FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Refaktoring testové části backendu portálu dbs.fit.cvut.cz
Student:	Andrii Plyskach
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem této práce je refaktoring testové částí portálu dbs.fit.cvut.cz, dále jen portál. Důvodem je nutná modernizace a pokrok k pohodlnějšímu a efektivnějšímu následnému vývoji.

Postupujte v těchto krocích:
1. Analyzujte současný stav portálu se zaměřením na testovou část. Nezapomeňte na provázanost s mimotestovými částmi.
2. Na základě analýzy důkladně navrhněte strukturu nové databáze testové části portálu.
3. Nad databází postavte vhodný backend. Vzhledem k rozsahu se počítá s prototypovou implementací.
4. Na Vámi implementovaný kód realizujte vhodné testy - minimálně ukázkové jako šablonu pro další vývoj.
5. Navrhněte budoucí směr rozvoje Vámi připraveného prototypu.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

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

V Praze dne 11. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Refaktoring testové části backendu portálu dbs.fit.cvut.cz

Andrii Plyskach

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

4. června 2020

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu, panu Ing. Jiřímu Hunkovi, za možnost volby tohoto tématu a také za veškeré poskytnuté informace tykající se funkcí současného testového modulu DBS portálu. Dále bych chtěl poděkovat Ing. Oldřichu Malcovi a Ing. Pavlu Kovářovi za jejich pomoc v pochopení systému a cenné rády ohledně refaktorování testového modulu. Nakonec bych též rad poděkoval své rodině za jejich podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). 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 zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. června 2020 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 Andrii Plyskach. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

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

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací prototypu nového backendu testového modulu DBS portálu. Na začátku je provedena kompletní analýza funkcí systému z pohledu uživatele pomocí případů užití. Dále se podrobně rozebírají hlavní nedostatky a problémy spojené s uložením dat a také obecné nevýhody současné implementace. Na základě této analýzy je pak upravena a rozšířena struktura databáze a vytvořen návrh testového modulu s důrazem na znovupoužitelnost a snadnou údržbu. Poslední část této práce se věnuje integraci knihoven a implementaci prototypového řešení, včetně jeho testování.

Klíčová slova webový portál, DBS portál, refaktorování, databázové systémy, návrh, PHP, ORM, Doctrine, Nette

Abstract

This bachelor thesis deals with the design and prototype implementation of a new backend for the testing module in the DBS portal. At the beginning, a complete analysis of the system functionality is performed using use cases from the user's point of view. Furthermore, the main shortcomings and problems associated with data storage as well as the general disadvantages of the current implementation are discussed in detail. Based on this analysis, the structure of the database is modified and expanded, and the testing module design is also created with an emphasis on reusability and easy maintenance. The last part of this work deals with library integration and implementation of a prototype solution, including its testing.

Keywords web portal, DBS portal, refactoring, database systems, software design, PHP, ORM, Doctrine, Nette

Obsah

Seznam obrázků

Úvod

Problém refaktorování je vždy aktuální, často se během provozu a rozvoje systému stane, že původní návrh už neodpovídá dnešním požadavkům. Proto se další rozšíření přidávají obtížněji, také velikost práce potřebné pro splnění úkolů pravidelně narůstá. Podobný problém nastal v DBS portálu, systému pro podporu výuky předmětu Databázové systémy (BI-DBS), který je vyučován na Fakultě informačních technologií ČVUT v Praze. Bylo zjištěno, že během častých modifikací systému a pár let provozu, současný design obtížně podporuje další rozšíření, která by se v dnešní době od tohoto systému požadovala. Proto bylo rozhodnuto vytvořit zcela nový testový modul, který by byl odolnější vůči budoucím změnám a nově kladeným požadavkům.

DBS portál byl vyvíjen v rámci předmětů Softwarový týmový projekt 1 a 2 (BI-SP1 a BI-SP2) a závěrečných prací [1–10]. Plánování vývoje systému začalo již v roce 2013 – v tomto roce byl také schválen jeho první návrh. Poprvé byl využit studenty na cvičeních až v letním semestru (LS) 2016, která vedl Ing. Jiří Hunka. V následujících semestrech byl již portál používán všemi cvičícími a od té doby se stále vyvíjí. [5]

Nápad napsat tuto bakalářskou práci vznikl ještě v době, kdy jsem se podílel na vývoji portálu v rámci předmětů BI-SP1 a BI-SP2. V tuto dobu jsem dostal za úkol na první pohled jednoduchou úlohu, přidat nový druh testů do systému. Jak se ukázalo během analýzy, tento úkol byl mnohem složitější. Původní návrh vůbec nepočítal s tím, že by mohl existovat automatický test. Časová náročnost tohoto úkolu byla nakonec násobně vyšší, než se odhadovalo na začátku. Práce na tomto problému a na projektu celkově mě hodně zaujala, proto jsem se rozhodl, že bych mohl systém něčím vylepšit, a to třeba touto bakalářskou práci.

Výstupy této práce by měly zjednodušit práci vývojářům a usnadnit další rozvoj nebo změny v portálu. Práce v jistém smyslu navazuje na diplomovou [9] a bakalářské práce [1–8, 10], které systém vyvíjely, a používá jejich výsledky.

Cílem práce je především návrh a tvorba prototypu backendu pro testový modul DBS portálu. Modul bude též rozšířen o aktuální požadavky. Nejprve bude provedena kompletní analýza současného řešení testového modulu, a to včetně struktury databáze, kde budou popsány současné funkce modulu s důrazem na možné nedostatky a problémy. Dále bude provedena analýza požadavků a specifikace rozšíření, které chceme implementovat navíc oproti současnému stavu. Na základě analýzy bude za prvé navrhnuto vhodné uložení dat a za druhé vytvořen kompletní návrh modulu včetně směru budoucího rozvoje. S využitím navržených modelů se vytvoří částečná implementace a její testování. Nakonec se práce bude zabývat problémy s budoucím integrováním testového modulu se zbytkem portálu.

Kapitola 1

Analýza

V současnosti se analýza stala jedním z nejdůležitějších kroků softwarového procesu. Její důležitost výrazně roste s velikostí projektu. Cílem refaktorování je zlepšení čitelnosti a porozumění kódu a také zvětšení jeho použitelnosti a rozšiřovatelnosti. Toto není možné bez důkladné a podrobné analýzy, nalezení současných problémů a obecného pochopení systému.

Refaktorování obecně je postupné zlepšování designu aplikace pomocí malých kroků bez vlivu na vnější chování systému. Při refaktorování se doporučuje dělat pouze malé změny a po každé takové modifikaci spouštět jednotkové[1] nebo integrační[2] testy, abychom snížili riziko vnesení chyb. Až po tom, co test skončí bezchybně, můžeme začít provádět další změnu. [11] V našem případě tento postup není možné použít, protože potřebujeme modifikovat strukturu databáze. Z tohoto důvodu změna původního kódu ztrácí smysl, jelikož kód je přímo vázán na datovou vrstvu. Proto slovo refaktorování pro nás bude znamenat tvorbu zcela nového testového modulu, který vychází z funkčností současného řešení. Modul navíc bude rozšířen o nové možnosti v souladu s nově kladenými požadavky.

Tato kapitola se věnuje kompletní analýze testového modulu DBS portálu. V podkapitole 1.1 je provedena analýza případů užití. Dále jsou popsané problémy, které se v současné době v systému vyskytují, viz podkapitola 1.2. Kromě toho je součástí této podkapitoly též stručná analýza současného stavu struktury databáze. V poslední podkapitole se práce zabývá specifikací požadavků, které rozšiřují stávající řešení a díky kterým má toto refaktorování smysl.

[1] Jednotkové testy ověřují konkrétní funkčnost v rámci jedné třídy. Veškeré závislosti, které obsahuje testovaná třída, je potřeba nahradit zástupnými kopiemi (např. „Mockování“), pokud to jde.

[2] Integrační testy se používají pro testování větší částí systému, např. testování spolupráci několika komponent.

1.1 Současné řešení

Současné řešení představuje systém učený pro podporu výuky předmětu Databázové systémy (BI-DBS), který se skládá z několika hlavních komponent:

- semestrální práce,
- testový modul,
- databázový modul,
- administrace uživatelů,
- kreslící nástroj.

Jelikož se tato práce bude zabývat testovou částí, bude pro nás důležitá pouze testová komponenta a na ni se v dané kapitole omezíme. Testová část je používána studenty a vyučujícími pro psání cvičných testů, testů v semestru a také zkoušek z předmětu BI-DBS, které se následně zcela automaticky vyhodnocují.

Původně se testy z tohoto předmětu psaly na papír a oprava byla pro učitele extrémně náročná. Proto bylo rozhodnuto implementovat testový modul, který tuto činnost na sebe převezme a alespoň částečně test vyhodnotí.

První implementace probíhala v rámci předmětů Softwarový projekt 1 (BI-SP1) a Softwarový projekt 2 (BI-SP2) a obsahovala množství nedostatků, proto byla využívaná pouze některými cvičícími. V roce 2016 vznikla bakalářská práce [10], jejíž cílem bylo refaktorování tohoto řešení.

V průběhu dalších několika let byl tento modul modifikován a rozšiřován. V současnosti se objevily nové požadavky, které by se obtížně implementovaly. Proto bylo rozhodnuto tyto problémy vyřešit pomocí zlepšení designu celkového modulu. Jelikož už přesně známe veškeré požadavky a také jaké změny se očekávají v budoucnu, můžeme vytvořit návrh tak, aby splňoval podmínky jednoduchosti, nízké provázanosti, rozšířitelnosti a snadného testování.

1.1.1 Aktéři

Aktéři jsou role, které v systému vykonávají určité činnosti. Aktér (účastník) může označovat člověka, skupinu lidí, systém nebo dokonce i čas, který má v systému právo provést nějakou konkrétní akci nebo sekvencí akcí. [12]

V DBS portálu existují následující účastníci: nepřihlášený uživatel, student, vyučující a garant. Jelikož jediná možná akce, která je dostupná nepřihlášenému uživatelovi, je přihlášení, dále se více rozebírat nebude. Kromě toho se táto práce věnuje pouze testovému modulu v DBS portálu, proto se omezíme pouze na aktivity prováděné v této části systému.

1.1.1.1 Student

Role student představuje osobu, která je studentem Českého vysokého učení technického (ČVUT) a má zapsán předmět BI-DBS. V systému může student vyplňovat několik druhů testů: *demo test, automatický demo test, test*

v semestru, zkouška. Na základě těchto testů získává body a pak i výslednou známku.

1.1.1.2 Vyučující

Zodpovědným za tvorbu a správu testů je vyučující. Systém dovoluje učitelům vytvářet nové otázky a zadání, které pak mohou být použity v testech. Vyhodnocení testů je zcela automatické, proto vyučující může automaticky ohodnocené testy procházet a opravovat hodnocení, pokud systém nezvládl ohodnotit otázku nebo ji ohodnotil špatně.

1.1.1.3 Garant

Garant je role, jež má stejná práva jako vyučující. Navíc ale může upravovat podmínky klasifikace, např. počet bodů, kterého je potřeba dosáhnout, aby student uspěl u testu.

1.1.2 Správa zadání

V této podkapitole je popsána tvorba testových zadání. Diagram případů užití je zachycen na obrázku 1.1.

1.1.2.1 UC1: Zobrazení seznamu zadání

Případ užití umožňuje uživateli zobrazit seznam zadání, které v systému existují.

Aktéři: vyučující, garant.

Scénář:

1. Případ užití začíná, když se uživatel rozhodne podívat se na všechna testová zadání v systému.
2. Uživatel zvolí v levém menu tlačítko „*Zobrazit zadání*“.
3. Systém zobrazí veškerá existující testová zadání.

1.1.2.2 UC2: Zobrazení konkrétního zadání

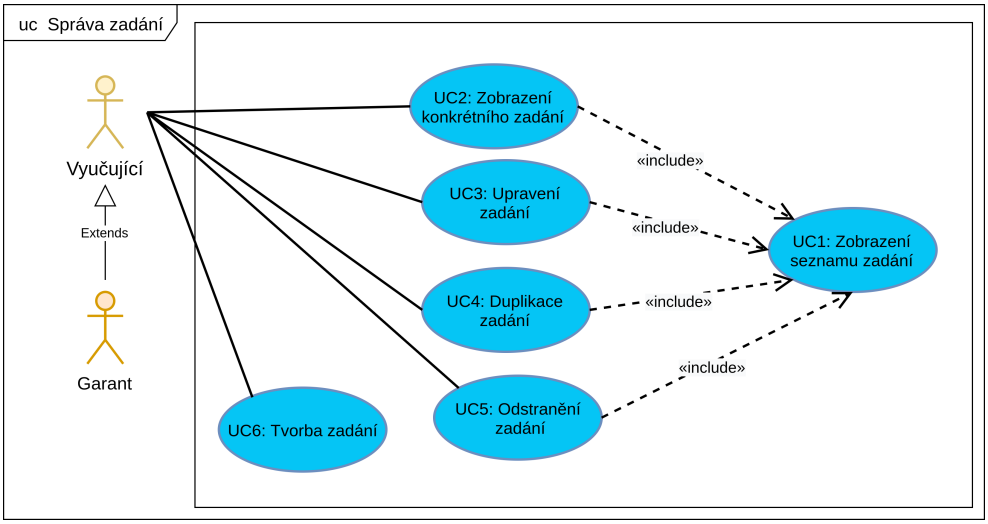
Případ užití umožní vyučujícímu zobrazit popis konkrétního zadání.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedno zadání.

Scénář:

1. Uživatel projde případem užití ”UC1: Zobrazení seznamu zadání”.
2. Ze seznamu vybere konkrétní zadání a klikne na „*Náhled*“.
3. Systém zobrazí informace o vybraném zadání.



Obrázek 1.1: Diagram případů užití: Správa testových zadání

1.1.2.3 UC3: Upravení zadání

Systém dovoluje garantovi a učiteli modifikovat zadání. Pokud zadání obsahuje alespoň jednu otázku, pak se před úpravou systém zeptá, jestli uživatel opravdu chce akci provést.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedno zadání.

Hlavní scénář:

1. Uživatel projde případem užití „UC1: Zobrazení seznamu zadání“.
2. Uživatel vyhledá v seznamu zadání potřebné a klikne „*Upravit*“.
3. Systém zobrazí formulář v závislosti na konkrétním typu zadání, které je buď: *text*, *diagram*, *obrázek* nebo *normalizace relačního schématu*.
4. Pokud zadání bylo textové, pak systém zobrazí formulář s údaji: název, popis a jazyk.
5. Uživatel upraví jednotlivá políčka a zvolí „*Uložit*“ nebo odmítne změny pomocí tlačítka „*Zpět*“.

Alternativní scénář: zvolené zadání bylo typu „Diagram“.

1. Scénář začíná ve 3. kroku hlavního scénáře.
2. Systém zobrazí formulář s prvky: název, jazyk a interaktivní diagram, který uživatel může modifikovat.

3. Dále scénář pokračuje v 5. kroku hlavního scénáře.

Alternativní scénář: zvolené zadání bylo typu „Obrázek“.

1. Scénář začíná ve 3. kroku hlavního scénáře.
2. Systém zobrazí formulář s prvky: obrázek, název zadání, jazyk, možnost volby jiného obrázku.
3. Dále scénář pokračuje v 5. kroku hlavního scénáře.

Alternativní scénář: zvolené zadání bylo typu „Normalizace relačního schématu“.

1. Scénář začíná ve 3. kroku hlavního scénáře.
2. Systém zobrazí formulář, kde vyučující může upravovat následující atributy: název, jazyk, množinu atributů, množinu funkčních závislostí s možností přidávání či odebírání dalších.
3. Po dokončení modifikace hodnot uživatel může uložit změny pomocí tlačítka „*Uložit*“ nebo je odmítnout pomocí tlačítka „*Zpět*“. Systém navíc dovoluje zobrazit ukázkové řešení zvolením tlačítka „*Ukázkové řešení*“.

1.1.2.4 UC4: Duplikace zadání

Případ užití umožňuje učiteli vytvořit kopii zvoleného zadání pomocí tlačítka „*Duplikovat*“. Systém požadavek zpracuje a zobrazí nově vytvořené zadání v seznamu a taky hlášku o úspěchu či chybě.

Aktéři: vyučující, garant.

1.1.2.5 UC5: Odstranění zadání

V systému existuje možnost testové zadání smazat, ale pouze tehdy, když zadání není přiřazeno k žádné otázce, v opačném případě tlačítko „*Odstranit*“ bude neaktivní.

Aktéři: vyučující, garant.

Scénář:

1. Vyučující ze seznamu vybere zadání a klikne „*Odstranit*“.
2. Systém se zeptá na potvrzení.
3. Vyučující zvolí „*Ok*“, pokud chce volbu potvrdit nebo „*Cancel*“ v opačném případě.

1.1.2.6 UC6: Tvorba zadání

Případ užití dovolí vyučujícímu přidávat do systému nová zadání. Vyučující vybírá z těchto druhů zadání: *text, diagram, obrázek* nebo *normalizace relačního schématu*. Proto se postup tvorby nepatrně liší v závislosti na druhu.

Aktéři: vyučující, garant.

Hlavní scénář:

1. Případ užití začíná, když se uživatel rozhodne vytvořit nové zadání. Proto z vedlejšího menu zvolí „*Vytvořit zadání*“.
2. Ve vedlejším menu systém zobrazí typy zadání, které je uživatel schopen vytvořit.
3. Učitel zvolí jeden ze druhů zadání, na jehož základě se mu zobrazí odpovídající formulář.
4. Systém zobrazí formulář v závislosti na druhu zadání.
5. Vyučující vyplní formulář a zvolí „*Uložit*“ pro uložení zadání.
6. Systém provede žádanou operaci a zobrazí seznam zadání nebo chybu, pokud nastala.

Alternativní scénář: uživatel zvolil typ „Normalizace relačního schématu“.

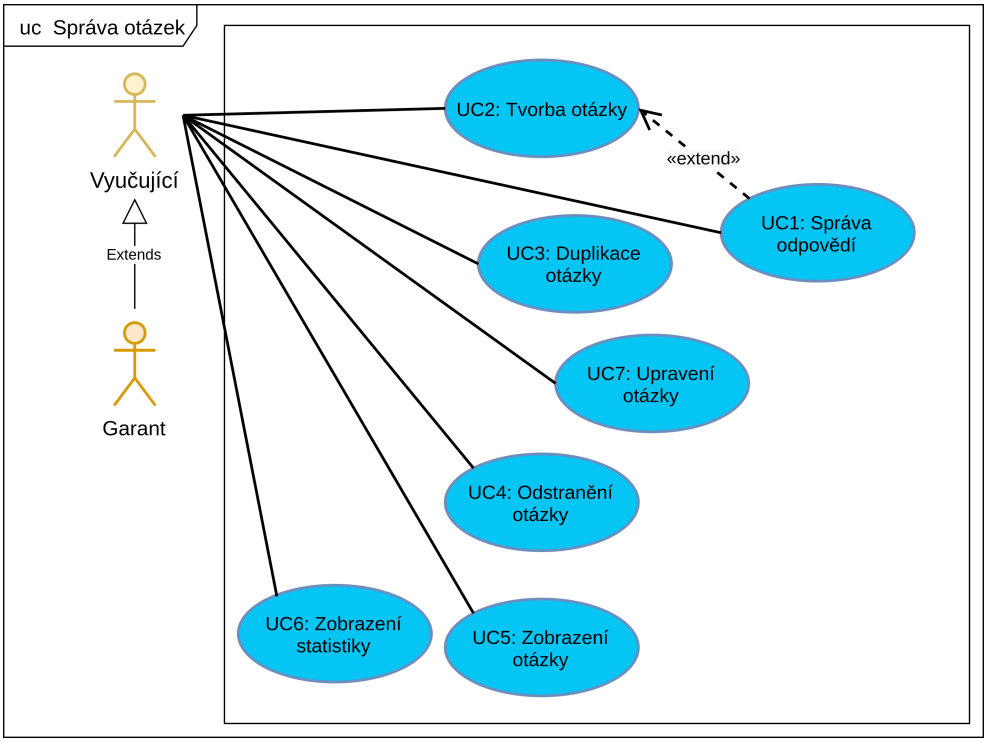
1. Scénář začíná ve 3. kroku hlavního scénáře.
2. Systém zobrazí formulář s údaji: název, jazyk, množina atributů a množina funkčních závislostí.
3. Poté, co vyučující vyplní formulář, může kliknout na tlačítko „*Uložit*“ a systém uloží provedené změny nebo zvolit „*Ukázkové řešení*“.
4. Scénář pokračuje v 6. bodě hlavního scénáře.

Alternativní scénář: vyučující klikl na tlačítko „Generovat zadání“.

1. Scénář začíná ve 2. kroku alternativního scénáře. Uživatel zvolil možnost automatického generování zadání.
2. Systém zobrazí formulář, kde lze nastavit množinu atributů a počet funkčních závislostí.
3. Vyučující vyplní formulář a zvolí „*Generovat zadání*“.
4. Systém vygeneruje zadání a scénář pokračuje ve 3. bodě alternativního scénáře.

1.1.3 Správa otázek

V této podkapitole je popsána tvorba a správa testových otázek. Diagram případů užití lze nalézt na obrázku 1.2.



Obrázek 1.2: Diagram případů užití: Správa testových otázek

1.1.3.1 UC1: Správa odpovědí

Systém umožňuje učiteli přiřazovat nebo odebírat odpovědi z otázky. Postup tvorby odpovědí se liší v závislosti na jejich druhu. Odpovědi můžou být následujících typů: *SQL dotaz, dotaz v relační algebře, textová odpověď, diagram, normalizace relačního schématu, transformace* a *zaškrtávací odpovědi*. Diagram aktivit popisující tento proces lze nalézt v příloze na obrázku D.2.

Aktéři: vyučující, garant.

Scénář:

1. Ze seznamu vyučující vyhledá potřebnou otázku a zvolí tlačítko „*Správa odpovědí*“.
2. Systém zobrazí odpovídající formulář dle druhu odpovědi. Obecně tento formulář obsahuje tyto části: zadání, nastavení otázky, referenční odpověď, všechny odpovědi a část, která dovoluje přidat novou odpověď. Také může obsahovat opravené odpovědi, které se staly referenčními.
3. Uživatel vyplní textová pole nebo nakreslí diagram v závislosti na

druhu odpovědi a klikne na „*Přidat odpověď*“.

4. Systém odpověď přidá a opětovně načte aktuální stránku, kde se tato odpověď objeví v sekci „Všechny odpovědi“.
5. Až učitel dokončí přidávání odpovědí může kliknout „*Zpět*“ a vrátit se k seznamu otázek.

1.1.3.2 UC2: Tvorba otázky

Vyučující a garant mohou vytvářet nové otázky v portálu. Pokud otázka nemá přiřazené zadání a alespoň jednu referenční odpověď, pak ji systém zobrazuje jako nepoužitelnou. Pro lepší porozumění postupu tvorby validní otázky jsou vytvořeny diagramy aktivit jazyka UML, které lze nalézt v příloze na obrázcích D.1, D.2 a D.4.

Aktéři: vyučující, garant.

Scénář:

1. Případ užití začne, když se uživatel rozhodne vytvořit novou otázku.
2. Z vedlejšího menu vybere „Vytvořit otázku“.
3. Systém zobrazí formulář s údaji: název, úkol, obtížnost, kategorie, úroveň, jazyk, typ odpovědi a možnost zařadit otázku do demo testu.
4. Vyučující zadá všechny údaje a klikne na „*Uložit*“ nebo „Uložit a spravovat zadání“.
5. Pokud učitel zvolil „*Uložit a spravovat zadání*“, systém otázku uloží a zároveň přejde do vyplňování „UC Přiřazování zadání“, viz podkapitola 1.1.4.

1.1.3.3 UC3: Duplikace otázky

Systém dovoluje učiteli vytvořit kopii konkrétní otázky.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedna otázka.

Scénář:

1. Případ užití začíná, když se uživatel rozhodne kopírovat otázku.
2. Systém zobrazí seznám otázek.
3. Uživatel v seznamu vyhledá otázku a klikne na „*Duplikovat*“.
4. Systém vytvoří novou otázku a následně zobrazí seznam otázek a odpovídající hlášku o úspěchu či chybě.

1.1.3.4 UC4: Odstranění otázky

Případ užití umožňuje vyučujícímu smazat otázku. Pokud je otázka použita alespoň v jednom testu, pak ji nelze smazat a tlačítko „*Odstranit*“ bude neaktivní.

Aktéři: vyučující, garant.

Počáteční podmínka: otázka není použita v žádném testu.

Scénář:

1. Případ užití začne, když se učitel nebo garant rozhodne smazat otázku, kterou nepotřebuje.
2. Systém zobrazí seznam otázek.
3. Uživatel vyhledá otázku a zvolí „*Odstranit*“.
4. Systém požádá o potvrzení.
5. Uživatel volbu potvrdí tlačítkem „*Ok*“ nebo odmítne tlačítkem „*Cancel*“.
6. Systém otázku odstraní a zobrazí seznam otázek.

1.1.3.5 UC5: Zobrazení otázky

Případ užití umožňuje vyučujícímu zobrazit detailní informace o otázce: popis, složitost, referenční odpovědi a zadání. Pro zobrazení informací učitel zvolí otázku a klikne na „*Náhled*“.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedna otázka.

1.1.3.6 UC6: Zobrazení statistiky

Vyučující je schopen monitorovat úspěšnost konkrétní otázky v testech. Statistika se sestaví ze správných a chybných odpovědí studentů ve všech testech, ve kterých byla otázka použita.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedna otázka.

Scénář:

1. Ze seznamu otázek uživatel zvolí tu, která ho zajímá a klikne na tlačítko s obrázkem statistiky.
2. Systém otevře modální okno, ve kterém se zobrazí graf úspěšnosti studentů u dané otázky.

1.1.3.7 UC7: Upravení otázky

Tento případ užití umožňuje měnit obecné údaje v otázce, např. název, popis apod. Vyučující nemůže měnit druh odpovědi, jestliže existuje alespoň jedna odpověď referenční.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje alespoň jedna otázka.

Scénář:

1. Systém zobrazí seznam otázek.
2. Vyučující si zvolí otázku, kterou potřebuje modifikovat a zvolí tlačítko „*Upravit*“.
3. Systém zobrazí formulář umožňující změnit následující údaje: název, úkol, obtížnost, kategorie, úroveň, jazyk, typ odpovědi, pokud otázka dosud neměla žádnou přidanou odpověď a zařazení do demo testu.
4. Uživatel upraví potřebné údaje a klikne na „*Uložit*“.
5. Systém otázku uloží a zobrazí hlášku o úspěchu či chybě, potom zobrazí seznam všech otázek.
6. Pokud uživatel zvolil „*Uložit a spravovat zadání*“, systém otázku uloží a přejde do případu užití „UC Přiřazování zadání“, viz podkapitola 1.1.4.

1.1.4 Přiřazování zadání

Tato podkapitola popisuje postup přiřazování zadání k otázce. Diagram případů užití je zachycen na obrázku 1.3.

1.1.4.1 UC1: Zvolení otázky

Před přiřazováním či odebíráním zadání si uživatel musí zobrazit seznam otázek, vyhledat otázku a zvolit „*Správa zadání*“ nebo kliknout na „*Uložit a spravovat zadání*“ během tvorby otázky, která je popsána případem užití v podkapitole 1.1.3.2. Nato systém zobrazí formulář, kde budou veškerá zadání a informace o zvolené otázce.

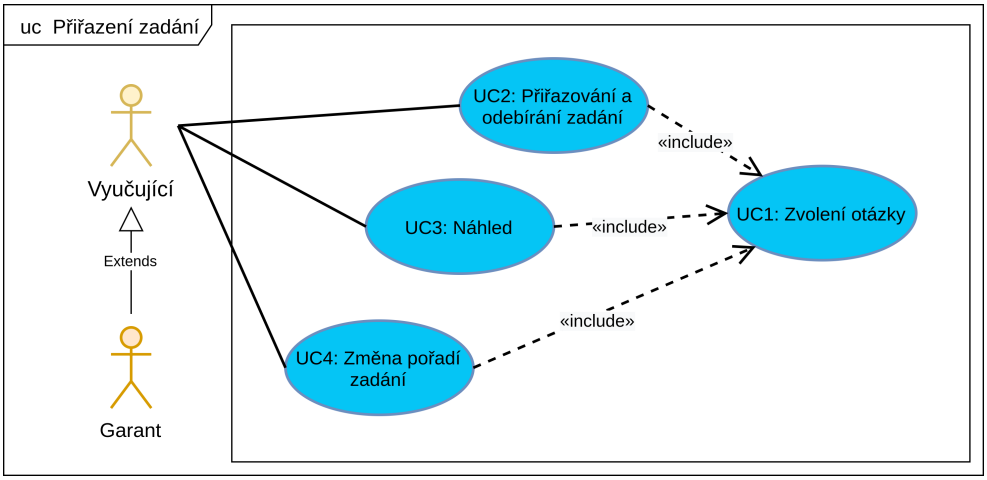
Aktéři: vyučující, garant.

1.1.4.2 UC2: Přiřazování a odebírání zadání

Případ užití dovoluje přidat nebo odebrat zadání z otázky.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje nejméně jedna otázka a jedno zadání.



Obrázek 1.3: Diagram případů užití: Přiřazování zadání

Scénář:

1. Uživatel splní případ užití „UC1: Zvolení otázky“.
2. Ze seznamu zadání vyučující vyhledá potřebné a klikne „Přidat zadání k otázce“ či „Odebrat zadání z otázky“.
3. Systém zvolenou akci provede a hned zobrazí změny.

1.1.4.3 UC3: Náhled

Systém umožňuje vyučujícímu zobrazit informace o konkrétním zadání, před jeho přiřazením k otázce.

Aktéři: vyučující, garant.

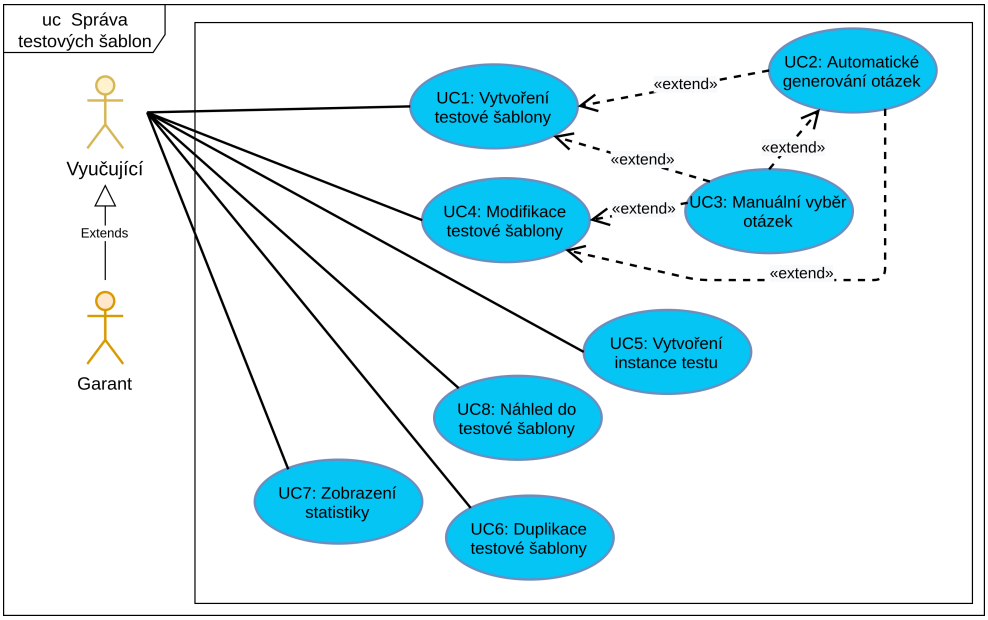
1.1.4.4 UC4: Změna pořadí zadání

Vyučující je schopen měnit pořadí zobrazování zadání studentům v testech. Tuto akci je možné provést pomocí šipek v seznamu zadání. Po tom, co uživatel zvolil jedno z tlačítek, systém akci provede a hned zobrazí nastalé změny.

Aktéři: vyučující, garant.

1.1.5 Správa testových šablon

V této podkapitole můžete nalézt případy užití, které popisují postupy pro tvorbu a správu testových šablon. Diagram případů užití je zobrazen na obrázku 1.4.



Obrázek 1.4: Diagram případů užití: Správa testových šablon

1.1.5.1 UC1: Vytvoření testové šablony

Případ užití umožní vyučujícímu vytvořit novou testovou šablonu, kterou lze následně využít pro vytvoření testů.

Aktéři: vyučující, garant.

Scénář:

1. Případ užití začne, když učitel chce vytvořit novou testovou šablonu a 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*“ či „*Manuální výběr*“.
4. Systém uloží šablonu a přejde do „UC2: Automatické generování otázek“ nebo „UC3: Manuální výběr otázek“ v závislosti na volbě uživatele.

1.1.5.2 UC2: Automatické generování otázek

Vyučující může nastavit počty otázek v jednotlivých kategoriích, na jejichž základě systém vygeneruje otázky a přiřadí je k šabloně.

Aktéři: vyučující, garant.

Scénář:

1. Případ užití začne tím, že uživatel v předchozím případě užití zvolil „*Automatické generování otázek*“.
2. Systém zobrazí formulář, ve kterém lze nastavit počty otázek pro každou z kategorií. Pokud v systému není dostatečný počet otázek příslušné kategorie, pak na to systém vyučujícího upozorní.
3. Uživatel nastaví počty otázek a klikne na „*Uložit*“.
4. Pokud druh testu byl *demo test*, systém uloží otázky a zobrazí seznam testových šablon. Jinak přejde do „UC3: Manuální výběr otázek“.

1.1.5.3 UC3: Manuální výběr otázek

Případ užití dovoluje přidávat nebo odebírat otázky z testu či šablony manuálně. Kromě toho může vyučující měnit pořadí otázek v testu.

Aktéři: vyučující, garant.

Scénář:

1. Případ užití je aktivován systémem po kliknutí uživatelem na tlačítko „*Manuální výběr*“ nebo po dokončení případu užití: „UC2: Automatické generování otázek“, pokud typ testu nebyl *demo test*.
2. Systém zobrazí seznam otázek v šabloně a seznam všech dostupných otázek.
3. Učitel může přidávat a odebírat otázky, měnit jejich pořadí nebo přejít do modifikace obsahu konkrétní otázky.
4. Až vyučující provede veškeré změny, které potřeboval, může kliknout na „*Uložit*“.
5. Systém uloží provedené změny a zobrazí seznam testových šablon.

1.1.5.4 UC4: Modifikace testové šablony

Případ užití umožňuje uživateli měnit parametry konkrétní testové šablony pouze v případě, pokud šablona nemá instanci testu (dosud nebyla použita v systému).

Aktéři: vyučující, garant.

Počáteční podmínka: neexistuje žádná instance testu, která by patřila zvolené šabloně.

Scénář:

1. Vyučující se rozhodne modifikovat šablonu, proto ze seznamu vybere potřebnou a zvolí tlačítko „*Upravit*“.

2. Systém zobrazí formulář, kde lze nalézt následující údaje: název, čas na vypracování, časové okno pro celý test, typ testu a maximální počet bodů.
3. Uživatel upraví hodnoty a klikne na „*Manuální výběr*“.
4. Systém uloží provedené změny a přejde do „UC3: Manuální výběr otázek“, který je možné najít v podkapitole 1.1.5.3.

1.1.5.5 UC5: Vytvoření instance testu

Z testové šablony vyučující může vytvořit instance testu. Druhy testových šablon jsou: *demo test, automatický demo test, test v semestru, zkouška*. Na základě zvoleného druhu se postup tvorby instance trochu liší.

Aktéři: vyučující, garant.

Počáteční podmínka: testová šablona existuje a je validní.

Hlavní scénář:

1. Případ užití začne, když uživatel bude potřebovat vytvořit novou instanci testu, a proto vybere potřebnou šablonu ze seznamu a klikne na „*Vytvořit instanci testu*“.
2. Pokud druh testové šablony byl *zkouška*, pak aplikace zobrazí seznam zkouškových termínů, exportovaných ze systému KOS s možností přiřazení všech studentů z termínu pomocí tlačítka „*Přidat studenty ze zkoušky*“. Kromě toho systém umožní přejít do nastavení kapacit místností.
3. Uživatel zvolil „*Přidat studenty ze zkoušky*“.
4. Systém zobrazí tři seznamy:
   - Přiřadit paralelku.
   - Přiřadit uživatele.
   - Přiřazení studenti.
5. Vyučující může studenty ze zkoušky odebírat či přidávat a pak přejít do nastavení kapacit místností.
6. Pokud uživatel zvolil „*Nastavit kapacitu místností*“, systém zobrazí seznam místností, kde vyučující může nastavovat jejich kapacity.
7. Učitel kapacity nastaví a zvolí „*Uložit*“.
8. Dále systém přejde do kontroly karet studentů.
9. Až vyučující karty zkontroluje může kliknout na „*Spustit*“, čímž spustí test.
10. Pokud někteří studenti nebyli zkontrolováni, systém na to vyučujícího upozorní.

11. Uživatel volbu potvrdí zvolením „*Přesto spustit*“.
12. Systém odebere z testu nezkontrolované studenty a test spustí.

Alternativní scénář:

1. Pokud byl druh testové šablony jiný než *zkouška*, systém zobrazí tři seznamy, ke kterým vyučující může studenty přidávat nebo je naopak odebírat:
   - Přiřadit paralelku.
   - Přiřadit uživatele.
   - Přiřazení studenti.

   Systém též zobrazí formulář „Možnosti testu“ s možnostmi zaškrtnutí:
   - Povolit vyplňování tohoto testu jenom z počítačů s progtest imagem.
   - Zkontrolovat studentské karty před samotným spuštěním testu.
2. Uživatel nakonfiguruje test a zvolí „*Spustit*“.
3. Pokud vyučující zaškrtl možnost kontroly studentských karet, pak systém přejde do 8. kroku hlavního scénáře, jinak spustí test.

1.1.5.6 UC6: Duplikace testové šablony

Systém dovoluje vyučujícímu zduplikovat testovou šablonu pomocí tlačítka „*Duplikovat*“ v seznamu šablon. Šablona bude okopírována včetně všech přiřazených otázek.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje nejméně jedna validní testová šablona.

1.1.5.7 UC7: Zobrazení statistiky

Případ užití umožňuje uživateli se podívat na úspěšnost studentů v testech, které byly vytvořeny z dané šablony.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje nejméně jedna testová šablona.

Scénář:

1. Vyučující se rozhodne podívat se na statistiku úspěšnosti studentů v rámci jedné šablony, proto ze seznamu šablon vybere tu, která ho zajímá a klikne na tlačítko s obrázkem statistiky.
2. Systém statistiku vypočítá a zobrazí graf úspěšnosti studentů.

1.1.5.8 UC8: Náhled do testové šablony

Vyučující má možnost si prohlédnout konkrétní testovou šablonu a zjistit, jaké otázky obsahuje testová šablona a měnit jejich pořadí zobrazování v testu.

Aktéři: vyučující, garant.

Počáteční podmínka: v systému existuje nejméně jedna testová šablona.

Scénář:

1. V seznamu testových šablon učitel vyhledá šablonu a zvolí „*Náhled*“.
2. Systém zobrazí modální okno, které bude obsahovat seznam přiřazených otázek k šabloně.
3. Uživatel může pořadí otázek v testu libovolně měnit a také nastavovat opravujícího u konkrétní otázky. Provedená změna se hned projeví po kliknutí na jedno z tlačítek.

1.1.6 Správa testů

V této podkapitole se budeme zabývat spouštěním a ukončením testů. Diagram případů užití je zachycen na obrázku 1.5.

1.1.6.1 UC1: Přiřazení studentů k testu

Případ užití dovoluje vyučujícímu přiřadit studenty k testu, který dosud nebyl spuštěn.

Aktéři: vyučující, garant.

Počáteční podmínka: test nebyl spuštěn.

Scénář:

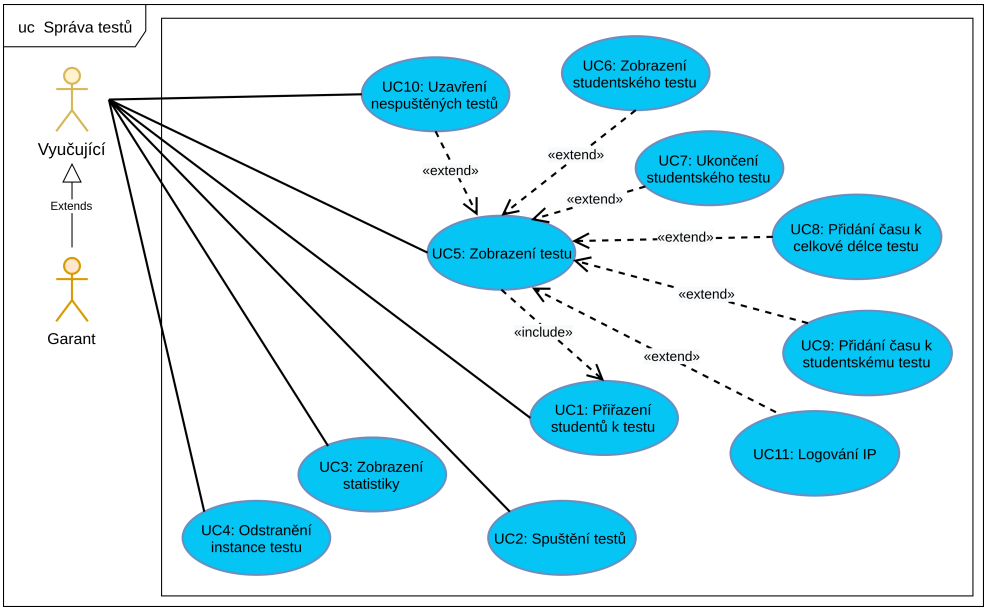
1. Případ užití začne, když učitel bude potřebovat přidat studenty k testu. V seznamu s testy vyučující vyhledá ten, který ho zajímá a klikne na tlačítko „*Přiřadit*“.
2. Dále přidávání studentů pokračuje v 1. kroku alternativního scénáře případu užití „UC5: Vytvoření instance testu“, který se nachází v podkapitole 1.1.5.5. Pokud druh zvoleného testu byl *zkouška*, pak pokračuje v hlavním scénáři v bodě 4.

1.1.6.2 UC2: Spuštění testu

Systém umožňuje vyučujícímu spustit obecný test (instanci šablony), pokud se tento test nachází ve stavu *připravený*.

Aktéři: vyučující, garant.

Počáteční podmínka: test nebyl spuštěn.



Obrázek 1.5: Diagram případů užití: Správa testů

Scénář:

1. Ze seznamu testů uživatel zvolí test a klikne na „*Spustit*“.
2. Systém test spustí a pokračuje v případě užití: „UC5: Zobrazení testu“, viz podkapitola 1.1.6.5.

1.1.6.3 UC3: Zobrazení statistiky

Případ užití umožňuje zobrazit graf úspěšnosti studentů ve vybraném testu. Pro zobrazení statistiky vyučující vybere test ze seznamu a klikne na tlačítko s obrázkem statistiky.

Aktéři: vyučující, garant.

1.1.6.4 UC4: Odstranění instance testu

Případ užití dovoluje učiteli smazat instanci testu, jestliže dosud nebyla spuštěna.

Aktéři: vyučující, garant.
Počáteční podmínka: test nebyl spuštěn.

Scénář:

1. Případ užití začne, když se vyučující rozhodne smazat instanci testu, kterou nepotřebuje, proto zvolí „*Odstranit*“.

2. Systém se zeptá uživatele na potvrzení.
3. Uživatel svou volbu potvrdí.
4. Systém smaže instanci testu a zobrazí seznam testů.

1.1.6.5 UC5: Zobrazení testu

Vyučující si může zobrazit konkrétní test (instanci šablony), v rámci kterého běží studentské testy a následně přejít do manipulace s nimi. Všechny následující případy užití budou na tento navazovat.

Aktéry: vyučující, garant.

Počáteční podmínka: test byl spuštěn.

Scénář:

1. Případ užití začne, když se učitel bude potřebovat podívat na studentské testy a proto ze seznamu testů vyhledá potřebný a klikne na tlačítko „*Zobrazit*“.
2. Systém zobrazí seznam studentských testů a seznam s uživateli.
3. Vyučující je schopen během testu uživatele přidávat a také vyplňovat akce, které jsou reprezentovány následujícími případy užití.

1.1.6.6 UC6: Zobrazení studentského testu

Případ užití umožňuje zobrazit modální okno, ve kterém lze vidět seznam otázek a další informace spojené s hodnocením a vyplňováním testu.

Aktéři: vyučující, garant.

Scénář:

1. Ze seznamu studentských testů vyučující vybere test a klikne na „*Zobrazit test*“.
2. Systém zobrazí modální okno s otázkami.
3. Po náhledu do testu může uživatel okno zavřít nebo si zobrazit informace o jednotlivých otázkách pomocí tlačítka „*Náhled*“ u konkrétní otázky.

1.1.6.7 UC7: Ukončení studentského testu

Vyučující má právo ukončit studentův test předčasně, pokud student podváděl nebo i z jiných důvodů.

Aktéři: vyučující, garant.

Scénář:

1. Ze seznamu studentských testů vyučující zvolí test a klikne na „*Ukončit test*“.

2. Systém test ukončí a zobrazí seznam studentských testů.

1.1.6.8 UC8: Přidání času k celkové době testu

Systém umožňuje učiteli přidávat čas k celkovému oknu, v rámci kterého běží jednotlivé studentské testy.

Aktéři: vyučující, garant.

Scénář:

1. Pro přidání času vyučující klikne na tlačítko „*Přidat 5 minut k celkové délce testu*“.
2. Systém čas k testu přidá a zobrazí odpovídající hlášku.

1.1.6.9 UC9: Přidání času k studentskému testu

Vyučující je schopen prodlužovat čas na vypracování testu konkrétnímu studentovi.

Aktéři: vyučující, garant.

Scénář:

1. V seznamu studentských testů učitel vyhledá test a zvolí tlačítko „*Přidat 5 minut*“.
2. Systém přidá čas k studentskému testu a zobrazí odpovídající hlášku.

1.1.6.10 UC10: Uzavření nespuštěných testů

V průběhu testu se může učitel rozhodnout uzavřít testy, které do této doby nebyly spuštěny studenty, např. z důvodu nepřítomnosti některých studentů.

Aktéři: vyučující, garant.

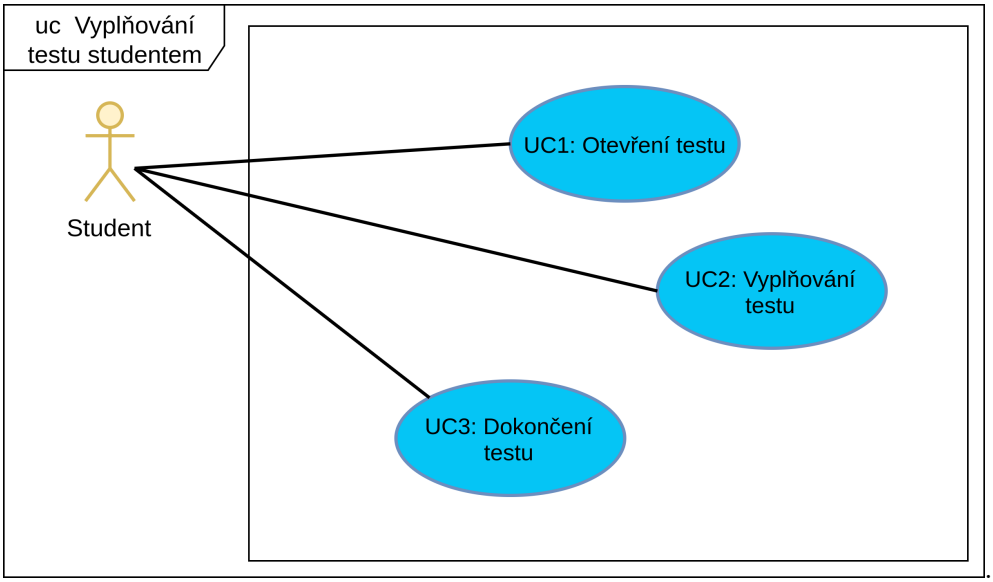
Scénář:

1. Pod seznamem s testy vyučující klikne na tlačítko „*Uzavřít nespuštěné testy*“.
2. Systém nespuštěné studentské testy uzavře, pokud takové existují, potom zobrazí odpovídající hlášku.

1.1.6.11 UC11: Logování IP adres

Případ užití umožňuje uchovávat záznamy o přístupu studentů během testu. Jestliže byl student během testu přihlášen ze dvou míst současně, systém na to upozorní. Učitel má možnost tyto logy procházet a vyhledávat v nich.

Aktéry: systém.



Obrázek 1.6: Diagram případů užití: Vyplňování testu studentem

1.1.7 Vyplňování testu studentem

V této podkapitole se podíváme na vyplňování studentských testů. Diagram případů užití lze najít na obrázku 1.6.

1.1.7.1 UC1: Otevření testu

Případ užití umožňuje studentovi spustit test.

Aktéři: student.

Počáteční podmínka: test byl aktivován.

Scénář:

1. Z menu student zvolí „Testy“.
2. Systém zobrazí seznam všech dostupných testů, ke kterým je student přiřazen.
3. Uživatel vybere test a klikne na „*Start test*“.
4. Pokud test nebyl spuštěn, systém test spustí a zobrazí seznam otázek, které má student za úkol zodpovědět.

1.1.7.2 UC2: Vyplňování testu

Student může postupně odpovídat na otázky a může si zobrazit seznam všech otázek v testu a přecházet mezi vyplňováním libovolné z nich. V seznamu

otázek je zobrazeno, jestli otázka byla vyplněna nebo ne.

Aktéři: student.

1.1.7.3 UC3: Dokončení testu

Systém umožňuje ukončit test předčasně pomocí tlačítka „*Ukončit test*“. Až student vyplní všechny otázky, může se rozhodnout jej ukončit. Po ukončení testu uživatelem, systém test automaticky opraví a zobrazí počet získaných bodů. Navíc zobrazí otázky, které se nepodařilo vyhodnotit automaticky a které jsou přesměrovány do manuální opravy.

Aktéři: student.

1.1.8 Hodnocení testů

Tato podkapitola popisuje hodnocení testů a nastavování podmínek klasifikace. Diagram případů užití je zobrazen na obrázku 1.7.

1.1.8.1 UC1: Zobrazení podmínek klasifikace

Systém dovoluje vyučujícímu zobrazit podmínky klasifikace.

Aktéři: vyučující, garant.

Scénář:

1. Z vedlejšího menu uživatel zvolí „*Podmínky klasifikace*“.
2. Systém zobrazí podmínky, které student musí splnit pro úspěšné ukončení předmětu.

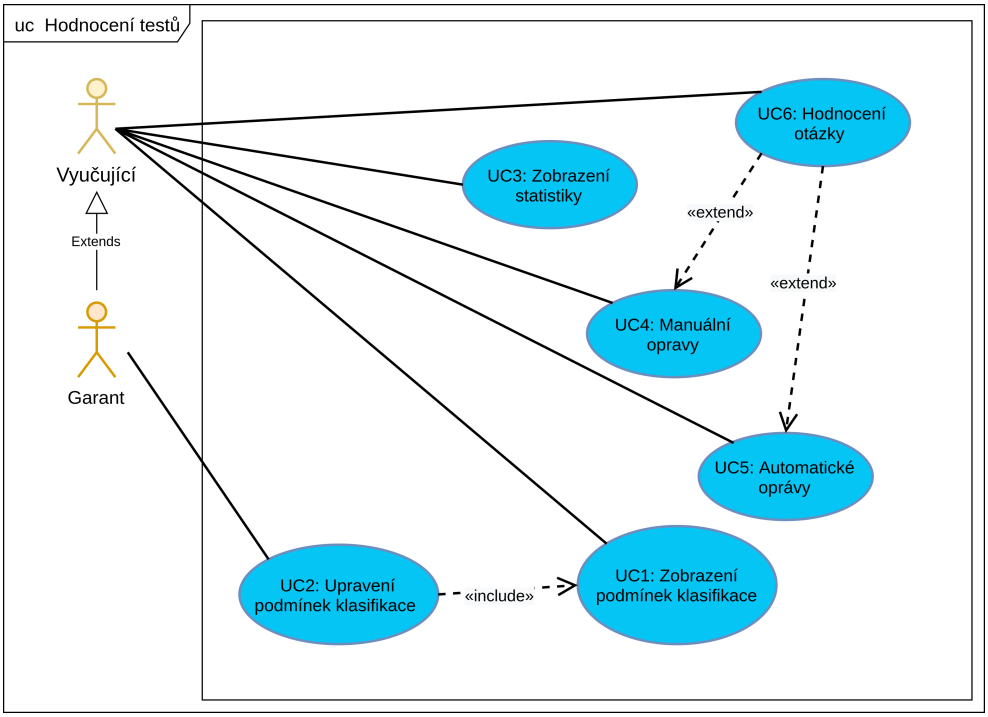
1.1.8.2 UC2: Upravení podmínek klasifikace

Případ užití umožňuje garantovi nastavovat podmínky klasifikace.

Aktéři: garant.

Scénář:

1. Případ užití začne až uživatel vyplní předchozí „UC1: Zobrazení podmínek klasifikace“.
2. Garant klikne na tlačítko „*Upravit požadavky*“.
3. Systém zobrazí formulář, kde vyučující může měnit:
   - minimální požadované body z testu,
   - maximum bodů z ústní zkoušky,
   - automaticky stržené body u ústní zkoušky.
4. Uživatel příslušné hodnoty upraví a zvolí „*Uložit*“ nebo odmítne pomocí tlačítka „*Zrušit*“.



Obrázek 1.7: Diagram případů užití: Hodnocení testů

5. Systém zobrazí podmínky klasifikace.

1.1.8.3 UC3: Zobrazování statistiky

V systému může vyučující nebo garant zobrazit celkovou statistiku úspěšnosti v testech, která je rozdělena v závislosti na konkrétním cvičícím. Pro zobrazení statistiky uživatel zvolí z vedlejšího menu tlačítko „*Statistika*“.

Aktéři: vyučující, garant.

1.1.8.4 UC4: Manuální opravy

Případ užití dovoluje uživateli přejít do bodování otázek v testu.

Aktéři: vyučující, garant.

Počáteční podmínka: alespoň jeden ukončený studentův test.

Scénář:

1. Vyučující z vedlejšího menu zvolí „*Manuální opravy*“.
2. Systém zobrazí dva seznamy, v prvním seznamu učitel uvidí pouze své testy, ve druhém všechny testy v systému.

3. Vyučující může vybrat test a kliknout „*Zobrazit neopravené*“ nebo „*Zobrazit vše*“.
4. Systém zobrazí všechny otázky v testu pokud uživatel klikl na „*Zobrazit vše*“, v opačném případě zobrazí pouze ty, ve kterých alespoň jeden student měl chybu.
5. Dále scénář pokračuje v případě užití „UC6: Hodnocení otázky“, který můžete najít v následující podkapitole. V případě, že uživatel zvolil tlačítko „*Zpět*“, končí.

1.1.8.5 UC5: Automatické opravy

Systém umožňuje vyučujícímu si prohlédnout automatické opravené testy studentů. V případě, že systém otázku vyhodnotil špatně, učitel může hodnocení opravit. Scénář je podobný případu užití „UC4: Manuální opravy“, s jediným rozdílem: vyučující klikne na tlačítko „*Automatické opravy*“. Z tohoto důvodu již podrobný scénář znovu neuvádíme. Příslušný případ užití lze nalézt v podkapitole 1.1.8.4.

Aktéři: vyučující, garant.

Počáteční podmínka: alespoň jeden ukončený studentův test.

1.1.8.6 UC6: Hodnocení otázky

Případ užití umožňuje vyučujícímu manuálně obodovat otázku, která nebyla automaticky vyhodnocena systémem.

Aktéři: vyučující, garant.

Počáteční podmínka: alespoň jeden ukončený studentův test.

Scénář:

1. Uživatel ze seznamu otázek v testu zvolí otázku a klikne „*Ohodnotit otázku*“.
2. V levé části obrazovky systém zobrazí:
   - Zadání, která jsou přiřazena k otázce.
   - Podrobné informace o hodnocené otázce.
   - Již ohodnocené odpovědi studentů.

   V pravé části obrazovky budou rozbalené neohodnocené studentské odpovědi.
3. Učitel postupně odpovědi hodnotí, čímž se formuláře zabalují. Ve formuláři může vyučující nastavit počet bodů za studentskou odpověď a přidat komentář, který student následně uvidí v náhledu testu. Pokud některá z odpovědí jiných studentů je stejná jako ohodnocená

učitelem, pak ji systém oboduje stejným počtem bodů. Po vyhodnocení testu systém vygeneruje upozornění pro studenta, jehož otázka byla ohodnocena.

4. Po dokončení hodnocení může uživatel kliknout na tlačítko „*Zpět*“, čímž se vrátí do seznamu otázek.

1.2 Hlavní problémy současného řešení

V předchozí podkapitole byla provedena podrobná analýza uživatelských případů, což lze považovat za analýzu funkčností systému z pohledu uživatele a vymezení toho, co systém v současné době umí. Dále se tato kapitola zabývá analýzou systému z technického pohledu a konkrétně současným stavem databáze. Jak už je z předchozí podkapitoly jasné, tato analýza je omezena na část, která souvisí s testovým modulem. Navíc se trochu dotýká aplikační vrstvy a technologií, které se v portálu používají.

1.2.1 Databáze

V DBS portálu se již od začátku používá databáze PostgreSQL, která je objektově-relačním databázovým systémem s otevřeným zdrojovým kódem a nabízí hodně zajímavých funkcí pro podporu snazšího vývoje aplikací, ochranu dat a spoustu dalších možnosti, např. rozšiřovatelnost: možnost definovat vlastní strukturované datové typy a funkce, které lze přidávat pomocí různých programovacích jazyků bez překompilování celé databáze, JIT kompilace (Just-in-time compilation), paralelní zpracování dotazů a další. [13] Tento databázový systém plně podporuje standard jazyka SQL SQL:2011 a poslední jeho verze, která byla vydána v roce 2019, už pokrývá většinu specifikací SQL:2016. [14]

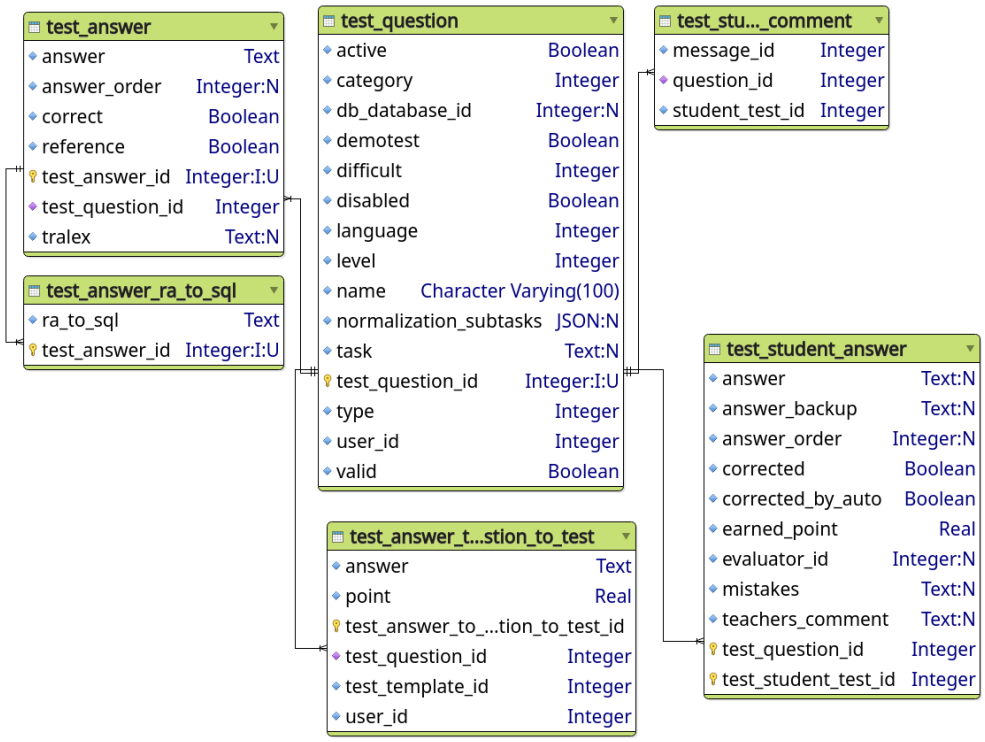
Tato podkapitola se bude zabývat analýzou databázových objektů a celkovou strukturou databáze testového modulu DBS portálu, kde se zaměříme hlavně na nevýhody, které tato struktura přináší pro další rozvoj systému.

1.2.1.1 Uložení otázek a odpovědí

Jak už jsme mohli vidět v podkapitole 1.1.3, systém umožňuje evidovat testové otázky, referenční a studentské odpovědi. V následující podkapitole se podíváme na to, jak jsou tato data v systému uchovávána. Relační schéma lze vidět na obrázku 1.8.

Nejprve se podíváme na tabulku *`test_question`* a její atributy: `category, demotest, difficult, level, disabled`. Na první pohled vypadají celkem správně, ale pokud si položíme otázku, jaký je jejich hlavní účel, tak pochopíme, že tyto atributy nesou stejný význam spočívající v rozdělení otázek na nějaké skupiny. Proto není důvod, aby je tabulka obsahovala. Bude mnohem lepší navrhnout vhodnou strukturu, která by reprezentovala rozdělení otázek.

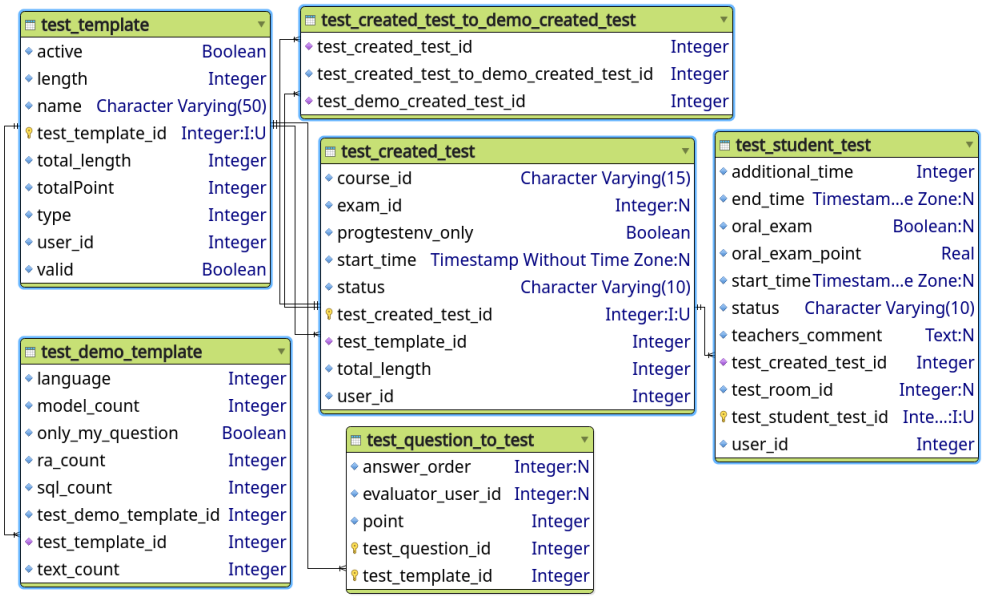
Dalšími nevhodnými můžeme jmenovat atributy `normalization_subtasks` a `db_database_id`. První z nich propojuje konkrétní otázku s databázovým přípojením a používá se během vyhodnocení SQL a RA odpovědí. Každá otázka má nějaká zadání, zadání může mít více otázek, tj. pokud se vyučující rozhodně změnit připojení u tohoto konkrétního zadání, tak ho musí změnit



Obrázek 1.8: Relační schéma: Současný stav otázek a odpovědí

v každé otázce, která zadání používá. Toto znemožňuje jednoduchou změnu přípojení, proto to pro nás není vhodné. Poslední atribut se nám také nebude hodit, jelikož se používá pouze pro konkrétní druh zadání, což znamená, že jej ve většině otázek vůbec nevyužijeme.

Uložení referenčních a testových odpovědí je rozdělené do dvou tabulek: *`test_answer`* a *`test_student_answer`*, viz obrázek 1.8. Kromě nich, můžeme vidět také pomocné tabulky, které v zásadě nejsou nutné. První z nich uchovává překlad z RA do jazyka SQL, což se využívá pro optimalizaci automatické opravy. Jelikož jiné druhy odpovědí také mohou obsahovat transformovanou odpověď, hodilo by se tyto informace držet v jedné tabulce s odpověďmi pomocí jednoho atributu, např. `answer_transformed`. Další tabulka reprezentuje vyhodnocené odpovědi studentů, které se staly referenčními, a využívají se v automatickém hodnocení testů. Tato tabulka obsahuje duplicitní informace, které už jsou obsažené ve studentských odpovědích. Navíc studentské a referenční odpovědi jsou v podstatě stejné s výjimkou některých atributů: `mistakes, teachers_comment, earned_points`. Také víme, že referenčních odpovědí je výrazně méně, proto by se tyto čtyři tabulky daly sjednotit a tím nejen zjednodušit strukturu databáze, ale i odstranit duplicity.



Obrázek 1.9: Relační schéma: Současný stav testových šablon a testů

1.2.1.2 Uložení testových šablon a testů

V této podkapitole se budeme zabývat analýzou ukládaní testových šablon a studentských testů. Podrobný popis funkcí těchto částí testového modulu pomocí případů užití můžete nalézt v podkapitolách 1.1.5, 1.1.6, 1.1.7 a 1.1.8. Relační schéma je zachycena na obrázku 1.9.

Jak už víme z analýzy případů užití, vyučující může vytvářet čtyři druhy testu: *zkouška, test v semestru, demo test* a *automatický demo test*. Typ testu se v databázi označuje pomocí atributu `type` v tabulce *`test_template`*, navíc testovou otázku lze přiřadit pouze k testové šabloně. Velkou nevýhodou tedy je, že testovou šablonu nelze použít opakovaně, i kdybychom chtěli udělat pouze drobnou změnu. Testy jsou pevně vázané na šablonu, proto šablona zcela ztrácí svůj význam, jelikož hlavním účelem šablony je opakované vytváření testů. Kvůli tomu se stává, že učitel potřebuje duplikovat testovou šablonu, protože v původní šabloně už nelze nic změnit, např. hodnocení otázek, druh testu apod.

Kvůli tomu, že jsou otázky propojeny pouze se šablonou, narážíme na další problém. Nemůžeme generovat automatický test, aniž bychom negenerovali testovou šablonu. Pro automatické generovaní testu se používají následující tabulky:

- *`test_demo_template`* – tabulka potřebná pro držení informací o počtu otázek v kategoriích. Tato informace se využívá pro generovaní otázek v automatických testech.

- *test_created_test_to_demo_created_test* – využívá se pro spojení generovaných testů s obecným testem vytvořeným vyučujícím.

Tato databázová struktura tabulek rovněž nedovoluje vytvářet různé varianty pro stejný test, což by se v tomto systému určeném hlavně pro testování studentů hodilo.

Vedle hlavních problémů můžeme také pozorovat méně podstatné problémy, např. atribut `answer_order` v tabulce *test_question_test* znamená pořadí otázek, ale ne odpovědí, nevhodný datový typ atributu `point` ve stejné tabulce, umožňuje pouze celočíselné bodování otázek a další podobné drobnosti.

1.2.1.3 Uložení předmětu a testových místností

Systém umožňuje import předmětů, zkoušek a studentských paralelek ze systému KOS[3], kde následovně může vyučující tato data spravovat. Tyto funkce jsou už mimo rozsah tématu této práce, proto se jimi nebudeme podrobněji zabývat. V této podkapitole se omezíme především na ukládaní předmětů a místností, ve kterých test může probíhat. Relační schéma této části databáze lze vidět na obrázku 1.10.

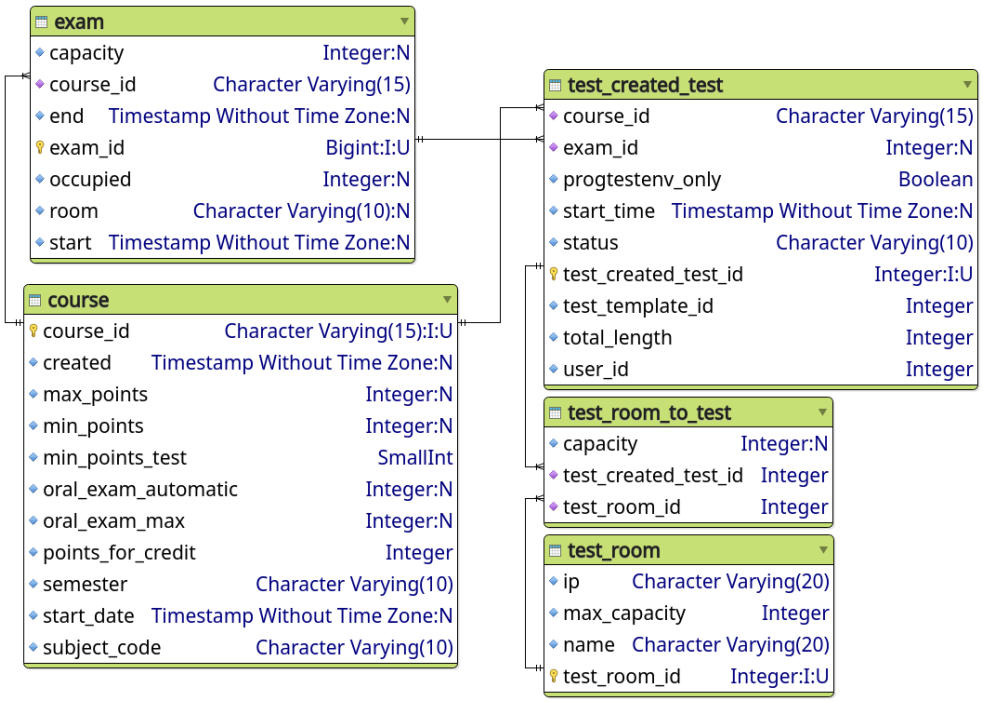
Nejprve provedeme analýzu tabulky *course* a ujasníme si, jakým způsobem souvisí s testovým modulem. Tato tabulka reprezentuje veškerou konfiguraci předmětu v konkrétním semestru, která je sdílená mezi všemi moduly v DBS portálu. Z ní se v testovém modulu používají jen tři atributy: `min_points_test, oral_exam_automatic` a `oral_exam_max`. Je zřejmé, že je tato tabulka používána skoro všemi částmi systému, proto zbytečně provazuje komponenty mezi sebou, které by měly být zcela nezávislé. Bylo by na místě tyto atributy vyčlenit do samostatné tabulky, která by byla vázána na tabulku *course*, čímž by bylo možné odstranit provázanost. Též můžeme pozorovat, že všechny tři atributy používají celočíselný datový typ pro uchování bodů, což není vždy žádoucí.

Dále se podíváme na ukládaní místností, ve kterých se může psát test. Jistě víme, že neexistují místnosti, které mají stejný název. Tedy umělý identifikátor (primární klíč) v tabulce *test_room* je zbytečný. V místo něj můžeme jednoduše použít název místnosti jako primární klíč[4]. Pak v tabulce *exam* můžeme vytvořit cizí klíč[5], který bude odkazovat do tabulky *test_room*.

[3] Systém KOS – studijní informační systém, který se využívá studenty a učiteli na ČVUT.

[4] Primární klíč – sloupec databázové tabulky sloužící pro jednoznačnou identifikace řádku v tabulce. Primárním klíčem může být libovolný unikátní atribut nebo kombinace více atributů, která musí být jednoznačná. [15]

[5] Cizí klíč – slouží pro vytvoření vztahu mezi tabulkami. Umožňuje jednoznačně identifikovat, které záznamy z různých tabulek mezi sebou souvisí (jsou v relaci). [15]



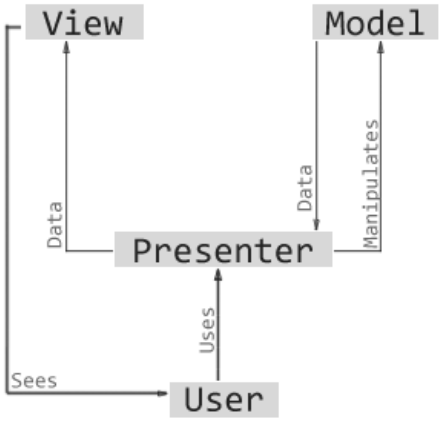
Obrázek 1.10: Relační schéma: Současný stav předmětů a testových místností

1.2.2 Architektura

Zkoumaný systém byl založen na frameworku Nette, proto má třívrstvou architekturu MVP (Model-View-Presenter), která je velice podobna MVC (Model-View-Controler) s jedním rozdílem: do presenteru v MVP se většinou přidává logika aplikace na rozdíl od MVC, kde je veškerá business logika aplikace převážně obsazená v Modelu. Ovšem není tomu tak vždy a liší se to v závislosti na konkrétní implementaci této architektury. [16] Příklad komunikace mezi jednotlivými vrstvami lze vidět na obrázku 1.11. Více o tomto frameworku a o použitých technologiích si povíme později.

Architektura MVP rozděluje program do tří vrstev, které by mezi sebou měly komunikovat přes jasně definované rozhraní, většinou jednosměrně. To občas neplatí u vrstvy View, např. v některých implementacích může prezentační vrstva používat vrstvu logickou. Obecně vrstva reprezentuje skupinu objektů, které mohou být mezi sebou propojené pomocí závislostí a definují rozhraní umožňující s touto vrstvou komunikovat dalším objektem či vrstvou.

Rozdělení na jednotlivé vrstvy je extrémně důležité pro střední a velké projekty. Pokud větší systém používá pouze jednu vrstvu (Monolitická architektura), velice snadno se může dostat do stavu, kde nároky na jeho údržbu budou mnohem vyšší, než přínosy, proto na tomto rozdělení závisí úspěch



Obrázek 1.11: Příklad komunikace mezi vrstvy Model-View-Presenter [17]

projektu. Díky vícevrstvé architektuře můžeme snadno rozdělit zodpovědnosti jednotlivých objektů, zabezpečit jejich nezávislost a spolupráci, což výrazně zjednoduší složitost systému, zajistí jeho snazší udržovatelnost a rozšiřitelnost oproti jednovrstvé. [16, 18] V případě DBS portálu jsou vrstvy následující:

- *Prezentační (Views)* – zodpovídá za zobrazování dat koncovému uživateli, např. HTML šablony.
- *Aplikační (Presenters)* – vrstva, pomocí které probíhá komunikace mezi uživatelem, modelem a view. Tady je implementována veškerá logika aplikace.
- *Datová (Models)* – tato vrstva je zodpovědná za ukládaní a selekci dat.

V testovém modulu DBS portálu, jak můžeme vypozorovat z kódu, rozdělení do těchto vrstev je více méně zachované. Přesto, že toto rozdělení máme, můžeme sledovat porušení principu Low Coupling[6] nejenom uvnitř jedné z vrstev, ale i mezi nimi. Vysoká provázanost spočívá hlavně v tom, že datová vrstva poskytuje data, která jsou přímo vázaná na názvy sloupečků v databázových tabulkách. Toto provazuje aplikační vrstvu s datovou, čímž komplikuje jakékoli změny tabulek a obecně změnu struktury databáze. Vzhledem k tomu, že v současnosti je potřeba měnit spoustu databázových objektů, viz předchozí podkapitola, toto bylo jedním z hlavních důvodu provedení refaktorování celého testového modulu.

Dále můžeme sledovat porušení principu High Cohesion[7] uvnitř aplikační vrstvy. Navíc se objevuje i porušení principu nízké provázanosti. Presentery

[6]Low Coupling neboli nízká provázanost je princip, který říká, že bychom měli přiřazovat zodpovědnosti tak, aby provázanost mezi komponentami byla co nejmenší. [19]

[7]High Cohesion neboli vysoká soudružnost je princip, jež doporučuje přidávat zodpovědnosti tak, aby byla zachována vysoká soudružnost. Každá třída by měla být zodpovědná za jednu konkrétní funkci. [19]

mimo zajištění komunikace mezi komponentami a uživatelem, obsahují vetší část business logiky aplikace, což pak způsobuje vysokou provázanost s ostatními objekty. Navíc se soubory, ve kterých je umístěn zdrojový kód, staly velkými a nepřehlednými. Častá výměna vývojářů na projektu přináší zvýšené riziko vnesení chyb do systému a činí jej složitějším. Proto se tato práce bude snažit tyto závislosti odstranit a rozdělit logiku aplikace na menší samostatné části, které by měly plnit jen jeden konkrétní úkol.

Prezentační vrstva se také stala docela významným důvodem pro provedení refaktorování. Zjistilo se, že Nette framework poskytuje nedostačující funkcionalitu pro další rozvoj frontendu DBS portálu. Proto bylo rozhodnuto pro implementaci prezentační vrstvy použít nějaký vhodný JS framework, který by nabízel dostačující možnosti pro budoucí vývoj. Vzhledem k tomu, že toto už je mimo téma této práce, dále se prezentační vrstvou zabývat nebudeme.

1.3 Definice rozšíření

Tato podkapitola se bude zabývat specifikací požadavků a vymezením toho, co by měl nový testový modul splňovat. Jelikož většina těchto funkcí systému už byla znázorněna pomocí případů užití v podkapitole 1.1, navíc je docela dobře popsal Ing. Petr Pejša ve své bakalářské práci [10], není důvod se jimi detailně zabývat. V této části se hlavně omezíme na definici rozšíření. Nejprve lehce uvedeme čtenáře do dáne problematiky, podíváme se proč vlastně specifikace požadavků je důležitá činnost a proč bychom se jí měli zabývat.

Požadavky z pohledu softwarového inženýrství jsou specifikací toho, co by mělo být implementováno, popisují veškeré chování systému nebo jeho části. Měly by jasně a jednoznačně definovat konkrétní omezení, hranice, kde by měl softwarový proces skončit. Nestanovení jasné hranice může způsobit problémy se zákazníkem, který nebude chtít platit za něco, co dle něj již bylo zaplaceno. Z tohoto důvodu se doporučuje jasně a jednoznačně vymezit to, co se udělat má a co se implementovat nebude. [20] Proto by každý funkční požadavek měl obsahovat:

- Aktéra – osobu, která akce provádí. Podrobný přehled účastníků v DBS portálu lze nalézt v podkapitole 1.1.1.
- Akci – nějakou konkrétní činnost, kterou může vykonávat aktér.
- Vymezující kriteria – konkrétní podmínky a omezení, která se na tento požadavek kladou, nebo na základě kterých účastník vykonává svou činnost.

Softwarové požadavky můžeme dělit dle následujících typů [21]:

- Funkční – požadavky popisující funkce systému, jeho chování.
- Nefunkční – zabývají se obecnými vlastnostmi systému. Typicky: *přenositelnost, bezpečnost, udržovatelnost, spolehlivost, škálovatelnost, výkon, opětovné použití, flexibilita.*
- Požadavky na doménu – liší se v závislosti na konkrétní doméně.

1.3.1 Funkční požadavky

1.3.1.1 FR1: Správa zadání

Systém umožňuje uživateli vytvářet, editovat, případně mazat testová zadání, která mohou být těchto druhů: *text, obrázek, diagram, normalizace, databáze.* Poslední druh zadání dosud v systému neexistoval, jak lze vidět z případů užití popsaných v podkapitole 1.1.2.6. Tento druh zadání bude obsahovat přípojení, které se pak bude využívat pro vyhodnocení otázek, jejichž odpovědi jsou typu *RA* nebo *SQL*.

Aktéři: vyučující, garant.

1.3.1.2 FR2: Správa přípojení

Nově se v testovém modulu objeví databázové přípojení, které bude podobné tomu, co je v současnosti v modulu „Databáze“. Toto přípojení je určeno pro tvorbu databázového zadání a vyhodnocení odpovědí, proto vůbec nebude mít vlastníka oproti přípojení v databázovém modulu, tj. bude sdíleno pro všechny vyučující. Navíc bude obsahovat atribut `createScript`, který bude určen pro budoucí rozšíření. Uživatel bude schopen přípojení vyrábět, upravovat a mazat.

Aktéři: vyučující, garant.

1.3.1.3 FR3: Správa otázek

Vytvoření otázek a jejich správa bude v podstatě podobná jako je popsáno v podkapitole 1.1.3.2. Otázky už nebudou obsahovat některé z atributů, např. přípojení. Také se nebudou dělit na kategorie a úrovně. V místo těchto vlastností, otázky budou obsahovat štítky, na základě kterých se bude provádět rozdělení do skupin, vyhledávání apod.

Aktéři: vyučující, garant.

1.3.1.4 FR4: Vytvoření štítků

Štítky umožní rozdělování otázek dle libovolného vymyšleného druhu, proto jejich tvorba bude dostupná pouze garantovi předmětu, aby bylo znemožněno vytvoření štítků se stejným účelem apod. Garant bude schopen tyto štítky vytvářet, modifikovat a mazat. Každý štítek musí mít pouze jednu štítkovou skupinu, která může být disjunktní nebo ne, což se pak bude využívat během generování otázek. Disjunktivnost skupiny znamená možnost přiřazení k otázce pouze jednoho štítků ze stejné skupiny. Pokud skupina disjunktní není, můžeme přiřadit více. Každý štítek nebo jejich skupina bude obsahovat název a popis, čímž poznáme k čemu slouží.

Aktéři: garant.

1.3.1.5 FR5: Nastavení štítků

Systém umožní uživateli jednotlivé štítky k otázkám přiřazovat nebo odebírat. Libovolná otázka může mít více štítků, štítek může obsahovat mnoho otázek. Otázka, která nemá přiřazený žádný štítek se stane neplatnou. Následně tuto otázku nelze použít při tvorbě testové šablony či testu. Štítek nelze přiřadit k otázce, pokud už otázka obsahuje štítek ze stejné skupiny, kromě případu, kdy skupina není disjunktní.

Aktéři: vyučující, garant.

1.3.1.6 FR6: Správa skupin otázek

Systém bude podporovat tvorbu skupiny otázek s konkrétním účelem. Každá otázka bude patřit k obecné skupině, ve které jsou obsazený všechny otázky v systému. Kromě toho otázka může být přiřazena k více skupinám a každá skupina může obsahovat libovolné množství otázek. Uživatel má možnost tyto skupiny vytvářet, modifikovat a zobrazovat. Navíc bude schopen do těchto skupin otázky přidávat a odebírat. Tyto skupiny otázek se především budou používat pro automatické generovaní testů. Ze kterých se otázky budou vybírat na základě nastavených štítků.

Aktéři: vyučující, garant.

1.3.1.7 FR7: Generování otázek

Systém bude schopen automaticky generovat otázky nejen pro automatický test, ale i obecně na základě zvolených štítků a skupiny otázek. Jsou čtyři možné případy, ve kterých se generují otázky:

- Generovaní otázek pro automatický test, otázky se generují před spuštěním konkrétního studentského testu a přiřazují se přímo k němu.
- Systém umožní na základě nastavených štítků v testové šabloně vygenerovat statické otázky a přiřadit je k šabloně.
- V některých případech se vyučujícímu bude hodit vygenerovat otázky pro statický test, proto systém otázky vygeneruje a přiřadí do obecného testu.
- Obecné generování otázek, otázky jsou vygenerovány, ale nejsou přiřazené k žádnému z testů.

1.3.1.8 FR8: Logování otázek

Systém bude umožňovat uchovávat historii změn otázky. Každá otázka může mít libovolní počet referenčních odpovědí, tyto odpovědi jsou součástí otázky, proto se bude jejich libovolná změna ukládat také. V historii se bude zaznamenávat: co se změnilo, kdo změnu provedl, datum provedení změny a druh operace. Dále systém bude umožňovat zobrazení všech změn, které byly provedené u konkrétní otázky.

Aktéři: vyučující, garant.

1.3.1.9 FR9: Správa a vyhodnocení odpovědí

Budoucí testový modul, stejně jako v současnosti, budeme umožňovat editovaní, tvorbu a mazaní testových referenčních odpovědí. Odpovědi bude možné rozdělit na referenční a studentské. Referenční odpovědi vytváří učitel a garant, testové – student. Vyučující bude schopen ze studenské odpovědi udělat

referenční v průběhu manuální opravy. Dále všechny odpovědi jsou rozdělené dle následujících druhů:

- text,
- diagram,
- relační algebra,
- SQL dotaz,
- výběr jedné z více možností,
- výběr mnoha z více možností,
- normalizace relačního schématu,
- transformace.

Pokud systém nebyl schopen vyhodnotit některou ze studentských odpovědí, tak ji vyhodnotí vyučující manuálně. Po tom, co učitel ohodnotil odpověď, všechny stejné studentské odpovědi budou také automaticky opravené. V podstatě nový testový modul nabízí stejné funkce jako předchozí, s jedním rozdílem, bude umožněno paralelní zpracování opravy odpovědí a testů. Popis současných funkcí systému můžete najít v podkapitole 1.1.

Aktéři: vyučující, garant, student.

1.3.1.10 FR10: Ověření správnosti odpovědí typu SQL/RA

Občas se může stát, že otázka obsahující odpovědi typu SQL/RA už nemůže být automaticky vyhodnocená, protože databázové přípojení přestalo fungovat nebo někdo smazal databázi potřebnou pro opravu této otázky. Proto nový testový modul by měl moct jednou za nějakou dobu, projít všechny otázky v systému a zkontrolovat jejich správnost. Pokud podobný problém nastane, systém označí otázku a všechny šablony, kde se dána otázka vyskytovala nevalidními, což znemožní jejich použití v testech.

1.3.1.11 FR11: Správa testových šablon

V systému bude uživatel schopen vytvářet, upravovat, duplikovat a mazat testové šablony. Šablona bude obsahovat *název, autora, dobu trvání obecného testu, dobu trvání studentského testu, celkový počet bodů, jazyk, otázky pro tvorbu statického testu, skupinu otázek a seznám štítků pro tvorbu dynamického testu*. Z testové šablony pak bude možné vyrobit jeden ze dvou obecných testů. Před vytvořením testu systém zkontroluje správnost testové šablony. Pokud se budeme snažit vytvořit statický test, šablona musí obsahovat alespoň jednu přiřazenou otázku. Ve případě dynamického testu, musí šablona obsahovat skupinu otázek a přiřazené štítky.

Aktéři: vyučující, garant.

1.3.1.12 FR12: Nezávislost testové šablony

Obecný test bude obsahovat veškeré informace, které jsou uložené v testové šabloně. Tento test ji vlastně bude kopírovat včetně nastavených otázek a štítků. Proto se testová šablona stane nezávislou na obecném testu, což umožní jednoduchou změnu vlastností šablony bez vlivu na vytvořené obecné testy. Toto výrazně zlepší znovupoužitelnost testové šablony během drobných modifikací. Obecný test na ní teď vůbec nebude závislý, ale bude zachovávat informace o šabloně, ze které byl vyroben.

1.3.1.13 FR13: Správa obecných testů

Systém bude umožňovat tvorbu obecného testu. Obecný test může být *statický* nebo *dynamický*. Statický test je testem, ve kterém uživatel může manuálně nastavovat otázky a jejich pořadí v testu. Dynamický (nebo také generovaný) test je testem s automaticky generovanými otázkami. Obecný test musí být v nějaké skupině testů. Skupina testů umožní vytvářet několik variant stejného testu a k nim přidělit studenty, což původně nebylo možné. Navíc bude podporovat provádění testů různých druhů současně, např. test v semestru a zkoušku, v rámci jednoho termínu. Podrobný přehled funkcí současného stavu lze nalézt v podkapitole 1.1.6.

Aktéři: vyučující, garant.

1.3.1.14 FR14: Správa studentských testů

Vyučující může přidávat či odebírat studenty z obecného testu, který může být následujících druhů: *demo test, test v semestru, zkouška*. Druh testu zadává vyučující během vytváření obecného testu. Demo testy se budou vyhodnocovat automaticky. Naproti tomu u testů v semestru a zkoušek budou automaticky hodnoceny pouze úplně správné odpovědi, zbytek zůstane na vyučujícím. V průběhu zkoušky a testu v semestru se bude logovat IP adresa každého studenta během ukládaní odpovědí, aby bylo možné zabránit podvádění. Oproti současnému stavu, kde pouze demo test může být automatický, bude umožněno vytvořit libovolný druh testu, který může být zároveň dynamický. Otázky v dynamickém testu se budou generovat během spouštění testu studentem. Opakované spuštění testu bude umožněno pouze u demo testu.

Aktéři: vyučující, garant, student.

1.3.2 Nefunkční požadavky

1.3.2.1 NFR1: Zachování původních dat

V současnosti existuje v systému velké množství vytvořených otázek. Je žádoucí tato data zálohovat, abychom o ně nepřišli. Až bude celkový systém dokončen, nahrají se do nových databázových struktur. Tento požadavek se týká pouze testových zadání, otázek a referenčních odpovědí.

1.3.2.2 NFR2: Volba technologií

Jelikož je zbytek systému naprogramován v jazyce PHP na základě frameworku Nette, v případě volby jiné technologie by integrace a budoucí údržba testového modulu byla složitá. Z tohoto důvodů se bude používat stejný jazyk a další knihovny tohoto jazyka, které nabízí potřebné funkce.

1.3.2.3 NFR3: Provázanost komponent

Nový testový modul by měl být co nejvíce nezávislý na ostatních modulech systému. Místa, kde dochází ke spojení, budou jasně definována, aby bylo umožněno provádění jednoduchých změn ostatních komponent bez velkého zásahu do testové části.

Kapitola 2

Návrh

V předchozí kapitole jsme se zabývali podrobnou analýzou systému, a to nejen jeho funkcí, ale i zhodnocením nevýhod a problémů, které současná implementace a databáze přináší. Na základě těchto zkoumání a definice rozšíření budeme konstruovat nový testový modul, který bude rozdělen na menší samostatné části, které budou mezi sebou nezávislé a budou komunikovat pomocí rozhraní. Toto by mělo výrazně zjednodušit systém, umožnit jeho snazší modifikace a rozvoj. Abychom těchto cílů dosáhli, na jedné ze schůzek ohledně projektu bylo rozhodnuto použít Object Relation Mapping (ORM) pro práci s databází. ORM umožní úplně oddělit datovou vrstvu od aplikační, což bude mít pozitivní dopad na budoucí rozvoj systému. Proto v následujícím návrhu už budeme počítat s tím, že databázové tabulky jsou mapované na softwarové třídy a dle toho stavět naši aplikaci.

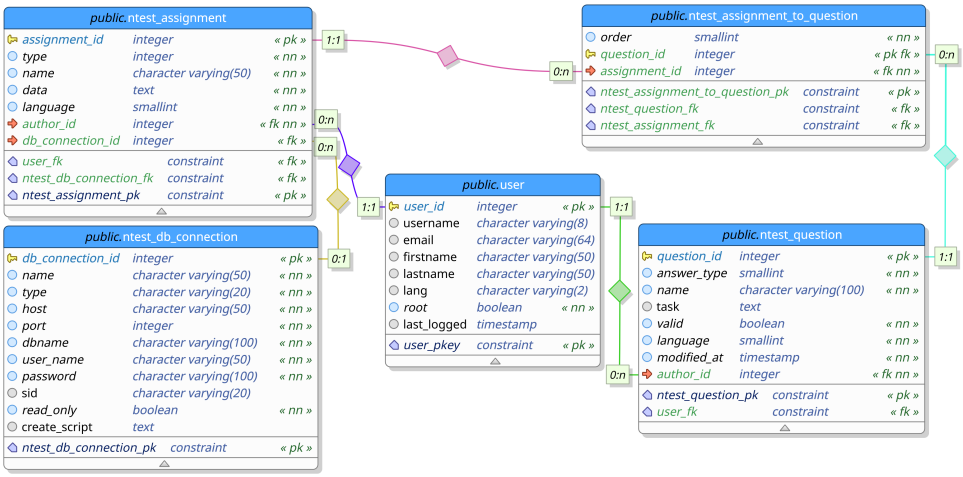
Návrh systému rozdělíme do dvou hlavních částí. První se bude věnovat návrhu vhodné struktury pro uchovávaní dat. Jak je patrné z podkapitoly 1.2.1, současný stav databáze je nevhodný, navíc nám nedovolí realizovat další funkce, které se od toho očekávají. V poslední části se budeme zabývat návrhem systému v souladu s nově kladenými požadavky a provedenou analýzou.

2.1 Návrh struktury databáze

2.1.1 Testové zadání

Testová zadání se budou ukládat do tabulky *ntest_assignment*, která obsahuje atributy:

- `name` – název zadání;
- `type` – druh zadání;
- `data` – obsah zadání v závislosti na konkrétním druhu;



Obrázek 2.1: Relační schéma: Testové zadání

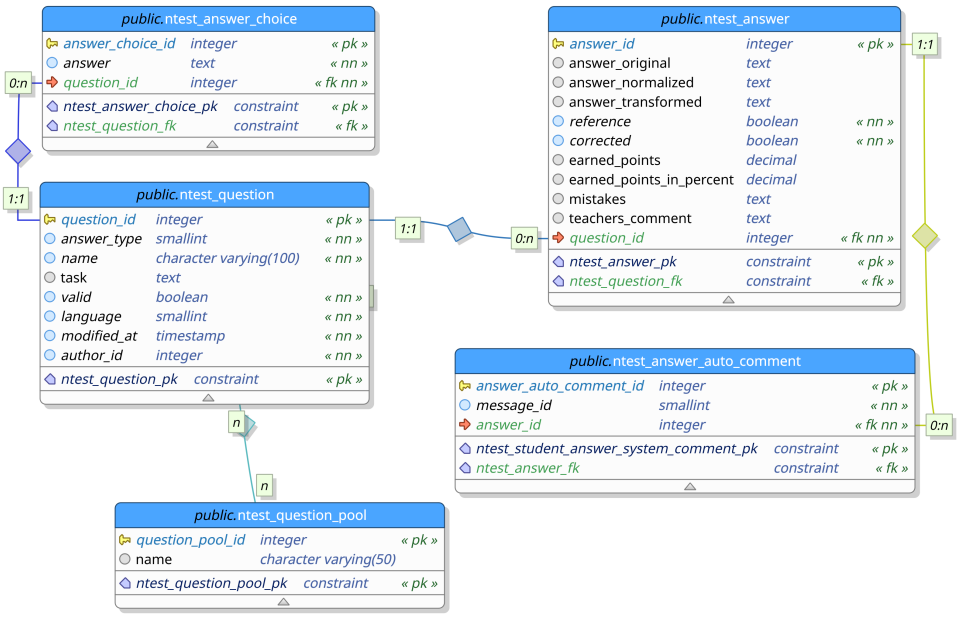
- `author_id` – cizí klíč odkazující do tabulky *`user`*. Každé zadání musí mít autora, tj. uživatele, který ho vytvořil;
- `db_connection_id` – cizí klíč odkazující do tabulky *`ntest_db_connection`*. Nastavuje se pouze v případě, když testové zadání má typ *databáze*.

Celková struktura této části databáze zůstává stejná a většinou se podobá současnému řešení. Nově zde bylo přidáno zadání typu *databáze*, které bude obsahovat databázové připojení. Tato připojení budou sdílená se všemi vyučujícími a určená pouze pro testový modul, což souvisí s požadavkem o novém druhu zadání, viz podkapitola 1.3.1.1. Také v tabulce *`ntest_question`* můžeme vidět zmizení několika atributů, ale o tom si povíme více v následujících podkapitolách. Kromě toho byly změněny databázové typy některých atributů, např. `language`, `order` a další. Jelikož tyto atributy nikdy nenabývají velkých hodnot, je logičtější použít *smallint* místo obyčejného *integer*. Relační schéma této části databáze je zakresleno na obrázku 2.1.

2.1.2 Otázky a odpovědi

V této podkapitole si detailněji povíme o změnách, které jsou spojeny s uložením otázek a odpovědí. Hlavně budeme navazovat na analýzu provedenou v podkapitole 1.2.1.1. Relační schéma vidíme na obrázku 2.2.

Na základě analýzy bylo rozhodnuto sjednotit studentskou a referenční odpověď do jedné tabulky. Navíc se objevuje tabulka *`ntest_answer_choice`*, která bude mít jednoduchý ukol. Tato tabulka bude uchovávat veškeré možnosti pro zaškrtávací typy odpovědí. Správné možnosti se budou ukládat jako obyčejné referenční odpovědi do tabulky *`ntest_answer`*. Do těla odpovědi

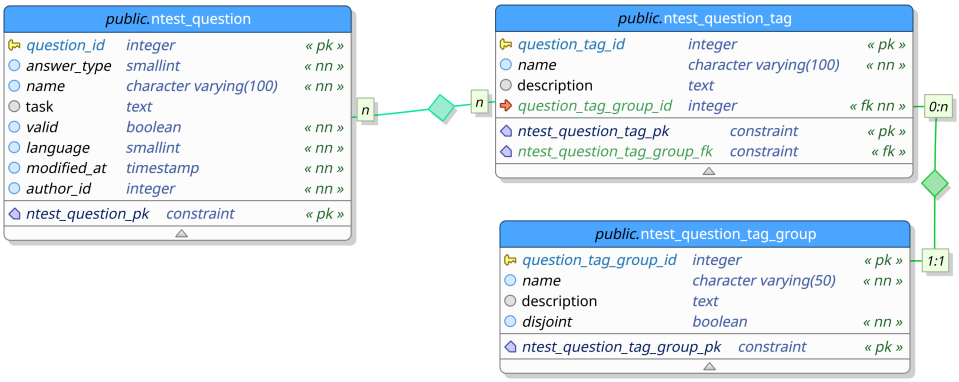


Obrázek 2.2: Relační schéma: Otázky a odpovědi

se budou ukládat identifikátory řádků, dle kterých poznáme správnou možnost. Podobným způsobem se budou uchovávat studentské odpovědi. Tabulka *`ntest_answer`* bude mít následující atributy:

- `answer_original` – originální odpověď;
- `answer_normalized` – normalizovaná odpověď;
- `answer_transformed` – modifikovaná odpověď;
- `reference` – označuje, jestli odpověď je referenční nebo ne;
- `corrected` – vyhodnocení, odpověď je ohodnocená nebo čeká na hodnocení;
- `earned_points` – body, které dostává student za svou odpověď;
- `earned_points_in_percent` – podíl bodů získaných studentem k celkovému počtu bodů za tuto otázku;
- `mistakes` – popis chyb, které byly udělané studentem, nastavuje vyučující;
- `teacher_comments` – komentář učitele.

Dále jsou přidány skupiny otázek (Question pools). Tyto skupiny umožňují učiteli si vytvořit skupinu otázek s konkrétním účelem, např. otázky pro zkoušku nebo demo test. Každá otázka může byt přiřazená do více skupin



Obrázek 2.3: Relační schéma: Štítky

a skupina může obsahovat více otázek. Tyto skupiny budou velmi užitečné při automatickém generování testů nebo při manuálním vyhledávaní v systému.

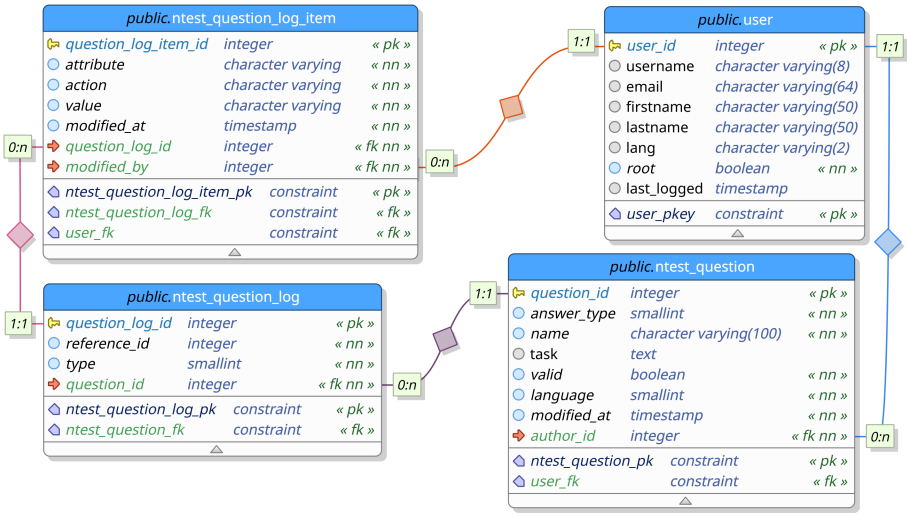
2.1.3 Štítky a otázky

Jak jsme si mohli všimnout v podkapitole 1.2.1.1, rozdělení otázek do skupin na základě několika atributů není vhodné, proto bylo rozhodnuto tyto atributy sjednotit, což bude hlavní zodpovědnost tabulky *`ntest_question_tag`*. Tato struktura dovolí nejen rozdělit otázky do nějakých základních skupin, ale též umožní uživateli vytvářet vlastní typy pro otázky, čímž jednoduše rozšiřovat jejich rozdělení. Každá otázka bude obsahovat minimálně jeden štítek, jeden stejný štítek může být přiřazen k několika otázkám. Relační schéma popisující tyto tabulky lze nalézt na obrázku 2.3. Tabulka *`ntest_question_tag`* bude obsahovat:

- `name` – pojmenování štítku;
- `description` – stručný popis štítku, k čemu slouží;
- `question_tag_group_id` – cizí klíč označující na skupinu štítků.

Tabulka *`ntest_question_tag_group`*, která popisuje skupinu štítků obsahuje atributy:

- `name` – pojmenování skupiny;
- `description` – popis hlavního účelu skupiny;
- `disjoint` – disjunktivnost skupiny – obecné otázky nemohou obsahovat štítky ze stejné skupiny (otázka nemůže být zároveň složitá a jednoduchá), přesto se to pro některé případy hodí a právě to je hlavním úkolem tohoto atributu.



Obrázek 2.4: Relační schéma: Historie změn otázek

2.1.4 Historie změn otázek

Občas se stává, že někdo z vyučujících změní některé vlastnosti otázky a další si ani toho nevšimne. Toto pak může způsobovat problémy. Abychom byli schopni tyto problémy odstranit, nově se v systému objeví historie změn otázky, která je spojena s požadavkem 1.3.1.8. Relační schéma popisující historii změn se nalézá na obrázku 2.4.

Tabulka *ntest_question_log* představuje historii změn otázky. Tato tabulka má následující atributy:

- `reference_id` – identifikátor záznamu v jedné z tabulek;
- `type` – představuje jednu z tabulek (jelikož máme tři různé tabulky, které souvisí s testovou otázkou, musíme vědět, ve které tabulce nastala změna);
- `question_id` – cizí klíč odkazující na tabulku *ntest_question*.

Tabulka, jež uchovává nastalé změny, má následující atributy:

- `attribute` – název atributu, který byl změněn;
- `action` – druh provedené akce, např. upravení, odstranění;
- `value` – nová hodnota atributu;
- `modified_at` – datum, kdy změna proběhla;
- `modified_by` – cizí klíč odkazující na tabulku *user*, dovolí zjistit, kdo otázku změnil.

2.1.5 Testové šablony

Tato podkapitola se bude věnovat ukládání testových šablon a hlavně změnami oproti současnému stavu. Na základě analýzy, viz podkapitola 1.2.1.2, bylo rozhodnuto odstranit typ studentského testu, který zbytečně provazoval šablonu a studentský test. Navíc bylo přejmenováno několik atributů, jelikož jejich pojmenování neodpovídalo úkolům, které plní. Některé atributy také změnily datový typ na vhodnější, např. `total_points`. Relační schéma popisující strukturu tabulek, která je spojena s testovou šablonou, je zachyceno na obrázku 2.5.

V souladu s kladenými požadavky, o kterých se zmiňuji v podkapitole 1.3.1.11, testová šablona bude navíc obsahovat skupinu otázek a přiřazené štítky. Šablona je reprezentována tabulkou *ntest_test_template*, která obsahuje následující atributy:

- `name` – název testové šablony;
- `student_test_time` – čas na vypracování testu studentem;
- `general_test_time` – celkové časové okno na vypracování testů;
- `total_points` - maximální počet bodů za test;
- `valid` – správnost testové šablony;
- `language` – jazyk;
- `active` – označení, jestli šablona už byla použita v některém z testů;
- `question_pool` – cizí klíč odkazující na tabulku *ntest_question_pool*, která znamená skupinu otázek (Question pool).

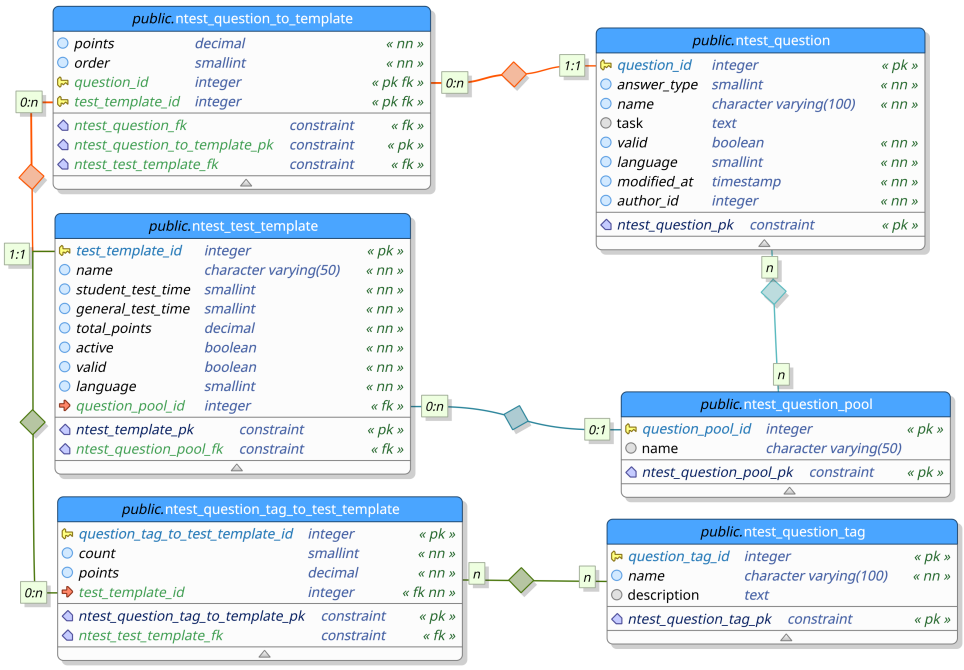
Další novou tabulkou je *ntest_question_tag_to_test_template*. Tato tabulka umožní nastavit veškeré potřebné informace pro generování otázek. Pro přehlednost budeme záznamy z této tabulky nazývat „dynamické otázky“. Toto řešení bylo navrhnuto společně během jedné ze schůzek ohledně projektu a poskytuje pouze základní možnosti pro práci se štítky, protože další rozšíření by už byly velice složité. Tato tabulka má atributy:

- `count` – počet generovaných otázek;
- `points` – maximální počet bodů, který student může získat za tuto otázku;
- `test_template_id` – testová šablona, ke které patří dynamická otázka.

2.1.6 Obecné testy

V této části se podíváme na ukládaní obecných testů, které bylo navrženo v souladu s požadavky specifikované v podkapitole 1.3.1.13. Relační model je zobrazen na obrázku 2.6.

Jak lze vidět z tohoto modelu, uložení obecného testu je trochu komplikovanější, navíc zčásti kopíruje atributy testové šablony. Z tohoto důvodu se



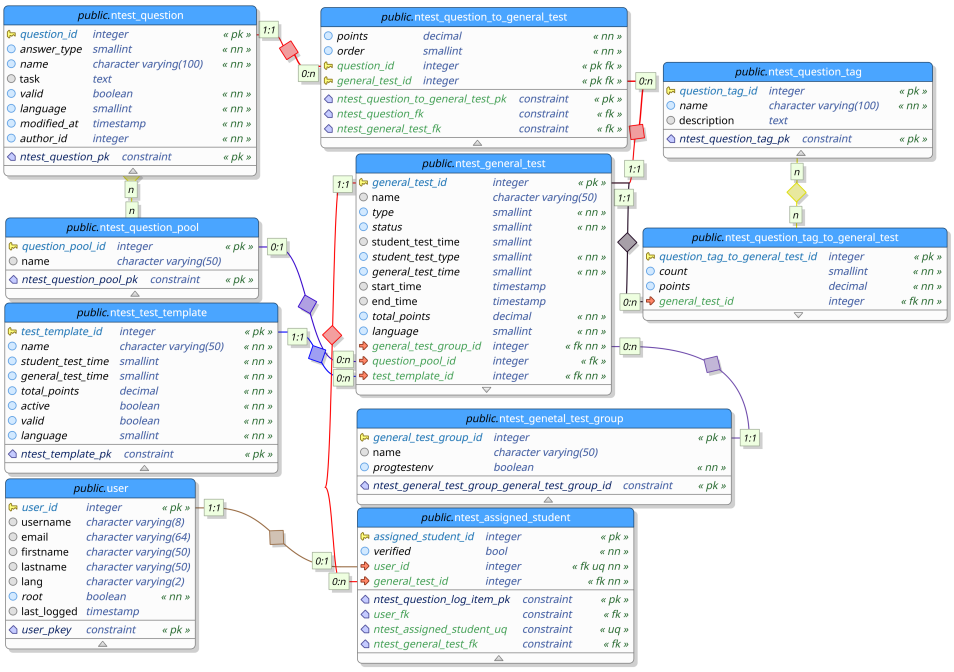
Obrázek 2.5: Relační schéma: Testová šablona

omezíme pouze na část atributů, které jsou odlišné. Obecný test je reprezentován tabulkou ***ntest_general_test***, která obsahuje následující atributy:

- `name` – název obecného testu;
- `type` – druh testu, může být *statický* nebo *dynamický*;
- `student_test_type` – druh studentského testu, může to být *demo test, zkouška* nebo *test v semestru*;
- `status` – stav, ve kterém se test nachází;
- `start_time` – čas spuštění testu;
- `end_time` – čas ukončení testu.

Tabulka ***ntest_assigned_student*** uchovává informace o přiřazených studentech k testu, a umožňuje kontrolu studentů před jeho začátkem.

Dále se podíváme na tabulky ***ntest_question_to_general_test*** a ***ntest_question_tag_to_general_test***, které uchovávají stejné informace o přiřazených statických a dynamických otázkách, které obsahuje testová šablona, a během vytváření testu se jednoduše zkopírují. Podobným způsobem se kopíruje skupina otázek. Přesto, že jsou tato data duplicitní, existuje rozumný důvod, proč by je obecný test měl obsahovat. Tato úprava úplně ruší závislost obecného testu na šabloně, ze které byl vyroben. Testovou šablonu



Obrázek 2.6: Relační schéma: Obecný test

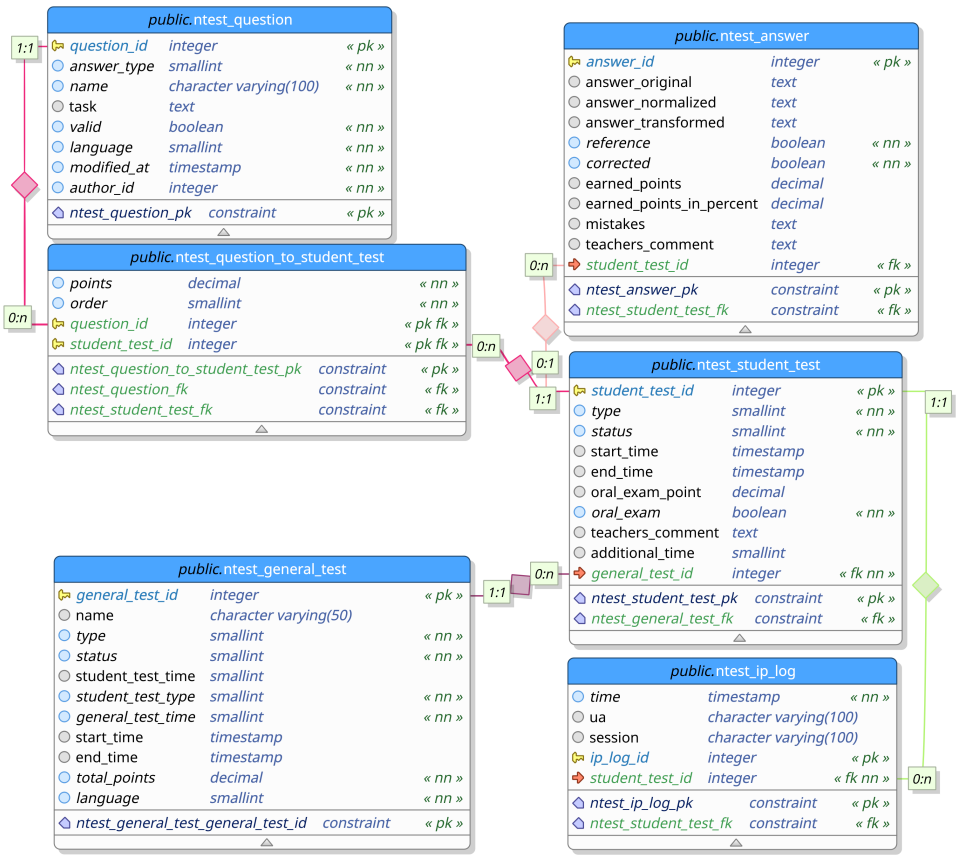
lze jednoduše modifikovat bez vlivu na obecné testy. Souvisí to s rozšířeními, která se očekávají, viz podkapitola 1.3.1.12.

Jak můžeme vidět na diagramu, všechny tyto vazby jsou nepovinné, protože to, které otázky budou přiřazeny, rozhoduje druh testu. Statický test bude obsahovat pouze statické otázky, na rozdíl od dynamického, který bude obsahovat otázky dynamické, a také skupinu otázek, ze které se tyto otázky budou generovat.

Nakonec se podíváme na testové skupiny a jaké nové možnosti tyto skupiny přináší. Testovou skupinu představuje tabulka ***ntest_general_test_group***, která má atributy:

- `name` – název testové skupiny;
- `progtestenv` – označení, dovolující studentům vyplňovat test z obyčejného operačního systému nebo pouze ze specifického, který byl instalován na školní počítače. Tento atribut byl původně obsazen v obecném testu. Vzhledem k tomu, že obecný test se stane pouze jednotlivou variantou zkoušky či testu, tak toto povolení bude docela zbytečné v rámci jedné varianty.

Testová skupina bude moct obsahovat několik obecných testů, které mohou být libovolného druhu, např. statická zkouška a zároveň dynamický test



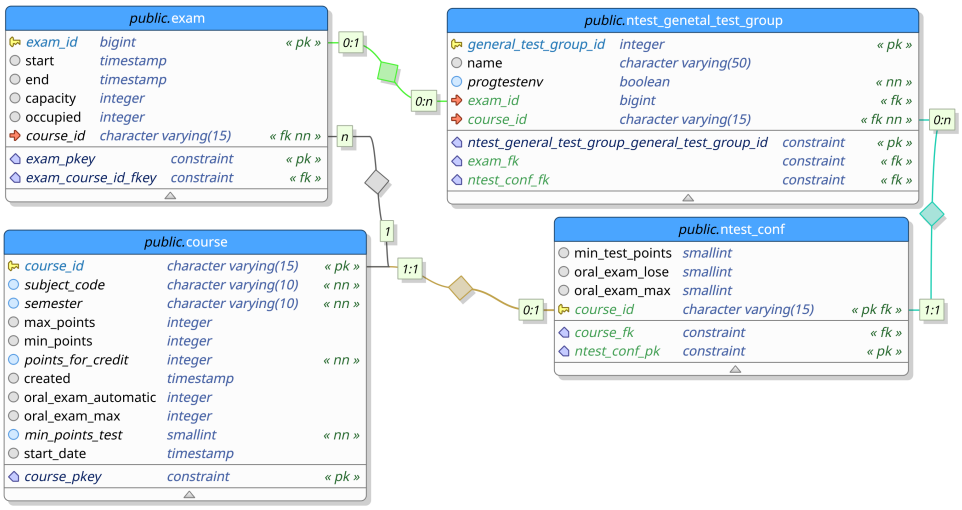
Obrázek 2.7: Relační schéma: Studentský test

v semestru apod. Navíc už by mělo být jasné, že můžeme tvořit různé varianty testu pro jeden termín, což původně bylo možné pouze pomocí různých testů. Tato skupina v podstatě představuje konkrétní termín testu. Toto přináší spoustu zajímavých možností pro další vývoj.

2.1.7 Studentské testy

Tato podkapitola se bude zabývat změnami, které jsou spojené s ukládáním studenských testů. Relační model je možné nalézt na obrázku 2.7. Studentský test representuje tabulka ***ntest_student_test***, která zůstala skoro stejná až na dva důležité rozdíly. Stručný popis současného řešení lze dohledat v podkapitole 1.2.1.2.

První rozdíl spočívá v tom, že se objevil atribut `type`, který nám říká, o jaký druh testu se jedná. Tuto činnost původně plnila šablona, proto jsme mohli sledovat silnou provázanost. Druhý rozdíl umožňuje systému ke kon-



Obrázek 2.8: Relační schéma: Konfigurace semestru

krétnímu testu přiřazovat generované otázky a tím zabezpečí existence dynamických testů, které systém generuje samostatně. Vazební tabulkou mezi testy a otázky je *ntest_question_to_student_test*. Tato tabulka obsahuje následující atributy:

- `order` – pořadí otázek v testu, v současnosti se využívat nebude. Tuto vlastnost pak budou generovat generátory otázek dle konkrétních požadavků;
- `points` – maximální počet bodů, které student může získat za danou otázku.

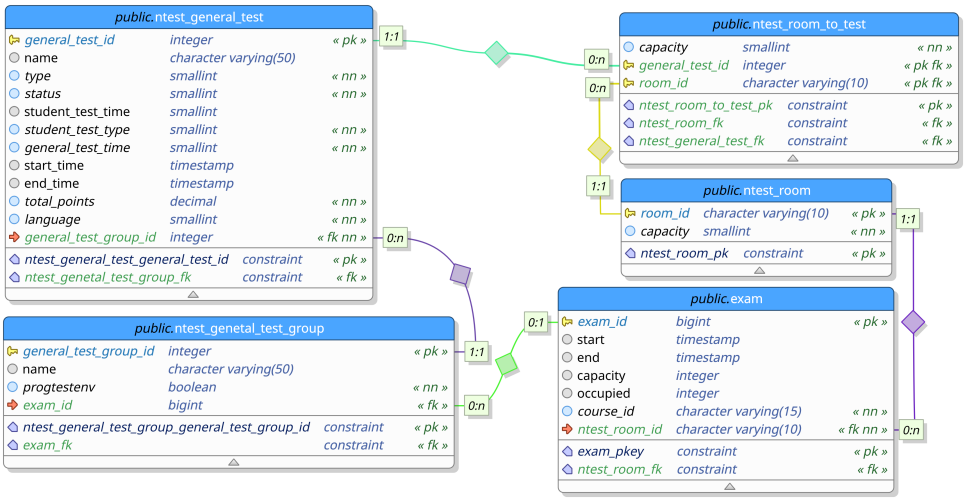
Také se podíváme na tabulku *ntest_ip_log*, jejímž hlavním účelem je uchování informace spojené s přístupem studenta k testu. Z této tabulky byl odstraněn atribut reprezentující uživatele, protože je duplicitní a lze ho získat z tabulky *ntest_student_test*.

2.1.8 Konfigurace semestru a místností

V souvislosti s nedostatky, jež byly popsány v podkapitole 1.2.1.3, konfigurace semestru bude vyčleněna do samostatné tabulky, což zmenší provázanost mezi komponentami a dovolí testovému modulu být nezávislým a samostatným celkem. Relační schéma znázorňuje obrázek 2.8.

Tabulka *ntest_conf* bude obsahovat celkovou konfiguraci modulu, její atributy jsou následující:

- `min_test_points` – minimální počet bodů, potřebný pro splnění testu v semestru;



Obrázek 2.9: Relační schéma: Testové místnosti

- `oral_exam_lose` – ztracené body, pokud se student rozhodne jít na ústní zkoušku;
- `oral_exam_max` – maximální počet bodů za ústní zkoušku.

Jelikož tabulka ***`course`*** je sdílená mezi všemi moduly, tak ji zatím necháme beze změn. Kromě oddělení těchto atributů, bylo lehce upraveno jejich pojmenování, abychom lépe rozuměli, co konkrétní atribut znamená. Navíc byl kvůli malým hodnotám upraven jejich datový typ.

Nakonec zmíníme změny, které nastaly v uchovávání testových místností. V této části databázové struktury jsou odstranění atributu `name` v tabulce ***`ntest_room`*** a změna umělého primárního klíče na název místnosti. Jelikož pojmenování místností v rámci univerzity je vždy unikátní, není důvod používat umělý identifikátor. S tímto pak souvisí změna v tabulce ***`exam`***, kam byl přidán cizí klíč odkazující na tabulku ***`ntest_room`***. Relační model je na obrázku 2.9.

2.2 Návrh systému

Tato podkapitola se bude věnovat podrobnému popisu návrhu softwarových struktur (tříd), které byly navržené na základě analýzy testového modulu a v souladu se současnými požadavky kladenými na systém. Tyto struktury budou reprezentovat nezávislé balíčky funkcí poskytující rozhraní pro presentery (Nette presenters), které budou následně zajišťovat pouze komunikaci mezi komponenty. V současnosti presentery vykonávají mimo komunikaci také velkou část práce v systému, což způsobuje značnou provázanost a nízkou soudržnost, viz kapitola 1.2.2.

Pro znázornění funkcí systému a lepšímu porozumění danému problému budeme využívat diagramy tříd jazyka UML (Unified Modeling Language).

Vzhledem k tomu, že tyto diagramy obsahují mnoho různých objektů, stávají se nepřehlednými. Proto si definujeme grafické rozdělení těchto tříd následovně:

- Class – entity,
- Class – třídy reprezentující datovou vrstvu,
- Class – třídy reprezentující logiku aplikace,
- Class – rozhraní či abstraktní třídy,
- Class – služby.

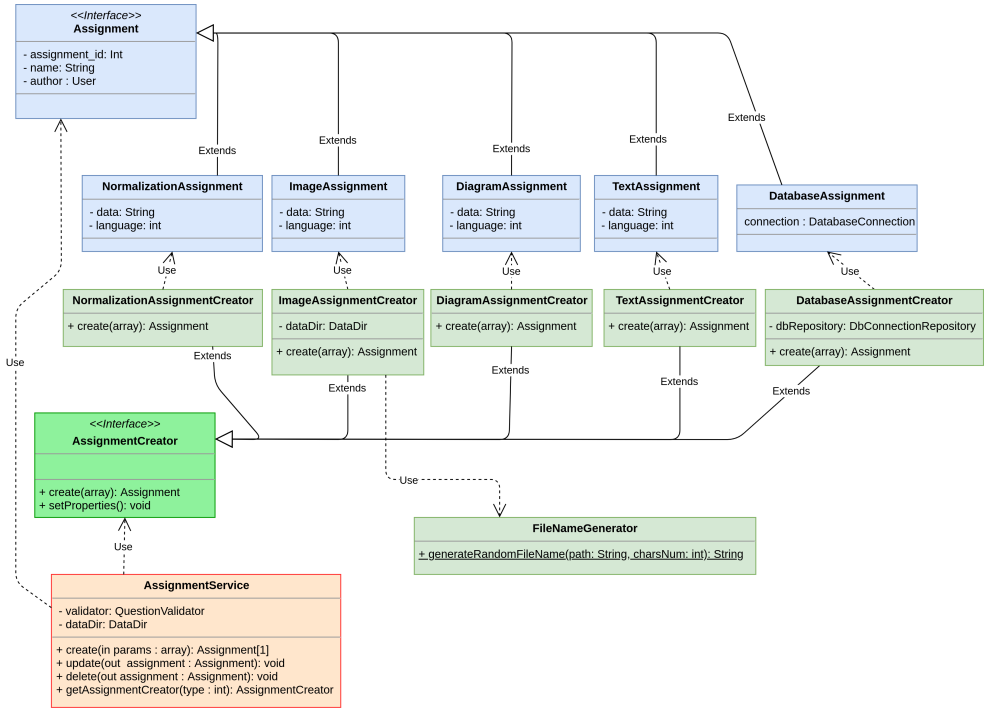
2.2.1 Assignment Creators

V této části se zaměříme na návrh tvorby testového zadání. Jak už víme z analýzy, máme tyto druhy testových zadání: *text, obrázek, diagram, normalizace* a *databáze*. Diagram tříd popisující strukturu zodpovědnou za tvorbu zadání vidíme na obrázku 2.10.

Každý druh se vytváří jiným způsobem, proto bylo rozhodnuto přidat rozhraní `AssignmentCreator`, jehož jednotlivé implementace určí, jakým způsobem se různé druhy zadání budou vytvářet. Pro klienta, který se bude zabývat tvorbou zadání, není důležité, s jakým druhem zadání pracuje. Toto umožní jednoduchou změnu postupu tvorby libovolného druhu zadání a přidání nových bez zásahu do fungujícího kódu. Toto řešení vlastně představuje takzvaný **factory method pattern**[8].

Třída `AssignmentService` představuje klienta, který se bude zabývat správou zadání. Tato služba obsahuje metodu *getAssignmentCreator()*, která vrací konkrétní implementaci rozhraní `AssignmentCreator` na základě druhu testové otázky. Můžeme si všimnout, že parametrem metody *create()* je pole.

[8] Factory method pattern je návrhový vzor poskytující rozhraní pro tvorbu objektů a zároveň umožnující podtřídám měnit typ vytvořeného objektu včetně způsobu jeho tvorby. [22]



Obrázek 2.10: Diagram tříd: Answer Creators

Toto nám umožní jednoduše předávat parametry mezi frontendem a backendem pomocí JSON[9] přes REST[10] rozhraní, jelikož se JSON snadno převádí na pole.

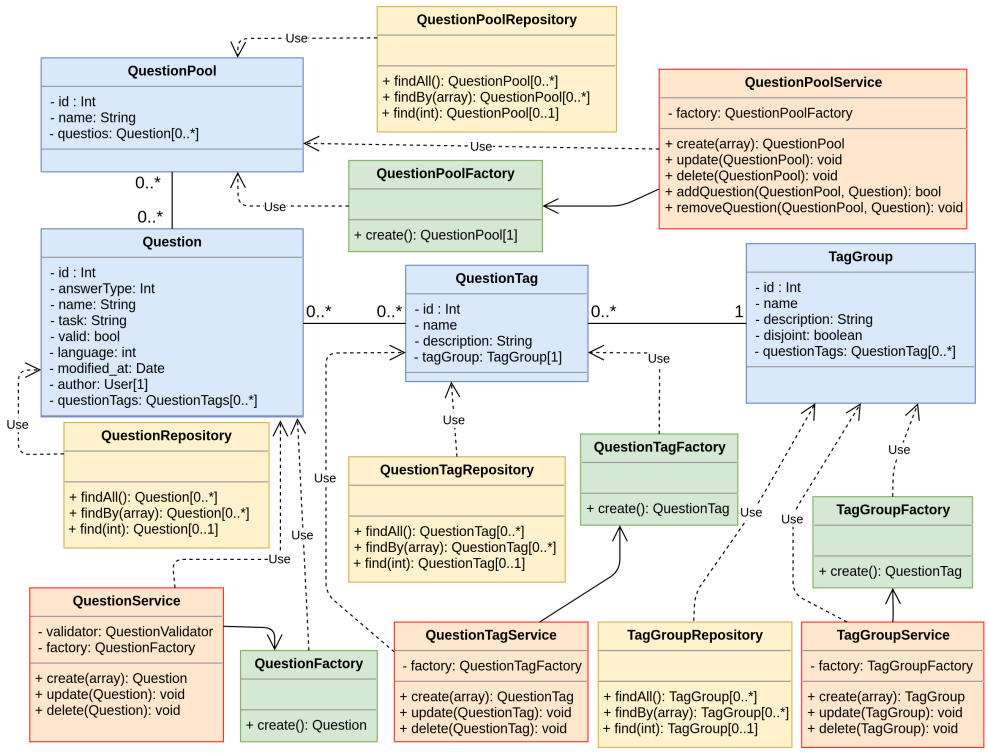
2.2.2 Questions and Tags

V této podkapitole se zaměříme na popis vytvoření otázek a štítků. Rozdělení zodpovědností jednotlivých tříd je zobrazeno na obrázku 2.11.

Jak je zřejmé z diagramu tříd, postup tvorby těchto entit je velmi jednoduchý. Přesto jsme jej rozdělili na čtyři samostatné nezávislé části, služby, které se mimo jiné zabývají tvorbou entit. Tyto služby obsahují metodu *create()*, která bere jako parametr pole s atributy potřebnými pro vytvoření příslušné entity. V případě, že atribut je entita, toto pole bude obsahovat její identifi-

[9] JSON – JavaScript Object Notation je jednoduchý standardizovaný textový formát reprezentující strukturovaná data a umožňující jejich přenos prostřednictvím internetových sítí. Nejčastěji je využíván webovými aplikacemi pro přenos dat mezi klientem a serverem. [23]

[10] REST – REpresentational State Transfer je architektura rozhraní poskytující standardy mezi počítačovými systémy na internetu, umožňující jednoduchou komunikací mezi klientem a serverem. [24]



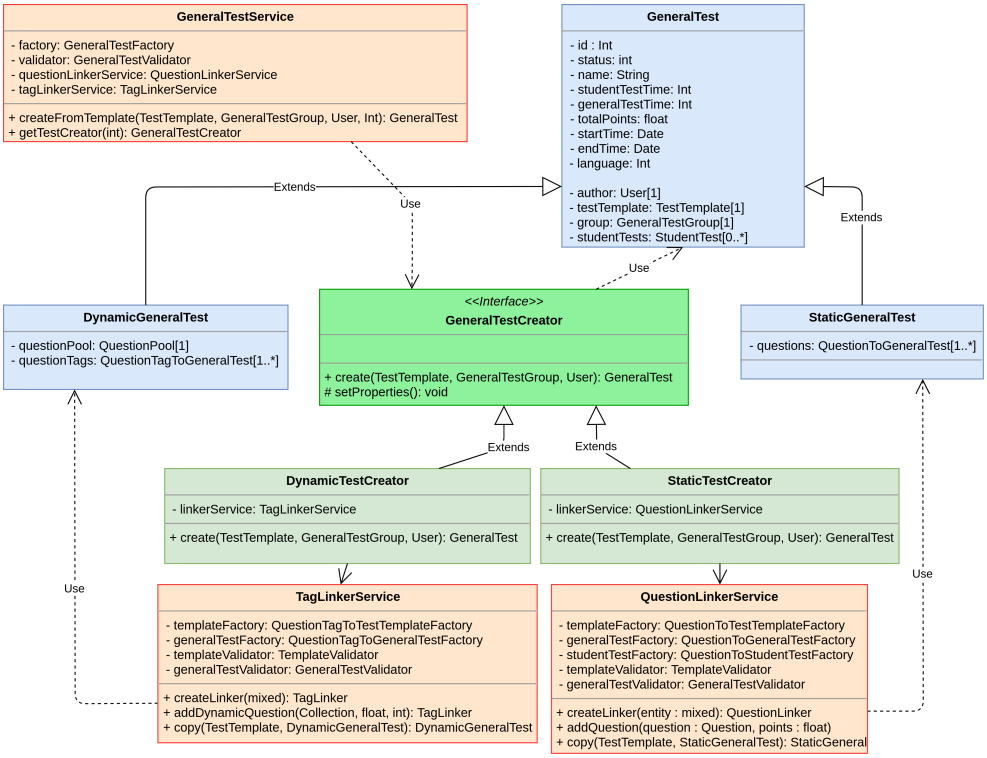
Obrázek 2.11: Diagram tříd: Tvorba otázek a štítek

kátor, na jehož základě se pak natáhne z databáze odpovídající entita. Například každý štítek obsahuje skupinu, tj. služba `QuestionTagService` použije metodu *find()* třídy `TagGroupRepository` pro získaní štítkové skupiny. Následovně služby předávají získaná data továrnám, které už vědí jakým způsobem vytvořit konkrétní entitu.

Služba `QuestionService` navíc obsahuje instanci třídy `QuestionValidator`, která se zabývá ověřením správnosti dané otázky. Po každém vytvoření či změně, bude otázka ověřena a atribut `valid` se nastaví na správnou hodnotu.

2.2.3 General test Creators

V souladu s požadavky kladenými na systém, viz podkapitola 1.3.1.13, bude obecný test rozdělen na dva odlišné testy: *statický* a *dynamický*. Postupy tvorby těchto testů jsou odlišné a proto, abychom dosáhli snadné údržby a rozšiřitelnosti, použijeme už známý **factory method pattern**, který nám dovolí rozdělit implementaci tvorby testu na samostatné třídy. Diagram tříd popisující tuto část systému můžete najít na obrázku 2.12.



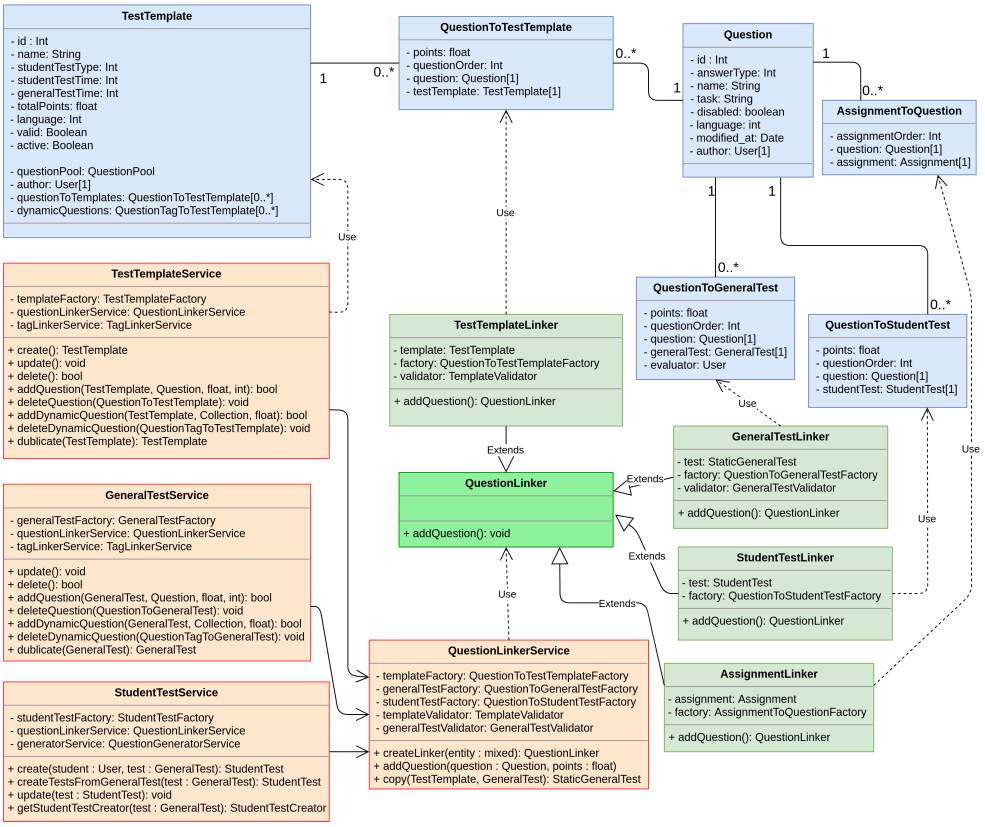
Obrázek 2.12: Diagram tříd: General test Creators

Služba `GeneralTestService`, která je zodpovědná za správu obecných testů, bude obsahovat jako závislost další dvě důležité služby: `QuestionLinkerService` a `TagLinkerService`. Obecně se tyto služby zabývají přiřazením otázek a štítků, a hlavně vědí jakým způsobem se s tím pracuje. O těchto službách a o úkolech, které budou splňovat, si povíme více v následující podkapitole. Teď jsou pro nás důležité pouze metody *copy()*, jejichž hlavním úkolem je kopírování veškerých otázek nebo štítků z testové šablony do testu v závislosti na tom, kterou službu voláme. To, o jaký test se bude jednat, rozhodne vyučující v době tvorby testu, podle čeho metoda *getTestCreator()* vrátí konkrétní implementaci abstraktní třídy `GeneralTestCreator`. Následně se tato třída bude využívat pro tvorbu obecného testu. Pokud při vytvoření testu nastane chyba, vyhodí se odpovídající výjimka.

2.2.4 Linkers

2.2.4.1 Question Linkers

Tato podkapitola se bude zabývat přiřazením otázek k testům a zadání. Diagram tříd popisující tento problém je zachycen na obrázku 2.13.



Obrázek 2.13: Diagram tříd: Question Linkers

Jak jsme si už mohli všimnout v průběhu návrhu struktury databáze, přiřazení otázek se obecně řeší pomocí vytvoření vazební entity mezi otázkou a další entitou. Tato operace je ovšem stejná pro všechny entity, ke kterým můžeme přiřadit otázku, dokonce tyto entity obsahují skoro stejné atributy. Z tohoto vyplývá, že bychom mohli přiřazení otázek standardizovat a zodpovědnost za tuto činnost nechat na nějakém samostatném balíčku tříd. Proto bylo rozhodnuto vytvořit abstraktní třídu `QuestionLinker`, která se bude zabývat přiřazením otázek v závislosti na druhu cílové entity. Toto řešení zabrání zbytečným duplicitám v kódu a zjednoduší jeho pochopení, protože veškeré přiřazování otázek máme na jednom místě. Navíc bude úplně nezávislé na zbytku kódu.

Služba `QuestionLinkerService`, kterou už známe z předchozí podkapitoly, se zabývá tvorbou konkrétních implementací rozhraní `QuestionLinker` na základě typu entity. Představuje klienta, který se zabývá přiřazením otázek pomocí odpovídajícího linkeru a také jejich kopírováním ze šablony do obecného testu. Otázky mohou být přiřazené k *zadání, testové šabloně, obec-*

ném testu nebo *studentskému testu*. Během přiřazování se vytvoří konkrétní vazební entita a nastaví se její potřebné atributy. Každá entita se vytváří pomocí továren (ty nejsou na diagramu zobrazeny kvůli nedostatku místa). Hlavní úkolem linkerů je použít správnou továrnu a nastavit správné hodnoty parametrů, proto jejich implementace je jednoduchá.

2.2.4.2 Tag Linkers

V této podkapitole se budeme zabývat přiřazováním štítků. Toto řešení je podobné tomu, co jsme probírali v předchozí podkapitole. Diagram tříd je zachycen na obrázku 2.14.

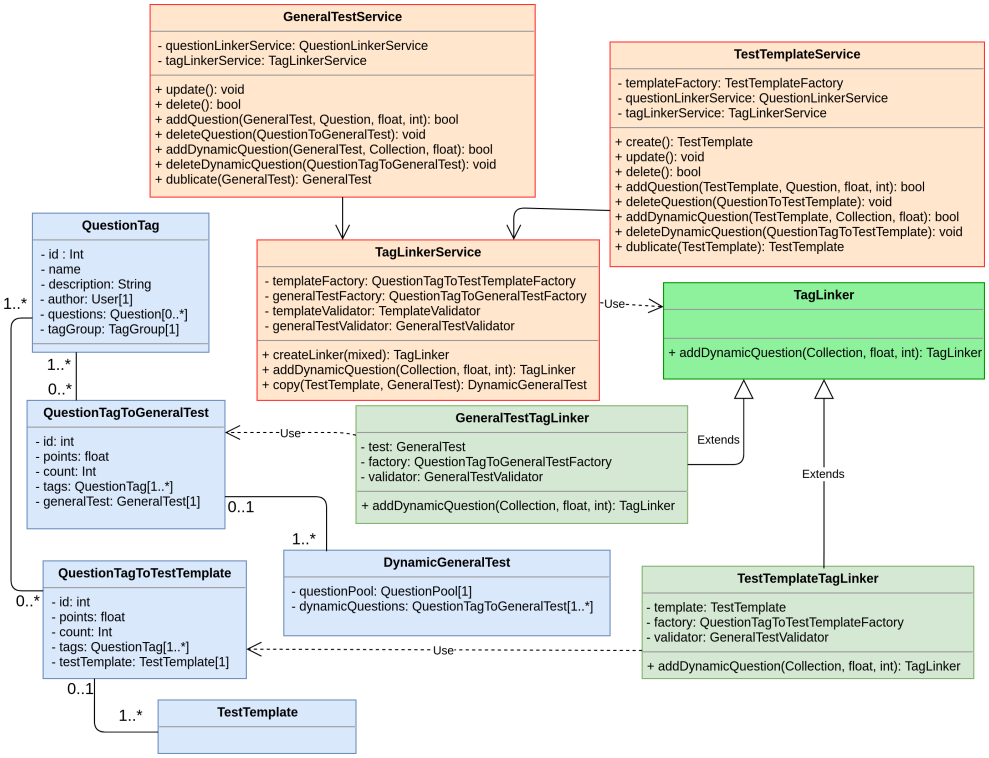
Štítky je možné přiřazovat pouze k entitám `TestTemplate` a `GeneralTest`. Zodpovědnost za přiřazení štítků bude přidělena abstraktní třídě `TagLinker` a jejím implementacím, které budou vytvářet odpovídající vazební entity. K vytvoření správného linkeru slouží metoda *createLinker()*, která na základě typu entity určí odpovídající podtřídu. Můžeme si vzpomenout, že jsme tabulky, které reprezentují tyto entity, pojmenovali *dynamické otázky*, viz podkapitola 2.1.5. Toto označení budeme využívat i dále.

Dynamické otázky, na rozdíl od statických, jsou komplikovanější, neboť každá z nich může obsahovat více štítků a také každý štítek může patřit k více než jedné otázce. Dynamické otázky jsou reprezentovány pomocí tříd `QuestionTagToTestTemplate` a `QuestionTagToGeneralTest`, jejichž atributy jsou následující:

- `points` – maximální počet bodů, který student může získat za tuto otázku;
- `count` – počet vygenerovaných otázek;
- `tags` – štítky, které musí obsahovat otázka, aby byla vybraná generátorem otázek;
- `generalTest` nebo `template` – obecný test nebo šablona v závislosti k čemu přiřazujeme.

Jak je vidět na diagramu, linkery také obsahují příslušně validátory, které ověřují dynamickou otázku před přiřazením. Pokud bude otázka nesprávná, vyhodí se výjimka. U každé další otázky se musí kontrolovat správnost nastavených bodů, abychom byli schopni zakázat přiřazení otázky, pokud součet bodů všech otázek je větší, než celkový počet bodů v testu.

Pro přidání dynamické otázky potřebujeme použít pouze metodu *addDynamicQuestion()* a předat odpovídající parametry. Tedy je nám jedno k čemu přiřazujeme dynamickou otázku, toto si služba vyřeší sama použitím příslušného linkeru na základě typu entity, kterou předáme jako parametr metody.



Obrázek 2.14: Diagram tříd: Tag Linkers

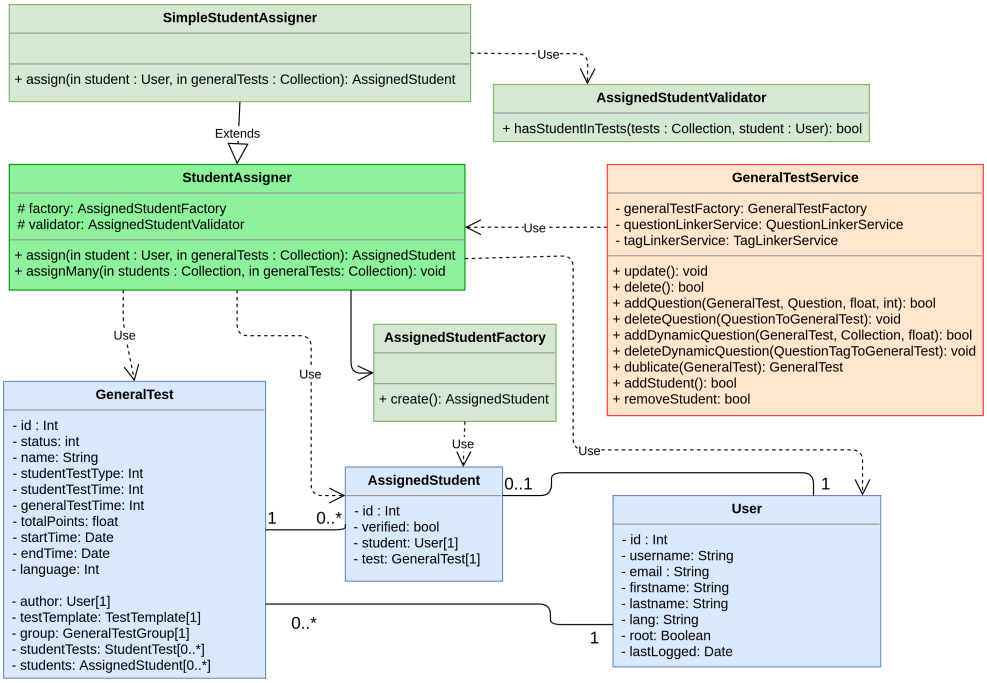
2.2.5 Student Assigners

Tato podkapitola se bude zabývat přiřazením studentů k obecnému testu. Diagram tříd popisující tyto funkce lze nalézt na obrázku 2.15.

Původně se přiřazení studentů řešilo pomocí metod v presenteru a každé přiřazení či odebrání studenta vedlo k vytvoření nebo odstranění studentského testu. Při přidání studenta k obecnému testu se vytvářel studentský test. Není žádoucí, aby se tento test vytvářel a my museli zasahovat tam, kde to není nutné. Je logičtější tento test vytvořit v době spuštění obecného testu nebo pomocí samostatné akce, kterou si vyvolá vyučující.

Z těchto důvodů bylo rozhodnuto o přidání nové entity `AssignedStudent`, která bude reprezentovat přiřazeného studenta k testu, viz podkapitola 2.1.6. Toto umožní jednoduše přiřazovat studenty k obecnému testu a následně před jeho spuštěním snadno provést kontrolu přítomnosti studentů, jak pomocí kontroly studentských karet, tak i manuálně vyučujícím. Navíc se teď nebude přistupovat k entitě `StudentTest`, což trochu sníží provázanost systému.

Jelikož bylo umožněno vytvářet více variant pro stejný test, viz podkapitola 2.1.6, přiřazené studenty bude možné rovnoměrně rozdělovat mezi několik obecných testů. Předpokládá se, že způsob přiřazení se může měnit, proto se

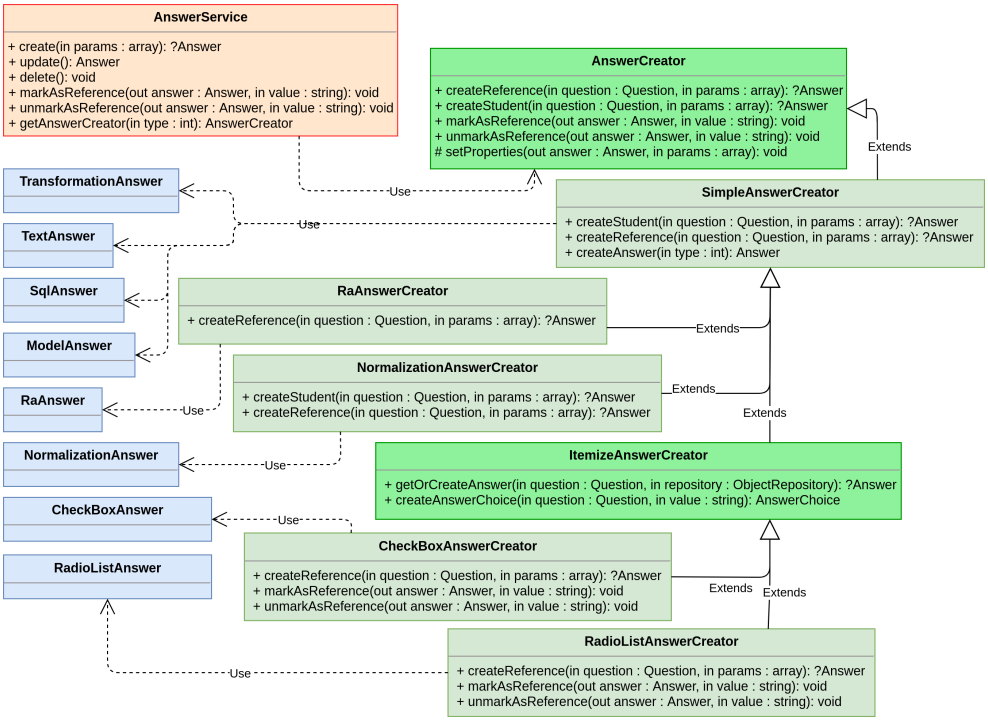


Obrázek 2.15: Diagram tříd: Student Assigners

rozhodlo přidat rozhraní `StudentAssigner`, jehož implementace se budou zabývat přiřazením studentů. Před samotným přiřazením je dobré ověřit, zda daný student už není přiřazen k tomuto testu, jinak bychom mohli spustit dva stejné testy pro jednoho studenta. Zodpovědnou za tuto činnost bude třída `AssignedStudentValidator`, která před přiřazením ověří přítomnost studenta v obecném testu.

2.2.6 Answer creators

Tvorba odpovědí v systému je komplikovaná, protože samostatná odpověď může být různých druhů, viz podkapitola 1.1.3.1, navíc odpovědi také dělíme na studentské a referenční, což výrazně zvětšuje složitost tvorby a obecně práci s odpovědmi. Na základě provedené analýzy, kterou jsme se zabývali v podkapitole 1.1.3.1 a návrhu struktury databáze, viz podkapitola 2.1.2, bylo rozhodnuto sjednotit studentskou a referenční odpověď a následně přidat abstraktní třídu `Answer` a její potomky reprezentující konkrétní druh odpovědí. Abychom byli schopni vyprodukovat kód, který bude splňovat podmínku jednoduché rozšiřitelnosti a bude odolnější vůči změnám, vytvoříme strukturu tříd zodpovědných za tvorbu odpovědí v závislosti na jejich typu. Diagram tříd popisující tuto strukturu se nachází na obrázku 2.16.

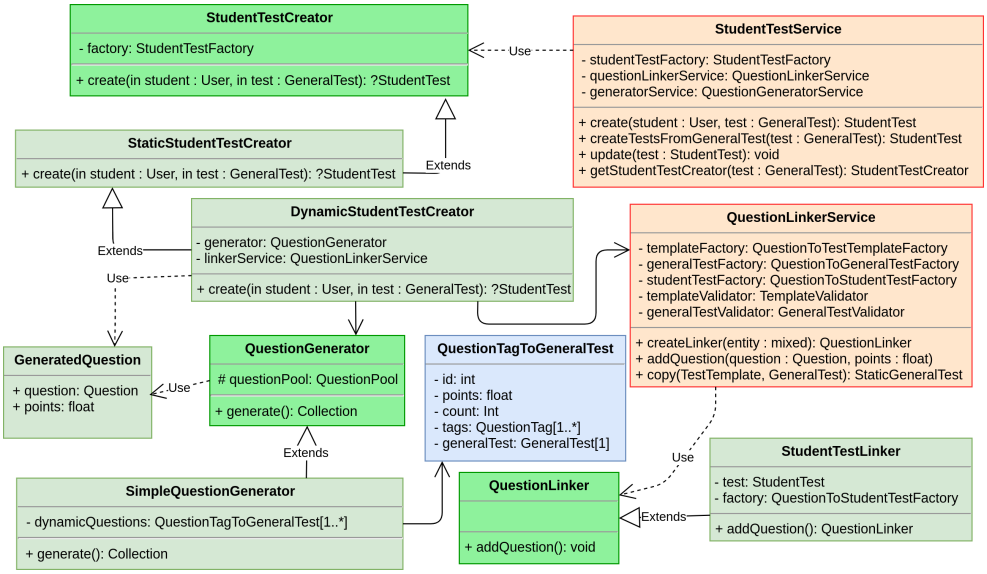


Obrázek 2.16: Diagram tříd: Answer Creators

Za tvorbu odpovědí bude zodpovědná služba `AnswerService` reprezentující klienta, který bude využívat rozhraní `AnswerCreator`, jehož chování je přepsané v potomcích v závislosti na typu odpovědi. Jelikož skoro půlka odpovědí má podobný způsob tvorby, jeho hlavní implementaci představuje třída `SimpleAnswerCreator`, která nabízí možnost tvorby odpovědí typu: *text, transformace, diagram* a *SQL*.

První odlišný postup tvorby má odpověď typu *RA*, která navíc potřebuje uložit transformovanou odpověď (překlad do jazyka SQL). Už víme z analýzy, že odpověď typu *normalizace* má úplně odlišný způsob tvorby, proto přidáváme podtřídu `NormalizationAnswerCreator`, která se ale bude implementovat v budoucnu.

Dále si můžeme všimnout ještě jedné třídy `ItemizeAnswerCreator`. Tato třída umožňuje ukládaní odpovědí, které jsou zaškrtávacího typu. Pro uložení těchto možností se bude využívat entita `AnswerChoice`, která reprezentuje jednu samostatnou referenční možnost, viz podkapitola 2.1.2. Abychom mohli zjistit, které z těchto možností jsou správné, musíme vytvořit referenční odpověď obsahující identifikátory správných možností. Jelikož máme dva druhy odpovědí, které obsahují možnosti, tak tvorbou odpovědí se budou zabývat podtřídy `CheckBoxAnswerCreator` a `RadioListAnswerCreator`. To, která im-



Obrázek 2.17: Diagram tříd: StudentTest Creators

plementace se použije, rozhodné klient pomocí metody *getAnswerCreator()*, která na základě typu odpovědi vrátí odpovídající implementaci.

Přesto, že toto řešení je docela složité, má své výhody. Snižuje počet duplikací kódu, které v současnosti existují ve tvorbě odpovědí. Navíc je zjevné, jak jednoduché je změnit některou z implementací. Vždy upravujeme pouze třídu, která souvisí s konkrétním druhem odpovědi.

2.2.7 Test Runner

V současnosti systém dovoluje studentovi sledovat, ke kterým testům je aktuálně přiřazen, ale neumožňuje tyto testy spustit, dokud vyučující nespustí obecný test. Až obecný test bude spuštěn, studenti mohou své testy otevřít a následně se pokusit o jejich splnění. Přesto, že je funkce zabývající se otevřením testu docela složitější na pochopení, operace je to jednoduchá a její hlavní činnost spočívá jen v tom, že se změní stav testu. V případě, že druh testu je *automatický demo test*, tak se navíc generují a přidávají otázky. Tato funkce je komplikovaná z několika důvodů. Musíme kontrolovat druh testu a pak ještě jeho stav, což přidává spoustu větvení a zhoršuje přehlednost a pochopení kódu. Ukázku této funkce uvádím v příloze na obrázku E.1.

Jelikož bylo umožněno tvorbu nejen automatického demo testu, ale i obecně libovolný test může být automatický, spouštění testu se stalo složitějším, protože automatický test mohl být generován studentem v době spouštění. Tuto možnost budeme chtít zachovat, navíc chceme umožnit generovaní testů vyučujícím před spouštěním obecného testu. Z těchto důvodů bylo rozhodnuto

oddělit vytváření studentských testů od jejich spuštění, což je lepší z pohledu rozdělení zodpovědností a snižuje provázanost systému. Proto byla přidána abstraktní třída `StudentTestCreator` zodpovědná za vytvoření testu v závislosti jen na druhu obecného testu, jelikož různé druhy studentského testu při tvorbě obsahují stejně informace. Diagram tříd popisující tvorbu studenského testu je zachycen na obrázku 2.17.

Tvorba statického testu je velice jednoduchá, proto se zaměříme na vytváření dynamického testu. Dynamický studentův test se bude vytvářet na základě nastavených štítků a skupiny otázek. Tyto testy lze pak vytvářet uprostřed běhu obecného testu a také před jeho spuštěním. Samotnou tvorbu umožňuje třída `DynamicStudentTestCreator`, která obsahuje generátor otázek a linker, pomocí kterých metoda *create()* nejdříve vygeneruje set otázek (toto generování záleží na konkrétní implementaci generátoru otázek) a následně je přiřadí k vygenerovanému studentskému testu pomocí linkeru.

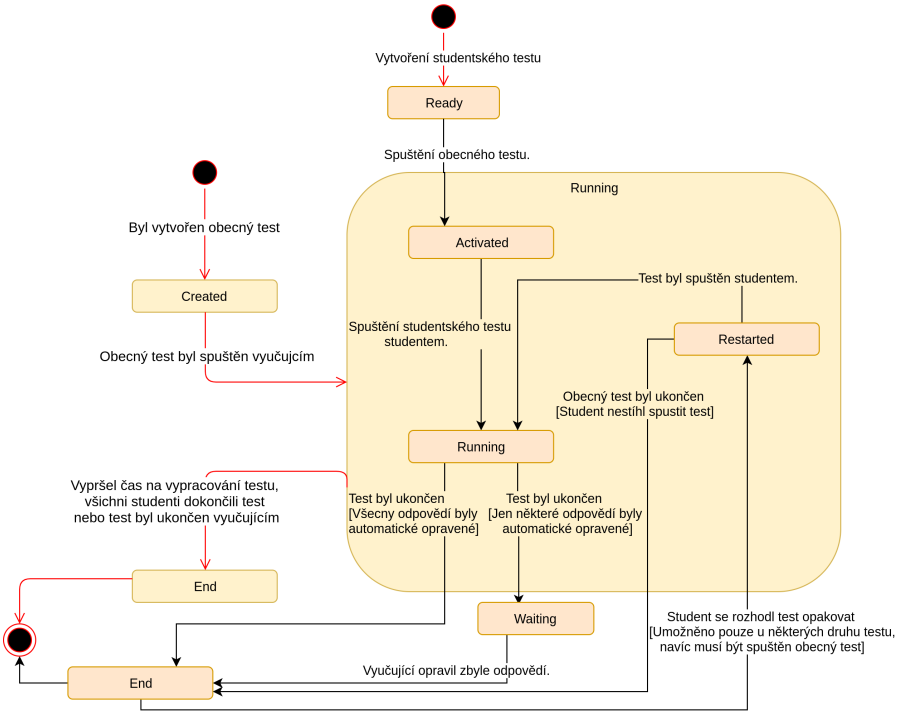
Konečně jsme se dostali do samotného spuštění a otevření testu. Už víme, že tato akce záleží na aktuálním testovém stavu. Abychom mohli zjednodušit kód vykonávající tuto činnost a odstranit zbytečné větvení, použijeme návrhový vzor **state**[11], který nám umožní rozhodovat v závislosti na stavu, ve kterém se test nachází. Jelikož přechody mezi jednotlivými stavy jsou komplikované, pro lepší porozumění těmto přechodům a testovým stavům použijeme stavový diagram jazyka UML, který můžete nalézt na obrázku 2.18.

Na tomto diagramu kromě stavů studentského testu vidíme také stavy testu obecného. Stavy studentského testu jsou závislé na stavech obecného, proto by je nemělo smysl zobrazovat zvlášť. Obecný test má pouze tři stavy: *Created, Running* a *End*. Studentův test jich má víc:

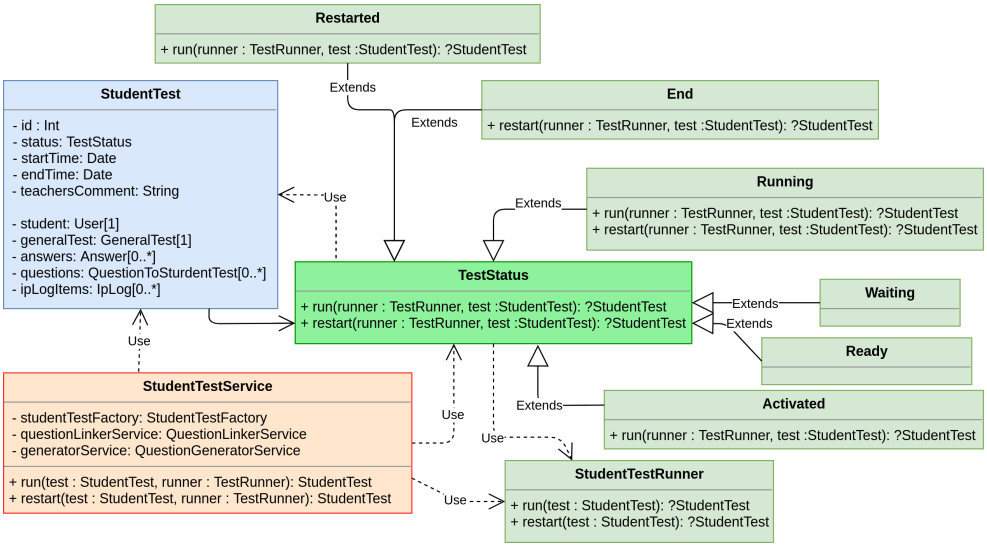
- *Ready* – test je vytvořen přiřazen studentovi;
- *Activated* – test je zpřístupněn, student ho může spustit;
- *Running* – test je spuštěn studentem;
- *Waiting* – test nebyl automaticky plně vyhodnocen, proto čeká na opravu;
- *End* – test byl ukončen a všechny otázky byly vyhodnocené;
- *Restarted* – test je opakovaně připraven, student ho může spustit.

Spouštěním testu se bude zabývat třída `StudentTestRunner`, která navíc bude umožňovat test opakovaně spouštět. Abychom plně využili **state pattern** služba `StudentTestService` dostane studentův test a následovně deleguje práce na testový stav. Na základe toho, ve kterém stavu se test aktuálně nachází, se zavolá odpovídající metoda u konkrétního stavu, která rozhodne, jestli může být test spuštěn nebo ne. V kladném případě zavolá metodu *run()* třídy `StudentTestRunner`, která test spustí a nastaví čas začátku a ukončení testu. Jestliže test už byl spuštěn, tak se tato metoda volat nebude, pouze

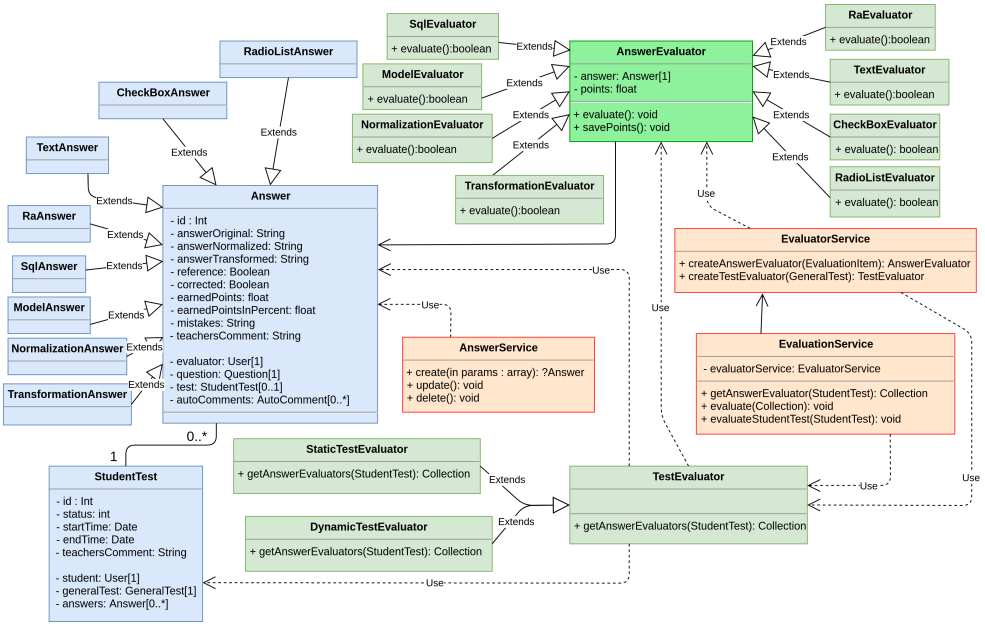
[11] State je návrhovým vzorem chování, který umožňuje měnit chování konkrétního objektu na základě změny jeho stavu. [25]



Obrázek 2.18: Stavový diagram: Stavy testů



Obrázek 2.19: Diagram tříd: Test Runner



Obrázek 2.20: Diagram tříd: Test Evaluators

se vrátí studentův test. Pokud test byl ve stavu, který nedovoluje jeho spuštění nebo otevření, vrací se `null` hodnota. Obdobně se toto provádí v případě opakování testu, navíc se tady kontroluje, jestli test může být opakován.

2.2.8 Test Evaluators

V této podkapitole se podíváme na automatické hodnocení odpovědí v testu a případné jeho budoucí rozšíření. Diagram tříd popisující tuto funkci lze nalézt na obrázku 2.20.

Současné řešení umožňuje automatické hodnocení odpovědí studentů v době odevzdání testu, což docela zatěžuje systém v případě, když se najednou odevzdá hodně testů. Rozhodlo se, že nový testový modul bude obsahovat frontu s neohodnocenými odpověďmi, které se budou postupně odebírat a hodnotit pomocí paralelního zpracovaní.

Víme, že testová odpověď může být různých druhů, viz podkapitola 1.1.3.1, proto je zřejmé, že každý z těchto druhů bude mít odlišný způsob vyhodnocení. Z tohoto důvodu se rozhodlo přidat abstraktní třídu `AnswerEvaluator`, jejíž implementace se budou zabývat hodnocením otázek v závislosti na typu odpovědi. Tyto podtřídy budou obsahovat studentskou odpověď a další parametry potřebné pro vyhodnocení. To umožní snadno přidávat objekty ke frontě a následně je zpracovávat, což souvisí s požadavkem o paralelním vyhodnocením testů, viz podkapitola 1.3.1.13.

Abychom byli schopni ohodnotit odpověď, potřebujeme znát maximální počet bodů, který student může dostat. Vzhledem k tomu, že existují dva typy testu *dynamický* a *statický*, zjištění maximálního počtu bodů není úplně triviální. Proto bylo přidáno rozhraní `TestEvaluator`, které se bude zabývat zjištěním bodování otázky v konkrétním testu a následovně generovat množinu evaluatorů. Pro správu těchto tříd budou sloužit dvě služby `EvaluatorService` a `EvaluationService`. První z nich se bude zabývat jen tvorbou evaluatorů, na základě poskytnutých parametrů. Další bude určená pro provádění hodnocení.

Vytvořené dynamické otázky se přiřazují přímo k studenskému testu. Tuto vazbu představuje entita `QuestionToStudentTest` obsahující maximální počet bodů, který student může získat za správně zodpovězenou otázku. Podtřída `DynamicTestEvaluator` tyto body získá a následně vytvoří datové objekty (třída `EvaluationItem`), které spojují odpověď a počet bodů. Navíc tyto datové objekty obsahují další vlastnosti potřebné pro vyhodnocení. Na základě těchto objektů se vytvoří odpovídající evaluator pomocí metody *createAnswerEvaluator()*. Obdobným způsobem podtřída `StaticTestEvaluator` získá počty bodů obsažené ve vazební entitě `QuestionToStudentTest`.

2.2.9 Adapters

Co když budeme potřebovat použít nějakou část systému, která už není podporována kvůli refaktorování a která souvisí s jinými komponenty v systému? Jakým způsobem můžeme zachovat nezávislost testového modulu? Abychom mohli tohoto dosáhnout, použijeme návrhový vzor **adapter**[12]. Pomocí tohoto návrhového vzoru můžeme jednoduše přizpůsobit třídy či rozhraní, které nejsou kompatibilní. Kromě toho tento návrhový vzor umožňuje určit místo, kde dojde ke spojení testového modulu s jinými částmi. Proto když se změní nějaká funkce jiných komponent, snadno najdeme a upravíme potřebný kód a navíc bude tato úprava vždy na jednom místě.

Během automatické opravy otázek se používá třída `SkriptRunner`, která je zodpovědná za vyhodnocení SQL a RA dotazů. Tato třída používá databázové připojení, které se nachází v jiné komponentě. V novém návrhu bude testový modul využívat svá vlastní připojení (třída `DbConnection`). Proto, abychom mohli použít třídu `SkriptRunner`, vytvoříme `SkriptRunnerAdapter`. Jeho hlavní úkolem bude přizpůsobit třídu `SkriptRunner`, aby mohla pracovat s novým typem databázového přípojení. Dále na všech místech, kde budeme potřebovat instanci třídy `SkriptRunner`, použijeme její obálku, čímž za prvé přizpůsobíme funkčnost původní třídy a za druhé celkový testový modul bude závislý pouze na třídě `SkriptRunnerAdapter`, nikoli na `SkriptRunner`, která není součástí tohoto modulu.

[12] Adapter je návrhovým vzorem umožňujícím spolupráci objektů s různými rozhraními. [26]

Kapitola 3

Realizace

V této kapitole se budeme věnovat zajímavým částem vývoje základního prototypu backendu testového modulu na základě vytvořeného návrhu, který jsme popisovali v předchozí kapitole.

Nejprve se podíváme na nastavení prostředí a integraci knihoven, které jsme se rozhodli používat. Dále se budeme zabývat implementací některých částí a provázaností testového modulu s ostatními moduly DBS portálu, jelikož mnoho částí systému tento modul používají. Je důležité je zmínit, abychom byli schopni naznačit, jaké části systému bude potřeba upravit v průběhu budoucí integrace testového modulu do zbytku systému. Na konec se podíváme na základní testování prototypu a použité technologie.

3.1 Konfigurace prostředí

Jak už bylo zmíněno, budeme používat technologii ORM, která umožňuje jednoduché mapování databázových tabulek na softwarové třídy, čímž odděluje ukládaní dat od logiky aplikace. To pak dovoluje nejen snadno získávat data, ale i postavit lepší architekturu aplikace. Jelikož v současnosti používaný framework Nette vůbec nepodporuje technologii ORM, použijeme knihovnu Doctrine 2. Tato knihovna plně podporuje funkce, které potřebujeme, navíc nabízí celou řadu zajímavých možností, které se budou dát využít při budoucím rozvoji systému, viz podkapitola 3.5.4.

Abychom byli schopni tuto knihovnu používat, potřebujeme ji integrovat s Nette. S tímto problémem nám pomůže balíček Nettrine, jehož hlavním účelem je tato integrace.

Bohužel v DBS portálu se využívá ještě starší verze Nette, proto budeme muset použít balíček Nettrine verze 0.3. Toto je jediná možná verze, která je v současnosti podporována systémem.

Po instalaci potřebných knihoven, jednoduše provedeme i konfiguraci. Jelikož je tento balíček určen pro tuto integraci, stačí pouze zaregistrovat další

```
extensions:
    # Registrace DBAL
    dbal: Nettrine\DBAL\DI\DbalExtension
    dbal.console: Nettrine\DBAL\DI\DbalConsoleExtension

    # Registrace ORM
    orm: Nettrine\ORM\DI\OrmExtension
    orm.cache: Nettrine\ORM\DI\OrmCacheExtension
    orm.console: Nettrine\ORM\DI\OrmConsoleExtension
    orm.annotations: Nettrine\ORM\DI\OrmAnnotationsExtension

dbal:
    # Konfigurace databázového připojení (lokální)
    connection:
        host: localhost
        user: adminDBS
        password: *******
        dbname: newdbs
        driver: pdo_pgsql

orm:
    configuration:
        autoGenerateProxyClasses: true
    # Nastavení uživatelského manažera entit
    entityManagerDecoratorClass: App\Modules\NewTest\Database\EntityManager

orm.annotations:
    # Cesta k entitám
    paths:
        - App/Modules/NewTest/Database/

orm.cache:
   defaultDriver: apcu
```

Obrázek 3.1: Konfigurační soubor config.neon

rozšíření v jednom z konfiguračních NEON souborů, které využívá Nette pro veškeré nastavení projektu. [27] Toto nastavení lze nalézt v konfiguračním souboru `config.neon` na obrázku 3.1.

3.2 Implementace

Implementace testového modulu začala ještě ve fázi návrhu, což umožnilo rychleji opravit některé chyby spojené s návrhem systému. Vzhledem k velikosti projektu, v některých případech byl implementován pouze design komponenty, např. komponenty: AnswerEvaluators, QuestionGenerators či StudentAssigners. Přehled všech hlavních komponent v systému znázorňuje diagram komponent jazyka UML, viz příloha F.1.

Navržené řešení obsahuje komponenty, které jsou na sobě nezávislé, a každá z nich vykonává určitou činnost a má svou omezenou oblast zodpovědnosti. Komponenty můžeme rozdělit do třech samostatných vrstev:

- Presentery – část zodpovědná za přijetí a odesílání dat na frontend (komunikace mezi komponentami, nebyla implementována, proto není zobrazena na diagramu).
- Logika aplikace – obsažená veškerá logika aplikace, presentery s ni komunikují pomocí služeb.
- Datová vrstva – zodpovědná za uložení a selekci dat. Obsahuje entity, reprezentující databázové tabulky a další pomocné třídy zabývající se vyhledáváním dat a jejich ukládáním.

Logiku aplikace můžeme dále rozdělit na 2 části: služby a funkční celky. Služby jsou třídy, které se registrují v konfiguračním souboru a které následně lze získat v libovolném místě programu pomocí Dependency Injection (DI). Tuto činnost nám zajistí Nette framework, tzn. vyžádáme si službu, kterou Nette nějakým způsobem vytvoří a dodá, aniž bychom se zabývali její tvorbou. Tyto služby jsou mezi sebou provázané a představují klienty, které používají vytvořené komponenty. Proto vykonávají především podpůrné činnosti, hlavní funkce si komponenty řeší samostatně. Toto umožňuje jednoduchou úpravu libovolné z nich bez zásahu do zbytku kódu a možnost snadného přidávání dalších komponent.

Datová vrstva obsahuje entity a další třídy, které jsou součástí knihovny Doctrine a zodpovídají za selekci, uložení a modifikaci dat. Tato knihovna nabízí svůj specifický jazyk dotazování Doctrine Query Language (DQL) umožňující psát dotazy, které využívají pouze vlastnosti entitních tříd. Tyto dotazy se následně transformují na SQL a vrací data, která jsou už v podobě entit. [28] Z tohoto důvodu datová vrstva vůbec není vázána na databázovou strukturu či konkrétní databáze, a proto bude snadné provést nějakou změnu.

Navíc Doctrine umožňuje vytvoření uživatelských databázových typů, které se nám hodí, například pro snadnou konverzi testových stavů. [29] Z návrhu víme, že testový stav je reprezentován abstraktní třídou `TestState`, viz podkapitola 2.2.7. Abychom mohli snadno tyto stavy ukládat, byl vytvořen uživatelský datový typ reprezentující testový stav. Tedy před uložením a načtením dat z databáze Doctrine provede příslušné konverze a změní celočíselný identifikátor testového stavu na odpovídající třídu nebo naopak. Nemusíme už

vůbec řešit, jestli testový stav existuje nebo provádět odpovídající konverzi ručně. Všechno za nás vyřeší tato knihovna sama. Můžeme se spolehnout na to, že testový stav bude vždy odpovídat příslušné třídě. Pokud by z nějakého důvodu konverze neproběhla úspěšně, tak bychom dostali výjimku.

3.3 Provázanost a budoucí integrace

Už víme, že testový modul byl docela provázán se zbytkem systému. Abychom mohli v budoucnu jeho novou verzi jednoduše integrovat, projdeme si v této podkapitole místa, kde by tato integrace mohla způsobit problémy v jiných částech systému.

První takové místo, kde dochází k využití struktur testového modulu je databázový pohled *test_students_points*. Tento pohled se snaží sečíst všechny body studenta ve všech testech, proto využívá hodně databázových tabulek testového modulu, např. *test_template*, *test_student_test*. V novém návrhu struktury databáze jsme změnili většinu tabulek, viz podkapitola 2.1, proto bude potřeba tento pohled vhodně upravit, aby byla zachována funkčnost jiných komponent, které jej používají.

Další část systému, která využívá komponenty současného testového modulu jsou e-mailové notifikace. Tato část se používá pro odesílání mailů, např. po vyhodnocení testu se odešle studentovi upozornění, že jeho test byl opraven. Upozornění dostává i vyučující, pokud některé z otázek byly přesměrovány do manuální opravy. Stejným způsobem to funguje v komponentě zodpovědné za semestrální práce: po ohodnocení semestrální práce dostane student upozornění o opravě a vyučující o odevzdání. Tato část systému je provázána s ostatními, proto integrace nového testového modulu v tomto místě nebude jednoduchá. Tato komponenta využívá několik služeb testového modulu, navíc jsou zde využívány databázové tabulky, což propojuje tyto dva moduly. Bylo také plánováno, že by tyto e-mailové notifikace mohly být refaktorovány. Předpokládáme, že toto refaktorování bude spojené s integrací testového modulu, jinak tato integrace bude dost složitá.

Poslední komponenta, která se bude muset do testového modulu integrovat, je normalizace. Tato část představuje kompletní řešení pro správu zadání, otázek a odpovědí typu *normalizace*. Toto řešení bylo navrženo a implementováno v bakalářské práci [7], jejímž autorem je Bc. Marek Erben. Přesto, že tato komponenta je nezávislá na zbytku testového modulu, bude ji potřeba mírně upravit. Při návrhu nové struktury databáze byly odstraněny některé atributy a veškeré tabulky související s touto částí byly modifikovány. Proto během budoucí integrace bude potřeba změnit část kódu, která se zabývá uložením dat, abychom mohli toto řešení dále používat.

3.4 Testování

Tato podkapitola se zabývá testováním vytvořeného prototypu. Pro jeho testování využijeme především jednotkové a integrační testy zabývající se testováním spolupráce komponent. Jelikož toto řešení představuje pouze prototyp, velké množství testů by mohlo zkomplikovat budoucí dokončení implementace. Proto omezíme automatizované testování především na hlavní funkce systému. V budoucnu se dále plánuje přidat vnější integrační testy, které testují spolupráci implementovaných komponent s databází. V současnosti tento druh testů ještě není plně podporován systémem.

Abychom mohli psát automatické testy, budeme využívat výsledky bakalářské práce [6] Ing. Pavla Kováře, jejímž cílem bylo navrhnout a vytvořit prostředí vhodné pro automatizované testování DBS portálu.

3.4.1 Testování formy a obsahu zdrojového kódu

Na vývoji DBS portálu stále pracuje velké množství lidí. Každý z programátorů většinou používá vlastní styl psaní zdrojového kódu, což může zhoršovat jeho celkovou čitelnost. Aby bylo možné sjednotit všechny styly a zlepšit čitelnost kódu, Ing. Pavel Kovář ve své práci [6] vytvořil testování formy kódu. Navíc bylo vytvořeno testování obsahu, které ověří, zda programátor neudělal chyby v datových typech, návratových hodnotách funkcí apod. V této práci tyto automatizované testy použijeme, a tím zajistíme, zda přidaný kód odpovídá stylu, který je využíván v DBS portálu.

3.4.2 Jednotkové a integrační testy

Jednotkové testy jsou určené pro testování nejmenších možných částí systému. Tyto testy obvykle ověřují konkrétní funkce v rámci jedné třídy. Většinou jsou psány programátory, které si těmito testy ověřují, že přidaná nebo změněná část kódu funguje dle očekávaní. Nemůžeme se ale spolehnout, že testované třídy budou spolu fungovat správně, proto by v systému měly být i další druhy testů, např. integrační testy. [30]

Integrační testy můžeme rozdělit na vnitřní a vnější. Vnitřní se používají pro testování větší části systému, např. testování spolupráce několika komponent. Obvykle se nejprve otestují menší funkční celky a pak se ověří spolupráce větších složitějších modulů. Vnější integrační testy se používají, když dochází k propojování aplikací do většího funkčního celku nebo komunikaci s databází. [30]

Jak už bylo zmíněno dříve, testovací prostředí bylo připraveno v bakalářské práci [6], která se zabývala testováním DBS portálu. V systému se používá framework Codeception, proto jej využijeme pro testování i my. Tento framework nám umožní elegantně a efektivně pracovat s jednotkovými a integračními testy, viz podkapitola 3.5.6.

3.5 Použité technologie

3.5.1 Jazyk PHP

PHP (Hypertext Preprocessor) je dynamický skriptovací jazyk, který je jedním z nejvyužívanějších programovacích jazyků k vytváření webových aplikací. Jedná se o jazyk s otevřeným zdrojovým kódem, který funguje na straně serveru a díky své široké a jednoduché podpoře je nainstalován a podporován na většině dostupných hostingů. [31, 32]

3.5.2 PostgreSQL

PostgreSQL je relačním databázovým systémem, který má otevřený zdrojový kód. Je podporován většinou operačních systémů a jeho údržba je jednoduchá, proto se často využívá pro webové aplikace. [13] Podrobněji tento databázový stroj už byl popsán v podkapitole 1.2.1.

3.5.3 Nette

Nette je open source framework pro tvorbu webových aplikací v PHP, který je tvořen množstvím vyspělých a samostatně použitelných komponent. Tento framework je založen na architektuře MVP, která rozděluje zodpovědnosti jednotlivých softwarových struktur do třech základních vrstev: prezentační, aplikační a datové, viz podkapitola 1.2.2. Jedná se o český framework, proto je rozšířen a má velkou komunitu v Čechách. Jeho autorem je David Grudl a v současné době se o další rozvoj tohoto frameworku stará organizace Nette Foundation. [33, 34]

3.5.4 Doctrine

Doctrine je knihovna pro jazyk PHP, která se skládá z několika menších knihoven a je určená především pro práci s databázovým úložištěm a objektovým mapováním. [35, 36] Mezi hlavní výhody této knihovny patří především:

- nízká náročnost na konfiguraci;
- možnost automatického generování entit, jejich ověření a vytvoření nové databáze pomocí několika příkazů.
- podpora stromových struktur dat;
- nezávislost na konkrétní databázi;
- Doctrine Query Language (DQL) – dotazovací jazyk umožňující nezávislou práci s databází, který vychází z principů Hibernate Query Language (HQL) a Java Persistence Query Language (JPQL);
- definování uživatelských databázových typů;
- několikaúrovňové cachování.

3.5.5 Nettrine

Nettrine je knihovna, jejímž hlavním účelem je umělá a jednoduchá integrace Doctrine 2 do Nette frameworku. DBS portál v současnosti využívá ještě starší verzi Nette, proto jediná možná verze této knihovny, která je podporována, je verze 0.3. [37]

3.5.6 Codeception - PHP Testing framework

Codeception je víceúčelový testovací framework napsaný v jazyku PHP a určen pro testování aplikací. Tento framework je založen na známém PHPUnit frameworku. Hlavní jeho výhodou je možnost pracovat s různými typy testování. Codeception se standardně skládá ze tří sad (kolekcí): Unit Test, Acceptance Test a Functional Test. [38, 39]

Závěr

Mezi hlavní cíle této práce patřila analýza a návrh backendu nového testového modulu v DBS portálu. Tento návrh měl za úkol snížit provázanost systému, zlepšit ukládaní dat a usnadnit přidávání budoucích rozšíření. Na základě tohoto řešení měl být dále implementován prototyp testového modulu a realizováno jeho základní testování. Tyto cíle byly splněny.

Nejprve byla provedena podrobná analýza systému pomocí případů užití, kde byly zjištěny jeho hlavní funkce. Dále byly hodnoceny veškeré nedostatky a problémy týkající se uložení dat a současné implementace. Jelikož jsem se podílel na vývoji tohoto systému v rámci předmětů BI-SP1 a BI-SP2, analýza struktury databáze a současné implementace začala ještě tehdy. Bylo to spojené s úkolem, jehož výsledkem bylo přidání nového druhu testu. Na základě tohoto úkolu jsem objevil hlavní nevýhody současného stavu databáze a testového modulu, které jsou podrobně popsané v podkapitole 1.2. Součástí této kapitoly je také definice funkcí, které by měl nový testový modul splňovat.

Následně se tato práce zabývala modifikací a návrhem struktury databáze, který byl vytvořen v souladu s nově kladenými požadavky a nabízí prostor pro budoucí rozšíření. Na základě této struktury byl postaven vhodný návrh systému, který byl rozdělen do třech hlavních částí (vrstev):

- Presentery – část zodpovědná za přijetí a odesílání dat na frontend (komunikace mezi komponenty, nebyla implementována).
- Logika aplikace – obsažená veškerá logika aplikace, presentery s ní komunikují pomocí služeb.
- Datová vrstva – zodpovědná za uložení a selekci dat. Obsahuje entity, reprezentující databázové tabulky a další pomocné třídy zabývající se vyhledáváním dat a jejich ukládáním.

Vrstva, která reprezentuje logiku aplikace je navíc rozdělena na menší samostatné nezávislé části. Toto rozdělení je velmi důležité, protože snižuje provázanost systému a každá z těchto částí odpovídá za nějakou určitou funkci. Toto umožňuje jednoduše měnit chování nějaké komponenty testového mo-

dulu, bez zásahu do funkcí, které s tím vůbec nesouvisí. Proto by výsledek této práce měl výrazně snížit složitost implementace budoucích změn a rozšíření.

Bohužel v současnosti ještě neexistuje implementace frontendu pro testový modul v DBS portálu. Návrh a implementace této části je naplánována na příští běh předmětu BI-SP1 a BI-SP2. Z tohoto důvodu implementovaný prototyp podporuje pouze základní možnosti, jelikož se ještě úplně neví, jaké funkce bude nabízet nový frontend. Přesto věřím, že po dokončení jeho implementace, tento prototyp bude možné velice snadno dopracovat, aby splňoval veškeré požadavky.

Bibliografie

1. SÝKORA, Jan. *Podpora automatizované kontroly semestrální práce z předmětu Databázové systémy*. Praha, 2015. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
2. KUBIŠ, Martin. *Překladač z relační algebry do SQL*. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
3. GLAZAR, Filip. *Systém pro podporu BI-DBS - semestrální práce*. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
4. SLAVOTÍNEK, Jiří. *Webová komponenta na kreslení ER diagramů*. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
5. MALEC, Oldřich. *Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
6. KOVÁŘ, Pavel. *Automatizované testování webového portálu dbs.fit.cvut.cz*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
7. ERBEN, Marek. *Automatické generování a oprava otázek na normalizaci databáze pro předmět BI-DBS*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.
8. MACHALA, Filip. *Automatická oprava zjednodušeného relačního zápisu*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.

9. FEDOR, Bc. Tomáš. *ER diagrams web component II*. Praha, 2017. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.

10. PEJŠA, Petr. *Systém pro podporu testování v BI-DBS*. Praha, 2016. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií. Vedoucí práce Ing. Jiří HUNKA.

11. FOWLER, Martin. *Refactoring: Improving the Design of Existing Code*. 2nd ed. United States: Pearson Education (US), 2019.

12. ČÁPKA, David. Lekce 2 - UML - Use Case Diagram. In: *Ajťácká sociální síť a materiálová základna pro C, Java, PHP, HTML, CSS, JavaScript a další.* [online] [cit. 2020-03-23]. Dostupné z: `https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram`.

13. PostgreSQL. In: *https://postgres.cz* [online] [cit. 2020-04-14]. Dostupné z: `https://postgres.cz/wiki/PostgreSQL`.

14. PostgreSQL: About. In: *www.postgresql.org* [online]. © 1996-2020 [cit. 2020-04-14]. Dostupné z: `https://www.postgresql.org/about/`.

15. Relační databáze. In: *https://cs.wikipedia.org* [online] [cit. 2020-05-01]. Dostupné z: `https://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze`.

16. Lekce 3 - Třívrstvá architektura a další vícevrstvé architektury. In: *https://www.itnetwork.cz* [online] [cit. 2020-04-15]. Dostupné z: `https://www.itnetwork.cz/navrh/architektury-a-dependency-injection/trivrstva-architektura-a-dalsi-vicevrstve-architektury`.

17. Ukázka komunikace mezi vrstvy Model-View-Presenter. In: *https://ru.wikipedia.org* [online] [cit. 2020-05-17]. Dostupné z: `https://ru.wikipedia.org/wiki/Model-View-Presenter#/media/%D0%A4%D0%B0%D0%B9%D0%BB:MVP-Pattern.png`.

18. Lekce 2 - Monolitická a dvouvrstvá architektura. In: *https://www.itnetwork.cz* [online] [cit. 2020-04-15]. Dostupné z: `https://www.itnetwork.cz/navrh/architektury-a-dependency-injection/monoliticka-a-douvrstva-architektura`.

19. Principy objektově orientovaného návrhu. In: *https://www.zdrojak.cz/* [online] [cit. 2020-04-16]. Dostupné z: `https://www.zdrojak.cz/serialy/principy-objektove-orientovaneho-navrhu/`.

20. HLAVATÝ, Martin. *Requirements Engineering: Sběr a analýzapožadavků* [online]. 2019 [cit. 2020-04-18]. Dostupné z: `https://moodle-vyuka.cvut.cz/pluginfile.php/171216/course/section/28533/3_RequirementsEngineering.pdf` [Soubor přístupný po přihlášení do sítě ČVUT – kopie souboru uložena na přiloženém mediu].

21. Software Engineering | Classification of Software Requirements. In: *https://www.geeksforgeeks.org* [online] [cit. 2020-04-18]. Dostupné z: `https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/?ref=rp`.

22. Factory Method Pattern. In: *https://refactoring.guru/* [online] [cit. 2020-05-07]. Dostupné z: `https://refactoring.guru/design-patterns/factory-method`.

23. Working with JSON. In: *https://developer.mozilla.org* [online] [cit. 2020-04-24]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON`.

24. What is REST? | Codecademy. In: *https://www.codecademy.com* [online] [cit. 2020-04-24]. Dostupné z: `https://www.codecademy.com/articles/what-is-rest`.

25. State Pattern. In: *https://refactoring.guru/* [online] [cit. 2020-05-07]. Dostupné z: `https://refactoring.guru/design-patterns/state`.

26. Adapter Pattern. In: *https://refactoring.guru/* [online] [cit. 2020-05-07]. Dostupné z: `https://refactoring.guru/design-patterns/adapter`.

27. How to use Doctrine ORM with Nette Framework. In: *https://blog.nette.org* [online] [cit. 2020-05-17]. Dostupné z: `https://blog.nette.org/en/how-to-use-doctrine-orm-with-nette-framework`.

28. DQL: Doctrine Query Language. In: *https://www.doctrine-project.org* [online] [cit. 2020-05-17]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine1/en/latest/manual/dql-doctrine-query-language.html`.

29. Custom Mapping Types. In: *https://www.doctrine-project.org* [online] [cit. 2020-05-16]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/cookbook/custom-mapping-types.html`.

30. Druhy testování v procesu vývoje SW. In: *http://test.swtestovani.cz* [online] [cit. 2020-05-20]. Dostupné z: `http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11`.

31. PHP: Hypertext Preprocessor. In: *https://www.php.net/* [online] [cit. 2020-05-13]. Dostupné z: `https://www.php.net/`.

32. Co je to PHP a k čemu mi bude dobré? In: *https://www.coreit.cz* [online] [cit. 2020-05-13]. Dostupné z: `https://www.coreit.cz/php/`.

33. Nette – Pohodlný a bezpečný vývoj webových aplikací v PHP. In: *https://nette.org/* [online] [cit. 2020-05-16]. Dostupné z: `https://nette.org/cs/`.

34. Nette Framework. In: *https://cs.wikipedia.org* [online] [cit. 2020-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/Nette_Framework#Vlastnosti_Nette_Framework.

35. Doctrine: PHP Open Source Project. In: *https://www.doctrine-project.org* [online] [cit. 2020-05-16]. Dostupné z: https://www.doctrine-project.org/index.html.

36. Doctrine (PHP). In: *https://cs.wikipedia.org* [online] [cit. 2020-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/Doctrine_(PHP).

37. Nettrine ORM. In: *https://github.com/nettrine/orm* [online] [cit. 2020-05-17]. Dostupné z: https://github.com/nettrine/orm/blob/master/.docs/README.md#setup.

38. Introduction to Codeception. In: *https://medium.com* [online] [cit. 2020-05-20]. Dostupné z: https://medium.com/tech-tajawal/introduction-to-codeception-e18503136ba8.

39. Codeception Documentation. In: *https://codeception.com/* [online] [cit. 2020-05-20]. Dostupné z: https://codeception.com/docs/01-Introduction.

Příloha A

Seznam použitých zkratek

BI-DBS Databázové systémy

BI-SP1 Softwarový týmový projekt 1

BI-SP2 Softwarový týmový projekt 2

DBS portál Systém pro podporu výuky předmětu BI-DBS

DI Dependency Injection

DQL Doctrine Query Language

HQL Hibernate Query Language

HTML Hypertext Markup Language

JIT kompilace Just-in-time compilation

JPQL Java Persistence Query Language

JS JavaScript

JSON JavaScript Object Notation

LS Letní semestr

MVC Model-View-Controler

MVP Model-View-Presenter

NEON Nette Object Notation

ORM Object Relation Mapping

PHP PHP: Hypertext Preprocessor

RA Relační algebra

REST Representational State Transfer

SQL Stands for Structured Query Language

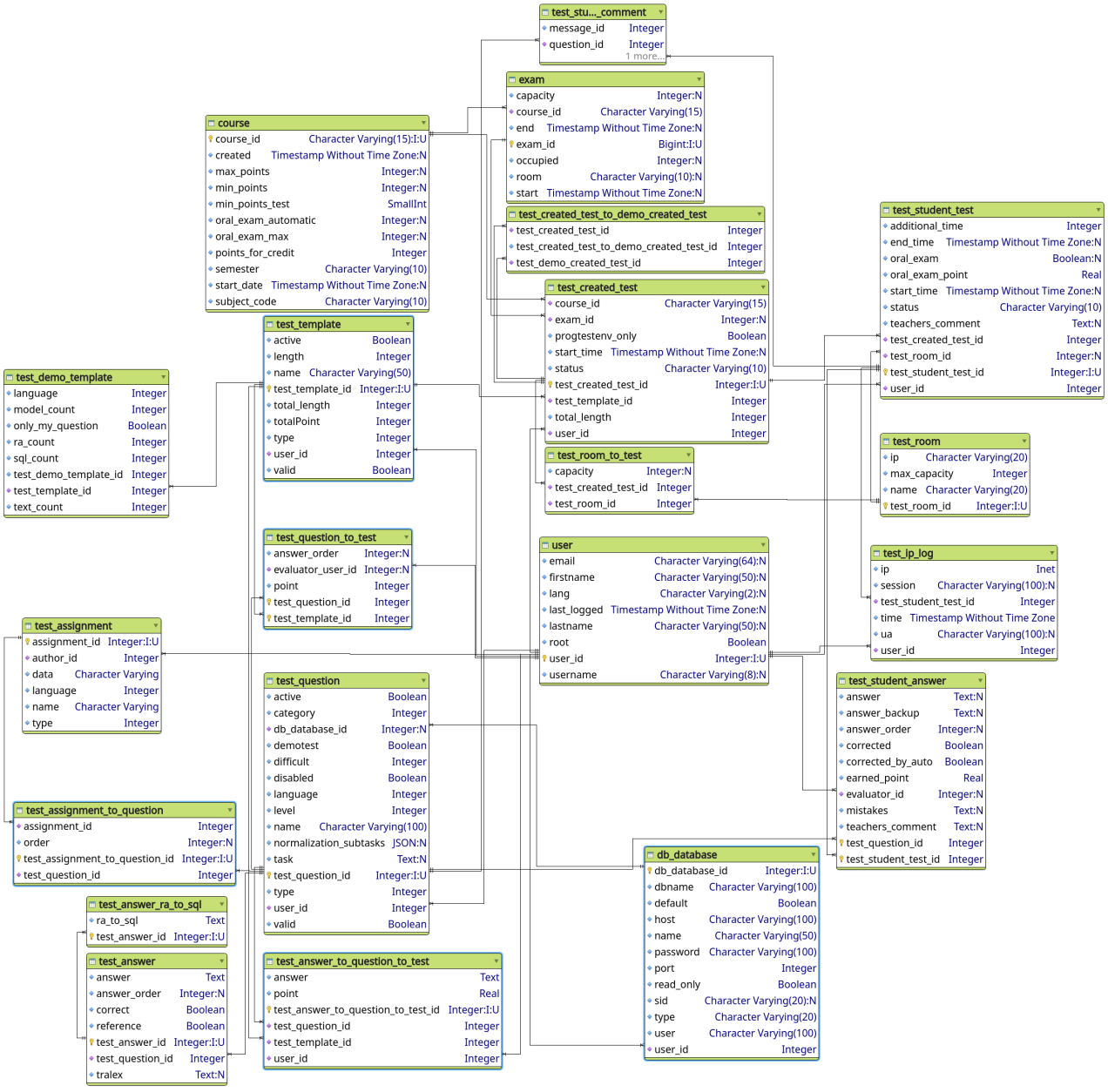
Systém KOS Studijní informační systém

UC Use case diagram

UML Unified Modeling Language

Příloha B

Současný stav databáze



Obrázek B.1: Schéma databáze: Současný stav

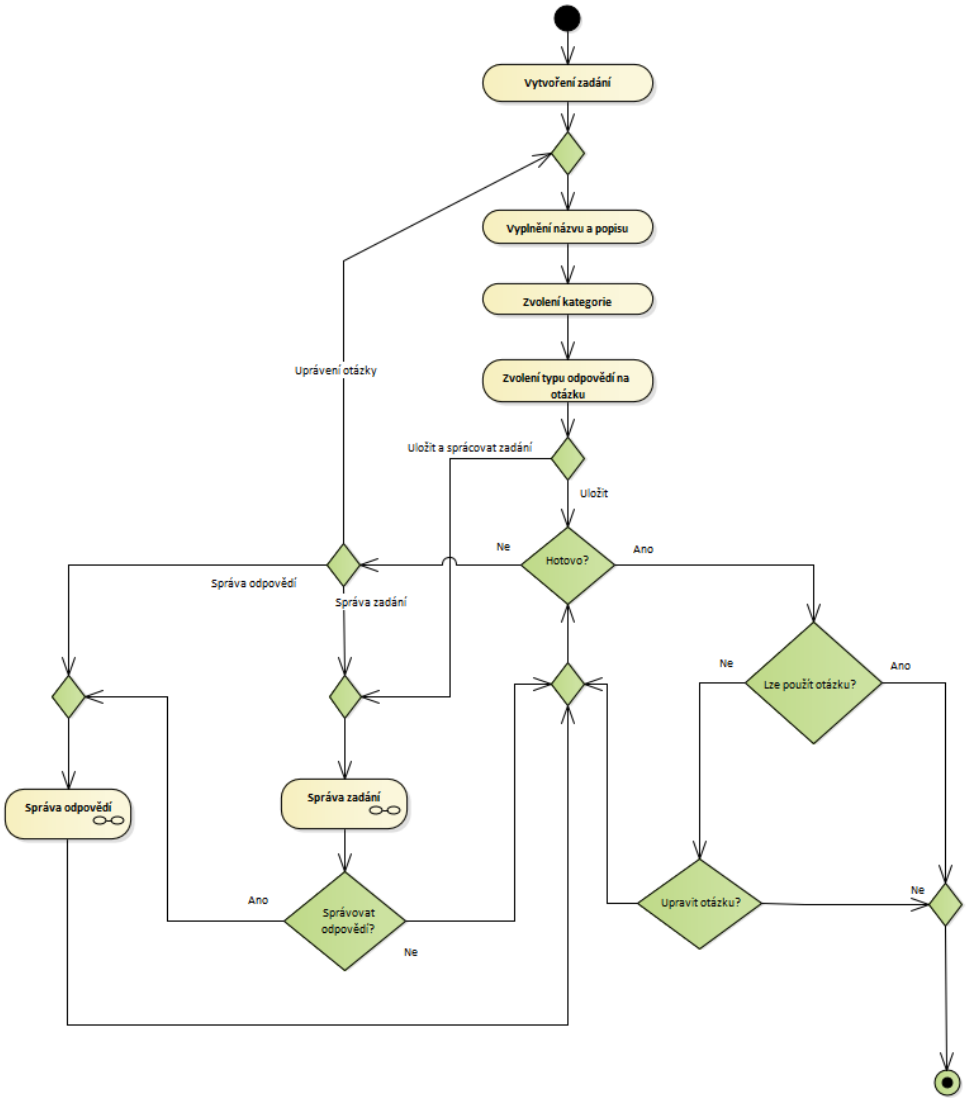
Příloha C

Budoucí stav databáze

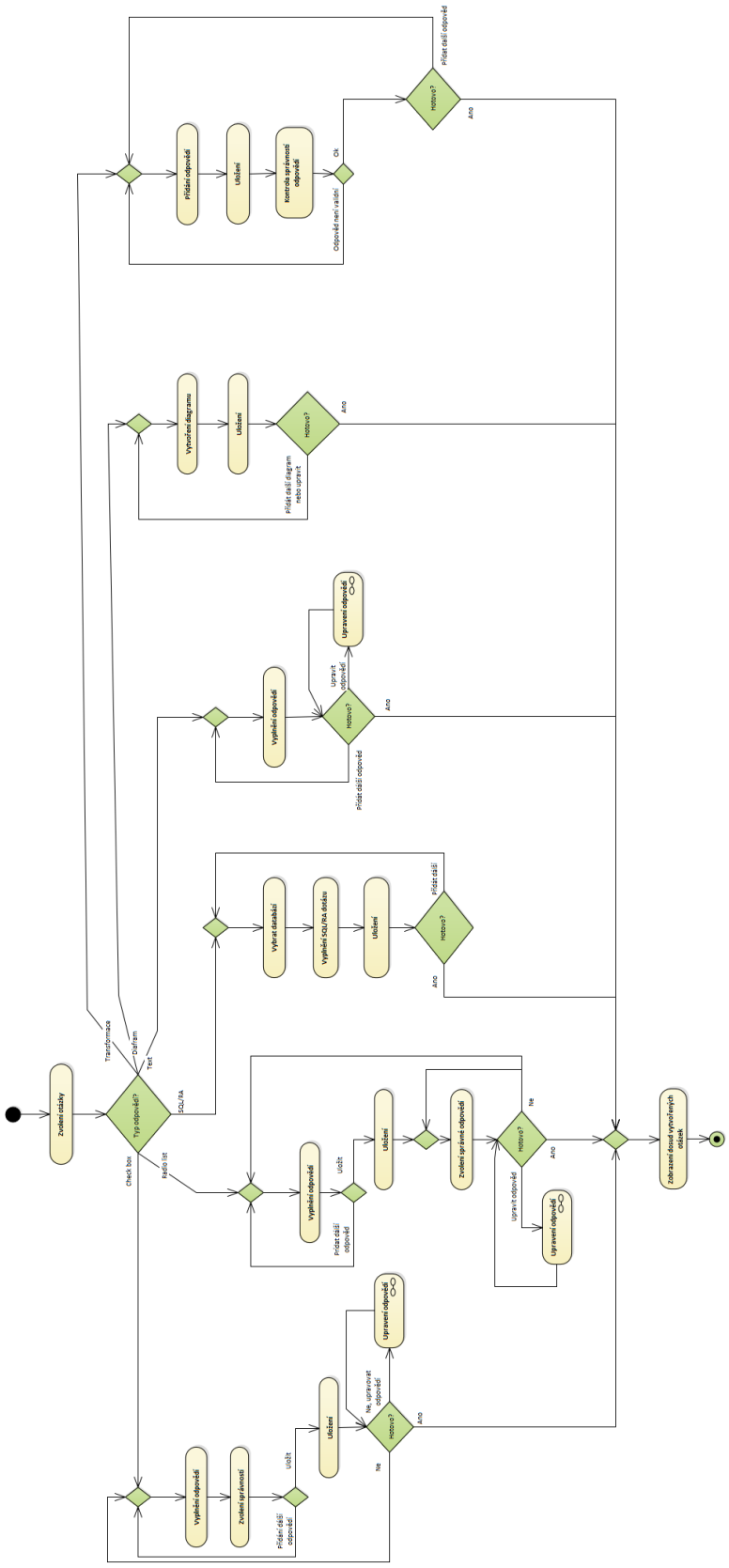
Obrázek C.1: Schéma databáze: Budoucí stav

Příloha D

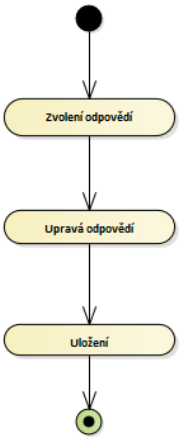
Diagramy aktivit



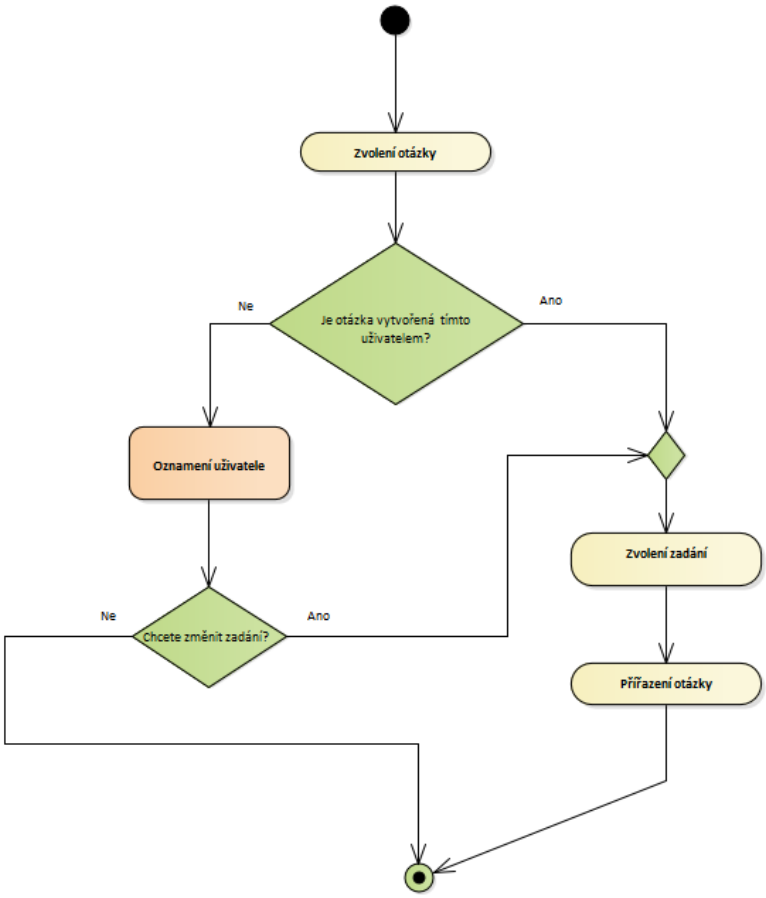
Obrázek D.1: Diagram aktivit: Vytvoření otázky



Obrázek D.2: Diagram aktivit: Správa odpovědí



Obrázek D.3: Diagram aktivit: Upravení odpovědí



Obrázek D.4: Diagram aktivit: Správa zadání

Příloha E

Ukázka metod současného řešení

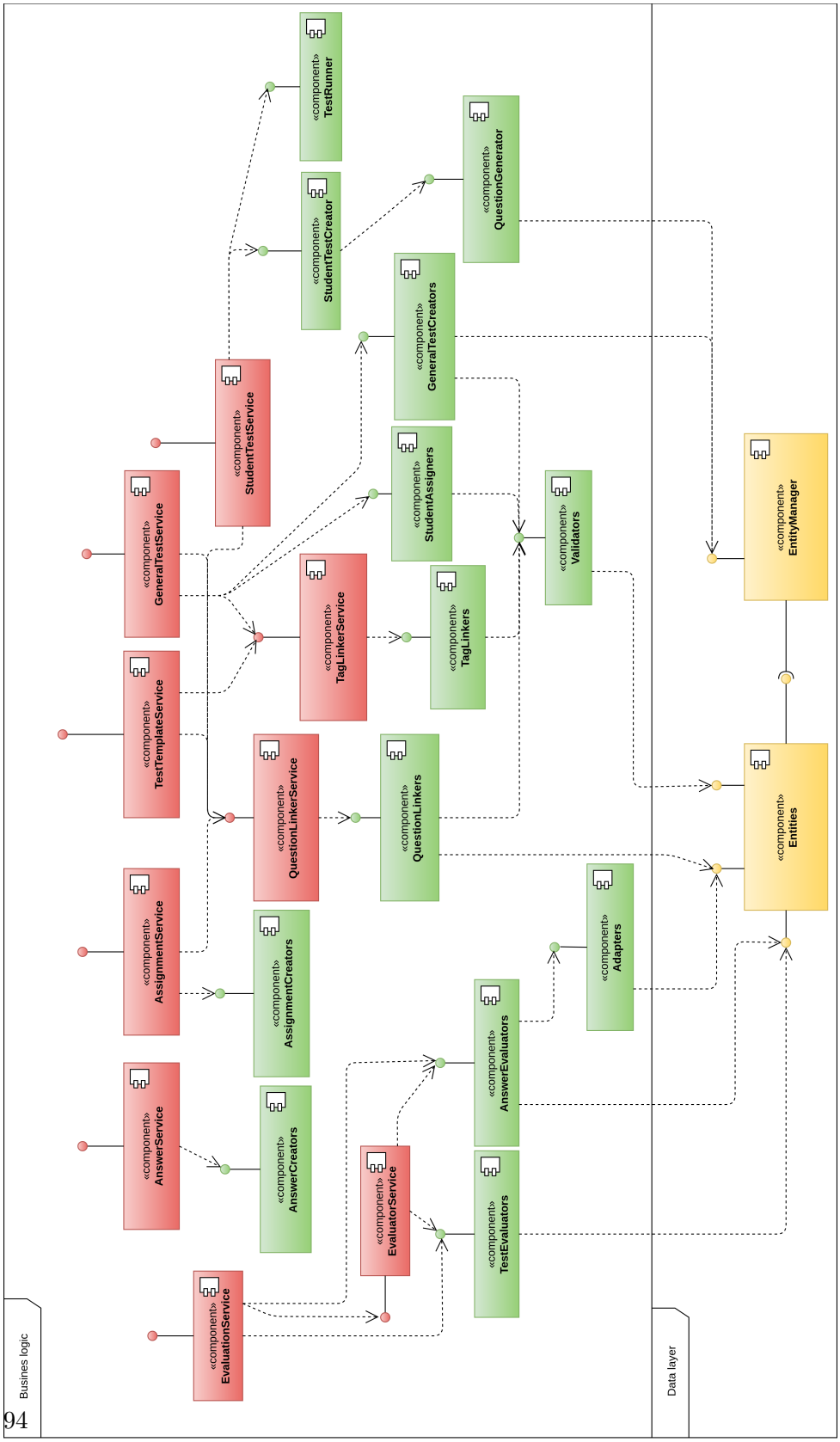
```php
public function handleStartTest(int $student_test_id): void
{
    $test_created_test_id = $this->createdTestModel->getCreatedTestByStudentTest($student_test_id)['test_created_test_id'];
    $testTemplate = $this->templateModel->getTemplateByStudentTest($student_test_id);                    // if type is auto demo test than create
    if ($testTemplate['type'] === TestType::AUTO_DEMO) {                                                 // a new template and a new test and start it
        $new_student_test_id = $this->autoDemoTestModel->generateAutoDemoTest($testTemplate, $test_created_test_id, $this->user->getId());
        if ($new_student_test_id != null) {
            $student_test_id = $new_student_test_id;
        }
    }

    $progtestEnvOnly = $this->createdTestModel->getCreatedTestByStudentTest($student_test_id)->progtestenv_only;
    $test = $this->studentTestModel->getStudentTestById($student_test_id);
    if ($test) {
        if (!$this->user->isInRole( role: Role::TEST_STUDENT) && $progtestEnvOnly && !$this->environment->isInProgtestEnv()) {
            //if set, allow non-test users to activate test only from progtest environment
            $this->flashMessage( message: 'test.test.message.notAllowed', type: 'danger');
            $this->redirect( code: ':Test:Test:default');
        } else if ($test->status == 'activated') {
            $dateTime = new Nette\Utils\DateTime();
            $length = $this->templateModel->getTemplateByStudentTest($student_test_id)['length'];
            $expectedEndTime = $dateTime->modifyClone( modify: '+ ' . $length . ' minutes');
            $this->studentTestModel->updateStudentTest($student_test_id, ['status' => 'running', 'start_time' => $dateTime, 'end_time' => $expectedEndTime]);
            $this->redirect( code: ':Test:Test:test', $student_test_id);
        } else if ($test->status == 'running') {
            if (!$this->isStudentAllowedToTest($this->user->id, $student_test_id)) {
                $this->flashMessage( message: 'test.test.message.notAllowed', type: 'danger');
                $this->redirect( code: ':Test:Test:default');
            }
            $this->redirect( code: ':Test:Test:test', $student_test_id);
        } else {
            $this->flashMessage( message: 'test.test.message.notAllowed', type: 'danger');
            $this->redirect( code: ':Test:Test:default');
        }
    }
}
```

Obrázek E.1: Současný stav: Ukázka metody spouštějící test

Příloha F

Diagram komponent



Obrázek F.1: Diagram komponent: Budoucí stav

Příloha G

Obsah přiloženého paměťového média

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `project` ..... zdrojové kódy DBS portálu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `create_script.sql` ..... SQL skript umožňující tvorbu nové databáze testového modulu
- `text` ..... text práce
  - `BP_Plyskach_Andrii_2020.pdf` ..... text práce ve formátu PDF
- `images` ..... složka s obrázky a diagramy
  - `Activity_Diagrams` ..... diagramy aktivit
  - `Class_Diagrams` ..... diagramy tříd
  - `DB_Diagrams` ..... databázové modely
  - `Use_Case_Diagrams` ..... diagramy případů užití
  - `Component_Diagrams` ..... diagramy komponent
  - `images` ..... obrázky