FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Elektronická sbírka příkladů z matematiky
Student:	Petr Pondělík
Vedoucí:	Mgr. Petr Matyáš
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

* Proveďte rešerši stávajících aplikací pro správu zadání a řešení matematických příkladů.
* Na základě rešerše navrhněte elektronickou sbírku příkladů z matematiky ve formě webové aplikace.
* Aplikace bude umožňovat:
  - správu uživatelů,
  - správu a klasifikaci matematických příkladů podle tématu či obtížnosti, a to včetně jejich řešení,
  - generování příkladů se zadanými vlastnostmi (např. lineární rovnice, kvadratické rovnice, posloupnosti, apod.),
  - generování testů dle parametrů s výstupem do LaTeXu,
  - sledování statistiky úspěšnosti příkladů v jednotlivých testech,
  - procvičování pro studenty.
* Navrženou aplikaci implementujte v jazyce PHP s využitím dalších webových technologií.
* Výslednou aplikaci důkladně otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 30. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Elektronická sbírka příkladů z matematiky

Petr Pondělík

Katedra softwarového inženýrství
Vedoucí práce: Mgr. Petr Matyáš

2. ledna 2020

Poděkování

Děkuji panu Mgr. Petru Matyášovi za vedení této bakalářské práce, za trpělivost a ochotu kdykoli řešit problematiku spojenou s prací. Dále děkuji všem lidem ve svém okolí za nezměrnou trpělivost a podporu během celého studia. Poděkování patří zejména mé rodině, nejbližším přátelům a kolegům v práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. ledna 2020 ……

Odkaz na tuto práci

Pondělík, Petr. *Elektronická sbírka příkladů z matematiky.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá návrhem a implementací elektronické sbírky úloh z matematiky zaměřené na středoškolskou matematiku. Návrh a implementace aplikace jsou založeny na analýze stávajících webových aplikací pro správu zadání a řešení matematických úloh.

Výstupem této práce je webová aplikace umožňující správu uživatelů. Výstupní aplikace dále umožňuje správu a klasifikaci matematických úloh dle tématu či obtížnosti a podporuje generování zadání testů a úloh. Výstupní aplikace je implementována v PHP frameworku Nette.

Přínosem této práce je online podpora výuky středoškolské matematiky jak pro studenty, tak pro učitele. Výstupní aplikace poskytuje studentům možnost procvičování úloh, jež jim byly poskytnuty učitelem. Učitelům pak poskytuje rozhraní umožňující udržovat a strukturovat vlastní sbírku úloh, částečně automatizovat tvorbu zadání testů a spravovat studenty.

Klíčová slova webová aplikace, sbírka úloh z matematiky, podpora výuky online, e-learning, generátor zadání testů, LaTeX, PHP, Nette Framework

Abstract

The purpose of this bachelor thesis is to design and implement electronic database of mathematical problems focused on high school mathematics. Design and implementation are based on the analysis of current web applications designed for management of mathematical problems and it's solutions.

The output of this bachelor thesis is web application designed to manage mathematical problems and to classify the problems by theme and difficulty. The application is also designed to generate assignments of exams and mathematical problems and to manage users. The application is implemented in Nette Framework for PHP.

The benefit of this bachelor thesis is online learning assistance for high school mathematics. The assistance is provided for both students and teachers. The application provides students the opportunity to practise on mathematical problems provided by teacher. For teachers, the application provides interface that enables to maintain and structure own mathematical problems collection, partially automatize creation of tests and to maintain students.

Keywords web application, mathematical problems database, online learning assistance, e-learning, exam generator, LaTeX, PHP, Nette Framework

Obsah

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Úvod

Dnešní společnost se již těžko obejde bez služeb dostupných online, jež nalézají uplatnění v celé řadě procesů současného života. Mezi takovéto procesy patří například vzdělávání, které je ve stále větší míře podporováno e-learningovými portály či jinými aplikacemi, jež studentům poskytují studijní materiály a vyučujícím umožňují jejich tvorbu a sdílení.

Pro podporu výuky matematiky, stejně jako v případě ostatních oblastí vzdělávání, je možné využívat některé z existujících e-learningových portálů. Výhoda takových portálů je v univerzálnosti jejich použití. Může však nastat situace, kdy portál nevyhovuje specifickým požadavkům, jež jsou na něj kladeny konkrétní oblastí vzdělávání.

Zadavatel aktuálně postrádá nástroj pro podporu výuky matematiky vyhovující jeho specifickým požadavkům. Mezi tyto požadavky patří například centrální správa matematických úloh, správa uživatelů, poskytnutí úloh k procvičování studentům, tvorba šablon úloh a generování úloh ze zadaných šablon spolu s generováním zadání testů s výstupem do jazyka pro systém LaTeX.

V současnosti existuje řada webových stránek či aplikací zaměřených na podporu výuky matematiky. Až na výjimky se však jedná o sbírky úloh, které nedisponují možností registrace uživatelů a neposkytují tak ani žádné uživatelské rozhraní. Takové sbírky nacházejí uplatnění pouze v rámci procvičení látky studentem.

Výše uvedené požadavky zadávajícího navíc nejsou uspokojivě splněny žádným z existujících řešení. Skutečnost, že záměrem zadávajícího je výslednou aplikaci reálně využívat a že výstup má mít přidanou hodnotu nad ostatními řešeními stejné problematiky motivovalo autora k volbě tématu.

Kapitola 1

Cíl práce

Teoretická část práce se zaměří na rešerši stávajících webových aplikací pro správu zadání a řešení matematických úloh. Na základě provedené rešerše budou zvoleny vhodné vlastnosti, jež budou integrovány do vznikající aplikace. Dále budou specifikovány funkční a nefunkční požadavky na tuto aplikaci. Nakonec bude představen doménový model a případy užití aplikace.

Cílem praktické části práce je na základě funkčních a nefunkčních požadavků navrhnout a za využití jazyka PHP implementovat elektronickou sbírku příkladů z matematiky ve formě webové aplikace.

Aplikace poskytne správu uživatelů. Dále umožní správu a klasifikaci matematických úloh dle tématu či obtížnosti a správu jejich řešení. Aplikace umožní rovněž generování úloh[1] dle zadaných vlastností. Aplikace poskytne rozhraní pro generování testů s výstupem do jazyka pro systém LaTeX. U jednotlivých úloh bude navíc možné sledovat statistiku úspěšnosti v testech a studentům bude umožněno procvičování úloh.

Výsledná aplikace bude na závěr důkladně otestována.

[1] např. lineární rovnice, kvadratické rovnice či posloupnosti

Kapitola 2

Rešerše stávajících řešení

Tato kapitola pojednává o stavu existujících webových aplikací zabývajících se správou zadání a řešení matematických úloh, případně webových aplikací částečně řešících tuto problematiku.

Za aplikaci částečně řešící problematiku je považována taková aplikace, která umožňuje správu studijních materiálů a procvičování látky za využití těchto materiálů.

Nejprve budou specifikovány oblasti, v rámci nichž budou jednotlivá řešení prozkoumána. Následně budou vybrané aplikace zmapovány v rámci specifikovaných oblastí. Na závěr kapitoly bude sestaven výstup z průzkumu, jenž poslouží jako základ pro návrh vznikající aplikace.

2.1 Oblasti průzkumu stávajících řešení

Správa úloh Pro elektronickou sbírku je rozumná správa matematických úloh nutností. V rámci této oblasti budou uvedeny možnosti, jež zkoumané aplikace poskytují pro správu úloh či studijních materiálů.

Generování úloh Z pohledu vznikající aplikace je neméně důležitá podpora generování úloh. Generováním úloh je myšlena tvorba nových, v jisté míře náhodných úloh sbírky. V této oblasti budou zmapovány možnosti generování úloh poskytované zkoumanými aplikacemi.

Generování testů Výslednou aplikaci je zamýšleno využívat pro generování zadání testů s výstupem do formátu pro LaTeX. Tato oblast se zaměří na možnosti generování testů, jimiž disponují zkoumané aplikace.

Správa uživatelů Bez možnosti správy uživatelů se aplikace neobejde. Tato oblast se zaměří na vlastnosti správy uživatelů zkoumaných aplikací.

Procvičování pro studenty Úlohy elektronické sbírky musejí být dostupné rovněž studentům k procvičování. Tato oblast je zaměřena na možnosti, jež analyzované aplikace poskytují studentům za účelem procvičování.

Sledování statistiky úspěšnosti úloh Za účelem získání přehledu o úspěšnosti úloh v testech umožní výsledná aplikace tuto statistiku zapisovat a sledovat. V rámci této oblasti budou zmapovány způsoby sledování statistiky úspěšnosti úloh ve zkoumaných aplikacích.

2.2 Průzkum aplikací

Pro analýzu byla zvolena následující trojice webových aplikací:

- `MARAST`[2], [1]
- `CK-12`[3], [2]
- `Sbírkapříkladů.eu`[4]. [3]

2.2.1 MARAST

`MARAST` je portál pro správu úloh využívaný na FIT ČVUT, jenž nalezl uplatnění i mimo matematické kurzy. Portál disponuje rozsáhlou databází úloh, které se dělí do několika skupin. Úlohy z jedné skupiny jsou dostupné ve cvičebnici, úlohy z další skupiny jsou pak využívány pro generování zápočtových či zkouškových písemek a poslední skupina slouží pro sestavování on-line kvízů. [4]

Na základě informací uvedených výše portál umožňuje:

- správu úloh, jejich zařazování do kurzů a skupin,
- správu kvízů,
- generování zápočtových a zkouškových písemek.

2.2.1.1 Správa úloh

Úlohy lze přiřadit do kurzů, v rámci nichž jsou dále strukturovány ve dvou úrovních[5]. Jednotlivé úlohy jsou propojeny s autorem, kurzem, tématem a obtížností. K úlohám lze mimo jiné uvést jejich řešení a přiřazovat jim tagy.

[2] dostupné z *www.marast.fit.cvut.cz*
[3] dostupné z *www.ck12.org*
[4] dostupné z *www.sbirkaprikladu.eu*
[5] na způsob témat a podtémat

2.2.1.2 Generování úloh

Dle dostupných informací portál neposkytuje možnost generování úloh.

2.2.1.3 Generování testů

Na základě informací uvedených v úvodu sekce umožňuje administrace portálu generování zápočtových a zkouškových testů. Pro generování jsou využívány úlohy zařazené do skupiny úloh určených pro generování testů.

2.2.1.4 Správa uživatelů

Možnosti správy uživatelů v rámci portálu neměl autor možnost prozkoumat.

2.2.1.5 Procvičování pro studenty

Systém studentům zpřístupňuje úlohy dle zapsaných kurzů. Studentům je umožněno procvičování ve formě řešení úloh ze cvičebnic těchto kurzů.

V rámci cvičebnice jsou úlohy stránkovány po jednom záznamu a studentovi je umožněn pohyb mezi úlohami[6]. Ve výchozím stavu je řešení úlohy skryto, přičemž student může řešení úlohy přepínat.

Cvičebnice umožňuje filtrování úloh dle obtížnosti, přítomnosti postupu řešení, obtížnosti, tématu, poznámek studenta či tagů. Dále umožňuje vyhledat konkrétní úlohu na základě jejího číselného identifikátoru. Úlohy je možné řadit dle obtížnosti. Nastavení filtrů se nachází na kartě, jež je ve výchozím stavu sbalena. V případě, kdy je karta s filtry sbalena, zobrazuje se v jejím záhlaví popis nastavených filtrů.

Úlohy je studentovi umožněno označovat[7]. Student navíc může přidat k úloze vlastní poznámku či se zapojit do diskuze týkající se dané úlohy.

2.2.1.6 Sledování statistiky úspěšnosti úloh

Možnosti sledování statistiky úspěšnosti úloh neměl autor možnost prozkoumat.

2.2.2 CK-12

Webový portál `CK-12` poskytuje ucelenou sadu výukových nástrojů. Každý z těchto nástrojů lze přizpůsobit potřebám studenta, učitele či školy. [5]

Portál poskytuje dva oddělené moduly, konkrétně `Student` a `Teacher`. Každý z těchto modulů disponuje vlastní registrací a registruje uživatele v rolích `student` či `učitel`.

Nejedná se o portál specializovaný výhradně na výuku matematiky, nýbrž

[6] pomocí tlačítek předchozí a následující

[7] např. hvězdičkou či jako vyřešené

o univerzální e-learningový portál. Tento portál byl do rešerše zahrnut z důvodu výborné podpory zadávání matematických úloh na portál, o níž blíže pojednává sekce 2.2.2.1. Na základě provedeného průzkumu bylo objeveno hned několik možností využití portálu:

- samostudium dle studijních materiálů poskytovaných portálem,
- centralizovaná správa vlastních studijních materiálů,
- podpora výuky pro učitele cílená konkrétním skupinám studentů.

Portál poskytuje širokou škálu studijních materiálů zahrnujících například matematiku, fyziku, chemii či biologii. Studijní materiály poskytované portálem jsou přístupné i anonymním uživatelům[8]. Za účely umisťování vlastních materiálů na portál je nutná registrace v libovolné roli. Studijní materiály je na portálu možné udržovat například ve formátu `FlexBook`[9].

2.2.2.1 Správa úloh

Uživatel má v rámci aplikace k dispozici knihovnu obsahující veškeré jeho materiály. V uživatelské knihovně se mohou nacházet vlastní materiály uživatele, výchozí materiály poskytované portálem, či jemu zpřístupněné materiály jiných uživatelů.

Materiálem může být `modalita`[10], `FlexBook` či `kvíz`. Modalita může obsahovat například text, seznamy, tabulky, obrázky či matematické výrazy zadávané ve formátu pro LaTeX. Z toho vyplývá, že modality poskytují výborné možnosti pro zadávání matematických úloh v rámci studijních materiálů.

Modality je možné zařazovat do `FlexBook` a za jejich využití tak modality strukturovat. Jakýkoli studijní materiál lze řadit do složek realizovaných ve formě tagů. Tato akce je k dispozici ve hromadné variantě.

Materiály umístěné do knihovny je možné filtrovat dle typu[11], způsobu přidání do knihovny[12] či dle složek.

2.2.2.2 Generování úloh

Generování úloh není v rámci aplikace k dispozici.

2.2.2.3 Generování testů

Aplikace neumožňuje generování testů. Díky své obecnosti však poskytuje možnost ručního vytvoření testu, jež spočívá ve vytvoření modality obsahující zadání testu a jejím následném stažení ve formátu PDF.

[8] neregistrovaným návštěvníkům

[9] interaktivní online učebnice umožňující např. vkládání poznámek či zvýraznění textu

[10] základní jednotka materiálu na portálu `CK-12`

[11] např. zda se jedná o `modalitu`, `FlexBook`, či `kvíz`

[12] materiál vytvořený uživatelem, či materiál převzatý od jiného uživatele

2.2.2.4 Správa uživatelů

Modul `Teacher` umožňuje učiteli zakládat třídy. Do tříd je následně možné zvát studenty, manuálně je přidávat či zaslat žádost o hromadný import studentů administrátorům portálu. Učiteli je dále umožněno studenty ze tříd odebírat.

Pozvání studentů do třídy je realizováno formou e-mailu. Žádost o hromadný import studentů obsahuje CSV soubor se seznamem studentů, jehož řádka povinně obsahuje uživatelské jméno, heslo, jméno, příjmení a e-mailovou adresu. Manuálně lze studenty přidat dvěma způsoby:

- založením účtu studenta bez e-mailové adresy,
- výběrem studentů, již patří do některé z dalších tříd učitele.

V případě účtu studenta, jenž byl manuálně vytvořen učitelem, je učiteli k dispozici editace tohoto uživatelského účtu. V případě účtu registrovaného studentem nemá učitel možnost takový účet editovat.

2.2.2.5 Procvičování pro studenty

Učiteli je umožněno poskytovat vedeným třídám studijní materiály z jeho uživatelské knihovny, jejíž možnosti byly popsány v sekci 2.2.2.1. Poskytnuté materiály jsou studentům dostupné v detailu konkrétních tříd, jejichž jsou členy.

Kromě sdílení materiálů je učiteli k dispozici možnost zadávat třídám úkoly. Úkolem může být stejně jako v případě studijních materiálů libovolná položka z knihovny učitele. Oproti běžným studijním materiálům je v případě úkolu možné stanovit termín pro jeho splnění.

2.2.2.6 Sledování statistiky úspěšnosti úloh

Modul `Teacher` poskytuje učiteli reporty pro jednotlivé třídy. Reporty jsou ve formě tabulky, kde jedna dimenze reprezentuje seznam studentů dané třídy a druhá dimenze reprezentuje seznam úloh přiřazených dané třídě. Buňky tabulky obsahují informaci o míře splnění úkolu[13].

2.2.3 sbirkaprikladu.eu

`Sbirkaprikladu.eu` je webový portál sloužící jako veřejná sbírka úloh z matematiky. Sbírka umožňuje uživatelům vyhledávat a filtrovat úlohy či nahlížet na matematické problémy z více úhlů pohledu. Úlohy jsou vytvářeny mnoha učiteli, čímž je zaručena jejich rozmanitost. [6]

Kromě toho, že portál je přístupný anonymním uživatelům, umožňuje re-

[13] (ne)splněno či procentuální úspěšnost v závislosti na typu úlohy

gistraci ve dvou rolích. Konkrétně se jedná o role `registrovaný uživatel` a `tvůrce`. [7]

2.2.3.1 Správa úloh

Aplikace umožňuje víceúrovňové strukturování úloh do témat a podtémat. Úlohy jsou provázány s autorem a lze jim přiřadit stupeň školy a obtížnost dle autora.

Uživateli v roli `registrovaný uživatel` je umožněno vytváření a zpřístupňování vlastních uživatelských sbírek. Pro vytvořenou sbírku lze nastavit, zda u obsažených úloh má být zobrazen výsledek, případně postup řešení. V detailu každé úlohy, jež je pro uživatele v rámci aplikace viditelná, je k dispozici akce zařazení úlohy do zvolené sbírky. `Registrovaný uživatel` nemá oprávnění ke správě úloh samotných.

Uživatel v roli `tvůrce` má rozšířená oprávnění oproti roli `registrovaný uživatel`. Disponuje navíc oprávněním vytvářet nové příklady, řadit je do témat a přiřazovat jim příznaky. Zároveň má oprávnění k editaci a odstranění jím vloženého obsahu. [7]

2.2.3.2 Generování úloh

Aplikace disponuje nástrojem pro jednorázové generování úloh dostupným v její veřejné sekci. Nástroj tvoří skupina generátorů, z nich každý je určený pro striktně specifikovaný[14] typ úlohy. K vygenerované úloze je každý z generátorů schopný získat řešení.

2.2.3.3 Generování testů

Aplikace neumožňuje generování testů.

2.2.3.4 Správa uživatelů

Správa uživatelů není k dispozici ani jedné z rolí, pod níž je registrace umožněna. Tímto oprávněním disponují výhradně administrátoři aplikace.

2.2.3.5 Procvičování pro studenty

Všem uživatelům, včetně anonymních, jsou k dispozici úlohy zveřejněné v aplikaci. K dispozici je procvičování ve formě řešení těchto úloh.

Aplikace úlohy stránkuje po pěti záznamech. V rámci položek stránkovaného seznamu je k dispozici stručný přehled dané úlohy[15]. Ze seznamu je možné přejít na detail konkrétní úlohy. Ten může obsahovat postup řešení a výsledek. V detailu je možné dynamicky skrývat a zobrazovat zadání, po-

[14] s daným schématem zadání, kde se mění pouze číselné hodnoty, nikoli struktura úloh

[15] obsahuje zadání, stupeň školy, obtížnost dle autora a hodnocení dle uživatelů

stup řešení i výsledek. Jednotlivé úlohy lze exportovat do formátu PDF.

Pokud je student registrovaným uživatelem, může sestavovat vlastní sbírky sdružující úlohy, jež zamýšlí procvičovat.

2.2.3.6 Sledování statistiky úspěšnosti úloh

Aplikace neposkytuje možnost sledovat statistiku úspěšnosti úloh.

2.3 Výstup z průzkumu

Na základě provedeného průzkumu byl získán přehled o vlastnostech a možnostech zkoumaných aplikací z pohledu požadavků na vznikající aplikaci. V této sekci bude proveden výběr vhodných vlastností, jež výsledná aplikace převezme. Ty se dále stanou součástí funkčních požadavků analyzovaných v sekci 3.1.

2.3.1 Obecné vlastnosti

2.3.1.1 Modularita

Aplikace `CK-12` zkoumaná v sekci 2.2.2 je rozdělena do dvou modulů, kdy modul `Student` je určen uživatelům v roli `student` a modul `Teacher` uživatelům v roli `učitel`.

Modulární architektura aplikace vede k její snazší rozšiřitelnosti a umožňuje zapouzdření jejích logických celků. Z těchto důvodů bude vznikající aplikace navržena rovněž v modulární architektuře.

2.3.1.2 Uživatelské role

Aplikace `Sbirkaprikladu.eu` implementuje práva uživatelských rolí systémem, kdy vyšší uživatelská role dědí všechna oprávnění od nižší uživatelské role a obdrží určitá oprávnění navíc. Stejné hierarchické pojetí uživatelských rolí převezme vznikající aplikace.

2.3.2 Správa úloh

Během průzkumu bylo zjištěno, že `sbirkaprikladu.eu` umožňuje úlohy vkládat do víceúrovňové struktury témat, `MARAST` podporuje strukturování úloh pomocí témat a podtémat, zatímco `CK-12` vkládání modalit do `FlexBook`. V rámci `CK-12` a `sbirkaprikladu.eu` byla navíc zjištěna podpora CRUD operací nad úlohami, případně studijními materiály.

Ze získaných poznatků vyplývá, že pro rozumnou správu úloh musí vznikající aplikace poskytovat:

- alespoň dvouúrovňové strukturování úloh[16],
- zařazování úloh do této struktury,
- CRUD operace nad úlohami, tématy a podtématy.

Zkoumané aplikace u jednotlivých úloh zobrazovaly sadu základních informací. Z tohoto důvodu budou úlohy v rámci vznikající aplikace provázány alespoň s **autorem**, **tématem** a **obtížností**.

2.3.3 Generování úloh

Během průzkumu bylo zjištěno, že generování úloh poskytuje ze zkoumaných aplikací pouze `sbirkaprikladu.eu`. Zde se však jedná o jednorázové generování za účelem procvičování, nikoli o generování ve smyslu vytváření nových úloh sbírky, což není pro vznikající aplikaci vyhovující. Generované úlohy mají navíc předem danou a neměnnou strukturu.

2.3.4 Generování testů

Generování testů je podporováno rovněž pouze jedinou z analyzovaných aplikací, konkrétně portálem `MARAST`. Na základě zjištěných informací poskytne vznikající aplikace možnost oddělení úloh dostupných studentům k procvičování od úloh využívaných pro generování testů.

2.3.5 Správa uživatelů

Vznikající aplikace musí poskytovat rozumnou práci s uživateli. V rámci provedeného průzkumu bylo zjištěno, že aplikace `CK-12` umožňuje zakládat třídy, do nichž lze studenty zařazovat několika způsoby.

Na základě tohoto poznatku výsledná aplikace umožní tvorbu uživatelských skupin, do nichž bude učitel moci studenty zařazovat. Dále bude umožněna správa uživatelských skupin

2.3.6 Procvičování pro studenty

Portál `MARAST` zpřístupňuje studentům úlohy dle zapsaných kurzů. Vznikající aplikace poskytne obdobný systém zpřístupnění úloh, a to na základě příslušnosti studentů k uživatelským skupinám.

`MARAST` poskytuje studentům cvičebnice jednotlivých kurzů. Stejně tak vznikající aplikace studentům poskytne zpřístupněné úlohy ve cvičebnici, jež se bude nacházet v modulu `Student`.

[16] na způsob témat a podtémat

Cvičebnice převezme vlastnosti od cvičebnice MARASTu a umožní:

- stránkování po jedné úloze,
- pohyb na předchozí a následující úlohu,
- přepínání zobrazení výsledku[17],
- filtrování a řazení úloh.

2.3.7 Sledování statistiky úspěšnosti úloh

Této oblasti byla věnována pozornost především během průzkumu portálu `CK-12`. Modul `Teacher` poskytuje pro jednotlivé třídy uživatelsky konfigurovatelné reporty zobrazující v tabulkách míru splnění či míru úspěšnosti úkolů přiřazených dané třídě.

Jedná se tedy o sledování úspěšnosti úkolů zadaných skupinám uživatelů, zatímco vznikající aplikace má umožnit sledování úspěšnosti úloh v rámci vytvořených testů.

[17] ve výchozím stavu bude skrytý

Kapitola 3

Analýza

V rámci této kapitoly bude provedena analýza funkčních a nefunkčních požadavků na vznikající aplikaci. Dále prozkoumá požadavky kladené na uživatelské role vystupující v aplikaci a zanalyzuje případy užití vyplývající z funkčních požadavků. Na závěr kapitoly bude zmapován doménový model aplikace rozdělený dle logických celků.

3.1 Funkční požadavky

Tato sekce přiblíží funkční požadavky kladené na vznikající aplikaci. Vybrané funkční požadavky jsou ovlivněny výstupem z průzkumu, jímž se zabývá sekce 2.3.

3.1.1 Filtrování a řazení

Z důvodu usnadnění práce s daty v rámci modulu `Teacher` musí aplikace poskytovat smysluplné filtrování a řazení při práci se seznamy entit.

3.1.2 Správa matematických úloh

Vznikající aplikace musí umožnit správu matematických úloh, a to včetně jejich řešení. V této sekci budou přiblíženy funkční požadavky kladené na zmíněnou sekci aplikace s ohledem na výstup z průzkumu v sekci 2.3.2.

Jednotlivé úlohy budou provázány s obtížností, podtématem a jejich autorem. Z povahy obsahu sbírky a na základě provedené rešerše je nezbytné, aby aplikace umožnila alespoň **dvouúrovňové strukturování úloh** ve smyslu **témat** a **podtémat**. Dle poznatků z rešerše dále aplikace umožní oddělit úlohy určené pro generování testů od úloh zobrazovaných studentům ve cvičebnici. Nad úlohami, jejich tématy a podtématy budou umožněny CRUD operace.

S ohledem na následující funkční požadavek musí aplikace rozlišovat mezi jednotlivými **typy šablon úloh** sloužícími pro generování **finálních úloh** a mezi finálními úlohami samotnými.

3.1.3 Generování úloh

Aplikace umožní generování úloh vybraných typů. Tato sekce blíže specifikuje aktuální funkční požadavek s ohledem na sekci 2.3.3.

3.1.3.1 Základní vlastnosti

Generování úloh bude probíhat na základě specifických **typů šablon úloh** zadávaných do aplikace. Je nutné umožnit zadávání šablon ve volném tvaru a neomezovat se na předem danou strukturu úloh. Generování finálních úloh z šablon bude využito v rámci generování zadání testů.

3.1.3.2 Náhodné parametry

Uživatel bude zadávat šablony ve formátu jazyka pro LaTeX, v rámci nichž specifikuje náhodné parametry formou nepárových XML značek. Pomocí atributů parametru uživatel určí interval hodnot, z něhož může být hodnota parametru vygenerována.

3.1.3.3 Podmínky šablon úloh

Kromě určení intervalu generovaných hodnot formou atributů parametru aplikace umožní přiřazovat k šabloně podmínky vztahující se k ní jakožto k celku. Tím uživateli umožní specifikovat globální podmínky pro generovaný výstup[18].

3.1.4 Generování a správa testů

Nezbytnou součástí modulu `Teacher` je sekce pro generování a následnou správu testů. Tato sekce přiblíží aktuální funkční požadavek s ohledem na sekci 2.3.4.

3.1.4.1 Základní vlastnosti

Nad testy budou k dispozici CRUD operace. Operace `update` přitom bude závislá na stavu editovaného testu.

Možnost oddělit úlohy určené pro zobrazení ve cvičebnici od úloh určených pro generování testů vyplývá z funkčního požadavku 3.1.2.

[18] např. podmínka odmocnitelnosti diskriminantu v případě šablony kvadratické rovnice

3.1.4.2 Vygenerování testu

Za vygenerovaný test je považována skupina souborů tvořená variantami zadání ve formátu pro LaTeX. Rozhraní pro generování testů musí uživateli umožnit zadání statických údajů, jež tvoří hlavičku testu a uvozují zadání. Konkrétně se jedná o:

- počet variant zadání testu[19],
- logo testu,
- cílové skupiny,
- školní rok,
- období školního roku[20],
- pořadové číslo testu,
- úvodní text[21].

Dále aplikace umožní přiřadit do testu úlohy. V rámci přiřazení úlohy je nutné umožnit filtrování pro specifikaci množiny úloh, z níž má být úloha zvolena a zařazena do testu. Uživatel bude mít dále kontrolu nad zalamováním stran vygenerovaných zadání.

3.1.4.3 Uzamčení zdrojových souborů

Aplikace poskytne uživateli akci pro převod vygenerovaného testu ze stavu `otevřený` do stavu `uzavřený`, která způsobí přepnutí režimu operace `update` nad daným testem. Uzavřený test bude možné editovat jen omezeně, a to v takové míře, která nezpůsobí nekonzistenci mezi entitou testu a vygenerovanými zdrojovými soubory testu.

3.1.4.4 Přegenerování testu

V rámci sekce pro správu testů bude uživateli k dispozici akce pro přegenerování uzavřeného testu. Na základě zvoleného testu pro přegenerování budou novému testu předvyplněny údaje, jež bude možné editovat. Uživatel dále bude moci zvolit, které úlohy umístěné do zadání testu mají být přegenerovány.

3.1.4.5 Stažení zdrojových souborů

Nutným požadavkem na aplikaci je možnost stažení archivu s vygenerovanými variantami zadání testu ve formátu jazyka pro LaTeX.

[19] na jeho základě dojde k vygenerování odpovídajícího počtu variant
[20] např. pololetí
[21] uvozuje zadání testu a slouží pro uvedení globálních informací souvisejících se zadáním

3.1.4.6 Usnadnění kompilace do PDF

Aplikace uživateli může, ale nemusí poskytovat usnadnění kompilace zdrojových souborů zadání do formátu PDF. Smyslem tohoto požadavku je poskytnout akci vázanou k testu, jež by z větší části automatizovala proces spojený s kompilací souborů testu ve formátu pro LaTeX do PDF.

3.1.5 Správa uživatelů

Aplikace musí umožnit správu uživatelů. Tato sekce specifikuje funkční požadavky kladené na správu uživatelů s ohledem na sekci 2.3.5.

3.1.5.1 Základní operace

Modul `Teacher` poskytne CRUD operace nad uživateli a uživatelskými skupinami. Aplikace umožní přiřazovat uživatele do více uživatelských skupin.

3.1.5.2 Úrovně uživatelských skupin

Systém uživatelských skupin musí být připraven k univerzálnímu použití. Tento požadavek vyplývá ze skutečnosti, že zadavateli může vzniknout potřeba vést v aplikaci uživatele pocházející z odlišných zdrojů[22]. Z tohoto důvodu musí být logika uživatelských skupin alespoň dvouúrovňová, kdy nad vrstvou uživatelských **skupin** bude existovat vrstva **superskupin**.

3.1.5.3 Pozvání do aplikace

Po úspěšném vytvoření uživatelského účtu aplikace sestaví a odešle pozvánku do aplikace na zadanou e-mailovou adresu uživatele. Tato pozvánka musí obsahovat všechny údaje potřebné pro vstup do aplikace a musí z ní být srozumitelné, jak do aplikace přistoupit.

3.1.5.4 Automatizovaná správa hesel

Uživatelská hesla nebudou v modulu `Teacher` poskytnuta k manuální správě. Aplikace musí implementovat automatizovaný mechanismus, jenž zajistí generování náhodných uživatelských hesel během operace `create` nad uživateli. Dále bude v modulu `Teacher` umožněno přegenerování a opětovné zaslání hesla na e-mailovou adresu daného uživatele.

Uživatel bude mít možnost změnit své heslo jak po přihlášení, tak v případě jeho zapomenutí.

[22] např. uživatelé ze střední školy a uživatelé z externích kurzů

3.1.6 Procvičování pro studenty

Aplikace poskytne studentům rozhraní, jež umožní procvičování úloh z témat zpřístupněných danému studentovi. Tato sekce přiblíží aktuální funkční požadavek s ohledem na sekci 2.3.6.

V rámci jednotlivých témat bude uživatel moci procházet příslušnou sadu úloh. Aplikace bude úlohy stránkovat po jednom záznamu a rozhraní dále umožní:

- filtrování a řazení úloh,
- pohyb na předchozí a následující úlohu,
- přepínání zobrazení výsledku.

3.1.7 Sledování statistiky úspěšnosti úloh

K vygenerovanému testu modul `Teacher` uživateli umožní zobrazit detail, jenž mu poskytne přehled úloh obsažených v daném testu. V rámci detailu bude možné editovat statistku úspěšnosti úloh v daném testu. Na základě těchto údajů bude aplikace sledovat celkovou průměrnou úspěšnost úloh.

3.1.8 Správa oprávnění k úlohám

Modul `Teacher` umožní správu přístupových oprávnění k vytvořeným úlohám. Tato funkcionalita umožní zpřístupňovat a skrývat jednotlivá témata úloh jak na úrovni skupin, tak na úrovni superskupin uživatelů.

3.2 Nefunkční požadavky

Tato sekce se zabývá analýzou nefunkčních požadavků na aplikaci, a to včetně nefunkčních požadavků, které jsou typické pro webovou aplikaci.

3.2.1 Webová aplikace

V této sekci budou popsány typické nefunkční požadavky kladené na webové aplikace. Dle [8] a [9] mezi ně patří například trojice dále popsaných vlastností.

3.2.1.1 Responzivní design

Responzivní design označuje design uživatelského rozhraní, jenž se snadno přizpůsobí rozlišení všech běžně používaných zobrazujících zařízení. Responzivní design reaguje na změny v rozlišení formou přeskupení prvků rozhraní tak, aby si rozhraní udrželo grafický vzhled v rámci dostupného prostoru. Responzivní design je blízký designu **adaptivnímu**. Rozdíl mezi těmito dvěma přístupy je dán skutečností, že responzivní design spočívá v dynamickém přizpůsobení de-

signu vzhledem k aktuálnímu rozhraní, zatímco v případě adaptivního designu jsou prvky rozhraní rozloženy fixně pro určená rozlišení. [10]

V současné době se navíc responzivní design stává zásadní složkou SEO pro mobilní zařízení, jelikož vyhledávače v mobilních zařízeních upřednostňují ve výsledcích vyhledávání právě webové stránky a aplikace disponující responzivním designem. [8]

Dle [11] navíc z mobilních platforem přistupuje na web v průměru více než 50 % uživatelů. Propad ve výsledcích vyhledávání u více než poloviny uživatelů představuje pro web značný problém.

3.2.1.2 Použitelnost

Vlastnost systému nazývaná použitelnost dle [9] specifikuje, jak obtížné je pro uživatele naučit se systém ovládat.

Použitelnost může být chápána několika různými způsoby:

Efektivita práce: průměrný čas potřebný pro dosažení cíle, kolik úkonů v systémů je uživatel schopný provést bez nápovědy.

Intuitivní rozhraní: jak obtížné je porozumět rozhraní, jeho struktuře či používaným ikonám.

Jednoduchost práce: kolik pokusů musí uživatel provést pro splnění úkolu.

3.2.1.3 Nevyužívat Adobe Flash

Využití technologie Adobe Flash na webu s sebou přináší řadu problémů. Mezi nejvýznamnější z nich patří:

- chybějící podpora u zařízení se systémem iOS,
- problematické indexování obsahu flash vyhledávačem Google,
- obtížné a pro vývojáře náročné updaty. [12]

První problém na základě [11] prakticky znamená, že veškerý obsah flash je nedostupný v průměru pro téměř 11 % uživatelů. Druhý problém může být závažným za podmínky, že se ve formátu Adobe Flash nachází sémantický obsah webu, jenž by měl být zohledňován během indexace. Třetí z problémů je překážkou pro vývojáře, jelikož realizace změn v HTML a CSS je časově efektivnější oproti změnám ve formátu Adobe Flash. [12]

3.2.2 Rozšiřitelnost a modularita

Dalším požadavkem kladeným na aplikaci je navrhnout architekturu aplikace tak, aby byla umožněna její budoucí **rozšiřitelnost**.

S přihlédnutím na výstup z průzkumu v sekci 2.3.1 musí být architektura aplikace navržena **modulárně**.

3.2.3 Technologie

Tato sekce se zabývá požadavky na serverové a databázové technologie, které vyplynuly z konzultací vedených se zadavatelem, kdy bylo přihlédnuto rovněž ke zkušenostem autora. Nyní bude provedena analýza požadovaných technologií na straně serveru a požadovaného systému řízení báze dat.

Následující technologie budou v sekci 4.1 upřesněny a doplněny o technologie na straně klienta zvolené pro implementaci aplikace.

3.2.3.1 PHP

PHP představuje rozšířený multiplatformní skriptovací jazyk, jenž je vhodný zejména pro tvorbu webových aplikací či dynamických webových stránek. Jedná se o serverovou technologii, zdrojový kód PHP je tedy prováděn na straně serveru. K uživateli[23] je po vykonání skriptů odeslán pouze výsledek jejich běhu.

Mezi výhody PHP patří:

- snadné základy jazyka,
- podpora funkcionálního i objektově orientovaného programování,
- velké množství návodů,
- početná komunita vývojářů,
- více než 240 000 open-source knihoven a balíčků,
- správa balíčků pomocí nástroje Composer,
- možnost využití knihoven napříč frameworky. [13] [14]

3.2.3.2 MySQL

MySQL je nejpopulárnější open-source SQL DBMS. DBMS je systém umožňující přistupovat k datům uloženým v databázi a tato data spravovat. MySQL je primárně open-source software distribuovaný pod bezplatnou licencí GPL. Existuje však i placená komerční licence, jež je potřebná v případě využití vlastního zdrojového kódu MySQL pro komerční účely. [15]

MySQL je multiplatformní relační databáze. Relační databáze je založena na **relačním modelu**, který spočívá ve sdružování dat do tzv. **relací**[24]. Tyto relace obsahují n-tice[25]. Relace tvoří základ relační databáze. Jedná se o strukturu záznamů s pevně danými atributy. Každý atribut má jednoznačně definován typ, název a rozsah. Tyto tři prvky tvoří dohromady **doménu atributu**. Atributy stejného typu mohou tvořit vazby mezi relacemi.

[23] na stranu klienta
[24] tabulek
[25] řádky

Mezi výhody MySQL patří:

- open-source licence,
- rychlost,
- spolehlivost,
- škálovatelnost,
- snadné použití. [15]

3.3 Uživatelské role

Vznikající aplikace má být určena pouze uživatelům vlastnícím uživatelský účet. Z toho důvodu nebude poskytovat veřejnou část přístupnou anonymním uživatelům. Pro vstup do neveřejných částí aplikace budou uživatelé muset projít procesy autentizace a autorizace, jejichž implementací se zabývá sekce 5.7.

Dle výstupu z provedené rešerše v sekci 2.3.1 budou oprávnění uživatelských rolí implementována hierarchicky. Tato sekce dále přiblíží uživatelské role vystupující v aplikaci a jejich oprávnění.

Administrátor

Administrátor má přístup do modulů `Student` a `Teacher`, v rámci nichž má plná oprávnění ke všem funkcionalitám vyplývajícím z funkčních požadavků uvedených v sekci 3.1.

Učitel

Učitel má přístup do modulů `Student` a `Teacher`. V rámci modulu `Student` má plná oprávnění, zatímco v modulu `Teacher` má oprávnění omezená. Na rozdíl od administrátora nemá oprávnění pro správu entit, jejichž tvůrcem je jiný uživatel.

Student

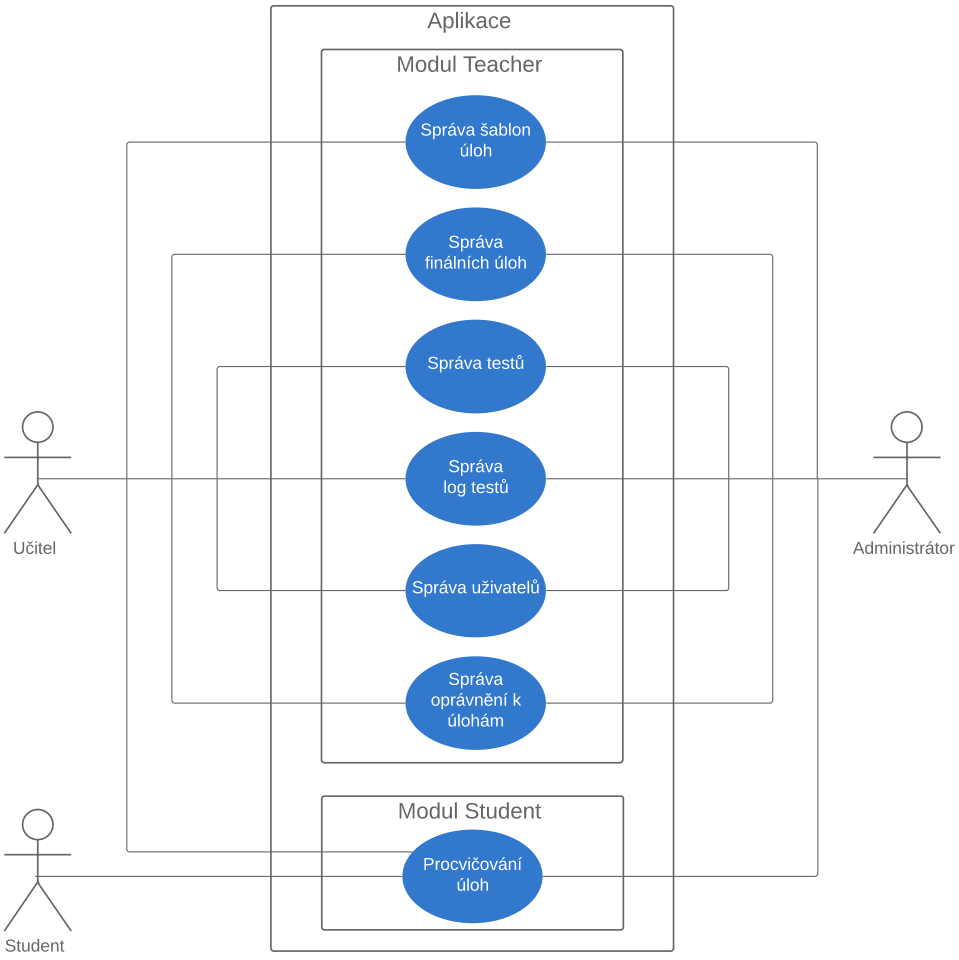
Student má oprávnění k přístupu pouze do modulu `Student`, jenž mu poskytne cvičebnici určenou k procvičování úloh. V rámci cvičebnice jsou studentovi přístupná pouze témata úloh, jež mu zpřístupnil uživatel[26] vyšší role.

[26] zpravidla `Učitel`

3.4 Případy užití

Tato sekce se soustředí na analýzu případů užití aplikace. Všechny případy užití vyplynuly z funkčních požadavků kladených na vznikající aplikaci uvedených v sekci 3.1 a zohledňují uživatelské role specifikované v rámci předchozí sekce. Role `Administrátor` a `Učitel` budou v rámci diagramů případů užití působit shodně, jelikož mají dostupné ty samé akce, pouze `Administrátor` má přístup i k entitám vytvořeným ostatními uživateli.

Z důvodu složitosti diagramů některých případů užití byl zvolen přístup, kdy bude nejprve uveden diagram případů užití znázorňující balíčky případů užití. Tyto balíčky budou následně rozvedeny v samostatných diagramech.



Obrázek 3.1: Diagram případů užití aplikace

3.4.1 Globální případy užití

Některé případy užití aplikace jsou naprosto či téměř totožné pro více balíčků případů užití. Tato sekce se zaměří na rozbor takovýchto případů užití. V analýze jednotlivých balíčků již tyto případy užití nebudou duplicitně analyzovány.

3.4.1.1 Zobrazení seznamu entit

Uživatel zobrazí seznam entit kliknutím na příslušnou položku v navigaci aplikace, čímž dojde k přesměrování na seznam entit příslušného typu. V rámci seznamu jsou uživateli v roli `Administrátor` zobrazeny všechny entity, zatímco uživateli v roli `Učitel` pouze jím vytvořené entity.

Filtrování seznamu entit uživatel navolí v záhlaví filtrovatelných sloupců, řazení pak kliknutím na titulek sloupce, dle kterého je řazení umožněno.

3.4.1.2 Odstranění entity

Uživatel u konkrétní entity v seznamu zvolí akci odstranění. V případě, že existuje entita jiného typu, jež je povinně v relaci s odstraňovanou entitou[27], k jejímu odstranění nedojde a aplikace uživatele na tuto skutečnost upozorní. V korektním případě dojde k odstranění entity a aktualizaci seznamu.

3.4.2 Správa šablon úloh

Tato sekce se zabývá analýzou případů užití v rámci správy **šablon úloh**. Jednotlivé případy užití jsou znázorněny na diagramu 3.2.

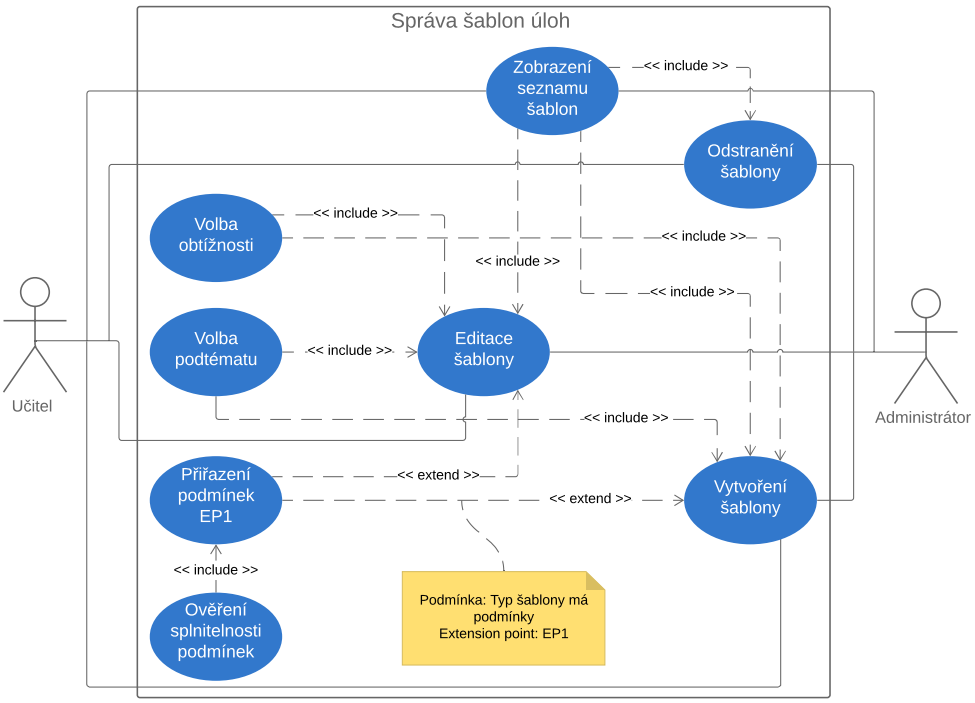
3.4.2.1 Zobrazení seznamu šablon úloh

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.1.

3.4.2.2 Vytvoření šablony úloh

Uživatel zvolí akci vytvoření dostupnou v hlavičce karty se seznamem, čímž dojde k zobrazení formuláře pro tvorbu šablon úloh v modálním okně. Typ šablony přiřadí aplikace automaticky. Uživatel zvolí podtéma a obtížnost. V případě, že k typu šablony jsou dostupné podmínky, může je uživatel k šabloně přiřadit. Pokud jsou k šabloně přiřazeny podmínky, uživatel musí ověřit jejich splnitelnosti. Uživatel vyplní zadání úlohy, případně úvod zadání a dodatek k zadání. V případě chybného vstupu bude upozorněn na chyby. V opačném případě bude vytvořena nová šablona úlohy a dojde k aktualizaci seznamu.

[27] např. šablona úlohy či finální úloha zařazená do některého z vygenerovaných testů



Obrázek 3.2: Diagram případů užití – správa šablon úloh

3.4.2.3 Editace šablony úloh

Uživatel zvolí akci editace konkrétní šablony ze seznamu a bude přesměrován na její editaci. Zde provede úkony popsané v předchozí sekci a kliknutím na `Uložit` potvrdí změny. V případě chybného vstupu bude upozorněn na chyby.

3.4.2.4 Odstranění šablony úloh

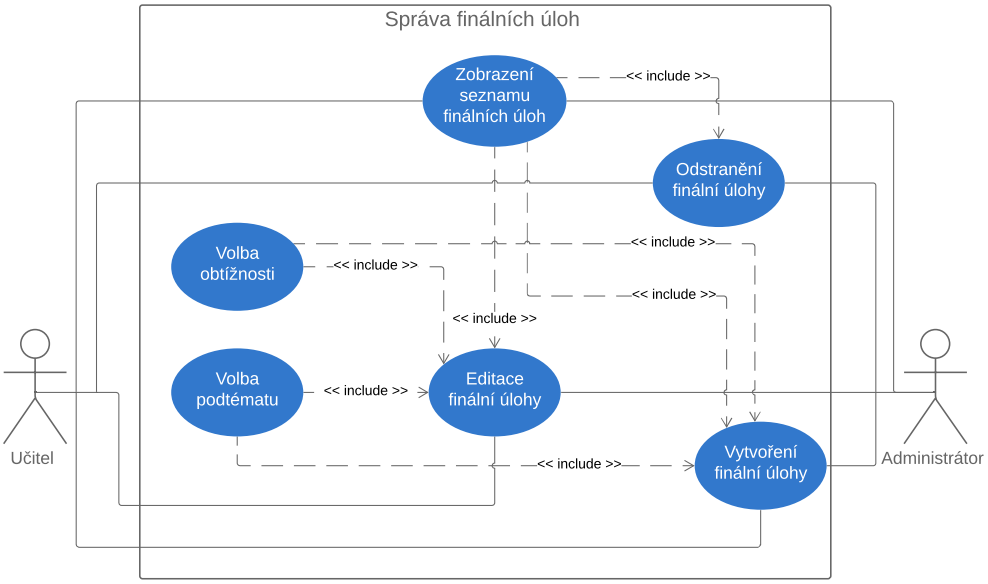
Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.2.

3.4.3 Správa finálních úloh

Tato sekce se zabývá analýzou případů užití v rámci správy **finálních úloh**. Jednotlivé případy užití jsou znázorněny na diagramu 3.3.

3.4.3.1 Zobrazení seznamu finálních úloh

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.1.



Obrázek 3.3: Diagram případů užití – správa finálních úloh

3.4.3.2 Vytvoření finální úlohy

Případ užití je založen na stejném principu jako případ užití 3.4.2.2. Jediným rozdílem je nepřítomnost přiřazení podmínek k finální úloze.

3.4.3.3 Editace finální úlohy

Případ užití je založen na stejném principu jako případ užití 3.4.2.3. Jediným rozdílem je nepřítomnost přiřazení podmínek k finální úloze.

3.4.3.4 Odstranění finální úlohy

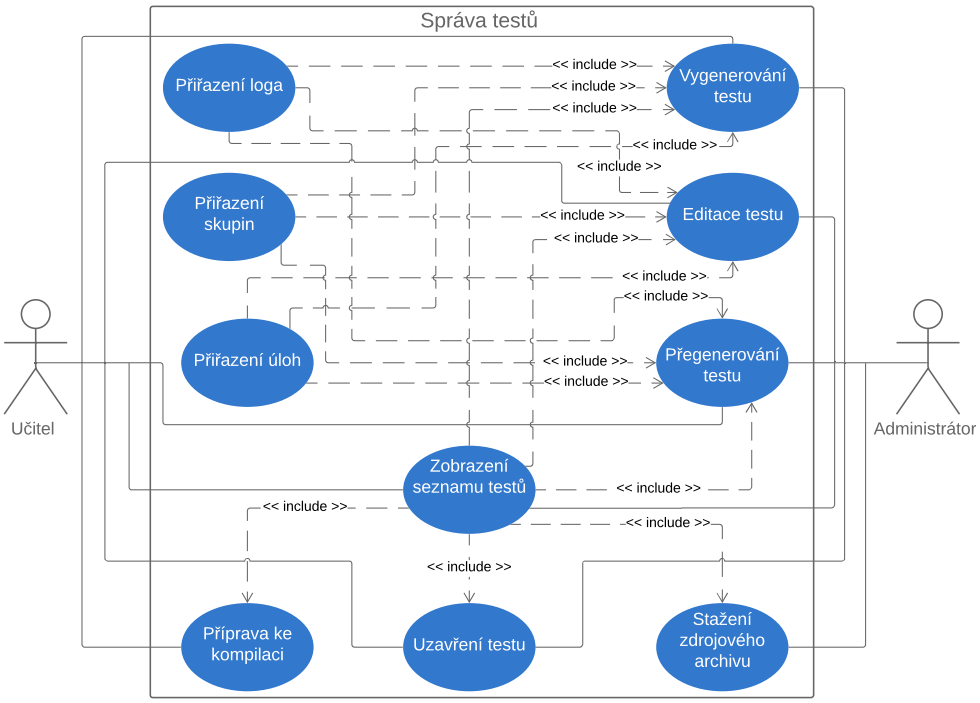
Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.2.

3.4.4 Správa testů

Tato sekce se zabývá analýzou případů užití v rámci **správy testů**. Jednotlivé případy užití jsou znázorněny na diagramu 3.4.

3.4.4.1 Zobrazení seznamu testů

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.1.



Obrázek 3.4: Diagram případů užití – správa testů

3.4.4.2 Vygenerování testu

Uživatel zvolí akci vytvoření dostupnou v hlavičce karty se seznamem, čímž dojde k přesměrování na tvorbu testu. V rámci tvorby testu uživatel zvolí počet variant testu, dále k testu přiřadí logo a uživatelské skupiny, jimž je test určen, zadá školní rok, období školního roku a číslo testu. Následně přidává jednotlivé úlohy zadání a pomocí filtrů či ručním výběrem specifikuje množiny úloh, z nichž dojde k náhodnému výběru úlohy pro jednotlivá zadání. Nakonec uživatel kliknutím na `Vytvořit` spustí generování testu. Pokud jsou všechny vstupy validní a jednotlivé varianty zadání lze vygenerovat bez duplicitních úloh, dojde k vygenerování entity testu dle zadaných parametrů.

3.4.4.3 Uzavření testu

Uživatel zvolí akci uzavření konkrétního testu ze seznamu, čímž dojde k jeho uzavření, vygenerování variant zadání ve formátu jazyka pro LaTeX a vytvoření archivu se zdrojovými soubory testu.

3.4.4.4 Editace testu

Uživatel zvolí akci editace konkrétního testu ze seznamu, čímž bude přesměrován na jeho editaci. Editace testu má dvě různé varianty v závislosti na stavu editovaného testu:

Editace údajů hlavičky je uživateli k dispozici v případě, kdy edituje neuzavřený test. V rámci editace údajů hlavičky má uživatel stejné možnosti jako v případu užití 3.4.4.2.

Zápis statistiky úspěšnosti úloh je uživateli k dispozici při editaci již uzavřeného testu.

3.4.4.5 Přegenerování testu

Volbou akce přegenerování již uzavřeného testu dojde k přesměrování na jeho přegenerování.

V rámci přegenerování testu dojde automaticky k vyplnění atributů testu a přiřazení filtrů aplikovaných na jednotlivé úlohy zadání dle původního testu. Dále má uživatel stejné možnosti jako v případě editace `neuzavřeného` testu popsané v sekci 3.4.4.4. Kliknutím na `Přegenerovat` dojde k vygenerování nové entity testu.

3.4.4.6 Stažení zdrojového archivu

Uživatel zvolí akci stažení konkrétního testu ze seznamu, čímž dojde ke stažení archivu se zdrojovými soubory testu.

3.4.4.7 Příprava ke kompilaci

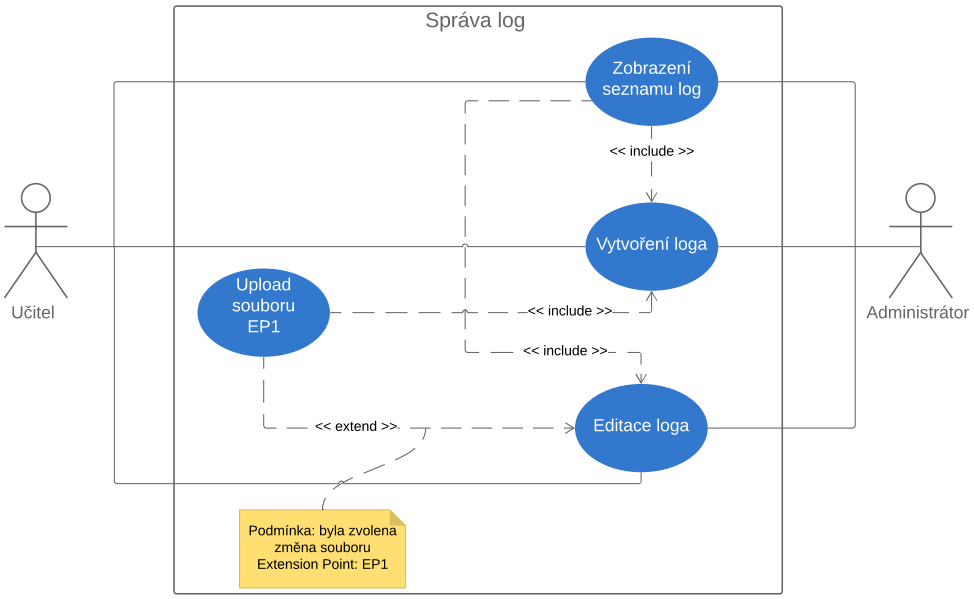
Uživatel zvolí akci PDF u konkrétního testu ze seznamu. Následně dojde k vytvoření projektu připraveného ke kompilaci variant zadání testu do PDF za využití vzdálené služby. Aplikace uživatele přesměruje do rozhraní této služby.

3.4.5 Správa log testů

Tato sekce se zabývá analýzou případů užití v rámci **správy log testů**. Jednotlivé případy užití jsou znázorněny na diagramu 3.5.

3.4.5.1 Zobrazení seznamu log

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.1.



Obrázek 3.5: Diagram případů užití – správa log testů

3.4.5.2 Vytvoření loga

Uživatel zvolí akci vytvoření dostupnou v hlavičce karty se seznamem, čímž zobrazí formulář tvorby loga v modálním okně. Uživatel provede nahrání souboru loga a zadá název loga. V případě validních vstupů dojde k vytvoření nového loga a aktualizaci seznamu. V opačném případě bude uživatel upozorněn na chyby.

3.4.5.3 Editace loga

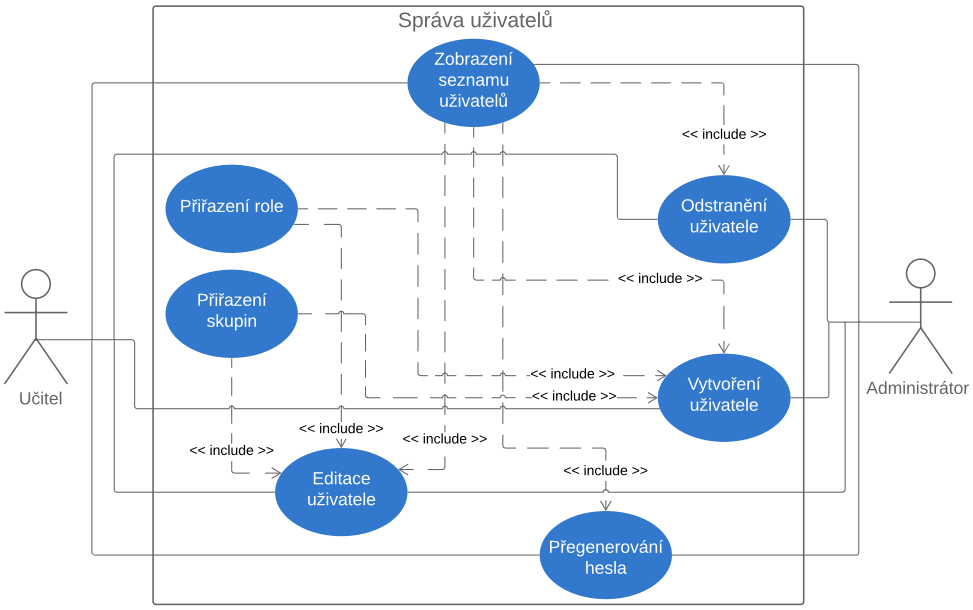
Uživatel zvolí akci editace konkrétního loga ze seznamu a bude přesměrován na jeho editaci. Dále zvolí, zda chce editovat soubor loga, či nikoli. V kladném případě musí provést nahrání souboru. Kliknutím na `Uložit` potvrdí změny. V případě chybného vstupu bude uživatel upozorněn na chyby.

3.4.5.4 Odstranění loga

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.2.

3.4.6 Správa uživatelů

Tato sekce se zabývá analýzou případů užití v rámci **správy uživatelů**. Jednotlivé případy užití jsou znázorněny na diagramu 3.6.



Obrázek 3.6: Diagram případů užití – správa uživatelů

3.4.6.1 Zobrazení seznamu uživatelů

Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.1.

3.4.6.2 Vytvoření uživatele

Uživatel zvolí akci vytvoření dostupnou v hlavičce karty se seznamem, čímž zobrazí formulář tvorby uživatele v modálním okně. Uživatel přiřadí novému uživateli roli, a uživatelské skupiny. Dále vyplní e-mailovou adresu, jméno, příjmení a nepovinně uživatelské jméno[28]. Každá z rolí může vytvářet pouze uživatele nižších rolí[29]. V případě chybného vstupu bude uživatel upozorněn na chyby. V opačném případě dojde k vytvoření uživatele, automatickému přiřazení náhodného hesla, odeslání pozvánky do aplikace na zadanou e-mailovou adresu a aktualizaci seznamu uživatelů.

[28] v případě, že není vyplněno, aplikace ho nastaví na hodnotu e-mailové adresy
[29] administrátor může vytvářet studenty a učitele, učitelé pouze studenty

3.4.6.3 Editace uživatele

Uživatel zvolí akci editace konkrétního uživatele ze seznamu a bude přesměrován na jeho editaci. Zde uživatel zvolí nové hodnoty atributů uživatele a kliknutím na `Uložit` potvrdí změny. V případě správných vstupů dojde k uložení změn. V opačném případě bude uživatel upozorněn na chyby.

3.4.6.4 Odstranění uživatele

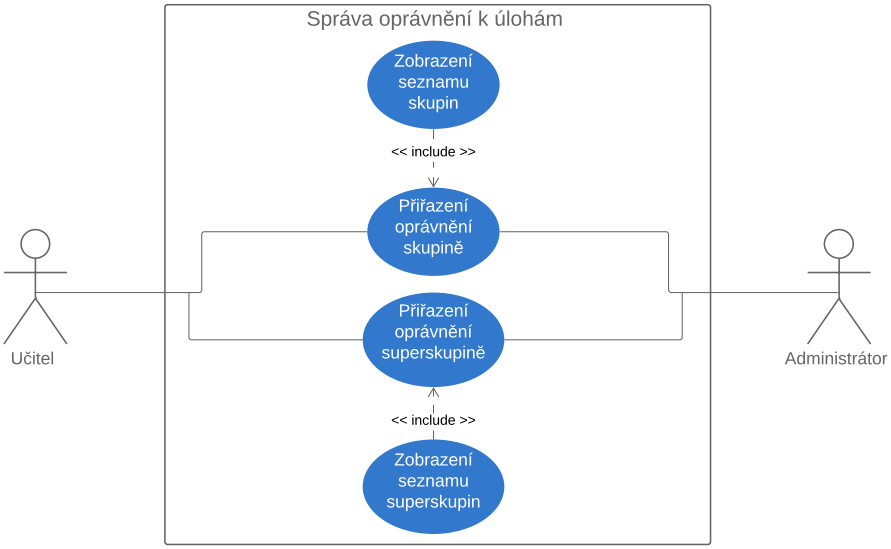
Případ užití odpovídá globálnímu případu užití ze sekce 3.4.1.2.

3.4.6.5 Přegenerování hesla uživatele

Uživatel zvolí akci přegenerovat heslo u konkrétního uživatele ze seznamu. Následně dojde ke změně hesla zvoleného uživatele na nové náhodné heslo a zaslání e-mailu s novým heslem na jeho e-mailovou adresu.

3.4.7 Správa oprávnění k úlohám

Tato sekce se zabývá analýzou případů užití v rámci **správy oprávnění k úlohám**. Jednotlivé případy užití jsou znázorněny na diagramu 3.7.



Obrázek 3.7: Diagram případů užití – správa oprávnění k úlohám

3.4.7.1 Zobrazení seznamu skupin

Uživatel v navigaci zvolí položku `Nastavení`. V nastavení zvolí `Oprávnění skupin` a dojde k přesměrování na seznam skupin, kde jsou skupiny zobrazeny

v závislosti na roli uživatele, jak je popsáno v sekci 3.3.

3.4.7.2 Přiřazení oprávnění skupině

Uživatel zvolí akci autorizace konkrétní skupiny ze seznamu, čímž dojde k přesměrování na editaci oprávnění. Zde uživatel navolí témata úloh, která mají být skupině zpřístupněna a změny potvrdí.

3.4.7.3 Zobrazení seznamu superskupin

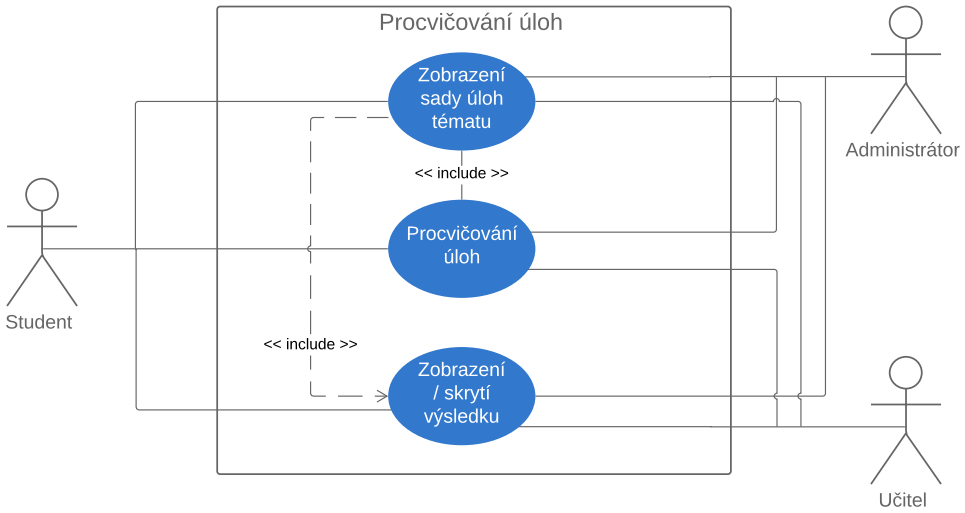
Uživatel v navigaci zvolí položku `Nastavení`. V nastavení zvolí `Oprávnění superskupin` a dojde k přesměrování na seznam superskupin, kde jsou superskupiny zobrazeny v závislosti na roli uživatele, jak je popsáno v sekci 3.3.

3.4.7.4 Přiřazení oprávnění superskupině

Případ užití je založen na stejném principu jako případ 3.4.7.2.

3.4.8 Procvičování úloh

Tato sekce se zabývá analýzou případů užití v rámci **procvičování úloh**. Jednotlivé případy užití jsou znázorněny na diagramu 3.8.



Obrázek 3.8: Diagram případů užití – procvičování úloh

3.4.8.1 Zobrazení sady úloh tématu

Studentovi budou zobrazena témata úloh, jež mu byla zpřístupněna administrátorem či učitelem. Po zvolení jednoho z tématu bude uživatel přesměrován do stránkované sady úloh daného tématu. Administrátorovi zde budou přístupná všechna témata úloh, zatímco učiteli pouze jím vytvořená témata.

V rámci sady úloh může přihlášený uživatel zobrazit filtrování rozbalením karty filtrovacího formuláře. Uživatel navolí příslušné filtry či řazení a potvrdí volbu. Po potvrzení dojde k filtrování sady úloh. Uživatel hromadně zruší filtry volbou `Resetovat filtry`.

3.4.8.2 Zobrazení / skrytí výsledku

U právě zobrazené úlohy uživatel ovládá zobrazení výsledku kliknutím na tlačítko `Zobrazit odpověď`.

3.4.8.3 Procvičování úloh

Uživatel prochází stránkovanou sadu úloh s využitím dostupných filtrů a s možností ovládání zobrazení výsledku aktuálního příkladu.

3.5 Doménový model

Tato sekce se věnuje analýze doménového modelu aplikace. Smyslem doménového modelu je získat přehled o entitách vyskytujících se v logické vrstvě aplikace a o vazbách mezi těmito entitami. Na základě provedené analýzy doménového modelu bude v sekci 5.4.1 zmapována jeho implementace.

3.5.1 Úlohy

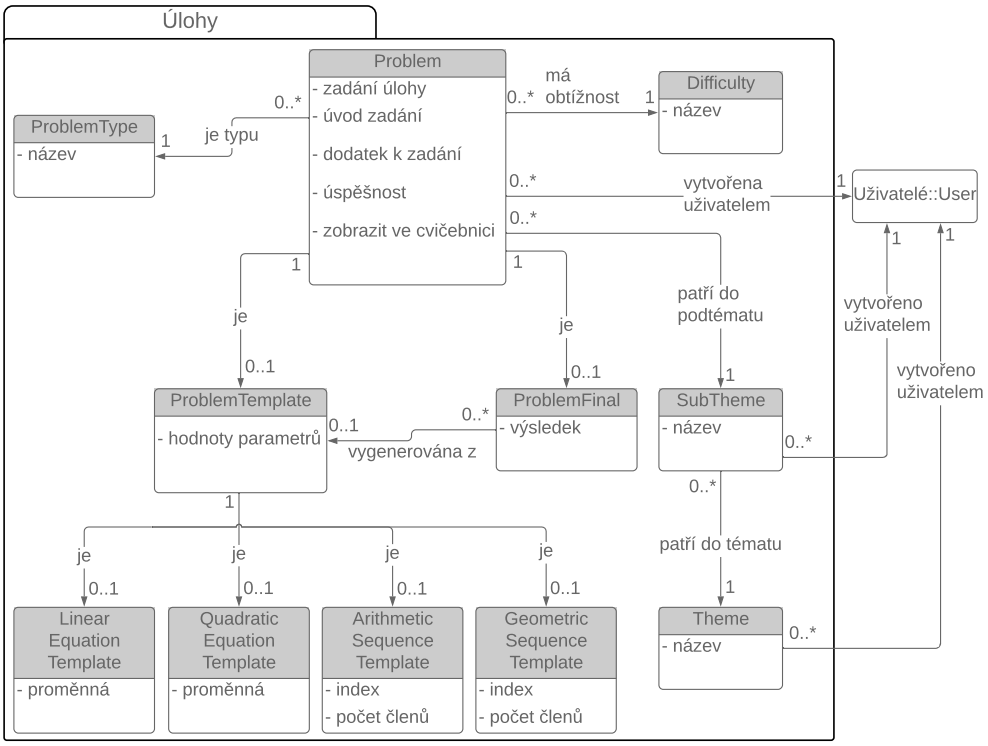
Úlohy představují v aplikaci důležitou jednotku a jsou reprezentovány entitou `Problem`.

Z funkčního požadavku 3.1.2 vyplývá třídění entity `Problem` do hierarchické struktury za využití entit `Theme` a `SubTheme`. Ze stejného funkčního požadavku vyplývá také existence entit `Difficulty` a `ProblemType`, včetně jejich vztahů s entitou `Problem`. Entita `Problem` musí být rovněž ve vztahu s uživatelskou entitou `User`. Na základě funkčních požadavků 3.1.2 a 3.1.3 existuje několik specializací entity `Problem` rozdělených do dvou větví.

První větev specializací reprezentuje entita `ProblemFinal`, jež představuje buď úlohu staticky zadanou uživatelem či úlohu vygenerovanou ze šablony úloh. V případě vygenerování ze šablony si `ProblemFinal` udržuje vztah se šablonou, z níž byla vygenerována.

Druhou větev specializací reprezentuje entita `ProblemTemplate`, která je dále specializována na konkrétní typy šablon. Rozhodnutí takto dále specializovat větev šablon plyne z nefunkčního požadavku 3.2.2, jelikož ke každé

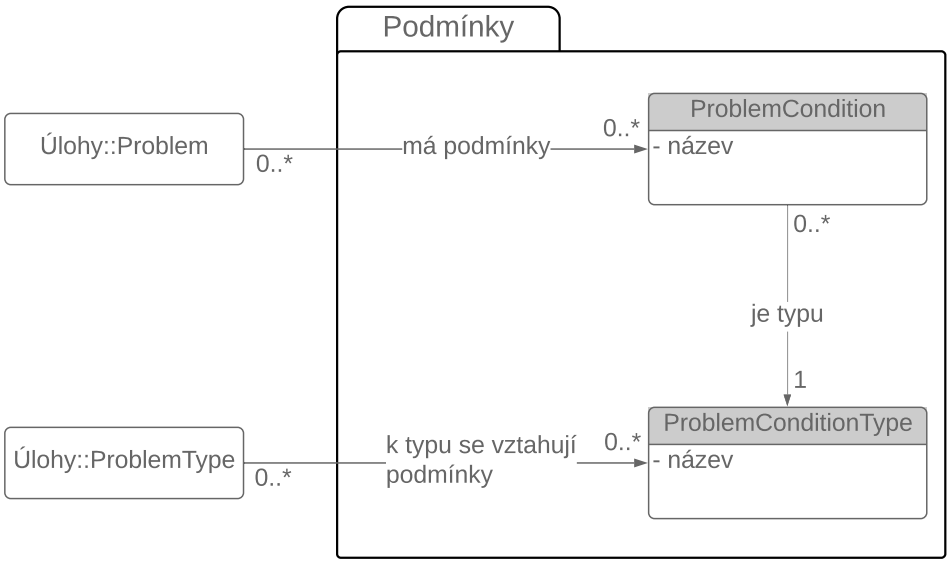
šabloně se mohou vázat specifické vlastnosti a logika zpracování. Rozdělení již na úrovni databáze umožní oddělit logiku zpracování jednotlivých typů šablon úloh.



Obrázek 3.9: Doménový model – Úlohy

3.5.2 Podmínky

Podmínky přímo souvisejí s úlohami. Entita `ProblemConditionType` vyjadřuje skupinu podmínek, jež lze přiřazovat k určitým typům úloh. Při omezení se na několik typů úloh a typů podmínek by mohl postačit návrh, který počítá pouze se situací, kdy se k jednomu typu úlohy váže jedna skupina podmínek. Za takové situace by byla postačující kardinalita 1:N mezi entitami `ProblemType` a `ProblemConditionType`. Podobná situace nastává mezi entitami `Problem` a `ProblemCondition`. S přihlédnutím k nefunkčnímu požadavku 3.2.2 však bylo navrženo obecnější řešení za využití kardinalit M:N.



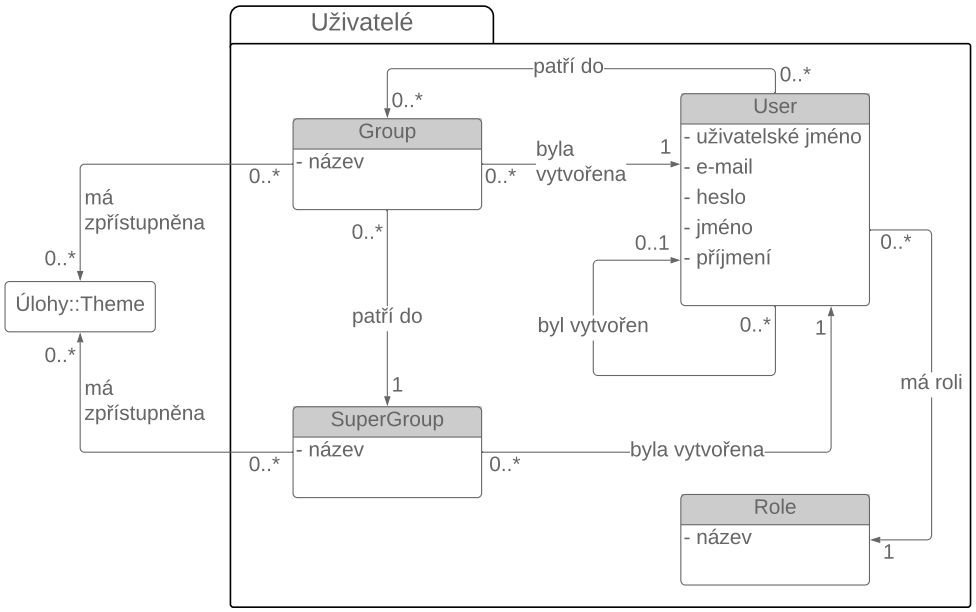
Obrázek 3.10: Doménový model – Podmínky

3.5.3 Uživatelé

Uživatelé jsou v aplikaci reprezentováni entitou `User`, již lze umístit do dvouúrovňové hierarchické struktury za využití entit `Group` a `SuperGroup`. Toto rozhodnutí vyplývá z funkčního požadavku 3.1.5. Vzhledem k tomu, že z požadavků na aplikaci nevyplývá potřeba rozlišovat jednotlivé role na úrovni uživatelské entity, byla pro reprezentaci rolí zvolena entita `Role`, která je ve vztahu s entitou `User`. Model tedy umožňuje přiřadit uživateli roli jakožto instanci entity `Role`.

Z funkčního požadavku 3.1.8 vyplývá potřeba provázanosti entit `Group` a `SuperGroup` s entitou `Theme`. Tato vazba umožní přiřazovat skupinám a superskupinám oprávnění k jednotlivým tématům úloh.

Na základě omezení pro uživatelskou roli `Učitel` představených v sekci 3.3, obsahuje doménový model další vazby zahrnující entitu `User`. Jedná se o vazby s entitami `Group` a `SuperGroup`. Kromě zmíněných dvou vazeb zahrnuje model rekurentní vazbu nad entitou `User`. Tyto tři vazby udržují informaci o autorovi entit. Díky této informaci je možné zpřístupnit uživateli správu pouze těch entit, u nichž je evidován jako autor, a realizovat tak požadovaná omezení.



Obrázek 3.11: Doménový model – Uživatelé

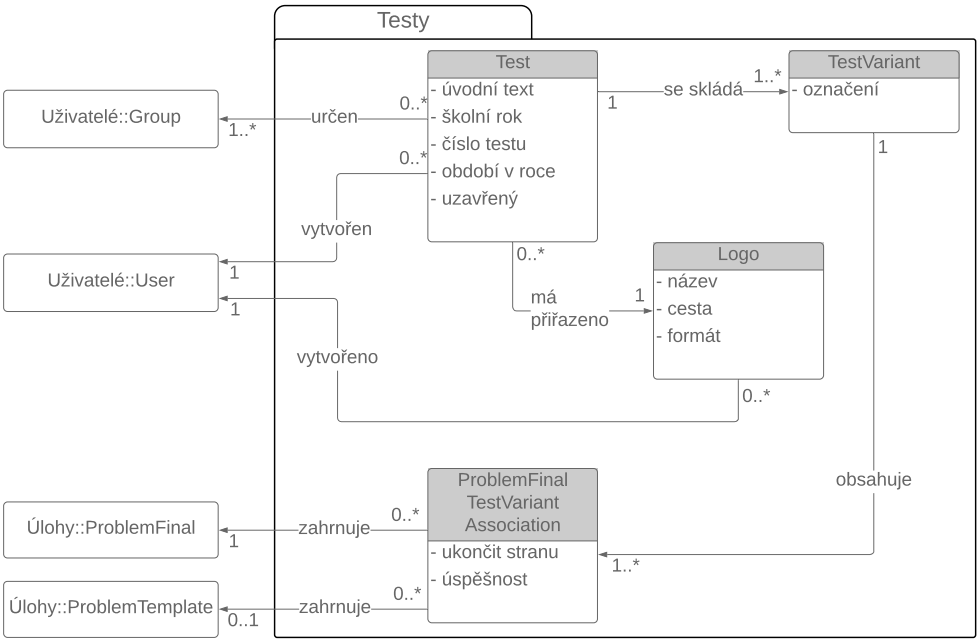
3.5.4 Testy

Návrh sekce doménového modelu, jež se věnuje testům, byl proveden s ohledy na funkční požadavek 3.1.4. Pro reprezentaci loga testu byla zvolena samostatná entita `Logo`. Tato entita je vázána na entitu `Test`. Vazba mezi entitami `Test` a `Group` umožní přiřazení více testů k více uživatelským skupinám.

Samotný test se skládá z jedné či více variant zadání reprezentovaných entitou `TestVariant`. Vlastní úlohy obsažené v zadání testu jsou ve vazbě právě s entitou `TestVariant`. V této vazbě se navíc mohou vyskytovat pouze úlohy ze specializace `ProblemFinal`.

K reprezentaci vazby mezi variantami testu a finálními úlohami zařazenými do varianty slouží vazební entita `ProblemFinalTestVariantAssociation`. Tato entita byla navržena za účelem udržování informací o úspěšnosti úloh a ukončení strany zadání, jež vyplývají z funkčních požadavků 3.1.4 a 3.1.7. Vazební entita je povinně ve vazbě s entitami `TestVariant` a `ProblemFinal`. Vazba s entitou `ProblemTemplate` je přítomna jen tehdy, pokud byla přiřazená finální úloha vygenerována ze šablony.

Pro účely omezení oprávnění uživatelské role `Učitel` v modulu `Teacher` dle požadavků popsaných v sekci 3.3 jsou entity `Test` a `Logo` provázány s entitou `User`, čímž je udržována informace o autorovi daných entit.



Obrázek 3.12: Doménový model – Testy

Kapitola 4

Návrh

Tato kapitola se věnuje návrhu aplikace, jenž poslouží jako základ pro její implementaci. V úvodu představí technologie, pomocí nichž bude aplikace implementována. Dále podrobněji přiblíží framework zvolený pro implementaci. Následně se kapitola zaměří na specifické prvky návrhu aplikace, které přesahují rámec architektury zvoleného frameworku.

4.1 Zvolené technologie

V sekci 3.2.3 předchozí kapitoly byly uvedeny technologie na straně serveru představující jeden z nefunkčních požadavků na vznikající aplikaci. Tato sekce dále upřesní volbu technologií využívaných na straně serveru. Zároveň tyto technologie doplní o technologie využívané na straně klienta.

4.1.1 PHP 7

O charakteristice jazyka PHP pojednávala sekce 3.2.3.1. Pro implementaci aplikace bylo využito PHP 7, aktuální verze tohoto jazyka.

PHP 7 prošlo oproti PHP 5 řadou změn včetně refactoringu[30] a přepracování datových struktur, což má za následek vyšší efektivitu jazyka a nižší spotřebu paměti. Zřejmě nejdiskutovanější novinkou, již PHP 7 přineslo, je možnost **typové kontroly pro skalární datové typy**. [16]

4.1.1.1 Framework

Využití frameworku pro realizaci aplikace není nutností, nicméně jeho využití s sebou přináší řadu výhod, mezi které patří například usnadnění práce programátorovi či vedení k nejlepší praxi.

[30] proces úpravy zdrojového kódu za účelem zvýšení jeho kvality bez vlivu na jeho rozhraní

Termín **softwarový framework** obecně označuje soubor knihoven, které dohromady tvoří základní prostředí pro vývoj aplikací. Framework představuje **kostru pro implementaci aplikace**. Smyslem frameworků je poskytnout řešení obecných problémů v rámci vývoje aplikací. Toho je dosaženo sdílením znovupoužitelného kódu napříč prvky aplikace. [17]

Webové aplikační frameworky zpravidla poskytují nástroje, které usnadňují řešení běžných problémů či problémů, jež se při vývoji opakují. Mezi takovéto problémy patří například:

- routování URL,
- připojení k databázi,
- rozhraní pro práci se sessions[31],
- nástroje pro autentizaci a autorizaci,
- zabezpečení aplikace proti známým útokům. [18]

Pro webové aplikační frameworky je navíc specifické využití vícevrstvé aplikační architektury. Více než 80 % těchto frameworků využívá architekturu MVC. [19]

Dle [20] patří mezi oblíbené frameworky pro jazyk PHP například `Laravel`, `Symfony`, `Zend`, `Code Igniter` či `CakePHP`. Vynechat nelze ani framework `Nette`, jenž byl zvolen pro implementaci a bude se mu podrobněji věnovat sekce 4.2.

4.1.2 Databáze

O databázové technologii MySQL jakožto o nefunkčním požadavku na aplikaci pojednávala sekce 3.2.3.2. Tato sekce se podrobněji zaměří na volbu databázových technologií, jež budou využity pro implementaci aplikace.

4.1.2.1 Doctrine

Vzhledem k doménovému modelu představenému v sekci 3.5 a požadavku na rozšiřitelnost aplikace je vhodné zvolit nad databází vyšší míru abstrakce namísto nízkoúrovňového přístupu. Z tohoto důvodu bylo nad databází nasazeno ORM za využití populární knihovny **Doctrine ORM** verze 2.

ORM poskytované Doctrine je knihovna vybudovaná na DBAL[32]. Doctrine ORM disponuje dotazovacím jazykem DQL, objektově orientovanou nadstavbou nad klasickým SQL. Díky skutečnosti, že Doctrine ORM je založeno na DBAL, je samotné ORM nezávislé na použité databázi. [21]

[31] předdefinované globální proměnné dostupné napříč aplikací

[32] abstrakční databázová knihovna, jež je součástí kolekce projektů Doctrine

Samotná DBAL podporuje například následující databáze:

- MySQL,
- Oracle,
- Microsoft SQL Server,
- PostgreSQL,
- SQLite. [22]

4.1.3 Strana klienta

Tato sekce pojednává o technologiích na straně klienta, jež byly zvoleny pro implementaci front-endu vznikající aplikace.

4.1.3.1 HTML5

Pro strukturování vygenerovaných dat přenesených na stranu klienta bude využito HTML5. Dle [23] je HTML **základní technologií pro tvorbu webových stránek**. Jedná se o značkovací jazyk sloužící pro popis jejich struktury. Samotný popis struktury tvoří HTML `značky`, neboli `elementy`, jež stránku rozdělují do menších částí, jako jsou odstavce, seznamy či tabulky.

HTML5 je nejnovější specifikací HTML, která cílí na **zesílení sémantiky** obsahu webu. Toho je dosaženo specifikací nových sémantických elementů. Zároveň se tato verze snaží z HTML odstranit prvky sloužící pro tvorbu vzhledu[33]. [24]

4.1.3.2 CSS3

Pro stylizování strukturovaného obsahu bylo zvoleno CSS3. Dle [23] je CSS jazyk pro popis vzhledu webových stránek. Toho je dosaženo vytvořením **seznamu pravidel**. Každé pravidlo je pak tvořeno `selektorem` a `seznamem vlastností`, jejichž hodnoty definují konkrétní stylizaci. Zadefinovaná pravidla umožňují mimo jiné docílit **responzivity** designu webových stránek. Jedná se o jazyk, který je nezávislý na HTML, tudíž CSS lze využít ve spojení s libovolným jazykem založeným na XML. Existence CSS umožňuje oddělit kód tvořící vzhled webu od kódu tvořícího jeho strukturu.

CSS3 je nejnovější specifikací CSS. Kromě nových vlastností přineslo například následující změny:

- návrh uživatelského rozhraní vycházející ze zobrazení na mobilních zařízeních,

[33] příkladem může být deklarace fontů a barev v rámci HTML

- modulární jádro jazyka,
- využití web fontů poskytovaných třetími stranami[34],
- vlastní renderování grafiky. [25]

4.1.3.3 Sass

Sass představuje populární CSS `preprocessor`, tedy nástroj umožňující vygenerovat CSS z vlastního jazyka, jenž zpravidla vytváří nadstavbu nad CSS a poskytuje řadu nových možností. Sass rozšiřuje klasické CSS například o proměnné, zanořená pravidla, funkce či mixiny[35]. Díky tomu umožňuje snazší udržitelnost rozsáhlých stylů či znovupoužití zadefinovaných bloků kódu. [26] [27]

4.1.3.4 jQuery

Pro potřeby vlastních dynamických funkcí a efektů front-endu aplikace byl zvolen framework jQuery. Jde o rozšířený JavaScriptový framework založený na filozofii „*write less, do more*“. Jeho účelem je **zjednodušit** používání JavaScriptu. Toho jQuery dosahuje díky nadstavbovým metodám, jež řeší časté JavaScriptové úkony. Dále zjednodušuje použití AJAXu či práci s DOM. [28]

4.1.3.5 Bootstrap

Jako základní stavební prvek uživatelského rozhraní aplikace poslouží front-endový framework Bootstrap postavený na Sass a jQuery. Je založený na mobile-first přístupu a jeho základ tvoří znovupoužitelné a modifikovatelné komponenty. [29]

4.2 Nette Framework

Nette Framework představuje PHP framework založený na architektuře MVP. Je tvořen skupinou knihoven, jež fungují nezávisle na sobě a tvoří tzv. **komponenty**. Mezi přednosti frameworku se řadí jeho výkon a bezpečnost. Součástí frameworku je také rychlý a bezpečný šablonovací systém Latte. [30]

4.2.1 Architektura MVC

„*Model-View-Controller je softwarová architektura, která vznikla z potřeby oddělit u aplikací s grafickým rozhraním kód obsluhy (Controller) od kódu aplikační logiky (Model) a od kódu zobrazujícího data (View). Tím jednak aplikaci zpřehledňuje, usnadňuje budoucí vývoj a umožňuje testování jednotlivých částí zvlášť.*“ [31]

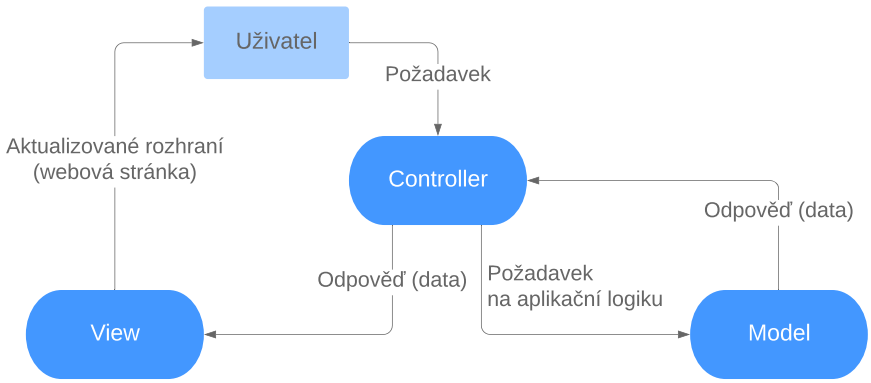
[34] např. Google Fonts
[35] definice bloku stylů, jež lze opakovaně využívat

MVC architektura je dle [32] velmi oblíbená v oblasti tvorby webových aplikací, přestože původně vznikla na desktopech. Znázornění samotné architektury je k dispozici na diagramu 4.1.

Model představuje vrstvu reprezentující datový model aplikace a její aplikační logiku. V modelu jsou definovány operace umožňující manipulaci s daty. [33]

View reprezentuje uživatelské rozhraní aplikace. Zajišťuje zobrazení dat obdržených od vrstvy `Controller`. [33]

Controller stojí mezi oběma předchozími vrstvami a slouží jako komunikační prvek (prostředník) mezi nimi. Jeho účelem je zpracování příchozích požadavků na aplikaci. [33]



Obrázek 4.1: Diagram architektury MVC

4.2.2 Architektura MVP

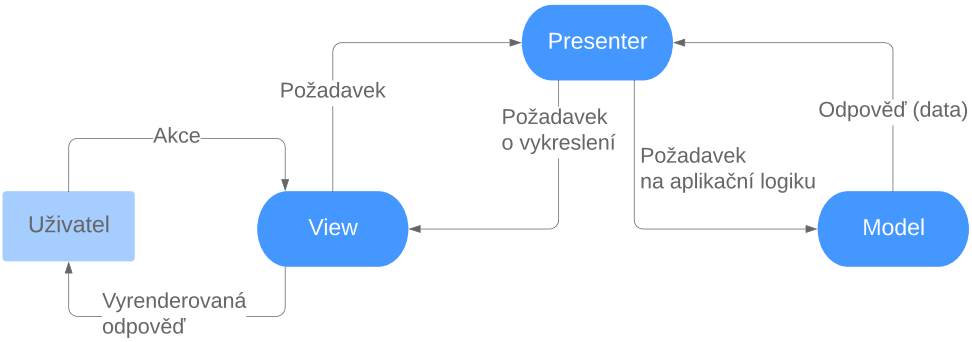
Architektura MVP vychází z architektury MVC. Architektury se vzájemně liší na úrovni vrstvy zajišťující komunikaci mezi vrstvami `View` a `Model`. V případě MVP je `Controller` nahrazen `Presenterem`. [33]

Znázornění samotné architektury je k dispozici na diagramu 4.2.

Model dle [33] plní v rámci MVC i MVP stejnou roli.

View je na základě [33] vrstvou architektury, jež slouží pro přímou interakci s uživatelem. Je tedy vstupním bodem celé architektury.

Presenter v rámci MVP kompletně odděluje vrstvy `View` a `Model`. Jeho činnost spočívá ve zpracování uživatelského vstupu z `View`, využití vrstvy `Model` pro zpracování vstupu a předání výsledku zpracování zpět do `View`. [33]



Obrázek 4.2: Diagram architektury MVP

4.2.3 Srovnání MVC a MVP

Ačkoli jsou si obě architektury v mnohém velmi blízké, lze mezi nimi najít dva základní rozdíly.

První odlišnost spočívá ve **vstupním bodu architektury**. V případě MVP zajišťuje `View` přímou interakci s uživatelem, zatímco v případě MVC zpracovává uživatelské akce `Controller`. Z toho vyplývá, že pro MVC je vstupním bodem architektury vrstva `Controller`, zatímco v případě MVP tuto roli plní vrstva `View`.

Druhý rozdíl je pak mezi **centrálními vrstvami architektury**. Zatímco v MVC plní `Controller` pouze roli prostředníka v komunikaci mezi ostatními dvěma vrstvami, v rámci MVP jsou vrstvy `View` a `Model` kompletně odděleny vrstvou `Presenter`, která na základě vstupu manipuluje s modelem a následně updatuje `View`.

4.2.4 Latte

Latte je šablonovací systém pro PHP představující jednu z komponent Nette Frameworku. Smyslem šablonovacích systémů je zastoupit přímé používání PHP uvnitř HTML za účelem jeho generování. Přestože PHP vzniklo jako šablonovací jazyk, ke kódování šablon není příliš vhodné. To je způsobeno především nepřehledností PHP šablon a nutností myslet na jejich zabezpečení. [34]

Mezi přednosti šablonovacího systému Latte patří:

- rychlost,
- bezpečnost,
- intuitivní syntaxe vycházející z PHP. [34]

Rychlosti Latte je dosaženo překladem na optimalizovaný PHP kód. Bezpečnost pak zajišťuje kontextově sensitivní escapování a kontrola odkazů. [34]

Kontextově sensitivní escapování je metoda zabezpečení před XSS[36] útoky. Je založena na detekci významu escapovaného kódu v rámci dokumentu. Na základě významu kódu pak escapuje znaky se speciálním významem metodou specifickou pro daný význam. [34]

Kontrola odkazů spočívá v ověření, zda proměnná používaná v atributech `src` či `href` obsahuje URL. [34]

4.2.5 Filozofie komponent

V úvodu sekce 4.2 bylo zmíněno, že Nette Framework je tvořený skupinou knihoven, jež jsou na sobě nezávislé a tvoří samostatné komponenty. Nette Framework se snaží vést programátora k dodržování této filozofie. Správné používání frameworku tedy vede k tvorbě znovupoužitelných komponent.

4.2.6 Adresářová struktura

Kostra Nette projektu disponuje doporučenou základní adresářovou strukturou. Tato struktura bude dále rozšiřována v závislosti na architektuře aplikace, jež bude představena v sekci 4.3. Doporučená adresářová struktura je znázorněna na obrázku 4.3.

4.2.7 Zpracování akce presenteru

Každý požadavek na aplikaci se přes soubory `index.php` a `bootstrap.php` dostane do objektu typu `Application`. Ten však nedovede HTTP požadavek zpracovat, zavolá proto router, jenž pro něj požadavek přeloží. Router aplikaci sdělí, kterému presenteru je požadavek určen a kterou akci daného presenteru vykonat. Jakmile objekt typu `Application` obdrží tuto informaci, začne požadavek zpracovávat. Požádá službu `PresenterFactory` o vyrobení objektu daného presenteru. Následně pošle presenteru požadavek na provedení akce. „*Presenter je objekt, který vezme požadavek přeložený routerem a vymyslí odpověď. Odpovědí může být HTML stránka, obrázek, XML dokument, soubor na disku, JSON, přesměrování nebo cokoliv potřebujete. Konkrétně presenter požádá model o data a ta poté předá do šablony k vykreslení. Tohle se zpravidla odehraje v metodě renderShow, kde slovo Show odpovídá názvu akce.*“ [31]

4.2.8 Životní cyklus presenteru

V předchozí sekci byla zmíněna metoda `render<View>`, jež má v rámci presenteru na starosti vykreslení požadované šablony. Nicméně presenter disponuje několika dalšími typy metod, jejichž volání je znázorněno na obrázku 4.4. Tyto

[36] metoda narušení webových stránek využitím bezpečnostních chyb ve skriptu

```
sandbox/
├── app ..........................................adresář s aplikací
│   ├── config ..................................konfigurační soubory
│   ├── forms ...................................třídy formulářů
│   ├── model ...................................modelová vrstva a její třídy
│   ├── presenters ..............................třídy presenterů
│   │   ├── HomepagePresenter.php ..... třída presenteru Homepage
│   │   └── templates ..........................adresář se šablonami
│   │       ├── @layout.latte ..... šablona společného layoutu
│   │       └── Homepage ..................šablony presenteru Homepage
│   │           └── default.latte ..... šablona akce default
│   ├── router ..................................třídy routerů
│   └── bootstrap.php ..... zaváděcí soubor aplikace
├── log ..... adresář pro logy, errory, atd.
├── temp .....dočasné soubory, cache, atd.
├── vendor ..... adresář pro knihovny
│   └── autoload.php ..... soubor pro načítání tříd
└── www ..... veřejný adresář
    ├── .htaccess .....pravidla pro mod_rewrite
    └── index.php .....spouštějící soubor aplikace
```

Obrázek 4.3: Doporučená adresářová struktura Nette [31]

metody budou nyní stručně popsány.

startup() „*Ihned po vytvoření presenteru se zavolá metoda* ***startup()****. Ta inicializuje proměnné nebo ověří uživatelská oprávnění.*“ [35]

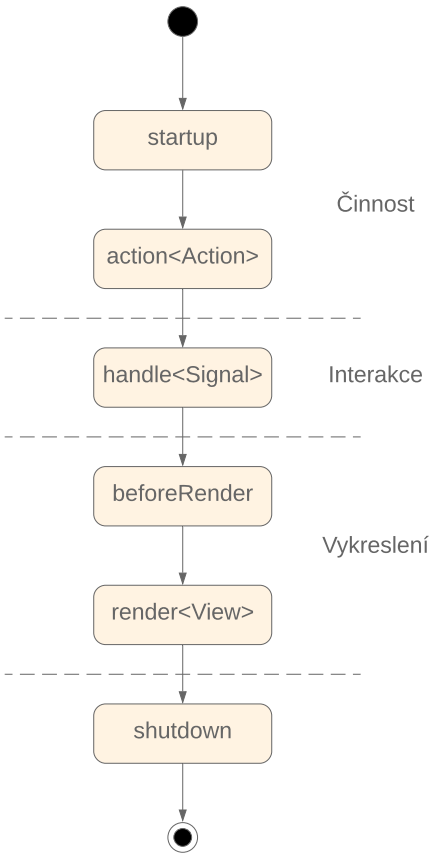
action<Action>() představuje alternativu pro `render<View>`. Využívá se v případě, kdy presenter provede operaci, po níž dojde k přesměrování. V takovém případě, kdy presenter nic nevykreslí, není vhodné volat metodu `render<View>`. [35]

handle<Signal>() „*Metoda zpracovává tzv. signály neboli subrequesty.*“ Využívá se především pro AJAXové požadavky či zpracování formulářů. [35]

beforeRender() se volá před `render<View>`. Lze využít pro nastavení šablony či předání proměnných společných pro více view. [35]

render<View>() obvykle předá data šabloně. [35]

shutdown() „*je vyvolána při ukončení životního cyklu presenteru.*“ [35]



Obrázek 4.4: Životní cyklus presenteru

4.3 Návrh aplikace

Sekce 4.2 se zabývala základní charakteristikou a filozofií Nette Frameworku. Tato sekce se přesune k vlastnímu návrhu vznikající aplikace. Ta musí být založena na modulární architektuře a navržena tak, aby byla zajištěna její budoucí rozšiřitelnost, jak vyplývá z nefunkčního požadavku 3.2.2.

Návrh aplikace bude vycházet ze sandboxu Nette projektu, jenž poskytuje rozšiřitelný základ pro tvorbu aplikace založený na architektuře MVP. MVP zajistí logické oddělení tří základních vrstev aplikace, jež byly popsány v sekci 4.2.2. Samotná vrstva `Model` zpravidla bývá složitější a vyžaduje další rozdělení. Touto vrstvou se bude podrobněji zabývat sekce 4.3.3.

4.3.1 Modulární architektura

Jak bylo zmíněno v předchozí sekci, vznikající aplikace bude založena na modulární architektuře.

Modularity lze v Nette Frameworku dosáhnout za pomoci vhodné adresářové struktury aplikace v závislosti na požadovaných modulech. Každý modul je pak umístěn ve vlastním podadresáři v adresářové struktuře projektu a je umístěn ve vlastním jmenném prostoru. Třída `RouterFactory`, jež se nachází v adresáři `router`[37], pak podporuje tvorbu objektů typu `RouteList` vázaných na konkrétní moduly.

Zmíněné pojetí modulů však trpí zásadním nedostatkem, jenž spočívá v nutnosti ručně registrovat služby všech modulů v hlavním konfiguračním souboru aplikace, což má za následek příliš zdlouhavou a složitou integraci modulů do aplikace.

Za účelem odstranění tohoto problému bude využita stejná technika, jíž využívá Nette Framework pro registraci jednotlivých komponent do svého jádra. Konkrétně se jedná o využití rozšíření `Extensions`, jež je obsaženo v základním balíčku Nette.

`Extensions` umožňují zapouzdřovat skupiny služeb do „balíčků“. Silná stránka `Extensions` pak spočívá ve skutečnosti, že při registraci rozšíření do aplikace proběhne automatické načtení všech zapouzdřených služeb do Dependency Injection kontejneru dané aplikace.

Návrh modulární architektury je znázorněn na obrázku 4.5. Vznikající aplikace bude rozdělena do následujících modulů:

Teacher Module zapouzdřující třídy v rámci administračního rozhraní určeného pro uživatele v rolích `Administrátor` a `Učitel`.

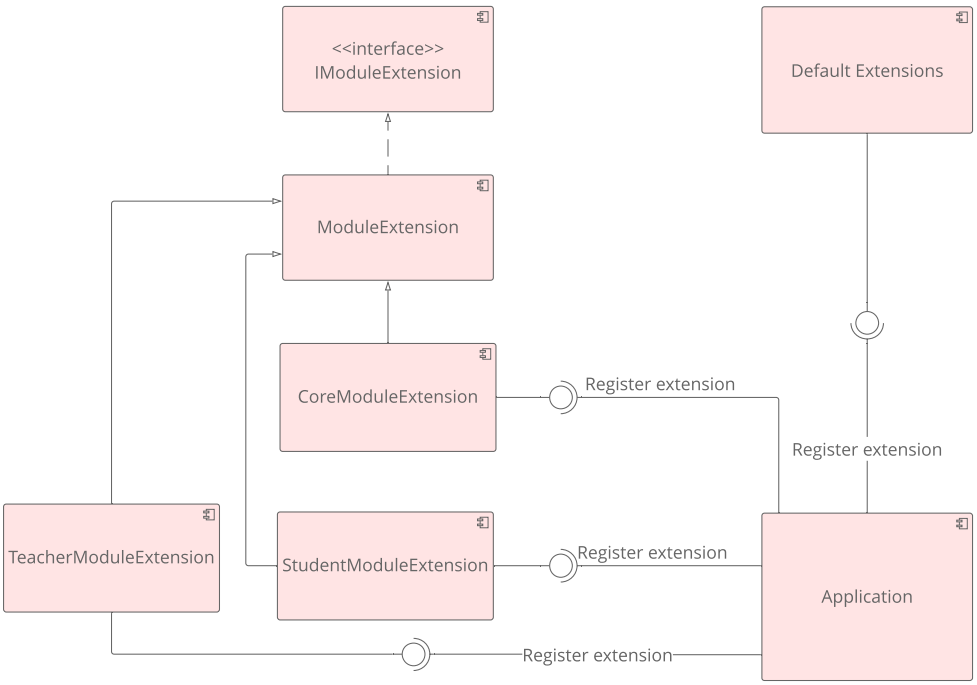
Student Module zapouzdřující třídy v rámci rozhraní určeného pro studenty, jež bude dostupné všem uživatelským rolím a uživatelům v roli `Student` tak umožní procvičování jim zpřístupněných úloh.

Core Module zapouzdřující základní třídy aplikace, jako jsou například abstraktní třídy presenterů, abstraktní třídy komponent aplikace, validátor či autorizátor. Dále tento modul zapouzdří persistentní entity využívané napříč výše zmíněnými moduly a třídy pro manipulaci s nimi.

4.3.2 Entity

Jak vyplývá ze sekce 4.1.2.1, budou pro reprezentaci dat v aplikaci využity persistentní entity. Každá entitní třída bude reprezentovat konkrétní databázovou tabulku. Instance entitních tříd pak jednotlivé záznamy daných tabulek. Entity budou vzájemně provázány za využití asociací.

[37] pro představu, viz 4.3



Obrázek 4.5: Návrh modulární architektury

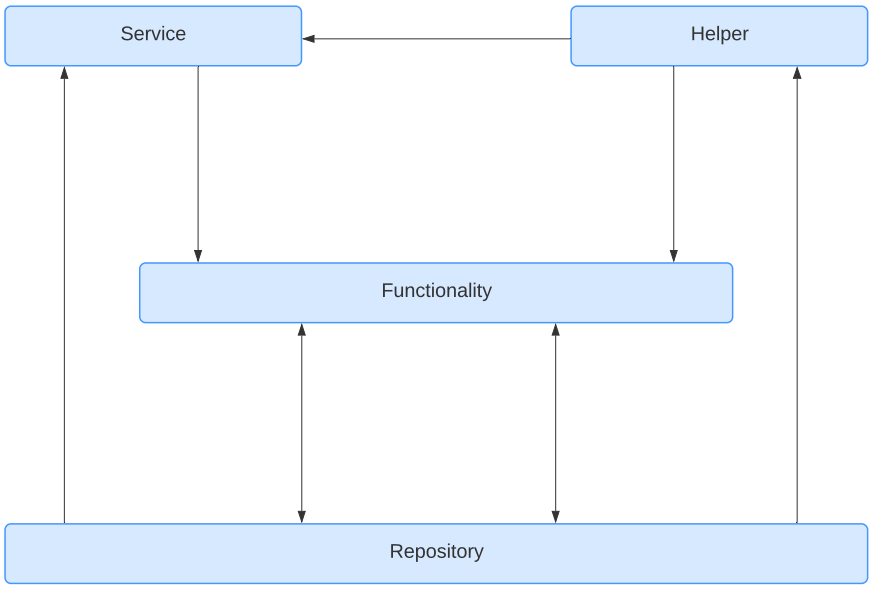
4.3.3 Podvrstvy aplikační logiky

V případě navrhované aplikace byla aplikační logika rozdělena do tří podvrstev, konkrétně `Repository` a `Functionality`, nad nimiž operuje `Service`, nejvyšší podvrstva aplikační logiky.

Vrstva `Presenter` bude mít možnost provádět operace na úrovni podvrstev `Service` a `Functionality`. Vstupními body aplikační logiky pro zápis jsou tedy podvrstvy `Service` a `Functionality`. Rozvrstvení aplikační logiky je znázorněno na obrázku 4.6.

4.3.3.1 Repository

`Repository` představují nejnižší podvrstvu aplikační logiky. V aplikaci bude pro každou entitní třídu existovat právě jedna třída typu `Repository`. Činnost tříd typu `Repository` spočívá ve čtení položek příslušné entitní třídy. Smyslem vrstvy `Repository` je tedy odstínit vyšší podvrstvy aplikační logiky od logiky zajišťující čtení dat. Jednotlivé třídy `Repository` smějí přistupovat pouze k datům vlastní entitní třídy a nekomunikují vzájemně mezi sebou.



Obrázek 4.6: Podvrstvy aplikační logiky

4.3.3.2 Functionality

Podvrstva `Functionality` poskytuje operace nad entitami, mezi jejichž příklady patří operace create, update a delete nad příslušnými entitami. Stejně jako v případě `Repository` platí, že pro každou entitní třídu bude exitovat právě jedna třída typu `Functionality`, jež poskytne operace pro manipulaci s příslušnou entitní třídou. Na rozdíl od vrstvy `Repository` však jednotlivé třídy této vrstvy mohou vzájemně komunikovat a využívat operace poskytované ostatními třídami dané vrstvy. Stejně tak může jedna třída typu `Functionality` využívat repozitáře různých entit.

4.3.3.3 Service a Helper

`Service` představuje nejvyšší podvrstvu aplikační logiky. Zajišťuje složitější procesy aplikace, mezi které patří například validace formulářů, práce se souborovým systémem, generování úloh ze šablon či generování testů. Třídy typu `Service` mohou využívat jak obou nižších podvrstev aplikační logiky, tak ostatních tříd typu `Service`. Jednotlivé třídy `Service` mohou navíc pro zvýšení abstrakce svých operací využívat pomocné třídy typu `Helper`.

4.3.4 Adresářová struktura aplikace

Adresářovou strukturu aplikace je nutné zvolit tak, aby vyhovovala potřebám navržené architektury. Tato sekce se zaměří na návrh dvojí adresářové struktury. První poslouží pro **aplikaci** jako celek, druhá pak pro **moduly aplikace**.

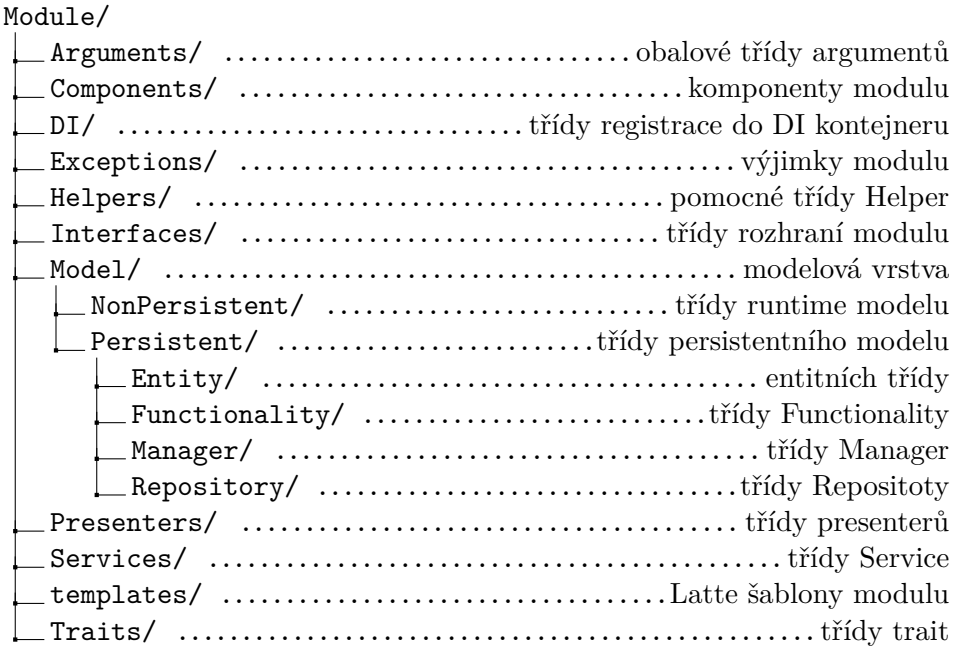
Adresářová struktura aplikace Moduly aplikace budou umístěny ve vlastních adresářích pojmenovaných dle modulu. V adresáři `resources` budou umístěny soubory využívané na front-endu aplikace[38]. Zkompilovaná verze `resources` bude umístěna v adresáři `assets`. Struktura zahrnuje rovněž adresáře `data` a `data_public`. Tyto dva adresáře poslouží jako úložiště pro soubory produkované aplikací[39]. Návrh adresářové struktury aplikace je k dispozici na obrázku 4.7.

Adresářová struktura modulů Pro všechny moduly bude adresářová struktura společná. Návrh adresářové struktury modulů aplikace je k dispozici na obrázku 4.8.

```
/
├── App ..................................... adresář aplikace
│   ├── Config/ ............................. konfigurační soubory
│   ├── CoreModule/ ......................... adresář modulu CoreModule
│   ├── TeacherModule/ ...................... adresář modulu TeacherModule
│   ├── StudentModule/ ...................... adresář modulu StudentModule
│   └── Router/ ............................. třídy routerů
├── resources/ .............................. adresář s resources (CSS, Sass, JS)
├── data/ ................................... adresář dat aplikace (např. archivy s testy)
├── log/ .................................... logy, errory atd.
├── temp/ ................................... dočasné soubory, cache atd.
├── Tests/ .................................. adresář s PHPUnit testy
├── webpack/ ................................ konfigurační soubory nástroje Webpack
└── www/ .................................... veřejný adresář document root projektu
    ├── assets/ ............................. adresář se zkompilovanými resources
    └── data_public/ ........................ adresář pro veřejná data
```

Obrázek 4.7: Adresářová struktura aplikace

[38] soubory stylů a JavaScriptu
[39] např. archivy s testy či nahranými soubory



Obrázek 4.8: Adresářová struktura modulu aplikace

Kapitola 5

Implementace

Tato kapitola se věnuje implementaci výsledné aplikace. Ve svém úvodu představí vybrané balíčky a služby, jež byly využity za účelem implementace. Následně se kapitola zaměří na popis implementace modelu za využití knihovny Doctrine ORM. Vzápětí zmapuje způsoby implementace jednotlivých sekcí aplikace vyplývajících z funkčních požadavků rozvedených v sekci 3.1. Kapitola následně přejde k realizaci autentizace a autorizace uživatelů a způsobu validování vstupních dat. Nakonec nastíní způsob implementace front-endové strany aplikace a představí technologie, jež byly během vývoje využívány pro správu `resources` aplikace.

5.1 Správa balíčků

Správu závislostí projektu implementované aplikace zajišťuje **Composer**.

Composer představuje nástroj pro správu závislostí na balíčkách v PHP. Pro účely správy závislostí umožňuje zadeklarovat balíčky, na nichž daný projekt závisí. Aby bylo možné Composer využívat, adresář projektu musí obsahovat konfigurační soubor `composer.json`. Tento soubor slouží pro specifikaci závislostí projektu, případně pro uvedení metadat projektu. [36] [37]

Jednotlivé produkční závislosti lze v rámci `composer.json` specifikovat uvnitř objektu s klíčem `require` a development závislosti v rámci objektu s klíčem `require-dev`. Pro každou uvedenou závislost lze specifikovat požadovanou verzi.

Pro základní správu závislostí Composer poskytuje následující příkazy:

composer install provede instalaci všech závislostí projektu[40]. Při prvním spuštění tohoto příkazu v rámci projektu Composer nainstaluje nejnovější povolené verze závislostí a následně vytvoří soubor `composer.lock` obsahující výčet balíčků s uzamčenými verzemi. V případě instalace

[40] uvedených jak v objektu `require`, tak `require-dev`

za přítomnosti souboru `composer.lock` instaluje Composer závislosti ve verzích uvedených v tomto souboru. [37]

composer update nainstaluje všechny závislosti projektu v nejnovějších povolených verzích a updatuje soubor `composer.lock`. [37]

Část souboru `composer.json` projektu aplikace je k dispozici v ukázce 1.

```
{
    "require": {
        "php": ">= 7.2",
        "nette/application": "^2.4",
        "nette/bootstrap": "^2.4.2",
        "nette/di": "^2.4"
    },
    "require-dev": {
        "phpunit/phpunit": "^8.1"
    }
}
```

Ukázka kódu 1: Ukázka souboru composer.json

5.2 Využívané balíčky

Tato sekce se zabývá stručným představením vybraných balíčků, jež byly využity za účelem implementace výsledné aplikace.

5.2.1 ublaboo/datagrid

`Ublaboo/datagrid`[41] představuje datagrid pro Nette Framework, jehož silnou stránkou je rozšiřitelnost a schopnost přijímat data z různých zdrojů. Jako zdroj dat může pro datagrid posloužit např. Doctrine `QueryBuilder`, kolekce Doctrine či Dibi fluent[42] dotaz. Nad daty lze konfigurovat řazení, filtrování či stránkování. Do datagridu lze dále integrovat akce jako editace či odstranění. Dokumentace je dostupná na *contributte.org/packages/contributte/datagrid*. K datagridu dále existuje sada ukázek pro doplnění dokumentace dostupná na adrese *examples.planette.io/contributte/datagrid*.

[41] dostupné z *packagist.org/packages/ublaboo/datagrid*
[42] dotaz budovaný objektově za využití zřetězeného volání metod

5.2.2 ipub/visual-paginator

`Visual-paginator`[43] umožňuje stránkování dat s podporou AJAXu. Komponentě paginátoru je možné nastavit vlastní latte šablonu. Kompletní dokumentace je dostupná na *packagist.org/packages/ipub/visual-paginator*.

5.2.3 kdyby/validator

`Kdyby/validator`[44] představuje integraci komponenty `Validator` frameworku Symfony do Nette Frameworku. `Validator` umožňuje například validovat atributy instance třídy za využití anotací. Dokumentace pro `symfony/validator` je dostupná na *symfony.com/doc/current/validation.html*.

5.2.4 Guzzle

`Guzzle`[45] představuje HTTP klienta pro PHP. Tento balíček poskytuje API pro práci s HTTP požadavky a usnadňuje tak propojit vlastní aplikaci s dalšími webovými službami. Dokumentace je dostupná na *docs.guzzlephp.org*.

5.2.5 jlawrence/eos

`jlawrence/eos`[46] je balíček pro parsování a vyhodnocování matematických výrazů. Dokumentace je dostupná z *packagist.org/packages/jlawrence/eos*.

5.3 Využívané služby

Tato sekce se zabývá stručným představením služeb, jichž aplikace pro své fungování využívá.

5.3.1 Newton API

Newton API představuje mikroslužbu pro pokročilé matematické operace. Podporuje jak vyhodnocení numerických matematických výrazů, tak úpravy a parsování symbolických výrazů.

API služby poskytuje několik endpointů[47]. Každý z těchto endpointů přijímá dotazy HTTP metody `GET`. Adresa pro volání endpointů je ve formátu `http(s)://server/operation/expression`, kde `server` představuje adresu serveru, na němž je služba dostupná, `operation` požadovanou matematickou operaci a `expression` matematický výraz ve formátu escapovaném pro URL. Přehled API endpointů, jež aplikace pro svůj běh využívá, je k dispozici v ta-

[43] dostupné z *packagist.org/packages/ipub/visual-paginator*
[44] dostupné z *packagist.org/packages/kdyby/validator*
[45] dostupné z *packagist.org/packages/guzzlehttp/guzzle*
[46] dostupné z *packagist.org/packages/jlawrence/eos*
[47] bod, jenž umožňuje komunikaci se softwarem

operace	API endpoint
`Simplify`	/simplify/<expression>
`Find 0's`	/zeroes/<expression>
`Factor`	/factor/<expression>

Tabulka 5.1: Využívané endpointy Newton API

bulce 5.1. Odpovědi služba vrací ve formátu JSON. [38]

Vzhledem k nestabilnímu hostingu služby na adrese *newton.now.sh* byla nasazena vlastní instance služby za využití cloudové platformy **Heroku**.

5.3.2 Overleaf API

Overleaf API poskytuje endpoint přijímající dotazy, jež obsahují adresu na veřejně dostupný zdrojový soubor ve formátu jazyka pro LaTeX či veřejně dostupný zip archiv se zdrojovými soubory a případnými grafickými soubory. Na základě tohoto dotazu a souboru nacházejícího se na předané veřejné adrese proběhne automatické založení projektu na službě Overleaf. [39]

5.4 Modul Core

Tato sekce se zaměří na implementaci modulu `Core` stručně popsaného v sekci 4.3.1. Nejprve bude přiblížena implementace persistentního modelu aplikace, jenž je využíván oběma zbývajícími moduly. Následně bude uveden způsob validace instancí entitních tříd před jejich persistencí. Na závěr kapitola popíše přínos modulu `Core` pro ostatní aplikační moduly.

5.4.1 Persistentní model aplikace

Tato sekce se zabývá postupem zvoleným pro implementaci persistentního modelu aplikace. Jak bylo zmíněno v sekci 4.1.2.1, nad databází je aplikováno ORM za využití Doctrine. Sekce nejprve přiblíží postup integrace Doctrine do Nette a následně přejde k samotné implementaci.

5.4.1.1 Integrace Doctrine ORM do Nette

K integraci byl využit balíček `kdyby/doctrine`, jenž lze pomocí Composeru nainstalovat příkazem `composer require kdyby/doctrine`. Rozšíření bylo dále potřeba zaregistrovat v konfiguračním souboru aplikace.

Jelikož je dle dokumentace Symfony [40] potřeba validovat data jak na vstupu po odeslání formuláře, tak před jejich persistencí do databáze, byla do Nette integrována komponenta `symfony/validator`.

5.4.1.2 Obousměrná asociace One-To-Many

Obousměrná asociace **One-To-Many** je realizována anotacemi `@ManyToOne` na straně Many a `@OneToMany` na straně One. Strana Many je v asociaci povinná a drží na databázové úrovni cizí klíč, z toho důvodu musí být v rámci této asociace vždy vlastnící stranou. Atribut `targetEntity` anotace `@ManyToOne` obsahuje entitu na straně One, zatímco atribut `inversedBy` obsahuje název asociačního atributu entity uvedené v `targetEntity`. Na straně One je asociace realizována anotací `@OneToMany`, jež má rovněž povinný atribut `targetEntity`. Kromě toho zahrnuje povinný atribut `mappedBy` obsahující název asociačního atributu entity na straně Many. [41]

V modelu aplikace je obousměrná asociace One-To-Many využita například pro asociaci mezi entitami `Theme` a `SubTheme`. Implementace je uvedena v ukázce 2. Na straně One je atributu `cascade` nastavena hodnota `all` pro kaskádové odstranění podtémat asociovaných s odstraněným tématem.

```php
<?php
class SubTheme extends BaseEntity
{
    /** @ORM\ManyToOne(
     *      targetEntity="Theme",
     *      inversedBy="subThemes",
     *      cascade={"persist", "merge"}
     * )
     */
    protected $theme;
}

class Theme extends BaseEntity
{
    /** @ORM\OneToMany(
     *      targetEntity="SubTheme",
     *      mappedBy="theme",
     *      cascade={"all"}
     * )
     */
    protected $subThemes;
}
```

Ukázka kódu 2: Obousměrná asociace One-To-Many

5.4.1.3 Obousměrná asociace Many-To-Many

Obousměrná asociace **Many-To-Many** je realizována anotací `@ManyToMany` na obou stranách. Na každé straně obsahuje anotace atribut `targetEntity`

s odkazovanou entitou. Anotace vlastnící strany obsahuje atribut `inversedBy`, zatímco anotace inverzní strany atribut `mappedBy`. Vlastnící strana navíc obsahuje anotaci `@JoinTable`, specifikující relační tabulku vazby. [41]

V modelu aplikace je obousměrná asociace Many-To-Many využita například pro realizaci vazby mezi entitami `Group` a `Theme`. Tato vazba je uvedena v ukázce 3.

```php
<?php
class Group extends BaseEntity
{
    /** @ORM\ManyToMany(
     *      targetEntity="Theme",
     *      inversedBy="groups",
     *      cascade={"persist", "merge"}
     * )
     * @ORM\JoinTable(name="group_theme_rel")
     */
    protected $categories;
}

class Theme extends BaseEntity
{
    /** @ORM\ManyToMany(
     *      targetEntity="Group",
     *      mappedBy="themes",
     *      cascade={"persist", "merge"}
     * )
     */
    protected $groups;
}
```

Ukázka kódu 3: Asociace Many-To-Many

5.4.1.4 Class Table Inheritance

Jednou ze základních vlastností objektově orientovaného programování je dědičnost, jež umožňuje vytvářet hierarchickou strukturu tříd. Relační databáze však ze své podstaty dědičnost nepodporují. Existují však strategie, jak hierarchii vytvořenou za pomoci dědičnosti rozložit do více tabulek.

Jednou z těchto strategií je **Class Table Inheritance**. *„Class Table Inheritance je strategie mapování dědičnosti, kde každá třída v hierarchii je mapována na několik tabulek: svoji vlastní tabulku a tabulky všech rodičovských tříd. Tabulka potomka je propojena s tabulkou rodiče pomocí cizího klíče. Doctrine 2 tuto strategii implementuje za využití sloupce zvaného discriminator, který se nachází v tabulce na vrcholu hierarchie, jelikož se jedná o nejjednodušší způsob pro dosažení polymorfního dotazování při využití Class Table Inheritance.“* [42] přeložil Petr Pondělík.

V Doctrine 2 se **Class Table Inheritance** realizuje za využití anotací `@InheritanceType`, `@DiscriminatorColumn` a `@DiscriminatorMap`, uváděných k třídě na vrcholu hierarchie dědičnosti. Anotace `@InheritanceType` specifikuje strategii uplatněnou pro dědičnost. Pro uplatnění Class Table Inheritance je nutné uvést hodnotu `JOINED`. Anotace `@DiscriminatorColumn` slouží pro specifikaci diskriminačního sloupce. Anotace `@DiscriminatorMap` popisuje mapování hodnot diskriminačního sloupce na entitní třídy v hierarchii. [42]

Class Table Inheritance byla využita pro realizaci specializací entity `Problem`. Samotnou entitu `Problem` přitom nemá smysl instanciovat, z toho důvodu byla definována jako abstraktní, díky čemuž nemusí být uvedena ve výčtu specializací uvnitř anotace `@DicsriminatorMap`. Použití Class Table Inheritance v rámci modelu aplikace je k náhledu v ukázce 4.

```php
<?php
 /**@ORM\Entity(repositoryClass="ProblemRepository")
 * @ORM\InheritanceType("JOINED")
 * @ORM\DiscriminatorColumn(name="discr", type="string")
 * @ORM\DiscriminatorMap({
 *      "problemfinal" = "ProblemFinal",
 *      "problemtemplate" = "ProblemTemplate",
 *      "linearеq" = "LinearEquationTemplate",
 *      "quadraticeq" = "QuadraticEquationTemplate",
 *      "arithmeticseq" = "ArithmeticSequenceTemplate",
 *      "geometicSeq" = "GeometricSequenceTemplate"
 * })
 */
abstract class Problem extends BaseEntity { ... }
```

Ukázka kódu 4: Class Table Inheritance

5.4.1.5 Validační pravidla

Validační pravidla nad entitami byla realizována formou anotací komponenty `symfony/validator` aplikovaných na atributy entitních tříd. Instance entitních tříd lze oproti anotovaným pravidlům validovat voláním metody `validate` validátoru komponenty `symfony/validator`, jíž je jako argument předána validovaná entita.

Za účelem automatizované validace entitních tříd oproti anotovaným pravidlům byl nad třídou `Kdyby\Doctrine\EntityManager` vytvořen wrapper[48] `ConstraintEntityManager`, jenž rozšiřuje metodu `persist` o validaci anotovaných pravidel. Implementace tohoto wrapperu je k dispozici v ukázce 5. Definice validačních pravidel pomocí anotací je pak v ukázce 6. Zde je vyžadována neprázdná hodnota atributu `schoolYear` zadaná v korektním formátu pro školní rok.

```php
<?php
public function persist($entity): ConstraintEntityManager
{
    $this->validateEntity($entity);
    $this->em->persist($entity);
    return $this;
}

public function validateEntity($entity): void
{
    $violations = $this->validator->validate($entity);
    if ($violations->count()) {
        throw new EntityException((string) $violations);
    }
}
```

Ukázka kódu 5: ConstraintEntityManager – persistence

5.4.1.6 Vygenerování databázového schématu

Po zadefinování entitních tříd byla využita možnost vygenerovat schéma relační databáze založené na těchto třídách. V případě frameworku Nette poskytuje pro tuto akci podporu například využitý balíček `kdyby/console`. Spuštěním příkazu[49] `php ./www/index.php orm:schema-tool:create` dojde k vygenerování schématu relační databáze.

[48] třída obsahující instanci jiné třídy, jejíž funkcionalitu modifikuje
[49] v kořenovém adresáři projektu

```php
<?php
class Test extends BaseEntity
{
    /**
     * @ORM\Column(type="string", nullable=false)
     * @Assert\NotBlank(
     *     message="SchoolYear can't be blank."
     * )
     * @Assert\Regex(
     *     value="/[0-9]{4}(\/|\-)([0-9]{4}|[0-9]{2})/",
     *     message="SchoolYear is not valid."
     * )
     */
    protected $schoolYear;
}
```

Ukázka kódu 6: Anotace – validační pravidla

5.4.1.7 Inicializace databáze

Pro účely naplnění databáze základními daty byly využity **Doctrine Migrations**. Dle [43] jsou Migrations založeny na projektech Doctrine DBAL a Doctrine ORM, pro něž poskytují funkcionalitu umožňující verzování databázového schématu. Přestože primárním účelem Migrations není manipulace s daty, lze pro ni Migrations bezproblémově využít. Jednotlivé třídy Migrations byly umístěny v adresáři `Migrations`. Každá z vytvořených migrací dědí od abstraktní třídy `EDOMPAbstractMigration`, jež rozšiřuje základní `AbstractMigration` a definuje výchozí chování metod `up` a `down` spočívající v provedení SQL skriptů načítaných ze souborů nacházejících se v adresářích `Migrations/scripts/<migrace>`. Za účelem inicializace databáze byla vytvořena migrace `InitData`, jejíž nasazení se provede příkazem `php ./www/index.php migrations:execute --up InitData`.

5.4.2 Základ aplikačního prostředí

Modul `Core` poskytuje **implementaci tříd Service a Helper**[50], u nichž se předpokládá využití v ostatních modulech aplikace. Mezi ně patří následující služby:

- autentikátor a autorizátor,
- validátor,
- služba pro práci se souborovým systémem,

[50] jmenný prostor `App\CoreModule\Services` či `App\CoreModule\Helpers`

- e-mailová služba,
- úložiště aplikačních konstant.

Dále modul `Core` poskytuje **abstraktní třídy komponent**[51] poskytující základ jejich implementace. Mezi ně patří například abstraktní třída základní komponenty aplikace, abstraktní třídy pro základní či entitní formulář či abstraktní třída komponenty datagridu.

Modul `Core` obsahuje také **implementace globálních komponent aplikace**. Patří mezi ně komponenty přihlašovacího formuláře, sidebaru, headerbaru či modální nápovědy.

5.5 Modul Teacher

Tato sekce se zaměří na implementaci modulu `Teacher` dle funkčních požadavků uvedených v sekci 3.1. V úvodu sekce bude přiblíženo využití balíčku `ublaboo/datagrid`, které bylo představeno v sekci 5.2. Následně bude popsána implementace jednotlivých sekcí modulu.

5.5.1 Ublaboo datagrid v modulu Teacher

Pro realizaci kostry modulu byl využit balíček `ublaboo/datagrid` pro Nette Framework. Tento doplněk poskytuje základ implementace pro každou sekci modulu.

Využití doplňku spočívá ve vytvoření sady komponent datagridů, jež jsou integrovány do aplikace na úrovni presenterů. Továrny datagridů jsou dostupné ve jmenném prostoru `App\TeacherModule\Components\DataGrids`. Továrny všech datagridů dědí z abstraktní třídy `BaseGrid` poskytované modulem `Core`. Továrny nastavují datový zdroj datagridů pomocí `Doctrine QueryBuilder`. Kromě nastavení zdroje dat obsahují továrny specifikaci sloupců datagridu. K jednotlivým sloupcům jsou pak nakonfigurována řazení či filtry. Náhled zdrojového kódu továrny datagridu je k dispozici v ukázce 7.

K jednotlivým datagridům jsou dále přiřazovány metody presenterů jakožto akce. Tato část konfigurace je zpravidla prováděna v příslušných presenterech po tom, co je vyrobena komponenta datagridu.

5.5.2 CRUD operace v modulu Teacher

Z důvodu konzistence byly CRUD operace všech sekcí modulu implementovány za využití stejného vzoru. Pro jednotlivé operace byla zvolena následující implementace:

[51] jmenný prostor `App\CoreModule\Components`

```php
<?php
public function create($container, $name): DataGrid
{
    ...
    $grid->setPrimaryKey("id");
    $grid->setDataSource(
        $this->problemRepository
             ->createQueryBuilder("er")
    );
    $grid->addColumnStatus(
        'difficulty', 'Obtížnost', "difficulty.id"
    )
        ->setOptions($difficultyOptions)
        ->onChange[] = [
            $container, 'handleDifficultyUpdate'
        ];
    return $grid;
}
```

Ukázka kódu 7: Metoda create továrny datagridu

Create je realizována pomocí metody `handleFormSuccess` komponenty příslušného entitního formuláře[52]. Volání této metody předchází validace pomocí metody `handleFormValidate`, jež využívá službu `Validator`[53]

Read seznamu entit poskytuje komponenta příslušného datagridu[54], jež dále umožňuje smysluplné filtrování a řazení.

Update je většinou dostupná ve dvou variantách. První variantou je rychlá editace dostupná jakožto akce datagridu. Rychlá editace je omezená pouze na jednoduché entitní atributy, u nichž není potřeba provádět složitější proces validace. Druhou variantou je klasická editace, jež je implementována metodou `handleUpdateFormSuccess` komponenty příslušného entitního formuláře. Volání této metody předchází validace pomocí metody `handleUpdateFormValidate`, jež využívá službu `Validator`.

Delete je implementována metodou `handleDelete` abstraktního presenteru `EntityPresenter`, jež volá metodu `delete` třídy `Functionality` pro příslušnou entitu. Tím je zajištěn **polymorfismus** operace.

[52] jmenný prostor `App\TeacherModule\Components\Forms`
[53] jmenný prostor `App\CoreModule\Services`
[54] jmenný prostor `App\TeacherModule\Components\DataGrids`

5.5.3 Správa uživatelů

Správa uživatelů je rozdělena do sekcí **superskupiny**, **skupiny** a **uživatelé**. Obsluhu sekcí zajišťují presentery `SuperGroupPresenter`, `GroupPresenter` a `UserPresenter`.

Správa uživatelů implementuje funkční požadavky popsané v sekci 3.1.5 následujícím způsobem:

CRUD operace dle vzoru popsaného v sekci 5.5.2.

Úrovně uživatelských skupin jsou realizovány formou skupin a superskupin.

Pozvání do aplikace je součástí operace `create`. Uživateli je zasláno pozvání na příslušnou e-mailovou adresu pomocí služby `MailService`.

Automatizovaná správa hesel je realizována vygenerováním hesla v rámci operace `create`. Datagrid `UserDatagrid` dále poskytuje akci pro přegenerování hesla uživatele, během níž je přegenerované heslo zasláno na e-mailovou adresu daného uživatele.

5.5.4 Správa matematických úloh

Správa matematických úloh je úzce spjata s generováním úloh popsaném v sekci 5.5.5. Správa matematických úloh je rozdělena do sekcí **témata**, **podtémata**, **šablony úloh** a **finální úlohy**.

5.5.4.1 Témata a podtémata

Témata spolu s podtématy slouží k hierarchickému strukturování úloh dle funkčního požadavku 3.1.2.

Obsluhu zajišťují presentery `ThemePresenter` a `SubThemePresenter`. Hierarchie úloh je založena na stejném principu jako v případě uživatelů. Nad tématy a podtématy jsou implementovány CRUD operace dle vzoru uvedeného v sekci 5.5.2.

5.5.4.2 Šablony úloh

Na základě šablon úloh je umožněno generování finálních úloh a testů, o němž pojednávají sekce 5.5.5 a 5.5.6.1.

Z důvodu obsáhlosti látky středoškolské matematiky podporuje aplikace pouze vybrané typy šablon úloh, jejichž přehled spolu se vstupním formátem je k dispozici v tabulce 5.2.

Jak bylo uvedeno v sekci 3.5.1, pro každý typ šablony existuje na úrovni modelu vlastní entita jakožto potomek entity `ProblemTemplate`. Tohoto rozdělení se drží rovněž uživatelské rozhraní modulu, díky čemuž vznikla v rámci

typ šablony	vstupní formát
lineární rovnice	`<výraz> = <výraz>`
kvadratická rovnice	`<výraz> = <výraz>`
aritmetická posloupnost (tvar: n-tý člen)	`<název>_<index> = <výraz>`
geometrická posloupnost (tvar: n-tý člen)	`<název>_<index> = <výraz>`

Tabulka 5.2: Podporované typy šablon

modulu samostatná podsekce pro každý typ šablony. Obsluhu těchto podsekcí zajišťují potomci abstraktního presenteru `ProblemTemplatePresenter`.

K takovému rozdělení uživatelského rozhraní vedly následující důvody:

Odlišné požadavky typů šablon na formuláře by vedly k rostoucí složitosti dynamicky tvořených univerzálních formulářů.

Nemožnost oddělit logiku zpracování formulářů jednotlivých typů šablon. Kromě předchozího problému by v aplikační logice bylo nutné ručně detekovat zvolený typ šablony a na jeho základě provádět odpovídající procesy.

Rostoucí složitost front-endu aplikace by byla důsledkem obou předchozích problémů.

Obtížná rozšiřitelnost front-endu aplikace by se neshodovala s nefunkčním požadavkem 3.2.2.

Správa šablon úloh na základě funkčních požadavků 3.1.2 a 3.1.3 implementuje CRUD operace následujícím způsobem:

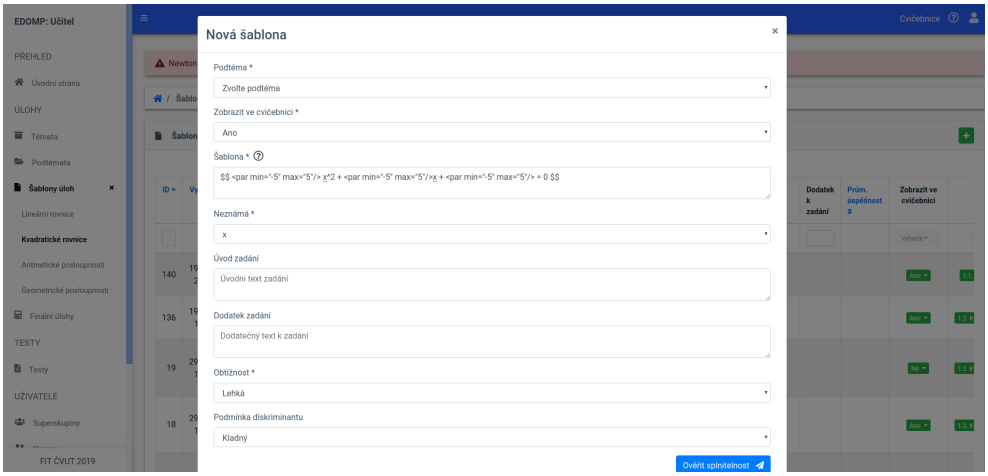
Create se návrhově drží vzoru ze sekce 5.5.2. Umožňuje zadávat šablony ve formátu popsaném v sekci 3.1.3. Na pozadí tvorby šablon jsou aplikací vykonávány komplexnější procesy. Aplikace například validuje, zda byl zadán povolený vstup v jazyce pro LaTeX či zadaná šablona odpovídá tvořenému typu šablony[55]. K některým typům šablon je umožněno přiřazovat podmínky. V případě volby některé z podmínek musí uživatel ověřit, zda je podmínka splnitelná pro zadaný vstup. Procesy vykonávané aplikací během detekce splnitelnosti podmínky souvisejí s generováním úloh a budou přiblíženy v sekci 5.5.5. Ukázku tvorby šablony lze vidět na obrázku 5.1.

Read odpovídá vzoru ze sekce 5.5.2.

Update je založen na stejném principu jako create.

[55] např. zda ze zadané šablony lineárních rovnic lze skutečně vygenerovat lineární rovnici

Delete odpovídá vzoru ze sekce 5.5.2.



Obrázek 5.1: Tvorba šablony úloh

5.5.4.3 Správa finálních úloh

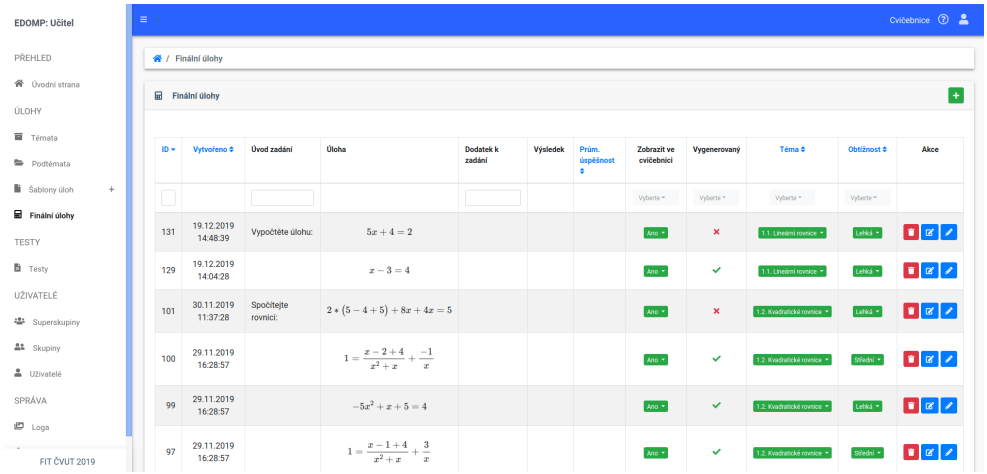
Správa finálních úloh představuje odlehčenou variantu správy šablon úloh, jež byla přiblížena v předchozí sekci. Ukázku zobrazení seznamu finálních úloh lze vidět na obrázku 5.2.

Hlavními rozdíly oproti správě šablon úloh jsou:

- jednotná správa bez rozlišování typu úloh,
- není přítomné přiřazovaní podmínek k úlohám,
- aplikace nevykonává komplexnější validaci vstupu.

5.5.5 Generování úloh

Generování finálních úloh je založeno na vytvořených šablonách úloh. V této sekci bude nejprve představen jednoduchý systém pro zapouzdření logiky související s konkrétními typy úloh, následně bude krok po kroku popsán proces generování úloh. Na závěr bude zdůvodněno, proč je podstatná část procesu generování úloh prováděna již během akcí **create** a **update** nad šablonami úloh.



Obrázek 5.2: Seznam finálních úloh

5.5.5.1 Pluginy typů úloh

V budoucnosti se očekává rozšiřování modulu `Teacher` o podporu generování nových typů úloh. S ohledem na budoucí rozšiřitelnost aplikace byl v rámci modulu implementován jednoduchý systém pluginů zapouzdřujících logiku vztahující se ke konkrétnímu typu úlohy. Na diagramu 5.3 je zachycena implementace systému pluginů.

5.5.5.2 Preprocessing šablon úloh

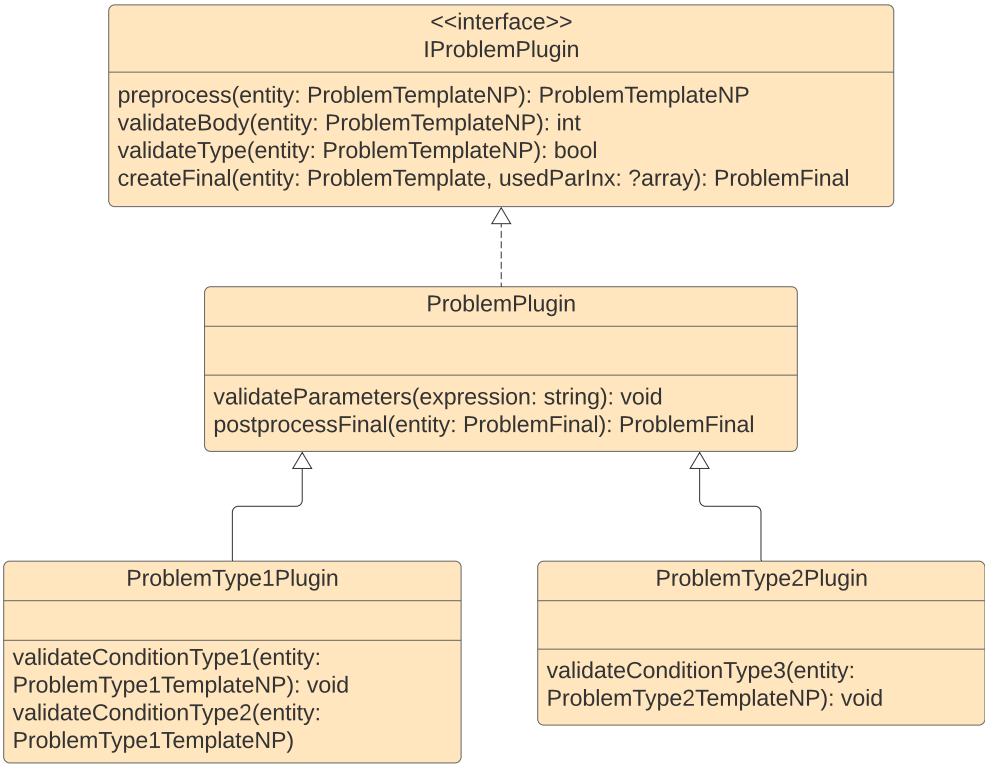
Preprocessing[56] šablon je součástí metody `handleFormValidate` abstraktní formulářové komponenty `ProblemTemplateFormControl`, z níž dědí formulářové komponenty všech typů šablon úloh. Komponentě jsou prostřednictvím jednoduchého kontejneru `PluginContainer` předány pluginy všech podporovaných typů úloh. V rámci `handleFormValidate` je pak volána metoda `preprocess` z pluginu odpovídajícího typu úlohy.

Preprocessing se skládá ze tří následujících kroků:

Převod vstupu v jazyce pro LaTeX na matematický výraz, jenž realizuje služba `LatexParser`.

Parametrizace výrazu spočívá v nahrazení nepárových XML tagů reprezentujících parametry šablony za neznámé[57]. Parametrizaci implementuje služba `ParameterParser`.

[56] proces transformace dat pro účely dalšího zpracování
[57] ve formátu pi, kde i je nezáporný identifikátor parametru



Obrázek 5.3: Systém pluginů úloh

Úpravy šablony úloh závislé na typu šablony, jejichž součástí jsou úpravy matematických výrazů, pro něž aplikace zpravidla využívá službu `Newton`. Výsledkem tohoto kroku je nepersistentní entita šablony úloh zahrnující vhodný parametrizovaný výraz standardizovaný pomocí služby `Newton`.

Pro komunikaci se službou `Newton` byla vytvořena třída `NewtonApiClient` jakožto wrapper nad klientem `Guzzle`. Ukázka 8 zachycuje část metody zajišťující dotazování na API point `/symplify` uvedený rovněž v tabulce 5.1. Hodnota proměnné `$newtonApiHost` je třídě předána z konfigurace prostředí[58].

Jednodušší úpravy matematických výrazů provádí aplikace interně. Jedná se například o převod rovnice na levou stranu či získání výrazu pro neznámou v lineární rovnici. Tyto úpravy prováděné aplikací počítají s výrazy upravenými do jednotného formátu, v němž vrací výsledky služba `Newton`.

[58] během vývoje bylo rozlišováno prostředí `local` pro vývoj a `prod` pro nasazenou aplikaci

```php
<?php
public function simplify(string $expression)
{
    try {
        $res = $this->client->request('GET',
            $this->newtonApiHost . self::SIMPLIFY . $expression);
    } catch (RequestException $e) { ... }
    $res = Json::decode($res->getBody())->result;
    ...
    return $res;
}
```

Ukázka kódu 8: Integrace endpointu simplify

5.5.5.3 Validace vstupní šablony

Po dokončení preprocessingu šablony úlohy je přistoupeno k její validaci, již za využití služby `ConditionService` zajišťuje metoda `validateType` pluginu příslušného typu úlohy. Validace šablony je službou `ConditionService` realizována postupným dosazováním kombinací hodnot parametrů do zkoumaného výrazu. Zkoumaný výraz zde představuje výstup z preprocessingu šablony popsaného v předchozí sekci.

Na výraz vzniklý dosazením kombinace hodnot parametrů jsou během validace kladeny podmínky na **matematickou validitu**[59] a **typovou validitu**[60]. Služba `ConditionService` je schopna validovat typ šablony díky skutečnosti, že k jednotlivým typům úloh v aplikaci existující podmínky reprezentující právě typovou validitu šablony úloh daného typu.

`ConditionService` sestavuje pole n-tic parametrů, jež při dosazení do výrazu s n parametry vyhověly dané podmínce. Pokud je po dokončení validace zmíněné pole **neprázdné**, je zakódováno do formátu JSON a dočasně persistováno entitou `TemplateJsonData` spolu s hodnotou sekvence databázové tabulky entity, která tvoří vrchol hierarchie úloh. V případě **prázdného** pole n-tic parametrů je vyhozena výjimka `ProblemTemplateException` a vstup je prohlášen za nevalidní.

5.5.5.4 Ověření splnitelnosti zadaných podmínek

V případě přiřazení podmínek k vstupní šabloně následuje po validaci šablony ověření jejich splnitelnosti. Ověření probíhá stejně jako v případě validace typu šablony popsané předchozí sekcí. Po tomto kroku jsou k výstupu ze sekce 5.5.5.2 připojeny entity typu `TemplateJsonData` zapouzdřující pole vyhovujících kombinací parametrů pro každou přiřazenou podmínku zvlášť.

[59] např. zda vzniklý výraz neobsahuje dělení nulou
[60] zda dosazením vznikla úloha požadovaného typu

5.5.5.5 Získání výsledných n-tic parametrů

Výstupem z předchozího kroku byla nepersistentní entita šablony úloh spolu se sadou dočasných entit `TemplateJsonData` zapouzdřujících n-tice parametrů vyhovujících konkrétním podmínkám šablony.

V rámci akcí **create** či **update** nad šablonami úloh aplikace získá pole n-tic parametrů, jež vyhovují všem podmínkám navázaným na danou šablonu. Takovéto n-tice parametrů aplikace získá jednoduše průnikem nad dočasně persistovanými n-ticemi.

5.5.5.6 Vygenerování finální úlohy ze šablony úloh

Generování finálních úloh je součástí generování testů blíže v sekci 5.5.6.1. Vygenerování finální úlohy je implementováno službou `ProblemGenerator`. Implementace generování spočívá v náhodné volbě jedné z vyhovujících n-tic parametrů a nahrazení jednotlivých parametrů za hodnoty dané n-tice.

5.5.5.7 Odůvodnění zvoleného řešení

Přístup spočívající v přípravě vyhovujících n-tic hodnot parametrů ještě před samotným procesem generování byl zvolen z výkonnostních důvodů. Problém hledání hodnot parametrů vyhovujících podmínkám stanoveným nad šablonou dosahuje složitosti odpovídající produktu intervalů hodnot jednotlivých parametrů. Navíc je před přistoupením k samotnému hledání vyhovujících hodnot potřeba provést několik dotazů na API služby `Newton` v rámci zpracování vstupu. Jedná se tedy o časově drahou operaci.

5.5.6 Správa a generování testů

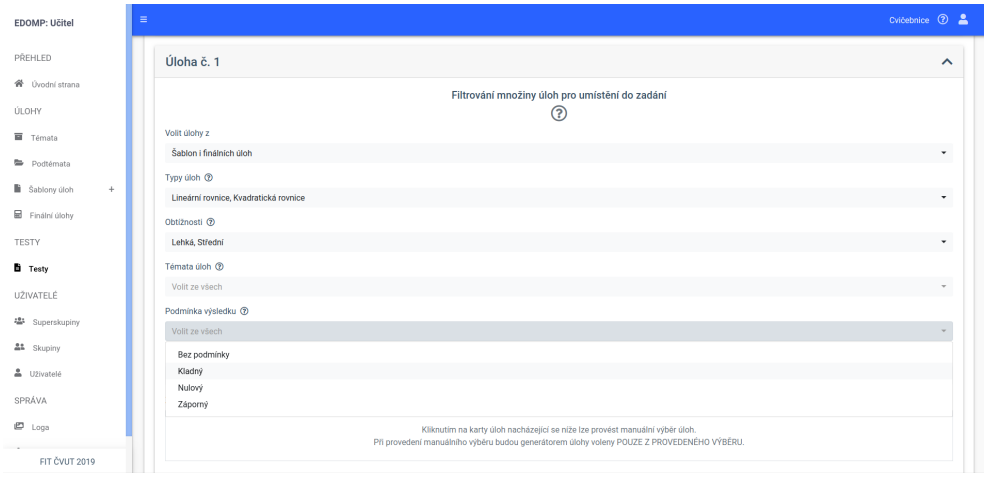
Správa testů poskytuje uživateli operace **read** a **delete** dle vzoru ze sekce 5.5.2. Kromě těchto dvou operací však poskytuje několik dalších akcí, jež tato sekce postupně zmapuje.

5.5.6.1 Generování testů

Tato funkcionalita vyplývá z funkčního požadavku uvedeného v sekci 3.1.4. Generování testu je spuštěno po validním odeslání formuláře sloužícího pro vytvoření testu, jež zohledňuje veškeré požadavky uvedené v sekci 3.1.4. Náhled části rozhraní pro tvorbu testů zaměřující se na přiřazení úloh do zadání je dostupný v ukázce 5.4.

Proces vygenerování testu zajišťuje služba `TestGenerator` a je rozdělen do následujících kroků:

Vytvoření základu entity obsahujícího atributy vycházející ze seznamu uvedeného v rámci funkčního požadavku 3.1.4.2. Výstupem je entita testu ve stavu **otevřený**, jež postrádá asociace na jednotlivé varianty zadání.



Obrázek 5.4: Tvorba testů – přiřazení úloh do zadání

Sestavení variant zadání testu zahrnující přiřazení navolených úloh k variantám zadání. V případě, že byla generátorem zvolena finální úloha, proběhne rovnou její zařazení do varianty testu. Jedná-li se o šablonu úlohy, je zavolána metoda `createFinal` pluginu příslušného typu úlohy, která za využití služby `ProblemGenerator` vygeneruje finální úlohu. Samotné vygenerování úlohy je jednoduché díky procesu popsanému v sekci 5.5.5. Služba `ProblemGenerator` náhodně zvolí jednu z vyhovujících n-tic parametrů persistovaných spolu s šablonou úlohy. V rámci přiřazení úlohy do první varianty testu jsou **persistovány filtry navolené uživatelem** pro danou úlohu za účelem využití během přegenerování testu.

Vygenerování zdrojových souborů jednotlivých variant zadání testu ve formátu jazyka pro LaTeX. Pro vygenerování zdrojových souborů využívá aplikace **uživatelskou či defaultní šablonu**[61] **latte**. Služba `TestGenerator` zajišťuje načtení šablony ze souboru a předání dat jednotlivých variant zadání do šablony. Na závěr je službou `FileService` vytvořen archiv daného testu obsahující logo a zdrojové soubory jeho variant ve formátu jazyka pro LaTeX. Tento krok se provede až po **uzavření testu**.

[61] nachází se v adresáři `App\TeacherModule\templates`

5.5.6.2 Uzamčení zdrojových souborů testu

Zdrojové soubory testu jsou uzamčeny v rámci akce uzavření testu. Test je označen jako **uzavřený** a dále ho není možné editovat[62]. Na základě entity uzavřeného testu dojde k vygenerování archivu zdrojových souborů testu.

5.5.6.3 Přegenerování testu

Implementace akce **přegenerování testu** se pohybuje na pomezí akcí **create** a **update**. Formulář komponenty `TestFormControl` je v rámci akce předvyplněn údaji originálního testu. K jednotlivým úlohám testu je k dispozici přehled filtrů navolených během generování originálního testu a možnost označit úlohu k přegenerování. Realizace procesu přegenerování je téměř totožná s vygenerováním testu ze sekce 5.5.6.1. Jediný rozdíl spočívá v předání originálního testu službě `TestGenerator`, na jehož základě je generován nový test. V případě úloh, jež nebyly označeny k přegenerování, dojde rovnou k asociování této úlohy s novým testem.

5.5.6.4 Stažení archivu testu

Správa testů poskytuje pro každý test akci **stažení archivu**. Samotné stažení je implementováno službou `TestDownloader`, jež na uživatelem vyvolaný požadavek zašle odpověď `FileResponse` s typem `application/zip`.

5.5.6.5 Automatizovaná příprava ke kompilaci

Modul poskytuje akci, jež automatizovaně vytváří projekt připravený ke kompilaci variant zadání testu do PDF. Akce využívá API webové služby `Overleaf`. Dle dokumentace [39] byla zvolena varianta dotazu s URL adresou veřejně dostupného zip archivu. `Overleaf` již zajistí rozbalení příslušného archivu, díky čemuž umožňuje připravit ke kompilaci projekt **včetně přiložených grafických souborů**. Pro využití této akce je nutné, aby uživatel **vlastnil účet** v aplikaci `Overfleaf`.

5.5.6.6 Update testu

Operace **update** se drží vzoru ze sekce 5.5.2, nicméně jak vyplývá z analýzy případů užití v sekci 3.4.4.4, update testu funguje ve dvou režimech. Údaj o režimu operace je předán službě `TestFunctionality`, jež na jeho základě provede příslušný update.

[62] umožněno je pouze zapisovat statistiku úspěšnosti úloh

5.5.7 Sledování statistiky úspěšnosti úloh

Na základě požadavku 3.1.7 aplikace implementuje **sledování statistiky úspěšnosti úloh**. Statistika je vedena na základě údajů, jež má uživatel možnost zadávat v rámci updatu uzavřených testů. Aplikace na základě asociací mezi testy a úlohami sleduje průměrnou statistiku úspěšnosti jak pro finální úlohy, tak pro šablony úloh.

5.5.8 Správa oprávnění k úlohám

Správa oprávnění skupin a superskupin je k dispozici v rámci nastavení. Pro skupiny a superskupiny je správa oddělena a je implementována formou editačních formulářů, jejichž odeslání způsobí aktualizaci témat úloh asociovaných s editovanou entitou. V případě updatu oprávnění superskupiny provede aplikace update oprávnění všech skupin náležících dané superskupině.

5.6 Modul Student

Tato sekce se zaměří na implementaci modulu `Student` dle funkčních požadavků ze sekce 3.1. Nejprve bude popsáno využití rozšíření `visual-paginator`, jež bylo představeno v sekci 5.2. Následně bude přiblížena implementace sbírky určené studentům k procvičování.

5.6.1 Visual-paginator v modulu Student

Doplněk byl využit pro implementaci stránkování úloh ve cvičebnici. Využití doplňku spočívá ve vytvoření komponenty paginátoru, přiřazení vlastní latte šablony a nastavení akce volané při přechodu mezi stránkami formou callbacku, jenž zajišťuje překreslování při AJAXovém stránkování. Paginátoru je předán počet finálních úloh v sadě určené ke stránkování a počet úloh pro zobrazení na jedné stránce. Za účelem procvičování je realizováno stránkování po jedné úloze. Na výběr úloh z repozitáře je aplikován `offset` a `limit`. Náhled použití paginátoru je k dispozici v ukázce 9.

5.6.2 Procvičování pro studenty

Cvičebnice dostupná v modulu `Student` je rozdělena na sady úloh dle témat. Nad každou sadou úloh je aplikováno stránkování za využití doplňku `visual-paginator`, jež bylo popsáno v předchozí sekci.

Nad každou sadou úloh lze aplikovat filtry. Cvičebnice podporuje filtrování dle obtížnosti, tématu či přítomnosti řešení úlohy. Umožňuje uživateli také řazení úloh dle obtížnosti. Filtrování je implementováno formou persistentního atributu presenteru `ThemePresenter`. Tato implementace způsobuje, že zvolené filtry zůstanou aplikovány při vyvolání přesměrování v rámci sady úloh. Jak je vidět v ukázce 9, filtry jsou předávány metodám repositáře úloh

```php
<?php
$problemsCnt = $this->repository->getFilteredCnt(
    $id, $this->filters
);

$paginator = $visualPaginator->getPaginator();
$paginator->itemsPerPage = 1;
$paginator->itemCount = $problemsCnt;

$problems = $this->repository->getFiltered(
    $id, $paginator->itemsPerPage,
    $paginator->offset, $this->filters
);
```

Ukázka kódu 9: Použití komponenty visual-paginator

sloužícím pro selekci příkladů z repositáře a získání počtu selektovaných úloh. Ukázka cvičebnice s otevřenou kartou filtrů je k dispozici na obrázku 5.5.

Obrázek 5.5: Cvičebnice

5.7 Autentizace a autorizace

Tato sekce pojednává o metodách, jimiž jsou v aplikaci reprezentováni uživatelé a uživatelské role. Dále se zabývá procesy autentizace a autorizace během přihlášení do aplikace, přístupu k sadám úloh v modulu `Student` a nakonec během přístupu k entitám v modulu `Teacher`.

5.7.1 Reprezentace uživatelů a rolí

Na základě analýzy provedené v sekci 3.5.3 bylo přistoupeno k reprezentaci uživatelů a uživatelských rolí ve formě entit `User` a `Role`. Mezi těmito entitami je realizována asociace `OneToOne`, jež je využívána pro získání role uživatele a jeho následnou autentizaci a autorizaci.

Jako přístupové údaje slouží login[63] a heslo. Hesla jsou při persistování hashována pomocí hashovací funkce `bcrypt` s deseti rundami a za využití soli.

5.7.2 Autentizace a autorizace během přihlášení

Autentizace a autorizace během přihlášení je implementována na úrovni metod `startup` abstraktních presenterů `TeacherPresenter` a `StudentPresenter` dědících z abstraktního presenteru `SecuredPresenter`, jenž implementuje rozhraní `ISecuredPresenter`. Od `TeacherPresenter` a `StudentPresenter` dědí všechny presentery daných modulů a získávají tak autentizační a autorizační funkci. Dle 4.4 je metoda `startup` prováděna před všemi ostatními metodami presenteru, lze tedy spolehlivě využít pro ověření přístupových oprávnění.

Samotný proces autentizace a přiřazení role službě `User`, již Nette dle [44] využívá pro reprezentaci uživatele, zajišťuje třída `Authenticator` implementující rozhraní `IAuthenticator`. Authenticator se nejprve pokusí vyhledat uživatele s daným loginem pomocí `UserRepository`. V případě autentizace pro modul `Teacher` navíc vyhledává pouze uživatele v rolích `Admin` a `Teacher`. Pokud byl nalezen vyhovující uživatel, přistoupí k ověření hesla pomocí služby `Passwords`. V případě, že autentizace uživatele skončila neúspěchem, přístup je odepřen a uživateli je tato skutečnost oznámena. Pokud autentizace uživatele proběhla v pořádku, přiřadí `Authenticator` uživateli roli. Pokud má uživatel přiřazenou roli `Admin`, udělí mu přístup ke všem sadám úloh ve cvičebnici. V opačném případě udělí uživateli přístup pouze k sadám úloh, jež mu byly zpřístupněny, či jejichž je autorem.

5.7.3 Autorizace při přístupu k tématu úloh

Autorizaci při přístupu k tématu úloh v modulu `Student` zajišťuje služba `Authorizator`. Autorizace spočívá v pokusu o vyhledání požadovaného tématu v identitě daného uživatele. O způsobu přiřazení povolených témat uživateli pojednává předchozí sekce. V případě, že je požadované téma v identitě nalezeno, je uživateli udělen přístup k sadě úloh odpovídající danému tématu.

5.7.4 Autorizace při přístupu k entitám v modulu Teacher

Autorizace během přístupu k entitám je zajištěna metodou `isEntityAllowed` služby `Authorizator`. Ta je volána během akce `update` abstraktního presen-

[63] e-mailová adresa či uživatelské jméno

teru `EntityPresenter`, od něhož dědí presentery všech entit. Při vytvoření entity s omezením přístupu pro roli `Teacher` je k této entitě přiřazen její autor. Toho využívá služba `Authorizator`, jež realizuje autorizaci porovnáním autora entity s identitou přihlášeného uživatele.

5.8 Správa assetů

Za účelem správy zdrojových souborů CSS, SASS a JavaScriptu, využívaných pro stylizování a dynamiku front-endu aplikace, byly využity nástroje **Webpack** a **NPM**. Tato sekce se zaměří na uplatnění těchto nástrojů v rámci vývoje aplikace.

5.8.1 Webpack

Webpack představuje nástroj pro vytváření **balíčků** z modulárního kódu za účelem interpretace v prohlížeči.

Node.js jakožto terminálový interpret JS podporuje code splitting[64], jehož však nepodporuje JS interpretovaný prohlížečem. Ve větších projektech pak nemusí být jasný původ závislostí a způsob fungování kódu. Webpack umožňuje psát kód modulárně. Jeho vstupem je modulární kód, případně další soubory a výstupem je balíček. [45]

Webpack umožňuje práci s různými druhy assetů. Při správné konfiguraci umožňuje vytvářet balíčky sestavené nejen z JS modulů, ale například také ze souborů CSS či SASS, jež lze importovat jako závislosti do modulů. Výstupem jsou primárně JS balíčky pro použití v prohlížeči. Assety ostatních typů musí být zpracovány pomocí **loaderů** a **pluginů**. [45]

Konfigurační soubor Webpacku je Node.js modul exportující konfigurační objekt. Výchozím konfiguračním souborem je `webpack.config.js`. Aplikace kromě výchozího souboru využívá konfigurační soubory prostředí. Ukázka 10 obsahuje část výchozího konfiguračního souboru. [45]

Základní klíče konfiguračního objektu mají následující význam:

entry nastavuje vstupní moduly pro zpracování Webpackem. V ukázce se jedná o modul `student.js`;

output nastavuje umístění a schéma názvu výstupních balíčků. V ukázce je název balíčků tvořen jménem vstupního modulu, hashem generovaným Webpackem a koncovkou;

module obsahuje nastavení zpracování modulů. Loadery jsou transformace aplikované na vstupní moduly před vytvořením balíčku. Klíč module je tvořen pravidly pro uplatnění loaderů. Každé pravidlo je tvořeno testem názvu modulu pomocí regulárního výrazu. Pokud modul vyhoví testu,

[64] rozdělení kódu do modulů závislostí

je na něj aplikován daný loader. Při využití více loaderů je Webpack aplikuje odspoda nahoru. V ukázce je pravidlo, které na assety formátu `.scss` či `.css` aplikuje `sass-loader`.

```
entry: {
    student: path.resolve(ROOT_DIR, 'resources', 'student.js')
},
output: {
    path: DIST_DIR,
    filename: '[name].[hash].bundle.js'
},
module: {
    rules: [
        {
            test: /\.(scss|css)$/,
            use : [ { loader: 'sass-loader' } ]
        }
    ]
}
```

Ukázka kódu 10: Konfigurace nástroje Webpack

5.8.2 NPM

NPM představuje nástroj pro správu balíčků. Balíčky dostupné skrze NPM se nacházejí ve stejnojmenném repozitáři. Pro konfiguraci se využívá soubor `package.json`, jenž musí obsahovat alespoň položky `name` a `version`. `Package.json` může obsahovat položky `dependencies` a `devDependencies`, které specifikují závislosti v produkčním a vývojovém prostředí, či položku `scripts`, v níž je možné definovat vlastní skripty, které lze následně volat skrze NPM command line. [46] [47]

Část souboru `package.json` využívaného aplikací je v ukázce 11. V položce `scripts` lze vidět integraci nástroje Webpack do NPM. Ta v uvedeném případě umožňuje sestavit balíček assetů za pomoci příkazů `npm run dev` pro vývoj či `npm run prod` pro produkci.

```
{
  "dependencies": { "bootstrap": "^4.3.1" },
  "devDependencies": { "@babel/core": "^7.1.2" },
    "scripts": {
      "dev": "webpack --config ./webpack/webpack.devel.config.js",
      "prod": "webpack --config ./webpack/webpack.prod.config.js"
  }
}
```

Ukázka kódu 11: Soubor package.json

Kapitola 6

Testování

Tato kapitola se zabývá testováním, jímž aplikace prošla v závěru vývoje. Provedená testování lze rozdělit do třech typů. První z nich je **automatizované testování** aplikace za využití frameworku `PHPUnit`. Druhým typem je diagnostika aplikace pomocí **Google Lighthouse**. Poslední typ pak představuje **testování aplikace z uživatelského pohledu**, jež zahrnuje autorem provedené testování použitelnosti aplikace a uživatelské testování dle připravených scénářů použití aplikace.

6.1 Testovací data

Za účelem testování byla vytvořena sada dat, jejíž nasazení a stažení z databáze zapouzdřuje migrace `TestingDataV1`. Tato testovací data budou využita pro automatizované testy vyžadující pro svůj běh přístup k databázi, především se jedná o testy tříd `Repository` a tříd zajišťujících autentizaci a autorizaci. Vytvořená sada testovacích dat nalezne uplatnění rovněž v rámci uživatelského testování.

6.2 Automatizované testy

Tato sekce se zaměří na testování tříd aplikace za využití automatizovaných testů. Nejprve budou představeny realizované vrstvy automatizovaného testování, načež bude popsána samotná realizace testů. V závěru sekce bude uveden postup pro spouštění testů z konzole.

Pro účely testování byl využit framework `PHPUnit`[65], a to zejména díky jeho snadnému použití a kvalitní dokumentaci.

V rámci práce byla vytvořena sady testů pro vrstvu aplikační logiky aplikace. Každá testovací třída je potomkem abstraktní třídy `EDOMPTestCase` dě-

[65] dostupné z *phpunit.de*

dící od třídy `TestCase`, jež představuje základní třídu testů ve frameworku `PHPUnit`. Pro vybrané typy testů byly vytvořeny abstraktní třídy poskytující společné atributy a rozhraní testům daného typu. Jedná se například o třídy `PersistentEntityTestCase` či `FunctionalityUnitTestCase`.

Třídy testů jsou umístěny v adresáři `Tests`. Struktura tohoto adresáře kopíruje strukturu samotné aplikace umístěné v adresáři `App`, a to v rozsahu vytvořených testů.

Rozložení testů do této adresářové struktury má několik výhod:

- testy jsou přehledně strukturovány,
- na první pohled je jednoznačné, k jaké třídě aplikace daný test patří,
- lze spouštět sady testů pro jednotlivé jmenné prostory aplikace.

6.2.1 Unit testy

Unit testy představují nejnižší úroveň testování softwaru a zpravidla je vytvářejí samotní programátoři. Unit testy jsou založeny na individuálním testování jednotek softwaru. Účelem těchto testů je ověřit, zda tyto jednotky fungují dle očekávání. **Jednotka** představuje nejmenší testovatelnou část softwaru. V procedurálním programování se jedná o **funkce** či **procedury**. V případě objektově orientovaného programování jde o **metody** jednotlivých tříd. Pokud testovaná jednotka vyžaduje interakci s jednotkami jiných objektů, využívají se v unit testech tzv. **mock objekty**[66]. [48]

Některé z testovaných tříd pro svoji funkčnost využívají metod poskytovaných řadou dalších tříd. V případě unit testů byly k takovýmto třídám vytvořeny objekty typu `MockObject`, jež ve frameworku `PHPUnit` slouží k zastoupení skutečných objektů a umožňují simulaci chování metod tříd řízenou programátorem. Jednotlivé objekty typu `MockObject` jsou sestavovány ve vlastních traitách za účelem znovupoužitelnosti napříč testy. Tyto traity byly umístěny do jmenného prostoru `App\Tests\MockTraits`.

V rámci testování aplikace byla zhotovena sada unit testů pokrývající persistentní entity a podvrstvu `Functionality` aplikační logiky. Náhled vytvořeného unit testu pro metodu `create` třídy `GroupFunctionality` je k vidění v ukázce 12.

Zmiňovaného principu tvorby mock objektů bylo využito především v případě testování podvrstvy `Functionality`, jelikož zde bylo potřeba odstínit testované třídy od závislostí na třídách podvrstvy `Repository`, případně na ostatních třídách podvrstvy `Functionality`. Příklad vytvoření mock objektu třídy `ProblemRepository` je k náhledu v ukázce 13.

Unit testy bylo dále pokryto použití vybraných tříd na úrovni `Service` a `Helper`.

[66] falešné objekty poskytující rozhraní reálného objektu simulující jeho chování

```php
<?php
    $data = [
        'label' => 'TEST_GROUP',
        'superGroup' => 1,
        'created' => DateTime::from($this->dateTimeStr),
        'userId' => 1
    ];

    $expected = new Group();
    $expected->setLabel($data['label']);
    $expected->setSuperGroup(
        $this->superGroupRepositoryMock->find($data['superGroup']
    );
    ...

    $created = $this->functionality->create($data);
    $this->assertEquals($expected, $created);
```

Ukázka kódu 12: Unit test metody create pro třídu GroupFunctionality

```php
<?php
$this->problemRepositoryMock = $this->getMockBuilder(
    ProblemRepository::class
)
    ->disableOriginalConstructor()
    ->getMock();
```

Ukázka kódu 13: Vytvoření mock objektu třídy ProblemRepository

6.2.2 Integrační testy

Další vrstvou testování, jež následuje po unit testech, jsou integrační testy. V rámci této vrstvy testování jsou jednotky testované v rámci unit testů kombinovány dohromady a testovány jakožto celek. [49]

Integrační testy byly využity pro většinu tříd na úrovni `Service` z důvodu jejich rozsahu. Dále byly integrační testy využity pro třídy podvrstvy `Repository` aplikační logiky, jejichž smysluplné testování vyžaduje přístup k databázi. Zde byla pro účely testování využita připravená sada testovacích dat.

Samotný princip testování na úrovni integračních testů představuje obdobu unit testů. Rozdíl spočívá v tom, že již nejsou vytvářeny objekty typu `MockObject` sloužící k odstínění závislostí testovaných tříd na metodách jiných tříd. Testované třídy jsou namísto vytváření v rámci testů s využitím objektů typu `MockObject` načítány přímo z DI kontejneru aplikace. Všechny

závislosti ve formě ostatních tříd jsou tedy testované třídě předány aplikací během konstrukce DI kontejneru.

6.2.3 Spuštění testů

Součástí `PHPUnit` je konzolová aplikace, jež poskytuje příkaz pro spuštění sady testů. Pro spuštění testů z kořenového adresáře projektu stačí zadat do konzole příkaz `./vendor/bin/phpunit --bootstrap vendor/autoload.php <adr>`. Cesta k příkazové řádce `PHPUnit` je dána využitím Composeru pro instalaci frameworku `PHPUnit`. Jako hodnotu parametru `<adr>` lze uvést adresář s kompletní sadou testů[67], cestu k jeho libovolnému podadresáři či souboru uvnitř adresářové struktury testů. Dle hodnoty parametru dojde ke spuštění příslušné sady testů.

6.3 Diagnostika Google Lighthouse

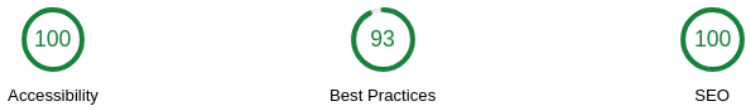
Nasazená aplikace byla podrobena diagnostice za pomoci nástroje Google Lighthouse. Diagnostika byla provedena se zaměřením na **SEO**[68], **přístupnost**[69] a **nejlepší praxi**.

Na základě provedené diagnostiky byly odhaleny nedostatky v oblasti přístupnosti a optimalizace pro vyhledávací služby.

V oblasti přístupnosti se jednalo o chybějící atributy `title` v případě některých odkazů a nedostatečný kontrast prvků hlavní nabídky oproti barvě pozadí. Autorem byla proto provedena revize HTML kódu a doplnění chybějících atributů. Dále byl vyladěn kontrast položek hlavní nabídky tak, aby vyhovoval požadavkům kladeným na přístupnost.

V případě SEO bylo zjištěno, že HTML kód aplikace neobsahuje meta tagy `author`, `description` a `keywords`.

Po odladění nedostatků byla aplikace ohodnocena 100 body v oblasti přístupnosti, 93 body v oblasti nejlepší praxe a 100 body v oblasti SEO, jak lze vidět na obrázku 6.1.



Obrázek 6.1: Výsledek diagnostiky Google Lighthouse

[67] tedy adresář `Tests`
[68] optimalizace pro vyhledávací služby
[69] zda je uživatelské rozhraní uzpůsobeno pro bezproblémové používání

6.4 Heuristická analýza

Tato sekce se zaměří na základní otestování použitelnosti aplikace, pro které bude využita heuristická analýza. Heuristická analýza spočívá v postupném procházení a hodnocení stránek uživatelského rozhraní aplikace. Během procházení je zkoumáno, zda stránky splňují doporučení týkající se použitelnosti. Po dokončení analýzy je vytvořen seznam nedostatků seřazených dle důležitosti. [50]

Sekce nejprve představí konkrétní heuristickou analýzu, jež byla pro účely testování zvolena. Následně bude vytvořen seznam zjištěných nedostatků. V závěru sekce budou uvedena řešení těchto nedostatků.

6.4.1 Nielsnova heuristická analýza

Heuristická analýza rozhraní byla autorem provedena dle desatera principů použitelnosti Jakoba Nielsna. [50]

1. **Viditelnost stavu systému** Uživatel by měl vždy vědět, co se v systému odehrává. V případě webu by uživatel měl například vždy vědět, kde se nachází a měl by dostávat zpětnou vazbu o tom, co právě řeší.

2. **Spojení mezi systémem a reálným světem** Systém by měl s uživatelem komunikovat uživatelsky příjemným způsobem.

3. **Uživatelská kontrola a svoboda** Uživatelé dělají při práci se systémem chyby a je nutné, aby měli možnost se pomocí rozhraní vrátit a chybnou volbu změnit či napravit.

4. **Konzistence a standardizace** Uživatelé by neměli být nuceni přemýšlet, zda různé termíny či symboly vyjadřují to samé.

5. **Prevence chyb** Systém by se měl bezpečným designem, který bude preventivně působit proti problémům, vyvarovat chybovým hlášením.

6. **Rozpoznání místo vzpomínání** Uživatel by neměl být nucen vzpomínat si na provádění operací v systému. Instrukce by měly být viditelně umístěny. Všechny vzájemně související prvky by měly být viditelné.

7. **Flexibilní a efektivní použití** Zkušenější uživatel by měl mít možnost urychlit některé frekventované akce.

8. **Estetický a minimalistický design** Rozhraní systému by mělo obsahovat pouze relevantní informace a vyhnout se zbytečnému textu.

9. **Pomoc uživateli poznat chyby a vzpamatovat se z nich** Chybové hlášky by měly být uživatelsky přívětivé, lehce pochopitelné a měly by navrhovat řešení.

10. Nápověda a návody Všechny potřebné informace musí být snadno dohledatelné. Nápověda by měla obsahovat postupy v krocích.

6.4.2 Seznam nedostatků

Na základě provedené heuristické analýzy bylo zjištěno několik nedostatků uživatelského rozhraní aplikace. Tato sekce uvede jejich seznam a zjištěné nedostatky rozvede.

6.4.2.1 Prevence chyb

Během testování byl zjištěn nedostatečný popis vstupů formulářů. V případě formulářových prvků chyběly upřesňující popisy vstupů, jež by uživateli poskytovaly informace navádějící ho ke korektnímu vyplnění formuláře. Zejména se jednalo o označení povinných vstupů či uvedení požadovaného formátu vstupních dat.

6.4.2.2 Rozpoznávání místo vzpomínání

Bylo zjištěno, že v případě složitějších formulářů se některé ovládací prvky nacházely mimo obrazovku a nebyly uživateli dostupné ihned po korektním vyplnění formuláře. Například při zadávání statistiky úspěšnosti úloh v testu musel uživatel scrollovat zpět ke tlačítku pro uložení.

Dále bylo zjištěno, že v případě, kdy neexistuje žádný záznam k entitě potřebné pro úspěšné odeslání formuláře, musel uživatel dohledávat sekci dané entity za účelem vytvoření záznamu.

6.4.2.3 Chybějící nápovědy

V aplikaci nebyla dostupná nápověda obsahující souhrn podpory jazyka pro LaTeX a popis tvorby šablon. Jelikož se jedná o úkony specifické pro výstupní aplikaci a z uživatelského rozhraní není možné tyto informace zjistit, dává zde smysl využití podrobnější nápovědy.

6.4.3 Řešení nedostatků

Tato sekce popíše řešení nedostatků zjištěných během heuristické analýzy, jež byly uvedeny v předchozí sekci.

6.4.3.1 Prevence chyb

Nedostatek byl vyřešen označením povinných vstupů formulářů uvedením hvězdičky na konci popisku daného formulářového prvku. Ke každému formulářovému prvku byla navíc přidána stručná instrukce ve formě zástupného textu.

6.4.3.2 Rozpoznávání místo vzpomínání

V případě nedostupnosti ovládacích prvků formulářů v rámci jedné obrazovky byl nedostatek vyřešen uvedením těchto prvků také na konec daných formulářů.

Další nedostatek spočíval v nutnosti dohledat sekci aplikace, jež poskytuje rozhraní pro vytvoření potřebné neexistující entity. Nedostatek byl vyřešen odstíněním uživatele od hledání napříč rozhraním. Toho bylo dosaženo poskytnutím odkazu do příslušné sekce v místě, kde měla být zobrazena potřebná entita.

6.4.3.3 Chybějící nápovědy

Poslední z nedostatků byl vyřešen formou modálního okna obsahujícího potřebnou nápovědu. Toto modální okno je vyvoláno kliknutím na ikonu otazníků v horní liště aplikace.

6.5 Uživatelské testování

Tato sekce se soustředí na testování aplikace uživateli. Uživatelské testování představuje další stupeň testování použitelnosti aplikace. Dle [51] je ideální počet účastníků uživatelského testování pět.

Uživatelské testování bylo provedeno formou průchodů několika připravenými scénáři použití aplikace. Jednotlivá testování byla prováděna v různých časových úsecích a jsou seřazena chronologicky. Mezi jednotlivými uživatelskými testováními byly průběžně opravovány zjištěné nedostatky v použitelnosti aplikace. Vzhledem ke skutečnosti, že modul `Teacher` je výrazně rozsáhlejší než modul `Student`, zaměří se většina scénářů právě na průchod modulem `Teacher`.

6.5.1 Testovací scénáře

Tato sekce představí připravené testovací scénáře, jimiž uživatelé procházeli během uživatelského testování. Pro prvních šest testovacích scénářů se předpokládá přihlášení pod účtem v roli `Učitel`, pro poslední scénář pak přihlášení v roli `Student`.

Účelem testovacích scénářů je otestovat **míru použitelnosti** jednotlivých sekcí aplikace, zjistit případné nedostatky aplikace a následně přijít s jejich řešením.

TS1: Tvorba a editace uživatele

1. Nalezení položky `Uživatelé` v hlavní nabídce a vstup do sekce.
2. Kliknutí na ikonu přidat a vytvoření nového uživatele.

3. Vyhledání zadaného uživatele v seznamu.
4. Nalezení ikony editace pro zadaného uživatele a vstup do editace.
5. Editace údajů uživatele a uložení změn.

TS2: Tvorba a editace finální úlohy

1. Nalezení položky `Finální úlohy` v hlavní nabídce a vstup do sekce.
2. Kliknutí na ikonu přidat a vytvoření nové finální úlohy.
3. Nalezení ikony editace vytvořené finální úlohy a vstup do editace.
4. Editace atributů finální úlohy a uložení změn.

TS3: Tvorba a editace šablony úloh

1. Nalezení položky `Šablony úloh` v hlavní nabídce a vstup do sekce.
2. Kliknutí na ikonu přidat a vytvoření nové šablony úloh včetně přiřazení a ověření podmínky.
3. Nalezení ikony editace vytvořené šablony úlohy a vstup do editace.
4. Editace atributů finální úlohy včetně změny podmínky a ověření její splnitelnosti. Uživatel nakonec uložení změny.

TS4: Nahrání a editace loga

1. Nalezení položky `Loga` v hlavní nabídce a vstup do sekce.
2. Kliknutí na ikonu přidat, zadání názvu loga a nahrání souboru.
3. Zobrazení náhledu loga v seznamu.
4. Nalezení ikony editace vytvořeného loga a vstup do editace.
5. Editace atributů loga včetně nahrání nového souboru. Nakonec uživatel uloží změny.

TS5: Tvorba testu

1. Nalezení položky `Testy` v hlavní nabídce a vstup do sekce.
2. Kliknutí na ikonu přidat a přechod do tvorby testu.
3. Volba loga testu a dalších parametrů testu.
4. Zadání úloh do testu a vygenerování testu.

5. Dohledání vygenerovaného testu, jeho uzavření a stažení.
6. Zápis statistiky úspěšnosti úloh v testu.

TS6: Nastavení oprávnění k úlohám

1. Nalezení položky `Nastavení` v hlavní nabídce a vstup do sekce.
2. Volba `Oprávnění skupin`.
3. Nastavení oprávnění zadaných skupin k zadaným tématům.

TS7: Použití sbírky úloh

1. Nalezení sbírky úloh zadaného tématu v hlavní nabídce.
2. Nalezení alespoň jedné finální úlohy vygenerované ze šablony vytvořené v testovacím scénáři 6.5.1.
3. Listování úlohami sbírky.
4. Filtrování dle obtížnosti a tématu.
5. Resetování filtrů.

6.5.2 Účastníci testování

Uživatelské testování bylo provedeno s následujícími pěti účastníky:

Účastník 1 je studentem FIT ČVUT. Účastník má zkušenosti s programováním v jazycích C, C++ a JavaScript. Dále účastník ovládá jazyk systému LaTeX. Zároveň má povědomí o základních požadavcích na použitelnost uživatelského rozhraní.

Účastník 2 je absolventem Střední průmyslové školy strojnické v Plzni. Jeho zkušenosti s užíváním webových aplikací jsou čistě na uživatelské úrovni.

Účastník 3 je absolventem FIS VŠE a pracuje jako vývojář webových aplikací. Je zkušeným programátorem v PHP a JavaScriptu. Orientuje se v požadavích na moderní webové aplikace a použitelné uživatelské rozhraní.

Účastník 4 je absolventem SPŠ Prosek. Pracuje jako front-endový webový vývojář. Má hlubší znalosti v oblasti použitelnosti uživatelského rozhraní a v oblasti moderního designu.

Účastník 5 působí jako vývojář webových aplikací. Orientuje se v požadavcích na použitelné uživatelské rozhraní.

Průchody **účastníků 1 a 2** proběhly za přítomnosti autora, kdy autor pozoroval chování účastníků během používání aplikace a konzultoval s nimi případné problémy. Průchody **účastníků 3 až 5** proběhly samostatně, bez přítomnosti autora. Přístupy do aplikace byly účastníkům poskytnuty zaslání pozvánky do aplikace za využití funkcionality modulu `Teacher`.

6.5.3 Průchody testovacími scénáři

Tato sekce rozvede zpětnou vazbu získanou z průchodů jednotlivých uživatelů skrze testovací scénáře. V každém průchodu budou uvedeny případné zjištěné nedostatky a jejich řešení.

6.5.3.1 Průchod účastníka 1

TS1: Aplikace účastníka přesměrovala na přihlášení do modulu `Student`, kde mu nebyl k dispozici odkaz na přihlášení do modulu `Teacher`, a účastník nevěděl, jak se k tomuto přihlášení dostat.

TS2: Účastník se pozastavil nad tím, kam vložit zadání úlohy.

TS3: Účastníkem bylo zjištěno, že formulářem prochází jakožto zadání šablon úloh vstupy, jež nejsou validně uzavřeny ve značkách pro matematický režim systému LaTeX. Dále bylo zjištěno, že zpráva o ověření splnitelnosti podmínky se nenachází na jedné obrazovce spolu s tlačítkem pro ověření.

TS4: Průchod testovacím scénářem proběhl bez výhrad účastníka.

TS5: Během vyplňování údajů testu bylo zjištěno, že lze zadat školní rok v nevalidním formátu. Dále bylo zjištěno, že na front-endu není omezena hodnota zadávaná jako číslo testu. Kromě toho nebylo účastníkovi příliš jasné, jaký vliv má na generování testu volba konkrétních šablon či finálních úloh.

TS6: Průchod testovacím scénářem proběhl bez výhrad účastníka.

TS7: Po odhlášení z modulu `Teacher` se účastník bez pomoci nedokázal dostat k přihlášení do modulu `Student`.

Nedostatek zjištěný v rámci **TS1** byl vyřešen doplněním odkazu na přihlášení do modulu `Teacher`.

Na základě nedostatku zjištěného v **TS2** byl upraven zástupný text příslušného formulářového prvku.

Chyba ve validaci vstupní šablony objevená v rámci **TS3** spočívala v možnosti opakovaně zadat značku pro inline matematický mód v kombinaci se značkou pro klasický matematický mód, tedy šablony ve formátu `$$ ... $` či `$ ... $$`. Tato chyba byla opravena. Zjištěný nedostatek v použitelnosti byl

vyřešen poskytnutím odkazu na kotvu tlačítka pro ověření podmínky v rámci zmíněné zprávy.

Chyba ve validaci školního roku zjištěná v **TS5** byla opravena. Omezení zadávaného čísla testu bylo vyřešeno uvedením atributů `min` a `max` k příslušnému formulářovému prvku. Možnosti generátoru testů byly popsány v nápovědě sekce a možnost jejího zobrazení byla dodána přímo do formuláře tvorby testu.

Na základě průchodu **TS7** bylo přihlášení do modulu `Teacher` rozšířeno o odkaz na přihlášení do modulu `Student`.

6.5.3.2 Průchod účastníka 2

TS2: Pořadí formulářových prvků během tvorby a editace finální úlohy bylo pro účastníka matoucí. V důsledku toho se zprvu pokoušel o zadání úlohy do prvního formulářového prvku typu `textarea`, jenž však reprezentoval vstup pro úvod zadání. Nad tímto se pozastavil již **Účastník 1**. Řešení spočívající ve směrodatnějším zástupném textu však nezabránilo tendenci vkládat zadání do prvního prvku typu `textarea`, tudíž je na místě tuto připomínku řešit.

TS3: Zde se účastník již s pořadím formulářových prvků vypořádal. Zpráva o úspěchu ve splnitelnosti zadané podmínky šablony se zobrazí v horní části editačního formuláře. Díky tomu se nenachází na jedné obrazovce spolu s tlačítkem akce pro ověření splnitelnosti. Zároveň toto chování nekoresponduje s chováním formulářů pro tvorbu úloh.

TS5: Během testovacího scénáře byla objevena chyba spočívající v neuložení filtrů podmínek šablon k vytvořenému testu. To způsobovalo zanedbání navoleného filtru v případě přegenerování testu.

Průchody testovacími scénáři **TS1**, **TS4**, **TS6** a **TS7** proběhly bez výhrad účastníka.

Matoucí pořadí prvků formulářů pro tvorbu a editaci finálních úloh a šablon úloh objevené v rámci scénáře **TS1** bylo vyřešeno změnou pořadí tak, aby odpovídalo očekávání účastníků testování.

Nedostatek zjištěný v rámci **TS2** byl vyřešen přesunutím zprávy o úspěchu ve splnitelnosti zadané podmínky šablon do spodní části formuláře, pod tlačítko akce pro ověření splnitelnosti příslušné podmínky.

Chyba objevená během **TS5** byla opravena.

6.5.3.3 Průchod účastníka 3

TS5: Chování volby loga testu bylo pro účastníka neintuitivní. Konkrétně zde pro účastníka bylo matoucí zrušení výběru loga kliknutím na logo ve výběru. Zároveň bylo účastníkovi nejasné ovládání formulářů filtrujících

množiny úloh pro umístění do zadání testu. Účastníkovi zde nebyl jasný rozdíl mezi typem úloh a tématem úloh. Dále mu celkově nebyly jasné možnosti filtrování.

TS7: Filtrování úloh ve sbírce přišlo účastníkovi nedostatečně výrazné.

Průchody testovacími scénáři **TS1**, **TS2**, **TS3**, **TS4** a **TS6** proběhly bez výhrad účastníka.

Nedostatek spočívající v neintuitivním chování volby loga objevený v rámci **TS5** byl vyřešen doplněním popisku rovněž k oblasti dostupných log. V oblasti zvoleného loga byl dále doplněn popisek pro stav aktivní volby loga popisující chování výměny a zrušení volby loga. Nejasnosti ve formulářích pro filtrování množin úloh pro umístění do zadání byly vyřešeny směrodatnějšími zástupnými texty a popisky jednotlivých prvků filtrovacích formulářů. Dále byly k popiskům doplněny rychlé nápovědy ve formě prvku `tooltip` z frameworku Bootstrap.

Nedostatek zjištěný během **TS7** byl vyřešen větší velikostí písma v hlavičce karty filtrování. Filtrování bylo dále umístěno nad stránkování úloh.

6.5.3.4 Průchod účastníka 4

TS1: Účastník testování očekával tlačítka `Uložení` a `Zpět` rovněž ve spodní části karty pro editaci. Vycházel z formulářů pro tvorbu, kde se tlačítka `Uložení` a `Zpět` nacházejí v patičce modálního okna.

TS5: Díky rozsáhlosti formuláře tvorby testu musel účastník po odeslání nevalidně vyplněného formuláře dohledávat chybové hlášky za pomoci scrollování.

TS6: Při editaci oprávnění se multiselect s dostupnými tématy úloh rozbalil nahoru a nezmizel, dokud účastník neklikl mimo rozbalený multiselect. Díky tomu multiselect překrýval hlášku aplikace.

Průchody testovacími scénáři **TS2**, **TS3**, **TS4** a **TS7** proběhly bez výhrad účastníka.

Nedostatek v použitelnosti odhalený během **TS1** se týká všech editací dostupných v modulu `Teacher`. Nedostatek byl vyřešen rozšířením karty editace o patičku, v rámci níž jsou k dispozici tlačítka `Uložit` a `Zpět`.

Nutnost vyhledávat chybové hlášky ve formulářích pomocí scrollování, objevená v rámci **TS5**, byla odstraněna rozšířením nad zpracováním AJAXových požadavků v Nette. Rozšíření je z back-endu aplikace předána hodnota atributu `name` prvního formulářového prvku, u něhož se vyskytla chyba. Samotné rozšíření následně zajistí automatický, animovaný přesun k tomuto formulářovému prvku.

Překrývání aplikační hlášky zjištěné v rámci **TS6** bylo vyřešeno globálním přesunutím vhodných hlášek do modálního okna zobrazeného po provedení

akce. Tím byl zároveň vyřešen problém nedostatečné výraznosti některých aplikačních hlášek, či viditelnost hlášek při rychlé inline editaci entit ve spodních řádcích datagridů.

6.5.3.5 Průchod účastníka 5

TS2: Účastníkovi testování chyběla v blízkosti vstupu pro zadání nápověda s popisem formátu zadání finálních úloh, případně s podporou jazyka pro LaTeX a příklady validních zadání.

TS3: Zde účastníkovi stejně jako v rámci **TS2** chyběla nápověda v blízkosti vstupu pro zadání šablony úloh popisující validní formát vstupu.

TS5: Účastníkovi nebyl ze zástupného textu příliš jasný formát školního roku. Dále mu chyběl podrobnější popis akcí v datagridu testů. Konkrétně se jednalo o chybějící popis obsahu staženého archivu a důvod k registraci na službě Overleaf.

TS7: Pro účastníka bylo obtížné rozeznat, zda se nachází v přihlášení pro učitele či studenta.

Průchody testovacími scénáři **TS1**, **TS4** a **TS6** proběhly bez výhrad účastníka.

Nedostatek objevený v rámci **TS2** a **TS3** byl vyřešen doplněním ikony pro zobrazení nápovědy sekce do blízkosti vstupů pro zadání finálních úloh a šablon úloh.

Na základě **TS5** byla k prvku školního roku doplněna ikona s nápovědou obsahující výčet validních formátů školního roku. U akcí pro stažení archivu a přípravy ke kompilaci variant zadání do PDF byly rozšířeny popisky v rámci atributu `title`.

Dle poznatku z **TS7** bylo do popisků přihlašovacích tlačítek doplněno, o jaké přihlášení se jedná.

6.6 Vyhodnocení testování

K vybraným třídám aplikace byly vedeny, případně dodatečně vytvořeny automatizované testy. V závěru implementace byla aplikace otestována z hlediska přístupnosti, SEO, použitelnosti a prošla uživatelským testováním o pěti účastnících.

Přístupnost byla otestována pomocí diagnostiky Google Lighthouse. Spuštěná diagnóza byla zaměřena mimo jiné také na SEO a nejlepší praxi. Z výsledků diagnózy vyplynulo například, že rozhraní aplikace porušovalo zásady přístupnosti, a to nedodržením dostatečného kontrastu mezi barvou písma a pozadí.

Následovalo důkladné testování použitelnosti aplikace. Nejprve byla pou-

žitelnost otestována autorem práce formou heuristické analýzy. Během ní bylo zjištěno, že u formulářových prvků často nebyly označeny povinné položky, chyběly nápovědy ve formě zástupných textů. Dále bylo například odhaleno, že aplikace neobsahuje popis podpory jazyka pro `LaTeX` či formátu pro zadávání šablon úloh.

Nakonec byla aplikace podrobena uživatelskému testování, během kterého bylo objeveno několik chyb v samotném běhu aplikace. To bylo způsobeno především častými úpravami v kódu prováděnými před či v průběhu uživatelského testování. První dva účastníci otestovali aplikaci za přítomnosti autora. Zbývající tři prováděli testování samostatně. Během uživatelského testování byla objevena řada nedostatků v použitelnosti rozhraní aplikace, jež autor v rámci heuristické analýzy z důvodu znalosti rozhraní neodhalil. Dle očekávání se tedy potvrdila důležitost uživatelského testování. Všechny objevené nedostatky v rozhraní aplikace byly odstraněny. Mimo jiné byla aplikace průběžně upravována dle připomínek a návrhů účastníků testování.

Závěr

Cílem práce bylo navrhnout a implementovat elektronickou sbírku příkladů z matematiky ve formě webové aplikace. Návrh aplikace měl být proveden na základě rešerše stávajících webových aplikací pro správu zadání a řešení matematických úloh. Aplikace měla být implementována za využití jazyka PHP. Implementovaná aplikace měla umožnit správu uživatelů a matematických úloh, jež mělo být umožněno klasifikovat dle tématu a obtížnosti. Kromě toho měla být umožněna správa řešení jednotlivých úloh. Aplikace měla dále podporovat generování zadání testů a úloh s výstupem do jazyka systému LaTeX. Pro generované úlohy mělo být umožněno volit jejich vlastnosti. Dále měla aplikace umožnit sledovat statistiku úspěšnosti úloh v testech a studentům umožnit procvičování úloh.

V samotném počátku práce bylo potřeba provést rešerši stávajících řešení v rozsahu požadavků kladených na implementovanou aplikaci. Na základě získaných poznatků byly zvoleny vhodné vlastnosti, jimiž měla vznikající aplikace disponovat.

Dále byla provedena analýza funkčních a nefunkčních požadavků kladených na vznikající aplikaci. Analyzovány byly rovněž uživatelské role vystupující v aplikaci a jednotlivé případy užití. Na závěr analýzy byl zmapován doménový model vznikající aplikace.

V rámci návrhu byl zvolen a představen framework využitý pro samotnou implementaci. Autor se zde zaměřil rovněž na návrh modulární architektury aplikace tak, aby byla umožněna její rozšiřitelnost.

Na základě provedené analýzy a návrhu byla implementována aplikace vyhovující všem kladeným požadavkům. Samotný způsob implementace je zmapován v příslušné kapitole, jež se zabývá implementací navržených modulů aplikace a jejich funkcionalit vycházejících z funkčních požadavků na aplikaci.

Vybrané třídy aplikace byly testovány pomocí automatizovaných testů za využití frameworku PHPUnit. Implementovaná aplikace byla podrobena diagnostice pomocí nástroje Google Lighthouse. Dále bylo přistoupeno k testování použitelnosti aplikace. Té se nejprve věnoval sám autor formou heuris-

tické analýzy uživatelského rozhraní aplikace. Posléze byla aplikace podrobena uživatelskému testování. Zjištěné nedostatky byly odstraněny.

Implementovaná aplikace byla úspěšně nasazena na server. Pro účely běhu aplikace byla rovněž nasazena vlastní instance služby Newton, jež běží na cloudové platformě Heroku. Aplikace pro účely prováděných úprav výrazů komunikuje s touto nasazenou instancí.

Díky modulární architektuře, třívrstvé architektuře MVP poskytované frameworkem Nette, členění do komponent a zapouzdření logiky zpracování jednotlivých typů šablon úloh do samostatných tříd je aplikace připravena pro případná budoucí rozšíření. V průběhu implementace bylo na konzultacích zjištěno, že aplikaci by v budoucnu bylo možné kromě podpory generování dalších typů úloh rozšířit například o modul poskytující komplexnější přehledy statistik.

Bibliografie

1. TOMÁŠ KALVODA Karel Klouda, KAM FIT ČVUT. *MARAST* [online]. 2018 [cit. 2019-10-18]. Dostupné z: https://marast.fit.cvut.cz.
2. FOUNDATION, CK-12. *Welcome to CK-12 Foundation | CK-12 Foundation* [online]. 2019 [cit. 2019-10-18]. Dostupné z: https://www.ck12.org.
3. PRIMMAT – SOUKROMÁ STŘEDNÍ ŠKOLA PODNIKATELSKÁ, s.r.o. *Sbírka příkladů z matematiky* [online]. 2012 [cit. 2019-10-20]. Dostupné z: http://sbirkaprikladu.eu.
4. TOMÁŠ KALVODA, Karel Klouda. *MARAST* [online]. 2018 [cit. 2019-10-18]. Dostupné z: https://marast.fit.cvut.cz/cs/about.
5. FOUNDATION, CK-12. *About Us | CK-12 Foundation* [online]. 2019 [cit. 2019-10-18]. Dostupné z: https://www.ck12info.org/about/about-us.
6. PRIMMAT – SOUKROMÁ STŘEDNÍ ŠKOLA PODNIKATELSKÁ S.R.O. *Sbírka příkladů z matematiky* [online]. 2012 [cit. 2019-10-20]. Dostupné z: http://sbirkaprikladu.eu/about.aspx.
7. PRIMMAT – SOUKROMÁ STŘEDNÍ ŠKOLA PODNIKATELSKÁ S.R.O. *Sbírka příkladů z matematiky* [online]. 2012 [cit. 2019-10-20]. Dostupné z: http://sbirkaprikladu.eu/Admin/Users/registerUser.aspx.
8. MACEK, David. *7 věcí, které nesmí chybět na moderním webu* [online]. 2017 [cit. 2019-10-22]. Dostupné z: https://blog.poski.com/7-pravidel-moderniho-webu/.
9. ALTEXSOFT INC. *Functional and Non-functional Requirements: Specification and Types* [online]. 2018 [cit. 2019-10-22]. Dostupné z: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.

10. INTERACTION DESIGN FOUNDATION. *What is Responsive Design? | Interaction Design Foundation* [online]. 2019 [cit. 2019-10-22]. Dostupné z: https://www.interaction-design.org/literature/topics/responsive-design.

11. W3COUNTER.COM. *W3Counter: Global Web Stats* [online]. 2019 [cit. 2019-10-22]. Dostupné z: https://www.w3counter.com/globalstats.php.

12. SWARTZ, Nathan. *Three Reasons not to Use Flash, and Why* [online]. 2019 [cit. 2019-10-22]. Dostupné z: https://clicknathan.com/about/no-flash/.

13. GAMMA, Smart. *What are the pros and cons of using PHP? – Smart Gamma – Medium* [online]. 2016 [cit. 2019-10-22]. Dostupné z: https://medium.com/@smartgamma/what-are-the-pros-and-cons-of-using-php-490553ed8ff2.

14. PACKAGIST.ORG. *Packagist* [online]. 2019 [cit. 2019-10-22]. Dostupné z: https://packagist.org/statistics.

15. ORACLE CORPORATION. *MySQL :: MySQL 8.0 Reference Manual :: 1.3.1 What is MySQL?* [online]. 2019 [cit. 2019-10-22]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html.

16. HUJER, Martin. *Jaké novinky přinese PHP 7 - Zdroják* [online]. 2015 [cit. 2019-11-03]. Dostupné z: https://www.zdrojak.cz/clanky/jake-novinky-prinese-php-7/.

17. TECHOPEDIA INC. *What is Application Framework? - Definition from Techopedia* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://www.techopedia.com/definition/6005/application-framework.

18. MOZILLA FOUNDATION. *Server-side web frameworks - Learn web development | MDN* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks.

19. YASKEVICH, Anastasia. *Web application framework: What it is, how it works, and why you need it* [online]. 2017 [cit. 2019-11-03]. Dostupné z: https://www.scnsoft.com/blog/web-application-framework.

20. NJENGA, Alice. *10 Popular PHP frameworks in 2019 · Raygun Blog* [online]. 2018 [cit. 2019-11-03]. Dostupné z: https://raygun.com/blog/top-php-frameworks/.

21. DOCTRINE-PROJECT.ORG. *Object Relational Mapper - Doctrine - PHP Database Tools* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://www.doctrine-project.org/projects/orm.html.

22. DOCTRINE-PROJECT.ORG. *Introduction - Database Abstraction Layer (DBAL) - Doctrine* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https:

//www.doctrine-project.org/projects/doctrine-dbal/en/2.9/reference/introduction.html#introduction.

23. W3C. *HTML & CSS - W3C* [online]. 2016 [cit. 2019-11-03]. Dostupné z: https://www.w3.org/standards/webdesign/htmlcss.
24. WOOD, Adam. *HTML5: What's New in The Latest Version of HTML?* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://html.com/html5/#What_is_HTML5.
25. WODEHOUSE, Carey. *CSS vs. CSS3* [online]. 2016 [cit. 2019-11-03]. Dostupné z: https://www.upwork.com/hiring/development/css-vs-css3/.
26. SASS-LANG.COM. *Sass: Documentation* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://sass-lang.com/documentation.
27. MOZILLA FOUNDATION. *CSS preprocessor - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor.
28. W3SCHOOLS.COM. *jQuery Introduction* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://www.w3schools.com/jquery/jquery_intro.asp.
29. GETBOOTSTRAP.COM. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://getbootstrap.com/.
30. NETTE FOUNDATION. *Rychlý a pohodlný vývoj webových aplikací v PHP | Nette Framework* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://nette.org/cs/.
31. NETTE FOUNDATION. *Životní cyklus aplikace | Nette Documentation* [online]. 2019 [cit. 2019-11-03]. Dostupné z: https://doc.nette.org/cs/2.4/request-lifecycle.
32. ČÁPKA, David. *MVC Architektura* [online]. 2018 [cit. 2019-11-04]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor.
33. SINHAL, Ankit. *MVC, MVP and MVVM Design Pattern - Ankit Sinhal - Medium* [online]. 2017 [cit. 2019-11-04]. Dostupné z: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad.
34. NETTE FOUNDATION. *Latte | Nette Framework* [online]. 2019 [cit. 2019-11-05]. Dostupné z: https://latte.nette.org/cs/guide.
35. NETTE FOUNDATION. *Presentery | Nette Documentation* [online]. 2019 [cit. 2019-11-04]. Dostupné z: https://doc.nette.org/cs/2.4/presenters.

36. NILS ADERMANN, JORDI BOGGIANO AND MANY COMMUNITY CONTRIBUTIONS. *Introduction - Composer* [online] [cit. 2019-11-07]. Dostupné z: `https://getcomposer.org/doc/00-intro.md`.

37. NILS ADERMANN, JORDI BOGGIANO AND MANY COMMUNITY CONTRIBUTIONS. *Basic usage - Composer* [online] [cit. 2019-11-07]. Dostupné z: `https://getcomposer.org/doc/01-basic-usage.md`.

38. NASH, Gerald. *GitHub - aunyks/newton-api: A really micro microservice for advanced math.* [online]. 2017 [cit. 2019-11-07]. Dostupné z: `https://github.com/aunyks/newton-api`.

39. OVERLEAF.COM. *Overleaf, Online LaTeX editor* [online]. 2019 [cit. 2019-11-07]. Dostupné z: `https://cs.overleaf.com/devs`.

40. SYMFONY.COM. *Validation (Symfony Docs)* [online]. 2019 [cit. 2019-11-09]. Dostupné z: `https://symfony.com/doc/current/validation.html`.

41. DOCTRINE-PROJECT.ORG. *Association Mapping - Object Relational Mapper (ORM) - Doctrine* [online]. 2019 [cit. 2019-11-13]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/association-mapping.html`.

42. DOCTRINE-PROJECT.ORG. *Inheritance Mapping - Object Relational Mapper (ORM) - Doctrine* [online]. 2019 [cit. 2019-11-13]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/inheritance-mapping.html`.

43. DOCTRINE-PROJECT.ORG. *Introduction - Doctrine Migrations* [online]. 2019 [cit. 2019-12-05]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-migrations/en/2.2/reference/introduction.html#introduction`.

44. NETTE FOUNDATION. *Přihlašování & oprávnění uživatelů | Nette Framework* [online]. 2019 [cit. 2019-11-16]. Dostupné z: `https://doc.nette.org/cs/2.4/access-control`.

45. MAREK JANČA. *Webpack - moderní Web Development | Ackee blog* [online]. 2017 [cit. 2019-11-17]. Dostupné z: `https://www.ackee.cz/blog/moderni-web-development-webpack/`.

46. NPMJS.COM. *About npm | npm Documentation* [online]. 2019 [cit. 2019-11-17]. Dostupné z: `https://docs.npmjs.com/about-npm/`.

47. NPMJS.COM. *Creating a package.json file | npm Documentation* [online]. 2019 [cit. 2019-11-17]. Dostupné z: `https://docs.npmjs.com/creating-a-package-json-file`.

48. SOFTWARE TESTING FUNDAMENTALS. *Unit Testing - Software Testing Fundamentals* [online]. 2019 [cit. 2019-12-09]. Dostupné z: `http://softwaretestingfundamentals.com/unit-testing/`.

49. SOFTWARE TESTING FUNDAMENTALS. *Integration Testing - Software Testing Fundamentals* [online]. 2019 [cit. 2019-12-09]. Dostupné z: `http://softwaretestingfundamentals.com/integration-testing/`.

50. PAVLA LICHNOVSKÁ A EVA KARBEROVÁ. *Heuristická analýza :: Testování a hodnocení rozhraní* [online]. 2019 [cit. 2019-12-11]. Dostupné z: `https://human-computer-interaction.webnode.cz/testovani-a-hodnoceni-rozhrani/metody-testovani/heuristicka-analyza/`.

51. JAKOB NIELSEN. *How Many Test Users in a Usability Study?* [online]. 2002 [cit. 2019-12-11]. Dostupné z: `https://www.nngroup.com/articles/how-many-test-users/`.

Příloha A

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

CRUD Create-Read-Update-Delete

CSS Cascading Style Sheets

CSV Comma-separated values

DBAL Database Abstraction Layer

DBMS Database Management System

DOM Document Object Model

DQL Doctrine Query Language

ES6 Ecma Script 6

GPL GNU General Public License

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JS JavaScript

JSON JavaScript Object Notation

LAMPP Linux + Apache + MariaDB + PHP + Pearl

MVC Model-View-Controller

MVP Model-View-Presenter

NPM Node Package Manager

ORM Object Relational Mapping

PDF Portable Document Format

PHP PHP: Hypertext Preprocessor

SASS Syntactically Awesome Style Sheets

SQL Structured Query Language

URL Uniform Resource Locator

XML Extensible Markup Language

XSS Cross-site scripting

SEO Search Engine Optimization

Příloha B

Obsah přiloženého CD

/
- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF