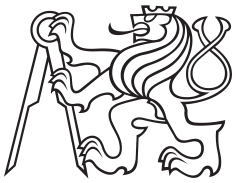
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Nástroj pro zadávání a správu zkouškových testů

Petr Ježek

Vedoucí: Ing. Karel Frajták, Ph.D.
Obor: Softwarové inženýrství a technologie
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Ježek** Jméno: **Petr** Osobní číslo: **474658**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Nástroj pro zadávání a správu zkouškových testů

Název bakalářské práce anglicky:

Exam test management tool

Pokyny pro vypracování:

Navrhněte a naimplementuje nástroj pro zadávání a správu zkouškových testů pro studenty.
Nástroj pomůže při organizaci zkouškových testů. Vyučující bude moci zadávat otázky (volný text, jedna odpověď je správna, více odpovědí je správných, apod.), seskupovat je dle definované kategorie, označovat štítky, definovat počet bodů za správně zodpovězenou otázku.
Nástroj sestaví test dle zadaného celkového počtu bodů za všechny otázky, celkového počtu otázek, kategorií a počtu otázek z každé kategorie.
Systém zaručí minimální opakování v testech zadávaných v jednom zkouškovém období. Nástroj upozorní obsluhu na nutnost doplnění otázek do zásobníku otázek v případě, že nelze další varianty testu sestavit.
Test bude pak možné vyexportovat do formátu pro import do Moodle nebo do PDF pro přímý tisk.
Vytvořená nástroj vhodným způsobem otestujte.

Seznam doporučené literatury:

https://docs.moodle.org/311/en/Quiz_activity

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Karel Frajták, Ph.D. laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Karel Frajták, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat svému vedoucímu panu doktorovi Karlu Frajtákovi za vstřícnost a trpělivost při spolupráci.

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně a veškeré použité zdroje jsou uvedeny v sekci Literatura na konci této práce v souladu s Metodickým pokynem č. 1/20.

Abstrakt

Cílem této práce je navrhnout, implementovat a otestovat nástroj pro zadávání a zprávu zkouškových testů. V nástroji lze generovat zadání z vybraných typů a kategorií otázek a exportovat ho ve zvoleném formátu pro Moodle, či v PDF pro přímý tisk. Obsahem práce je analýza zadávání testů v Moodle a jeho podporovaných formátů, návrh aplikace, výběr vhodných technologií a popis implementace a testování...

Klíčová slova: Nástroj, Webová aplikace, Moodle, Generátor

Vedoucí: Ing. Karel Frajták, Ph.D. Praha, Resslova 9, E-221 (vchod Karlovo náměstí 13)

Abstract

The goal of this thesis is to design, implement and test the exam test management tool. The tool can be used to generate tests from selected types and categories of questions and export them in a format supported by Moodle or PDF for print. This work contains analysis of the exam test creation process, Moodle and its supported formats, the design of the application as well as description of the technologies used, implementation details and the testing procedure...

Keywords: Tool, Web Application, Moodle, Generator

Title translation: Exam test management tool

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Primárním smyslem tohoto nástroje je usnadnit jeho uživatelům tvorbu zkouškových testů. Aktuální zadávání probíhá buď manuálně, nebo přes systém Moodle. Obě varianty jsou pracné a je zde mnoho prostoru pro vylepšení tohoto procesu.

V rámci této práce se nejprve zaměříme na pochopení principu zadávání zkouškových testů. Na základě provedené analýzy dále zkusíme najít vhodná již existující řešení, která splňují naše požadavky. Existující řešení rozebereme a porovnáme, případně identifikujeme nedostatky a navrhneme možná vylepšení. S těmito informacemi pak navrhneme řešení vlastní. Pro zvolené řešení vybereme vhodné technologie a provedeme implementaci. Aplikaci vhodným způsobem otestujeme.

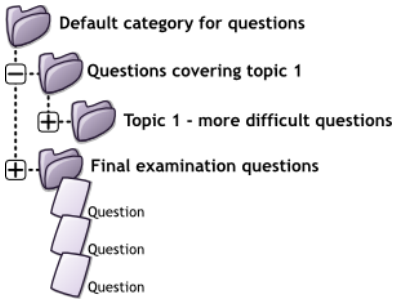
Kapitola 2
Analýza trhu

2.1 Tvorba zkouškových testů

Tvorba zkouškových testů je obecně pracná. Osoba, která zadání vytváří si většinou vede jakousi banku otázek, kterou využívá pro sestavení různých variant testu. Pro stejný termín zkoušky se test liší například pouze v pořadí otázek. Pro různé termíny zkoušky by měla zadání obsahovat zcela jiné otázky. Každá otázka má zpravidla textový popis, případně doplňující obrázky, správné a nesprávné odpovědi a bodové hodnocení. Otázky mají obvykle několik typů, například podle možných odpovědí - Ano/Ne, výběr jedné správné, výběr více správných, atp. Test se obvykle sestavuje z otázek s určitým bodovým hodnocením - například dvě otázky za tři body, jedna za šest a jedna za osm bodů.

2.2 Moodle

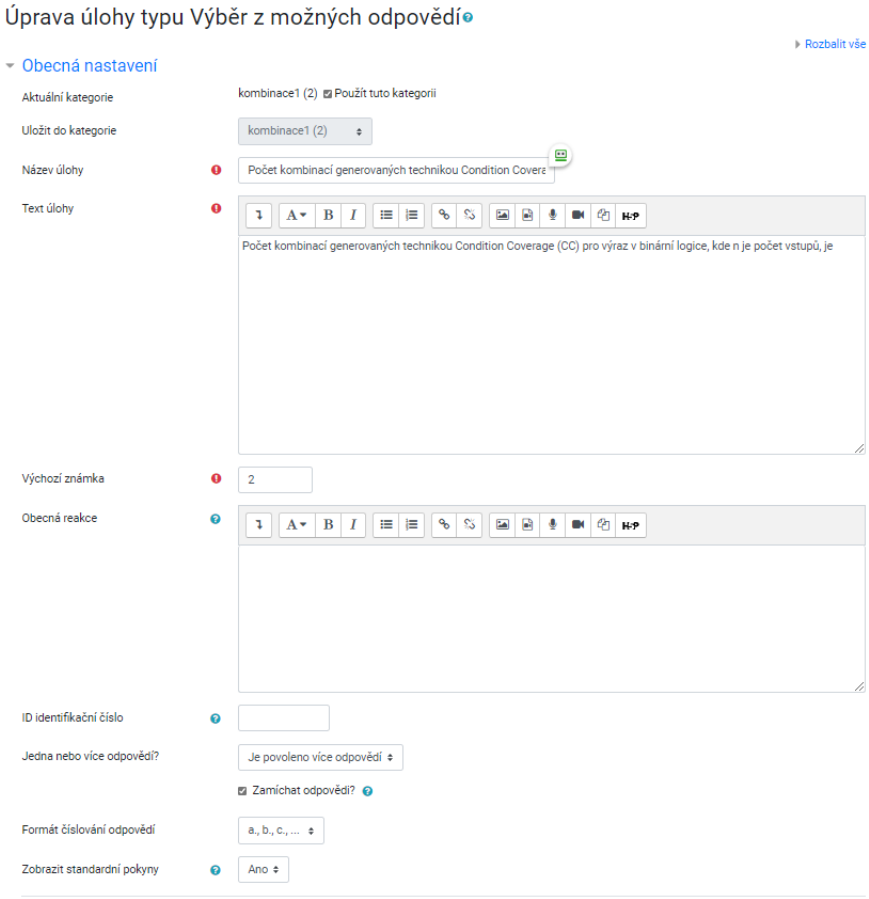
Zadávání zkouškových testů v Moodle probíhá obdobně ke způsobu popsanému výše [2]. Moodle umožňuje vytvářet kategorie a k nim úlohy. Kategorie tvoří stromovou strukturu, kterou lze vidět na obrázku 2.1.



Obrázek 2.1: Stromová struktura kategorií otázek v aplikaci Moodle

Úlohy mají kategorii, název, text, případně multimediální obsah, výchozí známku a možnost nastavit jednu či více odpovědí a jejich číslování. Odpovědi lze také automaticky zamíchat. Podle zvoleného typu odpovědi lze pak nastavit

odpovědi a jejich vliv na výslednou známku [1]. Ukázku uživatelského rozhraní editace úlohy lze vidět na obrázku 2.2.



Obrázek 2.2: Editace úlohy v aplikaci Moodle

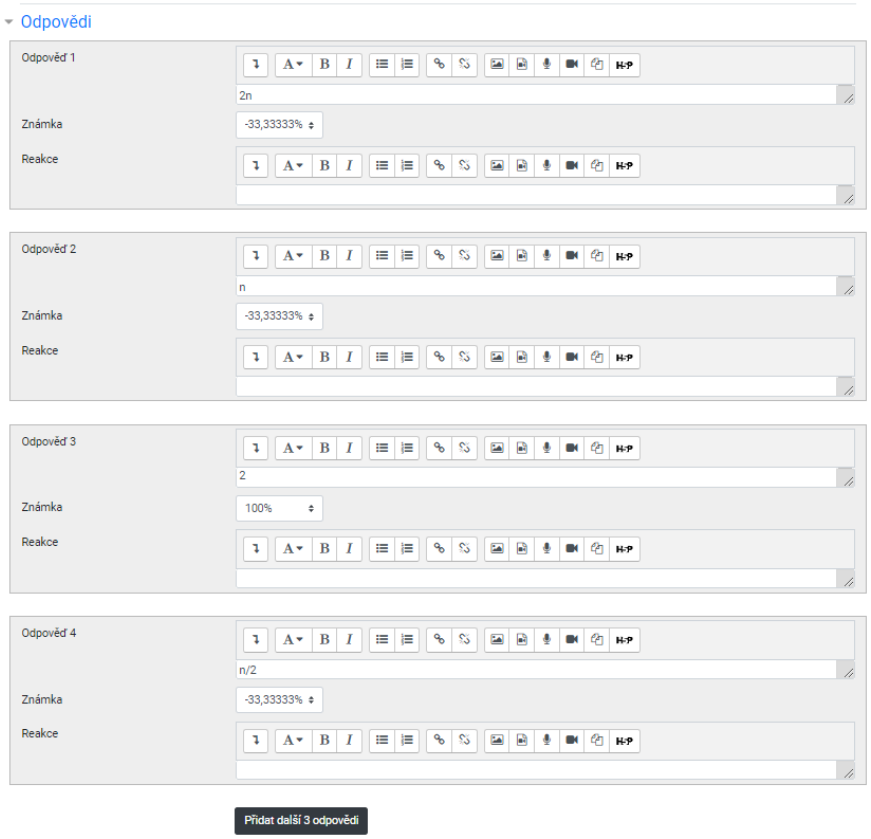
Známka je udávána v počtu bodů a vliv odpovědí na tuto známku je udáván v procentech. Pokud je jedna správná odpověď, tato odpověď bude mít vliv 100 procent a nesprávné odpovědi budou mít rovnoměrně rozděleno -100% - tedy pro 3 nesprávné odpovědi bude každá nesprávná odpověď mít -33.3333%. Editaci odpovědí ukazuje obrázek 2.3.

Z uživatelského pohledu je právě nastavování vlivu odpovědí na výsledné bodové hodnocení jednou ze slabých stránek zadávání testů v aplikaci Moodle.

2.2.1 Tvorba testů

Testy se v Moodle tvoří z otázek definovaných v bance úloh. Nejprve je třeba specifikovat kategorii, pod kterou se otázky nachází. Dále lze specifikovat jeden či více štítků, podle kterých chceme otázky filtrovat. Samotné otázky se pak vybírají pomocí zaškrtávacích polí.

[1] Dokumentace k otázkám v Moodle je dostupná na https://docs.moodle.org/311/en/Question_categories



Obrázek 2.3: Editace odpovědí v aplikaci Moodle

Hlavní nevýhodou způsobu, jakým Moodle řeší přidávání otázek je jejich výběr. Není například možné zadat, kolik otázek za určitý počet bodů se má do testu přidat. Z výběru ani není patrné, jaké bodové hodnocení vybíraná otázka má. Také se ve výběru neustále zobrazují všechny otázky.

2.2.2 Import a export

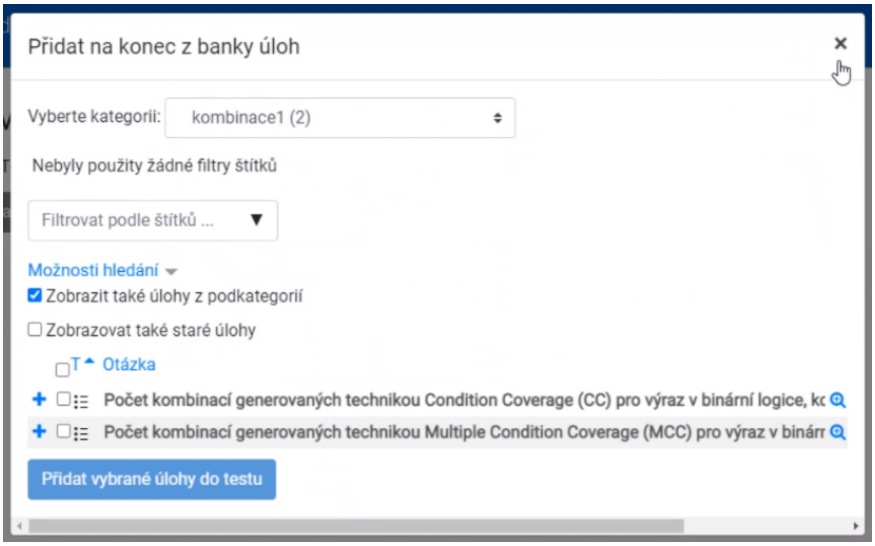
Moodle umožňuje import a export testových otázek. Lze exportovat libovolnou kategorii a s ní všechny podkategorie a otázky. Moodle dovoluje také import a export samotných otázek [2].

2.2.3 Podporované formáty

Moodle podporuje několik formátů uvedených ve vlastních odstavcích níže. Kategorie s otázkami lze v těchto formátech jak importovat tak exportovat.

GIFT. General Import Format Template, ve zkratce GIFT, se vyznačuje jednoduchou syntaxí, která obsahuje minimum přidaných znaků a tvoří ji

[2] Dokumentace importu a exportu otázek v Moodle je dostupná na https://docs.moodle.org/311/en/Managing_questions



Obrázek 2.4: Výběr testových otázek v aplikaci Moodle

převážně samotná data. Formát je známý primárně díky využití v Moodle a je určen pro snadné vytváření otázek typů: více odpovědí, ano-ne, či s krátkou textovou nebo číselnou odpovědí v textovém editoru [3]. Na počátku souboru je definována kategorie, do které budou následující otázky spadat. Poté následují otázky s jejich odpověďmi, jak lze vidět na obrázku 2.5

```
// question: 0  name: Switch category to $course$/top/mojeKategorie
$CATEGORY: $course$/top/mojeKategorie

// question: 235277  name: Jmeno otazky
// [tag:mujTag]
::Text otazky::{
        ~%-100%<p>Spatna odpoved</p>
        ~%100%<p>Spravna odpoved</p>
}
```

Obrázek 2.5: Ukázka struktury formátu GIFT

Výhody formátu GIFT jsou především v jednoduchosti a čitelnosti pro lidské oko. Otázky lze zadávat snadno pomocí textového editoru. Formát ovšem není zdaleka tak rozšířený jako jeho alternativy a neexistují prověřené parsery [4], mimo několika menších hobby projektů [5].

Moodle XML. Moodle XML je formát běžného XML, pouze s tagy specifickými pro Moodle. Data jsou uspořádána ve stromové struktuře definované

[3] Dokumentaci formátu GIFT lze nalézt na https://docs.moodle.org/311/en/GIFT_format

[4] Parser je program, který čte a kontroluje syntaktickou správnost dokumentu

[5] Jedním z parserů je GIFT escaper dostupný na http://selimtemizer.com/software/gift/

```
<?xml version="1.0" encoding="UTF-8"?>
<quiz>
  <!-- question: 0  -->
  <question type="category">
    <category>
      <text>$course$/top/mojeKategorie</text>
    </category>
    ...
  </question>

  <!-- question: 235277  -->
  <question type="multichoice">
    <name>
      <text>Text otazky s vice odpovedmi</text>
    </name>
    <questiontext format="html">
      <text>...</text>
    </questiontext>
    ...
  </question>
</quiz>
```

Obrázek 2.6: Ukázka formátu Moodle XML

značkami - tagy. Ukázku XML formátu lze vidět na obrázku 2.6. Z ukázky je patrná jedna z nevýhod formátu Moodle XML. Kategorie jsou definovány pomocí značky `<question type="category"/>`. Za takto definovanou kategorií následují otázky spadající v hierarchii pod tuto kategorii. Formát je tak méně přehledný a práce s ním náročnější, kvůli nutnosti implementace algoritmu pro čtení této struktury.

Struktura XML je výhodná zejména díky nespočtu existujících knihoven psaných v populárních programovacích jazycích jako je Java, které práci s tímto formátem umožňují. XML se navíc obecně používá jako formát pro přenos strukturovaných dat a jeví se pro naše účely jako vhodný kandidát.

Další nevýhodou tohoto formátu je, že je hůře čitelný a obsahuje množství pro lidské oko nepotřebných informací.

XHTML. Formát XHTML v porovnání se zmíněnými alternativami obsahuje největší množství nepotřebných informací. Zároveň v něm spoustu informací chybí. Jeho výhoda je primárně v tom, že je formát implicitně čitelný v prohlížeči. Umožňuje tedy například použít otázky nadefinované v Moodle v nějaké vlastní webové aplikaci. Úlohy jsou zde formátovány čitelně a jsou využity HTML elementy `<checkbox>` a `<textfield>`. Tyto elementy jsou navíc rovnou umístěny uvnitř formulářového bloku, který lze jednoduše upravit tak, aby odesílal zvolené odpovědi na zvolený server. Strukturu lze vidět na obrázku 2.7.

```
<form action="...REPLACE ME..." method="post">
  <!-- question: 235277  name: Jmeno otazky -->
  <div class="question">
    <h3>Jmeno otazky</h3>
    <p class="questiontext">Text otazky</p>
    <ul class="multichoice">
      <li>
        <input name="quest_23" type="checkbox" value="spravne" />
        <p>Spravna odpoved</p>
      </li>
      <li>
        <input name="quest_24" type="checkbox" value="spatne" />
        <p>Spravna odpoved</p>
      </li>
    </ul>
  </div>
  <p class="submit">
    <input type="submit" />
  </p>
</form>
```

Obrázek 2.7: Ukázka formátu XHTML

Možnosti, které XHTML nabízí z našeho pohledu nejsou příliš atraktivní, vzhledem k tomu že hledáme vhodný formát exkluzivně pro přenos informací mezi naším nástrojem a aplikací Moodle.

Shrnutí. Ze zmíněných formátů v této práci zvolíme formát XML pro import a export z Moodle, zejména kvůli již existujícím knihovnám, které práci s XML umožňují.

2.3 Existující nástroje

Aplikací pro zadávání online testů již na trhu existuje více. V rámci této práce se podíváme na 3 vybrané aplikace a porovnáme jejich funkcionalitu. Exempláře byly vybrány z patnácti nejlépe hodnocených nástrojů podle Capterra, 2021. [3]

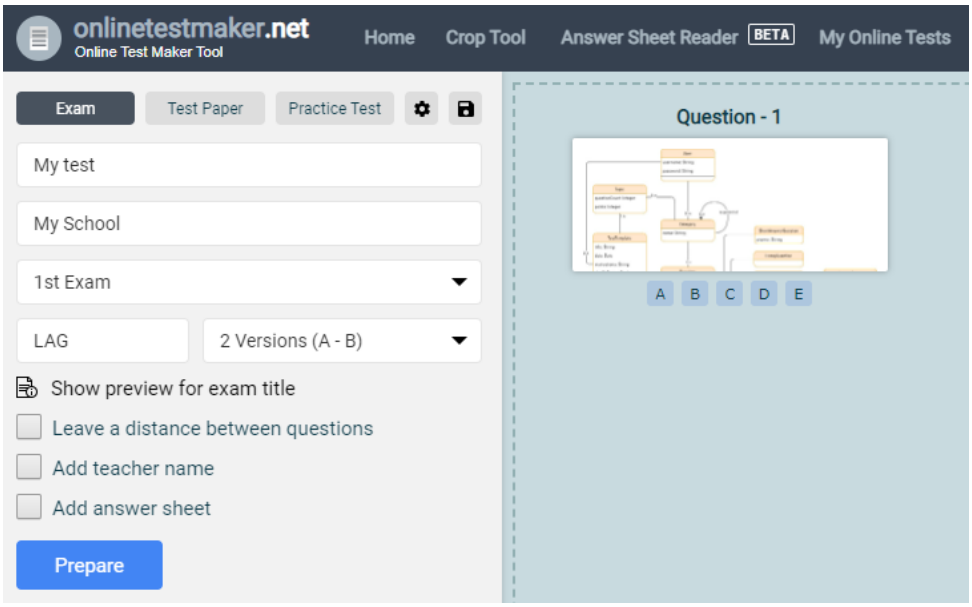
2.3.1 Online Test Maker

Online test maker působí na první pohled stroze [6]. Po jednoduché registraci je uživateli k dispozici několik základních funkcí

- Možnost vytvořit test s vlastním názvem, popisem, názvem tématu a školy.

[6] Online Test Maker je k dispozici na https://onlinetestmaker.net

- Do testu lze přidávat otázky ve formátu PNG, JPG a GIF, tedy pouze formou obrázků.
- Každá otázka má 5 možností odpovědi - A, B, C, D, E, které nelze nijak dále specifikovat
- Test lze vygenerovat v 1, 2, maximálně 4 verzích s jiným pořadím otázek
- Na konec testu lze přidat nápovědu se správnými řešeními
- Test lze stáhnout ve formátu PDF, odeslat emailem či publikovat jako online test
- V případě publikace jako online test lze zadat časový limit, začátek a konec testu, případně jestli se mají na konci testu zobrazit správná řešení



Obrázek 2.8: Tvorba testu v Online Test Makeru

Nástroj je velmi jednostranný a má omezenou funkcionalitu. Nelze například vytvářet kategorie otázek a specifikovat maximální počet získaných bodů. Hodnocení je binární, tedy splnil/nesplnil. Počet správných možností se také nedá měnit. Vygenerované testy se liší pouze pořadím otázek a vizuálně tvoří jakýsi seznam očíslovaných obrázků.

Tento nástroj se jeví vhodný například pro zadávání testů na základních školách nebo pro případy, kdy specificky potřebujeme zadávat otázky formou obrázků.

2.3.2 Speedexam

Tento nástroj je na rozdíl od předchozího více sofistikovaný [7]. Po registraci a ověření přes emailový odkaz může uživatel mimo jiné provádět následující:

- vytvářet vlastní otázky formou
  - otázky s výběrem a jednou správnou odpovědí
  - otázky s výběrem a více správnými odpověďmi
  - otázky s doplňováním slov do textu
  - otázky s přiřazováním
  - otázky se slovní odpovědí
  - otázky s odpovědí ano/ne
- definovat otázky formou textu s obrázky, přidávat matematické rovnice a tabulky
- definovat libovolný počet odpovědí formou textu, obrázků a matematických rovnic a jejich individuální hodnocení
- přidat vysvětlení ke správným odpovědím
- vytvářet vlastní zkouškové testy z definovaných otázek, buď náhodně nebo manuálně
- nastavit minimální počet bodů v procentech
- nastavit míchání otázek a odpovědí
- exportovat test do PDF či ho publikovat online

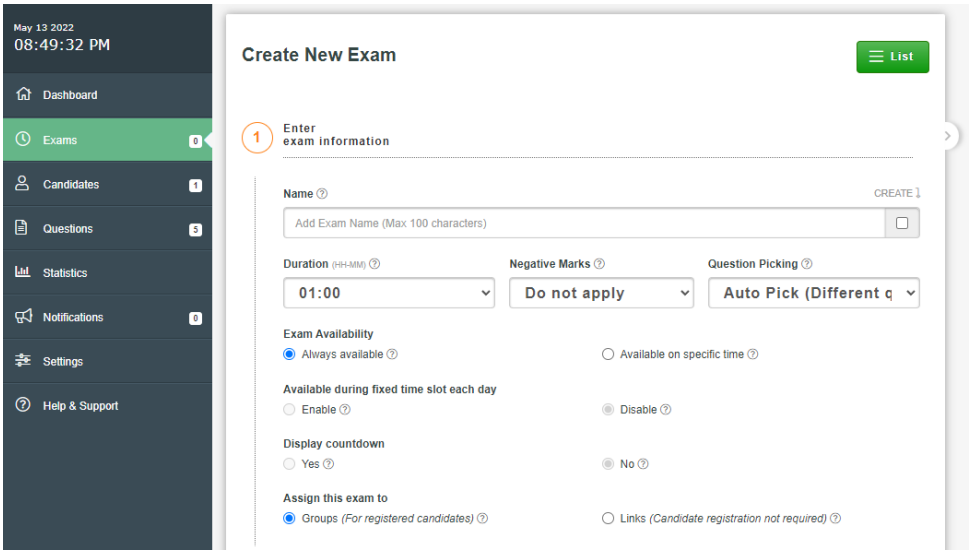
Nástroj Speedexam je v porovnání tedy o něco pokročilejší. Vítaná je možnost zadávat různé typy otázek. Hodnocení je tu také vyřešeno podstatně lépe než u předchozího příkladu. Je tu i možnost vytvořit si banku otázek a přidávat je do množiny, ze které se bude test generovat. Stále tu ovšem chybí například možnost vytvářet kategorie otázek nebo import či export ve formátech jako je XML.

2.3.3 Online Exam Builder

Tento nástroj je z našich ukázek nejpokročilejší. Je také cílen primárně na firmy a školy, nikoliv na jednotlivce. Nástroj je to placený, nicméně poskytuje sedmi denní demo verzi zdarma [8], v rámci které byla provedena tato analýza. Po registraci a přihlášení jsou uživateli k dispozici možnosti:

- vytvořit kategorie testů a otázek, z otázek vytvářet banku

[7] Nástroj Speedexam lze najít na adrese https://www.speedexam.net/
[8] Online Exam Builder je dostupný na adrese https://www.onlineexambuilder.com/



Obrázek 2.9: Tvorba testu v nástroji Speedexam

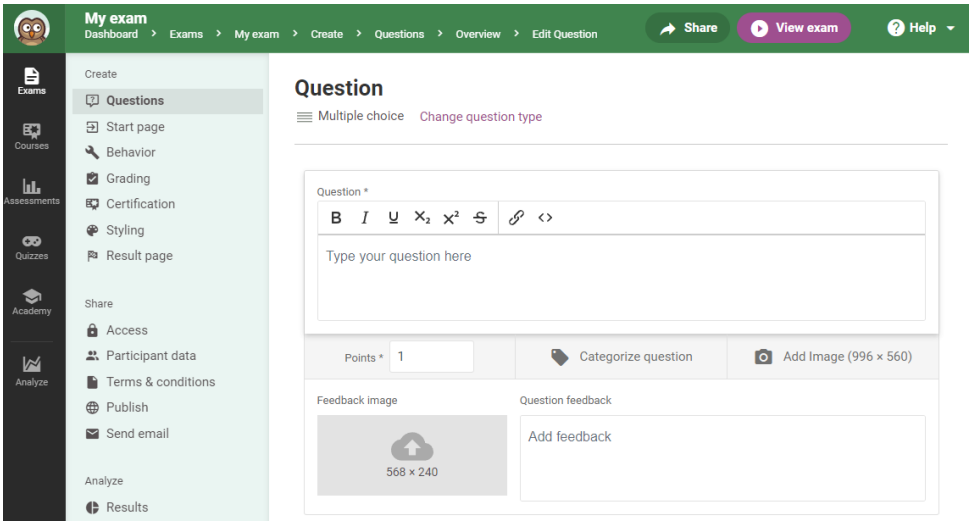
- ke kategoriím vyvářet jednotlivé otázky s podobným výběrem odpovědí jako u nástroje Speedexam
- lze specifikovat úvodní obrazovku či záhlaví testu
- lze specifikovat bodové hodnocení jednotlivých otázek a celkové hodnocení testu, také lze specifikovat minimální body v procentech
- k dispozici je také konfigurace vzhledu testu, od typu fontu až po barevné prvky a rozložení
- otázky lze importovat a exportovat ve formátu ODB[9] nebo XLSX
- Hotový test lze sdílet v online formě, není k dispozici export testu do formátu typu PDF ani XML

Poslední nástroj zaujal možností úpravy vzhledu testů a přívětivým uživatelským rozhraním. Nechyběl ani export, i když pouze do dvou formátů, z nichž ani jeden nebyl XML či PDF. Nástroj se soustředí primárně na tvorbu online testů a neposkytuje možnost stáhnout test v tisknutelné podobě. Cílovou skupinou jsou primárně malé a střední firmy, které potřebují nástroj pro tvorbu testů pro školení a další podobné případy.

2.3.4 Shrnutí

Vybrané příklady nástrojů nabízí dobré množství funkcionality a některé i zcela zdarma. Každý nástroj je zřejmě cílen na jinou skupinu uživatelů. Všem

[9]Soubory ODB obsahují archiv datových souborů, slouží u ukládání strukturovaných informací jako jsou tabulky a záznamy, soubor lze vytvořit v Apache OpenOffice Base



Obrázek 2.10: Tvorba testu v nástroji Online Exam Builder

třem nástrojům ovšem zcela chybí možnost exportu do formátů podporovaných aplikací Moodle. Stejně tak není možné vytvářet kategorie otázek, ze kterých by se generoval finální zkouškový test. Nástroj implementovaný v rámci této práce toto umožňovat bude. Naopak některé funkce, jako nastavitelný vzhled zkouškového testu, jsou mimo rozsah tohoto projektu.

Kapitola 3

Analýza a návrh aplikace

V této části se zaměříme již na samotnou aplikaci. Začneme sběrem požadavků, podle kterých připravíme případy užití a doménový model. Dále navrhneme high-level architekturou aplikace. Poté se soustředíme na komunikaci mezi jednotlivými částmi aplikace a na závěr zvolíme technologie pro jednotlivé části aplikace a způsob nasazení.

3.1 Funkční požadavky

V rámci analýzy jsme identifikovali seznam funkčních požadavků, které aplikace bude splňovat. Jsou rozděleny na kategorie, podle typu uživatele.

3.1.1 Požadavky pro nepřihlášeného uživatele

- nepřihlášený uživatel má možnost se zaregistrovat
- nepřihlášený uživatel se může po registraci přihlásit

3.1.2 Požadavky pro přihlášeného uživatele

Aplikace umožní přihlášenému uživateli

- vytvářet kategorie
- upravovat kategorie
- mazat kategorie
- zobrazit obsah kategorie strukturovaně
- vytvářet ke kategoriím podkategorie
- vytvářet otázky ke zvoleným kategoriím a specifikovat bodové ohodnocení
- otázky budou mít formu
  - otázky s textovou odpovědí
  - otázky zaškrtávací s jednou či více správnými odpověďmi

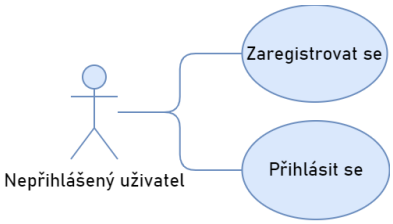
- vytvářet testové šablony z vybraných kategorií otázek
- u šablony specifikovat množství otázek za určitý počet bodů
- z šablony vygenerovat co nejvíce různých variant testového zadání s minimálním překryvem otázek mezi variantami, aplikace uživatele upozorní pokud nelze vygenerovat další varianty
- exportovat vygenerovaná zadání do formátu PDF pro tisk
- exportovat vygenerovaná zadání do formátu XML pro import do Moodle
- importovat kategorii včetně otázek a hierarchie z Moodle ve formátu XML

3.2 Případy užití

Podle funkčních požadavků sestavíme diagramy případů užití pro

- Nepřihlášené uživatele
- Přihlášené uživatele

Diagramy znázorňují přihlášeného a nepřihlášeného uživatele jako dva aktéry a zachycují jejich přístup k jednotlivým funkcím aplikace.



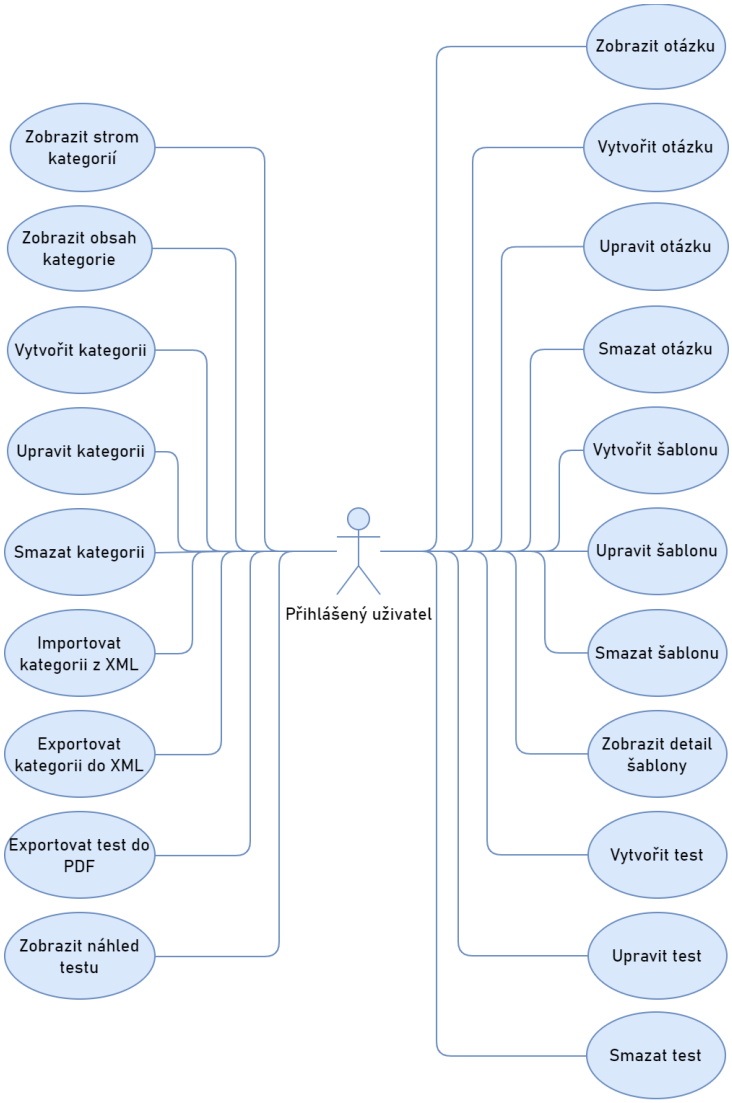
Obrázek 3.1: Případy užití pro anonymního uživatele

3.3 Doménový model

Doménový model zachycuje vztahy mezi entitami v aplikaci a jejich obsah včetně datových typů. Slouží jako základ pro definici datové vrstvy aplikace.

3.4 Generování kombinací testových otázek

Nedílnou součástí tohoto nástroje je i generování kombinací testových otázek. K dispozici máme množinu N otázek a cílem je vygenerovat k termínů zkouškového testu. V ideálním případě by se u různých termínů zkouškového testu neopakovala žádná otázka. Brát zřetel musíme také na fakt, že všechny termíny testu by měly mít stejný celkový počet bodů a stejné by měly být i

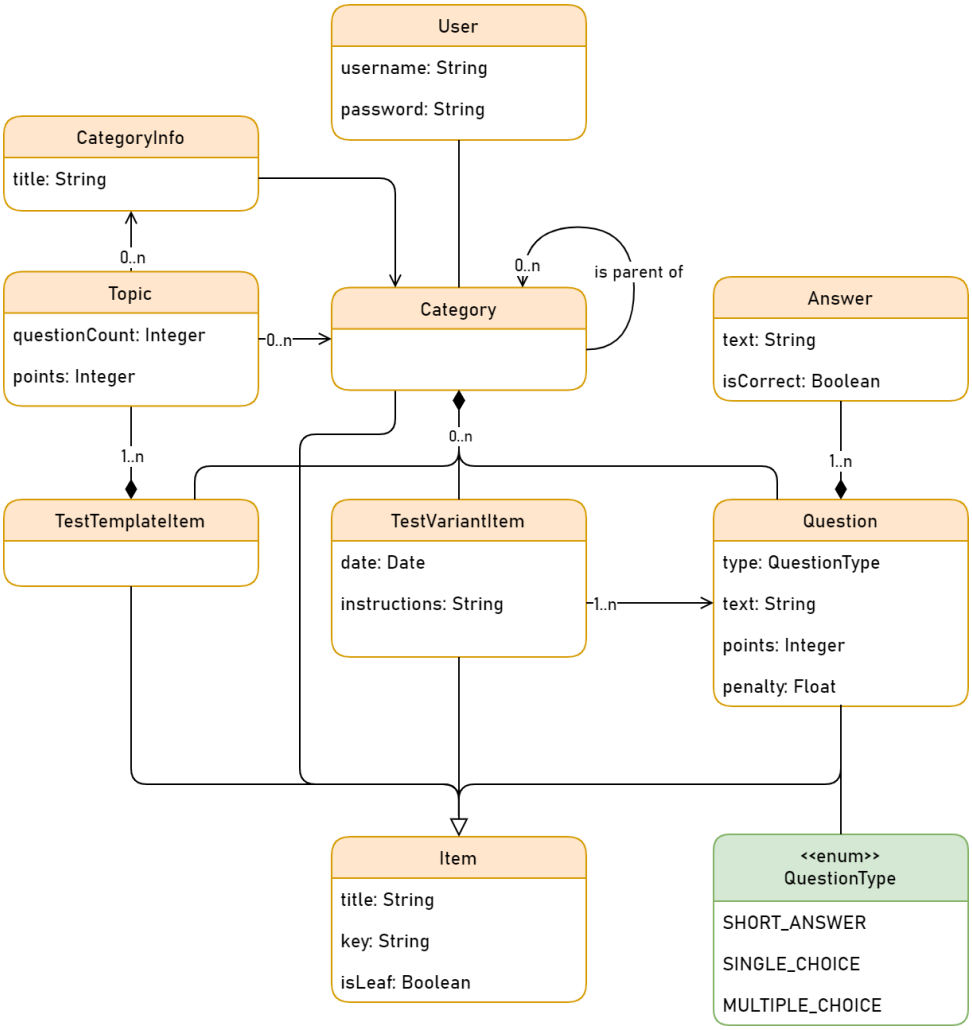


Obrázek 3.2: Případy užití pro přihlášeného uživatele

počty otázek za určitý počet bodů. Nemělo by se například stát, že jeden test bude mít 5 otázek za 3 body, a druhý 3 otázky za 5 bodů. V takovém případě by se totiž dalo argumentovat, že je jeden test více obtížný než druhý.

3.4.1 K-partition Problem

K-partition problém se v mnohém podobá našemu problému generování unikátních testových variant. V tomto problému máme rozdělit pole kladných celých čísel na k disjunktních množin, které mají stejný součet a zcela vyčerpají zdrojovou množinu. Problém k-partition je speciálním případem partition problému, který hledá pouze dvě takové množiny. V našem případě je množina kladných čísel nahrazena otázkami a jako celé číslo tu figuruje bodové ohodnocení dané otázky. Vyřešením tohoto problému dostaneme k unikátních



Obrázek 3.3: Doménový model aplikace

kombinací otázek, kde všechny kombinace mají stejný součet bodů a zároveň vyčerpáme všechny otázky, které máme k dispozici. *k*-partition problém má zásadní nedostatek, kterým je fakt, že pro mnoho případů neodkážeme nalézt řešení [1]. Příklad takového případu lze vidět na obrázku 3.4. Pro nás by to znamenalo, že bychom z některých kategorií nemohli daný počet testů vůbec vygenerovat. Také nelze specifikovat, kolik otázek za jaký počet bodů v testu požadujeme.

3.4.2 Kombinace

Kombinace poskytují opačný pohled na náš problém. V generování testů chceme nejen minimalizovat opakování otázek, ale zároveň získat maximální

[1] Implementace algoritmu řešící k-partition problém použitá v příkladu je dostupná na https://www.techiedelight.com/k-partition-problem-print-all-subsets/

```
# (<jméno otázky>, <počet bodů>)
S = [
    ("q1", 2), ("q2", 2), ("q3", 2)
    ("q4", 5),
]
k = 2
```

Obrázek 3.4: Příklad množiny kde nedokážeme nalézt 2 partitions se stejným součtem bodů

počet vygenerovaných variant. Všechny kombinace otázek nám poskytnou maximální počet různých testů. Abychom dostali správný počet otázek za určitý počet bodů, rozdělíme množinu všech otázek na podmnožiny podle počtu bodů. Z každé množiny poté utvoříme všechny kombinace. Finální složení testu bude obsahovat kombinaci těchto kombinací. Hlavním problémem této metody je, že nijak neredukuje překryv otázek mezi vygenerovanými testy. Podobně jako u problému k-partition toto řešení tedy splňuje pouze jednu z našich podmínek.

3.4.3 Kombinatorické testovací metody

Během analýzy algoritmů pro generování kombinací otázek byl mezi výsledky hledání i Github repozitář jesg/dither. [8] V tomto repozitáři jsou k dispozici algoritmy pro generování kombinací, které využívají principy kombinatorických testovacích metod, konkrétně metody z kategorie covering arrays [4].

3.4.4 Analogie s pairwise testováním

Pairwise testing, či obecně n-wise je metoda používaná v testování software pro generování testovacích scénářů s využitím covering arrays. Je založena na pozorování, že většina chyb vzniká vlivem nejvýše dvou faktorů, respektive kombinací dvou vstupních hodnot. Cílem je minimalizovat celkový počet testovacích scénářů a získat všechny kombinace vstupů s unikátními dvojicemi. Tím dosáhneme důkladného otestování i s malým počtem testovacích scénářů. Tato metoda je součástí kombinatorických testovacích metod, které mimo páry generují vstupy až s unikátními N-ticemi. Tyto metody sice nezaručí, že se jednotlivé vstupy ve vygenerovaných kombinacích nebudou opakovat, ale zaručí minimální opakování stejných N-tic vstupů [2].

Generování kombinací testových otázek metodou pairwise je střední cestou mezi generováním striktně disjunktních množin otázek v případě K-parition a generování všech kombinací. Získáme tak dostatek kombinací a zároveň redukujeme překryv otázek mezi různými testovými variantami. Další výhodou tohoto přístupu je, že nám rovnou umožňuje generovat kombinace otázek s

[2]Popis a nástroje pairwise testování jsou uvedeny na https://www.pairwise.org/

```
Object[][] questions = new Object[][]{
        // první otázka za 3 body
        new Object[]{"3b_q1", "3b_q2"},
        // druhá otázka za 3 body
        new Object[]{"3b_q4", "3b_q5", "3b_q6"},
        // otázka za 5 bodů
        new Object[]{"5b_q1", "5b_q2"},
        // otázka za 8 bodů
        new Object[]{"8b_q1", "8b_q2"}
};
Object[][] variants = Dither.aetg(t, questions);
```

Obrázek 3.5: Příklad generování kombinací otázek pomocí Dither knihovny v jazyce Java

variant / eq. class	eq 1	eq 2	eq 3	eq 4
v1	3b_q1	3b_q6	5b_q2	8b_q1
v2	3b_q2	3b_q4	5b_q1	8b_q2
v3	3b_q1	3b_q5	5b_q1	8b_q1
v4	3b_q2	3b_q5	5b_q2	8b_q2
v5	3b_q1	3b_q6	5b_q1	8b_q2
v6	3b_q2	3b_q4	5b_q2	8b_q1
v7	3b_q2	3b_q6	5b_q2	8b_q1
v8	3b_q1	3b_q4	5b_q1	8b_q2

Obrázek 3.6: Vygenerované varianty se zvýrazněnými páry otázek

ohledem na jejich bodové ohodnocení. Otázky rozdělíme do kategorií podle bodového hodnocení do tříd ekvivalence. Tyto třídy ekvivalence použijeme jako vstup pro pairwise algoritmus, jak lze vidět na obrázku 3.5. Výstupní kombinace pak budou z každé třídy obsahovat jednu otázku. Jednotlivé páry otázek u vygenerovaných variant jsou zvýrazněny na obrázku 3.6. V řádcích znázorněné tabulky jsou vygenerované varianty, ve sloupcích otázky z jednotlivých tříd ekvivalence. Barvy rozlišují dvojice tříd ekvivalence ze kterých jsou páry vybrány.

Pro generování kombinací testovacích vstupů existuje řada nástrojů. Nástroje se liší použitými algoritmy a cílem, který se snaží naplnit. Populárním příkladem takové knihovny je knihovna PICT [3] psaná v jazyce C++. Knihovna PICT klade důraz primárně na rychlost generování testů, snadnou použitelnost a rozšiřitelnost. [5]

Minimalizaci počtu vygenerovaných testovacích scénářů scénářů se za-

[3] Knihovna PICT je dostupná na https://github.com/microsoft/pict

bývá systém AETG, jehož implementaci poskytuje již zmíněná knihovna jesg/dither, psaná v jazyce Java [8]. AETG zaručuje logaritmický růst počtu vygenerovaných scénářů s počtem vstupních parametrů a jeví se jako vhodný kandidát pro generování našich testových variant. [6]

3.4.5 Shrnutí

Kombinatorické testovací metody nejlépe vyhovují našim požadavkům. Umožňují definovat počet otázek za určitý počet bodů a tím i celkový počet bodů za celý test. Navíc systémy jako je AETG redukují celkový počet vygenerovaných variant a s tím opakování otázek na rozdíl od běžných kombinací. Metoda k-partition je pak stále vhodná pro případ, že chceme zcela předejít opakování otázek v různých testových variantách.

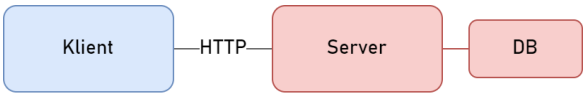
3.5 Architektura aplikace a volba technologií

Přecházíme k návrhu architektury aplikace. V následující sekci navrhneme nejprve high-level architekturu, poté vybereme vhodné technologie a nakonec vytvoříme finální architekturu včetně použitých technologií a jejich propojení.

3.5.1 High-level architektura

High-level architektura nastiňuje, na jaké části bude aplikace rozdělená a které části mezi sebou budou komunikovat.

Z high-level pohledu bude aplikace mít třívrstvou architekturu. Uživatel bude pracovat s klientskou vrstvou, která bude posílat HTTP požadavky na server. Server bude operovat nad databází, a poskytovat data klientské části. Bude zde sídlit také veškerá business logika aplikace. Takto docílíme návrhového principu "separation of concerns", který zjednodušuje jak návrh tak implementaci aplikace rozdělením na vrstvy vyvíjenými nezávisle, pouze se sdíleným rozhraním.



Obrázek 3.7: High-level architektura aplikace

3.6 Klientská aplikace

Klientská aplikace bude sloužit primárně jako grafické rozhraní pro uživatele. Pro tento účel je v dnešní době velice populární kombinace knihovny React.js s nějakou knihovnou poskytující hotové UI komponenty pro urychlení vývoje.

3.6.1 React.js

React je aktuálně obecně nejpopulárnějším řešením pro vývoj klientských aplikací [4]. Je vyvíjen společností Facebook od roku 2011. Mezi jeho hlavní výhody patří:

- jde o knihovnu vhodnou pro vývoj moderních single-page aplikací běžících v prohlížeči, aktualizace zobrazovaných dat probíhá bez překreslení celé stránky
- knihovna je založená na tvorbě komponent, které lze vyvíjet samostatně a snadno přepoužít
- je dostupná rozsáhlá dokumentace ať už oficiální nebo od komunity vývojářů

Pro implementaci lze u React.js využít buď JavaScript nebo TypeScript. Zde se přikloníme k TypeScriptu, díky jeho výhodám popsaným v sekci 3.6.2.

3.6.2 Typescript

TypeScript je silně typovaný programovací jazyk, který staví na JavaScriptu, rozšiřuje syntaxi a kompiluje se do JavaScriptu. [10] Díky kompilaci je TypeScript schopen odhalit chyby včas, na rozdíl od JavaScriptu, kde by se na chybu přišlo až později, při běhu aplikace. [9]

3.6.3 Ant Design

Ant Design [13] je jednou z knihoven nabízejících předpřipravené UI komponenty. Alternativní knihovna Material UI [5] poskytuje komponenty velice podobné. Rozhodujícím faktorem mezi těmito knihovnami je vzhled, který je u knihovny Ant Design více neutrální. Ant Design také nabízí pro nás zajímavější implementaci komponent jako je Tree než Material UI. Komponenty jsou vidět v náhledu a k dispozici je připravený kód ke zkopírování. Každá komponenta je dostupná v několika variantách. K dispozici je také popis API a příklady použití.

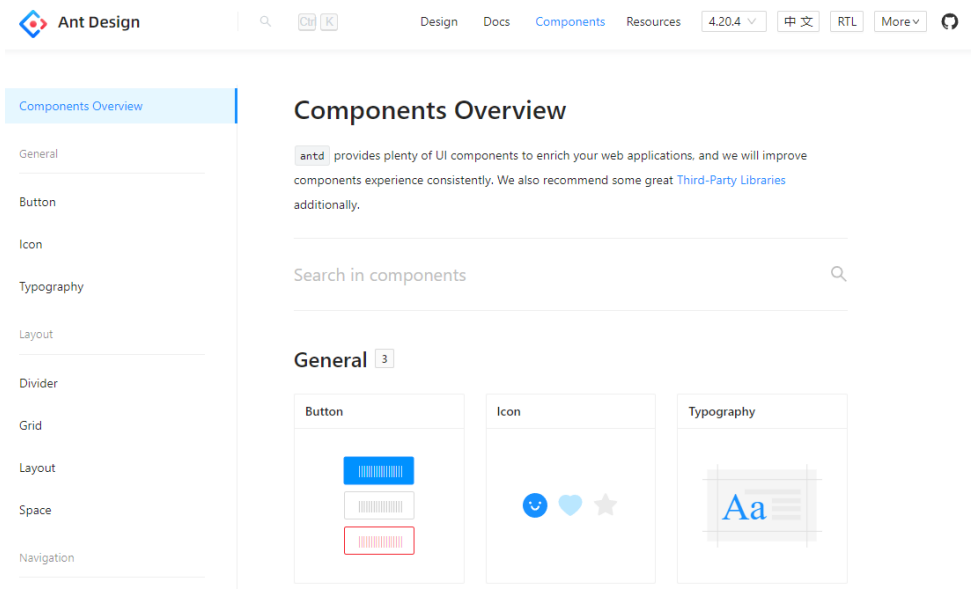
3.6.4 Node.js a NPM

Node.js je runtime prostředí pro běh kódu napsaném v jazyce JavaScript mimo prohlížeč. Je postavený na V8 JavaScript engine, open source projektu vyvíjeném společností Google a používaným mimo jiné i v prohlížečí Google Chrome. Node implicitně obsahuje NPM - node package manager, který využijeme pro správu závislostí v klientské části aplikace [6].

[4] Analýza popularity frameworků pro vývoj klientských aplikací: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

[5] Material UI je alternativou k AntD, lze najít na https://mui.com/

[6] Popis Node.js lze nalézt na https://nodejs.org/en/about/



Obrázek 3.8: Přehled komponent knihovny Atd Design

NPM je registr softwaru umožňující stahování a sdílení balíčků pro vývoj aplikací. Součástí NPM je i CLI [7], které umožňuje práci se samotným registrem.

NPM využívá soubor `package.json` viz obrázek 3.9 ke specifikaci verzí jednotlivých balíčků, které aplikace používá. Navíc umí automaticky dohledat a stáhnout závislosti a identifikovat problémy jako jsou nekompatibilní verze balíčků. Instalace balíčků probíhá pomocí příkazu `npm install <package-name>`, který v případě že baliček v registru existuje přidá závislost včetně její verze do souboru `package.json` a balíček nainstaluje.

3.7 Serverová aplikace

Serverová aplikace bude sloužit jako poskytovatel dat a bude provádět veškerou business logiku. Protože má naše aplikace využívat přihlašování uživatelů, bude vhodné zvolit knihovnu, která toto implicitně nabízí. Takovou knihovnou je Java Spring. Spring nabízí vše potřebné k rychlému vývoji serverových aplikací díky softwarovému paradigmatu "convention over configuration". Ten dovoluje přeskočit zdlouhavé nastavování v prospěch implementace samotné aplikační logiky. [14]

3.7.1 Maven

Jako správce závislostí a nástroj pro sestavení serverové aplikace využijeme Apache Maven 3. Maven funguje na bázi cílů (goals), které mají předdefinované

[7] CLI je zkratka pro "command line interface", neboli rozhraní v příkazové řádce

```
{
  "name": "etmt-fe",
  "version": "0.1.0",
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.2",
    "@testing-library/user-event": "^13.5.0",
    "@ant-design/icons": "^4.7.0",
    "antd": "^4.20.2",
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "react-router-dom": "^6.2.2",
    "react-scripts": "5.0.0",
    "ts-jest": "^28.0.2",
    "typescript": "^4.6.2",
    "web-vitals": "^2.1.4"
  },
}
```

Obrázek 3.9: Ukázka souboru package.json

kroky. Při spuštění jednoho z cílů maven na základě konfigurace provede například sestavení aplikace, otestování, vytvoření spustitelného artefaktu ale také deployment. Závislosti a konfigurace je definována pomocí XML v souboru `pom.xml`.

3.7.2 Spring Boot

Spring Boot je framework postavený nad Spring frameworkem, který umožňuje rychle připravit aplikaci vhodnou pro produkční prostředí bez nutnosti rozsáhlé konfigurace. Výchozí konfigurace lze přepsat a tím si aplikaci přizpůsobit podle potřeb. Aplikace bude dědit z předpřipraveného balíčku `spring-boot-starter-parent`, který lze dále přizpůsobit přidáním závislostí a konfigurací.

3.8 Rozhraní REST

Pro komunikaci mezi klientskou a serverovou částí aplikace využijeme REST rozhraní. REST je zkratka pro Representational State Transfer a následuje 5 principů [20]:

- uniform interface - obě strany využívají předem definované shodné rozhraní
- client-server model - "separation of concerns", oddělení klienta a serveru, klient zobrazuje data uživateli, server data poskytuje

```
@Document
public class Question {
    @Id
    private String id;
    private String title;
    private String text;
}
```

Obrázek 3.10: Mapování entity Question

- statelessness - server neudržuje stav, veškeré požadavky z klienta musí obsahovat všechny informace, které server potřebuje k obsoužení
- caching - data se ukládají v cache kvůli minimalizaci opakovaných dotazů
- layered architecture - jednotlivé komponenty mohou komunikovat pouze se sousedními vrstvami

Rozhraní definuje kontrakt na kterém spolu klient a server komunikují. My využijeme komunikaci přes HTTP, konkrétně dotazy GET, POST, PUT, DELETE.

3.9 Databáze

Výběr databáze se zakládá na několika hlavních faktorech. Pro nás nejdůležitějším faktorem je flexibilita. Podle doménového modelu uvedeném na obrázku 3.3 výše lze vidět, že primárním obsahem naší databáze jsou kategorie, které jsou naplněny mimo jiné otázkami s definovanými odpověďmi. Odpovědi by bylo vhodné ukládat dohromady spolu s otázkami. Dokumentově-orientované NoSQL databáze dovolují ukládat data v podobě dokumentů s JSON schématem, který nám s tímto usnadní práci.

3.9.1 MongoDB

MongoDB je nejvíc populárním exemplářem NoSQL databáze. Data jsou ukládána ve formátu JSON, což usnadňuje samotnou práci s databází. Během vývoje není třeba měnit databázové schéma ani provádět migrace dat [15]. Námi vybraný backend framework Spring navíc již implementuje rozhraní, pomocí kterého jde mapovat Java třídy na entity v databázi pomocí anotace `@Document`, jak lze vidět na obrázku 3.10 a 3.11.

K dispozici je také rozhraní `MongoRepository`, které umožňuje definovat metody provádějící databázové dotazy, viz obrázek 3.12.

```
{
    _id: ObjectId('61cdcc78926fc23fa07a7cc7'),
    title: 'My question',
    text: 'This is some question text ...',
    _class: 'cz.cvut.fel.pro.emtt.model.Question'
}
```

Obrázek 3.11: Podoba entity uložené v MongoDB

```
public interface QuestionRepository
    extends MongoRepository<Question, String> {
    @Query("{title:'?0'}")
    List<Question> findAllQuestionsByTitle(String title);
}
```

Obrázek 3.12: Databázový dotaz definovaný anotací v Javě

3.10 Export do PDF

K exportu zkouškového testu ve formátu PDF lze použít více alternativ. Například knihovna OpenPDF psaná v Javě poskytuje možnost snadno sestavit PDF dokument a uložit ho do souboru [16]. Její primární nevýhodou je poměrně omezená nastavitelnost vzhledu a rozložení obsahu na stránkách dokumentu.

LaTeX je z tohoto pohledu zajímavou alternativou. Obsah sám formátuje na základě předdefinované šablony. Dokumenty jsou uloženy ve formátu TEX, ze kterého se pak dají konvertovat do PDF, například pomocí balíčku PDFLatex. Tvorba dokumentu ve formátu TEX je poměrně jednoduchá díky syntaxi a struktuře tohoto formátu.

3.10.1 TeX a LaTeX

TeX. Tex je systém pro definici formátování dokumentů. Dovoluje specifikovat formátování do obrovského detailu. Mimo jiné využívá algoritmy pro počítání optimálního rozložení dokumentu.

LaTeX. Latex je systém pro psaní obsahu dokumentů, který využívá TeX k formátování. Je často používaný při psaní delších vědeckých dokumentů ale lze ho použít takřka k tvorbě jakéhokoliv dokumentu. Cílem LaTeXu je přenést pozornost autora dokumentu z formátování na samotný text, zatímco o formátování je postaráno díky TeXu [17]. LaTeX poskytuje sadu maker, respektive příkazů, pomocí kterých lze snadno definovat obsah a základní strukturu dokumentu, jak lze vidět na obrázku 3.14.

```
\documentclass[a4paper]{article}

\title{Example}
\author{Author}

\begin{document}
\maketitle

\end{document}
```

Obrázek 3.14: Ukázka LaTeXového kódu a vygenerovaného PDF

3.10.2 PDFLatex

PDFLatex je rozšíření LaTeXu, které dovoluje vygenerovat PDF přímo ze zdrojového TEX souboru. Abychom mohli používat v aplikaci PDFLatex, je potřeba mít nainstalovaný některý z mnoha balíčků, ve kterém se vyskytuje. Asi nejznámějším je TEX Live, který má několik variant. My zvolíme variantu `texlive-latex-recommended`, která představuje kompromis mezi celkovou velikostí a obsahem a jejíž součástí je implicitně i PDFLatex.

3.10.3 Shrnutí

K exportu do formátu PDF využijeme LaTeX, protože umožňuje rozsáhlejší a snazší formátování v porovnání s alternativami jako je OpenPDF. Také se zde otevírá možnost krom XML a PDF exportovat i samotný TEX soubor, který si může zadavatel testu upravit podle vlastního uvážení v TEX editorech, jako je například Overleaf.

3.11 Nasazení

Nasazení aplikace může být náročné zejména proto, že každá část aplikace vyžaduje specifické knihovny a prostředí, ve kterém může běžet. V dnešní době se proto využívají různé možnosti virtualizace, kde se jednotlivé části aplikace provozují v samostatných kontejnerech, ve kterých běží prostředí přizpůsobené potřebám té dané části aplikace. Jednotlivé kontejnery mezi sebou pak komunikují například pomocí HTTP dotazů.

3.11.1 Docker

Docker je příkladem takovéto virtualizace. Umožňuje definovat takzvané images, které slouží jako šablony pro tvorbu kontejnerů.

Image. Jednotlivé image lze buď najít a stáhnout z oficiální stránky Dockeru [22], nebo lze vytvořit vlastní image pomocí `Dockerfile` souboru, který specifikuje operační systém a proceduru nastavení výchozího stavu, ze kterého se poté vytvářejí kontejnery.

Kontejner. Instalací vzorového image je pak kontejner. Uvnitř běží specifikovaný systém a další přidané služby - například naše aplikace. Docker navíc umožňuje vytvářet několik images a posléze kontejnerů naráz podle specifikace v `docker-compose.yml` souboru, který definuje jednotlivé Docker služby a parametry pro jejich spuštění.

Volume. Docker volume slouží ke sdílení dat mezi kontejnery a vnějším systémem, na kterém běží Docker. Samotné kontejnery mají vlastní, oddělenou paměť. Volumes jsou vhodné například pro vývojovou fázi, kdy můžeme kód sdílet mezi vývojovým adresářem a adresářem uvnitř kontejneru, ve kterém je kód interpretován - například u vývoje klientské části aplikace.

Pro naše potřeby je Docker ideální a pomůže nám separovat prostředí pro běh jednotlivých částí aplikace. Těmi jsou:

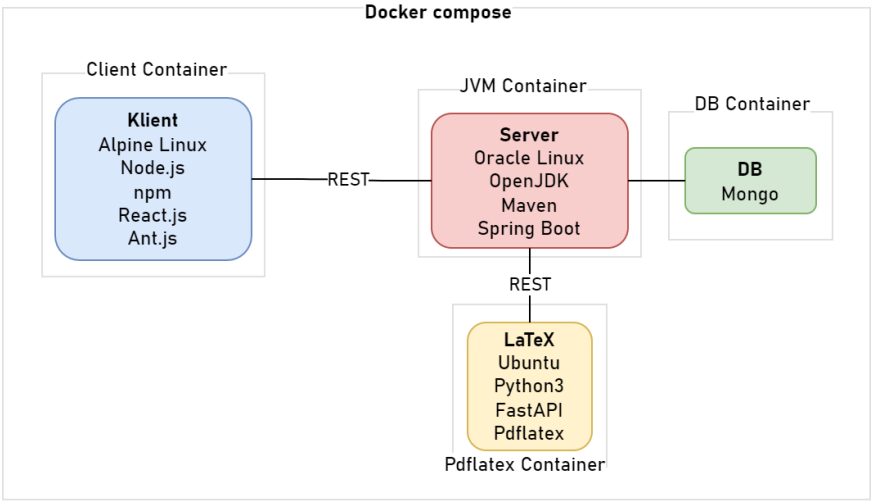
- Klientská aplikace - npm - React.js, Ant.js
- Serverová aplikace - Maven - Java
- Databáze - MongoDB
- PDFLatex

3.11.2 PDFLatex jako RESTová služba

PDFLatex je součástí balíčku TEX Live, který má více podob. Pro naše potřeby jsme vybrali variantu `texlive-latex-recommended`. Je vhodné stejně jako další části aplikace nasadit tuto službu do Docker kontejneru. K tomu budeme potřebovat implementovat jednoduché REST API, které bude umožňovat nahrát TEX soubor a vrátí vygenerovaný PDF soubor. Jedním z nejjednodušších možností pro implementaci REST api je FastAPI pro programovací jazyk Python. [18]

3.12 Finální architektura

S využitím vyjmenovaných technologií můžeme sestavit finální architekturu. Aplikace bude rozdělena na 4 částí, které poběží ve vlastních Docker kontejnerech.



Obrázek 3.15: Finální architektura aplikace včetně použitých technologií

Kapitola 4

Implementace aplikace

S hotovým návrhem architektury a vybranými technologiemi postupně implementujeme samotnou aplikaci. V následující kapitole se budeme věnovat principům a postupu při implementaci jednotlivých částí aplikace.

4.1 Vývojové prostředí

Aplikaci vyvíjíme v lokálním vývojovém prostředí.

Klient. Klientskou aplikaci poprvé vytvoříme příkazem `npx create-react-app etmt-fe`. Případné další závislosti instalujeme v adresáři `etmt-fe` příkazem `npm install`. Aplikaci poté spustíme příkazem `npm start`. S výhodou lze využít "live reload", kdy se aplikace po uložení změn znovu zkompiluje a aktualizuje se náhled v prohlížeči.

Server. Serverovou aplikaci spouštíme při vývoji z integrovaného vývojového prostředí IntelliJ IDEA. V konfiguraci spuštění aplikace definujeme systémové proměnné, které aplikace využije pro připojení k databázi:
`DB_PSW=rootpass;DB_HOST=localhost;DB_USER=rootuser;`

Databáze. Databázi pro testování provozujeme v Dockeru. Databázi pak lze po testování snadno smazat a znovu vytvořit. Využijeme jeden z existujících Docker images pod názvem "mongo"[1] a "mongo-express". Mongo Express je služba umožňující správu MongoDB databáze přes přehledné grafické rozhraní. Samotnou konfiguraci Dockeru probereme do detailu v sekci Nasazení aplikace.

PdfLaTex. Službu s PdfLaTeXem pro úvodní vývoj spustíme pomocí Python služby `uvicorn` s možností `--reload`, která funguje obdobně k funkci "live reload"u `npm`.

[1] Docker image mongo je k dispozici na https://hub.docker.com/_/mongo

4.2 Klientská aplikace

4.2.1 Základní principy a struktura

Klienskou aplikaci implementujeme v jazyku Typescript s použitím frameworku React.

React. React umožňuje stavět aplikaci z tzv. komponent, které tvoří stromovou strukturu. Každá komponenta si udržuje vlastní stav a předává ho jeho potomkům přes tzv. `props` (atributy). Jednotlivé komponenty jsou definovány funkcí a vrací tzv. JSX elementy. JSX elementy jsou kombinací HTML[2] a Javascriptu či Typescriptu, které React používá k sestavování HTML DOMu [3]. React hojně využívá observer pattern.

```
// definice atributů
interface ExampleProps {
  example: JSX.Element
}

// definice komponenty
const ExampleComponent: FunctionComponent<ExampleProps>
  = (props: ExampleProps) => {
    // komponenta vrací hodnotu atributu example
    return props.example;
  }

// rodičovská komponenta
const ParentComponent: FunctionComponent<{}> = () => {
  return (
    // komponenta vrací ukázkovou komponentu a specifikuje
    // hodnotu atributu example, jímž je prostý HTML div
    // element s textem uvnitř
    <ExampleComponent example={<div>EXAMPLE TEXT</div>}/>
  );
}
```

Obrázek 4.1: Ukázka struktury React komponent a JSX fragmentů

Observer pattern. Observer pattern je návrhový vzor kde observer je objekt, který se přihlásí jinému objektu, tzv. observable, že chce notifikovat při změně jeho stavu. V moment, kdy dojde u observable ke změně, notifikuje všechny observery, kteří se mu přihlásili. Tímto způsobem jsou minimalizovány

[2] HTML je zkratka pro Hyper Text Markup Langage, je to jazyk sloužící pro definici struktury webové stránky a jejího obsahu

[3] Popis JSX lze najít na https://reactjs.org/docs/introducing-jsx.html

závislosti mezi komponenty. React takto řeší mimo jiné aktualizaci dat, jako reakci na změnu stavu komponent. Poskytuje rozhraní, tzv. React Hooks, s jejichž použitím lze se stavy snadno pracovat. Nejtypičtějšími příklady jsou `useState` a `useEffect`. Hook `useState` udržuje stav zatímco `useEffect` sleduje stavy specifikované v hranatých závorkách a zavolá definovanou funkci jako reakci na změnu některého z těchto stavů [4].

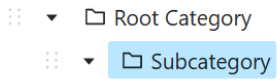
```
const Library: FunctionComponent<{}> = () => {
  const auth = useAuth();
  const [library, setLibrary] = useState();

  useEffect(() => {
    console.log("user changed: ", auth.user)
    fetchLibrary();
  }, [auth.user])

  const fetchLibrary = () => {
    var library;
    // library fetching logic
    setLibrary(library);
  }

  return (
    <div>{library}</div>
  );
}
```

Obrázek 4.2: Ukázka použití React hooks useState a useEffect pro načtení knihovny jako reakci na změnu uživatele

4.2.2 Komponenty - AntD

Ant Design nám při implementaci poskytuje nespočet hotových React komponent s definovaným API. Při implementaci samotného uživatelského rozhraní aplikace chceme tyto komponenty naplno využít, abychom nemuseli implementovat komponenty vlastní. Návrh samotného uživatelského rozhraní se tedy bude částečně odvíjet od komponent dostupných v AntD knihovně [13]. Na obrázku 4.4 lze vidět strukturu komponenty Tree, umožňující práci se stromovou hierarchií položek. Vidět je také struktura dat a část API, kterou komponenta poskytuje. Na obrázku 4.3 je pak vidět vzhled této komponenty.

[4] Oficiální dokumentaci React hooks lze najít na https://reactjs.org/docs/hooks-intro.html



Obrázek 4.3: Komponenta Tree knihovny AntD s kořenovou kategorií a vybranou podkategorií

```
const library = [{
    title: 'Root Category',
    key: '0-0',
    children: [{
        title: 'Subcategory',
        key: '0-0-0',
        children: [],
    }]
}]

const onSelect = (keys: any, info: any) => {
  console.log("selected item changed: ", keys, info);
};

export default () => (<Tree
  // komponenta poskytuje api
  onSelect={onSelect} // callback metoda při volbě položky
  treeData={library} // data zobrazená ve stromu
  draggable={true} // položky lze přemisťovat
  // ...
/>);
```

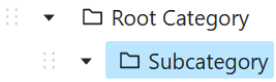
Obrázek 4.4: Ukázka použití komponenty Tree z knihovny AntD

4.2.3 Komponenty - Ostatní

React Router. React Router je knihovna Reactových komponent, která zprostředkovává navigaci pomocí URL cest. Lze snadno definovat, na jakém URL bude dostupná která část aplikace [12]. Na obrázku 4.6 lze vidět navigaci v naší aplikaci. Pod cestou `/` je dostupná samotná knihovna. K přihlašovací obrazovce a registraci lze přistoupit přes `/login` respektive `/signup`. Tyto cesty následně použijeme v již zmíněném hlavním menu, kde pomocí komponenty `Link` specifikujeme cestu pro jednotlivé položky.

Require Auth. Jde o komponentu umožňující přístup k jiným komponentám pouze pro autentizované uživatele. Ukázková implementace je dostupná v dokumentaci React Routeru [5]. Komponenta využívá vlastní triviální poskyto-

[5] Ukázková implementace autentizačního mechanizmu je k dispozici na https://reactrouter.com/docs/en/v6/examples/auth



Obrázek 4.5: Komponenta Tree knihovny AntD s kořenovou kategorií a vybranou podkategorií

```
<Routes>
  <Route path="/" element={
    <RequireAuth>
      <Library />
    </RequireAuth>}
  />
  <Route path="/login" element={<Login />} />
  <Route path="/signup" element={<Signup />} />
</Routes>
```

Obrázek 4.6: Navigace pomocí React Router knihovny, cesta k Library komponentě je zabezpečená

vatel autentizace, který v naší aplikaci nahradíme vlastní implementací. Ta bude komunikovat se serverem při přihlášení a při úspěšném přihlášení si uloží JSON Web Token (viz odstavec 4.4.2) poskytnutý serverem. Ten pak můžeme kdekoliv v aplikaci použít k odeslání dotazů na zabezpečené endpointy. Samotný stav autentizace včetně tokenu nám uchovává hook `useAuth`. V případě, že se nepřihlášený uživatel pokusí přistoupit k zabezpečené komponentě, je automaticky přesměrován na přihlašovací obrazovku.

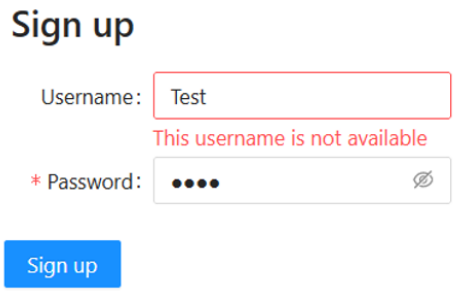
4.3 Popis uživatelského rozhraní

V této sekci se zaměříme na samotné uživatelské rozhraní, projdeme si jeho hlavní části a vysvětlíme jakým způsobem funguje.

Navigační menu. Toto menu poskytuje jednoduchou navigaci mezi aplikací, registrací, přihlášením a odhlášením. Menu dynamicky vypíná a zapíná jednotlivá tlačítka podle toho, jestli je uživatel přihlášený nebo ne. Navigační menu je vlastní komponenta, která získává stav aktuálního uživatele z autentifikačního hooku `useAuth`. V případě že je uživatel odhlášen jsou příslušná tlačítka deaktivována. Menu lze vidět na obrázku 4.7.

Obrázek 4.7: Navigační menu aplikace

Registrace a přihlášení. Formuláře registrace a přihlášení jsou si velmi podobné. Na ukázce 4.8 lze mimo jiné vidět chybovou hlášku, která se objeví když aktuálně zadané uživatelské jméno již existuje. K ověření, jestli uživatelské jméno existuje je použit systém ověření správnosti vstupu formulářové komponenty AntD, který při změně vstupu odešle požadavek na server s aktuálně zadaným uživatelským jménem.



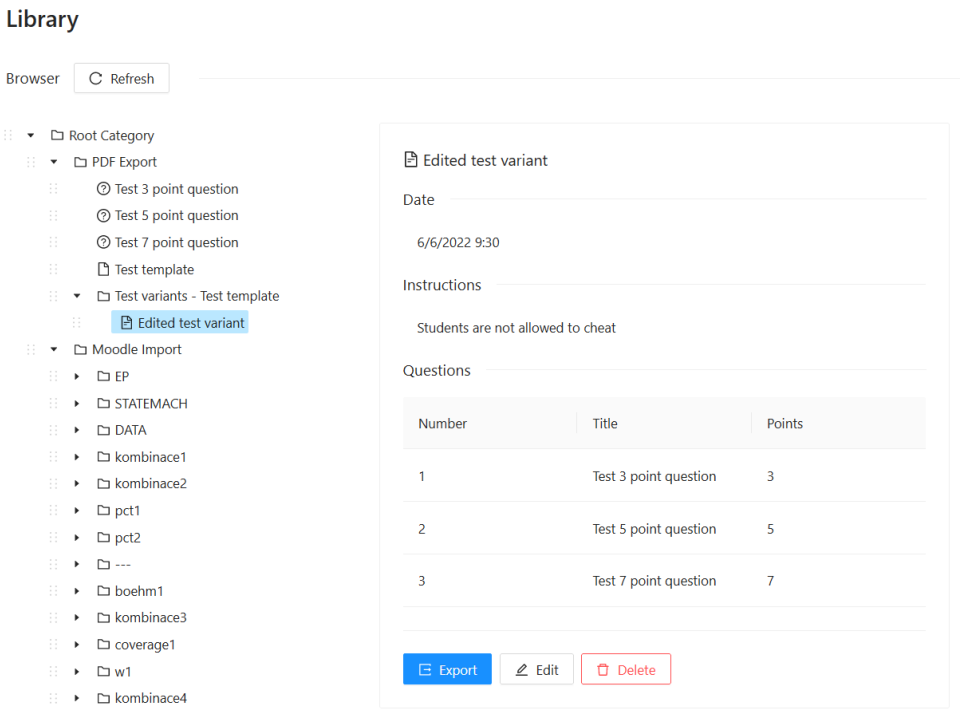
Obrázek 4.8: Ukázka registračního formuláře s chybovou hláškou

Knihovna. Po úspěšném přihlášení je uživateli zobrazena jeho knihovna. Nový uživatel bude mít ve stromu položek pouze kořenovou kategorii "Root Category". Na jednotlivé položky stromu vlevo lze kliknout. Kliknutí na položku zobrazí její náhled v sekci napravo od stromu. Podle typu položky náhled zobrazí odlišné informace. Náhled každého typu položky má separátní React komponentu, která při zvolení položky načte potřebná data ze serveru. Načtení dat je implementováno pomocí React hooků `useState` a `useEffect`. V moment kdy se změní stav zvolené položky `selectedItem` se provede metoda v `useEffect` hooku, která načte data pro nově zvolenou položku. Sekce také využívá komponentu `Skeleton` z knihovny AntD, která umožňuje zobrazit náhradu obsahu v moment, kdy se čeká na odpověď ze serveru. Ukázka knihovny s načtenou testovou variantou je zachycena na obrázku 4.9.

Přidání a editace otázek. Veškerá manipulace s položkami v hierarchií probíhá pomocí modálních oken. Ty umožňují zobrazit obsah, který by se normálně již nevešel na stránku a elegantním způsobem ho oddělí od zbytku obsahu. Další výhodou je fakt, že modální okno můžeme po zavření smazat a tím se automaticky resetují všechna inicializovaná data z předchozích operací.

Příklad použití modálního okna si uvedeme na přidávání a editaci otázek zejména proto, že je nejkomplexnější. Modální okno pro otázku umožňuje nastavit všechny parametry otázky. Je tu použito ověření vstupu pro povinné položky a minimální hodnotu bodového ohodnocení otázky. Ukázka modálního okna přidání otázky viz obrázek 4.10 s detailem odpovědí na obrázku 4.11.

Podle zvoleného typu otázky ve výběru nahoře se změní struktura odpovědí. Otázky s krátkou odpovědí (short answer) nemají možnost zaškrtnout, jestli je odpověď správná, protože to zde nedává smysl. U otázky s jednou správnou odpovědí resp. více správnými odpověďmi bude navíc možnost zaškrtnout jednu



Obrázek 4.9: Uživatelská knihovna s náhledem testové varianty

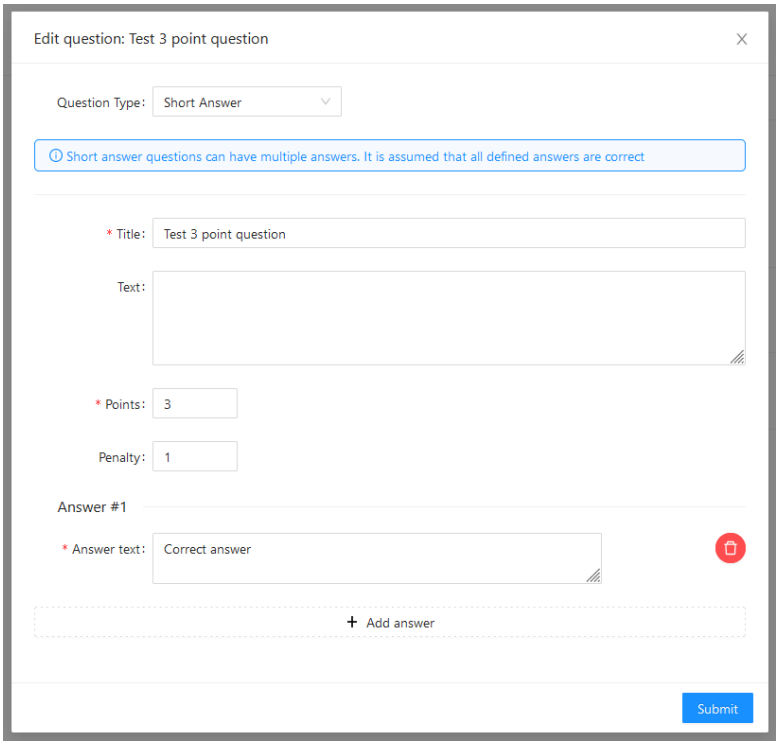
resp. více správných odpovědí. Tato implementace opět využívá `useState` pro udržování stavu formuláře podle jeho hodnoty se upraví zobrazované elementy.

Stejné okno i formulář je použito krom přidání i u editace. Hlavní rozdíl je v tom, že při otevření okna pro editaci se formulář naplní aktuálními daty.

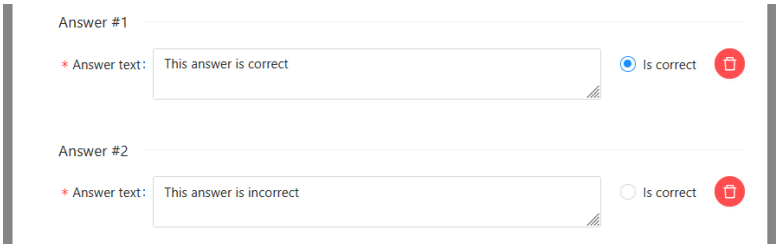
Tvorba a editace testových variant. Práce s testovými variantami je dalším příkladem použití modálního okna. Zde je zajímavostí výběr kategorií u jednotlivých témat (topic). K dispozici je pole, které umožní vybrat jednotlivé kategorie z podstromu s kořenem v rodičovské kategorie vybrané šablony. Na obrázku 4.12 lze vidět výběr kategorií při tvorbě šablony.

Generování testových variant. Generování je možné realizovat volbou příslušné šablony ze stromu položek v knihovně a následnou volbou tlačítka "Generate". Aplikace automaticky ověří, jestli lze podle definované šablony v tuto chvíli test vygenerovat. Klientská aplikace pošle dotaz na server, který toto ověří a odešle zpět informaci o tom, které otázky chybí ve které kategorii. V případě že test vygenerovat nelze, zobrazí se v modálním okně chybové hlášky, jak lze vidět na obrázku 4.13. Pokud test z nějakého důvodu vygenerovat nelze, tlačítko "Generate"zůstává neaktivní.

Export. Export do obou formátů je realizovaný stejným modálním oknem. K dispozici jsou dvě tlačítka "Export PDF"a "Export XML". Po kliknutí se odešle



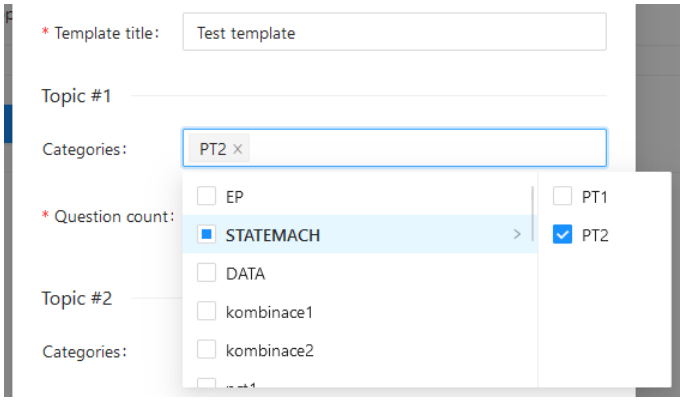
Obrázek 4.10: Modální okno pro přidání otázky s krátkou odpovědí



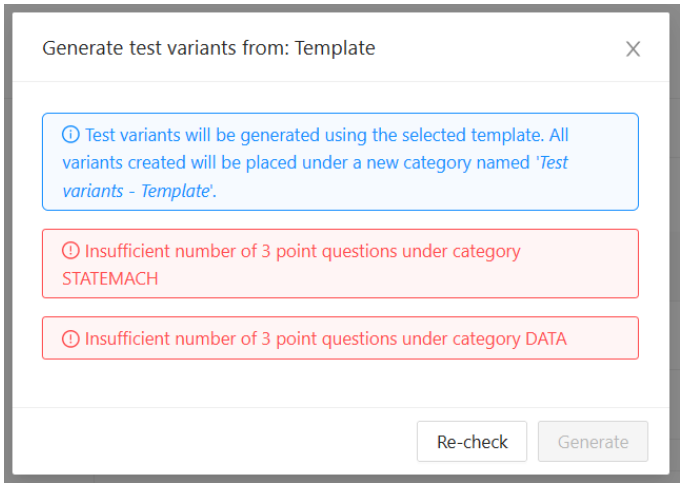
Obrázek 4.11: Seznam odpovědí se zaškrtávacími tlačítky pro otázku s jednou správnou odpovědí

požadavek na server a ten v případě úspěchu vrátí klientovi příslušný soubor. Soubor je nejprve převeden z binární formy do příslušného formátu. Poté je vytvořen skrytý HTML element `<a>` s dočasnou cestou k přijatému souboru. Pak je zavolána metoda `click()` tohoto elementu, která automaticky přijatý soubor stáhne. Tato funkčnost se mírně liší pro různé prohlížeče. Například v prohlížeči Firefox v100.0 se otevře systémové okno pro uložení souboru. Tento rozdíl je pravděpodobně daný nastavením prohlížeče a nikoliv samotnou implementací aplikace.

Import. Pro import XML souboru z Moodle využijeme element `Upload` z knihovny AntD. Ten umožňuje nahrát soubor přetažením či výběrem pomocí systémového okna. Po zvolení souboru se zobrazí soubor v dolní části okna s indikací průběhu nahrání. Element pro nahrání je opět umístěn v modálním



Obrázek 4.12: Výběr kategorie v modálním okně přidání testové šablony



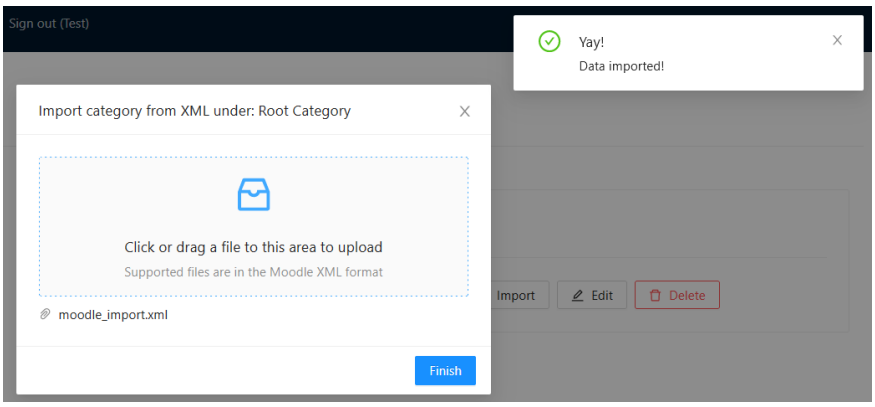
Obrázek 4.13: Modální okno generování testových variant s chybovými hláškami

okně. Po nahrání lze modální okno zavřít a v knihovně se objeví importované kategorie. Ukázka okna pro import je vidět na obrázku 4.14

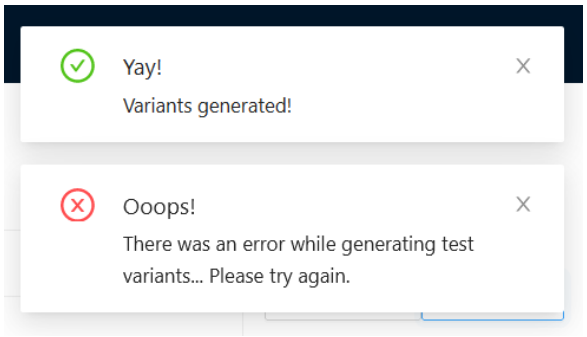
Notifikace. Aplikace sděluje uživateli úspěch či neúspěch operací a požadavků v podobě notifikací. Notifikace se objevují dynamicky v pravém horním rohu a jejich text indikuje, k jaké události došlo. Jsou použity dva typy této notifikace - úspěch a chyba. Obě notifikace lze vidět na obrázku 4.15.

4.4 Serverová aplikace

Serverovou aplikaci implementujeme v jazyce Java s využitím Mavenu jako správce balíčků. Spring Boot nám zde pomáhá oddělit data, business logiku a endpointy.



Obrázek 4.14: Úspěšný import kategorií z Moodle



Obrázek 4.15: Ukázka notifikací

4.4.1 Vrstvená architektura

Datová vrstva. Datová vrstva je tvořena entitami, které pomocí rozhraní `MongoRepository` ukládáme do databáze. `MongoRepository` je součástí výchozího balíčku Spring Data a poskytuje všechny základní operace typu `save`, `findById` a `delete`. Pro naše potřeby není nutné vytvářet vlastní implementaci pomocí tohoto rozhraní. Jednotlivé entity jsou jednoznačně identifikovány ID datového typu `String`, které je automaticky generováno při prvním uložení do databáze.

Servisní vrstva. Veškerá business logika aplikace je umístěna na servisní vrstvě, která komunikuje s vrstvou datovou. Servisní vrstva zajišťuje zejména:

- autentizaci
- stromovou strukturu knihovny otázek
- generování testových variant
- export do XML a PDF

Podstatná část servisní vrstvy pouze obaluje datovou vrstvu a žádnou další business logiku neprovádí.

Controller vrstva. Tato vrstva vystavuje RESTové rozhraní serverové aplikace. Na této vrstvě jsou definovány jednotlivé endpointy. Tato vrstva využívá vrstvu servisní k plnění příchozích požadavků. Také zde dochází k mapování DTO[6] entity. Důvodem pro toto mapování je oddělení objektů přenášejících data mezi klientem a serverem a samotnou datovou vrstvou aplikace. K mapování využijeme ModelMapper knihovnu, která umožňuje implicitně mapovat objekty se shodnými atributy a definovat vyjímky, viz obrázek 4.16.

```
Category category =
    modelMapper.map(categoryPayload, Category.class);
```

Obrázek 4.16: Mapování DTO na entitu pomocí ModelMapperu

4.4.2 Autentizace

Naše aplikace umožňuje registraci a přihlášení uživatelů. K tomu potřebujeme autentizační mechanismus. JSON Web Token (JWT) je velmi rozšířené řešení autentizace v RESTových aplikacích. Protože je naše serverová aplikace bezestavová (stateless, viz kapitola 3.8), server nijak neudržuje informace o přihlášených uživatelích. Při implementaci autentizačního mechanizmu využijeme vzorovou implementaci [7].

JSON Web Token. JSON Web Token je otevřený standard (RFC 7519), který definuje a zapouzdřené řešení jak bezpečně přenášet informace mezi protistranami v podobě JSON objektu. [19] Informace uložené v tomto tokenu jsou digitálně podepsané a jsou důvěryhodné. Při přihlášení server použije přihlašovací údaje uživatele k vygenerování JWT. Token je uživateli předán a je přiložen k uživatelovým požadavkům posílaným na server. Server podle konfigurace zamítá ty dotazy, které token přiložený nemají nebo pokud je přiložený token nevalidní.

4.4.3 Implementace knihovny

Každý uživatel naší aplikace má vlastní knihovnu. Knihovna sestává ze stromové hierarchie položek. Položky mají několik typů, konkrétně kategorie, otázka, šablona a testová varianta. Kategorie mohou mít další položky jako své potomky, včetně dalších podkategorií. Položky typu otázka, šablona a testová varianta tvoří listy této stromové hierarchie.

Každá položka má klíč, který indikuje její pozici v hierarchii. Tento klíč vychází ze specifikace React komponenty Tree, viditelnou na obrázku 4.4. Každý uživatel dostane po registraci kořenovou kategorii "Root category",

[6] DTO je zkratka pro Data Transfer Object, tyto objekty definují strukturu dat přenášených v těle požadavku

[7] Vzorová implementace autentizace s React a Spring boot je dostupná na https://www.bezkoder.com/spring-boot-react-jwt-auth/

jejíž klíč je `0-0`. Potomci této kategorie budou mít klíče `0-0-i` kde `i` je index potomka.

Stromová struktura je průběžně udržována. Při operaci přidání položky, odebrání položky a přesunu položky v hierarchii se aktualizují klíče ostatních položek.

4.4.4 Generování testových variant

Ke generování testových variant využijeme AETG, které jsme probrali v analýze v sekci Analogie s pairwise testováním.

Knihovna jesg/dither. Tato knihovna poskytuje API, pomocí kterého jde pro seznam tříd ekvivalence vygenerovat příslušné kombinace. Ke generování využijeme metodu `Dither.aetg()`. Podle autora jde o nedeterministickou pairwise strategii, díky čemuž pro opětovné generování můžeme dosáhnout různých výsledků, resp. různých testových zadání. Tato metoda vyžaduje minimálně 2 třídy ekvivalence, každou se dvěma vstupy. V našem případě třída ekvivalence odpovídá otázkám definovaným jedním tématem testové šablony, např. 2 otázky za 3 body z kořenové kategorie. V naší aplikaci však chceme generovat testy i z méně jak 4 otázek, a proto implementujeme vlastní řešení pro případy, kdy máme třídu ekvivalence pouze jednu, či pro případ že máme třídy ekvivalence dvě ale jejich obsah není dostačující. Jednotlivé případy jsou znázorněny v tabulce 4.1 Tyto případy využijeme také později při testování.

Vstup	Výstup
K témat, pro aspoň jedno téma ex. < 1 otázka	nelze vygenerovat
K témat, pro každé téma ex. právě 1 otázka	1 testová varianta, generujeme manuálně
K témat, pro právě jedno téma ex. N otázek pro ostatní témata ex. právě 1 otázka	N testových variant, generujeme manuálně
$K > 1$ témat, pro každé téma existuje $N > 1$ otázek	P testových variant, gen. pomocí AETG

Tabulka 4.1: Výčet jednotlivých případů generování testových variant

4.4.5 Import a Export

Aplikace umožňuje import z formátu XML a export do formátu XML a PDF. Importují se kategorie vyexportované z aplikace Moodle.

Import a Export XML. Import a export spočívá v konverzi XML v binární podobě do Java entit, resp. z Java entit do XML. Java poskytuje vestavěné API pro práci s XML, konkrétně `javax.xml`. Při importu postupným parsováním

jednotlivých kategorií a otázek konstruujeme požadovanou hierarchii položek, kterou poté uložíme do databáze. Při exportu naopak vytváříme objektovou reprezentaci XML, které pak převedeme do binární podoby a odešleme zpět klientské aplikaci.

Export do PDF. Přímý export do PDF sice v Javě možný je, nicméně dostupné knihovny z důvodů probraných v analýze nepoužíváme 3.10. Místo konstrukce PDF v naší serverové aplikaci sestavíme LaTeXový zdrojový kód, který pošleme POST dotazem separátní LaTeXové službě `fa-pdflatex` k vytvoření PDF. K odeslání HTTP dotazů v Javě použijeme Rest Template, což je vestavěné rozhraní ve Springu umožňující konstrukci a odeslání HTTP dotazů [8]. Služba `fa-pdflatex` vystavuje API, přes které je možné nahrát zdrojový soubor v binární podobě a zpět vrátí PDF soubor, opět v binární podobě.

Služba fa-pdflatex. Tato služba má poměrně jednoduchou implementaci. Vystavuje jediný endpoint `/upload`, na kterém přijímá v těle požadavku samotný TEXový soubor v binární formě. Tento soubor si aplikace uloží a spustí příkaz `pdflatex` se specifikovanou cestou ke zdrojovým souborům a cestou pro výstup. Po provedení tohoto příkazu je vygenerovaný PDF soubor opět načten v binárním formátu a odeslán v těle odpovědi.

4.5 REST API

Aby spolu mohly jednotlivé části aplikace komunikovat, definujeme RESTové API. Definice RESTového API je nejčastěji realizována pomocí OpenAPI specifikace, která popisuje aplikaci, její endpointy a povahu dat, která se přenáší[21]. Ke specifikaci OpenAPI se nejčastěji využívá Swagger [9]. Pro serverovou aplikaci tuto specifikaci po implementaci endpointů automaticky vygenerujeme v nástroji Intellij IDEA. Ukázka vygenerované specifikace je na obrázku 4.17.

4.6 Nasazení aplikace

Pro nasazení aplikace použijeme Docker. Aplikace bude mít celkem 5 kontejnerů:

- fe - npm, React - klientská aplikace
- jvm - maven, Java - serverová aplikace
- mongo - MongoDB - databáze

[8] Dokumentaci REST template lze nalézt na https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

[9] Swagger je sada nástrojů pro implementaci specifikace OpenAPI https://swagger.io/

```
openapi: 3.0.0
info:
  title: "ETMT API"
  description: "API for the Exam Test Management Tool"
  version: "1.0.0"
paths:
  /auth/checkUsernameAvailability:
    get:
      summary: "GET auth/checkUsernameAvailability"
      operationId: "checkUsernameAvailability"
      parameters:
        - name: "username"
          in: "query"
          required: true
          schema:
            type: "string"
      responses:
        "200":
          description: "OK"
```

Obrázek 4.17: Ukázka části OpenAPI specifikace vygenerované v IntelliJ IDEA

- mongo-express - Mongo Express pro správu databáze
- fa-pdflatex - python, fastAPI, PdfLatex - služba pro generování PDF

Jednotlivé kontejnery lze vidět na diagramu nasazení 4.18

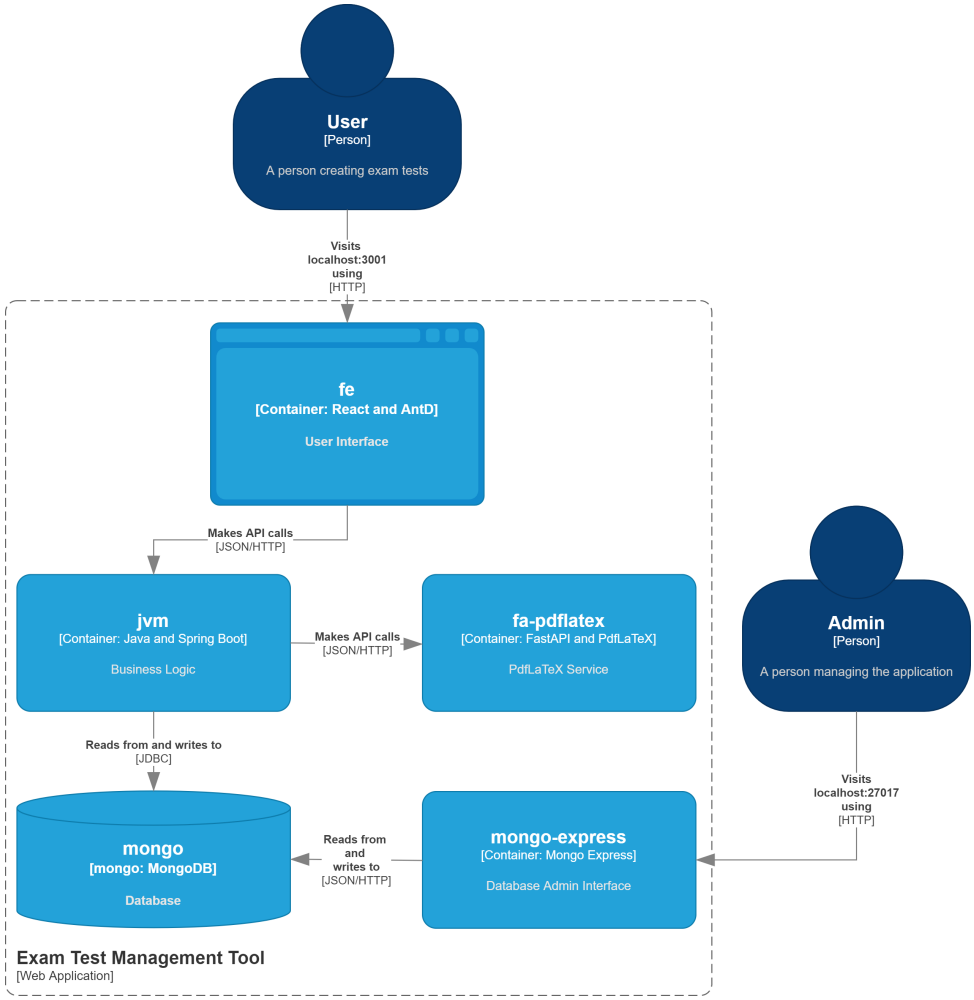
Pro konfiguraci jednotlivých kontejnerů použijeme Dockerfile. Dockerfile specifikuje proceduru, kterou Docker provede při vytváření Docker image. Po vytvoření Docker image se podle tohoto image vytvoří samotný kontejner.

Na obrázku 4.19 je Dockerfile naší klientské aplikace. Jako základ je použit image `node:17-alpine` dostupný na Dockerhubu [10]. Jako další krok jsou nastaveny systémové proměnné `PATH` a `NODE_OPTIONS` které naše aplikace vyžaduje. Poté jsou zkopírovány soubory `package.json` a `package-lock.json`, které jsou následně použity pro specifikaci verzí jednotlivých balíčků při instalaci v dalším kroku. Nakonec jsou přesunuty zdrojové soubory a aplikace je spuštěna.

Dockerfile ostatních částí aplikace je obdobný. Některé části aplikace jako je `mongo` Dockerfile nepotřebují. Na Dockerhubu je dostupný oficiální image `mongo`, který můžeme použít místo vytváření vlastního image.

Docker compose. Docker nám umožňuje spravovat všechny kontejnery naráz pomocí Docker compose. Zároveň poskytuje jednoduchý způsob, jak kontejnery konfigurovat, aby fungovaly dohromady. Soubor `docker-compose.yml` definuje konfiguraci všech našich služeb. Jsou to popsány jednotlivé služby,

[10] Použitý Node.js image na dockerhubu: https://hub.docker.com/_/node



Obrázek 4.18: Finální deployment diagram aplikace

jejich název, mapování portů, systémové proměnné a další. Ukázku struktury Docker compose souboru lze vidět na obrázku 4.20.

Za zmínku stojí mapování portů. Každý kontejner ve výchozím nastavení nevystavuje žádné vnější rozhraní. Aby mohly kontejnery mezi sebou komunikovat, je nutné namapovat porty používané aplikací uvnitř kontejneru na vnější port kontejneru. Příkladem je služba `fe`, u které mapujeme port `3000` na port `3001`. To znamená, že vnitřní aplikace běží na portu `3000` a náš kontejner s názvem `fe` vystavuje port `3001`. Tímto mapováním docílíme toho, že v našem systému kde běží kontejnery bude klientská aplikace dostupná, a to na portu `3001`.

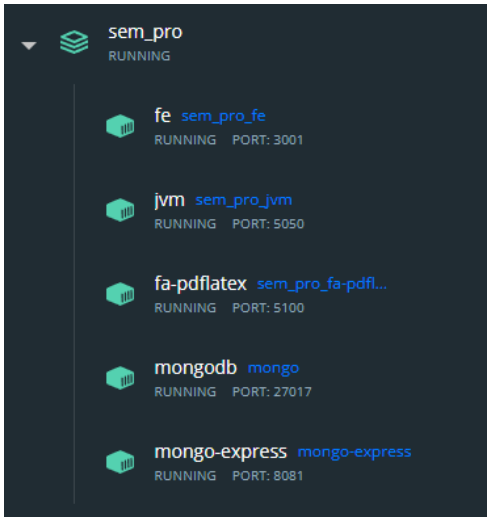
Celou aplikaci spustíme příkazem
`docker compose -f "docker-compose.yml" up -d --build`,
který vytvoří všechny kontejnery podle souboru `docker-compose.yml`. Po spuštění bude GUI Dockeru vypadat jako na obrázku 4.21. Uživatelské rozhraní najdeme na adrese `http://localhost:3001/`.

```
FROM node:17-alpine
WORKDIR /app
ENV PATH /app/node/node_modules/.bin:PATH
ENV NODE_OPTIONS --openssl-legacy-provider
COPY package.json ./
COPY package-lock.json ./
RUN npm install
COPY . ./
CMD ["npm", "start"]
```

Obrázek 4.19: Ukázka Dockerfile pro klientskou aplikaci

```
version: "3.8"
services:
  fe:
    container_name: fe
    build: etmt-fe
    ports:
      - "3001:3000"
    volumes:
      - "./etmt-fe:/app"
  jvm:
    container_name: jvm
    build: etmt-be
    ports:
      - "5050:5050"
```

Obrázek 4.20: Docker compose s definovanými službami klienta a serveru



Obrázek 4.21: Všech 5 běžících kontejnerů v Dockeru

Kapitola 5
Testování

V této kapitole se zaměříme na testování aplikace. Projdeme příklady testů pro klientskou i serverovou část aplikace. Otestujeme jak jednotlivé funkce izolovaně, tak celé funkční celky.

5.1 Unit Testy

Cílem unit testů je izolovat jednotlivé části aplikace a otestovat jejich funkčnost odděleně od ostatních. K tomu využijeme tzv. mockování. Mockování umožňuje nahradit v rámci testů závislosti testované části aplikace tzv. mocky, které působí jako náhrada za skutečnou implementaci. U mocků pak můžeme také sledovat, kolikrát bylo provedeno které volání a ověřit tak, že aplikace funguje správně.

5.1.1 JUnit a Mockito

JUnit a Mockito jsou knihovny umožňující unit testing v jazyce Java. JUnit umožňuje psát automatizované testy zatímco Mockito rozšiřuje možnosti těchto JUnit testů o již zmíněné mockování. V naší aplikaci se soustředíme na testování samotné business logiky. V této kapitole použijeme jako ukázku generování variant zkouškových testů, vzhledem k tomu, že jde o jednu z nejdůležitějších částí aplikace.

Testy generování zkouškových testů. Máme zde několik scénářů. Cílem je otestovat nejen tzv. "happy path"scénář, ale i případy, kdy dojde k chybě. V takovém případě sledujeme, že je chyba řádně ošetřená a nedojde k neočekávanému chování.

Samotné scénáře vychází z tabulky 4.1, kterou jsme zmínili již u implementace. Podle ní rozlišujeme různé případy, pro které jde a nejde vygenerovat zkouškový test, či očekávaný počet vygenerovaných testových variant. Podle této tabulky připravíme testovací scénáře, které lze vidět v tabulce 5.1.

Scénáře implementujeme v podobě unit testů s použitím knihovny JUnit 5 a Mockito. Ukázku unit testu pro scénář č.1 lze vidět na obrázku 5.1

Scénář	Vstup	Očekávaný výsledek
1	šablona s 1 tématem téma 1.: 1 otázka za 3 body v knihovně není žádná otázka	vyjímka: nedostatek otázek za 3 body
2	šablona s 1 tématem téma 1.: 1 otázka za 3 body v knihovně je 1 otázka za 3 body	úspěch: vygenerována 1 varianta
3	šablona se 2 tématy téma 1.: 1 otázka za 3 body téma 2.: 1 otázka za 5 bodů v knihovně je 1 otázka za 3 body a 2 otázky za 5 bodů	úspěch: vygenerovány 2 varianty
4	šablona se 2 tématy téma 1.: 1 otázka za 3 body téma 2.: 1 otázka za 5 bodů v knihovně jsou 2 otázky za 3 body a 2 otázky za 5 bodů	úspěch: vygenerovány 4 varianty
5	šablona se 2 tématy téma 1.: 2 otázky za 3 body téma 2.: 2 otázky za 5 bodů v knihovně je 1 otázka za 3 body a 2 otázky za 5 bodů	vyjímka: nedostatek otázek za 3 body

Tabulka 5.1: Testovací scénáře pro generování testových variant

5.1.2 Jest

Jest je testovací framework pro aplikace vytvořené pomocí Reactu. Jest umí mockovat funkce a React komponenty a ověřovat správné chování komponent pomocí příkazů `expect(<object>).toBe(<value>)` [23]. Při testování není spouštěna skutečná aplikace, ale jsou uměle renderovány testované komponenty, se kterými testy pracují. Jako příklad použití JESTu uvedeme test chování hlavního menu aplikace.

Test hlavního menu. U testování menu budeme mít následující scénář: Pokud je uživatel přihlášený, tlačítka `Sign in` a `Sign up` by mělá být vypnutá, zatímco tlačítko `Sign out` by mělo být aktivní. Jako druhý scénář bychom měli případ opačný: Pokud uživatel přihlášený není, očekáváme tlačítka `Sign in` a `Sign up` aktivní a tlačítko `Sign out` neaktivní. Na obrázku 5.2 lze vidět implementaci testu pro první scénář.

5.2 Integrační testy

Integrační testy se soustředí na testování více částí aplikace najednou, nebo testování integrace na další služby. Integračními testy můžeme otestovat

```
@Test
public void testGenerate_oneTopicOneQuestion_shouldThrowEx()
    throws Exception {
    // mock itemService po zavolání vrátí námi
    // specifikovanou prázndnou kategorii
    when(itemServiceMock.getItemById
        (eq(TEST_TEMPLATE_MINIMAL.getParentId()),
        any()))
    .thenReturn(TEST_ROOT_CATEGORY);

    // metoda generate nenajde otázky
    // proto očekáváme vyjímku
    var ex = assertThrows(
        Exception.class,
        () -> testSubject.generateTestVariantData(
                TEST_TEMPLATE_MINIMAL
            ));

    // ověříme zprávu u vyjímky
    assertEquals(
        ex.getMessage(),
        "not enough 3pt questions"
    );
}
```

Obrázek 5.1: Ukázka JUnit testu s použitím Mockita na scénáři č.1

například službu RESTovou službu PdfLaTeX, která funguje odděleně od serverové aplikace kterou testujeme.

5.2.1 Testování s aplikací Postman

Postman je nástroj pro testování endpointů API. Umožňuje vytvářet HTTP dotazy, ukládat je, organizovat do složek a vytvářet automatizované testy. Pro účely testování naší aplikace postačí testování jednotlivých endpointů serverové a fa-pdflatex aplikace.

Testování endpointů serverové aplikace. Začneme definicí samotného HTTP požadavku. Nejprve nastavíme URL požadavku a specifikujeme metodu POST, GET nebo DELETE. Poté určíme tělo požadavku a hlavičku. Pokud přistupujeme k zabezpečenému endpointu, je třeba nastavit autentizaci v sekci "Auth". V našem případě bychom použili možnost "Bearer Token"a vložili token, který dostaneme při přihlášení. Ukázku vzorového requestu v aplikaci Postman lze vidět na obrázku 5.3.

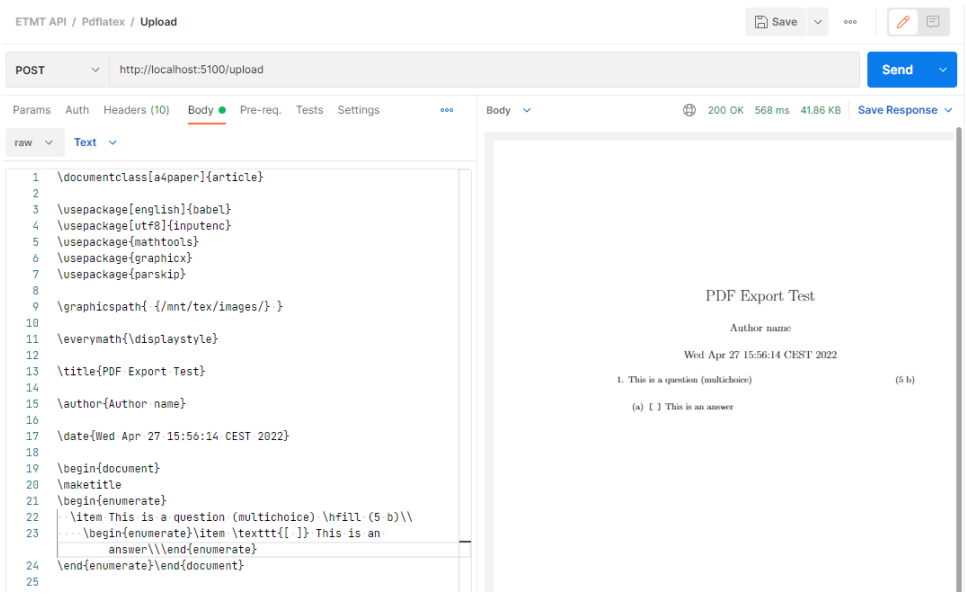
```
it('renders auth nav when signed in', async () => {
  // místo skutečné implementace autentizace použijeme mock
  // mock po zavolání isSignedIn() vrátí True
  useMockAuth.mockReturnValue(mockAuthContext)

  render(
    // testovaná komponenta
    <AuthNav />
  )

  // očekávané chování
  await waitFor(() => {
    // menu se úspěšně vyrenderovalo
    expect(screen.getByTestId("auth-nav-menu"))
        .toBeInTheDocument();

    // tlačítka se správně (ne)zobrazují
    expect(screen.getByTestId("auth-nav-menu-login"))
        .toHaveAttribute("aria-disabled", "true");
    expect(screen.getByTestId("auth-nav-menu-logout"))
        .toHaveAttribute("aria-disabled", "false");
  })
})
```

Obrázek 5.2: Ukázka testu React komponenty AuthMenu s použitím JEST

5.3 Uživatelské testování

Uživatelské testování proběhlo v přítomnosti vedoucího práce jakožto uživatele. Důraz byl kladen nejen na splnění požadavků vytyčených v analýze, ale i uživatelskou přívětivost a použitelnost v praxi. Hlavní oblastí testování bylo generování testových variant. Testování proběhlo jak s daty vloženými přímo přes aplikaci, tak s reálnými daty importovanými z Moodle.

Testování na datech z Moodle. Testování začalo importováním XML souboru vyexportované banky více jak 100 otázek z Moodle. Cílem bylo ověřit, že nástroj správně načte všechny otázky a jejich celkový obsah. Po úspěšném importu byla vytvořena šablona Test template se dvěma tématy - 5 otázek za 2 body a 2 otázky za 5 bodů. Následně byla šablona použita pro vygenerování variant zkouškových testů. Nástroj vygeneroval celkem 20 různých variant zkouškových testů. Několik vybraných variant bylo exportováno do formátu XML pro import v Moodle. Uživatel vznesl požadavek, že by nástroj měl dovolit označit použité testové varianty tak, aby šly rozpoznat od těch ještě nepoužitých. Také zmínil, že při opětovném generování testů by nástroj měl zabránit přepsání již použitých testů.



Obrázek 5.3: Úspěšný POST požadavek na fa-pdflatex server který vrátil vygenerované PDF jako odpověď

5.3.1 Změnové řízení

Během uživatelského testování prototypu vyvstaly následující změnové požadavky:

- CR1 - použité testové varianty by měly být označeny, aby je šlo rozlišit od těch nepoužitých a nedošlo k opětovnému zadání stejného testu
- CR2 - zvolená kategorie a její podstrom by se měly pro lepší čitelnost oddělit od hlavního stromu uživatelovy knihovny
- CR3 - v případě že v knihovně není dostatek otázek pro vygenerování testu z šablony, chybová hláška by se měla zobrazit už na náhledu samotné šablony
- CR4 - knihovna by měla být rozdělena na tři oddělené záložky v hlavním menu: banku otázek, šablony a vygenerované testy
- CR5 - před exportem do formátu PDF by se měl vyčistit [1] text otázek
- CR6 - při generování testových variant by měla aplikace zabránit opětnému vygenerování testu shodného s již existujícím, použitým zadáním

[1] alternativní termín pro vyčistění je desanitizace a znamená to odstranit mezery a speciální znaky

Kapitola 6
Závěr

Cílem této bakalářské práce bylo navrhnout, implementovat a otestovat nástroj pro zadávání a správu zkouškových testů. Tento cíl se podařilo naplnit. Během uživatelského testování byly vzneseny změnové požadavky, které bude vhodné implementovat v rámci dalších kroků.

6.1 Shrnutí

Tento projekt jsem si vybral jako příležitost implementovat funkční aplikaci s využitím frameworku React.

Projekt započal sběrem funkčních požadavků a analýzou tvorby zkouškových testů. Z této analýzy vznikl seznam funkčních požadavků a diagram případů užití, které jsem dále využil při analýze. Definice požadavků nebyla jednoduchá, nicméně s pomocí vedoucího práce se mi podařilo požadavky postupně specifikovat.

Následovala analýza dostupných nástrojů. Porovnal jsem několik vybraných nástrojů a došel k závěru, že žádný z nich nesplňuje definované požadavky. Během analýzy jsem se také zaměřil na generování zkouškových testů v aplikaci Moodle a identifikoval jeho slabé stránky.

V dalším kroku jsem se zaměřil na problém generování různých testových variant z množiny otázek, kde jsem prozkoumal možná řešení a vybral kombinatorické testovací metody jako kompromis mezi minimálním opakováním otázek mezi testovými variantami a maximálním počtem vygenerovaných variant.

S definovanými požadavky jsem zvolil vhodnou architekturu a technologie s ohledem na řešený problém a mé zkušenosti.

Při implementaci jsem využil své znalosti ze studia v oblasti Javy a Spring Boot. Technologie jako je TypeScript a React byly pro mne zcela nové. Při implementaci také došlo k několika změnovým požadavkům ze strany vedoucího práce z nichž byla implementována pouze část s nejvyšší prioritou, z časových důvodů spojených se studiem.

Na závěr jsem aplikaci otestoval pomocí unit a integračních testů. Uživatelské testování pak proběhlo s vedoucím práce, kdy jsme aplikaci s úspěchem otestovali na reálných datech exportovaných z aplikace Moodle.

6.2 Výstupy práce

Hlavním výstupem této práce je samotný nástroj v podobě webové aplikace. Zdrojový kód spolu s instalačními instrukcemi je umístěn v repozitáři na GitLabu, na adrese https://gitlab.fel.cvut.cz/jezekpe6/sem_pro. Obsah tohoto repozitáře je také jedinou přílohou této práce. Součástí repozitáře je soubor `README.md` s popisem adresářové struktury.

6.3 Další kroky

Nad rámec této práce je doporučena implementace následujících bodů:

- změnové požadavky zmíněné v sekci Změnové řízení
- uživatelský typ Admin s možností spravovat všechny uživatele a jejich knihovny
- nasazení do produkčního prostředí

6.4 Zhodnocení práce

Na úplný závěr bych chtěl krátce zhodnotit práci na tomto projektu, kterou hodnotím velice pozitivně. Projekt překonal má očekávání. Dal mi příležitost se naučit nové technologie a postupy a zároveň vytvořit funkční aplikaci řešící reálný problém. Rád bych vyzdvihl spolupráci s vedoucím projektu, který byl vždy ochoten konzultovat nejasnosti a navedl mě správným směrem, když jsem potřeboval.

Literatura

[1] tohecz, "Manual for LaTeX class ctuthesis", 2016
https://github.com/tohecz/ctuthesis/blob/master/ctuman.pdf

[2] Moodle Docs, version 3.11
https://docs.moodle.org/311/en/Main_page

[3] Capterra, "Best Exam Software", 2021
https://www.capterra.com/exam-software/?sortOrder=highest_rated

[4] NIST Covering Array Tables - "An Introduction to Covering Arrays", 2008
https://math.nist.gov/coveringarrays/coveringarray.html

[5] Czerwonka, Jacek. “Pairwise Testing in the Real World : Practical Extensions to Test-Case Scenarios.”, 2011

[6] Cohen, David M. et al. “The AETG System: An Approach to Testing Based on Combinatiorial Design.” IEEE Trans. Software Eng. 23 (1997): 437-444.

[7] Yu Lei and K. C. Tai, "In-parameter-order: a test generation strategy for pairwise testing,"Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231), 1998, pp. 254-261, doi: 10.1109/HASE.1998.731623.

[8] jesg/dither, "Collection of combinatorial test generation strategies", 2016
https://github.com/jesg/dither

[9] Avelon Pang, "TypeScript vs. JavaScript", Medium, 2021
https://medium.com/geekculture/typescript-vs-javascript-e5af7ab5a331

[10] TypeScript Documentation, version 4.6
https://www.typescriptlang.org/docs/

[11] React Documentation, version 18.1.0
https://reactjs.org/docs/getting-started.html

[12] React Router Documentation, version 6
https://reactrouter.com/docs/en/v6

[13] Ant Design, "Components Overview", version 4.20.5
https://ant.design/components/overview/

[14] Spring Framework Overview, version 5.3.20
https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html

[15] MongoDB Manual, version 5.0
https://www.mongodb.com/docs/manual/

[16] Open PDF, "Open source Java library for PDF files", version 1.3.28
https://github.com/LibrePDF/OpenPDF

[17] LaTeX Project, "An introduction to LaTeX"
https://www.latex-project.org/about/

[18] FastAPI Documentation, version 0.78.0
https://fastapi.tiangolo.com/

[19] Json Web Token, "Introduction"
https://jwt.io/introduction

[20] Lokesh Gupta, "What is REST", 2022
https://restfulapi.net/

[21] OpenAPI Specification, version 3.0.3
https://swagger.io/specification/

[22] Docker Documentation
https://docs.docker.com/

[23] Jest Documentation, version 28.1
https://jestjs.io/