Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Rozhraní pro administraci aplikace s možností injekce chyb

Plzeň 2019 Bc. Jakub Šmaus

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2019

Bc. Jakub Šmaus

Abstract

Interface for application administration and injection of errors

There is a web application UIS (University information system), which is a pseudo-realistic non-trivial application for research of newly developed test methods, which exists in its basic version since January 2018. Based on the verification of this version, there were requirements for adjustments and improvements that had to be made consistently. Another application for error injection into UIS has been prepared in parallel. One of the aims of the thesis is to connect these two parts to create a comprehensive project called TbUIS (Testbed UIS) and make this project available to the professional public. The key activity of this work will be the preparation of new error versions based on the analysis of most common errors. The primary programming language is Java.

Abstrakt

Existuje webová aplikace UIS (University Information System), což je pseudorealistická netriviální aplikace pro výzkum nově vyvíjených testovacích metod, která od ledna 2018 již funguje ve své prvotní verzi. Na základě ověřování této verze vyvstaly požadavky na její úpravy a vylepšení, které bylo potřeba konzistentně provést. Paralelně s aplikací UIS již byla připravena další aplikace pro injekci chyb do UIS. Jedním z cílů práce je spojení těchto dvou částí do uceleného projektu s názvem TbUIS (Testbed UIS) a tento projekt zpřístupnit odborné veřejnosti. Klíčovou aktivitou práce bude příprava nových chybových verzí podle analýzy nejčastěji se vyskytujících chyb. Primárním programovacím jazykem je Java.

Poděkování

Rád bych poděkoval doc. Ing. Pavlu Heroutovi, Ph.D. za odborné rady a cenné připomínky, které mi pomohly tuto diplomovou práci vypracovat.

Obsah

1 Úvod

V rámci diplomové práce *Aplikace s možností injekce chyb pro ověřování kvality testů* z roku 2018 byla vytvořena webová semirealistická aplikace s názvem University Information System (UIS), která bude sloužit jako SUT (System Under Test) pro ověřování správnosti a kvality nově vyvíjených testovacích nástrojů a metod. Aplikace UIS ve své prvotní verzi však ještě nebyla plně dokončena a začleněna do nově vznikajícího projektu TbUIS (Testbed UIS).

Součástí zmíněné diplomové práce je též aplikace ErrorSeeder umožňující injektovat softwarové chyby do UIS a tím vytvářet poruchové klony. Též bylo připraveno sedm poruchových verzí, ze kterých by bylo možné sestavit poruchové klony.

Hlavní cíl této diplomové práce bude pokračování v tomto projektu a to v několika úrovních. První úrovní bude doplnění bezporuchové aplikace UIS o vylepšení, která byla zjištěna až v provozu prvotní aplikace.

Druhou úrovní je spojení UIS a ErrorSeeder do uceleného projektu TbUIS a zpřístupnění tohoto projektu pro odbornou veřejnost. To prakticky znamená vytvoření webových stránek projektu.

Třetí úrovní budou „pomocné aktivity“ v projektu, např. nezávislé spouštění testů, komparátory logů atp.

Klíčovou a nejdůležitější úrovní bude příprava rozsáhlé sady poruchových verzí, kdy se využije všech již existujících možností ErrorSeederu a z těchto poruchových verzí budou připraveny poruchové klony. Nově vzniklé poruchové verze budou obsahovat vhodně zvolené případy typických funkcionálních chyb.

Při práci bude v maximální míře využito již existujících funkcionálních testů a v případě nutnosti budou tyto testy doplněny tak, aby spolehlivě zachytily všechny injektované chyby.

2 Testování pomocí testbedů

Testbed je označení pro platformu nebo soubor zařízení poskytující vhodné podmínky pro testování nových technologií nebo jejich výzkum.

V softwarovém inženýrství tento pojem obecně označuje platformu pro testování softwarových aplikací, tedy software nebo hardware, na kterých může být nasazen System Under Test (SUT), a je zaměnitelný s pojmem testovací prostředí. Slouží primárně k tomu, aby aplikace mohla být testována na bezchybném a pro všechny testery stejném prostředí a u všech nalezených chyb tak byla jistota, že jsou opravdu chybami aplikace, nikoliv že byly způsobené vnějšími vlivy. Proto jsou na testbedy kladeny vysoké nároky co se týká spolehlivosti a bezchybnosti. Vytvoření správného testbedu je sice časově náročné, ale výrazně minimalizuje riziko testery nahlášených chyb, jež se týkají například prostředí a dalších věcí přímo nesouvisejících s testovací aplikací [9].

2.1 Možné problémy testbedů

Testbedy mohou čelit různým problémům v závislosti na typu nebo implementaci. V této části je uveden výčet těch nejčastějších.

2.1.1 Vzdálené prostředí

Pokud je testbed provozovaný na vzdáleném prostředí, může čelit problémům s připojením, případně poruchami hardwaru a softwaru na vzdáleném serveru [9] [10]. Tým využívající testbed je tak odkázaný na vzdálenou podporu, což znesnadňuje komunikaci.

2.1.2 Sdílení několika testery/týmy

Testbedy jsou z praktických a ekonomických důvodů často využívány několika testery nebo testerskými týmy zároveň a dochází tak k situacím, kdy se výsledky jejich testování vzájemně ovlivňují a vznikají tak konflikty [9]. Za této situace mohou být jako chyby nahlašovány události, které jsou výsledkem působení jiného testera. V některých nejvíce sporných případech je jedna instance sdílena dokonce jako testovací i produkční prostředí zároveň [9].

2.1.3 Příliš složitá konfigurace

U testbedů, které mají příliš složitou konfiguaci, vzniká riziko, že ji testeři nebudou schopni správně nastavit. Na základě špatného nastavení testbedu se pak v testované aplikaci mohou objevovat chyby, které byly způsobeny touto příčinou [9].

2.1.4 Použití většího množství technologií

Testbedy, které využívají většího množství různých technologií, mohou být pro použití velmi časově náročné, obzvláště v případě, kdy je potřeba tyto technologie nakonfigurovat před každým testováním [9]. V takovém případě je velmi důležité dávat pozor na vzájemný vliv jednotlivých technologií, aby nedošlo k jejich konfliktu.

2.2 Správné postupy pro tvorbu testbedu

2.2.1 Porozumění požadavkům testování

Aby testbed plnil svůj účel, je nutné nejprve důkladně zanalyzovat požadavky na testování SUT. To může být učiněno typicky na základě dokumentu specifikace nebo přehledu případů užití. Podle nich je vytvořen dokument specifikace obsahující testovací případy a požadavky na testovací prostředí [1]. Postranním efektem důkladné analýzy je krom funkčního a použitelného testbedu také fakt, že zúčastnění testeři jsou lépe obeznámeni s požadovaným chováním aplikace a jejich výstupy jsou tak poté kvalitnější, rychleji získané a je zde menší pravděpodobnost výskytu nesrovnalostí [9].

2.2.2 Kontrola dostupnosti zdrojů

Aby byla zajištěna co nejvyšší spolehlivost a bezchybnost testbedu, je potřeba zkontrolovat dostupnost všech potřebných závislostí. Pokud například testbed využívá vzdálené rozhraní, je vhodné si ověřit spojení s ním „pingem“ nebo jinou metodou. Obdobně je dobré si například zjistit stav využívaného úložiště [9].

Často opomíjená je kontrola dostupnosti veškerého potřebného softwaru a též jejich licencí na konkrétní verze [9].

2.2.3 Přiblížení se cílové platformě

Pro co nejhladší nasazení výsledné verze aplikace je vhodné, aby se testbed co nejvíce podobal reálné cílové platformě. Kromě potřebných technologií je důležité si dát pozor i na jejich verze. V případě, že není možné použít stejné verze jako na cílové platfomě, mohou být použity nejnovější verze těchto technologií, ovšem vždy pouze v případě, že jsou tyto verze kompletně zpětně kompatibilní [9].

2.2.4 Automatizace

Pokud v rámci testování dochází k opakování některých testů, je vhodné uvažovat o jejich automatizaci. K tomu je potřeba výběr správného nástroje. Ten v závislosti na vhodnosti může i nemusí být součástí testbedu. V obou případech je však potřeba jej správně nakonfigurovat [9].

2.3 Typy testbedů

Často je tímto pojmem myšlen například server se správnou konfigurací, na kterém běží testovací verze vyvíjené aplikace (SUT). Tento způsob je nezbytný pro vývoj komerčního softwaru se specifickými podmínkami pro svůj běh.

Kromě jiného je ale možné testbedem označovat i simulaci určité aplikace, jejíž funkčnost SUT využívá. Příkladem může být například webový prohlížeč Arena, na kterém bylo v devadesátých letech testováno zobrazení například HTML, kaskádových stylů (CSS) nebo obrázků PNG [11]. V současné době je nahrazen modernějším prohlížečem Amaya, který vyhovuje novým standardům [12].

Někdy jsou jako testbedy označovány i „pískoviště“ (*sandbox*), v rámci kterých může vývojář interaktivně měnit kód a okamžitě pozorovat výsledek. Mezi ně patří například služby jako JSFiddle [13] a Plunker [14] pro testování JavaScriptového kódu nebo pískoviště pro uživatele na Wikipedii.

2.4 Testbed UIS

Tato práce se mimo jiné zabývá i testbedem UIS, který slouží k testování aplikací používaných pro výzkum vývoje nových druhů testů. UIS je tak simulací reálné aplikace, která je testována. Jinak řečeno by se dala označit za umělý SUT. Jedná se tedy o jistou modifikaci druhého příkladu. Testbedem zde není server, ale samotná aplikace. Z tohoto důvodu je kladen velký důraz

na její bezchybnost v jedné referenční instanci. Mnohem detailněji je tato aplikace popsána v části 3. K tomu, aby tento testbed mohli využívat vývojáři i mimo tým autorů, je zapotřebí mít veřejně přítupné všechny informace, které jsou k jeho užití potřebné. K tomuto účelu by tedy měla sloužit aplikace TbUIS – web, která je mimo jiné předmětem této práce. Její návrh je popsán v části 6 a implementace v části 7.

3 UIS

UIS (University Information System) je webová semirealistická aplikace, která slouží jako umělý System Under Test (SUT) s možností úpravy svojí chybovosti. Byla vytvořena v akademickém roce 2017/2018 v rámci diplomové práce Jiřího Matyáše [2].

UIS simuluje funkci reálného univerzitního informačního systému. Lze se do něj přihlásit buď pod učitelskou, nebo pod studentskou rolí. V závislosti na zvolené roli lze například vytvářet nové zkouškové termíny, hodnotit studenty nebo se přihlašovat na zkoušky.

3.1 Spring

Aplikace UIS je napsána za použití Spring Frameworku. Jedná se o open-source aplikační framework, jehož první verze byla vydána již v roce 2003. Slouží primárně pro vývoj Java Enterprise Edition (Java EE) aplikací [3]. Umožňuje tvorbu aplikací různých architektur, ale podporuje především využití POJO[1]. Skládá se z jednotlivých modulů, kterých je přibližně dvacet, a při vývoji aplikace lze využít pouze některé z nich.

3.1.1 Aspektově orientované programování

Významnou součástí Springu je paradigma aspektově orientovaného programování (AOP). Aspektem se v tomto případě rozumí modul nebo třída obsahující kód, který je často volán z jiných tříd [15]. V opačném případě by v nich byl tento kód duplikován. Typickým příkladem je autentifikace nebo logování. Frameworky, které podporují toto paradigma, zajišťují, že v případě volání určité metody bude vykonán kód aspektu [16]. Například v případě autentifikace by při volání jakékoliv metody byla nejprve zavolána metoda aspektu, jenž by kontrolovala přístupová práva.

3.1.2 Inversion of Control a Dependency Injection

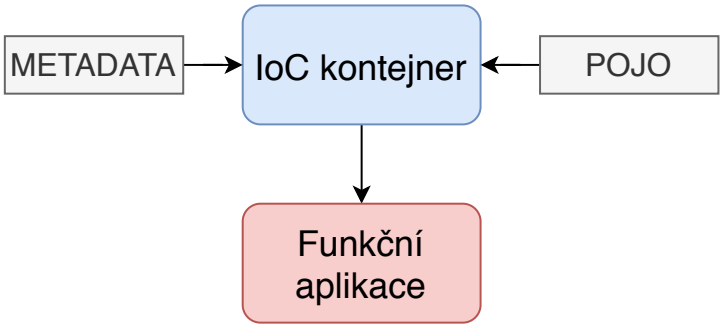
Důležitým návrhovým vzorem, který Spring využívá, je Inversion of Control (IoC). Tento návrhový vzor je velmi obecný a to znamená, že tok kódu není řízen přímo kódem samotným, ale často nějakým externím frameworkem.

[1]POJO – Plain Old Java Object, obyčejný javovský objekt, který není svázán žádnými pravidly

To je v kontrastu s klasickým programováním. Konkrétní implementací IoC je potom Dependency Injection (DI). V rámci DI aplikace za běhu vytváří instanci potřebné komponenty a zajišťuje vložení závislosti [17].

3.1.3 IoC kontejner

Dalším důležitým prvkem Springu je IoC kontejner. Ten se stará o životní cyklus objektů a jejich závislosti. Využívá POJO a konfigurační soubory/části kódu (metadata) a tyto transformuje do funkční aplikace, jak je naznačeno na obrázku 3.1. Těmito konfiguračními daty jsou například informace o životním cyklu a závislostech objektů a mohou být definovány například pomocí anotací nebo XML souborů. Spring nabízí dvě implementace IoC kontejneru – BeanFactory a ApplicationContext. BeanFactory obsahuje pouze základní funkčnost IoC a používá se tak především pro menší aplikace. ApplicationContext je bohatší. Obsahuje BeanFactory a další funkčnosti, například možnost posílat události aplikace určitým *event listenerům*. Pro svou multifunkčnost je tak ApplicationContext doporučován pro většinu aplikací.



Obrázek 3.1: Funkce IoC kontejneru

Objekty, jenž jsou spravovány IoC kontejnerem, se nazývají beany. IoC kontejner si je vytváří na základě POJO a konfiguračních dat a každé beaně na základě nich přiřadí soubor charakteristik. Mezi tyto charakteristiky patří například původní třída, ze které je beana vytvořena, nebo unikátní identifikátor.

3.2 JavaServer Pages

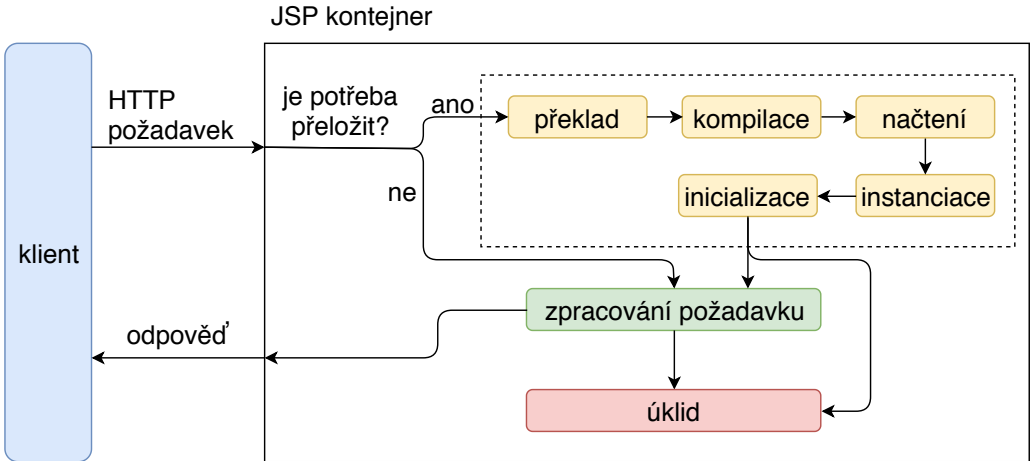
Nedílnou součástí webové aplikace založené na programovacím jazyku Java jsou JavaServer Pages (JSP). JSP jsou využívány pro tvorbu dynamických webových stránek v Javě [4]. Tyto soubory s dynamickým obsahem

mají příponu `.jsp`. Dynamický obsah může být do webových stránek obsahujících HTML přidán pomocí tagů `<% ... %>`, které umožňují snadný přístup k javovským objektům. Tyto tagy ohraničují užití speciálních JSP elementů a příkazů. Alternativou k nim je použití XML elementů `<jsp:scriptlet> ... <\jsp:scriptlet>`.

3.2.1 Životní cyklus

Životní cyklus JSP se skládá ze sedmi fází – překlad (*translation*), kompilace (*compilation*), načtení (*loading*), instanciace (*instantiation*), inicializace (*initialization*), obsloužení požadavku (*servicing request*) a úklidu (*cleanup*) [18].

Při načítání stránky posílá klient na server HTTP požadavek. Na serveru je tento požadavek předán JSP kontejneru, který začne vykonávat jednotlivé části životního cyklu JSP. Přehledný diagram těchto fází je znázorněn na obrázku 3.2.



Obrázek 3.2: Životní cyklus JSP

Překlad

Při přijetí požadavku od klienta server nejdříve kontroluje, jestli je potřeba soubor znovu přeložit. To je nutné v případech, kdy soubor ještě přeložen nebyl nebo byl od posledního překladu změněn.

Ve chvíli, kdy soubor přeložit není potřeba, jsou tyto první fáze přeskočeny a kontejner už požadavek pouze obslouží.

Pokud je potřeba soubor znovu přeložit, kontejner načte obsah příslušného JSP souboru z disku a všechny JSP elementy zkonvertuje na javovský kód

a zbytek HTML elementů vloží do `println()` příkazů. Tím vznikne servlet (`.java` soubor). Aby bylo možné pomocí něj obsluhovat HTTP požadavky, dědí od abstraktní třídy `HttpServlet`. Díky tomu poskytuje metody jako `doGet(...)` nebo `doPost(...)`. Od třídy `HttpServlet` však nedědí přímo, ale přes abstraktní třídu `GenericServlet`, jenž také dědí od `HttpServlet` a navíc poskytuje efektivnější implementaci metod `init` a `destroy` [4].

Kompilace

Vygenerovaný javovský servlet je JSP kontejnerem přeložen do spustitelné třídy (`.class` soubor) [18].

Načtení

Zkompilovaný `.class` soubor je načten do paměti kontejneru [18].

Instanciace

Kontejnerem je zavolán konstruktor servletu a tím vytvořena jeho instance [4].

Inicializace

V rámci inicializace je zavolána metoda standardního JSP aplikačního rozhraní `jspInit()`, která má na starosti navázání spojení s databází nebo otevírání souborů. Pokud je potřeba specifické inicializace, lze tuto metodu přepsat (*override*). Tato fáze a tedy i metoda jsou v rámci životního cyklu vždy provedeny pouze jednou [18].

Obsloužení požadavku

Ve chvíli, kdy již kontejner vykonal všechny předchozí fáze životního cyklu nebo v případě, kdy je na server přijat požadavek na JSP stránku, kterou není potřeba přeložit, je zavolána metoda `_jspService()`. Jejími parametry jsou instance tříd `HttpServletRequest` a `HttpServletResponse`. `HttpServletRequest` je třída nesoucí informace například o klientovi nebo parametry požadavku. `HttpServletResponse` obsahuje prostředky pro odeslání odpovědi klientovi. Metodu `_jspService()` nelze přepsat. Je volána vícekrát, v závislosti na počtu požadavků. Jejím výstupem je HTML kód, který je poslán zpět klientovi. Obsloužení každého požadavku probíhá ve vlastním vlákně [4].

Úklid

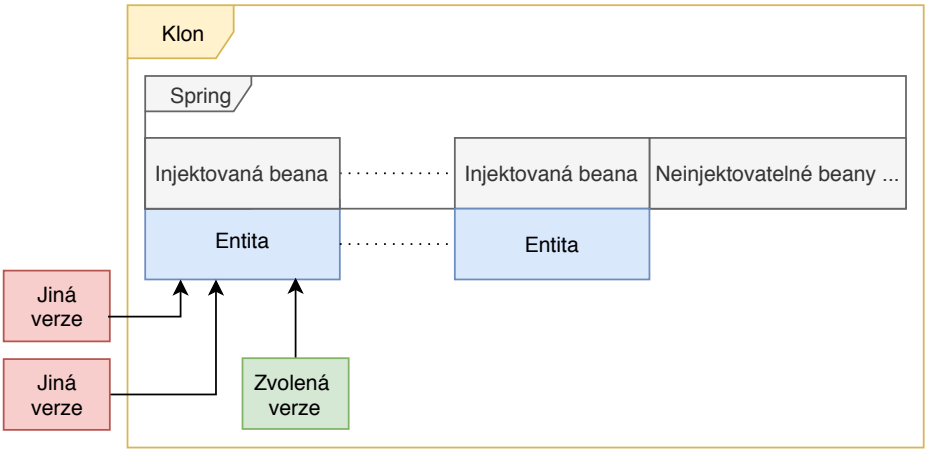
Když je JSP odstraňována kontejnerem, je zavolána metoda `jspDestroy()`, která je bez parametrů. Ta má za úkol například uvolnit všechny zdoje, jenž byly v rámci životního cyklu použity, tedy například databázová spojení nebo zavřít otevřené soubory. Pokud by bylo potřeba jejího specifického chování, lze ji rovněž přepsat [5]. K této fázi dochází většinou ve dvou případech, a to pokud je potřeba šetřit s přidělovanými zdroji a proto je uvolnit nebo v případě, že dojde k ukončení činnosti celého JSP kontejneru [4].

3.3 Používané pojmy v TbUIS

Napříč celým projektem je použito několik pojmů. Jejich uvedení je nutné pro plné pochopení dalšího textu.

- **entita** – dílčí část aplikace implementovaná buď v korektní (tj. bezchybné), nebo chybové verzi
- **verze**[2] – konkrétní implementace entity, může mít buď korektní, nebo chybovou podobu
- **bean** – objekt spravovaný IoC kontejnerem
- **injektovatelné beany** – aplikace zahrnuje celkem pět bean, kterým je umožněno vkládání korektních nebo chybových verzí
- **klon** – instance UIS sestavená z různých verzí entit, jeho schéma je znázorněno na obrázku 3.3

[2] verze zde neurčuje historický vývoj, ale konkrétní implementaci části kódu (konkrétně entity)



Obrázek 3.3: Znázornění klonu

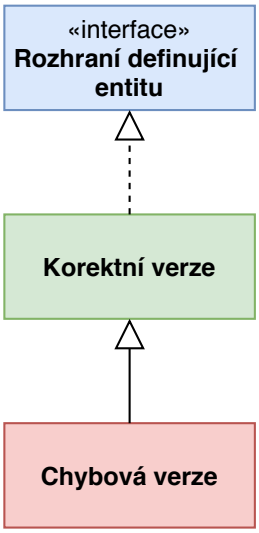
3.4 Injektování chyb

Aplikace UIS byla navržena tak, aby ji uživatel mohl jednoduše sestavit s různou mírou chybovosti. Obsahuje tak části, které jsou implementované jak v korektní, tak v chybové verzi. Tyto části jsou nazývány *entity*. Každá entita má své rozhraní, které deklaruje všechny metody, které musí korektní i chybové verze nějakým způsobem implementovat. Technologicky jsou tyto entity řešeny pomocí Java bean. Aplikace UIS má v souboru *seed.xml* definovanou množinu bean, kterou potřebuje ke svému úspěšnému sestavení. Ta je nazývána aplikační kontext. Z každé entity je tedy vybrána jedna verze, která bude beanou a příslušnou závislost bude splňovat. Samotný výběr a sestavování vybraných verzí je prováděno aplikací ErrorSeeder, jež je popsána v části 4.

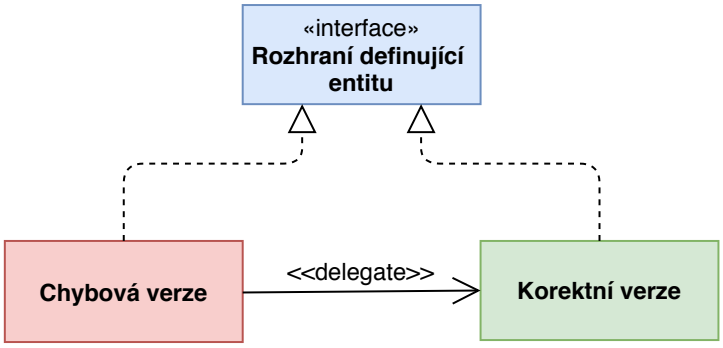
Při procesu injektování je vybraná verze uvedena jako součást aplikačního kontextu. IoC kontejner pak tyto objekty převede na beany a vytvoří funkční aplikaci.

Všechny verze entit se nacházejí v rámci aplikace. Korektní verze entit jsou vždy umístěny v balíčku `correct` a mají jmennou konvenci `Base<název_entity>`. Vždy dědí od rozhraní definující entitu a korektně implementují všechny metody.

Chybové verze entit se nacházejí ve zvláštním balíčku `error` a jejich jmenná konvence je `E<pořadí><název_beany>`. Chybové verze mohou být vytvořeny oddělením od korektní verze, to je znázorněno na obrázku 3.4. Další možností je implementace rozhraní, které entitu definuje. Tento způsob využívá návrhový vzor proxy a je znázorněn na obrázku 3.5.



Obrázek 3.4: Vytvoření chybové verze – rozšíření korektní verze o chybovou funkcionalitu



Obrázek 3.5: Vytvoření chybové verze – návrhový vzor proxy

Výhoda dědění od korektních verzí je v tom, že v chybových verzích je často potřeba ovlivnit funkčnost pouze jedné nebo několika metod a ostatní metody jsou korektní, jsou tedy využívány z rodičovské korektní verze.

V závislosti na výběru korektních nebo jednotlivých chybových verzí daných entit lze sestavovat rozdílné verze UIS s různou mírou chybovosti. Použití konkrétní verze entity v daném klonu je závislé na souboru *seed.xml*, který je importován do aplikačního kontextu UIS. Z aplikačního kontextu jsou pak tyto informace Spring frameworkem převedeny do Spring IoC kontejneru, který obsahuje ty beany, jež budou aplikací použity.

Současná implementace dovoluje chybové verze vytvářet v DAO, Utility a Servisní vrstvě. DAO vrstva obsahuje Data Access Objekty, které zajišťují manipulaci s entitami a databázové operace. Vrstva Utility sdružuje třídy pro

práci s daty, jako je export nebo formátování. Servisní vrstva je zodpovědná za funkční kód a je prostředníkem mezi zobrazením a databází. Do budoucna je možné rozšíření i pro kontrolery. To však nebylo v původní aplikaci implementováno.

Původní aplikace obsahovala již vytvořené chybové verze pro všechna injektovatelná rozhraní, konkrétně tedy pro `GradeTypeDao`, `UserDao`, `StudentService`, `TeacherService` a `DateUtility`. Jejich kompletní přehled je uveden v tabulce 3.1.

Kromě již hotových chybových verzí entit lze vytvořit i vlastní, což je mimo jiné cílem této práce. Všechny tyto verze by však měly splňovat náležitosti popsané výše. Podrobnější popis tvorby nových verzí entit je v části 9.

verze	**metoda**	**popis chyby**
E01DateUtility	`stringToDate`	vrací datum posunuté o jeden den dopředu
E02DateUtility	`stringToDate`	vrací datum, které je vždy stejné
E01StudentService	`getStudiedSubjectsList`	vrací `null` místo studovaných předmětů
E02StudentService	`getStudentTotalCredits`	vrací vždy 20 kreditů místo skutečného počtu
E01TeacherService	`getTaughtSubjectsList`	vrací `null` místo seznamu všech předmětů učitele
E01UserDAO	`findAllUsers`	vrací pouze studenty, bez ostatních uživatelů (učitelů)
E01GradeTypeDAO	`getAllGradeTypes`	vrací pouze známky A, B, C, D

Tabulka 3.1: Původní chybové verze

3.5 Hibernate

Komunikace s databází je v aplikaci UIS řešena pomocí frameworku Hibernate. Ten poskytuje možnost objektově relačního mapování (ORM),

což je způsob mapování objektově orientovaného javovského kódu do relační databáze.

3.6 Maven

Maven je nástroj pro automatické sestavování Java aplikací a správu jejich závislostí. Aplikace je reprezentována jako projekt, jenž se může skládat z dalších podprojektů. Tyto projekty jsou identifikovány pomocí skupiny (*group*), názvu (*artifact name*) a verze (*version*) [6].

3.6.1 POM

Konfiguranční jednotkou Mavenu je POM (Project Object Model), který je reprezentován v podobě XML souboru s názvem *pom.xml* [7]. Vždy se nachází v kořenovém adresáři daného projektu.

Kořenovým elementem souboru *pom.xml* je vždy `project`. V rámci něj je nutné nadefinovat čtyři elementy, a to `modelVersion`, `groupId`, `artifactId` a `version` [19]. Element `modelVersion` udává verzi POM a v Mavenu 2 a 3 je vždy nastaven na hodnotu *4.0.0*, protože se jedná o jedinou verzi modelu, kterou tyto verze Mavenu podporují. Element `groupId` značí skupinu příslušného projektu, `artifactId` jeho název a `version` současnou verzi.

Kromě těchto obsahuje `project` i nepovinný element `dependencies`. Ten v rámci vnitřních elementů `dependency` sdružuje všechny závislosti daného projektu.

3.6.2 Hiearchie projektů

Vzhledem k tomu, že jedna aplikace může být tvořena více projekty Mavenu, je často nutné mít mezi těmito projekty hiearchickou strukturu, tedy mít mezi projekty vztah rodič-potomek. Tento vztah má jistá pravidla, například že potomek může mít jen jednoho rodiče. Jeho realizace je proveditelná dvěma způsoby – oddědění nebo agregací. V rámci jednoho projektu lze použít i oba přístupy najednou [20].

Oddědění lze realizovat v *pom.xml* potomka, a to pomocí elementu `parent`. V rámci něj jsou definovány elementy určující rodičovský projekt, jako je `groupId`, `artifactId`, `version` nebo `relativePath`.

Agregace (nebo též multi-modulový model) lze realizovat pomocí *pom.xml* souborů rodičů. V nich je možné definovat element `modules`, jenž v rámci svých vnitřních elementů `module` specifikuje jednotlivé potomky jako relativní cesty do jejich domovských adresářů nebo příslušných *pom.xml* souborů. Tento

způsob lze využít například pokud rodič sám o sobě nemá žádné zdrojové soubory a jeho potomky tak jsou projekty, jejichž kořenové adresáře leží přímo v kořenovém adresáři rodiče.

3.6.3 Životní cyklus

Životní cykly Mavenu je složený z jednotlivých fází (*phase*) a definují pořadí, v jakém se fáze budou vykonávat. V rámci fáze jsou definovány cíle (*goals*) a jejich pořadí. Každý cíl provádí určitou činnost. Může být součástí jedné, vícero, ale také žádné fáze. Pokud není součástí žádné fáze, lze cíl spustit přímým voláním.

Maven definuje tři základní životní cykly – *clean*, *default* (nebo též *build*) a *site*. Cykly *clean* a *site* mají oba po třech fázích a cyklus *default* má fází 21.

Při spuštění životního cyklu jsou tedy vykonány všechny fáze a cíle, které jsou v rámci ní specifikovány. Kromě toho lze ale spustit i jednotlivé fáze. V tom případě jsou spuštěny všechny fáze v daném životním cyklu, které této fázi předcházejí, a poté tato fáze a tedy i všechny cíle na tyto fáze vázané.

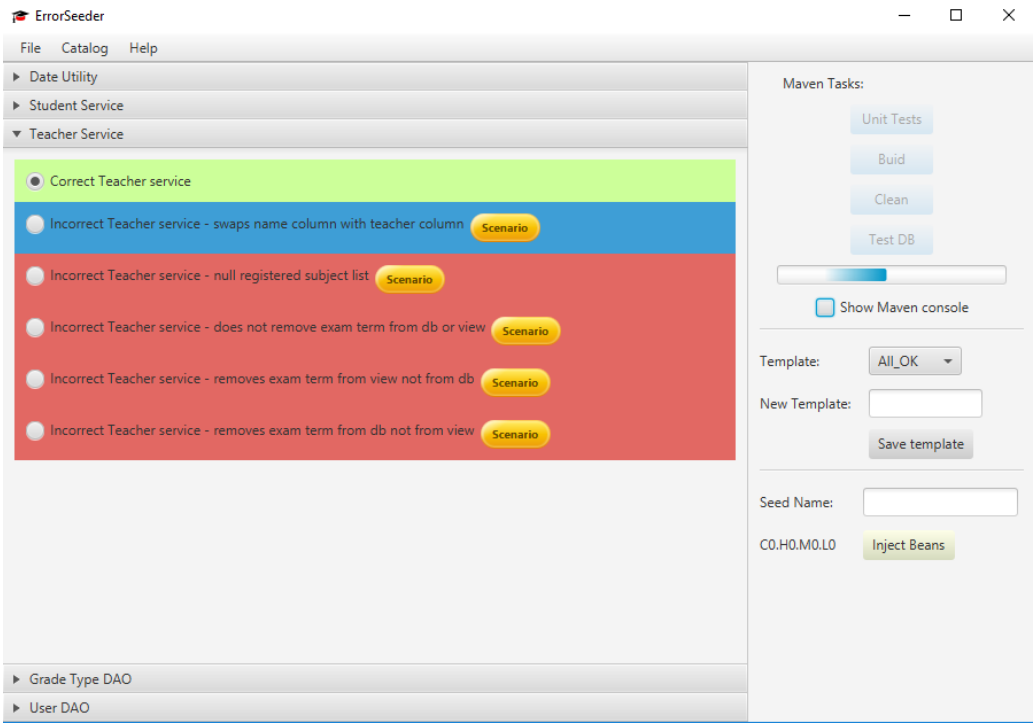
Pro spouštění slouží také pluginy. Každý obsahuje soubor cílů. Pluginy pro základní funkčnost jsou již součástí jádra Mavenu. Pokud by bylo potřeba specifického pluginu, jenž není nativní součástí, lze jej dodatečně implementovat. Mezi pluginy, jenž jsou součástí jádra, patří například `clean` (vyčištění projektu, smazání souborů vygenerovaných do adresáře `target`), `compiler` (přeložení javovských zdrojových souborů), `deploy` (nasazení projektu na vzdálené úložiště) nebo `install` (instalování balíku do lokálního úložiště) [21].

4 ErrorSeeder

ErrorSeeder (Správce poruchových entit) je desktopová aplikace určená k sestavování aplikace UIS z jednotlivých korektních či chybových entit. Ukázka použití ErrorSeederu je na obrázku 4.1. Každý UIS klon sestavený z jednotlivých verzí entit lze s pomocí ErrorSeederu vyexportovat jako WAR soubor. Zároveň si lze jednotlivé kombinace verzí uložit jako šablonu, jenž je možné poté opakovaně využít při exportu klonů UIS.

4.1 Testy

Kromě základních funkčností je také možné nad UIS spustit jednotkové testy, které ověřují přítomnost zanesených chyb. Tyto jednotkové testy se nacházejí v aplikaci UIS a ve většině případů pokrývají přes 80 % kódu pro jednotlivé entity. V mnoha případech se jedná i o pokrytí přes 90 %. Zároveň je možné spustit i test spojení s databází.



Obrázek 4.1: Ukázka použití ErrorSeederu

4.2 Katalog

Informaci o dostupných verzích získává ErrorSeeder z katalogu, jenž musí uživatel po přidání jakékoliv další poruchové verze sám „ručně“ upravit. Tento katalog je XML souborem obsahující informace o jednotlivých verzích, jako jsou název, popis, zda se jedná o korektní či chybovou verzi, případně závažnost chyby a testovací scénář.

Tento katalog může uživatel editovat v běžném XML editoru. V rámci ErrorSeederu je možné vytvořit si nové šablony a takto upravený katalog si lze uložit. Do ErrorSeederu jej lze načíst přes volbu *Catalog*, následně *Import* a výběr příslušného katalogu.

Bezchybně vyplněný katalog je nutnou podmínkou pro další práci s korektními a chybovými verzemi a proto musí být jeho úpravám věnována zvýšená pozornost.

4.3 Struktura katalogu

Kořenovým elementem katalogu je `catalog` bez atributů, uvozující jeho začátek a konec. Ten obsahuje vnitřní elementy `beans`, `pathToBeanSetFile`, `pathToPOM` a `templates`. Element `beans` v sobě obsahuje jednotlivé elementy `bean`, jenž reprezentují jednotlivé Java beany. Obecná struktura katalogu je naznačena v ukázce kódu 4.1.

```
<catalog>
    <beans>
        <bean>...</bean>
        ...
    </beans>
    <pathToBeanSetFile> ... </pathToBeanSetFile>
    <pathToPOM>...</pathToPOM>
    <templates>
        <template name="...">
            <versionId>...</versionId>
            ...
        </template>
    </templates>
</catalog>
```

Ukázka kódu 4.1: Struktura katalogu

Element `beans`

Element `beans` obsahuje elementy reprezentující všech pět injektovatelných bean, tedy elementů `bean`. Elementy `bean` jsou detailněji popsány v části 4.3.1.

Element `pathToBeanSetFile`

Element `pathToBeanSetFile` udává cestu k souboru *seed.xml*. Ten určuje, které verze bean budou ve výsledném klonu UIS.

Element `pathToPOM`

Podobně element `pathToPOM` specifikuje cestu k souboru *pom.xml* s úkoly Mavenu pro UIS, které lze spouštět z ErrorSeederu.

Element `templates`

Pro sestavení klonu UIS lze použít již předpřipravených šablon, ve kterých je specifikováno, jaká verze bude použita pro jakou beanu. Tyto šablony jsou specifikovány pomocí elementu `templates`. Ten nemá žádný atribut a obsahuje v sobě elementy `template` představující jednotlivé šablony. Implicitně ErrorSeeder obsahuje šablonu *All_OK*, obsahující pouze korektní verze, ale uživatel si může nadefinovat i své vlastní šablony.

Při specifikaci nové šablony stačí přidat nový element `template`. Ten musí mít určený atribut `name` se svým názvem a uvnitř elementy `versionId`. Jejich počet musí být roven počtu elementů `bean`. Obsah udává ID verze a to, který `versionId` se vztahuje ke které `bean`, je dáno pořadím.

4.3.1 Struktura elementu `bean`

Tento element obsahuje v rámci mnoha vnořených elementů vše potřebné pro popis jednotlivých injektovatelných bean.

Element `description`

Obsahuje základní popis beany.

Element `interfaceName`

Udává název rozhraní, kterému beana náleží.

Element `name`

Obsahuje název beany.

Element `versions`

Obsahuje v sobě jednotlivé verze, které mohou splňovat závislost dané beany. Vždy obsahuje jednu korektní a libovolný počet chybových verzí.

Element `version`

Jednotlivé verze, reprezentované elementem `version`, mají vždy unikátní atribut `id`, pomocí něhož lze pak na danou verzi odkazovat. Korektní verze obsahují čtyři elementy, chybové pět.

- `description` – udává popis verze pro uživatele, většinou obsahující informaci, zda je verze korektní, nebo chybová, případně popis dané chyby.
- `name` – představuje název, pod kterým se verze bude zobrazovat v Error-Seederu.
- `class` – obsahuje plný název dané třídy včetně balíků.
- `versionSeverity` – označuje chybovou závažnost verze a může nabývat hodnot `CORRECT`, `LOW`, `MEDIUM`, `HIGH` a `CRITICAL`, v závislosti na severitě dané verze.

Chybové verze – kromě výše uvedených – obsahují navíc ještě element `scenario`.

- `scenario` – popisuje scénář „ručního“ vyvolání chyby. Jednotlivé kroky scénáře jsou reprezentovány elementy `step`. Ty mají unikátní atribut `number` určující jejich pořadí a jejich obsahem je popis daného kroku.

```
<bean>
    <description>This bean serves teacher user business logic in whole
        application.</description>
    <interfaceName>teacherService</interfaceName>
    <name>Teacher Service</name>
    <versions>
        <version id="17">
            <description>This is correct version of Teacher Service bean.</
                description>
            <name>Correct Teacher service</name>
```

```
            <class>cz.zcu.kiv.matyasj.dp.service.users.correct.
            BaseTeacherService</class>
            <versionSeverity>CORRECT</versionSeverity>
        </version>
        <version id="18">
            <scenario>
                <step number="1">An unregistered user logs in as a teacher
                    strict.</step>
                <step number="2">Move to view \uv{My Exam Dates}.</step>
                <step number="3">Click on red button \uv{X} for deletion of
                     examination term.</step>
                <step number="4">Success message is given but exam date is
                    still in view and database.</step>
            </scenario>
            <description>This is error version of Teacher Service bean.
                This bean does not remove exam term from db or view.</
                description>
            <name>Incorrect Teacher service - does not remove exam term
                from db or view</name>
            <class>cz.zcu.kiv.matyasj.dp.service.users.error.
            E02TeacherService</class>
            <versionSeverity>HIGH</versionSeverity>
        </version>
    </versions>
</bean>
```

Ukázka kódu 4.2: Příklad definice beany v katalogu

Pokud uživatel potřebuje do ErrorSeederu doplnit další verzi, stačí jen do příslušného elementu `bean` přidat další element `version` s patřičně vyplněnými atributy a vnitřními elementy. Měl by si dát pozor, aby ji označil unikátním ID a aby dodržel jejich unikátnost napříč všemi verzemi.

5 Testování UIS

Aby mohlo UIS sloužit jako kvalitní prostředí pro testování, je velmi důležité, aby jeho srovnávací verze neobsahovala žádný defekt. Takového stavu bylo dosaženo důkladným testováním, které bylo prováděno průběžně během celého vývoje.

Provedeny byly dva druhy testů:

1. JUnit[1] testy vytvořené vývojářem UIS;
2. Funkcionální testy vytvořené vedoucím projektu.

Poznámka: Pro snazší funkcionální testování je aplikace UIS již přednastavena vhodnými testovacími daty. Toto přednastavení lze zobrazit z aplikace UIS *Home/Database content*.

JUnit testy jsou součástí aplikace UIS, zatímco pro funkcionální testy byl vytvořen separátní projekt se seleniovými[2] testy. K jejich provedení je tedy potřeba mít spuštěnou aplikaci.

Aplikace UIS je poměrně komplexní, jak lze vidět z ilustrativní ukázky stavů a přechodů aplikace v příloze D. Tomu odpovídá i rozsah testování. To je patrné z tabulky 5.1. Statistika v ní obsažená ukazuje, že testy samotné zabírají téměř třikrát více kódu než samotná aplikace. Je však důležité si uvědomit, že je to částečně způsobeno i tím, že část testů (jednotkové) je v samostatném projektu UIS. I přesto však lze tento poměr považovat za velmi vysoký.

Tato tabulka zobrazuje následující informace:

- počet souborů – počet souborů, převážně javovských, v případě UIS také `.jsp`
- velikost souborů – velikost souborů v kilobytech
- počet řádků celkem – počet všech řádků, včetně prázdných řádků, mezer, komentářů apod.
- počet řádků kódu – počet řádek, které obsahují kód

[1] JUnit – framework pro psaní jednotkových testů
[2] Selenium – framework pro testování webových aplikací

K získání těchto informací byl použit plugin Statistics pro IDE IntelliJ IDEA, kterým bylo možné dané údaje jednoduše zjistit. Tento plugin slouží pro přehledné zobrazení statistik projektu. Zobrazuje několik typů informací.

Ke každému typu souboru (například `.css` nebo `.java`) zobrazuje jejich počet, celkovou velikost všech těchto souborů v kilobytech, minimální a maximální velikost souboru tohoto typu v kilobytech, průměrnou velikost v kilobytech, celkový počet řádek kódu všech těchto souborů, minimální a maximální počet řádek souboru tohoto typu a průměrný počet řádek a mnoho dalších informací.

Kromě zobrazení celkových statistik pro typy souborů lze zobrazit statistiky i pro jednotlivé soubory zvlášť. Takto lze například zjistit celkový počet řádek nebo počty řádek kódu, komentářů a prázdných řádek a jejich procentuální zastoupení. Zároveň lze tyto statistiky zjistit i v součtu pro jednotlivé skupiny [22].

Pro vyhodnocení statistik byly použity pouze soubory, na kterých aplikace závisí.

	Počet souborů	**Velikost souborů**	**Počet řádků celkem**	**Počet řádků kódu**
Aplikace UIS				
Java	87	340	8 550	4 708
JSP	18	94	1 550	1 477
celkem	105	434	10 100	6 185
Testy				
JUnit	33	277	6 945	4 690
funkcionální	199	583	17 282	12 748
celkem	232	860	24 227	17 438
poměr testy/UIS	2,21	1,98	2,40	2,82

Tabulka 5.1: Porovnání aplikace UIS a jejích testů

5.1 Aplikace funkcionálních testů

Jak již bylo řečeno výše, pro seleniové testy funkcionálního charakteru je vyčleněna zvláštní aplikace. To, že je počet testů i v rámci této aplikace opravdu vysoký, lze vypozorovat ze statistiky uvedené v tabulce 5.2.

5.1.1 Balíky aplikace

Tato aplikace je členěna do balíků `support` a `tests`. Balík `support` obsahuje zejména pomocné třídy pro spouštění seleniových testů, konfigurační soubory, soubory s výchozími daty a javovské entity pro tato data. Balík `tests` pak již obsahuje jednotlivé testy aplikace UIS a tyto testy jsou rozděleny do dalších balíků v závislosti na jejich typu.

Balík `passive` je vyčlenen pro pasivní testy, které přímo nevykonávají žádné akce v aplikaci, ale pouze kontrolují správné zobrazení dat a prvků na obrazovce. Lze je na základě zaměření testů rozdělit do několika různých typů.

Prvním jsou testy zabývající se obsahem obrazovek, ty testují zobrazení jednotlivých elementů na obrazovce (tlačítka, nadpisy, tabulky, ...). To jsou obecně časově nejméně náročné pasivní testy. Jejich speciální podmnožinou jsou pak testy na obsah modálních oken, ty testují zobrazení jednotlivých elementů v modálních oknech.

Dalším typem jsou pak testy kompletního obsahu přednastavené databáze. Ty testují pouze výchozí nastavení databáze a správné zobrazení těchto databázových dat (např. zda jsou na předmětu zapsaní správní studenti). Ani ony však neprovádí žádné akce v aplikaci UIS. Jejich speciálním případem je opět testování obsahu databáze zobrazovaného v modálních oknech. To je obecně časově nejnáročnější typ pasivních testů.

Balík `active` obsahuje aktivní testy, které kontrolují správné provedení akcí v aplikaci. Dle složitosti je lze rozdělit do následujících kategorií. První tvoří testy, které provádí pouze jednu akci. Jejich významnou podmnožinou jsou pak testy kontrolující mezní hodnoty (např. situace, kdy si student odepíše svůj jediný zapsaný předmět). Další kategorií jsou komplexnější testy, v rámci kterých se provádí komplexnější scénář akcí, často ve střídající se roli učitele i studenta.

Balík `negative` zahrnuje sadu negativních testů, které kontrolují správné negativní vyhodnocení špatných uživatelských vstupů.

Některé z těchto testů byly v rámci této diplomové práce upravovány, ovšem změny byly natolik minoritní, že zde nebudou dále popisovány.

	Počet souborů	Velikost souborů	Počet řádků celkem	Počet řádků kódu	Počet testů
`support`	40	223	5 854	4 053	–
`passive`	79	165	5 222	3 840	456
`active`	54	151	4 812	3 794	548
`negative`	16	33	1 066	861	63
celkem	189	572	16 945	12 546	1 067

Tabulka 5.2: Statistiky testů pro jednotlivé balíky

5.1.2 Reprezentace dat

Vzhledem k tomu, že testovací aplikace není přímo součástí UIS, má jistá specifika, jako například způsob načítání výchozích dat nebo vlastní reprezentace entit.

Aplikace nepoužívá databázi a výchozí data jsou načítána ze souborů a uchovávána v paměti. To s sebou přináší určité výhody i nevýhody. Výhodou je bezesporu menší technologická zátěž, kdy kompletně odpadá nutnost konfigurace, správy a komunikace s databází a využívání dalších technologií. Nevýhodou je velmi náročná správa těchto souborů s daty. Při úpravách je totiž potřeba tyto změny ve správném formátu začlenit na správné místo. Takový způsob je už ze své podstaty velmi náchylný na lidské chyby. Pokud je struktura souboru jakkoliv porušena, aplikace jej nenačte a oznámí, že tyto soubory byly poškozeny. Vývojář se však nedozví, v čem přesně chyba spočívá a musí tak pozorně projít všechny své změny.

V aplikaci jsou jednotlivé entity reprezentovány vlastními, jinými třídami. A to má také svá pozitiva i negativa. Mezi hlavní výhodu patří nezávislost obou dvou kódů na sobě, kdy by mohla např. testovací aplikace přijmout zanesenou chybu od testované aplikace. Ovšem postupem času se ukázala spíše nevýhoda nedůsledné datové reprezentace entit v testech, což přináší problémy zmíněné dále. V ideálním případě by měly obě aplikace využívat stejné struktury tříd. Pokud by tedy v současné realizaci došlo k nějaké úpravě entit UIS, bylo by zapotřebí tutéž změnu zanést i do entit testovací aplikace.

V současné době se však třídy představující ty samé entity v obou aplikacích od sebe navzájem liší. Některé aspekty UIS pak není možné vůbec testovat, protože buď v testovací aplikaci chybí příslušné atributy entit, které by danou informaci nesly, nebo jsou špatně použité a danou informaci tak nelze v pravou chvíli získat. Navíc je tato „dvojí implementace“ velmi

náchylná na chyby v rámci samotné testovací aplikace.

Hlavní problém současné verze testovací aplikace však spočívá ve špatně navržené architektuře, kvůli které ani není možné bez velkých zásahů entity v obou aplikacích sjednotit.

Příkladem entity, jenž je v obou aplikacích reprezentována odlišně, což vede k problémům, je `ExaminationDate` v UIS a její nejbližší protějšky `ExamDates` a `ExamDatesGrades` v popisované aplikaci s testy. `ExaminationDate` je komplexní entita, obsahující informace o datu konání zkoušky, předmět, pořádajícího učitele, seznam účastnících se studentů a maximální počet účastníků. Oproti tomu `ExamDates` i `ExamDatesGrades` představují úplně jiný přístup k tomuto subjektu – termínu zkoušky. `ExamDates` obsahuje atribut představující termíny zkoušky vztahující se k jednomu předmětu, který je v podobě pole `řetězců`. Jako vnitřní statickou třídu má `ExamDates` výčtový typ `Term` obsahující jako hodnoty řetězce `FIRST`, `SECOND`, `THIRD`. `ExamDates` tedy představuje seznam zkouškových termínů, které jsou představovány řetězci výčtového typu `Term`. Třída `ExamDatesGrades` spojuje jednotlivé položky pole řetězců z `ExamDates` a položky výčtového typu `Grade`. Jak lze vidět z této ukázky, v testovací aplikaci chybí u zkouškového termínu například informace o datu konání zkoušky nebo seznam účastnících se studentů. Tyto informace pak při testování chybí.

Podobným případem je například i entita `Student`, která má totožný název jak v aplikaci UIS, tak v aplikaci s funkcionálními testy. V aplikaci UIS má tato entita dva atributy, a to seznam instancí třídy `Subject` představující aktuálně studované předměty a seznam předmětů absolvovaných. Naopak v aplikaci s funkcionálními testy obsahuje jednu mapu, jejímž klíčem jsou předměty studenta a hodnotou instance třídy `ExamDatesGrades`. Z toho vyplývá, že je v aplikaci s funkcionálními testy problém u studentových předmětů zjistit různé pokročilé informace o daném předmětu, jako je například seznam vyučujících nebo kreditové ohodnocení. Tyto informace nejsou těmito testy ověřovány.

Řešením tohoto problému by mohlo být například začlenění testů přímo do aplikace UIS s využitím již existujících databázových entit. Databáze by v tomto případě mohla buď zůstat stejná, což by ale mohlo způsobovat potíže s integritou, nebo v lepším případě by byla vytvořena ještě jedna databáze s identickou strukturou pro testy, která by byla využita vždy při testování a před jeho započetím naplněna správnými daty. Tato data by nebyla v běžných textových souborech jako doposud, ale byla by součástí nahrávaných SQL skriptů.

Druhou možností řešení by byla změna současné architektury a implementace všech entit přesně tak, jak jsou v původní aplikaci testbedu UIS, tedy

takzvaně „jedna ku jedné“. Tento přístup s sebou přináší výhodu oddělení obou aplikací, protože jejich účel je jiný. Aplikace UIS by samozřejmě měla být opatřena vlastními testy, které by byly její součástí. Tento požadavek je však již splněn přítomností jednotkových testů v testbedu UIS. Oproti tomu tyto funkcionální testy, které obsahuje aplikace s testy, mají jiný charakter. Jejich účelem je simulovat aplikace, jenž budou na testbedu UIS zkoušeny. Z tohoto důvodu je tak toto řešení vhodnější než dříve popsané. I přesto však má i své nevýhody. Tou je především již popsaná nutnost tuto aplikaci a její entity aktualizovat vždy, když se tak stane i v případě testbedu.

Třetí možný přístup k řešení této situace je nejvíce zachovávající současný stav aplikace s testy. Je jím rezignace na kompletní pokrytí všech případů užití. Tento přístup vychází především z úvahy, že pro kompletní otestování testovacího nástroje není potřeba všechny tyto případy užití projít. Z tohoto důvodu je preferován před ostatními možnostmi.

6 Návrh TbUIS – WEB

Tato část se zabývá návrhem webu reprezentující celý projekt TbUIS.

6.1 Požadavky a způsoby zveřejňování

Účelem webové aplikace pro testbed UIS je převážně prezentace celého projektu a poskytnutí všech důležitých informací a podkladů k bezproblémovému používání jeho aplikací.

To znamená konkrétně tyto sekce:

- obecné informace o projektu, jeho účel, součásti a vývojový tým, odkaz na univerzitní výzkumnou skupinu,
- detailní popis testbedu UIS, použitých technologií aplikace, případy užití, ukázka dostatečné komplexity aplikace, návod k sestavení a použití, včetně databáze, odkaz na běžící instanci,
- analýza testování UIS, popis funkcionálních testů,
- informace o ErrorSeederu, technologické požadavky, návod k sestavení a použití,
- možnost stažení obou aplikací včetně data jejich vydání, v případě UIS i starších verzí, ideálně i vybraných poruchových klonů včetně informace, ze kterých verzí entit jsou tyto klony sestaveny.

Jeden nepovinný požadavek:

- začlenění jednoduchého redakčního systému (umožňující určitým uživatelům přidávat články s aktuálními informacemi o UIS a větší skupině uživatelů tyto články komentovat).

Mezi požadavky plánované v rámci budoucích rozšíření webové aplikace patří:

- informace o relevantních publikacích na toto téma (především výstupy z konferencí, bakalářské a diplomové práce),
- odkazy na příbuzná témata,

- historie nahraných souborů,
- ukázky aplikace UIS včetně testů a jejich výsledků,
- diskusní fórum otevřené všem registrovaným uživatelům,
- slovníček terminologie obsahující základní pojmy.

Na základě těchto požadavků byla navržena aplikace obsahující pět hlavních obrazovek, konkrétně nazvané *About*, *Download*, *UIS*, *ErrorSeeder* a *News*, a další dvě obrazovky *Registration* pro registraci nových uživatelů a *Login* pro přihlášení.

6.2 Technologie

Během vytváření prototypu webové aplikace bylo vyzkoušeno několik technologií, a to konkrétně Node.js, Nette, Symfony a čisté PHP, a postupně byly také zjišťovány problémy. Níže je uveden přehled všech těchto technologií s hodnocením vhodnosti výběru pro implementaci takovéto aplikace, případně důvody jejich nepoužití.

6.2.1 Node.js

První verze webu pro TbUIS byla tedy vypracovávána jako MEAN stack, tj. kombinací moderních technologií MongoDB, Express.js, Angularu 6 a Node.js.

Node.js je platforma navržená pro psaní vysoce škálovatelných internetových aplikací, především webových serverů. Používá událostmi řízený, neblokující vstupně/výstupní (dále I/O) model, díky němuž se hodí pro datově náročné real-time aplikace, běžící na distribuovaných zařízeních. Node.js je postaven na renderovacím jádře užívající programovací jazyk JavaScript. Díky tomu aplikace napsané v této platformě využívají podobnou syntaxi jako frontendové JavaScriptové aplikace, včetně objektů a funkcí.

Express.js je webový framework pro Node.js, který se používá k vývoji aplikace na serverové straně. Samotný Node.js totiž nebyl původně vyvinutý pro vytváření webových stránek a neobsahuje tak pro tuto potřebu dostatek funkcionality. Tu Express.js dodává jako tzv. middleware. To je v obecné rovině software, který poskytuje funkce, jež nejsou běžně poskytované původní platformou. Například zde se jedná o již vytvořené komponenty JavaScriptu.

Angular 6 je framework vycházející z JavaScriptu. Umožňuje tvorbu zejména frontendu v HTML, JavaScriptu a TypeScriptu. Jeho předchůdci

jsou AngularJS (někdy též Angular 1), Angular 2, Angular 4 a Angular 5. Nejblíže má k poslednímu jmenovanému, se kterým je zpětně kompatibilní.

MongoDB je dokumentově orientovaná NoSQL databáze. K ukládání využívá dokumentů, které mají podobnou strukturu jako JSON.

Vzhledem k tomu, že NoSQL databáze se pro navrhovaný web příliš nehodí a na univerzitních produkčních serverech se MongoDB a Node.js nepoužívá, bylo nutné tyto technologie opustit. Řešením by mohl být vlastní virtuální server s těmito technologiemi, ale jeho administrace by byla příliš náročná a nevýhody by tak převyšovaly přínosy. Musely být zvoleny jiné technologie, které jsou pro univerzitní produkční server vhodné, konkrétně PHP a MySQL.

6.2.2 Nette

Nette je český framework pro psaní webových aplikací v PHP. Slouží především pro tvorbu událostmi řízených aplikací podle architektury Model-View-Presenter (MVP). Je silně inspirováno Ruby on Rails, Djangem a Springem.

V Nette byla tedy vytvořena ukázková aplikace. Protože je však byl tento framework moc rozsáhlý a v mnoha ohledech nedokonalý, bylo nakonec rozhodnuto, že bude použit framework Symfony, z něhož je možné použít pouze nutné komponenty.

6.2.3 Symfony

Symfony je druhý nejrozšířenější framework pro vývoj webových aplikací v PHP. Skládá se z jednotlivých komponent, které však lze použít i nezávisle na tomto frameworku. Využívá architektury Model-View-Controller (MVC). Mimo mnoha praktických komponent obsahuje funkcionality jako šablonování, automatickou validaci formulářů, cache nebo podporu jazykové lokalizace.

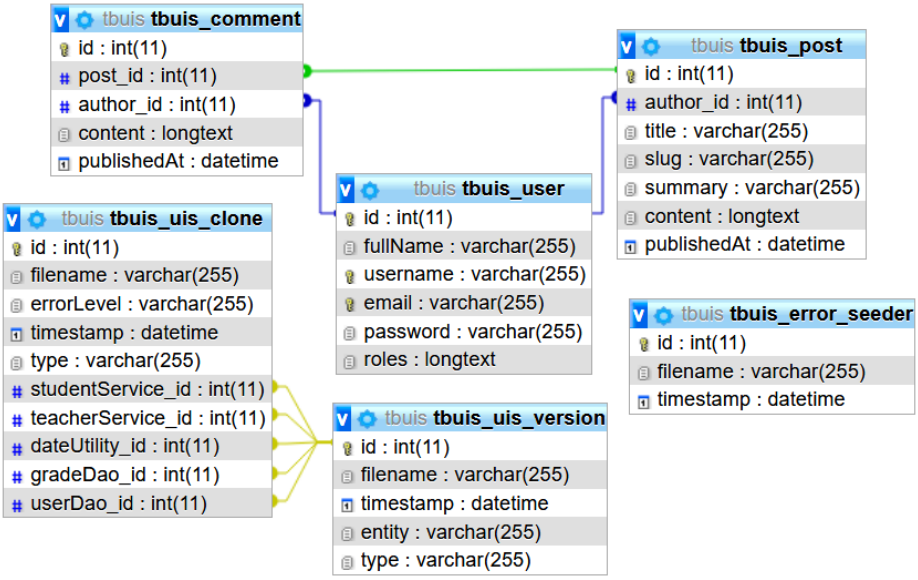
V průběhu implementace aplikace ve frameworku Symfony bylo rozhodnuto vedoucím práce, že zadání implementace webu bude splněno současně psanou bakalářskou prací *Mikroframework na bázi komponent Symfony pro publikování vědeckých projektů*. Dokončení aplikace v Symfony by tedy bylo plýtvání prostředky a bylo tedy rozhodnuto, aby byl tento bod zadání splněn ve statické podobě s nutnými případy užití.

Web projektu pomocí Symfony byl z převážné části realizován, ale z výše uvedených důvodů tato implementace nakonec nebyla dále rozvíjena a ani použita. I přesto je níže popsán model databáze, jelikož se jedná o část projektu, která byla shodná pro všechny předchozí implementace i v jiných

technologiích. Tento model lze využít při dalším rozvoji aplikace.

Databáze

Pro realizaci databáze dynamického webu bylo zvoleno MySQL a jako nástroj správy phpMyAdmin ve verzi 4.8.2. Na základě zadaných případů užití bylo navrhnuto šest tabulek a to tak, aby schéma bylo co nejjednodušší. Jejich model je znázorněn na obrázku 6.1.



Obrázek 6.1: Navržený databázový model

Tabulky `tbuis_uis_clone` a `tbuis_uis_module` obsahují informace o nahraných souborech – jednotlivých klonech a verzích. V případě verzí jsou ukládanými informacemi název souboru (`filename`), čas nahrání (`timestamp`), příslušná entita (`entity`) a typ (`type`; korektní, nebo chybový). Klon ukládá podobné infromace, tedy název souboru, čas nahrání a typ. Kromě toho však obsahuje i stupeň chybovosti (`errorLevel`) a sloupce s odkazy na všechny verze, jenž jsou v klonu obsaženy (`studentService`, `teacherService`, `dateUtility`, `gradeDao`, `userDao`), protože tato informace bude u každého klonu zobrazena. Podoba této tabulky je tak silně závislá na počtu injektovatelných bean.

Obdobně jsou v tabulce `tbuis_error_seeder` uloženy informace o nahraných `.war` souborech ErrorSeederu. Té však stačí shromažďovat informace pouze o názvu souboru (`filename`) a datu nahrání (`timestamp`).

Tabulka `tbuis_user` sdružuje jednotlivé uživatele a informace o nich potřebné, jako je celé jméno (`fullName`), uživatelská přezdívka (`username`),

e-mail (`email`), zahashované heslo (`password`) a seznam rolí (`roles`).

Tabulka `tbuis_post` ukládá informace o jednotlivých příspěvcích od uživatelů. Obsahuje tak sloupce s názvem (`title`), stručným shrnutím (`summary`) a obsahem příspěvku (`content`), datem přidání (`publishedAt`) nebo jako cizí klíč ID uživatele, jenž příspěvěk přidal (`author_id`).

Na ní poté závisí tabulka `tbuis_comment` s informacemi o komentářích k příspěvkům. Shromažďuje pouze nejdůležitější atributy, a to obsah komentáře (`content`), datum zveřejnění (`publishedAt`) a cizí klíče ID autora (`author_id`) a ID příspěvku (`post_id`), ke kterému se vztahuje.

6.2.4 Čisté PHP 5 + Twig

Aby byla výsledná implementace webu co se týká technologií co nejjednodušší, bylo rozhodnuto, že nebude použit žádný framework a stránky budou pouze statické, s využitím PHP a šablonovacího systému Twig. Pro zachování použití PHP bylo rozhodnuto zejména z důvodu nutnosti routování. Navíc lze samotné PHP bez problémů nasadit na školní produkční server. Aby bylo vytváření jednotlivých obrazovek co nejefektivnější, byl vybrán šablonovací systém Twig. Pro stylování bude použit Bootstrap a vlastní CSS. Případné ikony pro menu a další použití budou propůjčeny ze sbírky vektorových ikon Font Awesome.

Twig je open-source šablonovací systém pro PHP, který vhodně odděluje aplikační vrstvu od prezentační a není tedy potřeba používat PHP přímo v HTML kódu. To je využitelné především v aplikacích tohoto typu, kdy je PHP použito pouze na routování. Syntaxe Twigu je inspirována šablonovacími jazyky Django nebo Jinja. K oddělení HTML od kódu pro Twig využívá dvou typů oddělovačů – `{% ... %}` a `{{ ... }}`. Oba však mají rozdílné způsoby použití. První způsob se používá pro vykonání svého obsahu a druhý pro zobrazení hodnoty svého obsahu. Ve vyvíjeném projektu bude použito především prvního způsobu pro vkládání obalových šablon, jako je menu nebo patička.

6.3 Lokalizace

Po důsledném uvážení bylo rozhodnuto, že lokalizace celého webu bude pouze v angličtině, jelikož je testbed UIS určen pro vývojáře různých národností. Možná by byla i implementace zároveň v několika jazycích najednou, ale v tom případě by vzrostla režie a čas nutný pro případnou úpravu webu. To je však ve chvíli, kdy je web pouze v angličtině postačující, zbytečné.

6.4 Vývojové prostředí

Web TbUIS bude vyvíjen v prostředí PhpStorm od společnosti JetBrains, a to ve verzi 2018.3 pod studentskou licencí. PhpStorm poskytuje podporu jazyku PHP včetně jeho frameworků, jako je například Symfony. Mimo to je určený i pro vývoj v HTML, CSS, Sass, Less, JavaScriptu, Angularu a MySQL [23]. Z verzovacích nástrojů podporuje například Git, SVN, CVS nebo Mercurial [24].

7 Implementace webu TbUIS

Poznámka: Celá tato část bude popsána stručně, neboť v průběhu řešení bylo rozhodnuto, že web TbUIS bude řešen jinou kvalifikační prací. Přesto byla vytvoření prototypu webových stránek věnována relativně velká pozornost, byť bylo zřejmé, že stránky budou pouze dočasné. Ovšem stránky byly zprovozněny zhruba v září 2018, čili až do zprovoznění jiné realizaci slouží jako hlavní informační zdroj o projektu. Vedlejším efektem vytvoření stránek bylo, že pro ně bylo nutné doplnit realistický obsah, který byl potom snadno převzat do jiné realizace stránek.

7.1 Případy užití

Výsledná implementace webu je statická, umožňující uživateli se dozvědět základní informace o projektu, stáhnout si aktuální verzi ErrorSeederu, různé verze korektních klonů UIS a několik vybraných kombinací aktuálních verzí chybových klonů UIS, u kterých jsou informace o použitých verzích entit, získat detailní informace o UIS včetně statistik testů a přečíst si návod na použití ErrorSeederu.

Konkrétně tedy byly implementované obrazovky *About*, *Download*, *UIS* a *ErrorSeeder*.

7.2 Struktura

Výsledný projekt je rozdělen do několika hlavních částí:

- adresář `files` – obsahuje nahrané soubory všech typů, tedy aktuální verzi ErrorSeederu a korektní i chybové verze klonů UIS
- adresář `img` – obsahuje všechny obrázky na webu obsažené, tedy především obrázky týkající se UIS a návodu k ErrorSeederu
- adresář `static` – obsahuje stylování, konkrétně například zdrojové soubory Bootstrapu, CSS a Font Awesome
- adresář `templates` – obsahuje šablony Twigu pro jednotlivé obrazovky
- soubor `index.php` – PHP skript zajišťující routování

- soubor `composer.json` – soubor týkající se správy závislostí aplikace, obsahující závislost na knihovny šablonovacího systému `twig`

7.3 Způsob implementace

Během implementace bylo potřeba respektovat omezení daná katedrálním serverem, a to jak bezpečnostní, tak technologické, jako například verze PHP nebo chybějící podpora jiných technologií.

Web byl implementován v PHP bez pomoci frameworků, pouze s využitím šablonovacího nástroje Twig.

To znamená, že webové stránky jsou relativně jednoduché, ale od září 2018 plnily svůj účel a rovněž byly začleněny do stránek výzkumné skupiny. Ukázky jednotlivých obrazovek jsou v příloze C.

8 Příprava další verze UIS

Poznámka: Pokud bude dále zmiňován programovací jazyk Java, bude se vždy jedna o verzi 11.

Aplikace UIS v původní verzi byla během jejího vývoje průběžně testována. Ovšem testy většinou přicházejí o něco později než vývoj a došlo tedy k situaci, kdy bylo z důvodu termínu dokončení diplomové práce nutné vývoj verze 1.0 aplikace UIS ukončit [2]. Po dokončení sady funkcionálních testů bylo zjištěno, že aplikace stále obsahuje chyby, případně určitá funkčnost chybí. Na tuto situaci bylo nutné reagovat, to znamená zjištěné chyby odstranit a chybějící funkčnost doplnit. Dalším zdrojem námětů pro změny byla souběžně probíhající bakalářská práce [8].

Poznámka: Celá tato část nebyla součástí zadání diplomové práce.

8.1 Problémy s Javou 11

Aby byla aplikace UIS napsána v nejnovějších technologiích, bylo rozhodnuto o její migraci z Javy 8 na Javu 11. Při tom však bylo nutné vyřešit nekompatibilní verze některých knihoven. Konkrétně bylo potřeba zvýšit verzi knihovny *Hibernate* – 5.2.16.Final na verzi – 5.4.1.Final a *Log4j* z verze – 1.2.17 na verzi – 2.11.1. Dále knihovnu *com.sun.xml.bind* vyměnit za *javax.xml.bind* a *org.glassfish.jaxb* oboje verze – 2.3.0. Samotný kód aplikace s výjimkou logování nebylo potřeba měnit. Ale i změny v rámci logování byly minoritní, jak lze vidět z jejich popisu v části 8.3.

8.2 Opravy chyb

Tato část obsahuje výčet nejdůležitějších oprav a doplnění, přičemž minoritní opravy a změny nejsou uváděny.

Poznámka: Hned na prvním příkladě je vidět, jaké může mít zdánlivá nefunkční „maličkost“ rozsáhlé souvislosti.

8.2.1 Hodnocení F

Během doplňujícího testování bylo zjištěno, že pokud se studentovi změní známka z A na F, předmět stále zůstává v kategorii splněných a kredity

jsou stále započítané. Zároveň je předmět po změně hodnocení započítaný i zobrazený ve splněných hned dvakrát. Jediné k čemu skutečně dojde, je odstranění studenta ze zkouškového termínu. To je však ale nechtěná vlastnost, protože ačkoliv student neprospěl, byl na termínu přítomen.

Při očekávaném korektním chování by byl po změně hodnocení z A na F studentovi předmět odebrán z absolvovaných, včetně odečtení příslušného počtu kreditů, a přidán do seznamu studovaných. Zároveň by měl student i nadále zůstat účastníkem daného zkouškového termínu.

Tato zdánlivě jednoduše odstranitelná chyba při bližším zkoumání odhalila nedokončené řešení známky F, jenž má podstatně rozdílné chování, než ostatní známky. Zároveň nebylo počítáno s tím, že student může absolvovat více termínů (pokud dostane známku F, může jít na další).

Řešení tedy vyžadovalo rozsáhlejší zásah do aplikace. Bylo potřeba přidat kontroly, zda daná známka znamená, že student prospěl, či nikoliv. Dále bylo potřeba přidat podporu účasti studenta na více termínech. Ke každé známce byl přidán jednoznačný identifikátor termínu, neboli ID termínu, z něhož známka pochází, aby byla známka svázaná nejen s předmětem a studentem, ale i s konkrétním datem. To je důležité hlavně proto, aby student mohl jít na více termínů, případně pokud je potřeba známku změnit. V souvislosti s tím bylo potřeba rozlišit studenty, kteří mají předmět absolvovaný, a studenty, kteří z daného předmětu obdrželi známku. Toto rozdělení se následně promítlo do funkcionality mnoha případů užití, jako například zobrazení studentů, kteří z daného předmětu obdrželi nějakou známku.

S tím souvisí, že si v současné implementaci student nemůže zobrazit kompletní seznam známek, které obdržel. Může si sice zobrazit seznam absolvovaných předmětů včetně hodnocení, pokud však ale získá známku F, nemá možnost si tuto informaci nikde zobrazit. Tato chybějící funkcionalita je tak vhodným námětem pro budoucí rozšíření UIS.

8.2.2 Seznamy studentů zapsaných na předměty

Z důvodu neúplné analýzy byla v původním UIS opomenuta funkčnost, která dovoluje učiteli zjistit, jací studenti mají zapsán jeho předmět. Z tohoto důvodu bylo potřeba na obrazovce *List of Taught Subjects* zobrazující seznam vyučovaných předmětů ke každé položce přidat možnost si zobrazit modální okno s účastníky příslušného termínu.

Z toho důvodu bylo nutné rozšířit patřičný `.jsp` soubor o funkčnost otevírání modálního okna, které obsahuje tabulku se všemi účastníky.

8.3 Logování

V původní verzi UIS bylo logování sice použito, ale ve formě, která nebyla kompatibilní s nově vzniklými požadavkami a ani kompatibilní s použitým logováním funkcionálních testů. Mezi vzniklé požadavky na logování patří například nová struktura konfiguračního souboru v podobě XML, logování maximálního počtu informací v levelu `INFO` a zapisování z poruchových verzí do levelu `ERROR`. Nové logování, které bylo implementováno, tyto požadavky splňuje. K tomuto účelu byla využita knihovna Log4j. Původně se jednalo o verzi Log4j 1, ale nyní se z důvodu migrace a celkových upgradů na Javu 11 využívá vyšší verze Log4j 2.

Rozdíl mezi těmito verzemi spočívá především k přístupu k loggeru. Ve starší verzi Log4j 1 bylo k loggeru přistupováno voláním `Logger.getLogger(getClass())`, v novější verzi je k jeho získání potřeba využít `LogManageru`, konkrétně pomocí syntaxe `LogManager.getLogger()`. Tato změna je však minoritní a vyjma tohoto se projeví pouze v příslušných importech.

8.3.1 Konfigurační soubory

Konfigurace této knihovny může být napsána ve formátech XML, JSON, YAML a properties [25]. V původní verzi měl konfigurační soubor logování pro UIS podobu právě properties, konkrétně se jednalo o soubor *log4j.properties*. Vzhledem k tomu, že aplikace s funkcionálními testy používá formát konfiguračního souboru XML a formát properties nepodporuje všechny pokročilé způsoby konfigurace [26], bylo rozhodnuto o změně formátu konfiguračního souboru pro logování UIS na XML, a tedy i sjednocení formátů konfiguračních souborů napříč projekty. Nový konfigurační soubor se nachází v projektovém adresáři *src/main/resources* a nese název `uis-logger-config.xml`.

Tento konfigurační soubor v podobě XML má kořenový element `Configuration`. Ten obsahuje tři elementy `Properties`, `Appenders` a `Loggers`. Element `Properties` sdružuje elementy `Property` představující případné konstanty, které mohou být v konfiguračním souboru použity. Další element `Appenders` specifikuje, kam má být logování prováděno. V současné implementaci je logováno na konzoli a do souboru. Logování na konzoli je zajištěno přítomností elementu `Console`, který má definovaný název, cíl a v rámci elementu `PatternLayout` i podobu logování. Logování do souboru obstarává element `File`. Ten má specifikovaný název, cestu k souboru, informaci, zda se má daný soubor vždy přepsat, nebo nové logování připisovat na konec, a v rámci elementu `PatternLayout` opět specifikovanou podobu výpisu. Po-

slední element `Loggers` pak specifikuje pro jaké úrovně logování se má použít jaký appender.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Properties>
        <Property name="pFileName">./uis-log.txt</Property>
        <Property name="pFormat">%-5level %c{1}:%L - %msg%n</Property>
    </Properties>
    <Appenders>
        <Console name="screen" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} -- ${
                pFormat}"/>
        </Console>
        <File name="file" fileName="${pFileName}" append="false"
              immediateFlush="true">
            <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} -- ${
                pFormat}"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="INFO">
            <AppenderRef ref="file"/>
            <AppenderRef ref="screen"/>
        </Root>
    </Loggers>
</Configuration>
```

Ukázka kódu 8.1: Příklad konfiguračního souboru

Pokud by tedy v rámci budoucího vývoje UIS vznikla potřeba logovat do dalšího souboru, stačí pouze pod element `Appenders` přidat nový `File` s patřičně vyplněnými atributy a vnitřním elementem a odkaz na něj uvést v elementu `Loggers` pod správnou úrovní logování.

8.3.2 Logování v rámci aplikace

Logování bylo přidáno do všech vrstev aplikace. V rámci kontrolerů jsou logovány požadované akce, včetně případných ID a dalších informací o příslušných objektech. V beanech Student Servis a Teacher Servis jsou logovány všechny prováděné akce, opět včetně všech potřebných informací. V případě chybových verzí jsou logovací zprávy brány z aplikačního konfiguračního souboru *application.properties*, jenž se nachází v projektovém adresáři *src/main/resources*. Formát properties v tomto případě není poblematický, protože se jedná jen o informace v podobě klíč-hodnota. Ve vrstvě DAO jsou pak logovány všechny úkony, které jsou s databázovými daty prováděny,

včetně počtů navrácených dat při příkazu select.

Pokud by v budoucnu došlo k rozšíření aplikace UIS o další třídy, bylo by vhodné opatřit logováním i je. Pro to stačí pouze použít instanci `Logeru`, která lze získat voláním `LogManager.getLogger()`.

V současné verzi konfiguračního souboru `uis-logger-config.xml` je nastaveno, že všechna logování jsou zapisována do souboru `uis-log.txt`. Ten se nachází v adresáři *bin* v domovském adresáři Tomcatu, který je spouštěn. Případné změny nastavení by byly provedeny v konfiguračním souboru `uis-logger-config.xml`

8.4 Vývojové prostředí

Aplikace UIS bude dále vyvíjena v prostředí IntelliJ IDEA od společnosti JetBrains, konkrétně v jeho verzi 2018.3 pod studentskou licencí. IntelliJ IDEA a její pluginy poskytují podporu mnoha programovacím jazykům, jmenovitě například Javě, Groovy nebo Kotlinu, a frameworkům, jako je Spring [27]. Mimo to poskytuje uživatelské rozhraní verzovacím systémům jako jsou Git, SVN, CVS nebo Mercurial [28].

9 Nové chybové verze

Původní UIS aplikace obsahovala celkem sedm chybových verzí (jejich přehled je uveden v tabulce 3.1). Pro rozsáhlejší experimenty však tento počet není dostatečný. Z tohoto důvodu bylo potřeba vytvořit další chybové verze, a to tak, aby bylo možné vytvořit alespoň deset různých chybových klonů aplikace.

Zároveň by měly nově injektované chyby dodržovat určitá pravidla či omezení. Ty vyplývají jednak ze stávajícího funkcionálního testování seleniovými testy, a pak také generovanými funkcionálními testy z výše zmíněné bakalářské práce [8]. To znamená například, že injektovaná chyba by neměla narušovat URL, aby bylo možné dokončit zamýšlený test, tj. spuštěné testy nesmí být předčasně ukončeny chybou. Tato pravidla a omezení sice mohou být vědomě porušena, avšak při vytváření poruchových klonů na ně musí být brán zřetel.

9.1 Postup vytvoření chybové verze

Jak již byl uvedeno v části 3.4, architektura UIS je navržena tak, aby bylo možné vytvářet různé verze bean na DAO, Utility a Servisní vrstvě. To znamená, že implementace chybových verzí s injektovanými chybami, jenž vycházejí z těchto vrstev, by neměla být problém. Příkladem takovýchto „snadno implementovatelných“ chyb může být například změna v databázových či zobrazovaných datech.

9.1.1 Podrobný návod pro jednoduchou chybu

Pro detailní postup vytvoření entity byl vybrán způsob z části 3, kdy chybové verze dědí od korektní verze. Postup vytvoření standardní chybové verze, která mění výstup již vytvořených metod korektní verze, vypadá následovně:

1. Vytvořit v balíku `error` příslušné injektovatelné beany novou třídu – verzi, která bude mít unikátní název, jehož jmenná konvence je `E<pořadí><název_beany>`. Ta bude dědit od korektní verze.

2. Rovněž bude vytvořen stejný konstruktor kvůli závislostem rodiče. Konstruktor bude označen anotací `@Autowired`.

3. Injektování chyby se dosáhne tím, že se přepíše metoda, jejíž funkčnost je nutné změnit. Taková metoda bude označena anotací `@Override` a také `@ErrorMethod`, která obsahuje krátký popis, jenž chybovou funkcionalitu vystihuje.

4. Metodu je rovněž nutné doplnit logováním, aby bylo z logu patrné, že byla provedena zanesená chyba.

9.1.2 Podrobný návod složitější chyby

Do UIS je však vyjma výše popsaného typu chyb potřeba zanášet i chyby jiného charakteru – tím jsou myšleny zejména chyby na frontendu jako špatné nadpisy obrazovek, chybějící sloupce v tabulce a nebo špatně odkazující tlačítka. Pro tyto chyby by bylo ideální měnit buď příslušný kontroler nebo `.jsp` soubor. To však není při současné implementaci UIS možné a bylo tak nutné hledat jiná řešení.

Nejsnazší možností, jak tyto chyby zanášet, je využít současných možností UIS a v Servisách vytvořit metody, jejichž návratové hodnoty budou ovlivňovat dění na frontendu. Při současném stavu, kdy není možné vytvářet chybové verze kontrolerů nebo `.jsp` souborů, je toto jediný způsob, jak chyby tohoto typu injektovat.

Vytváření nových chybových verzí s jinými typy chyb tedy vyžadovalo rozsáhlejší zásah do aplikace a proto bylo nutné vykonat více změn. Tímto došlo ke změně hned v několika souborech. Opět byl vybrán způsob, kdy chybové verze dědí od korektní verze. Postup by se dal obecně popsat následovně:

1. Vytvořit v balíčku `error` příslušné injektovatelné beany novou třídu – verzi, která bude mít unikátní název, jehož jmenná konvence je `E<pořadí><název_beany>`. Ta bude dědit od korektní verze.

2. Rovněž bude vytvořen stejný konstruktor kvůli závislostem rodiče. Konstruktor bude označen anotací `@Autowired`.

3. V rozhraní, kterým je implementována rodičovská třída, se nadefinuje hlavička nové metody. Ta v nejzákladnější podobě vrací booleovskou hodnotu a tímto určuje, zda bude v kontroleru zanesena chyba a nebo nikoliv.

4. V rodičovské třídě bude naimplementována tato metoda s chováním, kterým nezanese žádnou chybu.

5. V nově vytvořené třídě bude tato metoda přepsána, aby implikovala chybovost. Bude označena anotací `@Override` a také `@ErrorMethod`, která obsahuje krátký popis, jenž chybovou funkcionalitu vystihuje.

6. Metodu je rovněž nutné doplnit logováním, aby bylo z logu patrné, že byla provedena zanesená chyba.

7. V metodě kontroleru, který zajišťuje změny v uvažovaném pohledu, pak bude tato metoda zavolána a na základě jejího výsledku i provedena změna. Taková změna může být různá a může tímto nabýt změn pouze kontroler a nebo i `.jsp` soubor. To záleží na konkrétním případu.

Příklad označení anotací `@ErrorMethod` s patřičně vyplněným parametrem `errorMessage` je v ukázce kódu 9.1. Tato anotace je důležitá při vytváření nové verze přímou implementací rozhraní definujícího entitu, aby bylo možné odlišit chybové metody od korektních. Její význam je i ve formě přehledné dokumentace injektované chyby.

```
@ErrorMethod (errorMessage = "This error method ...")
public boolean afterRemoveShowOverview() {
    ...
    log.error(propertyLoader.getProperty(
        "log.E04StudentService.afterRemoveShowOverview"
    ));
    return true;
}
```

Ukázka kódu 9.1: Příklad označení chybové metody

Ukázka chybové metody implikující změnu:

```
/**
 * DELIBERATE ERROR
 *
 * This error method changes view
 * that is shown after removal of subject
 * @return Indication for showing overview after removal of subject
 */
@Override
@ErrorMethod(errorMessage = "This error method shows overview view after
    removal of subject.")
public boolean afterRemoveShowOverview() {
    log.error(propertyLoader.getProperty("log.E04StudentService.
        afterRemoveShowOverview"));
    return true;
}
```

Ukázka kódu 9.2: Příklad chybové metody

Do TbUIS bylo tedy doplněno 20 nových chybových verzí. Deset z nich vychází z rozhraní `StudentService` a deset z `TeacherService`. Chyby do DAO a DateUtility již nemělo smysl injektovat, protože byly dostatečně pokryty chybovými verzemi z původní práce [2].

Tento výčet obsahuje všechny nové chybové verze:

- **E03StudentService** (metoda `setTitle`) – je zobrazený jiný textový popisek – místo *Student's View* je *Stud View*
- **E04StudentService** (metoda `afterRemoveShowOverview`) – pokud si student odepíše předmět, přesune se aplikace na pohled *Overview*
- **E05StudentService** (metoda `changeOverviewToOtherExam`) – odkaz *Overview* v menu studenta vede na *Other Exam Dates*
- **E06StudentService** (metoda `hideTeacherColumn`) – na obrazovce *Other Subjects* chybí sloupec *Teachers*
- **E07StudentService** (metoda `changeParticipantsButtonColor`) – na obrazovce *My Exam Dates* je tlačítko *Participants* podbarveno červeně
- **E08StudentService** (metoda `getStudentExaminationDatesList`) – tabulka v modálním okně *Participants* na obrazovce *My Exam Dates* obsahuje jednoho neznámého studenta navíc
- **E09StudentService** (metoda `duplicateLastParticipant`) – tabulka v modálním okně *Participants* na obrazovce *Other Exam Dates* obsahuje dvakrát posledního uvedeného studenta
- **E10StudentService** (metoda `hideUnenrollButton`) – v tabulce na obrazovce *My Subjects* chybí v některé náhodně zvolené řádce jedno tlačítko *Unenroll subject*
- **E11StudentService** (metoda `setExaminationDate`) – na obrazovce *Other Exam Dates* tlačítko *Register* nevykoná žádnou změnu
- **E12StudentService** (metoda `changeNumberOfParticipants`) – na obrazovce *My Exam Dates* ukazují všechna tlačítka *Participants* chybný počet studentů

- **E02TeacherService** (metoda `removeExaminationTerm`) – učitel při pokusu o zrušení zkouškového termínu nesmaže záznam v databázi a ani na obrazovce
- **E03TeacherService** (metoda `removeExaminationTerm`) – učitel při pokusu o zrušení zkouškového termínu smaže záznam na obrazovce, nikoliv v databázi
- **E04TeacherService** (metoda `removeExaminationTerm`) – učitel při pokusu o zrušení zkouškového termínu smaže záznam v databázi, nikoliv na obrazovce
- **E05TeacherService** (metoda `swapNameAndTeacher`) – na obrazovce *My Subjects* jsou prohozené sloupce pro název předmětu – *Name* a jména učitelů – *Teacher(s)*.
- **E06TeacherService** (metoda `setMySubject`) – učiteli se při pokusu o zapsání jiného předmětu neprovedou změny v databázi a ani na obrazovce.
- **E07TeacherService** (metoda `createNewExaminationTerm`) – učiteli je na obrazovce *New Exam Dates* umožněno vytvořit zkouškový termín bez uvedení dnu a času.
- **E08TeacherService** (metoda `getAllExaminationTermsByTeacherAndSubject`) – učiteli se na obrazovce *Evaluation Table* nezobrazí již ohodnocení studenti.
- **E09TeacherService** (metoda `createNewExaminationTerm`) – učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Tento termín je pro učitele viditelný, ale pro studenty nikoliv.
- **E10TeacherService** (metoda `createNewExaminationTerm`) – učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Tento termín se jeví, že se vytvořil, ale do databáze se neuloží a je tak pro učitele i studenty neviditelný.
- **E11TeacherService** (metoda `createNewExaminationTerm`) – učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Záporné číslo studentů je ale automaticky převedeno na kladné. Takový je i maximální počet studentů termínu.

Pro příklad je zde uvedený scénář odhalení jedné chybové verze. Všechny verze jsou popsané i se scénářem odhalení v příloze A.

Ukázka scénáře odhalení

Pokud si student odepíše předmět, přesune se aplikace na pohled *Overview*.

Scénář odhalení – E04StudentService

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Subjects* (položka uživatelského menu).
3. Vybere si z nabídky libovolný předmět a klikne na jeho tlačítko *Unenroll subject* (křížek).
4. Svůj výběr potvrdí v modálním okénku kliknutím na tlačítko *Unenroll subject*.
5. Uživatel se nyní nachází na pohledu *Overview*, i když měl zůstat na pohledu *My Subjects*.

9.1.3 Rozšíření katalogu ErrorSeederu

Spolu s přidáním nových verzí bean musel být rozšířen i katalog ErrorSeederu, aby tyto poruchové verze bylo možné začlenit do klonů. Obecný popis možnosti rozšíření je popsán v části 4.2. V nové verzi tak byly do příslušných elementů `bean` přidány nové elementy `version`, reprezentující tyto nové verze, se všemi potřebnými atributy a elementy, včetně scénářů pro opětovné simulace daných chyb.

9.1.4 Změna způsobu injekce složitější chyby

Umožnění injektování složitější chyby (tedy ovlivnění chování na vrstvě kontrolerů) je možné, ovšem v současné době není nutné. Taková případná změna injekce chyb by byla natolik rozsáhlá, že by překročila možnosti této diplomové práce. Ovšem není vyloučeno, že v budoucnu při zvýšené potřebě injekce tohoto typu chyb bude vhodné k této radikální změně přistoupit.

9.2 Pomocné aplikace pro testy

Pro lepší testování testbedu UIS byly vytvořeny dvě menší aplikace, které mají za úkol celý proces pro uživatele zjednodušit a zrychlit. Potřeba jejich vytvoření vzešla z častého experimentálního používání seleniových testů za účelem úspory práce.

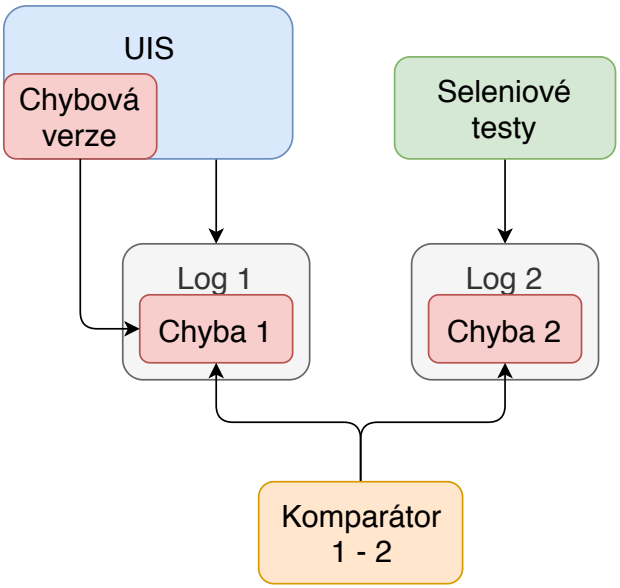
9.2.1 Program pro porovnávání logů

Výsledkem implementace logování pro UIS je po spuštění generovaný soubor s logy, který popisuje všechny prováděné akce na serveru, včetně vyhozených výjimek. V tomto souboru je taktéž zalogováno použití chybové verze, pokud je samozřejmě během činnosti UIS taková funkcionalita spuštěna.

Na druhé straně, výsledkem proběhnuvších scénářů seleniových testů nad běžící instancí UIS je rovněž soubor s logy. Pokud některý ze spuštěných testů neprojde, je takové selhání náležitě zalogováno. Během testování ovšem vznikla potřeba na základě logů získaných ze seleniových testů co nejjednodušeji zjistit informaci o případné zanesené chybě a zpětně potvrdit, že taková chyba opravdu nastala. Kvůli tomu bylo zapotřebí přidat pojítko, kterým by se dokázaly bezpečně spárovat chybové výpisy obou logů a tím by bylo ověřeno, že chybová verze je odhalena testy. Tímto pojítkem se staly ID jednotlivých elementů objevujících se na stránkách UIS. Chybové výpisy logů ze seleniových testů úvadějí právě ID elementu, kterého se chyba týká. Bylo tedy nutné opatřit pro každý výpis chybové verze hodnotami, představující ID elementu, nebo elementů, kterým svým chybovým chování ovlivní.

Pro tyto účely tak byla vytvořena pomocná konzolová aplikace zvaná LogComparator, jenž oba logy porovnává.

Vše je přehledně zobrazeno na obrázku 9.1.



Obrázek 9.1: Kontext pomocné aplikace určené k porovnání logů

Logy z aplikace UIS

Obsah souboru s logy z aplikace UIS obsahuje na každém řádku výpisy, jenž mají stále stejnou strukturu. Jejich podoba vypadá takto: `<datum a čas zalogování> -- <severita> <název souboru, z něhož je logováno>:<číslo řádku, z něhož je logováno> - <zpráva logu>`. Odchylku od této normy tvoří případy, kdy dojde k výjimce, poté je do řádek souboru vypsána zpráva této výjimky. Tyto stavy většinou nastávají společně s logem o severitě `ERROR`.

Speciálním typem chybového výpisu zde představují logy se severitou `ERROR`, jejichž zpráva začíná značením `INJECTED_ERROR:`. Tento typ chyby pochází z chybové verze některé z entit a je takto speciálně označen proto, že se jedná o záměrnou chybu. Po tomto značení následuje klíč chyby, který je složen z názvu chybové verze, příslušné metody a popisu chyby. Příklad takového logu je v ukázce 9.3.

```
2019-03-18 23:30:04.121 -- ERROR E01StudentService:57 - INJECTED_ERROR:
    E01StudentService_getStudiedSubjectsList_returns_always_null
_instead_of_list_of_subjects/CRITICAL
```

Ukázka kódu 9.3: Část logu z aplikace UIS

Logy ze seleniových testů

Struktura řádků druhého souboru, tedy souboru se seleniovými logy, je značně odlišná. Jednotlivé záznamy jsou vždy na několik řádek, minimálně však dvě. Jejich struktura je následující: `<datum a čas zalogování> -- <severita>: <pomocné pořadové číslo události použité pro ladění>> <název testu včetně balíků> > <název testové metody> UIS_ERR: <zpráva logu>`. Příklad takového logu je v ukázce 9.4.

```
2019-03-18 23:30:08.003 -- ERROR: 0441> uis.tests.active.onechange.
    stu_othersubjects.SeveralSubjects_Test > test_2_Stu_MySubjects
    UIS_ERR: stu.mySubjects.enrolledSubjects.table: NOT contains '
        Introduction to Algorithms' expected:<true> but was:<false>
```

Ukázka kódu 9.4: Část logu ze seleniových testů

Implemetace

Z logů aplikace UIS jsou vybírané jen ty typy logovacích zpráv, které odpovídají injektovaným chybám, tedy ty logy, jenž obsahují značení `INJECTED_ERROR:`. Aby bylo možné spárovat ty logy, jenž odpovídají stejné chybě, bylo potřeba manuálně vytvořit soubor, jenž bude mapovat klíč chyby

z aplikace UIS na zprávu ze seleniových testů. K tomuto účelu byl tedy vytvořen soubor *keys.txt*, který toto mapování obsahuje. Každá jeho řádka specifikuje jedno mapování, a to v podobě `<klíč>: <mapované hodnoty oddělené čárkou>`. Následně jsou logy náležící stejné chybě spárované podle nejbližšího výskytu v čase. Takto utvořené dvojice jsou vypsány do konzole.

Aby došlo k nalezení a spárování výše uvedených logů, je pro představu uvedena řádka s mapováním v ukázce 9.5.

```
E01StudentService_getStudiedSubjectsList_returns
_always_null_instead_of_list_of_subjects: stu.mySubjects.enrolledSubjects.
    table, stu.otherExamDates.table, testStu_MySubjects_DbContents,
    testStu_OthersExamDates_DbContents
```

Ukázka kódu 9.5: Příklad mapování chyby z aplikace UIS na zprávu z testů

Současné řešení sice není ideální, ale vzhledem k tomu, že aplikace je pouze pomocným nástrojem, je plně dostačující.

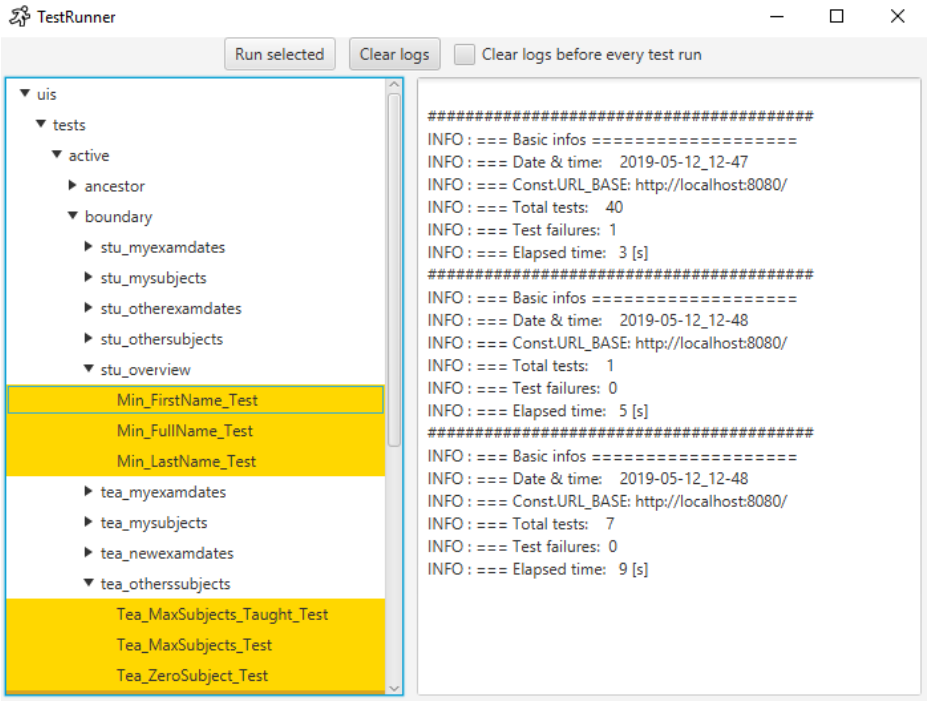
9.2.2 Desktopová aplikace pro spouštění jednotlivých testů

Seleniové testy nad aplikací byly spouštěny buď z vývojového prostředí nebo z příkazové řádky pomocí `bat` souboru s příkazy. Tomu bylo potřeba specifikovat cestu ke spouštěnému testu a který obsahoval v příkazu „na tvrdo“ uvedenou cestu k adresáři konfiguračním souborem.

Aby bylo spouštění testů pohodlnější a přehlednější, byla pro tento účel vytvořena malá desktopová aplikace s grafickým uživatelským rozhraním (GUI) zvaná TestRunner. Tato aplikace umožňuje jednoduše spouštět vybrané testy. Samostatné testy a testovací sady (*test suite*) jsou vizuálně odděleny odlišnou barvou. Adresář s konfiguračním souborem je nyní brán jako aktuální pracovní adresář. Výstup testů je vypisován jak do vlastní aplikace, tak do případné konzole. Logy lze z konzole i ze souboru promazávat kdykoliv pomocí tlačítka *Clear logs* nebo automaticky vždy před spuštěním dalšího testu zaškrtnutím volby *Clear logs before test run*. Ukázka použití této aplikace je na obrázku 9.2.

Aplikace je zároveň napsána natolik obecně, že výběr seleniových testů je určen pouze přítomností jejich zkompilovaných `class` souborů v aplikačním adresáři *uis*, případné vnitřní adresáře pak tvoří stromovou strukturu testů. Výjimku tvoří adresář *support*, který je jediný při načítání struktury vyřazený, protože je určen pouze pro pomocné soubory pro spouštění testů. Pokud tedy uživatel potřebuje spouštět nový test, stačí jeho `class` soubor přidat do příslušné složky. Je však potřeba dodržet jmennou konvenci, aby tento

soubor končil v případě samostatného testu řetězcem *_Test* a v případě testovací sady řetězcem *_Tests*, a zároveň se nenacházel v adresáři *support*. Obdobně funguje i odebrání testu z aplikace, kdy stačí `class` soubor daného testu pouze smazat. Implementačně je aplikace TestRunner řešena tak, že jsou v základu spouštěny příkazy na příkazové řádce, ale uživatel interaguje čistě s uživatelským rozhraním. Díky GUI je práce značně zjednodušená. Ke své činnosti využívá několika knihoven, a to JUnit ve verzi 4.12, Hamcrest[1] ve verzi 1.3, Log4j ve verzi 2.11.1, Selenium Standalone Server ve verzi 3.141.59 a Commons IO[2] ve verzi 2.6.



Obrázek 9.2: Ukázka použití TestRunneru

Při vytváření této aplikace byl kladen důraz na přehlednost a intuitivnost. Je rozdělena na tři hlavní části. Levá polovina obsahuje adresářový strom `.class` souborů, který je možné jednoduše procházet a kliknutím si vybrat test nebo testovací sadu. Samostatné testy jsou označeny žlutou barvou, testovací sady světle hnědou. V pravé polovině se nachází konzole, kam se po vykonání testů vypíše výsledek. Poslední třetí část obsahuje základní ovládací prvky a je umístěna v horní části aplikace.

Po spuštění a doběhnutí vybraného souboru s testem se do konzole vypíše informace o aktuálním čase, počtu všech spuštěných, úspěšných a neúspěšných

[1]Hamcrest – jedna z částí JUnit4
[2]Commons IO – knihovna pro psaní I/O v Javě

testů, použité prostředí a doba běhu všech spuštěných testů.

9.2.3 Git

Jako distribuovaný systém správy verzí byl použit Git, konkrétně GitLab na školním serveru. Jsou zde verzovány všechny důležité části projektu: UIS, ErrorSeeder, funkcionální testy atp., konkrétně se jedná o tyto projekty:

- TbUIS-UIS
- TbUIS-tests
- TbUIS-ErrorSeeder
- TbUIS-DP

10 Poruchové klony

To, jakým způsobem je klon sestaven a jaký je jeho vztah k verzím, bylo uvedeno v části 3 a zobrazeno na obrázku 3.3. Klon může obsahovat ve stávající době celkem pět injektovaných chybových verzí, do každé beany jednu. Jak již bylo uvedeno, 20 nových chybových verzí bylo do UIS doplněno. Ty spolu s původními 7 verzemi představují celkem 27 chybových verzí. Po všech úpravách, které byly v aplikaci UIS provedeny, vznikl pro každou chybovou verzi chybový klon. Každý klon tak obsahuje právě jednu zanesenou chybovost a to je výhodné pro jejich prvotní otestování. Je samozřejmě možné a v budoucnu se to očekává, že každý chybový klon bude sestaven z více chybových verzí.

10.1 Testování klonů

Postupně bylo testováno celkem 23 klonů UIS, z toho jeden plně korektní a zbylé do určité míry chybové. Ty chybové obsahují vždy pouze jednu chybovou verzi. Každá chybová verze, která byla vytvořena, tak má vlastní klon, jenž obsahuje pouze tuto chybovou verzi. Nově vytvořené verze si vyžádaly i přidání nových jednotkových testů ověřující jejich funkčnost a chybovost. To bylo provedeno pro všechny verze, ovšem u jedné konkrétní verze – `E07StudentService` to nebylo možné. V současnosti totiž nelze používaným způsobem odhalit takovou chybovost.

10.1.1 E07StudentService

Na obrazovce *My Exam Dates* je tlačítko *Participants* podbarveno červeně.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
3. Tlačítko pro zobrazení všech zapsaných studentů (*Participants*) není modré ale červené.

10.2 Výsledky

Přehled výsledků včetně statistik výsledků testů je v tabulce 10.1. První sloupec vždy obsahuje název klonu se jmennou konvencí, jenž začíná vždy *UIS*. Pokračuje severitou a řetězcem charakterizující daný klon, typicky například zratka názvu chybové verze entity, jenž je v klonu užita. Severita udává počet použitých chybových verzí a jejich závažnosti. Písmena *C*, *H*, *M* a *L* značí severity *critical* (kritická), *high* (vysoká), *medium* (střední) a *low* (nízká) a čísla následující udávají počet jednotlivých chybových verzí těchto severit. V dalších sloupcích následují počty všech spuštěných testů (*celkem*), úspěšně doběhlých testů (*passed*), selhaných testů (*failed*) a ignorovaných (*ignored*).

Jednotlivé moduly jsou v tabulce seřazeny podle severity, od nejnižší (korektní modul) po nejvyšší.

Poznámka: Drobné odchylky v celkovém počtu proběhlých testů jsou způsobeny parametrizovanými testy.

10.3 Zhodnocení výsledků

Z výsledků testování různých klonů lze vypozorovat, že se zvyšující se severitou dochází ke snížení počtu úspěšných testů, což je zcela očekávaný výsledek. Platí zde tedy nepřímá úměra. Opačný trend vykazují počty neprovedených testů, tj. testů, které z důvodu např. pádu aplikace již nebylo možné provést. Poznámka – pokud jsou testy spuštěny z vývojového prostředí IntelliJ IDEA, jsou tyto neproběhlé testy označeny jako ignorované.

U neúspěšných testů je rozložení četností u jednotlivých klonů značně nepravidelné, ačkoliv klony s větší severitou jsou na tom v tomto ohledu poněkud hůře. Dalším zajímavým poznatkem je, že chybová verze již implementované funkčnosti (tedy tzv. jednoduché chyby) vykazuje vyšší známku chybovosti, než u chyb, které ovlivňovaly chování kontrolerů a souborů `.jsp`.

Téměř všechny klony proběhly bez programové chyby, až na tři klony. Jedním z nich je UIS-C0.H0.M0.L1_S_S_04, kdy žádný test sice neselhal (*failed*), ale pět z nich skončilo programovou chybou (*error*). To je způsobeno tím, že zanesená chyba mění přepokládanou URL adresu a stránka tudíž postrádá elementy pro testy potřebné. Dalšími dvěma klony s podobnou chybovostí jsou UIS-C0.H1.M0.L0_T_S_06 a UIS-C0.H1.M0.L0_T_S_08. V tabulce s výsledky však byly i tyto testy zařazeny do sloupečku *failed*, jelikož se i tak jedná o odhalenou chybu.

Jediný klon (UIS-C0.H0.M0.L1_S_S_07), jehož chybovost nebyla testy odhalena, zanáší záměnu podbarvení tlačítka z modré barvy na červenou.

Nicméně tento klon vzniknul čistě z experimentálních důvodů a pro odhalení takového typu chybovosti nejsou funkcionální testy navrhnuty a implementovány.

klon	**celkem**	**passed**	**failed**	**neprovedeny**
UIS-C0.H0.M0.L0_ALL_OK	1069	1069	0	0
UIS-C0.H0.M0.L1_S_S_02	1069	1058	11	0
UIS-C0.H0.M0.L1_S_S_03	1069	1068	1	0
UIS-C0.H0.M0.L1_S_S_04	1074	1024	0 + 5	45
UIS-C0.H0.M0.L1_S_S_06	1069	1056	13	0
UIS-C0.H0.M0.L1_S_S_07	1069	1069	0	0
UIS-C0.H0.M0.L1_S_S_12	1069	1059	10	0
UIS-C0.H0.M0.L1_T_S_05	1078	964	59	55
UIS-C0.H0.M1.L0_S_S_09	1069	1068	1	0
UIS-C0.H0.M1.L0_T_S_07	1075	1064	7	4
UIS-C0.H0.M1.L0_T_S_09	1069	1068	1	0
UIS-C0.H0.M1.L0_T_S_10	1069	1068	1	0
UIS-C0.H0.M1.L0_T_S_11	1069	1068	1	0
UIS-C0.H0.M1.L0_U_D_01	1069	1062	7	0
UIS-C0.H1.M0.L0_D_U_01	1072	981	28	63
UIS-C0.H1.M0.L0_D_U_02	1072	980	29	63
UIS-C0.H1.M0.L0_S_S_05	1069	1068	1	0
UIS-C0.H1.M0.L0_S_S_08	1069	1049	20	0
UIS-C0.H1.M0.L0_S_S_10	1072	1033	4	35
UIS-C0.H0.M1.L0_T_S_01	1099	617	196	286
UIS-C0.H1.M0.L0_T_S_02	1069	1057	12	0
UIS-C0.H1.M0.L0_T_S_03	1069	1068	1	0
UIS-C0.H1.M0.L0_T_S_04	1069	1054	15	0
UIS-C0.H1.M0.L0_T_S_06	1072	992	29 + 1	50
UIS-C0.H1.M0.L0_T_S_08	1070	1028	20 + 5	17
UIS-C1.H0.M0.L0_G_T_D_01	1079	859	13	207
UIS-C1.H0.M0.L0_S_S_01	1086	822	86	178
UIS-C1.H0.M0.L0_S_S_11	1072	983	23	66
UIS-C2.H2.M1.L0_M_CR	1117	433	185	499

Tabulka 10.1: Statistiky testovaných klonů

11 Závěr

Pokud by bylo možné charakterizovat tuto práci jednou větou, bylo by to: „Pokračování ve vývoji již existujícího projektu TbUIS“.

Tato stručná charakteristika vcelku vystihuje charakter prací, které byly v rámci této diplomové práce vykonány. Nejednalo se o jednu konkrétní aktivitu, ale o sadu vzájemně souvisejících činností, které dohromady významně posunuly projekt TbUIS k jeho celkové použitelnosti.

Jednou z prvních aktivit bylo doplnění původní bezporuchové aplikace UIS o vylepšení, která byla zjištěna až v provozu prvotní aplikace, což je podrobně popsáno v části 8.2. Současně s tímto doplněním byly rozšířeny i příslušné funkcionální testy.

Ačkoliv v zadání diplomové práce byl kladen velký důraz na webové stránky projektu, ve skutečnosti v této práci na ně tak velký důraz kladen nebyl. Bylo to z toho důvodu, že webové stránky budou připraveny jako výsledek jiné kvalifikační práce. Ovšem v této práci vytvořené webové stránky ve své jednodušší podobě sloužily pro informaci o projektu TbUIS již od podzimu 2018 a tím splnily svůj účel.

Při pracích na projektu TbUIS vyvstala potřeba přípravy několika „pomocných“ aplikací. Byly určeny pro nezávislé spouštění testů (podrobně viz část 9.2.2) a porovnávání logů (část 9.2.1). Tyto aplikace nejsou velké svým rozsahem, ale výrazně zvyšují komfort při práci s TbUIS.

Klíčovou aktivitou v této práci byla příprava rozsáhlé sady poruchových verzí. Bylo připraveno celkem 20 poruchových verzí a z nich pak celkem 28 přímo spustitelných poruchových klonů UIS, které jsou umístěny na DVD. Všechny poruchové klony byly důkladně otestovány funkcionálními testy. Podrobnosti jsou uvedeny v částech 9 a 10.

V současné době je na http://oks.kiv.zcu.cz:10008/ nasazena verze 1.5.0 v Java 8 a na DVD je verze 1.5.2 v Java 11.

Všechny body zadání byly splněny.

Přehled zkratek

AOP	Aspect-Oriented Programming
CSS	Cascading Style Sheets
CVS	Concurrent Version System
DAO	Data Access Object
DI	Dependency Injection
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IoC	Inversion of Control
I/O	Input/Output
Java EE	Java Enterprise Edition
JSON	JavaScript Object Notation
JSP	JavaServer Pages
Less	Leaner Style Sheets
MEAN	MongoDB, Express.js, Angular, Node.js
MVC	Model-View-Controller
MVP	Model-View-Presenter
ORM	Object-Relational Mapping
PNG	Portable Network Graphics
POJO	Plain Old Java Object
POM	Project Object Model
Sass	Syntactically awesome style sheets
SUT	System Under Test

SVN Apache Subversion

TbUIS Testbed UIS

UIS University Information System

URL Uniform Resource Locator

WAR Web Application Resource nebo Web Application Archive

XML Extensible Markup Language

YAML YAML Ain't Markup Language

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu

Použité zdroje

Literatura

[1] Ammann, J. P. a. O. *Introduction to Software Testing*. Cambridge University Press, 1 edition, 2008. ISBN 978-1107172012.

[2] Matyáš, J. *Aplikace s možností injekce chyb pro ověřování kvality testu.* Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2018.

[3] Rod Johnson, A. A. T. R. C. S. J. H. *Professional Java Development with the Spring Framework*. Wrox, 2005. ISBN 978-0-7645-7483-2.

[4] Duane K. Fields, M. A. K. *Web Development with Java Server Pages*. Manning Publications Co., 2001. ISBN 9781930110120.

[5] Hanna, P. *Jsp 2.0: The Complete Reference*. McGraw-Hill Education (India) Pvt Limited, 2003. Dostupné z: `https://books.google.cz/books?id=u4owcUMALbMC`. ISBN 9780070531413.

[6] Timothy M. O'Brien, B. F. J. C. *Professional Java Development with the Spring Framework*. Sonatype, Inc.(2009-2011), 2011. ISBN 978-0-9842433-3-4.

[7] McIntosh, S. – Adams, B. – Hassan, A. E. *The evolution of Java build systems*. 2012. doi: 10.1007/s10664-011-9169-5.

[8] Poubová, J. *Generování automatizovaných funkcionálních testů.* Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2019.

Elektronické zdroje

[9] Nadig, S. *How to Effectively Prepare "Test Bed" and Minimize the Test Environment Defects (Part 1)* [online]. 2019. [cit. 2019/13/4]. Dostupné z: `https://www.softwaretestinghelp.com/test-bed-test-environment-management-best-practices/`.

[10] *Test Environment for Software Testing* [online]. 2019. [cit. 2019/15/4]. Dostupné z: `https://www.guru99.com/test-environment-software-testing.html`.

[11] Dave, H. Y. H. *Arena* [online]. 1996. [cit. 2019/13/4]. Dostupné z: `https://www.w3.org/Arena/`.

[12] Quint, V. *Amaya Overview* [online]. 2012. [cit. 2019/14/4]. Dostupné z: https://www.w3.org/Amaya/Overview.html.

[13] *JSFiddle Docs and Help* [online]. 2018. [cit. 2019/13/4]. Dostupné z: https://docs.jsfiddle.net/.

[14] *Plunker* [online]. 2019. [cit. 2019/13/4]. Dostupné z: http://plnkr.co/.

[15] *JBoss AOP - User Guide* [online]. 2019. [cit. 2019/7/4]. Dostupné z: https://docs.jboss.org/aop/1.1/aspect-framework/userguide/en/html_single/index.html.

[16] *Spring - Core Technologies* [online]. 2018. [cit. 2019/7/4]. Dostupné z: https://docs.spring.io/spring/docs/5.0.9.RELEASE/spring-framework-reference/core.html#aop.

[17] Crusoveanu, L. *Inversion of Control and Dependency Injection with Spring* [online]. 2019. [cit. 2019/2/4]. Dostupné z: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring.

[18] *Test Environment for Software Testing* [online]. 2019. [cit. 2019/28/4]. Dostupné z: https://www.guru99.com/jsp-life-cycle.html.

[19] Zyl, J. v. – See, F. A. V. – Porter, B. *Maven – Introduction to the POM* [online]. 2019. [cit. 2019/18/4]. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html.

[20] Zyl, J. v. – See, F. A. V. – Porter, B. *POM Relationships* [online]. 2019. [cit. 2019/18/4]. Dostupné z: http://maven.apache.org/pom.html#POM_Relationships.

[21] Zyl, J. v. – See, F. A. V. – Porter, B. *Available Plugins* [online]. 2019. [cit. 2019/18/4]. Dostupné z: https://maven.apache.org/plugins/index.html.

[22] Topinka, T. *Statistic* [online]. 2019. [cit. 2019/29/4]. Dostupné z: https://plugins.jetbrains.com/plugin/4509-statistic.

[23] *Php The Lightning-Smart PHP IDE* [online]. 2019. [cit. 2019/14/4]. Dostupné z: https://www.jetbrains.com/phpstorm/.

[24] *PhpStorm Features — Development Environment* [online]. 2019. [cit. 2019/14/4]. Dostupné z: https://www.jetbrains.com/phpstorm/features/development_environment.html.

[25] *Configuration* [online]. 2019. [cit. 2019/14/4]. Dostupné z: https://logging.apache.org/log4j/2.x/manual/configuration.html.

[26] Ceki Gülcü, A. K. *Documentation: Class PropertyConfigurator* [online]. 2019. [cit. 2019/14/4]. Dostupné z: `https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PropertyConfigurator.html`.

[27] *IntelliJ IDEA* [online]. 2019. [cit. 2019/14/4]. Dostupné z: `https://www.jetbrains.com/idea/`.

[28] *Making Development an Enjoyable Experience* [online]. 2019. [cit. 2019/14/4]. Dostupné z: `https://www.jetbrains.com/idea/features/`.

A Seznam nových chybových verzí

Každý scénář odhalení počítá s tím, že se nepřihlášený uživatel nachází na stránce *Login* (položka horního menu).

A.0.1 E03StudentService

Je zobrazený jiný textový popisek – místo *Student's View* je *Stud View*.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Nadpis pohledu není *Student's view* ale *Stu view*.

A.0.2 E04StudentService

Pokud si student odepíše předmět, přesune se aplikace na pohled *Overview*.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Subjects* (položka uživatelského menu).
3. Vybere si z nabídky libovolný předmět a klikne na jeho tlačítko *Unenroll subject* (křížek).
4. Svůj výběr potvrdí v modálním okénku kliknutím na tlačítko *Unenroll subject*.
5. Uživatel se nyní nachází na pohledu *Overview*, i když měl zůstat na pohledu *My Subjects*.

A.0.3 E05StudentService

Odkaz *Overview* v menu studenta vede na *Other Exam Dates*.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Měl by se nacházet na pohledu *Overview* (položka uživatelského menu).
3. Kliknutím na stejnou položku v menu (*Overview*) ovšem přejde na pohled *Other Exam Dates*.

A.0.4 E06StudentService

Na obrazovce *Other Subjects* chybí sloupec *Teachers*.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *Other Subjects* (položka uživatelského menu).
3. V tomto pohledu chybí sloupeček učitele (*Teacher(s)*).

A.0.5 E07StudentService

Na obrazovce *My Exam Dates* je tlačítko *Participants* podbarveno červeně.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
3. Tlačítko pro zobrazení všech zapsaných studentů (*Participants*) není modré ale červené.

A.0.6 E08StudentService

Tabulka v modálním okně *Participants* na obrazovce *My Exam Dates* obsahuje jednoho neznámého studenta navíc. Neznámý student není vůbec v přednastavených studentech (Tony Noname).

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
3. Vybere si z nabídky libovolný předmět a klikne na jeho příslušné tlačítko *Participants*.
4. Modální okno zobrazí seznam studentů i s neznámým studentem.

A.0.7 E09StudentService

Tabulka v modálním okně *Participants* na obrazovce *Other Exam Dates* obsahuje dvakrát posledního uvedeného studenta.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *blue* (heslo *pass*).
2. Přejde na pohled *Other Exam Dates* (položka uživatelského menu).
3. Vybere si z nabídky libovolný předmět a klikne na jeho příslušné tlačítko *Participants*.
4. Modální okno obsahuje dvakrát posledního uvedeného studenta.

A.0.8 E10StudentService

V tabulce na obrazovce *My Subjects* chybí v jedné náhodně zvolené řádce tlačítko *Unenroll subject*.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Subjects* (položka uživatelského menu).
3. Chybí možnost odebrání předmětu kvůli chybějícímu tlačítku u náhodného předmětu.

A.0.9 E11StudentService

Na obrazovce *Other Exam Dates* tlačítko *Register* nevykoná žádnou změnu.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *blue* (heslo *pass*).
2. Přejde na pohled *Other Exam Dates* (položka uživatelského menu).
3. Klikne na libovolné tlačítko *Register*.
4. Žádná změna se ovšem neprovede, i když bude uživatel informován o úspěchu operace.

A.0.10 E12StudentService

Na obrazovce *My Exam Dates* ukazují všechna tlačítka *Participants* chybný počet studentů.

Scénář odhalení

1. Uživatel se přihlásí za studenta s uživatelským jménem *cyan* (heslo *pass*).
2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
3. Tlačítka *Participants* ukazují chybný počet studentů zapsaných na zkouškových termínech. Jejich počet je vždy o jeden vyšší, než je ve skutečnosti.

A.0.11 E02TeacherService

Učitel při pokusu o zrušení zkouškového termínu nesmaže záznam v databázi a ani na obrazovce.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).

3. Klikne na tlačítko pro zrušení zkouškového termínu (červené tlačítko X).

4. Uživatel bude informován o úspěchu operace, i když se žádná změna, jak v databázi tak na obrazovce neprovede.

A.0.12 E03TeacherService

Učitel při pokusu o zrušení zkouškového termínu smaže záznam na obrazovce, nikoliv v databázi.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).

2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).

3. Klikne na tlačítko pro zrušení zkouškového termínu předmětu *Software Engineering* (červené tlačítko X).

4. Uživatel bude informován o úspěchu operace, avšak smazán je obsah na obrazovce, nikoliv v databázi.

5. Uživatel se odhlásí stitknutím *Logout*.

6. Uživatel se přihlásí za studenta s uživatelským jménem *red* (heslo *pass*).

7. Přejde na pohled *Other Exam Dates* (položka uživatelského menu).

8. Uživateli bude nabídnut zkouškový termín k termínu *Software Engineering*, i když by tento termín měl být smazán.

A.0.13 E04TeacherService

Učitel při pokusu o zrušení zkouškového termínu smaže záznam v databázi, nikoliv na obrazovce.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).

2. Přejde na pohled *My Exam Dates* (položka uživatelského menu).

3. Klikne na tlačítko pro zrušení zkouškového termínu (červené tlačítko X).

4. Uživatel bude informován o úspěchu operace, avšak smazán je obsah v databázi, nikoliv na obrazovce.

A.0.14 E05TeacherService

V tabulce na obrazovce *My Subjects* jsou prohozené sloupce pro název předmětu – *Name* a jména učitelů – *Teacher(s)*.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *My Subjects* (položka uživatelského menu).
3. V tabulce jsou prohozené sloupce pro název předmětu – *Name* a jména učitelů – *Teacher(s)*.

A.0.15 E06TeacherService

Učiteli se při pokusu o zapsání jiného předmětu neprovedou změny v databázi a ani na obrazovce.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *Others Subjects* (položka uživatelského menu).
3. Klikne na tlačítko pro přidání si předmětu *Computer System Engineering* (modré tlačítko *Participate*).
4. Uživatel bude informován o úspěchu operace a předmět z nabídky zmizí.
5. Přejde na pohled *My Subjects* (položka uživatelského menu).
6. V tabulce učených předmětů se nenachází právě zapsaný předmět *Computer System Engineering*.

A.0.16 E07TeacherService

Učiteli je na obrazovce *New Exam Dates* umožněno vytvořit zkouškový termín bez uvedení dnu a času.
Poznámka: Chybová verze způsobuje, že čas termínu zkoušky není validován.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *New Exam Dates* (položka uživatelského menu).
3. Zkontroluje, že je již předvybrán libovolný předmět a maximální počet zapsaných studentů. Pole pro určení času zkoušky ponechá nevyplněné.
4. Kliknutím na tlačítko pro uložení zkouškového termínu *Save new exam date* uloží termín. Následně bude informován o úspěchu operace.
5. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
6. V tabulce se bude nacházet zkouškový termín s nevyplněným časem zkoušky.

A.0.17 E08TeacherService

Učiteli se na obrazovce *Evaluation Table* nezobrazí již ohodnocení studenti.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *Evaluation Table* (položka uživatelského menu).
3. Z nabídky předmětů vybere předmět *Programming in Java* a stiskne tlačítko pro filtrování studentů (modré tlačítko *Filter*).
4. Z tabulky pak vybere studentovi se jménem *Ethan Cyan* známku B a to potvrdí příslušným modrým tlačítkem *Submit*.
5. Uživatel bude informován o úspěchu operace a student se v tabulce opravdu neobjeví.
6. Zaškrtne možnost pro zahrnutí studentů s již nastavenou známkou (checkbox *Includes students with already set grade.*) a opětovně stiskne tlačítko pro filtrování studentů.
7. V tabulce se stále nebude nacházet student *Ethan Cyan*, ačkoliv by měl.

A.0.18 E09TeacherService

Učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Tento termín je pro učitele viditelný, ale pro studenty nikoliv.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *New Exam Dates* (položka uživatelského menu).
3. Z nabídky předmětů vybere předmět *Programming in Java*.
4. Do pole *Max participants* určující maximální počet studentů napíše hodnotu -1.
5. Do pole *Date and time of examination* vyplní den a čas, který je v budoucnosti, minimálně posunutý o 24 hodin po předchozím termínu.
6. Uživatel bude informován o úspěchu operace i když vyplnil nesmyslný počet studentů.
7. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
8. V tabulce bude přítomný nový termín se stejnými parametry, které byly uvedeny.
9. Uživatel se odhlásí stitknutím *Logout*.
10. Uživatel se přihlásí za studenta s uživatelským jménem *blue* (heslo *pass*).
11. Přejde na pohled *Other Exam Dates* (položka uživatelského menu).
12. V tabulce nebude přítomný nový termín vypsaný učitelem strict pro předmět *Programming in Java*, i když byl učitel informován o úspěchu vytvoření.

A.0.19 E10TeacherService

Učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Tento termín se jeví, že se vytvořil, ale do databáze se neuloží a je tak pro učitele i studenty neviditelný.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *New Exam Dates* (položka uživatelského menu).
3. Z nabídky předmětů vybere předmět *Programming in Java*.
4. Do pole *Max participants* určující maximální počet studentů napíše hodnotu -1.
5. Do pole *Date and time of examination* vyplní den a čas, který je v budoucnosti, minimálně posunutý o 24 hodin po předchozím termínu.
6. Uživatel bude informován o úspěchu operace i když vyplnil nesmyslný počet studentů.
7. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
8. V tabulce se ovšem termín nalézat nebude.

A.0.20 E11TeacherService

Učiteli je umožněno vytvořit zkouškový termín, i když zadá záporný maximální počet studentů. Záporné číslo studentů je ale automaticky převedeno na kladné. Takový je i maximální počet studentů termínu.

Scénář odhalení

1. Uživatel se přihlásí za učitele s uživatelským jménem *strict* (heslo *pass*).
2. Přejde na pohled *New Exam Dates* (položka uživatelského menu).
3. Z nabídky předmětů vybere předmět *Programming in Java*.
4. Do pole *Max participants* určující maximální počet studentů napíše hodnotu -1.
5. Do pole *Date and time of examination* vyplní den a čas, který je v budoucnosti, minimálně posunutý o 24 hodin po předchozím termínu.
6. Uživatel bude informován o úspěchu operace i když vyplnil nesmyslný počet studentů.
7. Přejde na pohled *My Exam Dates* (položka uživatelského menu).
8. V tabulce se ovšem termín bude nalézat, ovšem s kladným číslem určující maximální počet studentů termínu.

B Obsah DVD

- **bin** – složka s přeloženými a spustitelnými aplikacemi
  - **catalog.xml** – soubor katalogu
  - **ErrorSeeder.jar** – spustitelná verze aplikace ErrorSeeder
  - **LogComparator.jar** – spustitelná verze aplikace LogComparator
  - **poruchove_klony** – složka se všemi `.war` soubory chybových klonů
  - **TestRunner** – složka obsahující spustitelnou verzi aplikace TestRunner
  - **UIS.war** – bezchybná nasaditelná verze aplikace UIS
  - **uis-logger-config.xml** – konfigurační soubor pro Log4j
  - **LogComparator-files** – složka obsahující soubory, používané aplikací LogComparator
- **ErrorSeeder** – složka projektu aplikace ErrorSeeder
  - **src** – složka se zdrojovými soubory aplikace ErrorSeeder
  - **pom.xml** – ErrorSeeder Maven Project Object Model
- **JavaDoc** – JavaDoc dokumentace aplikací
  - **ErrorSeeder** – složka s dokumentací aplikace ErrorSeeder
  - **LogComparator** – složka s dokumentací aplikace LogComparator
  - **TestRunner** – složka s dokumentací aplikace TestRunner
  - **UIS** – složka s dokumentací aplikace UIS
- **LogComparator** – složka projektu programu pro porovnávání logů
- **Poster** – složka se soubory posteru této práce
  - **Smaus_Jakub_2019.pdf** – pdf soubor posteru
  - **Smaus_Jakub_2019.pub** – Microsoft Publisher soubor posteru

- **Smaus_Jakub_Diplomova_prace.pdf** – pdf soubor této práce
- **TBUIS** – složka projektu statických webových stránek
- **TestRunner** – složka projektu desktopové aplikace pro spouštění jednotlivých testů
- **TeX** – složka zdrojových souborů diplomové práce
- **UIS** – složka projektu aplikace UIS
  - **src** – složka se zdrojovými soubory aplikace UIS
  - **pom.xml** – UIS Maven Project Object Model

C Screenshoty z webové aplikace

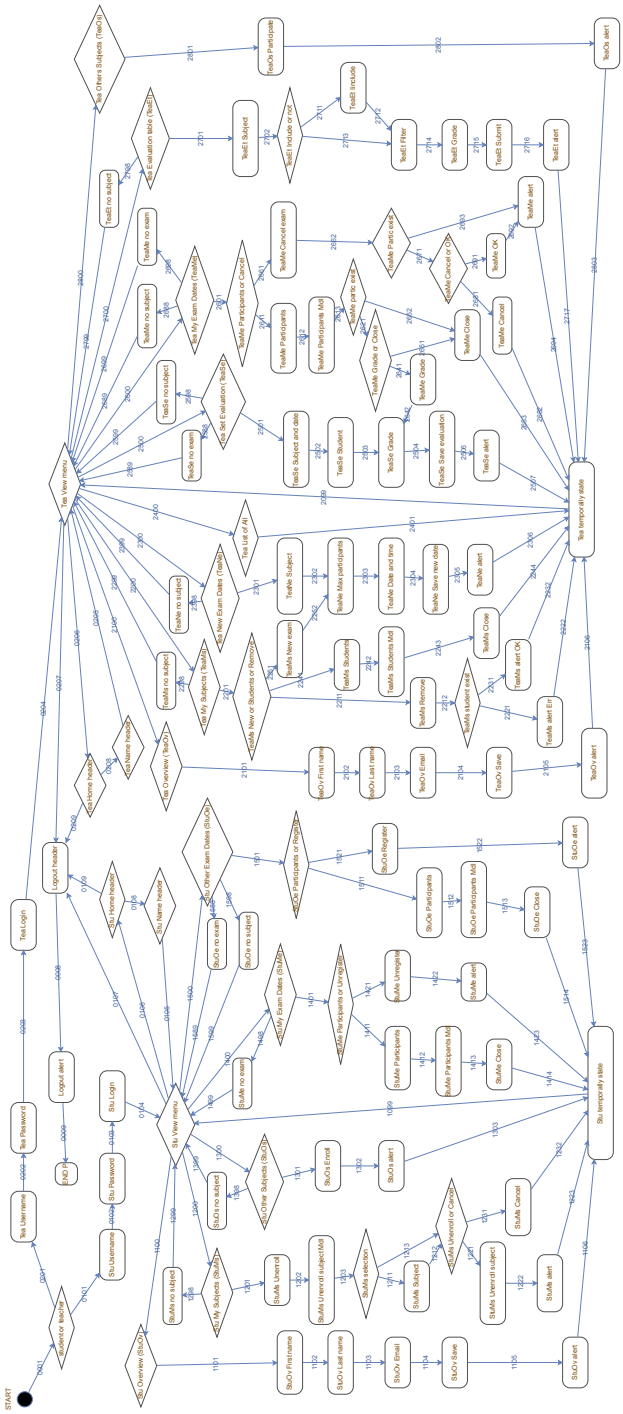
Obrázek C.1: Obrazovka *About*

Obrázek C.2: Obrazovka *Download*

Obrázek C.3: Obrazovka *UIS*

Obrázek C.4: Obrazovka *ErrorSeeder*

D Diagram stavů a přechodů



Obrázek D.1: Ilustrativní ukázka komplexnosti UIS