



Střední průmyslová škola a Vyšší odborná škola, Písek, Karla Čapka 402, Písek

18-20-M/01 Informační technologie

Maturitní práce

Správa povinných prací

Téma číslo 1

autor:

Richard Kropáček, B4.I

vedoucí maturitní práce:

Mgr. Milan Janoušek

Písek 2020/2021

Anotace

Maturitní práce se zaměřuje na návrh a realizaci webové aplikace, která má na starosti správu povinných prací odevzdaných žáky. Vyučující má možnost vytvořit zadání a případně ho upravit, nebo odstranit. Žáci mají možnost procházet jak aktuální práce, tak práce, které již odevzdali například v minulých ročnících.

Klíčová slova: C#; ASP.NET Core; Entity Framework Core; HTML; JavaScript; CSS; Bootstrap; MySQL; WebApp

Annotation

The graduation thesis focuses on the design and implementation of a web application, which is in charge of managing the compulsory work submitted by students. The teacher has the opportunity to create an assignment and possibly modify or delete it. Pupils have the opportunity to go through both current work and work that they have already submitted, for example, in previous years.

Keywords: C#; ASP.NET Core; Entity Framework Core; HTML; JavaScript; CSS; Bootstrap; MySQL; WebApp

Poděkování

Tímto bych chtěl poděkovat Mgr. Milanu Janouškovi za vedení mé Maturitní práce, cenné rady a odborný dohled. Děkuji také za pomoc všem, kteří se jakkoliv podíleli na realizaci této práce.

Obsah

1	Úvod	5
2	Teoretický úvod	6
2.1	Programovací jazyk C#	6
2.2	Razor	6
2.3	ASP.NET, ASP.NET CORE	7
2.3.1	ASP.NET	7
2.3.2	ASP.NET Core	8
2.4	EF6 a EF Core	8
2.4.1	Entity Framework 6	8
2.4.2	Entity Framework Core	8
2.5	MVC	8
2.5.1	Model	9
2.5.2	View	9
2.5.3	Controller	10
2.6	MySQL	11
3	Základní struktura ASP.NET Core	12
3.1	Model	12
3.2	View	14
3.2.1	_Layout	15
3.2.2	_LoginPartial	16
3.2.3	_MenuPartial	17
3.3	Controller	19
3.3.1	Seznam entit	21
3.3.2	Detail	22
3.3.3	Editace	24
3.3.4	Smazání	27

4	Přihlašování a registrace	30
4.1	Přihlašování	30
4.2	Registrace	30
5	Role	31
5.1	Student	31
5.2	Učitel	31
5.3	Administrator	31
5.4	Vedení školy	31
6	Databáze	32
6.1	Připojení databáze	32
6.2	Návrh databáze	32
7	Závěr	33
	Přílohy	38
A	Příloha	39

Kapitola 1

Úvod

Výsledkem této maturitní práce by měla být plně funkční webová aplikace, která by umožňovala efektivní správu povinných prací, které studenti odevzdávají za všechny roky studia.

Pro realizaci jsem se rozhodl využít otevřený framework pro tvorbu webových aplikací APS.NET Core vyvinutý společností Microsoft Corporation. Pro tento framework jsem se rozhodl hlavně díky jeho jednoduché implementaci, možnosti ho vyvíjet a spouštět na platformách Windows, Linux i macOS a vysokému výkonu.

Pro návrh stránky využiji architektonický vzor MVC, ten mi pomůže oddělit logiku webové aplikace (Backend) od výstupu (Frontend). Pro to, aby tento vzor byl účinný, musím si rozvrhnout oprávnění, tedy role, které se ve webové aplikaci budou vyskytovat.

Mimo jiné musím vyřešit způsob registrování a následného přihlašování uživatelů, tak aby se eliminovala možnost neoprávněného přihlášení. To vše ale nebude správně fungovat, pokud nebudu mít připravenou databázi. Tu mám v plánu vytvořit v MySQL, což je velice rozšířený otevřený systém řízení báze dat.

A toto všechno je potřeba vložit do uživatelsky přívětivého rozhraní, které bude jednoduché na správu a zároveň přehledné a profesionální. S tímto mi pomůže Bootstrap 4, což je front-endový open-source určený pro návrh responzivních webů.

Kapitola 2

Teoretický úvod

2.1 Programovací jazyk C#

C# je moderní, vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý společností Microsoft. Jazyk C# se řadí mezi typově bezpečné programovací jazyky. To znamená, že nedovoluje provádět operace, které mohou vést k chybám. Tento jazyk umožňuje vývojářům vytvářet mnoho druhů zabezpečených a robustních aplikací, které běží na platformě .NET¹.

C# je zároveň objektově orientovaný programovací jazyk orientovaný na součásti². Z toho vyplývá, že se zaměřuje na vytváření komponent, které jsou tvořeny často se opakujícími částmi kódu. C# také poskytuje jazykové konstrukce pro přímou podporu těchto konceptů, což z něj dělá přirozený jazyk pro tvorbu a používání těchto softwarových komponent.[14]

Právě pomocí tohoto jazyka se tvoří backend, tedy celá logika webové aplikace založené na ASP.NET Core.

2.2 Razor

Razor je syntaxe pro ASP.NET používaná pro tvorbu dynamických webových stránek společně s programovacím jazykem C#. Pomocí Razor syntaxe lze vložit do webové stránky blok kódu, který se provede na straně serveru. Soubory využívající tuto syntaxi mají příponu .cshtml.

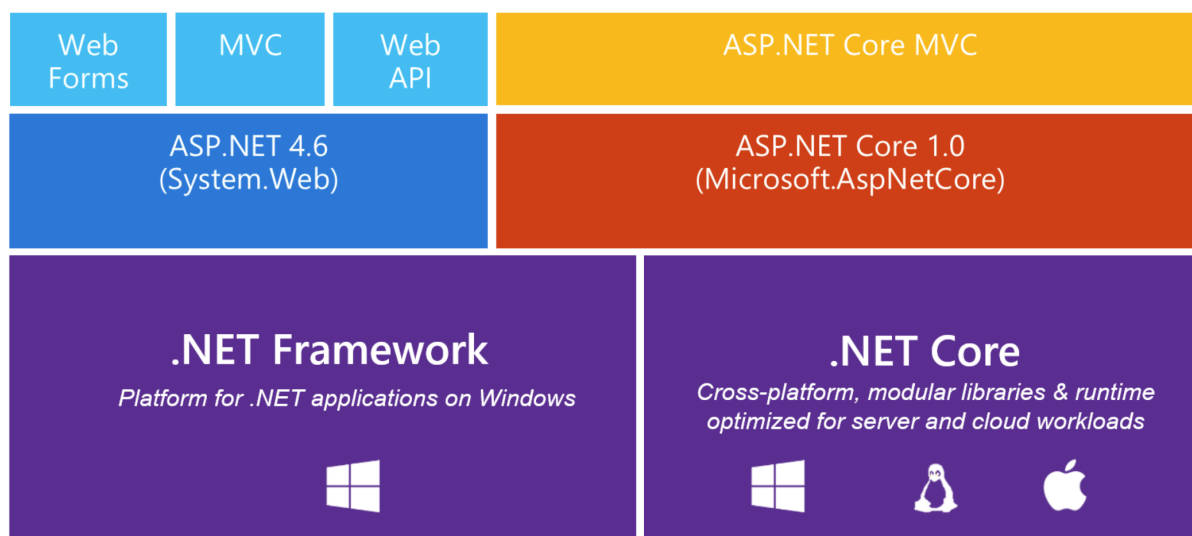
Razor syntaxe se skládá z Razor značek, C# a HTML. Výchozím Razor jazykem je

¹Vývojářská platforma pro vytváření webových, mobilních, desktopových, herních, IoT a dalších aplikací. Je podporovaná v systémech Windows, Linux a macOS.

²Component-oriented programming language

HTML. Pro označení C# kódu využijeme symbolem @. Tímto dojde k přechodu z formátu HTML do C#. Razor vyhodnotí výrazy jazyka C# a vykresluje je ve výstupu HTML. Kód HTML v souborech s příponou .cshtml se vykreslí serverem beze změny.[2]

2.3 ASP.NET, ASP.NET CORE



Obrázek 2.1: Porovnání ASP.NET Core a ASP.NET [3]

ASP.NET je webový framework obsahující sadu knihoven, které obsahují hotová řešení mnoha základních problémů, které ve webových technologiích vyvstávají. Může se jednat např. o bezpečnost, autentifikaci uživatele, práci s databázemi apod.

Tato technologie je založena na architektuře klient-server. Výstupem ASP.NET aplikace je HTML stránka. ASP.NET tedy běží na serveru a reaguje na dotazy uživatele/klienta. Pro tvorbu ASP.NET aplikace je potřeba znalost především programovacího jazyku C# a značkovacího jazyku HTML.[11]

2.3.1 ASP.NET

ASP.NET je webový framework s otevřeným zdrojovým kódem, vytvořený společností Microsoft, pro vytváření moderních webových aplikací. ASP.NET rozšiřuje platformu .NET o nástroje a knihovny určené právě pro vytváření webových aplikací.[6]

2.3.2 ASP.NET Core

ASP.NET Core je open-source a multiplatformní verze ASP.NET. Tato platforma je navržena tak, aby umožnila rychlé vyvíjení runtime komponent, API, překladačů apod. a zároveň poskytovala stabilní a podporovanou platformu pro udržení běhu aplikací.

Aplikace v ASP.NET Core lze, oproti dřívější verzi APS.NET Windows-only version, vyvíjet a spouštět v systémech Windows, Linux, macOS a Docker.[7]

2.4 EF6 a EF Core

Entity Framework je Object-relational mapping (ORM)³. To znamená, že se databázové tabulky přímo mapují na C# třídy. V projektu následně pracujeme pouze s objekty a framework za nás na pozadí generuje SQL dotazy. Díky tomu je výsledná aplikace tvořena především pomocí objektů.[12]

2.4.1 Entity Framework 6

Entity Framework 6 (EF6) je ORM, primárně navržený pro .NET Framework, ale zároveň s podporou pro .NET Core. EF6 je stabilní, podporovaný produkt, ale již se aktivně nevyvíjí.[13]

2.4.2 Entity Framework Core

Entity Framework Core (EF Core) je moderní ORM pro .NET. Podporuje dotazy LINQ, sledování změn, aktualizace a migrace schématu.

EF Core pracuje s SQL Server/SQL Azure, SQLite, Azure Cosmos DB, MySQL, PostgreSQL a mnoha dalšími druhy databází.[13]

2.5 MVC

Model-View-Controller (MVC) je návrhový vzor používaný k rozdělení webové aplikace na 3 komponenty.

³Objektově-relační mapování

- Model - Práce s daty
- View - Uživatelské rozhraní
- Controller - Logická část aplikace

Pomocí vzoru MVC pro webové aplikace jsou požadavky směřovány na controller, který je zodpovědný za práci s modelem. Model provádí akce a načítá data z databáze. Controller poté zvolí zobrazení (view), které se má zobrazit, a poskytne mu model. View už pouze vykreslí stránku na základě dat získaných z modelu.[5]

2.5.1 Model

Jedná se o dynamickou datovou strukturu aplikace, nezávislou na uživatelském rozhraní. Jejím úkolem je spravovat data, pravidla a logiku aplikace.

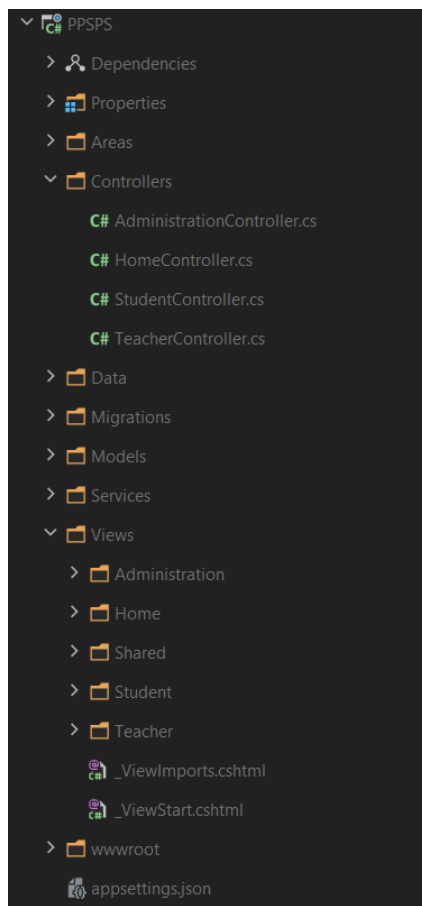
V ASP.NET Core MVC má model podobu třídy, kde veřejné metody představují položky v databázi, s kterou je třída provázána. Ukládáme ji do adresáře Models v rámci projektu. Pomocí atributů definuje pravidla, která se aplikují na klienta a server.[9]

2.5.2 View

Převádí data reprezentovaná objekty modelu do podoby vhodné k interaktivní prezentaci uživateli.[8] Může se jednat o jakoukoliv reprezentaci dat (graf, diagram, tabulka, atd.).

V ASP.NET Core se využívá syntaxe Razor. Ta poskytuje jednoduchý, čistý a lehký způsob vykreslení obsahu HTML stránek na základě view. Razor umožňuje vykreslit stránku pomocí C# a vytvářet webové stránky plně kompatibilní s HTML5.[9]

Každý controller může pro jedno view generovat vlastní obraz. Aby tedy nedocházelo ke kolizi 2 controlleru pro jedno view, vyžaduje MVC uložení view do adresáře Views v rámci projektu, a zde do podadresáře s názvem controlleru, který ho generuje (viz obr. 2.2).



Obrázek 2.2: Ukládání View v rámci projektu

2.5.3 Controller

Controller reaguje na vstup uživatele a provádí interakce s objekty datového modelu. Zjednodušeně to znamená, že řadič přijme vstup, ověří jej a následně ho předá modelu.[9]

Třída controlleru obsahuje veřejné metody označené jako Action method. V ASP.NET MVC musí každý název třídy controlleru končit klíčovým slovem "Controller". To znamená, že pro domovské stránky třídu nazýváme HomeController, pro stránky studenta StudentController, apod. Všechny tyto třídy ukládáme v rámci projektu do složky Controllers.

```
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

Listing 1: Controller - Action Method

2.6 MySQL

MySQL je otevřený systém řízení báze dat uplatňující relační databázový model. Jedná se o multiplatformní databázi. Komunikace s databází probíhá pomocí dotazovacího jazyka SQL.[10]

Právě díky těmto vlastnostem jsem se rozhodl využít MySQL jako databázi pro tuto webovou aplikaci.

Kapitola 3

Základní struktura ASP.NET Core

Pro tuto webovou aplikaci jsem použil verzi .NET Core 3.1, ta byla vydána v prosinci roku 2019. V době, kdy jsem začínal pracovat na tomto projektu to také byla nejnovější verze.

V tomto projektu jsem dále použil ASP.NET Core Identity¹, které za mě vygenerovalo základní strukturu, se kterou jsem dále pracoval.

V poslední řadě bylo zapotřebí stáhnout přes NuGet package manager² Entity Framework Core a k němu nástroje potřebné k práci s MySQL databází, kterou v projektu využívám.

3.1 Model

Jak již bylo zmíněno v teorii (viz kapitola 2.5.1 Model), model se zabývá především prací s daty uloženými v databázi. Pro to, abych s nimi mohl efektivně pracovat, potřebuji vytvořit z jednotlivých položek databáze objekty. Všechny objekty musí být veřejné, tedy public.

Pro každou tabulku z databáze jsem si tedy vytvořil objekt (třídu) ve složce **Models** (viz obr. 3.1). Poté jsem si potřeboval nadefinovat každou její položku jako vlastnost tohoto objektu, kde datový typ reprezentuje způsob zápisu do databáze tzn.

- VARCHAR() - Public string NazevObjektu { get; set; }
- INT - Public int NazevObjektu { get; set; }
- DATETIME - public DateTime NazevObjektu { get; set; }

¹Rozhraní API, které podporuje funkce přihlášení uživatelského rozhraní (UI). Spravuje uživatele, hesla, data profilu, role, deklarace identity, tokeny, potvrzení e-mailu a další.[1]

²NuGet je správce balíčků, který vývojářům umožňuje sdílet opakovaně použitelný kód.

Dalším krokem bylo přiřazení atributů k jednotlivým vlastnostem. V ASP.NET Core se můžeme setkat jak s atributy, které definují vlastnosti (jedná se o primární klíč, název, formát, ...), např.

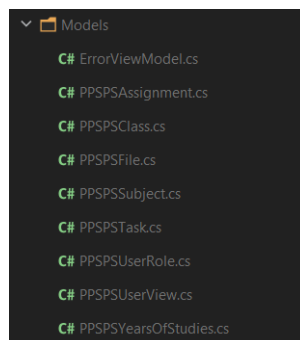
- Key - Označuje vlastnost jako primární klíč.
- Display("Název") - Název vlastnosti, která se zobrazí ve View.
- DisplayFormat(DataFormatString = "0: yyyy-MM-dd", ApplyFormatInEditMode = true) - Nastavuje formát zobrazení pro hodnotu vlastnosti.

Tak i s tzv. Atributy ověřování. Tyto atributy nám umožňují zadat pravidla pro vlastnosti objektu. V ASP.NET Core existuje několik předdefinovaných atributů, např.

- EmailAddress - Ověřuje, zda má vlastnost formát e-mailu.
- Range - Ověřuje, že hodnota vlastnosti spadá do zadaného rozsahu.
- Required - Ověří, že pole nemá hodnotu null.
- StringLength - Ověří, že hodnota vlastnosti řetězce nepřekračuje zadané omezení délky.

Atributy ověření umožňují vypsání chybové zprávy, která se zobrazí, pokud je zadán neplatný vstup do hodnoty vlastnosti. Tyto atributy je možné ověřit jak na straně klienta, tak na straně serveru.

K vytvoření modelu byla nutná již navržená databáze, tu popisuji v kapitole 6 Databáze níže. Pro lepší představu přikládám zdrojový kód PPSPSubject.cs, modelu popisující tabulku předmětů (viz zdrojový kód 2).



Obrázek 3.1: Složka Models

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace PPSPS.Models
{
    public class PPSPSSubject
    {
        [Key]
        [Column(TypeName = "varchar(767)")]
        public string Id { get; set; }

        [StringLength(45, ErrorMessage = "Název předmětu nesmí" +
            "mít víc, jak 45 znaků.")]
        [Display(Name = "Název předmětu")]
        [Column(TypeName = "varchar(45)")]
        public string SubjectName { get; set; }

        [StringLength(5, ErrorMessage = "Zkratka předmětu nesmí" +
            "mít víc, jak 5 znaků.")]
        [Display(Name = "Zkratka předmětu")]
        [Column(TypeName = "varchar(5)")]
        public string SubjectAbbreviation { get; set; }
    }
}

```

Listing 2: Model - Předmět

3.2 View

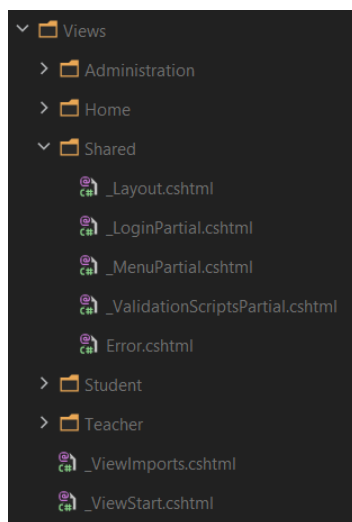
View lze v ASP.NET Core MVC chápat, jako HTML šablonu, do které pomocí Razor elementů vkládáme data. Pro každý controller je ve složce **Views** vytvořen vlastní adresář s názvem controlleru viz obr. 3.2. V každém z těchto adresářů je poté uloženo view

ve formátu .cshtml.

Ve Views adresáři je umístěn mimo jiné podadresář Shared. V něm se ukládají view, které jsou sdílené v rámci všech controllerů. Nachází se v něm většinou Layout, Partial view³, nebo stránky s chybovým hlášením.

V tomto projektu jsem si vytvořil 2 partial view. První je určený pro menu (viz kapitola `_MenuPartial`) a druhý pro lištu, kde zobrazuje email přihlášeného uživatele a možnost se odhlásit (viz kapitola `_LoginPartial`).

Pro to, abych mohl k těmto partial přistupovat, musím do `_Layout.cshtml` dopsat kód, pomocí kterého je volám, viz 5 pro menu a viz 4 pro zobrazení emailu na liště, a to v místě, kde je chci zobrazit.



Obrázek 3.2: Složka Views

3.2.1 `_Layout`

V záhlaví a zápatí html souborů se nachází metadata a odkazy na CSS/Javascript. Pro to, abych nemusel zápatí a záhlaví psát do každého view, můžu je napsat právě do souboru `_Layout.cshtml`. Když se následně generuje stránka pro uživatele, načte se prvně `_Layout.cshtml` a k němu se na místě, kde je volána funkce `RenderBody()` (viz zdrojový kód 3) doplní view, které uživatel požaduje.

Podobně se pracuje i s partial. Pro jejich zavolání je potřeba napsat tag `<partial />` s parametrem `name`, jehož hodnota je název požadovaného partial (viz zdrojové kódy 4

³Částečný pohled; Snižuje množství duplicitních kódů správou opakovaně používaných částí pohledů.

pro `_LoginPartial.cshtml` a 5 pro `_MenuPartial.cshtml`). Opět se volají z místa, ve kterém je chceme zobrazit.

```
<main role="main" class="pb-3">
    @RenderBody()
</main>
```

Listing 3: View - `RenderBody()`

```
<partial name="_LoginPartial.cshtml"/>
```

Listing 4: View - Volání `_LoginPartial.cshtml`

```
<partial name="_MenuPartial.cshtml"/>
```

Listing 5: View - Volání `_MenuPartial.cshtml`

3.2.2 `_LoginPartial`

Hlavník úkolem tohoto partial je zobrazit jméno uživatele, pokud je přihlášen a nabídnout mu možnost se odhlásit (viz obr. 3.3). Prvním krokem je, pomocí podmínky zjistíme, zda je uživatel přihlášen (`SignInManager.IsSignedIn(User)`). Pokud není, podmínka není splněna a partial se nezobrazí. Pokud ale je, požádá se o uživatelské jméno přihlášeného uživatele (email) a to se následně zobrazí s tlačítkem pro odhlášení (viz zdrojový kód 6).



Obrázek 3.3: Lišta s uživatelským jménem a tlačítkem odhlášení.

```

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    <li class="nav-item">
        <a id="manage" class="nav-link text-dark" asp-area="Identity"
            asp-page="/Account/Manage/Index" title="Manage">
            @UserManager.GetUserName(User)
        </a>
    </li>
    <li class="nav-item">
        <form id="logoutForm" class="form-inline" asp-area="Identity"
            asp-page="/Account/Logout" asp-route-returnUrl=
                "@Url.Action("Index", "Home", new { area = "" })">
            <button id="logout" type="submit" class="nav-link btn btn-link
                text-dark">
                Odhlásit se
            </button>
        </form>
    </li>
}
</ul>

```

Listing 6: View - _LoginPartial.cshtml

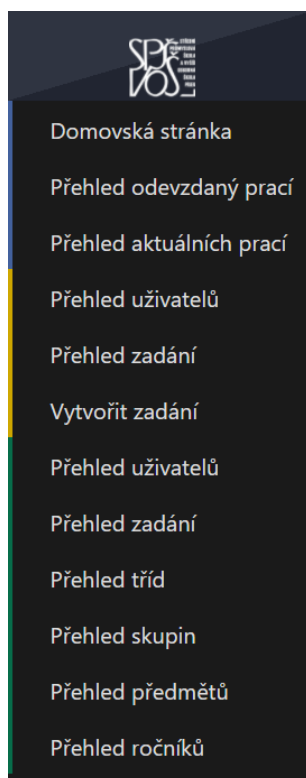
3.2.3 _MenuPartial

Dalším úkolem bylo zobrazení menu (viz obr. 3.4). Jako v předešlém případě ověříme, zda je uživatel přihlášen pomocí podmínky (`SignInManager.IsSignedIn(User)`). Pokud je uživatel přihlášen, zjišťuje se, jako má roli. Příkládám jako příklad část zdrojového kódu 7, kde ověřuji, zda je uživatel ověřený (`User.Identity.IsAuthenticated`)⁴, a pokud ano, zda má uživatel roli Administrator (`User.IsInRole("Administrator")`). V tomto případě můžou

⁴Pokud je uživatel přihlášený pomocí uživatelského jména a hesla, je i ověřený.

nastat 3 situace:

1. Pokud je uživatel přihlášen a má roli administrátora, zobrazí se mu menu se všemi položkami.
2. Pokud je uživatel přihlášen ale nemá roli administrátora, zobrazí se mu menu pouze s položkou Domovská stránka.
3. Pokud není uživatel přihlášen, nezobrazí se mu menu vůbec.



Obrázek 3.4: Uživatelské menu se všema stránkami.

```

@if (SignInManager.IsSignedIn(User)){
    @if (User.Identity.IsAuthenticated){
        <li><a asp-area="" asp-controller="Home"
            asp-action="Index">Domovská stránka</a></li>
    }
    @if (User.IsInRole("Administrator")){
        <li><a asp-area="" asp-controller="Administration"
            asp-action="UsersOverview">Přehled uživatelů</a></li>
        <li><a asp-area="" asp-controller="Administration"
            asp-action="TasksOverview">Přehled zadání</a></li>
        <li><a asp-area="" asp-controller="Administration"
            asp-action="ClassesOverview">Přehled tříd</a></li>
        <li><a asp-area="" asp-controller="Administration"
            asp-action="SubjectsOverview">Přehled předmětů</a></li>
        <li><a asp-area="" asp-controller="Administration"
            asp-action="YearsOfStudiesOverview">Přehled ročníků</a></li>
    }
}

```

Listing 7: View - _MenuPartial.cshtml

3.3 Controller

Poslední položkou v MVC vzoru je controller. Každá role (kromě vedení, to má stejná práva jako administrator) bude mít vlastní controller. V projektu mám celkem 4 role, jejich bližší popis je v kategorii 5Role, to znamená, že zde budou 3 controllery pro role a k tomu 1 controller pro home adresář, kde je Domovská stránka (viz obr. 3.5).

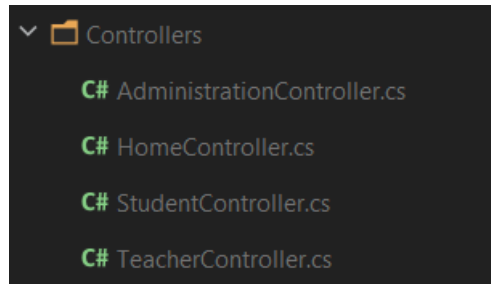
Protože každá role má mít vlastní controller, musel jsem nějaký způsobem ošetřit, aby uživatel např. s rolí studenta nemohl načíst view z controlleru pro administratora. Uživatele je tedy potřeba autorizovat⁵ k přístupu do controlleru. Pro tuto akci stačí

⁵Autorizace je proces získávání souhlasu s provedením nějaké operace, povolení přístupu někam, k někomu nebo něčemu.[4]

napsat nad třídu kód viz zdrojový kód 8.

```
[Authorize(Roles = "Administrator")]
```

Listing 8: Controller - Autorizace



Obrázek 3.5: Složka Controllers

Jak již bylo zmíněno v teorii (viz kategorie 2.5.3Controller), controller pracuje s tzv. Action Method. V celé aplikaci mi postačí pouze 4 varianty Action Method, ve kterých se budou pouze měnit data. Ty varianty jsou:

- Zobrazit seznam entit
- Zobrazit detail položky
- Editovat položku
- Smazat položku

Před tím, než začnu popisovat jednotlivé varianty Action Method vypíši zde pro lepší přehlednost a srozumitelnost seznam využívaných metod a jejich výstup:

- NotFound() - NotFoundResult, který vrátí chybovou hlášku Error 404, stránka nenalezena.
- View() - Vytvoří objekt ViewResult, který vykreslí view na výstup.

A zde metody, které se využívají pro práci s entitami v Entity Framework:

- .Include() - Načte i související data. Např. K tabulce uživatelů načte i tabulku tříd.

- `.AsNoTracking()` - Entity Framework načte entity a již je dále nesleduje ani neukládá.
- `.FirstOrDefaultAsync()` - Asynchronně vrátí první prvek sekvence nebo výchozí hodnotu, pokud sekvence neobsahuje žádné prvky.
- `.FindAsync()` - Asynchronně nalezne entitu s danými hodnotami primárního klíče, pokud ji nalezne, vrátí ji okamžitě. V opačném případě vrací null⁶.
- `.OrderBy()` - Seřadí prvky v kolekci na základě zadaných polí ve vzestupném pořadí. Např. Na základě příjmení.
- `.Where()` - Načte pouze data, která splňují nějakou podmínku. Např. Načte pouze práce, kde je ID uživatele rovno přihlášenému uživateli.

3.3.1 Seznam entit

Jednou ze základních funkcí, kterou by měla tato webová aplikace disponovat je výpis seznamu entit (viz zdrojový kód 9). Pro to, abychom mohli entity vypsát, je potřeba je nejprve uložit do implicitně typované lokální proměnné `var`⁷. Do této vytvořené proměnné přiřadíme `_Contex.Users`, což je metoda v `AuthDBContext.cs`. Právě tato třída je zodpovědná za interakci s databází. Poté pomocí `.OrderBy(u => u.LastName)` určím, že chci řadit vzestupně podle proměnné `LastName`, tedy příjmení. Protože s entitami již dále nepotřebuji pracovat, zakážu trasování pomocí `.AsNoTracking()`.

Takto jsem nadefinoval proměnnou `var users`, kterou nyní stačí pouze uložit do `List<T>`⁸, a protože `Task<IActionResult>` požaduje asynchronní výstup, přidám k `.ToList()` klíčové slovíčko `Async`.

Ná výstup se nám tedy vrátí seznam uživatelů seřazený vzestupně podle příjmení, s kterým dále pracuji ve view pomocí `foreach()`.

⁶Označení pro žádnou hodnotu.

⁷Typ instance je odvozen z prvku zadaného v inicializátoru. To znamená, že kompilátor určí a přiřadí nejvhodnější typ proměnné.

⁸List (česky seznam) se řadí do kolekci, které umožňují prvky za běhu programu přidávat a mazat.

```

public async Task<IActionResult> UsersOverview()
{
    var users = _context.Users
        .OrderBy(u => u.LastName)
        .AsNoTracking();

    return View(await users.ToListAsync());
}

```

Listing 9: Controller - Seznam entit

3.3.2 Detail

Pokud chceme zobrazit detail některé entity, např. uživatele (viz zdrojový kód 10), potřebejeme nějakým jednoznačným způsobem definovat, o jakou entitu se jedná. Jedním z takových způsobů může být třeba primární klíč⁹. A nyní, jakým způsobem ji předat do controlleru, aby s ní mohl pracovat? Přesně k tomu se využívají parametry v url¹⁰ adrese. Jak je vidět ve zdrojovém kódu¹⁰, metoda `UserOverview` přebírá nějaký parametr, který nazýváme `id` a má datový typ `string`. Výsledná url může tedy vypadat například takto:

```
https://localhost:5001/Administration/UserOverview/f0a5d163-b81c-41d1-a488-89753377fd1c
```

Jako první se zde nachází protokol (`https`), po něm následuje doména (`localhost:5001`¹¹), a za doménou se vypíše jméno controlleru, ve kterém se nachází naše požadovaná stránka `UserOverview`. url zakončuje řetězec náhodných znaků, a to je právě `id` uživatele, tedy náš parametr.

Tímto způsobem jsem tedy vyřešil identifikaci entity, které detail požadujeme zobrazit. A nyní zpět ke controlleru. Pokud controller dostane požadavek na detail entity, ověří nejprve, zda obdržel parametr `id`. Pokud ne, vyvolá metodu `NotFound()` a vrátí ji na výstup. V opačném případě se zahájí proces, která jsem podrobněji popisoval již v předchozí kapitole (viz kapitola 3.3.1 Seznam entit).

⁹Jednoznačná identifikace entity v rámci databáze.

¹⁰Uniform Resource Locator, neboli webová adresa.

¹¹Aplikace momentálně funguje na mém počítači, proto je doména `localhost`.

V detailu chci zobrazit základní údaje o uživateli a třídu, ve které se uživatel nachází. Controller prvně načte všechny uživatele z databáze, ale jak jsem zmínil, požaduji načtení i třídy. Protože u uživatele ukládám pouze ID třídy, ve které se nachází, musím přidružit i tabulku tříd. Toho docílím zavoláním metody `.Include(c => c.Class)`. S daty nehodlám po načtení dále pracovat, proto mohu zakázat trasování.

Úkolem metody `FirstOrDefaultAsync(u => u.Id == id)` je procházet entity, dokud nenajde první, jejíž `Id` je shodné s parametrem `id`. Pokud takovou entitu najde, vrátí ji na výstup. Pokud takovou entitu nenajde, vrátí výchozí hodnotu, kterou je `null`.

Poslední podmínka ošetřuje situaci, kdy je zadán parametr, ale žádná entita tomuto parametru neodpovídá. Pokud tedy nenajdeme žádného uživatele, který by měl shodné `Id` s parametrem `id`, zavolá se metoda `NotFound()`, která se vrátí na výstup.

```
public async Task<IActionResult> UserOverview(string? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.Users
        .Include(c => c.Class)
        .AsNoTracking()
        .FirstOrDefaultAsync(u => u.Id == id);

    if (user == null)
    {
        return NotFound();
    }

    return View(user);
}
```

Listing 10: Controller - Detail

3.3.3 Editace

Třetí typ stránky je určená k editaci. Pro to, abychom mohli entity upravovat, potřebujeme využít dvě metody. První metoda (viz zdrojový kód 11) má za úkol načíst data o entitě z databáze, druhá následně upravená data odeslat zpět do databáze.

Při načítání dat je princip stejný, jak u načítání detailu (viz kapitola 10Detail), opět přebíráme parametr z url. Pokud se tam nachází, hledáme pomocí metody FindAsync(id) identitu, jejíž primární klíč je shodný s naším parametrem. Pokud se parametr nenalezne, vyvolá se metoda NotFound(), která vrátí na výstup chybovou hlášku. Následně ověříme, zda se nějaká entita našla, pokud ne, controller vrací chybovou hlášku.

Protože se jedná o editaci, potřebuji načíst seznam všech tříd tak, abych mohl případně změnit uživateli třídu. Z toho důvodu jsem si vytvořil metodu s názvem PopulateClassesWithIdDropDownList() (viz zdrojový kód 15), která má za úkol načíst všechny entity z tabulky tříd, a jejich Id a název vložit do drop-down menu.

```
public async Task<IActionResult> UserEdit(string? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.Users.FindAsync(id);
    if (user == null)
    {
        return NotFound();
    }

    PopulateClassesWithIdDropDownList(user.ClassId);
    return View(user);
}
```

Listing 11: Controller - Editace, načtení entit)

Máme načtenou stránku s entitami, provedeme změny a chceme je odeslat zpět do

databáze. V tuto chvíli využijeme druhou metodu (viz zdrojový kód 12), která má provést změny v databázi. Protože se dvě metody nemohou nazývat stejným jménem, musím této metodě přidělit pomocí atributu tzv. `ActionName`, které následně volám z view. V attributech mimo jiné nastavím i dotazovací metodu (tedy `HttpPost`) a `ValidateAntiForgeryToken`. `ValidateAntiForgeryToken` je ochrana proti padělání požadavků mezi weby. MVC zapíše jedinečnou hodnotu do cookies a tatáž hodnota se zapíše i do formuláře. Před odesláním formuláře se tyto hodnoty porovnají a pokud jsou různé, formulář se neodešle.

Controller zkontroluje parametr a pokud nenalezl chybu, načte první entitu, jejíž `Id` je shodné s parametrem, předaným v url. Následuje metoda `TryUpdateModel`, ta aktualizuje zadanou instanci modelu pomocí hodnot v paramteru (`FirstName`, `LastName`, atd.). Metodu zahrneme do podmínky. Pokud je podmínka splněna (nenastane chyba), asynchronně uloží všechny změny provedené v tomto kontextu a uloží je do databáze. Poté se přesměruje na stránku `UsersOverview`. Pokud nastane chyba (podmínka není splněna), vyvolá se větev `catch`, která vrátí chybovou hlášku.

```

[HttpPost, ActionName("UserEdit")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> UserEdit_Post(string? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var userToUpdate = await _context.Users.FirstOrDefaultAsync(s => s.Id == id);
    if (await TryUpdateModelAsync<PPSPSUser>(
        userToUpdate, "",
        s => s.FirstName, s => s.LastName, s => s.Email,
        s => s.EmailConfirmed, c => c.ClassId))
    {
        try
        {
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(UsersOverview));
        }
        catch (DbUpdateException ex)
        {
            ModelState.AddModelError("", "Nebylo možné uložit změny. " +
                "Zkuste to znovu později a pokud " +
                "problém přetrvává, " +
                "obratte se na správce systému.");
        }
    }

    PopulateClassesWithIdDropDownList(userToUpdate.ClassId);
    return View(userToUpdate);
}

```

Listing 12: Controller - Editace, odeslání změn)

3.3.4 Smazání

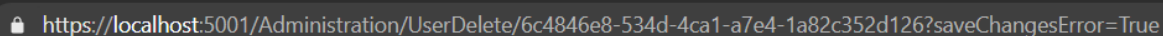
Posledním typem, který v práci využívám je smazání. Entity, které mají relaci se některou jinou netitou nelze smazat.

Podobně, jak tomu bylo u editace a vytvoření, i zde využívám 2 metody. První, která načítá stránku s možností potvrdit smazání nebo ho zrušit. Druhá, která provede samotné smazání entity.

Pokud uživatel v seznamu entity vybere možnost smazat entitu, aplikace vyšle požadavek GET a načte view `UserDelete` (viz zdrojový kód 13). Na této stránce má uživatel možnost potvrdit smazání entity, nebo ho zrušit. Pokud se rozhodne pro smazání, vytvoří se požadavek POST, který zavolá metodu `HttpPost UserDelete` (viz zdrojový kód 14), která smazání entity provede. V `HttpPost UserDelete` se nachází blok try-catch, který bude kontrolovat, zda nedošlo při aktualizaci databáze k chybě. Pokud k chybě dojde, metoda `HttpPost UserDelete` zavolá metodu `HttpGet UserDelete` a předá ji parametr, který indukuje, že při aktualizaci databáze došlo k chybě (viz obr. 3.6).

Metoda `HttpGet UserDelete` poté opět zobrazí potvrzovací stránku, společně s chybovou zprávou. Uživatel poté může požadavek opakovat, nebo ho zrušit.

Parametr `saveChangesError` je při prvním volání metody `HttpGet UserDelete` `false`, to znamená, že tato metoda není volaná z důvodu chyby a nevypisuje tedy chybovou zprávu. Pokud ale tuto `HttpGet UserDelete` metodu volá metoda `HttpPost UserDelete` v návaznosti na chybu při aktualizaci databáze, nastaví se `saveChangesError` na `true` a proto se chybová zpráva vykreslí na stránce (viz obr. 3.7).



<https://localhost:5001/Administration/UserDelete/6c4846e8-534d-4ca1-a7e4-1a82c352d126?saveChangesError=True>

Obrázek 3.6: Url při chybě aktualizace databáze.

```

public async Task<IActionResult> UserDelete(string? id,
bool? saveChangesError = false)
{
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.Users
        .AsNoTracking()
        .FirstOrDefaultAsync(m => m.Id == id);
    if (user == null)
    {
        return NotFound();
    }
    if (saveChangesError.GetValueOrDefault())
    {
        ViewData["ErrorMessage"] =
            "Smazání se nezdařilo. Zkuste to znovu později "
            + "a pokud problém přetrvává, " +
            "obratte se na správce systému.";
    }
    return View(user);
}

```

Listing 13: Controller - HttpGet UserDelete

Odstranit uživatele

Smazání se nezdařilo. Zkuste to znovu později a pokud problém přetrvává, obraťte se na správce systému.

Opravdu si přejete odstranit uživatele Hánský Karel?

Křestní jméno	Karel
Příjmení	Hánský
E-mail	khansky@sps-pi.cz

[Odstranit](#) | [Zpět na přehled](#)

Obrázek 3.7: Chybová zpráva při chybě aktualizace databáze.

```
[HttpPost, ActionName("UserDelete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> UserDelete_Post(string? id)
{
    var user = await _context.Users.FindAsync(id);
    if (user == null)
    {
        return RedirectToAction(nameof(UsersOverview));
    }
    try
    {
        _context.Users.Remove(user);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(UsersOverview));
    }
    catch (DbUpdateException ex)
    {
        return RedirectToAction(nameof(UserDelete), new { id = id,
            saveChangesError = true });
    }
}
```

Listing 14: Controller - HttpPost UserDelete

Kapitola 4

Přihlašování a registrace

4.1 Přihlašování

4.2 Registrace

Kapitola 5

Role

5.1 Student

5.2 Učitel

5.3 Administrator

5.4 Vedení školy

Kapitola 6

Databáze

6.1 Připojení databáze

6.2 Návrh databáze

Kapitola 7

Závěr

Seznam tabulek

Seznam obrázků

2.1	Porovnání ASP.NET Core a ASP.NET [3]	7
2.2	Ukládání View v rámci projektu	10
3.1	Složka Models	13
3.2	Složka Views	15
3.3	Lišta s uživatelským jménem a tlačítkem odhlášení.	16
3.4	Uživatelské menu se všema stránkami.	18
3.5	Složka Controllers	20
3.6	Url při chybě aktualizace databáze.	27
3.7	Chybová zpráva při chybě aktualizace databáze.	29

Seznam zdrojových kódů

1	Controller - Action Method	11
2	Model - Předmět	14
3	View - RenderBody()	16
4	View - Volání _LoginPartial.cshtml	16
5	View - Volání _MenuPartial.cshtml	16
6	View - _LoginPartial.cshtml	17
7	View - _MenuPartial.cshtml	19
8	Controller - Autorizace	20
9	Controller - Seznam entit	22
10	Controller - Detail	23
11	Controller - Editace, načtení entit)	24
12	Controller - Editace, odeslání změn)	26
13	Controller - HttpGet UserDelete	28
14	Controller - HttpPost UserDelete	29
15	Drop-down list ze seznamu tříd	39

Literatura

- [1] ANDERSON, R.: *Úvod do Identity ASP.NET Core*. Microsoft Corporation, 2020, [cit. 2021-4-10].
Dostupné z: <https://docs.microsoft.com/cs-cz/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>
- [2] ANDERSON, R.; Mullen, T.; Vicarel, D.: *Razor Referenční informace k syntaxi pro ASP.NET Core*. Microsoft Corporation, 2020, [cit. 2021-4-8].
Dostupné z: <https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/razor?view=aspnetcore-5.0>
- [3] ASP.NET vs ASP.NET Core. 2017, [cit. 2021-4-4].
Dostupné z: <https://thienn.com/aspnet-vs-aspnetcore/>
- [4] Autorizace. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2021, [cit. 2021-04-12].
Dostupné z: <https://cs.wikipedia.org/wiki/Autorizace>
- [5] Microsoft Corporation: *ASP.NET MVC Pattern*. 2021, [cit. 2021-4-5].
Dostupné z: <https://dotnet.microsoft.com/apps/aspnet/mvc>
- [6] Microsoft Corporation: *What is ASP.NET?* 2021, [cit. 2021-4-4].
Dostupné z: <https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet>
- [7] Microsoft Corporation: *What is ASP.NET Core?* 2021, [cit. 2021-4-4].
Dostupné z: <https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core>
- [8] Model-view-controller. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2021, [cit. 2021-04-05].
Dostupné z: <https://cs.wikipedia.org/wiki/Model-view-controller>

- [9] Model-view-controller. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2021, [cit. 2021-04-05].
Dostupné z: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [10] MySQL. 2021, [cit. 2021-04-06].
Dostupné z: <https://cs.wikipedia.org/wiki/MySQL>
- [11] ČÁPKA, D.: *Lekce 1 - Úvod do ASP.NET*. 2012, [cit. 2021-4-4].
Dostupné z: <https://www.itnetwork.cz/csharp/asp-net-core/zaklady/tutorial-uvod-do-asp-dot-net>
- [12] ČÁPKA, D.: *Lekce 8 - Scaffolding a Entity Framework v ASP.NET MVC*. 2014, [cit. 2021-4-5].
Dostupné z: <https://www.itnetwork.cz/csharp/asp-net-mvc/zaklady/asp-dot-net-mvc-tutorial-scaffolding-entity-framework-editor-clanku>
- [13] VICKERS, A.; OLPROD: *Porovnání EF Core a EF6*. Microsoft Corporation, 2019, [cit. 2021-4-5].
Dostupné z: <https://docs.microsoft.com/cs-cz/ef/efcore-and-ef6/>
- [14] WAGNER, B.; OLPROD: *Prohlídka jazyka C#*. Microsoft Corporation, 2021, [cit. 2021-3-25].
Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp/>

Příloha A

Příloha

```
private void PopulateClassesWithIdDropDownList(object selectedClass = null)
{
    var classesQuery = from c in _context.Classes
        orderby c.ClassName
        select c;
    ViewBag.ClassId =
        new SelectList(classesQuery.AsNoTracking(), "Id",
            "ClassName", selectedClass);
}
```

Listing 15: Drop-down list ze seznamu tříd