FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání diplomové práce

| | |
|---|---|
| **Název:** | dbs.fit.cvut.cz – testy |
| **Student:** | Bc. Radoslav Hašek |
| **Vedoucí:** | Ing. Jiří Hunka |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2023/2024 |

## Pokyny pro vypracování

Portál dbs.fit.cvut.cz, dále jen portál, momentálně prochází přechodem od těžko udržitelné architektury monolitu k mikroslužbám navrženým Bc. Andriiem Plyskachem. Cílem této práce je s přihlédnutím k tomuto návrhu architektury, tedy s ohledem na modularitu, a implementovat tvorbu testových šablon a psaní testů. Dále bude potřeba nové moduly integrovat se zbývajícími aktuálně vyvíjenými částmi testového modulu a celého portálu.
Postupujte v těchto krocích:
1. Analyzujte stávající testový modul a plánovanou architekturu aktuálně vyvíjeného portálu.
2. Navrhněte nový systém pro psaní a vytváření testů (včetně vytváření testových šablon) v souladu s novou architekturou.
3. Implementujte systém dle návrhu.
4. Při vývoji řádně testujte. Navrhněte a aplikujte vhodné testy s ohledem na budoucí vývoj této části backendu.
5. Vytvořený systém integrujte se zbytkem nového portálu, zaměřte se na celkovou funkčnost testové části.
6. Proveďte zkušební nasazení.
7. Navrhněte budoucí směr rozvoje, shrňte dosažené výsledky.

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2023 v Praze.*

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

# dbs.fit.cvut.cz – testy

***Bc. Radoslav Hašek***

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

19. června 2023

# Poděkování

Děkuji Ing. Jiřímu Hunkovi na vedení a možnost práce na zajímavém projektu. Také bych chtěl poděkovat všem kolegům, se kterými jsem na vývoji DBS portálu spolupracoval (Ing. Andrii Plyskach, Jakub Pavličko, Tomáš Douba, Bc. Max Hejda, Dana Schomelová i další), a své rodině, která mě při studiu vždy podporovala.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, 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 19. června 2023 . . . . . .

### Odkaz na tuto práci

Hašek, Radoslav. *dbs.fit.cvut.cz – testy*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

# Abstrakt

Tato diplomová práce se zabývá tvorbou backendové části testového modulu nového portálu dbs.fit.cvut.cz, který je vyvíjen již delší dobu. Konkrétně řeší systém pro tvorbu testových šablon, systém pro průběh testu a část ohledně posílání notifikací. Kromě toho popisuje i integraci těchto součástí s ostatními nově vznikajícími částmi testového modulu a s celým DBS portálem. Práce pokrývá celý vývojový cyklus softwaru včetně analýzy současného testového modulu, identifikování požadavků, vytvoření návrhu s ohledem na dříve zavedenou architekturu mikroslužeb, popisu samotné implementace až po testování a zkušební nasazení. Mezi použité technologie patří programovací jazyk PHP a framework Symfony, které jsou stanoveny jako standard pro celý portál. Na konci je nastíněn možný budoucí vývoj a jsou shrnuty dosažené výsledky.

**Klíčová slova** DBS portál, PHP, Symfony, webová aplikace, backend, mikroslužby, integrace

# Abstract

This master's thesis deals with the development of the backend part of the test module for the new dbs.fit.cvut.cz portal, which has been under development for some time. Specifically, it addresses the system for creating test templates, the system for tests management, and sending notifications. Additionally, it describes the integration of these components with other parts of the test module that are currently being developed and with the entire DBS portal. The thesis covers the entire software development life cycle, including analysis of the current test module, requirement identification, design that complies with the previously established microservices architecture, description of the implementation itself, testing and deployment on a test server. Used technologies include the PHP programming language and the Symfony framework, which are defined as the standard for the entire portal. At the end, there is an outline of a possible future development and a summary of the achieved results.

**Keywords** DBS portal, PHP, Symfony, web application, backend, microservices, integration

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Valná většina studentů Fakulty informačních technologií Českého vysokého učení technického v Praze zná portál dbs.fit.cvut.cz (dále též jako portál) a ví, že se v něm mimo jiné píší zápočtové a zkouškové testy v předmětu Databázové systémy. Daleko méně jich už ale ví, jak portál funguje na pozadí, a že momentálně se vyvíjí úplně nová verze, která bude používat mírně odlišné technologie a hlavně bude postavena okolo úplně jiné architektury.

Cílem této práce je zapojit se do tohoto procesu tím, že vyvine nový backend testového modulu, konkrétně část pro správu testových šablon a pro správu a psaní testů. Aby byly tyto části po dokončení použitelné, je do rozsahu zahrnut i modul starající se o správu notifikací. Bude ukázán celý proces softwarového vývoje od analýzy a návrhu (který bude proveden s ohledem na architekturu nového portálu), přes implementaci a testování, až po zkušební nasazení. Bude ovšem potřeba dbát i na integraci se zbytkem aplikace, novým frontendem, jehož část je obsahem bakalářské práce Dany Suchomelové, i ostatními částmi testového modulu, jež vznikají jako bakalářské práce kolegů Tomáše Douby a Jakuba Pavlička. Tato práce má za úkol i zajistit celkovou funkčnost zmíněné části portálu.

## Struktura práce

Text je členěn dle fází procesu vývoje softwaru. Níže v tomto úvodu se nachází stručný popis DBS portálu, jeho funkcionalit a historie vývoje. V první kapitole samotného textu je podrobná analýza současného stavu testového modulu a správy notifikací včetně identifikování aktérů, případů užití a popisu architektury. Dále se zde nachází funkční a nefunkční požadavky na nově vyvíjenou verzi. Návrh je rozdělen podle systémů, kterých se práce týká. Ke každému patří popis plánovaných odlišností oproti stávajícímu stavu, návrh databáze a API. Kromě toho jsou zmíněny použité technologie, celková architektura a způsob zajištění funkčnosti s ostatními částmi portálu. Následuje kapitola

popisující podobu implementace a některá z použitých technických řešení. Poté je uveden způsob testování (druh použitých testů, pokrytí kódu atd.) a průběh zkušebního nasazení. Na závěr je nastíněn možný budoucí vývoj.

## DBS portál

Portál DBS slouží na Fakultě informačních technologií ČVUT jako nástroj pro podporu a automatizaci předmětu Databázové systémy (BI-DBS), který je povinný pro celý bakalářský studijní program. S touto aplikací se tedy během svého studia setkají všichni studenti tohoto programu.

Návrh na vývoj nástrojů, které by výuku Databázových systémů usnadnily, pochází už z roku 2013 [1]. Nejprve vznikaly systémy pro automatizovanou kontrolu semestrálních prací nebo např. překladač z relační algebry do SQL. Celý portál byl poprvé použit ve výuce v omezené míře v roce 2016, o rok později ho už používali všichni studenti, kteří měli zapsaný předmět BI-DBS. V dnešní době portál podporuje kromě psaní a kontroly již zmíněné semestrální práce také psaní a (částečně automatické) opravování semestrálních a zkouškových testů, modelování konceptuálního schématu, pouštění skriptů a dotazů nad databázemi, notifikace o hodnocení atd.

Projekt je od svého začátku zaštiťován Ing. Jiřím Hunkou z Katedry softwarového inženýrství, pod kterou předmět spadá. Na portálu se za oněch téměř 10 let vystřídalo velké množství vývojářů. Jednalo se výhradně o studenty fakulty, z nichž většina tuto práci odváděla v rámci předmětů Softwarový týmový projekt 1 a 2 (BI-SP1 a BI-SP2), které jsou povinné pro specializaci Softwarové inženýrství. Kromě toho na DBS portálu vznikla řada bakalářských i diplomových prací jako je například tato.

V současné době je aplikace realizovaná v jazyce PHP s použitím frameworku Nette a monolitické architektury. O co se jedná, a jaké to s sebou nese výhody a nevýhody, je popsáno dále v sekci 1.4.2. Jak již bylo řečeno, pracuje se na velkých změnách v architektuře jak na straně backendu, tak na frontendu; např. Ing. Andrii Plyskach se ve své diplomové práci [2] zaměřuje na to, jak zajistit modernizaci celého projektu s ohledem na budoucí udržitelnost. Moje práce je součástí těchto změn a navazuje na některé poznatky a návrhy Andriiho.

Kapitola 1

# Analýza

Analýza je velmi důležitý krok softwarového procesu, který se odehrává převážně na počátku vývoje (v agilních metodikách téměř po celou dobu, ale stále je jí více na počátku). Je nutné seznámit se s doménou a porozumět ji, aktivně komunikovat se zadavatelem potažmo zákazníkem, identifikovat požadavky, aktéry, případy užití atd. Opomenutí nějaké části analýzy nebo její nedostatečná příprava může mít za důsledek to, že se bude pracovat na něčem jiném, než bylo požadováno [3]. V takové situaci je odvedenou práci nutno upravit nebo i zahodit, nelze totiž očekávat, že by zadavatel akceptoval software, který neřeší jeho problém. A to samozřejmě stojí čas a peníze. Pochopitelně, že dobře provedená analýza nezajistí, že následný návrh bude kvalitní, nebo že implementace nebude obsahovat chyby. Jedná se ale o nutnou podmínku úspěchu. Není ale dobré se u analýzy naopak zaseknout, tomuto fenoménu se říká „analysis paralysis“ [4]. Člověk má někdy pocit, že nemá dost informací, aby začal pracovat na samotné aplikaci. Neustále proto zkoumá daný problém a snaží se ho popsat do nejmenších podrobností. To ale neúměrně prodlužuje dobu strávenou nad analýzou a obvykle takový popis nemá žádnou přidanou hodnotu a zpomaluje práci i v pozdějších fázích projektu.

V rámci této kapitoly bude popsána analýza tak, jak byla provedena na testovém modulu DBS portálu (včetně notifikací). Budou popsány objekty, které se v systému vyskytují a vztahy mezi nimi. Následuje popis aktérů a případů užití při současném stavu. Vzhledem k tomu, že tato práce je součástí projektu přechodu DBS portálu na novou architekturu a nejedná se tedy o vývoj úplně prvního řešení, budou v další části kapitoly zmíněny i některé technické aspekty současného řešení, které budou užitečné v kapitole 2 pro popsání odlišností obou přístupů. Na závěr budou identifikovány a sepsány požadavky na nový systém.

Na úvod je rovněž vhodné říci, že v minulosti již vzniklo několik prací, které popisovaly stejnou doménu, jako například bakalářské práce Bc. Martina Hanzla [5], Bc. Pavla Jordána [6], Ing. Andriiho Plyskache [7] i dalších.

V jejich pracích už byla analýza testového modulu provedena. Avšak tato díla vznikala v různých letech a během té doby se samozřejmě DBS portál vyvíjel. Jednotlivé analýzy se tedy neshodují ve všech detailech se současným stavem ani mezi sebou navzájem. Aspekty, které jsou stále stejné, nemusí být nutně v této kapitole popsány do úplných detailů. Větší důraz bude kladen na odlišnosti.

## 1.1 Současný testový modul

Testový modul v současném systému slouží k tomu, aby bylo možné řešit veškerou agendu spojenou se semestrálními a zkouškovými testy v online formě. Je podporován celý proces, což znamená, že učitelé mohou testy vytvořit, přiřadit je studentům a zorganizovat termíny, kdy se testy budou psát. Studentům je umožněno test napsat na počítači a učitelé ho jsou následně schopni opravit a obodovat. Studenti jsou poté upozorněni notifikací na opravený test a jeho hodnocení je jim zobrazeno.

Tato práce se sice konkrétně zabývá systémy pro tvorbu testových šablon, pro průběh testu a pro správu notifikací, nejdříve je ale třeba představit si dva pojmy, které jsou pro testy velmi důležité - *zadání* a *otázka*. Tyto objekty vytváří učitelé a dají se použít opakovaně v různých testech. Výše uvedené systémy s nimi musí pracovat minimálně na úrovni identifikátorů entit, které budou tyto objekty reprezentovat v současně vznikajícím novém systému pro správu zadání, na kterém pracuje Tomáš Douba.

Zadání se dá popsat jako problém, který je studentovi představen, nepatří k němu ale žádné úkoly, ty zahrnuje otázka postavená okolo tohoto zadání. Jako většinu entit, je i zadání možné duplikovat a upravit. Zadání je několik typů:

- Text
- Obrázek
- Diagram – popis relačního modelu zobrazovaný v nástroji Data modeller, který rovněž vznikl během práce na DBS portálu [8].
- Normalizace relačního schématu – poskytuje relace a funkční závislosti, na kterých se má provést normalizace. Tento typ zadání se dá v současné době generovat na základě parametrů, jako je počet relací.
- Databáze – zvláštní typ zadání, které není zobrazováno studentovi. Slouží k uchování databázového připojení, pokud ho otázka potřebuje. Dříve se připojení vázalo na odpověď místo na zadání, s návrhem na změnu se přišlo v práci Andriiho Plyskache [7] a počítá se s jeho realizací v novém testovém modulu.

Otázka už dává studentovi úkoly nad zadáním, které je jí přiřazeno. Zadání může být i více v jedné otázce. Aby byla otázka validní a dala se použít, musí mít minimálně jedno zadání a minimálně jednu referenční odpověď (může mít i další odpovědi, které byly označeny jako správné při opravování). Mezi povinné údaje o otázce patří název, textový popis úkolu, kategorie (teorie, model, SQL a relační algebra), obtížnost, použitelnost v demo testu nebo typ. Typů je opět více:

- Text
- Radio list – výběr jedna z n.
- Checkbox – výběr nula až n.
- Diagram – odpovědí je diagram relačního schéma, opět namodelovaný pomocí Data modelleru.
- SQL – odpovědí je dotaz v SQL, studentovi je při odpovídání kontrolována syntaxe.
- RA – podobný princip jako u SQL, ale dotaz je psán v relační algebře.
- Transformace – textový převod konceptuálního schématu na relační.
- Normalizace relačního schématu - student zaškrtává check boxy a vyplňuje formulář s otázkami ohledně funkčních závislostí atd.

Nyní už přichází na řadu pojmy, se kterými pracují zde vyvíjené systémy. Jedná se o *testovou šablonu*, *testový termín*, *instanci testu* (dále v textu je popisována *varianta testu*, což je úprava tohoto objektu), *studentský test* a *notifikaci*.

Testová šablona je vzor pro instanci testu. Má svůj název, časové okno pro celý test, čas na vyplnění (počítá se od chvíle, kdy student začne vyplňovat, ale nemůže překročit konec okna pro celý test), typ (test, zkouška, demotest a automatický demo test) a maximální počet bodů, který může student získat. Význam typu šablony bude patrný z popisu instance testu. Kromě toho jsou k šabloně přiřazeny otázky. Otázka se dají přiřazovat manuálně z tabulky, v takovém případě je potřeba u každé z nich nastavit počet bodů a učitele, který bude otázku opravovat v případě nutnosti manuální opravy. Pokud je typ automatický demo test, přiřazují se otázky automaticky po nastavení počtu požadovaných otázek v jednotlivých kategoriích. Šablona včetně výběru otázek a změny jejich pořadí se dá upravovat do chvíle, než je z ní poprvé vytvořena instance testu. Od té doby se dá pouze duplikovat, aby se zabránilo nekonzistenci mezi šablonou a již proběhlými testy. Šablona se dá použít pouze, pokud je validní. To znamená, že součet bodů za otázky se rovná nastavenému maximálnímu počtu bodů a všechny otázky jsou validní. Druhá podmínka vypadá triviálně, ale otázka se může dodatečně stát neplatnou třeba, pokud přestane

fungovat databázové připojení, na kterém je závislá. Pokud už byla šablona použita, učitelé mohou sledovat statistiky úspěšnosti studentů, kteří psali testy z ní vytvořené. Tato statistika se dá zobrazit i pro jednotlivé otázky (které mohou být vloženy do více šablon).

Testový termín je objekt, který reprezentuje zkoušku vypsanou v systému KOS (komponenta studium ČVUT) a studenty, kteří jsou na ni přihlášeni. Uplatňuje se při vytváření instance testu ze šablony s typem zkouška nebo test.

Jak bylo řečeno, instance testu se vytváří z platné testové šablony. Po vytvoření je nutné přiřadit studenty. To se dá dělat jednotlivě, po paralelkách nebo se dá využít testového termínu a přiřadit všechny studenty zapsané na akci v KOSu. Pokud se jedná o zkoušku, nastavují se i kapacity místností. Do těch jsou pak studenti přiřazeni. Také se nastavuje vynucení použití speciálního Progtest image na počítačích, na kterých se bude test psát, a fyzická kontrola studentských karet. Současný systém umožňuje učiteli při spuštěné instanci testu přidávat čas všem studentům, uzavírat dosud nespuštěné studentské testy, uzavírat konkrétní test (třeba v případě podvádění) a zobrazovat si log IP adres použitých pro vyplňování. Pokud dojde čas daný oknem pro celý test, uzavřou se všechny studentské testy, nehledě na to, kolik jim zbývá času.

Studentský test je na rozdíl od výše uvedených pojmů, které jsou důležité hlavně pro učitele, klíčový pro studenty. Skrz něj je možné vyplňovat otázky a případně ukončit test ještě před vypršením času. Při vyplňování otázek typu SQL a RA si může student validovat odpověď z hlediska syntaktické správnosti a ověřit si, že dotaz vrací neprázdnou odpověď. Demo testy jsou specifické, protože student je může opakovat, pokud nevypršelo časové okno na instanci testu, a jejich výsledky se nepočítají do hodnocení. Jakmile je test ukončen, začíná fáze hodnocení.

Notifikací se myslí upozornění uživatele na událost nebo změnu stavu, která se ho týká. Notifikace se nevyskytují pouze v testovém modulu, ale i ve zbytku portálu. Například u semestrálních prací je učitel upozorněn na nové odevzdání kontrolního bodu, student je následně informován o jeho ohodnocení nebo odmítnutí. Důležitější pro tuto práci jsou upozornění týkající se testů. Portál dá studentovi vědět, když je jeho test plně opraven, nebo když je mu zapsáno hodnocení z ústní zkoušky. Učitelé se skrz tento kanál dozví o tom, že v testu je potřeba některé otázky opravit ručně. Notifikace se zobrazují jednak ve webovém rozhraní a jednak přichází uživateli na email, po kliknutí na ni se uživatel dostane do příslušné části portálu. Posílání emailu je možno nastavit, případně úplně vypnout.

Ve výčtu důležitých pojmů nebyl zmíněn *štítek*. Ten se v současném DBS portálu ještě nenachází, avšak počítá se s jeho použitím v nové verzi, a to skrze několik systémů. Popsán bude v kapitole 2, zde je jen pro úplnost.

Zde končí popis entit a procesů nacházejících se v současnosti v části systému, jejíž nová implementace je předmětem této práce. Do testového mo-

dulu ale patří i systém pro hodnocení testů, který rovněž vzniká touto dobou, a to v rámci bakalářské práce Jakuba Pavlička. Zaslouží si proto taktéž alespoň stručný popis. Po ukončení testů nastává pokus vyhodnotit otázky automaticky. To lze udělat, pokud se odpověď přesně shoduje s referenční nebo jinou dřívější správnou odpovědí nebo pokud jde o otázku na normalizaci relačního schématu. V opačném případě nastává manuální hodnocení. Jak již bylo řečeno, při přiřazování otázky do šablony je třeba vyplnit opravujícího. Po ukončení testů dostane každý učitel přiřazený k nějaké z použitých otázek notifikaci, kolik otázek mu zbývá opravit manuálně. Učitel může nastavit počet bodů a textový komentář. Student si pak může svá hodnocení prohlédnout, nevidí už ale obsah zadání v otázkách (motivací je snaha omezit šíření otázek mezi studenty). V novém systému se počítá se sofistikovanějším způsobem automatické opravy.

## 1.2 Aktéři

Před psaním případů užití je nutné uvědomit si, jací aktéři se v systému nachází a jak se systémem mohou interagovat. Aktérem nemusí být pouze lidský uživatel, ale i části téhož systému nebo úplně jiné aplikace [3]. Vzhledem k tomu, že práce pojednává o vývoji backendové části systému, veškerá interakce bude uživatelům zprostředkována skrz nově vznikající frontend, který ale v následujícím výčtu chybí. V této fázi je důležitější popsat, jak budou systém používat koncoví uživatelé, což pomůže při návrhu, než rozebírat interakci backendu s frontendem. Aktéři zároveň ani neodpovídají seznamu rolí, které v portálu existují (např. speciální role „root“ nebo nepřihlášený uživatel), ale ostatní role nemají buď do daných systémů přístup nebo se jejich možnosti neliší od níže uvedených.

### 1.2.1 Student

Tuto roli má v DBS portálu jakýkoliv student předmětů BI-DBS, BIK-DBS nebo BIE-DBS (obecně předmětů, pro které se portál pouští). Může zde vidět svoje hodnocení, pracovat na semestrální práci atd. Co se týče interakce se zde popisovanými částmi portálu, student nemá žádným způsobem přístup do systému pro tvorbu testových šablon. V systému pro průběh testu může vidět svoje studentské testy, spustit je (za předpokladu, že nevyplňuje zároveň demotest), zobrazit aktuálně si vyplňovaný test, ukládat odpovědi na otázky a test předčasně ukončit. U již opravených testů si může zobrazit hodnocení jednotlivých otázek, ale v této fázi již nevidí jejich zadání. U demo testů jsou možnosti mírně odlišné, zde jsou zadání otázek vidět po celou dobu a studentovi je umožněno test opakovat. Stejně jako ostatní níže uvedení, dostává i student pro něj určené notifikace.

Student je do portálu importovaný pro určitý semestr. To znamená, že pokud měl některý z předmětů zapsaný již před rokem, nemůže být přiřazen do testů v aktuálním semestru.

### 1.2.2 Učitel

Vyučující daných předmětů mají v portálu podstatně rozsáhlejší oprávnění. Patří mezi ně například hodnocení semestrálních prací, správa zadání a otázek, ale zejména vytváření, duplikace a úprava testových šablon a následné vytváření instancí testů. Do instance mohou učitelé přidávat studenty, nastavovat v ní kapacity místností, instanci spustit a poté přidávat čas, předčasně ukončovat studentské testy a ukončit všechny studentské testy, na kterých se ještě nezačalo pracovat. Následně vypracované testy opravují v závislosti na přiřazení k otázkám na úrovni šablony.

### 1.2.3 Garant

Jedná se o speciální roli v portálu, která náleží garantovi daných předmětů. Ten může kromě všech akcí učitelů ještě upravovat podmínky hodnocení v daném semestru, např. změnit požadovaný počet bodů ze zkoušky.

## 1.3 Současné případy užití

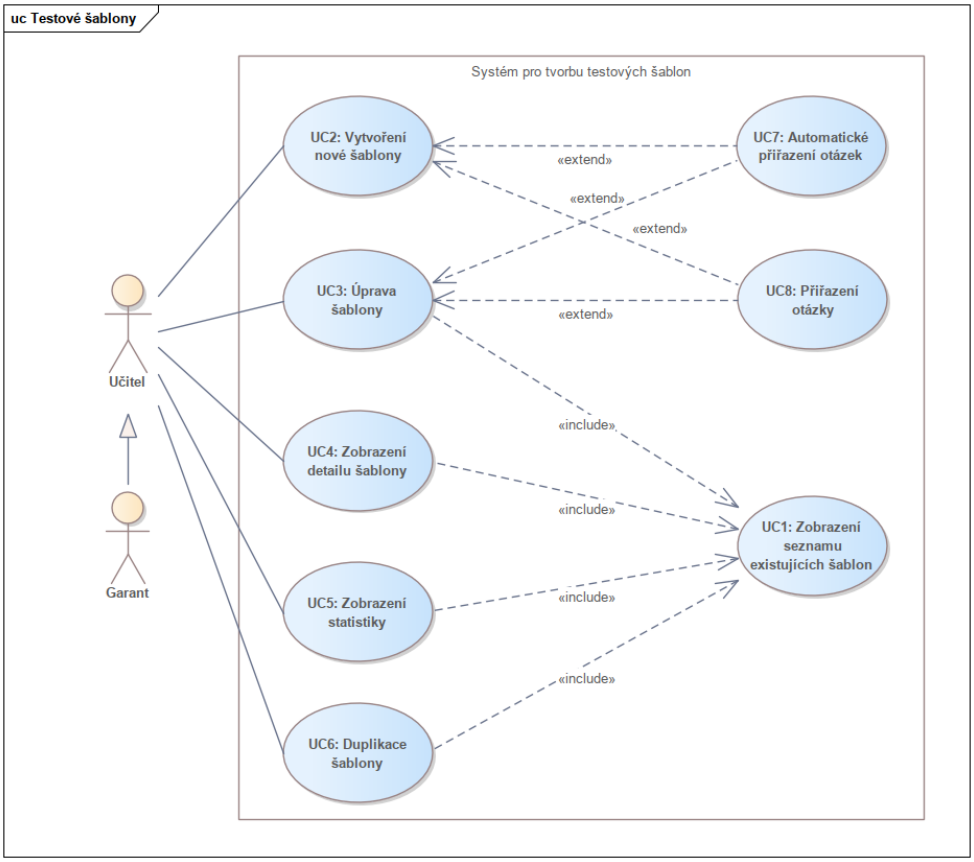
Případ užití (i v češtině se někdy používá anglický termín „use case“) je popis toho, jak uživatel interaguje se systémem [9]. Uživateli jsou aktéři, kteří jsou popsáni výše. Jedná se o velmi důležitý nástroj, se kterým umí pracovat nejen analytici a vývojáři, ale je vypovídající i pro projektové manažery a zákazníka. Každý případ užití by měl kromě názvu obsahovat i seznam aktérů, kteří v něm vystupují, a jeden či více scénářů [10]. Scénář je sekvence akcí a interakcí mezi aktéry a popisovaným systémem. Často se uvádí jako očíslovaný seznam. Scénáře mohou být hlavní a alternativní. Jako hlavní scénář se volí cesta, ve které nenastane žádná chyba nebo nebo kroky specifické pro určitou situaci. Alternativní scénáře popisují odlišnosti od této cesty. Vždy musí být jasné, kde se oddělují a do kterého kroku hlavního scénáře se opět připojují. Jednoduchým příkladem je placení kartou v e-shopu. Zákazník zadá údaje ke kartě, potvrdí formulář, následně potvrdí platbu u své banky a nákup je zaplacen. Pokud je zadaná karta neplatná, oddělí se alternativní scénář, ve kterém je zobrazena chyba a je možno zkusit platbu znovu nebo vybrat jinou platební metodu, případně objednávku zrušit. Z popisu musí být jasné počáteční podmínky, za kterých celý případ use case začíná. Ty mohou být napsány zvlášť nebo jako součást prvního kroku scénáře.

Pro celkový pohled se hodí diagram případů užití, který vizuálně ukazuje interakce mezi aktéry a systémem [9]. V příslušném UML diagramu je systém zobrazen jako obdélník, v němž se nachází názvy jednotlivých případů užití,

mezi kterými mohou být dva vztahy: *include* a *extend* [11]. V obou případech je jeden use case součástí druhého. Rozdíl je zjednodušeně řečeno v povinnosti tohoto vztahu. Pokud je použito *include* z A do B, vložený případ užití B je nedílnou součástí A a je spuštěn vždy. V případě *extends*, nemusí být B spuštěn vždy, ale jen za určitých podmínek. Vně obdélníku jsou aktéři, kteří jsou propojeni s příslušnými use cases. Pokud jeden aktér dědí z druhého, znamená to, že potomek může figurovat ve všech případech užití, ve kterých se vyskytuje jeho rodič.

V pracích mých předchůdců ([6], [7] a [5]) se nachází jak tyto diagramy, tak velmi podrobné popisy jednotlivých případů užití, není účelné je do jednoho přepisovat i do tohoto textu. Tato kapitola bude zejména o srovnání jejich přístupů a případném doplnění a vyjasnění rozdílů. Všechny diagramy byly vytvořeny v nástroji Enterprise Architect a jejich zdrojové soubory jsou k dispozici na přiloženém médiu.

### 1.3.1 Testové šablony




Obrázek 1.1: Diagram případů užití tvorby testových šablon.

Z diagramu 1.1 je vidět, že bylo identifikováno pět základních případů a to *vytvoření nové šablony, úprava šablony, zobrazení detailu, zobrazení statistiky a duplikace šablony.* Součástí prvních dvou jmenovaných může být *automatické přiřazení otázek* nebo (manuální) *přiřazení konkrétní otázky.* U zobrazení detailu, zobrazení statistiky, duplikace a úpravy je prvním krokem ještě *zobrazení seznamu všech existujících šablon*, které se dá i filtrovat.

Jeden z use cases, který se vyskytuje ve všech zmíněných pracích kolegů i na diagramu 1.1 je zde označený jako **UC2: Vytvoření testové šablony**. Je tedy ideálním kandidátem pro srovnání stylu popisu. Pavel Jordán a Andrii Plyskach popsali scénář velmi podobně, nemá tedy cenu uvádět dvě citace, popis Martina Hanzla se mírně odlišuje. Následují přímé citace ze dvou prací, první zní:

***UC1: Vytvoření testové šablony***
*Případ užití umožní uživateli vytvořit novou testovou šablonu, kterou lze následně využít pro vytvoření testů.*
***Aktéři:*** *vyučující, garant.*
***Scénář:***

1. *Případ užití začne, když uživatel chce vytvořit novou testovou šablonu a v levém menu klikne na „Vytvořit testovou šablonu.“*

2. *Systém zobrazí formulář s údaji: název, čas na vypracování, časové okno pro celý test, typ testu a maximální počet bodů.*

3. *Uživatel formulář vyplní a zvolí „Automatické generování otázek“ nebo „Manuální výběr“.*

4. *Systém uloží šablonu a přejde do „UC2: Automatické generování otázek“ nebo „UC3: Manuální výběr otázek“ v závislosti na volbě uživatele. [6]*

A druhá ukázka:

***UC5: Tvorba testové šablony***
*Testovou šablonu vytvářejí aktéři s právy alespoň jako vyučující, později z této šablony mohou vytvořit skutečný test, který budou vyplňovat studenti. Testová šablona obsahuje základní údaje o testu a také drží množinu otazek, které budou studenti zodpovídat v rámci psání testu vytvořeného z téže šablony. Šablona také drží informace o tom, kdo bude jakou otázku opravovat v tomto testu a kolika body bude otázka hodnocena.*
***Aktéři:*** *Vyučující, Garant*
***Scénář:***

1. *Aktér si v testovém modulu porálu zvolí v postraní liště položku Tvorba testové šablony.*

2. *Systém aktérovi zobrazí formulář pro vyplnění s položkami Název, Typ, Čas na vypracování, Celkový čas, Celkový počet bodů.*

3. *Aktér vyplní formulář a uloží kliknutím na tlačítko „Uložit“. Aktér může dále automaticky vygenerovat otázky do testu nebo přejít na Manuální přiřazování (UC6).*

4. *Po uložení je šablona k dispozici ve správě šablon, nicméně nemusí být validní. Pokud validní je, zle z ní vytvořit instanci testu. [5]*

Oba popisy jsou jasné a scénář není nikterak komplikovaný. Ovšem i tak lze najít rozdíly. Druhá varianta je o něco obsáhlejší, protože na začátku rekapituluje význam šablony v systému a zmiňuje některé další detaily jako je poznámka v závěrečném kroku, že vytvořená šablona nemusí být nutně použitelná, pokud není validní. Věcně i formátem jsou obě varianty správně a nevynechávají nic podstatného.

Tento úvodní příklad posloužil jako demonstrace toho, že stejný případ užití může mít různé reprezentace v závislosti na tom, co považuje autor za vhodné zmínit.

Co se týče celého diagramu, tak v práci Martina Hanzla je část o testových šablonách velmi stručná, obsahuje pouze tvorbu šablony, jejíž součástí je přiřazení otázky, vynechává tedy možnosti úpravy, duplikace atd. Diagramy ostatní dvou kolegů se liší zejména v orientaci šipek u vztahu *extends*. Správný směr je vidět u Andriiho Plyskache [7].

V jeho diagramu je ještě navíc případ *vytvoření instance testu*. Ten byl v této práci zařazen až do diagramu popisující správu testů. Jedná se sice o operaci se šablonou, ale vzniká tím instance, se kterou se dá následně manipulovat. Logicky se dá tím pádem zařadit do obou skupin, zde ale převážil druhý argument.

V diagramu vytvořeném v této práci přibylo *zobrazení seznamu existujících šablon*, které je v pracích kolegů zmíněné jen implicitně jako první krok některých scénářů. Tím pádem ale není řečeno, že v seznamu je možno používat filtry.

**UC1: Zobrazení seznamu existujících šablon**
Učitel, potažmo garant, si může zobrazit seznam testových šablon, které již byly vytvořeny. Lze aplikovat filtry, které tento seznam omezí. Pro každou šablonu z výběru se dá přejít do některých z ostatních scénářů.
**Aktéři:** Vyučující, Garant
**Scénář:**

1. Aktér klikne na tlačítko „Testové šablony“.

2. Systém aktérovi zobrazí stránkovaný seznam všech testových šablon nacházejících se v databázi. U každé šablony se na základě jejího stavu

zobrazí tlačítka, která zahajují „UC3: Úprava šablony“, „UC4: Zobrazení detailu šablony“, „UC5: Zobrazení statistiky“ nebo „UC6: Duplikace šablony“ nebo šablonu smažou.

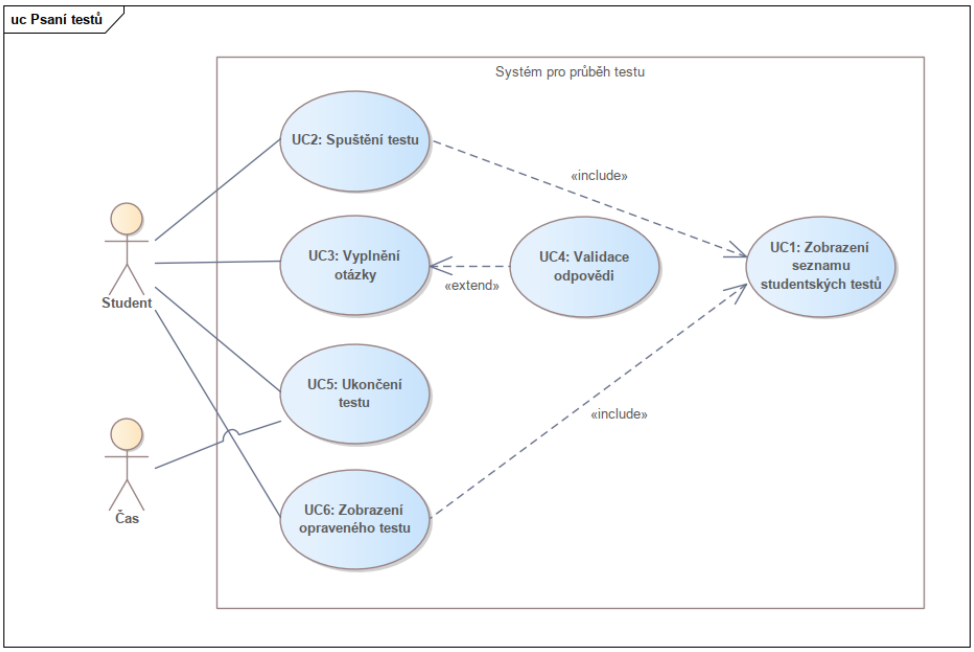
3. Volitelně aktér vyplní některé z filtrů. Atributy, které mohou sloužit pro filtrování, zahrnují všechny parametry šablony vyplňované při jejím zakládání, autora a přítomnost validní otázky.

4. Po vyplnění systém aktualizuje zobrazen seznam tak, aby obsahoval pouze šablony, které splňují všechny filtry.

Šablonu lze i smazat, pokud ještě nebyla použita pro vytvoření instance testu. Tuto akci zde nepopisuje žádný případ užití, jedná se totiž o jednoduchý jednokrokový scénář a jeho rozepisování nemá žádnou přidanou hodnotu.

### 1.3.2 Průběh testu

Případy užití týkající se samotných testů jsou pro přehlednost rozděleny do dvou diagramů. Diagram 1.3 ukazuje hlavně, jak s testy pracují učitelé, diagram 1.2 potom popisuje, jak s nimi manipulují studenti.




Obrázek 1.2: Diagram případů užití psaní testů.

V diagramu use cases týkajících se psaní testu studentem lze vidět celý proces psaní testu, to znamená *spuštění testu, vyplnění jedné otázky* (v závislosti na typu otázky může být součástí i základní *validace odpovědi*) a *ukončení*

*testu* ať už studentem, nebo na základě vypršení času. Nachází se zde i *zobrazení opraveného testu*, které stejně jako spuštění zahrnuje i *zobrazení seznamu studentských testů*, do kterých je daný student přiřazen. Vyhodnocování napsaných testů sice není součástí popisované části systému, následné zobrazení do ní ale logicky náleží.

Andrii Plyskach popsal tuto část systému stručně s pomocí tří případů užití (otevření testu, vyplňování a dokončení). Diagram v práci Martina Hanzla je už podobnější tomu, který je uveden v tomto textu, jelikož zahrnuje i možnou validaci odpovědi. Martin uvedl samostatně i uložení odpovědi, zde bude tato akce považována za součást vyplnění, jelikož vyplnění bez uložení nepřináší samo o sobě aktérovi žádný užitek.

Ani jeden z kolegů nezahrnul zobrazení opraveného testu do této skupiny případů užití, nýbrž ho buď v práci neuvedli, nebo z něj udělali samostatnou kategorii a rozpadli ho na několik use cases. Uvedení si konkrétně zde zaslouží, protože tato práce se nezabývá systémem pro hodnocení testů a nebude ho podrobně analyzovat, není tedy možnost ho dát jinam.

V této práci nachází je uvažován i nový aktér - čas. Andrii možnost vypršení času na test zmiňuje v jiných částech své práce, ve scénáři o ukončení testu nikoliv, Martin potom uvedl jako podmínku příslušného případu užití, že čas ještě nevypršel. Čas je ale v tomto diagramu validní aktér [12] a dobře ilustruje obě situace, kdy může dojít k ukončení testu.

V diagramu 1.2 také přibylo zobrazení seznamu dostupných testů, které je spolu s notifikacemi primárním způsobem, jak se student může dostat k vyplňování nebo zobrazení výsledků.

Následuje popis těch případů užití, kterých se týkají výše uvedené rozdíly.

**UC1: Zobrazení seznamu studentských testů**
Student si může zobrazit seznam testů (nebo demotestů), do kterých je přiřazen. Pokud je test aktivní, dá se přejít na jeho vyplňování (nebo opakovat demotest). U již opravených testů je poté možno si zobrazit náhled.
**Aktéři:** Student
**Scénář:**

1. Aktér klikne v hlavní nabídce na tlačítko „Testy“.

2. Systém zobrazí stránku s postranní nabídkou, ve které jsou na výběr „Testy“ a „Demotesty“.

3. Aktér klikne na jedno z těchto dvou tlačítek.

4. Systém zobrazí stránkovaný seznam dostupných a ukončených testů z dané kategorie. U testů, které je možno spustit, se zobrazí tlačítko „Spustit test“ nebo „Opakovat demotest“, kterým může aktér spustit „UC2: Spuštění testu“. U vyhodnocených testů se pak bude nacházet

tlačítko „Náhled“, kterým může započít „UC6: Zobrazení opraveného testu“.

**UC3: Vyplnění otázky**

Během práce na testu vyplňuje student jednotlivé otázky. Příslušný formulář se liší v závislosti na typu odpovědi u dané otázky. Odpověď může být validována a poté je uložena.

**Podmínka:** Čas na vypracování testu ani časové okno pro celou instanci testu ještě nevypršelo. Student již dokončil „UC2: Spuštění testu“

**Aktéři:** Student

**Hlavní scénář:** první vyplnění otázky

1. Aktér klikne na číslo otázky.
2. Systém zobrazí zadání, úkol a prázdný formulář pro konkrétní typ odpovědi.
3. Aktér vyplní formulář. V případě, že se jedná o odpověď typu RA nebo SQL, může kliknout na tlačítko „Ověřit“, čímž zahájí „UC4: Validace odpovědi“. Jakmile je aktér s odpovědí spokojen, klikne na tlačítko „Uložit“.
4. Systém uloží odpověď do databáze.

**Alternativní scénář:** změna dříve uložené odpovědi

1. Scénář začíná v kroku 2 hlavního scénáře v případě, že aktér již odpověď na otázku uložil a nyní se k otázce vrátil. Systém v tomto případě zobrazí formulář, ve kterém již je vyplněna předchozí odpověď. Pokračuje se krokem 3 hlavního scénáře.

**UC5: Ukončení testu**

Jakmile vyprší čas, nebo pokud již student nechce dále odpovídat a ukončí test manuálně, nastává fáze hodnocení.

**Aktéři:** Student, Čas

**Podmínka:** Existuje test, který student aktuálně vyplňuje.

**Hlavní scénář:** ukončení studentem

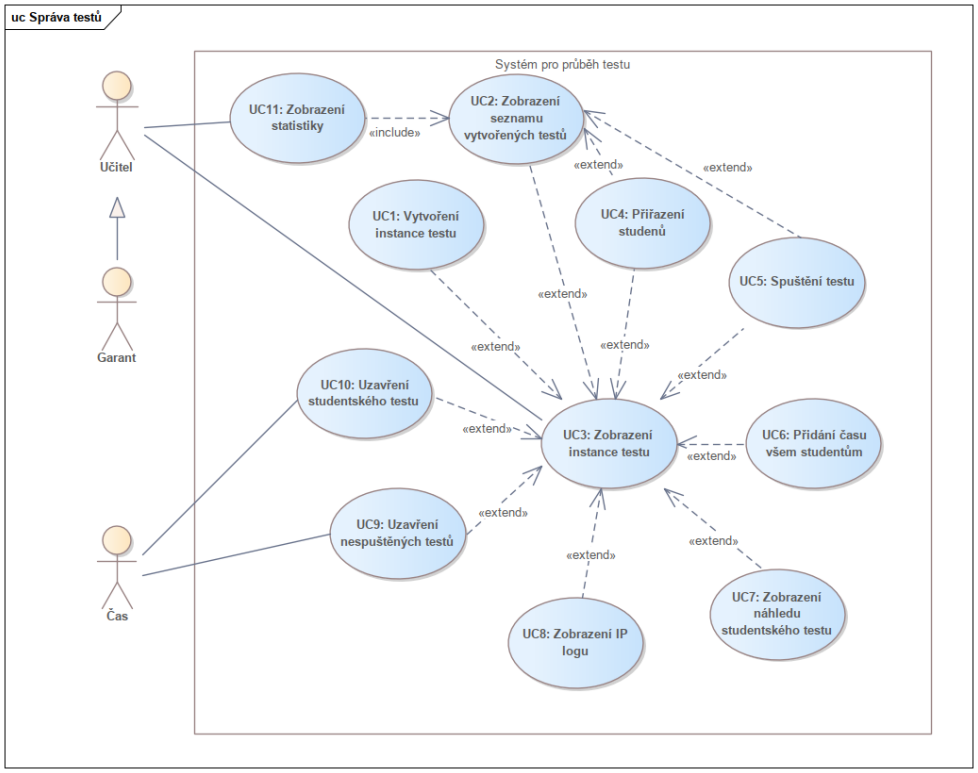
1. Student klikne na tlačítko „Ukončit test“ na přehledu testu.
2. Systém ukončí test a student již nemůže odpovídat na otázky.
3. Test je odeslán k ohodnocení.

**Alternativní scénář:** vypršení času

1. Pokud student test neukončil sám, ale vypršel čas nebo skončilo časové okno pro celou instanci testu, provede systém ukončení sám. Pokračuje se krokem 3 hlavního scénáře.

**UC6: Zobrazení opraveného testu**
Jakmile je test ohodnocen (ať už manuálně, nebo automaticky), může si student zobrazit náhled, ve kterém uvidí hodnocení jednotlivých otázek.
**Podmínka:** Existuje alespoň jeden opravený test přiřazený studentovi.
**Aktéři:** Student
**Scénář:**

1. Aktér buď klikne na notifikaci o opravení testu, nebo na tlačítko „Náhled“ v „UC1: Zobrazení seznamu studentských testů“.
2. Systém zobrazí tabulku, ve které je vidět seznam otázek. U demo testů je zobrazena jen jejich kategorie a typ odpovědi. U ostatních testů i získané hodnocení, opravující atd.
3. U demotestů může aktér kliknout na tlačítko „Náhled“ u některé z otázek.
4. Systém zobrazí zadání otázky, úkol, studentovu odpověď a její hodnocení spolu s tlačítkem pro návrat na seznam otázek.




Obrázek 1.3: Diagram případů užití správy testů.

Problematika správy testů je z hlediska případů užití nejsložitější, jak lze vidět už při pohledu na diagram 1.3, a proto byl její popis ponechán na

konec. Ústředním use case je *zobrazení instance testu*, v rámci něho se dá provádět zbytek činností. *Vytvoření instance testu* a *zobrazení seznamu vytvořených testů* jsou způsoby, jak se do zobrazení instance dostat. V rámci zobrazení poté může učitel manipulovat s testem, konkrétně lze provádět *přiřazení studentů, spuštění testu, přidání času všem studentům, zobrazení náhledu studentského testu, zobrazení IP logu* a *uzavření nespuštěných studentských testů* nebo *konkrétního studentského testu*. Poslední dva jmenované případy užití spouští kromě učitele i vypršení času. Asociace s učitelem patří ke všem případům na diagramu, pro zachování přehlednosti je ale znázorněna jen někde. Jedinou akcí, která se provádí ze seznamu všech vytvořených instancí je *zobrazení statistiky*, které rovněž spouští učitel. Po přečtení předchozí části kapitoly se může zdát, že chybí nastavování kapacity místností. Zde je ale zahrnuto ve vytváření instance testu jako jeden krok.

Popis této části DBS portálu není předmětem prací Pavla Jordána ani Martina Hanzla, srovnání se tedy nabízí s Andriim Plyskachem. Při pohledu na jím vytvořený diagram je vidět shoda v základní struktuře. Zobrazení instance bylo shodně identifikováno jako základní use case.

Andrii popsal navíc odstranění instance testu a přidání času ke konkrétnímu studentskému testu. Odstranění bylo vynecháno ze stejných důvodů jako v případě odstranění testové šablony. Co se týče přidávání času konkrétnímu studentovi, v době psaní této práce se tato funkcionalita v současném systému nenachází, čas lze přidat pouze všem studentům najednou.

Co je naopak navíc, je vytvoření instance testu, které je oproti Andriiho diagramům z již vysvětlených důvodů přesunuto ze správy šablon. Dále je zde vidět zobrazení seznamu vytvořených testů, které je vhodné zmínit samostatně, protože je nutné pro zobrazení statistiky a je to jeden ze způsobů, jak spustit test, dostat se do detailu instance a do přiřazování studentů.

Posledním rozdílem je přítomnost času jakožto aktéra, který hraje stejnou roli jako v případě předchozího diagramu. Zde ukončuje doposud nespuštěné studentské testy nebo konkrétní test.

**UC2: Zobrazení seznamu vytvořených testů**
Učitel si může zobrazit seznam vytvořených instancí testů. V závislosti na stavu testu je možné provádět různé akce.
**Aktéři:** Učitel, Garant
**Scénář:**

1. Aktér klikne v postranní nabídce na tlačítko „Vytvořené testy“.

2. Systém aktérovi zobrazí stránkovaný seznam všech instancí testů nacházejících se v databázi. U každé instance se na základě jejího stavu zobrazí tlačítka, která zahajují „UC3: Zobrazení instance testu“, „UC4: Přiřazení studentů“ nebo „UC5: Spuštění testu“.

3. Volitelně aktér vyplní některé z filtrů. Atributy, které mohou sloužit pro filtrování, zahrnují název použité šablony, autora, čas spuštění nebo stav.

4. Po vyplnění systém aktualizuje zobrazený seznam tak, aby obsahoval pouze instance, které splňují všechny filtry.

**UC9: Uzavření nespuštěných testů**
Jakmile vyprší čas, nebo pokud učitel už nechce umožnit studentům, kteří dosud nespustili test, aby začali pracovat, ukončí se všechny dosud nespuštěné testy. Tyto testy budou hodnoceny 0 body.
**Aktéři:** Učitel, Garant, Čas
**Hlavní scénář:** ukončení učitelem

1. Učitel se nachází v detailu testu (z „UC3: Zobrazení instance testu“) a klikne na tlačítko „Ukončit nespuštěné“.

2. Systém ukončí všechny testy, který nebyly doposud spuštěny. Daní studenti již nemohou test spustit.

**Alternativní scénář:** vypršení času

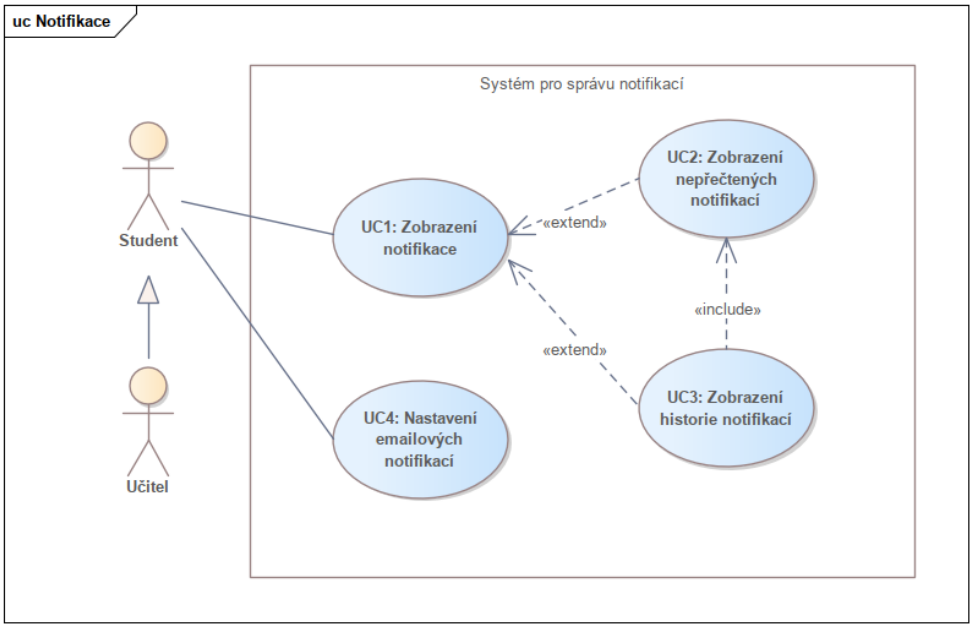
1. Pokud vyprší časové okno pro celý test, provede systém ukončení všech nespuštěných testů sám.

### 1.3.3 Notifikace

Poslední analyzovanou částí je systém pro správu notifikací, jeho případy užití jsou zobrazené na diagramu 1.4. Tato část portálu nebyla analyzovaná v rámci předchozích prací, proto zde bude popsána kompletně. Základním případem užití je zobrazení konkrétní notifikace, čímž uživatel (jak učitel tak student) přejde na stránku, jíž se upozornění týká (detail testu, opravy atp.). To může zahrnovat zobrazení stručného seznamu posledních notifikací nebo kompletní historie. Kromě toho lze nastavovat preference posílání emailů.

**UC1: Zobrazení notifikace**
Pokud aktérovi přišla nová notifikace, může na ni kliknout a zobrazit si událost, která způsobila její odeslání. Počet nepřečtených notifikací se neustále zobrazuje vedle tlačítka pro rozbalení seznamu upozornění. Alternativní scénáře týkající se semestrálních prací jsou vynechány.
**Aktéři:** Student, Učitel, Garant
**Podmínka:** Aktér má v systému alespoň jednu notifikaci.
**Hlavní scénář:** opravení testu nebo hodnocení ústní zkoušky

1. Aktér klikne na notifikaci buď v rámci „UC2: Zobrazení posledních notifikací“ nebo „UC3: Zobrazení historie notifikací“ nebo klikne na odkaz v emailu, který mu přišel.




Obrázek 1.4: Diagram případů užití správy notifikací.

2. Systém zobrazí stránku se seznamem ukončených testů daného studenta. Z této obrazovky je možné přejít na náhledy těchto testů.

**Alternativní scénář:** manuální oprava testu

1. Scénář začíná v bodě 2 hlavního scénáře. Systém zobrazí seznam otázek z testu, které jsou k opravě přiřazeny danému učiteli. Poté je může učitel postupně ohodnotit.

**UC2: Zobrazení posledních notifikací**

Systém umožňuje rychlý přístup k malému množství posledních notifikací daného uživatele.
**Aktéři:** Student, Učitel, Garant
**Scénář:**

1. Aktér klikne na tlačítko zvonku na horní liště na jakékoliv obrazovce DBS portálu.

2. Systém zobrazí tabulku obsahující detaily několika notifikací, které byly vytvořeny naposledy, tlačítko pro zobrazení celé historie a pro nastavení emailových upozornění.

3. Aktér může kliknout na některou z notifikací, čímž začne „UC1: Zobrazení notifikace“, na tlačítko historie, které spustí „UC1: Zobrazení histo-

rie notifikací“, nebo na tlačítko emailů - začátek „UC4: Nastavení emailových notifikací“.

**UC3: Zobrazení historie notifikací**
Ne všechna upozornění se ukážou v přehledu posledních notifikací. Kompletní historie nabízí pohled do vzdálenější minulosti a umožňuje s notifikacemi více manipulovat.
**Aktéři:** Student, Učitel, Garant
**Scénář:**

1. Scénář začíná posledním krokem „UC2: Zobrazení posledních notifikací“. Aktér klikne na tlačítko „Zobrazit všechny notifikace“.
2. Systém zobrazí stránkovatelný seznam všech notifikací, které pro jeho účet kdy vnikly. U každé je možnost označit ji jako přečtenou nebo nepřečtenou, anebo na ni přejít.
3. Pokud aktér klikne na tlačítko „Přejít“, začíná „UC1: Zobrazení notifikace“.

**UC4: Nastavení emailových notifikací**
Zároveň s upozorněním ve webovém rozhraní chodí uživatelům portálu i emailová zpráva. Seznam událostí, při kterých má být tato zpráva odeslána je možné upravit.
**Aktéři:** Student, Učitel, Garant
**Scénář:**

1. Uživatel klikne na tlačítko „Nastavení notifikačních emailů“, které se nachází v přehledu posledních notifikací.
2. Systém zobrazí seznam zaškrtávacích políček s popisem událostí, které mohou vyvolat odeslání emailu. Políčka jsou předvyplněná podle aktuálního nastavení.
3. Aktér může změnit vyplnění check boxů a následně klikne na tlačítko „Uložit“.
4. Systém uloží změny a bude při budoucím odesílání emailů reflektovat nové nastavení preferencí.

## 1.4 Současná architektura

Softwarová architektura popisuje organizaci a strukturu aplikace. Její výběr je zásadním rozhodnutím, které bude mít vliv na celý další vývoj softwaru, jeho testování, nasazení a samozřejmě na jeho údržbu. Od architektury se také odvíjí možnosti škálování aplikace. Pokud je vývoj prováděn bez jasně stanovené architektury, trpí výsledek často vysokou provázaností, nejasnou

strukturou a je obtížné provádět podstatnější změny [13]. To znamená, že bez znalosti každé komponenty v systému je těžké určit, za co který modul zodpovídá, jak se systém škáluje atd.

DBS portál se v tuto chvíli skládá z několika komponent, mezi nejdůležitější z nich patří:

- semestrální práce
- testový modul
- databázový modul
- administrace uživatelů
- kreslící nástroj [6]

Všechny komponenty jsou součástí jediného projektu a nasazují se najednou, jedná se o architekturu monolitu.

### 1.4.1 Použité technologie

Pro současnou aplikaci byl použit programovací jazyk PHP s frameworkem Nette. PHP je skriptovací jazyk navržený pro tvoru webových aplikací, který je interpretován webovým serverem. Jedná se o projekt s otevřeným zdrojovým kódem, který vytvořil v roce 1994 Rasmus Lerdorf. Během let se jazyk vyvíjel a dnes pohání velkou většinu webových stránek [14]. Nette je český framework původně vytvořený Davidem Grudlem, který má za cíl usnadnit vývoj aplikací právě v PHP. Je rozdělený do samostatných balíčků a podporuje například Dependency Injection Containers, má šablonovací systém a zaměřuje se na bezpečnost. Staví na architektonickém vzoru MVC - Model-View-Controller - byť část Controller je v Nette nazývána Presenter [15]. O MVC a jeho vztahu k Nette se již rozepsali kolegové [6] [7] [5].

### 1.4.2 Monolitická architektura vs. architektura mikroslužeb

Současná verze DBS portálu je napsaná jako monolitická aplikace. Architektura navržená Ing. Andriim Plyskachem [2] je architektura mikroslužeb. Nás tedy bude zajímat zejména srovnání těchto dvou přístupů.

Hlavním rysem architektury monolitu je, že všechny funkcionality se nasazují jako jeden celek [16]. Existují sice i varianty (např. takzvaný modulární monolit), které umožňují aplikaci rozdělit na více částí, na kterých se dá pracovat nezávisle, ale při nasazení musí být tyto části nutně zkompletovány. Monolity mívají větší provázanost a jejich škálování má určitá úskalí. Aplikace se dá škálovat horizontálně současným během více instancí a použitím load balanceru, to však ale nemusí odpovídat reálnému zatížení, kdy některé moduly jsou vytíženější než jiné. U malých aplikací to nemusí být problém,

ale u větších systémů může monolitická architektura časem způsobit mimo jiné obtížnější hledání chyb a celkové zpomalení vývoje.

Naproti tomu architektura mikroslužeb dělí aplikaci do jednotek (mikroslužeb), které jsou samostatně nasaditelné [13] a které jsou modelovány okolo byznys domény. Někdy je tato architektura považovaná za jeden z typů SOA (Service Oriented Architecture), ale někdy se hovoří spíše o její evoluci. Mikroslužby mají jasně definované rozhraní a komunikují se svým okolím skrze síť (např. REST) [16]. Každá mikroslužba má obvykle svoji vlastní databázi a je dokonce možné, aby každá byla napsána v jiném programovacím jazyce. Tohle uspořádání ulehčuje vývoj tím, že změna v jedné mikroslužbě nemusí nijak ovlivnit zbytek systému, pokud rozhraní zůstane stejné. Škálování lze provádět více cíleně, kdy je v provozu více instancí jen některých mikroslužeb. V dnešní době je obvyklý scénář, kdy se již existující monolitická aplikace přepisuje právě do této architektury, učinil tak např. Netflix, Amazon [17] stejně jako právě DBS portál. Samozřejmě všechno má svoje pro a proti. Problémem architektury mikroslužeb je přidaná komplexita. V systému se nachází hned několik databází a mezi mikroslužbami dochází k síťové komunikaci, která má určité zpoždění a zpráva nemusí vždy dorazit. Při výběr vhodné architektury je tedy vždy potřeba brát do úvahu povahu projektu.

## 1.5 Požadavky

Požadavky říkají, co má vyvíjený systém splňovat, tedy jaké funkcionality má nabízet, ale rovněž specifikují např. výkon, který je požadován [18]. Jedná se o dokument, který odráží potřeby zákazníka a usnadňuje komunikaci mezi ním a vývojovým týmem. V případě, že má zákazník připomínky k dodanému produktu, jsou to právě požadavky, které ukážou, jestli zadavatel na počátku např. zapomněl zmínit některou funkcionalitu, nebo jestli je chyba na straně vývojářů, kteří nerespektovali některý z požadavků. To samozřejmě platí pouze v případě, že jsou požadavky sepsány jasně a úplně a jejich splnění musí být testovatelné. Například požadavek, aby se webová aplikace zobrazovala korektně na všech moderních prohlížečích, může být v budoucnu problematický, pokud se ukáže, že zákazník do prohlížečů splňujících tuto definici zahrnuje i Operu, ale vývojáři ne. Lepším řešením je explicitně prohlížeče vyjmenovat včetně jejich verzí, aby nemohlo dojít k nejasnostem. Z této definice je vidět podobnost s případy užití. Požadavky jsou většinou obecnější formulace, kdežto případy užití ukazují větší míru detailu. Někdy se používá tabulka ukazující, které případy užití pokrývají jaké požadavky.

Požadavky se velmi často dělí na *funkční* a *nefunkční* [18]. Funkční požadavky popisují, co má systém dělat, jak má reagovat na určité vstupy a chovat se v určitých situacích. Příkladem pro vývoj e-shopu by mohla být následující formulace: „Aplikace umožní zákazníkovi řadit zobrazené produkty podle ceny vzestupně i sestupně“. Někdy může být i ve formě negativního

vymezení, tedy co systém dělat nebude. Naproti tomu nefunkční požadavky nepopisují chování ale kritéria, která musí systém respektovat. Mezi obvyklé typy patří: *bezpečnost, kapacita, kompatibilita, dostupnost, škálovatelnost* [19] atd. Příkladem je: „Při 1 000 současně pracujících uživatelích bude odpověď na požadavek vygenerována do 3 vteřin".

Dalším způsobem, jakým lze požadavky klasifikovat, jsou modely FURPS a MoSCoW nebo jejich kombinace. FURPS je zkratkové slovo pro „Functionality, Usability, Reliability, Performance a Supportability". Jedná se o kategorie požadavků původně definované společností Hewlett-Packard již v 80. letech [20] a někdy bývá rozšířen například o kategorii „Implementation". Model MoSCoW je o něco novější a soustředí se na prioritizaci. Rozděluje požadavky na „Must have, Should have, Could have a Won't have", z čehož plyne i jeho název. Oba modely budou dále použity v českých variantách. Priority budou označeny jako „vysoká, střední, nízká, vynecháno".

Zde již končí popis současné podoby DBS portálu. Níže uvedené požadavky jsou pokryty stávajícími případy užití z velké části, avšak ne úplně, protože požadavky patří k nové podobě systému, která nebude ve všem stejná. Vychází se z nich v dalších kapitolách.

### 1.5.1 Funkční požadavky

#### F1: Správa testových šablon

Backend nového testového modulu musí zachovat entitu testová šablona jakožto vzor pro vytváření instancí testů, který lze použít opakovaně. Šablona bude i nadále obsahovat název, čas pro vypracování, maximální počet bodů a seznam přiřazených otázek, u kterých půjde nastavit opravující učitel, počet bodů a pořadí. Učitel bude schopen šablony vytvářet, duplikovat, upravovat a mazat.
**Kategorie**: funkčnost
**Priorita**: vysoká

#### F2: Používání štítků

V novém DBS portálu budou důležité štítky. Půjdou přiřazovat k otázkám i k testovým šablonám. Šablona bude mít nula, jeden nebo více štítků. Bude možné je jednoduchým způsobem filtrovat, kdy učitel bude moci vybrat do vyhledávání seznam štítků a budou mu vráceny šablony, které mají přiřazené všechny z nich.
**Kategorie**: funkčnost
**Priorita**: střední

#### F3: Vytváření instancí testů

Učitel bude schopen vytvořit instanci testu z některé validní šablony. Do vzniklé instance půjde přiřadit studenty jednotlivě nebo hromadně podle událostí vypsaných v KOSu. Dále bude možné nastavovat časové okno, kapacity dostupných místností a nutnost použití Progtest image nebo manuální kontroly karet před začátkem testu. Také se budou dát vložit instrukce, které se studentům zobrazí před vyplňováním testu.
**Kategorie**: funkčnost
**Priorita**: vysoká

#### F4: Generování dynamických testů

Nový DBS portál bude obsahovat bohatší možnosti automatického generování testů. Doposud bylo generování možné pouze u šablon typu *automatický demo test* a otázky se vybíraly čistě podle typu - SQL, RA, atd. Nově ale musí jít otázky do šablony volit na základě nastavených štítků. K popisu tohoto výběru bude sloužit jazyk obsahující štítky a logické operátory mezi nimi, jehož přesná podoba a zpracování je vyvíjeno v rámci bakalářské práce Tomáše Douby. Ze šablon obsahujících automatické generování otázek bude možné vytvářet všechny typy testů, nejenom demo testy.
**Kategorie**: funkčnost
**Priorita**: střední

#### F5: Zobrazení absolvovaných a dostupných testů

Student bude schopen si zobrazit seznam testů (včetně demotestů), které v minulosti vyplnil. Dále musí vidět aktuálně běžící testy, do kterých je přiřazen, a z tohoto seznamu musí být možné se dostat přímo do testu a začít jej vyplňovat. Pokud byl test již opraven, půjde si zobrazit náhled, kde bude vidět celkové hodnocení, otázky (zobrazené bez zadání, pokud byl testový termín tak nastaven) a jejich bodové, případně slovní hodnocení.
**Kategorie**: funkčnost
**Priorita**: vysoká

#### F6: Vyplňování studentských testů

Ve chvíli, kdy učitel připraví variantu testu a následně ji spustí, bude mít student i učitel řadu možností interakce s testem. Student si může zobrazit svůj studentský test a vyplňovat ho. Odpovědi na již vyplněné otázky musí být vidět i v případě znovunačtení stránky. Student může ukončit svůj test ještě před koncem časového limitu. Nově musí být možné během určeného času (požadavkem frontendu bylo nastavit tento čas na 15 vteřin) ukončení odvolat a vrátit se k vyplňování, nebo ukončení potvrdit. Učitel může ukončit

dosud nespuštěné studentské testy dříve, než skončí časové okno pro celou instanci testu. Rovněž může přidávat čas všem studentům a nově i konkrétnímu studentovi. V případě potřeby může předčasně ukončit test konkrétního studenta. Systém automaticky ukončí studentský test v případě, že vyprší čas na test nebo celkové časové okno podle toho, co nastane dříve. Student by neměl být schopný si během vyplňování testu zobrazovat jiné části portálu, obzvláště ne vyplněné demotesty nebo semestrální práci.
**Kategorie**: funkčnost
**Priorita**: vysoká

#### F7: Opakované vyplňování demo testů

Demo testy si zachovají svoji funkcionalitu. To znamená, že student může demo test vyplňovat opakovaně, získané body se nepočítají do hodnocení předmětu a odpovědi nejsou posílány na případnou manuální opravu učitelem.
**Kategorie**: funkčnost
**Priorita**: vysoká

#### F8: Zasílání notifikací

Uživatelé musí být informování prostřednictvím notifikací o událostech v testovém modulu minimálně ve stejném rozsahu jako ve stávajícím systému. To znamená, že učitelé musí být upozorněni na ukončený testový termín a na otázky vyžadující manuální opravu. Studenti zase musí vidět, že mají test připravený k vyplnění, jejich test byl opraven a že jim byly přiděleny body za ústní zkoušení. Uživatelé musí mít možnost nastavit si, na které události jim budou chodit notifikace.
**Kategorie**: funkčnost
**Priorita**: nízká

### 1.5.2 Nefunkční požadavky

#### N1: Architektura mikroslužeb

Systémy pro tvorbu testových šablon i pro průběh testu budou realizovány v rámci architektury mikroslužeb, kterou pro DBS portál navrhl Ing. Andrii Plyskach. Přesně rozdělení na mikroslužby nemusí být identické s jeho návrhem, výstup této práce ale bude dodržovat zásady této architektury.
**Kategorie**: implementace
**Priorita**: vysoká

#### N2: Technologie PHP + Symfony

Mikroslužby tvořící dané systémy budou napsány v programovacím jazyce PHP s použitím Symfony frameworku. Díky výhodám zvolené architektury by

bylo možné tuto část nového DBS portálu napsat a nasadit i s použitím jiných technologií, ale neslo by to s sebou řadu nevýhod do budoucna. Při zachování jednotného jazyka v rámci celého systému stačí, pokud se budou lidé podílející se na vývoji orientovat právě v onom jazyce. Nestane se tedy, že o určitou věc se bude moct postarat jen jeden člověk, který bude umět daný problém řešit například v Javě. Na projektu často pracují studenti předmětu BI-SP1, pro které to může být první setkání s podobným projektem a tato strategie jim urychlí proces zaučení. Tyto dvě technologie tedy byly dané předem.
**Kategorie**: implementace
**Priorita**: vysoká

### N3: REST API

Mikroslužby budou komunikovat mezi sebou, se zbytkem backendové části a s frontendem skrze předem definované REST API. Toto rozhraní bude respektovat zásady správné práce se zdroji a u velkých entit bude ve vhodných situacích vracet pouze její stručnou reprezentaci (bez vnořených entit) v zájmu nižšího vytížení sítě a hardwaru.
**Kategorie**: implementace
**Priorita**: vysoká

Kapitola 2

# Návrh

Další fází softwarového procesu, která v klasickém vodopádovém modelu následuje po analýze, je návrh. Výstupem je popis struktury implementovaného softwaru, datových modelů, rozhraní sloužící pro komunikaci aplikace s okolím i pro zasílání zpráv mezi jednotlivými moduly [18]. Při návrhu je potřeba opírat se o analýzu, zvláště pak o požadavky, které návrh musí respektovat. Pokud by se tak nedělo, nastala by stejná situace jako v případě špatné analýzy, to jest, že software nebude dělat to, co zákazník očekává. Problémy způsobené špatným designem ale nemusí být takto patrné. Mohou se projevit až později při změnách v aplikaci, které nevyhnutelně nastanou. Může se ukázat, že provedení i malé změny ve funkčnosti bude vyžadovat velký zásah do zdrojových kódu, protože celý systém trpí např. vysokou provázaností. Přidání nového typu nějaké entity zase může znamenat, že místo přidání jedné třídy se musí značně upravit databázové schéma. Dobrý návrh by proto měl mířit na flexibilitu, udržitelnost a znovupoužitelnost [21].

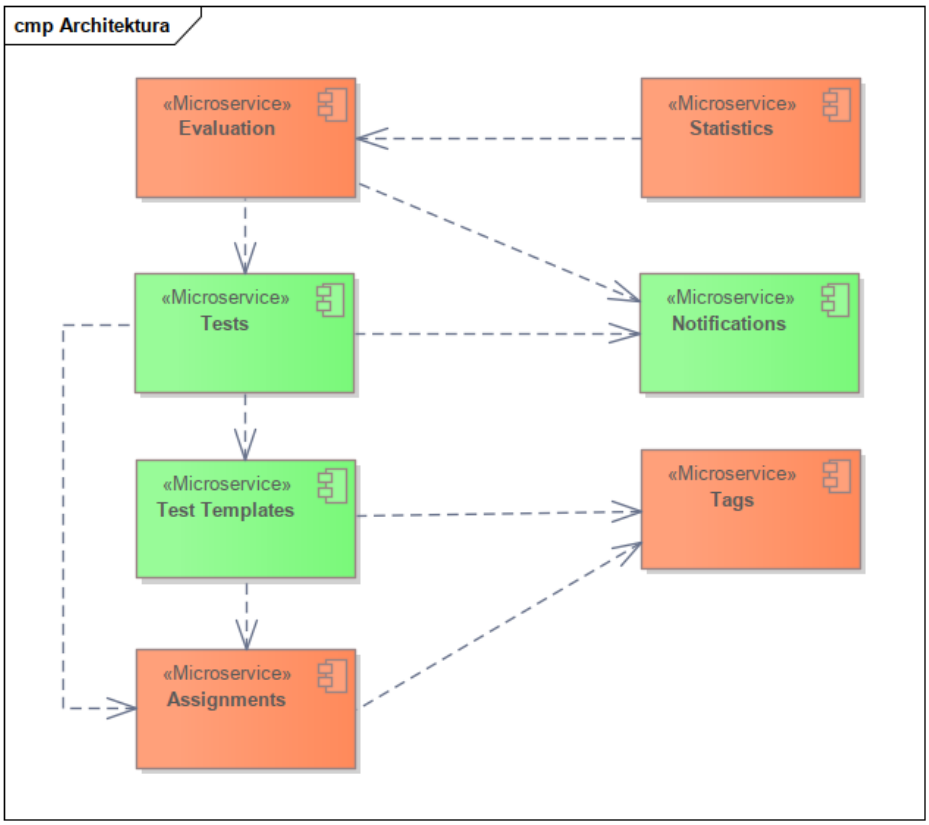
Návrh nemusí popisovat systém do sebemenších detailů, ty mohou být nechány na dořešení při implementaci. Části, které tato fáze obsahuje, se také liší podle druhu vyvíjeného systému. V této kapitole bude nejprve představena zvolená architektura a poté technologie, které budou použity při implementaci. Následně bude popsána problematika integrace komponent vytvořených v této práci se zbytkem testového modulu a dalších částí systému. Další podkapitoly jsou organizovány podle části systému, kterých se týkají. Vždy jsou vysvětleny změny ve funkčnosti oproti současnému testovému modulu, jelikož tato práce nespočívá pouze v reimplementaci stávajícího DBS portálu pod jinou architekturou, ale počítá se například i s rozšířením možnosti automaticky generovat testy atd. Následuje popis navržené databáze a rozhraní. Popis třídního modelu bude ponechán do kapitoly 3.

Počátečním záměrem bylo postupovat podle vodopádového modelu, to jest udělat kompletní návrh, teprve poté se pustit do implementace a do návrhu již nezasahovat. Nakonec se ukázalo, že změny v návrhu bylo potřeba dělat i v pozdní fázi implementace. Použitý postup včetně příslušné teorie je blíže

popsán v kapitole 3.

## 2.1 Architektura

Na obrázku 2.1 je vidět rozdělení části DBS portálu na mikroslužby a jejich komunikace v podobě, kterou navrhl Ing. Andrii Plyskach, jedná se o část diagramu obsaženého v jeho diplomové práci [2]. Kromě čtyř komponent přímo se týkajících testů jsou do diagramu zahrnuty i moduly starající se o sledování statistik, správu notifikací a štítků, které s testy souvisí a jsou také aktuálně vyvíjeny. Zeleně označené mikroslužby vznikají nebo jsou doplňovány v rámci této práce, červené jsou součástí bakalářských prací kolegů Pavlička a Douby. Rozdělení zbytku systému, jako je třeba modul pro semestrální práce, je k nalezení v uvedené práci.




Obrázek 2.1: Původně navržené mikroslužby týkající se testů.

Toto rozdělení bylo k dispozici předem, samozřejmě ale nebylo nutné se ho při návrhu do detailu držet. Co ale bylo nutné respektovat, byla celková koncepce. Nejde jenom o architekturu mikroslužeb, ale i o strukturu jednot-

livých modulů. Jak bude popsáno dále, v rámci celého portálu existují jmenné konvence pro databáze i API. Především je ale pro mikroslužby stanovena čtyřvrstvá architektura.

Jednou z uvažovaných změn bylo sjednotit testové šablony a správu testů do jedné mikroslužby. Nevýhodou rozdělení na dva moduly je totiž nutnost dotazovat se na testové šablony pokaždé, když je potřeba získat seznam otázek v testu. Alternativou je pak otázky do testu nakopírovat, což ale způsobí duplikaci dat v databázích dat. Nakonec ale převážily argumenty pro zachování stavu zachyceného v diagramu s tím, že otázky budou při vytvoření varianty testu zkopírovány. Díky tomu budou na sobě šablony a testy méně závislé, čehož se využije pro možnost upravovat šablonu i poté, co byla použita, a zároveň automaticky generovat otázky do dynamických testů až při přiřazování studentů.

Dalším dilematem bylo umístění dat a endpointů souvisejících se šablonami. V první fázi byla zamýšlená samostatná mikroslužba s názvem *Notifications*. Při bližším pohledu na systém pro správu předmětu bylo ale jasné, že tato varianta by znamenala velké množství volání navíc. Při každé zaslané notifikaci by se musela získat data o uživateli, jeho nastaveních a také kurzu. To vedlo k závěru, že lepší bude začlenit tyto funkcionality do mikroslužby Configurations. To je jediná změna oproti původnímu diagramu.

Z obrázku je také vidět, jaké závislosti bude potřeba řešit (nejedná se o kompletní seznam závislostí v celém portálu). Testy závisí na testových šablonách a obě tyto části používají data z mikroslužby starající se o otázky a zadání. Část konfigurace předmětu zabývající se notifikacemi zase závisí na datech z testů a z jejich hodnocení. Endpointy týkající se testů budou volány Evaluations mikroslužbou. Všechny tyto součásti budou zároveň komunikovat s nově vznikajícím frontendem a také s se zbytkem modulu Configurations, který uchovává široce používané informace o uživatelích, kurzech, paralelkách atd. a jehož další závislosti pro přehlednost na diagramu nejsou znázorněny.

Použitím této architektury je splněn nefunkční požadavek „N1: Architektura mikroslužeb“.

## 2.2 Použité technologie

### PHP

Vývoj nového portálu probíhá rovněž v jazyce PHP z důvodů již zmíněných v nefunkčním požadavku *N2: Technologie PHP + Symfony*.

### Symfony

Místo Nette bude v backendu celého portálu použit framework Symfony, který je rovněž vyžadován i nefunkčním požadavkem. Symfony je framework určený pro tvorbu webových aplikací v jazyce PHP, který staví na sadě komponent,

při jejichž vývoji je dbáno na vysokou znovupoužitelnost a nízkou provázanost. Je vyvíjen již od roku 2005 jako projekt s otevřeným zdrojovým kódem, na němž se podílí několik tisíc přispěvatelů [22]. Původní autoři se inspirovali například Springem a v současné době se jedná o jeden z nejpoužívanějších PHP frameworků [23].

### Composer

Pro správu závislostí v projektu je používán nástroj Composer. Ten umožňuje formou *.json* souboru definovat přímo konkrétní verze knihoven a po zavolání příslušných příkazů je nainstalovat nebo aktualizovat. Díky tomu je spouštění na libovolném stroji - nebo v Docker kontejneru - výrazně jednodušší, než kdyby bylo nutné vše instalovat ručně [24].

### Docker + Docker Compose

Docker je platforma s otevřeným zdrojovým kódem pro sestavování softwaru v balíčcích nazvaných *kontejnery* pomocí virtualizace na úrovni operačního systému. Kontejnery jsou na sobě nezávislá prostředí, ve kterých aplikace běží. Postup jejich sestavení je popsán v tzv. *Dockerfile* souboru. Jedná se o poměrně novou technologii, první verze byla vydána v roce 2013. Unixové systémy již kontejnery podporovaly dříve, ale Docker výrazně zjednodušil práci s nimi [25]. Vývojáři mohou kontejnery nasazovat, replikovat, přesouvat a sdílet obrazy, ze kterých se běžící kontejnery tvoří, a to vše bez závislosti na prostředí konkrétního stroje [26].

Docker Compose je nástroj umožňující definování a jednoduché spouštění aplikací skládajících se z více Docker kontejnerů. K definování slouží soubor ve formátu *yaml*. S pomocí tohoto nástroje je možné i spouštět, zastavovat a monitorovat jednotlivé kontejnery i se do nich připojovat [27].

### PostgreSQL

Objektově relační databázový systém PostgreSQL je rovněž open-source projekt, který je vyvíjen již 35 let. Jedná se tedy o již zavedený a velmi populární systém zaměřující se na spolehlivost, integritu dat, bohatou sadu funkcionalit a rozšiřitelnost. PostgreSQL se velmi dobře škáluje s v současné době ho používají projekty od těch nejmenších až po prostředí, které operují nad petabajty dat. Žádná relační databáze v současné době nesplňuje všechny požadavky standartu *SQL:2016*, ale právě PostgreSQL je tomu velmi blízko [28].

### Traefik

Traefik je moderní nástroj pro směrování, vyvažování zátěže a proxy jak pro prostředí běžící lokálně, tak i pro cloud. Podporuje řadu často používaných

technologií jako Docker, Kuberneetes atd. a dokáže i sám upravovat svoji konfiguraci. Jedná se v podstatě o další vrstvu abstrakce nad komunikací mezi jednotlivými službami. Tím se vývojářům zjednoduší mimo jiné diagnostika chyb souvisejících právě s komunikací [29].

Z dalších nástrojů lze zmínit Redmine, Slack nebo Gitlab pro zaznamenávání a koordinaci práce a pro verzování.

## 2.3 Integrace

Jedním z cílů práce je i integrace se zbytkem nového portálu se zaměřením na celkovou funkčnost testové části. Vzhledem k navržené architektuře a k použití REST API by se mohlo dát, že jde o triviální úkol a půjde víceméně jen o zajištění toho, že dvě mikroslužby nepoběží na stejném portu atp.

Opak je ale pravdou. To, že například systém pro průběh testu vystaví nějaké rozhraní, neznamená, že systém pro hodnocení testů ho bude moci snadno využít. Toto rozhraní nemusí poskytovat všechna potřebná data, nebo je může posílat ve formě, která nevyhovuje potřebám jiné mikroslužby nebo frontendu, které mohou mít úplně jiná očekávání nebo i odlišné pochopení některé stránky byznysu. V tom případě mohou být sice všechny části systému korektně navrženy a implementovány, ovšem jako celek nepůjdou použít.

Riziko, že taková situace nastane, se dá omezit diskuzí v průběhu vývoje, zejména během analýzy a návrhu. Mimo to bylo nutné dodržovat koncepce celého projektu, které říkají např., jak se pracuje s autorizačními tokeny. A to bylo také mým úkolem. Během tvorby závěrečných prací byly organizovány pravidelné schůzky s oběma kolegy vyvíjejícími další součásti backendu i schůzky s Danou Suchomelovou, která se spolu s jedním z SP týmů zabývá tvorbou frontendu testové části portálu z pohledu studenta (rovněž v rámci své bakalářské práce). Na těchto schůzkách byl probírán postup vývoje jednotlivých modulů a zejména důležité byly ve fázi návrhu API, kdy poskytly možnost všem zainteresovaným stranám návrh připomínkovat. Co dalšího tato část zadání obnášela bude vidět v kapitole 3.

## 2.4 Systém pro správu testových šablon

### 2.4.1 Změny oproti současnému stavu

Níže uvedené změny byly navrženy v souladu s funkčními požadavky „F1: Správa testových šablon, F2: Používání štítků“ a „F4: Generování dynamických testů“.

Testové šablony bude nově možné upravovat, i pokud už byly někdy použity pro tvorbu varianty testu. Na dříve vytvořené testy to nebude mít vliv, jelikož důležité informace ze šablony se budou kopírovat do varianty testu. To umožní

větší flexibilitu, kdy nebude nutné kvůli změně jedné otázky celou šablonu duplikovat. Pokud by se tato funkce využívala často pro kompletní změnu sady otázek, snížila by se vypovídací hodnota statistik úspěšnosti studentů ve vytvořených testech, ale neočekává se, že by učitelé se šablonami pracovali tímto způsobem. Navíc v současné podobě šablon statistiky ztrácí na využitelnosti právě duplikací kvůli menším změnám.

Při mazání nedojde k úplnému odstranění, ale pouze k archivaci. Taková šablona se bude zobrazovat pouze ve zvláštním seznamu archivovaných šablon a nepůjde upravovat ani použít pro vytváření testů. Jedná se o jednu z metod historizace databáze [30] (zde je vhodná, protože varianta testu si stále udržuje informaci o tom, z jaké šablony vznikla).

K šablonám bude možné přiřazovat štítky. Podle těch bude možné filtrovat vyhledávání šablon (seznam štítků s logickým operátorem AND). Štítky se udržují v mikroslužbě nazvané Tags.

Otázky ve formě podobné té ze stávajícího systému, jsou nyní nazvané statické otázky a udržují si pouze externí identifikátor do Assignment mikroslužby, nikoliv obsah otázky. Kromě toho přibyly dynamické otázky. Ty jsou náhradou stávajícího jednoduchého generování testů na základě zadaného počtu otázek z jednotlivých kategorií. Dynamická otázka si udržuje identifikátor zadání, do kterého mají následně vygenerované otázky patřit - v nové podobě správy zadání a otázek je invertovaný vztah mezi otázkou a zadáním. Dále tento typ otázky textový výraz obsahující štítky a logické operátory z jazyka navrženého Tomášem Doubou společně s počtem otázek, které se mají dle výrazu vygenerovat. Jakmile to bude potřeba, tyto údaje se odešlou systému pro správu zadání a otázek a ten vrátí seznam odpovídajících otázek. Dostatečné množství validních otázek splňujících daná kritéria se nemusí v danou chvíli v systému nacházet. Jak byl tento problém řešen je popsáno níže.

Šablona obsahuje i pořadí otázek jak statických, tak dynamických. Tyto kategorie se řadí samostatně, jelikož umožnit prokládání by bylo složité a není očekáváno využívání takové funkce ze strany učitelů. Při vytváření je rovněž možné nastavit použití náhodného pořadí dynamických otázek ve vzniklých testech.

### 2.4.2 Databáze

Součást návrhu jednotlivých mikroslužeb jsou i databáze. Relační databázová schémata se vzhledem ke své velikosti nachází v příloze D. Konkrétně schéma databáze testových šablon ukazuje obrázek D.1.

Ve všech třech schématech je vidět používání externích identifikátorů do dalších mikroslužeb. Např. v tabulce *test_template_tag* se nachází atribut *tag_id*, jenž nabývá hodnot primárního klíče příslušné tabulky v systému pro správu štítků. Zajištění platnosti takových identifikátorů skrz několik databází zvyšuje nároky na jednotlivé mikroslužby a je jednou z nevýhod této architektury [16]. Celá tabulka *test_template_tag* je vlastně rozkladová tabulka M:N vztahu mezi

šablonou a štítkem s tím, že entita štítek se nachází v jiné databázi. Tabulka ale nemá žádné další atributy, proto je použit složený primární klíč.

Při návrhu databází bylo dbáno na dodržování pravidel pojmenování tabulek, atributů, cizích klíčů atd., které sepsal Ing. Andrii Plyskach [2] a které zajistí jednotný styl pojmenování v celém portálu, přestože se na něm podílí více autorů.

Za zmínku stojí příznak *is_valid* v tabulce *test_template*, který je nastavený na „true“, pokud se součet bodů za jednotlivé otázky rovná maximálnímu počtu bodů v šabloně a zároveň byly při poslední kontrole všechny statické i dynamické otázky platné. Statická otázka je platná tehdy, pokud je příslušná otázka (*question_id*) platná. Dynamické otázky se z hlediska platnosti hodnotí všechny najednou a zkoumá se, zda-li je možné všechny požadavky dané výrazy splnit tak, aby se žádná otázka nevygenerovala dvakrát. Tento příznak je použit primárně pro filtrování, protože může být zastaralý. Do budoucna se počítá i s implementací mechanismu pravidelného kontrolování otázek z hlediska jejich platnosti, který může nastavit šablonu jako nevalidní, pokud shledá, že některé z vložených otázek se nedají použít.

### 2.4.3 API

Popisy API (Application Programming Interface) všech tří mikroslužeb jsou k nahlédnutí v příloze E. Na obrázcích je jen seznam endpointů, ale kompletní dokument ve formátu *yaml* se nachází na přiloženém médiu společně s Enterprise Architect projektem obsahujícím všechny diagramy v této práci. Kromě toho je dokumentace i ve stejném Gitlab repozitáři jako zdrojový kód.

I pro návrh API jsou v DBS portálu stanovené určité konvence [2].

Rozhraní pracuje se čtyřmi zdroji, šablonou, štítky přiřazenými k šabloně, statickými a dynamickými otázkami, jak je vidět na obrázku E.1. Ze seznamu je patrné, že je možné se šablonami provádět všechny výše popsané akce včetně získání seznamu archivovaných šablon.

Neobvykle působí předposlední endpoint v sekci *template* - metoda POST na URL `/templates/templateId/check-validity`. Jeho úkolem je zkontrolovat a vrátit aktuální platnost šablony a aktualizovat již diskutovaný atribut *is_valid*. Učiteli se tak může zobrazit varování už při vytváření varianty testu. Jak je ale níže vysvětleno, ani tato kontrola nezaručí, že problém nenastane později.

## 2.5 Systém pro průběh testu

### 2.5.1 Změny oproti současnému stavu

Stávající pojmy „termín“, „instance testu“ a „studentský test“ jsou matoucí a navíc není aktuálně žádná možnost, jak ve zkoušce nebo zápočtovém testu

dát každému studentovi jinou sadu otázek, proto bude ve zde vytvořené implementaci tato logika fungovat odlišně.

Termín bude obsahovat několik variant testu, které se budou vytvářet ze šablon (funkční požadavek „F3: Vytváření instancí testů“). Termín nově nebude znamenat pouze událost v KOSu, ale bude na ní případně jen odkazovat, přičemž se do něj přesune část informací, které byly doposud v instanci testu. Časové okno pro celý test tím pádem nemůže být v šabloně, protože nedává smysl, aby bylo pro dva studenty jiné. To s sebou nese nevýhodu v podobě nutnosti nastavovat tento údaj při každém vytváření termínu. Jedná se avšak jen o jedno políčko ve formuláři, které navíc může být předvyplněné nějakou výchozí hodnotou. Studenti, paralelky a místnosti s kapacitou budou nastavovány u termínu, přičemž přidělení studentů k variantám a do místností bude provádět interní algoritmus. Termínu mohou být tři typy - demotest, semestrální test a zkouška.

Varianta testu je potom hlavně kopií šablony a obsahuje všechny její důležité údaje včetně množiny otázek. Pro účely statistik si drží i odkaz na šablonu, byť ta mohla být mezitím upravena. U varianty se rozlišují dva typy - statická a dynamická. Pokud je varianta statická, znamená to, že statické otázky budou ve všech studentských testech patřících do dané varianty stejné (což ani jinak zařídit nelze) a dynamické otázky se rovněž vygenerují pro všechny studenty shodně. Dynamická varianta naopak znamená, že dynamické otázky se pro každého studenta vygenerují zvlášť a každý z nich tedy může vyplňovat jinou množinu otázek. Generování dynamických otázek z jejich logických výrazů se u statické varianty provede při vytváření, u dynamické poté pro každého studenta ve chvíli, kdy jsou generovány studentské testy. Pokud by neexistoval dostatečný počet otázek pro vygenerování, bude učitel upozorněn a může případně upravit šablonu. Pokud by generování probíhalo až při spuštění testů, nebyl by už čas problém řešit. Samozřejmě vzhledem k faktu, že otázky ze zadání, které má část typu RA nebo SQL, se mohou stát nevalidní - a tedy nepoužitelné pro generování - kdykoliv, žádná strategie nezajistí zamezení vuniu této situace na 100%. Riziko bude ale v novém portálu menší než v současném, jelikož informace o databázovém připojení se nyní nachází přímo v části zadání.

Během návrhu této části portálu vyvstal jeden zásadní problém. Na systém pro hodnocení testů je kladen požadavek, aby se manuální hodnocení určité otázky uložilo a další identická odpověď jiného studenta se už automaticky ohodnotila stejným počtem bodů, čímž se opravujícím výrazně ušetří práce. Maximální počet bodů za otázku je ale uložen v šabloně, kterých bude nově možné do termínu vložit několik. To může vést k situaci, kdy bude v termínu stejná otázka několikrát a pokaždé za jiný počet bodů. To by značně zhoršilo právě automatické hodnocení na základě předchozího manuálního obodování. Může se zdát, že je to nepravděpodobný jev a jednalo by se o chybu učitele, ale toto riziko se zvyšuje při použití dynamických otázek. Pro ilustraci předpokládejme, že v systému existuje otázka $X$ se štítky „SQL“, „Těžká“ a „Left-

join“. Šablona *A* obsahuje dynamickou otázku s požadavkem na jednu otázku splňující „SQL AND Těžká“ za 4 body. V šabloně *B* je dynamická otázka, jež obsahuje štítky „SQL AND Left-join“ a dává za vybranou otázku 3 body. Do termínu jsou vloženy varianty vzniklé z obou těchto šablon. Jednomu studentovi se otázka *X* vybere v rámci šablony *A* (tedy za 4 body), druhý student bude vyplňovat také otázku *X*, ale v šabloně *B* za 3 body. A toto je jen jedna ze situací, kdy může tento konflikt nastat. Bylo uvažováno několik možných řešení:

- Opustit celou myšlenku více variant v jednom termínu a zachovat termíny a instance testů ve stávající podobě. V takovém případě by nebylo možné, aby měl každý student jinou podobu studentského testu.
- Problém neřešit a různé bodování otázky umožnit bez úpravy způsobu automatického hodnocení, což by znamenalo, že oprava odpovědi, která je identická s nějakou dříve opravenou, by nutně nefungovala ani v rámci termínu. Musel by být roven maximální počet bodů za obě odpovědi.
- Zamezit celé situaci striktnějšími požadavky na varianty a generování otázek. Učitel by byl upozorněn, pokud by se problém vyskytl a musel by ho nejprve vyřešit. U statických otázek by stačilo vybrat jinou šablonu nebo ji upravit. V případě dynamických otázek je ale situace složitější. Mohlo by se také stát, že se konflikt objeví až po přiřazení dalšího studenta do termínu. Teoreticky by se dalo Assignments mikroslužbě spolu s textovým výrazem posílat i seznam otázek, se kterými nemůže nastat konflikt. To by ale na ni kladlo o dost větší nároky a častěji by se stávalo, že otázky nebudou k dispozici. Kromě toho celé uvedené řešení zhoršuje uživatelskou přívětivost portálu. Učitel by během tvorby termínu dostával na první pohled s jeho akcemi nesouvisející chybové hlášky o konfliktu v bodování jedné otázky. Smysluplné chybové hlášky jsou ale jedním z principů návrhu uživatelského rozhraní [31].
- Zavést procentuální hodnocení otázek. Učitelé by mohli stále manuální hodnocení zadávat v bodech, ale backend si spočítá, kolik procent z maxima bylo studentovi uděleno a při následných automatických opravách se jich bude držet. Přidaná komplexita bude při takovém řešení v Evaluation mikroslužbě, jejíž autor bude muset analyzovat případné problémy s přesností strojových čísel a zaokrouhlováním.

Nakonec byl po diskuzi s Jakubem Pavličkem zvolen poslední přístup, který vyučujícím usnadní tvorbu termínů s více variantami a dynamickými otázkami.

Podoba studentského testu zůstává bez velkých změn, přibývá pouze možnost do nastaveného počtu vteřin odvolat rozhodnutí ukončit předčasně test, nebo souhlas potvrdit, čímž možnost odvolání zanikne.

### 2.5.2 Diagramy aktivit a stavové diagramy

Ze všech tří vyvíjených součástí portálu je problematika správy a psaní testů nejsložitější a uvažované entity mají množství stavů, ve kterých se mohou nacházet. Zaslouží si proto znázornění v UML diagramech aktivit. Vzhledem ke své velikosti se diagramy nachází v příloze B. Pro přehlednost jsou aktivity týkající se termínů ukázány na obrázku B.1 a testy jsou vidět odděleně na druhém diagramu B.2. Akce manipulující s termínem mají vliv i na stav studentských testů, které do něj náleží, proto na druhém diagramu začíná studentský test již ve stavu „připravený“. Vyhodnocení testů je v obou případech reprezentováno jedinou aktivitou, přestože se jedná o komplexní proces. Návrh jeho přesné podoby ale není předmětem této práce.

Je vhodné si zároveň popsat, co vlastně jednotlivé stavy znamenají. V příloze C jsou stavové diagramy termínu i testu, na kterých jsou přechody patrnější než na diagramech aktivit. Termín se může nacházet v jednom z následujících stavů:

- **Nový** – Termín byl vytvořený, ale nemůže být spuštěn, jelikož neobsahuje žádné varianty, nejsou přiřazení studenti nebo není nastavena dostatečná kapacita místností.
- **Připravený** – Je možné termín spustit, jelikož splňuje všechny náležitosti. Z tohoto stavu se dá vrátit zpět na stav předchozí, pokud jsou provedeny změny, které smažou vygenerované studentské testy.
- **Probíhající** – Začalo časové okno pro termín a studenti mohou začít pracovat na testech.
- **Ukončený** – Vypršel čas pro termín nebo již byly ukončeny všechny studentské testy. Odpovědi čekají na vyhodnocení.
- **Vyhodnocený** – Všechny studentské testy byly opraveny, jedná se o koncový stav termínu.

Studentské testy mají podobné stavy. Kromě nich ještě jeden přibyl:

- **Nový** – Studentský test vzniká při vygenerování testů v termínu a nejprve se nachází v tomto stavu.
- **Připravený** – Student může začít pracovat. Nový test se do tohoto stavu přesouvá, pokud se spustí termín. Jestliže se jedná o zkoušku, musí být zároveň fyzicky zkontrolován průkaz studenta.
- **Probíhající** – Časový limit na test se začíná počítat ve chvíli přesunu testu do tohoto stavu. Student může odpovídat na otázky, dokud nevyprší tento limit nebo okno pro celý termín, nebo dokud test předčasně neukončí sám student, či ho neukončí učitel.

- **Podmínečně ukončený** – student klikl na možnost ukončit test předčasně, ale ještě neproběhl čas, během kterého se dá rozhodnutí zrušit (požadavkem frontendu bylo nastavit 15 vteřin). Kdykoliv se dá ukončení i potvrdit, čímž dojde k okamžitému přesunu do následujícího stavu.

- **Ukončený** – Test skončil z jakéhokoliv z výše uvedených důvodů a odpovědi čekají na vyhodnocení.

- **Vyhodnocený** – Všechny otázky byly automaticky či manuálně opraveny a student si může hodnocení prohlédnout. Zde životní cyklus testu končí. Pokud dojde k doplnění hodnocení ústní části zkoušky, stav se již nemění.

Diagramy i popisy stavů platí pro semestrální testy a zkoušky, u demotestů je situace lehce odlišná. Jelikož je možné demotesty opakovat, termín se nepřesouvá do stavu „ukončený“ jindy, než při vypršení času (učitelé ve stávajícím systému otevírali demotesty na dobu v řádu týdnů až měsíců, což bude možné i v této nové podobě).

### 2.5.3 Databáze

Schéma databáze testů najdeme na diagramu D.2. K vytvoření číselníku místností s jejich maximální kapacitou - tabulce *room* bylo přistoupeno i přesto, že u zkoušky je evidovaná místnost už v mikroslužbě Configurations. Problémem bylo, že místnosti musí být možné upravovat i v samotném testovém modulu a pro tento účel je mnohem lepší, pokud má učitel možnost výběru a vidí právě i maximální kapacity (ty nebudou systémem vynucovány, slouží hlavně jako informace). Při vytváření zkouškového termínu je nutné se ujistit, že v této databázi existuje místnost se stejným názvem jako ta importovaná z KOSu.

U varianty se kromě identifikátorů původní šablony uchovává i časová značka, která jednoduše umožní říci, jestli byla od doby vytvoření varianty šablona upravena.

Statické otázky ze šablony se převedou do *test_question* a dynamické do *dynamic_test_question*. Během generování se vniklé otázky uloží rovněž do tabulky *test_question*, která má vazbu jak na studentský test tak na variantu. Její atribut *type* si uchovává infcormaci o tom, jak otázka vznikla - generováním či přímo. Vazba na variantu slouží pro statické varianty a statické otázky u dynamických variant, u kterých nemá smysl otázky nakopírovat pro každého studenta zvlášť a vytvářet tak velké množství identických záznamů. Pro získání všech otázek ze studentského testu je tedy potřeba udělat sjednocení množin otázek přes obě vazby.

Sloupečky s názvem *expected_end_at* v termínu a testu budou většinu času nastavené na NULL kromě chvíle, kdy jsou dané entity ve stavu Probíhající. V tu dobu se bude ve sloupci nacházet čas, kdy uplyne doba na vypracování.

Na tyto atributy se bude systém často dotazovat, proto jsou nad nimi vytvořeny indexy.

Tabulka *generate_tests_request* slouží k uložení požadavků na vygenerování studentských testů do termínu. Tato operace bude totiž asynchronní, jak uvidíme dále.

### 2.5.4 API

Rozhraní této mikroslužby (obrázek E.2) obsahuje o něco více endpointů oproti testovým šablonám. V tomto případě bylo obzvláště důležité respektovat potřeby frontendu a ostatních mikroslužeb, jelikož s daty o testech musí jít manipulovat z pohledu učitele i studenta, a to takovým způsobem, aby byly splněny požadavky „F5: Zobrazení absolvovaných a dostupných testů“, „F6: vyplňování studentských testů“ a „F7: Opakované vyplňování demotestu“.

Pracuje se se zdroji reprezentujícími termín, variantu, studenta přiřazeného k termínu, studentský test a místnost a přiřazenou místnost. Je dobře vidět, že například ke studentskému testu musí být možné se dostat skrz identifikátor studenta ale i přes id termínu (aby bylo pro učitele například možné přidat čas k testu). PATCH na `/student-tests/{studentTestId}` je sdílený, ale některé úpravy budou umožněné pouze studentovi a jiné učiteli, což bude v implementaci bráno v potaz. To samé platí i pro úpravy přiřazeného studenta nebo testového termínu. Alternativou by bylo mít jeden endpoint pro každou z těchto úprav, což by ale učinilo API velmi nepřehledným.

Při vytváření varianty je proveden dotaz na validitu šablony. Pokud je úspěšný, otázky jsou zkopírovány a dále už se berou jako validní. V případě statické šablony jsou dynamické otázky rovnou vygenerovány.

Za zmínku stojí i poslední dva endpointy v sekci *student-test*. Ty slouží k vygenerování studentských testů. První z nich vygeneruje do dosud nespuštěného termínu studentské testy, pokud je to možné. To znamená, že jsou přiřazeni studenti, varianty a kapacity místností jsou dostatečné. Jakákoliv akce, která mění tyto entity (přenastavení místností, přidání studentů, ...) způsobí, že se testy smažou. Je to nejjednodušší způsob, jak umožnit tyto úpravy v jakoukoliv chvíli před spuštěním, byť bude nutné testy poté přegenerovat. Jiná řešení by vyžadovala složité mechanismy reakce na každou možnou změnu. Druhý endpoint slouží k vytvoření dalšího pokusu pro termín typu *Demo*. První test bude vygenerován při zakládání termínu, další si student opatří právě tímto způsobem.

Prvně jmenovaná z těchto dvou operací funguje asynchronně, jelikož v praxi může znamenat velké množství volání do jiných mikroslužeb. V dynamické variantě se musí pro každou dynamickou otázku každého studenta otázky vygenerovat zvlášť. Pokud by se tak dělo synchronně, uživatel by mohl dlouho čekat na odpověď, teoreticky by mu mohlo i vypršet spojení. Proto bude při zavolání endpointu požadavek pouze zaznamenán a uživatel dostane odpověď s odkazem, na kterém může sledovat stav vyřízení. Pro sledování slouží en-

dpoint `/generate-tests-requests/{generateTestsRequestId}`. Pokud byl již požadavek dokončen, uživatel bude přesměrován na seznam vygenerovaných testů. Příslušná tabulka v databázi obsahuje i možnost zaznamenávat případnou chybu, prozatím ale nebude využita.

## 2.6 Systém pro správu notifikací

Zahrnutím implementace této části DBS portálu do rozsahu práce bude naplněn funkční požadavek „F8: Notifikace".

### 2.6.1 Změny oproti současnému stavu

Notifikace budou prozatím implementovány jednodušeji. V současném portálu se sice v databázi uchovává i informace o tom, kam by měl vést proklik na danou notifikaci (u upozornění na opravený test by se student logicky mohl dostat na detail daného testu), ale implementován bude pouze typ notifikace, který umožní frontendu použít obecnější přesměrování (na seznam všech testů atp.). Byla zvážena možnost dodat do nového systému i zobrazování bodů přímo v textu notifikace, jedná se ale pouze o doplňkovou mikroslužbu, která není hlavním zaměřením práce. Prokliky budou pouze obecné a notifikace prozatím nebudou chodit emailem. Tato rozšíření jsou vhodným úkolem pro budoucí SP týmy.

### 2.6.2 Databáze

Na diagramu D.3 je vidět, že databáze (respektive změna databáze Configurations) je velmi jednoduchá, nová tabulka má vazby na *user* a *course*. Ty jsou v diagramu pro jednoduchost znázorněny pouze s primárním klíčem, jejich zbytek je k nalezení v dokumentaci Configurations. Možné hodnoty atributu *event* ukazují, které události budou notifikace prozatím podporovat. Přidání dalšího druhu bude spočívat pouze v úpravě těchto hodnot v aplikaci.

### 2.6.3 API

Posledním navrženým API je to pro správu notifikací, které je ukázáno na obrázku E.3. Obsahuje pouze zobrazení notifikací (filtrovatelné podle uživatele i jiných atributů), přidání notifikace, úpravu jejího stavu a smazání. Není zde přítomna žádná historizace. Příjemce je daný identifikátorem uživatele v cestě.

Uživatel má možnost se skrze endpointy manipulující s nastaveními přihlásit nebo odhlásit k odběru notifikací určitého typu. Pokud nastavení neexistuje, bere se to jako, že uživatel má notifikace zapnuté. Pokud má uživatel danou notifikaci skutečně vypnutou, při volání POST pro vytvoření notifikace se vrátí status kód 204 a notifikace nebude vytvořena.

Kapitola 3

# Realizace

Tato kapitola popisuje samotnou tvorbu řešení, k němuž předchozí kapitoly popisující analýzu a návrh směřovaly. Tyto činnosti ale nemusí jít během celého vývojového cyklu striktně za sebou, byť by to tak mohlo působit. Existují i postupy, které práci organizují jinak. V úvodu kapitoly jsou popsány některé metodiky, které tyto způsoby vývoje používají, aby s nimi mohl být následně srovnán postup, který se použil při realizaci této práce. Následuje samotná implementace rozdělená opět podle dotčených systémů, jako tomu bylo v případě návrhu.

## 3.1 Metodiky vývoje

Metodika softwarového vývoje je strukturovaný postup používaný při práci na projektu, souboru takových procesů se říká metodologie [32]. Jejich význam narůstá s velikostí projektu a počtem vývojářů. Ve velkém týmu je náročnější rozdělovat práci, efektivně komunikovat a sdílet informace, což právě metodiky formalizují za účelem zefektivnění. Metodik je celá řada, z nichž každá má svoje pozitiva i negativa a nelze tak vybrat jednu, která by byla nejlepší ve všech případech. Spíše je nutné vybírat s ohledem na povahu projektu. Dále budou popsány dvě z nich.

### 3.1.1 Vodopád

Vodopád (Waterfall model) je lineární přístup k vývojovému cyklu [33], někdy se označuje za první metodiku vůbec. Definuje několik fází vývoje: sběr požadavků, analýzu, návrh, implementaci, testování, nasazení a údržbu, které v podstatě odpovídají i kapitolám této práce. Fáze následují striktně za sebou. To znamená, že jakmile je například návrh prohlášen za hotový, pozdější změny nejsou možné, proto se obvykle před koncem fáze provádí důkladná revize, aby se minimalizovala nutnost pozdějších změn.

Tato striktnost ale přináší některé nevýhody této metodiky, jako je špatná možnost zapracování zpětné vazby od zákazníka, pozdržení testování do velmi pozdní fáze a celkově nižší flexibilita a kvůli způsobu dělení práce i o něco nižší efektivita.

Vodopád se stále při vývoji softwaru používá, protože mezi jeho výhody patří mimo jiné to, že jasně stanovuje cíle a termíny a zjednodušuje plánování.

### 3.1.2 Agilní metodiky

Jedná se o množinu metodik, které vycházejí z Manifestu agilního vývoje software, který klade důraz na schopnost reagovat na změny spíše než na dodržování předem stanoveného plánu atd. [34] Už to napovídá, jakým směrem se tyto přístupy ubírají. Obecně se dá říci, že upouští od jednoho velkého cyklu, podle kterého se řídí vodopád [35]. Místo toho se neustále opakují cykly nad menší části produktu. Jedna funkcionalita se zanalyzuje, navrhne, implementuje a rovnou otestuje a nasadí. Poté se pokračuje s další funkcionalitou.

To umožňuje například daleko větší flexibilitu a schopnost reakce na změnu požadavků zákazníka a jeho zpětnou vazbu. Na druhou stranu je tento způsob práce náročnější na organizaci a je nutné dbát na dodržování určitých zásad, s čímž někdy pomáhá role agilního kouče.

Agilní metodiky jsou v současné době široce používané, adoptovala je i řada korporátů [35]. Příkladem je SCRUM, který definuje sprinty, tedy cykly, během kterých se pracuje na jednotlivých úkolech [36]. Sprinty obvykle trvají dva týdny nebo měsíc, aby se udržela flexibilita. Během jednoho sprintu se také odehrává řada schůzek jako je plánování, retrospektiva nebo denní „stand-up“, na kterém každý člen týmu informuje, na čem aktuálně pracuje a jestli narazil na nějaký problém.

### 3.1.3 Použitý postup

Během vývoje testového modulu DBS portálu nebyl striktně použit ani jeden z výše uvedených postupů, ale objevily se určité znaky obou dvou. Analýza a návrh sice byly provedeny na začátku a implementovat se začalo až poté, což by sedělo k vodopádu, ale návrh byl v průběhu ještě upravován. Někdy změna nastala kvůli drobné změně požadavků na funkcionality, někdy kvůli potřebám frontendu dostávat data v určité podobě a někdy i kvůli tomu, že původní řešení se ukázalo jako technicky složité pro implementaci. Příkladem posledně zmíněného důvodu je absence `UNIQUE` omezení na index otázky a id šablony v tabulkách *static_question* a *dynamic_question* na obrázku D.1. Dlouhou dobu se v databázi tato omezení vyskytovala, ovšem ukázalo se, že při implementaci změny pořadí otázek způsobují problémy a Doctrine nenabízí dostatečně jednoduchou možnost je dočasně vypnout. Proto bylo přistoupeno k tomu, že unikátnost se kontroluje jen na úrovni aplikace s důrazem na důkladné otestování tříd, které zajišťují změnu pořadí.

Byly organizovány pravidelné schůzky s kolegy, kteří pracovali na backendové části, na nichž jsme sdíleli aktuální postup, koordinovali práci tak, abychom se navzájem neblokovali s vývojem frontendu, a řešili věci, které se týkají více mikroslužeb, jako byl například problém s procentuálním hodnocením otázek zmiňovaný v návrhu systému pro správu testů. Také byl stanoven způsob, jakým pracovat s verzovacím systémem Git a jak hlídat dodržování standardů v kódu. Další detaily budou popsány v následující podkapitole a v kapitole o testování.

## 3.2 Implementace

Zdrojové kódy vytvořených částí backendu se nachází na přiloženém médiu. Ty však nejsou samostatně použitelné, protože jim chybí řada závislostí. Kód celého backendu, jehož jsou tyto mikroslužby součástí, se nachází v repozitáři na Gitlab, do něhož mají přístup členové DBS týmu.

Před samotným začátkem implementace bylo nutné vytvořit kostry jednotlivých mikroslužeb. Na to v repozitáři existovala zvláštní větev `base_app` obsahující adresář, který se měl zkopírovat a sloužit po úpravě jako ona kostra. V této složce byly ale definovány zastaralé závislosti, jiné chyběly a také již nebyl plně funkční návod. Nejdříve byla upravena tato větev, aby se mohlo začít pracovat na nových mikroslužbách.

Jako první se začalo pracovat na systému pro správu testových šablon, tedy Test templates mikroslužbě, jelikož na ní závisí vytváření samotných testů. V ní byly nejdříve implementovány CRUD operace a základní části business logiky. Poté bylo nutné to samé provést pro systém pro průběh testu (Tests mikroslužbu), jelikož se už paralelně začínalo s vývojem frontendu z pohledu učitele, který potřeboval rychle např. zobrazovat studentské testy, aby si jeho vývojáři nemuseli zbytečně přidělávat práci s mockováním backendu. V této fázi se primárně neřešily funkce, které se týkají výhradně pohledu učitele, například dynamické otázky a jejich generování, některé validace atd.

Ve chvíli, kdy byla tato základní část dokončena, pracovalo se na složitější logice v obou mikroslužbách, zejména na integraci Test Templates s Assignments a Tags nebo generování studentských testů. Zároveň se začalo pracovat na testování.

Jako poslední přišly na řadu notifikace, které měly nižší prioritu, a jejich integrace s Tests. Rozhodnutí začlenit je do systému pro konfiguraci předmětu, kde jsou uloženy informace o uživatelích, padla až v pozdní fázi vývoje celého modulu.

### 3.2.1 Code review

Bylo jasné, že celý testový modul, nepočítaje zbytek nového DBS portálu, se bude včetně testů skládat z desetitisíců řádků kódu tvořených několika vývojáři. Aby byla zajištěna určitá kvalita a uniformita kódu a také jednodušší

integrace s frontendem, bylo hned na začátku určeno, jakým způsobem se bude pracovat s Git repozitářem a jak budou probíhat vzájemná code review.

Veškerá práce se mergovala do vytvořené větve `test_module_integration`, ze které se oddělily větve pro jednotlivé mikroslužby (`test_templates_master` a další), které již byly v režii jednotlivých vývojářů. Po dokončení vhodně velkého množství kódu byly zakládány merge requesty do integrační větve, které vždy musel schválit alespoň jeden ze zbývajících dvou lidí. Všechny změny z ostatních mikroslužeb se musely do dané větve dostat výhradně přes integrační větev, bylo zakázáno mergovat ostatní větve mezi sebou, aby se udržela přehlednost historie a omezilo množství konfliktů. Jedná se o klasický Git workflow, kdy integrační větev sloužila jako `master` potažmo `develop` a zbylé jako tzv. `feature` větve.

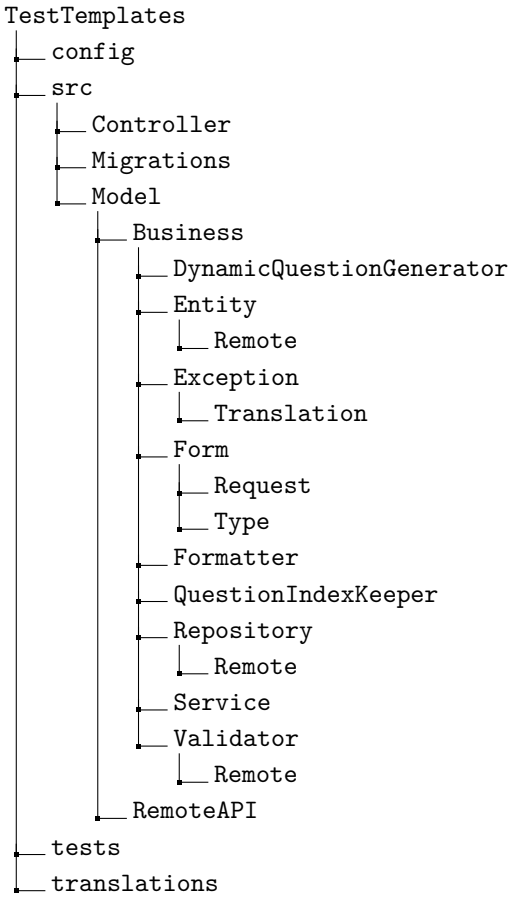
Po každé publikované změně byl Daně Suchomelové a SP týmu, se kterým pracovala, dán seznam změn relevantních pro pohled studenta. Frontend pracoval výhradně s integrační větví.

### 3.2.2 Testové šablony

#### 3.2.2.1 Struktura

Na obrázku 3.1 je vidět rozdělení této mikroslužby do balíčků. Následuje jejich stručný popis, přičemž balíčky v sekci `Model`, u kterých není napsané, jaké úrovni architektury se týkají, patří do úrovně domény.

- **config** – Konfigurace závislostí a Symfony služeb.
- **Controller** – Všechny controllery mikroslužby, které zpracovávají požadavky a dále je předávají třídám z balíčku `Service`. Patří do úrovně uživatelského rozhraní.
- **Migrations** – Migrační skripty pro databázi.
- **Model/Business** – Business logika mikroslužby.
- **DynamicQuestionGenerator** – Generování dynamických otázek pro účely validace šablony, obsahuje volání Assignments.
- **Entity** – Datové třídy, které se pomocí Doctrine ORM mapují na tabulky v databázi. V balíčku `Remote` se nachází datové třídy, které se získávají z ostatních mikroslužeb a jsou tedy uloženy v jiných databázích.
- **Exception** – Uživatelské výjimky, v balíčku `Translation` jsou třídy uchovávající zprávu výjimky tak, aby ji bylo možné celou přeložit.
- **Form** – Datové třídy, na které Symfony mapuje těla příchozích HTTP požadavků.




Obrázek 3.1: Struktura balíčků systému pro tvorbu testových šablon.

- **Formatter** – Třídy zpracovávající entity do formátu vhodného pro odpověď na požadavek.
- **QuestionIndexKeeper** – Třídy obstarávající změnu pořadí otázek (obou typů) v šabloně.
- **Repository** – Úroveň infrastruktury, která slouží pro získávání instancí entit z databáze. V podadresáři `Remote` najdeme třídy, které získávají entity z ostatních mikroslužeb po síti.
- **Service** – Služby patřící do operační úrovně, které vykonávají jednoduchou logiku a složitější úkony delegují na ostatní třídy.
- **Validator** – V mikroslužbě se nachází velké množství validací, které se vyhodnocují právě zde. Přímo v balíčku jsou složitější validace entit uložených v lokální databázi (např. testová šablona), v `Remote` potom

nalezneme validátory entit z ostatních mikroslužeb (např. otázka z Assignments).

- **RemoteAPI** – Pomocné třídy pro posílání dotazů mezi mikroslužbami.
- **tests** – Struktura testů bude popsána v další kapitole.
- **translations** – Překlady chybových hlášek v češtině a angličtině.

#### 3.2.2.2 Popis

Za znínku stojí, že během vývoje této mikroslužby se přišlo na to, že některé věci by se mohly přesunout do *Utils bundlu*, tedy knihovny, která sdružuje různé pomocné třídy používané skrz větší množství mikroslužeb pro zamezení duplikace. Jednalo se o drobnosti jako přesun jedné třídy nebo doplnění několika metod, takže byly tyto úkoly evidovány ve školním Redminu, aby si je mohli rozebrat studenti z SP týmů. Nakonec se jich ujal David Ratimec. Možnost přímého zapojení SP týmů do vývoje zde popsaných mikroslužeb využita nebyla.

K zajímavějším technickým řešením, která se následně použila i v Tests a mohla by být časem rovněž přesunuta do *Utils bundlu* patří způsob překládání chybových hlášek nebo třída `AbstractRemoteRepository` sloužící k zobecnění získávání entit z jiných mikroslužeb. Tyto problémy se momentálně v každém modulu řeší trochu jiným způsobem a do budoucna by se mohla řešení sjednotit.

Třída `TranslatableException` umožňuje vkládat do výjimky strukturovanou zprávu, která poté půjde celá najednou přeložit. Obsahuje totiž pole objektů - částí zprávy, které si udržují samotnou zprávu a její argumenty. Konkrétnější výjimky, které z ní dědí, je možné odchytit a vyhodit obecnější výjimku, čímž dojde k řetězení zpráv. Výsledkem může být zpráva „Kontrola platnosti šablony selhala: Neplatná statická otázka: Daná otázka je v šabloně použita dvakrát: {id}.“, přičemž {*id*} se nahradí argumentem zadaným při vyhazování první výjimky. Přeložená zpráva jde poté vrátit pomocí `TranslationService`.

`AbstractRemoteRepository` obsahuje prozatím metody *findById* a *findByIds*, které úmyslně názvem připomínají metody ze skutečných repozitářů. Konkrétním vzdáleným repozitářům poté stačí specifikovat typ objektu, který má být deserializován, a do proměnných prostředí přidat URL, na které se pošle požadavek. Mohou si zároveň přidat vlastní metody, pokud to bude potřeba. Získávání dat z jiných mikroslužeb je častým úkonem a tato třída zamezuje duplikaci kódu.

### 3.2.3 Průběh testu

#### 3.2.3.1 Struktura

Diagram 3.2 opět ukazuje rozdělení do balíčků. Komentovat má smysl jen ty, které se v Test Templates nevyskytovaly nebo zde fungují jinak.

- **DynamicQuestionGenerator** – Zde už se otázky negenerují pouze pro validace, ale pro vložení do studentských testů.
- **Enum** – Výčtové typy jako jsou typ termínu nebo stav testu. Ve vnořeném balíčku jsou výčty operací, které mohou být prováděny při úpravě některých entit, jejich význam bude popsán níže.
- **Message** – Zprávy posílané asynchronně.
- **NotificationSender** – Stará se o zasílání notifikací.
- **Service/MessageHandler** – Třídy zpracovávající zprávy z balíčku Message.
- **StudentTestGenerator** – Generuje studentské testy do termínu nebo nový pokus demotestu. Jeho podadresáře obsahují možné strategie přiřazování studentů do místností a k variantám.
- **UpdateOperation** – S některými entitami je možné provádět více druhů úprav v rámci jednoho endpointu. Zde obsažené třídy identifikují, o kterou operaci jde, zda je možné ji provést a případně ji provedou.
- **ScheduledCommand** – Příkazy, které se pravidelně pouští a mají za úkol identifikovat a zastavit termíny a testy, u kterých vypršel čas. Způsob jejich pouštění bude opět popsán dále.
- **Security/Voter** – Část logiky řízení přístupu ke zdrojům, ve které má neoprávněný pokus vrátit HTTP status kód 403. Např. studentský test si může zobrazit pouze učitel nebo přiřazený student.

#### 3.2.3.2 Popis

Tato část je ze všech tří zdaleka největší, pokud se jedná o množství kódu. Kromě překladů a vzdálených repozitářů popsaných výše se zde používají i další řešení, která si zaslouží popis.

Kontrola vypršení času u termínů a testů je realizována pomocí systémového plánovače úloh - cronu. Mezitím co většina mikroslužeb používá stejný Dockerfile, kvůli podpoře cronu bylo nutné pro Tests vytvořit nový, který z původního vychází. Ten obsahuje navíc instalaci cronu do prostředí, jeho spuštění a konfiguraci tak, aby každou minutu kontroloval, které příkazy se mají provést. Příkazy v adresáři `ScheduledCommand` jsou rovněž nastavené

tak, aby se prováděly každou minutu a po svém spuštění zkontrolují hodnoty ve sloupcích *expected_end_at* v příslušných tabulkách. Pokud je nějaký čas nastavený a již odkazuje na minulost, dojde k ukončení příslušné entity. Tohle řešení není ideální ze dvou důvodů. Zaprvé se cron správně nakonfiguruje jen při vytvoření Docker kontejneru, po jeho zastavení a opětovném spuštění už příkazy nebudou prováděny. Tento fakt je nepohodlný pro vývoj a testování, ale pro nasazení není překážkou, jelikož kontejnery se na serveru vždy znovu vytváří. Zadruhé není cron technologie, která by se hodila pro plánování s přesností na vteřiny, zde použitá jedna minuta je minimální interval. Studenti by tedy mohli test vyplňovat v některých případech i necelou minutu po vypršení času. Pokud ale frontend podporuje funkci automatického poslání požadavku na ukončení zobrazeného testu, když vyprší čas, problém se z velké části vyřeší. Nejedná se tedy o zásadní překážku.

Generování studentských testů využívá návrhový vzor *Strategy* pro způsoby přiřazování studentů do místností a k variantám testu. Použitá strategie se nastavuje mimo kód v souboru `services.yaml` a je možno použít jinou strategii pro přiřazování k variantám při generování všech testů a při generování nového pokusu demo testu. Strategie, které už jsou pro ukázku implementované, jsou přiřazování do variant náhodně nebo rovnoměrně a přiřazování do místností tak, aby se plnily od začátku seznamu do plné kapacity. Jednou z motivací pro mazání vygenerovaných testů po úpravě dosud nespuštěného termínu bylo i zajištění, že všechny testy budou vytvořeny v souladu s vybranou strategií. Pokud by se některé generovaly dodatečně, nemuselo by tomu tak být.

Zároveň generování testů funguje asynchronně. K realizaci byla použita komponenta Symfony Messenger. Požadavek je uložen do databáze a uživateli se vrátí cesta, kde může sledovat jeho stav. Následně je vytvořena zpráva obsahující identifikátor požadavku (předávání celých Doctrine entit ve zprávě způsobuje chyby), která se zařadí do fronty, odkud si ji vyzvedne třída s názvem `GenerateTestsRequestHandler` a zpracuje příslušný požadavek. Implementace v tuto chvíli nevyužívá možnost uložení případných chybových hlášek. Konfigurace opět probíhá přes Dockerfile a provede se při vytvoření kontejneru, nikoliv při jeho opětovném spuštění. Konfigurační skript tentokrát vytvořil Jakub Pavličko, který ho používá i v systému pro hodnocení testů.

Při pohledu na popis API v kapitole 2.5.4 a na jeho kompletní dokumentaci je vidět, že u termínů, studentských testů a přiřazených studentů je možné provádět najednou více druhů úpravy skrze jeden endpoint. Např. lze studentovi kontrolovat kartu před začátkem zkoušky, ale také si student může sám vyžádat ústní zkoušku. Každá z operací vyžaduje jiná oprávnění a dokonce je lze provádět v jiných fázích životního cyklu termínu. Proto implementace umožňuje provádět vždy jen jednu operaci naráz. Požadavek na provedení více z nich najednou skončí odpovědí s chybou. Při přijetí takového požadavku se tedy nejdříve identifikuje, o kterou operaci jde (balíček `UpdateOperation/Identifier`), poté se zkontrolují přístupová práva v pří-

slušné třídě ve `Voter`. Následně se podle druhu operace získá třída z balíčku `UpdateOperation/Updater`, jejíž vrácení zajistí návrhový vzor *Factory* a která zkontroluje, že operace není konfliktní (např. pokus o zastavení dosud nespuštěného testu) a nakonec ji provede.

Posílání notifikací při automatickém ukončení termínu nebo testu a asynchronní generování testů naráží na problém s autorizací. Pro generování dynamických otázek v rámci Assignments a pro poslání notifikace v rámci Configurations je potřeba dodat token. Standardně se při volání jiné mikroslužby přepoužívá token z původního volání. U výše zmíněných operací ale tento postup použít nelze, protože se provádí později a ukládat původní token do databáze není vhodné, navíc by již nemusel být platný. V plánu je zavedení systémového uživatele, jehož token by se dal získat z Auth mikroslužby a to jen v rámci interní sítě. V době odevzdání této práce se na tomto řešení ještě nezačalo pracovat, a tak je v kódu natvrdo použit testovací token. To je v pořádku pro testování, pro produkci však ne. Je to ovšem jediné řešení, které se v dané chvíli nabízelo.

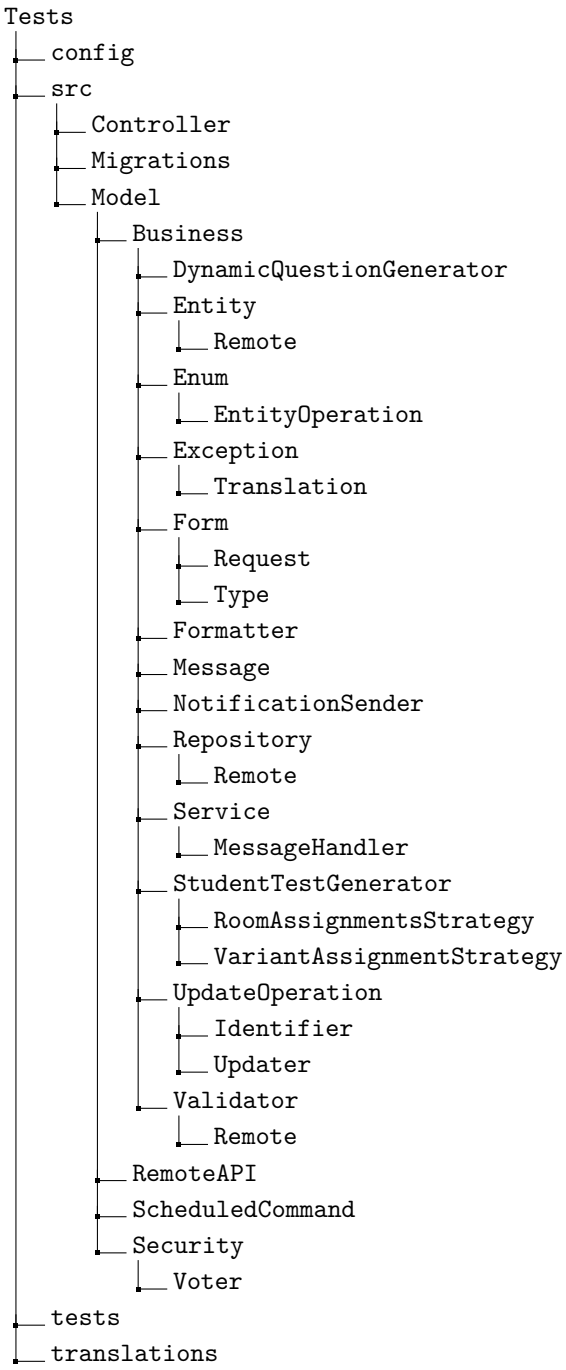
### 3.2.4 Notifikace

#### 3.2.4.1 Struktura

Notifikace byly implementovány v rámci Configurations mikroslužby a jelikož se jedná o vcelku jednoduchou funkčnost, stačilo nové třídy zakomponovat do stávajících balíčků. Diagram by neukázal nic nového.

#### 3.2.4.2 Popis

Notifikace se prozatím ukládají do databáze, aby mohly být zobrazeny frontendem, zatímco posílání emailů nebylo řešeno. Nastavení, kterými je možné s odběrem notifikací manipulovat, se jmenují vždy podle typu notifikace s předponou „notification“, tedy např. „notification_test_evaluated“.




Obrázek 3.2: Struktura balíčků systému pro průběh testu.

Kapitola 4

# Testování

K implementaci neoddělitelně patří i testování. Kvalitně vytvořené testy mohou zavčas odhalit chyby v aplikaci, které mohou mít zejména u velkých projektů nedozírné následky. Také poskytují zpětnou vazbu. Pokud vývojář udělá změny v části systému, nechá proběhnout určité testy, aby měl určitou úroveň jistoty, že změna negativně neovlivnila funkcionality zbylé části. Druhů automatických testů je celá řada od rychlých jednotkových testů, které testují aplikaci po nejmenších částech až po skripty vytvořené pro specializované nástroje, jako je Robot framework, které simulují interakci s frontendem v prohlížeči [37].

Pro použití v tomto projektu připadalo v úvahu mimo jiné použití ručních vývojářských testů, tedy ručního volání endpointů nebo proklikávání frontendu. Kromě toho se počítalo i s automatickými testy. Ty obvykle obsahují části týkající se přípravy dat, provedení testované operace a následné kontroly výsledků. Pro účely testování zde vytvořených mikroslužeb bude stačit popis dvou typů testů.

- **Jednotkové testy** – Anglicky zvané *unit tests*. Zaměřují se většinou na jedinou metodu, přičemž případné závislosti dané třídy jsou tzv. mockované. To znamená, že místo reálných objektů jiných tříd se použijí mocky - objekty, které se nakonfigurují, aby na určitý vstup odpovídaly určitým výstupem, ale samy nic nepočítají. Díky tomu je chování testované třídy izolované a v případě mockování databázového připojení je test i mnohem rychlejší [38].

- **Integrační testy** – Tyto testy se týkají více komponent najednou. Zkoumá se, jak spolu tyto komponenty fungují jako celek, případně jak fungují při připojení k reálné databázi atd. Bývají složitější na přípravu a pomalejší.

## 4.1 Použité testy a pokrytí

Jak bylo řečeno na začátku předchozí kapitoly, nejdříve se implementovala jednoduchá logika a CRUD operace s požadavkem na brzké dodání. V této chvíli postačovalo provolávat ručně jednotlivé endpointy a kontrolovat návratové kódy a těla odpovědí. To samozřejmě nemohlo stačit ve chvíli, kdy se přidala složitější logika. Tehdy se přidaly automatické testy, avšak testování skrze endpointy probíhalo i nadále. Součástí testování byla i komunikace s vývojáři frontendu, kteří s backendem také pracovali, a zapracování jejich zpětné vazby.

Ve všech třech vytvářených komponentách byly v kódu použity dva výše popsané typy automatických testů s pomocí frameworku PHPUnit. Unit testy pokryly třídy patřící do úrovně domény a integrační testy se použily pro úroveň infrastruktury neboli repozitáře. Struktura balíčků testů kopíruje balíčky v samotném kódu, aby byly testy snáze dohledatelné. Přibyl balíček `Data`, který obsahuje třídy vracející data, se kterými mohou testy pracovat. Pokud test pracuje s databází, musí si data sám uložit skrze Doctrine. Alternativou by bylo použití tzv. Fixtures, kdy by byla práce s databází jednodušší. V případě jednotkových testů, kterých je většina a ve kterých se s databází nepracuje, by ale došlo ke zbytečnému zpomalení.

Nestihlo se provést testování zbylých části, které dohromady tvoří většinu kódu avšak menší část složité business logiky. Testy by si ale správně zasloužily i ony. Třídy v balíčku `Service` by šlo pokrýt jednotkovými testy, kontroléry by zase šlo provolat a kontrolovat výsledek, který se zpracovával celou mikroslužbou.

Pokrytí pro jednotlivé mikroslužby je znázorněno v tabulce 4.1. Už na první pohled nejde o ideální čísla, obecně se doporučuje pokrytí kolem 70-80% [39]. Kód řešící notifikace nemá smysl zobrazovat, jednak je ho relativně malé množství a jednak je součástí Configurations mikroslužby, která nemá zbytek otestovaný vůbec, čísla by tedy nebyla relevantní.

| Mikroslužba | Třídy (%) | Metody (%) | Řádky (%) |
|---|---|---|---|
| Test Templates | 38.16 | 44.70 | 36.60 |
| Tests | 34.27 | 47.06 | 37.73 |

Tabulka 4.1: Pokrytí testy.

V ukázce 4.1 je vidět jednotkový testující metodu `canBeUsedInATemplate` ze třídy `QuestionValidator`. Jsou dobře patrné všechny tři části zmíněné výše. Součástí přípravy dat je i nastavení namockovaného vzdáleného repozitáře.

```
public function testCanBeUsedInATemplateNotFound(): void
{
  $questionId = 1;
  $caller = FakeUserInfo::createCaller();

  $this->mockedRepository->expects($this->once())
    ->method('findById')
    ->with(
      $questionId,
      $caller->getToken()
    )->willReturn(null);

  $result = $this->validator->canBeUsedInATemplate($questionId,
    $caller);

  self::assertFalse($result);
}
```

Ukázka kódu 4.1: Ukázka jednotkového testu.

V následující kapitole bude popsána ještě jedna fáze testování, kdy byla kontrolována funkčnost po nasazení na server.

Kapitola 5

# Zkušební nasazení

Dalším krokem softwarového procesu je distribuce vytvořené aplikace směrem k zákazníkovi. V případě webové aplikace je nutné, aby kód běžel na nějakém serveru, na který bude mít zákazník přístup. Provozování aplikace na produkci má také svá specifika, nelze ji pouze na serveru pustit stejně, jako se to dělá při lokálním vývoji. I pokud by zákazník takto nasazenou aplikaci dokázal používat, neslo by to s sebou bezpečnostní rizika [40].

## 5.1 Postup

Zkušební nasazení backendu bylo provedeno na server dbs3.fit.cvut.cz, který je rovněž používán pro nasazení části nového frontendu v rámci bakalářské práce Volhy Chukavy řešící DevOps koncepty na frontendu.

Vzhledem k tomu, že se jedná pouze o zkušební nasazení, nebyl proces integrován do Gitlab CI/CD. Navíc se do budoucna plánuje rozdělit backend do více repozitářů podle jednotlivých mikroslužeb (a případně některé mikroslužby sjednotit), což znamená, že by se proces musel z větší části měnit. Tato kapitola tedy hlavně ověřuje, že je jednotlivé komponenty možné nasazovat a komunikace mezi nimi probíhá správně i na produkčním sestavení.

Prvním krokem bylo vytvořit pro každou mikroslužbu nový Dockerfile. Během vývoje nevadí sdílení jednoho Docker image mezi mikroslužbami, pro nasazení je ale jednak potřeba obrazy samostatně verzovat a jednat v sobě musí obsahovat kód a závislosti. Obrazy vytvořené z těchto Dockerfiles byly následně nahrány do repozitáře v rámci Gitlabu. Pro automatizaci tohoto procesu slouží skript `build-images.sh` ukázaný v kódu 5.1. V něm se nejprve definují sestavované mikroslužby a jejich verze. Následně se pustí vývojová verze mikroslužeb, v rámci každého kontejneru se nainstalují závislosti a z příslušného Dockerfile (obsahujícího instrukci pro zkopírování zdrojového kódu) se sestaví a pushne nový obraz. Tento skript se pouští lokálně na vývojářském stroji, nikoliv na serveru.

```
#!/bin/bash

cd "$(dirname "$(readlink -f "${BASH_SOURCE[0]}")")"
cd ../..

# Built microservices - name, prefix of the produced image, directory, version fo the produced image
microservices=(
  "assignments assignments Assignments 1.0.0"
  "auth auth Auth 1.0.0"
  "configurations configurations Configurations 1.0.0"
  "connections connections Connections 1.0.0"
  "tags tags Tags 1.0.0"
  "test_evaluations test-evaluations TestEvaluations 1.0.0"
  "test_templates test-templates TestTemplates 1.0.0"
  "tests tests Tests 1.0.0"
)

git pull

# For installing dependencies
docker-compose down
cp docker/docker-compose.yml .
docker-compose up -d

# Building image for each microservice
for microservice in "${microservices[@]}"; do
  name=$(echo "$microservice" | awk '{print $1}')
  prefix=$(echo "$microservice" | awk '{print $2}')
  directory=$(echo "$microservice" | awk '{print $3}')
  version=$(echo "$microservice" | awk '{print $4}')

  echo "----------------------------"
  echo "Building $name".
  echo "----------------------------"

  # Skip if the same version already exists
  if docker manifest inspect "gitlab.fit.cvut.cz:5000/dbs/dbs-microservices/$prefix-deploy:$version" > /dev/null; then
    echo "Skipping $name because the same version already exists in the registry."
    continue
  fi

  # Installing dependencies
  docker-compose exec "$name" sudo -HEu web composer install

  # Building and pushing image
  cd ./"$directory"
  docker build -t "gitlab.fit.cvut.cz:5000/dbs/dbs-microservices/$prefix-deploy:$version" -f ../docker/deploy/Dockerfile.$prefix .
  docker push "gitlab.fit.cvut.cz:5000/dbs/dbs-microservices/$prefix-deploy:$version"
```

```
49
50      cd ..
51
52      echo "----------------------------"
53      echo "Finished building $name".
54      echo "----------------------------"
55  done
56
57  # Cleaning
58  docker-compose down
```

Ukázka kódu 5.1: Skript build-images.sh

Následně byla vytvořena upravená verze souboru `docker-compose.yml`. Ta se od původní verze používané pro vývoj liší použitím jiných Docker images, vypnutým debuggerem, odstraněným portem pro přístup k Traefik dashboardu nebo odstraněnými Docker volumes. Ty ve vývojové verzi slouží pro namapování adresáře s kódem dovnitř kontejneru, zde už ale nejsou žádoucí. Výjimkou je mikroslužba Assignments, která používá volume pro ukládání obrázků jako součástí zadání. Tyto soubory nemohou být v kontejneru, aby se nesmazaly při restartu. Upravena byla také konfigurace Traefiku, aby nebylo možné volat napřímo jeho endpointy, jako je např. `/api/entrypoints`.

Druhým vytvořeným skriptem je `deploy.sh` (ukázka 5.1), jenž se pouští po naklonování repozitáře na serveru. Ten nastaví upravenou konfiguraci Traefiku, pustí specifikované mikroslužby z upraveného `docker-compose.yml`, v případě nutnosti připraví úložiště pro obrázky a pro každou mikroslužbu pustí databázové migrace, které mohly potenciálně přibýt v nové verzi.

Postup při vytvoření nové verze některé mikroslužby je následující:

1. V `build-images.sh` zvednout verzi u dané mikroslužby.
2. Skript lokálně pustit. Předtím je nutné být přihlášený do repozitáře s obrazy (`docker login gitlab.fit.cvut.cz:5000`).
3. Upravit v produkční verzi konfigurace Docker Compose verzi obrazu.
4. Na serveru pustit `deploy.sh`, opět je nutné být přihlášen do repozitáře.

```
1   #!/bin/bash
2
3   cd "$(dirname "$(readlink -f "${BASH_SOURCE[0]}")")"
4   cd ../..
5
6   # Only those microservices that have a database
7   microservices=(
8     "assignments"
9     "configurations"
10    "connections"
11    "tags"
12    "test_evaluations"
```

```
13      "test_templates"
14      "tests"
15  )
16
17  git pull
18
19  # Use the correct traefik configuration
20  cp docker/traefik/traefik-deployment.yaml docker/traefik/
      traefik.yaml
21
22  docker-compose down
23  cp docker/deploy/docker-compose.yml .
24  docker-compose up -d
25
26  # Creating a folder for images in assignments - will be mapped
      as a volume with compose
27  uploads_path="../uploads"
28  images_path="$uploads_path/assignment_images"
29  chmod -R 777 $uploads_path
30  mkdir -p $images_path
31  chmod -R 777 $images_path
32
33  # Running migrations for each microservice
34  for microservice in "${microservices[@]}"; do
35    echo "----------------------------"
36    echo "Running migrations for $microservice".
37    echo "----------------------------"
38
39    docker-compose exec "$microservice" sudo -HEu web php bin/
      console --no-interaction do:mi:mi
40  done
```

Ukázka kódu 5.2: Skript deploy.sh

Pro nasazení frontendu navrhla Volha Chukava odlišný způsob i s použitím Gitlab CI/CD, přičemž frontend se skládá pouze z jednoho obrazu [41]. V jejím případě se nejednalo o nasazení zkušební ale o návrh celého procesu, který by se později měl pro frontend používat.

Po nasazení současných verzí mikroslužeb Assignments, Auth, Configurations, Connections, Tags, Test Evaluations, Tests a Test Templates byl proveden test funkčnosti celého business procesu i na produkční verzi. Mikroslužba SwConfigurations, která je již také v pokročilém stádiu vývoje, nasazena nebyla, jelikož s ní testový modul nijak neinteraguje. Přidat ji na server je ovšem velmi jednoduché. Ideální by bylo testovat i interakci s frontendem, práce Dany Suchomelové ovšem nebyla v této chvíli přítomna v master větvi repozitáře frontendu, která se pomocí Gitlab CI/CD nasazuje. Testování s frontendem tedy bylo možné jen na lokálním prostředí, na serveru se použilo API.

Na serveru byl importován semestr, vytvořena zadání, skupiny štítků, štítky patřící do skupin a otázky s těmito štítky. Dále byla založena testová šablona s jednou statickou a jednou dynamickou otázkou a proběhla kontrola, že šablona prochází validací. Poté byla do vytvořeného testového termínu

přidána šablona jako statická varianta, byli přidáni studenti a místnosti. Po asynchronním vygenerování testů byl termín spuštěn. Jeden ze studentů začal pracovat na svém testu a odpověděl na otázku. Po automatickém ukončení testu byla otázka ručně ohodnocena. Během celého procesu byla skrze API kontrolována vytvořená data a spolupráce všech mikroslužeb, přičemž pro autorizaci byly používány testovací přístupové tokeny s příslušnými rolemi - student, učitel, ... Ukázalo se, že celý proces proběhl v pořádku dle očekávání a integrace mezi mikroslužbami zafungovala. Kolekce použitých requestů je k dispozici na přiloženém médiu a na Redminu spolu s kolekcemi obsahující vzorová volání pro množství dalších endpointů.

Kapitola 6

# Budoucí vývoj

Konečný stav mikroslužeb tvořených přímo v této práci je následující. Testové šablony jsou plně funkční skrze API, učitel si je může zobrazovat, vytvářet a upravovat. Lze do nich vkládat statické i dynamické otázky a manipulovat s nimi. Je implementovaná logika jejich validace, která se pouští na vyžádání. API systému pro průběh testu umožňuje projít celým procesem. Učitel je schopen vytvořit termín, přiřadit do něj šablonu jakožto statickou nebo dynamickou variantu testu, studenty, místnosti a případně importovat studenty ze zkoušky v KOSu. Lze vytvořit i demotesty. Funguje asynchronní vygenerování studentských testů včetně použití otázek ze štítků a zjištění stavu vytvořeného požadavku na generování. Termín lze spustit, studenti jsou schopni si test zobrazit a začít na něm pracovat. Během toho se dá test ukončit předčasně, případně může učitel přidávat čas k jednotlivým testům nebo ke všem najednou. Po vypršení času dojde k automatickému ukončení. Studentům přijde do systému notifikace o dostupném testu atd. a přidávat další typy notifikací je velmi jednoduché.

Naopak se nestihlo zajistit, aby student v průběhu testu nemohl zobrazovat jiné části portálu jako demotesty a semestrální práci. Asynchronní generování testů nepracuje s chybovými hláškami a spolu s posíláním notifikací při automatickém ukončení termínu nebo testu využívá testovací token kvůli chybějící funkčnosti v mikroslužbě Auth starající se o autorizaci uživatelů. Tu je nutné vyřešit před spuštěním nového portálu.

Co se týče testového modulu jako celku (tedy včetně prací kolegů), tak integrace je ve stavu, kdy funguje správa štítků a otázek, které se pak používají v testech. Systém pro hodnocení testů komunikuje s Tests mikroslužbou, dokáže vyhodnotit všechny typy otázek automaticky nebo je předat učiteli k ručnímu ohodnocení. Chybí v něm několik funkcionalit jako možnost zobrazit celkové hodnocení pro konkrétní studentský test nebo odesílání notifikací o opravení, které se nestihly. Frontendová část z pohledu studenta umí zobrazovat seznam testů, vyplňovat je a odeslat k hodnocení, byť malé množství volání backendu zůstává namockovaných. Na druhé části, tedy pohledu učitele,

se během akademického roku nakonec vůbec nepracovalo, jelikož SP tým, který ji měl původně na starost, nakonec pomáhal s jinými věcmi. Stejně tak plánovaná mikroslužba na zobrazování statistik o úspěšnosti studentů v testech (a v semestrálních pracích) teprve čeká na to, až se jí někdo bude věnovat, jedná se ale o vysloveně doplňkovou funkčnost.

Zbytek frontendu může být námět pro další závěrečnou práci, drobné dodělávky na backendu jsou potom vhodné pro příští SP týmy. Právě pro tyto týmy byly na Redminu vytvořeny úkoly, které popisují, co je potřeba dodělat. Studenti také mohou využít kolekce exportované z nástroje Postman, které obsahují vzorová volání všech endpointů z mikroslužeb Test Templates a Tests a některá volání z mikroslužeb mých kolegů, která se hodí pro přípravu dat. Kolekce se nachází na přiloženém média a na Redminu.

# Závěr

Cílem dilomové práce bylo podílet se na tvorbě nového portálu pro výuku databází na FIT ČVUT, konkrétně testového modulu a jeho backendu. Byly realizovány systémy pro správu testových šablon, průběh testu a správu notifikací. Další součásti backendu i frontendu vznikaly v závěrečných pracích mých kolegů, se kterými se po celou dobu spolupracovalo.

Byl analyzován současný testový modul, a jelikož této problematice se již věnovalo několik prací v minulých letech, současné případy užití byly popsány stylem srovnání popisu právě z těchto prací a vyjasnění a doplnění některých detailů. V rámci analýzy byly rovněž identifikovány a kategorizovány požadavky na zde vyvíjené systémy.

V další části byla navržena nová podoba uvedených systémů. Popis se zaměřuje na změny oproti stávajícímu stavu ve starém systému, popisu navržené databáze a rozhraní včetně vyzkoušených slepých uliček a zvažování různých variant u konkrétní funkcionality. Při návrhu bylo dbáno na to, aby byla respektována navržená architektura nového DBS portálu.

Proběhla implementace dle návrhu včetně testování. V příslušných kapitolách textu se nachází popis některých technických řešení, druhy použitých testů a jejich pokrytí. Otestovány jsou vrstvy obsahující složitější business logiku pomocí jednotkových a integračních testů, u zbylých částí je pouze nastíněno, jak by mohlo testování probíhat.

Důležitým bodem zadání bylo integrovat zde vytvářené systémy se zbytkem testového modulu a celého portálu. To bylo zajištěno hlavně pravidelnými schůzkami s ostatními vývojáři backendu, vzájemnou revizí návrhu i kódu, během kterých byly eliminovány možné budoucí problémy. Rovněž probíhala komunikace s frontendem, aby se zajistila spolupráce obou komponent.

Pomocí vytvořených skriptů bylo provedeno zkušební nasazení na server dbs3.fit.cvut.cz, na který se rovněž nasazuje nový frontend. Pomocí volání API celého procesu od vytváření otázek přes psaní testů až po jejich hodnocení byla vyzkoušena integrace mikroslužeb i na testovacím serveru.

Na konci je popsán možný budoucí vývoj a shrnuty dosažené výsledky.

Backend testového modulu v tuto chvíli obsahuje všechny důležité funkčnosti, které jsou potřeba pro jeho použití. Lze projít celým procesem od vytváření otázek, testových šablon, termínů a testů samotných (včetně použití otázek generovaných ze štítků). Testy lze spustit, vyplnit a opravit (automaticky a manuálně). Bohužel se nestihlo zajistit, aby student nemohl během testu používat ostatní součásti portálu. Aby bezpečně fungovalo asynchronní generování testů a posílání notifikací, je potřeba nejdříve dořešit autorizaci v rámci celého portálu. Systémy vyvíjené mými kolegy mají také drobné nedodělky, nejedná se ale o zásadní překážku. Frontend vznikl zatím z pohledu studenta, učitelská část se bude teprve tvořit. Po jejím zhotovení a po dodělání několika menších věcí na backendu bude testový modul připraven.

# Literatura


[1] Autoři BI-DBS. `https://dbs.fit.cvut.cz/authors/`, accessed: 2022-12-02.

[2] Plaskach, A.: *Modernizacce a migrace DBS portálu*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

[3] Stephens, R.: *Beginning Software Engineering*. Wrox, první vydání, 2015, ISBN 978-8126555376.

[4] SourceMaking.com: Analysis Paralysis. `https://sourcemaking.com/antipatterns/analysis-paralysis`, accessed: 2023-01-23.

[5] Hanzl, M.: *dbs.fit.cvut.cz - Refaktoring testů I.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

[6] Jordán, P.: *dbs.fit.cvut.cz - Refaktoring testu II.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

[7] Plyskach, A.: *Refaktoring testové části backendu portálu dbs.fit.cvut.cz.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[8] Fedor, T.: *ER Diagrams Web Component II.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[9] Daly, N.: What Is a Use Case. `https://www.wrike.com/blog/what-is-a-use-case/`, accessed: 2023-01-29.

[10] Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Pearson, třetí vydání, 2004, ISBN 978-0131489066.

[11] Seidl, M.; Scholz, M.; Huemer, C.; aj.: *UML @ Classroom: An Introduction to Object-Oriented Modeling*. Springer, 2015, ISBN 978-3319127415.

[12] Ambler, S. W.: UML 2 Use Case Diagramming Guidelines. `http://www.agilemodeling.com/style/useCaseDiagram.htm`, accessed: 2023-02-03.

[13] Richards, M.: *Software Architecture Patterns*. O'Reilly Media, 2015, ISBN 9781491924242.

[14] Welling, L.; Thomson, L.: *PHP and MySQL Web Development*. Addison-Wesley Professional, páté vydání, 2016, ISBN 978-0275967598.

[15] Grudl, D.: Historie Nette. `https://nette.org/cs/history`, accessed: 2023-02-04.

[16] Newman, S.: *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, první vydání, 2019, ISBN 978-1492047841.

[17] Richardson, C.: Pattern: Microservice Architecture. `https://microservices.io/patterns/microservices.html`, accessed: 2022-12-03.

[18] Sommerville, I.: *Software Engineering*. Pearson, 9 vydání, 2010, ISBN 978-0137035151.

[19] Rome, P.: What are Non Functional Requirements — With Examples. `https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples`, accessed: 2023-01-27.

[20] Dyson, J.: Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements. `https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson/`, accessed: 2023-05-25.

[21] McLaughlin, B. D.; Pollice, G.; West, D.: *Head First Object-Oriented Analysis and Design*. O'Reilly Media, první vydání, 2006, ISBN 978-0596008673.

[22] What is Symfony. `https://symfony.com/what-is-symfony`, accessed: 2023-02-14.

[23] Potencier, F.: Open-Source cross-pollination. `https://www.jmix.io/blog/to-delete-or-to-soft-delete-that-is-the-question/`, accessed: 2023-02-14.

[24] Introduction - Composer. `https://getcomposer.org/doc/00-intro.md/`, accessed: 2023-05-22.

[25] Carey, S.: What is Docker? The spark for the container revolution. https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html, accessed: 2023-02-14.

[26] What is a container. https://www.docker.com/resources/what-container/, accessed: 2023-02-14.

[27] Docker Compose overview. https://docs.docker.com/compose/, accessed: 2023-02-14.

[28] PosgreSQL:About. https://www.postgresql.org/about/, accessed: 2023-02-14.

[29] Singh, M.: What is Traefik & How to Learn Traefik? https://www.devopsschool.com/blog/what-is-traefik-how-to-learn-traefik/, accessed: 2023-02-14.

[30] Belyaev, A.: To Delete or to Soft Delete, That is the Question! https://www.jmix.io/blog/to-delete-or-to-soft-delete-that-is-the-question/, accessed: 2023-02-10.

[31] Lidwell, W.; Holden, K.; Butler, J.: *Universal Principles of Design*. Rockport Publishers, 2003, ISBN 978-1592530076.

[32] Nikolaieva, A.: 8 Best Software Development Methodologies. https://www.uptech.team/blog/software-development-methodologies/, accessed: 2023-05-24.

[33] Lutkevich, B.: definition: waterfall model. https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model/, accessed: 2023-05-24.

[34] Beck, K.; a další: Agile Manifesto. https://agilemanifesto.org/, accessed: 2023-05-24.

[35] Stellman, A.; Greene, J.: *Learning Agile: Understanding Scrum, XP, Lean, and Kanban*. O'Reilly Media, první vydání, 2013, ISBN 978-1449331924.

[36] Sutherland, J.; Sutherland, J. J.: *Scrum: The Art of Doing Twice the Work in Half the Time*. Currency, první vydání, 2014, ISBN 978-0385346450.

[37] Pittet, S.: The different types of software testing. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing, accessed: 2023-05-24.

[38] What are mock objects? https://www.agilealliance.org/glossary/mocks, accessed: 2023-06-11.

[39] Arguelles, C.; Ivanković, M.; Bender, A.: Code Coverage Best Practices. https://testing.googleblog.com/2020/08/code-coverage-best-practices.html, accessed: 2023-05-25.

[40] Kazim, W.: What Is Software Deployment? Process and Best Practices. https://learn.g2.com/software-deployment, accessed: 2023-06-19.

[41] Chukava, V.: *DevOps concepts – CI/CD, implementation of authorization & authentication, presented on a BI-DBS portal frontend.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Příloha A

# Seznam použitých zkratek

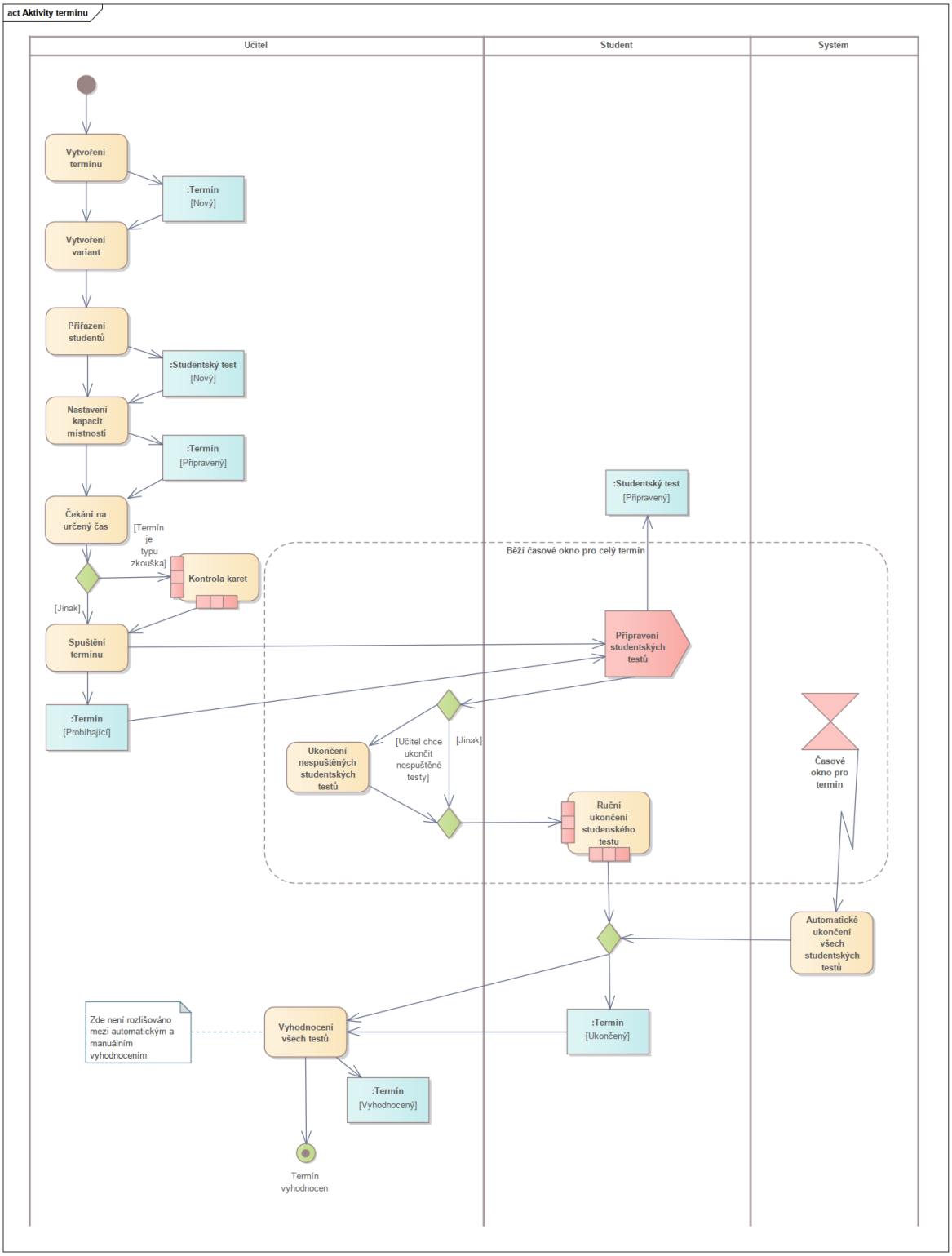
**REST** Representational State Transfer

**HTTP** Hypertext Transfer Protocol

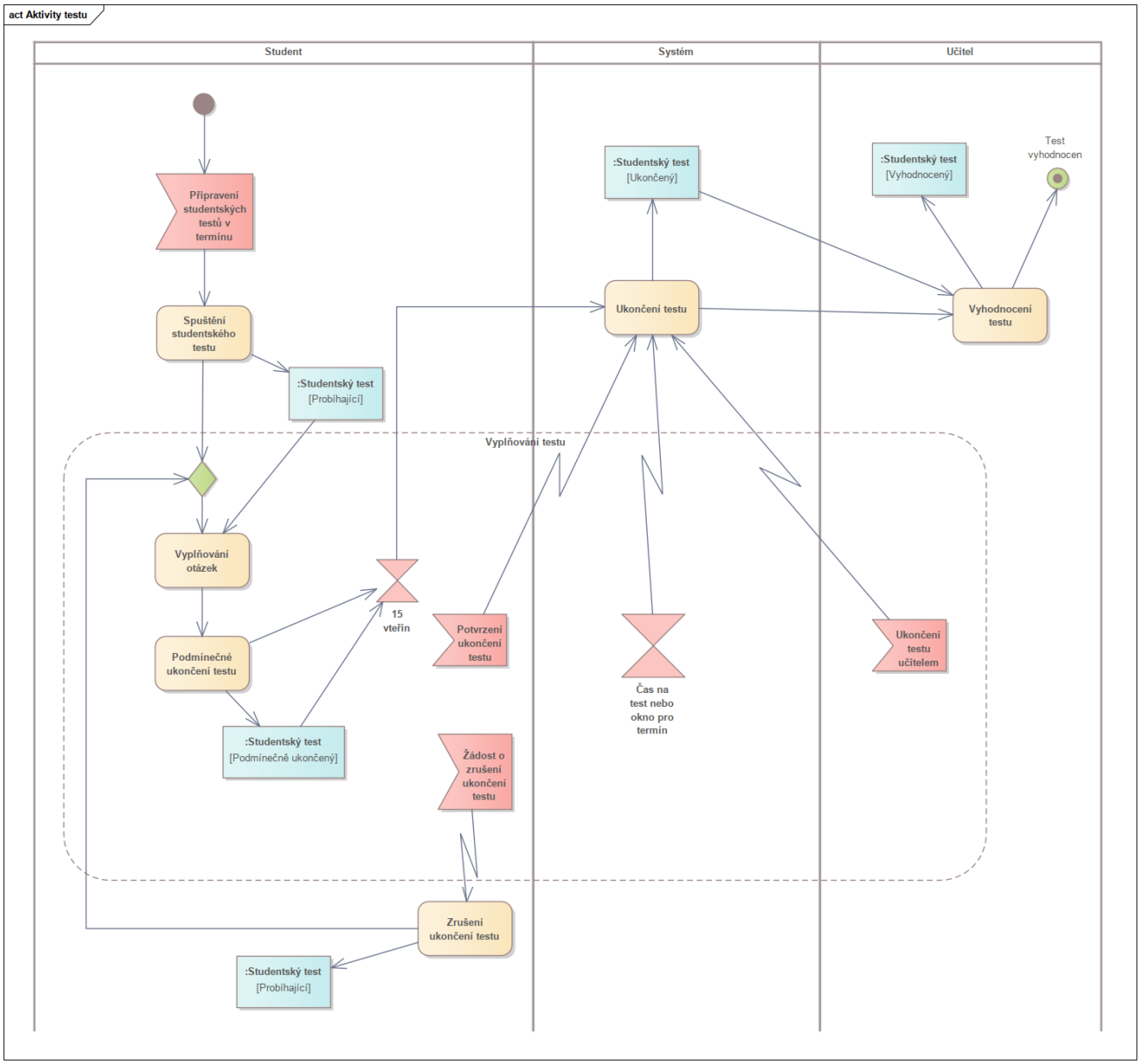
**ORM** Object–Relational Mapping

Příloha B

# Diagramy aktivit



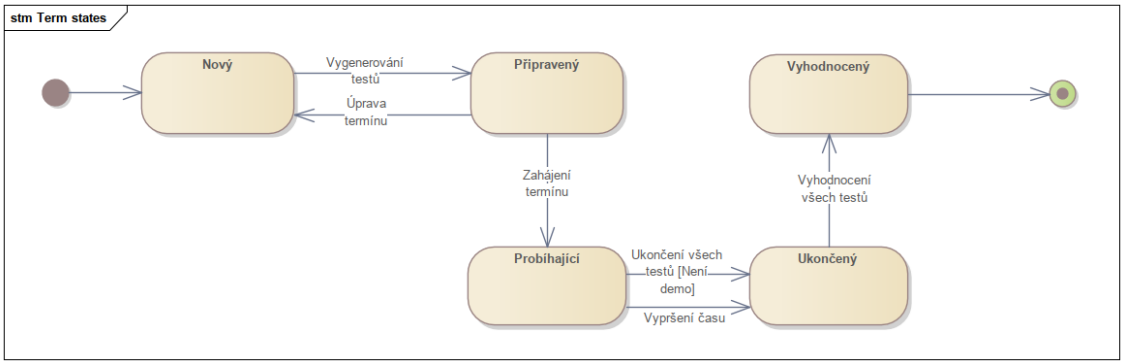

Obrázek B.1: Diagram aktivit průběhu testového termínu.



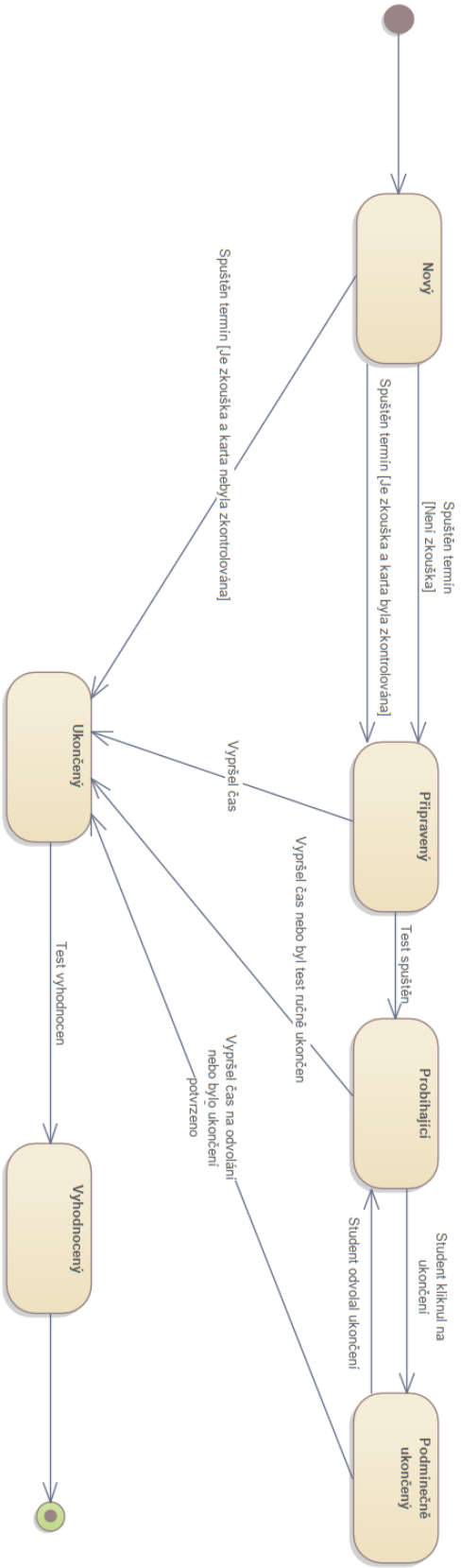

Obrázek B.2: Diagram aktivit psaní testu.

Příloha C

# Stavové diagramy



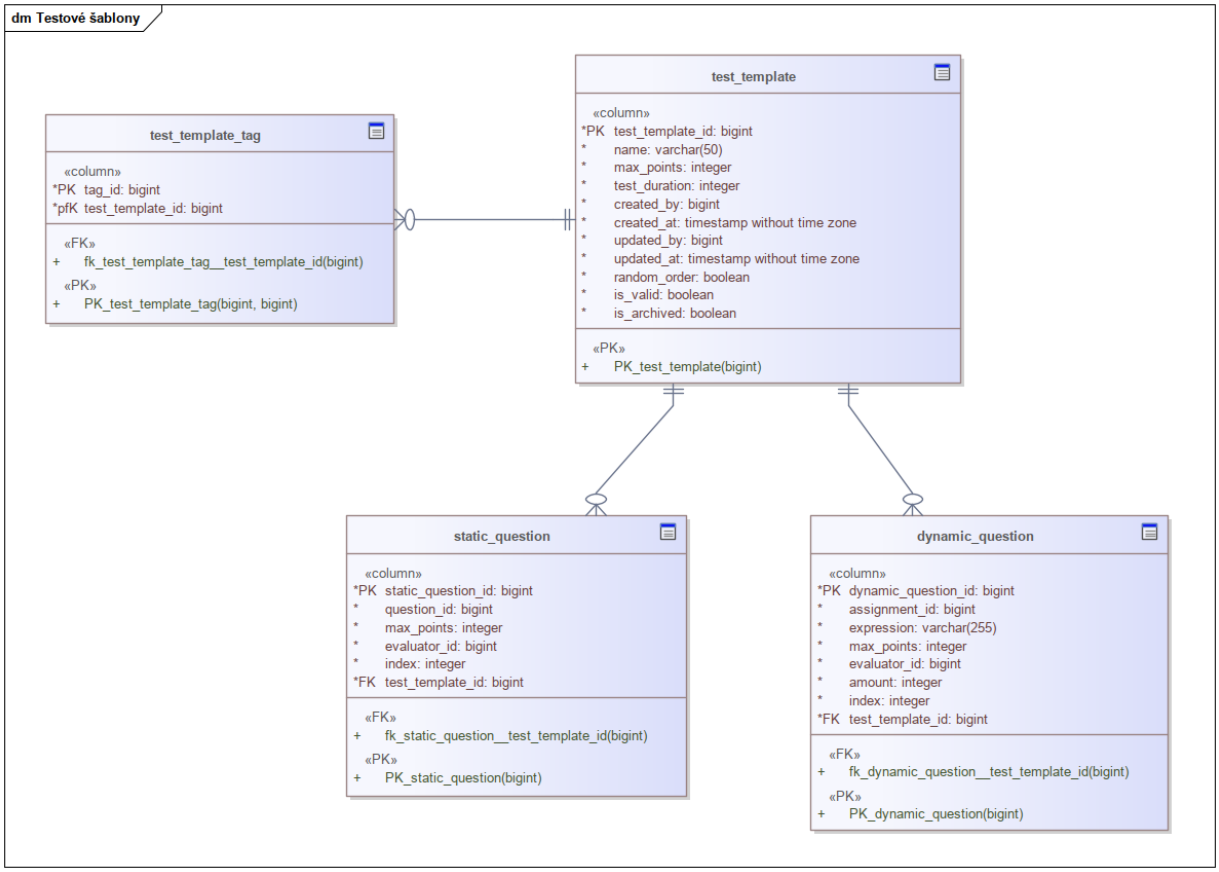

Obrázek C.1: Diagram stavů termínu.



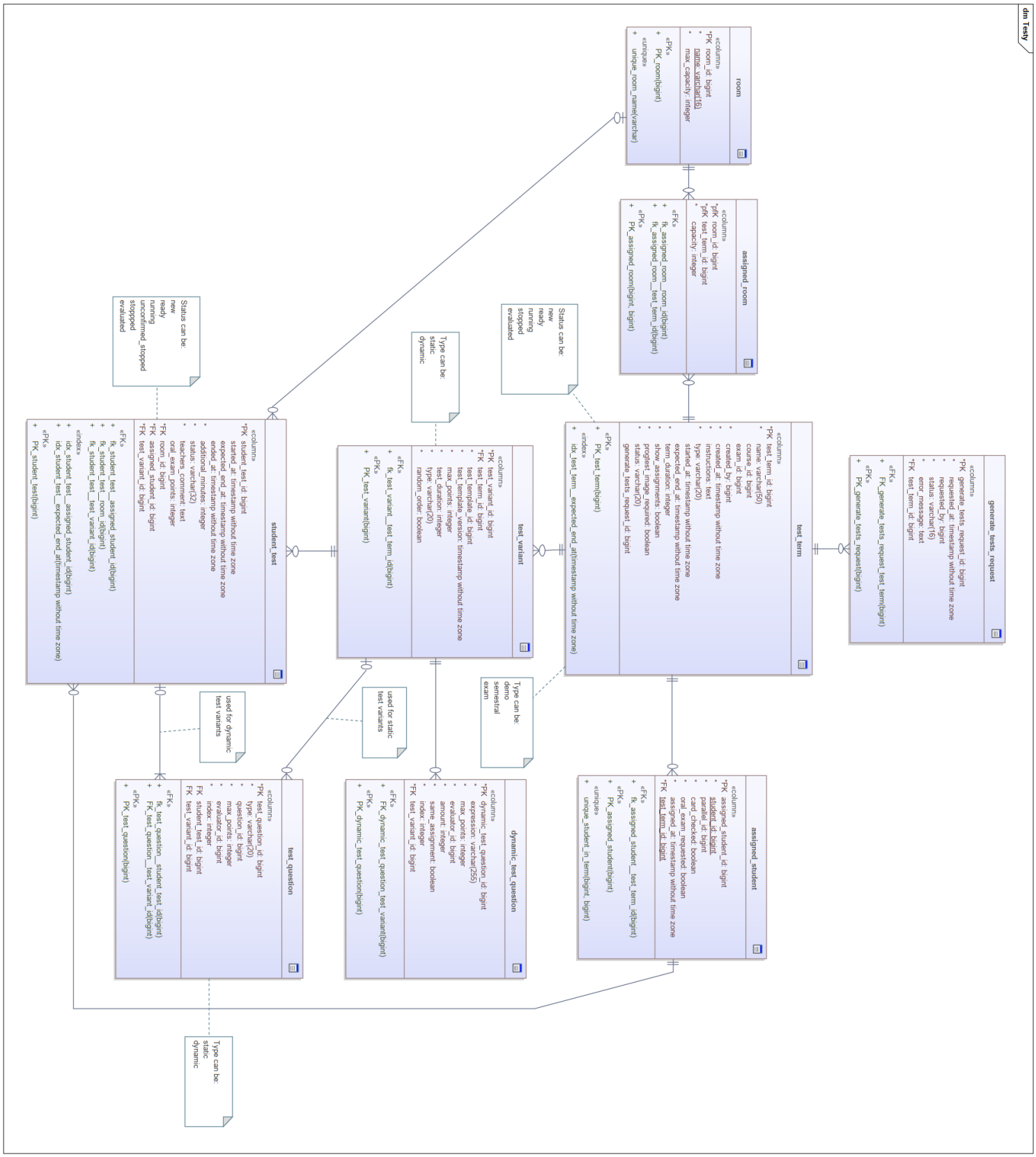

Obrázek C.2: Diagram stavů studentského testu.

Příloha D

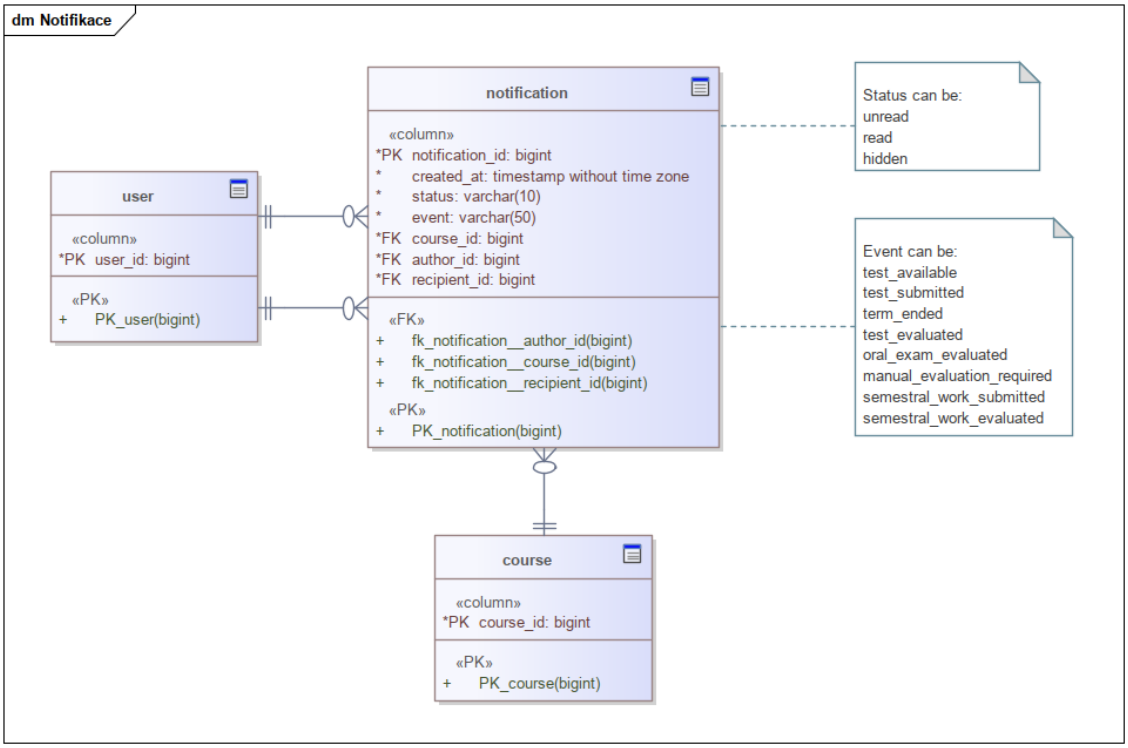
# Relační databázová schémata




Obrázek D.1: Relační schéma databáze mikroslužby správy testových šablon.




Obrázek D.2: Relační schéma databáze mikroslužby pro průběh testu.




Obrázek D.3: Relační schéma části databáze mikroslužby konfigurace předmětu týkající se notifikací.

Příloha **E**

# Seznamy endpointů

# DBS - Test Templates microservice 1.0.0 OAS3

Servers
https://dbs.fit.cvut.cz/test-templates

## template

| | | | |
|---|---|---|---|
| GET | /templates | Returns all (active) templates | getAllTemplates |
| POST | /templates | Adds a new template | addTemplate |
| GET | /templates/{templateId} | Finds a template by id | getTemplateById |
| PATCH | /templates/{templateId} | Updates a template specified by id | updateTemplate |
| DELETE | /templates/{templateId} | Deletes (archives) a template given by id. Deleted template cannot be modified or used, but can be read through id or through /templates/archived | removeTemplate |
| POST | /templates/{templateId}/duplicate | Duplicates a template given by id | duplicateTemplate |
| POST | /templates/{templateId}/check-validity | Checks and updates validity of a template. | checkValidityOfTemplate |
| GET | /templates/archived | Returns all archived (previously deleted) templates | getAllArchivedTemplates |

## tag

| | | | |
|---|---|---|---|
| PUT | /templates/{templateId}/tags | Updates (replaces) tags by id of a template they belong to | replaceTagsOfTemplate |

## static-question

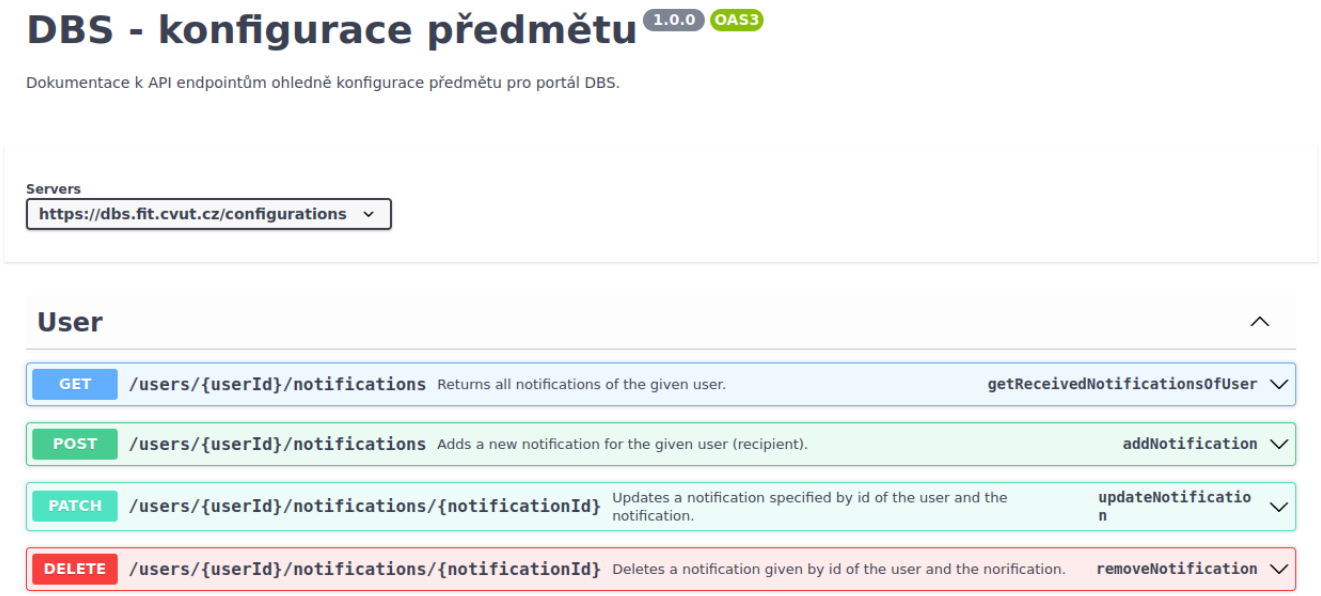
| | | | |
|---|---|---|---|
| POST | /templates/{templateId}/static-questions | Adds a new static question to the specified template | addStaticQuestionToTemplate |
| DELETE | /templates/{templateId}/static-questions | Deletes all static questions assigned to the given template | removeAllStaticQuestionsFromTemplate |
| GET | /templates/{templateId}/static-questions/{staticQuestionId} | Finds a static question by id | getStaticQuestionById |
| PATCH | /templates/{templateId}/static-questions/{staticQuestionId} | Updates a static question of a template specified by template id and static question id | updateStaticQuestionOfTemplate |
| DELETE | /templates/{templateId}/static-questions/{staticQuestionId} | Removes a static question of a template specified by template id and static question id | removeStaticQuestionFromTemplate |

## dynamic-question

| | | | |
|---|---|---|---|
| POST | /templates/{templateId}/dynamic-questions | Adds a new dynamic question to the specified template | addDynamicQuestionToTemplate |
| DELETE | /templates/{templateId}/dynamic-questions | Deletes all dynamic questions assigned to the given template | removeAllDynamicQuestionsFromTemplate |
| GET | /templates/{templateId}/dynamic-questions/{dynamicQuestionId} | Finds a dynamic question by id | getDynamicQuestionById |
| PATCH | /templates/{templateId}/dynamic-questions/{dynamicQuestionId} | Updates a dynamic question of a template specified by template id and dynamic question id | updateDynamicQuestionOfTemplate |
| DELETE | /templates/{templateId}/dynamic-questions/{dynamicQuestionId} | Removes a static question of a template specified by template id and dynamic question id | removeDynamicQuestionFromTemplate |

Obrázek E.1: Seznam endpointů Test Templates mikroslužby.


Obrázek E.2: Seznam endpointů Tests mikroslužby.




Obrázek E.3: Seznam endpointů Configurations mikroslužby týkajících se notifikací.

Příloha F

# Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové kódy
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `docs` ..... dokumentace
  - `test_module.eap` ..... diagramy v nástroji Enterprise Architect
  - `postman.zip` ..... Kolekce Postman requestů na vytvořená API
  - `API` ..... popisy rohraní ve specifikaci OpenAPI
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF