Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Aplikace pro studijní agendu základní školy s webovou službou

Plzeň 2021 Martin Dub

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Martin DUB**
Osobní číslo:	**A20B0080P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Studijní obor:	**Informatika**
Téma práce:	**Aplikace pro studijní agendu základní školy s webovou službou**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s typy dat, které je potřeba uchovávat ve studijní agendě základní školy.
2. Seznamte se webovými službami.
3. Navrhněte aplikaci pro studijní agendu základní školy, jejímž primárním rozhraním bude webová služba. Při návrhu dbejte důsledně na zamýšlený účel aplikace, kterým bude benchmark testovacích nástrojů.
4. Navrženou aplikaci implementujte. Dbejte na řádné oddělení jednotlivých vrstev/částí aplikace dle zvolené architektury.
5. Vytvořenou aplikaci důkladně otestujte tak, aby množství chyb bylo minimální.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Tomáš Potužák, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **5. října 2020**
Termín odevzdání bakalářské práce: **6. května 2021**

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 26. října 2020

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2021

Martin Dub

Abstract

This bachelor thesis deals with design and implementation of application for study agenda of elementary school. The interface of the application is provided using web services. The main purpose of the application is to be part of a benchmark for software testing tools. For this reason, there was a great emphasis on the testing of the application and eliminating as many bugs as possible. The goal is to create a "bug-free" application, to which known errors will be introduced. Regardless of this main purpose, the application still provides enough functionality to be at least partially used for study agenda of elementary school. Theoretical part describes data, which are needed in study agenda of elementary school, and web services. In practical part, the design and implementation of the application including its thorough testing is described.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací aplikace pro studijní agendu základní školy. Rozhraní aplikace je poskytnuto pomocí webových služeb. Hlavním účelem aplikace je být součástí benchmarku testovacích nástrojů. Z tohoto důvodu byl kladen velký důraz na testování aplikace a odstranění co největšího počtu chyb. Cílem je vytvořit "bezchybnou" aplikaci, do které budou zavedeny známé chyby. Bez ohledu na tento hlavní účel, aplikace poskytuje dostatek funkcí pro alespoň částečné využití pro studijní agendu základní školy. Teoretická část práce popisuje data, která jsou potřebná ve studijní agendě základní školy, a webové služby. V praktické části je popsána konečná implementace aplikace včetně jejího důsledného otestování.

Poděkování

Velmi rád bych poděkoval panu Ing. Tomášovi Potužákovi, Ph.D. za vedení mé bakalářské práce, cenné rady a postřehy, které mi poskytl při konzultacích.

Obsah

1 Úvod

Tato bakalářská práce se zabývá návrhem a implementací aplikace pro studijní agendu základní školy s webovou službou. Hlavní motivací vzniku této aplikace je benchmark testovacích nástrojů. Přesto by bylo možné tuto aplikaci alespoň částečně použít pro studijní agendu základní školy.

V teoretické části se zabývám daty, která jsou potřeba uchovávat ve studijní agendě. Jsou zde uvedeny zákony, které určují, jaká data mají školy povinnost ukládat. Také jsou zde rozebrány existující školní informační systémy a jejich moduly.

Následuje popis webových služeb. Popisuji zde nejčastější formáty dat přenášených webovými službami. Následně se zabývám nejpoužívanějšími typy webových služeb a způsoby jejich zabezpečení.

V praktické části se zabývám návrhem aplikace. Je zde popsána zamýšlená funkcionalita aplikace, návrh databáze, zdůvodnění výběru použitých technologií a typy architektur webových služeb.

Dále popisuji implementaci aplikace. Rozebírám zde strukturu balíků aplikace a jejich funkcionalitu. Také jsem uvedl obecný tvar dotazů pro práci s aplikací.

Na závěr se věnuji testování. Uvedl jsem typy prováděných testů a zdůvodnil jsem výběr těchto typů. Také jsem zmínil výsledky testování a chyby, které testování odhalilo.

2 Typy dat studijní agendy základní školy

V této kapitole se zabývám typy dat, které základní školy musí uchovávat. Také jsou zde uvedeny nejpoužívanější školní informační systémy a funkce, které poskytují.

2.1 Povinná dokumentace školy

Povinná dokumentace škol je dána ustanovením § 28 odstavce 1 školského zákona [9]:

Školy a školská zařízení vedou podle povahy své činnosti tuto dokumentaci:
a) rozhodnutí o zápisu do školského rejstříku a o jeho změnách a doklady uvedené v § 147,
b) evidenci dětí, žáků nebo studentů (dále jen "školní matrika"),
c) doklady o přijímání dětí, žáků, studentů a uchazečů ke vzdělávání, o průběhu vzdělávání a jeho ukončování,
d) vzdělávací programy podle § 4 až 6,
e) výroční zprávy o činnosti školy,
f) třídní knihu, která obsahuje průkazné údaje o poskytovaném vzdělávání a jeho průběhu,
g) školní řád nebo vnitřní řád, rozvrh vyučovacích hodin,
h) záznamy z pedagogických rad,
i) knihu úrazů a záznamy o úrazech dětí, žáků a studentů, popřípadě lékařské posudky,
j) protokoly a záznamy o provedených kontrolách a inspekční zprávy,
k) personální a mzdovou dokumentaci, hospodářskou dokumentaci a účetní evidenci a další dokumentaci stanovenou zvláštními právními předpisy.

2.2 Obsah matriky školy

Údaje, které musí obsahovat školní matrika je dána ustanovením § 28 odstavce 2 školského zákona [9]:

Školní matrika školy podle povahy její činnosti obsahuje tyto údaje o dítěti, žákovi nebo studentovi:
a) jméno a příjmení, rodné číslo, popřípadě datum narození, nebylo-li rodné číslo dítěti, žákovi nebo studentovi přiděleno, dále státní občanství, místo narození a místo trvalého pobytu, popřípadě místo pobytu na území České republiky podle druhu pobytu cizince nebo místo pobytu v zahraničí, nepobývá-li dítě, žák nebo student na území České republiky,
b) údaje o předchozím vzdělávání, včetně dosaženého stupně vzdělání,
c) obor, formu a délku vzdělávání, jde-li o střední a vyšší odbornou školu,
d) datum zahájení vzdělávání ve škole,
e) údaje o průběhu a výsledcích vzdělávání ve škole, vyučovací jazyk,
f) údaje o znevýhodnění dítěte, žáka nebo studenta uvedeném v § 16, údaje o mimořádném nadání, údaje o podpůrných opatřeních poskytovaných dítěti, žákovi nebo studentovi školou v souladu s § 16, a o závěrech vyšetření uvedených v doporučení školského poradenského zařízení,
g) údaje o zdravotní způsobilosti ke vzdělávání a o zdravotních obtížích, které by mohly mít vliv na průběh vzdělávání,
h) datum ukončení vzdělávání ve škole; údaje o zkoušce, jíž bylo vzdělávání ve střední nebo vyšší odborné škole ukončeno,
i) jméno a příjmení zákonného zástupce, místo trvalého pobytu nebo bydliště, pokud nemá na území České republiky místo trvalého pobytu, a adresu pro doručování písemností, telefonické spojení.

2.3 Školní informační systém

Školní informační systémy umožňují bezpečné ukládání dat, které školy musí ze zákona uchovávat, jejich zpracovávání a poskytování dalším orgánům. Jsou to komplexní webové aplikace rozdělené na moduly. Toto rozdělení umožňuje školám výběr pouze funkcí, které opravdu potřebují [12].

2.4 Existující školní informační systémy

V rámci této bakalářské práce jsem prozkoumal několik školních informačních systémů, abych zjistil, jaká data jsou potřeba uchovávat v rámci studijní agendy. Nejvíce jsem se zaměřil na školní systém Bakaláři, který je jedním z nejpoužívanějších a nejrozsáhlejších. Dále jsem prozkoumal systémy Etřídnice a Škola OnLine

2.4.1 Bakaláři

Bakaláři [8] je nejrozšířenější školní informační systém v České republice. Tento systém používá dle výrobce přes 60% škol v ČR [8]. Systém je rozdělen na moduly, kde každý modul představuje jeden funkční celek.

Hlavními moduly systému Bakaláři jsou:

- *Evidence* – obsahuje osobní data žáků, klasifikaci a tisk vysvědčení. Je zde předpřipraveno mnoho sestav, které lze libovolně modifikovat. Díky tomu lze vytvářet požadované výkazy daným orgánům.
- *Internetová žákovská knížka* – umožňuje rodičům prohlížet klasifikaci a docházku žáků, rozvrh a jeho změny. Dále umožňuje komunikaci mezi rodiči a školou, omlouvání žáků a záznamy z elektronické třídní knihy.
- *Rozvrh* – napomáhá rozvrháři s hledáním kolizí, výměnou a přesunem hodin. Také umí generovat rozvrh.
- *Třídní kniha* – umožňuje zapisovat informace o hodině – téma, poznámky a absence žáků.

2.4.2 Etřídnice

Etřídnice [4] je informační systém pro školy, který je funkcemi velmi podobný systému Bakaláři.

Hlavními moduly systému Etřídnice jsou:

- *Třídní kniha* – umožňuje zápis předmětů, učiva, absence, hospitace, inspekce, projektů, akcí a kurzů. Dále poskytuje přehled zameškaných hodin žáků, zameškaných hodin v předmětech a odučených hodin učitelů za období. Také dovoluje rodičům přistupovat k absenci žáka.
- *Žákovská knížka* – dovoluje zápis známek, váženého průměru a slovního hodnocení. Dále výpis a tisk známek a slovního hodnocení. Rodičům umožňuje přístup ke klasifikaci.
- *Rozvrh hodin* – umí vytvářet rozvrh podle zadaných kritérií s možností manuálních úprav, eliminuje kolize. Dále umožňuje tisk rozvrhu a suplování.
- *Vysvědčení* – dovoluje zadávání konečných známek, slovního hodnocení a hodnocení chování na vysvědčení. Automaticky vyplňuje známky,

údaje a absence o žácích. Umožňuje hromadnou úpravu informací třídy a hromadný tisk vysvědčení třídy.

- *Komunikace* – zajišťuje komunikaci mezi rodiči a školou.
- *Úkoly* – umožňuje zadávání domácích úkolů celé třídě a nahrávání příloh. Poskytuje přehled splněných úkolů.

2.4.3 Škola OnLine

Škola OnLine [7] je školní informační systém, který je také rozdělen na moduly. Na rozdíl od ostatních systémů umožňuje evidovat úrazy.

Hlavními moduly systému Škola OnLine jsou:

- *Evidence úrazů* – umožňuje vést knihu úrazů a odesílat záznamy o úrazech České školní inspekci.
- *Komunikace* – umožňuje komunikaci mezi školou a rodiči.
- *Evidence osob* – umožňuje evidovat žáky, učitele, zákonné zástupce a další uživatele.
- *Domácí úkoly* – umožňuje zadávat a odevzdávat domácí úkoly.
- *Rozvrh, suplování a školní akce* – zobrazuje žákům i učitelům jejich rozvrhy včetně změn. Obsahuje nástroje pro kontrolu duplicit v rozvrhu a suplování, přehled o absencích učitelů a suplovaných hodinách. Také lze plánovat události zasahující do rozvrhu, například školní výlety nebo exkurze.
- *Tiskové sestavy* – obsahuje předpřipravené šablony, které lze vytisknout. Jedná se například o vysvědčení, seznamy žáků, přehledy hodnocení a docházky nebo záznam o úrazu.

2.5 Shrnutí

Každý ze zkoumaných školních informačních systémů se dělí do modulů. Každý z modulů se zabývá určitou podmnožinou dat, které musí školy uchovávat. Každý systém se liší svými moduly, které obsahuje. Například systém Škola OnLine obsahuje modul Evidence úrazů, zatímco ostatní školní systémy nikoli.

Pro návrh aplikace pro studijní agendu základní školy (který následuje v kapitole 4) zahrnu pouze uchovávání a zpracování základních dat jako údaje o žácích a jejich hodnocení a absenci, učitelích a rozvrhu a to z důvodu, že zamýšleným účelem aplikace není studijní agenda, ale benchmark testovacích nástrojů.

3 Webové služby

Tato kapitola se zabývá webovými službami, druhy webových služeb a typy přenášených dat.

Existuje mnoho definic pojmu webová služba. Jednou z nich je, že webová služba je aplikace nebo zdroj dat přístupná přes webový protokol, jako například *HTTP* nebo *HTTPS*. Webové služby jsou určeny pro komunikaci s ostatními programy, nikoli pro přímou komunikaci s uživatelem [16].

3.1 Formáty dat přenášených webovými službami

Nejčastěji používanými formáty přenášených dat jsou *XML* a *JSON*, protože jsou snadno rozpoznatelné ostatními programy, které tato data přijímají [6].

3.1.1 Extensible Markup Language

XML (eXtensible Markup Language) je značkovací jazyk podobný *HTML*, ale bez předdefinovaných tagů. Místo toho lze definovat vlastní tagy pro potřeby programu, což umožňuje ukládat data ve formátu vhodném pro ukládání, vyhledávání a sdílení. Tím, že základní formát XML je standardizován, tak i po sdílení XML dat mezi různými systémy a platformami příjemce bude schopen rozdělit a rozpoznat data. [20].

```
<?xml version="1.0"?>
<doc>
 <article id="0527">
   <updInfo>Aug-04-2006(NOT TO BE TRANSLATED)</updInfo>
   <title>XML formats</title>
   <par id="par1">
     This is a short tutorial for <ref target="#89">XML</ref> formats.
     This <img target="diagram1" alt="Diagram for illustration purposes" />
     diagram is provided for illustration only. All material is copyrighted
     (&copyright;).
   </par>
 </article>
</doc>
```

Obrázek 3.1: Příklad XML kódu (převzato z [19])

Pro správnost XML dokumentů musí být dodrženo následující:

- dokument musí mít správný formát
- dokument musí dodržovat syntaxi XML

- dokument musí dodržovat sémantická pravidla, která jsou většinou nastavena v XML schématu

Na obrázku 3.1 je zobrazen příklad XML se správným kódem. Prvek `xml` není tag, ale deklarace. Používá se pro přenos metadat dokumentu. Tagy mohou mít parametry, jak můžeme vidět u tagu `article`. Všechny tagy musí být nadefinovány, jinak je dokument nevalidní.

3.1.2 JavaScript Object Notation

JSON (JavaScript Object Notation) je standardní textový formát pro reprezentaci strukturovaných dat, který je založen na syntaxi objektů jazyka JavaScript. Často je používán pro přenos dat webových aplikací (například přenos dat mezi serverem a klientem). JSON existuje jako text, což je vhodné pro síťový přenos dat. Pro přístup k datům je třeba konverze do nativního JavaScriptového objektu (JavaScript poskytuje globální JSON objekt, který obsahuje metody pro konverzi mezi textem a objektem) [18].

```
[
  {
    "name": "Molecule Man",
    "age": 29,
    "secretIdentity": "Dan Jukes",
    "powers": [
      "Radiation resistance",
      "Turning tiny",
      "Radiation blast"
    ]
  },
  {
    "name": "Madame Uppercut",
    "age": 39,
    "secretIdentity": "Jane Wilson",
    "powers": [
      "Million tonne punch",
      "Damage resistance",
      "Superhuman reflexes"
    ]
  }
]
```

Obrázek 3.2: Příklad JSON kódu (převzato z [18])

Ve formátu JSON lze používat některé základní datové typy jako text, čísla, pole, boolean a objekty. To umožňuje vytvořit hierarchii dat jako je na obrázku 3.2.

3.2 Typy webových služeb

V dnešní době se používají převážně dva typy webových služeb a to webové služby typu *SOAP* a *REST*.

3.2.1 SOAP

Webové služby typu SOAP využívají Simple Object Access Protocol. SOAP je protokol založen na XML pro výměnu zpráv v distribuovaném decentralizovaném prostředí. SOAP lze použít v kombinaci s celou řadou dalších protokolů, ale typicky se používá v kombinaci s protokolem HTTP. SOAP zpráva je XML dokument, který se z několika částí [14]:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Header>
       <t:Transaction
         xmlns:t="some-URI"
         xsi:type="xsd:int" mustUnderstand="1">
           5
       </t:Transaction>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse
         xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 3.3: Příklad SOAP zprávy s povinným zpracováním hlavičky (převzato z [14])

- *Envelope* (obálka) – povinný element XML dokumentu reprezentující zprávu.
- *Header* (hlavička) - nepovinný element, slouží k přidávání funkcí do SOAP zprávy decentralizovaným způsobem bez předchozí dohody komunikujících stran. V hlavičce lze definovat několik atributů:
  - *actor* – používá se ke směrování hlavičky konkrétnímu příjemci
  - *mustUnderstand* – určuje, zda příjemce musí hlavičku zpracovat
- *Body* (tělo) – Obsahuje informace pro konečného příjemce zprávy

- *Fault* (chyba) – Používá se k přenosu chybových nebo stavových informací, skládá se ze čtyř elementů:
  - *faultcode* – kód pro identifikaci druhu chyby
  - *faultstring* – informace vysvětlující chybu
  - *faultactor* – informace o tom, kdo chybu způsobil
  - *faultstring* – detailnější informace o chybě

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
           <faultcode>SOAP-ENV:Server</faultcode>
           <faultstring>Server Error</faultstring>
           <detail>
               <e:myfaultdetails xmlns:e="Some-URI">
                 <message>
                   My application didn't work
                 </message>
                 <errorcode>
                   1001
                 </errorcode>
               </e:myfaultdetails>
           </detail>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 3.4: Příklad SOAP zprávy s chybou (převzato z [14])

3.2.2 REST

Dalším často používaným typem webových služeb jsou služby typu *REST*. Representational State Transfer je architektonický styl, který usnadňuje vzájemnou komunikaci systémů. Systémy REST, často nazývané RESTful systémy, se vyznačují tím, že jsou bezestavové a oddělují část klienta a serveru [11].

V architektonickém stylu REST lze obě části implementovat nezávisle na sobě, což umožňuje měnit serverovou část bez nutnosti zásahu do kódu klienta a naopak. Použitím REST rozhraní mohou různí klienti dosáhnout stejných zdrojů, provádět stejné akce a dostávat stejné odpovědi [11].

RESTful systémy jsou bezestavové, což znamená, že server nemusí znát aktuální stav klienta a naopak. Tímto způsobem může server i klient rozumět jakékoli přijaté zprávě bez znalosti předchozích zpráv. Bezestavovost je dána pomocí zdrojů, nikoli příkazů. Zdroje jsou objekty popisující dokument nebo věc, která bude potřeba k uložení nebo odeslání dalším službám. REST systémy interagují pomocí standardních operací se zdroji. Díky tomu lze dosáhnout rychlého výkonu a škálovatelnosti jako komponent, které lze spravovat, aktualizovat a znovu použít bez ovlivnění systému jako celku, a to i během provozu [11].

REST vyžaduje, aby klient vytvářel žádosti na server, pokud chce získat nebo upravovat data na serveru. Žádost obecně obsahuje tyto části [11]:

- *HTTP metoda* - definuje, která z operací se vykoná, existují čtyři základní pro interakci se zdroji v REST systému:
  - *GET* – získání specifického zdroje (podle id) nebo kolekce zdrojů
  - *POST* – vytvoření nového zdroje
  - *PUT* – úprava konkrétního zdroje (podle id)
  - *DELETE* – odstranění specifického zdroje (podle id)
- *header* (hlavička) – umožňuje klientovi přidat informace o žádosti, nejčastěji se zde určuje, jaký typ dat je klient ochoten přijmout od serveru. Toto je dáno polem *Accept*, které se skládá z typu a podtypu.
- *path* – cesta ke zdroji
- *body message* – nepovinné, používá se k vložení údajů zdroje pro POST a UPDATE requesty

Pokud server odesílá v odpovědi data klientovi, musí obsahovat *content-type* v hlavičce odpovědi. Toto pole v hlavičce říká, jaký typ dat je použit v těle odpovědi. Odpověď také musí obsahovat stavový kód, který informuje o úspěchu či neúspěchu provedené operace. Základní typy stavových kódů jsou [11]:

- *OK* (200) – standardní odpověď pro úspěšnou HTTP žádost
- *CREATED* (201) – standardní odpověď pro HTTP žádost, po které se úspěšně vytvořil nový prvek
- *NO CONTENT* (204) – standardní odpověď pro úspěšnou HTTP žádost s prázdným tělem odpovědi

- *BAD REQUEST* (400) – žádost nemůže být provedena kvůli nesprávné syntaxi, překročené velikosti nebo jinému problému u klienta
- *FORBIDDEN* (403) – klient nemá oprávnění přistupovat k tomuto zdroji
- *FORBIDDEN* (404) – zdroj nelze nalézt v daném čase, je možné že byl smazán nebo neexistuje
- *INTERNAL SERVER ERROR* (500) – obecná odpověď pro nečekanou chybu, pokud nejsou dostupné konkrétnější informace

```
GET http://fashionboutique.com/customers
Accept: application/json
```

Obrázek 3.5: Příklad dotazu klienta na REST server (převzato z [11])

Na obrázku 3.5 je uveden dotaz klienta, který slouží pro zobrazení všech zákazníků. Možná odpověď serveru je na obrázku 3.6, která by byla doplněna o data zákazníků ve formátu application/json.

```
Status Code: 200 (OK)
Content-type: application/json
```

Obrázek 3.6: Příklad odpovědi REST serveru na dotaz klienta (převzato z [11])

3.3 Zabezpečení webových služeb

Tato podkapitola se zabývá možnostmi zabezpečení webových služeb proti možným útokům.

3.3.1 Autentizace

Autentizace slouží k identifikaci totožnosti uživatele. Identita uživatele je kotrolována na základě údajů od uživatele jako [13]:

- věci, které uživatel vlastní a jednoznačně ho identifikují – například doklady vydané důvěryhodným orgánem jako občanský průkaz, cestovní pas nebo čipová karta

- údaje, které by měl vědět pouze daný uživatel – například sdílené tajné heslo nebo jiný tajný údaj
- jiné vlastnosti identifikující uživatele – biometrické informace

Použitím kombinace několika typů uvedených zabezpečení lze dosáhnout většího zabezpečení.

3.3.2 Autorizace

Autorizace (řízení přístupu) umožňuje přidělovat práva uživatelům pro přístup ke konkrétním zdrojům. Autorizace dovoluje rozhodnout, které operace ověření uživatelé mohou provádět. Existují tři základní pohledy na autorizaci [13]:

- *Na základě rolí* – základní myšlenkou je, že množinu identit lze uskupit do rolí a práva pak přiřadit každé roli zvlášť
- *Na základě identit* – v závislosti na údajích poskytnutých uživatelem může služba udělit nebo odepřít přístup ke zdroji, umožňuje jemnější autorizaci, než autorizace na základě rolí
- *Na základě zdrojů* – každý ze zdrojů je zabezpečen přístupovými právy

3.3.3 Zabezpečení zpráv

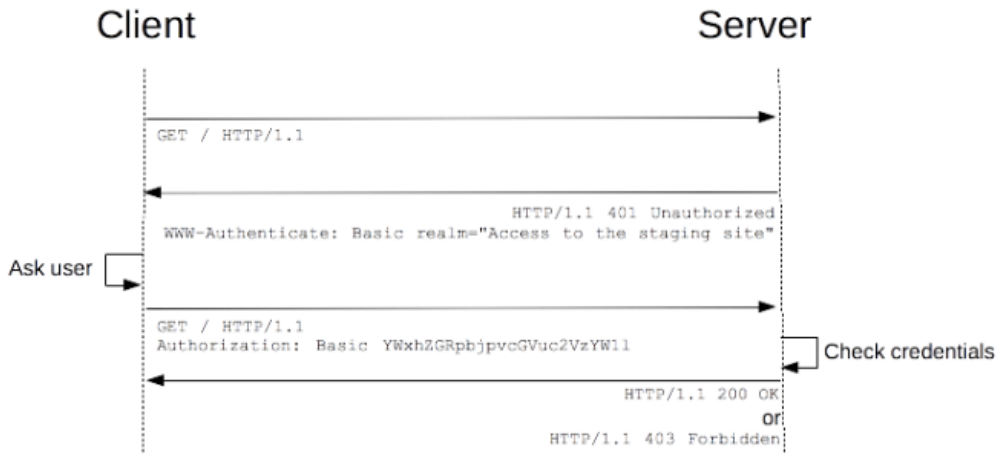
Zabezpečení zpráv řeší dva koncepty – důvěryhodnost zpráv a integritu zpráv.

Důvěryhodnost zpráv zahrnuje zachování obsahu v tajnosti, i totožnosti obou komunikujících stran. Důvěryhodnosti lze dosáhnout šifrováním obsahu zpráv a skrytím totožnosti odesílatele a příjemce. Odesílatel používá veřejný klíč příjemce k zašifrování zprávy, ovšem pouze soukromým klíčem lze dešifrovat obsah zprávy. Tím je zajištěno, že obsah zprávy nelze přečíst třetí stranou během přenosu [13].

Integrita zpráv může být zajištěna digitálním podpisem autority. Digitální podpisy se používají k ověření odesílatele zprávy a k zajištění integrity zprávy. Pokud je zpráva SOAP podepsána digitálním podpisem, je ze zprávy vytvořen unikátní hash, který je následně zašifrován soukromým klíčem odesílatele. Příjemce poté dešifruje zprávu pomocí veřejného klíče odesílatele. Toto slouží k ověření odesílatele, protože pouze odesílatel mohl zašifrovat zprávu soukromým klíčem. Slouží také k tomu, aby nebyla při přenosu přenášena zpráva SOAP, protože příjemce může porovnat hash odeslaný se zprávou s hashem vytvořeným u příjemce [13].

3.3.4 HTTP Basic Authentication

Jedním ze způsobů, jak zabezpečit webovou službu používající HTTP hlavičku je *HTTP Basic Authentication*.



Obrázek 3.7: Princip HTTP Basic Authentication (převzato z [5])

Na obrázku 3.7 je popsán princip HTTP Basic Authentication. Poté, co klient pošle dotaz na server, tak server odpoví se statusem 401 (Unauthorized) a vyzve klienta k zadání jména a hesla. Pokud klient zadá správné údaje do hlavičky HTTP požadavku, je ověřen a má přístup k serveru. Údaje jsou uloženy na straně serveru a jsou konfigurovatelné [5].

```
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

Obrázek 3.8: Příklad údajů v HTTP hlavičce pro ověření pomocí HTTP Basic Authentication (převzato z [2])

Na obrázku 3.8 je příklad pole pro ověření, který se vloží do HTTP hlavičky požadavku. Údaj *YWxhZGRpbjpvcGVuc2VzYW1l* je výsledek po zakódování *aladdin:opensesame* pomocí kódování *base64* [3], kde aladdin je jméno uživatele a opensesame je heslo.

4 Návrh

Tato kapitola se zabývá návrhem požadované aplikace. Obsahuje specifikaci požadavků aplikace a diagram případů užití. Také jsou zde nastíněna různá řešení a zdůvodnění výběru konkrétních řešení.

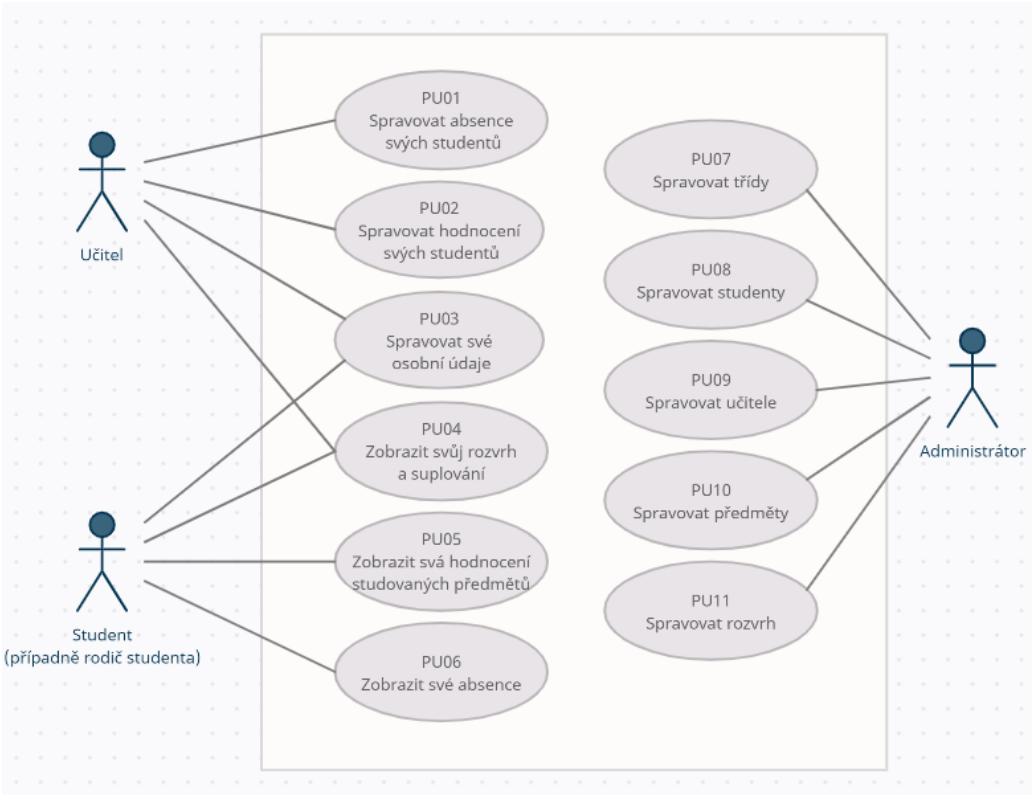
4.1 Specifikace požadavků

Specifikace požadavků popisuje funkcionalitu požadované aplikace. Požadavky na aplikaci jsou následující:

- aplikace bude napsána v jazyce Java
- primárním rozhraním aplikace bude webová služba
- pro ukládání dat bude použita databáze
- komunikace mezi aplikací a databází bude řešena pomocí frameworku Hibernate
- jako rozhraní budou použity webové služby typu REST
- přenos dat bude realizován ve formátu JSON
- aplikace bude zpracovávat data pro běžnou agendu základní školy, nemusí obsahovat veškerou agendu (kvůli použití jako část benchmarku testovacích nástrojů)
- aplikace bude důkladně otestována, aby množství chyb bylo co nejmenší
- webové služby budou používat alespoň základní formu zabezpečení, aby přístup nebyl zcela veřejný

4.2 Funkcionalita

Vzhledem k tomu, že zamýšlený účel této aplikace je součást benchmarku testovacích nástrojů, tak jsem vybral pouze podmnožinu funkcí, které budu implementovat. Popis funkcí aplikace je popsán diagramem případů užití na obrázku 4.1



Obrázek 4.1: Diagram případů užití

V systému vystupují tři aktéři - student (případně rodič studenta), učitel a administrátor, jak je vidět na obrázku 4.1.

4.2.1 Popis případů užití

V této podkapitole popisuji jednotlivé případy užití uvedené na obrázku 4.1.

- PU01 Spravovat absence svých studentů – učitel může zobrazit, vytvářet, upravovat a mazat absence studentů z předmětů, kteří studují předměty vyučované tímto učitelem
- PU02 Spravovat hodnocení svých studentů – učitel muže zobrazit, vytvářet, upravovat a mazat hodnocení z předmětů u studentů, kteří studují předměty vyučované tímto učitelem
- PU03 Spravovat své osobní údaje – učitel a student může zobrazit a upravovat své osobní údaje
- PU04 Zobrazit svůj rozvrh a suplování – učitel a student si mohou zobrazit svůj rozvrh a suplování

- PU05 Zobrazit hodnocení studovaných předmětů – student si muže zobrazit hodnocení z předmětů, které studuje
- PU06 Zobrazit své absence – student si může zobrazit své absence z předmětů, které studuje
- PU07 Spravovat třídy – administrátor může zobrazit, vytvářet, upravovat a mazat školní třídy
- PU08 Spravovat studenty – administrátor může zobrazit, vytvářet, upravovat a mazat studenty
- PU09 Spravovat učitele – administrátor může zobrazit, vytvářet, upravovat a mazat učitele
- PU10 Spravovat předměty – administrátor může zobrazit, vytvářet, upravovat a mazat předměty
- PU11 Spravovat rozvrh – administrátor může zobrazit, vytvářet, upravovat a mazat rozvrh

4.3 Návrh databáze

ERA model databáze je přiložen v příloze A. Tabulka `students` představuje entitu studenta. Každý student musí patřit do nějaké třídy, což zajišťuje tabulka `classes`. Třída musí mít nějakou svoji místnost - tabulka `rooms`. Všechny místnosti jsou umístěny v budovách - tabulka `buildings`. Každá třída má třídního učitele, kteří budou uloženi v tabulce `teachers`. V tabulce `subjectlist` budou uloženy obecné typy předmětů jako matematika, fyzika nebo český jazyk. Tabulka `teacherssubjects` představuje seznam obecných předmětů, které může daný učitel vyučovat, což bude kontrolováno při přiřazování konkrétních předmětů učiteli. V tabulce `subjects` jsou konkrétní předměty například matematika 9.A, které jsou odvozeny od obecných typů předmětů. Tabulkou `classessubjects` bude zajišťeno přidávání konkrétních předmětů dané třídě, což bude sloužit jako kontrola při přidávání předmětů určitému studentovi - tabulka `studentssubjects`. Každý studentův předmět bude moci mít ohodnocení - tabulka `evaluations`.

Pro tvorbu rozvrhu budou potřeba rozvrhové akce - tabulka `scheduleactions`. Rozvrhová akce potřebuje vyučujícího učitele, místnost a vyučovací hodinu (tabulka `schoolhours`), kde a kdy proběhne. Protože se jedná o pravidelné rozvrhové akce, je zde také uveden den (tabulka `schooldays`) například pondělí. Konkrétní rozvrhové akce budou uloženy

v tabulce `scheduleactionsdate`. Tato akce obahuje konkrétní datum, kdy akce proběhne. Stejně jako u pravidelných rozvrhových akcí, je zde den, hodina a místnost. Pokud se alespoň jeden z těchto údajů bude lišit (případně ještě učitel) oproti pravidelné rozvrhové akci, jedná se o suplování. Absence žáků v tabulce `absences` jsou vztaženy ke konkrétní rozvrhové akci a studentovi.

4.4 Použité technologie

Jak již bylo zmíněno v kapitole 4.1, aplikace musí být napsána v programovacím jazyce Java. Pro snažší správu aplikace jsem použiji framework *Spring Boot*.

4.4.1 Spring Boot

Spring boot je volně dostupný framework založený na Javě, který se používá k vytváření mikroslužeb. Výhody frameworku Spring Boot jsou [15]:

- ve frameworku spring boot je vše automaticky konfigurováno, nejsou nutné žádné další konfigurace
- nabízí Spring aplikaci založenou na anotacích
- zjednodušuje správu závislostí
- vyhýbá se komplexním konfiguracím z frameworku Spring

Tento framework jsem zvolil především kvůli jednoduché konfiguraci a použitelnosti s frameworkem *Hibernate*.

4.4.2 Hibernate

Hibernate je Java framework, který zjednodušuje vývoj aplikací, které komunikují s databází. Výhodou je, že používá objektově-relační mapování [10].

Objektově-relační mapování umožňuje ukládat objekty objektově orientovaných jazyků (například Java) do relační databáze. Java třídám přísluší databázové tabulky, kde datové typy Javy se přemění na datové typy SQL. Toto mapování také zpracovává vztahy mezi objekty, které jsou pak reprezentovány spojováním tabulek a cizími klíči. Hibernate implementuje Java Persistence API, které můžeme používat přímo voláním jeho metody nebo obecně přes rozhraní definované JPA [10].

Tento framework použiji z důvodu uvedeného ve specifikaci požadavků a také pro zjednodušení komunikace mezi aplikací a databází.

4.4.3 Upřesnění webové služby

Pro potřeby webové služby použiji webovou službu typu REST a formát pro přenos dat webovou službou bude JSON, protože je to vyžadováno zadavatelem.

4.4.4 Zabezpečení webové služby

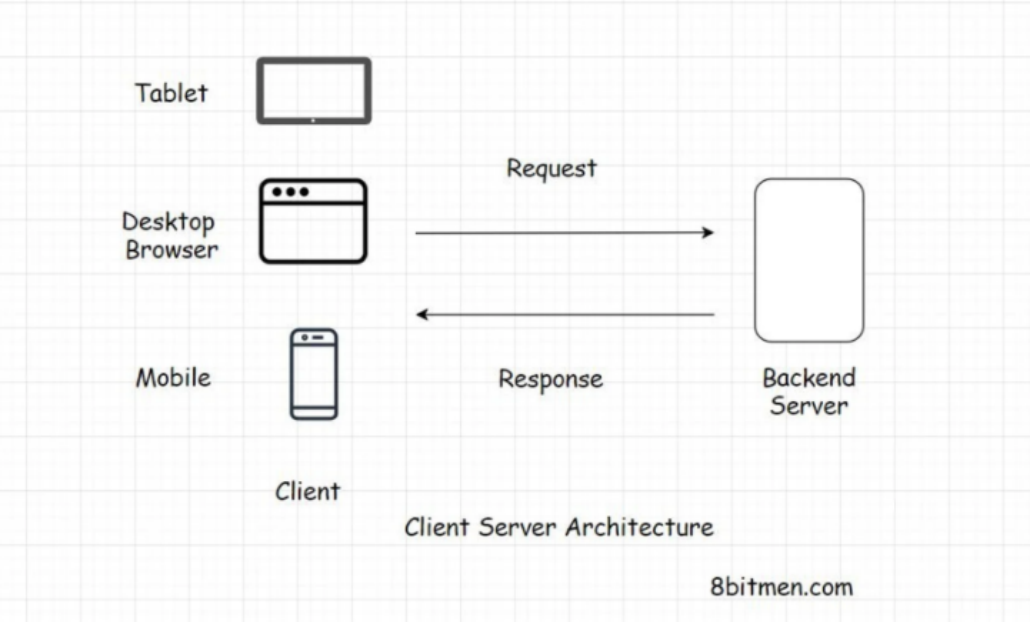
Jelikož bude v této bakalářské práci kladen velký důraz na testování a eliminaci co největšího počtu chyb, tak zabezpečím aplikaci pouze pomocí HTTP Basic Authentication (kapitola 3.3.4). Role a přístupová práva ke zdrojům nebudou implementována z důvodu, že primárním účelem této aplikace bude součást benchmarku testovacích nástrojů a také proto, že by to snížilo čas určený k testování, což by vedlo k eliminaci menšího počtu chyb.

4.5 Architektura aplikace

Pro architekturu aplikace byly zvažovány varianty uvedené v následujících podkapitolách.

4.5.1 Klient-server

Klient-server architektura funguje na principu žádost-odpověď. Klient pošle žádost serveru, který klientovi zpět odpoví, jak je znázorněno na obrázku 4.2 [17].



Obrázek 4.2: Klient-server architektura (převzato z [17])

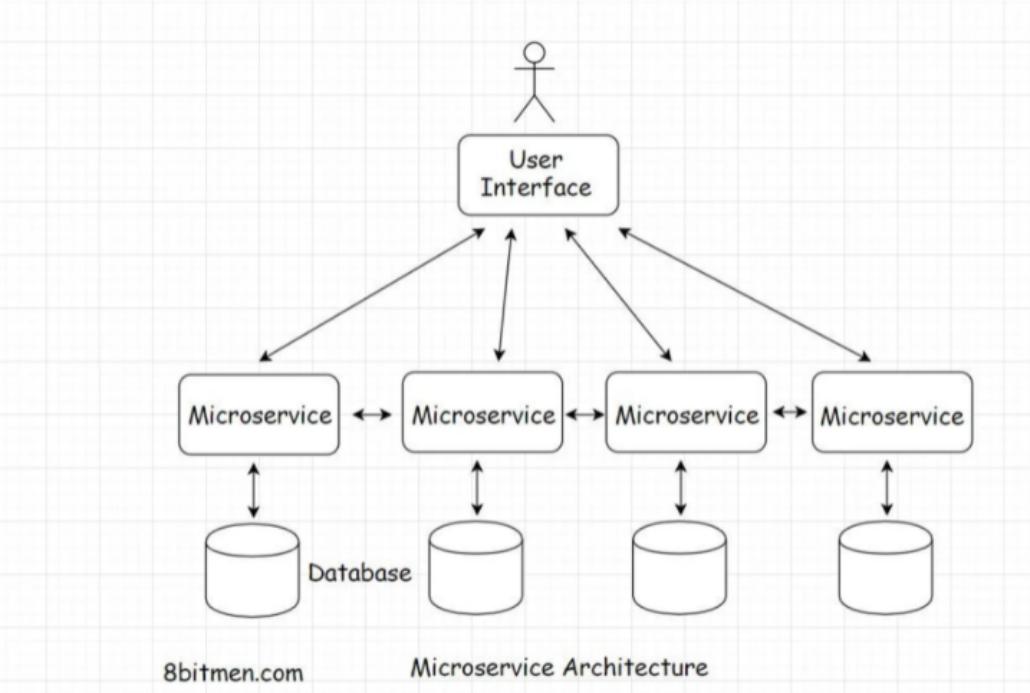
4.5.2 Model-View-Controller

MVC (Model-View-Controller) architektura je architektonický vzor, kde logika aplikace je rozdělena na tři komponenty podle funkčnosti [17]:

- *Models* (modely) – představují způsob dat uložených v databázi
- *Views* (pohledy) – komponenty viditelné pro uživatele, například grafické uživatelské rozhraní
- *Controllers* (kontrolery) – komponenty, které fungují jako rozhraní mezi modely a pohledy

4.5.3 Mikroslužby

V architektuře mikroslužeb jsou různé funkce, které jsou rozdělené do příslušných modulů. Tyto moduly vzájemně spolupracují a tvoří jednu velkou službu. Tato architektura usnadňuje údržbu aplikací, vývoj a nasazení aplikací [17].



Obrázek 4.3: Architektura mikroslužeb (převzato z [17])

4.5.4 Vrstvená architektura

Vrstvená architektura se používá pro strukturování programů, které lze rozdělit na vrstvy, kde každá z nich je na určité úrovni abstrakce. Každá vrstva poskytuje služby další vyšší vrstvě. Mezi běžně používané vrstvy patří [17]:

- *Prezentační vrstva*
- *Aplikační vrstva*
- *Logická vrstva*
- *Vrstva přistupující k datům*

4.5.5 Použitá architektura

Z uvedených architektur odpovídá nejvíce povaze této práce vrstvená architektura (kapitola 4.5.4). Budu potřebovat vrstvu přistupující k datům z databáze, logickou vrstvu pro práci daty z databáze a také vrstvu pro zpracovávání příchozích HTTP dotazů, která bude využívat metody logické vrstvy.

5 Implementace

V této kapitole je popsána implementace této bakalářské práce - aplikace pro studijní agendu s webovou službou. Implementace vychází z návrhu představeného v kapitole 4. Návod pro sestavení a spuštění programu je v příloze B.

Jak už jsem uvedl v kapitole 4, tato práce byla implementována v programovacím jazyku Java (verze 11.0.9) s použitím frameworku Spring Boot verze 2.4.1 k ulehčení programování a konfigurace a framework Hibernate verze 5.4.25.Final pro komunikaci s databází.

5.1 Struktura aplikace

Tato aplikace je rozdělena na následující balíky:

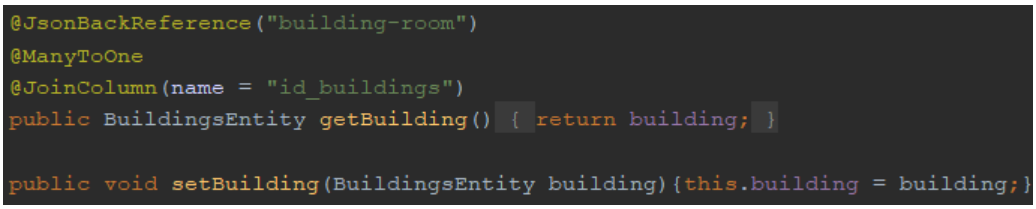
- `database` – dělí se na další balíky související s databází:
  - `entities` – představují objekty, které reprezentují entity tabulek databáze
  - `repositories` – zde jsou rozhraní, která poskytují základní metody pro práci s daty z databázových tabulek
  - `validators` – v tomto balíku jsou třídy kontrolující správnost entit a také podpůrné metody pro služby
- `DTO` (Data Transfer Object) – reprezentují objekty ve vhodném formátu pro přenos entit
- `service` – třídy tohoto balíku představují služby
- `controller` – v tomto balíku jsou kontrolery

5.1.1 Entities

Každá třída balíku `entities` představuje tabulku databáze. Název třídy je odvozen od názvu tabulky, tedy například entita pro tabulku `buildings` se jmenuje `BuildingsEntity`. Každá entita obsahuje atributy stejné, jako atributy dané tabulky v databázi. Dále jsou zde gettery a settery pro tyto atributy a metoda `equals()` pro porovnávání těchto objektů.

Obrázek 5.1: Příklad mapování atributů

Mapování atributů je zajištěno anotacemi. Na obrázku 5.1 je znázorněn příklad mapování atributů. Anotace mohou být použity přímo u atributů nebo u getterů, já používám u všech tříd jednotně připnutí ke getterům. U základního atributu, jako je název budovy, používám anotaci `@Basic` a `@Column`, čímž definuji, který z atributů databázové tabulky chci mapovat na tento atribut Java třídy. Primární klíč je označen `@Id` a `@GeneratedValue` pro určení typu generování hodnoty klíče. Dalším typem jsou cizí klíče, v tomto případě se jedná o vazbu 1:N, kde budova může obsahovat více místností a místost patří jedné budově. Proto jsem použil anotaci `@OneToMany`, kde parametr `building` je název atributu ve třídě `RoomsEntity`, jak je vidět na obrázku 5.2.



Obrázek 5.2: Příklad mapování atributů

Pomocí `@JoinColumn` specifikuji sloupec tabulky pro spojení entit. `@JsonManagedReference` a `@JsonBackReference` pouze upravuje serializaci entit, tedy místnost nemá referenci na budovu, zatímco budova na místnosti ano.

5.1.2 DTO

V balíku `DTO` jsou třídy, které obsahují podmnožinu atributů dané entity. Tím si mohu vybrat, které atributy entity budu přenášet. Tento modul jsem vytvořil také proto, že například entita `ScheduleactionEntity` je složená pouze z primárního klíče a cizích klíčů a jak už jsem popisoval v kapitole 5.1.1, tak tato entita nemá z důvodu N:1 vazby k ostatním entitám referenci. Ale díky této třídě mohu naplnit vytvořený DTO objekt v mnou definovaném formátu, který bude dále použit pro přenos.

5.1.3 Repositories

Třídy balíku `repositories` jsou rozhraní, která poskytují základní metody pro získání dat z databáze. Metody stačí pouze definovat a za dodržení správné konvence není třeba nic implementovat, Hibernate sestaví databázový dotaz na základě definice metody. Každé z těchto rozhraní je nutné označit anotací `@Repository`, jak je vidět na obrázku 5.3.

```
@Repository
public interface SubjectRepository extends JpaRepository<SubjectsEntity, Integer>
{
    List<SubjectsEntity> findBySubjectName(String name);
    List<SubjectsEntity> findBySubjectlist(SubjectlistEntity subjectlist);
    List<SubjectsEntity> findByTeacher(TeachersEntity teacher);
    List<SubjectsEntity> findByClasses(ClassesEntity classesEntity);
    SubjectsEntity findByScheduleactions(ScheduleactionsEntity scheduleactionsEntity);
    List<SubjectsEntity> findByStudents(StudentsEntity student);
}
```

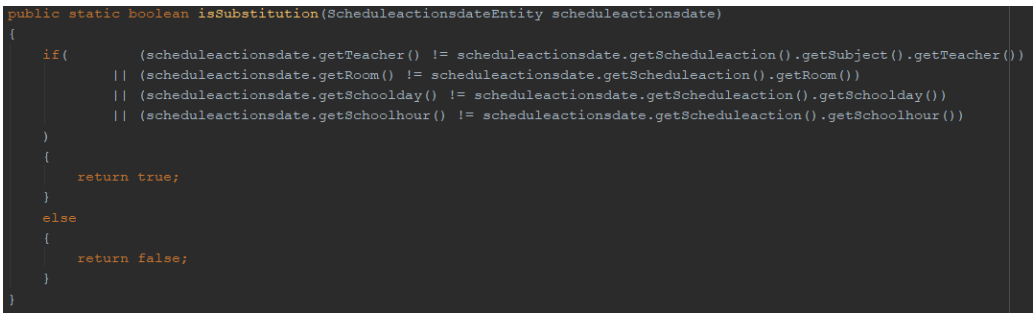
Obrázek 5.3: Příklad repository

5.1.4 Validators

Primární funkcí balíku `validators` je validovat, zda konkrétní entita existuje v databázi. K tomu slouží metoda `validateExist()`. Existenci entity lze kontrolovat podle zadaného primárního klíče nebo názvu, pokud nějaký takový název entita obsahuje. Jak je vidět na obrázku 5.4, primární klíč nebo název je předáván entiou v parametru metody. Jako návratový typ jsem použil `ResponseEntity`. Je to výhodné, protože tím sdělím návratový kód i informaci o výsledku, což jsem dále použil u služeb a kontrolerů. Pokud entita existuje, tak se naplní do předávané entity.

Obrázek 5.4: Příklad validátoru

Také se zde vyskytují metody `convertToTransfer()` a `convertListToTransfer`, které slouží ke konverzi entit na objekty vhodné pro přenos (DTO). Dále metody `validatePost()` a `validatePut()`, které používám pro validaci základních předávaných údajů u POST a PUT requestů. U některých validátorů jsem také implementoval další podpůrné metody týkající se entit, například metodu `isSubstitution()`, která zjišťuje, zda konkrétní rozvrhová akce je suplováním, jak je prezentováno na obrázku 5.5. Pokud se oproti pravidelné rozvrhové akci liší místnost, učitel, den nebo hodina, je tato konkrétní rozvrhová akce suplováním.



Obrázek 5.5: Určení, zda rozvrhová akce je suplováním

5.1.5 Service

Třídy balíku `service` představují služby. V těchto třídách jsem implementoval logiku pro práci s daty z databáze. Pro komunikaci s databází využívám

metody balíku `repositories`, pro ověřování předávaných entit využívám balík `validators`.

Každá služba obsahuje metodu pro výpis všech daných entit, výpis entity podle primárního klíče, vytvoření nové entity, úpravu entity podle primárního klíče a smazání entity podle primárního klíče. Také jsem vytvořil mnoho metod, které vrací danou entitu podle entity jiného typu, jak můžeme vidět na obrázku 5.6. Nejprve je třeba zjistit, zda student se zadaným jménem existuje, což jsem zjistil pomocí metody z třídy `StudentsEntityValidator`. Jak už jsem zmínil v kapitole 5.1.4, typ `ResponseEntity` se zde hodí, protože ho mohu rovnou vrátit i z této služby a předat dál do kontroleru. Pokud daný student nemá žádné absence, vrátím příslušnou informaci, jinak seznam jeho absencí.

```
public ResponseEntity getAbsencesByStudentName(String firstName, String lastName)
{
    StudentsEntity student = new StudentsEntity();
    student.setFirstName(firstName);
    student.setLastName(lastName);
    ResponseEntity response = StudentsEntityValidator.validateExist(student, studentRepository);
    if(response.getStatusCode() != HttpStatus.OK)
    {
        return response;
    }

    List<AbsencesEntity> absences = absenceRepository.findByStudent(student);

    if (absences.isEmpty())
    {
        return ResponseEntity.ok().body( "Student with name " + firstName + " " + lastName + " has no absences.");
    }

    return ResponseEntity.ok().body(AbsencesEntityValidator.convertListToTransfer(absences));
}
```

Obrázek 5.6: Získání absence podle jména studenta

Metody pro vytáření a úpravu dat využívají metody `validatePost()` a `validatePut()` pro kontrolu základních údajů, jako zda je jméno žáka zadáno. Také je potřeba stejně jako u příkladu na obrázku 5.6 kontrolovat, zda existují entity jiného typu, pokud je daná entita používá. U metod pro mazání jsem musel ověřovat, zda po smazání nevznikne konflikt, například budovu nelze smazat pokud obsahuje místnosti.

5.1.6 Controller

Balík `controller` slouží ke správě příchozích dotazů. Na základě typu dotazu a tvaru *URI* (Uniform Resource Identifier - jednotný identifikátor zdroje) se zavolá příslušná metoda určité služby. Každá třída tohoto modulu je označena anotací `@RestController`. Také jsem je označil anotací `@RequestMapping` pro určení první části URI, která představuje entity vztahující se k danému kontroleru. Například `@RequestMapping("/absences")`

jsem použil v kontroleru `AbsenceController`, který správuje dotazy na absence.

Podle kapitoly 3.2.2 existují 4 typy zálkadní HTTP dotazů - GET, POST, UPDATE a DELETE. Tomu odpovídají anotace `@GetMapping`, `@PostMapping`, `@PutMapping` a `@DeleteMapping`, které jsem použil u metod kontrolerů. Určují, při jaké kombinaci typu dotazu a URI se metoda zavolá. Na obrázku 5.7 je příklad metody, která se zavolá, pokud bude dotaz typu GET a URI bude `/absences`.

```
@RestController
@RequestMapping("/absences")
public class AbsenceController extends Controller
{
    @Autowired
    AbsencesService absencesService;

    @GetMapping
    public List<AbsenceDTO> getAllAbsences() { return absencesService.getAllAbsences(); }
```

Obrázek 5.7: Příklad metody kontroleru

Všechny kontrolery tohoto balíku dědí ze třídy `Controller`, která zajišťuje ošetření výjimek, jako například chybný formát vstupních dat.

5.1.7 Zabezpečení

Zabezpečení jsem implementoval způsobem Basic HTTP Authentication podle návhu. Zabezpečení je implementováno ve třídě `AuthConfig.java`.

```
spring.security.user.name=user
spring.security.user.password=password
```

Obrázek 5.8: Příklad konfigurace přihlašovacích údajů

Přihlašovací jméno a heslo lze konfigurovat v souboru `resources/application.properties.java` jak je vidět na obrázku 5.8.

```
C:\Users\Dubs>curl http://localhost:8080/buildings --user user:password
```

Obrázek 5.9: Příklad volání služeb s použitím zabezpečení

Pokud chceme používat služby serveru z příkazové řádky, vložíme přihlašovací údaje způsobem, jako je na obrázku 5.9.

5.2 Validní dotazy

Již jsem zmínil, že metoda služby, která bude zavolána kontrolerem, je dána typem dotazu a URI. Základní typy dotazů jsou dány, ale tvar URI se může lišit. V rámci této aplikace používám následující tvary URI:

- `/zpracovávaná_entita`
- `/zpracovávaná_entita/atribut/hodnota_atributu`
- `/zpracovávaná_entita/jiná_entita/atribut/hodnota_atributu`

Zpracovávaná entita je například budova, pak příklad prvního tvaru je `/buildings`, což by v kombinaci s typem HTTP metody get znamenalo výpis všech budov. Pokud chceme vyhledat budovy podle jména, použijeme `/buildings/name/{jméno}`, kde jméno je název hledané budovy. Některé entity lze hledat i podle jiných entit. Například `/buildings/room/id/{id}` znamená vyhledání budovy podle primárního klíče místnosti. Tímto způsobem jsou skládány URI této aplikace. Seznam všech validních HTTP dotazů je v příloze B.

6 Testování

V této kapitole popisuji testování implementované aplikace. Jedním z nejdůležitějších bodů této bakalářské práce bylo otestovat aplikaci tak, aby množství chyb v aplikaci bylo co nejmenší. Pro testování jsem použil různé druhy testů a snažil se pokrýt co největší část možných situací, které mohou nastat při práci s touto aplikací.

6.1 Unit testy

Cílem unit (jednotkových) testů je otestovat nejmenší části kódu. Tyto testy by měly být nezávislé na ostatních částech kódu, proto u částí, které jsou závislé na jiných objektech, používám *mocking*.

Cílem mockingu je izolovat testovaný objekt od ostatních objektů, na kterých je tato jednotka závislá a ověřit správnost jeho chování. Mock objektům můžeme nastavit, jaké hodnoty budou vracet při zavolání jejich konkrétní metody. Tímto způsobem jsem otestoval metody balíku `controller`. Kontrolery používají metody služeb, proto je potřeba označit objekt služby atotací `@Mock`, čímž se stane mock objektem. Pak v testech pouze nastavím, jakou hodnotu vrátí konkrétní metoda dané služby.

```
@Test
public void getAllBuildings() throws Exception
{
    when(buildingsService.getAllBuildings()).thenReturn(buildings);

    mockMvc.perform(get( urlTemplate: "/buildings").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$[0].buildingName", is(buildings.get(0).getBuildingName())))
            .andExpect(jsonPath( expression: "$[1].buildingName", is(buildings.get(1).getBuildingName())));
}
```

Obrázek 6.1: Příklad testu kontroleru

Na obrázku 6.1 je test metody kontroleru `BuildingController` pro výpis všech budov. Nejprve nastavím, aby metoda `getAllBuildings()` vracela seznam mnou definovaných metod a následně ověřím, zda se po provedení příslušného HTTP request metoda skutečně zavolá a vrátí požadovaná data.

6.2 Integrační testy

Integrační testy oproti jednotkovým testům testují funkčnost komunikujících částí aplikace. Ve své práci tento typ testování používám u služeb a

validátorů, které obsahují podpůrné metody služeb, protože služby pracují s databází prostřednictvím metod balíku `repositories`, což je odlišná část aplikace, na které jsou služby závislé.

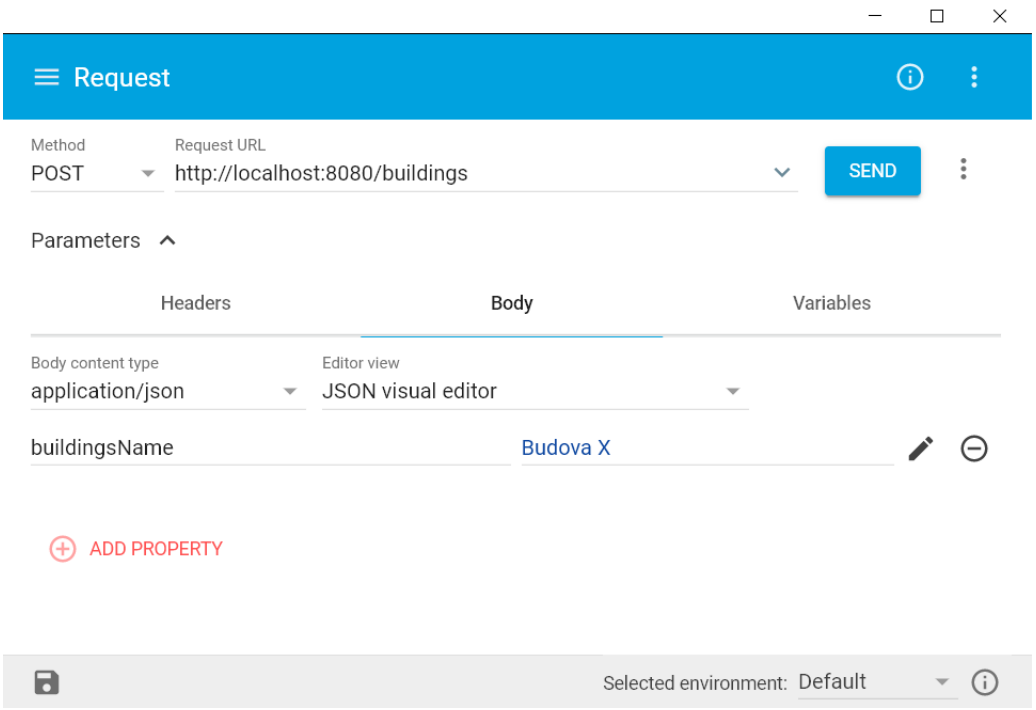
K testování práce s databázovými daty používám testovací databázi, která má stejnou strukturu jako databáze jako databáze aplikace. Struktura databáze je uložena v souboru `test/resources/schema.sql`. Obsah této databáze se po dokončení testů smaže. Pomocí metod instance třídy `TestEntityManager` lze přidat testovací data, která jsem v testech použil. Výhodou použití testovací databáze místo databáze aplikace je, že neovlivním skutečná data uložena v databázi aplikace a také při každém spuštění testů mám v testovací databázi stejná data. Konfiguraci databáze lze změnit v souboru `test/resources/application.properties`.

```
@Test
void getSubjectById_NotExists() {
    final int id = Integer.MAX_VALUE;
    ResponseEntity response = subjectService.getSubjectById(id);
    assertEquals(HttpStatus.UNPROCESSABLE_ENTITY, response.getStatusCode());
    assertEquals(response.getBody(),  actual: "Subject with id " + id + " does not exist.");
}
```

Obrázek 6.2: Příklad testu služby

6.3 Testy podle testovacích scénářů

Aplikaci jsem také otestoval provedením testů podle testovacích scénářů. Přestože rozhraním této aplikace je webová služba, která nemá žádné uživatelské grafické rozhraní, bylo potřeba otestovat výstupy, aby bylo jasné, že aplikace jako celek funguje správně a dává očekávané hodnoty. Tyto testy jsem provedl manuálně za použití nástroje *Advanced REST Client* [1], což je volně dostupný nástroj určený k testování REST služeb.



Obrázek 6.3: Advanced REST Client

Tento nástroj, jak je vidět na obrázku 6.3, poskytuje vše potřebné pro manuální testování, tedy výběr typu HTTP metody, zadání URI, vkládání atributů do hlavičky a také podporuje snadné vkládání JSON objektů do těla požadavku. Testovací scénáře jsou uvedeny v příloze C.

6.4 Výsledky testování

Celkově bylo provedeno 655 jednotkových a integračních testů. Těmito testy bylo pokryto 100% kódu balíčků `service` a `validators` a 98% kódu balíčku `controller`, kde neotestované případy byly otestované následně manuálními testy podle testovacích scénářů.

Balík `validators` byl otestován 122 testy, balík `service` 383 testy a balík `controller` 150 testy.

Průměrný počet jednotkových a integračních testů na třídu balíku `validators` je 8,4, balíku `service` 27,4 a balíku `controller` 15.

Počet jednotkových a integračních testů na metodu je v průměru 3,15 pro balík `validators`, 2,8 pro balík `service` a 1,1 pro balík `controller`.

Integrační a jednotkové testy byly doplněny o 111 manuálních testů podle testovacích scénářů (příloha C).

Tabulka 6.1: Statistika provedených jednotkových a integračních testů podle balíků

Název balíku	Počet provedených jednotkových a integračních testů	Průměrný počet jednotkových a integračních testů na třídu balíku	Průměrný počet jednotkových a integračních testů na metodu tříd balíku
validators	122	8,4	3,2
service	383	27,4	2,8
controller	150	15	1,1

Testování odhalilo následující chyby:

- unit testy odhalily špatný formát URI pro výpis konkrétní rozvrhové akce podle id
- integrační testy odhalily:
  - špatný návratový text u metody pro smazání třídy, pokud smazání proběhlo úspěšně
  - u metody pro vytvoření třídy vyhozena výjimka, pokud nebyl zadán učitel nebo místnost
  - u metody pro přiřazení předmětu studentovi vyhozena výjimka, pokud byla překročena kapacita místností, kde je předmět vyučován
  - špatný návratový text u metody pro zobrazení studenta podle id, pokud student neexistuje
- manuální testy podle scénářů odhalily špatný návratový text u metody pro vytvoření školního dne, pokud nebyl zadán

Všechny popsané chyby jsem opravil a zopakoval testy. Při opakovaných testech již další chyby odhaleny nebyly.

7 Závěr

V teoretické části této práce jsem prostudoval typy dat, které je potřeba uchovávat ve studijní agendě základních škol. Prostudoval jsem zákony nařizující školám povinnost uchovávat konkrétní data. Následně jsem prozkoumal existující školní informační systémy, které tato data uchovávají.

Dále jsem se věnoval webovým službám. Jsou zde popsány nejpoužívanější formáty dat přenášených webovými službami. Dále jsem uvedl a popsal webové služby typu SOAP a REST. A také jsem popsal možnosti zabezpečení webových služeb proti útokům.

V praktické části práce jsem se zabýval návrhem aplikace. Návrh jsem uzpůsobil specifikaci požadavků, která je zde také uvedena. Dále jsem popsal zamýšlenou funkcionalitu aplikace, navrhnul model databáze a zdůvodnil, jaké technologie použiji. Rozebral jsem také druhy architektur webových služeb a zdůvodnil výběr konkrétní architektury.

Následně jsem se zabýval implementací aplikace. Popsal jsem strukturu aplikace, funkcionalitu balíků a také důležité implementační detaily. Je zde také zmíněna obecná forma dotazů pro práci s aplikací.

Nakonec jsem popsal testování aplikace. Popsal jsem několik typů testů, kterými jsem aplikaci otestoval. Nakonec jsem zmínil výsledky testování a nalezené chyby pomocí těchto testů.

Výsledkem této bakalářské práce je funkční aplikace, která poskytuje základní funkce pro studijní agendu základní školy s webovou službou. Tato aplikace bude využita jako součást benchmarku testovacích nástrojů. Z tohoto důvodu byla aplikace otestována nad rámec běžných zvyklostí, aby se co nejvíce omezilo množství neznámých chyb obsažených v aplikaci.

Literatura

[1] *Advanced REST Client* [online]. 2021. [cit. 2021/04/30]. Dostupné z: https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo.

[2] *Authorization* [online]. 2021. [cit. 2021/05/03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization.

[3] *Base64* [online]. 2021. [cit. 2021/05/03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Base64.

[4] *Etřídnice* [online]. 2021. [cit. 2021/04/11]. Dostupné z: https://www.etridnice.cz/.

[5] *HTTP authentication* [online]. 2021. [cit. 2021/05/03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication.

[6] Jakob, J. *Web Service Message Formats* [online]. 2015. [cit. 2021/04/30]. Dostupné z: http://tutorials.jenkov.com/web-services/message-formats.html.

[7] *Škola OnLine* [online]. 2021. [cit. 2021/04/11]. Dostupné z: https://www.skolaonline.cz/.

[8] *Bakaláři* [online]. 2021. [cit. 2021/04/11]. Dostupné z: https://www.bakalari.cz/.

[9] *Školský zákon* [online]. 2020. [cit. 2021/04/11]. Dostupné z: https://www.msmt.cz/dokumenty-3/skolsky-zakon-ve-zneni-ucinnem-od-11-7-2020.

[10] Peterková, A. *Seznámení s Hibernate ORM* [online]. 2014. [cit. 2021/04/29]. Dostupné z: https://blog.bcvsolutions.eu/seznameni-s-hibernate-orm/.

[11] *What is REST?* [online]. 2021. [cit. 2021/04/29]. Dostupné z: https://www.codecademy.com/articles/what-is-rest.

[12] Růžičková, M. *Informační systémy pro ZŠ a SŠ* [online]. Medium, 2018. [cit. 2021/04/11]. Dostupné z: https://medium.com/edtech-kisk/informa%C4%8Dn%C3%AD-syst%C3%A9my-pro-z%C5%A1-a-s%C5%A1-b861ab00594a.

[13] *Understanding Web Service Security Concepts* [online]. 2021. [cit. 2021/04/29]. Dostupné z: https://docs.oracle.com/cloud/latest/fmw122100/OWSMC/owsm-security-concepts.htm#OWSMC116.

[14] *Simple Object Access Protocol (SOAP) 1.1* [online]. 2000. [cit. 2021/04/29]. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383497.

[15] *Spring Boot - Introduction* [online]. 2020. [cit. 2021/04/29]. Dostupné z: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm.

[16] *Web Service* [online]. 2017. [cit. 2021/04/11]. Dostupné z: https://techterms.com/definition/web_service.

[17] Wilson, C. *How to Design a Web Application: Software Architecture 101* [online]. 2020. [cit. 2021/04/29]. Dostupné z: https://www.educative.io/blog/how-to-design-a-web-application-software-architecture-101.

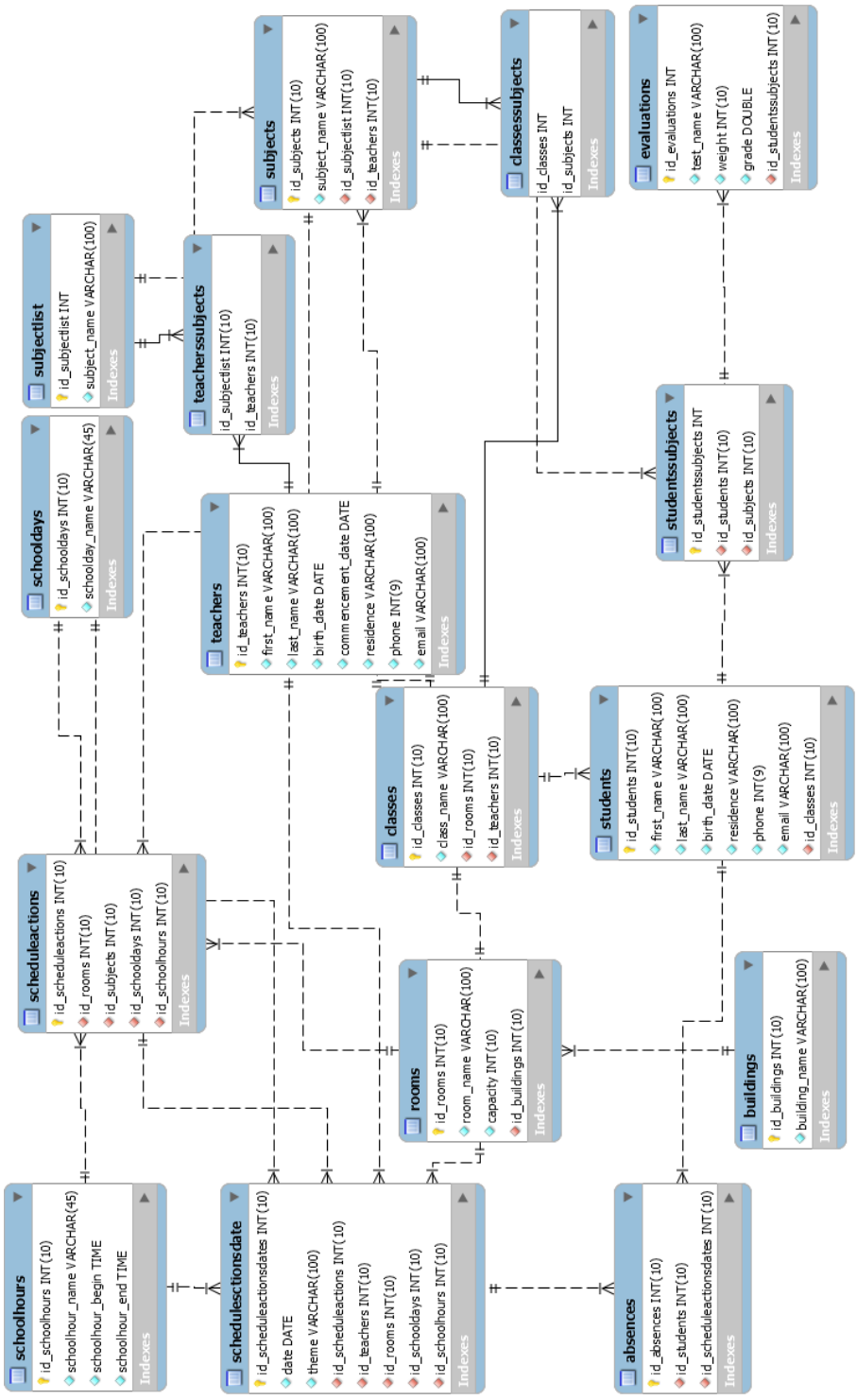
[18] *Working with JSON* [online]. 2021. [cit. 2021/04/29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

[19] *XML (eXtensible Markup Language) files* [online]. 2021. [cit. 2021/04/30]. Dostupné z: docs.memoq.com/current/en/Places/xml-extensible-markup-language.html.

[20] *XML introduction* [online]. 2021. [cit. 2021/04/29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction.

Přílohy

A ERA model databáze



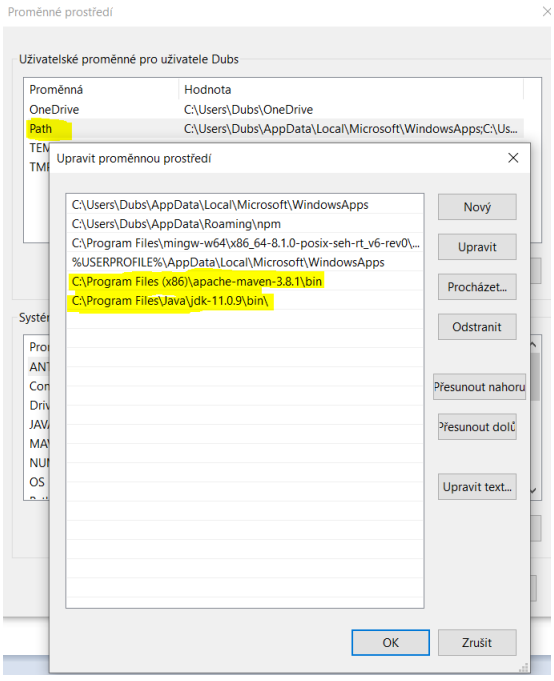
B Uživatelská příručka

Sestavení programu

Sestavení programu je otestováno pouze na systému Windows 10. Nejprve je potřeba nainstalovat *Oracle JDK* verze 11 pro překlad jazyka Java a také *Maven* verze 3.8.1 pro sestavení programu.

- Oracle JDK lze stáhnout po bezplatné registraci zde: *https://www.oracle.com/cz/java/technologies/javase-jdk11-downloads.html*
- Maven lze stáhnout zde: *https://maven.apache.org/download.cgi*

Po stažení a rozbalení je třeba přidat cesty k těmto nástrojům do systémové proměnné PATH (obrázek B.1)



Obrázek B.1: Nastavení systémové proměnné PATH

Sestavení programu nástrojem Maven se provede následovně:

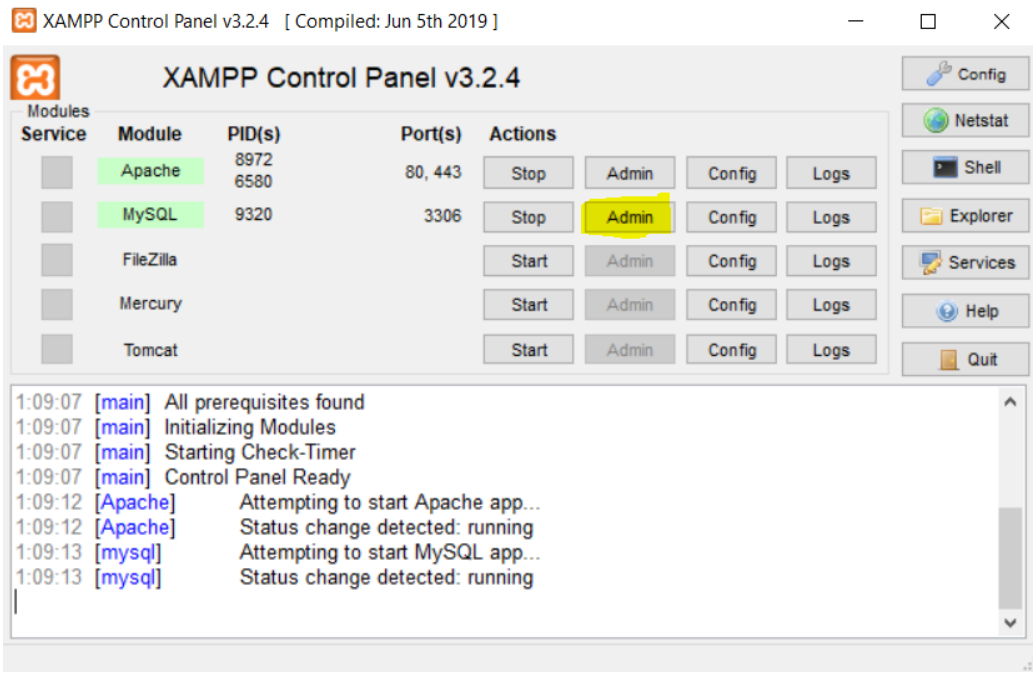
- nejprve je potřeba se přepnout do kořenového adresáře programu – *zdrojove_kody* (kde se nachází pom.xml)

- zadáme příkaz `mvn clean package`

Po dokončení příkazu se vytvoří spustitelný soubor *schoolagenda-final.jar* ve složce *target*.

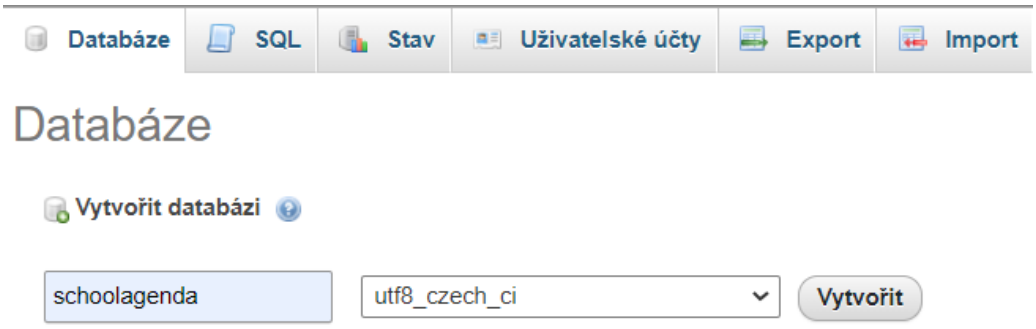
Spuštění programu

Pro spuštění programu je třeba nainstalovat volně dostupný softwarový balíček *xampp*, který obsahuje webový server *Apache* a databázi *MariaDB*. Dostupný ke stažení zde: *https://www.apachefriends.org/index.html*.



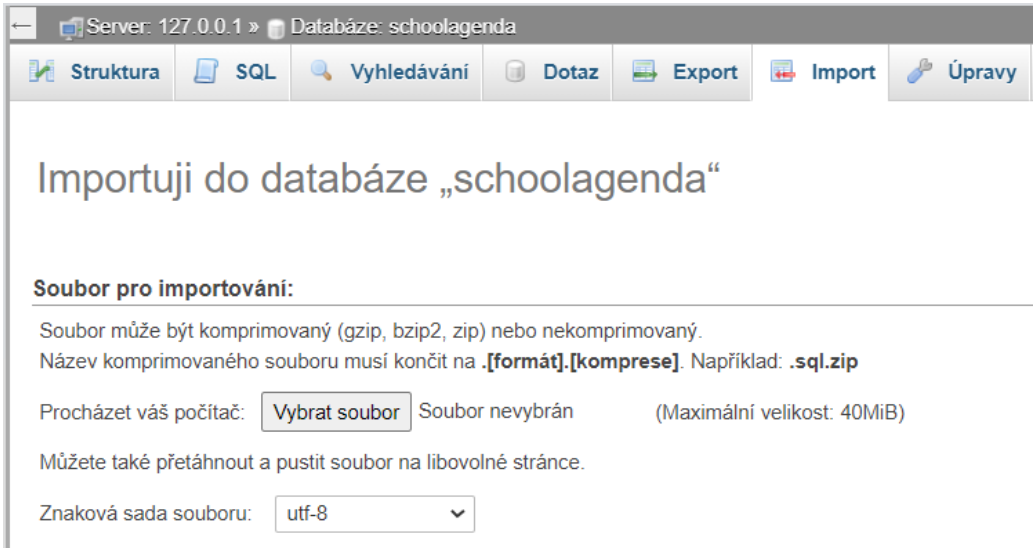
Obrázek B.2: Nastavení v programu xampp

Po jeho spuštění je třeba zapnout moduly *Apache* a *MySQL*, jak vidět na obrázku B.2. Poté kliknutím na tlačítko *Admin* v řádku modulu *MySQL* se dostaneme do webového prostředí *phpMyAdmin*.



Obrázek B.3: Vytvoření databáze v prostředí phpMyAdmin

Pak klikneme na záložku *Databáze*, dále do pole *Název databáze* napíšeme *schoolagenda* a z rozbalovací nabídky vybereme *utf8_czech_ci* (obrázek B.3). Poté klikneme na tlačítko *Vytvořit*, čímž se vytvoří nová databáze. Dále z nabídky databází v levé části klikneme na vytvořenou databázi *schoolagenda* a pak stiskneme na záložku *Import* z horní nabídky, jak je vidět na obrázku B.4.



Obrázek B.4: Import databáze

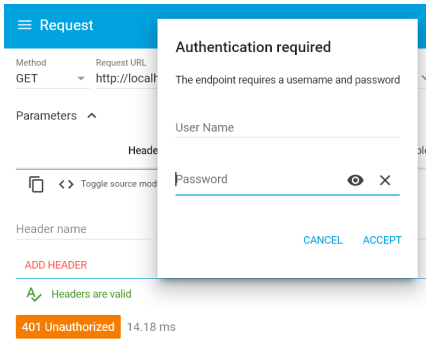
Nakonec stiskneme tlačítko *Vybrat soubor*, vybereme soubor se schématem databáze *schema.sql* ze složky *databaze* a stiskneme tlačítko *Proveď* v pravém dolním rohu. Tímto máme připravenou databázi.

Pro spuštění programu je také třeba nainstalovant Oracle JDK 11, jehož instalaci je již popsána. Program lze spustit rovnou ze složky *aplikace* nebo nejprve sestavit a poté spustit ze složky *zdrojove_kody/target*.

Po přesunutí do příslušné složky spustíme program příkazem `java -jar schoolagenda-final.jar`.

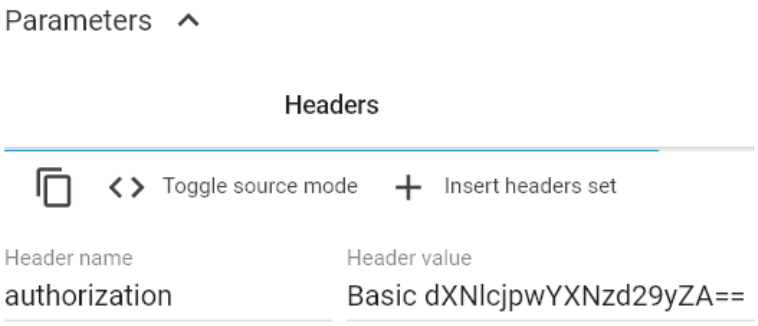
Používání aplikace

Přestože HTTP requesty je možné provádět z příkazového řádku pomocí příkazu curl, přívětivější je použít například volně dostupný *Advanced REST Client* dostupný zde jako rozšíření prohlížeče *Google Chrome*: *https://chrome.google.com/webstore/detail /advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo* nebo zde jako samostatná aplikace: *https://install.advancedrestclient.com/install*.



Obrázek B.5: Zadání ověřovacích údajů

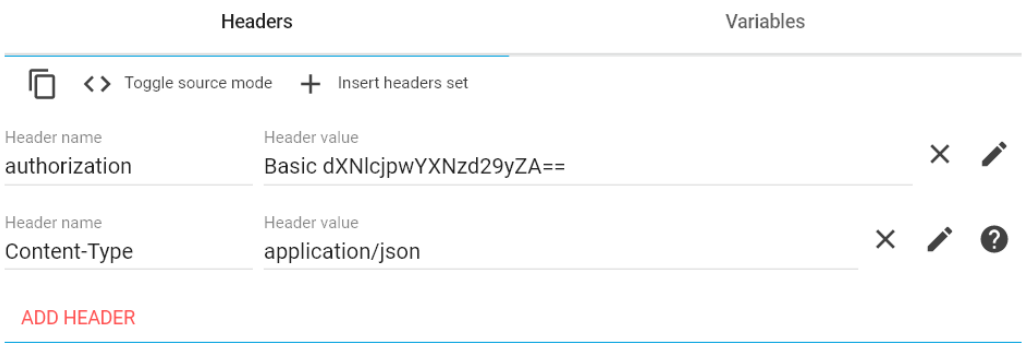
Adresa, na které aplikace běží je *http://localhost:8080*. Pokud pošleme první HTTP dotaz na tuto adresu, tak server požádá o přihlašovací údaje. Výchozí uživatelské jméno je *user* a výchozí heslo *password*.



Obrázek B.6: Zadání ověřovacích údajů

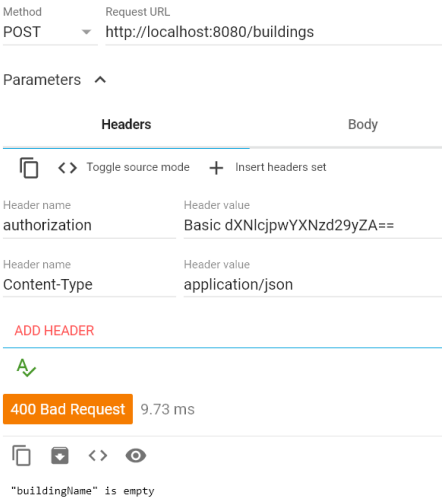
Po zadání správných údajů se vyplní hlavičce pole autorizace, jak je

na obrázku B.6 a od té doby lze pokládat dotazy bez nutnosti zadávání přihlašovacích údajů.



Obrázek B.7: Nastavení typu přenášených dat na JSON

Pro přenos dat v požadovaném formátu JSON je potřeba vyplnit atribut hlavičky `Content-Type` naplnit hodnotou `application/json` (obrázek B.7).



Obrázek B.8: POST dotaz s prázdným tělem

Pokud provedeme změnu typu metody na POST, pak lze naplnit tělo dotazu v záložce *Body*. I zde nastavíme atribut *Body content type* na hodnotu `application/json` pro zadávání dat ve formátu JSON. Pokud provedeme dotaz typu POST na adresu *http://localhost:8080/buildings* s prázdným JSON objektem, tak návratový kód odpovědi je 400 a v těle odpovědi je informace, že chybí atribut `buildingName`, jak je vidět na obrázku B.8. Pokud nastane chyba, tak aplikace se vždy snaží informovat, v čem je chyba.

Obrázek B.9: POST dotaz s vyplněným tělem

Pokud je JSON objekt správně vyplněn a dotaz zopakujeme, už je vše v pořádku a vytvoří se nová budova.

```
C:\Users\Dubs>curl --header "Content-Type:application/json" --request POST
 --data "{\"buildingName\":\"Budova X\"}" http://localhost:8080/buildings
--user user:password
```

Obrázek B.10: Příklad HTTP dotazu z příkazové řádky

Obdobně tento příkaz lze vykonat z příkazové řádky, jak je zobrazeno na obrázku B.10. Seznam všech validních dotazů je popsán na tabulkách B.1 až B.21.

Seznam validních dotazů

Adresy uvedené v tabulkách jsou suffixy za adresou *http://localhost:8080*.

Tabulka B.1: Seznam dotazů pro práci s absencemi

HTTP metoda	URI	Popis
GET	/absences	Vrací seznam všech záznamů absencí
GET	/absences/id/{id}	Vrací záznam absence s číslem, které je zadáno pomocí *id*
GET	/absences/student/id/{id}	Vrací seznam záznamů absencí podle zadaného čísla studenta *id*
GET	/absences/student/name/{firstName}/ {lastName}	Vrací seznam záznamů absencí podle zadaného křestního jména *firstName* a příjmení *lastName* studenta
GET	/absences/scheduleactiondate/id/{id}	Vrací seznam záznamů absencí podle zadaného čísla konkrétní rozvrhové akce *id*
POST	/absences	Vytvoří záznam absence na základě parametrů v požadavku
PUT	/absences/id/{id}	Upraví záznam absence s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/absences/id/{id}	Odstraní záznam absence s číslem, které je zadáno pomocí *id*

Tabulka B.2: Seznam dotazů pro práci s budovami

HTTP metoda	URI	Popis
GET	/buildings	Vrací seznam všech záznamů budov
GET	/buildings/id/{id}	Vrací záznam budovy s číslem, které je zadáno pomocí *id*
GET	/buildings/name/{name}	Vrací seznam záznamů budov podle zadaného jména *name*
GET	/buildings/room/id/{id}	Vrací záznam budovy podle zadaného čísla místnosti *id*
POST	/buildings	Vytvoří záznam budovy na základě parametrů v požadavku
PUT	/buildings/id/{id}	Upraví záznam budovy s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/buildings/id/{id}	Odstraní záznam budovy s číslem, které je zadáno pomocí *id*

Tabulka B.3: Seznam dotazů typu GET pro práci s třídami

HTTP metoda	URI	Popis
GET	/classes	Vrací seznam všech záznamů tříd
GET	/classes/id/{id}	Vrací záznam třídy s číslem, které je zadáno pomocí *id*
GET	/classes/name/{name}	Vrací seznam záznamů tříd podle zadaného jména *name*
GET	/classes/student/id/{id}	Vrací záznam třídy podle zadaného čísla studenta *id*
GET	/classes/student/name/ {firstName}{lastName}	Vrací záznam třídy podle zadaného čísla křestního jména *firstName* a příjmení *lastName* studenta
GET	/classes/teacher/id/{id}	Vrací záznam třídy podle zadaného čísla učitele *id*
GET	/classes/teacher/name/ {firstName}{lastName}	Vrací záznam třídy podle zadaného čísla křestního jména *firstName* a příjmení *lastName* učitele
GET	/classes/room/id/{id}	Vrací záznam třídy podle zadaného čísla místnosti *id*
GET	/classes/room/name/{name}	Vrací záznam třídy podle zadaného jména místnosti *name*
GET	/classes/subject/id/{id}	Vrací seznam záznamů tříd podle zadaného čísla konkrétního předmětu *id*
GET	/classes/subject/name/{name}	Vrací seznam záznamů tříd podle zadaného jména konkrétního předmětu *name*

Tabulka B.4: Seznam dotazů typu POST, UPDATE a DELETE pro práci s třídami

HTTP metoda	URI	Popis
POST	/classes	Vytvoří záznam třídy na základě parametrů v požadavku
PUT	/classes/id/{id}	Upraví záznam třídy s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
PUT	/classes/id/{id}/subject/add	Přidá třídě zadané pomocí čísla *id* konkrétní předmět, který je uveden v těle požadavku
PUT	/classes/id/{id}/subject/remove	Odstraní z třídy zadané pomocí čísla *id* konkrétní předmět, který je uveden v těle požadavku
DELETE	/classes/id/{id}	Odstraní záznam třídy s číslem, které je zadáno pomocí *id*

Tabulka B.5: Seznam dotazů pro práci s hodnocením

HTTP metoda	URI	Popis
GET	/evaluations	Vrací seznam všech záznamů hodnocení
GET	/evaluations/id/{id}	Vrací záznam hodnocení s číslem, které je zadáno pomocí *id*
GET	/evaluations/studentssubject/id/{id}	Vrací záznam hodnocení podle zadaného čísla předmětu studenta *id*
GET	/evaluations/average/studentssubject/ id/{id}	Vrací vážený průměr předmětu studenta zadaného pomocí čísla předmětu *id*
POST	/evaluations	Vytvoří záznam hodnocení na základě parametrů v požadavku
PUT	/evaluations/id/{id}	Upraví záznam hodnocení s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/evaluations/id/{id}	Odstraní záznam hodnocení s číslem, které je zadáno pomocí *id*

Tabulka B.6: Seznam dotazů typu GET pro práci s místnostmi

HTTP metoda	URI	Popis
GET	/rooms	Vrací seznam všech záznamů místností
GET	/rooms/id/{id}	Vrací záznam místnosti s číslem, které je zadáno pomocí *id*
GET	/rooms/name/{name}	Vrací seznam záznamů místností podle zadaného jména *name*
GET	/rooms/capacity/{capacity}	Vrací seznam záznamů místností podle zadané kapacity *capacity*
GET	/rooms/capacity/between/ {minCapacity}/{maxCapacity}	Vrací seznam záznamů místností, jejichž kapacita je v rozmezí *minCapacity* a *maxCapacity*
GET	/rooms/building/id/{id}	Vrací seznam záznamů místností podle zadaného čísla budovy *id*
GET	/rooms/building/name/{name}	Vrací seznam záznamů místností podle zadaného jména budovy *name*
GET	/rooms/class/id/{id}	Vrací záznam místnosti podle zadaného čísla třídy *id*
GET	/rooms/class/name/{name}	Vrací záznam místnosti podle zadaného jména třídy *name*

Tabulka B.7: Seznam dotazů typu POST, UPDATE a DELETE pro práci s místnostmi

HTTP metoda	URI	Popis
POST	/rooms	Vytvoří záznam místnosti na základě parametrů v požadavku
PUT	/rooms/id/{id}	Upraví záznam místnosti s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/rooms/id/{id}	Odstraní záznam místnosti s číslem, které je zadáno pomocí *id*

Tabulka B.8: Seznam dotazů pro práci s pravidelnými rozvrhovými akcemi

HTTP metoda	URI	Popis
GET	/scheduleactions	Vrací seznam všech záznamů pravidelných rozvrhových akcí
GET	/scheduleactions/id/{id}	Vrací záznam pravidelné rozvrhové akce s číslem, které je zadáno pomocí *id*
GET	/scheduleactions/room/id/{id}	Vrací seznam záznamů pravidelných rozvrhových akcí podle zadaného čísla místnosti *id*
GET	/scheduleactions/room/name/{name}	Vrací seznam záznamů pravidelných rozvrhových akcí podle zadaného jména místnosti *name*
GET	/scheduleactions/subject/id/{id}	Vrací seznam záznamů pravidelných rozvrhových akcí podle zadaného čísla konkrétního předmětu *id*
GET	/scheduleactions/subject/name/{name}	Vrací seznam záznamů pravidelných rozvrhových akcí podle zadaného jména konkrétního předmětu *name*
POST	/scheduleactions	Vytvoří záznam pravidelné rozvrhové akce na základě parametrů v požadavku
PUT	/scheduleactions/id/{id}	Upraví záznam pravidelné rozvrhové akce s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/scheduleactions/id/{id}	Odstraní záznam pravidelné rozvrhové akce s číslem, které je zadáno pomocí *id*

Tabulka B.9: Seznam dotazů pro práci s konkrétními rozvrhovými akcemi

HTTP metoda	URI	Popis
GET	/scheduleactionsdates	Vrací seznam všech záznamů konkrétních rozvrhových akcí
GET	/scheduleactionsdates/id/{id}	Vrací záznam konkrétní rozvrhové akce s číslem, které je zadáno pomocí *id*
GET	/scheduleactionsdates/subject/id/{id}	Vrací seznam záznamů konkrétních rozvrhových akcí podle zadaného čísla konkrétního předmětu *id*
GET	/scheduleactionsdates/substitutions	Vrací seznam záznamů konkrétních rozvrhových akcí, které jsou suplování
GET	/scheduleactionsdates/substitutions/ date/{date}	Vrací seznam záznamů konkrétních rozvrhových akcí podle datumu *date*, které jsou suplování
GET	/scheduleactionsdates/substitutions/ date/{date}/teacher/id/{id}	Vrací seznam záznamů konkrétních rozvrhových akcí podle datumu *date* a čísla učitele *id*, které jsou suplování
GET	/scheduleactionsdates/substitutions/ date/{date}/student/id/{id}	Vrací seznam záznamů konkrétních rozvrhových akcí podle datumu *date* a čísla studenta *id*, které jsou suplování
POST	/scheduleactionsdates	Vytvoří záznam konkrétní rozvrhové akce na základě parametrů v požadavku
DELETE	/scheduleactionsdates/id/{id}	Odstraní záznam konkrétní rozvrhové akce s číslem, které je zadáno pomocí *id*

Tabulka B.10: Seznam dotazů pro práci se školními dny

HTTP metoda	URI	Popis
GET	/schooldays	Vrací seznam všech záznamů školních dnů
GET	/schooldays/id/{id}	Vrací záznam školního dne s číslem, které je zadáno pomocí *id*
GET	/schooldays/name/{name}	Vrací seznam záznamů školních dnů podle zadaného jména *name*
GET	/schooldays/scheduleaction/id/{id}	Vrací záznam školního dne podle zadaného čísla pravidelné rozvrhové akce *id*
GET	/schooldays/scheduleactionsdate/id/{id}	Vrací záznam školního dne podle zadaného čísla konkrétní rozvrhové akce *id*
POST	/schooldays	Vytvoří záznam školního dne na základě parametrů v požadavku
PUT	/schooldays/id/{id}	Upraví záznam školního dne s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/schooldays/id/{id}	Odstraní záznam školního dne s číslem, které je zadáno pomocí *id*

Tabulka B.11: Seznam dotazů pro práci se školními hodinami

HTTP metoda	URI	Popis
GET	/schoolhours	Vrací seznam všech záznamů školních hodin
GET	/schoolhours/id/{id}	Vrací záznam školní hodiny s číslem, které je zadáno pomocí *id*
GET	/schoolhours/name/{name}	Vrací seznam záznamů školních hodin podle zadaného jména *name*
GET	/schoolhours/scheduleaction/id/{id}	Vrací záznam školní hodiny podle zadaného čísla pravidelné rozvrhové akce *id*
GET	/schoolhours/scheduleactionsdate/id/{id}	Vrací záznam školní hodiny podle zadaného čísla konkrétní rozvrhové akce *id*
POST	/schoolhours	Vytvoří záznam školní hodiny na základě parametrů v požadavku
PUT	/schoolhours/id/{id}	Upraví záznam školní hodiny s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/schoolhours/id/{id}	Odstraní záznam školní hodiny s číslem, které je zadáno pomocí *id*

Tabulka B.12: Seznam dotazů typu GET pro práci se studenty

HTTP metoda	URI	Popis
GET	/students	Vrací seznam všech záznamů studentů
GET	/students/id/{id}	Vrací záznam studenta s číslem, které je zadáno pomocí *id*
GET	/students/name/{firstName}{lastName}	Vrací seznam záznamů studentů podle zadaného křestního jména *firstName* a příjmení *lastName*
GET	/students/class/id/{id}	Vrací seznam záznamů studentů podle zadaného čísla třídy *id*
GET	/students/class/name/{name}	Vrací seznam záznamů studentů podle zadaného jména třídy *name*
GET	/students/subject/id/{id}	Vrací seznam záznamů studentů podle zadaného čísla konkrétního předmětu *id*
GET	/students/subject/name/{name}	Vrací seznam záznamů studentů podle zadaného jména konkrétního předmětu *name*

Tabulka B.13: Seznam dotazů typu POST, PUT a DELETE pro práci se studenty

HTTP metoda	URI	Popis
POST	/students	Vytvoří záznam studenta na základě parametrů v požadavku
PUT	/students/id/{id}	Upraví záznam studenta s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
PUT	/students/id/{id}/subject/add	Přidá studentovi zadaném pomocí čísla *id* konkrétní předmět, který je uveden v těle požadavku
PUT	/students/id/{id}/subject/remove	Odstraní studentovi zadaném pomocí čísla *id* konkrétní předmět, který je uveden v těle požadavku
DELETE	/students/id/{id}	Odstraní záznam studenta s číslem, které je zadáno pomocí *id*

Tabulka B.14: Seznam dotazů pro práci s předměty studenta

HTTP metoda	URI	Popis
GET	/studentssubjects	Vrací seznam všech záznamů předmětů studentů
GET	/studentssubjects/id/{id}	Vrací záznam předmětu studenta s číslem, který je zadán pomocí *id*
GET	/studentssubjects/student/id/{idStudents}/ subject/id/{idSubjects}	Vrací záznam předmětu studenta podle čísla studenta *idStudents* a čísla konkrétního předmětu *idSubjects*

Tabulka B.15: Seznam dotazů typu GET pro práci s konkrétními předměty

HTTP metoda	URI	Popis
GET	/subjects	Vrací seznam všech záznamů konkrétních předmětů
GET	/subjects/id/{id}	Vrací záznam konkrétního předmětu s číslem, které je zadáno pomocí *id*
GET	/subjects/name/{name}	Vrací seznam záznamů konkrétních předmětů podle zadaného jména *name*
GET	/subjects/subjectlist/id/{id}	Vrací seznam záznamů konkrétních předmětů podle zadaného čísla obecného předmětu *id*
GET	/subjects/subjectlist/name/{name}	Vrací seznam záznamů konkrétních předmětů podle zadaného jména obecného předmětu *name*
GET	/subjects/class/id/{id}	Vrací seznam záznamů konkrétních předmětů podle zadaného čísla třídy *id*
GET	/subjects/class/name/{name}	Vrací seznam záznamů konkrétních předmětů podle zadaného jména třídy *name*
GET	/subjects/scheduleaction/id/{id}	Vrací záznam konkrétního předmětu podle zadaného čísla pravidelné rozvrhové akce *id*

Tabulka B.16: Seznam dotazů typu GET pro získání konkrétních předmětů pomocí učitele a studenta

HTTP metoda	URI	Popis
GET	/subjects/teacher/id/{id}	Vrací záznam konkrétního předmětu podle zadaného čísla učitele *id*
GET	/subjects/teacher/name/ {firstName}{lastName}	Vrací záznam konkrétního předmětu podle zadaného křestního jména *firstName* a příjmení *lastName* učitele
GET	/subjects/student/id/{id}	Vrací seznam záznamů konkrétních předmětů podle zadaného čísla studenta *id*
GET	/subjects/student/name/ {firstName}{lastName}	Vrací seznam záznamů konkrétních předmětů podle zadaného křestního jména *firstName* a příjmení *lastName* studenta

Tabulka B.17: Seznam dotazů typu POST, UPDATE a DELETE pro práci s konkrétními předměty

HTTP metoda	URI	Popis
POST	/subjects	Vytvoří záznam konkrétního předmětu na základě parametrů v požadavku
PUT	/subjects/id/{id}	Upraví záznam konkrétního předmětu s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/classes/id/{id}	Odstraní záznam konkrétního předmětu s číslem, které je zadáno pomocí *id*

Tabulka B.18: Seznam dotazů typu GET pro práci s obecnými předměty

HTTP metoda	URI	Popis
GET	/subjectlists	Vrací seznam všech záznamů obecných předmětů
GET	/subjectlists/id/{id}	Vrací záznam obecného předmětu s číslem, které je zadáno pomocí *id*
GET	/subjectlists/name/{name}	Vrací seznam záznamů obecných předmětů podle zadaného jména *name*
GET	/subjectlists/subject/id/{id}	Vrací záznam obecného předmětu podle zadaného čísla konkrétního předmětu *id*
GET	/subjectlists/subject/name/{name}	Vrací záznam obecného předmětu podle zadaného jména konkrétního předmětu *name*
GET	/subjects/teacher/id/{id}	Vrací seznam záznamů obecných předmětů podle zadaného čísla učitele *id*
GET	/subjects/teacher/name/ {firstName}{lastName}	Vrací seznam záznamů obecných předmětů podle zadaného křestního jména *firstName* a příjmení *lastName* učitele

Tabulka B.19: Seznam dotazů typu POST, UPDATE a DELETE pro práci s obecnými předměty

HTTP metoda	URI	Popis
POST	/subjectlists	Vytvoří záznam obecného předmětu na základě parametrů v požadavku
PUT	/subjectlists/id/{id}	Upraví záznam obecného předmětu s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
DELETE	/subjectlists/id/{id}	Odstraní záznam obecného předmětu s číslem, které je zadáno pomocí *id*

Tabulka B.20: Seznam dotazů typu GET pro práci s učiteli

HTTP metoda	URI	Popis
GET	/teachers	Vrací seznam všech záznamů učitelů
GET	/teachers/id/{id}	Vrací záznam učitele s číslem, které je zadáno pomocí *id*
GET	/teachers/name/{firstName}{lastName}	Vrací seznam záznamů učitelů podle zadaného křestního jména *firstName* a příjmení *lastName*
GET	/teachers/subjectlist/id/{id}	Vrací seznam záznamů učitelů podle zadaného čísla obecného předmětu *id*
GET	/teachers/subjectlist/name/{name}	Vrací seznam záznamů učitelů podle zadaného jména obecného předmětu *name*
GET	/teachers/subject/id/{id}	Vrací záznam učitele podle zadaného čísla konkrétního předmětu *id*
GET	/teachers/subject/name/{name}	Vrací záznam učitele podle zadaného jména konkrétního předmětu *name*

Tabulka B.21: Seznam dotazů typu POST, PUT a DELETE pro práci s učiteli

HTTP metoda	URI	Popis
POST	/teachers	Vytvoří záznam učitele na základě parametrů v požadavku
PUT	/teachers/id/{id}	Upraví záznam učitele s číslem, které je zadáno pomocí *id*, záznam je upraven podle parametrů v požadavku
PUT	/teachers/id/{id}/subjectlist/add	Přidá učiteli zadaného pomocí čísla *id* obecný předmět, který je uveden v těle požadavku
PUT	/teachers/id/{id}/subjectlist/remove	Odstraní učiteli zadaného pomocí čísla *id* obecný předmět, který je uveden v těle požadavku
DELETE	/teachers/id/{id}	Odstraní záznam učitele s číslem, které je zadáno pomocí *id*

C Seznam testovacích scénářů

Pro všechny testovací scénáře kromě TS01 se předpokládá vyplnění Basic HTTP Authorization, kde uživatelské jméno je *user* a heslo *password*.

TS01: Nezadané ověření

Popis: Zadejte HTTP požadavek typu GET s adresou.
http://localhost:8080/buildings, Basic HTTP Authorization v požadavku odstraňte
Očekávaný výsledek: Stavový kód odpovědi je 401.

TS02: Zadané ověření

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/buildings s Basic HTTP Authorization, kde uživatelské jméno je *user* a heslo *password*.
Očekávaný výsledek: Stavový kód odpovědi není 401.

TS03: Vytvoření budovy s chybějícím jménem

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/buildings, tělo JSON objektu nechte prázdné.
Očekávaný výsledek: Stavový kód odpovědi je 400, v těle odpovědi oznámení *"buildingName"is empty.*

TS04: Správné vytvoření budovy

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/buildings, v těle požadavku zadejte atribut JSON objectu s názvem *buildingName* a hodnotou *Budova A*.
Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi je vytvořená budova.

TS05: Výpis všech budov

Popis: Zadejte stejný dotaz jako v TS04, ale s hodnotou atributu *buildingName Budova B*. Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/buildings

Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi je seznam budov již vytvořených budov.

TS06: Výpis budovy podle id – budova neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/buildings/id/100000*
Očekávaný výsledek: Stavový kód odpovědi je 422, v těle odpovědi oznámení *Building with id 100000 does not exist.*

TS07: Výpis budovy podle id – budova existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/buildings/id/hodnota*, kde hodnota je číslo *idBuildings* budovy vytvořené v TS05.
Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi budova, jejíž hodnota *idBuildings* je stejná jako hodnota zadávaná v dotazu.

TS08: Výpis budovy podle jména – budova neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/buildings/name/Budova X.*
Očekávaný výsledek: Stavový kód odpovědi je 422, v těle odpovědi oznámení *Building with name Budova X does not exist.*

TS09: Výpis budovy podle jména – budova existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/buildings/name/Budova A*
Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi budova, jejíž hodnota *buildingName* je *Budova A*.

TS10: Úprava budovy

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/buildings/id/hodnota*, kde hodnota je číslo *idBuildings* jedné budovy vytvořené v TS04, v těle požadavku zadejte atribut JSON objectu s názvem *buildingName* a hodnotou *Budova C.*
Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi je budova, jejíž hodnota *idBuildings* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *buildingName* je Budova C.

TS11: Smazání budovy

Popis: Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/buildings/id/hodnota*, kde hodnota je číslo *idBuildings* budovy s názvem Budova C.
Očekávaný výsledek: Stavový kód odpovědi je 200, v těle odpovědi je oznámení *Building Budova C deleted.*

TS12: Vytvoření místnosti - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/rooms*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"roomName"is empty*

TS13: Správné vytvoření místnosti

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/rooms* a v těle požadavku zadejte atribut JSON objectu s názvem *roomName* a hodnotou *Místnost A* a object *building* s atributem *buildingName* a hodnotou *Budova A*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená místnost

TS14: Výpis všech místností

Popis: Zadejte stejný dotaz jako v TS13, ale s hodnotou atributu *roomName Místnost B*, poté zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/rooms*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je seznam již vytvořených místností

TS15: Výpis místnosti podle id – místnost neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/rooms/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Room with id 100000 does not exist.*

TS16: Výpis místnosti podle id – místnost existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/rooms/id/hodnota*, kde hodnota je číslo *idRooms* místnosti vytvořeného v TS14
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi místnost, jejíž hodnota *idRooms* je stejná jako hodnota zadávaná v dotazu

TS17: Výpis místnosti podle jména – místnost neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/rooms/name/Místnost X*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Room with name Místnost X does not exist.*

TS18: Výpis místnosti podle jména – místnost existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/rooms/name/Místnost A*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi místnost, jejíž hodnota *roomName* je Místnost A

TS19: Úprava místnosti

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/rooms/id/hodnota*, kde hodnota je číslo *idRooms* místnosti vytvořené v TS14, v těle požadavku zadejte atribut JSON objectu s názvem *roomName* a hodnotou *Místnost C*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je místnost, jejíž hodnota *idRooms* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *roomName* je Místnost C

TS20: Smazání místnosti

Popis: Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/rooms/id/hodnota*, kde hodnota je číslo *idRooms* místnosti s názvem Místnost C.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Room Místnost C deleted*

TS21: Vytvoření obecného předmětu - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/subjectlists*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"subjectName"is empty*

TS22: Správné vytvoření obecného předmětu

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/subjectlists* a v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *Matematika*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořený obecný předmět

TS23: Výpis všech obecných předmětů

Popis: Zadejte stejný dotaz jako v TS22, ale s hodnotou atributu *subjectName Fyzika*, poté zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjectlists*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je seznam již vytvořených obecných předmětů

TS24: Výpis obecného předmětu podle id – obecný předmět neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjectlists/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Subject list with id 100000 does not exist.*

TS25: Výpis obecného předmětu podle id – obecný předmět existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjectlists/id/hodnota*, kde hodnota je číslo *idSubjectlist* je je číslo obecného předmětu vytvořeného v TS23
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi obecný předmět, jehož hodnota *idSubjectlist* je stejná jako hodnota zadávaná v dotazu

TS26: Výpis obecného předmětu podle jména – obecný předmět neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjectlists/name/Prvouka*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Subject list with name Prvouka does not exist.*

TS27: Výpis obecného předmětu podle jména – obecný předmět existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjectlists/name/Matematika*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi obecný předmět, jehož hodnota *subjectName* je Matematika

TS28: Úprava obecného předmětu

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/subjectlists/id/hodnota*, kde hodnota je číslo *idSubjectlist* obecného předmětu vytvořeného v TS23, v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *Prvouka*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je obecný předmět, jehož hodnota *idSubjectlist* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *subjectName* je *Prvouka*.

TS29: Smazání obecného předmětu

Popis: Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/subjectlists/id/hodnota*, kde hodnota je číslo *idSubjectlist* obecného předmětu s názvem Prvouka.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Subject list Prvouka deleted*

TS30: Vytvoření učitele - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/teachers*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"firstName"is empty*

TS31: Správné vytvoření učitele

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/teachers* a v těle požadavku zadejte atribut JSON objectu s názvem *firstName* a hodnotou *Jan*, *lastname* s hodnotou *Novák*, *birthDate* s hodnotou *1980-05-06*, *commencementDate* s hodnotou *2021-03-01*, *residence* s hodnotou *Praha*, *phone* s hodnotou *"123456789"* a *email* s hodnotou *novakj@seznam.cz*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořený učitel

TS32: Výpis všech učitelů

Popis: Zadejte stejný dotaz jako v TS31, ale s hodnotou atributu *firstName* *"Josef"*, poté zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/teachers*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je seznam již vytvořených učitelů

TS33: Výpis učitele podle id – učitel neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/teachers/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Teacher with id 100000 does not exist.*

TS34: Výpis učitele podle id – učitel existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/teachers/id/hodnota*, kde hodnota je číslo *idTeachers* učitele vytvořeného v TS32
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi učitel, jehož hodnota *idTeachers* je stejná jako hodnota zadávaná v dotazu

TS35: Výpis učitele podle jména – učitel neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/teachers/name/Daniel/Zelený*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Teacher Daniel Zelený does not exist.*

TS36: Výpis učitele podle jména – učitel existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/teachers/name/Jan/Novák*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi učitel, jehož hodnota *firstName* je *Jan* a *lastName* je *Novák*.

TS37: Úprava učitele

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/teachers/id/hodnota*, kde hodnota je číslo *idTeachers* učitele vytvořené v TS32, v těle požadavku zadejte atribut JSON objectu s názvem *phone* a hodnotou *987654321*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je učitel, jejíž hodnota *idTeachers* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *phone* je *987654321*.

TS38: Smazání učitele

Popis: Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/teachers/id/hodnota*, kde hodnota je číslo *idTeachers* učitele se jménem *Josef Novák*.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Teacher Josef Novák deleted*

TS39: Vytvoření konkrétního předmětu - učitel nemůže vyučovat předmět

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/subjects* a v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *"Matematika 9.Třída"*, objekt teacher s atributy *firstName* s hodnotou *"Jan"* a *lastName* s hodnotou *"Novák"*, objekt subjectlist s atributem *subjectName* s hodnotou *"Matematika"*
Očekávaný výsledek: stavový kód odpovědi je 409 a v těle odpovědi oznámení *Jan Novák cannot teach Matematika*

TS40: Přidání obecného předmětu učiteli

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/teachers/id/hodnota/subjectlist/add* a v těle požadavku

zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *"Matematika"*
Očekávaný výsledek: stavový kód odpovědi je 200

TS41: Správné vytvoření konkrétního předmětu

Popis: Zopakujte TS39
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořený konkrétní předmět

TS42: Výpis všech konkrétních předmětů

Popis: Zadejte stejný dotaz jako v TS39, ale s hodnotou atributu *subjectName* *Maatematika 8.Třída*, poté zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjects*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je seznam již vytvořených konkrétních předmětů

TS43: Výpis konkrétního předmětu podle id – konkrétní předmět neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjects/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Subject with id 100000 does not exist.*

TS44: Výpis konkrétního předmětu podle id – konkrétní předmět existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjects/id/hodnota*, kde hodnota je číslo *idSubjects* konkrétního předmětu vytvořeného v TS42
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi konkrétní předmět, jehož hodnota *idSubjects* je stejná jako hodnota zadávaná v dotazu

TS45: Výpis konkrétního předmětu podle jména – konkrétní předmět neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjects/name/Matematika 1.Třída*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Subject with name Matematika 1.Třída does not exist.*

TS46: Výpis konkrétního předmětu podle jména – konkrétní předmět existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/subjects/name/Matematika 9.Třída*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi konkrétní předmět, jehož hodnota *subjectName* je *Matematika 9.Třída.*

TS47: Úprava konkrétního předmětu

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/subjects/id/hodnota*, kde hodnota je číslo *idSubjects* konkrétního předmětu vytvořeného v TS42, v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *Matematika 1.Třída*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je konkrétní předmět, jehož hodnota *idSubjects* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *subjectName* je *Matematika 1.Třída.*

TS48: Smazání konkrétního předmětu

Popis: Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/subjects/id/hodnota*, kde hodnota je číslo *idSubjects* konkrétního předmětu se jménem *Matematika 1.Třída.*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Subject Matematika 1.třída deleted*

TS49: Vytvoření třídy - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/classes*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"className"is empty*

TS50: Správné vytvoření třídy

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/classes* a v těle požadavku zadejte atribut JSON objectu s názvem *className* a hodnotou *9.A*, objekt *teacher* s atributy *firstName* s hodnotou *"Jan"* a *lastname* s hodnotou *"Novák"*, objekt *room* s atributem *roomName* s hodnotou *"Místnost A"*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená třída

TS51: Výpis všech tříd

Popis: zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/classes*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořená třída

TS52: Výpis třídy podle id – třída neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/classes/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Class with id 100000 does not exist.*

TS53: Výpis třídy podle id – třída existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/classes/id/hodnota*, kde hodnota je číslo *idClasses* třídy vytvořené v TS50
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi třída, jejíž hodnota *idClasses* je stejná jako hodnota zadávaná v dotazu

TS54: Výpis třídy podle jména – třída neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/classes/name/9.B*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Class with name 9.B does not exist.*

TS55: Výpis třídy podle jména – třída existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/classes/name/9.A
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi třída, jejíž hodnota *className* je *9.A*

TS56: Úprava třídy

Popis: Zadejte HTTP požadavek typu PUT s adresou
http://localhost:8080/classes/id/hodnota, kde hodnota je číslo *idClasses* třídy vytvořené v TS50, v těle požadavku zadejte atribut JSON objectu s názvem *className* a hodnotou *9.B*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je třída, jejíž hodnota *idClasses* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *className* je *9.B*.

TS57: Smazání třídy

Popis: Zadejte HTTP požadavek typu DELETE s adresou
http://localhost:8080/classes/id/hodnota, kde hodnota je číslo *idClasses* třídy s názvem *9.B*.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Class 9.B deleted*

TS58: Vytvoření studenta - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/students, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"firstName"is empty*

TS59: Správné vytvoření studenta

Popis: Zopakujte TS50, poté zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/students a v těle požadavku zadejte atribut JSON objectu s názvem *firstName* s hodnotou*David*, *lastName* s hodnotou *Krátký*, *birthDate* s hodnotou *2010-05-30*, *residence* s hodnotou *Praha*, *phone* s hodnotou *123123123*, *email* s hodnotou *kratkyd@gmail.com*, objekt *classesEntity* s atributem *idClasses* s hodnotou *"idClasses"* vytvořené třídy v tomto TS.

Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořený student

TS60: Výpis všech studentů

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/students
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořený student

TS61: Výpis studenta podle id – student neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/students/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Student with id 100000 does not exist.*

TS62: Výpis studenta podle id – student existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/students/id/hodnota, kde hodnota je číslo *idStudents* studenta vytvořeného v TS59
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi student, jehož hodnota *idStudents* je stejná jako hodnota zadávaná v dotazu

TS63: Výpis studenta podle jména – student neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/students/name/Karel/Krátký
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Student Karel Krátký does not exist.*

TS64: Výpis studenta podle jména – student existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/students/name/David/Krátký
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi student, jehož hodnota *firstName* je *David* a *lastName* je *Krátký*

TS65: Úprava studenta

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/students/id/hodnota*, kde hodnota je číslo *idStudents* studenta vytvořeného v TS59, v těle požadavku zadejte atribut JSON objectu s názvem *lastName* a hodnotou *Dlouhý*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je student, jehož hodnota *idStudents* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *lastName* je *Dlouhý*.

TS66: Přidání konkrétního předmětu třídě

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/classes/id/hodnota/subject/add*, kde hodnota je číslo *idClasses* třídy vytvořené v TS59, v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *Matematika 9.Třída*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je třída, jejíž hodnota *idClasses* je stejná jako hodnota zadávaná v dotazu a také objekt subjects s atributem *subjectName* a hodnotou *Matematika 9.Třída*.

TS67: Přidání konkrétního předmětu studentovi

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/students/id/hodnota/subject/add*, kde hodnota je číslo *idStudents* studenta vytvořeného v TS59, v těle požadavku zadejte atribut JSON objectu s názvem *subjectName* a hodnotou *Matematika 9.Třída*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je student, jehož hodnota *idStudents* je stejná jako hodnota zadávaná v dotazu a také objekt studentssubjects s atributem *idStudentssubjects*.

TS68: Smazání studenta

Popis: Zopakujte TS59, ale s hodnotou atributu *firstName Pavel* a *lastName* s hodnotou *Široký*. Poté zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/students/id/hodnota*, kde hodnota je číslo *idStudents* studenta vytvořeného v tomto TS.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Student Pavel Široký deleted.*

TS69: Vytvoření hodnocení - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/evaluations*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"testName"is empty*

TS70: Správné vytvoření hodnocení

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/evaluations* a v těle požadavku zadejte atribut JSON objectu s názvem *testName* s hodnotou *Násobení*, *grade* s hodnotou *2*, *weight* s hodnotou *5*, objekt *studentssubject* s atributem *idStudentssubjects* s hodnotou atributu *idStudentssubjects* objektu *studentssubjects* studenta vytvořeného v TS59.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořené hodnocení

TS71: Výpis všech hodnocení

Popis: zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/evaluations*
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořené hodnocení

TS72: Výpis hodnocení podle id – hodnocení neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/evaluations/id/100000*
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Evaluation with id 100000 does not exist.*

TS73: Výpis hodnocení podle id – hodnocení existuje

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/evaluations/id/hodnota*, kde hodnota je číslo *idEvaluations* hodnocení vytvořeného v TS70
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi hodnocení, jehož hodnota *idEvaluations* je stejná jako hodnota zadávaná v dotazu

TS74: Výpis váženého průměru hodnocení podle čísla studentova předmětu

Popis: Zadejte HTTP požadavek typu GET s adresou *http://localhost:8080/evaluations/average/studentssubject/id/hodnota*, kde hodnota je atribut *idStudentssubjects* studenta vytvořeného v TS59
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi oznámení *Average of this students subject is 2.0.*

TS75: Úprava hodnocení

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/evaluations/id/hodnota*, kde hodnota je číslo *idEvaluations* hodnocení vytvořeného v TS70, v těle požadavku zadejte atribut JSON objectu s názvem *grade* a hodnotou *3*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je hodnocení, jehož hodnota *idEvaluations* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *grade* je *3*.

TS76: Smazání hodnocení

Popis: Zopakujte TS70, ale s hodnotou atributu *testName Dělení*. Poté zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/evaluations/id/hodnota*, kde hodnota je číslo *idEvaluations* hodnocení vytvořeného v tomto TS.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Evaluation with id hodnota deleted*, kde hodnota je číslo *idEvaluations* hodnocení vytvořeného v tomto TS.

TS77: Vytvoření školního dne - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/schooldays*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"schooldayName"is empty*

TS78: Správné vytvoření školního dne

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/schooldays* a v těle požadavku zadejte atribut JSON objectu s názvem *schooldayName* a hodnotou *Pondělí*.

Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořený školní den

TS79: Výpis všech školních dnů

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schooldays
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořený školní den

TS80: Výpis školního dne podle id – školní den neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schooldays/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schoolday with id 100000 does not exist.*

TS81: Výpis školního dne podle id – školní den existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schooldays/id/hodnota, kde hodnota je číslo *idSchooldays* školního dne vytvořeného v TS78
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi školní den, jehož hodnota *idSchooldays* je stejná jako hodnota zadávaná v dotazu

TS82: Výpis školního dne podle jména – školní den neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schooldays/name/Sobota
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schoolday with name Sobota does not exist.*

TS83: Výpis školního dne podle jména – školní den existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schooldays/name/Pondělí

Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi školní den, jehož hodnota *schooldayName* je *Pondělí*

TS84: Úprava školního dne

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/schooldays/id/hodnota*, kde hodnota je číslo *idSchooldays* školního dne vytvořeného v TS78, v těle požadavku zadejte atribut JSON objectu s názvem *schooldayName* a hodnotou *Úterý*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je školní den, jehož hodnota *idSchooldays* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *schooldayName* je *Úterý*.

TS85: Smazání školního dne

Popis: Zopakujte TS78, ale s hdnotou atributu *schooldayName* a hodnotou *Pátek*. Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/schooldays/id/hodnota*, kde hodnota je číslo *idSchooldays* školního dne vytvořeného v tomto TS.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Schoolday Pátek deleted*

TS86: Vytvoření školní hodiny - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/schoolhours*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"schoolhourName"is empty*

TS87: Správné vytvoření školní hodiny

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/schoolhours* a v těle požadavku zadejte atribut JSON objectu s názvem *schoolhourName* a hodnotou *Hodina 1*, *schoolhourBegin* a hodnotou *08:00:00*, *schoolhourEnd* a hodnotou *08:45:00*.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená školní hodina

TS88: Výpis všech školních hodin

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schoolhours
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořená školní hodina

TS89: Výpis školní hodiny podle id – školní hodina neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schoolhours/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schoolhour with id 100000 does not exist.*

TS90: Výpis školní hodiny podle id – školní hodina existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schoolhours/id/hodnota, kde hodnota je číslo *idSchoolhours* školního dne vytvořeného v TS87
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi školní hodina, jejíž hodnota *idSchoolhours* je stejná jako hodnota zadávaná v dotazu

TS91: Výpis školní hodiny podle jména – školní hodina neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schoolhours/name/Hodina 2
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schoolhour with name Hodina 2 does not exist.*

TS92: Výpis školní hodiny podle jména – školní hodina existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/schoolhours/name/Hodina 1
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi školní hodina, jejíž hodnota *schoolhourName* je *Hodina 1*

TS93: Úprava školní hodiny

Popis: Zadejte HTTP požadavek typu PUT s adresou *http://localhost:8080/schoolhours/id/hodnota*, kde hodnota je číslo *idSchoolhours* školního dne vytvořeného v TS87, v těle požadavku zadejte atribut JSON objectu s názvem *schoolhourName* a hodnotou *Hodina 2*
Očekávaný výsledek: stavový kód odpovědi je 200, v těle odpovědi je školní hodina, jejíž hodnota *idSchoolhours* je stejná jako hodnota zadávaná v dotazu a hodnota atributu *schoolhourName* je *Hodina 2*.

TS94: Smazání školní hodiny

Popis: Zopakujte TS87, ale s hdnotou atributu *schoolhourName* a hodnotou*Hodina 3*. Zadejte HTTP požadavek typu DELETE s adresou *http://localhost:8080/schoolhours/id/hodnota*, kde hodnota je číslo *idSchoolhours* školní hodiny vytvořené v tomto TS.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je oznámení *Schoolhour Hodina 3 deleted*

TS95: Vytvoření pravidelné rozvrhové akce - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/scheduleactions*, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"room"object missing*

TS96: Správné vytvoření pravidelné rozvrhové akce

Popis: Zadejte HTTP požadavek typu POST s adresou *http://localhost:8080/scheduleactions* a v těle požadavku zadejte object JSON objectu s názvem *room* s atributem *roomName* a hodnotou *Místnost A*, objekt *subject* s atributem *subjectName* a hodnotou *Matematika 9.Třída*, objekt *schoolday* s atributem *schooldayName* a hodnotou *Úterý*, objekt *schoolhour* s atributem *schoolhourName* a hodnotou *Hodina 2*.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená pravidelná rozvrhová akce.

TS97: Výpis všech pravidelných rozvrhových akcí

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactions
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořená pravidelná rozvrhová akce.

TS98: Výpis pravidelné rozvrhové akce podle id – pravidelná rozvrhová akce neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactions/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schedule action with id 100000 does not exist.*

TS99: Výpis pravidelné rozvrhové akce podle id – pravidelná rozvrhová akce existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactions/id/hodnota, kde hodnota je číslo *idScheduleactions* pravidelné rozvrhové akce vytvořené v TS96
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi pravidelná rozvrhová akce, jejíž hodnota *idScheduleactions* je stejná jako hodnota zadávaná v dotazu

TS100: Vytvoření konkrétní rozvrhové akce - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/scheduleactionsdates, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"date"is empty*

TS101: Správné vytvoření konkrétní rozvrhové akce

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/scheduleactionsdates a v těle požadavku zadejte atribut JSON objectu *date* s hodnotou *2021-04-04*, atribut *theme* s hodnotou *Zlomky*, objekt *scheduleaction* s atributem *idScheduleactions* a hodnotou *idScheduleactions* pravidelné rozvrhové akce vytvořené v TS96.

Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená konkrétní rozvrhová akce.

TS102: Výpis všech konkrétních rozvrhových akcí

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactionsdates
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořená konkrétní rozvrhová akce.

TS103: Výpis konkrétní rozvrhové akce podle id – konkrétní rozvrhová akce neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactionsdates/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Schedule actions date with id 100000 does not exist.*

TS104: Výpis konkrétní rozvrhové akce podle id – konkrétní rozvrhová akce existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/scheduleactionsdates/id/hodnota, kde hodnota je číslo *idScheduleactionsdates* pravidelné rozvrhové akce vytvořené v TS101
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi konkrétní rozvrhová akce, jejíž hodnota *idScheduleactionsdates* je stejná jako hodnota zadávaná v dotazu

TS105: Vytvoření absence - chybějící údaje

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/absences, tělo požadavku nechte prázdné
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *"student"object is empty*

TS106: Správné vytvoření absence

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/absences a v těle požadavku zadejte objekt JSON objectu s názvem student a atributem *firstName* a *lastName* s hodnotami *David*

a *Dlouhý*, objekt *scheduleactionsdate* s atributem *idScheduleactionsdates* a hodnotou konkrétní rozvrhové akce vytvořené v TS101.
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je vytvořená absence.

TS107: Výpis všech absencí

Popis: zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/absences
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi je již vytvořená absence.

TS108: Výpis absence podle id – absence neexistuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/absences/id/100000
Očekávaný výsledek: stavový kód odpovědi je 422 a v těle odpovědi oznámení *Absence with id 100000 does not exist.*

TS109: Výpis absence podle id – absence existuje

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/absences/id/hodnota, kde hodnota je číslo *idAbsences* pravidelné rozvrhové akce vytvořené v TS106
Očekávaný výsledek: stavový kód odpovědi je 200 a v těle odpovědi absence, jejíž hodnota *idAbsences* je stejná jako hodnota zadávaná v dotazu

TS110: Špatný typ proměnné v URI

Popis: Zadejte HTTP požadavek typu GET s adresou
http://localhost:8080/absences/id/hodnota, kde hodnota je číslo *Text*.
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *Bad path variable type.*

TS111: Špatný tvar JSON objektu

Popis: Zadejte HTTP požadavek typu POST s adresou
http://localhost:8080/buildings a v těle požadavku zadejte atribut JSON objectu s názvem *buildingsName* bez vyplněné hodnoty.
Očekávaný výsledek: stavový kód odpovědi je 400 a v těle odpovědi oznámení *Bad format of data.*