

Autor **Petr Horák**

Obor **Informační technologie**

Vedoucí práce **Ing. Tomáš Kazda, DiS.**

Školní rok **2020/2021**

Střední průmyslová škola strojní   
a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3

Revize aplikace Rentals

Maturitní práce

Anotace (Resumé)

Tato maturitní práce se zabývá odstraněním chyb a úpravami webové aplikace Rentals používané na SPŠSE a VOŠ Liberec. Aplikace slouží k evidenci výpůjček vybavení ze školního ateliéru. Výchozí verzí této aplikace byla aplikace Rentals vytvořená v rámci maturitní práce bývalého studenta SPŠSE a VOŠ Liberec Marka Honce ve školním roce 2018/19. Při provozu původní aplikace se však vyskytly chyby narušující její chod a funkčnost. Proto vznikla tato práce, která přináší řešení těchto chyb a zároveň řeší a vylepšuje další nedostatky původní verze.

Summary

This work focuses on elimination of errors and modifications of the Rentals web application used at SPŠSE and VOŠ Liberec. The application is used to record equipment borrowings from the school atelier. The initial version of this application was the Rentals application created within the graduation work of the former student of SPŠSE and VOŠ Liberec Marek Honc in the school year 2018/19. During the operation of the original application occurred errors that disrupted its operation and functionality. Therefore, this work was created. It provides a solution to these errors and at the same time solves and improves other shortcomings of the original version.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní/ročníkovou práci vypracoval(a) sám(a) a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

Petr Horák

Obsah

[Úvod 1](#_Toc66272443)

[1 Nasazení ve vývojovém prostředí 2](#_Toc66272444)

[2 Přenos databáze do testovacího prostředí 3](#_Toc66272445)

[2.1 Vložení dat do lokální databáze 3](#_Toc66272446)

[2.1.1 Vložení dat do databáze pomocí souboru formátu *.bak* 3](#_Toc66272447)

[2.1.2 Vložení dat do databáze pomocí SQL příkazů 4](#_Toc66272448)

[3 Analýza zdrojových kódů 5](#_Toc66272449)

[3.1 Projekt *Rentals.Common* 5](#_Toc66272450)

[3.2 Projekt *Rentals.DL* 5](#_Toc66272451)

[3.2.1 Struktura projektu 5](#_Toc66272452)

[*3.3* Projekt *Rentals.Web* 6](#_Toc66272453)

[3.3.1 Zákaznická část aplikace 6](#_Toc66272454)

[3.3.2 Administrační část aplikace 7](#_Toc66272455)

[4 Analýza a dokumentace chyb 8](#_Toc66272456)

[4.1 Přihlašování uživatelů 8](#_Toc66272457)

[5 Oprava chyb a úprava stávajících řešení 12](#_Toc66272458)

[5.1 12](#_Toc66272459)

[Závěr 19](#_Toc66272460)

[Seznam obrázků 20](#_Toc66272461)

[Použitá literatura 21](#_Toc66272462)

[A. Seznam přiložených souborů 1](#_Toc66272463)

[B. Další příloha 2](#_Toc66272464)

Úvod

Téma této práce jsem si zvolil, neboť splňuje vše podstatné, co jsem od dlouhodobé maturitní práce očekával. Mým cílem bylo, aby má práce byla prakticky využitelná, a aby čas strávený s její tvorbou byl přínosný jak pro mě, tak alespoň částečně pro mé okolí. Proto, když mi bylo nabídnuto pracovat na úpravě školního výpůjčkového systému, dlouho jsem neváhal.

Webová aplikace Rentals byla vytvořena pomocí programovacího jazyka C# a frameworku .NET Core. Aplikace je v provozu od roku 2019.

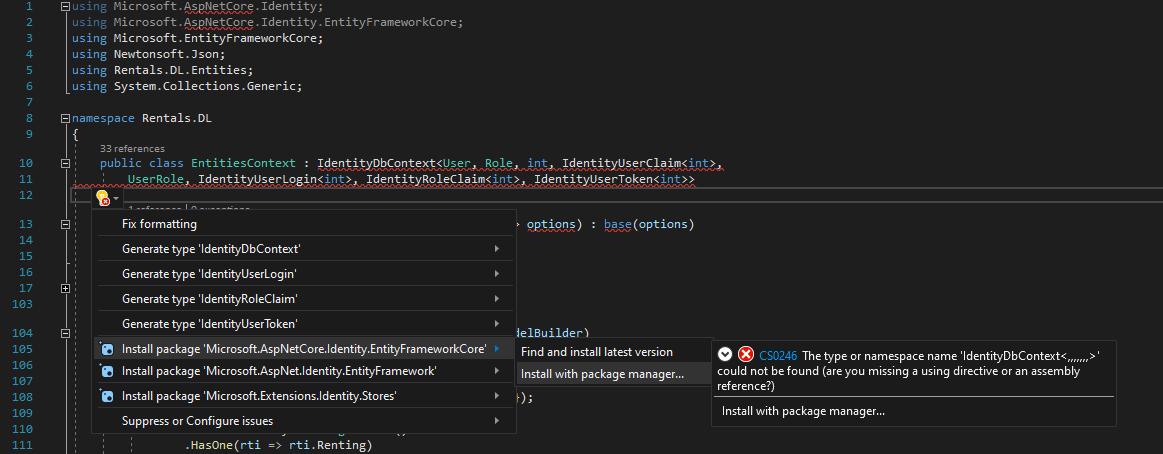
V tomto dokumentu popisuji postup, jakým jsem řešil úpravy původní verze aplikace, tak, aby nová verze byla plně funkční, méně problémová a přehlednější pro její uživatele.

1. Nasazení ve vývojovém prostředí

Jakožto vývojové prostředí jsem používal počítačový program Microsoft Visual Studio 2019, se kterým mám za dobu studia nemalé zkušenosti. Použití kombinace Visual Studia, C# a .NET Core se při debugování a práci s kódem ukázalo jako výhodné.

Výchozím stavem, ze kterého jsem při nasazení aplikace do vývojového prostředí vycházel byl repozitář na platformě GitHub obsahující zdrojové kódy původní aplikace Rentals. Data z repozitáře jsem pomocí funkce *Clone* importoval do Visual Studia.

Jak je již zmíněno výše, Visual Studio zprovoznění aplikace značně ulehčuje například svými chybovými hlášeními, které buď přímo odkazují na problém bránící chodu aplikace, nebo poskytují dobrý přehled nad tím, co je špatně. Postupoval jsem tedy podle tohoto seznamu chyb. Spuštění aplikace bránila absence NuGet balíčku *Microsoft.AspNetCore.Identity.EntityFrameworkCore.* Tento balíček do projektu Rentals.DL imlementuje ASP.NET Core Identity, což je API (Application Programming Interface), které umožňuje do aplikace zavést přihlašování jejich uživatelů uživatelskými účty. Z důvodu kompatibility je použita verze ASP.NET Core Identity 2.1.3.

Po přidání Identity balíčku již nebyly vývojovým prostředím hlášeny žádné chyby a bylo tak možné řešení sestavit.

Obrázek Příklad generovaného návrhu řešení vývojovým prostředím MS Visual Studio

1. Přenos databáze do testovacího prostředí

Dalším krokem po tom, když bylo řešení možné bez chyb sestavit, bylo připojení databáze.

Aplikace používá MSSQL (Microsoft SQL) databázi. Při přenosu databáze do testovacího prostředí jsem nejprve pomocí *SQL Server Object Explorer* panelu v MS Visual Studio vytvořil na lokálním serveru novou databázi. Z vlastností této nově vytvořené databáze jsem zkopíroval parametr *connection string*, který bylo třeba vložit do souboru *appsettings.json*, pro konfiguraci připojení projektu k databázi. Následně jsem použil příkaz *Update-Database*, který inicializuje nově vytvořenou databázi dle konfigurace obsažené v souboru s kontextem databáze (konkrétně *EntitiesContext.cs*).

* 1. Vložení dat do lokální databáze

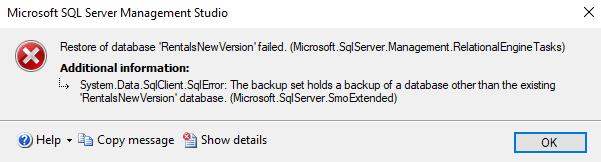
K co nejpřesnějšímu napodobení podmínek verze aplikace provozované na webu s verzí v testovacím prostředí bylo vhodné použít data přímo z aktivní databáze, kterou používá přímo webová aplikace.

* + 1. Vložení dat do databáze pomocí souboru formátu *.bak*

Abych naplnil nově nakonfigurovanou databázi daty, na kterých bych testoval funkčnost aplikace, měl jsem k dispozici soubor s daty z originální databáze ve formátu .*bak*. Tento formát souborů se používá k zálohování souborů. (1)

K přesunutí dat ze souboru *.bak* do databáze bylo třeba použít program *Microsoft SQL Server Management Studio*, který slouží ke správě Microsoft SQL Serveru a Microsoft SQL databází. Oproti *SQL Server Object Explorer* panelu v MS Visual Studio nabízí rozšířené funkce, mezi které patří i funkce *Restore*, která přijímá soubory formátu *.bak*. Tato funkce slouží k obnovení databáze z vybraného zdroje dat. V tomto případě byl zdrojem dat onen soubor formátu *.bak*. Po použití funkce *Restore* na nově vytvořenou databázi se však vyskytlo několik problémů.

Prvním problémem byly rozdílné verze serverů. SQL Server, na kterém byla v provozu databáze, ze které byla pořízena záloha v podobě zmiňovaného soubor formátu *.bak,* měl jinou verzi než ten lokální, na kterém byla vytvořena nová databáze. Po změně verze lokálního SQL serveru nastal další poněkud zásadnější problém.

Další chybová hláška programu *Microsoft SQL Server Management Studio* upozorňovala na skutečnost, že zálohu jedné databáze nelze vložit do databáze jiné, i když se svojí strukturou a vlastnostmi shodují.

Obrázek Chybová hláška programu Microsoft SQL Server Management Studio

* + 1. Vložení dat do databáze pomocí SQL příkazů

Chybové hlášení (viz Obrázek 2) naznačovalo, že dosavadní postup s využitím funkce *Restore* a vložením souboru formátu *.bak* úspěšné nebude. Data z původní databáze jsem proto dostal k dispozici v souborech ve formátu *.sql*. Tyto soubory obsahovali SQL příkazy jak pro vytvoření struktury databáze, tak pro naplnění databáze daty. Jelikož databáze, kterou jsem zkoušel naplnit daty s použitím funkce *Restore* a souboru *.bak*, už strukturu tabulek měla díky provedenému příkazu *Update-Database*, vytvořil jsem databázi novou, kterou jsem nakonfiguroval SQL příkazy ze souboru formátu .sql a dalším souborem .sql obsahujícím INSERT příkazy jsem do tabulek databáze vložil data.

Bylo možné použít první verzi databáze a zkusit ji pouze naplnit daty použitím INSERT příkazů, z důvodu zachování integrity a eliminaci potencionálních komplikací jsem se ale – jak je již zmíněno výše – rozhodl vytvořit databázi novou. Toto řešení se ukázalo jako úspěšné a po proběhnutí všech SQL příkazů byla databáze funkční a naplněna daty. Aplikace se již při spuštění z vývojového prostředí maximálně podobala jak vzhledem, tak funkčností verzi, jež byla v provozu na webu.

Další možností, jak naplnit lokální databázi daty, by bylo „seedování“ vlastních dat. Tento způsob jsem ale zavrhl kvůli časové náročnosti tvorby rádoby relevantních dat.

1. Analýza zdrojových kódů

Zdrojové kódy aplikace jsou psány programovacím jazykem C#. C# je jedním z nejrozšířenějších vysokoúrovňových programovacích jazyků. Používá se k objektovému programování. Byl vyvinut firmou *Microsoft*. (2)

Dále je zde využito frameworku *.NET Core*, který usnadňuje tvorbu aplikací v jazyce C#.

Struktura souborů se zdrojovými kódy aplikace je tvořena třemi projekty, které jsou popsány v podkapitolách níže.

* 1. Projekt *Rentals.Common*

Jedná se o nejméně rozsáhlou část aplikace obsahující například definici základních výčtových typů, které jsou použity v dalším projektu *Rentals.DL* obstarávajícím databázi aplikace. Dle těchto výčtových typů, které se definují klíčovým slovem *enum*, se v aplikaci stanovují například stavy výpůjček a role uživatelů.

V další části také obsahuje třídy, jejichž metody rozšiřují funkčnost základní datových typů používaných v dalších projektech aplikace.

* 1. Projekt *Rentals.DL*

Tento již poněkud rozsáhlejší projekt obsahuje všechny zdrojové kódy týkající se databáze. Využívá se zde struktury kódu, kdy mimo to, že tento projekt obsahuje třídy definující entity s jejich parametry, tak obsahuje také třídy a rozhraní definující metody pro každou z entit v databázi. Využití tohoto postupu v praxi znamená, že všechny dotazy na získání dat z databáze zůstávají v tomto projektu a v projektu *Rentals.WEB* obsahujícím logiku aplikace se pouze volají metody z *Rentals.DL*, jež tyto dotazy zahrnují.

* + 1. Struktura projektu

Pomyslným základem projektu je adresář tříd *Entities*, jež obsahují předpis databázové tabulky každé z entit. Vlastnosti těchto tříd předznamenávají sloupce tabulky. Jednotlivé vlastnosti pak lze označit validátory zvanými *Data Annotations*, které označují např. vlastnost, jež má sloužit jako identifikátor prvku. Jsou zde i třídy, které definují tzv. spojovací tabulky M:N vazby.

V adresáři s názvem *BussinessLogic* jsou třídy odpovídající třídám v adresáři *Entities*, které jsou o onu „Busssiness logiku“ rozšířeny, tzn. obsahují metody, které určují, jak data v daný tabulkách vytvářet a měnit.

Další částí projektu je adresář *Repositories*, která obsahuje třídy pro tabulky, ze kterých se pomocí metod obsažených v těchto třídách získávají data. Tím, že jsou všechny metody pro získávání dat jedné entity v jedné třídě v jednom souboru výrazně usnadňuje orientaci v souborech aplikace.

Každá třída v adresáři *Repositories* je závislá na odpovídajícím rozhraní nacházejícím se v adresáři *Interfaces*. Každá třída je totiž závislá na svém rozhraní, které osahuje deklarace metod obsažených v konkrétní třídě. Přes rozhraní třídy se také k metodám při jejich použití přistupuje.

Dále projekt obsahuje třídu *EntitiesContext.cs,* která konfiguruje databázi, určuje její tabulky a vztahy mezi nimi.

V adresáři *Migrations* se nachází automaticky generovaný kód, který provádí změny v databázi na základě kódu v souborech a třídách zmíněných výše.

* 1. Projekt *Rentals.Web*

V tomto projektu se nachází aplikace jako taková. Využívá se zde MVC (Model-View-Controller) architektury. To znamená, že každá zobrazovaná stránka v prohlížeči, která je generována dle kódu v souborech formátu *.cshtml* (View) má svůj model. Tento model představuje klasickou třídu, která obsahuje všechna data v podobě vlastností, ke kterým lze přistupovat v souborech formátu *.cshtml*. Onen zmíněný model se plní daty v kontroleru. Kontroler obsahuje metody vracející datový typ *ActionResult* s označením *Data Annotations* specifikujícím URL adresu. Takto označená metoda se poté volá pokaždé, když je v prohlížeči načtena stránka s definovanou URL adresou.

* + 1. Zákaznická část aplikace

Projekt je rozdělen do dvou oblastí. První z nich není ve struktuře souborů aplikace explicitně označena, adresáře se soubory do ní spadající jsou v hlavním adresáři projektu. Tato oblast primárně obstarává část aplikace věnovanou zákazníkům, obsahuje ale také například prostředí pro přihlašování uživatelů. I kvůli tomu je tato část aplikace přístupná všem úspěšně přihlášeným uživatelům. Uživatelé s rolí Zaměstnanec anebo Administrátor mají z této základní oblasti přístup do administrační části aplikace.

* + 1. Administrační část aplikace

Soubory týkající se této části aplikace se nachází v adresáři *Areas.* Oblast je od výchozí oblasti pro zákazníky oddělena. Má vlastní kontrolery i modely pro zobrazování dat. Webové rozhraní této oblasti aplikace se nachází na URL adrese */Admin*.

Uživatelé s rolemi Administrátor anebo Zaměstnanec mohou pomocí funkcí implementovaných funkcí spravovat aplikaci. To zahrnuje správu předmětů, výpůjček a přehled zákazníků.

1. Analýza a dokumentace chyb

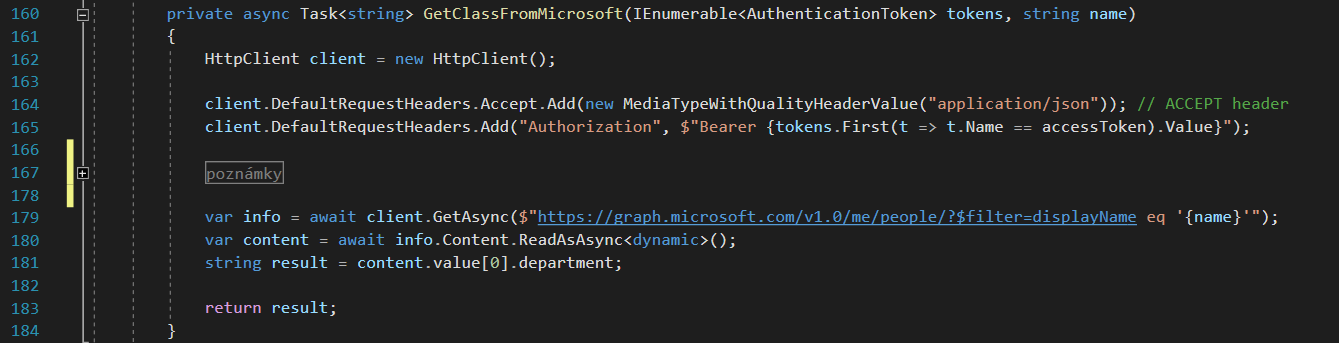
Tato kapitola obsahuje výčet několika hlavních chyb a nedostatků původní verze aplikace, které přímo ovlivňovaly její funkčnost. Jednotlivým chybám se věnují podkapitoly níže.

* 1. Přihlašování uživatelů

Problémy s přihlášením byly hlášeny několika uživateli, kteří tak nemohli využívat aplikace. Počty neúspěšně přihlášených uživatelů se také postupně zvyšovali. Jedním z uživatelských účtů, u kterého nastávala chyba při přihlášení byl i ten můj, což bylo výhodné při odstraňování chyby.

Chyba nastávala v kotroleru *AccountController* v metodě *GetClassFromMicrosoft*, která slouží k získání údaje o tom, do jaké třídy přihlašovaný student chodí.

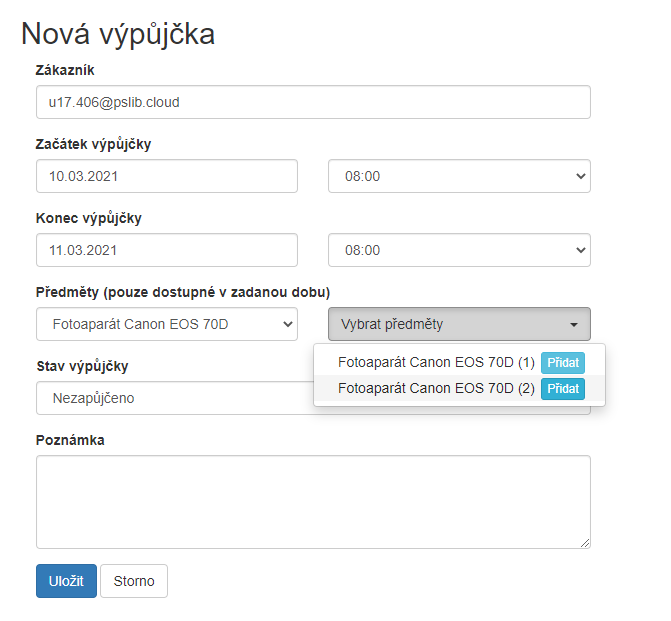
Metoda také obsahovala komentáře autora aplikace upozorňující na poněkud nevhodné řešení metody. Konkrétně zmiňuje problém se získáváním dat o uživateli z *Microsoft Graph API*. V době uvedení aplikace do provozu nebylo jiné řešení možné, a uživatelé v té době s přihlášením problémy neměli.

Metoda přijímá parametr *name*, který představuje jméno přihlašovaného uživatele. Podle hodnoty této proměnné se poté získávají ostatní data o uživateli z *Microsoft Graph API*, mezi něž patří i proměnná *department*, která obsahuje potřebné údaje o třídě přihlašovaného studenta.

Obrázek Původní verze metody GetClassFromMicrosoft

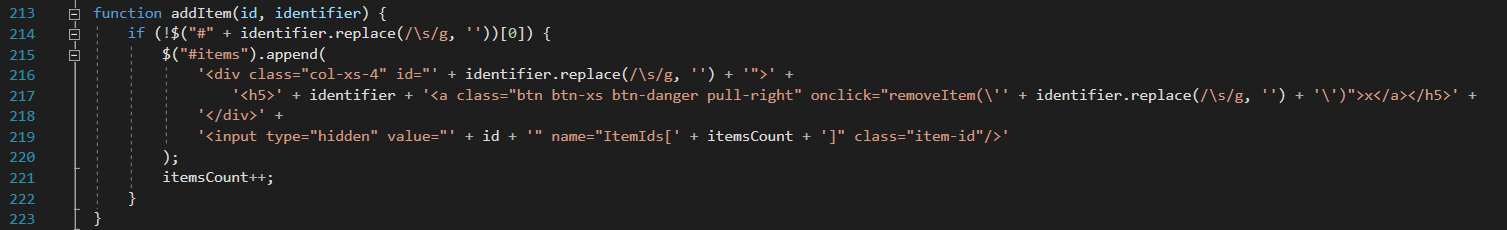
Chyba se projevovala konkrétně tím, že proměnná *info* nebyla naplněna daty z *Microsoft Graph API*. Její hodnota byla tudíž *null*, a proto pokus o přihlášení končil u některých uživatelů chybou.

* 1. Přidávání předmětů do vytvářené výpůjčky

Tato chyba se projevovala při přidávání dostupných předmětů do nově vytvářené výpůjčky v administrační části aplikace. Tato funkce vytváření výpůjček umožňuje uživatelům s rolí administrátora vytvářet výpůjčky pro uživatele registrované v aplikaci.

Obrázek Formulář pro vytvoření výpůjčky

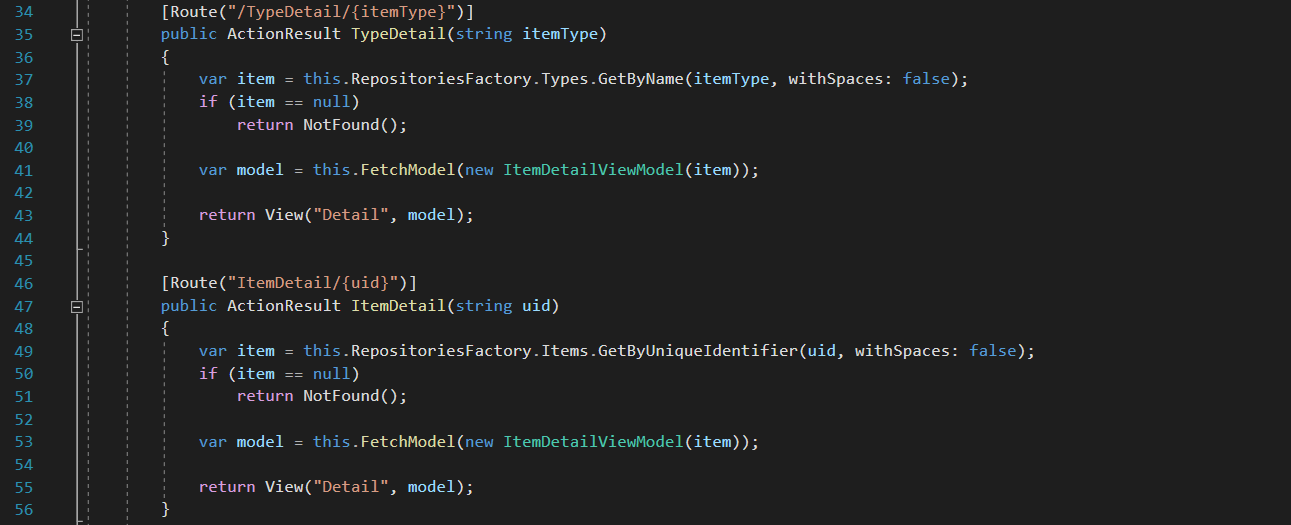
Funkcionalita formuláře (viz Obrázek 4) je řízena pomocí skriptu v jazyce javascript. Pro přidání předmětů do vytvářené výpůjčky je třeba nejdříve zvolit typ předmětu, který má být přidán. Tím je pomocí knihovny *AJAX* asynchronně volána metoda *GetAvalibeItems*, která vrací všechny předměty zvoleného typu dostupné ve zvoleném časovém rozmezí. Z těchto předmětů pak lze přidat konkrétní předměty do výpůjčky.

Při kliknutí na tlačítko *Přidat* (viz Obrázek 4) je volána funkce *addItem* (viz Obrázek 5), která přidá předmět do seznamu předmětů, který je touto funkcí generovaný. Předmět je do seznamu přidán v závislosti na jeho atributu *uniqueIdentifier*, který je ve funkci *addItem* znázorněn jejím parametrem *identifier*. Jedná se o proměnnou typu *string*, která slouží i jako zobrazovaný název předmětu.

Obrázek Původní verze funkce addItem

Chyba nastává při pokusu o přidání předmětu, jehož *uniqueIdentifier* obsahuje znaky kulatých závorek. Funkce *addItem* neproběhne, neboť podmínka na řádku 214 (viz Obrázek 5) není splněna, což je zapříčiněno využitím funkcionality knihovny jQuery. Konkrétně chybu způsobuje využití „[0]“ pro získání prvního HTML objektu, jehož hodnota atributu id obsahuje kulaté závorky, v podmínce.

* 1. Zobrazení detailu předmětu

Tato chyba nastávala při pokusu o zobrazení stránky s detailem předmětu nebo typu předmětu, jehož atribut *UniqueIdentifier* obsahoval znak lomítko „/“. Chybě se předcházelo nepoužíváním tohoto znaku v označení předmětů. Jelikož se ale atribut předmětu *UniqueIdentifier* používá i jako jeho název zobrazovaný zákazníkům a uživatelům aplikace, bylo toto omezení v podobě nemožnosti označit předmět názvem obsahujícím lomítko velmi nevyhovující.

Obrázek Původní verze metod TypeDetail a ItemDetail

Konkrétně chybu způsobují metody *TypeDetail* a *ItemDetail*. Tyto metody slouží jako tzv. endpoint. Liší se tím, že metoda *ItemDetail* je volána, pokud má být zobrazena stránka s detailními informacemi o jednom konkrétním předmětu. Zato *TypeDetail* vrací detailní stránku, pokud je k dispozici více předmětů stejného typu. Metody přijímají jako parametr datový typ *string*, který získávají přímo z URL adres, kterými jsou volány. Formát těchto adres je určen na řádcích číslo 34 a 46 (viz Obrázek 6).

Metody jsou volány po tom, co uživatel zvolí některý z nabízených předmětů na hlavní stránce aplikace a URL adresa se změní tak, že odkazuje na jednu z metod *TypeDetail* nebo *ItemDetail.*

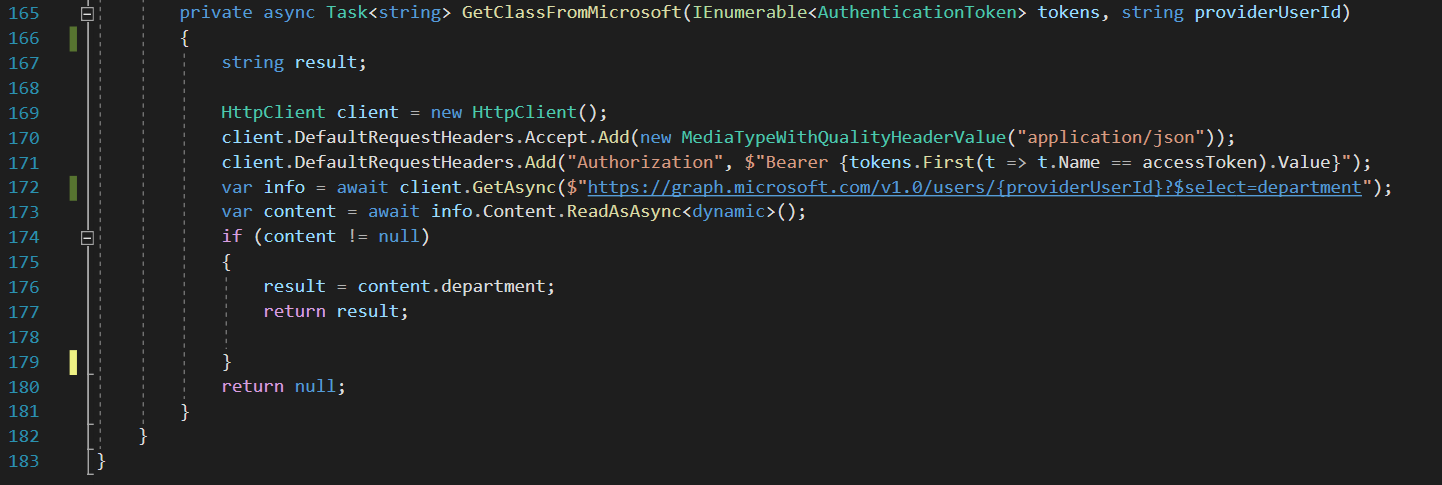
Tento způsob zápisu metod a použití datového typu *string* pro odkázání na stránku s detailem předmětu způsobuje, že URL adresa může v její části obsahující identifikátor předmětu obsahovat téměř jakékoli znaky. Pokud se v této části objeví například zmiňovaný znak lomítka, nebo jiný z tzv. rezervovaných znaků (3), adresa může být prohlížečem misinterpretována. Důsledkem toho je nenalezení stránky.

1. Oprava chyb a úprava stávajících řešení

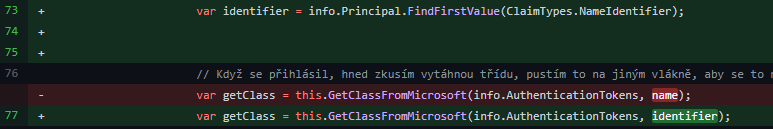
Tato kapitola je věnována popisu oprav chyb popsaných ve čtvrté kapitole a některým provedeným úpravám, které zlepšují orientaci uživatele v aplikaci a zpřehledňují strukturu zobrazovaných dat. Seznam všech provedených změn se nachází v příloze této práce.

* 1. Přihlašování uživatelů

Jak je již zmíněno výše v první podkapitole čtvrté kapitoly, chyba byla v získávání záznamu o třídě přihlašovaného uživatele. Díky tomu nebylo přihlášení některých uživatelů možné.

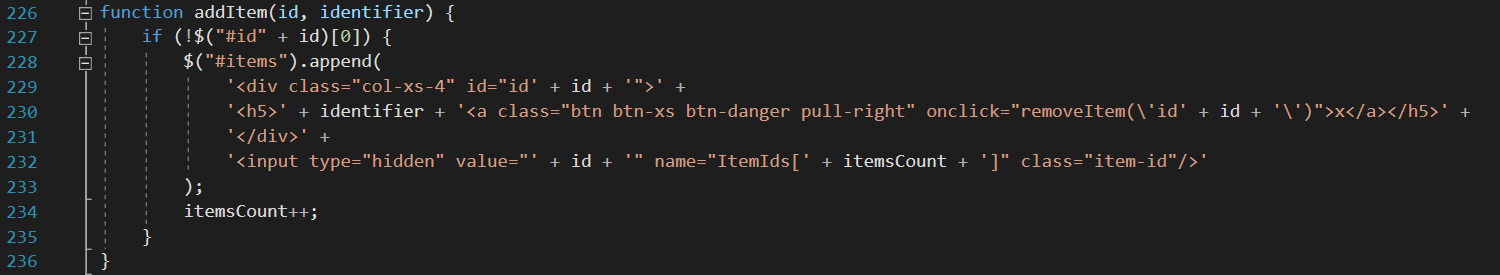
Inspirací při úpravách metody *GetClassFromMicrosoft* v kotroleru *AccountController* mi byl poskytnutý kód používaný pro získání dat o přihlašovaném uživateli z *Microsoft Graph API* v jiných školních aplikacích. Z tohoto kódu pochází formát URL adresy odkazující na API, který byl ale oproti svému vzoru značně zjednodušen, kvůli potřebě získání pouze atributu *department*.

Obrázek Opravená metoda GetClassFromMicrosoft

Zásadním rozdílem oproti původní verzi je ale získávání dat o uživateli z *Microsoft Graph API* na základě jeho identifikátoru, nikoli jeho jména, jak tomu bylo u původní verze. Bylo proto třeba plnit parametr metody datového typu *string* identifikátorem uživatele místo jména uživatele. Hodnota tohoto identifikátoru se získává z tzv. Claims uživatele, které obsahují základní data o přihlášeném uživateli.

Obrázek Změny ve volání metody GetClassFromMicrosoft

* 1. Přidávání předmětů do vytvářené výpůjčky

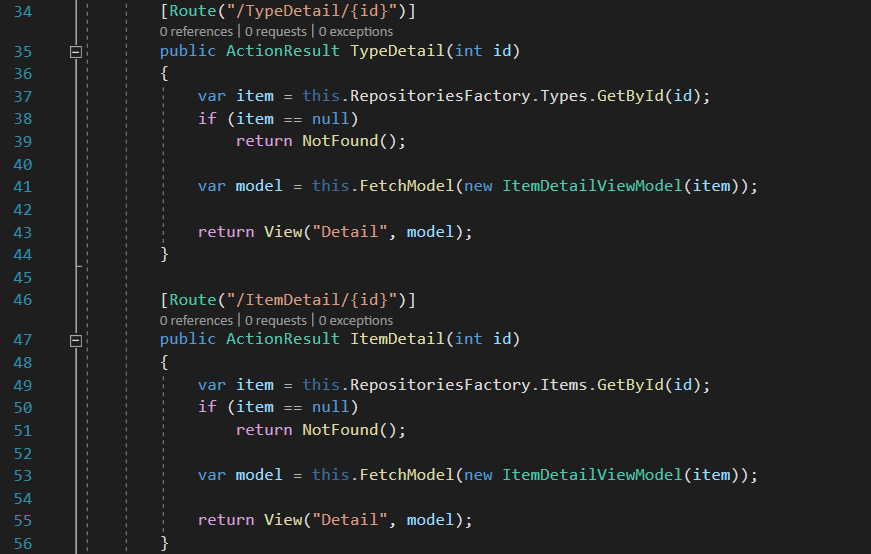
Chyba popsána ve druhé podkapitole čtvrté kapitoly, spočívala v kontrolování přítomnosti html prvku na webové stránce pomocí JavaScript knihovny jQuery. Přítomnost prvku se kontroluje ve funkci *addItem* na základě jeho atributu *id*. Funkce je psána programovacím jazykem JavaScript na stránce pro vytváření nových výpůjček v administračním prostředí aplikace. Chyba nastává v případě, pokud má být zjištěna přítomnost HTML prvku jehož hodnota atributu *id* obsahuje znaky kulatých závorek.

Obrázek Opravená funkce addItem

Opravená verze funkce *addItem* (viz Obrázek 9) přijímá stejně jako její původní verze (viz Obrázek 5) dva parametry. Prvním z nich je parametr *id* obsahující unikátní identifikátor předmětu datového typu *integer* v databázi. Druhým je parametrem je *identifier*, který představuje atribut předmětu *uniqueIdentifier*, jenž slouží jako unikátní název předmětu.

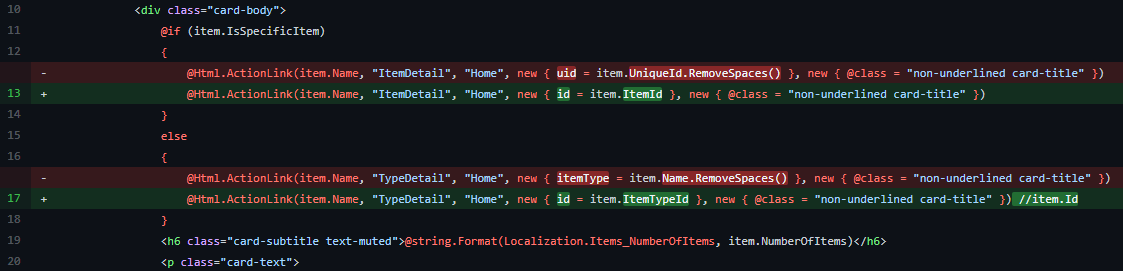
Oprava spočívá ve využití parametru *id* pro nastavení atributu *id* nově vytvářeného HTML prvku (viz řádek 229, Obrázek 9). Jelikož hodnota atributu *id* HTML prvků musí začínat některým ze znaků [A-Za-z] (4), přidává se k parametru funkce *id* navíc „id“, neboť jak je již zmíněno výše parametr *id* obsahuje hodnotu v podobě datového typu *integer*. Tím se eliminuje riziko, že atribut HTML prvku *id* bude obsahovat znaky, které by mohly působit problémy při kontrole přítomnosti HTML prvku s daným atributem *id* pomocí knihovny jQuery.

* 1. Zobrazení detailu předmětu

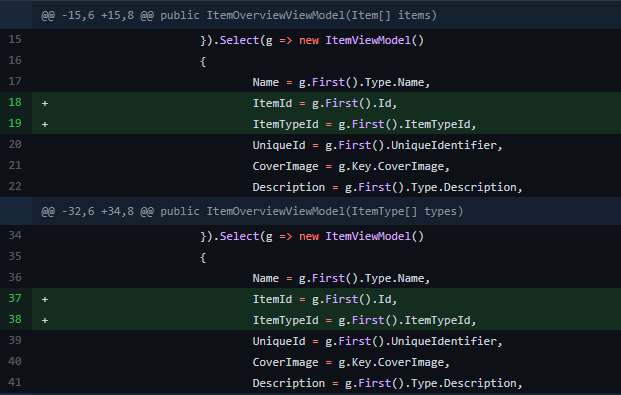
Oprava tohoto problému, detailně popsaného ve třetí podkapitole čtvrté kapitoly, si vyžádala úpravy zdrojových kódů napříč několika soubory aplikace. Chyba nastávala při pokusu o zobrazení stránky s detailními informacemi o předmětu, jehož název obsahoval některý z rezervovaných znaků URL adresy. Příčina této chyby byla ve využívání atributu předmětu *UniqueIdentifier* k odkázání na stránku s jeho detailními informacemi. Tento atribut je datového typu *string* a využívá se i jako zobrazovaný název předmětu. Řešením proto bylo využití unikátního identifikátoru předmětu datového typu *integer* z databáze.

Obrázek Upravené metody TypeDetail a ItemDetail

To znamenalo předělat metody *TypeDetail* a *ItemDetail* kontroleru *HomeController* sloužící jako tzv. endpointy zobrazující ony stránky s detailními informacemi o předmětech. Metoda *TypeDetail* je volána, pokud se zobrazuje detailní stránka pro typ předmětu. To se děje, pokud je dostupných více předmětů stejného typu. Metoda *ItemDetail* je pak volána pro zobrazení detailní stránky jednoho konkrétního předmětu, tzn. volá se, pokud je dostupný pouze jeden konkrétní předmět.

Změny byly provedeny v parametru obou metod, kdy místo proměnné datového typu *string* metody nyní přijímají proměnnou datového typu *integer*. V závislosti na této změně musely být změněny i metody, které vrací instanci předmětu s danou hodnotou identifikátoru z databáze. V obou případech (na řádcích 37 a 49) byla použita v projektu již zavedená metoda *GetById*, což bylo možné díky využití generického datového typu v definici této metody. 

Obrázek Změny ve volání metod TypeDetail a ItemDetail v komponentu ItemsOverview

Změny v parametrech metod se projevily i ve způsobu jejich volání (viz Obrázek 11), kde bylo třeba předat metodě identifikátor předmětu datového typu *integer* namísto identifikátoru datového typu *string*.

Obrázek Úpravy třídy ItemOverViewModel

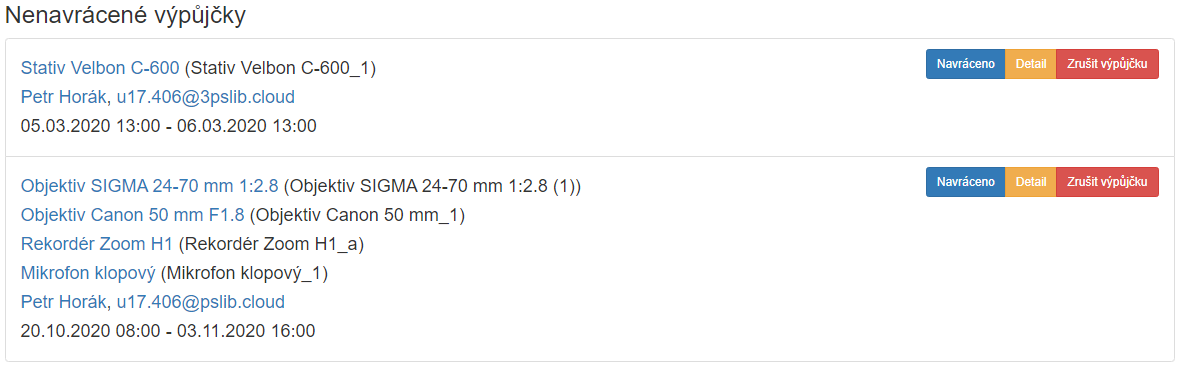
Tyto změny ve volání metod *TypeDetail* a *ItemDetail* si vyžádaly změny v modelu *ItemOverviewViewModel* komponentu *ItemsOverview*, aby bylo možné přistupovat k identifikátorům předmětů. *ItemOverviewViewModel* obsahuje pouze jednu vlastnost *Items*. Jedná se o pole datového typu *ItemViewModel*, které obsahuje data pro každý z vypůjčitelných předmětů. Bylo tak třeba upravit i *ItemViewModel*, kam byly přidány vlastnosti *ItemId* a *ItemTypeId*, které jsou plněny hodnotami stejnojmenných vlastností databázové entity *Item*, respektive *ItemType* (viz Obrázek 12).

* 1. Úpravy stávajících řešení

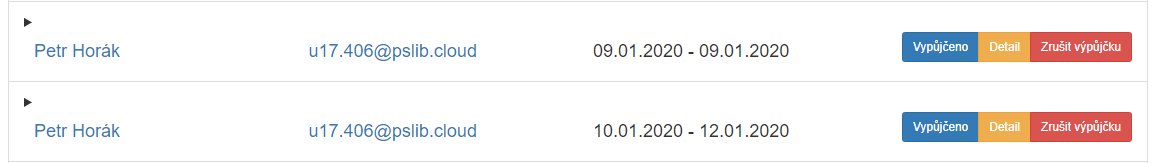
Tato podkapitola se zaobírá provedenými úpravami některých částí původní aplikace, které se během používání aplikace projevily jako nevhodně řešené.

* + 1. Zobrazení náhledu výpůjčky

Cílem této úpravy bylo zpřehlednit zobrazení náhledů výpůjček používaných na více stránkách aplikace.

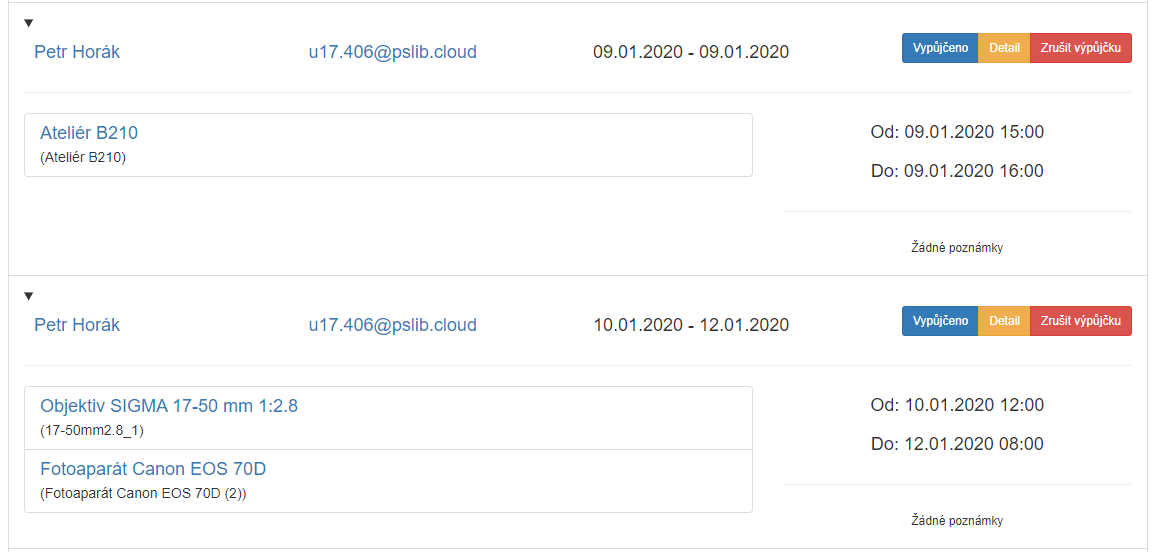
Úpravy byly provedeny v tzv. partialview *\_Rentings.cshtml*, který se používá k zobrazení výčtu výpůjček. Modelem tohoto partialview je kolekce *RentingViewModel*, což je viewmodel obsahující detailní informace o výpůjčce.

Obrázek Původní verze zobrazení náhledů výpůjček

Původní verze sice obsahovala všechny potřebné informace (viz Obrázek 13), ale struktura zobrazovaných dat byla nepřehledná a matoucí. Při zobrazení výčtu s vyšším počtem výpůjček byla přehlednost kvůli nestále výšce sekcí s informacemi o jednotlivých výpůjčkách ještě nižší.

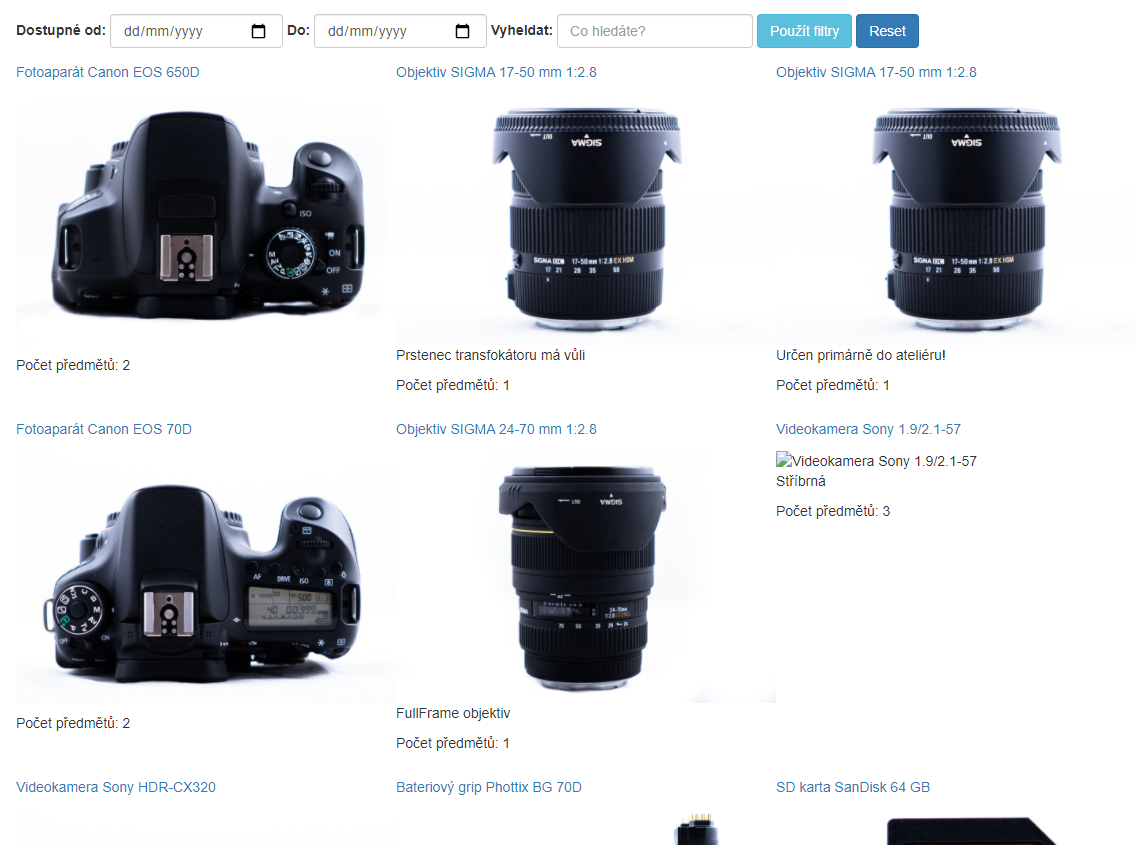
Obrázek Upravená verze náhledu výpůjček se základními informacemi

Nová struktura zobrazení dat jednotlivých výpůjček ve výčtu výpůjček byla inspirována poskytnutými návrhy. Využívá se zde HTML značky *details*, která obsahuje značku *summary*. Značka *summary* v rámci značky *details* definuje oblast s HTML elementy, jež mají být zobrazeny vždy (viz Obrázek 14). Tato oblast obsahuje základní data o výpůjčce a tlačítka sloužící pro správu výpůjčky. Díky její zpravidla fixní výšce jsou výčty více výpůjček přehlednější a čitelnější.

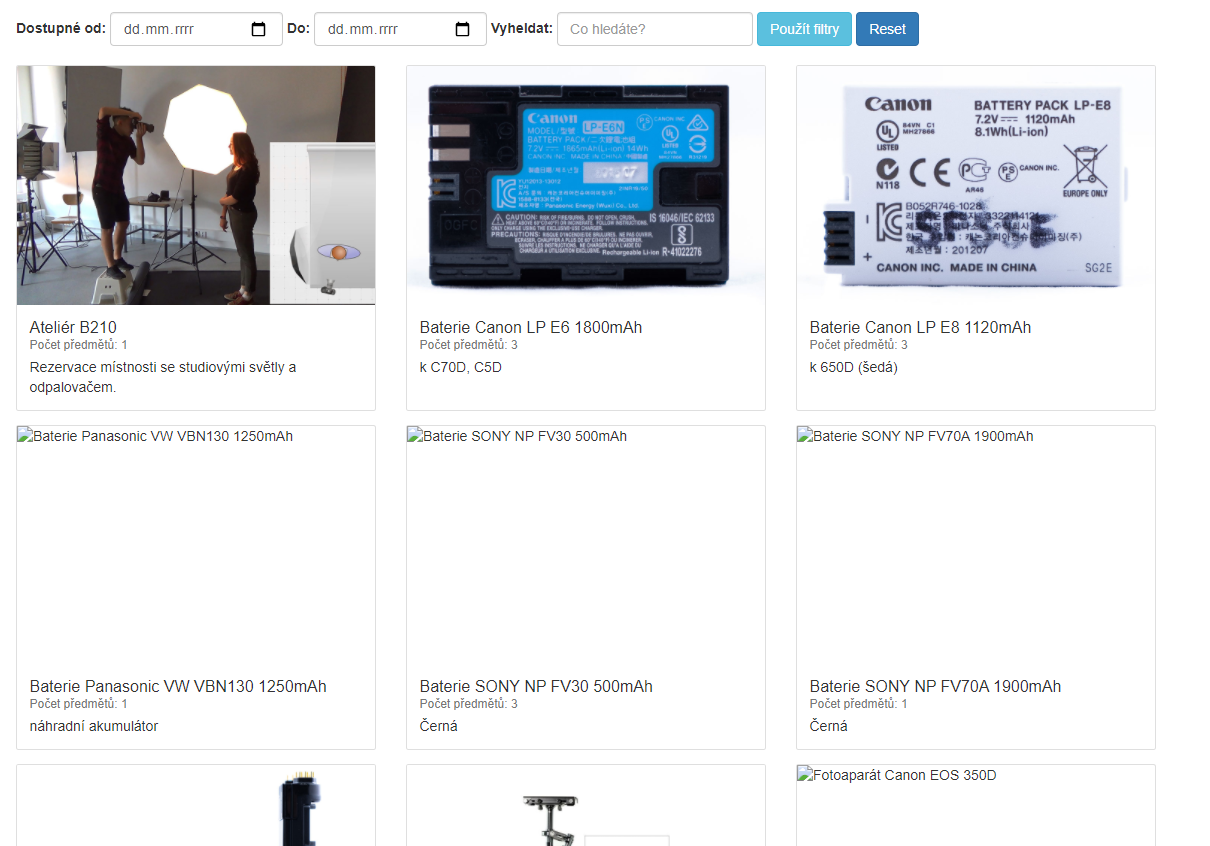
Po kliknutí na oblast značky *summary* jsou zobrazeny i ostatní HTML elementy v rámci značky *details*. Tím lze mimo jiné zobrazit i seznam vypůjčených předmětů. Této nové verze zobrazení se využívá na všech stránkách aplikace, kde se zobrazuje výčet výpůjček.

Obrázek Upravená verze náhledu výpůjček se podrobnými informacemi

* + 1. Zobrazení přehledu předmětů

Tato úprava se týkala zobrazení přehledu předmětů k vypůjčení. Při úpravách jsem vycházel z vlastní zkušenosti, neboť když jsem se s aplikací setkal poprvé, přišlo mi původní řešení matoucí. Proto jsem se při úpravách soustředil na zpřehlednění zobrazovaného výčtu předmětů, aby byla ulehčena orientace uživatele na stránce.

Obrázek Původní zobrazení přehledu předmětů k vypůjčení

Úprava spočívala i v seřazení předmětů podle abecedy, tudíž bylo nutné přidat k dotazu pro získání dat předmětů z databáze funkci *OrderByDescending*.

Obrázek Upravené zobrazení přehledu předmětů k vypůjčení

Stylování vychází z CSS knihovny *Bootstrap*. Údaje týkající se jednoho předmětu jsou zřetelněji oddělené od ostatních. Výška jednotlivých řádků s kartami předmětů je responzivní, tzn. výška karet v jednom řádku se přizpůsobí té nejvyšší.

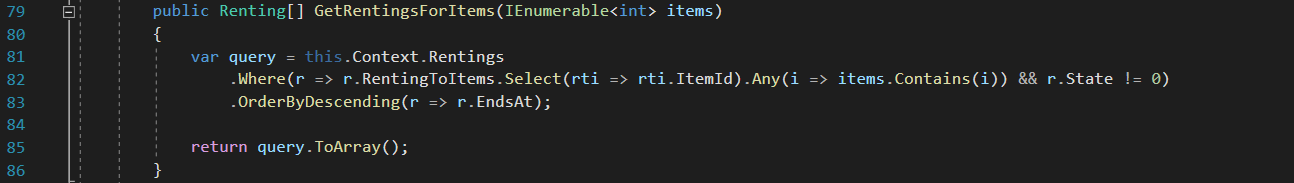
Obdobného stylování zobrazení výčtu předmětů se používá u přehledu příslušenství na stránkách s detailními informacemi jednotlivých předmětů.

* + 1. Funkcionalita tabulky zákazníků

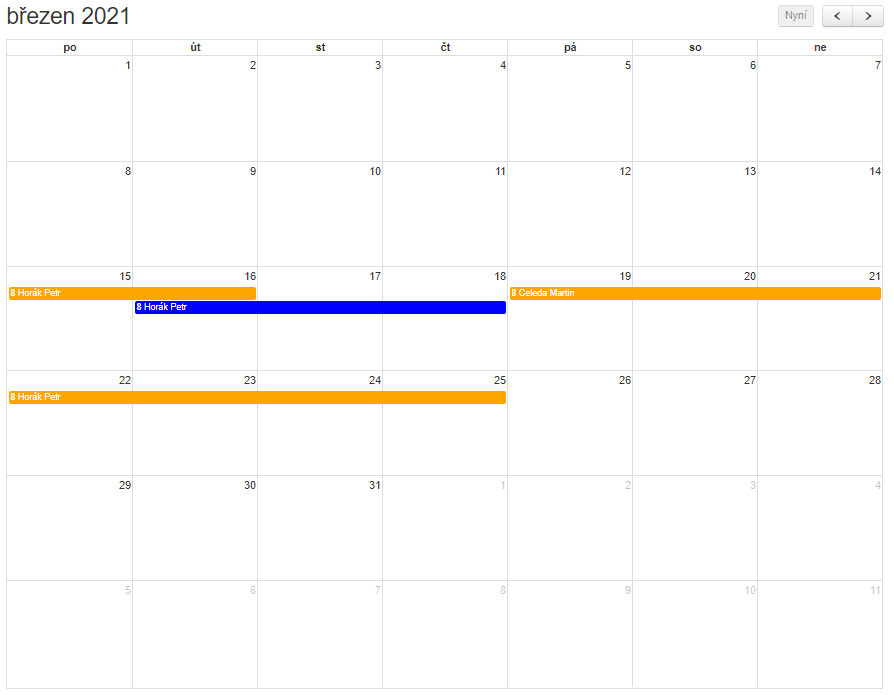
V administračním prostředí lze zobrazit seznam všech registrovaných zákazníků, který měl v původní verzi aplikace podobu prosté tabulky. S přibývajícími počty registrovaných zákazníků je toto řešení nevyhovující.

Využito bylo JavaScript knihovny *SimleDatatables*. Díky této úpravě je možné rozdělit rozsáhlou tabulku na více stran, mezi kterými se dá přecházet. Dále je možné vyhledávat v tabulce nebo řadit data v tabulce dle jednotlivých sloupců.

* + 1. Detail předmětu v administračním prostředí

Na stránku s detailem typu předmětu v administračním prostředí sloužící ke správě předmětu byla přidána historie všech výpůjček obsahujících tento některý z konkrétních předmětů tohoto typu. Využívá se tudíž partialview *\_Rentings*, jehož úpravy jsou popsány v první podkapitole této kapitoly.

Obrázek Nová metoda GetRentingsForItems

Zobrazování historie výpůjček jednotlivých předmětů si vyžádalo vytvoření nové metody získávající data z databáze. Metoda *GetRentingsForItems* přijímá jako parametr kolekci identifikátorů konkrétních předmětů, pro které vrací pole výpůjček. Následně musel být změněn model stránky *ExtendedItemTypeViewModel*, ve kterém přibylo pole datového typu *RentingViewModel* obsahující data všech zobrazovaných výpůjček. Tato vlastnost modelu stránky je plněna daty skrze nový parametr modelu přijímající pole výpůjček. Data pro tento parametr obstarává metoda *GetRentingsForItems* (viz Obrázek 18) volaná v kontroleru *ItemsController*.

Obrázek Nový kalendář na stránce /Admin/Items/Detail/

Dále byl na tuto stránku aplikace přidán měsíční kalendář zobrazující historii výpůjček graficky. Barvy výpůjček znázorňují stav výpůjčky. Po kliknutí na výpůjčku v kalendáři lze přejít na stránku s detailními informacemi o výpůjčce. Při najetí myší nad výpůjčku se zobrazí malý seznam názvů všech vypůjčených předmětů v rámci výpůjčky.

Kalendář je generován pomocí funkce v programovacím jazyce JavaScript využívající knihovny *Fullcalendar*.

Závěr

Tak jsem to dokázal! A teď ještě, co jsem mohl udělat jinak a jak by se dalo v práci pokračovat.

Práce mi pomohla zdokonalit se v oblasti, která mě při studiu zaujala nejvíce, v tvorbě webových aplikací.

a navíc díky ní vznikla pro uživatele – studenty a personál SPŠSE a VOŠ – pohodlnější a méně problémová webová aplikace.

Seznam obrázků

[Obrázek 1 Příklad generovaného návrhu řešení vývojovým prostředím MS Visual Studio 2](file:///D:\Osobní%20-%20záloha%20(foto,%20kazety,%20důležité)\Škola\SPŠSE%20Liberec\MATURITA\MP2021-Petr-Horák-Revize_aplikace_Rentals.docx#_Toc65840742)

[Obrázek 2 Chybová hláška programu Microsoft SQL Server Management Studio 4](file:///D:\Osobní%20-%20záloha%20(foto,%20kazety,%20důležité)\Škola\SPŠSE%20Liberec\MATURITA\MP2021-Petr-Horák-Revize_aplikace_Rentals.docx#_Toc65840743)

Použitá literatura

1. **Wikipedia. *Bak file.* [Online] 18. Únor 2021. https://en.wikipedia.org/wiki/Bak\_file.**

**2. učitelé SPŠSE. Úvod. *SPŠSE a VOŠ Liberec.* [Online] 01. 09 2016. [Citace: 01. 09 2016.] https://www.pslib.cz.**

**3. nuget. [Online] Microsoft. https://www.nuget.org/.**

**4. Introduction to Identity on ASP.NET Core. *Microsoft Docs.* [Online] Microsoft , 15. 07 2020. https://docs.microsoft.com/cs-cz/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio.**

1. Seznam přiložených souborů

Text

1. Další příloha