Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Poskytovatel dat (API) k front-end aplikaci pro správu fondů v systému BlackBoard Learn

(Back-end)

Bakalářská práce

Autor: Ondřej Kašpar
Studijní obor: ai3-p

Vedoucí práce: prof. RNDr. Peter Mikulecký, PhD.

Odborný konzultant: Ing. Jan Budina

Hradec Králové Květen 2021

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

…………………………………..

V Hradci Králové dne 30.4.2021 Jméno a Příjmení

Poděkování:

Děkuji prof. RNDr. Peteru Mikuleckému, PhD. a Ing. Janu Budinovi za odborné vedení práce, poskytování rad a ochotu při konzultacích.

Anotace

Cílem závěrečné práce je vytvoření poskytovatele dat (API) pro front-end aplikaci, která umožní jednodušší úpravu testových fondů vyexportovaných ze systému BlackBoard Learn. V analytické části jsou poznatky z analýzy systému BlackBoard, jak pracuje se soubory otázek a předešlé neúspěšné pokusy, na které se narazilo během vývoje. Také je zde popis použitých technologií. Návrhová část obsahuje požadavky funkčnosti a návrh databáze. Implementační část popisuje založení a spuštění projektu, práce s databází a popis klíčových funkcionalit.

Annotation

Title: Data Provider (API) for the Front-End Pool Management Application in the BlackBoard Learn System

The aim of the final work is to create a data provider (API) for the front-end application, which will allow easier modification of test pools exported from the BlackBoard Learn system. The analytical section contains findings from the analysis of the BlackBoard system, how it works with files of questions and describes previous unsuccessful attempts, which were encountered during the development. There is also a description of the technologies used. The design section contains functionality requirements and database design. The implementation section describes the establishment and launch of the project, work with the database and a description of key functionalities.

Obsah

1 Úvod ... 1
2 Cíl práce ... 2
3 Metodika zpracování ... 3
4 Analytická část ... 4
4.1 Použité technologie ... 4
4.1.1 REST API ... 4
4.1.2 Node.js ... 5
4.1.3 XML ... 5
4.1.4 JSON ... 5
4.1.5 PostgreSQL ... 6
4.2 Analýza exportu a importu ... 6
4.2.1 Oliva ... 6
4.2.2 Standardní formát ... 7
4.2.3 QTI 2.1 ... 8
4.3 Popis historie implementací ... 10
4.3.1 Prvotní verze ... 10
4.4 Rozpis úkolů a zaznamenání chyb ... 12
4.5 GitFlow ... 12
4.5.1 Develop a master větve ... 13
4.5.2 Feature větev ... 13
4.5.3 Hotfix větev ... 14
4.5.4 Release větev ... 14
5 Návrhová část ... 15
5.1 Obecné požadavky ... 15
5.2 Funkční požadavky ... 15

5.3 Návrh databáze ... 16
5.3.1 Tabulka upload_file ... 17
5.3.2 Tabulka question ... 17
5.3.3 Tabulka answer ... 18
5.3.4 Tabulka image ... 18
6 Implementační část ... 20
6.1 Založení a spuštění projektu ... 20
6.1.1 Použité balíčky ... 20
6.1.2 Příprava serverové části ... 21
6.1.3 Swagger ... 23
6.2 Připojení PostgreSQL databáze a modul PG ... 26
6.2.1 Instalace lokální databáze ... 26
6.2.2 Heroku databáze ... 26
6.2.3 Instalace na školní server ... 26
6.2.4 Implementace v projektu ... 27
6.3 Popis klíčových funkcionalit aplikace ... 28
6.3.1 Nahrání a rozbalení zip souboru ... 28
6.3.2 Projití souborů a vložení do databáze ... 29
6.3.3 Zpracování otázek ... 31
6.3.4 Export fondu ... 36
7 Shrnutí výsledků ... 42
8 Závěry a doporučení ... 43
9 Seznam použité literatury ... 45
10 Přílohy ... 47

Seznam obrázků

Obr. 1 Ukázka exportu z Olivy 7
Obr. 2 Struktura ZIP souboru – standartní formát 8
Obr. 3 Struktura XML souboru assessmentItem formátu QTI 2.1 10
Obr. 4 Prvotní verze aplikace 11
Obr. 5 Tabulka pro rozpis úkolů 12
Obr. 6 Zobrazení principu GitFlow [8] 13
Obr. 7 Diagram návrhu databáze 16
Obr. 8 Zobrazení Swagger dokumentace 24
Obr. 9 Rozbalení GET metody pro export 25
Obr. 10 Atributy elementů k určení typu otázek 30
Obr. 11 Zobrazení odpovědi pro vrácení otázek 33
Obr. 12 Příklad, jak má vypadat tělo požadavku 34
Obr. 13 Ukázka Heroku scheduler 36
Obr. 14 Vygenerovaný imsmanifest soubor 40

1 Úvod

Systém BlackBoard Learn, který na Univerzitě Hradec Králové dostal název Oliva, je složité a komplexní virtuální vyučovací prostředí pro školy a univerzity, které mnoho nabízí a je navrhnuto tak, aby vyhovělo co nejlépe všem uživatelům a jejich požadavkům. Část, kterou se bakalářská práce zabývá, je úprava testových fondů (otázek/odpovědí) a příprava následného nahrání zpět do Olivy. Momentálně Oliva nabízí chaotická a neintuitivní prostředí pro úpravu testových fondů a nabízí možnost exportu otázek ve dvou formátech, které se dají následně nahrát zpět.

Po důkladné analýze bylo rozumné práci rozdělit na dvě části, a to na front-end (dále FE) a back-end (dále BE).

FE je pojem, který je použitý v souvislosti s webovými aplikacemi, tedy s částí, kterou uživatel vidí a interaguje s ní. Aplikace, která je pod tímto pojmem vytvořena, odesílá požadavky na server a přijímá odpovědi. Tuto část vypracoval kolega Michal Petras a je předmětem jeho závěrečné bakalářské práce. [22]

BE je pojem, související s částí aplikace, v níž se zpracovávají data, které pak FE zobrazuje. Tuto část vytvořil autor této závěrečné bakalářské práce.

Výsledkem spojení těchto dvou částí je jedna funkční aplikace, která nese název BBeditor.

2 Cíl práce

Cílem závěrečné práce bylo vytvoření poskytovatele dat (API) pro front-end aplikaci, která umožní jednodušší úpravu testových fondů vyexportovaných ze systému Oliva ve formátu QTI 2.1. Aplikace bude přijímat od uživatele vyexportovaný soubor, který se na straně serveru zpracuje a převede do požadované podoby, aby se dále mohlo s daty pracovat a provádět na nich CRUD operace, následně ze systému vyexportovat a nahrát upravený soubor do Olivy. Data se budou ukládat do databáze vytvořenou v jazyce SQL. Aplikace bude nahrána na příslušný server, na který bude moci uživatel přistupovat. Bude kladen důraz na funkčnost zejména u předmětů Principy počítačů (PRIPO i KPRIP) a Architektura počítačů (ARCH i KARCH), ale i dalších. Spojením obou částí (front-end a back-end) by mělo vyučujícím přinést mnohem jednodušší a uživatelský přívětivější prostředí pro úpravu testových otázek.

3 Metodika zpracování

Analytická část práce obsahuje analýzu, jak Oliva pracuje s testovými fondy, jejich formáty, rozbor jednotlivých typů otázek, možnost exportu a následně importu a přechozí nepovedený vývoj aplikace. Bude následovat popis pojmů a použitých technologií.

Návrhová část obsahuje návrh databáze a požadavky na funkčnost.

Implementační část popisuje použité balíčky pro vývoj, použití databáze včetně instalace a popis klíčových funkcionalit aplikace.

4 Analytická část

4.1 Použité technologie

4.1.1 REST API

Rest je architektura rozhraní, navržená pro distribuované prostředí. Tato architektura je založená na HTTP protokolu. **[1]**

REST je, na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům. **[1]**

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim. **[1]**

Rest implementuje čtyři základní metody, jinak nazývané jako CRUD – Create, Read, Update, Delete. Metody jsou implementovány pomocí metod HTTP protokolu. Celá interakce mezi klientem a serverem je popsána základními HTTP metodami **[1]**

- **GET** – metoda pro získání zdroje neboli klasický požadavek na stránku. V tomto požadavku je v hlavičkách v odpovědi přiložena informace, která popisuje data obsažená v těle požadavku. Server by měl odpovědět patřičným HTTP kódem – 200 OK
- **POST** – metoda pro vytvoření nového zdroje. Zde nevoláme konkrétní zdroj, protože ještě neexistuje a teprve ho vytváříme. Pokud požadavek proběhl v pořádku, tak server odpoví kódem – 201 Created
- **PUT** – metoda pro upravení již existujícího zdroje. Operace je podobná jako u POST metody, akorát že voláme konkrétní zdroj
- **DELETE** – metoda pro zmazání konkrétního zdroje

Reprezentace popisuje formáty dat, které jsou použity při přenosu mezi serverem a klientem. Zdroj lze reprezentovat mnoha formáty, nejčastěji používané jsou JSON, XML a HTML. [1]

4.1.2 Node.js

Node.js je open-source prostředí v jazyce Javascript, které složí pro psaní webových aplikací. Důraz je kladen na vysokou škálovatelnost, tzn. schopnost obsloužit mnoho připojených klientů najednou. Tato škálovatelnost zajišťuje velký výkon a v dnešní době je Node.js velmi oblíbený pro psaní API serverů pro klientské single-page aplikace. [2]

Node.js pracuje asynchronně, to znamená, že když odešle požadavek, tak nemusí čekat na odpověď, aby mohl pokračovat, ale mezitím může provádět další potřebné kroky, než dostane odpověď. [3]

4.1.3 XML

XML (Extensible Markup Language), je rozšiřitelný značkovací jazyk, který je kompatibilní s mnoha programovacími jazyky. Hodí se hlavně na předávání dat mezi různými aplikacemi. Tento jazyk je čitelný jak uživatelsky, tak i strojově. Každý XML soubor začíná hlavičkou, která udává verzi souboru a použité kódování. Data samotná jsou v souboru obaleny tagy (elementy), které mohou obsahovat atributy pro konkrétnější informace. [4]

Například jednoduchý zápis vypadá takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<address>
  <state>Czech republic</state>
  <postcode>44466</postcode>
</address>
```

4.1.4 JSON

JSON (JavaScript Object Notation) je z jedním nejvyužívanějších formátů pro výměnu dat na Webu. Tato data jsou organizována v polích a objektech a jsou platná se zápisem jazyka JavaScript a je tedy sám o sobě již připravenou datovou strukturou ke zpracování. Ze zápisu v JSON je rovnou vidět, jak s ním bude moct programátor pracovat. JSON nekonkuruje formátu XML při zápisu obsáhlých dokumentů, ale při krátkých zápisech je mnohem lepší řešení. I když je JSON

odvozen z Javascriptu, tak je tento formát jazykově nezávislý tzn. dá se použít i v PHP, RUBY, nebo Pythonu. [5]

Příklad napsání dat ve formátu JSON:

```
{
  "name": "Ondrej",
  "surname": "Kaspar",
  "school": {
    "name": "UHK",
    "fieldOfStudy ": "ai-3p",
  }
}
```

4.1.5 PostgreSQL

Jedná se o open-source objektově relační databázový systém, na kterém se podílí globální komunita vývojářů a firem. Využívá se standardní jazyk SQL. [6]

4.2 Analýza exportu a importu

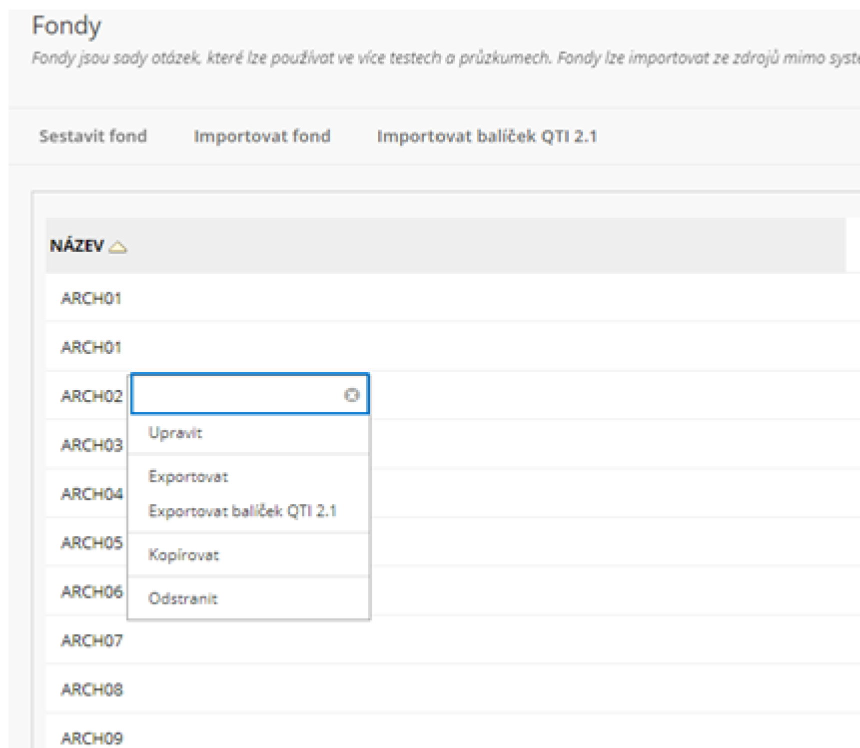
4.2.1 Oliva

Oliva je webový serverový software, který obsahuje správu kurzů, přizpůsobitelnou otevřenou architekturu a škálovatelný design, který umožňuje integraci se studentskými informačními systémy. [7]

Oliva podporuje mnoho typů otázek pro tvorbu testů, ale stěžejní pro tuto práci se staly:

- Otázka s výběrem více odpovědí
- Otázka s více možnými odpověďmi
- Otázka s odpovědí pravda/nepravda
- Esej
- Otázka vyžadující doplnění

Část, kterou se zabýváme, je export a import testových fondů. Zip soubor, který nám vydá oliva po kliknutí na tlačítko export, je složen ze souborů ve formátu XML. Oliva nabízí dva formáty exportu a ty se od sebe vzájemně liší.



Obr. 1 Ukázka exportu z Olivy

4.2.2 Standardní formát

První je výchozí formát, který je obsáhlý a obsahuje všechny informace o otázkách v jednom souboru. Na obr.2 je zobrazena struktura vyexportovaného souboru z Olivy.

Ve složce *csfiles* jsou obsažené obrázky, pokud se u otázek nějaké nachází. Soubor *.bb-package-info* obsahuje systémové informace, například ze které složky se soubor stáhnul. Soubor *imsmanifest* uvádí všechny uvedené soubory a cesty k nim. První *res* soubor obsahuje veškeré informace o otázkách, jedná se o velmi obsáhlý XML soubor. Každá otázka má svůj vlastní identifikátor, a to samé i odpověď. Soubor dále obsahuje mnoho dalších informací, které nemají pro tuto práci využití. Další *res* soubory jsou buďto prázdné, nebo obsahují metadata.

Tento základní formát se nevyužil, protože by příliš zatěžoval databázi. Navíc použití formátu QTI 2.1 je mnohem lepší řešení, protože má mnohem jednodušší XML strukturu.

Name	Size
..	
csfiles	
.bb-log-info	48
.bb-package-info	2 708
.bb-package-sig	32
imsmanifest.xml	1 048
res00001.dat	90 608
res00002.dat	897
res00003.dat	55
res00004.dat	788
res00005.dat	67
res00006.dat	127

Obr. 2 Struktura ZIP souboru – standartní formát

4.2.3 QTI 2.1

Druhý formát je QTI 2.1 (Question and Test Interoperability Specification**).** Tato specifikace (standard) popisuje datový model pro reprezentaci testových otázek, jejich výsledků a také mechanismů pro posouzení jejich správnosti. Systém je v základu navržen abstraktně pomocí UML diagramů, a pro jeho přenositelnost je implementován pomocí jazyka XML.

Specifikace je navržena tak, aby byla přenositelnost mezi vícero systémy snadná, protože s nimi pracují různí uživatelé s různými rolemi. Hlavní úloha systému je poskytnout dokumentovaný formát pro uchování a tvorbu testových otázek a odpovědí a ty se uchovávají v bankách úloh. V neposlední řadě poskytuje systémy pro reportování výsledků testu.

Banka úloh (*question_bank*) je také XML soubor, který obsahuje identifikátory všech testových úloh, které mají být zaregistrovány.

4.2.3.1 Elementy

Jedna úloha (samostatný XML soubor) je definovaná kořenovým elementem *assessmentItem*, který popisuje právě jednu testovou úlohu.

Atributy elementu *assesmentItem* nesou základní informace o úloze, například:

- **Identifier –** identifikátor úlohy, který je důležitý pro banku úloh
- **Title –** titulek úlohy

Tělo testové otázky je reprezentováno elementem *itemBody*, který obsahuje informace o názvech otázky i odpovědí.

Element *responseProcessing* slouží ke zpracování odpovědi a je v něm popsáno, jakým způsobem vyhodnotit uživatelský vstup. Vyhodnocení probíhá pomocí základních podmínek používaných i v programovacích jazycích. Hodnotíme-li odpověď, uzavřeme tento proces do elementu *responseCondition*, který musí povinně obsahovat element *responseIf,* obsahující podmínku.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<assessmentItem xmlns="http://www.imsglobal.org/xsd/imsqti_v2p1"
 xmlns:ns9="http://www.imsglobal.org/xsd/apip/apipv1p0/imsapip_qtiv1p0" xmlns:ns8="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" title="Arch13-16"
 xsi:schemaLocation="http://www.imsglobal.org/xsd/imsqti_v2p1 http://www.imsglobal.org/xsd/qti/qtiv2p1/imsqti_v2p1.xsd"
 adaptive="false" timeDependent="false" identifier="QUE__400580_1">
 ▼<responseDeclaration cardinality="single" baseType="identifier" identifier="RESPONSE">
  ▼<correctResponse>
     <value>QUE__400580_1_false</value>
   </correctResponse>
  </responseDeclaration>
 ▼<outcomeDeclaration identifier="SCORE" cardinality="single" baseType="float">
  ▼<defaultValue>
     <value>0</value>
   </defaultValue>
  </outcomeDeclaration>
  <outcomeDeclaration identifier="FEEDBACKBASIC" cardinality="single" baseType="identifier"/>
 ▼<outcomeDeclaration identifier="MAXSCORE" cardinality="single" baseType="float">
  ▼<defaultValue>
     <value>0</value>
   </defaultValue>
  </outcomeDeclaration>
 ▼<itemBody>
  ▼<div>
     Grafické karty se k základní desce připojují prostřednictvím IDE.
     <br/>
   </div>
  ▼<choiceInteraction responseIdentifier="RESPONSE" maxChoices="1" shuffle="false">
     <simpleChoice identifier="QUE__400580_1_true" fixed="true">Pravda</simpleChoice>
     <simpleChoice identifier="QUE__400580_1_false" fixed="true">Nepravda</simpleChoice>
   </choiceInteraction>
  </itemBody>
 ▼<responseProcessing>
  ▼<responseCondition>
   ▼<responseIf>
    ▼<match>
       <variable identifier="RESPONSE"/>
       <correct identifier="RESPONSE"/>
     </match>
    ▼<setOutcomeValue identifier="SCORE">
       <variable identifier="MAXSCORE"/>
     </setOutcomeValue>
    ▼<setOutcomeValue identifier="FEEDBACKBASIC">
       <baseValue baseType="identifier">correct_fb</baseValue>
     </setOutcomeValue>
    </responseIf>
   ▼<responseElse>
    ▼<setOutcomeValue identifier="FEEDBACKBASIC">
       <baseValue baseType="identifier">incorrect_fb</baseValue>
     </setOutcomeValue>
    </responseElse>
   </responseCondition>
  </responseProcessing>
 </assessmentItem>
```

Obr. 3 Struktura XML souboru assessmentItem formátu QTI 2.1

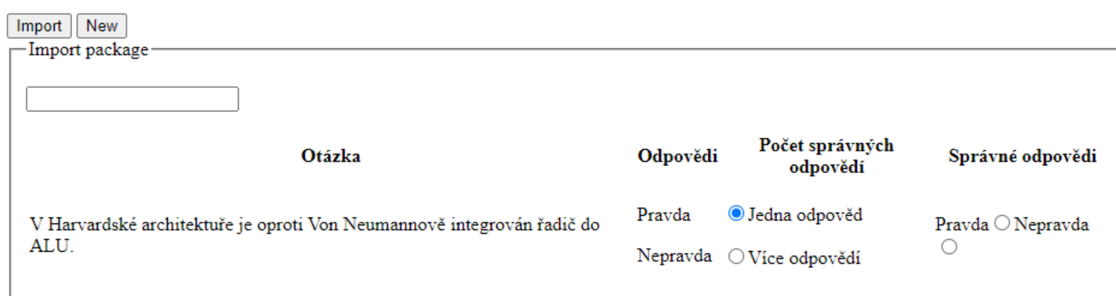
4.3 Popis historie implementací

4.3.1 Prvotní verze

S kolegou, který tvoří FE aplikaci, jsme v prvotní části vývoje neměli rozdělenou práci na FE a BE, ale míchali jsme vše dohromady.

Má práce byla realizována pomocí jazyka PHP. Připravil jsem funkčnost pro nahrání a rozbalení ZIP souboru a následné předání. ZIP soubor nebyl formátem QTI 2.1, ale standartní formát systému Oliva. Neupravené XML soubory jsem předával kolegovi, který tyto soubory upravoval pomocí Javascriptu na klientovi.

Kámen úrazu bylo upravování názvů otázek a odpovědí přímo na klientovi a následné uložení dat. Tento krok byl až příliš komplikovaný a bez databáze prakticky nerealizovatelný, i z hlediska bezpečnosti postrádal smysl. Grafické rozhraní vypadalo v prvotní verzi takto:



Obr. 4 Prvotní verze aplikace

Po důkladnější analýze jsme práci rozdělili na BE (API) a FE (Uživatelská aplikace zpracovávající požadavky od BE) a začali používat formát QTI 2.1, který obsahoval mnohem méně dat potřebných pro úpravu otázek, přičemž data by se ukládaly do databáze. Rozhodli jsme se také používat modernější jazyky a technologie:

- BE – Node.js, PostgreSQL
- FE – Vue.js

Důvody použití Node.js místo PHP byly hlavně škálovatelnost, což znamená schopnost obsloužit více klientů naráz a psaní přívětivějších API.

4.4 Rozpis úkolů a zaznamenání chyb

V průběhu vývoje je normální narazit na chyby, popřípadě na nedostatky funkčnosti. K tomu sloužila Excel tabulka obr. 5 navržená tak, aby se zde dalo zapsat o jakou chybu se jedná, popis, v jakém se to odehrává prostředí, odkaz na fond, kde se chyba projevila, jedná-li se o BE či FE, verze aplikace a jestli se chyba vyřešila nebo ne. Tato tabulka byla velmi užitečná hlavně pro připomenutí daného problému zejména proto, aby se na něj nezapomnělo.

1	Chyba	Popis	Prostředí	Odkaz na fond	FE/BE	Verze	Vyřešeno

9	Automatické mazání databáze	Automaticky mazat z databáze soubory, které tam jsou déle než 24 od nahrání. (kontrolovat fileID):
		if (currentTimeStamp - fileId > 86400) {
		zmaž všechny záznamy patřící k fileId
		}
			DEV	jakýkoliv	BE	develop-v2	ANO

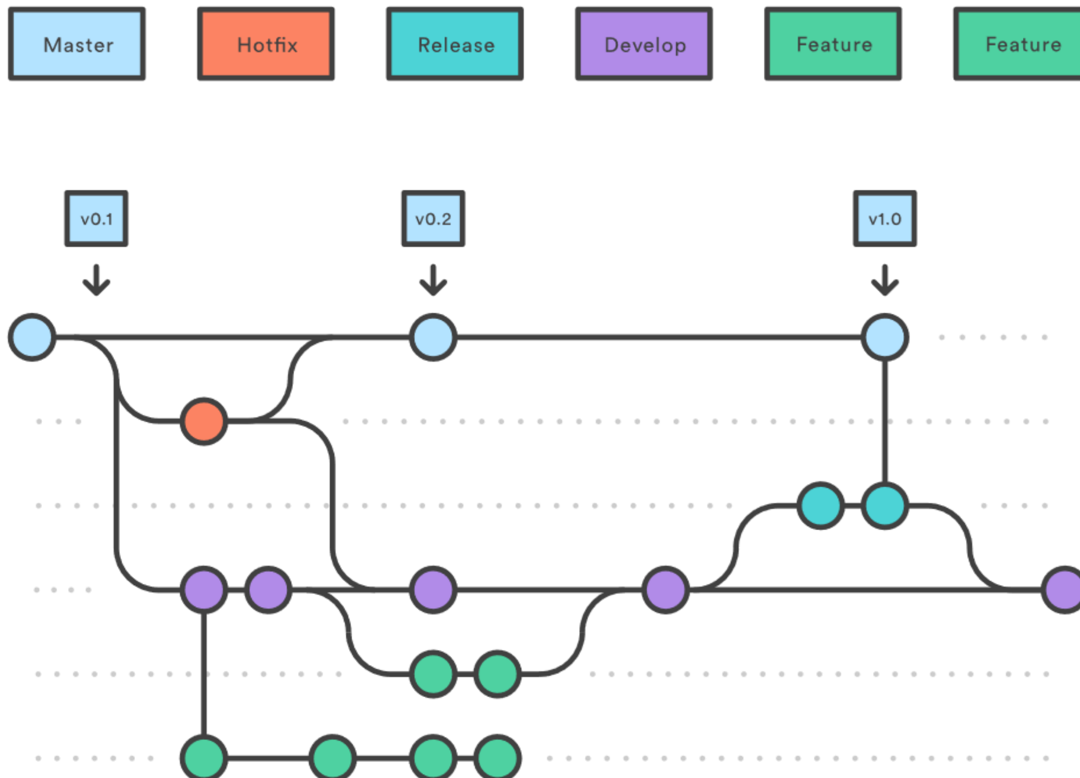
28	Nové API na main page	Potřebuji vytvořit nové GET API, které bude vracet všechny pooly, které byly uploadovány max před 24h. API vrátí název poolu, čas nahrání a datum nahrání + (možná ještě počet otázek)	DEV	jakýkoliv	BE	develop-v2	ANO
34	vytvořit js soubory pro herocu cron	mazání dat z db, mazání souborů uploaded file	DEV		BE	develop-v2	ANO
35	Přeházení odpovědí	Při výběru nové správné odpovědi a uložení se přehází seskupení odpovědí na výpisu https://prnt.sc/w3xtng	PROD		BE	develop-v2	ANO
38	Nenačtení otázky	Fond z PSIT1 nenačítá u odpovědi jednu otázku: https://ibb.co/xzgh1D3 - jedná se o tu první Chyba v parsování	PROD	Otázky - Přednáška 11	BE	develop-v2	ANO
	Chybí mezery v textu	V otázkách (možná i odpovědích) se spojují slova.	PROD	Otázky - Přednáška 11	BE	develop-v2	ANO

Obr. 5 Tabulka pro rozpis úkolů

4.5 GitFlow

Jelikož máme s kolegou rozdělenou práci na FE a BE, tak jsme se rozhodli částečně držet pracovního postupu GitFlow, který využívají vývojové týmy. GitFlow je ideální pro projekty, které mají cyklus plánovaného vydávání verzí aplikace.

GitFlow je pracovní postup nástroje Git, který pomáhá s nepřetržitým vývojem softwaru a implementací postupů DevOps – sada postupů, která pracuje na automatizaci a integraci procesů mezi týmy pro vývoj softwaru. [8]



Obr. 6 Zobrazení principu GitFlow [8]

Tento postup je založen na vytváření a přiřazování funkcí větví. Větve Master a Develop by měly být pevné a neměli by se mazat, pouze aktualizovat.

4.5.1 Develop a master větve

Hlavní větev celého projektu je master, viz obr. 6. Zde je uložen aktuální a stabilní kód, který je na produkci. Do master se slučují hotfix a release větve. [8]

Develop je hlavní větev, do které se slučují nové funkcionality z větví feature. Nachází se zde kód, který je připraven ke sloučení do master větve. [8]

4.5.2 Feature větev

Každá nová funkcionalita aplikace by měla mít svou feature větev, například *feature/addNewEndpoint*. Po dokončení a odsouhlasení této větve, se musí sloučit do Develop, nikoli do Master. Feature vychází z aktuální větve Develop.

4.5.3 Hotfix větev

Tato větev má za úkol opravit chyby na produkci, to znamená chyby, které se objeví v Master větvi, například *hotfix/fixCondition*. Jakmile je oprava hotová, měla by být sloučena do obou Master a Develop, nebo do aktuální Release větve. [8]

Díky této větvi, může být oprava aplikována ihned co bude hotová a nemusí se čekat na další cyklus vydání.

4.5.4 Release větev

Pokud je ve větvi Develop dostatek nových funkcí, které by mohli tvořit novou verzi aplikace a je tím pádem připravena na vydání, tak se vytvoří z Develop větev Release s daným číslem verze, například *release/1.20.1*. Větev se sloučí do Master větve a také by se měla sloučit do Develop – zde mohly mezitím přibýt nové funkcionality. Vytvoření této větve spustí další cyklus vydání, takže po tomto bodě nelze přidávat žádné nové funkce. [8]

Vytváření Release větví je další přímá operace větvení. Stejně jako Feature větve je založena na větvi Develop.

5 Návrhová část

5.1 Obecné požadavky

Jedním z požadavků na funkčnost je, aby FE aplikace dostala funkční endpointy (http požadavky), které zpracují a dodají potřebná data, tzn. výstupní data otázek musí být v JSON formátu.

Potřeba bude také, aby výstupní zip soubor byl validní s formátem QTI 2.1 a tím pádem i se systémem Oliva.

5.2 Funkční požadavky

Funkčnost API závisí na FE aplikaci, která posílá požadavky na BE a ten je zpracovává. S API lze také pracovat pomocí Swagger dokumentace (open-source framework pro tvorbu a návrh dokumentace pro REST API [12]), která nabízí popis a interakci s jednotlivými endpointy.

Prvním krokem je nahrání vyexportovaného QTI 2.1 zip souboru do aplikace, proběhne rozbalení a následné upravení XML souborů s otázkami a odpověďmi. Data se zpracují a uloží do databáze. Každý nahraný soubor dostane vygenerované unikátní ID, podle kterého se pozná, s jakým fondem otázek se pracuje. Názvy otázek a odpovědí se ukládají včetně HTML tagů, aby uživatel viděl přesně to, co si z Olivy vyexportoval.

API bude schopné provádět následující operace:

- Nahrát a rozbalit soubor, upravit XML soubory na JSON a uložit data do databáze
- Dodat všechny otázky a odpovědi v daném fondu
- Dodat danou otázku s odpověďmi podle ID
- Vytvořit novou otázku s požadovanými odpověďmi
- Upravit danou otázku
- Smazat danou otázku
- Smazat všechny otázky
- Vyexportovat soubor s validními daty pro import do Olivy

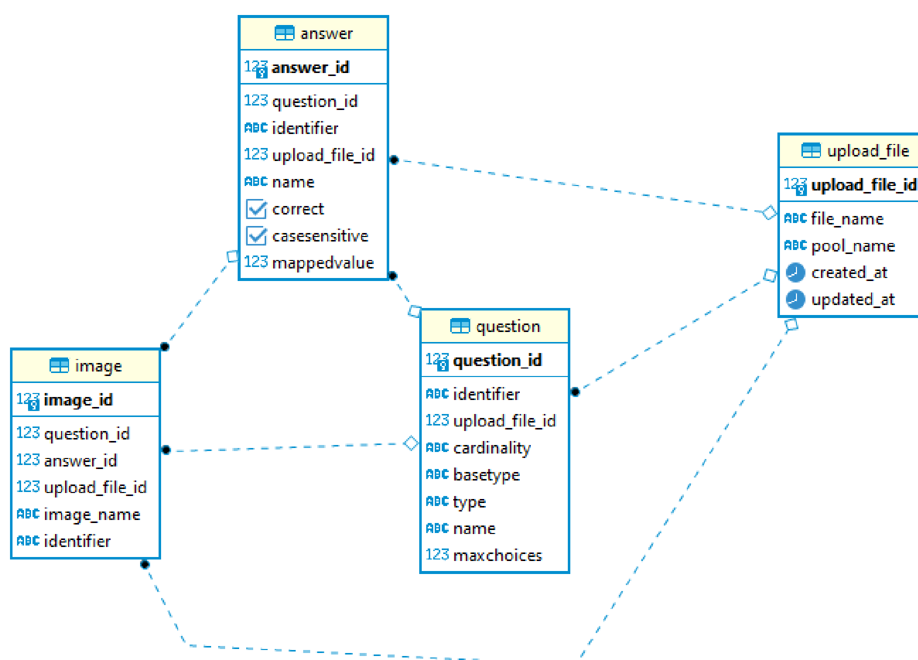
- Možnost zobrazovat výstupní hlášky v češtině a angličtině

5.3 Návrh databáze

V databázi bude potřeba udržovat data otázek, odpovědí, obrázků, informace o importovaném souboru. Datová struktura není nijak komplexní a obsahuje jednoduché typy. Databáze byla navrhnuta, aby nebyla nijak složitá a zároveň aby datová struktura byla dostatečná pro aplikaci a práci s ní.

Při návrhu jsem vybíral ze dvou databází – MongoDB a PostgreSQL. MongoDB je nejpopulárnější NoSQL technologie a PostgreSQL je velmi rozšířená v kategorii relačních databází.

ER diagram byl vygenerovaný v klientské aplikaci DBeaver. Jedná se o univerzální open-source nástroj pro správu databází. Nástroj podporuje velkou škálu relačních a NoSql databází. [9]



Obr. 7 Diagram návrhu databáze

Diagram uvádí spojení a závislost mezi jednotlivými tabulkami pomocí cizích klíčů. Tabulky jsou celkem čtyři a každá je naplněna potřebnými atributy, kde některé vycházejí z formátu QTI 2.1.

5.3.1 Tabulka upload_file

Tato tabulka obsahuje informace o souboru nahraného z Olivy, její atributy jsou:

- **upload_file_id** – primární klíč
- **file_name** – název rozbaleného zip souboru
- **pool_name** – název fondu, který se zobrazuje v Olivě
- **created_at** – datum, kdy byl fond nahrán
- **updated_at** – datum, kdy byl fond naposledy aktualizován, například změna názvu otázky

5.3.2 Tabulka question

Tabulka uchovává data o otázce. Její atributy jsou:

- **question_id** – primární klíč
- **identifier** – unikátní identifikátor pro otázku typu String, například QUE__483372_1. Tento identifikátor byl převzat z QTI 2.1 formátu, aby se dodržoval určitý standart pojmenovávání
- **upload_file_id** – cizí klíč odkazující do tabulky upload_file na daný nahraný soubor
- **cardinality** – udává, zda lze vybrat více možných odpovědí, nebo pouze jednu. Možnosti jsou *multiple*, nebo *single*
    - **multiple –** možné je více správných odpovědí
    - **single –** možná je pouze jedna správná odpověď. Tento typ se udává i u eseje a otázky vyžadující doplnění
- **basetype**
    - **identifier** – uživatel si vybírá odpovědi, které jsou dopředu připravené tzn. uživatel vybírá z možností a) b) c), True/False atd.
    - **string** – toto platí u otázek, které vyžadují doplnění přímo uživatelem tzn. esej a otázky vyžadující doplnění

- **type –** po rozparsování a určených kritérií se přiřadí správný typ otázky např. MULTIPLE_ANSWER, TRUE_FALSE, MULTIPLE_CHOICE. Formát QTI 2.1 neuchovává přímo informaci o typu otázky, ale ke každému typu sedí sada parametrů, dle kterých se dá typ určit
- **name –** název otázky
- **maxchoises –** hodnota, která říká kolik možností může uživatel vybrat. Při hodnotě 1, může uživatel vybrat pouze jednu odpověď (True/False), ale při hodnotě 0 jich může vybrat více (MULTIPLE_ANSWER)

Tyto atributy jsou potřeba při znovusložení XML souborů, který je využívá také jako atributy a následný import do Olivy.

5.3.3 Tabulka answer

Tabulka uchovává data o odpovědi. Atributy jsou:

- **answer_id** – primární klíč

- **identifier** – unikátní identifikátor pro otázku typu String, například answer_2. Tento identifikátor byl převzat z QTI 2.1 formátu, aby se dodržoval určitý standart pojmenovávání
- **question_id** – cizí klíč odkazující na tabulku *question*, na danou otázku
- **upload_file_id –** cizí klíč odkazující do tabulky upload_file na daný nahraný soubor
- **name –** název odpovědi
- **correct –** udává, jestli je odpověď správná
- **casesensitive –** udává, jestli se mají rozlišovat malá a velká písmena
- **mappedvalue –** vyskytuje se pouze u otázky vyžadující doplnění, hodnota udává mapování uživatelské odpovědi na správně předefinované odpovědi

5.3.4 Tabulka image

Tabulka sama o sobě neobsahuje přímo informace o obrázcích při prvotním importu dat, ale naopak až při exportu. Je to dáno tím, že obrázky, pokud tedy otázka nebo odpověď nějaké obsahují, se ukládají do tabulky *question*, protože se ukládá

celé znění otázky i s HTML tagy, kde se nachází i cesta k obrázku. Jedná se spíše o pomocnou tabulku při finálním skládání XML souborů s otázkami.

- **image_id** – primární klíč
- **question_id** – cizí klíč odkazující na tabulku question, na danou otázku
- **answer_id** – cizí klíč odkazující na tabulku answer, na danou odpověď. Pokud odpověď neobsahuje obrázek, hodnota se nevyplňuje
- **upload_file_id** – cizí klíč odkazující do tabulky upload_file na daný nahraný soubor
- **image_name** – název obrázku
- **identifier** – unikátní textový identifikátor pro obrázek typu String. Tento identifikátor byl převzat z QTI 2.1 formátu, aby se dodržoval určitý standart pojmenovávání

6 Implementační část

Tato část popisuje postup od založení projektu a popis klíčových funkcionalit aplikace.

6.1 Založení a spuštění projektu

Pro tuto část musí být na počítači nainstalovaná, nejlépe nejnovější verze, Node.js a balíčkového manažera npm. Příkazem *npm init* se spustí krátký průvodce, který vytvoří soubor *package.json* a do něho se vyplní základní informace o projektu. Informace jsou například název projektu, popis nebo počáteční verze. Tyto informace se zadávají ve spuštěném průvodci, ale pokud byl projekt vytvořen například na GitHubu – webová služba podporující vývoj softwaru za pomocí verzovacího nástroje Git [10], tak informace vyplněné tam se objeví přímo v souboru *package.json*.

V tomto souboru je také blok *dependencies*, který obsahuje všechny nainstalované balíčky. Prvotní instalace balíčku se provádí příkazem *npm i <balíček>.* Po instalaci se v projektu vytvoří složka *node_modules*, ve které jsou zdrojové kódy daného balíčku. Standardní projekt využívá velké množství balíčků a tím pádem se jedná o stovky souborů navíc, což nemělo být obsažené v Gitu, protože by zbytečně docházelo k velkému zaplnění uložiště. K tomu slouží soubor *.gitignore*, ve kterém se registrují soubory či složky.

6.1.1 Použité balíčky

Node.js využívá npm správce balíčků, jedná se o podobnou technologii jako v PHP, který využívá composer. Baliček je sada modulů, která se po nainstalování do projektu dá využívat, ale nedá se upravovat. Po nainstalování se přidá do seznamu závislostí a příkazem *npm install* se dají všechny balíčky, zde uvedené, jednoduše doinstalovat. [11]

- **Express** – aplikační framework pro jednodušší vytváření webových aplikací a API
- **Nodemon** – po změně kódu automaticky restartuje server, aby se projevily změny

- **Xml-stream** – nástroj, který načte XML soubor a převede ho na JSON podobu
- **Xmlbuilder2** – nástroj, který pomocí metod, vytvoří XML soubor
- **Glob** – rekurzivně projde složku a najde všechny soubory, které se ve složce nachází
- **Dotenv** – dokáže načítat proměnné z *.env* souboru, které se mohou v každém prostředí lišit. Jedná se hlavně o databázový konektor, protože testovací prostředí aplikace by měla používat i testovací databázi a totéž platí pro ostrý provoz aplikace
- **swagger-ui-express** – slouží pro generování dokumentace API
- **extract-zip** – slouží pro rozbalení zip souboru a následné uložení obsahu
- **express-easy-zip** – jedná se o middleware, který dokáže v odpovědi serveru vytvořit a poslat zip soubor
- **connect-history-api-fallback** – Middleware na požadavky proxy prostřednictvím zadané indexové stránky, užitečné pro jednostránkové aplikace

6.1.2 Příprava serverové části

Vše se připravuje a odvíjí ze souboru *index.js*, který je umístěn v kořenové struktuře projektu. Z tohoto souboru jdou veškeré akce ze serveru, tzn. obsluha http požadavků atd. Zde je potřeba:

- importovat všechny potřebné moduly pro běh serveru
- vytvořit instalaci express
- připojit potřebný middleware
- nastavit obsluhu http požadavků

Nejdřív je potřeba importovat modul Express a nastavit na jakém portu poběží:

```
const express = require('express')
const app = express()
require('dotenv').config()
const port = process.env.PORT || 3000
```

```
app.listen(port, () => {
  console.log(`Server running on port ${port}`)
})
```

Konstanta port má dvě možnosti, jak nastavit port, buď přímo řekneme, že se jedná o 3000, nebo si číslo portu načteme z env proměnné. Dále nastavíme, že má server poslouchat na výše uvedeném portu.

Dále bude potřeba připojit potřebný middleware.

- **express.json()** má na starosti aby příchozí požadavky byly zpracovávány do formátu JSON
- **zip()** je použití modulu **express-easy-zip**, aby odpověď serveru byla schopná předat zakódovaný ZIP soubor, který si dále zpracuje klient
- **history()** je použití middleware **connect-history-api-fallback**

```
app.use(express.json({ limit: '200mb' }))
app.use(zip())
app.use(history())
```

Kvůli přehlednosti by nebylo vhodné mít všechny importované moduly a http požadavky v tomto jednom souboru. Modul Express nabízí směrování, které umožňuje jejich rozdělení do více souborů.

```
const uploadRouter = require('./routes/upload')
const questionRouter = require('./routes/question')
const exportRouter = require('./routes/export')

app.use(uploadRouter)
app.use(questionRouter)
app.use(exportRouter)
```

Soubory jsou pojmenovány podle toho, s jakou událostí pracují:

- Upload má na starosti nahrání, rozbalení zip souboru, zpracování souborů v něm a úprava XML dat do vlastní JSON podoby a nahrání do databáze.
- Question zpracovává vše ohledně práce s otázkami, jako je nahrání, editace, mazání, nebo vrácení seznamu otázek/otázky

- Export vytváří z finální úpravy fondu nové XML soubory s otázkami a následně vytvoření nového ZIP souboru.

Je potřeba připravit spojení s FE aplikací. Po dokončení nějakého vývoje se musí na FE vytvořit pomocí příkazu *npm build* složka *dist*, ve které jsou veškeré funkcionality z Vue.js zkompilovány do podoby, se kterou je schopen pracovat prohlížeč. Express umí k této složce přistupovat a pracovat s ní. Zde je použito zpracování http požadavku GET, který na adrese „/“ načte index.html a tím pádem i zkompilovanou FE aplikaci.

```
app.use(express.static(__dirname + './../client/dist/'))
app.get('/', function (req, res) {
  res.sendFile(__dirname + './../client/dist/index.html')
})
```

Server je možné spustit pomocí příkazu *npm run dev*. Díky použití balíčku Nodemon se po každé úpravě kódu server sám restartuje, aby se aplikovaly provedené změny.

6.1.3 Swagger

Swagger je open-source framework pro tvorbu a návrh dokumentace pro REST API, obsahuje i nástroje pro automatizovanou dokumentaci a testování existujícího API. Swagger je pouze nástroj, který využívá OpenAPI specifikaci. [12]

Použitý modul swagger-ui-express generuje dokumentaci ze *swagger.json* souboru. Výsledkem je živá dokumentace pro API hostované ze serveru přes routu. [13]

Swagger
Supported by SMARTBEAR

Blackboard editor API 1.0.0

API that edit QTI 2.1 format questions form Blackboard (oliva)

MIT

Upload zip API for upload file to system

POST `/api/{lang}/upload` Upload zip in system

Questions API for Questions in the system

GET `/api/{lang}/upload/{upload_id}/questions/{id}` Get one question in system

PUT `/api/{lang}/upload/{upload_id}/questions/{id}` Update question

DELETE `/api/{lang}/upload/{upload_id}/questions/{id}` Delete question

GET `/api/{lang}/upload/{upload_id}/questions` Get all questions in system

POST `/api/{lang}/upload/{upload_id}/questions` Create new questions in system

DELETE `/api/{lang}/upload/{upload_id}/questions` Delete all questions

Export API for export file from the system

GET `/api/{lang}/upload/{upload_id}/export/{pool_name}` Get exported file

Obr. 8 Zobrazení Swagger dokumentace

V souboru *swagger.json* jsou uvedené základní informace o projektu, názvy serverů, na kterých je tato dokumentace povolena a následné endpointy aplikace.

Například pro endpoint na export hotového fondu je zápis:

```
"/api/{lang}/upload/{upload_id}/export/{pool_name}": {
      "parameters": [
        {
          "name": "lang",
          "in": "path",
          "required": true,
          "description": "Language of page (cs, en)",
          "type": "string"
        },
        …
        {
          "name": "pool_name",
          "in": "path",
          "required": true,
          "description": "Pool name",
```

```
          "type": "string"
        }
      ],
      "get": {
        "tags": [
          "Export"
        ],
        "summary": "Get exported file",
        "responses": {
          "200": {
            "description": "OK"
          }
        }
      }
    }
```

Zapisují se zde parametry a jejich typ. Parametry jsou jazyk ({lang}), upload_id souboru se kterým se pracuje ({upload_id}) a vlastní název fondu ({pool_name}). U parametrů se dá nastavit, jestli mají být povinné, datový typ nebo popisek. Dále je zde uvedená metoda GET a s tím do jaké patří kategorie a co jaká má přijít odpověď serveru.

Výsledná podoba exportu je na obr. 9.

Export API for export file from the system

GET /api/{lang}/upload/{upload_id}/export/{pool_name} Get exported file

Parameters — Try it out

Name	Description
lang * required string (path)	Language of page (cs, en) lang - Language of page (cs, en)
upload_id * required integer (path)	File upload ID upload_id - File upload ID
pool_name * required string (path)	Pool name pool_name - Pool name

Responses — Response content type: application/json

Code	Description
200	OK

Obr. 9 Rozbalení GET metody pro export

6.2 Připojení PostgreSQL databáze a modul PG

6.2.1 Instalace lokální databáze

Lokální databáze vyžaduje instalaci nejlépe aktuální verze PostgreSQL, následné nakonfigurování portů, názvů a hesel. PostgreSQL umožňuje mít tabulky uložené ve schématech, což přidává přehlednost, pokud využívá více aplikací stejnou databázi. Tato aplikace má tabulky uložené ve schématu *bb_editor*.

6.2.2 Heroku databáze

Databáze byla v brzké době vývoje uložena na Heroku – jedná se o cloudovou platformu pro uložení projektů a obsahuje i rozšíření pro vytvoření uložiště databáze. K tomu slouží oficiální PostgreSQL plugin, který vše nakonfiguruje sám a stačí se akorát připojit pomocí databázového Stringu. Verze zdarma umožnuje uložení pouze 1000 záznamů, což je nedostatečné, kdyby aplikaci používalo více lidí najednou. Rozšíření uložiště vyžaduje placení v měsíčních cyklech, proto se nabízí využití školního serveru, kde bude databáze uložena. [14]

6.2.3 Instalace na školní server

Server *edu.uhk.cz* umožnuje studentům a učitelům publikovat své webové aplikace. Využití tohoto serveru se zvažovalo pouze pro uložiště databáze, protože zde nejdou uložit aplikace vytvořené v Node.js a Vue.js. Bohužel typ relační databáze je zde pouze MySQL a 100 MB volného místa pro projekty. [15]

Po kontaktování a následné konzultaci s Ing. Janem Budinou, byl pro tento projekt zpřístupněn čistý Ubuntu server. Server neobsahuje grafické rozhraní, tak se vše muselo instalovat přes příkazy. Například instalace PostgreSQL serveru:

```
$ sudo apt install postgresql
```

Následně bylo potřeba otevřít databázový port pro veřejné připojení přes databázový String, protože na server se dá připojit pouze přes SSH – zabezpečený komunikační protokol v počítačových sítích, které používají TCP/IP. [16]
Bez otevření portů, by se dalo na databázi připojit pouze ve stejné síti.

Na tento server se nainstalovala pouze databáze, nikoli projekt. Uložení projektu se již od začátku zvažovalo Heroku, protože zde je připravená veškerá konfigurace pro běh aplikace včetně nasazení z GitHubu.

6.2.4 Implementace v projektu

O komunikaci aplikace s databází se stará modul PG. Jedná se o kolekci modulů pro propojení PostgreSQL databáze s aplikací psanou v Node.js. Podporuje callback funkce, promises, async/await, connection pooling. Jedná se o klient, takže je zde mnoho funkcionalit jako v samotném PostgreSQL. [17]

Pro toto spojení je vytvořen samostatný soubor *db.js*, ve kterém je napsaná veškerá potřebná konfigurace:

```
const Pool = require('pg').Pool
require('dotenv').config()
const connectionString = process.env.DATABASE_URL

const pool = new Pool({
  connectionString
})
module.exports = pool
```

Nejdříve se importuje modul PG a také dotenv pro načtení databázového Stringu pro připojení. String pro lokální databázi vypadá takto:

postgres://postgres:admin@localhost:5433/bc-api

Popisuje, o jaký typ databáze se jedná, heslo, hosta, port a databáze, se kterou se pracuje. Zápis pro připojení na produkční databázi, která je uložena na školním serveru, vypadá takto:

postgres://postgres:postgres@imitgw.uhk.cz:59752/bc-api

Následně je potřeba toto nastavení exportovat pro použití v následujících částech aplikace.

Pro vrácení například všech otázek z konkrétního fondu, si aplikace nejdřív připojí databázový soubor a následně provede dotaz.

```
const pool = require('../db/db')
const questions =
await pool.query('SELECT * FROM bb_editor.question WHERE upload_file_id =
 $1', [uploadId])
```

6.3 Popis klíčových funkcionalit aplikace

V této části jsou rozebrány klíčové funkcionality aplikace.

6.3.1 Nahrání a rozbalení zip souboru

První krok, který uživatel potřebuje udělat, je nahrát vyexportovaný soubor z Olivy. FE aplikace proto využívá endpoint „*/api/{lang}/upload*“, který přijme a rozbalí ZIP soubor a následně zpracuje data v něm.

Po nahrání ZIP souboru je potřeba mít soubor kam na server uložit, proto se vytvoří (pokud ještě není) složky *upload_files a unzip_files* ve složce *data* v kořenovém adresáři. Soubor se musí zkontrolovat, jestli nepřesahuje povolenou velikost a nastaví se mu nové jméno pro uložení, aby nedošlo ke konfliktu se soubory, které mají stejný název, k tomu se generuje aktuální čas přesný na sekundy, aby byl vždy unikátní. Po uložení na disk do složky *upload_files*, se soubor může přesunout k dalšímu kroku, a to je rozbalení ze ZIP formátu. K tomu slouží modul *extract-zip*, který vykonává veškerou práci s rozbalením. Modul přijímá parametry cesty k ZIP souboru a kam má uložit jeho obsah, k tomu slouží složka unzip_files. Zde se vytvoří složka s názvem ZIP souboru a za ní se opět přidá přípona aktuálního unikátního času. Po tomto kroku se může původní ZIP soubor smazat, aby se nezaplňovalo zbytečně místo na disku.

```
const extract = require('extract-zip')
const extractZip = async (file, destination, deleteSource) => {
  await extract(file, { dir: destination }, (err) => {
    if (!err) {
      if (deleteSource) fs.unlinkSync(file)
      nestedExtract(destination, extractZip)
    } else {
      console.error(err)
    }
  })
}
```

Informace o složce se dále uloží do databáze do tabulky *upload_file*, zaznamená se jméno a aktuální čas vložení. Název fondu se doplní v následujícím kroku při procházení jednotlivých souborů.

6.3.2 Projití souborů a vložení do databáze

Další a nejsložitější krok je projití jednotlivých souborů a vložení jejich obsahu do databáze. Modul *glob* umožnuje procházet jednotlivé složky a podsložky a vrací seznam souborů v poli. Toto pole se poté projede v cyklu a s každým souborem, který obsahuje otázku a odpovědi, se pracuje jednotlivě.

Zde se využije modul *xml-stream*, který převádí XML na JSON a je potřeba mu nastavit konfiguraci:

```
const XmlStream = require('xml-stream')
const xml = fs.createReadStream(itemFile) // cesta k xml
const xmlStream = new XmlStream(xml)

xmlStream.collect('value')
xmlStream.collect('mapEntry')
xmlStream.collect('p')
xmlStream.collect('img')
xmlStream.collect('simpleChoice')
xmlStream.collect('setOutcomeValue')
xmlStream.preserve('div')
xmlStream.preserve('simpleChoice')
```

Nastavení *collect* znamená, že pokud má element více potomků (dalších elementů), tak je potřeba aby se vrátilo pole všech potomků. Pokud by toto nebylo nastaveno, tak by poslední potomek přepsal předešlé a xml-stream by vrátil pouze jeden prvek. Například pokud by měl název otázky rozdělený do více *<p>* tagů, tak by se vrátila nekompletní otázka. [18]

Nastavení *preserve* řeší problém u zanořených tagů. Například pokud by bylo znění otázky: „*<p>Toto <i>je</i> otázka</p>*“, tak by výsledná upravená podoba z JSON vypadala takto: „Toto otázka je“. Toto je považováno za obecný problém u převodu XML na JSON či naopak. Tento modul poskytuje dostatečné řešení tohoto problému, ale nikoli ideální. [18]

6.3.2.1 Určení typu otázky

Typ otázek se určí podle atributů elementů. V QTI 2.1 není žádná přímá informace o jaký typ se jedná. Například otázka s více možnými odpověďmi obsahuje v elementu *responseDeclaration* atribut *cardinality="multiple"* a *baseType="identifier"*. V elementu *correctResponse* musí být zaznamenány více než jeden identifikátor správné odpovědi.

```
▼<responseDeclaration cardinality="multiple" baseType="identifier"
  ▼<correctResponse>
      <value>answer_1</value>
      <value>answer_2</value>
      <value>answer_3</value>
    </correctResponse>
  </responseDeclaration>
```

Obr. 10 Atributy elementů k určení typu otázek

6.3.2.2 Složení názvu otázky/odpovědi

Pokud je název otázky obalený do html tagu, tak *xml-stream* vrátí tento tag jako klíč objektu, objekt může mít i další objekty, pokud se jedná o otázky s vnořenými tagy, nebo pokud otázka obsahuje obrázky. Tyto objekty je potřeba při větším zanoření projet vícekrát až na poslední objekt, ve kterém jsou uloženy informace. V kódu je toto řešeno vlastní funkcí, která volá sama sebe, dokud se nenajde objekt, který má klíče obsahující potřebné informace. V průběhu běhu funkce se skládá String, který přidává tagy, na které po cestě narazí a po nalezení názvu otázky tyto tagy uzavře, aby byl výsledek validní. Stejný postup platí i pro odpovědi. Výsledek se vrátí přes callback funkce.

U obrázků se musí nejdříve ověřit, zda uvedená cesta v atributu *src* existuje v rozbalené složce, protože pokud se přímo v olivě obrázky vloží špatně, tak po exportu QTI 2.1 se nemusí obrázek ve složce objevit. Výjimku mají obrázky, které jsou vloženy do otázky jako odkaz, ty nepotřebují být uloženy ve složce. Pokud cesta existuje, obrázky se zakódují do Base64 – jedná se o kódování, které převádí binární data na posloupnost tisknutelných znaků. Dokáže obrázek zakódovat do textové podoby. [19]

Tento krok je nutný pro FE aplikaci, která využívá editor, který pomocí tohoto kódování zobrazuje a ukládá obrázky.

Výsledek se vloží do databáze spolu s dalšími hodnotami, které se vezmou přímo z XML souboru převedeného na JSON.

Po úspěšném projití všech souborů s otázkami a odpověďmi server vrátí odpověď složenou z *upload_file_id* neboli primární klíč z tabulky *upload_file*. Toto id se na straně klienta uloží do paměti prohlížeče, aby mohlo kdykoli vložit jako parametr do dalších http požadavků, kde je potřeba. Pokud se v průběhu narazí na typ otázky, která není podporovaná, například otázka interaktivní, tak se průběh souborů ukončí a vyskočí chybová hláška.

6.3.3 Zpracování otázek

U zobrazení a práce s otázkou jako takovou, se vždy posílá JSON text, kde je uveden typ otázky, název otázky a pole odpovědí. To celé je v jednom požadavku či odpovědi serveru. Požadavky na server jsou:

- GET
  - Vrať všechny otázky
  - Vrať jednu otázku
- POST
  - Vytvoř novou otázku
- PUT
  - Uprav danou otázku
- DELETE
  - Smaž jednu otázku
  - Smaž všechny otázky

6.3.3.1 GET – vrácení otázek

Ve FE aplikaci se hned po nahrání souborů odešle požadavek na vrácení všech otázek, jedná se o endpoint „*/api/{lang}/upload/{upload_id}/questions*“ zavolaný metodou GET. Parametr *lang* určuje, ve kterém jazyce má být uživatelské rozhraní a chybové či informativní hlášky a *upload_id* se převezme na straně klienta z paměti prohlížeče. Pokud bude požadavek pouze na jednu danou otázku, zavolá se podobný endpoint, akorát se přidá parametr *{id}*.

Nejdříve se provede výběr z databáze z tabulky *question* podle *upload_id* a vytvoří se objekt do kterého se vloží atributy, které vrátil výběr. Při požadavku na jednu otázku se do výběru z databáze přidá podmínka WHERE a vybere se otázka podle primárního klíče.

```
qArray.push({
  id: q.question_id,
  type: q.type,
  question: q.name,
  folder: uploadFile.rows[0].file_name,
  answers: []
})
```

Pro naplnění klíče *answers* se provede další výběr z databáze, tentokrát z tabulky *answer* podle *upload_id* a *question_id*. Server vrátí jeden velký JSON objekt, který má v klíči rows pole objektů s otázkami a v nich další pole objektů s odpověďmi. Statusové kódy se liší podle situace:

- 200 – nenastala žádná chyba a otázky se načetly
- 204 – nenastala žádná chyba, ale žádné otázky nejsou k dispozici
- 410 – požadovaný fond nebyl nalezen

Code	Details
200	Response body


```
{
  "status": 200,
  "poolName": "Otázky - Přednáška 11",
  "updatedAt": "2021-04-21T08:12:41.000Z",
  "rows": [
    {
      "id": 258,
      "type": "MULTIPLE_ANSWER",
      "question": "<p>Pod Managementemsítě je v nejobecnějším smyslu myšleno: </p>",
      "folder": "Pool_ExportFile_KIT-PSIT1-TEST_Otazky-Prednaska11_1618992761801",
      "answers": [
        {
          "id": 855,
          "name": "<p>Tvorba statistik a přehledů. </p>",
          "correct": true
        },
        {
          "id": 860,
          "name": "<p>Konfigurace sítě a řešení závad. </p>",
          "correct": true
        },
        {
          "id": 865,
          "name": "<p>Dozor nad službami v síti. </p>",
          "correct": true
        },
        {
          "id": 870,
```


Obr. 11 Zobrazení odpovědi pro vrácení otázek

6.3.3.2 POST – vytvoření otázky/odpovědi

K vytvoření otázky do nového fondu slouží endpoint „/api/{lang}/upload/{upload_id}/questions“ zavolaný metodou POST. Jako požadavek musí přijít JSON objekt s již vyplněnými daty na vytvoření otázky. Tento JSON se generuje ve FE aplikaci, kde uživatel má k dispozici formulář a několik

možností, aby si vytvořil správný typ otázky. V rámci Swaggeru se dá přednastavit template, jak má JSON vypadat a podle něho vyplnit požadovaná data.

Obr. 12 Příklad, jak má vypadat tělo požadavku

Po odeslání požadavku se tyto data na straně serveru zpracují a následně se vkládají do databáze. Podle typu otázky se naplní do databáze ostatní potřebná data, aby se shodovala s QTI 2.1 formátem.

```
 if (req.body.type == 'MULTIPLE_ANSWER') {
      cardinality = 'multiple'
      maxchoices = 0
    } else if (req.body.type == 'ESSAY' || req.body.type == 'FILL_BLANK')
 {
      basetype = 'string'
    } else if (req.body.type == 'TRUE_FALSE' || req.body.type == 'MULTIPL
E_CHOICE') {
      maxchoices = 1
    }
```

Pokud chce uživatel vložit obrázek, tak ten se zakóduje do Base64. Po úspěšném vložení server vrátí kód 200 a informativní hlášku, že vše proběhlo v pořádku.

U otázky vyžadující doplnění, pokud uživatel vloží do možných odpovědí obrázek, je potřeba na straně serveru tento obrázek odebrat včetně HTML tagů,

protože student nemůže poté v testu do odpovědi zadat obrázek. Toto je řešeno pomocí regulárního výrazu.

```
a.name = a.name.replace(/<([^>~]+)>/gi, '')
```

Funkce *replace* najde podle regulárního výrazu HTML tagy a nahradí je prázdným řetězcem.

6.3.3.3 PUT – aktualizace otázky/odpovědi

Aktualizace funguje na podobném principu jako vložení. Na server přijde požadavek, že se jedná o aktualizaci a s ním i JSON objekt obsahující kompletní data otázky a jejích odpovědí. Název a typ otázky se aktualizuje v databázi pomocí UPDATE, také je potřeba aktualizovat čas úpravy fondu v tabulce *upload_file*. Odpovědi se nejdřív z tabulky vymažou a poté se nahrají znovu upravené, je to kvůli tomu, že by bylo komplikované řešit každou otázku zvlášť.

6.3.3.4 DELETE – vymazání otázky/odpovědí

Uživatel může smazat buď to jednu otázku a s ní odpovědi, anebo všechny otázky s odpověďmi z fondu najednou. Pro smazání jedné slouží endpoint „/api/{lang}/upload/{upload_id}/questions/{id}“, pro všechny je název ednpointu stejný, až na poslední část, kde se zadává {id} otázky. Oba se volají metodou DELETE.

```
await updateTimeStamp(upload_id)

await pool.query('DELETE FROM bb_editor.answer WHERE upload_file_id = $1'
, [upload_id])
await pool.query('DELETE FROM bb_editor.question WHERE upload_file_id = $
1', [upload_id])
```

Nejdříve je potřeba aktualizovat datum úpravy fondu a následně vymazat všechny odpovědi a otázky. Kdyby se nejdřív vymazaly otázky, tak by došlo k databázové chybě kvůli relaci s odpověďmi.

6.3.3.5 Cron script pro mazání

Cron je softwarový démon, který v operačních systémech automatizovaně spouští nějaký proces (program, script), který má za úkol něco vykonat. Jedná se o plánovač úloh, jenž umožňuje opakované spouštění periodicky se opakujících procesů. [20]

Je potřeba myslet na to, aby databáze nebyla zbytečně naplněna daty, se kterými už nebude pracovat. Ve FE aplikaci je funkcionalita vrátit se k fondu, který byl v posledních 24 hodinách naposledy upravován. Je to možné tím, že je jeho id stále uloženo v paměti prohlížeče. Po 24 hodinách je tedy rozumné tento fond smazat. K tomu slouží script, který si pomocí databázového příkazu, vybere z databáze data fondů, u kterých nebyla v posledních 24 hodinách aktualizována hodnota *updated_at*.

```
const uploadFile = await pool.query('SELECT * FROM bb_editor.upload_file
WHERE updated_at < current_timestamp - interval \'24 HOURS\'')
```

Dále je potřeba projít složku s fondem, která je uložena fyzicky na disku a smazat jí. K tomu slouží modul *rimraf*, který po přijetí cesty ke složce začne mazat její obsah, a nakonec složku samotnou. V průběhu vykonávání funkčnosti smažeme obsah z tabulek *image, answer, question a upload_file*.

Na Heroku, kde jsou uloženy prostředí projektu (testovací, staging, produkční) lze přidat doplněk *Heroku Scheduler*, který podporuje spouštění scriptů na bázi Cronu. Zadá se, jaký script se má spouštět (v Node.js se před název scriptu musí napsat příkaz node) a časový interval.

Job	Dyno Size	Frequency	Last Run	Next Due
`$ node cron/delete-data.js`	Free	Hourly at :50	April 24, 2021 11:50 AM UTC	April 24, 2021 12:50 PM UTC

Obr. 13 Ukázka Heroku scheduler

6.3.4 Export fondu

Uživatel po úpravě fondu musí mít možnost tento fond exportovat do ZIP souboru a nahrát do systému Oliva. K tomu slouží endpoint „/api/{lang}/upload/{upload_id}/export/{pool_name}" zavolaný metodou GET. Z FE aplikace přijde požadavek v JSON objektu, kde včetně jazyka a *upload_id*, je i název fondu, který si uživatel může zvolit podle sebe. Zde je využit modul *xmlbuilder2* pro sestavení a vytvoření vlastních XML souborů a *express-easy-zip* pro vytvoření ZIP souboru obsahující XML soubory. *Xmlbuilder2* je obal kolem uzlů DOM, který přidává propojitelné funkce, které usnadňují vytváření a práci s dokumenty XML. [21]

Je potřeba vytvořit 3 typy souborů:

- **assessmentItem –** pro každou otázku jeden soubor
- **Question_bank** – seznam všech vygenerovaných assessmentItem souborů
- **Imsmanifest** – zaregistrování souboru assessmentItem a question_bank

6.3.4.1 Assessment item

Každá otázka s odpověďmi má svůj vlastní *assessmentItem* soubor. Z databáze se pomocí SELECT vyberou všechny otázky podle id souboru, ke kterému patří a v cyklu, pro každou zvlášť se začne vytvářet XML struktura souboru.

```
const root = create({ version: '1.0', encoding: 'UTF-8' })
        .ele('assessmentItem',
          {
            xmlns: 'http://www.imsglobal.org/xsd/imsqti_v2p1',
            'xmlns:ns9': 'http://www.imsglobal.org/xsd/apip/apipv1p0/imsa
pip_qtiv1p0',
            'xmlns:ns8': 'http://www.w3.org/1999/xlink',
            'xmlns:xsi': 'http://www.w3.org/2001/XMLSchema-instance',
            'xsi:schemaLocation': 'http://www.imsglobal.org/xsd/imsqti_v2
p1 http://www.imsglobal.org/xsd/qti/qtiv2p1/imsqti_v2p1.xsd',
            adaptive: 'false',
            timeDependent: 'false',
            identifier: q.identifier
          })
```

Nejdříve se vytvoří konstanta *root* a zde se uvede funkce *create*, která vytvoří počáteční XML element. Uvede se zde verze a kódování souboru. Na to navazuje funkce *ele*. Tato funkce v prvním parametru vytvoří element *„assessmentItem“* a ve druhém se za pomocí objektů doplňují atributy. Tyto atributy byly převzaty z QTI 2.1 formátu ze systému Oliva, proto jsou napsány na pevno a zůstanou vždy stejné. U atributu *identifier* se doplní z databázové tabulky *question* hodnota z atributu *identifier*.

Pro vytvoření potomka elementu *„assessmentItem“* se musí vytvořit nová konstanta a použít funkci *ele* na předchozí konstantu root.

```
const resDecla = root.ele('responseDeclaration',
  {
```

```
    cardinality: q.cardinality,
    baseType: q.basetype,
    identifier: 'RESPONSE'
  })
```

Element *„responseDeclaration"* bude tedy potomkem předchozího elementu a jeho atributy se opět převezmou z databázové tabulky *question*.

Ve FE aplikaci, u otázky vyžadující doplnění, má uživatel možnost vytvořit odpovědi, kde se dají zadat HTML tagy. Tento typ odpovědí má využití u předmětů TNPW1 a TNPW2, kde by mohla nastat otázka, že má student doplnit konkrétní HTML tagy. FE aplikace používá textový editor, který HTML tagy rovnou aplikuje a vizualizuje. Aby mohl uživatel, který odpovědi tvoří, vidět tyto tagy, tak musí zadat před každou špičatou závorku znak vlnovky. Například zápis bude vypadat takto: *„<~strong~>example text<~/strong~>".* Při exportu a vytváření *assessmentItem* souborů se každá otázka a odpověď projde a pomocí regulárního výrazu se špičaté závorky s vlnovkou nahradí obyčejnými špičatými závorkami.

```
a.identifier = a.identifier.replace(/(<(~))/gi, '<')
a.identifier = a.identifier.replace(/(~(>))/gi, '>')
```

Při nahrání fondu do aplikace, když se jedná o otázku vyžadující doplnění, tak tento proces funguje obráceně a každou špičatou závorku nahradí za špičatou závorku s vlnovkou.

```
item.$.mapKey = item.$.mapKey.replace(/</gi, '<~')
item.$.mapKey = item.$.mapKey.replace(/>/gi, '~>')
```

Pokud otázka/odpověď obsahuje obrázek, tak se musí projít celý název a zkontrolovat, zda obsahuje HTML tag *<img>* a jak je obrázek zapsaný. Když je v atributu *src* URL adresa obrázku, tak se nic neděje a pokračuje se dál, protože takto uložený obrázek Oliva podporuje. U typu, kde je obrázek zakódován do Base64 se obrázek musí dekódovat a uložit do složky fondu na disk a do atributu *src* přidat cestu k obrázku. Oliva bohužel nepodporuje Base64 kódování. Do databázové tabulky *image* vložíme název obrázku, id složky, id otázky a identifikátor obrázku. Pokud se obrázek nachází v odpovědi, tak přibude ještě id odpovědi. Následně se obrázek uloží na disk do aktuálně používaného fondu pomocí funkce *writeFileSync*.

Díky tomu, že jsou v databázi názvy otázek a odpovědí uložené včetně HTML tagů, tak *xmlbuilder2* dokáže z těchto tagů rovnou poskládat a vytvořit XML elementy.

Po vytvoření XML struktury, za pomocí funkcí, se musí proces ukončit funkcí *end* s parametrem *prettyPrint*. Tato funkce se přidá za počáteční konstantu root. Jelikož se XML struktura vytváří pro každý soubor zvlášť, tak se popořadě přidává do pole pro pozdější vytvoření souborů.

```
const xml = root.end({ prettyPrint: true })
      const i = index + 1
      qArray.push({
        name: (i.toString().length > 1) ? 'assessmentItem000' + i : 'asse
ssmentItem0000' + i,
        xml: xml
      })
```

6.3.4.2 Question bank

Tento soubor slouží jako seznam všech použitých *assessmentItem* souborů. Vytvoření XML struktury funguje stejně jak je popsáno v předešlé části za pomocí funkcí. Podle výběru z databázové tabulky *question*, se podle počtu záznamů vygenerují názvy *assessmentItem* a přidají se do atributů *identifier* a *href*. *Href* atribut slouží jako cesta, kde jsou soubory *assessmentItem* uložené – jsou uloženy ve stejné složce.

```
for (const [index] of questions.rows.entries()) {
      const i = index + 1
      let name = 'assessmentItem0000' + i
      if (i.toString().length > 1) {
        name = 'assessmentItem000' + i
      }
      root.ele('assessmentItemRef', {
        identifier: name,
        href: name + '.xml'
      })
    }
```

6.3.4.3 Imsmanifest

Tento soubor popisuje celou složku fondu. Jsou zde uvedené identifikátory a cesty souborů *question_bank*, *assessmentItem* a obrázky (pokud existují). Sestavení XML struktury je opět stejné jako je popsáno v předešlé části.

Element *„resources“* má pod sebou několik potomků jménem *„resource“*, kde se uvedou jednotlivé položky. Nejdříve se načtou všechny obrázky z tabulky *image* podle *upload_file_id* a jako atributy se zde uvedou identifikátory a cesty. Poté se zaregistruje soubor *question_bank* a do elementu *file* (typu potomek) se uvedou všechny *assessmentItem* soubory. Tyto soubory je potřeba zaregistrovat také zvlášť a pokud obsahují obrázek, tak se zde uvedou také. Příklad výsledku popsané části imsmanifest je uvedený na obr. 14:


```xml
<resources>
  <resource href="image00.jpg" identifier="ccres00001" type="webcontent">
    <file href="image00.jpg"/>
  </resource>
  <resource href="qti21/question_bank00001.xml" identifier="question_bank00001" type="imsqti_test_xmlv2p1">
    <file href="qti21/question_bank00001.xml"/>
    <dependency identifierref="assessmentItem00001"/>
    <dependency identifierref="assessmentItem00002"/>
    <dependency identifierref="assessmentItem00003"/>
    <dependency identifierref="assessmentItem00004"/>
    <dependency identifierref="assessmentItem00005"/>
    <dependency identifierref="assessmentItem00006"/>
    <dependency identifierref="assessmentItem00007"/>
    <dependency identifierref="assessmentItem00008"/>
    <dependency identifierref="assessmentItem00009"/>
    <dependency identifierref="assessmentItem00010"/>
    <dependency identifierref="assessmentItem00011"/>
    <dependency identifierref="assessmentItem00012"/>
    <dependency identifierref="assessmentItem00013"/>
    <dependency identifierref="assessmentItem00014"/>
  </resource>
  <resource href="qti21/assessmentItem00001.xml" identifier="assessmentItem00001" type="imsqti_item_xmlv2p1">
    <file href="qti21/assessmentItem00001.xml"/>
    <dependency identifierref="ccres00001"/>
  </resource>
  <resource href="qti21/assessmentItem00002.xml" identifier="assessmentItem00002" type="imsqti_item_xmlv2p1">
    <file href="qti21/assessmentItem00002.xml"/>
  </resource>
```


Obr. 14 Vygenerovaný imsmanifest soubor

6.3.4.4 Vytvoření ZIP souboru

Vygenerované XML struktury je potřeba vložit do nových souborů, jelikož modul *xmlbulder2* vytvoří pouze text. Modul *express-easy-zip* dokáže po přijetí JSON objektů s validními klíči, vytvořit soubory a rovnou je předat v odpovědi jako ZIP. Objekty, s informacemi pro vytvoření souboru, se přidávají do pole fileArray.

Klíč *content* očekává data, kterýma se má soubor naplnit, v tomto případě XML struktura *question_bank* načítající se z externí funkce. Klíč *name* očekává název nového souboru, pokud se zde vyplní název s lomítkem, tak z názvu před lomítkem

vytvoří složku. Poslední klíč *type* udává, jaký typ souboru má vytvořit, pokud se má vytvořit obrázek, tak se tento klíč musí odebrat.

```
fileArray.push({
  content: await questionBank.getQuestionBankXml(upload_id, pool_name),
  name: 'qti21/question_bank00001.xml',
  type: 'file'
})
```

Pro vytvoření a předání ZIP souboru se na serverové odpovědi (res) zavolá funkce *zip*, kde se vloží pole se soubory fileArray. Pomocí regulárního výrazu se ořeže název ZIP souboru o české znaky, protože modul *express-easy-zip* s nimi neumí pracovat.

```
await res.zip({
      files: fileArray,
      filename: uploadFile.rows[0].file_name.replace(/_[^_]+$/g, '').norm
alize('NFD').replace(/[\u0300-\u036f]/g, '') + '.zip'
    })
```

7 Shrnutí výsledků

Výsledků v této práce je několik. Implementace funkčního API se kterým se dá komunikovat, vytvořená a nahraná databáze na školním serveru, nebo validní nahrání testových fondů zpět do Olivy. Za hlavní výsledek se dá považovat, že spojení mé BE části s kolegovou FE částí tvoří jednu funkční aplikaci pro jednodušší editaci fondů pro systém Oliva. Tato aplikace by měla přinést vyučujícím mnohem jednodušší a rychlejší úpravu svých testových fondů.

Aplikace je dostupná na této URL adrese:

https://bb-editor.herokuapp.com

Swagger dokumentace na této URL na adrese:

https://bb-editor.herokuapp.com/api-docs

8 Závěry a doporučení

Cílem bakalářské práce bylo navrhnout a implementovat poskytovatele dat, se kterým bude komunikovat FE aplikace a spolu budou tvořit jeden funkční celek – aplikaci. Tato aplikace vznikla ve spolupráci s Michalem Petrasem. [22]

Při prvotním návrhu se ukázalo, že je potřeba důkladnější analýza, jak bude celá aplikace fungovat. Postupně jsme zkoumali a zkoušeli nové způsoby implementace až jsme došli k výsledku rozdělit aplikaci na BE a FE. Toto je možné vyčíst z prvních návrhů řešení, že směr, kterým jsme se zprvu vydali, nebyl správný, ale nakonec jsme našli vhodné řešení. Analýza Olivy ukázala, jak se pracuje s testovými fondy a podle toho se začala přizpůsobovat, v mém případě, BE část. Ukázalo se, že databáze je klíčovou částí a bez ní by funkčnost těžko byla proveditelná.

Při implementaci jsem si osvěžil technologie, se kterými jsem již v minulosti pracoval. Tyto technologie jsou Node.js, PostgreSQL, nebo organizovanou práci s Gitem. Naopak jsem se naučil nové způsoby, jak s těmito technologiemi pracovat, jedná se hlavně o využití balíčků, které je potřeba si nastudovat a aplikovat jejich funkčnost do vlastní práce.

Samotná implementace funkcionalit byla náročnější, hlavně v oblasti vymýšlení postupů a aby následně použité testové fondy, byly plně kompatibilní se systémem Oliva. U použitého balíčku *xml-stream* byla škoda, že nedokázal lépe převést data z XML na JSON u otázek/odpovědí, kde byly zanořené tagy. To vedlo ke složité úpravě JSON dat, aby výsledný název otázky/odpovědi byl pro uživatele přijatelný.

Výslednou aplikaci se podařilo vyvinout do stavu, ve kterém splňuje svůj účel a je schopna provozu na Univerzitě Hradec Králové. V průběhu vývoje a průběžného učení se ukázalo, že nějaké funkcionality by šly udělal jinak a lépe. Například by se mohlo využít oficiální API systému Oliva, které by zajišťovalo přihlášení vyučujícího a následně by vyučující viděl svoje fondy a mohl je rovnou do aplikace nahrát a pracovat s nimi. Nabízí se také, aby aplikace dokázala pracovat s ostatními typy otázek jako je například interaktivní otázka, kde uživatel musí kliknout na obrázek

a označit správně vybranou oblast. Do budoucna, nebo v další diplomové práci, bych na tuto aplikaci rád navázal a rozšířil ji o výše zmíněné návrhy.

9 Seznam použité literatury

[1] *REST: architektura pro webové API - Zdroják* [online]. [vid. 2021-04-04]. Dostupné z: https://zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[2] *Node.js – Wikipedie* [online]. [vid. 2021-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Node.js

[3] *Lekce 1 - Úvod do Node.js* [online]. [vid. 2021-04-04]. Dostupné z: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs

[4] *Co je to XML, k čemu se užívá a jaké jsou jeho hlavní výhody? | FastCentrik* [online]. [vid. 2021-04-26]. Dostupné z: https://www.fastcentrik.cz/blog/co-je-to-xml,-k-cemu-se-uziva-a-jake-jsou-jeho-vyh

[5] *JSON : jednotný formát pro výměnu dat - Zdroják* [online]. [vid. 2021-04-26]. Dostupné z: https://zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/

[6] *PostgreSQL – Wikipedie* [online]. [vid. 2021-03-31]. Dostupné z: https://cs.wikipedia.org/wiki/PostgreSQL

[7] *Blackboard Inc. - Wikipedia* [online]. [vid. 2021-04-04]. Dostupné z: https://en.wikipedia.org/wiki/Blackboard_Inc.

[8] *Pracovní postup Gitflow | Výukový program Atlassian Git* [online]. [vid. 2021-04-05]. Dostupné z: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[9] *DBeaver - Wikipedia* [online]. [vid. 2021-03-31]. Dostupné z: https://en.wikipedia.org/wiki/DBeaver

[10] *GitHub – Wikipedie* [online]. [vid. 2021-04-07]. Dostupné z: https://cs.wikipedia.org/wiki/GitHub

[11] *What is npm* [online]. [vid. 2021-04-14]. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp

[12] *Swagger (software) – Wikipedie* [online]. [vid. 2021-04-14]. Dostupné z: https://cs.wikipedia.org/wiki/Swagger_(software)#cite_note-:0-1

[13] *swagger-ui-express - npm* [online]. [vid. 2021-04-14]. Dostupné z: https://www.npmjs.com/package/swagger-ui-express

[14] *Heroku - Wikipedia* [online]. [vid. 2021-04-16]. Dostupné z: https://en.wikipedia.org/wiki/Heroku

[15] *Vítejte na serveru EDU* [online]. [vid. 2021-04-16]. Dostupné z: https://edu.uhk.cz/

[16] *Secure Shell – Wikipedie* [online]. [vid. 2021-04-16]. Dostupné z: https://cs.wikipedia.org/wiki/Secure_Shell

[17] *Welcome | node-postgres* [online]. [vid. 2021-04-14]. Dostupné z: https://node-postgres.com/

[18] *xml-stream - npm* [online]. [vid. 2021-04-21]. Dostupné z: https://www.npmjs.com/package/xml-stream

[19] *Base64 – Wikipedie* [online]. [vid. 2021-04-23]. Dostupné z: https://cs.wikipedia.org/wiki/Base64

[20] *Cron – Wikipedie* [online]. [vid. 2021-04-24]. Dostupné z: https://cs.wikipedia.org/wiki/Cron

[21] *xmlbuilder2 - npm* [online]. [vid. 2021-04-24]. Dostupné z: https://www.npmjs.com/package/xmlbuilder2

[22] *BBeditor – aplikace k editování fondů z BlackBoard Learn*. Závěrečná bakalářská práce – Kašpar Michal, FIM, UHK, 2021

10 Přílohy

Příloha č. 1

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Ondřej Kašpar**
Studium:	**I1700097**
Studijní program:	**B1802 Aplikovaná informatika**
Studijní obor:	**Aplikovaná informatika**
Název bakalářské práce:	**Poskytovatel dat (API) k frontend aplikaci pro správu fondů v systému BlackBoard Learn**
Název bakalářské práce AJ:	**API for FE Application for the BlackBoard Learn System Fonds Maintenance**

Cíl, metody, literatura, předpoklady:

Cílem závěrečné práce bude vytvoření poskytovatele dat (API) pro frontend aplikaci, která umožní jednodušší úpravu testových fondů vyexportovaných ze systému BlackBoard Learn (Oliva) ve formátu QTI 2.1. Aplikace bude přijímat od uživatele vyexportovaný soubor, který se na straně serveru zpracuje a převede do požadované podoby, aby se dále mohlo s daty pracovat a provádět na nich CRUD operace, následně ze systému vyexportovat a nahrát upravený soubor do Olivy. Data se budou ukládat do databáze vytvořenou v jazyce SQL. Aplikace bude nahrána na příslušný server, na který bude moci uživatel přistupovat. Bude kladen důraz na funkčnost zejména u předmětů Principy počítačů (PRIPO i KPRIP) a Architektura počítačů (ARCH i KARCH), ale i dalších.

V teoretické části práce bude provedena analýza systému BlackBoard, jak pracuje s otázkami, jak funguje proces vstupu a výstupu testovacích fondů a v jakém formátu je schopen přijímat cizí fondy, vytvořené z aplikací třetích stran.

Literatura bude poskytnuta zadavatelem

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	**prof. RNDr. Peter Mikulecký, Ph.D.**

Datum zadání závěrečné práce: **21.10.2019**