení počtu zakoupených artiklů na skladě v databázi Artikl a posléze vytvořit pohledávky v tabulce ObjednavkaArtikl. Jestliže selže jeden z příslušných kroků, je třeba, aby vše bylo vráceno do původního stavu. Tedy chceme, aby se buď nevykonal žádný z kroků, nebo všechny. Z tohoto důvodu je tato metoda implementována jako transakce.

private static int VytvoreniZaznamuVDatabaziObjednavka(string IdUzivatel, DateTime datum, SqlConnection spojeni, SqlTransaction transakce)

Obecně: Vytvoří záznam o objednávce v tabulce dbo.Objednavka v rámci probíhající transakce.

Parametr IdUzivatel: Identifikátor uživatele, jež přidal objednávku.

Parametr datum: Datum, při kterém byla objednávka vytvořena.

Parametr spojeni: Spojení, na kterém probíhá transakce.

Parametr transakce: Transakce, v rámci které je třeba vykonat dotaz.

Návratová hodnota: Vrací počet záznamů, které byly vytvořeny.

private static bool VytvoreniZaznamuVObjednavkaArtikl(int idObjednavka, int IdArtikl, int PocetKusu, SqlConnection spojeni, SqlTransaction transakce)

Obecně: Ukládá položku objednávky do tabulky dbo.ObjednavkaArtikl v rámci probíhající transakce.

Parametr idObjednavka: Identifikátor objednávky, k níž patří daná položka.

Parametr idArtikl: Identifikátor artiklu, příslušícího výsledné položce objednávky.

Parametr PocetKusu: Počet kusů artiklu, jež byly objednány.

Parametr spojeni: Spojení, v rámci kterého probíhá transakce.

Parametr transakce: Transakce, v rámci které je třeba vykonat dotaz

Návratová hodnota: Informace o tom, jestli byla položka v tabulce dbo.Objednávka vytvořena úspěšně.

public static List<PolozkaObjednavkyBModel> NactiPolozkyPodleIdObjednavky(int idObjednavka)

Obecně: Načte z databáze položky objednávky se specifikovaným identifikátorem.

Parametr idObjednavka: Identifikátor načítané objednávky.

Návratová hodnota: Seznam položek objednávky se specifikovnaným identifikátorem.

public static List<ObjednavkaBModel> NacteniObjednavekVykonanychUzivatelem(string IdUzivatel)

Obecně: Načte z databáze seznam objednávek vykonaných uživatelem se specifikovaným identifikátorem.

Parametr IdUzivatel: Identifikátor uživatele, jehož objednávky chceme načíst.

Návratová hodnota: Seznam objednávek, vytvořených uživatelem s identifikátorem zadaným ve vstupu.

#### PolozkaKosikuProcesor

public static class PolozkaKosikuProcesor

Obecně: Třída datového přístupu k tabulce dbo.PolozkaKosiku obsahující data o položkách košíku jednotlivých uživatelů.

public static void UlozPolozkuUzivatele(int idArtikl, int pocet, string idUzivatel)

Obecně: Uloží položku košíku uživatele se zadaným identifikátorem do databáze. Položka se sestává z identifikátoru artiklu a jeho kvantity v košíku (a identifikátoru uživatele).

Parametr idArtikl: Identifikátor artiklu, ke kterému se daná polžka vztahuje.

Parametr pocet: Kvantita artiklu v košíku.

Parametr idUzivatel: Identifikátor uživatele, o jehož košík se jedná.

public static List<PolozkaKosikuBModel> NactiPolozkyKosikuUzivatele(string idUzivatel)

Obecně: Načte z databáze položky košíku uživatele, jehož identifikátor se nachází na vstupu. V případě zjištění, že některá z položek se již nenachází na skladě v dostatečné kvantitě, sníží podle potřeby příslušnou kvantitu v košíku na maximální možnou hodnotu.

Parametr idUzivatel: Identifikátor uživatele, jehož položky košíku chceme načíst

Návratová hodnota: Seznam položek košíku zadaného uživatele.

public static bool SmazPolozkuZKosiku(int idArtikl, string idUzivatel)

Obecně: Smaže položku košíku vztahující se k artiklu a uživateli se zadanými identifikátory.

Parametr idArtikl: Identifikátor artiklu, ke kterému se váže mazaná položka košíku

Parametr idUzivatel: Identifikátor uživatele, ke kterému se váže mazaná položka košíku

Návratová hodnota: Informace o tom, jestli položka košíku byla smazána úspěšně.

public static bool ZmenKvantitu(int idArtikl, string idUzivatel, int pocetKusu)

Obecně: Položce košíku vztahující se uživateli a artiklu se zadanými identifikátory nastaví tato metoda v databázi kvantitu na zadaný počet prvků.

Parametr idArtikl: Identifikátor artiklu, kterému se váže odkazovaná košíku.

Parametr idUzivatel: Identifikátor uživatele, ke kterému se váže odkazovaná položka košíku.

Parametr pocetKusu: Počet kusů, na který chceme změnit kvantitu košíku odkazované položky.

Návratová hodnota: Informace o tom, jestli kvantita položky košíku byla změněna úspěšně.

public static int ZjistiKvantitu(int idArtikl, string idUzivatel)

Obecně: Zjistí kvantitu položky košíku vztahující se k artiklu a uživateli se zadanými identifikátory.

Parametr idArtikl:Identifikátor artiklu, jehož kvantitu v košíku chceme zjistit.

Parametr idUzivatel: Identifikátor uživatele jemuž přísluší odkazovaná položka košíku.

Návratová hodnota: Zjištěná kvantita dotazované položky košíku.

public static bool SmazPolozkyZKosikuUzivatele(string idUzivatel)

Obecně: Smaže z tabulky položky košíku uživatele se zadaným identifikátorem.

Parametr idUzivatel: Identifikátor uživatele, jehož položky chceme smazat.

Návratová hodnota: Informace, jestli smazání proběhlo úspěšně.

public static bool SmazPolozkyZKosikuUzivatele(string idUzivatel, SqlConnection spojeni, SqlTransaction transakce)

Obecně: Smaže položky z košíku uživatele se zadaným identifikátorem v rámci probíhající transakce.

Parametr idUzivatel: Identifikátor uživatele, jehož položky chceme smazat.

Parametr spojeni: Spojení, na kterém probíhá aktuální transakce.

Parametr transakce: Transakce, v rámci které má být metoda vykonána.

Návratová hodnota: Informace o tom, jestli smazání proběhlo úspěšně.

# MVC

Naše webová aplikace je implementována podle MVC architektury. Zkratka MVC znamená Model-View-Controller. Kde Model je datová struktura (front-endu), View je soubor definující, jak má vypadat stránka, která se zobrazí klientovi v prohlížeči a Kontroler je třída, která zpracovává příchozí dotazy a odesílá vhodnou odpověď (opět ve formě zobrazení).

### Podrobněji

ModelVětšina stránek zpravidla souvisí s nějakým úkonem, který se poměrně přirozeně váže k některému z frontendových modelů. Jedna stránka třeba typicky bude odpovídat registraci uživatele. To je tedy stránka, která přímo koresponduje s nějakými daty, v našem případě modelem (resp. třídou) uživatele. Podobně stránka zobrazující obsah nákupního košíku bude zase korespondovat s nějakým modelem seznamu položek nákupního košíku.

**Poznámka**  Model je tedy prostá C# třída která bývá krom konstruktorů prostá dalších metod.

**Poznámka** Od frontendového modelu jsou zpravidla vyžadovány trochu odlišné vlastnosti než od backendového modelu. Frontendový model uživatele, který budeme používat k verifikaci dat z formuláře, například bude obsahovat anotace s chybovými hláškami. Ty se se pak uživateli zobrazují v rámci formuláři v případě, kdy zadá chybné údaje. Od backendového modelu, zas vyžadujeme konstruktory, které souvisí s objektovým mapování vztahujícím se k databázi. Z odlišných požadavků na příslušné třídy se tedy doporučuje mít frontendové a backendové modely oddělené. Tento přístup má tu nevýhodu, že vede k poměrně repetitivnímu kódu. I přesto se však domnívám, že se jedná o čistší implementaci, která zachovává dělbu na frontend a backend.

ZobrazeníZobrazení určuje, jak se mají data z modelu vykreslit klientovi v prohlížeči. Typicky se jedná o soubory typu .cshtml (c# html). To je jakási směsice html kódu, neboli té „statické“ části struktury stránky a C# kódu. Skrze C# udáváme serveru určité výrazy k ohodnocení. Výstupem tohoto výpočtu je opět html kód. Nepochybnou výhodou tohoto kódu je, že server jej vyhodnocuje „skrytě“. Neboli klient se o něm nemá jak dozvědět. V zásadě jsme tak schopni skrýt vnitřní strukturu naší aplikace před klientem.

KontrolerMezi jednotlivými zobrazeními se obvykle přechází pomocí určitých dotazů vyvolaných pomocí interaktivních prvků na stránce– jako je tlačítko. Tyto dotazy bývají následně zpracovány metodou (tzv. akcí) kontroleru, na kterou daný dotaz odkazuje. Daná akce nějak zpracuje dotaz, případně upraví data aplikace a posléze odešle zpátky odpověď opět ve formě zobrazení. Tím završuje cyklus stránky.

**Poznámka:** Http je bezstavový protokol. Jinými slovy, nedojde-li k nějaké manipulaci s daty v databázi serveru, tak příští vyvolání této akce bude na kontroleru odpovídat stejné zobrazení.

**Definice** Kontroler je pouze třída, která dědí od třídy Controller.

**Poznámka** Tato aplikace je implementována v souladu s konvencí, kdy každému modelu přísluší shodně pojmenovaný kontroler se sufixem Controller. Tzn. například modelu Uživatel přísluší kontroler UzivatelController a ten zpracovává dotazy, které se týkají modelu Uživatel.

ArtiklyController:Kontroler zpracovávající žádosti směřující ke zobrazení jednotlivých stránek katalogu našeho eshopu. Katalog je implementován tak, že jednotlivé artikly jsou sdružovány do skupin po 12. Podle toho, jaký seznam si uživatel zvolí, se mu může zobrazit seznam baskytar, akustických kytar či elektrických kytar.

public IActionResult VratVhodnouStranku(Kytary.Backend.BModels.TypyArtiklu typ, int KolikataStranka)

Obecně: Na vstupu akce obdrží typ artiklů na skladě, které si chce uživatel zobrazit a o kolikátou stránku seznamu se jedná. Artikly jsou sdružovány do skupin po 12. Na výstup dá zobrazení s příslušnými artikly a výstupními daty.

Parametr typ: Typ artiklu, které si chce uživatel zobrazit.

Parametr KolikataPolozka: Kolikátou stránku katalogu si chce uživatel zobrazit.

Návratová hodnota: Zobrazení příslušné stránky katalogu.

public IActionResult VratVhodnouStranku(Kytary.Backend.BModels.TypyArtiklu typ, int KolikataStranka)

Obecně: Akce vrací některou ze stránek seznamu baskytar, které má eshop na skladě. Artikly jsou sdružovány do stránek po 12. Na vstupu akce obdrží kolikátou stránku seznamu si chce uživatel zobrazit, na výstup dá její zobrazení společně s příslušnými daty.  
Parametr KolikataPolozka: Kolikátou stránku katalogu si chce uživatel zobrazit.  
Návratová hodnota: Zobrazení požadované stránky katalogu.

public IActionResult StrankaElektrickeKytary(int KolikataPolozka)

Obecně: Akce vrací některou ze stránek seznamu elektrických kytar, které má eshop na skladě. Artikly jsou sdružovány do stránek po 12. Na vstupu akce obdrží kolikátou stránku seznamu si chce uživatel zobrazit, na výstup dá její zobrazení společně s příslušnými daty.

Parametr KolikataPolozka: Kolikátou stránku katalogu si chce uživatel zobrazit.

Návratová hodnota: Zobrazení požadované stránky katalogu.

public IActionResult StrankaAkustickeKytary(int KolikataPolozka)

Obecně: Akce vrací některou ze stránek seznamu akustických kytar, které má eshop na skladě. Artikly jsou sdružovány do stránek po 12. Na vstupu akce obdrží kolikátou stránku seznamu si chce uživatel zobrazit, na výstup dá její zobrazení společně s příslušnými daty.

Parametr KolikataPolozka: Kolikátou stránku katalogu si chce uživatel zobrazit.

Návratová hodnota: Zobrazení požadované stránky katalogu.

public IActionResult StrankaVsechnyArtikly(int KolikataPolozka)

Obecně: Akce vrací některou ze stránek seznamu směsi artiklů, které má eshop na skladě. Artikly jsou sdružovány do stránek po 12. Na vstupu akce obdrží kolikátou stránku seznamu si chce uživatel zobrazit, na výstup dá její zobrazení společně s příslušnými daty.

Parametr KolikataPolozka: Kolikátou stránku katalogu si chce uživatel zobrazit.

Návratová hodnota: Zobrazení požadované stránky katalogu.

KosikController: Kontroler zpracovávající žádosti směřující ke zobrazení a případné modifikaci košíku přihlášeného uživatele. Obsahuje dependency injection.

**Poznámka**(k implementaci) To, jakým způsobem je vhodné implementovat košík uživatele, se liší na základě určitých parametrů. Zásadní problém s implementací košíku je v bezstavovosti HTTP protokolu. Data z košíku jsou podobně jako jiná data příslušná přihlášenému uživateli, společná pro všechna dílčí zobrazení. Tzn. data košíku musí být nutně uložena nějak mimo HTML kód jednotlivých stránek. Na to existují v zásadě tři recepty.

1. Košík lze uložit v relaci (anglicky Session). To znamená že data jsou uložena v paměti serveru po dobu, po jakou je uživatel přihlášen. Což má tu zřejmou nevýhodu, že při vyšším počtu aktivních uživatelů roste zásadním způsobem i nárok na paměť. Tím se snižuje celková efektivita aplikace. Což jinými slovy jde proti výhodám http protokolu. Dalším problémem je, že data relace jsou po ukončení spojení ze serveru smazána. Jinými slovy by tak uživatel po odhlášení či zavření prohlížeče přišel o všechna data z košíku.
2. Další možností je uložit data v Cookies prohlížeče, tzn. na straně klienta. To by svým způsobem řešilo problém ztráty dat při ukončení relace. Problém je, že klient může mít Cookies zablokovány. Může mezi dvěma připojeními na náš eshop použít dvě různá zařízení. Případně si v nějakou dobu může vyčistit Cookies z prohlížeče. Ve všech těchto případech nebude pak schopen použít data z předešlého výběru.
3. Z tohoto důvodu jsem košík implementoval tak, že uživatel má své položky košíku uloženy v databázi na serveru. Při každém dotazu přistupuje server k databázi a načítá z ní příslušná data nanovo. Když si uživatel vyžádá změnu kvantity v košíku, tak dochází prvotně ke změně dané kvantity v databázi na serveru.   
   Jednou z nevýhod této implementace je navýšení času potřebného k odeslání vhodného zobrazení při dotazech, které vyžadují data z košíku. Server totiž musí přistupovat k datům na pevném disku, což trvá déle, než přístup k datům v paměti (ať už na straně serveru, či klienta). Stejně se však domnívám, že výše uvedené výhody značně převyšují tento drobný nedostatek.

public IActionResult ZobrazKosik()

Obecně: Načte z databáze prvky košíku přihlášeného uživatele, převede je na typ frontendového modelu kosikKomplet a tato data společně se zobrazením ZobrazKošík dá na výstup.

public int MinusTlacitkoKontroler(int idArtikl, int pocet)

Obecně: Metoda zpracovává požadavek na dekrementaci počtu artiklů v tabulce košíku přihlášeného uživatele. Vrací výsledný počet artiklů.

Parametr idArtikl: Identifikátor artiklu, jehož četnost v košíku chceme snížit.

Parametr počet: Aktuální četnost odkazovaného artiklu v košíku.

Návratová hodnota: Počet artiklů v tabulce košíku po skončení této metody.

public int PlusTlacitkoKontroler(int idArtikl, int pocet)

Obecně: Metoda zpracovává požadavek na inkrementaci počtu artiklů s identifikátorem idArtikl v tabulce košíku přihlášeného uživatele. Vrací výsledný počet artiklů.

Parametr idArtikl: Identifikátor artiklu, jehož četnost v košíku chceme zvýšit.

Parametr počet: Aktuální četnost odkazovaného artiklu v košíku.

Návratová hodnota: Počet artiklů v tabulce košíku po skončení této metody.

public IActionResult VyridObjednavku()

Obecně: Předá uživatelův košík k odbavení do datové vrstvy (metodě ObjednavkaProcesor.PridejObjednavku). Posléze vrátí zobrazení informující uživatele, o tom, jestli vyřízení objednávky proběhlo úspěšně.

Návratová hodnota: Zobrazení informující klienta o průběhu vyřízení objednávky.

UzivatelController: Kontroler zpracovávající žádosti směřující k uživatelským profilům - v našem případě pouze k přihlášení a registraci. Obsahuje dependency injection. Jinými slovy obsahuje proměnné \_userManager a \_signInManager skrze které přistupujeme k bezpečnostním metodám Identity Coru. To je framework Asp.Net, který se stará o bezpečné přihlášení, registraci, atp. Danému postupu se říká injection, neboť tyto proměnné jsou umístěny do konstruktoru kontroleru.

public IActionResult Registrace()

Obecně: Akce zpracovávající žádost ke zobrazení stránky s formulářem registrace uživatele

Návratová hodnota: Vrací stránku s formulářem registrace uživatele.

[HttpPost]

public async Task<IActionResult> Registrace(UzivatelModel vystup)  
Obecně: Verifikace dat zadaných do vstupního formuláře zobrazení Registrace.cshtml. V případě kladného výstupu verifikace následuje vytvoření účtu a přesměrování na domovskou stránku, nebo vrácení na zobrazení Registrace společně s výčtem chyb.  
Parametr vystupZFormulare: Frontendový model registrovaného uživatele sloužící k verifikaci a ukládání dat uživatelského vstupu.  
Návratová hodnota: V závislosti na průběhu verifikace buď úvodní stránka, nebo stránka registrace.

public IActionResult Prihlaseni()  
Obecně: Akce zpracovávající žádost ke zobrazení stránky s formulářem přihlášení uživatele  
Návratová hodnota: Vrací stránku s formulářem přihlášení uživatele.

[HttpPost]

public async Task<IActionResult> Prihlaseni(UzivatelModel vystupZFormulare)

Obecně: Verifikace dat zadaných do vstupního formuláře zobrazení Prihlaseni.cshtml. V případě kladného výstupu verifikace následuje přihlášení do systému a přesměrování na domovskou stránku, nebo vrácení na zobrazení Prihlaseni.cshtml společně s výčtem chyb.

Parametr vystupZFormulare: Frontendový model přihlašovaného uživatele sloužící k verifikaci a ukládání dat uživatelského vstupu.

Návratová hodnota: V závislosti na průběhu verifikace buď úvodní stránka, nebo stránka přihlášení.

[Authorize]

public async Task<ActionResult> ZobrazHistoriiObjednavek()  
Obecně: Zpracovává žádost ke zobrazení historie objednávek přihlášeného uživatele.  
Návratová hodnota: Stránka s přehledem objednávek provedených přihlášeným uživatelem.

### Views

Každému kontroleru přísluší podsložka ve složce Views, která nese stejný název. V ní se nachází .cshtml soubory definující vzhled jednotlivých stránek. Ty slouží jako odpovědi na dotazy pro jednotlivé kontrolery. Jinými slovy například ke Kontroleru UzivatelController existuje složka Views->Uzivatel, ve které se nachází zobrazení Prihlaseni.cshtml a Registrace.cshtml.

**Poznámka k implementaci**: Všechna zobrazení byla v zásadě navržena jednoduchým vanila HTML a CSS v editoru Atom. Návrh jsem ze cvičných důvodů vytvářel bez použití externých CSS knihoven (včetně bootstrapu). Rozložení stránky bylo povětšinou navrženo technologií cssGrid.   
Protože struktura CSS a HTML kódu je poměrně přímočará, tak nebudu vypisovat dokumentaci po jednotlivých souborech, jak tomu bylo u objektového návrhu. Okomentuji pouze ty části kódu, které si to žádají.

**Layout:** Ve složce Shared se nachází částečná zobrazení. V našem případě se jedná pouze o zobrazení \_Layout.cshtml. To je kód definující layout pro naše stránky. Kde layoutem myslíme tu část stránky (menu a patka), která je společná pro všechna dílčí zobrazení.

Příslušný kód, který posléze slouží jako odpověď na jednotlivé dotazy se pak zobrazuje na místě, označeném příkazem

@RenderView

**Poznámka:** Znak @ uvozuje C# kód v rámci našeho .cshtml kódu. Jedná se o tzv Razor syntax.

**Data:** Data příslušná ke stránce jsou povětšinou uložena v modelu. Příslušný model bývá uvozován klíčovým slovem Model (na začátku zobrazení Registrace.cshtml tomu například odpovídá kód @Model UzivatelModel).   
Data, která se do modelu nevejdou, se výjimečně vkládají do ViewBagu. Což je ovšem seznam dynamického typu (datový typ každé z jeho proměnných je určován až za běhu) a práce s ním je tedy pro nějaký větší objem dat značně neefektivní. V našem případě však jeho použití bylo nezbytné pouze v jednom případě a to u zobrazení Artikly->ZobrazeniArtiklu.cshtml, kde se ve ViewBagu nachází typ položky (Informace o tom, jestli se jedná o stránku katalogu pro akustické či například elektrické kytary).

**Skripty**

1. **AJAX:** Zobrazení *Kosik->ZobrazKosik.cshtml* a *Artikly->ZobrazeniArtiklu.cshtml* vztahující se k výčtu prvků katalogu obsahují tlačítka plus-mínus. Ta nastavují četnost prvku v košíku uživatele. V tomto případě bylo nezbytné použít JQuery a nastavit posluchače události poklepání na příslušná tlačítka. JQuery umožňuje přiřadit společný posluchač pro celou třídu naráz.   
   *Syntax:* $(".minusTlacitkoArtikly").click( … ) )   
   Ve chvíli, kdy se tak stane, je třeba , aby příslušná funkce javascriptu, nějak předala řešení dál na server, který je schopen komunikovat s databází. Jinými slovy je třeba nějak z HTML stránky vyvolat akci kontroleru a posléze (bez načítání stránky nové) nastavit výslednou kvantitu ve formě HTML kódu na příslušné místo stávající HTML stránky. K tomu slouží v rámci JavaScriptu technologie Ajax. Ta právě obsahuje metody, které nám umožňují komunikaci s webovým serverem. Výslednou četnost, kterou nám dá server jako odpověď, pak příslušný EventHandeler vkládá na patřičné místo v HTML kódu.

**Poznámka (Regex)** Ajax musí předat dané akci kontroleru identifikátor artiklu, který si chtěl uživatel přidat do košíku. Ten je uložen v id atributu HTML elementu daného tlačítka. Event handler se pak k danému číslu dostává skrze jednoduchý Regex výraz. "[0-9]+";

**Formuláře:** Asp.Net framework umožňuje některé věci, které by jinak člověk musel dlouze vypisovat, napsat v poměrně komprimované podobě. Typickým příkladem, který zde využíváme jsou formuláře. Místo toho, aby člověk implementoval verifikační skripty, k tomu již ASP.NET dodává předpřipravé knihovny. Práce s formulářem je potom nesmírně jednoduchá. Prve formulář uvozujeme skrze klíčové slovo:  
 @using (Html.BeginForm()) { … }

Každá položka formuláře je pak uvnitř elementu <div class="form-group">. Na místo, na kterém chceme zobrazit popisek vstupu, vkládáme @Html.LabelFor( … ). Tam, kde má být editační položka píšeme @Html.EditorFor( … ). A tam, kde chceme v případě chybné validace vlastnosti našeho modelu vložit chybovou hlášku @Html.ValidationMessageFor( … ).