FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	dbs.fit.cvut.cz - Refaktoring testů I
Student:	Martin Hanzl
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem této práce je úprava a přepis části testů v portálu dbs.fit.cvut.cz, dále jen portál, s ohledem na zobrazování otázek a jejich ohodnocení studentům a vyučujícím. Důvodem je aktuální nespokojenost studentů s informací, kde v testu pochybili. Dále má tato práce navázat na práci Bc. Andriiho Plyskache, který připravil nový backend testů.

Postupujte v těchto krocích:
Analyzujte současný stav vyhodnocení a zobrazení otázek z testů u studentů a vyučujících v portálu.
Na základě analýzy navrhněte lepší možnosti vyhodnocení a následného zobrazení tak, abyste minimalizovali nárůst složitosti pro vyučující a zároveň zachovali nebo zlepšili edukativní dopad zobrazení pro studenty. Zamyslete se i nad možností chytrého zobrazení vzorových řešení.
Na základě analýzy implementujte funkční prototyp.
Realizujte vhodné sady testů a ověřte použitelnost Vašeho řešení.
Integrujte Vaše řešení po odladění prototypu do portálu pro budoucí možné použití.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 19. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

dbs.fit.cvut.cz - Refaktoring testů I

Martin Hanzl

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Hunka

24. června 2021

Poděkování

Rád bych poděkoval svému vedoucímu, Ing. Jiřímu Hunkovi za veškorou poskytnutou zpětnou vazbu, která pomohla zlepšit tuto práci. Dále bych rád poděkoval Ing. Oldřichu Malcovi a Ing. Pavlovi Kovářovi za jejich rady pro řešení technických potíží. Rád bych poděkoval Andriimu Plyskach za zodpovězení otázek k jeho práci. Chtěl bych poděkovat i vývojovému týmu z předmětu BI-SP1 za jejich spolupráci. V neposlední řadě bych rád poděkoval své mamince za podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 24. června 2021 . . . . . .

Odkaz na tuto práci

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

Abstrakt

Tato bakalářská práce se zaobírá úpravou a přepisem části testů v portálu dbs.fit.cvut.cz. Konkrétně se zaměřuje na zlepšení zpětné vazby při zobrazování výsledků testů studentům. Práce začíná analýzou současného řešení s ohledem na změny provedené v nově vznikajícím testovém modulu, na který tato práce navazuje. Analýza pokrývá případy užití, strukturu testů a algoritmy automaticky vyhodnocující otázky v portálu. Na základě této analýzy je proveden návrh nového zobrazování pomocí drátěného modelu. Následně je popsána tvorba prototypu od převzetí projektu po jeho integraci.

Klíčová slova webový portál, portál DBS, databáze, podpora výuky, Případy užití, Drátěný model, SQL, PHP, Nette, framework

Abstract

This bachelor thesis deals with the modification and rewriting of part of the tests in the portal dbs.fit.cvut.cz. Specifically, it focuses on improving feedback when displaying test results to students. The work begins with an analysis of the current solution with respect to the changes made in the newly emerging test module, which this work builds on. The analysis covers use cases, test structure and algorithms that automatically evaluate questions in the portal. Based on this analysis, a new frontend solution is designed using a wireframe model. Lastly, the creation of a prototype from the takeover of the project to its integration is described.

Keywords web portal, portal DBS, database, teaching support, Use case, Wireframe, SQL, PHP, Nette, framework

Obsah

Seznam obrázků

Úvod

Práce se zabývá úpravou a přepisem části testového modulu v portálu dbs.fit.cvut.cz, dále jen portál. Práce navazuje na nově vzniklé backendové řešení, jehož základ položil Andrii Plyskach ve své bakalářské práci [2]. V té popsal současné nedostatky systému související s budoucím rozvojem portálu a navrhl nové databázové schéma. Při tvorbě backendu využil technologii ORM (Objected Related Mapping).

Důvodem této práce je aktuální nespokojenost studentů s informací, kde v testu pochybili. Současné řešení studentům nabízí náhled na absolvované testy, ale toto řešení má jisté nedostatky. Také je důvodem potřeba tvorby frontendu pro nový backend. Starý frontend nelze použít, neboť došlo ke změně databázového schématu a bylo kompletně předěláno třídní schéma, aby lépe odpovídalo potřebám pro budoucí rozvoj.

Portál slouží jako podpora nejen online výuky v předmětu BI-DBS a jiných na fakultě Informačních technologií, ČVUT. Pandemická situace v posledním roce ukázala, jak je kvalita online výuky důležitá. Proto vnímám zlepšení zpětné vazby pro studenty v portálu za důležitý přínos.

Konkrétně se tato práce zaobírá částí portálu, kde si studenti zobrazují výsledky svých testů a jak si počínali v řešení jednotlivých otázek. Pro pochopení této problematiky, bude ale třeba se seznámit i s částmi portálu, kde se otázky a testy tvoří/používají. Bude třeba brát v potaz i změny vzniklé při návrhu nového testového modulu.

Cílem práce je analýza současného řešení se zaměřením na otázky, testy a jejich vyhodnocování. K analýze bude využito prostředků softwarového inženýrství. Na základě poznatků z analýzy a sestavených požadavků bude navrženo nové frontendové řešení. Také bude navržen koncept realizace vyhodnocování otázek nad novým backendem. Dle návrhu bude realizován funkční prototyp frontendu, který bude následně otestován uživatelskými testy a následně integrován pro budoucí možné použití.

Kapitola 1

Analýza

V úvodu této kapitoly je seznámení s portálem. Dále následují případy užití situací, při kterých se aktéři portálu mohou setkat s otázkami a testy. Poznatky z případů užití jsou shrnuty v kapitole 1.3, která zahrnuje i změny nového backendového řešení. Následně je provedena analýza algortimů vyhodnocování otázek. Na základě analýzy jsou sestavy požadavky pro následný návrh.

1.1 portál DBS

Portál slouží zejména jako podpora výuky předmětu BI-DBS na fakultě Informačních technologií, ČVUT. S nápadem na zautomatizování výuky přišel Jiří Hunka v roce 2013 a garant předmětu Michal Valenta tento nápad schválil. Portál byl vyvýjen od roku 2014 z velké části studenty fakulty pod vedením Jiřího hunky. Prvně byl použit v letním semestru 2016 na menším vzorku studentů. [3] Dnes je již plnou součástí výuky předmětu BI-DBS ale i BIE-DBS a BIK-DBS. Také je dále rozvíjen studenty v rámci předmětů BI-SP1 a BI-SP2. S funkcemi portálu se budeme seznamovat v následujících kapitolách. Zde jen pro zmíňku uvedeme nejpodstatnější:

- Tvorba semestrální práce
- Vyplňování testů
- Zobrazení studentských výsledků
- Modelovací nástroj
- Připojení k databázi
- Administrace uživatelů

V rámci vývoje vznikla celá řada bakalářských prací na toto téma. Andrii Plyskach ve své bakalářské práce [2] vytvořil nový backend pro testový modul.

1.2 Případy užití

Případy užití se používají k zachycení funkcionalit celého systému. Poskytují abstraktní náhled na systém zejména z pohledu uživatelů. Zachycují aktéry a jejich aktivity, které v daném systému provádějí. Jsou tedy zejména užitečné na počátku analýzy, kdy je třeba zachytit co systém umí, případně má umět.

V Use Case diagramu podle UML by měli být na levé straně aktéři, kteří iniciují aktivity ve prostřed diagramu a jsou s nimi spojeny. Na pravé straně jsou naopak aktéři, kteří jsou ovlivněni aktivitami. V diagramu se může objevit i aktivita, která není iniciována žádným aktérem a žádného aktéra neovlivňuje, nicméně to naznačuje, že pravděpodobně analýza ještě nezahrnuje všechny elementy. Diagram nezachycuje žádné pořadí provádění aktivit.

Je vhodné diagramy případu užití doplnit o jejich popis. Takový popis by měl zachycovat aktéry, kteří figurují v aktivitě, počáteční podmínku pro započatí dané aktivity a scénář této aktivit. Aktivita může mít i více scénářů, je tedy vhodné je uvést například jako alternativní scénáře.[4]

Bude proveden tento druh analýzy nad vývojářskou verzí portálu DBS se zaměřením na otázky a testy. Ve vývojářské verzi je možný přístup ke všem rolím, bude tedy možné si vyzkoušet jaké je odpovídat na otázky jako student, ale také jak je vytvářet a opravovat jako vyučující, což je vhodné pro pochopení problematiky otázek. Analýzu testového modulu portálu provedl i Andrii Plyskach ve své bakalářské práci, kde se zaměřil i na funkce, které se netýkají otázek [2] .

1.2.1 Aktéři

Aktéry z DBS portálu zadefinujeme jako uživatele s rolí a právy k pravádění určitých činností. Aktérem může být osoba, skupina osob, externí systém nebo i čas. Pro zaměření této práce budou nejdůležitějšími aktéry student, vyučující a garant.

V portálu se vyskytují i další aktéři. Root je hlavní administrátor, který má právo na import dat z portálu KOS, má také právo přihlásit se jako jakýkoli jiný uživatel, tím jsou mu zpřístupněna všechna práva daného uživatele. Dalším aktérem by mohl být nepřihlášený uživatel, ten naopak moc činností v portálu provádět nemůže, je mu dostupná pouze možnost přihlásit se. Těmito aktéry se práce dále zabývat nebude.

1.2.1.1 Student

Aktéry student jsou v portálu studenti českého vysokého učení technického (ČVUT) - fakulty Informačních technologií, kteří se zapsali do předmětu BI-DBS, případně BIE-DBS, BIK-DBS.

V portálu tvoří svou semestrální práci, kterou v portálu i odevzdávají. Také píší v portálu semestrální test pro získání zápočtu a následně opět v portálu

skládají závěrečnou zkoušku. Aby se na tyto aktivity mohli připravit jsou jim zpřístupněny demotesty. Studenti si v portálu mohou prohlédnou zpětnou vazbu od systému nebo učitele a podívat na své známky, body ze semestru.

1.2.1.2 Vyučující

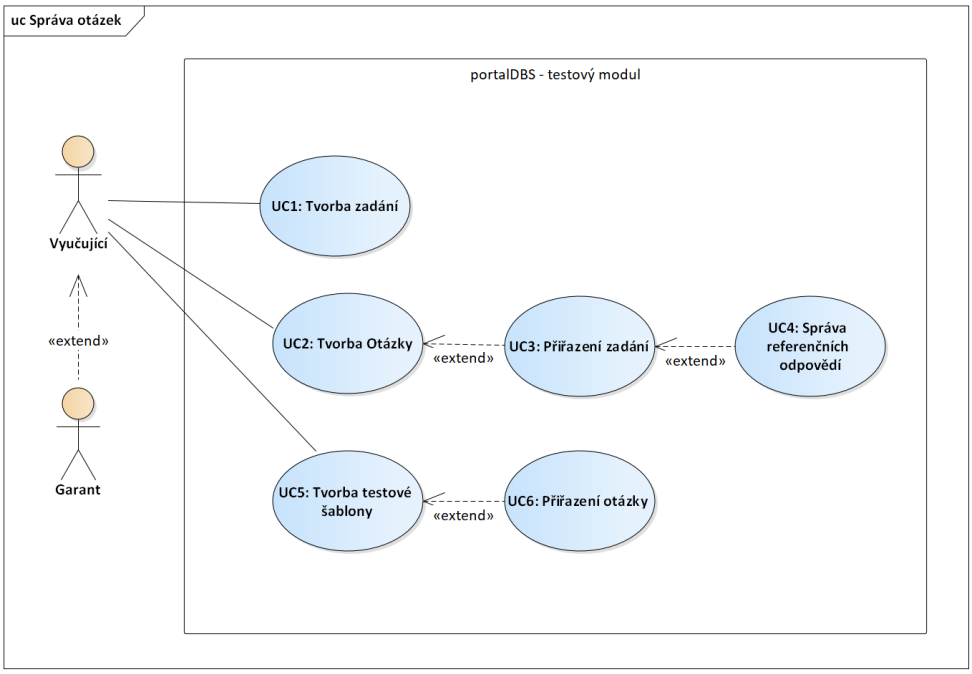
Vyučující mají v portálu možnost tvorby zadání, otázek a testů, které mohou následně spouštět pro studenty. Dale musí manuálně opravovat o tázky ze semestrálního testu a zkoušky, které nebylo možné vyhodnotit automatickou opravou. Také opravují iterační body semestrálních prací studentů. Vše plně v portálu. Pro vyučující jsou dostupné i různé statistiky, například úspěšnost studentů v testu apod. Tyto statistiky jim mohou sloužit například pro reflexi obtížnosti testu, který vytvořili.

1.2.1.3 Garant

Protože garant zodpovídá za běh předmětu, má stejná práva jako aktér vyučující s rozšířením o upravování podmínek klasifikace.

1.2.2 Správa otázek

Pro vhodné navrhnutí frontendu pro zobrazování výsledků testů studentům bude vhodné se napřed seznámit s tím jak otázky fungují od vzniku po opravu. Tato kapitola se zaměří na pohled vyučijících a garanta na otázky. Zejména na tvorbu zadání, otázek a testů, na jejich vzájemné propojení. Cílem této kapitoly je seznámit se se strukturou a kompozicí otázek. Analýza některých aktivit, která by již nepřinesla žádné nové informace bude vynechána. Často bude zmiňováno, že zadání, otázky i testy mají více typů. Podrobný popis těchto typů bude proveden v kapitole 1.3. Zde je cílem zachytit aktivity.



Obrázek 1.1: Diagram případu užití: Správa otázek

1.2.2.1 UC1: Tvorba zadání

Zadání musí být součástí každé validní otázky, pouze validní otázky mohou být součástí testů, které vyplňují studenti. Následující scénář se věnuje popisu tvorby zadání vyučujícím, případně garantem.

Aktéři: Vyučující, Garant
Scénář:

1. Aktér si v testovém modulu porálu zvolí v postraní liště položku *Tvorba zadání*.
2. Systém tubo položku rozbalí v rámci postraního panelu a nabídne aktérovi jaký typ zadání chce vytvořit.
3. Aktér si zvolí typ.
4. Systém zobrazí aktérovi formulář pro vyplnění s daty o zadání, formulář se liší dle typu zadání. Položka *Název* je společná.
5. Aktér vyplní formulář a klikne na tlačítko *Uložit*.
6. Systém uloží zadání do databáze. Zadání je nyní dostupné ve správě zadání a v přiřazování zadání k otázce.

1.2.2.2 UC2: Tvorba otázky

Tento případ užití zachycuje prvotní tvorbu otázky. Jedná se pouze o nastavení jejích zakladních vlastností. Proto aby otázka byla validní, je třeba jí přiřadit zadání v UC3 a alespoň jednu referenční odpověď v UC4.

Aktéři: Vyučující, Garant
Scénář:

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

2. Systém zobrází aktérovi zákaldní formulář pro otázku. Položky tohoto formuláře jsou *Název, Typ otázky, Jazyk, Úkol, Vhodnost pro demotest.*

3. Aktér vyplní formulář a otázku uloží a vrátí se na předchozí stránku nebo pokračuje dále k přiřazení zadání (UC3).

4. Ať už se uživatel rozhodl jakkoli, otázka je uložena v databázi a je dostupna ve správě otázek, nicméně není považována za validní.

1.2.2.3 UC3: Přiřazení zadání

Aktivita popisuje, jak vyučující nebo garant přiřazuje nebo odebírá již vytvořené zadání k otázce. Jsou zachyceny i další možnosti, které aktér má v tomto náhledu.

Aktéři: Vyučující, Garant
Scénář:

1. Aktér se do této aktivity dostane přímo z tvorby otázky nebo přes editaci otázky.

2. Systém zobrazí vícestránkovou tabulku se všemi zadáními v databázi. Zadání již přidaná k této otázce jsou zobrazena na první stránce a jsou zvýrazněna.

3. Aktér má možnost si zadání prohlédnout, v tomto případě mu systém poskytne náhled zadání. Další možností aktéra je přiřadit vybrané zadání k otázce, nebo již přiřazené zadání odebrat.

4. Systém vždy po vykonání akce aktérem aktualizuje tabulku se zadáními. Aktérovi změny jsou také ihned uloženy v databázi.

5. Aktér z tohoto bodu může pokračovat na Správu referenčních odpovědí otázky (UC4) nebo se vrátit na předchozí stránku.

1.2.2.4 UC4: Správa referenčních odpovědí

Aby systém mohl automaticky vyhodnocovat studenské testy, je často nutné mít referenční řešení. V tomto procesu aktér s právy alespoň vyučujícího spravuje referenční řešení vybrané otázky.

Aktéři: Vyučující, Garant
Scénář:

1. Aktér se do této aktivity dostane z náhledu s přiřazováním zadání (UC3).

2. Systém aktérovi zobrazí přehled již přidaných referenčních odpovědí. Také formulář pro přidání nové referenční odpovědi (formulář je dle typu otázky). Poslední zobrazenou komponentou je formulář, kde aktér volí kolik má být minimálně zobrazeno správných odpovědí za předpokladu, že se jedná o uzavřenou otázku, kde student pouze odpovědi vybírá z předem zobrazených.

3. Aktér může klasicky přidávat, odebírat nové referenční odpovědi. Nastavovat počet minimálních správných a zobrazovat si náhled již přidaných referenčních odpovědí. Své změni pak uloží tlačítkem „Uložit“.

4. Změny jsou zapsány do databáze po uložení.

1.2.2.5 UC5: Tvorba testové šablony

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

Aktéři: Vyučující, Garant
Scénář:

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

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

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

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

1.2.2.6 UC6: Přiřazení otázky

Při popisu tvorby testové šablony bylo zmíněno, že je možnost do ní manuálně přidat otázky. Zde se na tento případ užití podíváme blíže.

Aktéři: Vyučující, Garant

Scénář:

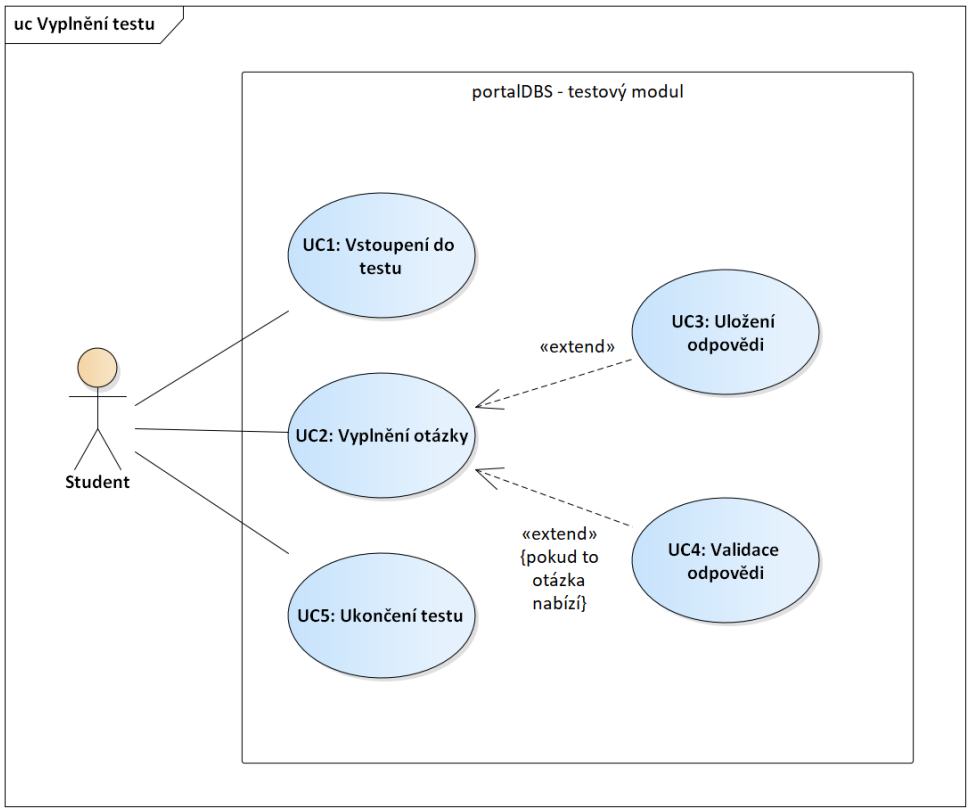
1. Systém aktérovi zobrazí vícestránkovou tabulku s otázkami. Na začátku tabulky jsou otázky již přidané do této testové šablony, jsou také zvýrazněny zeleně. Žlutě jsou označeny otázky, které byly automatickou kontrolou označeny za nevalidní a pravděpodobně tak nepůjdou do testu přidat.Červeně jsou označeny otázky přiřazené v této testové šabloně, ale automatickou kontrolou byly označeny za nevalidní.

2. Aktér má u každé otázky možnost přidat nebo odebrat podle toho zda je již přidaná. Může si také zobrazit náhled otázky. Akce přidání otázky vyvolá formulář, kde aktér napřed musí vyplnit, kdo z vyučujících je zodpovědný za manuální opravu daté otázky v případě, že se nepovede otázku opravit automaticky. Také vyplní kolika body má být otázka maximálně hodnocena v tomto testu.

3. Změny provedené v testové šabloně jsou ihned uloženy sestémem do databáze, když aktér provede nějakou akci.

1.2.3 Vyplnění testu

Tento případ užití se zaměřuje na proces vyplnění testu studentem. Proces bude zachycen od vstupování do testu po jeho manuální, případně automatické ukončení. Budou zachyceny i důležité, menší aktivity při vyplňování testu.



Obrázek 1.2: Diagram případu užití: Vyplnění testu

1.2.3.1 UC1: Vstoupení do testu

Student touto akcí zahajuje vyplňování testu. Začíná mu oficiálně běžět čas na vyplnění a nemůže po dobu této aktivity vyplňovat jiný test.

Aktéři: student

Počáteční podmínka: Student má oprávnění vstoupit do testu. Podmínky vstoupení jsou následující:

- Student je přiřazen do testu.
- Pokud je u testu nastaven požadavek na progtestový image, student se musí do testu připojit ze zařízení s progtestovým imagem.
- Student se nepokouší vstoupit do demotestu, pokud běží semestrální test nebo zkouška.

Scénář:

1. Student se nachází v testovém modulu portálu a v postranní liště si zvolí položku reprezentující test, který se chystá vyplnit. Z nabídky je test (semestrální test, zkouška) nebo demotest.
2. Systém studentovi zobrazí pouze testy do kterých je student přiřazen.
3. Student akcí *Spustit test* vstoupí do testu.

1.2.3.2 UC2: Vyplnění otázky

Tento případ užití popisuje zodpovězení testové otázky studentem. Předpokladem je, že nastal scénář z případu UC1.

Aktéři: student

Počáteční podmínka: Čas na vypracování testu ještě nevypršel.
Hlavní scénář: První vyplňení otázky

1. Student ze seznamu otázek zvolí, kterou otázku chce vyplňovat.
2. Systém studentovi zobrazí potřebné informace, aby otázku mohl zodpovědět(název, úkol, zadání).
3. Systém také zobrazí formulář pro vyplnění odpovědi. Položky formuláře jsou v tomto scénáři prázdné.
4. Student vyplní formulář odpovědi.

Alternativní scénář: Vyplnění již zodpovězené otázky

1. Student znovu zvolí vyplnění otázky, kterou již zodpověděl.
2. Systém studentovi zobrazí potřebné informace, aby otázku mohl zodpovědět(název, popis, zadání).
3. Systém také zobrazí formulář pro vyplnění odpovědi. Pokud student při zodpovězení této otázky použil funkcionalitu „Uložit odpověď“ (UC3), položky formuláře jsou v tomto scénáři předvyplněné studentovou poslední odpovědí.
4. Student vyplní formulář odpovědi.

1.2.3.3 UC3: Uložení odpovědi

Jedná se o krátký případ užití, nicméně je velmi podstatný pro správné zpracování studentova testu. Student svou odpověď musí manuálně uložit, jinak by se mohlo stát, že jeho odpověď nebude zaznamenána.

Aktéři: student

Počáteční podmínka: Čas na vypracování testu ještě nevypršel.

Scénář:

1. Student se nachází při vyplňování otázky (UC2) a klikne na tlačítko „Uložit odpověď“.

2. Systém uloží formulář se studentovými daty.

3. Systém studentovi zobrazí zprávu, že jeho odpověď byla úspěšně uložena.

1.2.3.4 UC4: Validace odpovědi

Tento případ užití má za účel pomoct studentovi, zda je jeho odpověď z hlediska syntaktické stránky validní a dává smysl vůči kontextu otázky, nicméně to neříká nic o správnosti jeho odpovědi.

Aktéři: student

Scénář:

1. Student se nachází při vyplňování otázky (UC2) a klikne na tlačítko „Ověřit odpověď“.

2. Systém provede kontrolu.

3. Systém studentovi zobrazí zprávu o validitě jeho otázky.

1.2.3.5 UC5: Ukončení testu

Touto funkcí může student ukončit test předčasně. Učiní tak například, když je přesvědčen, že má všechny odpovědi zodpovězeny správně. Pokud tuto funkcionalitu student nevyužije, test se ukončí automaticky při vypršení časového limitu.

Aktéři: student

Počáteční podmínka: Čas na vypracování testu ještě nevypršel a nastal scénář z UC1, tedy student vstoupil do testu.

Scénář:

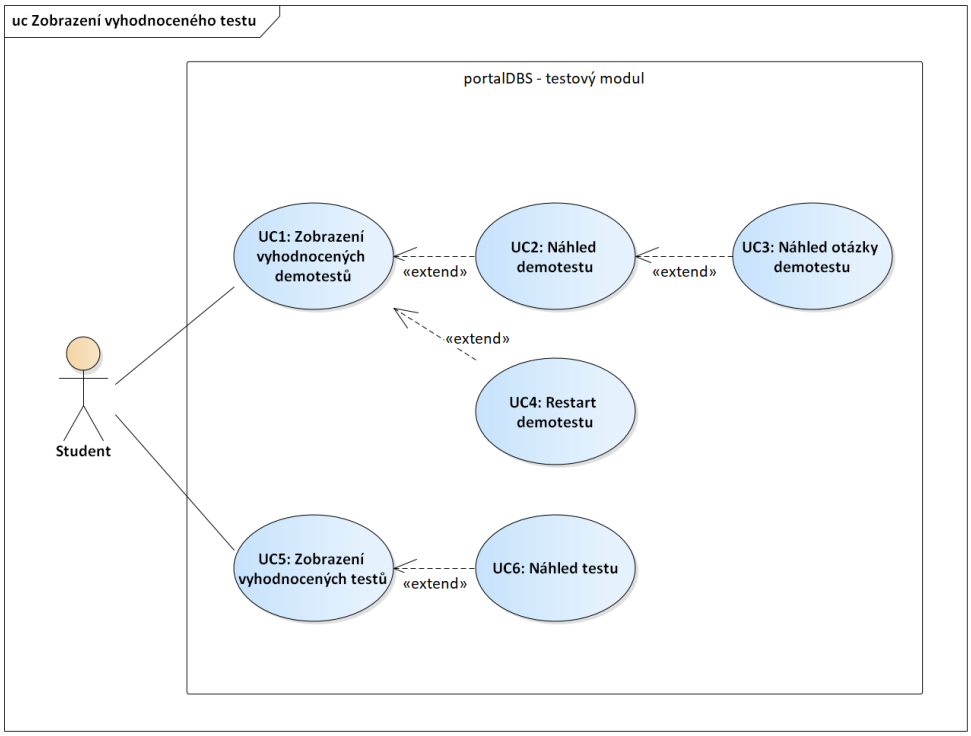
1. Student se nachází v seznamu otázek daného testu, do kterého vstoupil a klikne na a tlačítko *Ukončit test*.

2. Systém se studenta zeptá na potvrzení, zda chce opravdu test ukončit a pokud obdrží kladnou odpověď, test ukončí.

3. Systém odešle test k vyhodnocení.

1.2.4 Zobrazení vyhodnoceného testu

Student si může zobrazit svůj vyhodnocený test v testovém modulu portalu. Pro zobrazení semestrálních testů a zkoušek student zvolý v postraní liště položku testy. Zde se mu zobrazí tabulka aktuálně běžících testů a tabulka již dokončených, vyhodnocených testů. Zobrazení vyhodnocených demotestů je odlišné. Základní rozdíl si ukážeme v této kapitole.



Obrázek 1.3: Diagram případu užití: Zobrazení vyhodnoceného testu

1.2.4.1 UC1: Zobrazení vyhodnocených demotestů

Tento scénář umožnuje studentovi zobrazit si přehled demotestů, které napsal.

Aktéři: student
Scénář:

1. Student v postraní liště testového modulu portálu DBS zvolil položku *Demotesty*.

2. Systém studentovi zobrazí tabulky s aktvivními demotesty, které jsou mu dostupny vyplnit.

3. Systém pod tabulkou dostupných demotestů zobrazí další tabulku *Ukončené demotesty*.

4. Student v této tabulce vidí sloupce s daty o demotestech. Sloupce jsou *Název, Čas spuštění, Čas ukončení, Stav, Čas na vypracování*.

5. Poslední sloupec jsou akce, které se může student pokusit vykonat. Akce jsou dvě, první je *Náhled*, která vede na UC2 a druhou je *Opakovat demotest*, která vede na UC4.

1.2.4.2 UC2: Náhled demotestu

Touto akcí si student může zobrazit demotest, aby zjistil něco více o tom jak si v demotestu vedl.

Aktéři: student

Počáteční podmínka: Student má v tabulce *Ukončené demotesty* alespoň jeden záznam.
Scénář:

1. Student u záznamu v tabulce *Ukončené demotesty* zvolí akci *Náhled*.

2. Pokud je test již vyhodnocen, systém studentovi zobrazí přehled otázek vybraného demotestu formou tabulky. Tato tabulka obsahuje sloupce *Název, Typ odpovědi, Kategorie, Akce*.

3. Student z tohoto náhledového okna může pokračovat zvolením akce *Náhled* na UC3, nebo tlačítkem zavřít se vrátit na původní náhled demotestů (UC1).

1.2.4.3 UC3: Náhled otázky demotestu

Tato aktivita studentovi umožnuje podívat se na otázku, kterou vyplňoval v demotestu. Akce je dostupna pouze u demotestů. Demotesty zpravidla obsahují pouze otázky pro ně určené. Zamezuje se tak úniku informací o otázkách, které se používají v hodnocených testech, tedy semestrální test a zkouška.

Aktéři: student

Počáteční podmínka: Test byl již vyhodnocen a studentovi byl zobrazen náhled otázek z UC2.
Scénář:

1. Student v UC2 zvolil u vybrané otázky akci *Náhled*.

2. Systém zobrazí studentovi okno s detaily o otázce a jeho odpovědi.

3. Student v zobrazeném okně má dostupné celé zadání a úkol otázky. Vidí svou vyplňenou odpověď.

4. V závislosti na typu otázky a studentově odpovědi systém zobrazuje také automatickou nápovědu v čem student chyboval. Výstup této automatické nápovědy může být například, že studentovo SQL dotaz vrátil nesprávný počet sloupců.

1.2.4.4 UC4: Restart demotestu

Demotesty studentovi slouží pro přípravu na semestrální test, případně zkouškový. Tento nebodovaný test může student opakovat kolikrát chce, ale jen po určitou časovou dobu. Pro opakování demotestu si jej student musí znovu nechat vygenerovat.

Aktéři: student

Počáteční podmínka: Student má v tabulce *Ukončené demotesty* alespoň jeden záznam.

Scénář:

1. Student u zvoleného demotestu použije akci *Opakovat demotest*.
2. Pokud ještě nevypršel čas na restartování, systém vygeneruje nový demotest a přidá ho studentovi do Spuštěných demotestů, kde do něj může student vstoupit a začít ho vyplňovat.
3. V opačném případě, kdy již byla možnost opakovat demotest uzavřena, je studentovi zobrazena patřičná hláška o nemožnosti opakování.

1.2.4.5 UC5: Zobrazení vyhodnocených testů

Pro náhled výsledku v semestrálním či zkouškovém testu student provede tuto aktivitu.

Aktéři: student

Scénář:

1. Student v postraní liště testového modulu portálu DBS zvolil položku *Testy*.
2. Systém studentovi zobrazí tabulky s aktvivními testy, které jsou mu dostupné vyplnit.
3. Systém pod tabulkou dostupných testů zobrazí také tabulku *Ukončené testy*.
4. Student v této tabulce vidí sloupce s daty o testech. Sloupce jsou *Název, Typ, Stav, Čas na vypracování, Získané body, Body ústní zkouška, Maximální počet bodů*.
5. Poslední sloupec jsou akce, které se může student pokusit vykonat. Pro testy narozdíl od demotestů je dostupná pouze akce *Náhled*, která se ovšem liší od UC2. Náhled testu vede na UC6.

1.2.4.6 UC6: Náhled testu

Tato aktivita slouží studentovi, aby mohl lépe nahlédnout jak si vedl v konkrétním testu. Zobrazí si přehled otázek v testu, na kolik bodů je zodpověděl a kdo mu jeho odpovědi opravoval.

Aktéři: student

Počáteční podmínka: Student má v tabulce *Ukončené testy* alespoň jeden záznam.

Scénář:

1. Student u záznamu v tabulce *Ukončené testy* zvolí akci *Náhled*.
2. Pokud je test již vyhodnocen, systém studentovi zobrazí přehled otázek vybraného testu formou tabulky obdobně jako v UC2. Tabulka se ovšem liší, neboť se jedná o bodovaný test. Tabulka obsahuje sloupce *Název, Typ odpovědi, Kategorie, Odpověď, Získané body, Maximum bodů, Komentář opravujícího, Opravujicí*. Sloupec odpověď poskutuje informaci o stavu odpovědi, zda již byla opravena a jakým způsobem byla opravena. Možnosti jsou *Opravená* pro případ, kde systém nezvládl vyhodnotit studentovu odpověď a otázka musela být opravena manuálně a *Automaticky opravená* pro úspěšně automaticky vyhodnocenou.
3. Akce náhled není v tomto případě dostupná narozdíl od UC2. Slouží k uchování informací o otázkách používaných v bodovaných testech. Tento scénář tedy končí kliknutím na tlačítko *Zavřít* a student je přesměrován zpět na UC5.

1.3 Struktura otázek a testů

V této kapitole jsou stručně shrnuty poznatky z kapitoly předchozí. Jsou popsány atributy a rozdíly mezi jednotlivými typy otázek. Rozdíly mezi druhy testů, tj. zkouška, semestrální test a demotest. Nejsou opomenuta ani zadání.

1.3.1 Otázky

S otázky se setkávají studenti při vyplňování testů v portálu. Otázky jsou tvořeny aktéry s alespoň právy vyučujícího, tento aktér na začátku tvorby otázky vyplní následující parametry:

- *Název* – Slouží zejména pro následné nalezení otázky v systému uživatelem. Například při správě otázek nebo při jejím přiřazování do testové šablony.
- *Úkol* – Toto políčko obsahuje jednoduchý textový popis co má student v otázce dělat. Je zobrazeno zejména studentovi při vyplňování otázky.
- *Kategorie* – Určuje na jaké téma se otázka zaměřuje. Možnosti jsou teorie, model, SQL a relační algebra.
- *Jazyk* – Tímto atributem se rozhoduje, pro kterou jazykovou skupinu studentů je otázka určena. V současné době volba mezi českým nebo anglickým jayzkem.
- *Obtížnost* – Udává informaci o obtížnosti otázky. Tento atribut se zobrazuje například při přiřazování otázek do testu a může vyučujícím poskytnou lepší náhled o obtížnosti testu. Je na výběr mezi lehká, střední a těžká.
- *Vhodná pro demotest* – Pouze pokud je toto políčko zaškrtnuto, je možné otázku přiřadit do demotestu.
- *Typ* – Určuje typ otázky. Podle typu vypadá formulář odpovědi k otázce. O typech otázek dále.

Parametr *Úkol* není většinou dostačující a ani vhodný pro zadání komplexního úkolu otázky. Z tohoto důvodu existuje objekt *Zadání*. Pro použití otázky v testu je třeba, aby měla přiřazeno alespoň jedno zadání, může jich mít přiřazeno i více.

Zadání jsou opět tvořeny aktéry s právy vyučující a jsou spravovány ve správě zadání. Tento objekt je klíčové zobrazit studentům, když odpovídají na otázku, ale né vždy jej chceme zobrazovat studentům u jejich vyhodnocených testů, aby nedošlo k úniku informací o zadání testu do širšího povědomí. Zadání jsou následujících typů:

- *Text* – Textové pole s popisem úkolu k otázce.

- *Obrázek* – Lze nahrát libovolný obrázek, v databázi je uložena cesta k obrázku. Často používáno jako doplňující zadání.

- *Diagram* – Webová komponenta vyvýjena v rámci závěrečných prací [5] [6]. Obsahuje diagram databázového modelu s vlastností pouze pro čtení. Diagram komponenty je uložen jako JSON [7]. Komponenta je dostupná jako open-source [8].

- *Normalizace relačního schématu* – Zadání pro otázku typu Normalizace, poskytuje data, na kterých má student provést normalizaci databáze.

- *Databáze* – Jedná se o nový typ zadání, který přibyl v práci Andriiho Plyskach [2]. Má sloužit k uchování databázového připojení nad kterým daná otázka operuje. Dříve se toto databázové připojení přiřazovalo k odpovědím na otázku což bylo nepraktické. Tento druh zadání není vhodné zobrazovat studentům.

Víše již bylo zmíněno, že otázky jsou vícero typů. Různé typy otázek jsou vhodné pro ověření znalostí z různých témat. V systému typ otázky určuje jakého formátu bude odpověď k dané otázce. Těmito typy jsou:

- *Text* – Jedná se o otevřenou otázku. Odpověď je zadávána jako textové pole. Používá se převážně s kategorií *teorie*.

- *Checkbox* – Uzavřená otázka na principu klasického checkboxu. Studentovi je prezentována skupina výroků a student zaškrtává ty, které považuje za validní odpověď na otázku.

- *RadioList* – Uzavřená otázka, obdobně jako při otázce *Checkbox* je studentovi prezentována skupina výroků, tentokrát s právě jednou správnou odpovědí, student může zvolit pouze jednu možnost.

- *SQL* – Odpověď na tuto otázku vyžaduje dotaz v jazyce SQL, který má vrátit výsledek požadovaný ze zadání nad konkrétní uvedenou databází. Student si při vyplňování testu může tuto otázku zvalidovat, zda je syntakticky správně.

- *RA* – Podobný princip jako při typu *SQL*. Jen v tomto případě je dotaz zadáván v relační algebře, nikoli SQL. Opět je dostupná syntaktická validace.

- *Transformace* – Zde je po studentovi požadován převod konceptuálního schématu na relační. V zadání studenti například obdrží model databáze a do textového pole jej přepíšou relačním zápisem [9].

- *Normalizace relačního schématu* – Po studentovi je požadováno provedení normalizace databáze nad relačním schématem, aby splňovala určité normy [10]. V současné době otázka nepodporuje připojení referenční odpovědi.

Proto, aby student mohl v testu zodpovědět otázku je tedy nejdůležitější atribut *Úkol* a objekt *Zadání* (bez typu zadání databáze). Ostatní parametry slouží zejména pro vyučující, případně garanta, aby v systému mohl otázky lépe spravovat.

1.3.2 Testy

Testy, stejně jako otázky jsou tvořeny aktéry s právy vyučující a vyšší. Zejména slouží pro otestování znalostí studentů. V systému evidujeme tři typy testů – demotest, semestrální test a zkouška. Alternativně tyto testy lze rozdělit na bodované a nebodované. Bodované testy jsou takové jejichž bodový zisk studentem ovlivňuje studentovo hodnocení předmětu. Nebodovaným testům nejsou přiřazovány body a celkové hodnocení studenta neovlivňují. Pro obě skupiny testů je důležité evidovat tyto vlastnosti:

- *Čas zahájení* – Ve formě časové značky udávající datum a čas, kdy konkrétní student začal test vyplňovat.
- *Čas ukončení* – Ve formě časové značky udávající datum a čas, kdy konkrétní student test manuálně ukončil nebo byl jeho pokus o vyplnění ukončen systémem například z důvodu vypršení času na vypracování.
- *Čas na vypracování* – Zpravidla udáváno v minutách. Reprezentuje kolik minut má student na vypracování testu od zahájení vyplňování testu.

Jediným *nebodovaným* typem testu je demotest. Demotest je studentům zpřístupněn, aby se mohli připravit na testy bodované. Studenti tento druh testu mohou v rámci časového okna opakovat kolikrát chtějí. Při každém opakování je jim vygenerována nová instance tohoto testu často s různými otázkami. Při náhledu tohoto testu si student může zobrazit všechny otázky v testu, u otázek vidí úkol, zadání, svojí odpověď a v některých případech systémem vyhodnocenou nápovědu v čem chyboval.

Bodovanými druhy testů jsou semestrální test a zkouška. Semestrální test ověřuje znalosti z předmětu studentů v průběhu semestru. Zkouška pak ověřuje finální znalosti studenta nabyté z předmětu při zkouškovém období. Tyto testy lze opakovat dle pravidel předmětu/fakulty. Při opakování tohoto druhu testu studentem je mu opět vytvořena nová instance patřičného testu s možnými různými otázkami. V náhledu bodovaných testů není studentům zpřístupněn náhled otázky. Jedinou zpětnou vazbou pro ně v tomto případě je textový komentář vyučujícího, který nemusí být vždy dostupný. U bodovaných testů je navíc důležité evidovat následující infromace:

- *Požadované body* – Tato vlastnost reprezentuje kolik bodů musí student z testu minimálně získat, aby mohlo jeho absolvování testu být považováno za úspěšné.
- *Získané body* – Uvádí kolik bodů z testu získal konkrétní student. Součet všech bodových zisků otázek.
- *Maximum bodů* – Udává kolik bodů je maximálně možné získat z testu. Součet maximálních možných získatelných bodů ze všech otázek v daném testu musí být roven této hodnotě.

Test je tedy soubor otázek s vlastními atributy. Při zobrazování vyhodnocených testů studentovi je vhodné poskytnout mu výše zmíněné informace o času a bodování.

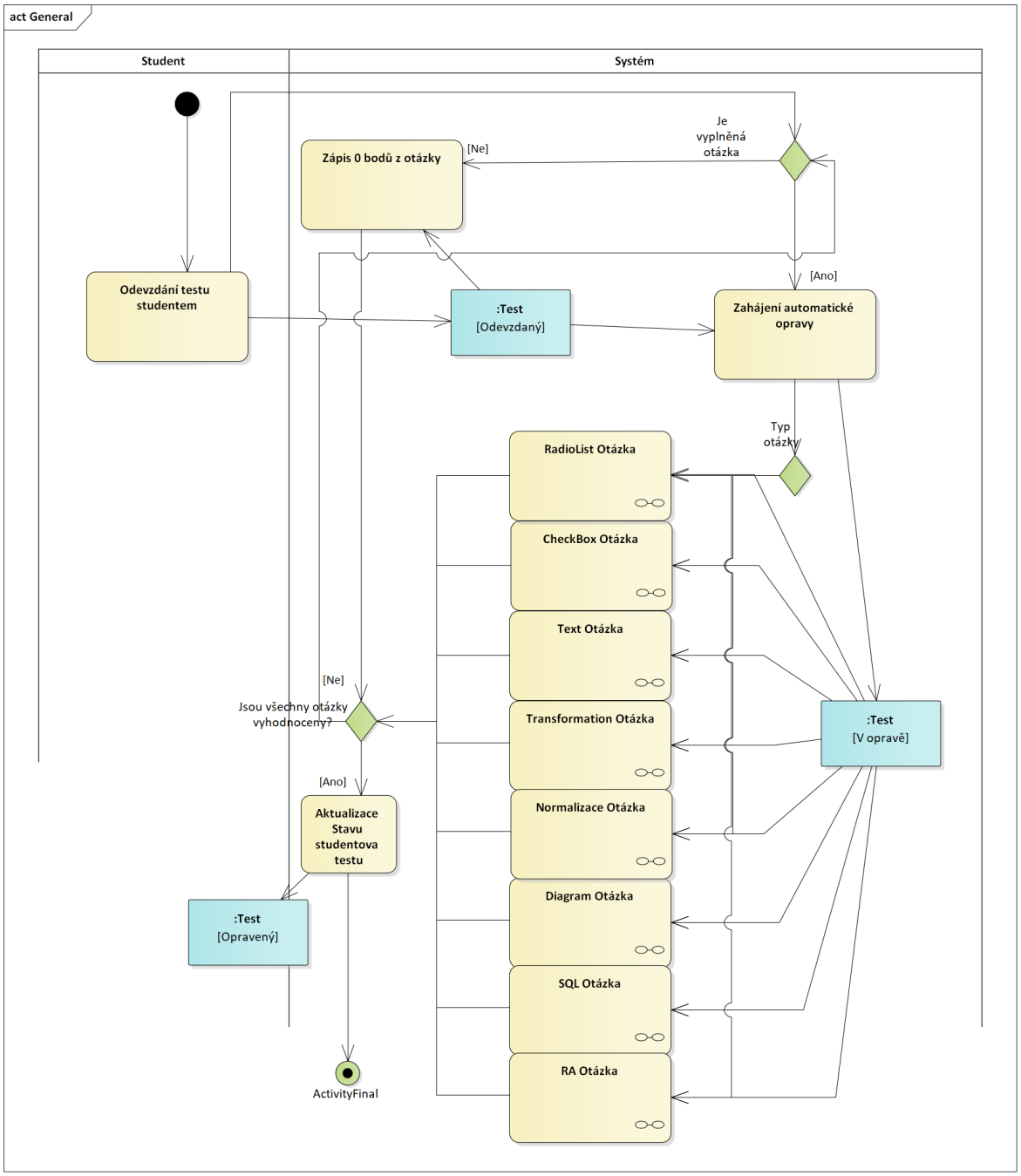
1.4 Vyhodnocování odpovědí

Poté co studenti v portálu zodpoví otázky v testu, je třeba jejich odpovědi vyhodnotit. Kvůli velkému počtu studentů by bylo pouhé manuální opravování odpovědí příliš náročné pro vyučující. Proto systém poskytuje funkci automatické opravy.

Při analýze vyhodnocování odpovědí bylo spolupracováno se studenty předmětu BI-SP1. Studenti se na začátku předmětu seznamují s DBS portálem, aby jej později mohli dále vyvýjet. Také se učí modelovat diagramy, konkrétně modelují části portálu. Na společných schůzkách jsme debatovali na co by se měli zaměřit při seznamování s portálem a následně jak postupovat při vývoji. Jako výstup jejich seznamování s portálem byla řada UML diagamů. Aktivity diagramy vyhodnocování odpovědí budou použity v této kapitole. Celkový přehled aktivity diagramů a výčet jejich autorů je dostupný v příloze C.

Automatická oprava testu je zachycena na obrázku 1.4. Systém postupně bere všechny otázky v testu a vyhodnocuje je. Můžeme vidět, že na začátku vyhodnocování samostatné otázky systém nejprve zkontroluje, zda má otázka vyplňenou odpověď. Samotná oprava otázky může být časově náročná i pro systém, tímto se tedy systém vyhne zbytečné opravě otázky, která nemůže být správně. Otázka s prázdnou či vynechanou odpovědí je ohodnocena 0 body. Pokud otázka obsahuje vyplňenou odpověď, je na ní dále aplikováno automatické vyhodnocování. Z kapitoly 1.3 víme, že existuje více druhů otázek, každý typ otázky má svůj vlastní algoritmus vyhodnocovaní, více si o nich řekneme později v této kapitole. Pokud skončila automatická oprava pro všechny otázky v testu (ať už úspěšně či neúspěšně), je test označen za opravený a jeho automatická oprava je tímto ukončena. V opraveném testu stále mohou existovat otázky, které nebyly automaticky vyhodnoceny úspěšně, takové otázky jsou zařazeny k manuální kontrole. Student v náhledu opraveného testu nemusí mít tedy všechny otázky opravené, některé mohou čekat na manuální opravu.

Přestože každá otázka má svůj vlastní algortimus vyhodnocení, mnoho z nich sdílí podobné koncepty. Například porovnávání s ostatními referenčními odpověďmi a následné porovnávání s částečně správnými. U otázky typu SQL, kde student zadává jako svou odpověď dotaz v jazyce SQL je samotný dotaz dokonce volán nad kontextovou databází a výstup dotazu je dále zpracováván. Protože zabývat se každým typem otázky v této kapitole by bylo zdlouhavé a nepřineslo mnoho nových informací, zaměří se tato kapitola na rozbor otázky pokrývající co nejvíce společných konceptů. Všechny strageie vhodnocování otázek jsou zaznamenány v příloze C.



Obrázek 1.4: Diagram aktivit: Automatické vyhodnocení testu

1.4.1 Rozbor vyhodnocení otázky typu RA

Algoritmus vyhodnocování otázek se lehce liší pro každý typ otázky.
V následujícím textu je popsán algortimus vyhodnocení otázky typu RA, který je také zachycen na obrázku 1.5. Tato otázka obsahuje mnoho konceptů vyhodnocování, které se používají i u jiných typů otázek.

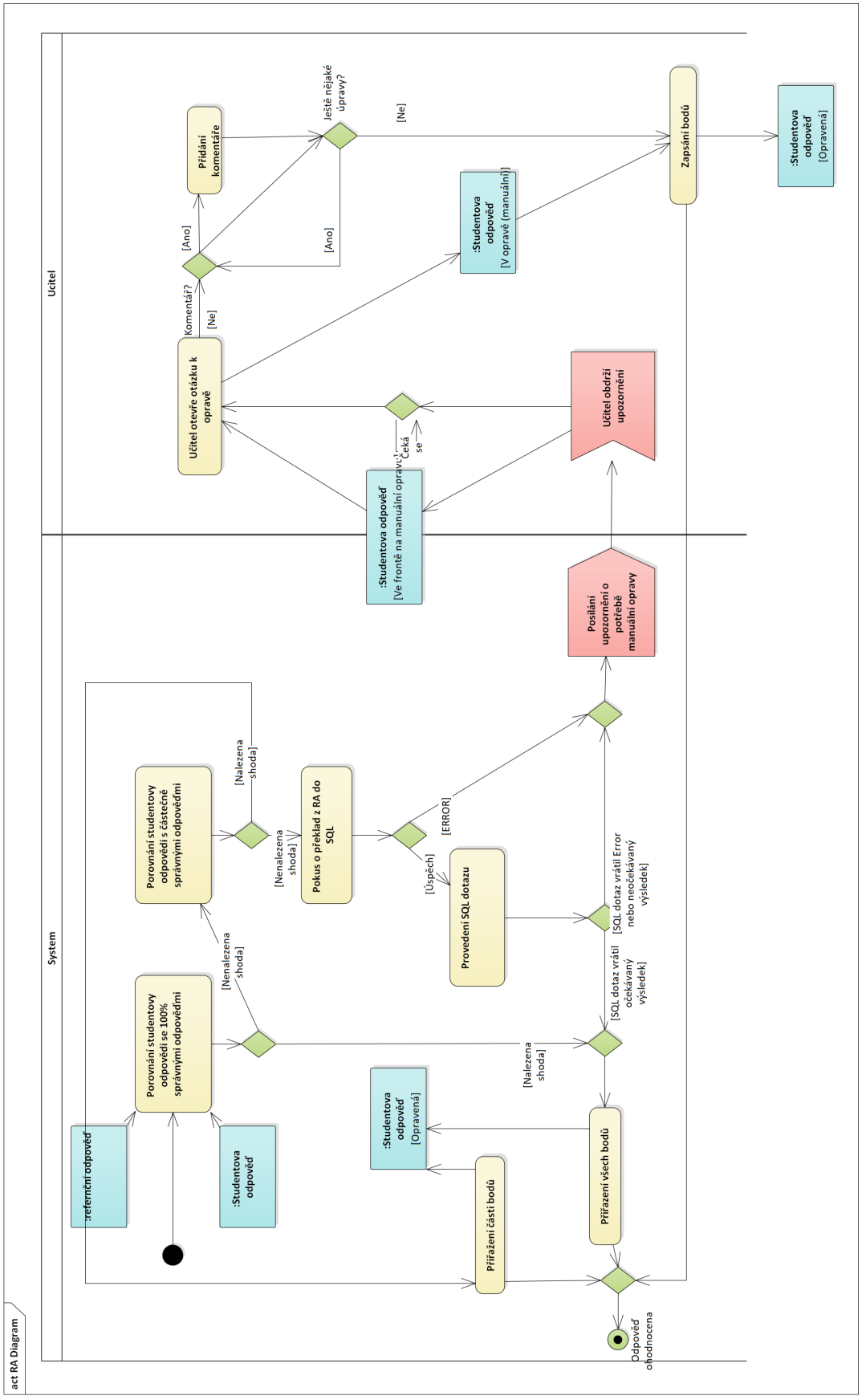
Na začátku je odpověď porovnána s odpověďmi, které jsou úplně správné. Takové odpovědi mohou být buď referenční, které přidal vyučující nebo jsou to odpovědi studentů, jenž byly ohodnoceny za plný počet bodů automatickou či manuální opravou. Pokud dojde ke shodě při tomto porovnávání, je otázka také hodnocena plným počtem bodů a automatická oprava je ukončena. Nedojde-li ke shodě, algortimus pokraćuje dál.

V dalším kroku se odpověď porovná se všemi částečně správnými odpověďmi na tuto konkrétní otázku. Částečně správně odpovědi jsou již opravené odpovědi od studentů, které získali více jak 0 bodů. Obdobně jako v předchozím kroku, při shodě porovnání je otázka ohodnocena stejným počtem bodů jako otázka, s kterou se úspěšně porovnala. Pokud opět nedošlo ke shodě, přejde se na další krok.

Otázka typu RA jak víme z kapitoly 1.3.1 testuje studenty na znalost relační algebry, odpověď se tedy očekává zadána v relační algebře. Automatická oprava se nyní pokusí přeložit odpověď z relační algebry do jazyka SQL. Využívá k tomu překladač jehož vývojem se zaobíral Martin Kubiš jak ve své bakalářské, tak později i v diplomové práci. [11]. V případě, kdy překlad skončí úspěšně, algortimus může dále pracovat s nově získaným SQL dotazem. Opačně, pokud překlad skončil neúspěchem, je otázka předána k manuální kontrole.

Při úspěšném překladu je na nově vzniklý SQL dotaz zavolána kontrola na SQL injection. Pokud by dotaz obsahoval kód, který by mohl poškodit databázi, automatická kontrola končí a odpověď je předána na manuální kontrolu. V jiném případě algortimus zavolá SQL dotaz na databázi, jenž je přiřazena u otázky. Pokud toto volání skončí jakoukoli chybou, otázka je opět zařazena k manuální kontrole. Pokud volání dotazu chybou nekončí je porovnán výstup tohoto dotazu s výstupem dotazu referenční odpovědi. Jsouli výstupy shodné, otázka je ohodnocena plným počtem bodů a algoritmus končí. V případě neshody výstupů je otázka předána na manuální kontrolu.

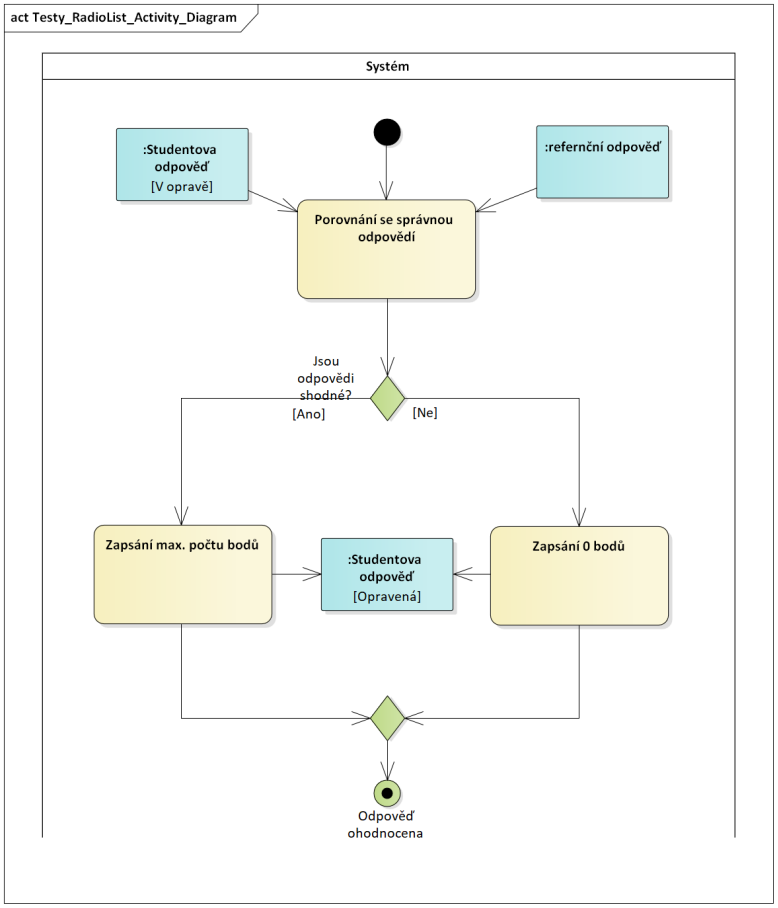
Každá otázka v testu má přiřazeného svého kontrolujícího. Kontrolující je vyučující, případně garant. Kontrolující je zodpovědný za manuální opravu otázky. Kontrolující tedy otevře otázku k opravě a zhodnotí ji. Má možnost přidat textový komentář, ale musí zapsat body. Po zapsání bodů je otázka vyhodnocena a vyhodnocování je u konce.



Obrázek 1.5: Diagram aktivit: Algoritmus vyhodnocení RA otázky

1.4.2 Uzavřené otázky

Za zmíňku stojí otázka typu Radio, která využívá radio button. V této otázce student jako svou odpověď zadává volbou právě jednoho zaškrtávacího políčka z nabídky. Narozdíl od otevřených otázek je zde tedy správná odpověď vždy jednoznačná. To značně zjednodušuje vyhodnocování této otázky. Systém je schopen jí vždy plně automaticky vyhodnotit, není tedy potřeba manuální kontroly čímž není přidávána práce na vyučujícího. Strategii vyhodnocení zachycuje obrázek 1.6. Podobně funguje i otázka typu Checkbox, její algoritmus vyhodnocení lze dohledat v příloce C.



Obrázek 1.6: Diagram aktivit: Algoritmus vyhodnocení Radio otázky

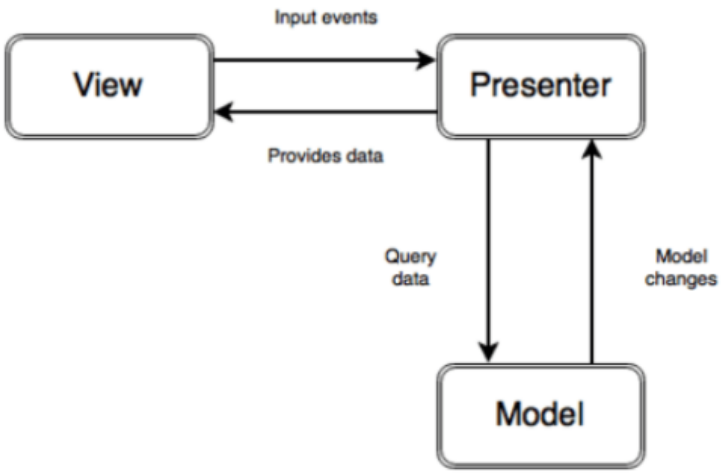
1.5 Architektura

Webový portal DBS je napsán v Nette. Nette je PHP framework využívající architekturu MVP (Model-View-Presenter), která je velice podobná architektuře MVC (Model-View-Controller). Aplikace je rozdělena do tří vrstev:

- *Model* – Datová vrstva se stará o zápis a čtení dat z databáze
- *View* – Neboli prezentační vrstva, zobratuje obsah uživateli. V Nette se typicky jedná o šablony stránek a komponenty na nich zobrazené, např. datagridy, formuláře.
- *Presenter* – Aplikační vrstva která sprostředkovává komunikaci mezi model a view, také je zde obsažena logika aplikace. Presenter je varianta controlleru.

MVP se od MVC liší hlavně v tom, že view nemůže komunikovat přímo s modelem, presenter tedy tvoří takové prostředníka pro tyto dvě vrstvy. V Nette je také zpravidla použit jeden presenter pro jednu stránku webové aplikace, zatímco v klasickém MVC přístupu se controller může starat i o více stránek [1] [12].

Ve vrstvě view se v případě MVP architektury také provádí validace vstupů od uživatele, například ve formulářích. Tato vrstva je v Nette realizována pomocí šablon napsaných v šablonovacím jazyce Latte. Šablony mohou obsahovat tzv. AJAX snippety což jsou úryvky obsahu vykresleny na dané stránce bez nutnosti znovu překreslení celé stránky [13] [14].



Obrázek 1.7: Princip MVP architektury [1]

1.6 Požadavky

Tato kapitola shrne požadavky, na které se bude třeba zaměřit při návrhu nového frontendu. V předchozích kapitolách byl zdokumentován současný stav systému a společně s ním zachyceny i nedostatky, které nabízí. Frontend bude zaměřený na studentskou část portálu, kde si studenti zobrazují výsledky svých testů. Je důležité, aby zachovával užitečné současné funkce, ale i rozšířil o funkce, které zlepší zpětnou vazbu pro studenty.

Analýza požadavků v softwarovém inženýrství je souhrn toho co by měl výsledný systém umět. Požadavky popisují co by systém měl umět, ale neřeší jak by tyto funkce měli být realizovány. Funkční požadavky popisují funkce systému prováděné uživateli. Nefunkční požadavky se pak soustředí na vlastnosti systému, například bezpečnost, stabilita, výkon, flexibilita [15].

V praxi by tato ustanovení měla mít jasné hranice, aby nedošlo například k tomu, že ve výsledném produktu bylo dodáno něco co nemělo, nebo něco za co zákazník nezaplatí. V tomto případě budou požadavky sepsány s volnějšími hranicemi, protože slouží zejména jako podklad, na co se zaměřit při návrhu.

1.6.1 Funkční požadavky

F1: Náhled absolvovaných testů
V současném řešení jsou aktivní i uzavřené testy na jedné stránce. Student tedy jde do stejné části modulu ať už se test chystá psát nebo si jde zobrazit výsledky testu již napsaného. V případě bodovaných testů tato stránka pak obsahuje semestrální test i zkoušku v jedné tabulce. Toto řešení není úplně vhodné, neboť například semestrální test pak obsahuje položku *Body ústní zkouška*, ale pro semestrání test se ústní zkouška nepoužívá. V novém řešení by jsme navíc u zkoušky chtěli studentovi udávat i přehled o jeho celkových bodech ze semestru a kolik bodů potřebuje pro hodnocení o stupeň lepší. Nové řešení by mělo oddělit aktivní testy od již uzavřených a poskytnout samostatné tabulky pro každý typ testu.
Priorita: Vysoká
Kategorie: Funkčnost
Náročnost: Střední

F2: Tabulky umožnující řazení/filtraci dat
Aby se studenti v přehledu svých absolvovaných testů lépe vyznali, je vhodné jim poskytnout možnost řazení a filtraci záznamů v tabulkách. Klíčem pro řazení a filtraci by měl jít použít každý sloupec.
Priorita: Vysoká
Kategorie: Funkčnost
Náročnost: Střední

F3: Náhled demotestu
U již vyhodnoceného demotestu má student možnost náhledu. Je mu napřed zobrazena obrazovka s přehledem otázek, dále má u každé otázky možnost zobrazení detailu otázky. V tomto náhledu otázky jsou studentovi zobrazeny všechny důležité informace o otázce. Detailní přehled je k dispozici v kapitole 1.2.4.

Funkcionalita je studentům prospěšná, při zpětném náhledu absolvovaného testu si mohou připomenout na co odpovídali, jakou odpověď zadali a v některých případech zobrazit správnost odpovědi. Nový frontend by měl tuto funkcionalitu zachovat.
Priorita: Vysoká
Kategorie: Funkčnost
Náročnost: Střední

F4: Náhled bodovaného testu
Vyhodnocený bodovaný test nabízí studentům pouze zjednodušený náhled. Detailní rozdíl je dostupný v 1.2.4.

Tato část současného řešení je vnímána jako největší nedostatek. Studenti u bodovaných testů nevidí ani část kontextu otázky, ani svou vlastní zadanou odpověď. Nedostatečná zpětná vazba zejména u bodovaných testů, které mají výrazný vliv na to, zda studenti uspějí v předmětu, není výtána studenty kladně. Nové řešení by mělo tedy zejména adresovat tento bod.

V novém řešení bude možné si u bodovaných testů otevřít detail otázek. Tento detail se bude lišit od detailu v testech nebodovaných. Bude obsahovat pouze základní informace a zpětnou vazbu opravujícího.
Priorita: Vysoká
Kategorie: Funkčnost
Náročnost: Střední

1.6.2 Nefunkční požadavky

NF1: Multijazyčnost portálu
Portál se používá i pro výuku předmětu BIE-DBS, kde je výuka vedena v anglickém jazyce, proto je potřeba multijazyčnost portálu. V současné době lze přepínat mezi jazyky *Čestina* a *Angličtina*. Nový frontend musí reagovat na tyto změny jazyka.
Priorita: Střední
Kategorie: Vhodnost k použití
Náročnost: Jednoduchá

NF2: Návaznost na nově vzniklý backend
Jedním z důvodů pro vznik tohoto řešení je potřeba frontendu pro nový testový modul, jehož backend začal vyvýjet Andrii Plyskach ve své BP [2]. Toto řešení bude tedy dále rozvýjet nový testový modul portálu DBS. Měla by být zachována nezavislost na ostatních modulech a brán ohled na novinky zavedené v novém řešení.
Priorita: Vysoká
Kategorie: Podporovatelnost
Náročnost: Vysoká

NF3: Technologie pro tabulky
V portálu se pro tabulky používají v souČasné době dvě různé komponenty – *Ublaboo/datagrid* a *o5/Grido*. Na první pohled nabízejí shodnou funkcionalitu (řazení, filtraci a další), z hlediska konzistence je vhodné používat pouze jeden typ tabulkové komponenty. Dle budoucí rešerše by měli být tabulky realizovány vhodnější komponentou z nabízených. Rešerše v 3.4.8.
Priorita: Vysoká
Kategorie: Implementace
Náročnost: Střední

Kapitola 2

Návrh

Následující návrh se zaobírá návrhem drátěného modelu pro tvorbu frontendu na základě zjištěných vlastností portálu a následně sestavených požadavků na konci analýzy. Dále je využito doménového modelu pro abstraktnější návrh komponent, který bude podklad pro implementaci. Poslední část se zaobírá jak implementovat vyhodnocovací algoritmy nad vzniklým backendem pro nový testový modul portálu.

2.1 Drátěný model

Drátěný model neboli Wireframe je návrh rozmístění funkčních prvků na stránce. Jeho cílem není zachytit výsledný grafický vzhled, neobsahuje obrázky ani loga a je tvořen pouze pomocí čar a textu. Použití barev by mělo být jen za účelem zvýraznění, kde je třeba. Drátěný model připravuje informační architekt, ten přenese do nákresů požadavky zákazníka. Následný model je pak prezentován zákazníkovi, ale i programátorům a grafikům, kteří přesně vidí co na webu bude za komponenty a kde. Ideálně by tento model měl být přikládán ke smlouvě se zákazníkem, aby v pozdějších fázích nemohlo dojít k tomu, že jedna strana ve výsledku vyžaduje co nebylo sjednáno ve smlouvě.

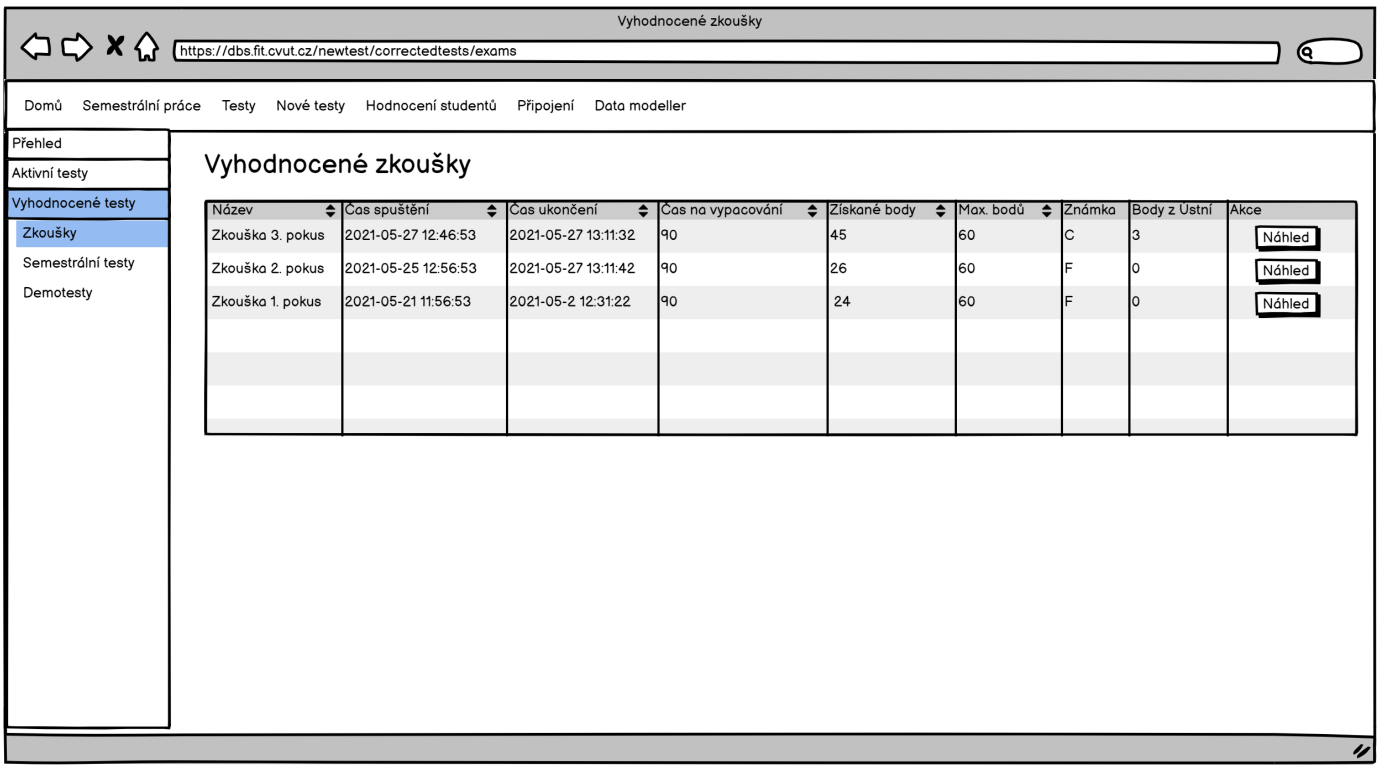
Drátěné modely můžeme rozdělit na několik typů. Textový wireframe pouze slovně definuje struktury stránek. Podrobný wireframe je už vymodelování stránek využitím nějakého softwaru, je tedy grafickou reprezentací rozmístění komponent. HTML prototyp je také druhem drátěného modelu, přidává chování a naplno tak ukazuje provázanost a smysl celého systému [16].

V této kapitole byl pro návrh nového frontendu použit podrobný wireframe vytvořený v programu Balsamiq Wireframes. Tento program umožnuje tvorbu drátěných modelů stránek z různých komponent, ať už předpřipravených nebo i vlastních. Výsledné drátěné modely stránek se pak dají přes odkazy propojit a vytvořit tak interaktivní model webu.

V následujících kapitolách jsou zobrazeny a popsány nejdůležitější stránky z navrženého drátěného modelu. Projekt s celým navrženým drátěným mo-

delem je pak dostupný na přiloženém paměťovém médiu k této práci. Tento model byl prezentován vedoucímu práce Ing. Jiřímu Hunkovi a po zakomponování pár jeho nápadů byl schválen.

2.1.1 Přehled vyhodnocených zkoušek



Obrázek 2.1: Drátový model: Přehled absolvovaných zkoušek

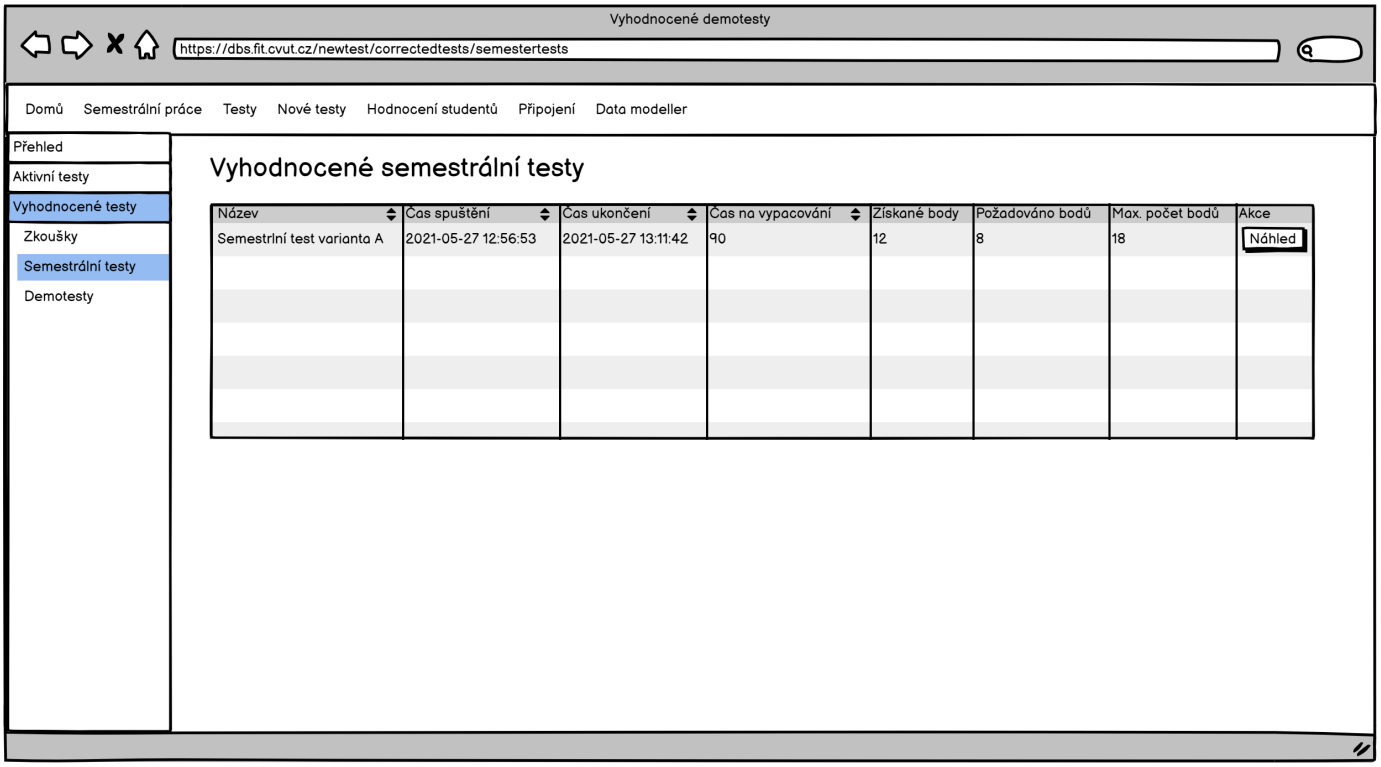
Obrazovka na obrázku 2.1 zachycuje jak bude vypadat zobrazení vyhodnocených zkoušek studentovi. Lze vidět, že aby se student na tuto obrazovku dostal, musel v postraní liště zvolit rozbalovací položku *Vyhodnocené testy* a z následně rozbalených vybrat *Zkoušky*.

Stránka obsahuje tabulku, která studentovi vylistuje všechny zkoušky z předmětu BI-DBS, které již absolvoval. Za předpokladu, že student má tedy předmět zapsán poprvé a ještě u zkoušky nebyl, bude tato tabulka prázdná.

Každý záznam tabulky prezentuje jednu zkoušku, které se student účastnil. Studentovi jsou prezentovány informace o jaký pokus se jednalo, kdy zkoušku začal psát, kdy jí ukončil a kolik měl dostupného času na celou zkoušku. Dále je v tabulce sloupec pro body, které získal ze zkoušky a sloupec pro maximální bodový zisk. Sloupec *Známka* studentovi sděluje jeho celkový výsledek zakončení předmětu, základě toho sloupce se student může rozhodnout, zda

chce zkoušku opakovat, neboť není spokojen se svým výsledkem. Informace o celkovém výsledku studenta je dostupna i v modulu s hodnocením, zde je tato informace explicitně přidána, aby byl student na výsledek upozorněn. *Body z ústní* prezentují bodový zisk z ústního zkoušení, pokud se ho student neúčastnil, v políčku je nula. Akcí *Náhled* pak student přechází na náhled testu popsaného v 2.4.

2.1.2 Přehled vyhodnocených semestrálních testů

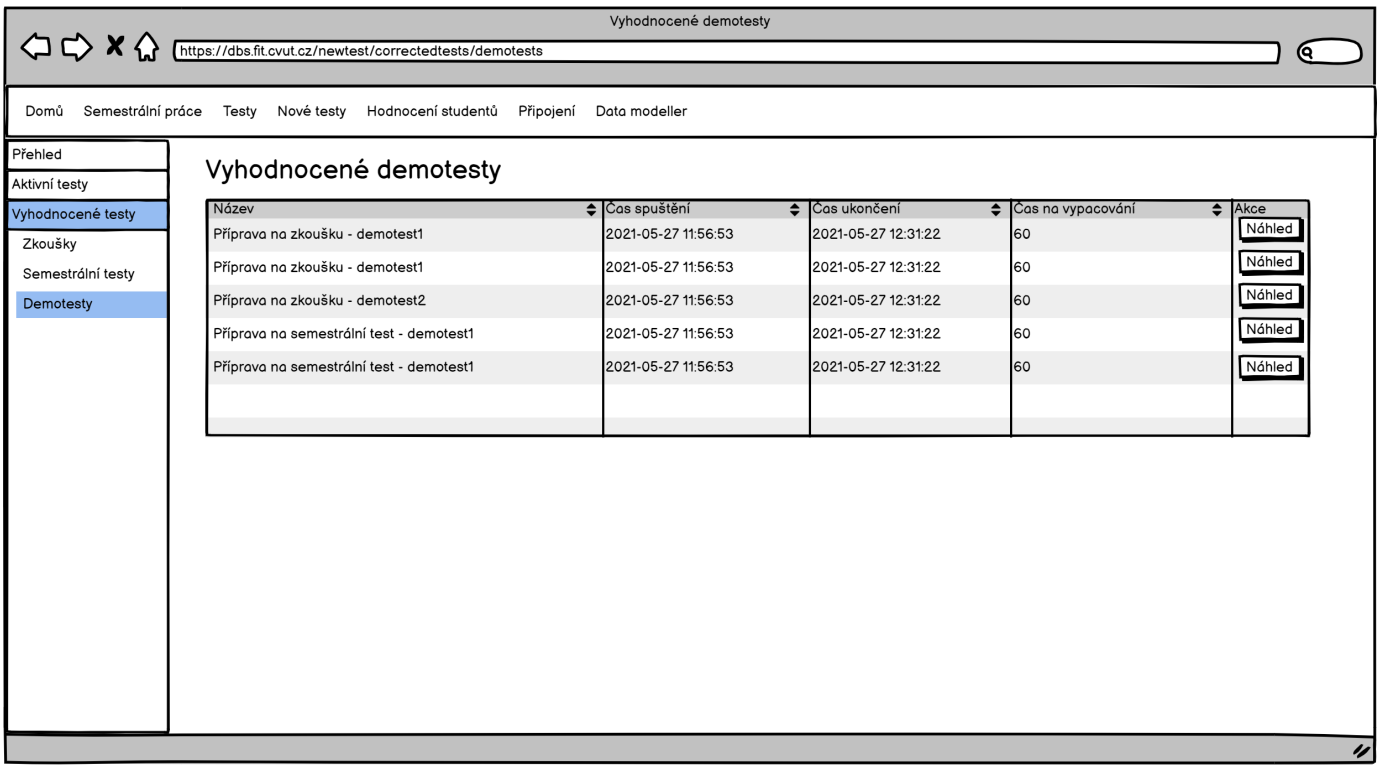


Obrázek 2.2: Drátový model: Přehled absolvovaných semestrálních testů

Na obrázku 2.2 je zachycen pohled studenta na přehled absolvovaných semestrálních testů. V ideálním případě student píše tento druh testu jen jednou v semestru, za předpokladu, že jej absolvuje na poprvé. Může se však stát, že student napoprvé neuspěje a pak tato tabulka obsahuje i opravné pokusy.

Podobně jako u přehledu vyhodnocených zkoušek je studentovi zobrazeno pro každý záznam tabulky čas spuštění, čas ukončení, čas na vypracování, získané body a maximální možný bodový zisk. Student z tohoto druhu testu musí získat alespoň požadovaný počet bodů, ten je také uveden v tabulce. Akcí *Náhled* opět student přejde na náhled daného testu, tento pohled je dále popsán v kapitole 2.4.

2.1.3 Přehled vyhodnocených demotestů

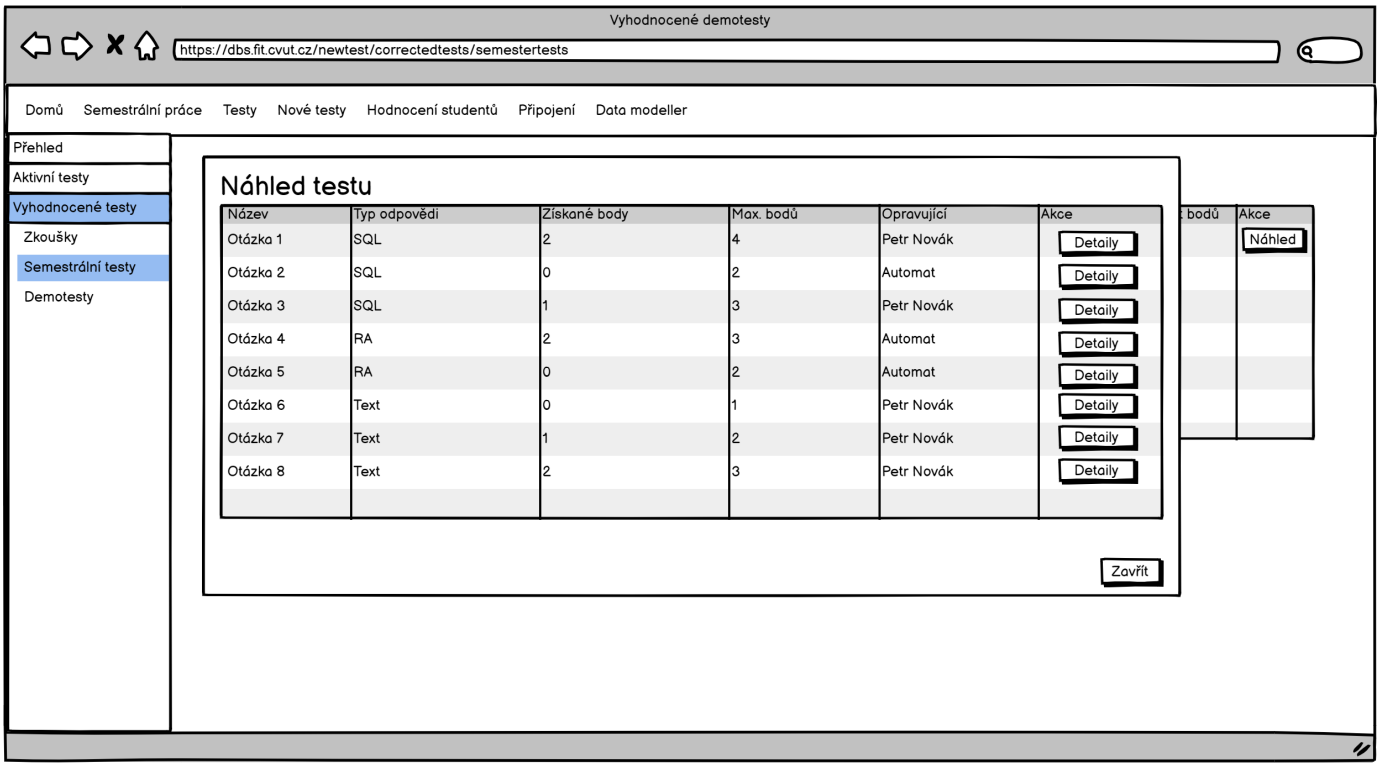


Obrázek 2.3: Drátový model: Přehled absolvovaných demotestů

Výše uvedený obrázek 2.3 zachycuje jak jsou studentovi zobrazeny jím absolvované demotesty. Demotesty jsou nebodovaným typem testu, tabulka s absolvovanými demotesty tedy obsahuje pouze pět sloupců.

V tabulce student nalezne stejně jako v předchozí náhledech testů sloupec *Čas spuštění* udávající datum a čas, kdy student začal test v portálu vyplňovat. Obdobně také sloupec *Čas ukončení* udávající čas konce testu a sloupec s časem na vypracování. V posledním sloupci je opět dostupná akce na náhled testu. Tento náhled se liší od náhledu pro testy bodované. Je zpracován v kapitole 2.6.

2.1.4 Náhled bodovaného testu

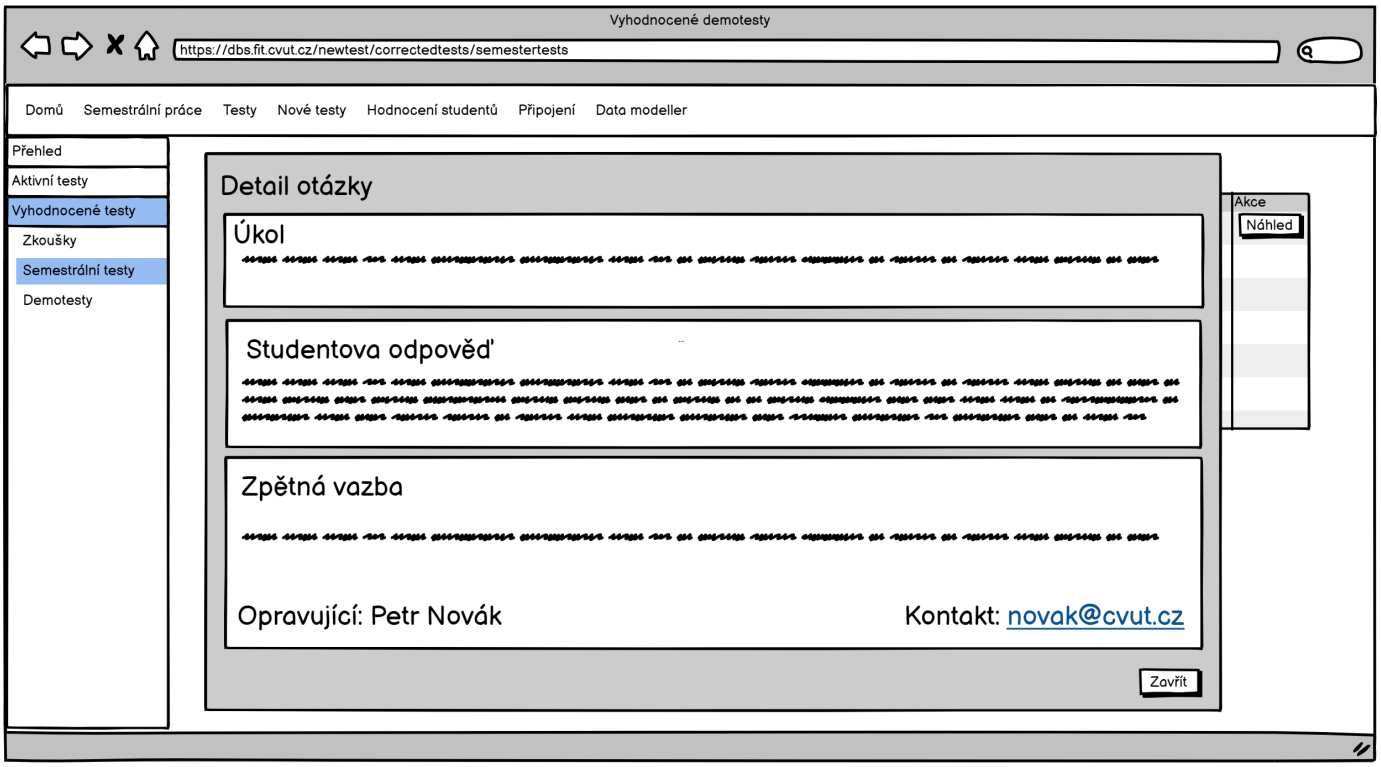


Obrázek 2.4: Drátový model: Náhled semestrálního testu

Obrazovka zachycena na obrázku 2.4 zachycuje náhled bodovaného testu studentem. V tomto případě jde o náhled semestrálního testu, náhled zkoušky ovšem funguje na stejném principu. Pro připomenutí – bodované testy tvoří dílčí část hodnocení studentů z předmětu BI-DBS a pokud automatická oprava otázky z bodovaného testu neskončí úspěšně, otázka je poslána na kontrolu manuální.

Náhled testu obsahuje tabulku, jejíž položky tvoří otázky obsažené v tomto testu. U každé otázky je uveden typ odpovědi reprezentující, zda otázka chtěla jako odpověď např. SQL dotaz nebo textovou odpověď. Dále získané body z otázky a maximální možný bodový zisk za otázku. Sloupec *Opravující* udává, kdo otázku opravil. Pokud otázka byla vyhodnocena automaticky úspěšně, v tabulce je uveden „Automat“. V opačném případě, kdy otázku bylo nutné vyhodnotit manuálně, je v tabulce jméno vyučujícího, který otázku opravil. Pro každou otázku je dostupná akce *Detaily*, která vede na detail otázky z bodovaného testu popsaného v kapitole 2.5.

2.1.5 Detail otázky z bodovaného testu



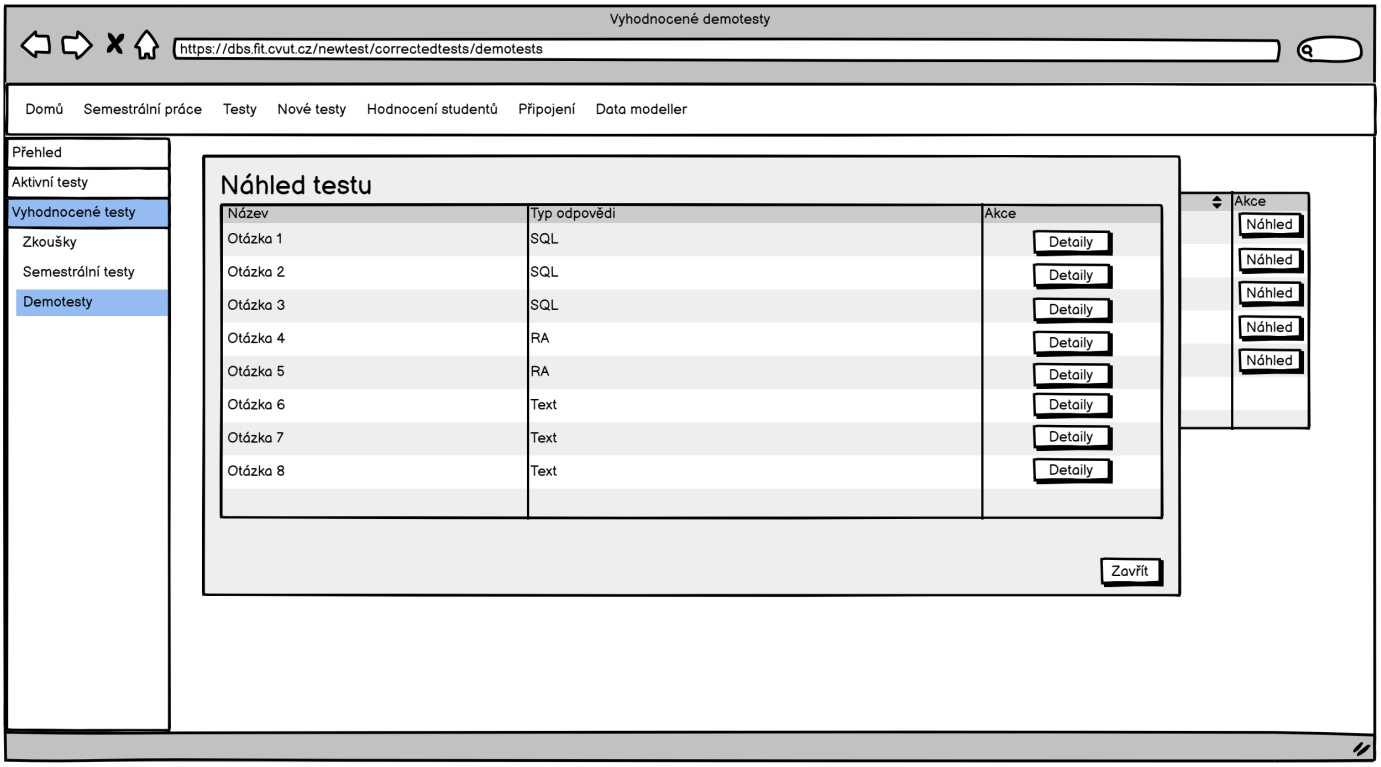
Obrázek 2.5: Drátový model: Detail otázky v semestrálním testu

Podkapitola rozebírá zobrazení detailu otázky z bodovaného testu. Obrázek 2.5 zachycuje konkrétně otázku ze semestrálního testu, ale detail je stejný jako pro otázku ze zkoušky. V současné verzi portálu tento pohled neexistuje, jedná se o rozšíření zpětné vazby pro studenta u bodovaných testů.

Student si může u každé bodované otázky zobrazit tento náhled, kde je nově studentovi zobrazen *Úkol* otázky. Jak bylo řečeno v předchozích částech této práce, úkol otázky je pouze stručně zadáno co má student v otázce dělat, neobsahuje konkrétní zadání. Dříve také studentům nebyla zobrazována jejich vyplněná odpověď. Student si pak mohl myslet, že odpověděl něco jiného než skutečně odpověděl a to mohlo vést na nespokojenost s hodnocením.

Do položky zpětná vazba se nově přemístil komentář opravujícího. Komentář již tedy není omezen na nízký počet znaků, aby se vešel do tabulky. Dále tato položka obsahuje jméno opravujícího a kontakt na něho, aby si student mohl případně domluvit individuální konzultaci v čem chyboval. Pokud byla otázka vyhodnocena automaticky úspěšně, tato položka nebude v tomto řešení dostupná. Přemístění komentáře do samostatné komponenty umožnuje jednodušší rozšiřování zpětné vazby v budoucnu.

2.1.6 Náhled demotestu

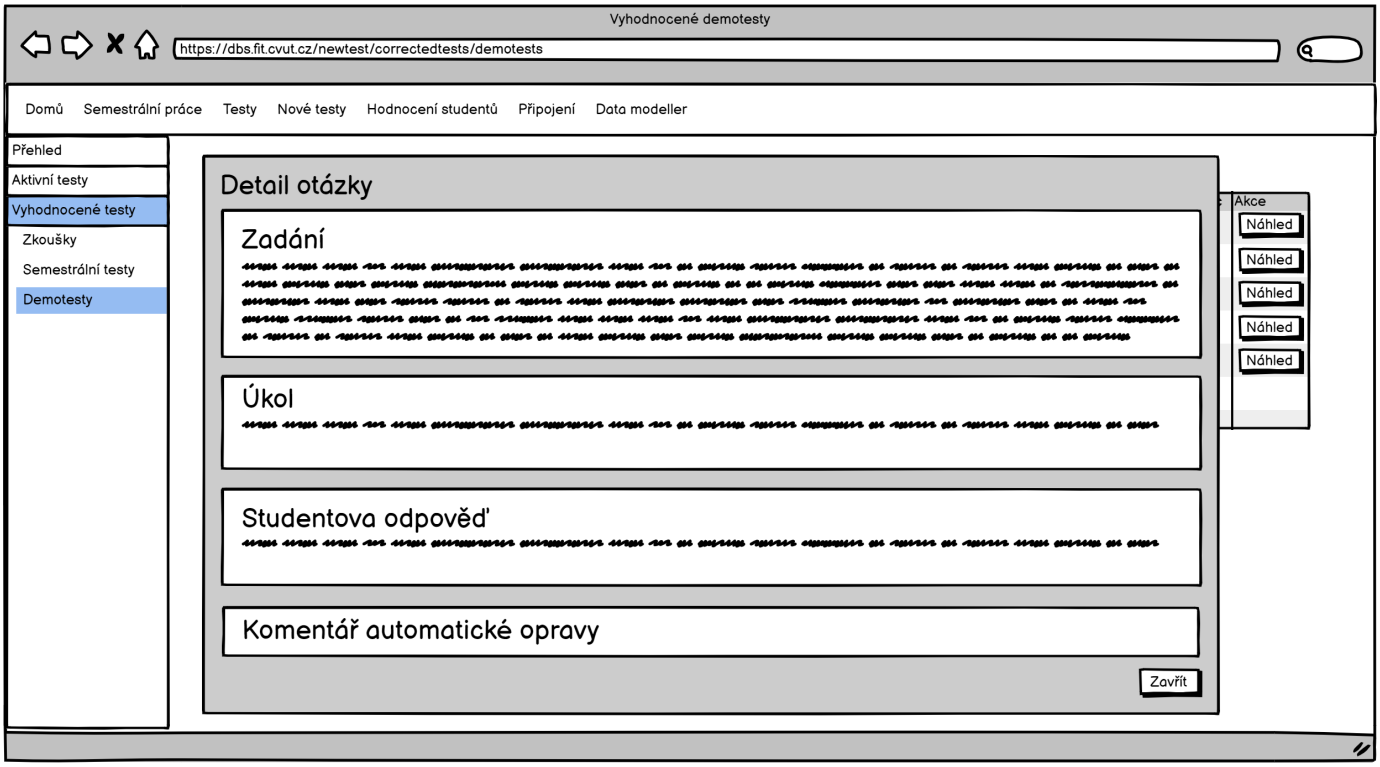


Obrázek 2.6: Drátový model: Náhled demotestu

Náhled demotestu je komponenta zoprazující tabulku s otázkami použitými v daném demotestu. Demotesty nejsou bodovány a tak ani otázky v nich přiřazené nemají svá bodová ohodnocení.

Student se na tento náhled dostane pokud na obrazovce s náhledem absolvovaných demotestů zachycené na obrázku 2.6 klikne na akci *Náhled*. V samotné tabulce student vidí očíslované otázky v pořadí v jakém je měl v testu, když test vyplňoval. U každé otázky je sloupec *Typ odpovědi* udávající v jakém formátu otázka vyžadovala odpověď, např. SQL dotaz nebo zápis v relační algebře. Pro každou otázku je dostupna akce *Detaily*, které zobrazují podrobnější informace o otázce. Detaily otázky v demotestu jsou dále zachyceny v kapitole 2.1.7. Tlačítkem *Zavřít* se student vrací na předchozí obrazovku.

2.1.7 Detail otázky z demotestu



Obrázek 2.7: Drátový model: Detail otázky v demotestu

V této kapitole je popsán detail otázky z demotestu. Pohled je zachycen na obrázku 2.7. Tento detail se liší od detailu otázky z bodovaného testu zachyceném v kapitole 2.5. Demotesty jsou zpravidla tvořeny jinou sadou otázek než testy bodované. U těchto otázek nám nevadí zpřístupnit více informací jako je například zadání.

První položkou v detailu je *Zadání* obsahující zadání otázky. Zadání jsou popsána v kapitole 1.3. Jako u všech detailů otázek i zde je přítomna položka *Úkol* s jednoduchou informací co měl student v otázce dělat. Předposlední položkou je studentova odpověď pro připomenutí studentovi, jakou odpověď vyplnil. Student má v tomto typu detailu veškeré informace pro něho užitečné.

I přestože otázky použíté v demotestech musí mít v systému alespoň jednu referenční odpověď, tato referenční odpoveď nebude studentům zobrazována. Důvodem je podnítit ve studentech zájem, aby si otázku zkusili vyřešit znovu sami a jen si nezapamatovali referenční řešení. Poslední položkou je *Komentář automatické opravy*. V budoucnu by tato položka měla studentovi dávat lehkou nápovědu v čem chyboval.

2.2 Návrh komponent

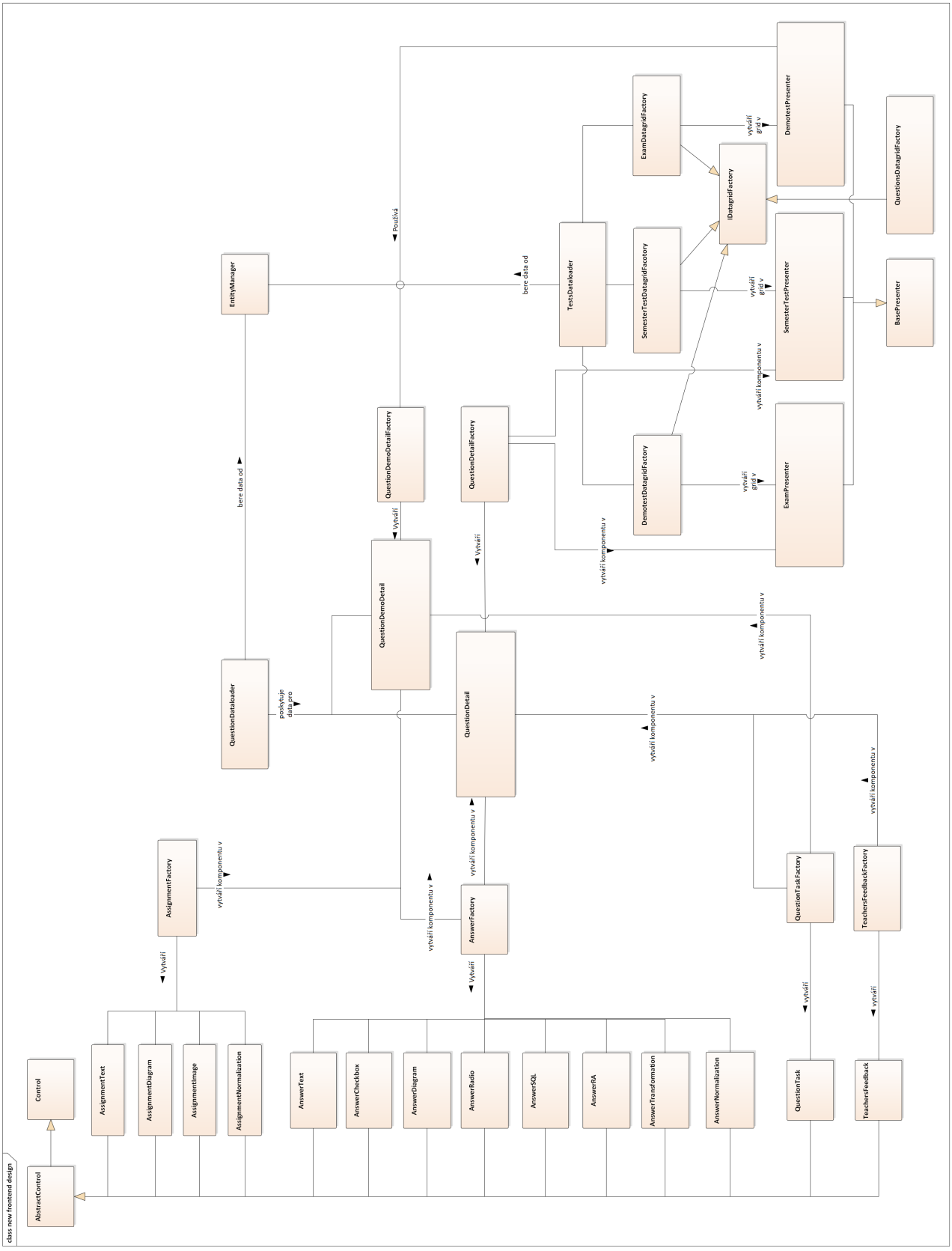
K tvorbě GUI použijeme vestavěný komponentový systém v Nette framework. V současné době je toto řešení použito pro většinu frontendového řešení v portálu. Jak sami tvůrci Nette frameworku tvrdí, mezi PHP frameworky jde o unikátní přístup. Nicméně něco podobného lze nalézt v Delphi nebo ASP.NET Web Forms. V Nette má každá komponenta svojí třídu, která musí dědit od třídy *Control*. Tato komponenta musí mít také svojí šablonu, kterou musíme třídě specifikovat [17]. V portálu se používá pro šablony šablonovací jazyk *latte*. Komponenta může obsahovat i další komponenty a o její vykreslení se postará v presenter. Tvorba komponenty je více popsána v kapitole Realizace.

2.2.1 Doménový model

Tato kapitola dále popisuje navrhnuté třídní schéma pro implementaci nového frontendu. Je využit doménový model k zachycení struktury komponent a vztahů s presentery. V této části jsem se pro zachycení konceptu návrhu rozhodl použít doménový model narozdíl od klasického diagramu tříd z důvodu větší abstraktnosti doménového modelu, která mi pomůže lépe zachytit některé vztahy. Jsou zahrnuty i některé významější pomocné třídy, ale hlavním cílem modelu byl návrh komponent ,a proto menší, pomocné třídy jsou ve schématu pro přehlednost vynechány.

Doménový model je forma class diagramu, kde jsou třídy zapsány zjednodušeně. Třídy nemusí obsahovat atributy a metody a jejich názvy nemusí odpovídat realitě. Často v praxi tento model neodpovídá implementační realitě, není závislí na technologii a slouží hlavně k pochopení problematiky dané domény [18]. Třídy jsou krabičky, které mohou obsahovat atributy. Mezi třídami existují následující vztahy:

- *Asociace* – Určuje základní vztah mezi dvěma entitami. Ty mohou existovat nezávisle na sobě. Zakresluje se jednoduchou plnou čáru.
- *Agregace* – Jedná se o vztah celek - část. Zakresluje se plnou čáru, zakončenou na jedné straně prázdným kosočtvercem. Kosočtverec je umístěn na straně celku. Aneb třídy, která slouží jako kolekce tříd na druhé straně.
- *Kompozice* – Podobná vztahu agregace, ale třída, která slouží jako kolekce (celek) musí mít vždy alespoň jednu část. Kompozice je zakreslena stejně jako agregace, ale s plným kosočtvercem.
- *Generalizace* – Z hlediska implementace se jedná o dědičnost. Jedna entita dědí vlastnosti a chování jiné. Zakresluje se plnou čáru, zakončenou na jedné straně prázdnou uzavřenou šipkou. Šipka je na straně rodiče třídy.



Obrázek 2.8: Doménový model: Návrh komponent

2.2.2 Komponenty

Pod komponenty se skrývají všechny třídy z doménového modelu na obrázku 2.8, které jsou v generalizačním vztahu s třídou *AbstractControl* a tranzitivitou tedy ve vztahu s třídou *Control.* Každá tato třída reprezentuje jednu komponentu, kterou budeme na webu zobrazovat. Komponenty by měli samostatné a znovupoužitelné na více místech v portálu [17].

Třídy začínající v názvu předponou *Assignment* slouží pro vykreslení zadání otázky na webu. Jak bylo rozebráno v kapitole 1.3 Struktura a otázek a testů, zadání je vícero druhů a každý typ bude mít svojí vlastní třídu, která bude zodpovídat za vykreslení zadání. Výjimkou je zadání typu *Databáze*, které slouží jako dodatečná datová strktura pro otázky s databází. V současné době není třeba tento druh zadání zobrazovat. Protože zadání se zobrazují v režimu *read only* budou tyto komponenty použitelné například i ve správě otázek v náhledu otázky vyučujícím.

Třídy začínající v názvu předponou *Answer* budou mít na starost vykreslení komponenty s odpovědí na otázku. Opět dle kapitoly 1.3 víme, že je více typů otázek a každá má svůj vlastní typ odpovědi. Každý typ má svou vlastní třídu zodpovědnou za zobrazení. U odpovědí v databázi ukládáme zda se jedná o referenční odpověď. Podle toho můžeme poznat zda se jedná o odpověď zadanou studentem, nebo odpověď referenční zadanou vyučujícím. Tyto třídy si tuto vlastnost budou umět zjistit a následně své zobrazení upravit. Použití tříd tak bude i na jiných místech v portálu.

QuestionTask je třída, která má za úkol vykreslit úkol otázky – pro připomenutí úkol je krátký textový komentář, který stručně shrnuje co má student v otázce dělat. Komponenta bude obsahovat nadpis „Úkol“ a následný text. Obdobně třída *TeachersFeedback*, která bude zobrazovat textový komentář od vyučujícího u otázky. Komponenta také zobrazí celé jméno autora komentáře a kontakt na něho.

Posledními komponentami jsou třídy *QuestionDetail* a *QuestionDemoDetail.* Obě se starají o zobrazení detailu otázky v testech. Pro třídu QuestionDetail odpovídá orbazovka zachycena na drátovém modelu na obrázku 2.5 a třída QuestionDemoDetail o obrazovku 2.7. Komponenty jsou složeny z některých výše uvedených komponent. Třídy QuestionDetail a QuestionDemoDetail mají v konstruktoru předány továrny komponent, které obsahují.

2.2.3 Presentery

Presenter je třída, která představuje nějakou konkrétní stránku ve webové aplikace. V architektuře MVC tuto roli zastává controller, ten ale může, často řešit logiku i více stránek najednou. Nette framework podporuje architekturu MVP, presenter se bude starat o logiku jedné stránky [19].

Třídy *DemotestPresenter, SemesterTestPresenter* a *ExamPresenter* se starají o vykreslení stránek s přehledy absolvovaných testů. Tyto přehledy dále

odkazují na náhledy testů a detail otázky, proto presentery musí zajistit handle metody, které po zavolání zobrazí tabulku otázek z testu a komponentu s informacemi o otázce. Tvorba těchto datagridů a komponent bude zajištěna patřičnými továrními metodami, které budou zaregistrovány v konfiguračních souborech jako *Service* a bude je tak možne vložit do presenterů.

2.2.4 Továrny

Tovární třídy se používají k tvorbě nových objektů, slouží k odstínění klienta od této tvorby [20]. Nette framework umí jednodušší továrny vygenerovat, ale Složitější si programátor musí napsat sám. V Nette tyto továrny zaregistrujeme jako služby, které pak můžeme pohodlně vkládat do jiných tříd.

Továrními třídami z modelu 2.8 jsou *AssignmentFactory, AnswerFactory, QuestionTaskFactory, TeachersFeedbackFactory*, které mají za úkol vytvořit jednodušší dílčí komponenty, které jsou dále použity. Továrny *QuestionDetailFactory* a *QuestionDemoDetailFactory* pak tvoří komponenty náhledů otázek. Posledním typem továren v modelu jsou třídy dědící od rozhraní *IDatagridFactory*. Toto rozhraní je poskytnuto balíčkem (knihovnou) Ublaboo/datagrid, který poskytuje řadu metod pro tvorbu datagridu. Od tohoto rozhraní dědí továrny *DemotestDatagridFactory, SemesterTestDatagridFactory, ExamDatagridFactory, QuestionsDatagridFactory* ve kterých budou implementovány metody pro tvorbu příslušných datagridů.

2.2.5 API pro komunikaci s databází

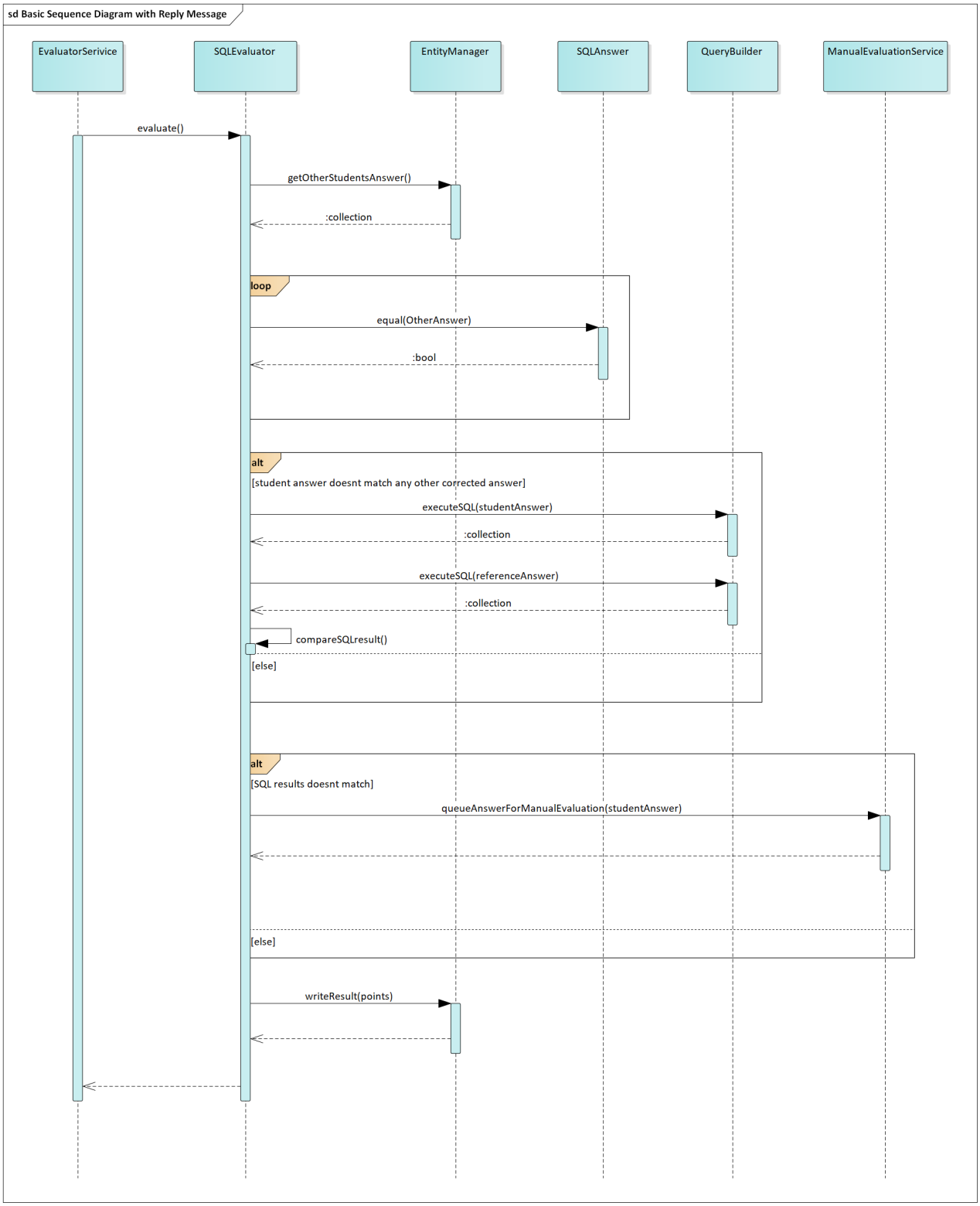
Komunikace s databází je zprostředkována třídou *EntityManager*. Přes tuto třídu zapisujeme i čteme data z databáze. Protože tato třída nám při čtení napřed vrací instance objektů repository, nad kterými až následně voláme query pro získání dat, kód této komunikace je vhodné někam „uklidit“. K tomu slouží pomocné třídy *TestDataloader, QuestionDataloader*, kde je prováděno volání metod nad EntityManagerem pro udržení čitelnějšího kódu aplikace.

2.3 Návrh implementace vyhodnocování

Třídy, které budou vykonávat vyhodnocování testů v portálu byly již navrhnuty v práci Andriiho Plyskach [2]. Nicméně implementování logiky vyhodnocování nebylo ještě realizováno. Tato část práce stručně seznámí s dosavadním návrhem a pokusí se navrhnout jak nad tímto návrhem realizovat logiku vyhodnocování.

Třída *AnswerEvaluator* je rodičovská třída, která poskytuje obecené rozhraní pro její děti, které mají za úkol vykonávat opravu odpovědí. Pro každý typ odpovědi je určena jedna třída poděděná od třídy AnswerEvaluator. Tyto třídy vytváří služba *EvaluatorService*. V konstruktory těchto tříd je jim předána instance třídy Answer, která obsahuje informace dané studentovi odpovědi a dále je předána instance třídy EntityManager. Jak již bylo zmíněno v předchozích částech testu, EntityManager slouží ke komunikaci s databází. Každá podtřída třídy AnswerEvaluator musí obsahovat metodu evaluate(), tato metoda nepřijímá žádné parametry a vrací void. Z tohoto důvodu se bude muset v této metodě provést načtení všech potřebných dat k vyhodnocení odpovědi (jako jsou například odpovědi dalších studentů na totožnou otázku a referenční odpovědi). Dále se v této metodě vykoná algoritmus vyhodnocení otázky, těmto algoritmům se věnovala kapitola 1.4 a následně příloha C. Metoda nevrací žádný parametr, a proto tato metoda musí také provést zápis výsledků (bodový zisk studenta z otázky).

Následující text se věnuje popisu konceptu volání funkce evaluate() zachyceném na obrázku 2.9. EvaluatorService vytvořil třídu SQLEvaluator, v konstruktoru této třídě předal instance tříd SQLAnswer a EntityManager. Následně zavolal metodu evaluate(). Odpověď je dle algoritmu vyhodnocení otázky typu SQL napřed porovnána s jinými již opravenými odpověďmi na tuto otázku. Pro získání těchto odpovědí si je třída SQLEvaluator bude muset napřed vyžádat od třídy EntityManager. Dále se opravovaná odpověď ve smyčce porovná se získanými odpověďmi. Pro porovnání dvou instancí třídy SQLAnswer by bylo nejlepší využit metodu, která zjsití zda vyplněná část studentem je shodná. Nehledáme úplnou identitu těchto instancí. Pokud není nalezena shoda se žádnou jinou již opravenou odpovědí, algoritmus se pokusí provést SQL dotaz nad databází otázky. K tomu je potřeba nějaký *QueryBuilder*, který dotaz provede a zároveň ověří, že dotaz neobsahuje nějakou variantu SQL injection. QueryBuilder také provede dotaz z nějaké z referenčních odpovědí k této otázce. Následě dojde k porovnání výstupů těchto dotazů. Pokud nejsou shodné nastane poslední alternativa v diagramu, ve které se otázka zařadí do fronty pro manuální opravu. Posledním krokem metody evaluate() je zápis výsledků, ten se opět provede přes EntityManagera. V případě kdy je otázka zařazena do manuální opravy, při zápisu výsledku se zatím zapíše nula.



Obrázek 2.9: Sekvenční diagram: Vyhodnocení SQL otázky

Při pokusu o realizaci se ukázalo, že backendová implementace není zatím plně funkční. Při komunikaci s databází často dochází k chybě a jsou i některé nedostatky v návrhu. Tento fakt byl konzultován s tvůrcem Andriim Plyskach, který konstatoval, že na backendu a obecně novém testovém modulu plánuje dále pracovat v rámci své diplomové práce a některé části nejsou zatím použitelné. Z tohoto důvodu tento návrh zůstane pouze konceptem, jako odrazový můstek pro budoucí možné použití.

2.3.1 Úprava třídního rozhraní

Tato kapitola poskytne stručný návrh na úpravu třídního rozhraní tříd, jež mají na starost vyhodnocování studentských odpovědí.

V kapitole 2.3 lze vidět, že metoda evaluate() pro vyhodnocení otázky potřebuje další závislosti. Třídě SQLEvaluator, ale i jiným třídám, které mají na starost vyhodnocování odpovědí, bude třeba v konstruktoru předat QueryBuilder a službu ManualEvaluationService.

Dále lze vidět, že metoda provádí spoustu dotazování dat na EntityManager. Toto dotazování nemusí vést na přehledný kód, proto navrhuji část komunikace s API implementovat jinde, ve třídách modelové vrstvy. EntityManager je schopen vracen tzv. repositories, což jsou třídy realizující metody pro správu dat jedné konkrétní entity. Do těchto tříd můžeme implementovat metody vlastní. Přes EntityManager tak dostaneme vyžádanou repository, z které zavoláme novou metodu. Druhou variantou je implementovat novou třídu modelové vrstvy a tam řešit tyto operace, závislost na ni pak předat třídám vyhodnocující odpovědi. Další výhodou tohoto řešení je dodržení principu DRY (don't repeat yourself). Metody pro získání potřebných dat pro vyhodnocení odpovědi půjde využit ve všech typech tříd vyhodnocující odpovědi.

Poslední úpravou, kterou navrhuji na zvážení, je změna návratového typu metody evaluate(). V současné době metoda vrací void. Metodu, která nepřijímá parametry a „nic“ nevrací je obtížné testovat. Navrhuji, aby metoda například vracela bodový zisk z otázky. To by vedlo na další zlepšení, kdy by se o zápis bodů mohla starat jiná třída. Metoda evaluate() by pak měla opravdu jen princip vyhodnotit odpověď.

Kapitola 3

Realizace

Následující text se věnuje převzetí projektu, komplikacím které ho provázeli a jejich řešení. Dále je stručně popsána implementace na ukázkách ze zdrojového kódu. Následuje kapitola, která obsahuje stručné teoreticé shrnutí o použitých technologiích. Poté se text věnuje teorii testování a následně popisuje jak byl testován a následně odladěn vytvořený prototyp. Poslední kapitola shrnuje budoucí nasazení projektu a budoucí rozvoj.

3.1 Vývojový proces

Kapitola podává teoretický základ o vývojových procesech a nakonec popisuje jakým procesem se postupovalo při tvorbě frontendového řešení, kterým se tato práce zabývá.

Vývojový proces nebo také model životního cyklu vývoje softwaru je množina aktivit popisující jak bude postupováno při tvorbě softwaru a jeho případné podpoře. Každy proces by měl obsahovat specifikaci požadavků co má systém dělat. Rozhodnutí o volbě architektury a designu pro systém. Samotnou implementace – vlastní výrovba systému. A zvalidnění výstupu, ověření, že systém dělá co má [21]. Rozlišujeme následující základní přistupy:

- *Vodopád* – Fáze vývoje jsou oddělené, vždy se pracuje jen na jedné fázi. Zpravidla v pořadí analýza požadavků, design softwaru, implementace, testování, údržba. Výhodou je jasně definovaný plán, z kterého plyne predikovatelnost zdrojů a snadná koordinace práce. Nevýhodou pak je nutno chápat co se chce již na začátku.

- *Iterativní* – Oproti vodopádu existuje několik verzí produktu, jednotlivé verze se dělají vodopádem. Zákazník tak může dostat přístup k jednotlivým verzím a mít lepší přehled o postupu. Nevýhodou zůstává nutnost chápání, cose chce, již na začátku.

- *Agilní* – Oproti iterativnímu přistupu má mnohem kratší iterace, ale jednotlivé verze nejsou vždy produkční. Výhodou je rychlá reakce na změny. Nevýhoda je ve velkých nárocích na intenzivní a souvislou práci celého týmu.

Při tvorbě praktické části této práce se nedrželo strikně žádného z uvedených přístupů. Problematika práce byla obsáhlá a já nebyl tolik zkušený v návrhu a implementaci frontendu, proto by při striktní volbě vodopádu mohlo dojít k nedostatkům v práci.

Vývojový proces použitý bych nazval jako obousměrný vodopád. Začalo se analýzou a pokračovalo se na návrh, když byly ale zjištěny nějaké nejasnosti při návrhu, vrátilo se zpět k analýze a doplnili se důležité informace. Při implementaci protoypu jsem narazil na jistá omezení s zanořením více komponent do sebe, vrátil jsem se tedy k návrhu a upravil jej, poté jsem realizoval, dle upraveného návrhu. Text této práce tak obsahuje v analýze všechny potřebné informace, které jsme potřeboval pro návrh, a ten je upraven, aby odpovídal realizačním omezením.

3.2 Převzetí projektu

Kapitola se zabývá problémy, které bylo nutné vyřešit při navázání na implementaci nového testového modulu. Následující text obsahuje popis problému a jeho řešení. Zároveň řešení těchto problému bylo předáno dalším vývojářům portálu, kdyby se některý problém vrátil.

3.2.1 Chybějící APCu driver

Rozšíření APCu slouží pro ukládání PHP proměnných v mezipaměti [22]. Je potřeba pro správný chod Doctrine v projektu. Portál DBS je vyvýjen ve virtuálním prostředí automaticky nakonfigurované softwarem Vagrant. APCu driver je třeba doinstalovat v tomto prostředí. Samotná instalace se provede následovně:

```
sudo su
apt install php7.2-apcu
```

Tato instalace v projektu často končila chybou o nedostupnosti archivů. Bylo třeba tedy pročistit apt metadata, natáhnout nové package listy a provést update všech php balíčků. Lze tak učinit například následujícím způsobem, s kterým přišel Pavel Jordán.

```
sudo apt-get clean
cd /var/lib/apt
sudo mv lists lists.old
sudo mkdir -p lists/partial
sudo apt-get clean
sudo apt-get update
```

A poté už jen stačí znovu provést instalaci uvedenou výšše. Každý uživatel si ale musí instalaci APCu driveru provést sám ve svém prostředí.

3.2.2 Nepřístupnost php console v projektu

V projektu je využito řady rozšíření pro Nette framework. Konkrétně balíček Contributte\Console do projektu integruje nástroj symphony\console [23]. Tento nástroj umožnuje spouštět z konzole předvytvořené příkazy, které se mohou hodit při vývoji. Balíčkem Nettrine\ORM pak do této konzole přidává nástroje spojené se správou databáze a namapovaných entit za pomoci Doctrine (ORM technologie).

Pokus o zavolání těchto příkazů přes `php bin\console` ovšem končí neúspěšně, uživatelovi je zobrazena chyba o chybějícím konfiguračním souboru pro produkci. V lokálním prostředí se tedy nepoužívá tento přístup. Je třeba příkazy z `php bin\console` předat dalšímu nástroji.

Tímto nástrojem je taskrunner realizovaný pomocí technologie Robo.li. Pokud chceme přes tento taskrunner volat například příkaz `orm:validate-schema` je třeba do souboru „RoboFile.php“ implementovat následující funkci:

```
//Validates orm shcema.
public function ormValidate(string $env = 'local'): void
{
    $this->taskExec("php␣bin/console␣orm:validate-schema")
        ->env(['APP_ENV' => $env])
        ->run();
}
```

Poté již stačí zavolat `./robo.sh orm:validate` a příkaz se normálně provede. Je možné uvést parametr tohoto příkazu pro nastavení v jakém prostředí se má příkaz vykonat, bez uvedení parametru se příkaz aplikuje na lokální prostředí. V mé implementaci je tento taskrunner rozšířen pro příkazy:

- `orm:validate-schema`
- `orm:schema-tool:create`
- `orm:schema-tool:update`
- `orm:schema-tool:drop`

3.2.3 Úprava schématu

Zprvu přidané příkazy balíčkem Contributte\Console a Nettrine\ORM nefungovali správně. Nebylo tak možné použít příkaty pro validaci schématu a jeho případnou úpravu. Chyba spočívala v neschopnosti volaných skriptů nalézt namapované entity. Bylo třeba upravit cestu k namapovaným entitám v konfiguračním souboru „Config.neon“.

```
orm.annotations:
   # paths to entity classes
   paths:
       - %appDir%/Modules/NewTest/Database
   #   - App/Modules/NewTest/Database/
```

Výšše je uvedena část konfiguračního souboru s částí konfigurující Doctrine. Předposlední řádek je cesta nová, poslední řadek je zakomentovaná cesta původní, jenž způsobovala chybu.

Po této opravě byly již příkazy schopné nalézt namapované entity. Schéma ovšem nešlo upravit, neboť nebylo validní. Validace schématu lze ověřit zavoláním příkazu `orm:validate-schema`. Tento příkaz hlásil chybu „Invalid forgei key action: SET NULLS“ na jednom řádku. Dle dokumentace nic jako tag *SET NULLS* neexistuje, pravděpodobně se tedy jednalo o syntaktickou chybu. Tag byl nahrazen za známy tag *SET NULL*, který je používán na mnoha dalších místech v implementaci. Poté již bylo schéma dle příkazu validní a šlo použít i příkazy pro úpravu schématu.

3.3 Tvorba komponent

Následující text na ukázkách zdrojového kódu z implementovaného prototypu vysvětluje základní principy tvorby komponent. Uvedené třídy v tomto textu byly již popsány v kapitole 2.2.

V následující ukázce kódu je zachycen konstruktor třídy TeachersFeedback, tato třída má za úkol zobrazit zpětnou vazbu učitele na studentovu odpověď. Aby tak mohla učinit, potřebuje informace o jméni opravujícího, kontaktu na něho a komentáři, který k otázce napsal. Tyto informace jsou předány instanci třídy při jejím vytvoření.

```
public function __construct(string $teachersComment, string
    $evaluatorName, string $email )
{
    $this->teachersComment = $teachersComment;
    $this->name = $evaluatorName;
    $this->email = $email;
}
```

Klíčovou metodou pro třídy, které dědí od třídy Control je metoda render(). Ta má za úkol nastavit šablonu komponenty. Následně jí předat data, která se mají v šabloně zobrazit a šablonu vykreslit voláním render() na šablonu. V některých případech lze v metodě render() též provádět nějakou práci s daty před vykreslením šablony.

```
public function render()
{
    $this->template->setFile(__DIR__ . '/TeachersFeedback.latte');
    $this->template->teachersComment = $this->teachersComment;
    $this->template->email = $this->email;
    $this->template->name = $this->name;
    $this->template->render();
}
```

Aby jsme mohli komponentu pro zpětnou vazbu učitele vytvářet pohodlně, napíšeme na ní tovární metodu. V tomto případě, kdy je tovární metoda jednoduchá, lze jí vygenerovat Nette frameworkem. Stačí nadeklarovat její interface jako na následující ukázce. Továrnu následně zaregistrujeme jako službu v konfiguračních souborech, to nám umožní snadné dependency injection do dalších tříd.

```
interface TeachersFeedbackFactory
{
    public function create(string $teachersComment, string
        $evaluatorName, string $email): TeachersFeedback;
}
```

V předchozí části byly ukázány klíčové aspekty tvorby komponenty pro zobrazování zpětné vazby učitele. Nyní se zaměříme na komponentu, která využívá tu předchozí. Další ukázka zachycuje konstruktor tentokrát třídy QuestionDetail. Tato třída zobrazuje detail otázky studentovi. K tomu potřebuje komponenty, které jsou její součástí. Je třeba jí tedy v konstruktoru předat potřebné továrny, také lze vidět, že je jí předáván QuestionDetailDataloader. To je třída z modelové vrstvy, která má za úkol z databáze dostat data, na kterých závisí její dílčí komponenty.

```
public function __construct(Answer $answer,
    QuestionDetailDataloader $questionDetailDataloader,
    QuestionTaskFactory $questionsTaskFactory, AnswerFactory
    $answerFactory, TeachersFeedbackFactory
    $teachersFeedbackFactory)
{
    $this->answer = $answer;
    $this->questionDetailDataLoader = $questionDetailDataloader;
    $this->questionsTaskFactory = $questionsTaskFactory;
    $this->answerFactory = $answerFactory;
    $this->teachersFeedbackFactory = $teachersFeedbackFactory;
}
```

Poslední ukázka zachycuje funkci třídy QuestionDetail. Tato metoda slouží pro vytvoření komponenty TeachersFedback v komponentě QuestionDetail. V metodě se načtou potřebná data pro tvořenou komponentu pomocí třídy QuestionDetailDataLoader, následně je použita továrna popsaná výšše pro tvorbu instance třídy TeachersFeedback. Metoda tedy vrací potomka třídy Control.

```
public function createComponentFeedback($feedName)
{
    $teachersComment = $this->questionDetailDataLoader
        ->getTeachersCommentByAnswer($this->answer);
    $evaluator = $this->questionDetailDataLoader
        ->getEvaluatorByAnswer($this->answer);
    $name = $evaluator->getFirstname() . "␣" .
        $evaluator->getLastname();
    $email = $evaluator->getEmail();

    $feedback =
        $this->teachersFeedbackFactory->create($teachersComment,
        $name, $email);
    return $feedback;
}
```

Presentery fungují obdobně jako komponenty. Mají také vlastní šablonu, a pokud v ní chtějí zobrazit nějakou komponentu musejí jí vytvořit metodou typu createComponent.

3.4 Použité technologie

Kapitola poskytuje teoretický základ pro technologie, které se používají při vývoji portálu DBS a v rámci vývoje prototypu s nimi bylo pracováno.

3.4.1 PHP

PHP (PHP: Hypertext Preprocessor) je skriptovací jazyk běžící na straně serveru (server-side) speciálně navržený pro potřeby tvorby webových stránek a aplikací. díky své široké a jednoduché podpoře je nainstalován a podporován na většině dostupných hostingů. PHP má otevřený zdrojový kód [24] [25].

3.4.2 Vagrant

Vagrant pomocí dalšího nástroje na virtualizaci, například Virtual Box z konfiguračního souboru Vagrantfile spustí na našem zařízení další virtuální stroj. V tom pak může běžet námi psaná aplikace. Pomocí sdílené složky mezi strojem, kde píšeme kód a kde běží aplikace jsou změny ihned promítány do aplikace. Vývojáři vagrant šetří čas při konfiguraci prostředí nového projektu [26].

3.4.3 Nette

Nette je PHP framework pro tvorbu webových aplikací. Webové aplikace se v Nette frameworku skládají ze samostatně požitelných komponent a presenterů. Framework využívá architektury MVP, dále popsané v 1.5. Nabízí skoro uniká Autorem Nette je David Grudl a dále je vyvýjeno organizací Nette Foundation. Je pro něho také dostupná celá řada rozšíření [27] [28].

3.4.4 Nettrine

Jedná se o knihovnu, která je dále složena z dalších balíčků, jejichž cílem je integrace Doctrine 2 do Nette framework [29]. Nový testový modul využívá právě tuto technologii k realizaci ORM.

3.4.5 Knihovny Contributte

Je sada knihoven pro Nette framework umožnují jednoduchou implementaci různých funkcí [30]. Například balíčkem Contributte\Translation je v portálu realizován překlad mezi jazyky. Contributte\Console pak integruje do projektu nástroj Symphony\Console, jehož využití je popsáno v 3.2.2.

3.4.6 Codeception

Codeception je PHP testing framework určen pro jednoduché psaní testů pro aplikaci [31]. Je založen na známém PHPUnit frameworku. Nabízí celou řadu testování jako je unit, integrační, data driven a další.

3.4.7 Robo.li

Je nástroj pro tvorbu automatických scriptů v objektovém PHP. Také obsahuje určitou sadu předem připravených spustitelných úkonů. Lze jednodušše přidat do projektu přes Composer [32]. V této práci byl využit pro napsání úkonů, které mají za úkol provádět operace nad ORM schématem.

3.4.8 Ublaboo\Datagrid

Pro tvorbu datagridů v projektu byla vybrána knihovna Ublaboo\Datagrid, nahradí tak na některých místech knihovnu o5\Grido. Dle srovnání na diskuzním fóru Nette framework má vývoj Ublaboo do budoucna větší plány a funkce, které nabízí jsou identické s o5\Grido [33]. Ublaboo také zvítězilo v rychlostních porovnáních [34].

3.5 Testování

Tato kapitola poskytuje krátké teoretické shrnutí o testování softwaru a následně popisuje jak byl testován prototyp nového frontendu zaměřující se na zobrazování vyhodnocených testů studentům v portálu DBS.

Testování je důležitou fází vývoje softwaru. Pomáhá zjistit chyby programu a nalézt nedostatky v návrhu. Proto by se s testováním mělo začít již v rané fázi vývoje, aby případné nedostatky mohli být adresovány co nejdříve. Software lze testovat na mnoho vlastností – funkčnost, stabilitu, bezpečnost, výkon a mnohé další [35]. Testování lze rozdělit dle způsobu realizace:

- *Manuální* – Testy provádí uživatel/tester, který postupuje dle připraveného scénáře a nejčastěji zadává vstupy programu přes GUI. V dnešní době je již toto testování ve spoustě oblastech na ústupu a je nahrazováno automatizovaným testováním.
- *Exploratory* – Opět se testuje manuálně, ale za jiným účelem a jiným způsobem. Již se nepostupuje dle předem připravených scénářů a cílem testování je objevení funkcí a vlastností systému.
- *Automatizované* – Software je testován jiným softwarem. Programátor zpravidla připravý sadu testů, program pak tyto testy vykonává a vyhodnocuje. Výhodou tohoto testování je časová úspora a opakovatelnost.

Testovat můžeme různé vlastnosti a v různých fázích vývoje. Pro různé případy jsou vhodné různé testy. Proto dále rozlišujeme testy na:

- *Unit testy* – Také zvané jako jednotkové, testují se základní prvky softwaru. Z pohledu objektově orientovaného programování je jednotkou obvykle třída, či konkrétní metoda. Každý testový případ by měl být nezávislý na ostatních.
- *Integrační testy* – Oveřují fungování jednotlivých tříd a komponent dohromady. Zpravidla se ještě netestuje celý sytém dohromady, jen jeho části.
- *Systémové testy* – z pohledu hierarchie se jedná o nejvyšší úroveň testování, která má za účel prověřit celkovou funkčnost softwaru. Testují se již všechny části dohromady.

Pro automatické testování se v portálu využívá PHP framework *Codeception*. Který umožnuje tvorbu Unit, integračních, systémových ale i dalších typů testů. Právě dalším typem testu, který se v portálu používá je kontrola na *Code style*. Vývojářů portálu je mnoho a každý může využívat odlišné formátování kódu. Tato kontrola má za účel ověřit, zda kód který chce vývojář do projektu nahrát splňuje požadované formátování kódu. Toto testování kódu vytvořil

Ing. Pavel Kovář v rámci své práce [36]. Mimojiné byla přidána i kontrola, zda implementované funkce vrací požadované datové typy apod. Při implementaci bylo využito právě těchto funkcí pro zajištění správného formátování kódu.

3.5.1 UX testování prototypu

Automatické testování frontendu je v současné době stále složité, protože je těžké zajistit, aby program rozpoznal správnost výsledku testu. Automatizace frontendu se tedy provádí hlavně na úrovní generování uživatelských vstupů programem a následný foto nebo video záznamem výstupů, tyto výstupy jsou pak předány člověku.

Protože většina tříd implementovaných v prototypu slouží k vykreslení komponent a zajištění interakcí mezi nimi na webu, zautomatizované testy typu jednotkové nebo integrační nejsou příliš vhodné. Implementace obsahuje i třídy z modelové vrstvy, ale z důvodu jistých nedostatků na backendu museli být jejich metody realizovány jako mockup. Rozhodl jsem se tedy výsledný prototyp testovat využitím UX testů.

UX testy jsou formou manuálních testů, kde testeři vykonávají úkoly, dle připraveného scénáře v aplikaci a následně hodnotí jejich funkčnost, obtížnost, intuitivnost [35] [37]. Jako testeři byla vybrána skupina studentů, kteří již absolvovali předmět BI-DBS a byli tak již seznámeni s portálem. Dále jsem pro ně připravil úkoly, které budou provádět v protypu a formulář, který následně vyplní.

3.5.1.1 Úkoly formuláře

V této části textu jsou popsány úkoly, které plnili testeři v rámci testovacího scénáře.

Úkol 0: Prohlédněte jak vypadá/funguje zobrazování absolvovaných testů studentem v současné verzi portálu DBS.

V tomto úkolu si testeři měli připomenout jak funguje zobrazování absolvovaných testů v současné době v portálu. Provedli tak přihlášením do portál DBS a následným proklikáním svým absolvovaných testů.

Úkol 1: V prototypu si otevřete nový testový modul pod kolonkou „Nové testy“ a následně zvolte položku „Vyhodnocené testy“.

Testeři měli za úkol se v prototypové verzi portálu dostat do nového testového modulu a následně přejít na své vyhodnocené testy.

Úkol 2: Zobrazte si úkoly v druhém pokusu zkoušky a následně si zobrazte detail vybrané otázky.

Cílem úkolu bylo ve vyhodnocených testech si zobrazit absolvované zkoušky, zobrazit náhled druhého pokusu zkoušky a poté si zobrazit detail k otázkám v tomto pokusu.

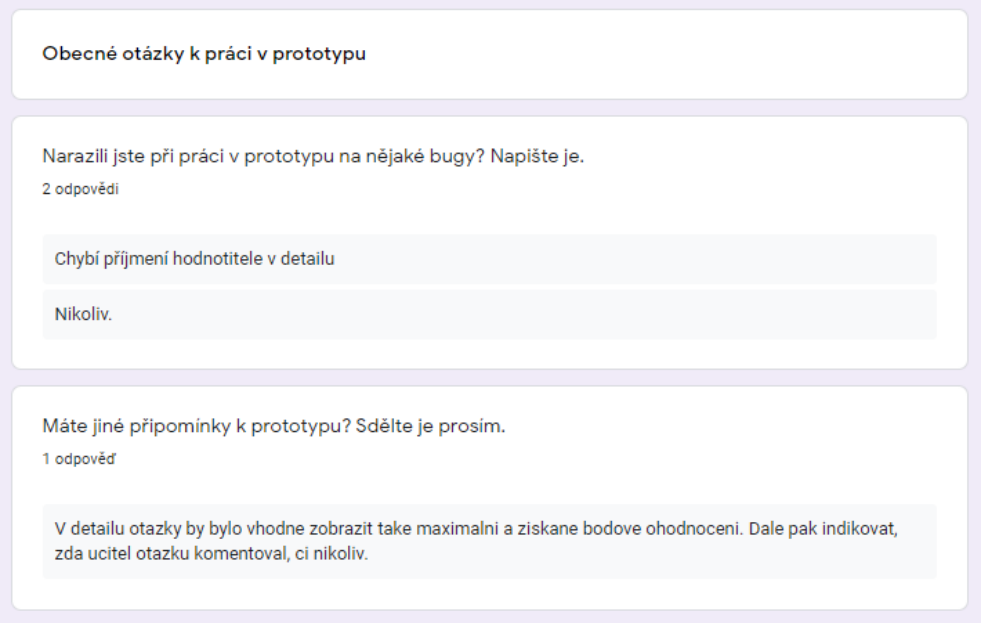
Úkol 3: Nalezněte demotest s nekratším a nejdelším časem na vypracování. Nalezněte demotest, který jste psali 1. 6. 2021 a otevřete si detail jeho otázek.

Úkol měl zejména otestovat použitelnost třídicí a vyhledávací funkce tabulek s testy. Dále si testeři měli zobrazit opět detail otázek u demotestů, aby se seznámili s odlišností od zobrazení detailu otázky v bodovaném testu.

3.5.1.2 Počty testerů

Dle Jakob Nielsenovi teorie z počátku devadesátých let je nejúčinější provádět krátké testovací scénáře s nízkým počtem testerů (například 5 testerů), neboť zásadní nedostatky se zpravidla projevý již u prvních testerů. Následné zvyšování testerů již nepřináší mnoho nových objevů. Jak ale později dodal, testování na vetším počtu uživatelů je potřeba, pro odhalení komplikovaných chyb v designu [38].

3.5.1.3 Výstupy testování



Obrázek 3.1: Formulář pro UX testování - vylepšení a chyby

Testováním si prošli tři studenti, veškeré jejich odpovědi s kontaktním emailem jsou dostupné na přiloženém paměťovém médiu. Dále je také na médiu odkaz na daný formulář, ukázka formuláře je také dostupná v příloze E.

Formulář se dotazoval testerů na intuitivnost a obtížnost jednotlivých úkolů. Všichni respondenti uvedli, že úkoly byly jednoduché a intuitivní. V posledních dvou nepovinných otázkách byla dána možnost testerům, aby nahlásili případné nalezené chyby nebo poskytli nápady na zlepšení. Výstup těchto odpovědí je zachycen na obrázku 3.1.

Nahlášenou chybou bylo chybějící přijmení vyučujícího v detailu otázky. Tato chyba byla ihned opravena. Nahlášeným podmětem pro zlepšení bylo zobrazovat získané a maximální body z otázky i v detailu otázky (obrazovka z obrázku 2.5), v současné době jsou body z otázky vidět jen v náhledu testu (obrazovka na obrázku 2.4). Toto vylepšení je určitě dobrý nápad a bude realizováno v budoucnu.

3.6 Integrace a budoucí rozvoj

Implementace frontendového prototypu, kterým se tato práce zabývala, plně navazuje na řešení backendu pro nový testový modul portálu DBS, jehož základy položil Bc. Andrii Plyskach ve své bakalářské práci [2]. Ten v rámci své práce vytvořil nový modul v projektu portálu DBS, ve kterém následně implementovat. Ve svém řešení jsem navázal a tuto implementaci a modul dále rozšiřoval.

Abych mohl pokračovat v rozšiřování, bylo třeba vyřešit některé problémy popsané v kapitole 3.2. Dle návrhu jsem implementoval jádro frontendu. V současné době je třeba doimplementovat komponenty pro zobrazování některých typů odpovědí, například *transformace, normalizace*. Implementace je na toto doplňení plně připravena, v továrních metodách stačí doplnit implementaci pro další druhy. Frontendový prototyp musel být v současné době realizován jako mockup z důvodu neodladěné API pro komunikaci s databází. V budoucnu bude tedy třeba prototyp propojit se zmíněným API. Třídy modelové vrstvy mají ve svých závislostech již přidaný *EntityManager*, který se k tomu bude používat. Ve zdrojovém kódu jsou přidány patřičné komentáře, kde je třeba provést odmockování. Komentáře obsahují tag *TODO*, takto je zle snadno najít například využitím funkcí IDE PhpStorm.

Využitím UX testů na prototypu se odhalily některé chyby, které byly následně opraveny. Také byl získán podmět k zlepšení, ten je zdokumentován na konci kapitoly 3.5.1. Toto zlepšení bude realizováno v budoucnu.

Své řešení jsem v průběhu implementace verzoval verzovacím systémem git, to je nyní dostupné v repozitáři newDBS ve větvy *bw_newQuestionsFrontend* na fakultním GitLabu. Prototyp je tedy připraven na případné rozšíření a následný merge do hlavní vývojové větve portálu DBS. Nový testový modul zůstal plně samostatný.

V průběhu tvorby prototypu a této práce jsem spolupracoval s vývojovým týmem portálu z předmětu BI-SP1 (tým 3, vedoucí týmu byl František Sciranka). V rámci této spolupráce jsem členům pomáhal jak začít rozšiřovat nový testový modul. Sepsal jsem manuál, jak se vypořádat s chyby popsanými v kapitole 3.2. Manuál je publikován na týmové wiki v Redmine. Dále jsem radil jak navrhovat frontend v nástroji Balsamiq Wireframes a následně tento návrh prezentovat. Seznámil jsem je, na jakých dalších částech nového modulu je třeba pracovat a dle tohoto podnětu si vybrali tvorbu frontendu pro psaní testů v portálu. Věřím tedy, že své znalosti předají opět dále a nový testový modul se bude dále rozšiřovat. V době započatí této práce také začínal rozvíjet nový testový modul Pavel Jordán, který se věnuje frontendu pro tvorbu otázek.

Závěr

Prvním cílem práce byla analýza současného řešení portálu DBS se zaměřením na vyhodnocení a zobrazení otázek z testů studentům. Na začátku bylo využito případů užití k zachycení funkcí systémů a k pochopení souvislostí mezi otázkami, testy a zadáními. Následovala analýza struktury otázek a testů s ohledem na provedené změny v novém testovém modulu. Algoritmus vyhodnocování otázek byl popsán a zachycen pomocí diagramů aktivit. Na základě zjištěných vlastností současného řešení byly stanoveny požadavky pro řešení nové (Dostupné v kapitole 1.6). Významnějším požadavkem bylo rozšíření zpětné vazby studentům u bodovaných testů.

V návrhu bylo využito drátěného modelu pro model stránek a komponent na webu pro nový frontend. Návrh adresoval požadavky stanovené na konci analýzy a povedlo se přijít s řešením zlepšujícím zpětnou vazbu. Dále tato část obsahovala návrh tvorby komponent pomocí komponentového systému Nette framework. Byla zachována samostatnost a nízká provázanost nového testového modulu. Poslední část se pak zaměřila na návrh implementace vyhodnocovacích algoritmů nad novým třídním schématem.

Kapitola Realizace popisuje vývojový proces použitý při vývoji funkčního prototypu frontendového řešení. Dále v kapitole 3.2 jsou poskytnuta řešení problému, které nastaly při převzetí projektu, na který tato práce navazuje. Tato řešení již pomohla s započatím vývoje novým vývojářům portálu a dalším studentům při tvorbě jejich závěrečných prací. Také je v této kapitole na ukázkách ze zdrojového kódu vysvětlen princip tvorby komponent.

Výsledný porotyp byl otestován UX testy. Testeři byli bývalí studenti předmětu BI-DBS, kteří již měli zkušenosti s původním řešením v portálu. Na základě testování byly odladěny chyby prototypu a byly získány nové podněty pro další zlepšování. Více detailů v kapitole 3.5.1.

Poslední kapitola se věnuje integraci prototypu do portálu DBS. Prototyp je připraven v repozitáři newDBS ve větvy bw_newQuestionsFrontend. Tato kapitola také popisuje, jaké věci je v prototypu třeba dodělat před plným nasazením, jednou z věcí je odmockování tříd z modelové vrstvy, neboť v současné

době nový testový modul nemá odladěné API pro komunikaci s databází. Také je popsána spolupráce s vývojovým týmem z předmětu BI-SP1, kteří již začali rozvíjet nový testový modul. Veřím, že budou schopni dále využít můj prototyp, ale případně i poznatky z analýzy a návrhu.

Literatura

[1] Úvod do MVP. *V: https://www.ackee.cz/blog/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://www.ackee.cz/blog/uvod-do-mvp-na-androidu/`

[2] Andrii, P.: *Refaktoring testové části backendu portálu dbs.fit.cvut.cz.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[3] dbs.fit.cvut.cz. [online], [cit. 2021-04-22]. Dostupné z: `https://dbs.fit.cvut.cz/`

[4] Project development with UML and Enterprise Architect. *V: https://www.sparxsystems.eu/ [online]*, [cit. 2021-04-22]. Dostupné z: `https://www.sparxsystems.eu/resources/project-development-with-uml-and-ea`

[5] Slavotínek, J.: *Webová komponenta na kreslení ER diagramů.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

[6] Fedor, T.: *Webová komponenta na kreslení ER diagramů II.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[7] Working with JSON. *V: : https://developer.mozilla.org [online]*, [cit. 2021-04-15]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON`

[8] Data Modeler. *V: https://github.com/ [online]*, [cit. 2021-04-15]. Dostupné z: `https://github.com/tfedor/datamodeler?fbclid=IwAR1veMWufKeW0qRTFHsyi2EHkx7v9NGbEQ5tmGpuRrDBFDI1pnWg7iuwx3w`

[9] Valenta, M.: Transformace konceptuálního modelu na relacní. *V: https://courses.fit.cvut.cz/ [online]*, [cit. 2021-05-24]. Dostupné z: `https://courses.fit.cvut.cz/BI-DBS/materials/slides/handles05-transformace.pdf`

[10] Normalizace databáze. *V: https://en.wikipedia.org/ [online]*, [cit. 2021-04-18]. Dostupné z: `https://en.wikipedia.org/wiki/Database_normalization`

[11] Kubiš, M.: *Rozšíření překladače relační algebry*. Praha: České vysoké učení technické v Praze, 2019.

[12] Architektura MVC: definice, struktura, frameworky. *V: https://www.rascasone.com/cs/blog/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky`

[13] AJAX. *V: https://cs.wikipedia.org/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://cs.wikipedia.org/wiki/AJAX`

[14] Snippet (úryvek). *V: https://www.seoprakticky.cz/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://www.seoprakticky.cz/slovnik-pojmu/snippet/`

[15] Analytická specifikace a její zpracování. *V: http://ecom.ef.jcu.cz/web2/ [online]*, [cit. 2021-06-01]. Dostupné z: `http://ecom.ef.jcu.cz/web2/download/teorie/04_analyticka_specifikace_a_jeji_zpracovani.pdf`

[16] Wireframe. *V:https://cs.wikipedia.orge [online]*, [cit. 2021-06-03]. Dostupné z: `https://cs.wikipedia.org/wiki/Wireframe`

[17] Komponenty a ovládací prvky. *V: https://doc.nette.org/cs/3.1/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://doc.nette.org/cs/3.1/components`

[18] Úvod do Domain-Driven Designu 2 – co je doménový model? *V: https://www.trigama.eu/cs/blog/ [online]*, [cit. 2021-06-03]. Dostupné z: `https://www.trigama.eu/cs/blog/detail/uvod-do-domain-driven-designu-2-co-je-domenovy-model`

[19] Presentery. *V: https://doc.nette.org/cs/3.1/ [online]*, [cit. 2021-06-01]. Dostupné z: `https://doc.nette.org/cs/3.1/presenters`

[20] Factory Method. *V: https://refactoring.guru/ [online]*, [cit. 2021-06-03]. Dostupné z: `https://refactoring.guru/design-patterns/factory-method`

[21] Hlavatý, M.: Softwarový proces. *V: https://moodle-vyuka.cvut.cz/course/view.php?id=3917 [online]*, [cit. 2021-06-03]. Dostupné z: `https://moodle-vyuka.cvut.cz/pluginfile.php/308910/course/section/46035/01_SoftwareProcess.pdf`

[22] APC User Cache. *V: https://www.php.net/ [online]*, [cit. 2021-06-06]. Dostupné z: `https://www.php.net/manual/en/book.apcu.php`

[23] Contributte/console. *V: https://github.com/ [online]*, [cit. 2021-05-30]. Dostupné z: `https://github.com/contributte/console`

[24] Co je to PHP? *V: https://www.artic-studio.net/slovnicek-pojmu/ [online]*, [cit. 2021-06-06]. Dostupné z: `https://www.artic-studio.net/slovnicek-pojmu/skriptovaci-jazyk-php/`

[25] PHP. *V: https://cs.wikipedia.org/wiki/ [online]*, [cit. 2021-06-06]. Dostupné z: `https://cs.wikipedia.org/wiki/PHP`

[26] Úvod do vagrantu. *V: https://zdrojak.cz/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://zdrojak.cz/clanky/uvod-do-vagrantu/`

[27] Proč používat Nette? *V: https://doc.nette.org/cs/ [online]*, [cit. 2021-06-06]. Dostupné z: `https://doc.nette.org/cs/3.1/why-use-nette`

[28] Nette Foundation. *V: https://github.com/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://github.com/nette`

[29] Nettrine. *V: https://github.com/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://github.com/nettrine`

[30] Contributte. *V: https://github.com/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://github.com/contributte`

[31] Codeception quicstart. *V: https://codeception.com/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://codeception.com/quickstart`

[32] Robo Modern Task Runner for PHP. *V: https://robo.li/ [online]*, [cit. 2021-06-19]. Dostupné z: `https://robo.li/`

[33] Benchmarking (v1). *V: https://ublaboo.org/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://ublaboo.org/datagrid/benchmarking`

[34] Jaký DataGrid použít – srovnání. *V: https://forum.nette.org/cs/ [online]*, [cit. 2021-06-09]. Dostupné z: `https://forum.nette.org/cs/26583-jaky-datagrid-pouzit-srovnani`

[35] Kitner, R.: Typy testování software (třídění testů). *V: https://kitner.cz/ [online]*, [cit. 2021-06-03]. Dostupné z: `https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/`

[36] Kovář, P.: *Automatizované testování webového portáludbs.fit.cvut.cz.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[37] UX - USER EXPERIENCE. *https://www.antee.cz/ [online]*, [cit. 2021-06-06]. Dostupné z: https://www.antee.cz/ux-testovani

[38] Number of test subjects. *V: https://en.wikipedia.org/ [online]*, [cit. 2021-06-22]. Dostupné z: https://en.wikipedia.org/wiki/Usability_testing

[39] SQL injection. *V: : https://portswigger.net/ [online]*, [cit. 2021-05-29]. Dostupné z: https://portswigger.net/web-security/sql-injection

Příloha A

Seznam použitých zkratek

AD Activity Diagram

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

BI-DBS Databázové systémy

BI-SP1 Softwarový týmový projekt 1

BI-SP2 Softwarový týmový projekt 2

ČVUT České vysoké učení technické v Praze

GUI Graphic User Interface

HTML Hypertext Markup Language

JSON JavaScript Object Notation

KOS Studijní informační systém

MVC Model-View-Controler

MVP Model-View-Presenter

ORM Object Relation Mapping

PHP PHP: Hypertext Preprocessor

RA Relační algebra

SQL Structured Query Language

UC Use case

UML Unified Modeling Language

Příloha **B**

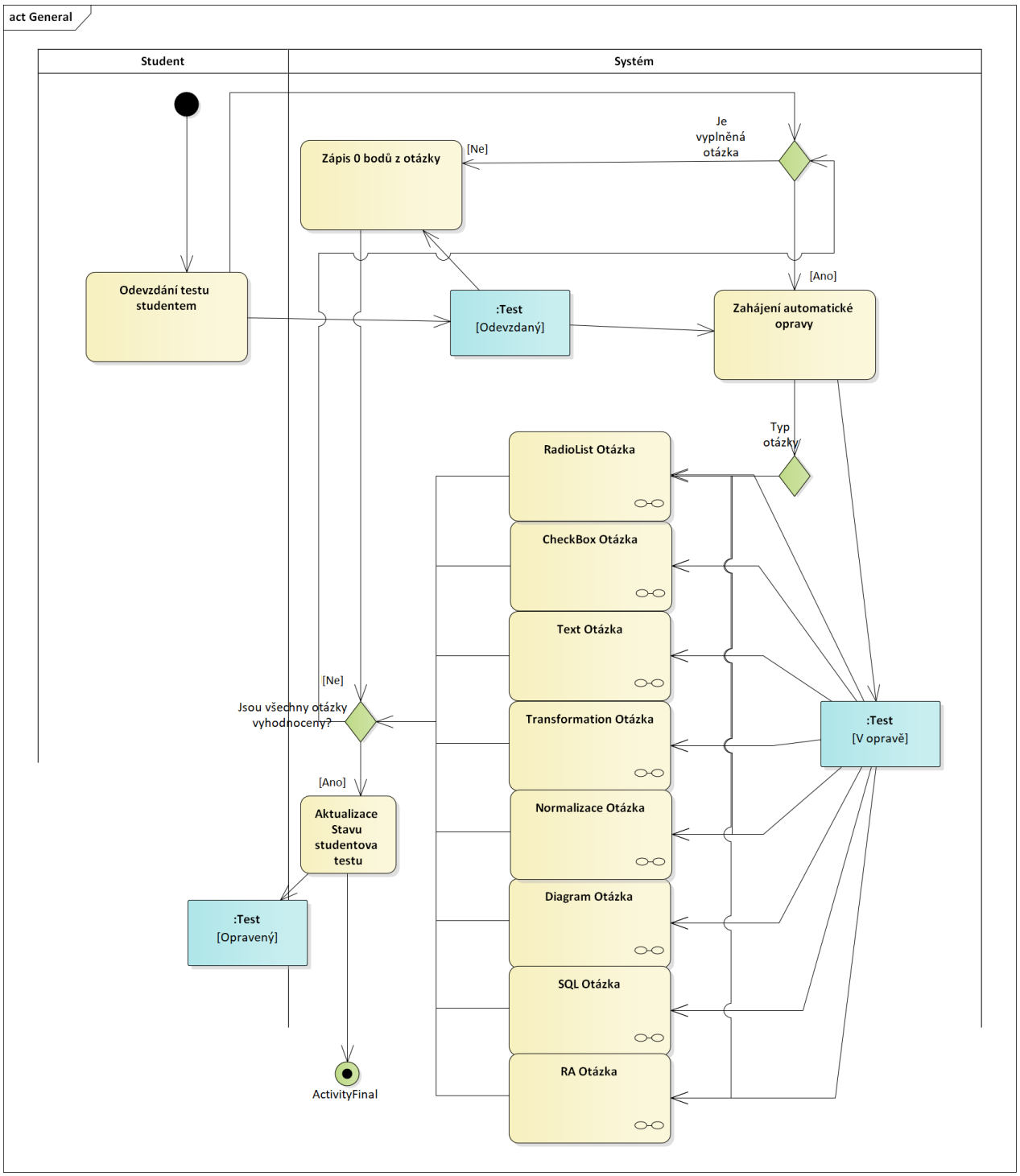
Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `BPtext.pdf` ..... text práce
- `Balsamiq Wireframes project` ..... projekt drátěného modelu
- `Enterprise Architect projects` ..... projekty použitých diagramů
- `Implementation` ..... implementace prototypu
- `LaTeX project` ..... zdrojová forma práce ve formátu LaTeX
- `UX testing` ..... záznamy z testování
  - `answers.pdf` ..... tabulka všech odpovědí z formuláře
  - `form.txt` ..... odkaz na formulář

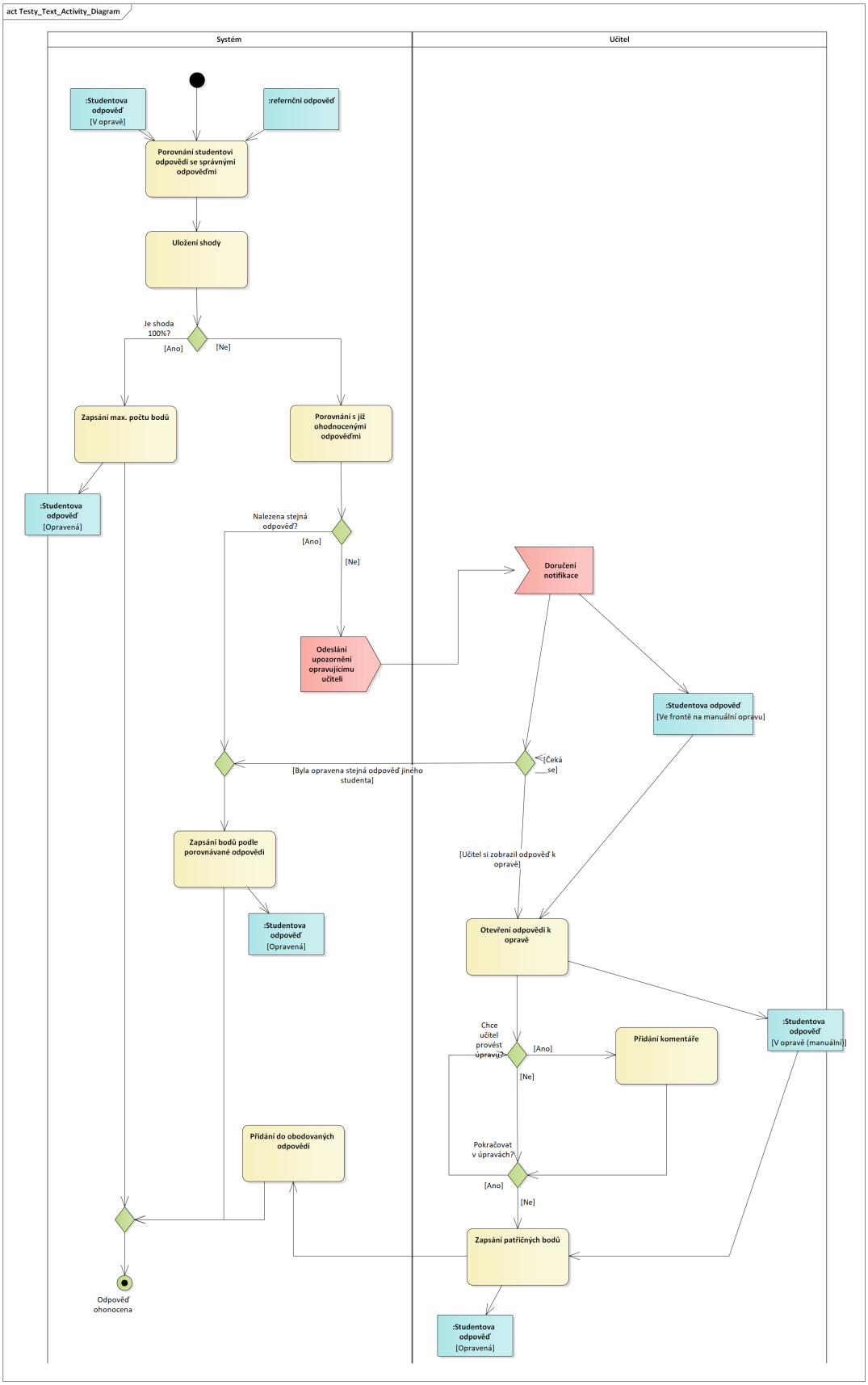
Příloha C

Aktivity diagramy vyhodnocování odpovědí

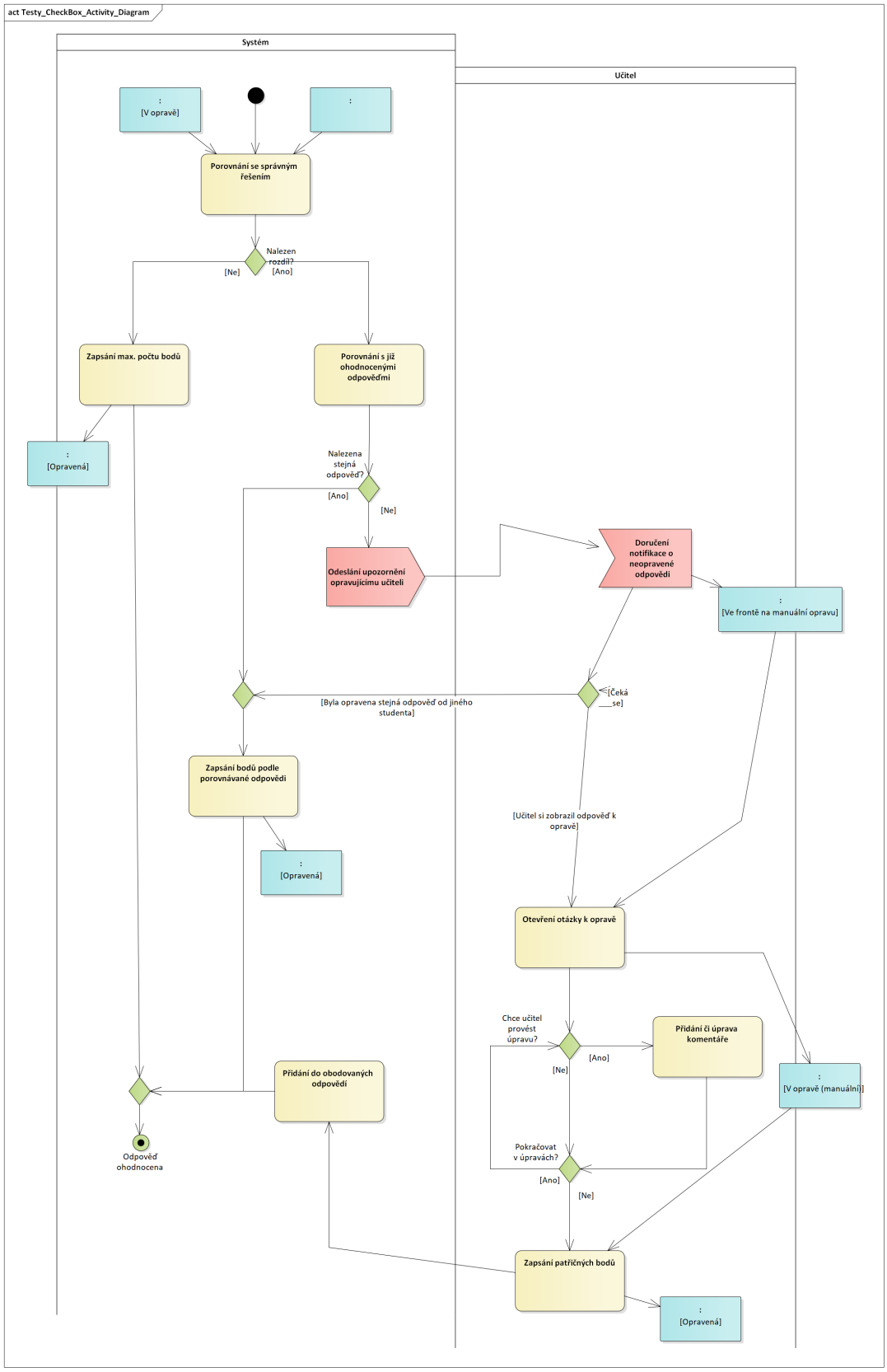
Příloha obsahuje diagramy aktivit zachycující algoritmy vyhodnocování otázek v portálu. Jsou dostupné také na přiloženém paměťovém médiu. Diagramy byly vytvořeny v rámci spolupráce s vývojovým týmem portálu z předmětu BI-SP1 v roce 2021. Členové týmu jsou:
Dana Suchomelová, František Sciranka, Matyáš Frank, Max Hejda, Samuel Švalich, Viktor Vysoký.



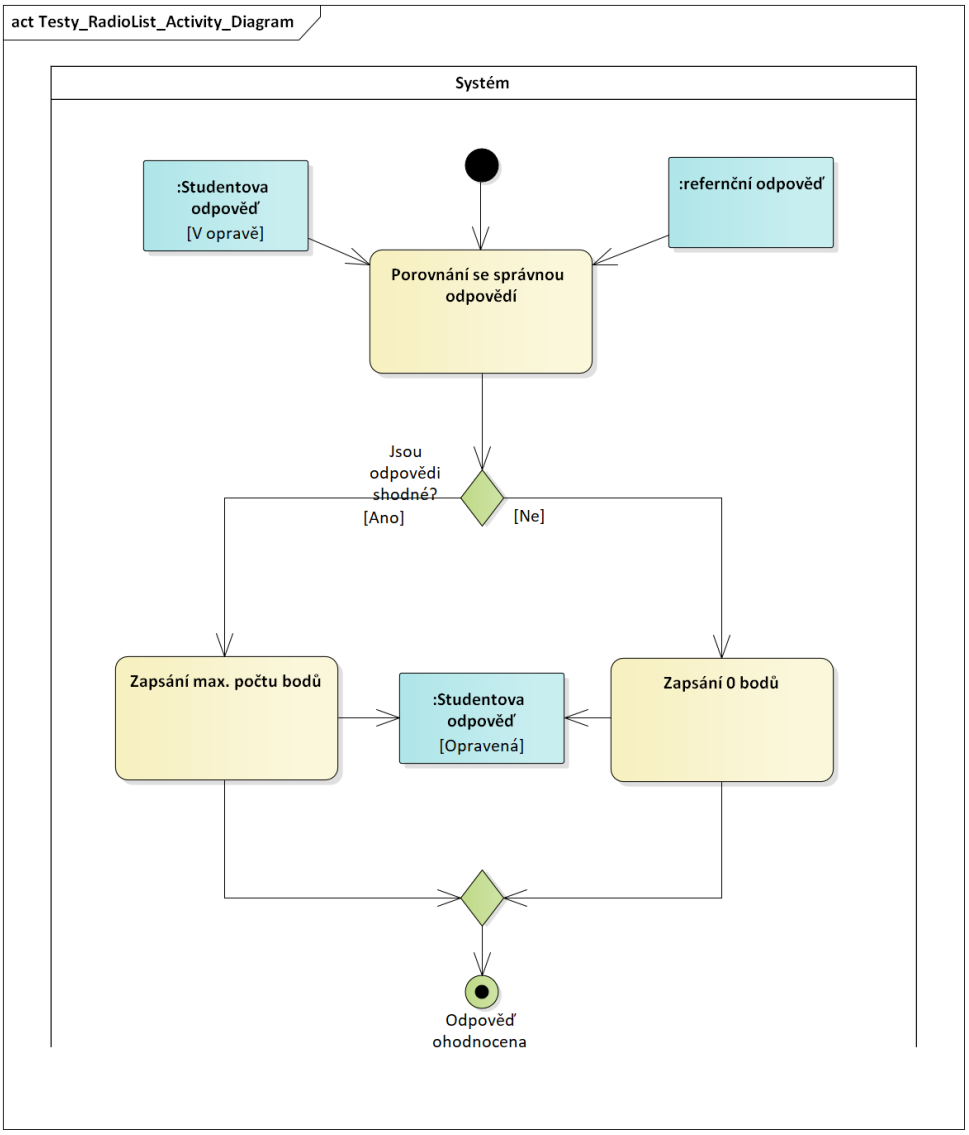
Obrázek C.1: Diagram aktivit: Obecné vyhodnocení odpovědi



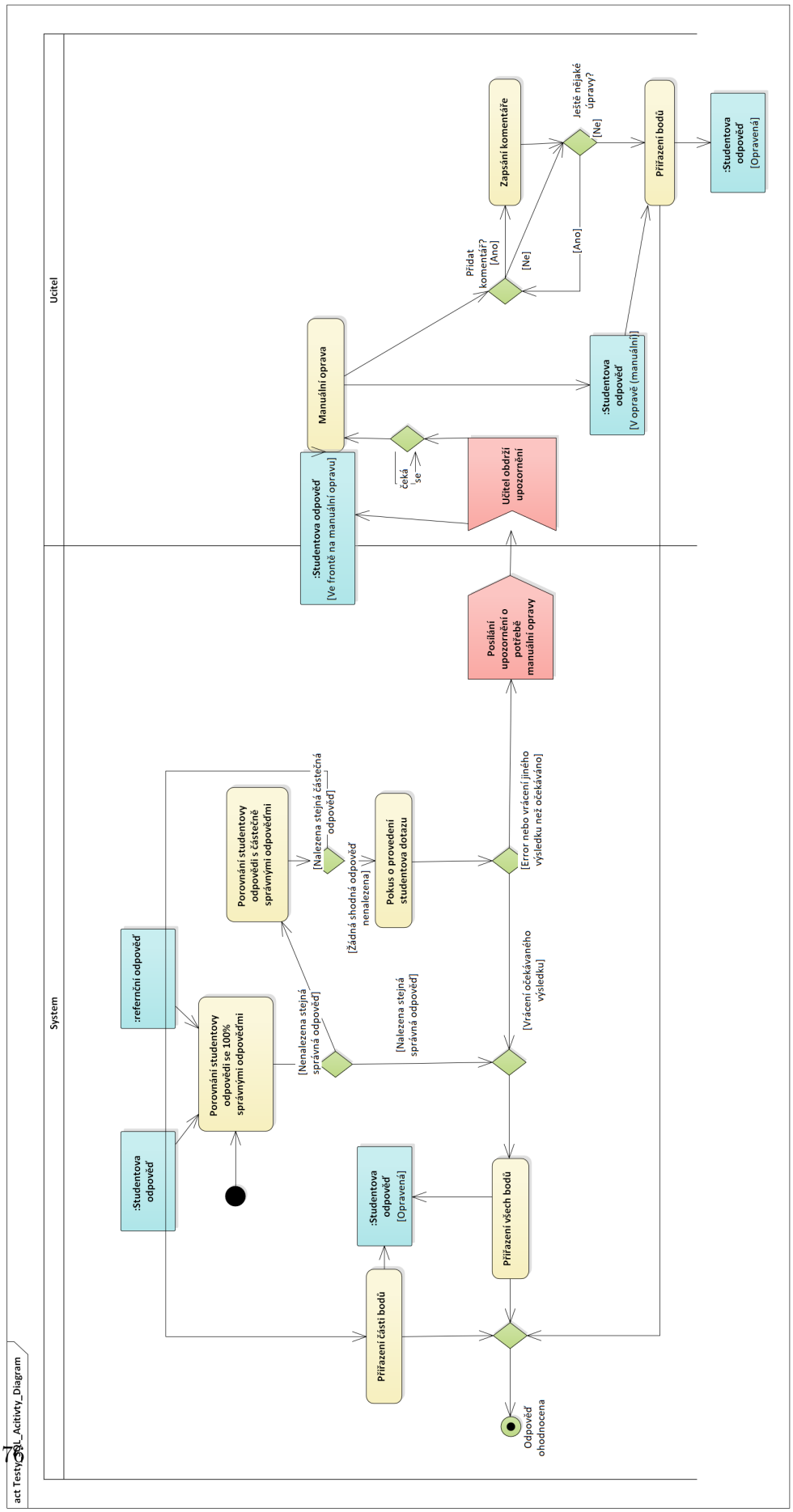
Obrázek C.2: Diagram aktivit: Vyhodnocení odpovědi - Text



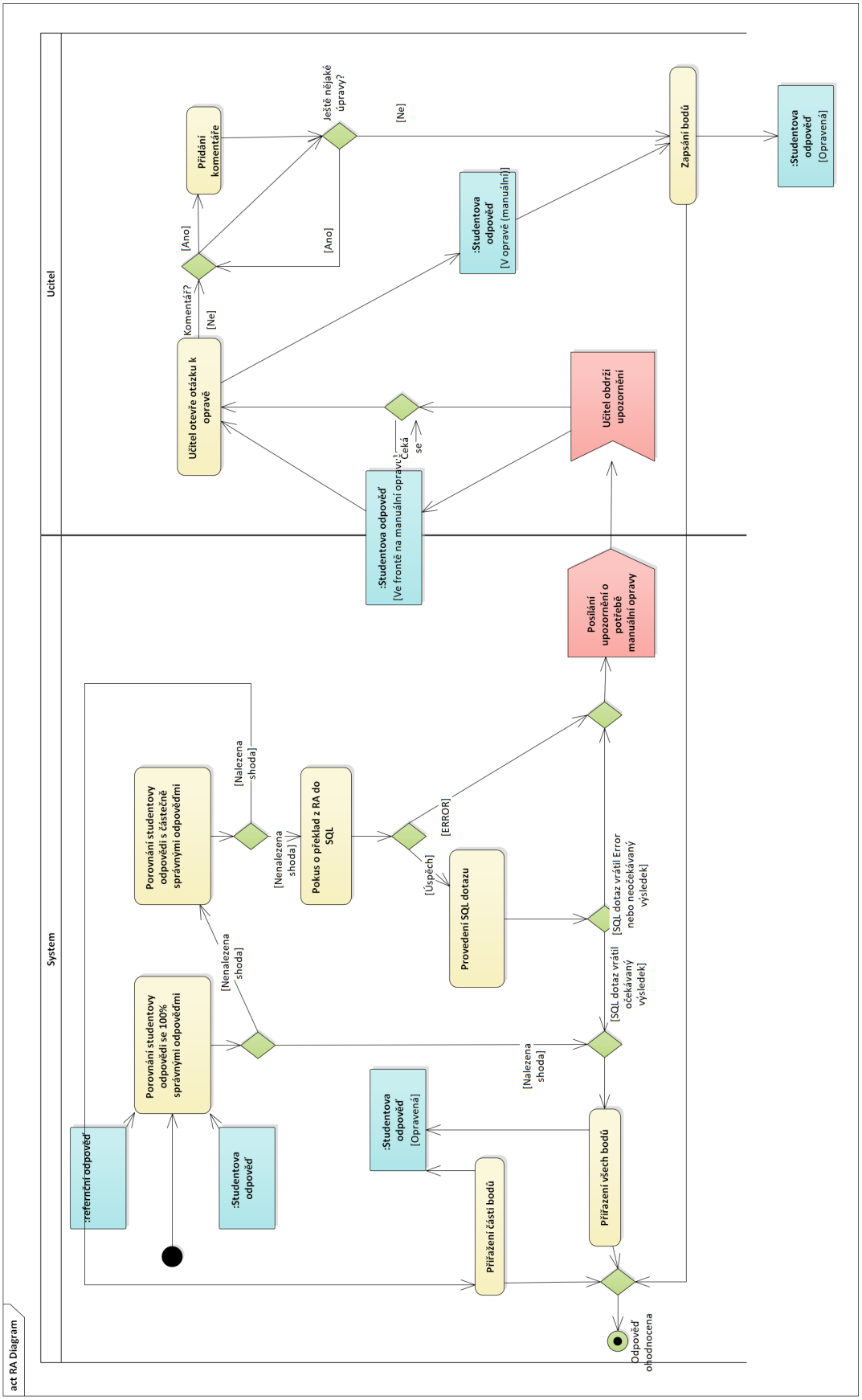
Obrázek C.3: Diagram aktivit: Vyhodnocení odpovědi - Checkbox



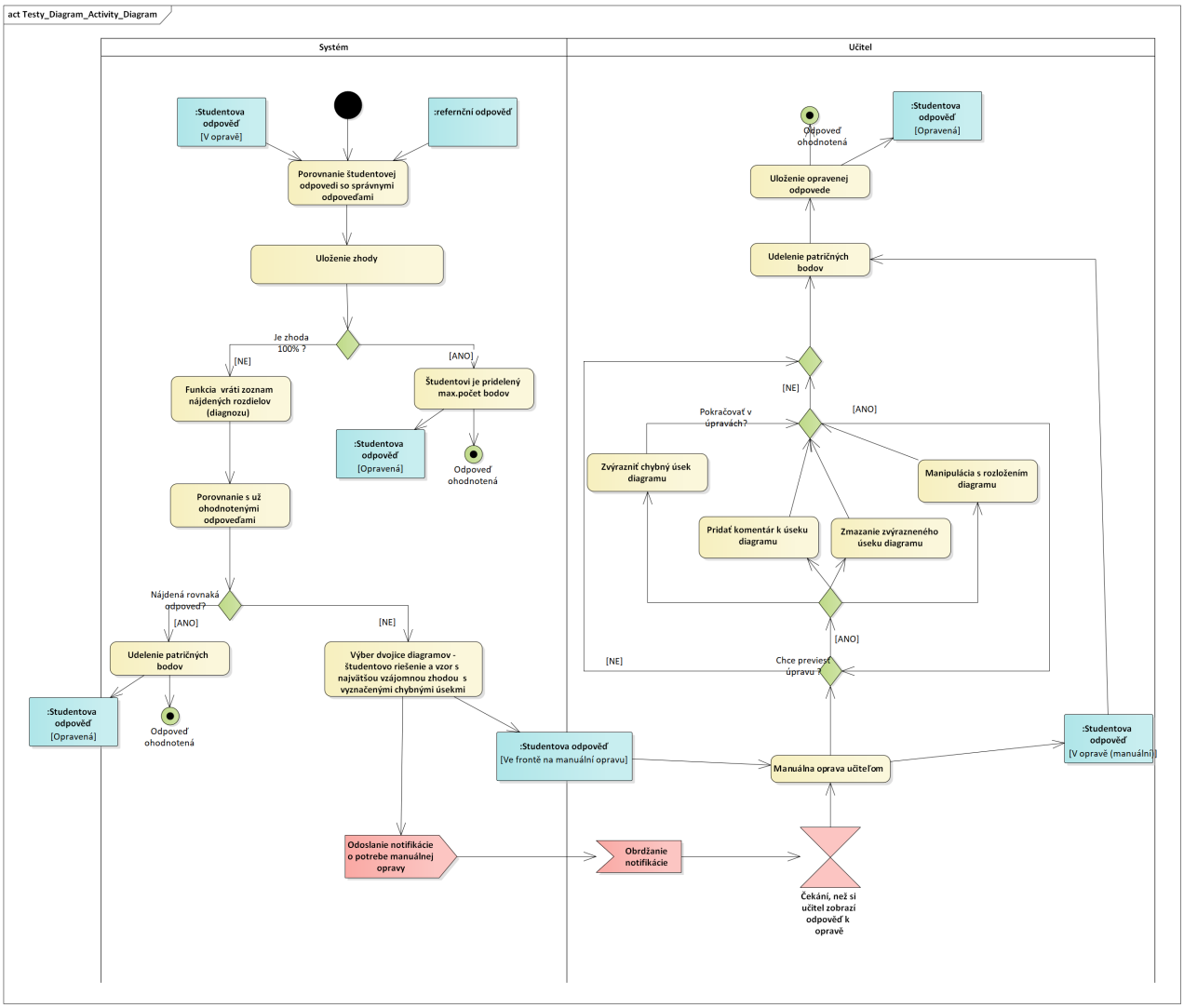
Obrázek C.4: Diagram aktivit: Vyhodnocení odpovědi - Radio



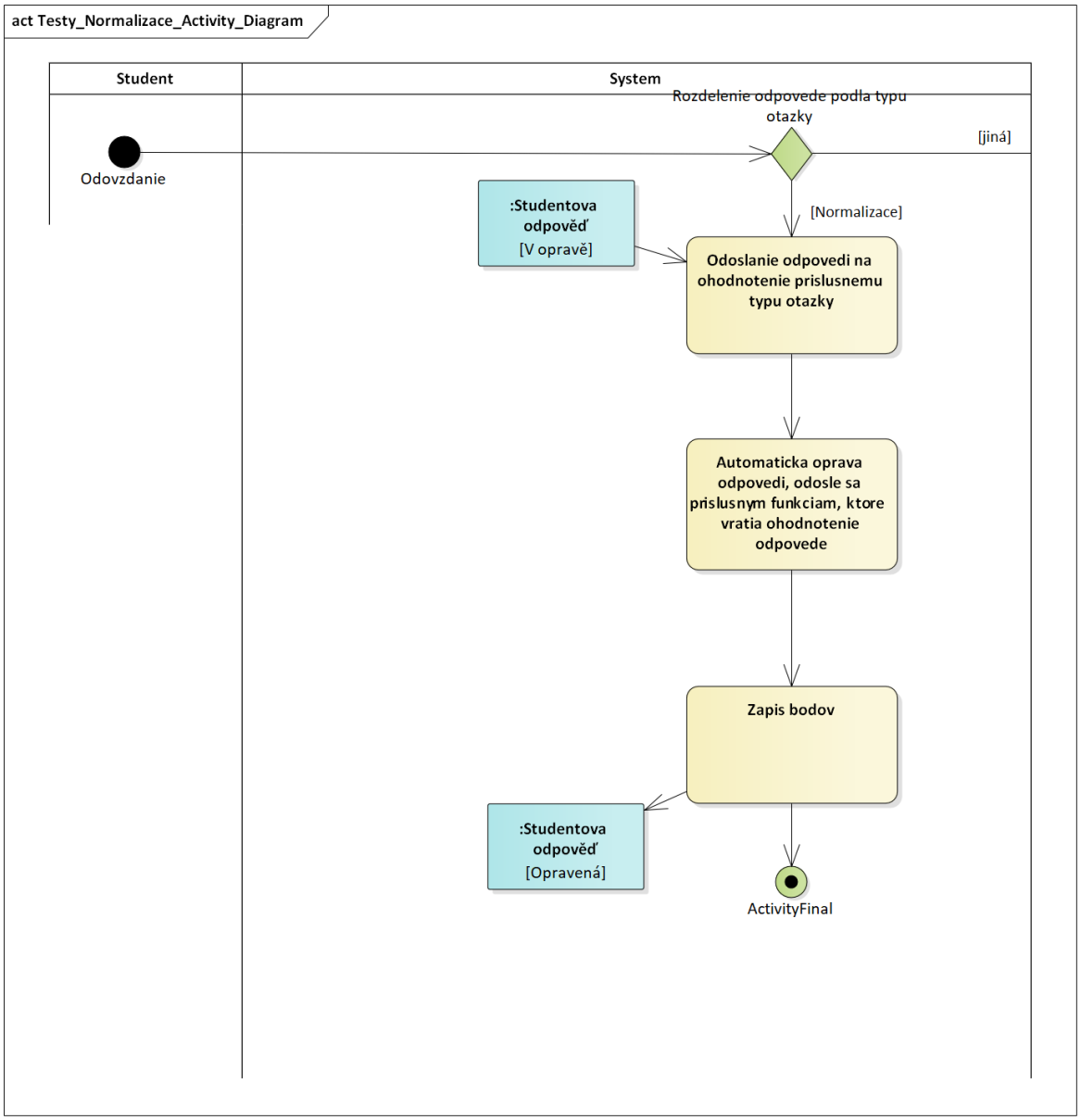
Obrázek C.5: Diagram aktivit: Vyhodnocení odpovědi - SQL



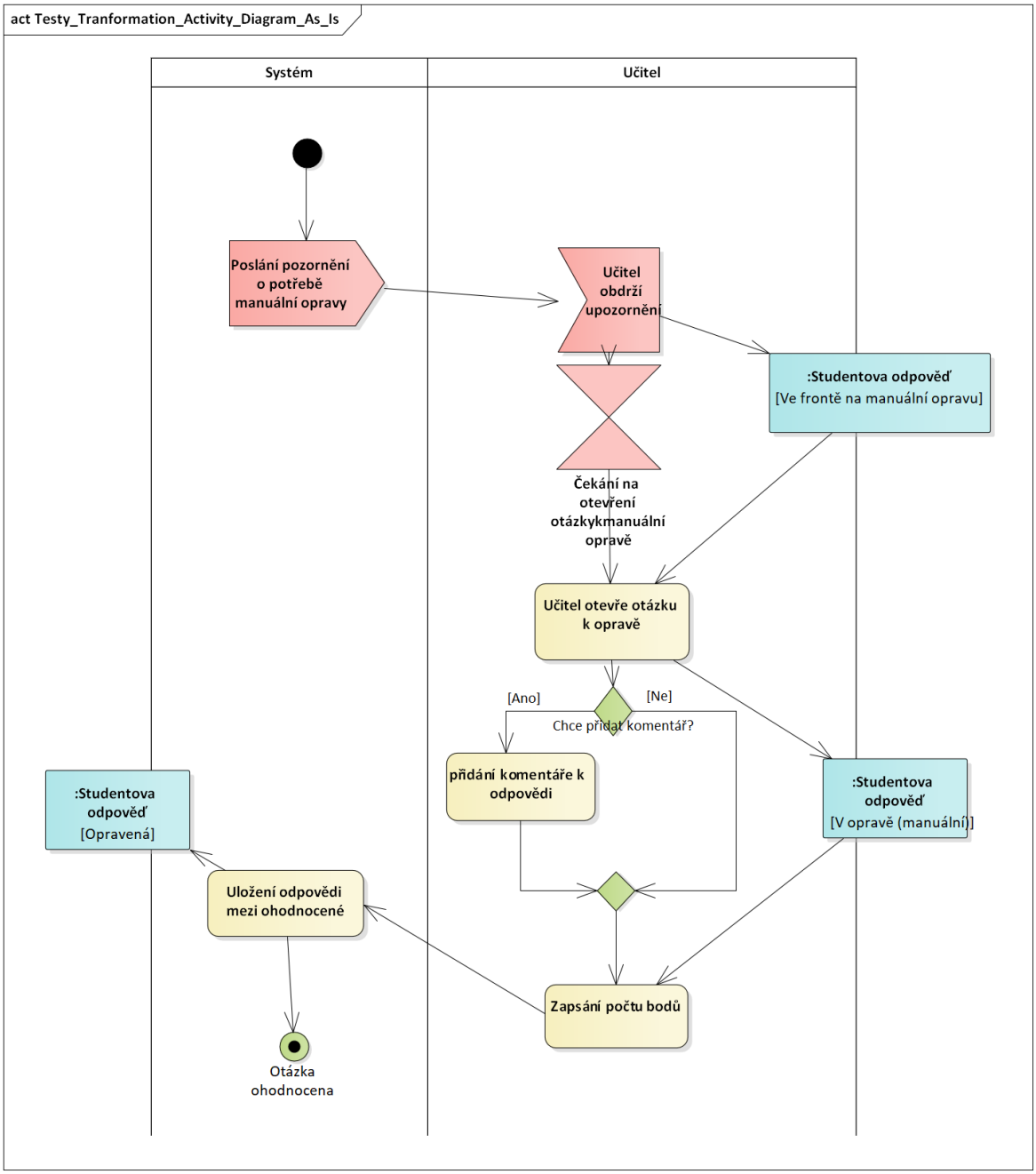
Obrázek C.6: Diagram aktivit: Vyhodnocení odpovědi - RA



Obrázek C.7: Diagram aktivit: Vyhodnocení odpovědi - Diagram



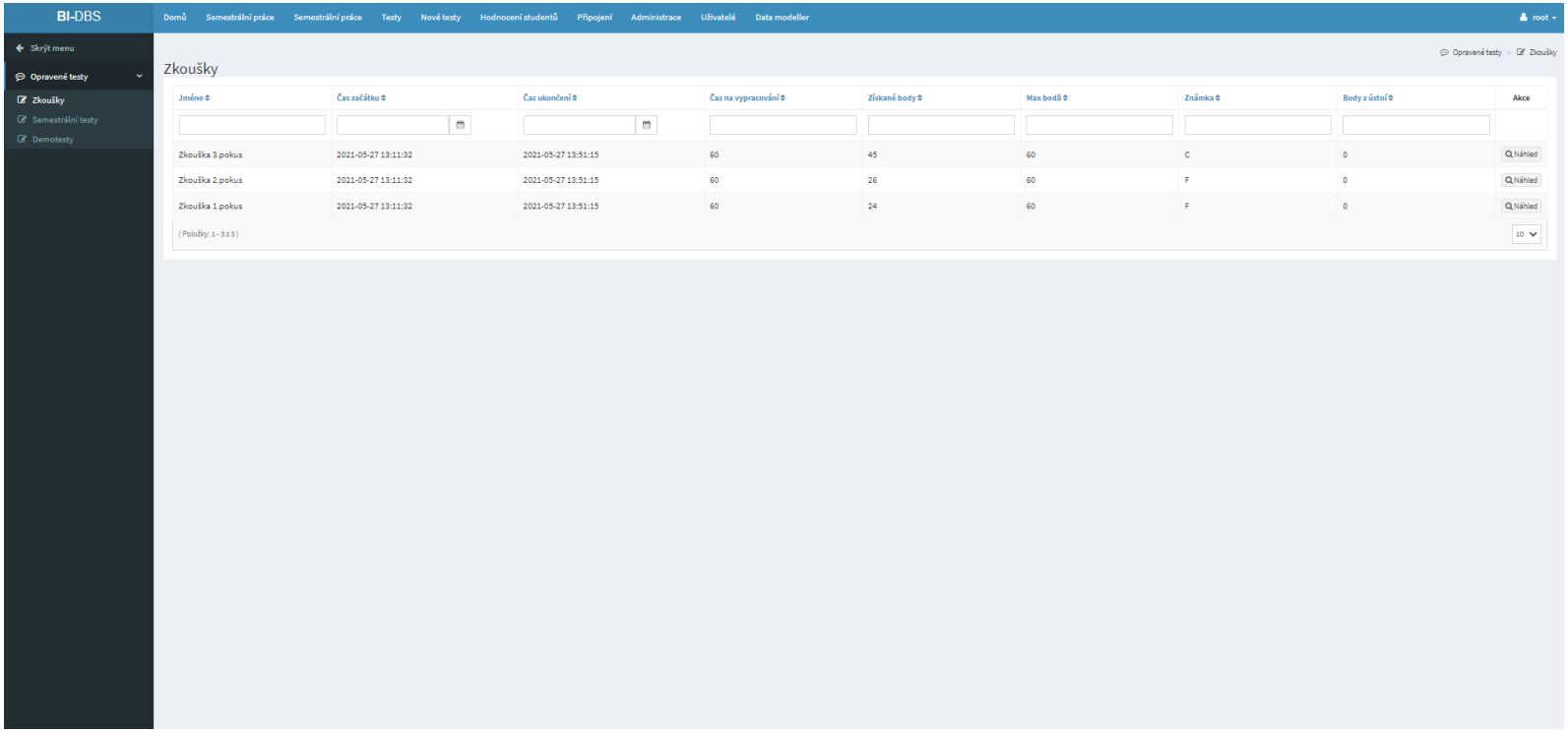
Obrázek C.8: Diagram aktivit: Vyhodnocení odpovědi - Normalizace



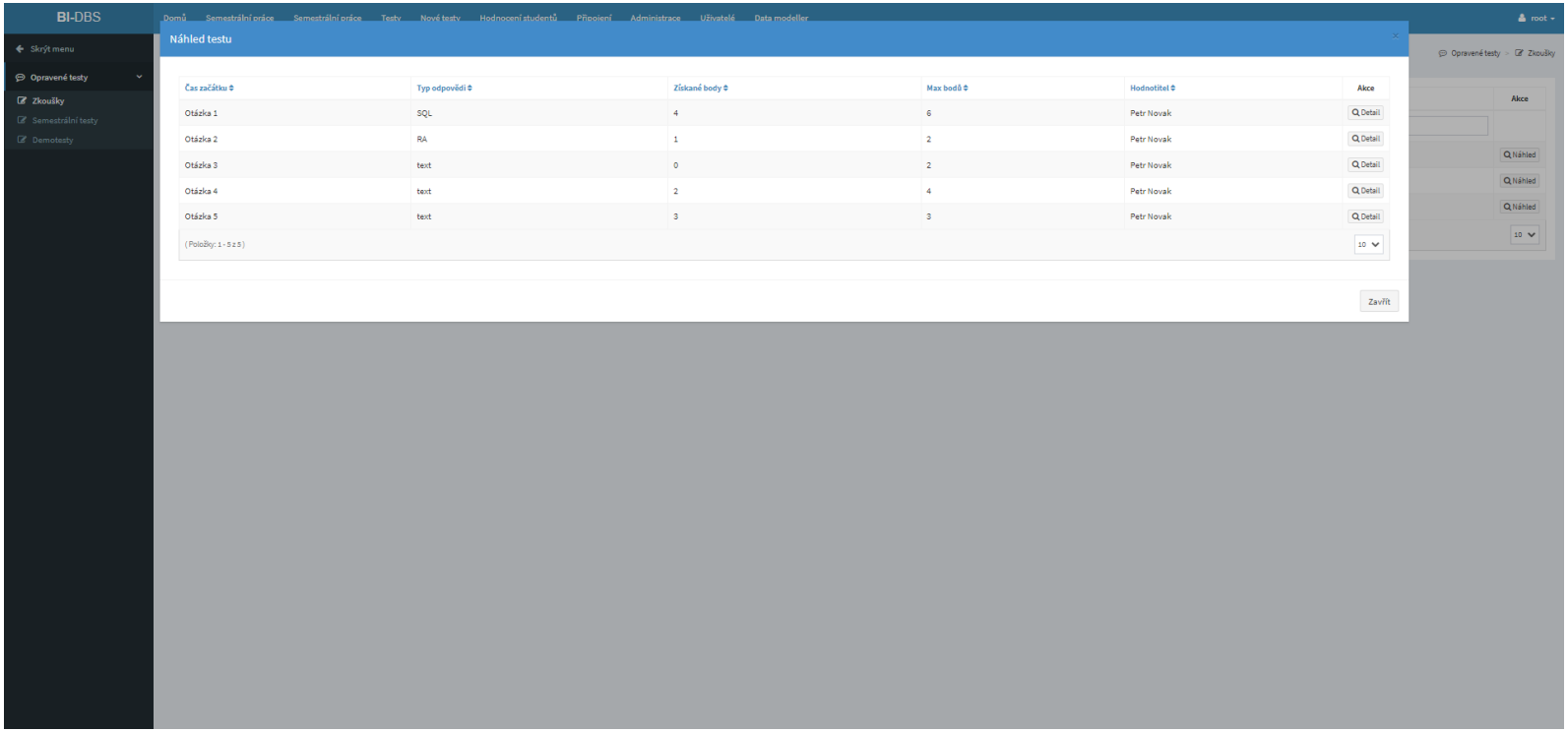
Obrázek C.9: Diagram aktivit: Vyhodnocení odpovědi - Transformace As Is

Příloha D

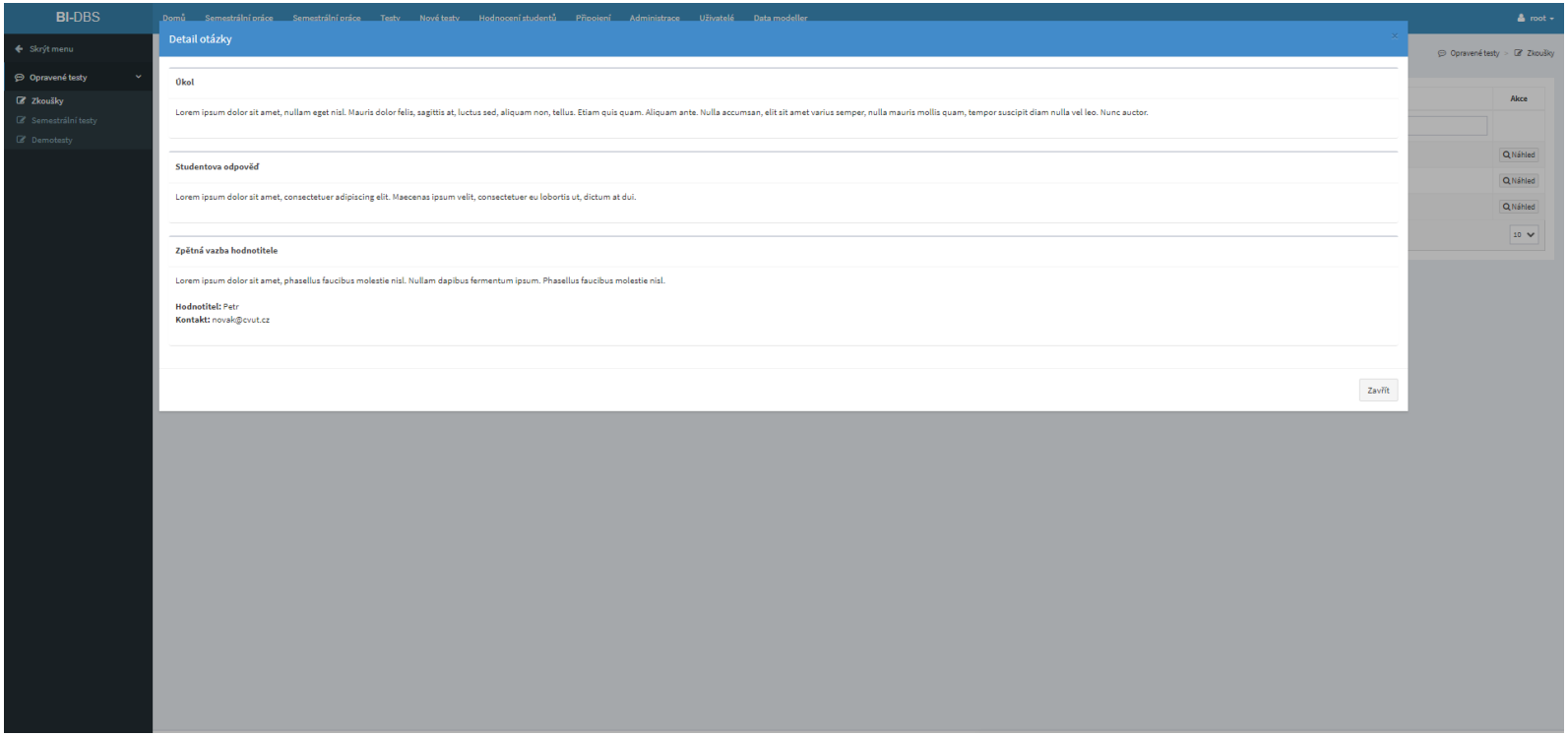
Ukázka vytvořeného prototypu



Obrázek D.1: Ukázka prototypu - Přehled zkoušek



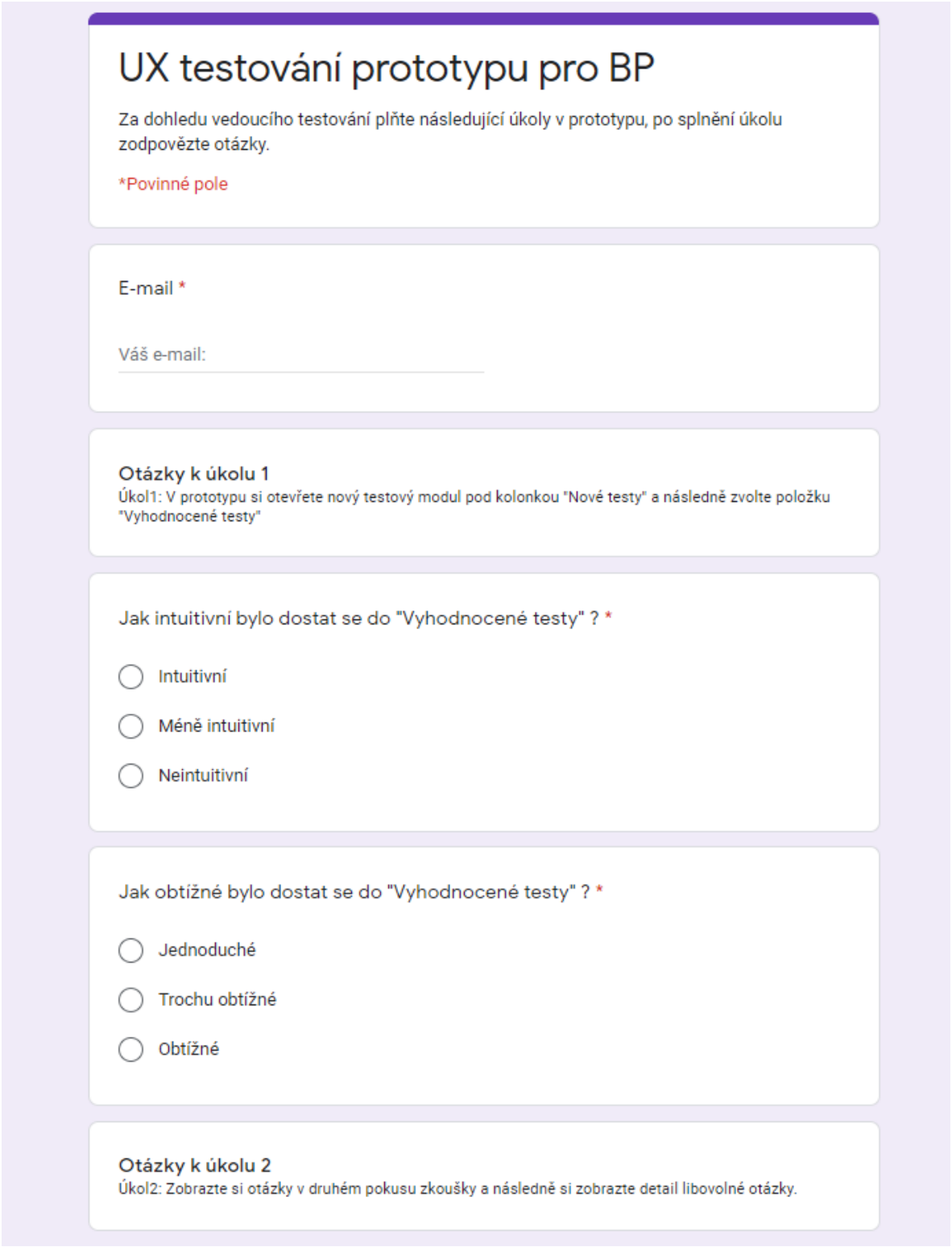
Obrázek D.2: Ukázka prototypu - Náhled testu



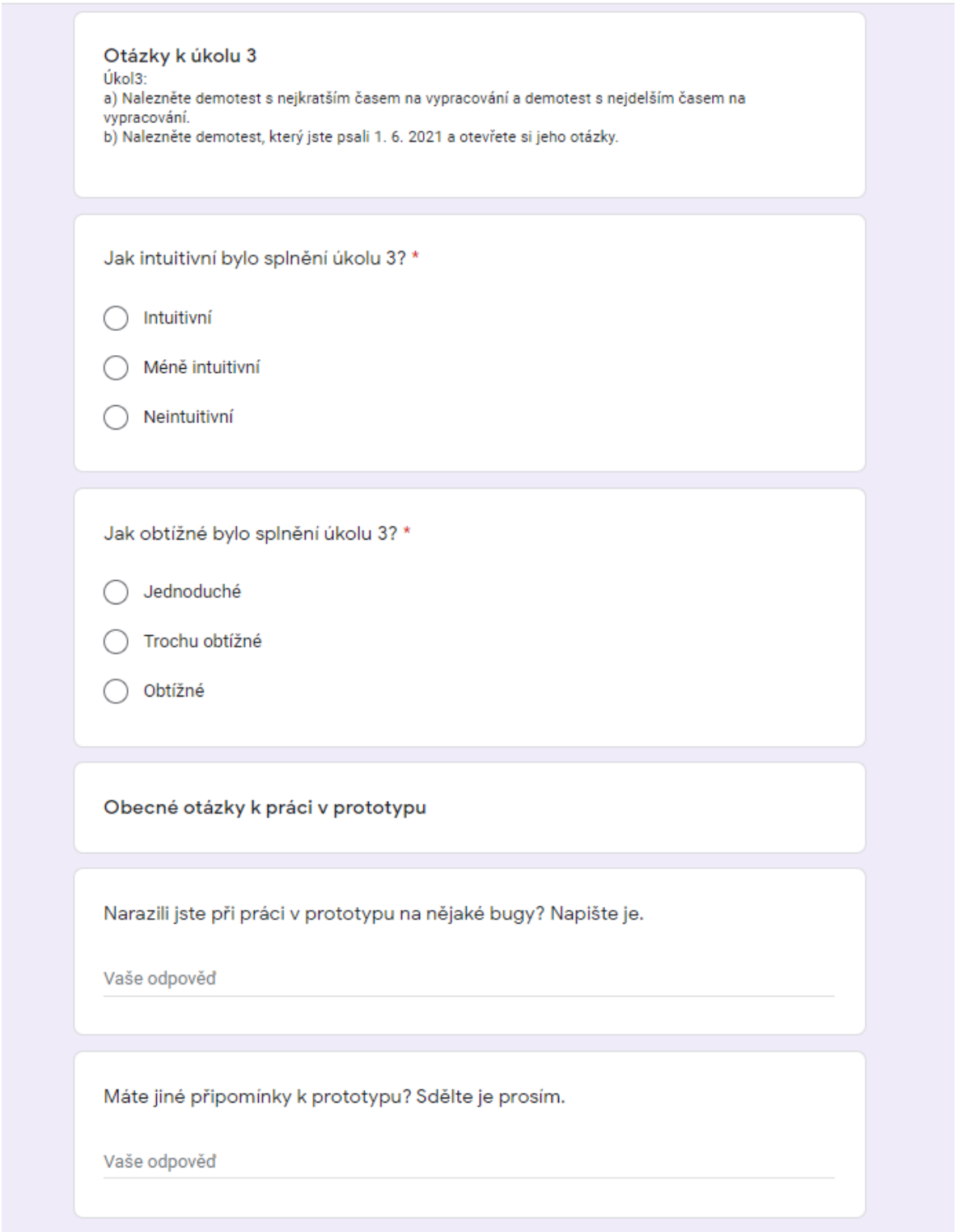
Obrázek D.3: Ukázka prototypu - Detail otázky

Příloha E

Ukázka formuláře pro UX testování



Obrázek E.1: Formulář pro UX testování - ukázka 1



Obrázek E.2: Formulář pro UX testování - ukázka 2