Masarykova Univerzita
Fakulta informatiky

Funkční testování aplikace pro výuku zeměpisu

Bakalářská práce

Erik Sýkora

Brno, Podzim 2015

Místo tohoto listu vložte kopie oficiálního podepsaného zadání práce a prohlášení autora školního díla.

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Erik Sýkora

Vedoucí práce: Mgr. Jan Papoušek

Poděkování

Rád bych zde poděkoval zejména svému vedoucímu Mgr. Janu Papouškovi za cenné rady a odbornou pomoc při tvorbě této práce. Dále chci také poděkovat své rodině za jejich podporu a pochopení během tvorby této práce.

Shrnutí

Cílem této práce je pomocí vhodných nástrojů vytvořit sadu funkčních testů pro webovou aplikaci Slepé Mapy. Součástí práce je rozbor použitých technologií a různých metod vhodných pro funkční testování webové aplikace. Část práce je také věnována analýze funkcionality testované aplikace. Důraz je také kladen na ukázku práce se zvolenými nástroji na konkrétních příkladech. Závěrem je popsána automatizace vytvořené testové sady.

Klíčová slova

funkční testování, webová aplikace, Java, Arquillian, automatizace testování, průběžná integrace

Obsah

1 Úvod

Vývoj softwaru je komplexní proces sestávající z mnoha kroků. Tento proces se liší dle použitého vývojového modelu, nicméně vždy by měl zahrnovat určitou formu testování. Náklady na opravu chyby totiž strmě rostou při jejím pozdním nalezení, v případě kritického softwaru může neošetřená chyba dokonce způsobit ztráty na životech.

Webové aplikace jsou dnes velice rozšířeným druhem softwaru. Testování webové aplikace zahrnuje spoustu oblastí, například funkční testování, testování kompatibility či regresní testování. Tyto a další způsoby testování se běžně používají pro testování softwaru obecně, nicméně některé z nich lze velice efektivně využít pro testování webových aplikací. Způsobům testování webové aplikace se bude zabývat druhá kapitola (2).

Cílem práce je provést funkční testování webové aplikace pro výuku zeměpisu. K tomuto účelu je možno vybírat ze široké škály nástrojů. V této práci bude vybrána testovací platforma Arquillian, která vylepšuje velice populární testovací platformu Selenium. Obě tyto platformy si zaslouží podrobnější popis, proto se třetí kapitola (3) bude zabývat představením těchto a dalších použitých nástrojů.

Jelikož testovaná aplikace postrádá specifikaci softwaru, bude důležitou částí práce analýza aplikace, čemuž bude věnována čtvrtá kapitola (4). Tato práce je zaměřena na funkční testování, proto zde bude hlavně popsáno chování a funkcionalita aplikace.

Na získaných znalostech o použitých technologiích bude stavět pátá kapitola (5), ve které bude představen postup při tvorbě testové sady. Vysvětlen bude kompletní postup použitý při testování webové aplikace Slepé mapy [12]. Popsány budou všechny kroky od přípravy vývojového prostředí, konfigurace použitých nástrojů až po tvorbu abstrakcí stránek a následně i testových tříd.

Hotové testy budou spouštěny automaticky při aktualizování aplikace. Jelikož by bylo nepraktické po každé aktualizaci testy spouštěn manuálně, bude součástí práce automatizace testové sady. Způsob realizace automatického testování bude popsán v šesté kapitole (6).

Závěrem (7) bude vyhodnocen přínos této práce a stručně budou shrnuty výsledky testování. Představeny budou také další možnosti rozvoje testovací sady.

2 Testování webové aplikace a použité přístupy

Testování softwaru je komplexní proces, jehož cílem je v programu či aplikaci nalézt softwarové chyby a ověřit, zda splňuje požadavky na něj kladené. Chybu lze obecně definovat jako neočekávané chování programu, lišící se od specifikace softwaru [4]. Za software obsahující chyby lze dle [2] považovat i obtížně srozumitelný, pomalý, či dle názoru testera program z pohledu koncového uživatele jakkoliv nesprávný. Při testování softwaru je nutno mít na paměti, že je prakticky nemožné v rozumném čase kompletně otestovat byť jednoduchý program. I menší programy mají příliš mnoho možných vstupů, výstupů a možných cest skrze software, proto je třeba při testování zúžit tuto množinu na zvládnutelnou podmnožinu. Toho je v případě této práce docíleno vytyčením klíčových či rizikových částí aplikace a aplikováním vhodných testovacích metod.

Výběr vhodné strategie je klíčový pro optimální výsledek testování, proto v této kapitole bude představen výběr některých běžně používaných přístupů pro testování webové aplikace. Většina zde uvedených postupů a metod byla využita v této práci při testování aplikace na procvičování zeměpisu. Je ovšem nutno zmínit, že seznam zde zmíněných principů testování webových aplikací není vyčerpávající. Je možné podrobit aplikaci důkladnějšímu testování pomocí dalších, zde nezmíněných metod, účelem této práce je ale provedení funkčního testování. Toto zúžení oblasti zájmu pouze na funkční testování umožňuje otestovat klíčové vlastnosti softwaru a podat tak informaci o funkčnosti – jednom z důležitých kritérií kvality softwaru. Nevýhodou je pak nemožnost otestovat některé ostatní kvality softwaru, jako například jeho spolehlivost, kompatibilitu či bezpečnost.

2.1 Typy testování

2.1.1 Funkční testování

Jelikož je stěžejním bodem této práce vypracování funkčních testů, je vhodné si uvést cíle tohoto typu testování. Cílem je otestování správného chování funkcí části aplikace či aplikace jako celku. Chování

softwaru může být popsáno v jeho specifikaci. Pokud není specifikace k dispozici, tak je testováno předpokládané chování aplikace. [3]. Během funkčního testování tester nahlíží na aplikaci očima uživatele a jeho úkolem je ověřit, zda aplikace funguje správně. Ve webové aplikaci mohou být testovány například přechody mezi jednotlivými stránkami aplikace, ošetření zápisu neplatných hodnot do formulářů nebo ošetření chybových stavů aplikace [2].

Při funkčním testování je nutné nejprve identifikovat funkce testovaného softwaru a poté pro nalezené funkce vytvořit testy. V této práci nelze k identifikaci funkcí využít specifikaci softwaru, proto se třetí kapitola (3) bude věnovat analýze webové aplikace a následného výběru funkcí vhodných k testování. Samotné tvorbě testů bude věnována pátá kapitola (5).

2.1.2 Jednotkové testování

Tento typ testování využívá malých, testovatelných částí aplikace. Touto takzvanou jednotkou mohou být například funkce, třídy, procedury či rozhraní. Cílem jednotkového testování je izolovat části programu a otestovat, zda splňují svou funkci a je možné je použít v celku, jehož jsou součástí. Výhodou tohoto přístupu je jednodušší objevení případných chyb v aplikaci. Jednotkové testování navíc bývá prováděno v ranných fázích vývoje softwaru, proto je také objevení chyb v programu výrazně levnější [5].

Některé programovací jazyky(Java, C#, Python, Ruby, PHP a další) mají přímou podporu jednotkového testování bez závislosti na externích knihovnách, nicméně běžně se používají aplikační rámce třetích stran, které tvorbu jednotkových testů usnadňují.

2.1.3 Statické a dynamické testování

Těmito pojmy lze rozlišit dva základní způsoby přístupu k testování softwaru. Statickým testováním je myšleno testování nespustitelné části programu, jde převážně o zkoumání dokumentace a specifikace softwaru. Tento přístup lze aplikovat ještě dříve, než je vytvořena spustitelná verze softwaru[2]. Hlavním cílem tohoto druhu testování je nalezení chyb v ranné fázi vývoje softwaru. Odstranění chyb v softwaru díky včasnému odhalení nejasnosti ve specifikaci může

být mnohem levnější, než opravování chyby v již hotovém kódu softwaru.

Dynamické testování naopak vyžaduje spustitelný kód, na základě jehož chování je software testován. Při testování zadáváme softwaru vstupní data, kontrolujeme reakci systému a ověřujeme korektnost výstupních dat. Dynamické testování lze také dále dělit na funkční a nefunkční testování. Funkční testování bylo v této kapitole popsáno, nefunkční testování se zabývá těmi aspekty softwaru, které přímo nesouvisí s jeho funkcionalitou. Pod nefunkčním testováním si lze představit například výkonnostní, zátěžové či bezpečnostní testování [10]. Předmětem zájmu této práce je dynamické funkční testování.

2.1.4 Regresní testování

Po opravě chyb v softwaru se může stát, že sice byla chyba opravena, ale jako vedlejší efekt se objevila chyba na jiném místě. Proces opakovaného spouštění testů při změně softwaru za účelem ověření existující funkcionality se nazývá regresní testování. Jeho cílem je zajištění, že změna kódu programu negativně neovlivnila funkčnost softwaru zavedením nových chyb [6]. Jelikož je testována funkčnost softwaru, tak lze regresní testování považovat za druh funkčního testování.

Regresní testování může probíhat manuálně, často ovšem bývá automatizováno. K testování je připravena testovací sada. Do této testovací sady je u větších projektů rozumné zahrnout pouze podmnožinu všech testů, jelikož spouštění velké testové sady vyžaduje větší množství času a zdrojů. V případě této práce jsou do testové sady zahrnuty všechny testy, což je díky jejich počtu stále ještě rozumné.

2.2 Použité metody a postupy

2.2.1 Testování černé, šedé a bílé skříňky

Testování softwaru lze dělit pomocí pojmů testování černé, bílé a šedé skříňky. Při postupu testování černé skříňky má tester k dispozici pouze popis funkčnosti aplikace, nemůže využít zdrojového kódu softwaru. Lze tedy nesprávnou funkčnost pouze odpozorovat dle od-

lišného chování od specifikace či předpokládaného chování. Jako příklad testování černé skříňky může být uvedeno funkční testování.

Pokud je k testování využit i zdrojový kód programu nebo znalost vnitřních principů, jedná se o testování bílé skříňky. Testování založené na tomto principu bývá velmi důsledné, ale nemusí odpovídat realistickým scénářům užití. Tester využívající tuto techniku také musí mít programátorské znalosti, aby byl schopen porozumět kódu programu. Nevýhodou také je nemožnost testovat, zda nějaká funkcionalita chybí, jelikož tento přístup se zabývá testováním již napsaného kódu.

Třetí možností je testování šedé skříňky, která kombinuje předchozí dva postupy. Nejprve jsou testy navrhovány z pohledu uživatele (testování černé skříňky), následně je ale k návrhu testů využit také postupy testování bílé skříňky. Tento přístup zajistí efektivitu aplikace a dostatečné pokrytí kódu testy [1]. Postup testování šedé skříňky lze díky dostupnosti zdrojového kódu každé webové stránky využít i při testování webových aplikací.

2.2.2 Validace a verifikace

Oba tyto pojmy lze přeložit jako ověřování či kontrola, nicméně význam těchto dvou výrazů se liší a pro účely testování je tento rozdíl velmi důležitý. Verifikace je proces, jehož cílem je potvrzení, že testovaný software vyhovuje své specifikaci. Účelem procesu validace je kontrola, zda testovaný software vyhovuje požadavkům uživatele. V případě, že byla specifikace softwaru vytvořena nesprávně a nevyhovuje původním požadavkům uživatele, může dojít k situaci, kdy software sice splňuje verifikaci, ale validaci nikoliv [2]. V této práci dochází k validaci aplikace, jelikož se při testování hledí na správnost aplikace z pohledu uživatele.

2.2.3 Automatizace testování

Často se stává, že napsané testy bude nutné spouštět vícekrát, například v případě regresního testování. Nástroje automatizace testovacího procesu jsou vhodným řešením pro tyto situace. Mezi jejich hlavní výhody patří například rychlejší provádění testů, než je možné manuálně. Dále je testování efektivnější, jelikož při manuálním testo-

vání není možné provádět jinou činnost. Nespornou výhodou je také správnost a přesnost, jelikož člověk provádějící manuální testování může provést při testování chybu. Testovací nástroj je vždy ve své práci konzistentní [2]. Jednou z elegantních možností automatizace testování je využití systému průběžné integrace. Konkrétní postup využití systému průběžné integrace k automatizaci testování bude dále představen v šesté kapitole (6).

2.2.4 Průběžná integrace

Průběžná integrace je definována jako metoda vývoje softwaru, při které je práce každého vývojáře integrována v častých a pravidelných intervalech. K použití této metody je zapotřebí, aby vývojáři vkládali zdrojový kód do centrálního úložiště zdrojových kódů. V tomto repozitáři se nachází veškeré soubory nutné k sestavení projektu. Po zaznamenání změny v repozitáři se provedou veškeré automatizované činnosti, hlavní z nich je sestavení projektu. Sestavení projektu může zahrnovat kompilaci, přesun souborů nebo stažení potřebných závislostí projektu [9]. Sestavení aplikace je pak v systému průběžné integrace automatizováno pomocí nástrojů jako například Ant, Maven, Gradle či Make. Nástroj Maven byl využit i v této práci a bude přiblížen v části 3.3.3.

Dalším aspektem průběžné integrace je využití automatizovaného testování. Mezi výhody patří možnost nové chyby rychle odhalit a ihned informovat o chybě zainteresované osoby. Testovací sada by měla být spustitelná jednoduchým příkazem a měla by poukazovat na selhané testy. Automatické testování by také mělo při objevení chyby způsobit přerušení sestavení projektu [9].

Pro správné využití průběžné integrace je nutné udržovat repozitář projektu co nejaktuálnější, proto by každý vývojář měl alespoň jednou denně (ale pokud možno častěji) veškeré provedené změny ukládat do repozitáře. Čím častěji je repozitář aktualizován, tím rychleji a jednodušeji lze nalézt a vyřešit případný konflikt v kódu vývojářů. Okamžitá zpětná vazba je jednou z hlavních vlastností průběžné integrace [9].

3 Technologie pro funkční testování webové aplikace

V dnešní době existuje pro testování webových aplikací spousta různých nástrojů. Tyto technologie se liší jak v podpoře různých webových prohlížečů, skriptovacích jazyků nebo cenou. Hlavním cílem této kapitoly není představení velkého množství těchto nástrojů, ale spíše podrobněji představit ty, které byly použity či s použitým řešením nějakým způsobem souvisí.

Hlavní technologií, která byla při vytváření testů použita, je testovací platforma Arquillian a její rozšíření Graphene a Drone. Vlastnosti a výhody použití této platformy budou hlavní náplní této kapitoly. Populárnější alternativou je testovací nástroj Selenium, jehož komponenta WebDriver je využita i ve dvou použitých rozšířeních platformy Arquillian. Kromě těchto nástrojů zde budou představeny i další specializované nástroje využité například k sestavení aplikace, průběžné integraci či verzování projektu.

3.1 Selenium

Selenium je jeden z nejpopulárnějších nástrojů sloužících k automatickému testování webových aplikací. Jedná se o open source aplikační rámec, jehož součástí je několik komponent, nyní hlavně Selenium IDE a Selenium WebDriver. Původní část projektu Selenium Remote Control je již zastaralá a jejím nástupcem je Selenium Webdriver, který také obsahuje schopnosti původně samostatný nástroj Selenium Grid. Selenium jako celek nabízí velice flexibilní platformu umožňující tvorbu testů pro širokou škálu webových prohlížečů a operačních systémů.

Ve své podstatě Selenium umožňuje ovládat instance webového prohlížeče a emulovat jeho interakci s uživatelem. Touto interakcí mohou být různé běžně prováděné úkony, jako například pohyb myší, vyplnění textového pole nebo kliknutí na tlačítka či odkazy. Jedním z principů projektu Selenium je také jednotné rozhraní pro všechny běžné prohlížeče, což umožňuje jeden test spouštět na růz-

ných webových prohlížečích bez nutných úprav. Selenium nástroje jsou také vhodné k dalšímu rozšíření jako součást jiných aplikací [8].

3.1.1 Selenium WebDriver

Po dlouhou dobu byl hlavním modulem Selenium projektu Selenium Remote Control (nazývaný také Selenium RC či Selenium 1.0), ten měl ale značné nedostatky. Tím hlavním je samotný princip fungování Selenium RC, a to vkládání funkcí psaných pomocí JavaScriptu do webového prohlížeče po jeho načtení. Tento princip činil práci s dynamickými webovými stránkami náročnou. Další nevýhodou bylo rozdílné chování JavaScriptu v různých webových prohlížečích, což mohlo způsobit neočekávané chování testů.

Přirozenou evolucí projektu Selenium RC se stal Selenium WebDriver (nazývaný také Selenium 2.0). Ten vznikl sloučením projektu WebDriver a původního Selenium 1.0 a oproti starší verzi nabízí mnohá vylepšení. Tím hlavním je využití tzv. *WebDriver API*[1], které využívá nativní podpory komunikovat přímo s webovým prohlížečem. Každý podporovaný webový prohlížeč má svůj vlastní WebDriver, který zajišťuje specifika práce s tímto prohlížečem. Uživatel pak už jen využívá WebDriver rozhraní, aniž by musel řešit rozdílné chování webových prohlížečů.

Podpora webových prohlížečů a programovacích jazyků

Výhodou Selenium WebDriveru je podpora běžných prohlížečů a programovacích jazyků. Podporováný jsou následující prohlížeče: Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera a také dva prohlížeče bez uživatelského rozhraní, HtmlUnit a PhantomJS. Selenium WebDriver podporuje velké množství programovacích jazyků, mezi ty hlavní patří například Java, C#, Ruby, Python a JavaScript.

1. Application Programming Interface

3.1.2 Selenium IDE

Druhým Selenium nástrojem pro tvorbu testů je Selenium IDE[2]. Jedná se o doplněk pro internetový prohlížeč Mozilla Firefox, díky čemuž je jeho instalace jednoduchá. Stejně tak je snadné i jeho použití, není nutné mít rozsáhlé programátorské znalosti. Tento nástroj slouží k nahrání interakce s internetovým prohlížečem. Tato interakce je uložena jako posloupnost speciálních Selenium příkazů, nazývaných *Selenese*. Ty se skládají z příkazu a až dvou argumentů, většinou jde o identifikátor nějakého prvku na webové stránce a hodnoty, která je příkazu předávána. Posloupnost těchto příkazů je možné dále upravovat a vkládat další *Selenese* příkazy. Tato posloupnost příkazů tvoří jeden testový případ, který lze zpětně přehrát. Více testových případů lze seskupit do testovací sady. [7]

Jednoduchost tohoto nástroje je jeho silná stránka, nicméně není příliš vhodný k tvrobě složitějších testů. Pro větší množství robustnějších testů je vhodné použít spíše nástroj Selenium WebDriver. Další nevýhodou je také vázanost pouze na internetový prohlížeč Mozilla Firefox. Tuto vlastnost lze částečně obejít exportem testových případů do programovacího jazyka Java, Ruby, Python nebo C# a následným udržováním v některém z testovacích aplikačních rámců.

3.1.3 Selenium Grid

Za zmínku stojí také nástroj původně nazývaný Selenium Grid. Ve své podstatě se jedná o prostředí k distribuovanému provádění testů. To sebou přináší dvě hlavní výhody. Je možné testy současně provádět na více strojích, každý s různým webovým prohlížečem (případně jinými verzemi stejného prohlížeče) nebo operačním systémem. Další z výhod je snížení času potřebného k provedení testů díky paralelnímu zpracování na více strojích. Momentálně není tento nástroj dostupný samostatně, ale jeho funkcionalita je obsažena v nástroji Selenium RC, který je také součástí nástroje Selenium WebDriver [7].

---

2. Integrated Development Environment

3.2 Arquillian

Arquillian je testovací platformou vyvinutou za účelem tvorby automatizovaných integračních, funkčních a akceptačních testů v prostředí Java EE[3]. Zjednodušuje testování pomocí správy běhu programu, čehož dosahuje pomocí [11]:

- Správy životního cyklu kontejneru (kontejnerů)
- Zabalením testovacího případu, všech závislých tříd a zdrojů do ShrinkWrap [13] archivu (archivů)
- Nasazení archivu (archivů) do kontejneru (kontejnerů)
- Obohacení testového případu poskytnutím injekce závislostí a ostatních deklarativních služeb
- Provádění testů uvnitř (nebo proti) kontejneru
- Zachycení výsledků a jejich předání spouštěči testů, který o výsledcích reportuje

Projekt Arquillian se řídí třemi základními principy. Prvním z nich je přenositelnost testů do jakéhokoliv podporovaného kontejneru. Jelikož rozhraní specifické pro jednotlivé kontejnery není v testech používáno, lze verifikovat přenositelnost aplikace spuštěním testů v různých kontejnerech.

Dalším principem je možnost spouštět testy jak přímo z vývojového prostředí, tak pomocí nástroje pro sestavení aplikace. Z vývojového prostředí je při testování možné přeskočit krok sestavení aplikace a ušetřit čas. Navíc vývojář může pracovat ve svém vývojovém prostředí, s nímž je obeznámen. Tyto výhody neznemožňují využít testy v systému průběžné integrace.

Poslední z řídících principů je rozšiřování či integrování stávajících testovacích aplikačních rámců. Spouštění Arquillian testů je proto jednoduché jak přímo z vývojového prostředí výběrem *„Run As > Test“* nebo spuštěním cíle *„test“* v nástroji pro sestavení aplikace.

Aby nebylo zbytečně komplikováno sestavení aplikace, tak lze Arquillian integrovat nástroji jako Maven (3.3.3) nebo Ant, které se

3. Java Platform, Enterprise Edition

o sestavení aplikace postarají. Arquillian také nabízí několik rozšíření. Těmi hlavními pro účely této práce jsou Arquillian Graphene a Arquillian Drone, které budou nyní představeny. Konkrétní způsob použití těchto rozšíření bude dále vysvětlen v páté kapitole (5).

3.2.1 Arquillian Graphene

Arquillian Graphene si klade za cíl rozšířit možnosti Selenium WebDriver technologie a přidává spousu vlastností umožňujících psát znovu použitelné a udržovatelné funkční testy. Důraz je také kladen na jednoduché rozhraní, přenositelnost mezi webovými prohlížeči a psaní robustních testů podporujících AJAX[4] technologii. Arquillian Graphene je závislý na rozšíření Arquillian Drone, které se stará o životní cyklus použitých webových prohlížečů. Rozšíření Drone bude podrobněji popsáno v části 3.2.2.

Jednou z vlastností rozšíření Graphene je podporování ve využití abstrakcí webových stránek (*Page Abstractions*), konkrétně objekty stránky (*Page Objects*) a fragmenty stránky (*Page Fragments*). Objekt stránky zapouzdřuje strukturu testované stránky do jediného objektu, se kterým následně test komunikuje. Takto zapouzdřený objekt by měl vývojáři nabízet stejné služby jako modelovaná stránka. Zároveň by ale měl být jediným místem, které pracuje s vnitřní strukturou webové stránky. Jednoduše řečeno se jedná o rozhraní testované stránky. Díky tomu je při změně uživatelského rozhraní testované webové stránky nutné provést změnu jen na jednom místě v objektu stránky. Výhodou využití objektů stránky je také omezení duplicity a tvorba robustního kódu [7].

Druhou možnou abstrakcí webové stránky jsou fragmenty stránky. Jde o velice podobný koncept jako v případě objektů stránky. Také jde o zapouzdření struktury, ale na rozdíl od objektů stránky nejde o zapouzdření jedné určité webové stránky. Fragment stránky zapouzdřuje pouze určitou malou součást stránky, která je znovu použitelná napříč všemi testovanými stránkami [14]. Příkladem může být navigační menu, které je součástí více webových stránek. Pokud by mělo být využito pouze objektů stránek, tak by struktura tohoto navigačního menu musela být opakovaně definována ve všech objektech

4. Asynchronous JavaScript and XML

stránky, které jej využívají. Lze ale definovat pouze jeden fragment stránky pro toto navigační menu a v objektech stránky pouze využívat tento fragment. Konkrétní použití obou druhů abstrakcí bude popsáno v části (5.4).

Dále Graphene nabízí vylepšení synchronizace s webovým prohlížečem. Toto je velice důležitá vlastnost, jelikož WebDriver příkazy jsou vykonávány rychleji, než dokáže webový prohlížeč provést změnu stavu stránky. Pomocí tzv. *Waiting API* rozhraní [14] lze test synchronizovat čekáním, například dokud se na stránce nezobrazí určitý element nebo dokud nebude splněna nějaká podmínka. Další možností synchronizace je využití tzv. střežení požadavků(*Request Guards*). To umožňuje ověřit, zda daná interakce s webovým prohlížečem vyvolala příslušný požadavek na server. Je možno ověřovat, zda došlo k HTTP nebo AJAX požadavku, případně zda nedošlo k žádnému požadavku. Vykonávání interakce s prohlížečem musí skončit v časovém intervalu příslušném druhu požadavku. Pokud skončí dříve, tak se okamžitě pokračuje ve vykonávání testu.

Dalším vylepšením oproti Selenium WebDriver technologii je také rozšíření možností, jak nalézt požadované prvky testovaného uživatelského rozhraní. Kromě metody *@FindBy*, kterou využívá i WebDriver, podporuje Graphene také možnost vytvořit vlastní vyhledávací strategie. Metoda *@FindByJQuery*, která umožňuje vyhledávat na základě JQuery funkce, je v Graphene již vytvořená a okamžitě použitelná.

3.2.2 Arquillian Drone

Rozšíření Drone umožňuje platformě Arquillian využívat technologii Selenium WebDriver, která se stará o práci s uživatelským rozhraním webových stránek. WebDriver poskytuje vlastní rozhraní, pomocí nějž lze komunikovat s webovým prohlížečem podobným způsobem, jakým uživatel využívá běžný webový prohlížeč. Arquillian Drone umožňuje využít toto rozhraní a také přináší další výhody [15]:

- Správa životního cyklu webového prohlížeče
- Interakce s kontejnery, které poskytuje Arquillian
- Jednoduché použití více prohlížečů v jednom testu

- Konfigurace je na jediném místě mimo Java kód
- Podpora injektování objektů stránek, fragmentů stránek, střežení požadavků a dalších vlastností rozšíření Graphene
- Testování mobilních prohlížečů (Arquillian Droidium)
- Integrace JavaScriptového nástroje QUnit pro vykonávání testové sady
- Kompatibilní s technologiemi Selenium WebDriver a Grid

Práce s tímto nástrojem bude vysvětlena v části 5.4.1.

3.3 Ostatní nástroje

Pro psaní testů byla zvolena platforma Arquillian, jelikož výše popsané výhody oproti Selenium nástrojům značně zjednodušují psaní testů. To sebou také nese jistá omezení spojená s tímto výběrem. Jelikož Arquillian podporuje pouze programovací jazyk Java, budou testy psány v tomto jazyce. Dále jsou popsány některé další nástroje a technologie, které byly použity.

3.3.1 Git

Testovaná aplikace je vyvíjena za pomoci open source verzovacího systému Git [16]. S tím je také spojena webová služba GitHub, která obsahuje repozitář projektu. Repozitář, v němž je obsažena testovaná aplikace bude následně využit pro systém průběžné integrace. Vytvořené testy se ale nacházejí v jiném repozitáři, který jsem pro tyto účely vytvořil. Práci s oběma repozitáři zajišťuje systém průběžné integrace Travis CI, jehož použití je popsáno v kapitole šest (6).

3.3.2 JUnit

JUnit [17] je aplikační rámec pro psaní jednotkových testů v jazyce Java. Tento aplikační rámec využívá k psaní testů také platforma

Arquillian. Pro psaní jednotkových testů jsem mohl vybrat také nástroj TestNG, který také Arquillian podporuje. Nicméně JUnit je standardně dostupný ve vývojovém prostředí Eclipse, které jsem pro psaní testů využíval, proto se jevil jako rozumnější volba.

3.3.3 Maven

Apache Maven [18] je nástroj pro správu, řízení a sestavení aplikací. Základem fungování Mavenu je tzv. *Project Object Model* soubor. Jde o soubor, v němž je popsána struktura projektu, všechny použité rozšíření, závislosti a další užitečné funkce, jako například nastavení profilů či definování vlastních proměnných. Tento soubor je popsán pomocí značkovacího jazyka XML[5]. V projektu je také využit Maven Surefire Plugin, což je rozšíření zajišťující mimo jiné spouštění jednotkových testů. Konkrétní Maven nastavení použité v této práci bude popsáno v části (5.2).

3.3.4 Travis CI

Travis CI je webová služba umožňující testovat a sestavovat projekty pomocí systému průběžné integrace. Ta umožňuje po zaznamenání změny ve vybraném repozitáři provést automatizované sestavení aplikace dle konfigurace nástroje. Kromě sestavení aplikace lze také spouštět připravené testy a o výsledku testování informovat zainteresované osoby. Důvodem pro výběr tohoto nástroje oproti jiným populárním řešením průběžné integrace (například Jenkins CI) byla jednoduchost použití i konfigurace a zároveň podpora všech používaných technologií. Použitím tohoto systému se bude podrobněji zabývat šestá kapitola (6).

3.3.5 Webové prohlížeče

V projektu bylo při testování využito více různých prohlížečů. Díky využití platformy Arquillian s rozšířením Drone je využití více prohlížečů rozumné a samotné testy jsou na výběru prohlížečů nezávislé. Z nejběžnějších prohlížečů byly vybrány prohlížeče Google

5. Extensible Markup Language

Chrome a Mozilla Firefox, jelikož jeden z těchto prohlížečů používá více než 85% uživatelů [20].

Dále byl vybrán také webový prohlížeč bez uživatelského rozhraní PhantomJS. Výhodou prohlížečů bez uživatelského rozhraní jsou mimo jiné nižší požadavky na zdroje, mírné zrychlení vykonávání testů a jednodušší využití v systémech průběžné integrace. Tento prohlížeč má vestavěnou podporu v použitém systému průběžné integrace Travis CI a je také podporován WebDriver technologií a tím pádem i použitým rozšířením Arquillian Drone.

4 Aplikace Slepé Mapy a její analýza

Hlavní částí této práce je vytvoření funkčních testů pro aplikaci Slepé Mapy. V této kapitole bude aplikace přiblížena a analyzována její funkcionalita, pro kterou budou navrhnuty testovací scénáře. Testovacím scénářem je myšlen popis funkcionality, která se bude testovat. Na základě testovacích scénářu budou dále vytvořeny testovací případy, které krok za krokem popisují testovací proces. Z těchto testovacích případů budou následně odvozeny samotné testy napsané pomocí platformy Arquillian v jazyce Java.

Cílem aplikace je hravou formou procvičovat znalosti zeměpisu. Podle vědomostí uživatele aplikace volí podobu a frekvenci opakování otázek. Znalosti uživatele jsou barevně zobrazeny na mapě, což uživateli vizuálně pomáhá určit jeho míru znalosti. Tuto znalost může poměřovat jak sám se sebou, tak proti průměru všech uživatelů. Na výběr k procvičování jsou slepé mapy světa, všech kontinentů a vybraných států. Na těchto mapách lze studovat státy(regiony, provincie, kraje, atd…), města, řeky, jezera, pohoří a ostrovy.

Procvičování probíhá v sadě otázek po deseti. Po dokončení sady otázek dojde k vyhodnocení úspěšnosti a uživatel dostane body, pomocí kterých zvyšuje úroveň svého profilu. Pokud není uživatel spokojený s dosaženým výsledkem, může opakovat procvičování stejné mapy, případně vybrat mapu jinou. Po několika sadách otázek je uživatel vyzván k subjektivnímu určení obtížnosti otázek, na které odpovídal. Existují 3 typy otázek, lišící se způsobem výběru správné odpovědi:

- kliknutím na specifické místo do slepé mapy
- pojmenováním jednoho místa vyznačeného na slepé mapě
- kliknutím na jedno z dvou nebo více míst zaznačených na slepé mapě

Aplikace má také vlastní *staging server*, na němž jsou zkoušeny změny v aplikaci před zavedením do hlavní webové stránky aplikace. Testování nejprve probíhalo na webové stránce určené i pro uživatele, nicméně cílem je testovat aplikaci běžící na *staging serveru*.

4.1 Identifikace testovacích scénářů

Testovaná aplikace Slepé Mapy nedisponuje vlastní specifikací, proto je nutné věnovat zvláštní pozornost analýze aplikace a výběru správných částí aplikace k testování. Jednoduše řečeno jsem se nejdříve zabýval hledáním testovacích scénářů, které lze chápat jako odpověď na otázku „Co se bude testovat?“. Identifikace scénářů probíhala tak, že jsem systematicky procházel možné stránky aplikace a kladl si otázku: „Jakou funkcionalitu webová aplikace nabízí uživateli?“ Cílem bylo nalézt veškeré funkce, které má uživatel dostupné a z těchto vybrat podmnožinu těch nejdůležitějších, které budou testovány. Níže bude stručně popsána testovaná funkcionalita aplikace, rozdělená na ucelené části. Tato funkcionalita bude dokumentována pro každou testovanou část aplikace v podobě následující tabulky obsahující identifikátor a účel scénáře a také identifikátor testovacího případu (či případů), které se zabývají jeho řešením. Celá dokumentace se nachází v příloze.

ID scénáře	Účel scénáře	Testovací případ
A_SC1	Ověřit funkčnost odeslání zpětné vazby	A1
A_SC2	Ověřit funkčnost zavření formuláře zpětné vazby	A2
A_SC3.1	Ověřit funkčnost chybových hlášek formuláře zpětné vazby	A3
A_SC3.2	Ověřit zavření chybových hlášek formuláře zpětné vazby	A4

Tabulka 4.1: Testovací scénáře zpětné vazby

4.1.1 Přehledová mapa

Jednou z nejdůležitějších částí testované aplikace jsou bezesporu samotné mapy. Jejich správná funkčnost bude pro bezchybný běh aplikace klíčová a bude zahrnuta do testovací sady. Konkrétně lze na

mapě testovat spoustu vlastností. Hlavní bude ověření správnosti zobrazených informací o místech na mapě, všechna místa musí mít svůj popis zobrazený po najetí myší. Veškerá procvičovaná místa musí být zahrnuta v seznamu v ovládacím panelu mapy, který mimo jiné obsahuje i přepínání mezi druhy map, případně skrytí a odkrytí části mapy (například všech států) a odkazy na procvičování. Zmíněná funkcionalita bude také otestována. Dále bude testována funkčnost tlačítek pro přepínání znalostí mezi průměrným a aktuálním uživatelem a také možnost přiblížit či oddálit mapu. Ve spojení s procvičováním bude také vhodné otestovat, zda se na mapách správně zaznačují změny ve znalostech uživatele.

4.1.2 Procvičování

Procvičování slepých map je hlavní funkcionalitou této aplikace, proto se procvičování bude věnovat podstatná část testů. Při procvičování mají mapy trochu jiné vlastnosti oproti přehledovým mapám. Tou hlavní je možnost při odpovídání na otázku kliknout do mapy na vybrané místo, které se následně obarví dle správnosti odpovědi. Je také třeba otestovat vyznačení míst, která lze vybrat jako odpověď. Při výběru odpovědi z více míst vyznačených na mapě by kliknutí na jiná než vyznačená místa neměla provést žádnou akci. Dalším prvkem při procvičování je ovládací panel, který obsahuje aktuální otázku, tlačítka pro pokračování a dle typu otázky i seznam možných odpovědí a tlačítko pro zvýraznění možných odpovědí v mapě. Kromě zde zmíněné funkcionality bude předmětem testování ukazatel postupu, který dává uživateli vizuální informaci o zbývajícím počtu otázek. Poslední testovanou funkcionalitou bude panel vyhodnocení, který zobrazuje přehled úspěšnosti v procvičované sadě otázek a obsahuje tlačítka pro pokračování procvičování, přechod na světovou mapu a na výběr jiné mapy.

4.1.3 Přihlášení a registrace uživatele

Pro zachování svého postupu se může uživatel registrovat. Pro vytvoření účtu se lze registrovat kliknutím na možnost „Přihlásit se“ v navigačním menu, což vyvolá přihlašovací formulář. V tomto formuláři lze kromě možnosti přihlášení se vybrat také možnost „Registrovat

se“, což vyvolá registrační formulář pro zadání registračních údajů. Je možné provést registraci vytvořením nového účtu na stránce aplikace, což vyžaduje zadat platný e-mail a heslo. Dále aplikace nabízí možnost registrovat se a následně se přihlašovat pomocí uživatelova existujícího Facebook či Google účtu. Tato funkcionalita si vynucuje interakci s webovou stránkou mimo testovanou aplikaci, i přesto se ale testování bude věnovat i této části. Nicméně je nutné mít na paměti, že funkcionalita registrace a přihlášení pomocí Google a Facebook účtu se může nečekaně změnit, testy zabývající se touto částí aplikace proto budou méně robustní.

4.1.4 Navigační menu

Navigační menu je hlavním nástrojem k pohybu po stránce. Testování funkcionality této části aplikace bude přímočaré, půjde hlavně o funkčnost všech odkazů a rozbalovacích menu, které navigační menu poskytuje. Dále lze z navigačního menu vyvolat formulář pro přihlášení uživatele a možnost změnit jazyk. Po přihlášení uživatele se také objeví dříve skryté rozbalovací menu umožňující dostat se do profilu uživatele a případně se odhlásit.

4.1.5 Zpětná vazba

Poslední testovanou částí bude správná funkcionalita odesílání zpětné vazby. Tlačítko odkazující na formulář zpětné vazby se nachází na každé stránce aplikace. Zpětnou vazbu lze zaslat buďto anonymně, nebo lze nepovinně vyplnit svou e-mailovou adresu. Tento formulář o úspěšném či neúspěšném odeslání zpětné vazby informuje krátkou zprávou.

4.2 Tvorba testovacích případů

Pomocí testovacích případů je na základě testovacích scénářů přesně definován postup kroků, které se pro otestování daného scénáře musí vykonat. Testovací případy jsou proto založené na identifikovaných testovacích scénářích. Každý testovací scénář musí být zcela pokryt jedním či více testovacími případy. Stejně jako testovací scénáře jsou

i testovací případy zdokumentovány tabulkou obsahující nejpodstatnější informace o testovacím případu. Pro jednoduchost jsem vybral jen ty nejnutnější údaje, které lze využít pro napsaní testů. Těmi hlavními údaji jsou účel a kroky nutné k vykonání testovacího případu a také očekávaný výsledek testování. Zde je pro upřesnění uveden příklad dokumentace testovacího případu pro ověření funkčnosti odesílání zpětné vazby.

ID	A1
Název	Odeslání zpětné vazby
Účel	Ověřit, zda lze odeslat text jako zpětnou vazbu
Vstupní podmínky	Je dostupné tlačítko pro zpětnou vazbu
Kroky	1. Kliknout na tlačítko pro otevření formuláře zpětné vazby 2. Vepsat text do textového pole pro zpětnou vazbu 3. Vyplnit email 4. Kliknout na tlačítko pro odeslání zpětné vazby
Očekávaný výsledek	1. Objeví se zpráva o úspěšném odeslání 2. Formulář zůstane otevřený
Poznámky	

Tabulka 4.2: Testovací případ odeslání zpětné vazby

5 Tvorba testů

V třetí kapitole (3) byly představeny nástroje, pomocí nichž lze provádět testování webové aplikace, zatímco čtvrtá kapitola (4) se zabývala identifikací funkcionality, která bude testována. Toto nám dává dostatečné teoretické základy pro samotnou tvorbu testů pomocí vhodných technologií, tudíž se již v této kapitole můžeme zabývat praktickou částí této práce. Hlavním tématem této kapitoly bude představení tvorby funkčních testů pomocí testovací platformy Arquillian a rozšíření Graphene a Drone. Podstatnou část před psaním testovacích tříd tvoří také nastavení vývojového prostředí a nástroje pro sestavení aplikace. Větší část kapitoly bude také věnována tvorbě abstrakcí testované aplikace pomocí fragmentů stránek a objektů stránek. Až po jejich vytvoření bude možné se věnovat psaní testovacích tříd.

5.1 Příprava vývojového prostředí

Prvním krokem v přípravě na testování bylo nastavení vývojového prostředí. Jelikož jsem měl předchozí zkušenosti s nástrojem Eclipse, zvolil jsem právě tento. Výhodou je také fakt, že v nástroji Eclipse je standardně zahrnut i testovací aplikační rámec JUnit, nebylo tedy třeba vývojové prostředí pro práci s JUnit připravovat. Spouštění JUnit testů z vývojového prostředí je jednoduché, stačí kliknout pravým tlačítkem myši na požadovanou testovací třídu a zvolit možnosti *Run As -> JUnit Test*. Tento způsob testování umožňuje pohodlně spouštět testy a využívat ladící nástroje. Jelikož ale v projektu bude využit nástroj pro sestavení aplikace Maven, je vhodné nainstalovat rozšíření M2Eclipse. Všechna rozšíření lze jednoduše nainstalovat pomocí služby Eclipse Marketplace také přímo z vývojového prostředí. Rozšíření M2Eclipse není nutností, s nástrojem Maven lze také pracovat z příkazové řádky.

Dalším využitým rozšířením je EGit, který umožňuje využívat verzovací systém Git přímo z vývojového prostředí Eclipse. Toto rozšíření v mém případě práci se systémem Git značně zjednodušuje, například nejběžnější prováděnou akci *commit* lze provést jednoduše kliknutím pravým tlačítkem myši na třídu a volbou možností *Team -> Commit...*, vyplněním zprávy o provedených změnách a potvrzením

možností *Commit and Push*. I toto rozšíření není nutné, pokud je pro vývojáře pohodlnější používání příkazové řádky, v mém případě ale zvyšuje efektivitu práce.

5.2 Konfigurace a použití nástroje Maven

Nástroje Maven a Arquillian byly popsány ve třetí kapitole (3), zde se budeme zabývat jejich konkrétním nastavením. Maven je konfigurován v souboru *pom.xml*, který se nachází v kořenovém adresáři sestavovaného projektu. V tomto projektu konfigurační soubor slouží ke 3 hlavním účelům – správě závislostí a všech rozšíření, nastavení uživatelských proměnných a správě profilů. Celý soubor se nachází v příloze.

Projektem využívané závislosti jsou definovány v elementu *Dependency Management*. Každá závislost má vlastní *Dependency* element, v němž jsou uvedeny jednotlivé informace, podle nichž Maven vyhledá požadovanou závislost ve vzdálených repozitářích. Maven zároveň tranzitivně vyhledá veškeré závislosti, které námi specifikovaná závislost vyžaduje.

Zdrojový kód 5.1: Ukázka zápisu závislosti v souboru *pom.xml*

```
<dependency>
    <groupId>org.jboss.arquillian</groupId>
    <artifactId>arquillian-bom</artifactId>
    <version>1.1.7.Final</version>
    <scope>import</scope>
    <type>pom</type>
</dependency>
```

Kromě vlastních proměnných Maven také nabízí možnost definovat uživatelské proměnné, které se definují v elementu *properties*. V těchto proměnných mimo jiné ukládám informace o uživatelských jménech a heslech používaných pro přihlášení do aplikace pomocí různých služeb. Jelikož jsou tyto údaje definované jen na tomto místě, je možné při změně těchto údajů provést změnu jen v souboru *pom.xml* a nikoliv na všech místech výskytu v testech. Hodnoty proměnných lze také jednoduše měnit z příkazové řádky přidáním *-DnázevProměnné=hodnotaProměnné* do Maven příkazu. Jednou z uži-

tečných standardních Maven proměnných použitých během vytváření testů je proměnná *test*, která umožňuje spouštět pouze jednu testovací třídu z celé testovací sady, případně jen jeden test z testovací třídy.

Pro využití uživatelem definovaných proměnných v testech je ale nutné proměnné definovat také v použitém rozšíření Maven Surefire Plugin, jehož proměnným předávám hodnoty uložené v definovaných Maven proměnných. Ty pak budou dostupné i při spouštění testů, což lze provést pomocí příkazu *mvn clean verify*, případně *mvn test*. Rozdíl mezi těmito způsoby spuštění testů je takový, že *mvn test* pouze spustí test pomocí vhodného testovacího aplikačního rámce (v tomto případě JUnit), zatímco *mvn clean verify* navíc provede kontrolu validity a splnění určité úrovně kvality projektu.

Posledním nastavením nástroje Maven je definování vlastních profilů. Tyto profily slouží k výběru webového prohlížeče, který bude k testování použit. Jeden z profilů je označen jako výchozí, ten je používán pokud není specifikován jiný profil. To lze provést jednoduše z příkazové řádky pomocí Maven příkazu *-PnázevProfilu*. Nyní jsou nastaveny profily pro tři prohlížeče: Mozilla Firefox, Google Chrome a PhantomJS. Přidání podpory dalších prohlížečů lze provést jednoduše přidáním dalšího profilu do souboru *pom.xml* a dle druhu prohlížeče také konfigurací souboru *arquillian.xml*, který se stará o nastavení nástroje Arquillian.

5.3 Konfigurace nástroje Arquillian

Nástroj Arquillian a jeho rozšíření jsou konfigurovány v souboru *arquillian.xml*, dostupný v adresáři projektu *\src\test\resources*. Tento konfigurační soubor není povinný, v případě této práce slouží pouze pro nastavení rozšíření Arquillian Drone. V konfiguračním souboru jsou nastavovány tři proměnné, hodnoty těchto proměnných jsou přebírány z Maven nastavení. Proto lze nastavení tohoto souboru měnit stejně jako konfiguraci Maven i přes příkazový řádek. Proměnná *browser* určuje, který webový prohlížeč bude Arquillian Drone využívat pro testování. Proměnná *dimensions* slouží k nastavení velikosti okna webového prohlížeče. Toto nastavení je nutné hlavně pro prohlížeč PhantomJS, jelikož ten má ve výchozím nastavení velice malé

rozměry, což vedlo k chybám v testech. Poslední proměnnou je cesta ke spustitelnému souboru prohlížeče Google Chrome.

5.4 Abstrakce testovaných stránek

Základními stavebními kameny při tvorbě testů budou abstrakce stránek (fragmenty stránek a objekty stránek). Ty byly představeny v části 3.2.1, zde se budeme zabývat převážně jejich vytvářením. Postup při jejich tvorbě bude vždy podrobně vysvětlen na jednom konkrétním případu pro oba druhy abstrakcí.

5.4.1 Fragmenty stránek

Tvorba fragmentů stránky bude stěžejní částí v procesu tvorby testů, jelikož testy budou právě využívat funkcionalitu poskytovanou vytvořenými fragmenty stránek. Fragmenty budou představovat takové části uživatelského rozhraní, které se buďto opakují na více stránkách testované aplikace, případně seskupují určitou funkcionalitu. Každý fragment stránky se skládá z jedné třídy jazyka Java a všechny fragmenty jsou obsaženy v balíčku *cz.muni.proso.geography.fragment*. Tvorba konkrétního fragmentu stránky bude podrobněji představena na fragmentu pro přihlášení uživatele, nicméně alespoň zběžně budou představeny všechny vytvořené fragmenty.

Tvorba fragmentu stránky

Jako každá běžná třída jazyka Java se i fragment skládá z atributů a metod. V případě fragmentu stránky jsou atributy tvořeny jednotlivými elementy uživatelského rozhraní, které fragment představuje. Fragmenty stránek do sebe můžou být také vnořovány, atributem může tedy být i fragment stránky, jehož deklarace je identická s deklarací obyčejného elementu webové stránky. Příkladem takového elementu může být element textového pole pro přihlašovací heslo.

Zdrojový kód 5.2: Atribut představující textové pole pro heslo

```
@FindBy(xpath = "./div[2]/form/div[2]/input")
private WebElement loginPassword;
```

Anotace *@FindBy* slouží k nalezení elementu (případně kořenu celého fragmentu stránky nebo objektu stránky) podle lokační strategie zvolené v argumentu anotace. Použití jazyka XPath lze považovat za přehlednější než jiné lokační strategie, jelikož umožňuje jednoduše využívat relativní cestu k elementu, což zkracuje zápis a zvyšuje přehlednost oproti využití například CSS lokátorů. Spolehlivější a robustnější je hledání elementů podle ID atributu, nicméně podstatná část elementů využitých v testech tímto atributem nedisponuje.

WebElement je datový typ reprezentující HTML element. Většina operací s uživatelským rozhraním probíhá zavoláním příslušné metody na tomto datového typu. *WebElement* umožňuje provádět běžné operace, jako například kliknutí na element, zjištění jeho polohy, zjištění hodnot atributů elementu či nalezení jiného elementu uvnitř tohoto elementu. Rozšířením tohoto datového typu je *GrapheneElement*, což je datový typ specifický pro rozšíření Arquillian Graphene. Ten nabízí veškerou funkcionalitu datového typu *WebElement* a navíc má velice užitečnou metodu *isPresent()*. Tato metoda umožňuje rozhodnout, zda se element nachází na stránce. Použití této metody předchází výjimkám typu *NoSuchElementException()*.

Vlastností fragmentu stránky je zapouzdření určité funkcionality, kterou část webové stránky nabízí. Metody fragmentu stránky by proto měly nabízet přesně takovou funkcionalitu, kterou disponuje modelovaná webová stránka.

Zdrojový kód 5.3: Metoda pro vepsání hesla do formuláře

```
/**
 * Writes the specified password into the password
 * field. Clears the field first.
 *
 * @param password
 *  password to write into the password field
 */
public void inputPassword(String password) {
    Graphene.waitAjax().until().
        element(loginPassword).is().present();
    loginPassword.clear();
    loginPassword.sendKeys(password);
}
```

První příkaz metody *inputPassword* slouží k synchronizaci s webovou stránkou. Tento příkaz zajistí, že následující příkazy nebudou vykonány, dokud se používaný element neobjeví na obrazovce. Doba čekání je nastavena na 2 sekundy (což zhruba odpovídá době, za kterou by měla být vykonána AJAX akce) pomocí metody *waitAjax()*. Je zde využito Graphene rozhraní pro čekání *Waiting API*, které umožňuje pracovat s čekáním jednodušeji a přehledněji. Pokud by zde toto čekání nebylo, mohlo by se stát, že testy využívající tuto metodu by skončily s chybou oznamující, že na webové stránce nelze nalézt určitý element.

Druhý a třetí příkaz metody *inputPassword* pracují přímo s výše deklarovaným atributem *loginPassword*. Zde jsou využity dvě metody datového typu *WebElement*. Metoda *clear()* zajistí, že pole pro heslo bude prázdné. Následující metoda *sendKeys()* slouží k vepsání textového řetězce do elementu. Pokud by zde nebyla metoda *clear()*, mohlo by v případě neprázdného textového pole dojít k vepsání jiného hesla, než bylo zadáno v argumentu metody.

V případě, že je potřeba poskytnout ve fragmentu stránky, objektu stránky či testu také funkce WebDriver rozhraní, je možné pomocí anotace *@Drone* poskytnout instanci webového prohlížeče. Arquillian Drone umožňuje držet více WebDriver instancí, což ale v této práci nebylo potřeba.

Zdrojový kód 5.4: Injekce instance webového prohlížeče pomocí Arquillian Drone

```
@Drone
private WebDriver browser;
```

Fragmenty pro přihlášení

Kromě fragmentu *Login.java* se přihlašováním zabývají fragmenty *FacebookAuth.java*, *FacebookConfirmAuth.java*, *GoogleAuth.java* a *GoogleConfirmAuth.java*. Lze zde také zahrnout fragment pro registraci nového uživatele, *SignUp.java*. Důvodů pro toto rozdělení přihlašování do více tříd je několik. Prvním z nich je omezení fragmentů stránek na použití pouze těch elementů uživatelského rozhraní, které jsou

zahrnuty v DOM[1] struktuře kořenu fragmentu stránky. To vylučuje zahrnutí přihlašování pomocí Google a Facebook účtu do jediného fragmentu, jelikož tyto služby přesměrují uživatele na jinou webovou stránku.

Dalším důvodem rozdělení je možná změna funkcionality přihlašování pomocí těchto služeb, a to i bez vědomí vývojářů testované aplikace. Toto řešení umožňuje jednoduše spravovat tuto funkcionalitu a případně přidávat nové fragmenty, pokud se v budoucnu bude možné přihlašovat pomocí účtů jiných služeb. Pro tento případ je vytvořeno rozhraní *SocialMediaLogin.java*, které vynucuje základní funkcionalitu přihlašování.

Fragmenty pro mapu

Fragmenty stránek zabývající se funkcionalitou map se nacházejí ve třídách: *Map.java*, *MapControl.java*, *MapControlContext.java*, *MapTooltip.java*, *OverviewMap.java* a *PracticeMap.java*. Dále se prací s mapou zabývá také pomocná třída *ColourHashMap.java*, která umožňuje zjistit informace o vztahu mezi barvami vyznačujícími znalost uživatelů a jejich odpovídajícími významy. Třída *Map.java* je abstraktní třída představující obecnou mapu, zatímco třídy *OverviewMap.java* a *PracticeMap.java* jsou její implementací a představují přehledovou mapu a procvičovací mapu. Třída *MapControl.java* představuje ovládací panel mapy, obsahující přepínače druhů map a jejich viditelnost, ukazatel znalosti mapy a seznam procvičovaných položek. Procvičované položky jsou definovány ve fragmentu *MapControlContext.java*.

Fragmenty pro procvičování

Funkcionalita procvičování je řešena v třídách *Practice.java*, *PracticeFeedback.java*, *PracticeMap.java* a *PracticeSummary.java*. Třída *PracticeMap.java* se stará o funkcionalitu procvičovací mapy, například umožňuje do mapy klikat nebo získat správné odpovědi. Navíc tato třída obsahuje metodu na rozlišení typu otázky, jelikož ten lze odvodit pouze od počtu vyznačených míst na mapě. Třída *Practice.java* pak představuje panel obsahující otázku a tlačítka pro interakci s aplikací. Zároveň se v této třídě nachází ukazatel postupu a čítač otázek,

1. http://www.w3.org/DOM/Overview

který umožňuje zjistit, kolikátá otázka je momentálně zobrazena. Tato informace je důležitá, jelikož otázky jsou generovány pomocí JavaScriptu postupně a na začátku procvičování nejsou přitomny v DOM struktuře webové stránky. Čítač otázek je proto využit při hledání elementů pomocí anotace *@FindBy()*. Jedním z atributů třídy *Practice.java* je také fragment *PracticeSummary.java*, který představuje vyhodnocení procvičování, jenž se zobrazí po zodpovězení všech otázek. Posledním fragmentem procvičování je *PracticeFeedback.java*, ten představuje vyskakovací okno umožňující uživateli upravit obtížnost otázek.

Ostatní fragmenty

Zbývajícími fragmenty, zajišťující různé užitečné služby, jsou uvedeny zde. Třída *AlertMessage.java* představuje zprávy, jimiž aplikace v různých částech aplikace dává informace o úspěchu či neúspěchu interakce uživatele. Třída *NavigationMenu.java* zajišťuje funkcionalitu navigačního menu. Dalším vytvořeným fragmentem je *PracticedContext.java*, který představuje jednu položku v přehledu map. Součástí tohoto fragmentu jsou také fragmenty *ProgressButton.java* a *ProgressBar.java*. *ProgressButton.java* představuje tlačítko složené z tlačítka a ukazatele znalosti, který je definovaný ve vlastní třídě *ProgressBar.java*. Tento ukazatel znalosti je použitý na více místech aplikace, nejen v přehledu map.

5.4.2 Objekty stránek

Objekty stránek reprezentují testovanou stránku webové aplikace a jejich identifikace byla v porovnání s fragmenty stránky jednodušší. Veškeré testovací scénáře totiž probíhají buď na jednotlivých fragmentech nebo na jedné ze tří hlavních stránkách aplikace: světové mapě, přehledu map a procvičování konkrétní mapy. Stejně jako v případě fragmentů stránek každý objekt stránky představuje jednu třídu jazyka Java, nachází se v balíčku *cz.muni.proso.geography.page*. Pro společné fragmenty všech objektů stránek byla vytvořena abstraktní třída *CommonPageFragments.java*, jejíž potomci jsou samotné objekty stránek využívané v testech.

Tvorba objektu stránky

Objekt stránky je stejně jako fragment stránky tvořen jednou třídou jazyka Java. Oproti fragmentu stránky bude ale význam atributů a metod rozdílný. Atributy objektu stránky jsou tvořeny převážně dříve vytvořenými fragmenty stránky, které musí mít své *get* metody, které pak tvoří rozhraní pro komunikaci. V praxi je samozřejmě reálné, že veškerá funkcionalita modelované stránky není úplně zahrnuta v použitých fragmentech stránky. Proto je možné v objektech stránky stejně jako ve fragmentech stránky deklarovat elementy uživatelského rozhraní a metody, které s nimi pracují. Cílem je, aby objekt stránky nabízel úplně stejnou funkcionalitu jako modelovaná stránka.

Princip tvorby objektu stránky je až na zmíněné rozdíly velice podobný tvorbě fragmentů stránek. Lze využívat vlastností jako WebDriver rozhraní, anotace*@FindBy* a *@Drone*. Rozdíl ale nastává při deklaraci objektu stránky v testech. Oproti fragmentům není využita anotace *@FindBy*, ale anotace *@Page*, která nepobírá žádné argumenty. Další vlastností objektů stránek je možnost využití anotace *@Location*, které lze v argumentu předat místo, na němž se nachází modelovaná stránka. Toho lze poté využít v testech, kde není třeba deklarovat použitý objekt stránky anotací *@Page*, ale je možné využít anotace *@InitialPage* v argumentu testovací metody. Výhodou tohoto přístupu je, že se testovaná stránka při vstupu do testovací metody automaticky otevře.

5.5 Tvorba testových tříd

Po vytvoření abstrakcí stránek je již příprava na psaní testových tříd hotová. Pokud jsou fragmenty a objekty stránek vytvořeny správně, bude tvorba testů velice jednoduše realizovatelná využitím dříve vytvořených testovacích scénářů. Testovací metody budou posloupností akcí, které by tester prováděl na modelované stránce, provázané s vyhodnocováním správnosti testu na vhodných místech. Všechny tyto možné akce a funkcionalitu pomáhající s vyhodnocením výsledku testu (například metoda zjišťující přítomnost formuláře) musí zajišťovat abstrakce stránek. Konkrétní postup bude vysvětlen na testovací třídě *WorldMapTest.java*.

Zdrojový kód 5.5: Deklarace třídy *WorldMapTest.java* a jejích atributů

```
@RunWith(Arquillian.class)
public class WorldMapTest extends TestUtilityClass {

    @Page
    private WorldMap page;

    @FindBy(className = "bottom-alert")
    private AlertMessage alert;

    ...
```

Definice testovací třídy začíná anotací *@RunWith(Arquillian.class)*. Ta zajistí, aby pro testování byla využita platforma Arquillian a její rozšíření. V těle třídy jsou vidět dva atributy, a to objekt stránky představující stránku světové mapy a atribut obsahující fragment stránky reprezentující informační zprávu zobrazující se ve spodní části obrazovky.

Dále je v deklaraci třídy vidět, že rozšiřuje pomocnou třídu *TestUtilityClass.java*. Ta je společný předek všech testovacích tříd a umožňuje testům využít několik vlastností. Prvním účelem této třídy je držení informací o proměnných, které byly definovány při nastavení nástroje Maven v souboru *pom.xml*. Těmito proměnnými jsou URL aplikace, uživatelská jména, hesla a e-maily používané při přihlašování do aplikace. Jelikož je pro testování vytváření nového účtu nutné generovat vždy nový, náhodný e-mail a uživatelské jméno, obsahuje tato třída i metody pro jejich náhodné generování. Dále pomocná třída obsahuje injekci webového prohlížeče pomocí anotace *@Drone*. Každý test pracuje s webovým prohlížečem, proto injekce webového prohlížeče na tomto místě vyhovuje přístupu DRY[2]. Poslední funkcionalitou třídy *TestUtilityClass.java* je vlastní řešení čekání na dokončení načtení stránky metodou *waitUntilPageLoaded()*. Toho je docíleno čekáním na element indikující, že aplikace pracuje na zpracování požadavku a periodické dotazování se, zda tento element již není viditelný.

Následující metoda využívající anotaci *@Before* zajišťuje provedení úkonů nutných vykonat před spuštěním testovacích tříd. Ve

2. Don't Repeat Yourself

všech testovacích třídách půjde o otevření testované stránky pomocí metody *browser.get()*. Obdobným způsobem lze v testech využít metodu využívající anotaci *@After*, která se vykoná po skončení každého testu. Obě tyto anotace jsou vlastností aplikačního rámce JUnit.

Zdrojový kód 5.6: Metoda vykonávající příkazy před každým testem

```
@Before
public void getWebpage() {
    browser.get(BASE_URL + "/view/world/average");

    if (!page.getNavMenu().getActiveLanguage()
            .equals("en")) {
        page.getNavMenu().switchLanguage("en");
    }

    waitUntilPageLoaded();
}
```

Poslední částí testových tříd jsou samotné testové metody. Následující kus zdrojového kódu je dobrou ukázkou typického testu. Testová metoda je nejprve označena anotací *@Test*(také součást JUnit aplikačního rámce). V těle metody jsou deklarovány pomocné proměnné, do nichž je následně nahrána hodnota získaná pomocí metody obsažené v dříve vytvořeném fragmentu. Tato hodnota je pak vyhodnocována pomocí metody *assertTrue()* nebo *assertFalse()*. Pokud toto vyhodnocení selže, je test označen jako neúspěšný (*failure*). Pokud se objeví chyba kdykoliv během vykonávání testové metody (např. nenalezení elementu, na němž je prováděna akce), je test označen jako chybný (*error*). Struktura testových metod se samozřejmě může lišit dle vykonávaného scénáře, nicméně tento způsob vyhodnocování je základem každé testové metody.

Zdrojový kód 5.7: Část třídy testující přiblížení a oddálení mapy

```
@Test
public void testZoom() {
    String placeToTest = "Australia";

    int heightBefore = page.getWorldMap()
            .getPlaceSize(placeToTest).getHeight();
```

```
        for (int i~= 0; i~< 5; i++) {
            page.getWorldMap().zoomIn();
        }

        int heightAfter = page.getWorldMap()
                .getPlaceSize(placeToTest).getHeight();

        assertTrue(heightBefore < heightAfter);

            ...
```

6 Systém automatizovaného testování

Jednou z podmínek v zadání této práce bylo integrovat vytvořené testy do vývojového procesu aplikace. K tomu je využita webová služba Travis CI. Tato služba využívá služby GitHub, na níž jsou uloženy repozitáře jak testované aplikace, tak testů vytvořených pro tuto aplikaci.

Travis CI lze začít používat po provedení následujících kroků. Nejprve je nutné se přihlásit do služby Travis CI s GitHub účtem obsahujícím testovanou aplikaci. Poté se provede automatická synchronizace GitHub repozitářů, po jejímž dokončení je možné v profilové stránce zvolit, pro které repozitáře chceme využívat sestavení pomocí služby Travis CI. Posledním povinným krokem je nahrání souboru *.travis.yml* do kořenového adresáře repozitáře. Tento soubor obsahuje konfiguraci a dává službě vědět, jaké úkony se mají provést. V tomto projektu vypadá soubor následovně:

Zdrojový kód 6.1: Soubor .travis.yml

```
language: java

script:
  - chmod a+x scripts/tests.sh
  - ./scripts/tests.sh
```

První řádek specifikuje jazyk, ve kterém je projekt napsán. Podle tohoto výběru je také zvolen nástroj pro sestavení aplikace. Pokud Travis CI nalezne kromě souboru *.travis.yml* také soubor *pom.xml*, automaticky zvolí k sestavení aplikace nástroj Maven. Druhou informací, kterou *.travis.yml* předává, je umístění *Bash* skriptu, v němž jsou definovány příkazy zajišťující testování aplikace. Tomuto skriptu jsou nastavena práva, která zajišťují spuštění v prostředí nástroje Travis CI.

Skript byl napsán tak, aby se nespouštěl při každé *commit* akci v repozitáři testované aplikace. Spuštěn bude jen v případě, že se jedná o větší *commit* značící novou verzi aplikace. Ta je v repozitáři aplikace označena značkou (*tag*) *release-datum-vydani*. Skript proto nejprve hledá tuto značku a až poté přejde k přípravě testování. Ta probíhá tak, že se do vzdáleného počítače poskytovaného službou Travis CI nakopíruje obsah GitHub repozitáře obsahující testy. V tomto re-

pozitáři poté zavolá příkaz nástroje Maven, kterým bude testování zahájeno. O výsledcích testování jsou prostřednictvím e-mailu automaticky informovány dvě osoby: autor *commitu* a osoba, která *commit* provedla. Notifikace lze detailně konfigurovat v souboru *.travis.yml*.

Zdrojový kód 6.2: Bash skript zajišťující testování aplikace

```
#!/bin/bash
# test script for .travis.yml

set -e
COMMIT_TAG=\$(git tag --contains "\${TRAVIS_COMMIT}")

if [[ \$COMMIT_TAG =~ release.* ]]; then

git clone https://github.com/SykoraErik/BachelorThesis.git
    testRepository
cd testRepository
mvn -Pphantomjs -Dbase.url=http://staging-new.slepemapy.cz/
    clean verify;

fi
```

O výsledku sestavení aplikace a provedených testů se lze také informovat na webových stránkách Travis CI. Po přihlášení ke svému účtu lze procházet všechny v minulosti spuštěné i právě probíhající sestavení aplikace. Lze mimo jiné zjistit, zda sestavení aplikace proběhlo v pořádku, jak dlouho trvalo, kdo byl autorem *commit* a také se lze odkazem ihned dostat na GitHub stránku *commit*. Navíc je zde zobrazen i kompletní záznam průběhu sestavení aplikace, který poskytuje nástroj Maven.

7 Závěr

Cílem mé bakalářské práce bylo vytvoření sady funkčních testů pro aplikaci na výuku zeměpisu. Nejprve jsem se zabýval metodami testování webové aplikace a vymezil tak rámec, v němž se při tvorbě testů pohybujeme. Dále byly představeny dva vybrané nástroje (Selenium a Arquillian), pomocí nichž lze funkční testy pro webovou aplikaci vytvořit. Popsal jsem, jakým způsobem fungují a jaké jsou jejich výhody. Zmíněny byly také ostatní nástroje použité v práci, jako například nástroj pro sestavení aplikace Maven a webovou službu Travis CI, zajišťující průběžnou integraci. Dále jsem se v práci zabýval analýzou testované aplikace a tvorbou testové dokumentace, která slouží jako základ pro tvorbu testů. Před samotnou tvorbou testů pomocí nástroje Arquillian jsou ale představeny koncepty abstrakce webové stránky, jejichž využití je názorně ukázáno na příkladech.

Výsledkem práce jsou testové třídy testující nejdůležitější funkcionalitu aplikace Slepé Mapy. Společně s testovými třídami jsou také výstupem práce vytvořené abstrakce stránek, jejichž využití zjednoduší následnou údržbu testů a také další rozšiřování testovací sady. Testovací sada je připravena k nasazení do vývojového procesu pomocí nástroje průběžné integrace Travis CI. Po vložení souborů *.travis.yml* a *tests.sh* do repozitáře testované aplikace budou testy součástí vývojového procesu. Spouštění testů je nastaveno tak, aby se prováděly pouze při vydání nové verze aplikace.

Vytvořené řešení je připravené na další rozvoj. Během vývoje testované aplikace se počítá s tím, že provedené změny mohou ovlivnit použitelnost testů, proto bylo již od počátku myšleno na udržovatelnost testové sady i vytvořených abstrakcí webových stránek. Testování budoucí funkcionality testované aplikace lze také jednoduše realizovat přidáním nových testových tříd, případně nových abstrakcí webové stránky. Výhodou je znovupoužitelnost již vytvořených webových abstrakcí v budoucích testových třídách. Možnými vylepšeními do budoucna je rozšíření sady testů a pokrýt tak testy větší část funkcionality aplikace. Lze uvažovat také o využití nástroje Selenium Grid a umožnit tak provádět testy na několika prohlížečích současně.

Literatura

[1] PAGE, Alan – JOHNSTON, Ken – ROLLISON, Bj. *Jak testuje software Microsoft.* Vyd. 1. Brno: Computer Press, 2009, 384 s. ISBN 978-80-251-2869-5.

[2] PATTON, Ron. *Testování softwaru.* Vyd. 1. Praha: Computer Press, 2002, xiv, 313 s. Programování. ISBN 80-7226-636-5.

[3] *What is Functional testing (Testing of functions) in software?* ISTQB Exam Certification [online]. 2015 [cit. 2015-04-12].Dostupné z: `http://istqbexamcertification.com/what-is-functional-testing-testing-of-functions-in-software/`

[4] *What is Software Testing?* ISTQB Exam Certification [online]. 2015 [cit. 2015-04-12]. Dostupné z: `http://istqbexamcertification.com/what-is-a-software-testing/`

[5] *What is Unit testing?* ISTQB Exam Certification [online]. 2015 [cit. 2015-11-06]. Dostupné z: `http://istqbexamcertification.com/what-is-unit-testing/`

[6] MYERS, Glenford J, Corey SANDLER a Tom BADGETT. *The art of software testing.* 3rd ed. Hoboken, N.J.: John Wiley & Sons, c2012, xi, 240 p. ISBN 1118133145.

[7] *Selenium documentation.* [online]. 12.11.2015 [cit. 2015-11-12]. Dostupné z: `http://docs.seleniumhq.org/docs/index.jsp`

[8] *Selenium documentation* [online]. 12.11.2015 [cit. 2015-11-12]. Dostupné z: `https://seleniumhq.github.io/docs/index.html`

[9] Fowler, Martin. *Continuous integration.* Martin Fowler. [online]. 2006 [cit. 2015-11-22]. Dostupné z: `http://www.martinfowler.com/articles/continuousIntegration.html`

[10] *Static Vs Dynamic Testing.* Guru99. [online]. © 2015 [cit. 2015-11-26]. Dostupné z: `http://www.guru99.com/static-dynamic-testing.html`

[11] *Arquillian.* Arquillian. [online]. © 2009-2015 [cit. 2015-12-08]. Dostupné z: http://arquillian.org/

[12] *Slepé Mapy.* Slepé Mapy. [online]. © 2015 [cit. 2015-12-10]. Dostupné z: http://slepemapy.cz/

[13] *Creating Deployable Archives with ShrinkWrap* Arquillian Guides. [online]. © 2009-2015 [cit. 2015-12-11]. Dostupné z: http://arquillian.org/guides/shrinkwrap_introduction/

[14] *Graphene 2.* Project Documentation Editor. [online]. 5.8.2014 [cit. 2015-12-13]. Dostupné z: https://docs.jboss.org/author/display/ARQGRA2

[15] *Drone.* Project Documentation Editor. [online]. 16.6.2014 [cit. 2015-12-13]. Dostupné z: https://docs.jboss.org/author/display/ARQ/Drone

[16] *Git* Git. [online]. 2014 [cit. 2015-12-15]. Dostupné z: https://git-scm.com/

[17] *About.* JUnit. [online]. 12.3.2015 [cit. 2015-12-15]. Dostupné z: http://junit.org/

[18] *Welcome to Apache Maven* Maven [online]. 1.12.2015 [cit. 2015-12-15]. Dostupné z: https://maven.apache.org

[19] *Test and Deploy with Confidence.* Travis CI. [online]. 2015 [cit. 2015-12-15]. Dostupné z: https://travis-ci.org/

[20] *Browser Statistics.* W3Schools. [online]. 2015 [cit. 2015-12-15]. Dostupné z: http://www.w3schools.com/browsers/browsers_stats.asp

Přílohy

- text práce ve formátu PDF
- vytvořené testové třídy, objekty stránek, fragmenty stránek a konfigurační soubory
- vytvořený skript a konfigurační soubor pro nástroj Travis CI
- výstupní report nástroje Maven obsahující dobu trvání testové sady
- testová dokumentace