STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

REST API APLIKACE RENTALS
REST API APPLICATION RENTALS

Autoři: Kateřina Daňková
Škola: Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3, příspěvková organizace; Masarykova 3, 460 01 Liberec 1
Kraj: Liberecký kraj
Konzultant: Ing. Tomáš KAZDA, DiS.

Liberec 2022

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 15. 3. 2022 ……

Kateřina Daňková

Anotace

Má práce vychází ze původního backendu aplikace Rentals pro správu výpůjček ze školního ateliéru SPŠSE a VOŠ Liberec. Práce se zabývá vytvořením nové databáze, REST API, a povýšením na framework .NET 6, který vyšel v průběhu vývoje.

Klíčová slova

REST API;databáze;backend

Annotation

My work is based on the original backend of the Rentals application for the management of rentals from the school studio of the SPŠSE Liberec. The thesis deals with the creation of a new database, REST API, and promotion to the .NET 6 framework that came out during development.

Keywords

REST API;database;backend

Obsah

Úvod

Mým cílem bylo vytvořit přehlednou a pro práci jednoduchou databázi výpůjček, předmětů a všeho co k tomu patří. K tomu jsem použila Entity Framework Core pro relační databáze. Vytvořit API odpovídající navrženému wireframu za pomocí ASP.NET Web API, které je konkrétně REST API.

Tuto práci jsem si vybrala pravděpodobně proto, že jsem si během druhé poloviny ročníku, kdy jsme byli nuceni zůstat doma, musela zvyknout na to se učit sama. Mnohdy jsme se při dlouhodobějších projektech z webů nebo programování museli sami vypořádat s řešením. V tu dobu se změnil můj pohled na programování, a od té doby se nebojím věcí, které vidím poprvé. Začali jsme probírat ER modely a framework .NET, a možná právě proto, že to byla první věc, co jsem s tímto přístupem zvládla, jsem se rozhodla databázím a tvorbě API více věnovat. Tudíž jsem si chtěla vyzkoušet tvorbu databáze a API v praxi a získat tak více zkušeností.

1 Analýza navrženého prostředí

Za použití wireframu navrženého jako MP v repozitáři pslib-cz/MP2021-JulieSanetrnikova-Redesign-UI-UX-aplikace-Rentals a dostupné databáze Rentals jsem analyzovala, jak by měla vypadat struktura databáze a endpointy v REST API.

Rentals aplikace byla navržená jako e-shop, což pro výpůjčky ze školního ateliéru, kdy se jeden předmět opakuje maximálně pětkrát, je zbytečně složité řešení. Také předmět v databázi byl rozdělen do dvou tabulek, a to *Typ předmětu* (např. kamera) a poté ve druhé tabulce byla tato kamera na dvou řádcích jako *Kamera_1* a *Kamera_2.* V mé práci jsou obě kamery v tabulce předmětu každá jako vlastní záznam s téměř identickými vlastnostmi (popis se může lišit).

Databáze byla odlehčena o tabulku míst, ze kterých se dá půjčovat, jelikož jde o vypůjčovaní věcí pouze z ateliéru naší školy, a naopak se nově přidaly kategorie předmětů, podle kterých se dá v aplikaci lépe vyhledávat. Také byly místa, kde databáze nesplňovala ani první normální formu – v tabulce uživatele, byl jeho košík zapsán jako textový atribut, kde byly čárkami odděleny předměty, které obsahuje.

Basket
{"Fotoaparát Canon EOS 650D":1}
{}
{}
{}
{}
{"Fotoaparát Canon EOS 70D (1)":-1,"17-50mm2.8_2":-1,"Baterie Canon LP E6 1800mAh":1,"Nabíječka baterií Canon LC E6E":1}
{}
{"Ateliér B210":-1}

Obrázek 1 Ukázka porušení 1. normální formy (1NF)

1.1 Změny

Nově se pro správu uživatelů používá školní autorizační server *oauth.pslib.cloud*, díky čemuž se v databázi o uživateli vyskytují pouze potřebné a ověřené informace, které se získávají z oauth.pslib.cloud pomocí Bearer tokenu. Došlo také k přechodu .NET 6 a přidání knihovny tus pro nahrávání obrázků.

1.2 Knihovna tus.io

Tato knihovna se stará o nahrávání obrázků tak, že ho rozdělí na několik menších částí a ty postupně nahrává. Bez použití knihovny se mohlo stát, že když uživatel nahrál příliš velký soubor na méně výkonném zařízení, tak celá aplikace zamrzla a muselo se počkat několik sekund, než se obrázek kompletně nahraje. Další výhodou této knihovny je, že se dá nahrávání pozastavit. Zároveň je jednoduše nakonfigurovatelná jak na backendu a frontendu. Na frontendu nám také během nahrávání umožňuje ukazovat progres.

```
app.UseTus(httpContext => new DefaultTusConfiguration
{
    Store = new TusDiskStore(@"C:\Users\katul\source\repos\TUS_Demo_Own\TUS_Demo_Own\Files\"),
    UrlPath = "/files",
    Events = new Events
    {
        OnFileCompleteAsync = async eventContext =>
        {
            ITusFile file = await eventContext.GetFileAsync();
            Dictionary<string, Metadata> metadata = await file.GetMetadataAsync(eventContext.CancellationToken);
            Stream content = await file.GetContentAsync(eventContext.CancellationToken);
        }
    }
});
```

Obrázek 2 Základní konfigurace knihovny tus v .NET 6

2 Databáze

> Databáze je systém souborů s pevnou strukturou záznamů. Tyto soubory jsou mezi sebou navzájem propojeny pomocí klíčů. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. (1)

Existuje několik druhů databází, v tomto případě jsem použila relační databázi hlavně z důvodu přehlednosti a také zkušeností ze školy.

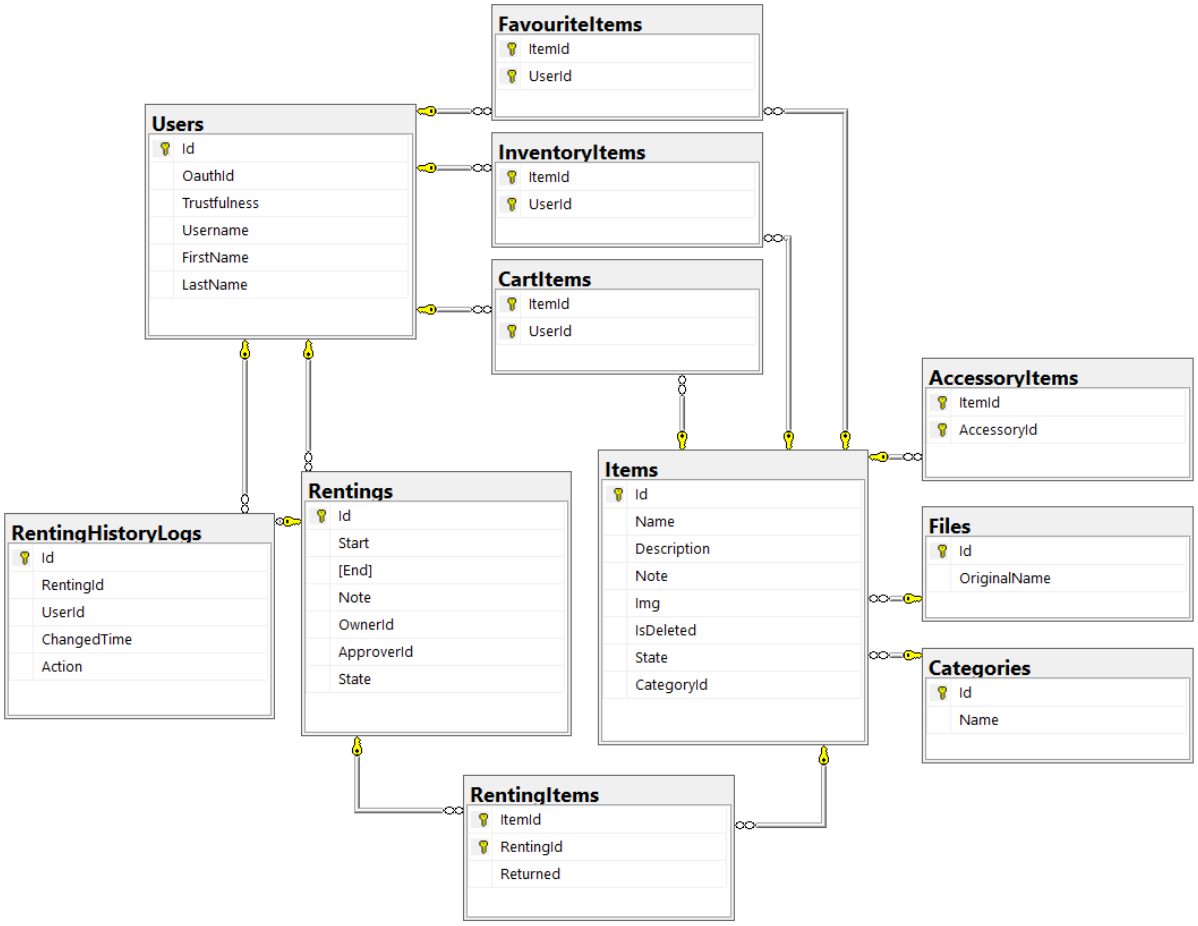
2.1 Relační databáze

Relační databáze se skládá z tabulek, které se obsahují záhlaví a tělo s daty. Sloupce v tabulkách se nazývají atributy, každý atribut má svůj datový typ (int, string, enum, float, …). Záznamy, nebo také řádky se nazývají entity.

	Id	Name	Index
	3e28741ca0c74...	Samoyed-MP.jpg	2
	7e693d3f513040...	cat-1101064_19...	1
	dc9b98b63ed94...	rabbit.jpg	3

Obrázek 3 Tabulka nahraných souborů

2.2 Návrh databáze



Obrázek 4 E-R model nové databáze

2.2.1 Items

Tabulka předmětů obsahuje název předmětu, popis, poznámku (např. zda byl poškozen, nebo něco chybí), cizí klíč ID nahraného obrázku. Obsahuje také cizí klíč ID kategorie, do které patří, každý předmět spadá pouze do jedné kategorie. Dále má výčtový typ stav, který může být buď dostupný, vypůjčený, nebo nedostupný. Jako poslední má předmět vlastnost, zda je smazán. Z bezpečnostních důvodů se data v aplikaci skutečně nemažou a pouze skrývají, aby nedošlo k jejich ztrátě.

2.2.1.1 AccesoryItems

Jedná se o vazbu mezi příslušenstvím a předmětem. Jelikož je příslušenství předmět, vzniká zde vazba M:N mezi předmětem a předmětem, protože předmět může být příslušenstvím pro mnoho předmětu a zároveň předmět může mít mnoho

příslušenství. Tato vazba by se od verze .NET 5 měla vytvářet automaticky, ale pro přehlednost je vždy vytvářím.

2.2.2 Categories

Definuje jednoduchou tabulku pro kategorii, která obsahuje jen své jméno.

2.2.3 Files

Tabulka pro nahrané obrázky předmětů. Ukládá zatím pouze jejich původní název. Jejich ID je název fyzického souboru.

2.2.4 Rentings

Výpůjčka se skládá z datumů, kdy by měla začínat a končit a dodatečné poznámky. Má vlastníka a osobu, která výpůjčku schválí. Výčtový typ *Stav*, který může být *začne, probíhá, ukončená, zrušená*.

2.2.4.1 RentingItems

Vazba mezi výpůjčkou a předmětem navíc obsahuje, jestli byl už v dané výpůjčce předmět navrácen.

2.2.4.2 RentingHistoryLog

Tabulka pro historii úprav předmětu, kam se při každé změně výpůjčky zaznamená, kdo provedl změnu, na jaké výpůjčce, o jakou akci se jednalo a kdy ke změně došlo.

2.2.5 Users

Tabulka uživatelů vychází z dat oauth.pslib.cz, proto obsahuje jen informace jako je ID z Oauthu, křestní jméno, příjmení a uživatelské jméno, kterým je e-mail. Veškeré ostatní informace se dají získat z tokenu na frontendu aplikace.

2.2.5.1 FavouriteItems, CartItems, InventoryItems

Tyto tři vázací tabulky mezi uživatelem a předmětem, které slouží pro oblíbené předměty, košík a inventář uživatele, jsou naprosto stejné. Každý předmět tu má vlastní

záznam, kdežto v původní databázi byl například inventář jako textová vlastnost uživatele, což nesplňuje 1. normalizační normu databáze.

2.3 Data

Do databáze bylo potřeba předdefinovat data předmětů a jejich příslušenství a kategorií. Pro přepsání předmětů jsem musela vycházet z původní tabulky ItemTypes kde bylo jméno, popis a zda je předmět smazán. Z této tabulky jsem musela vzít ID předmětu do další tabulky, kde jsem zjistila, který obrázek se k nim váže a spočítala si, kolikrát se má předmět v databázi nacházet, a získala ke každému konkrétnímu kusu poznámku.

S příslušenstvím to bylo o něco jednodušší, tam už mi stačilo pouze ID z jedné tabulky, ale i přesto se orientovat mezi ID předmětu, ID příslušenství a ID mého předmětu a ID mého příslušenství, bylo celkem obtížné na udržení pozornosti.

Kategorie zde byly přidány nově, tudíž jsem pouze od vedoucího práce zjistila, jaké kategorie by se zde měly vyskytovat, a podle názvu předmětu jsem je rozřadila. Prvotní myšlenka byla, že předmět může mít více kategorií, a tak byla mezi nimi vazba M:N. Když jsem poprvé přiřazovala předměty ke kategoriím, sledovala jsem tedy ID předmětu a ID kategorie, kam patří, podobně jako u příslušenství, jen méně obtížněji. Až díky testování na frontendu jsem si uvědomila, že kategorie jsou už udělané jako 1:N, ale toto přepsání mi zabralo tak 5 minut.

Z důvodu zmatených tabulek u předmětu jsem se rozhodla všechny záznamy psát kompletně ručně, a i když jsem si mohla alespoň trochu ulehčit práci tím, že ponechám původní ID předmětů, přišlo mi lepší, když ID půjdou popořadě od 1 do 100.

2.3.1 Kategorie

- přístroje (to jako fotoaparáty a videokamery)
- objektivy
- stativy (stativ, gimbal, flycam, rigy...)
- příslušenství (sd karty, akumulátory, nabíječky brašny)
- audiotechnika (mikrofon, rekordér)
- ostatní (ateliér a co se "nevejde" jinam)

3 REST API

RESTful API je architektonický styl, který používá http požadavky k přístupu a práci s daty. API je kód, který umožňuje dvěma softwarovým programům vzájemně komunikovat, zároveň určuje správný způsob, jakým napsat program požadující služby od operačního systému nebo jiné aplikace. (2)

REST je, na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům.

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim s názvem CRUD. (3)

3.1 CRUD

Obrázek 5 CRUD (4)

3.1.1 Create – POST

Metoda POST se používá, když chceme odeslat na server nějaká data, například data z formuláře, nejčastěji pro vytvoření nového záznamu do databáze. Metodě můžeme přiřadit parametry dvěma různými způsoby:

– v URL požadavku, může jich být několik za sebou (v našem případě pouze ID předmětu 14 pro přidání do uživatelova košíku)

```
https://example.pslib.cloud/api/User/Cart/14
```

– v těle požadavku zapsán v JSON formátu (Vytvoření nového uživatele s parametry jméno a příjmení, pomocí axiosu v Reactu)

```
axios({
    method: 'post',
    url: "api/User",
    data: {
        Name: "Karel Novák",
        Email: "karel.novak@pslib.cz"
    }
})
```

Metoda nám může vracet například námi vytvořená data (uživatele, jeho košík apod.), cokoli jiného (vyhledávání, kdy hledáme zadaný termín, a metoda nám vrátí data, která termín obsahují => žádná data se nevytváří) anebo prázdnou odpověď.

3.1.2 Read – GET

Metoda GET se používá k vyžádání informací ze serveru, požadavky by tedy neměly mít žádný vliv na data a díky tomu se považuje za bezpečnou. Tato metoda přijímá pouze parametry v URL, nemá totiž žádné tělo. S její pomocí můžeme získat data z databáze, nebo například soubory.

3.1.3 Update – PUT

Metoda PUT se používá k aktualizaci či úpravě všech vlastností záznamu, tudíž se všechny vlastnosti přepíší daty od uživatele, a to i v případě, kdy původně hodnota již byla zapsána a uživatel vlastní hodnotu nezadá, tato vlastnost se přepíše na prázdnou (null). Parametry i návratovými hodnotami se shoduje s metodou POST, parametry ale musí obsahovat identifikátor záznamu, který chceme aktualizovat.

3.1.4 Delete – DELETE

Metoda DELETE se používá pro smazání záznamů z databáze, nebo odstranění souboru na serveru. Stejně jako u metody GET nemůže mít požadavek této metody tělo, a tak se parametr záznamu, který chceme smazat píše do URL požadavku. Z principu metoda DELETE by neměla nic vracet, jelikož už není co.

3.2 Další metody

3.2.1 PATCH API pro úpravu

V projektu jsem použila PATCH API pro úpravu záznamů. Jedná se o rozšiřující metodu PATCH, jako je například GET, POST atd. Je to metoda podobná PUT, ten ale mění všechny vlastnosti objektu a v případě kdy bychom chtěli změnit například jméno u osoby, která má jméno a příjmení, příjmení se nastaví na null. Díky metodě PATCH lze upravit pouze jednu (nebo i více) vlastností beze změny ostatních.

PATCH metoda přijímá parametr navíc a to `JsonPatchDocument`. Jedná se v podstatě o kolekci vlastností, které chceme změnit. Pro každou měněnou vlastnost se udává

- *path* (hodnota se píše s lomenem – v příkladu měním popis obrázku tudíž *"/Description")*
- *op* (operace kterou chceme provést, v našem případě nahradit – *replace*)
- *value* (hodnota na kterou se má vlastnost popis přepsat)

```
[
  {
    "path": "/Description",
    "op": "replace",
    "value": "Velmi"
  },
  {
    "path": "/Note",
    "op": "replace",
    "value": "Užitečný"
  }
]
```

Obrázek 6 Příklad JsonPatchDocument

3.2.2 Ostatní

- HEAD – Metoda HEAD požaduje odpověď stejnou jako u požadavku GET, ale bez těla odpovědi
- CONNECT – Metoda CONNECT vytvoří tunel na server identifikovaný cílovým prostředkem
- OPTIONS – Metoda OPTIONS popisuje možnosti komunikace pro cílový prostředek
- TRACE – Metoda TRACE provede test zpětné smyčky zpráv podél cesty k cílovému prostředku (5)

3.3 Stavové kódy HTTP

Stavový kód je součást hlavičky odpovědi na požadavek, který určuje, jak byla odpověď zpracována. Každý kód je zapsán ve formě tříciferného čísla, kdy první určuje kategorii odpovědi a zbylá čísla ji blíže specifikují. Kódy se rozdělují do pěti kategorií podle charakteru odpovědi. (6)

3.3.1 1xx – Informační

Tyto kódy oznamují že požadavek byl úspěšně přijat.

- 100 Continue – čeká se na tělo zprávy (např. POST)
- 101 Switching protocol – server souhlasí se změnou protokolu
- 102 Processing – požadavek přijat, ale ještě se zpracovává

3.3.2 2xx – Úspěch

Tyto kódy oznamují, že požadavek byl přijat a úspěšně zpracován.

- 200 OK – nejčastější odpověď na požadavek, kdy vše proběhlo v pořádku
- 202 Created – požadavek byl zpracován a výsledkem došlo k vytvoření dat
- 204 No Content – požadavek byl úspěšně zpracován, ale nic nevrací

3.3.3 3xx – Přesměrování

Tyto kódy se používají, když má server několik možností, jak pokračovat.

- 300 Multiple Choices – dokument dostupný na více místech
- 301 Moved Permanently – všechny požadavky směrovány na dané URI
- 303 See Other – odpověď může být nalezená na jiném URI

3.3.4 4xx – Chyba klienta

Tyto kódy se použijí, když se server domnívá, že klient udělal chybu.

- 400 Bad Request – požadavek je špatně formulován
- 401 Unauthorized – k tomuto požadavku nemá uživatel práva
- 403 Forbidden – přístup odepřen
- 404 Not found – data nenalezena
- 429 Too many Request – příliš mnoho požadavku v krátkém časovém úseku

3.3.5 5xx – Chyba serveru

Tyto kódy oznamují, že na serveru došlo k chybě.

- 500 Internal Server Error – obecná chybová hláška, většinou když se požadavek na API vůbec nedostal
- 501 Not Implemented – metoda nerozpoznána nebo nepodporována

4 Dokumentace API

API se skládá ze tří kontrolérů. Kontrolér uživatele, ten se stará o založení záznamu v databázi a umožňuje uživateli například ukládat předměty do košíku, či si je přidat do oblíbených. Kontrolér předmětů obsahuje metody pro různé druhy změn na předmětu + kategorií. Nakonec kontrolér pro administraci výpůjček. Na některých místech se pro přehlednost používá jako parametr objekt, který obsahuje parametry, které potřebujeme. Pro všechny metody je potřeba aby byl uživatel přihlášen.

4.1 Kontrolér uživatele

4.1.1 Kontrola uživatele

Slouží pro zjištění, zda uživatele máme již v databázi a pokud ne, vytvoří ho. Zavolá se ihned po přihlášení do aplikace. Jako parametr přijímá objekt, který obsahuje ID z oauthu, uživatelské jméno, křestní jméno a příjmení.

4.1.2 Košík uživatele

Slouží pro vypsání předmětů v košíku právě přihlášeného uživatele. Díky atributu *[Authorize]* nejsou potřeba parametry. Vrátí list předmětů.

4.1.3 Přidání do košíku uživatele

Slouží pro přidání předmětu do košíku právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme uložit. Vrací metodu pro vypsání košíku uživatele.

4.1.4 Odebrání z košíku uživatele

Slouží pro odebrání předmětu z košíku právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme odebrat. Vrací metodu pro vypsání košíku uživatele.

4.1.5 Vypsání oblíbených předmětů uživatele

Slouží pro vypsání předmětů, které má právě přihlášený uživatel uložené v oblíbených. Nepotřebuje žádný parametr. Vrací list předmětů.

4.1.6 Přidání do oblíbených předmětů uživatele

Slouží pro přidání předmětu do oblíbených právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme přidat. Vrací metodu pro vypsání oblíbených předmětů uživatele.

4.1.7 Odebrání z košíku uživatele

Slouží pro odebrání předmětu z oblíbených právě přihlášeného uživatele. Jako parametr přijímá ID předmětu, který chceme odebrat. Vrací metodu pro vypsání oblíbených předmětů uživatele.

4.1.8 Vypsání všech uživatelů

Slouží pro vypsání všech uživatelů, například pro správu výpůjček dané osoby. Na toto má právo pouze zaměstnanec nebo administrátor. Nepřijímá žádné parametry a vrací list uživatelů.

4.1.9 Vypsání předmětů v inventáři uživatele

Slouží pro vypsání předmětů, které by měl mít uživatel u sebe. Na toto má právo pouze zaměstnanec nebo administrátor. Jako parametr přijímá ID uživatele (oauth id) a vrací list předmětů.

4.2 Kontrolér výpůjčky

4.2.1 Vytvoření nové výpůjčky

Slouží pro zadání nové výpůjčky a při tom se smaže uživatelův košík. přijímá objekt, který obsahuje kolekci ID předmětů (z košíku), datum, kdy by měla výpůjčka začít a datum kdy by měla výpůjčka skončit a poznámku. Vrací právě vytvořenou výpůjčku.

4.2.2 Vrácení předmětů výpůjčky

Slouží pro vrácení předmětů z dané výpůjčky, pokud se vrátí všechny předměty, dojde k její ukončení. V průběhu se předměty odeberou z uživatelova inventáře a jejich stav se nastaví na dostupný. Na toto má právo pouze zaměstnanec nebo administrátor. Zároveň změnu zapíše do tabulky s historií. Jako parametr přijímá objekt, který obsahuje ID výpůjčky a kolekci ID předmětů, které chceme vrátit. Vrací změněnou výpůjčku.

4.2.3 Zrušení neuskutečněné výpůjčky

Slouží pro případ, že by si někdo nevyzvedl výpůjčku, a tak bylo potřeba výpůjčku zrušit. Na toto má právo pouze zaměstnanec nebo administrátor. Tuto změnu zapíše do tabulky s historií. Jako parametr přijímá ID výpůjčky, kterou chceme zrušit. Vrací zrušenou výpůjčku.

4.2.4 Vypsání všech výpůjček

Slouží pro vypsání všech výpůjček. Na toto má právo pouze zaměstnanec nebo administrátor. Tato metoda nepřijímá žádný parametr. Vrací list výpůjček.

4.2.5 Vypsání výpůjček dle uživatele

Slouží pro vypsání všech výpůjček daného uživatele. Přijímá jako parametr oauth ID uživatele, jehož výpůjčky chceme vypsat. Na toto má právo buď vlastník výpůjček, anebo zaměstnanec / administrátor. Vrací list výpůjček.

4.2.6 Aktivace výpůjčky

Slouží pro aktivaci, respektive fyzické vyzvednutí předmětů výpůjčky, kterou si uživatel vytvořil. Na toto má právo pouze zaměstnanec nebo administrátor. Dojde k vložení předmětů do uživatelova inventáře a nastavení stavu předmětů na vypůjčený. Zároveň se tato změna uloží do tabulky s historií. Jako parametr přijímá ID výpůjčky, kterou chceme aktivovat. Vrací aktivovanou výpůjčku.

4.2.7 Vypsání dat (od kdy do kdy) všech výpůjček

Slouží pro vypsání dat všech výpůjček pro zobrazení v kalendáři. Jako parametr bude přijímat rozsah dat, od kdy do kdy chceme výpůjčky vypsat. Vrací list objektů, který obsahuje ID, stav, začátek, konec a vlastníka výpůjčky.

4.2.8 Detail výpůjčky

Slouží k vypsání všech předmětů z výpůjčky, které ještě nebyly navráceny. Používá se například pro vytvoření seznamu k navrácení předmětů. Jako parametr přijímá ID probíhající výpůjčky a vrací list nevrácených předmětů.

4.3 Kontrolér předmětu

4.3.1 Vytvoření nového předmětu

Slouží pro přidání nového předmětu do databáze. Na toto má právo pouze administrátor. Jako parametr přijímá objekt, který obsahuje název předmětu, popis, poznámku a cestu k obrázku. Vrací právě vytvořený předmět.

4.3.2 Úprava předmětu

Slouží k úpravě vlastností předmětu. I když na metodu má právo zaměstnanec nebo administrátor, zaměstnanec může měnit pouze poznámku a popis obrázku. Nelze zde měnit ID předmětu a zda je smazaný. Jako parametry přijímá ID předmětu, který chceme měnit, a objekt JsonPatchDocument. Vrací upravený předmět.

4.3.3 „Smazání“ předmětu

Slouží k nastavení vlastnosti, zda je předmět smazaný. Na toto má právo pouze administrátor. Jako parametry přijímá ID předmětu, který chceme smazat, a objekt JsonPatchDocument (protože měníme jednu vlastnost). Vrací „smazaný“ předmět.

4.3.4 Vypsání „smazaných“ předmětů

Slouží pro vypsání všech předmětů, které mají vlastnost IsDeleted nastavenou na true. Možnost použití k vytvoření seznamu pro navrácení předmětu. K tomuto má

přístup pouze zaměstnanec nebo administrátor. Nepřijímá žádné parametry a vrací list „smazaných“ předmětů.

4.3.5 Navrácení „smazaného“ předmětu

Slouží k navrácení smazaného předmětu, tím že nastaví jeho vlastnost IsDeleted na false. K tomuto má oprávnění pouze administrátor. Jako parametr přijímá pouze ID předmětu, který chceme obnovit, a daný předmět vrátí.

4.3.6 Vypsání detailu předmětu

Slouží k vypsání vlastností předmětu, pokud není „smazaný“. Jako parametr přijímá ID předmětu a vrací daný předmět.

4.3.7 Získání obrázku předmětu

Slouží k získání souboru obrázku daného předmětu. Jako parametr přijímá ID předmětu, jehož obrázek se snažíme získat a vrací soubor s obrázkem. Pokud na předmět není navázán žádný obrázek, nahradí se zástupným obrázkem Placeholder.jpg.

4.3.8 Vypsání všech předmětů

Slouží k vypsání všech předmětů, které nejsou smazané. Nepřijímá žádný parametr. Vrací list předmětů.

4.3.9 Vypsání příslušenství předmětu

Slouží k vypsání všech předmětů, které jsou příslušenstvím k danému předmětu. Použije se například u zobrazení předmětu, aby uživatel věděl, co se k danému předmětu hodí. Jako parametr přijímá ID předmětu, ke kterému chceme vypsat příslušenství a vrací list předmětů.

4.3.10 Změna příslušenství předmětu

Slouží k přiřazení předmětů jako příslušenství k danému předmětu, původní nevybrané se odeberou. Na toto má právo pouze administrátor. Jako parametr přijímá objekt, který obsahuje ID předmětu, kterému chceme příslušenství přiřadit a kolekci ID předmětů, které mají být příslušenstvím. Vrací upravený předmět.

4.3.11 Kontrola oblíbenosti

Slouží k ověření, zda má přihlášený uživatel daný předmět v oblíbených. Používá se pro vybarvení srdíčka, které označuje, že je předmět oblíbený. Jako parametr přijímá ID předmětu, na který se dotazujeme a vrací hodnotu true nebo false.

4.3.12 Vypsání dat, kdy je předmět výpůjček

Slouží k vypsání dat, kdy je daný předmět vypůjčen. Slouží pro zobrazení v kalendáři na detailu předmětu. Jako parametr přijímá ID předmětu, kterého data chceme vypsat a vrací list objektů, které obsahují ID, stav, začátek, konec a jméno vlastníka výpůjčky, ve které se předmět nachází.

4.3.13 Změna kategorií předmětu

Slouží k přiřazení kategorií k danému předmětu, původní se odeberou. Jako parametr přijímá objekt, který obsahuje ID předmětu, kterému chceme kategorie přiřadit a kolekci ID kategorií. Vrací upravený předmět.

4.3.14 Vypsání všech kategorií

Slouží pro vypsání všech kategorií, použije se například k filtrování. Nepřijímá žádný parametr a vrací list kategorií.

4.3.15 Vytvoření nové kategorie

Slouží pro vytvoření nové kategorie. Jako parametr přijímá název kategorie a vrací úspěšně vytvořenou kategorii.

4.3.16 Úprava kategorie

Slouží pro změnu jména kategorie. Jako parametr přijímá ID kategorie, kterou chceme měnit a objekt JsonPatchDocument a vrací upravenou kategorii.

4.3.17 Smazání kategorie

Slouží pro odstranění kategorie. Jako parametr přijímá ID kategorie a vrací smazanou kategorii.

5 Zabezpečení

Do aplikace Rentals mají mít přístup pouze žáci naší školy, proto využívá školní autorizační server oauth.pslib.cz, který se používá i pro ostatní školní aplikace.

```
builder.Services.AddAuthentication("Bearer").AddJwtBearer("Bearer", options =>
{
    options.Authority = builder.Configuration["Authority:Server"];
    options.RequireHttpsMetadata = true;
    options.Audience = "RentalsApi";
});
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("Administrator", policy =>
    {
        policy.RequireClaim(AuthorizationConstants.ADMIN_CLAIM, "1");
    });
    options.AddPolicy("Employee", policy =>
    {
        policy.RequireClaim(AuthorizationConstants.EMPLOYEE_CLAIM, "1");
    });
});
```

Obrázek 7 Konfigurace zabezpečení a práv

Na ukázce výše je konfigurace pomocí Bearer tokenu pro náš autorizační server. Pod ním nastavení policy – práv, díky kterým mohou mít vybraní uživatele větší nebo menší práva než jiní.

V našem případě jsou zde tři třídy práv a každá další obsahuje práva předchozí třídy:

1. Student – přihlášený uživatel se základními právy, není pro něj definovaná policy
2. Employee – pracovník, který může spravovat výpůjčky (vypůjčit, ukončit…)
3. Administrator – správce celé aplikace má přístup ke všem metodám

Ke každé z metod má přístup pouze přihlášený uživatel. To je zajištěno atributem [Authorize] na úplném začátku kontroléru.

```
[Authorize]
[Route("api/[controller]")]
[ApiController]
public class ItemController : ControllerBase
{
```

Obrázek 8 Atribut authorize pro zajištění přihlášeného uživatele

V případě, kdy chceme, aby byla pouze konkrétní metoda zaštítěna oprávněními, píšeme atribut [Authorize] nad danou metodou s parametrem policy. Takto zapsaná metoda značí, že použít ji může pouze správce:

```
/// <summary>
/// Vytvoření nového předmětu
/// </summary>
[Authorize(Policy = "Administrator")]
[HttpPost]
public async Task<ActionResult<Item>> AddNewItem([FromBody] ItemRequest request)...
```

Obrázek 9 Authorize s policy

6 Testování

První fáze testování probíhala mnou za pomocí Swaggeru.

6.1 Swagger

Swagger je open source framework pro návrh a tvorbu dokumentace pro RESTful API. Obsahuje nástroje pro automatické vygenerování dokumentace a testování existujícího API, vizualizaci a vyzkoušení navrženého API.

Společnost SmartBear Software se podílela na vývoji OpenAPI a darovala svoji specifikaci swaggeru do nově vzniklé iniciativy. OpenAPI 2.0, která je obsahově shodná s původní specifikací Swagger 2.0, je dnes možné chápat jako specifikaci a Swagger jako nástroj pro implementaci této specifikace. (7)

6.1.1 Konfigurace Swaggeru v C#

```
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "API_Rentals", Version = "v1" });

    var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
    var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
    c.IncludeXmlComments(xmlPath);

    var securityScheme = new OpenApiSecurityScheme
    {
        Name = "JWT Authentication",
        Description = "Enter JWT Bearer token **_only_**",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Scheme = "bearer", // must be lower case
        BearerFormat = "JWT",
        Reference = new OpenApiReference
        {
            Id = JwtBearerDefaults.AuthenticationScheme,
            Type = ReferenceType.SecurityScheme
        }
    };
    c.AddSecurityDefinition(securityScheme.Reference.Id, securityScheme);
    c.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        {securityScheme, new string[] { }}
    });
});
```

Obrázek 10 Konfigurace Swaggeru v C#

Na přiloženém obrázku je konfigurace Swaggeru, která se skládá ze tří částí. První, červená část, je základní konfigurace, díky které se generuje Swagger.

V další zelené části je kód pro slovní popis metod, díky které můžeme použít nad každou metodou v kontroléru tag summary, jehož obsah se nám propíše k metodám ve Swaggeru.

```
/// <summary>
/// Košík uživatele
/// </summary>
[HttpGet("Cart")]
public async Task<ActionResult<IEnumerable<Item>>> GetUserCart()...
```

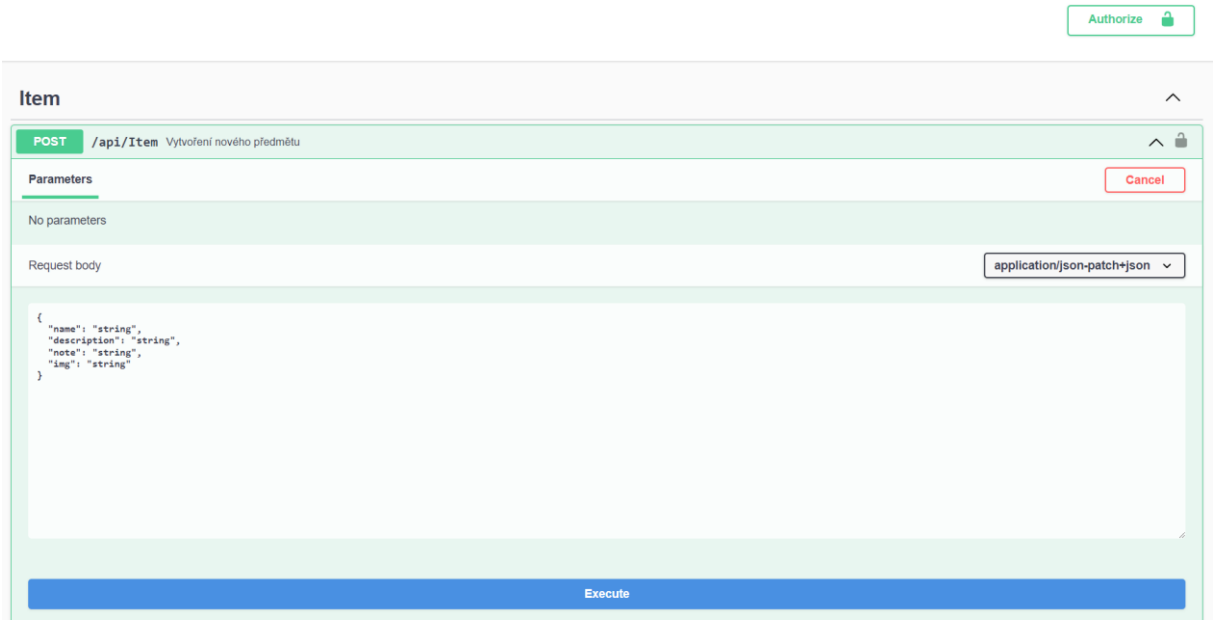
Obrázek 11 Ukázka použití summary

Poslední modrá část zajišťuje možnost testovat zabezpečení – autorizaci API, pomocí JWT Bearer tokenu.

Zatímco se tato konfigurace vyskytovala v programu v builder.Services, je potřeba pro funkční Swagger ještě napsat jeden řádek kódu na závěr Program.cs.

```
app.UseSwagger();
```

6.1.2 Použití Swaggeru

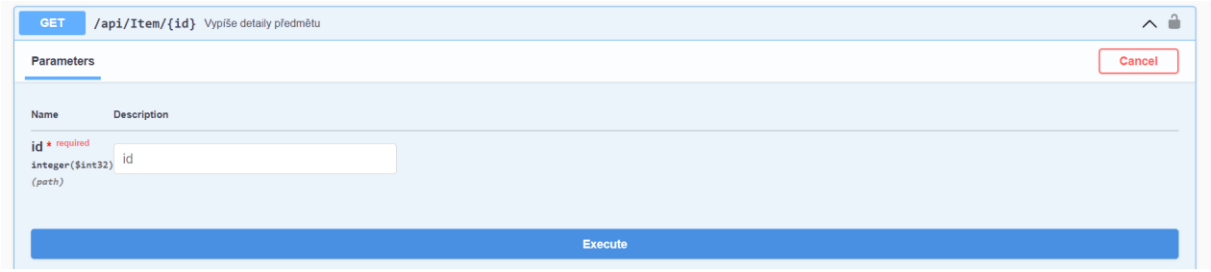


Obrázek 12 Příklad metody ve Swaggeru

V pravém horním rohu se nachází Authorize tlačítko, pro vložení Bearer tokenu, kterým se uživatel bude vůči aplikaci autorizovat.

V levém horním rohu vidíme název kontroléru, a pod ním seznam všech metod, které obsahuje. V tomto případě již máme otevřenou metodu pro vytvoření nového předmětu. Je zde vidět že se jedná o metodu POST (každá metoda má i vlastní barvu), adresu, na kterou endpoint odkazuje, a za tím popis, který jsme předtím napsali k metodě do summary.

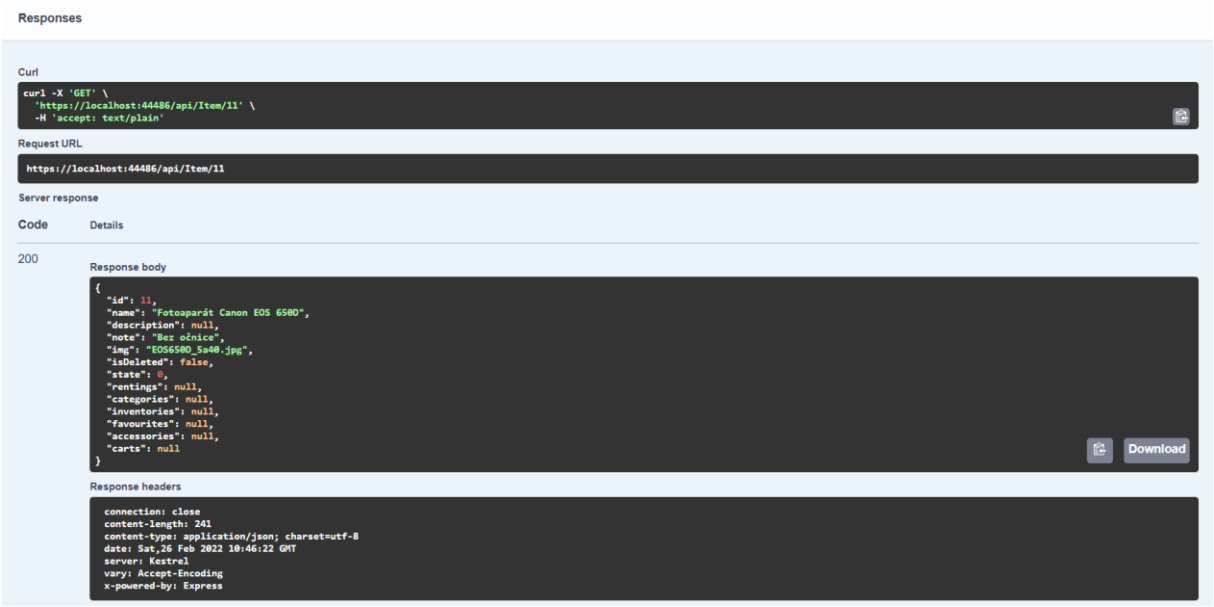
Tato metoda nemá žádné parametry, které by se psali přímo do URL, ale obsahuje tělo požadavku, kam se zapíší vlastnosti, které chceme, aby předmět obsahoval. Vlastnosti se oddělují čárkou, a vlastnosti, které nechceme vyplňovat, nemusíme udávat.



Obrázek 13 Metoda s parametry v URL

Když zkusíme metodu zavolat, v Responses uvidíme Request URL, která reprezentuje adresu, na které bychom metodu zavolali. Dostaneme stavový kód odpovědi (200 pro úspěšné splnění požadavku, 401 pokud uživatel nemá přístup a 404 pokud požadovaná data neexistují).

Odpověď na požadavek nám také vrátí data, v našem případě předmět a hlavičku odpovědi.



Obrázek 14 Odpověď na požadavek

6.2 Samotné testování

Pro každou metodu jsem testovala nejdřív s validními parametry, zda dělá, co se od ní očekává, poté jsem zkoušela i zadat parametry neexistující položky, zda vyhodí kód 404. V případě předmětu dostaneme 404 i pro předmět, který má vlastnost IsDeleted nastavenou na true.

V počátcích vývoje a testování, kdy ještě nebylo API napojeno na autorizační server zde byly metody pro vytvoření uživatele, jeho úpravu a mazání, které ve finální verzi nejsou potřeba, zbyla zde pouze metoda pro přidání uživatele přihlášeného k autorizačnímu serveru do databáze Rentals aplikace.

Pro testování přihlášených uživatelů a jejich práv jsem využívala pomocnou aplikaci (8) napsanou v Reactu, která slouží k vypsání Bearer tokenu a profilu přihlášeného uživatele k oauth.pslib.cz. Tento token jsem pokaždé zadala do Swaggeru a otestovala, zda nastavená práva fungují.

Závěr

Téma odborné práce jsem si zvolila, abych si vyzkoušela navržení reálné databáze a API a abych si rozšířila své znalosti. Trochu jsem se obávala, že moje znalosti na databázi v praxi nebudou dostačující, ale nakonec jsem použila převážně věci, co jsem znala ze školy. V odborné práci jsem musela některé informace nastudovat dopředu, čímž jsem měla výhodu v následujících projektech.

Zjistila jsem, jak se nahrávají obrázky do API a jak je poté přes endpoint získat zpět, to pro mě asi byla největší novinka. I když prvotní implementace byla přímo přes nahrávání souboru jako parametr endpointu. Práce s ním by nebyla problém, kdyby aplikace nezamrzala. Pak ale přišel vedoucí práce s tím, že bude lepší, a i pro obsah endpointu jednodušší použít tus. Příjemná novinka pro mě bylo také použití PATCH API, které velmi zjednodušuje úpravu entit. Do té doby jsem používala pouze PUT, který je přímo v REST API. Ujistila jsem se, jak se píšou metody a k čemu slouží jaký stavový kód.

Autorizace pro mě nebyla úplná novinka, i když jsme ve škole pro správu uživatele použili pouze identitu ASP.NET, na praxi ve firmě mi autorizace za použití Bearer tokenu byla přiblížena, a tak mě nepřekvapilo, když jsem dostala za úkol implementovat autorizaci vůči školnímu serveru. Ovšem claimy a práci s nimi, jsem si vyzkoušela až v tomto projektu úplně poprvé, i když si myslím, že jsme je již ve druháku zmiňovali.

Při praxi ve firmě jsem se také seznámila s použitím objektu jako parametr endpointu, který všechny potřebné parametry obsahuje. Bez tohoto jsem měla jen díky parametrům některé metody dlouhé přes dva řádky.

Problémy vyloženě s vývojem jsem neměla, veškeré moje chyby byly naprosto hloupé, i přesto že jsem u jejich řešení strávila i několik dní. Například pozůstatky v databázi po identitě ASP.NET, mi dělaly neplechu při implementaci vlastní Bearer autentizace, nebo SPA proxy mělo problém s novou verzí Node.js.

V budoucnu by dala implementovat důvěryhodnost uživatele, která je již v databázi připravená, ale nebyla vymyšlená funkcionalita, proto se v aplikaci zatím nevyskytne. Také by se dali přidat e-maily na upozorňování, že výpůjčka končí, že je předmět dostupný, nebo že uživatel přesáhl termín vrácení.

Seznam zkratek a odborných výrazů

URL

Uniform Resource Locator – webová adresa.

API

Application Programming Interface – sbírka procedur, které může programátor využívat.

JSON

JavaScript Object Notation – formát zápisu dat odvozený z JavaScriptu.

E-R model

Entity-Relationship model – model pro návrh vazeb mezi entitami v databázi.

Frontend

Část aplikace, kterou vidí uživatel.

Backend

Část aplikace, která se stará o funkcionalitu.

Endpoint

Koncový bod – adresa, na které se nachází API metoda.

React

JavaScriptová knihovna pro tvorbu webových aplikací.

Seznam obrázků

Použité zdroje

1. **Wikipedia foundation.** Databáze. *Wikipedie Otevřená encyklopedie.* [Online] 1. Březen 2022. [Citace: 5. Březen 2022.] https://cs.wikipedia.org/wiki/Datab%C3%A1ze.

2. **Gillis, Alexander S.** REST API (RESTful API). *TechTarget SearchApp Architecture.* [Online] Září 2020. [Citace: 2. Březen 2022.] https://www.techtarget.com/searchapparchitecture/definition/RESTful-API.

3. **Malý, Martin.** REST: architektura pro webové API. *zdroják.cz.* [Online] 3. Srpen 2009. [Citace: 2. Březen 2022.] https://zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

4. **Cirkovic, Uros.** MongoDB | CRUD Operations. *GitHub.* [Online] 19. Duben 2020. [Citace: 3. Březen 2022.] https://github.com/ross-u/MongoDB-CRUD-Introduction-Exercise/blob/master/README.md.

5. **MDN contributors.** HTTP request methods. *MDN Web Docs.* [Online] 3. Říjen 2021. [Citace: 4. Březen 2022.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

6. **Wikipedia foundation.** Stavové kódy HTTP. *Wikipedie Otevřená encyklopedie.* [Online] 13. Listopad 2021. [Citace: 5. Březen 2022.] https://cs.wikipedia.org/wiki/Stavov%C3%A9_k%C3%B3dy_HTTP.

7. —. Swagger (software). *Wikipedie otevřená encyklopedie.* [Online] 16. Červenec 2021. [Citace: 26. Únor 2022.] https://cs.wikipedia.org/wiki/Swagger_(software).

8. **Stehlík, Michal.** oauth-token-retriever. *GitHub.* [Online] 30. Září 2021. [Citace: 14. Listopad 2021.] https://github.com/MichalStehlik/oauth-token-retriever.

9. **Matsson, Stefan.** tus. *tus Open Protocol for Resumable File Uploads.* [Online] 18. Únor 2022. [Citace: 8. Březen 2022.] https://tus.io/.

10. **Dykstra, Tom a Larkin, Kirk.** JsonPatch in ASP.NET Core web API. *Microsoft.* [Online] 28. Únor 2022. [Citace: 8. Březen 2022.] https://docs.microsoft.com/cs-cz/aspnet/core/web-api/jsonpatch?view=aspnetcore-6.0.

A. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

- **MP2022-Daňková-Kateřina-REST-API-aplikace-Rentals.docx** – editovatelná verze dokumentace odborné práce
- **MP2022-Daňková-Kateřina-REST-API-aplikace-Rentals.pdf** – tisknutelná verze dokumentace odborné práce
- **Aplikace** – zdrojové kódy