

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Revize aplikace Rentals

Petr Horák
Liberecký kraj

Liberec 2020

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Revize aplikace Rentals

The revision of Rentals application

Autoři: Petr Horák

Škola: Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3, příspěvková organizace, 460 01, Liberec

Kraj: Liberecký kraj

Konzultant: Ing. Tomáš Kazda, DiS.

Liberec 2020

Anotace (Resumé)

Tato odborná 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 napravuje 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.

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Liberci dne 19.03.2021.

.....

Petr Horák

Obsah

Úvod.....	1
1 Nasazení ve vývojovém prostředí.....	2
2 Přenos databáze do testovacího prostředí.....	3
2.1 Vložení dat do lokální databáze.....	3
2.1.1 Vložení dat do databáze pomocí souboru formátu <i>.bak</i>	3
2.1.2 Vložení dat do databáze pomocí SQL příkazů.....	4
3 Analýza zdrojových kódů.....	5
3.1 Projekt <i>Rentals.Common</i>	5
3.2 Projekt <i>Rentals.DL</i>	5
3.2.1 Struktura projektu.....	5
3.3 Projekt <i>Rentals.Web</i>	6
3.3.1 Zákaznická část aplikace.....	6
3.3.2 Administrační část aplikace.....	7
4 Analýza a dokumentace chyb.....	8
4.1 Přihlašování uživatelů.....	8
4.2 Přidávání předmětů do vytvářené výpůjčky.....	9
4.3 Zobrazení detailu předmětu.....	10
5 Oprava chyb a úprava stávajících řešení.....	12
5.1 Přihlašování uživatelů.....	12
5.2 Přidávání předmětů do vytvářené výpůjčky.....	13
5.3 Zobrazení detailu předmětu.....	13
5.4 Úpravy stávajících řešení.....	15
5.4.1 Zobrazení náhledu výpůjčky.....	16
5.4.2 Zobrazení přehledu předmětů.....	17
5.4.3 Funkcionalita tabulky zákazníků.....	18
5.4.4 Detail předmětu v administračním prostředí.....	18
5.4.5 Kalendář výpůjček.....	21
5.4.6 Ostatní úpravy.....	26
6 Návrh řešení nedostatků aplikace.....	27
Závěr.....	28
Seznam obrázků.....	29
Použitá literatura.....	30
A. Seznam přiložených souborů.....	1

Úvod

Téma této práce jsem si zvolil, neboť splňuje vše podstatné, co jsem od dlouhodobé odborné 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.

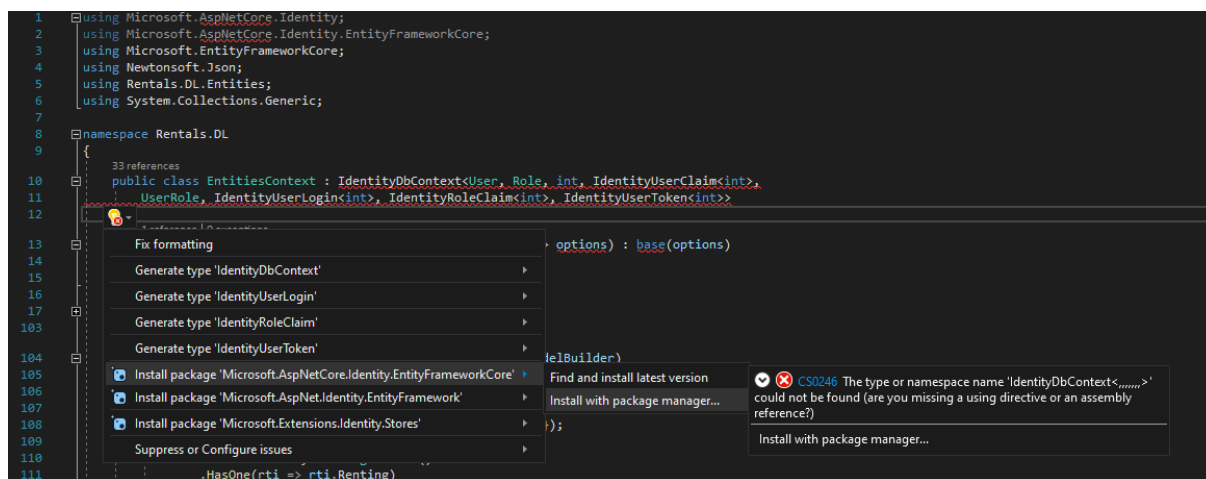
V tomto dokumentu popisuji postup, jakým jsem řešil úpravy původní verze aplikace, tak, aby nová verze byla plně funkční, méně problémová a přehlednější pro její uživatele.

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 programu Visual Studio, jazyka 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 implementuje ASP.NET Core Identity, což je API (Application Programming Interface), které umožňuje do aplikace zavést přihlašování jejích uživatelů uživatelskými účty. Z důvodu kompatibility je použita verze ASP.NET Core Identity 2.1.3.



Obrázek 1 Příklad generovaného návrhu řešení vývojovým prostředím MS Visual Studio

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

2 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*).

2.1 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 webová aplikace.

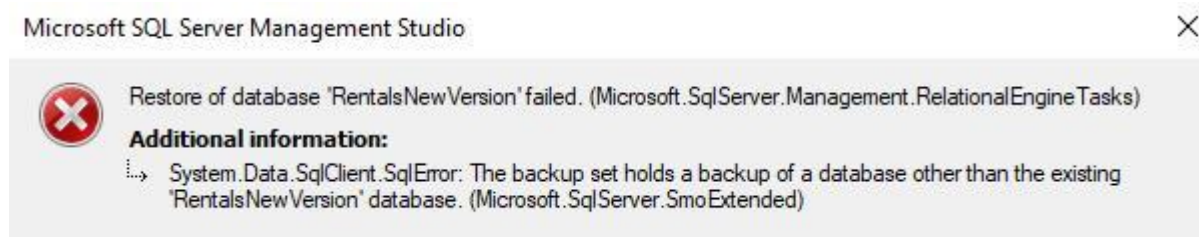
2.1.1 Vložení dat do databáze pomocí souboru formátu *.bak*

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 souboru 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 2 Chybová hláška programu *Microsoft SQL Server Management Studio*

2.1.2 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* úspěšný nebude. Data z původní databáze jsem proto dostal k dispozici v souborech ve formátu *.sql*. Tyto soubory obsahovaly 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 provedení příkazu *Update-Database*, vytvořil jsem databázi novou. Tuto databázi 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 rádob relevantních dat.

3 Analýza zdrojových kódů

Zdrojové kódy aplikace jsou psány programovacím jazykem C#. Je to jeden z nejrozšířenějších vysokoúrovňových programovacích jazyků. Používá se k objektovému programování. Byl vyvinut společností *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.

3.1 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.

3.2 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.

3.2.1 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 „Bussiness logiku“ rozšířeny. Obsahují metody, které určují, jak data v daných 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. To, 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 obsahuje 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.

3.3 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.

3.3.1 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.

3.3.2 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 implementovaných funkcí spravovat aplikaci. To zahrnuje správu předmětů, výpůjček a přehled zákazníků.

4 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.

4.1 Přihlašování uživatelů

Problémy s přihlášením byly hlášeny několika uživateli, kteří tak nemohli využívat aplikace. Počty neúspěšně přihlášených uživatelů se také postupně zvyšovaly. 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.

```
160 private async Task<string> GetClassFromMicrosoft(IEnumerable<AuthenticationToken> tokens, string name)
161 {
162     HttpClient client = new HttpClient();
163
164     client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json")); // ACCEPT header
165     client.DefaultRequestHeaders.Add("Authorization", $"Bearer {tokens.First(t => t.Name == accessToken).Value}");
166
167     poznámky
168
169     var info = await client.GetAsync($"https://graph.microsoft.com/v1.0/me/people/?$filter=displayName eq '{name}'");
170     var content = await info.Content.ReadAsAsync<dynamic>();
171     string result = content.value[0].department;
172
173     return result;
174 }
```

Obrázek 3 Původní verze metody *GetClassFromMicrosoft*

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.

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.

4.2 Přidávání předmětů do vytvářené výpůjčky

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

Nová výpůjčka

Obrázek 4 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 *GetAvalibelItems*, která vrací všechny předměty zvoleného typu dostupné ve zvoleném časovém rozmezí. Z těchto předmětů pak lze přidat konkrétní předměty do výpůjčky.

```

213 function addItem(id, identifier) {
214     if (!$("#" + identifier.replace(/\s/g, ''))[0]) {
215         $("#items").append(
216             '<div class="col-xs-4" id="' + identifier.replace(/\s/g, '') + '">' +
217             '<h5>' + identifier + '<a class="btn btn-xs btn-danger pull-right" onclick="removeItem(\'' + identifier.replace(/\s/g, '') + '\')">x</a></h5>' +
218             '</div>' +
219             '<input type="hidden" value="' + id + '" name="ItemIds[' + itemsCount + ']" class="item-id"/>'
220         );
221         itemsCount++;
222     }
223 }
```

Obrázek 5 Původní verze funkce addItem

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.

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.

4.3 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.

```

34 [Route("/TypeDetail/{itemType}")]
35 public ActionResult TypeDetail(string itemType)
36 {
37     var item = this.RepositoriesFactory.Types.GetByName(itemType, withSpaces: false);
38     if (item == null)
39         return NotFound();
40
41     var model = this.FetchModel(new ItemDetailViewModel(item));
42
43     return View("Detail", model);
44 }
45
46 [Route("ItemDetail/{uid}")]
47 public ActionResult ItemDetail(string uid)
48 {
49     var item = this.RepositoriesFactory.Items.GetByUniqueIdentifier(uid, withSpaces: false);
50     if (item == null)
51         return NotFound();
52
53     var model = this.FetchModel(new ItemDetailViewModel(item));
54
55     return View("Detail", model);
56 }

```

Obrázek 6 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 URL adresy (3), adresa může být prohlížečem misinterpretována. Důsledkem toho je nenalezení stránky.

5 Oprava chyb a úprava stávajících řešení

Tato kapitola je věnována popisu oprav chyb popsanych ve čtvrté kapitole a některým provedeným úpravám, které zlepšují orientaci uživatele v aplikaci a zpřehledňují strukturu zobrazovaných dat.

5.1 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*.

```

165 private async Task<string> GetClassFromMicrosoft(IEnumerable<AuthenticationToken> tokens, string providerUserId)
166 {
167     string result;
168
169     HttpClient client = new HttpClient();
170     client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
171     client.DefaultRequestHeaders.Add("Authorization", $"Bearer {tokens.First(t => t.Name == accessToken).Value}");
172     var info = await client.GetAsync($"https://graph.microsoft.com/v1.0/users/{providerUserId}?select=department");
173     var content = await info.Content.ReadAsAsync<dynamic>();
174     if (content != null)
175     {
176         result = content.department;
177         return result;
178     }
179     return null;
180 }
181
182
183

```

Obrázek 7 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.

```

73 +         var identifier = info.Principal.FindFirstValue(ClaimTypes.NameIdentifier);
74 +
75 +
76 // Když se přihlásil, hned zkusím vytáhnout třídu, pustím to na jiném vlákně, aby se to
77 -         var getClass = this.GetClassFromMicrosoft(info.AuthenticationTokens, name);
77 +         var getClass = this.GetClassFromMicrosoft(info.AuthenticationTokens, identifier);

```

Obrázek 8 Změny ve volání metody *GetClassFromMicrosoft*

5.2 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.

```

226 function addItem(id, identifier) {
227     if (!$("#id" + id)[0]) {
228         $("#items").append(
229             '<div class="col-xs-4" id="id" + id + ">' +
230             '<h5>' + identifier + '<a class="btn btn-xs btn-danger pull-right" onclick="removeItem(\'id\' + id + '\')">x</a></h5>' +
231             '</div>' +
232             '<input type="hidden" value="" + id + "" name="ItemIds[' + itemsCount + ']" class="item-id"/>'
233         );
234         itemsCount++;
235     }
236 }

```

Obrázek 9 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 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.

5.3 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.



```

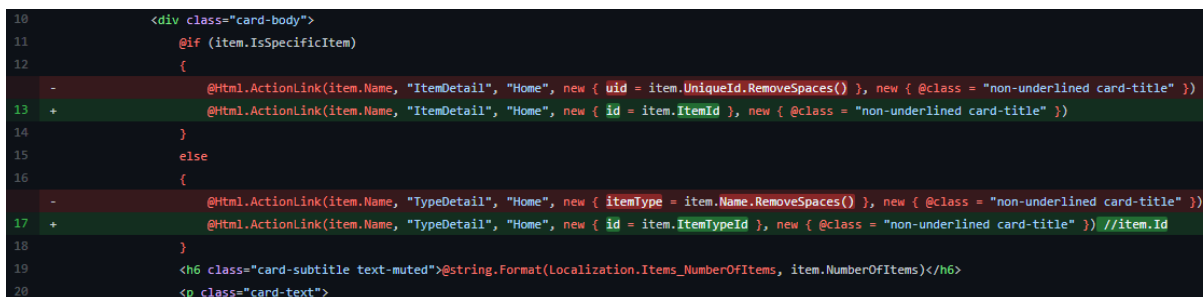
34 [Route("/TypeDetail/{id}")]
35 0 references | 0 requests | 0 exceptions
36 public ActionResult TypeDetail(int id)
37 {
38     var item = this.RepositoriesFactory.Types.GetById(id);
39     if (item == null)
40         return NotFound();
41
42     var model = this.FetchModel(new ItemDetailViewModel(item));
43
44     return View("Detail", model);
45 }
46
47 [Route("/ItemDetail/{id}")]
48 0 references | 0 requests | 0 exceptions
49 public ActionResult ItemDetail(int id)
50 {
51     var item = this.RepositoriesFactory.Items.GetById(id);
52     if (item == null)
53         return NotFound();
54
55     var model = this.FetchModel(new ItemDetailViewModel(item));
56
57     return View("Detail", model);
58 }

```

Obrázek 10 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.



```

10 <div class="card-body">
11     @if (item.IsSpecificItem)
12     {
13         - @Html.ActionLink(item.Name, "ItemDetail", "Home", new { uid = item.UniqueId.RemoveSpaces() }, new { @class = "non-underlined card-title" })
14         + @Html.ActionLink(item.Name, "ItemDetail", "Home", new { id = item.ItemId }, new { @class = "non-underlined card-title" })
15     }
16     else
17     {
18         - @Html.ActionLink(item.Name, "TypeDetail", "Home", new { itemType = item.Name.RemoveSpaces() }, new { @class = "non-underlined card-title" })
19         + @Html.ActionLink(item.Name, "TypeDetail", "Home", new { id = item.ItemTypeId }, new { @class = "non-underlined card-title" }) //item.Id
20     }
21     <h6 class="card-subtitle text-muted">@string.Format(Localization.Items_NumberOfItems, item.NumberOfItems)</h6>
22     <p class="card-text">

```

Obrázek 11 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*.

```

@@ -15,6 +15,8 @@ public ItemOverviewViewModel(Item[] items)
15         }).Select(g => new ItemViewModel()
16         {
17             Name = g.First().Type.Name,
18 +         ItemId = g.First().Id,
19 +         ItemTypeId = g.First().ItemTypeId,
20             UniqueId = g.First().UniqueIdentifier,
21             CoverImage = g.Key.CoverImage,
22             Description = g.First().Type.Description,
@@ -32,6 +34,8 @@ public ItemOverviewViewModel(ItemType[] types)
34         }).Select(g => new ItemViewModel()
35         {
36             Name = g.First().Type.Name,
37 +         ItemId = g.First().Id,
38 +         ItemTypeId = g.First().ItemTypeId,
39             UniqueId = g.First().UniqueIdentifier,
40             CoverImage = g.Key.CoverImage,
41             Description = g.First().Type.Description,

```

Obrázek 12 Úpravy třídy *ItemOverViewModel*

Tyto změny ve volání metod *TypeDetail* a *ItemDetail* si vyžádaly změny v modelu *ItemOverviewViewModel* komponenty *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).

5.4 Úpravy stávajících řešení

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

5.4.1 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.

Nenavrácené výpůjčky

Stativ Velbon C-600 (Stativ Velbon C-600_1) Petr Horák, u17.406@pslib.cloud 05.03.2020 13:00 - 06.03.2020 13:00	Navraceno Detail Zrušit výpůjčku
Objektiv SIGMA 24-70 mm 1:2.8 (Objektiv SIGMA 24-70 mm 1:2.8 (1)) Objektiv Canon 50 mm F1.8 (Objektiv Canon 50 mm_1) Rekordér Zoom H1 (Rekordér Zoom H1_a) Mikrofon klopový (Mikrofon klopový_1) Petr Horák, u17.406@pslib.cloud 20.10.2020 08:00 - 03.11.2020 16:00	Navraceno Detail Zrušit výpůjčku

Obrázek 13 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 nestálé výšce sekcí s informacemi o jednotlivých výpůjčkách ještě nižší.

▶ Petr Horák u17.406@pslib.cloud 09.01.2020 - 09.01.2020	Vypůjčeno Detail Zrušit výpůjčku
▶ Petr Horák u17.406@pslib.cloud 10.01.2020 - 12.01.2020	Vypůjčeno Detail Zrušit výpůjčku

Obrázek 14 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.

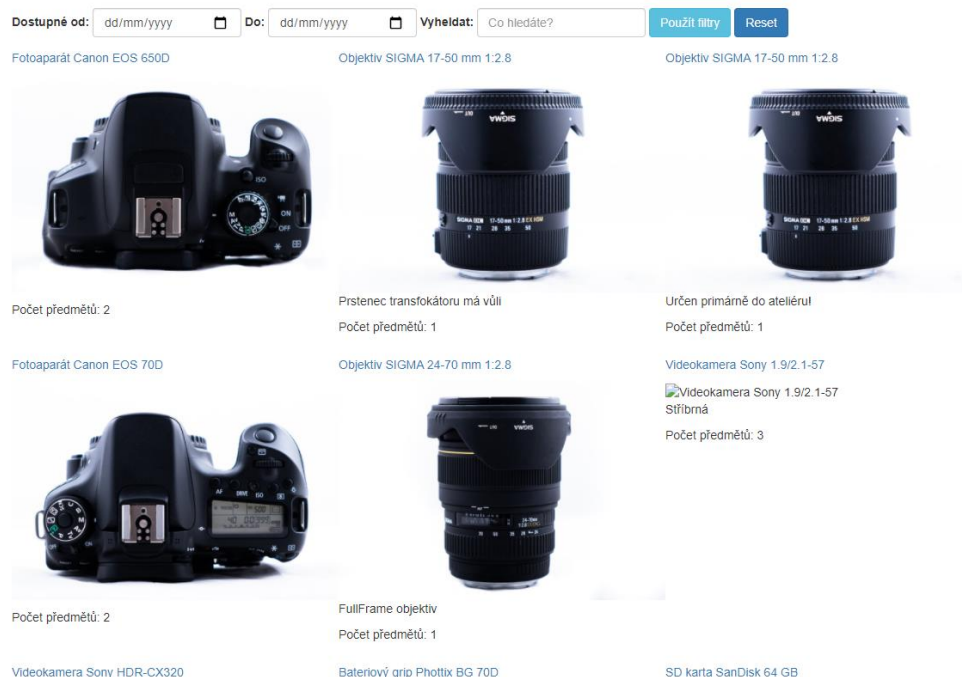
▼	Petr Horák	u17.406@pslib.cloud	09.01.2020 - 09.01.2020	Vypůjčeno	Detail	Zrušit výpůjčku
<div>Ateliér B210 (Ateliér B210)</div>				<div>Od: 09.01.2020 15:00</div> <div>Do: 09.01.2020 16:00</div>		
				Žádné poznámky		
▼	Petr Horák	u17.406@pslib.cloud	10.01.2020 - 12.01.2020	Vypůjčeno	Detail	Zrušit výpůjčku
<div>Objektiv SIGMA 17-50 mm 1:2.8 (17-50mm2.8_1)</div> <div>Fotoaparát Canon EOS 70D (Fotoaparát Canon EOS 70D (2))</div>				<div>Od: 10.01.2020 12:00</div> <div>Do: 12.01.2020 08:00</div>		
				Žádné poznámky		

Obrázek 15 Upravená verze náhledu výpůjček s podrobnými informacemi

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.

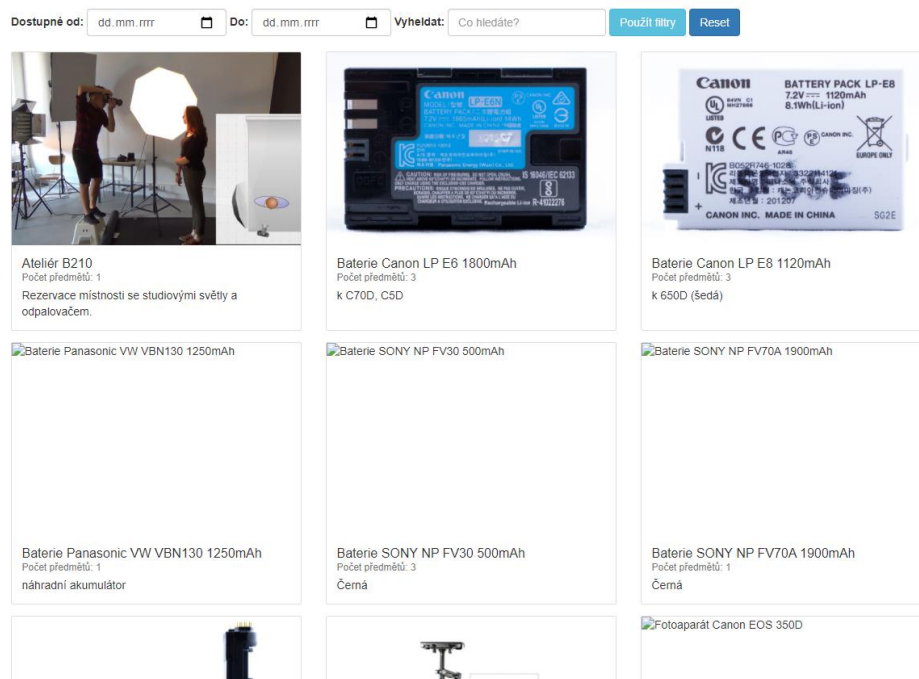
5.4.2 Zobrazení přehledu předmětů

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 16 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 17 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.

5.4.3 Funkcionalita tabulky zákazníků

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

Využito bylo JavaScript knihovny *SimpleDatatables*. 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.

5.4.4 Detail předmětu v administračním prostředí

Na stránku s detailem typu předmětu v administračním prostředí sloužící ke správě předmětu byla přidána historie všech výpůjček obsahujících některý z konkrétních

předmětů tohoto typu. Využívá se tudíž partialview *_Rentings*, jehož úpravy jsou popsány v první podkapitole této kapitoly.

```

79 public Renting[] GetRentingsForItems(IEnumerable<int> items)
80 {
81     var query = this.Context.Rentings
82         .Where(r => r.RentingToItems.Select(rti => rti.ItemId).Any(i => items.Contains(i)) && r.State != 0)
83         .OrderByDescending(r => r.EndsAt);
84
85     return query.ToArray();
86 }

```

Obrázek 18 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 přibyl 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*.

Neboť výpis historie všech výpůjček některých předmětů obsahuje i několik desítek záznamů, bylo třeba zajistit paginaci tohoto seznamu. K tomu je využita JavaScript funkce, jejíž definice je v souboru *pagination.js*. Uvnitř této funkce je definována funkce knihovny jQuery *pagify*, díky níž lze paginovat libovolné HTML prvky.

březen 2021

po	út	st	čt	pá	so	ne
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
8 Horák Petr	8 Horák Petr			8 Celeda Martin		
22	23	24	25	26	27	28
8 Horák Petr						
29	30	31	1	2	3	4
5	6	7	8	9	10	11

Obrázek 19 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 myši 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*. Pro získávání dat výpůjček zobrazovaných v kalendáři bylo třeba vytvořit novou metodu *GetCalendarEvents* v kontroleru *CalendarController*. Tato metoda vrací všechny nezrušené výpůjčky pro daný typ předmětu ve formátu *Json*. Pro určení struktury vracených dat se používá nově vytvořený *CalendarEventViewModel*, díky kterému lze předat i informace o barvě události v kalendáři aj.

5.4.5 Kalendář výpůjček

Kalendář v administračním prostředí aplikace zobrazující všechny nezrušené výpůjčky všech předmětů se ukázal jako nevyhovující. Hlavní problém byl se zobrazením kalendáře na mobilních zařízeních, kde bylo téměř nemožné kalendář ovládat. Cílem této úpravy bylo zpřehlednit kalendář použitím měsíčního zobrazení, namísto zobrazení na časové ose.

Kalendář					Nová výpůjčka	
15.03.2021 - 22.03.2021						
► Ateliér B210						
► Baterie Canon LP E8 1800mAh						
► Baterie Canon LP E8 1120mAh						
► Baterie Panasonic VW VBN130 1250mAh						
► Baterie SONY NP FV30 500mAh						
► Baterie SONY NP FV70A 1900mAh						
► Bateriový grip Phottix BG 70D						
► Bezdrátové mikrofony CVM-WM100						
► Bezdrátový mikrofon WM100 plus HTX						
► Flycam HD-3000						
► Fotoaparát Canon EOS 350D						
► Fotoaparát Canon EOS 650D						
► Fotoaparát Canon EOS 70D						
► Fotoaparát Canon EOS 90D						
► Fotoaparát Canon EOS RP						
► Gimbal Moza Air 2						
► Gimbal Snoppa Atom						
► Mikrofon klopový						
► Nabíječka baterií Canon LC E8E						
► Nabíječka baterií Canon LC E8E						
► Nabíječka baterií FK technics BC 450						
► Objektiv Canon 50 mm F1.8						
► Objektiv Canon ULTRASONIC 70-200 mm F4						
► Objektiv SIGMA 17-50 mm 1:2.8						
► Objektiv SIGMA 18-200 mm 1:3.5-6.3						
► Objektiv SIGMA 24-70 mm 1:2.8						
► Objektiv Sigma 30mmF1.4						
► Rekordér Zoom H1						
► SD karta Kingston SD10G3 32 GB						
► SD karta Lexar 64GB						
► SD karta SanDisk 64 GB						
► Stativ Hama Star 82						
► Stativ Rig Spider FT-10						
► Stativ Sony VCT-D680RM						
► Stativ Velbon C-800						
► Stativ video MS-007H						
► Světlo Viltrox VL-D640T						
► Videokamera Panasonic HC-X920						
► Videokamera Sony 1.9/2.1-57						
► Videokamera Sony HDR-CX320						
► Zhiyun Crane 3 LAB						
► stativ MiniTripod plochý						
► stativ Rig Hawk 35						
► stativ pro světla						
	st 17 březen 2021	čt 18	pá 19	so 20		

Obrázek 20 Původní kalendář výpůjček – klasické zobrazení



Obrázek 21 Původní kalendář výpůjček – zobrazení na mobilním zařízení

Původní kalendář zobrazoval výpůjčky po zvolení typu předmětu na časové ose v časovém rozmezí zvoleném v poli nad kalendářem. Zobrazení kalendáře nebylo responzivní, což znamenalo, že na zařízeních s menší úhlopříčkou obrazovky bylo téměř nemožné zobrazit časovou osu s výpůjčkami (viz Obrázek 21).

Pro generování původního kalendáře byla využívána knihovna *Vis.js*, která slouží k vizualizaci dat nejenom na časové ose. Data je pomocí této JavaScript knihovny možné zobrazovat i v různých grafech (5). Knihovna *Vis.js* ale neobsahuje možnost zobrazení měsíčního kalendáře, bylo proto nutné využít knihovny *Fullcalendar*.

Kalendář Nová výpůjčka

Vyberte předměty, pro které chcete zobrazit jejich výpůjčky:

<input checked="" type="checkbox"/> Ateliér B210	<input checked="" type="checkbox"/> Baterie Canon LP E6 1800mAh	<input checked="" type="checkbox"/> Baterie Canon LP E8 1120mAh
<input checked="" type="checkbox"/> Baterie Panasonic VW VBN130 1250mAh	<input checked="" type="checkbox"/> Baterie SONY NP FV30 500mAh	<input checked="" type="checkbox"/> Baterie SONY NP FV70A 1900mAh
<input checked="" type="checkbox"/> Bateriový grip Phottix BG 70D	<input checked="" type="checkbox"/> Flycam HD-3000	<input checked="" type="checkbox"/> Fotoaparát Canon EOS 350D
<input checked="" type="checkbox"/> Fotoaparát Canon EOS 650D	<input checked="" type="checkbox"/> Fotoaparát Canon EOS 70D	<input checked="" type="checkbox"/> Kamera sony
<input checked="" type="checkbox"/> Mikrofon klopový	<input checked="" type="checkbox"/> Nabíječka baterií Canon LC E6E	<input checked="" type="checkbox"/> Nabíječka baterií Canon LC E8E
<input checked="" type="checkbox"/> Nabíječka baterií FK technics BC 450	<input checked="" type="checkbox"/> Objektiv	<input checked="" type="checkbox"/> Objektiv Canon 50 mm F1.8
<input checked="" type="checkbox"/> Objektiv Canon ULTRASONIC 70-200 mm F4	<input checked="" type="checkbox"/> Objektiv SIGMA 17-50 mm 1:2.8	<input checked="" type="checkbox"/> Objektiv SIGMA 18-200 mm 1:3.5-6.3
<input checked="" type="checkbox"/> Objektiv SIGMA 24-70 mm 1:2.8	<input checked="" type="checkbox"/> Objektiv Sigma 30mm/F1.4	<input checked="" type="checkbox"/> Rekordér Zoom H1
<input checked="" type="checkbox"/> SD karta Kingston SD10G3 32 GB	<input checked="" type="checkbox"/> SD karta Lexar 64GB	<input checked="" type="checkbox"/> SD karta SanDisk 64 GB
<input checked="" type="checkbox"/> Stativ Hama Star 62	<input checked="" type="checkbox"/> stativ MiniTripod plochý	<input checked="" type="checkbox"/> Stativ Rig Spider FT-10
<input checked="" type="checkbox"/> Stativ Sony VCT-D680RM	<input checked="" type="checkbox"/> Stativ Velbon C-600	<input checked="" type="checkbox"/> Stativ video MS-007H
<input checked="" type="checkbox"/> Videokamera Panasonic HC-X920	<input checked="" type="checkbox"/> Videokamera Sony 1.9/2.1-57	<input checked="" type="checkbox"/> Videokamera Sony HDR-CX320

březen 2021 Nyní < >

po	út	st	čt	pá	so	ne
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

Detail výpůjčky (kliknutí na 18. března):

Čeleda Martin

Baterie Canon LP E6 1800mAh_2 ,
Flycam HD-3000_1 , Objektiv Sigma
30mm/F1.4_1

Obrázek 22 Nový kalendář výpůjček

Využití knihovny *Fullcalendar* znamenalo vyřešit způsob volby předmětů, pro které mají být v kalendáři zobrazovány výpůjčky. Jako nejvhodnější se ukázalo využití HTML prvku *input* typu *checkbox* pro potvrzení viditelnosti výpůjček každého z typů předmětů v databázi. Výchozí stav kalendáře je vidět na obrázku 22, kdy jsou v kalendáři zobrazovány výpůjčky pro všechny předměty v databázi. Při kliknutí na výpůjčku zobrazenou v kalendáři je uživatel přesměrován na detailní stránku dané výpůjčky. Při najetí myši nad výpůjčku v kalendáři je zobrazen stručný seznam názvů všech konkrétních předmětů vypůjčených v rámci této výpůjčky. Pokud není v seznamu předmětů výše na stránce označen ani jeden z předmětů ve výpůjčce, tak je výpůjčka skryta. Barvy výpůjčky v kalendáři znázorňují její stav.

```

51  jq = jQuery.noConflict(false);
52
53  jq(document).ready(function () {
54      var searchIDs = jq("#itemChooser input:checkbox:checked").map(function () {
55          return parseInt(jq(this).val());
56      }).get();
57
58      jq("#itemChooser :checkbox").change(function () {
59          console.log(searchIDs);
60          searchIDs = jq("#itemChooser input:checkbox:checked").map(function () {
61              return parseInt(jq(this).val());
62          }).get();
63          jq("#calendar").fullCalendar('refetchEvents');
64      });
65
66      jq("#calendar").fullCalendar({
67          defaultView: "month",
68          height: "auto",
69          eventOverlap: false,
70          locale: "cs",
71          slotDuration: "00:30:00",
72          allDaySlot: false,
73          eventRender: function (eventObj, $el) {
74              $el.popover({
75                  title: eventObj.title,
76                  content: eventObj.description,
77                  trigger: 'hover',
78                  placement: 'top',
79                  container: 'body'
80              });
81          },
82          events: function (start, end, timezone, callback) {
83              console.log(searchIDs);
84              jq.ajax({
85                  url: '@Url.Action("GetCalendarEventsForSelectedItemTypes", "Calendar")',
86                  method: 'get',
87                  traditional: true,
88                  data: {
89                      itemtypes: searchIDs,
90                      from: start.toISOString(),
91                      to: end.toISOString()
92                  },
93                  success: function (data) {
94                      callback(data);
95                  },
96                  error: function () {
97                      alert("Kalendářová chyba");
98                  }
99              });
100          }
101      });
102  });

```

Obrázek 23 Kód obstarávající chod kalendáře výpůjček

Při tvorbě nového kalendáře s možností výběru zobrazovaných výpůjček bylo podstatné využití knihovny jQuery, díky které se jednoduše získávají reference HTML prvků, se kterými lze dále pracovat. Při načtení stránky a následně při každé změně ve výběru předmětů se aktualizuje pole *searchIDs*. Toto pole obsahuje identifikátory všech označených typů předmětů datového typu *integer*, pro které mají být zobrazeny výpůjčky. Toto pole se předává metodě *GetCalendarEventsForSelectedItemTypes*, která je asynchronně volána pomocí knihovny *AJAX*. Zároveň se také metodě předávají údaje o zobrazovaném časovém rozmezí. Kalendář se aktualizuje při změně ve výběru

předmětů, což zajišťuje kód na řádku 63 obrázku 23, nebo při změně časového rozmezí kalendáře.

```

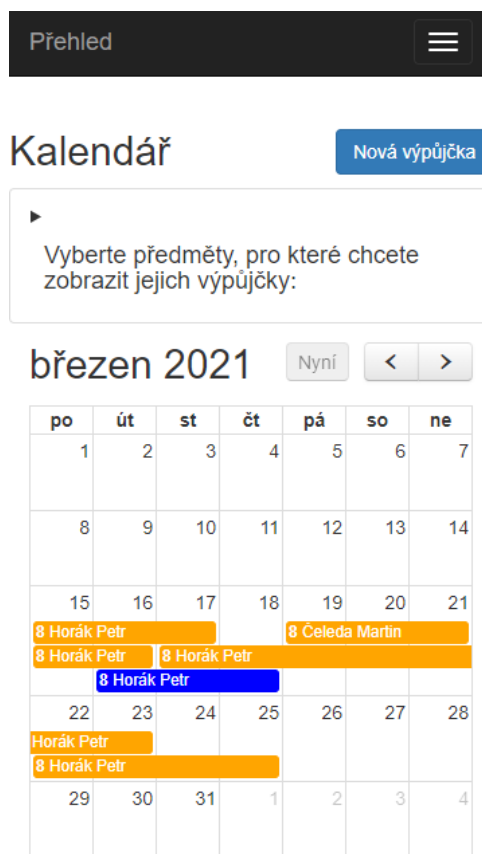
179 public JsonResult GetCalendarEventsForSelectedItemTypes(IEnumerable<int> itemtypes, DateTime from, DateTime to)
180 {
181     if (itemtypes.Count() == 0)
182     {
183         return Json(new int[0]);
184     }
185     var types = this.RepositoriesFactory.Types.GetItemTypes().Where(t => itemtypes.Contains(t.Id)).ToList();
186     var results = new List<CalendarEventViewModel>();
187
188     foreach (var itemType in types)
189     {
190         var rentings = this.RepositoriesFactory.Rentings
191             .GetRentingsInTimeForItems(itemType.NonSpecificItems.Select(i => i.Id), from, to);
192
193         if (rentings.Count() != 0)
194         {
195             foreach (var referenceRenting in rentings)
196             {
197                 results.Add(new CalendarEventViewModel(
198                     referenceRenting.User.Name,
199                     String.Join(" ", referenceRenting.Items.Select(p => p.UniqueIdentifier).ToArray()),
200                     referenceRenting.StartsAt,
201                     referenceRenting.EndsAt,
202                     Url.Action("Detail", "Renting", new { id = referenceRenting.Id, Area = "Admin" }),
203                     DecideColor(referenceRenting.State, referenceRenting.EndsAt)
204                 ));
205             }
206         }
207     }
208     return Json(results.Distinct(new CalendarEventCompare()));
209 }

```

Obrázek 24 Nová metoda *GetCalendarEventsForSelectedTypes*

Metoda *GetCalendarEventsForSelectedItemTypes* najde v databázi všechny nezrušené výpůjčky pro typy předmětů, jejichž identifikátor je v jejím parametru *itemtypes*, v daném časovém rozmezí. Následně data každé výpůjčky přidá do listu datových typů *CalendarEventViewModel*.

Důležitá část metody se nachází na jejím konci (viz Obrázek 24, řádek 208), kdy se na list výpůjček použije funkce *Distinct*. Tato funkce odstraňuje duplicitní objekty v kolekci na základě daných pravidel. Tato pravidla se v tomto případě nacházejí ve třídě *CalendarEventCompare*. Použití funkce *Distinct* bylo třeba, aby se zamezilo zobrazení jedné a té samé výpůjčky pro každý její předmět.



Obrázek 25 Nový kalendář výpůjček – zobrazení na mobilním zařízení

Důsledkem využití zobrazení formou měsíčního kalendáře se zvýšila přehlednost a čitelnost kalendáře zobrazeného na mobilních zařízeních. Seznam s výběrem předmětů k zobrazení lze zobrazit kliknutím na oblast nad kalendářem.

5.4.6 Ostatní úpravy

V této podkapitole jsou popsány ostatní drobné a méně významné úpravy, které byly provedeny.

Do lišty s navigací v administračním prostředí aplikace byla přidána možnost pro přechod do zákaznického prostředí aplikace.

Pokud nejsou u výpůjčky uvedeny žádné poznámky, je zobrazena hláška upozorňující na tento stav. Podobně je tomu u stavu výpůjčky, kde, pokud je výpůjčka vrácena, je to v náhledu výpůjčky zmíněno.

Dále byl upraven kalendář na stránce s detailem předmětu v zákaznickém prostředí, kde byl problém s jeho nekonzistentní výškou.

6 Návrh řešení nedostatků aplikace

I přes odstranění chyb a nedostatků popsaných v kapitolách výše, lze v aplikaci narazit na některá nevyhovující řešení. Většina z nich je však zapříčiněna vlastnostmi použité technologie *ASP*.

Příkladem takového nedostatku je stránka v administračním prostředí aplikace sloužící pro vytváření výpůjček. Funkcionalita formuláře pro zadání údajů o výpůjčce je zde řešena složitou kombinací kódu v jazyce JavaScript a voláním *C#* metod.

Vhodnějším řešením by mohlo být využití JavaScript knihovny *React* pro frontend aplikace společně s API tvořícím backend aplikace.

Proto vznikla nová aplikace využívající technologie knihovny *React*, která obsahuje výše zmíněný formulář pro vytváření nových výpůjček. Jako backend aplikace slouží samotná aplikace *Rentals*. Tato aplikace vznikla v rámci návrhů řešení nedostatků původní aplikace za účelem porovnání výhod a nevýhod obou technologií.

Závěr

Výsledkem této práce je nová verze aplikace pro evidenci výpůjček, kde nic nebrání uživatelům v jejím používání. Aplikace je díky úpravám přehlednější a méně problémová. Nové pohledy na data v administračním prostředí usnadňují administraci aplikace a výpůjček v ní evidovaných.

Budoucí úpravy aplikace by se mohly zabývat podrobnějšími úpravami vzhledu jednotlivých částí aplikace, nebo další optimalizací vzhledu aplikace na mobilních zařízeních.

Značnou část času stráveného s revizí aplikace jsem věnoval studiu a porozumění původním zdrojovým kódům. Dále bylo třeba doplnit si znalosti programovacího jazyka JavaScript a některých jeho knihoven, jako například *AJAX* nebo *jQuery*.

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
Obrázek 2 Chybová hláška programu Microsoft SQL Server Management Studio	4
Obrázek 3 Původní verze metody GetClassFromMicrosoft	8
Obrázek 4 Formulář pro vytvoření výpůjčky	9
Obrázek 5 Původní verze funkce addItem	9
Obrázek 6 Původní verze metod TypeDetail a ItemDetail	10
Obrázek 7 Opravená metoda GetClassFromMicrosoft	12
Obrázek 8 Změny ve volání metody GetClassFromMicrosoft	12
Obrázek 9 Opravená funkce addItem	13
Obrázek 10 Upravené metody TypeDetail a ItemDetail	14
Obrázek 11 Změny ve volání metod TypeDetail a ItemDetail v komponentu ItemsOverview	14
Obrázek 12 Úpravy třídy ItemOverViewModel	15
Obrázek 13 Původní verze zobrazení náhledů výpůjček	16
Obrázek 14 Upravená verze náhledu výpůjček se základními informacemi	16
Obrázek 15 Upravená verze náhledu výpůjček s podrobnými informacemi	17
Obrázek 16 Původní zobrazení přehledu předmětů k vypůjčení	17
Obrázek 17 Upravené zobrazení přehledu předmětů k vypůjčení	18
Obrázek 18 Nová metoda GetRentingsForItems	19
Obrázek 19 Nový kalendář na stránce /Admin/Items/Detail/	19
Obrázek 20 Původní kalendář výpůjček – klasické zobrazení	21
Obrázek 21 Původní kalendář výpůjček – zobrazení na mobilním zařízení	22
Obrázek 22 Nový kalendář výpůjček	23
Obrázek 23 Kód obstarávající chod kalendáře výpůjček	24
Obrázek 24 Nová metoda GetCalendarEventsForSelectedTypes	25
Obrázek 25 Nový kalendář výpůjček – zobrazení na mobilním zařízení	26

Použitá literatura

1. Bak file. *Wikipedia*. [Online] 18. Únor 2021. https://en.wikipedia.org/wiki/Bak_file.
2. C Sharp. *Wikipedia*. [Online] 13. Duben 2020. https://cs.wikipedia.org/wiki/C_Sharp.
3. Uniform Resource Identifier (URI): Generic Syntax. *IETF*. [Online] Network Working Group, Leden 2005. <https://www.ietf.org/rfc/rfc3986.txt>.
4. Basic HTML data types. *W3C*. [Online] W3C. <https://www.w3.org/TR/html4/types.html#type-id>.
5. vis.js community edition. *vis.js*. [Online] 19. Únor 2019. <https://visjs.org/>.
6. učitelé SPŠSE. Úvod. *SPŠSE a VOŠ Liberec*. [Online] 01. 09 2016. [Citace: 01. 09 2016.] <https://www.pslib.cz>.
7. nuget. [Online] Microsoft. <https://www.nuget.org/>.
8. 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>.
9. Enumeration types. *Microsoft Docs*. [Online] Microsoft, 13. Prosinec 2019. <https://docs.microsoft.com/cs-cz/dotnet/csharp/language-reference/builtin-types/enum>.
10. Documentation. *FullCalendar*. [Online] FullCalendar LLC, 2021. <https://fullcalendar.io/docs/v3#toc>.

A. Seznam příložených souborů

Obsah příloženého CD:

- Zdrojové kódy upravené aplikace Rentals
- Dokumentace práce