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

Bakalářská práce

Testování EEG/ERP portálu

Plzeň, 2013 Tomáš Pokryvka

Prohlášení

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

V Plzni dne 27. června 2013 Tomáš Pokryvka .....

Abstract

This thesis deals with the common testing theory and describes various types of tests used in testing of the portal-like applications. The main subject is the proposal and the subsequent implementation of tests for the EEG/ERP Portal, which serves for storing and managing data and metadata gained from the EEG/ERP experiments. The second part describes specific tests for EEG/ERP Portal. After decomposition to the particular components there was developed a set of unit and system tests verifying the functionality of every component. In the second phase of testing are prepared tests of usability and load tests applied on the whole EEG/ERP Portal. In conclusion, there is a list of proposed changes created according to the results obtained in tests. The benefit of this work is verifying the EEG/ERP Portal functionality and discovering the ability of managing the predicted load.

Abstrakt

Práce pojednává o obecné teorii testování a typech testů, se kterými je možné se setkat při testování portálových aplikací. V druhé části se zabává návrhem a následnou implementací testů pro EEG/ERP portál, který slouží pro ukládání a správu dat získaných z EEG/ERP experimentů. Celý portál byl nejprve dekomponován na jednotlivé funkční celky, přičemž pro každý z nich byla vytvořena sada jednotkových a systémových testů ověřující jeho funkčnost. Po otestování jednotlivých částí došlo k testování portálu jako celku za pomoci zátěžových testů a testů použitelnosti. Výsledkem práce je ověření funkčnosti portálu, návrh změn vytvořený na základě výsledků dosažených v testech a ověření, zdali je portál dimenzován na očekávanou zátěž.

Obsah

1 Úvod

Současným trendem ve vývoji aplikací je začlenění testování paralelně s programováním[3, strana 24]. Obvykle se testuje každý funkční celek na konci dané vývojové iteraci, kdy dochází k jeho ladění a začlenění do aplikačního jádra či modulu. Takový přístup umožňuje detekci odchylek oproti funkční specifikaci projektu v rané fázi [4, strana 15], tudíž odstranění této nesrovnalosti je levné jak z hlediska času, tak ostatních nákladů [1, strana 18]. Klesá rovněž procentuelní výskyt chyb v kódu a roste stabilita aplikace jako celku. Na tomto základě bylo nutné vytvořit pro EEG/ERP portál množinu testů validující existující kód během každé iterace.

První část práce (kapitola 2.1) se zabývá obecným popisem webové aplikace s názvem EEG/ERP portál, kde je vysvětlen účel vzniku a popis technologií, kterých portál pro svou činnost využívá. Na tomto základě je možné získat představu o testované aplikaci, jejím účelu a rozsahu.

V další části (kapitola 3) jsou vysvětleny základní pojmy z oblasti testování software a především testování webových aplikací. Zde je možné seznámit se s terminologií používanou v oblasti testování software, a se základními způsoby testování (kapitoly 3.1 a 3.2) . Pozornost je dále věnována typům testů, kde je v krátkosti uveden jejich popis a způsob využití v rámci testované aplikace, stejně jako nástroje, které se pro daný typ testů hodí. Díky tomu je možné získat představu typech testů, kterých bylo využito i dále pro testování samotného portálu (kapitola 4).

Následující část (kapitola 5) obsahuje seznam nástrojů použitých pro testování EEG/ERP portálu, včetně jejich zevrubného popisu a způsobu použití.

Poslední oddíl práce tvoří praktická část (kapitola 6.1), kde je portálová aplikace nejprve rozdělena na datovou, aplikační a prezentační část a pro každou z těchto částí navržen vhodný způsob testování. Další kapitoly jsou tvořeny jednotlivými typy testů, které byly pro portál použity. Jsou jimi jednotkové (kapitola 6.2), systémové (kapitola 6.4), zátěžové (kapitola 6.5) testy a testy použitelnosti (kapitola 6.6). U každého typu testu jsou společně s popisem testovacích scénářů uvedeny také dosažené výsledky.

2 EEG/ERP portál

2.1 Obecný popis

EEG/ERP Portal je webová aplikace vyvíjená v rámci neuroinformatické skupiny na Katedře informatiky a výpočetní techniky (dále KIV) Fakulty aplikovaných věd na Západočeské univerzitě v Plzni. Doposud byl portál používán výhradně v rámci této skupiny, což se postupně mění po úspěšném začlenění mezi registrované zdroje Neuroscience Information Framework (dále jen NIF), který tvoří stále rostoucí databázi relevantních neuroinformatických zdrojů, dat a nástrojů [11]. Portál slouží k ukládání a správě dat získaných z EEG/ERP experimentů provedených dle určitého, předem připraveného, scénáře, stejně jako ke správě skupin a osob spojených s výzkumem.

V současné době se funkcionalita portálu stále rozrůstá za účelem usnadnění samotného výzkumu a činností s ním spojených. Příkladem může být online rezervace laboratoří, ve kterých se prování EEG/ERP měření.

Portál je určen pro omezenou skupinu lidí, která se zabývá výzkumem v oblasti neuroinformatiky, přičemž každý z uživatelů zaujímá v rámci výzkumu určité postavení. Na tomto základě byly zavedeny takzvané role, které uživatelům přidělují určitá oprávnění. Aby bylo možné s portálem pracovat (bez ohledu na oprávnění), je nutné se nejprve přihlásit pomocí úvodního přihlašovacího formuláře. Tím je zajištěno, že data obsažená v portálu nebudou dostupná široké veřejnosti a registrovaní úživatelé budou mít odpovídající práva.

EEG/ERP portál je webová aplikace postavená na frameworku Apache Wicket[1] s podporou Spring MVC. Spring MVC je framework pro vývoj webových aplikací postavený na MVC[2] architektuře, který disponuje mnoha rozšiřujícími pluginy. V rámci portálu byly použity pluginy Spring Security řešící problematiku přihlašování a zabezpečeného posílání parametrů a Spring Social poskytující propojení aplikace se sociálními sítěmi (Facebook, LinkedIn), díky nimž je možné data buď sdílet nebo se pomocí účtů na sociální síti do portálu přihlašovat.

[1] http://wicket.apache.org/
[2] MVC (Model-View-Controller)

Veškerá data, se kterými portál pracuje, jsou ukládána do databáze Oracle. Propojení databáze s portálem je uskutečněno pomocí frameworku Hibernate a objektově-relačního mapování dat (dále jen ORM[3]), které mapuje vytvořené JAVA objekty (POJO[4]) na entity v relační databázi.

[3]ORM (Object Relational Mapping)
[4]POJO (Plain Old Java Object)

3 Testování software

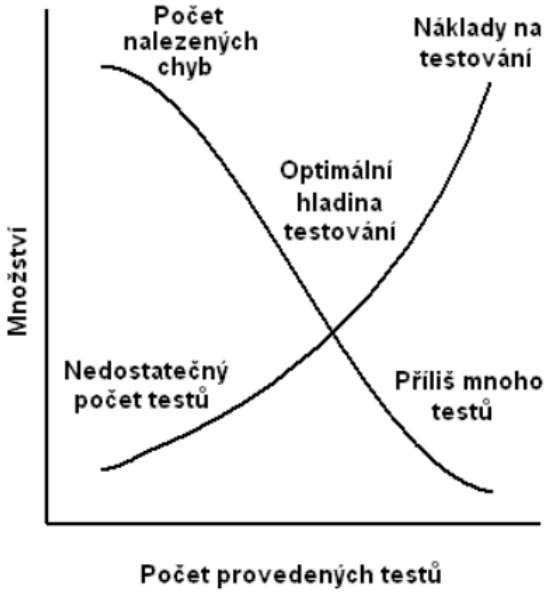
Testování software se stává nedílnou součástí vývoje aplikace. Jeho úkolem je nalezení odchylek (chyb) aplikace od specifikace, zjistit nekorektní či nesprávné chování a určit slabá místa systému pro případ vysokého zatížení. Tyto chyby je důležité odhalit a opravit v co nejkratším čase, protože čím později se chyba najde, tím je její oprava nákladnější [1]. Důvodů, proč aplikaci testovat, je hned několik. V našem případě je testován EEG/ERP portál pro správu naměřených dat. V případě zanesení vážné chyby, která není odhalena, může přijít několikaletý vývoj nazmar. Příkladem může být nežádnoucí změna hodnot při vytváření objektů z dat uložených v databázi.

Právě kvůli takovýmto problémům je snahou aplikace testovat co možná nejdůkladněji. Bohužel 100% otestování aplikací není možné, protože má příliš velké množství vstupů a výstupů. V praxi se tak určují pouze určité hranice a kvóty, které by měly být nastavené tak, aby byly odhaleny všechny závažné chyby a většina chyb méně závažných. Testování je stejně jako vývoj aplikací časově omezeno a finančně limitováno, takže definované kvóty tvoří také kompromis mezi vynaloženými prostředky a časem stráveným nad testováním aplikace. Proto je důležité určit vhodný poměr tak, aby aplikace byla dostatečně otestována a zároveň množství peněz vynaložených na testování přiměřené.

Mezi obvyklé kvóty určující rozsah testování aplikace je pokrytí kódu testy či pokrytí rozhodovacích hran. Například Microsoft má vývoj aplikací rozdělen do několika vývojových etap, přičemž v každé etapě je požadavek na pokrytí kódu větší [2, strana 68].

3.1 Testování z pohledu znalosti kódu

V závislosti na tom, jaké máme o aplikaci (produktu) znalosti a jak k ní přistupujeme, můžeme testování rozdělit na dvě, respektive tři způsoby přístupu. Navzájem jsou odlišeny množstvím informací, které o testovaném produktu máme, tj. jestli známe vnitřní uspořádání a implementační detaily nebo ne.



Obrázek 1: Vztah mezi množstvím provedených testů a financemi vynaloženými na testování [1, strana 40].

3.1.1 Testování černé skříňky

Testování černé skříňky (Black box testing) se nazývá takový typ testování, kdy tester nezná vnitřní strukturu aplikace a nemá k dispozici její zdrojové kódy. Díky tomu je skutečně možné na aplikaci nahlížet jako na černou skříňku, která má definované vstupy (zdrojová data, podněty od uživatele) a ta tyto vstupy převede na výstup, aniž by tester přesně věděl jak [5, strana 17]. Tester tak pouze porovnává získané výsledky s výsledky očekávanými. Jedná se o primitivní způsob testování, který neumožňuje důkladné otestování celé aplikace (neznáme podmínky a nemůžeme tak pokrýt všechny možné průchody programem), ale jde o velice efektivní způsob odhalení velkého množství počátečních chyb. Protože je tester odstíněn od implementačních detailů a na aplikaci nahlíží úplně jiným způsobem než vývojář, může najít chyby, kterých si vývojář nevšiml.

Do testování černé skříňky spadají téměř všechny testy uživatelského rozhraní, akceptační testy (kapitola 4.6) nebo testy použitelnosti (kapitola 4.5). Příkladem testování černé skříňky může být v pozdější fázi vývoje předložení testovací verze programu třetí straně nebo beta testování koncovým zákazníkem.

3.1.2 Testování bílé skříňky

Testování bílé skříňky (White box testing) je takový způsob testování, kdy tester zná implementační detaily aplikace (použité algoritmy, struktury, technologie) a na jejich základě vytvoří příslušné testy. Při tomto testování ztrácí tester určitý nadhled nad aplikací z pohledu uživatele, protože se více soustřeďuje na implementaci, ale naopak získá ponětí o tom, kde jsou slabá místa, co je potřeba detailněji otestovat a jak jsou nastaveny podmínky větvení. Je tak možné komplexněji otestovat celou aplikaci, včetně částí, které se metodou testování černé skříňky nemusí vůbec vykonat. Testeři mají často nastavená meřítka pro pokrytí kódu, kterých musí dosáhnout pro adekvátní otestování aplikace.

- **Pokrytí příkazů** - Nejjednoduší a zároveň nejméně přesná metrika, která kontroluje pouze to, zdali byl příkaz vykonán. Častokrát lze navrhnout testy, které se blíží s pokrytím kódu sto procentům, ale kód může obsahovat velké množství neodkrytých chyb.
- **Pokrytí hran** - Metrika o něco přesnější nežli v případě pokrytí příkazů, ovšem v mnoha případech i tak nedostatečná. Pokrytí hran sleduje, zdali byly pokryty všechny větve podmínky. Pro případ jednoduché podmínky `IF` to jsou případy, kdy je podmínka vyhodnocena jako TRUE a FALSE.
- **Pokrytí podmínek** - Pokročilejší metrika nežli pokrytí hran, kdy jsou testovány všechny kombinace vstupů. Tato metrika má význam především v případě, že testovaný kód obsahuje složené podmínky, protože dochází k vyhodnocení všech variant jednotlivých částí podmínky.
- **Pokrytí cest** - Nejpodrobnější a nejdůkladnější typ metriky, kdy jsou kontrolovány všechny vstupy a výstupy. Problémem této metriky je praktická použitelnost, protože počet testů roste exponenciálně se složitostí. [13]

Mezi typické zástupce testování bílé skříňky patří jednotkové testy (kapitola 4.1), které si buď píší vývojáři sami nebo je píší testeři na základě poskytnutého interface či kódu.

Speciálním způsobem testování aplikace postaveném na testování bílé skříňky je vývoj řízený testy (TDD[5]) [4]. Jedná se o vývoj aplikace zaměřený především na

[5]TTD (Test Driven Development)

automatické testování (kapitola 3.2.2). Myšlenka tohoto vývoje je taková, že se nejprve nadefinuje API a struktura aplikace. Poté se napíší testy, které budou validovat vyvíjený kód a jako poslední krok se píše kód, který je hotov v době, kdy projde validačními testy. Tím je zajištěno, že takto napsaný kód je otestovaný a správný.

3.1.3 Testovaní šedé skříňky

Testování šedé skříňky (Gray box testing) se nachází na pomezí testování černé a bílé skříňky (proto testování šedé skříňky) a rozsah tohoto pojmu není přesně definován. Do testů šedé skříňky lze však zařadit takové typy testů, které byly vytvořeny pro systém u něhož neznáme přesné implementační detaily, ale známe například jeho rozhodovací logiku a přibližnou strukturu. Díky tomu lze částečně uzpůsobit testy pro daný systém a otestovat ho tak důkladněji než v případě testů černé skříňky.

Zástupcem těchto těstů může být systémové testování (kapitola 4.3) systému či logických jednotek a jejich vzájemné provázanosti.

3.2 Dělení testů dle jejich provedení

3.2.1 Manuální testování

Ačkoli je dnes snaha většinu testů automatizovat (především z časového a finančního hlediska), tvoří manuální testy nedílnou součást testování aplikace. Důvodů je několik. Ne všechny testy je možné automatizovat, a ne vždy se to vyplatí (viz dále (kapitola 3.2.2)).

Hlavní uplatnění manuálního testování najdeme při testech černé skříňky, u webových aplikací to je například testování kompatibity mezi prohlížeči (kapitola 4.5), kdy je potřeba ověřit, že se aplikace ve všech podporovaných prohlížečích chová a zobrazuje korektně (stejně).

3.2.2 Automatické testování

Automatický test je takový test, který probíhá bez zásahu člověka. Jedná se tedy o program, který simuluje činnost testera a porovnává získané výsledky s výsledky očekávanými. Hlavní výhodou těchto testů je opakovatelnost, velká časová úspora při spouštění a vyhodnocování testů a vyloučení chyby testera při opakovaném provádění. Automatizovat se snažíme takové testy, které se příliš nemění a jsou často vykonávány (např. pro ověření integrity při změně některé funkční části).

Bohužel ne všechny testy se dají automatizovat. Záleží jakou část aplikace se snažíme pokrýt automatickými testy. Nejdůležitější je velká míra abstrakce jednotlivých částí zdrojového kódu a tím i jejich co možná největší nezávislost. Pokud jsou části kódu na sobě příliš závislé, není možné aplikaci dekomponovat a je nutné aplikaci testovat jako celek. To přináší značná rizika, protože aplikace nemusí být dostatečně otestována a zvyšuje se riziko přehlédnutí chyby.

Automatické testování je například nepostradatelné při zátěžových testech (kapitola 4.4), kdy v případě webové aplikace simulujeme činnost několika stovek či tisíc uživatelů najednou, což je bez automatického testování příliš nákladné a často i téměř nemožné.

3.3 Shrnutí

V rámci testování portálu je možné se setkat s různými typy testování, které umožňují komplexní ověření všech jeho částí, a které byly popsány v předchozích kapitolách. Není příhodné testovat portál pouze jako černou skříňku (kapitola 3.1.1), protože nemusí dojít k otestování celé aplikace z důvodu neznalosti implementačních detailů. Též není vhodné kontrolovat jednotlivé příkazy s využitím metod testování bílé skříňky (kapitola 3.1.2), kdy dochází ke ztrátě nadhledu nad aplikací jako celkem.

Pro portál bylo využito jak automatických (kapitola 3.2.2), tak manuálních (kapitola 3.2.1) testů. Protože byl kladen důraz především na automatizaci testů, tvořilo manuální testování menší část.

4 Typy testů

4.1 Jednotkové testy

Jednotkové testy (Unit tests) slouží k otestování malých bloků kódu, přičemž testují aplikaci na nejnižší úrovni. Pro objektově orientované jazyky to jsou jednotlivé metody, maximálně třídy. Nějvětší uplatnění najdou na začátku vývoje dané části kódu (než se začlení do většího funkčního bloku), kdy je možné ověřit, jestli metoda (třída) dělá to co má.

Tento typ testů si v jednoduché podobě často píší sami programátoři, aby si ověřili funkčnost svého kódu. Takto ověřený kód následně prochází i jednotkovými testy psaný testery, který je nezávislý na programátorovi (programátor je často vůči svému kódu "slepý"a má snahu své chyby přehlížet) a může zapadat do většího bloku testů.

Pro psaní jednotkových testů existuje několik nástrojů. Jeden z nejpoužívanějších a nejrozšířenějších je framework xUnit, kde písmeno `X` před názvem udává, pro který jazyk je framework určen [6, strana 11]. EEG/ERP portál je vyvíjen ve frameworku Wicket, který je postaven na programovacím jazyku JAVA, a proto byl použit testovací framework JUnit[6] (kapitola 5.1).

4.2 Integrační testy

Integrační testy slouží pro otestování větších funkčních celků aplikace (jedná se o spojení několika jednotek - tříd). Hlavní náplní těchto testů je ověření správné komunikace (integrity) mezi komponentami aplikace či mezi aplikací a systémem. Příkladem integračních testů může být ověření nové funkcionality do systému nebo validity vstupních dat a jejich předání nižší (vyšší) vrstvě, která má za úkol tato data dále zpracovávat.

Pro integrační testy je, stejně jako pro jednotkové testy (kapitola 4.1), využit nástroj JUnit, který umožňuje kromě tvorby jednoduchých jednotkových testů také testy

[6] http://junit.org/

integrační.

4.3 Systémové testy

Jedná se o testy, které testují aplikaci jako funkční celek. Tyto testy jsou vykonávány v poslední části vývoje, těsně před předáním produktu zákazníkovi. Simulují činnost uživatele a provádějí netriviální operace. Příkladem systémového testu může být vytvoření nového uživatele a jeho registrace do systému. Také tyto testy jsou častokrát automatizovány a často opakovány. Především u webových aplikací se často stává, že s novou funkčností dochází ke kolizi s již existující částí kódu. Díky automatizaci systémových testů je možné existující problém rychle objevit a chybu odstranit.

Příkladem systémového testu pro webové aplikace je nástroj Selenium[7], který umožňuje zaznamenávat určitou posloupnost příkazů a tu pak následně opakovat. Nástroj je navíc vybaven celou sérií porovnávacích funkcí, takže je i velice snadné kontrolovat obsah testovaných stránek. Více v kapitole 5.2.

4.4 Zátěžové testy

U webových aplikací je potřeba zjistit, kolik uživatelů může naráz s aplikací pracovat a kolik uživatelů už je kritických. K tomu slouží právě zátěžové testy, které ověřují chování aplikace při velkém vytížení a umožňují tak najít její hranice. Nejčastěji se sleduje počet požadavků, který je aplikace schopna obsloužit za určitou časovou jednotku nebo jak se aplikace chová při dlouhodobém vytížení. Při dlouhodobém vytížení, které může trvat řádově hodiny či dokonce dny se sleduje, jak aplikace pracuje s přidělenou pamětí a jak využívá procesorový čas. Tyto testy tak odhalí špatné uvolňování paměti, které by se jinak odhalovalo jen velice těžko.

Protože vytvářet testy, které simulují činnost několika set nebo dokonce tisíci uživatelů je značně náročné, využívá se k tomu speciální software. Tomu je na vstup přivedena množina testů simulující činnost uživatelů a tento program je paralelně

[7] http://docs.seleniumhq.org/

spouští. Zároveň sleduje i chování testovaného systému, jeho odezvy na poslané požadavky, množství poslaných dotazů či velikost přenášených dat. Výsledkem je pak sumární report obsahující veškerá tato data, případně jejich zobrazení v grafech.

Pro tento typ testů je často používán nekomerční nástroj JMeter[8] , jež podkytuje široké možnosti nastavení, a který je podrobněji popsán v kapitole 5.4.

4.5 Testy použitelnosti

V současné době je kladen velký důraz na grafickou prezentaci a interaktivnost webových stránek, kdy je výsledná webová aplikace velice podobná aplikaci desktopové. Díky tomu vzniklo nové odvětví testování nazývané testování použitelnosti. To má za úkol určit, jak moc jsou internetové stránky uživatelsky přívětivé, tj. vyhodnocuje se, jak jsou stránky intuitivní (Nemusí uživatel informace na stránkách příliš hledat? Ví uživatel, na co má kliknout?), konzistentní (Zobrazuje se webová stránka všude stejně? Jsou použity všude stejné konvence?), správné (Nejsou na stránkách neplatné odkazy?). Aplikace, která není intuitivní, obsahuje nefunkční odkazy nebo se nesprávně zobrazuje, způsobuje, že ztrácí na důvěryhodnosti a uživatelé s ní budou neradi pracovat. V případě komerční aplikace, která byť svou funcionalitou několikanásobně překonává konkurenci, může být tento problém dokonce fatální a stane se neúspěšnou.

Mezi testy použitelnosti patří také testování napříč webovými prohlížeči. Bohužel ne všechny prohlížeče zobrazují stejný kód stejným způsobem, a tak dochází k tomu, že se stejná stránka zobrazuje v různých prohlížečích trochu jinak (často u stejného prohlížeče v různých verzích). Tomuto je potřeba se vyvarovat a kód aplikace optimalizovat alespoň pro prohlížeče s největším zastoupením na trhu.

Některé testy použitelnosti jsou dosti specifické (zvláště testy pro ověření intuitivnosti aplikace) a často ho neprovádějí SW testeři (ti pouze píší testovací scénáře a vyhodnocují získané výsledky), ale množina vhodně vybraných uživatelů, kteří vykonávají předem určené úkony. Chování uživatelů je zaznamenáno, vyhodnoceno a na základě výsledů je možné aplikaci optimalizovat. Takovýmto způsobem dochází

[8] http://jmeter.apache.org/

především k optimalizaci intuitivnosti aplikace.

4.6 Akceptační testy

V koncové části vývoje, kdy je již webová aplikace schopná provozu (ovšem ještě ne ostrého), dochází k testování u zákazníka. Zákazník si tak ověří funkčnost produktu pomocí předem připravených scénářů, které simulují běžnou činnost uživatelů. V této chvíli by měli být nalezeny a opraveny všechny závažné chyby a většina chyb menších. Neočekává se, že aplikace bude bez chyb, a tak dochází k reportování zákazníkem nalezených chyb do sjednaného bugtrackeru[9] a jejich následné opravě.

Fáze tohoto testování může trvat od několika dní po několik měsíců v závislosti na rozsahu aplikace a požadavku garantované úrovně chybovosti. Úrovní chybovosti je myšleno počet odhalených chyb za časový úsek.

4.7 Shrnutí

V kapitole 4 se bylo možné seznámit se všemi základními typy testů, které se běžně pro webové aplikace používají, a zároveň byly použity pro testování EEG/ERP portálu. U všech dříve uvedených typů testů je možné najít jejich zevrubný popis, část aplikace, kde se s nimi můžeme setkat, případně jaké nástroje lze pro jejich vytvoření a spuštění využít.

[9]Systém pro zaznamenávání chyb a správu nalezených chyb. Příkladem takovýchto systémů jsou aplikace Bugzilla (www.bugzilla.org) či Mantis bug tracker (MantisBT.org).

5 Testovací nástroje

5.1 JUnit

Testovací nástroj JUnit je jedním z nejpoužívanějších open-source frameworků určených pro psaní jednotkových testů v programovacím jazyku Java. U jeho zrodu stáli dva softwaroví vývojáři Kent Beck a Erich Gamma, kteří vymysleli základní koncept. Aktuální verze frameworku je 4.x, která postupně nahrazuje předchozí verzi 3.x. Hlavní předností nové verze je podpora Java verze 5 a vyšší a podpora anotací, které značně zpřehledňují kód a usnadňují vývoj.

Základní ideou je porovnávání získaného výsledku s očekávaným výsledkem. Framework obsahuje porovnávací funkce typu `assert` pro všechny základní datové typy včetně polí. Pokud je test neúspěšný, zobrazí se chybové hlášení s lokalizací chyby a výpisem očekávaného a získaného výsledku.

Testy je možné spouštět buď pomocí poskytovaného GUI[10] napsaného též v jazyce Java nebo jako běžné testy v pokročilých vývojových nástrojích. Příkladem takovýchto nástrojů může být vývojové prostředí (Eclipse, Netbeans), nástroje pro překlad aplikace (ANT) či automatickou tvorbu buildů (Maven).

5.2 Selenium

Selenium je sada testovacích nástrojů sloužících k tvorbě automatických testů pro webové aplikace. Hlavní uplatnění má především pro tvorbu systémových testů, kdy se aplikace testuje jako celek, případně jsou testovány větší funkční celky. Příkladem může být testování administrátorské části webového portálu. V současné době (leden 2013) jsou podporovány webové aplikace vytvořené v programovacích jazycích Java, Javascript, Ruby, PHP, Python, Perl a C#.

Ačkoli je nástroj určen především pro tvorbu systémových testů, využívá po exportu frameworku JUnit, který dále doplňuje o své specifické třídy a rozhraní.

[10] GUI (Graphics User Interface) neboli grafické uživatelské rozhraní, které umožňuje práci s programem pomocí grafických prvků, jakými jsou například tlačítka.

Těmito třídami jsou hlavně ovladače (drivery) pro spuštění testů v různých webových prohlížečích jakými jsou například Internet Explorer nebo Opera.

Selenium IDE je plugin vytvořený pro webový prohlížeč Mozila Firefox (dále jen Firefox). Plugin poskytuje přehledné GUI pro tvorbu a správu testovacích případů. Ty lze vytvořit několika způsoby.

Prvním ze způsobů je zapnout mód pro zachytávání údálostí nad webovým prohlížečem Firefox a ty pak automaticky převádět na příkazy nástroje Selenium. Současně s tím jsou automaticky zaznamenávány do jednoduchého textového souboru ve formátu HTML. Pro každý testovací případ je vytvořen nový HTML soubor, což umožňuje velice přehlednou správu všech případů. Seznam všech testovacích případů a pořadí, v jakém se mají spouštět, jsou též uloženy v jednom speciálním HTML souboru, který má obdobu `TestSuite` v nástroji JUnit. Všechny testové případy je možné exportovat do Java kódu ve formě JUnit 3, JUnit 4 a TestNG testů (Platí pouze pro jazyk Java. Pro zbylé podporované jazyky je možný export v odpovídajícím formátu.).

Druhým ze způsobů je zadávat příkazy ručně pomocí připravených dialogů, kde se vyskytuje seznam všech dostupných příkazů a parametry příkazu. Tento způsob je sice obtížnější a vyžaduje hlubší znalosti nástroje Selenium, ale poskytuje větší možnosti pro konfiguraci a výběr prvků na webové stránce. Nástroj umí pracovat s XPath, takže je díky nim možné například pracovat se specifickým řádkem v tabulce.

V praxi se používá kombinace obou výše zmíněných metod. Nejčastější použití je zaznamenání testovacího případu pomocí prohlížeče a následné manuální upravení vygenerovaných příkazů, případně jejich doplnění. Poslední možnou variantou psaní testů pro Selenium IDE je jejich tvorba přímo v HTML kódu. Soubor s testem lze pak pomocí v pluginu otevřít a spustit. Tento způsob je sice možný, ale značně nepraktický a zde je uveden pouze pro doplnění.

5.3 BadBoy

Program Badboy[11] je testovací nástroj pro zaznamenávání činnosti uživatele pomocí integrovaného webového prohlížeče. Ten odchytává všechny akce, které uživatel provedl a zaznamenává je do své interní struktury, kterou lze dále upravovat a zpracovávat. Činností je tak značně podobá programu Selenium IDE (kapitola 5.2). Na rozdíl od Selenia IDE však disponuje funkcemi pro vytváření cyklů a spouštění vytvořených scénářů v několika vláknech, které lze dále exportovat do formátu JMX[12] a vytvářet tak zátěžové testy.

Nástroj disponuje diagnostickými funkcemi pro záznamenávání časů potřebných k načtení obsahu webové stránky a přehledné zobrazení zaznamenaných hodnot do grafů. Pro testovací scénář jsou údaje o době načítání a jejich zobrazení do grafu vytvořeny pro každou z načítaných stránek zvlášť, což umožňuje velice detailní vyhodnocování chování stránky v čase. Navíc je možné zobrazit hodnoty jednotlivých stránek pro celý testovací scénář v jednom grafu a sledovat tak webovou aplikaci jako celek. Je tak možné odhalit kumulativní chybu, která se projevuje stále pomalejším načítáním celé stránky.

5.4 JMeter

JMeter je open-source nástroj napsaný v programovacím jazyku Java a vyvinutý firmou Apache pro testování webových aplikací. Program je zaměřen především na tvorbu a spouštění zátěžových a stresových testů. Ty umožňují otestovat jak samotnou webovou aplikaci, tak i částečně hardware, na kterém webová aplikace běží. Dále disponuje přehledným GUI, kde lze testy jak vytvářet a také spuštět. Poskytuje však i možnost spustit všechny testy bez GUI a nezatěžovat tak zařízení, na kterém testy běží. To je velice užitečné v době, kdy na jednom zařízení běží jak samotné testy, tak i testovaná aplikace.

Všechny testovací scénáře lze v testu rozdělit do samostatných bloků, které běží nezávisle na sobě a navíc je lze spustit v definovaném počtu vláken, které před-

[11] http://www.badboy.com.au/
[12] JMeter File

stavují jednotlivé uživatele. Omezení se vyskytuje pouze z hardwarového hlediska testovacího zařízení, kdy na jednom zařízení není možné pustit více jak několik desítek až stovek (v závislosti na výkonu zařízení) testovacích vláken naráz. Pokud je potřeba vytvořit zátěž větší, je možné spustit testy v režimu Remote Control, kdy jsou jedny a tytéž testy spuštěny na několika zařízení najednou a vytvářejí tak potřebnou výslednou zátěž.

Krom sledování výkonostních údajů typu délka odezvy na požadavek či počtu zpracovaných požadavků za jednotku času, disponuje JMeter i funkcemi pro sledování vytížení webové aplikace během testů. To je však možné pouze v případě, že JMeter je spuštěn na stejném zařízení jako webová aplikace a všechny dotazy jsou směrovány přes Proxy server.

Testovací scénáře je možné doplnit o několik typů monitoringu, které zobrazí naměřené a dopočítané (průměrné) hodnoty ve formě tabulek nebo grafů, jež je možné vyexportovat.

5.5 IETester

Při testování použitelnosti je potřeba otestovat, zdali se obsah webové aplikace zobrazuje stejně v různých verzích daného prohlížeče. Pro Internet Explorer k tomuto účelu slouží aplikace IETester[13], která se vyznačuje intuitivností, jednoduchostí a správností interpretace zobrazení prohlížeče IE. Program využívá jádra prohlížeče Internet Explorer nainstalovaného v systému (díky tomu je program lokalizován pouze na MS Windows) a za jeho pomoci zobrazuje obsah požadovaných webových stránek ve svém interním prohlížeči. Zobrazení stránek respektuje všechny vlastnosti a pravidla zvolené verze prohlížeče internet explorer.

IETester podporuje všechny prohlížeče vydané od dnes již historické verze IE 5.5 do aktuálně nainstalované verze. To znamená, že pokud je potřeba testovat zobrazení v pohlížeči IE 10, je nutné mít v počítači nainstalovánu verzi IE 10.

[13] http://www.debugbar.com/

5.6 Xenu's Link Sleuth

Xenu je freeware testovací nástroj pro automatickou kontrolu odkazů na webové stránce. Disponuje jednoduchým ovládáním a vysokou rychlostí kontroly s podporou až sta testovacích vláken. Výsledkem testu je přehledná tabulka obsahující stav testované položky, tj. zdali je odkaz funkční, včetně úrovně zanoření. Podporuje též možnost využití autentizace, kdy je při nalezení přihlašovacího formuláře zobrazeno dialogové okno s možností zadání přihlašovacích údajů[14]. Výsledky testů je možné exportovat do CSV souboru a dále zpracovávat.

5.7 Fast Link Checker v 2.1

Jedná se o komerční nástroj umožňující kontrolu odkazů na webových stránkých s podporou přihlášení a disponujícím velice přehledným a intuitivním ovládáním. Nástroj byl zvolen právě pro svou podporu přihlášení, protože na rozdíl od Xenu (kapitola xenu) funguje přihlášení i v rámci EEG/ERP portálu. Výslekem aplikace je přehledná tabulka obsahující kontrolovanou stránky a stavový či chybový HTML kód, který je vrácen. Celý seznam prověřených odkazů je možné exportovat do formátu HTML nebo CSV, přiřemž je možné využít i některý z podporovaných filtrů, které do výsledného souboru zapíší pouze požadovaný typ záznamu.

Pro testování portálu byla využita TRIAL verze podporující kontrolu maximálně 2500 odkazů[15].

5.8 Shrnutí

Zvolené testovací nástroje byly vybírány tak, aby poskytovaly pohodlné a intuitivní ovládání a zároveň umožňovaly účinně otestovat danou část portálu.

Pro jednotkové testy byl zvolen nástroj JUnit, který tvoří jeden ze základních pilířů testování aplikací napsaných v jazyce JAVA. Má vysokou podporu integrace ve vývo-

[14] V rámci EEG/ERP portálu nebylo formulářové okno rozpoznáno a tudíž bylo nutné využít pro otestování té části portálu, vyžadující přihlášení, jiného nástroje.

[15] Pro kontrolu všech nalezených odkazů je nutné zakoupit plnou verzi programu.

jových prostředí a zároveň mnoha jiných testovacích nástrojů, které umožňují export testů do JUnit. Příkladem takového programu je Selenium. Zároveň disponuje velikou univerzálností, kdy lze pomocí JUnit testů vytvořit jak jednotkové, tak i integrační či systémové testy.

Selenuim IDE bylo zvoleno pro tvorbu systémových testů, a pro svou jednoduchost a variabilitu. Vytvořené testy je možné exportovat do pěti různých formátů java testů. Přesněji pro JUnit 4 Web Driver, které testují chování pro nastavení prohlížeč, JUnit 3 a 4 Remote Control a TestNG, který tvoří alternativu k JUnit.

Pro výkonové testování byly zvoleny programy JMeter a BadBoy. JMeter slouží pro tvorbu a spouštění zátěžových testů, které lze spouštět buď z jednoho počítače nebo distribuovaně, což poskytuje možnost dalšího využití pro budoucí testování a vývoj portálu. Program BadBoy byl zvolen pro svou možnost vytvořit a exportovat test do formátu JMF, tedy do formátu pro JMeter. Díky tomu je možné jednoduše vytvořit kostru testu v programu BadBoy, a tu následně optimalizovat v nástroji JMeter.

Pro testování použitelnosti portálu bylo využito nástrojů IETester, Xenu's Link Sleuth a Fast Link Checker. IETester pro snadnou kontrolu podpory různých verzí prohlížeče Internet Explorer. Xenu pro kontrolu odkazů v části před přihlášením se do portálu, protože disponuje přehlednými reporty a vysokou rychlostí kontroly. Navíc podrobně kontroluje CSS soubory a odhaluje neplatné cesty na soubory. Poslední nástroj Fast Link Checker byl vybrán pro svou podporu přihlášení v EEG/ERP portálu a tudíž kontrolu odkazů v části po přihlášení.

6 Implementační část

6.1 Testování portálu

6.1.1 Datová vrstva

Datová vrstva portálu je tvořena POJO objekty, které jsou využit pro objektově-relační mapování dat na databázi a DAO objekty, které obsahují metody pro práci s daty. Příkladem takovýchto metod může být vytvoření nové osoby či výzkumné skupiny.

Při testování je nejdůležitějším úkolem ověřit korektní mapování objektů na databázi a zkontrolovat správnou činnost datových objektů. Portál je aplikace, která je vyvíjena několik let a datová vrstva je již praxí ověřena. Proto testování mapování dat na databázi má spíše charakter validace pro budoucí vývoj, kdy je potřeba ověřit, že úpravou datové vrstvy nedošlo k zanesení chyby do již existujícího mapování a vše pracuje korektně.

6.1.2 Aplikační vrstva

Veškerou logiku aplikace obstarává komponentový nástroj Apache wicket. S jeho pomocí se generuje obsah pro prezentační vrstvu. Apache wicket tvoří celý Framework a zahrnuje v sobě nástroje pro ladění a testování vyvíjené aplikace. Hlavním z testovací třídou je `org.apache.wicket.util.tester.WicketTester` [10], která poskytuje porovnávací funkce pro testování parametrů stánky a komponent. Ve spojení s JUnit je tak možné otestovat celou aplikační logiku pomocí jednotkových testů. Nevýhodou testů postavených na WicketTester je jejich kódová rozsáhlost a pro aplikace velikosti portálu těžkopádnost. Na [12] je uveden článek pojednávající o vytvoření jednoduché stránky využívající wicket a její testování. Za pomoci jednotkových testů je možné simulovat chování uživatele a sledovat a vyhodnocovat činnost systému. Nutností ovšem je implementovat všechny požadované akce, které se mají vykonat, což může být pro formulář, obsahující například pět polí, několik desítek činností. Proto byla aplikační vrstva testována pomocí systémových testů za

pomoci nástrojů Selenium (kapitola 5.2) a BadBoy (kapitola 5.3).

Za pomoci výše zmíněných nástrojů a znalosti vnitřní logiky aplikace je možné portál testovat dostatečně důkladně. Při využití nástroje Code Coverage[16] je možné odhalit kód, který není pokryt systémovými testy a případně ho otestovat testy jednotkovými.

Velký důraz je při testování kladen na správnou validaci formulářových dat, kdy je nutné zajistit, že systém nebude pracovat s nesprávnými vstupními daty. Proto je potřeba každý formulář ověřit množinou testů kontrolující jeho správnou činnost při validaci údajů. Dalším krokem jsou systémové testy, které mají za úkol ověřit správnou součinnost jednotlivých částí postálu a zároveň jejich funkčnost. Základní části portálu lze určit pomocí hlavního menu, kdy každá z položek představuje jeden velký funkční blok. Ten je pak dále rozdělen dle menu druhé úrovně a jako poslední na jednotlivé části v rámci webové stránky. Příkladem posledního dělení může být ověření validace formuláře a následně jeho úspěšné odeslání.

6.1.3 Prezentační vrstva

Prezentační vrstvu tvoří grafický výstup aplikace společně s obsahem stránek. Při testování této vrstvy má uplatnění především manuální testování, kdy je potřeba otestovat korektní zobrazení aplikace ve všech aktuálně nejpoužívanějších prohlížečích či funkčnost odkazů na stránce. Nedílnou součástí je také kontrola obsahu stránek správnosti nadpisů v různých kategoriích či zobrazování odkazů pro správu portálový ch dat (odkazy a tlačítka pro mazání dat) pouze oprávněným uživatelům.

Pomocí automatických testů je možné u formulářů kontrolovat činnost AJAXu[17] a správnost chybových hlášení, tj. zdali se k nesprávným vstupním datů zobrazuje adekvátní chybová zpráva informující o daném problému. Validní odkazy na stráce lze ověřit některým z automatických robotů, kteří ze stránek získávají všechny dostupné odkazy a postupně je procházejí a vyhodnocují.

[16] Nástroj po sledování pokrytí testovaného kódu

[17] AJAX (Asynchronous JavaScript and XML) je technologie postavená na JavaScriptu sloužící pro tvorbu interaktivních webových aplikací. Hlavní výhodou je změna obsahu stránky bez nutnosti jejího opětovného načtení.

6.2 Jednotkové testy

Jednotkové testy byly v rámci portálu použity pro testování datové vrstvy. Zde bylo kontrolováno správné mapování objektů na databázi pomocí ORM a funkce DAO.

K implementaci testů byla využita rozhraní DAO objektů obsahující definice metod, které jsou v datové vrstvě používány. Všechna rozhraní mají jednoho společného potomka. Tímto potomkem je rozhraní `GenericDao.java`, které definuje všechny metody potřebné pro vytvoření a základní správu všech parametrů (hodnot). Jednotlivá rozhraní DAO objektů jsou dále doplněna o metody pro správu relací mezi objekty a případně dalšími specifickými metodami pro danou třídu. Testovány byly především uvedené typy metod.

- Vytvoření objektu
- Vznik relace
- Ověření možnosti změny dat
- Další specifické metody

Pokud test vyžadoval vazbu na další objekty (například na výzkumnou skupinu), byly využity objekty již v databázi existující. Obecně není dobré spoléhat se na existující data, protože není zaručeno, že je někdo nesmaže a testy neprojdou jen kvůli chybějícím datům. Ovšem v tomto případě se jedná o dobrou volbu, protože objekty uvedené v následujícím seznamu není možné pomocí GUI portálu smazat (data jsou navíc chráněna integritními omezeními na úrovni databáze). Krom toho, je možné třídu otestovat izolovaně, tj. není nutné vytvářet stále nové objekty. Proto vytvořenými testy kontrolujeme pouze kód pro který je určen a nezahrnujeme do nich i kód nutný k vytvoření pomocných objektů.

Všechny jednotkové testy byly vytvářeny oděděním od již existující třídy třídy `AbstractDataAccessTest`, která obsahuje odkaz a parsery na XML soubor s testovacím kontextem (`classpath:/test-context.xml`).

6.3 Testovací scénáře

Testovací scénář pro kontrolu vytvoření daného objektu, měl pro všechny třídy stejnou kostru. Základem bylo vytvořit nový objekt a po jeho přidání do databáze sledovat změnu počtu záznamů v databázi. Podrobnější popis je uveden v následujícím seznamu.

- Importování závislostí pro vytvoření objektu (DAO)
- Vytvoření nového objektu (POJO)
- Kontrola počtu záznamů v databázi
- Uložení nového objektu do databáze
- Kontrola počtu záznamů v databázi (počet musí být o 1 větší)
- Odstranění nového záznamu z databáze

Odstranění záznamu z databáze proběhlo automaticky díky anotaci `@Transactional`, která po dokončení testu provedla Rollback databáze. Pro ověření vzniku vazeb byl vytvořen velice podobný scénář, který se skládá z kroků uvedených v seznamu.

- Importování závislostí pro vytvoření objektu (DAO)
- Vytvoření nového objektu (POJO)
- Importování existujících záznamů z databáze (např. výzkumné skupiny)
- Kontrola existujících relací
- Vytvoření nové relace
- Kontrola počtu relací (počet musí být o 1 větší)
- Odstranění záznamu a nové relace z databáze

6.3.1 Výsledky

Během testování datové vrstvy, kde bylo využito jednotkových testů k ověření činnosti ORM a správné činnosti DAO, nebyly nalezeny žádné chyby. Vytvořené automatické testy mají tedy především validační význam pro budoucí vývoj portálu, kdy

bude možné snadno ověřit funkčnost datové vrstvy. Celkem bylo vytvořeno 39 testů datové vrstvy, přičemž byl vynechán test pro vytvoření nového experimentu. Důvodem byla nutnost vytvořit velký kontext, pro splnění všech integritních omezení databáze. To by mělo za následek otestování velkého množství dalšího kódu, který do testu nepatří, a který může ukrýt existující chyby. Proto byly otestovány všechny související objekty samostatně pomocí jednotkových testů a test experimentu byl přesunut mezi systémové testy.

6.4 Systémové testy

Pro otestování větších funkčních celků, jakým může být například přihlašovací formulář na hlavní straně portálu slouží systémové testy. Ty jsou důležité pro ověření funkčnosti daných celků a zároveň pro ověření kompatibility mezi komponentami, kdy mohou zastupovat integrační testy. Pro tvorbu automatických systémových testů existuje několik komerčních i nekomerčních nástrojů. Protože je EEG/ERP portál vyvíjen na akademické půdě, dává se při vývoji přednost nekomerčnímu softwaru. Jeden z nejpoužívanějších nástrojů je program Selenium vycházející pod licencí Apache 2.0 License.

Před použitím nástroje Selenium IDE (kapitola 5.2) bylo nejprve nutné provést jeho speciální konfiguraci, která ošetřovala měnící se ID[18] prvků nacházejících se v potrálu v důsledku použití frameworku Apache Wicket. Ten totiž způsobuje změnu ID při každém načtení stránky, což komplikovalo tvorbu automatických testů. Selenium funguje na principu zaznamenávání či volání událostí nad prvkem se specifickým ID, které se s časem nemění. Pokud je ale toto ID změněno, zahlásí Selenium pokus o volání události nad neexistujícím prvkem a dojde k chybě. Zmíněnou úpravou konfigurace dojde k zaznamenávání ID prvků pomocí takzvané `WicketPath`, díky které je zaznamenávána pevná část ID a číselná část, která se s časem mění, není brána v potaz. Na tomto základě je pak možné k prvku přistupovat kdykoli během testu. Aby vše fungovalo tak jak má, je ještě nutné v portálu povolit použití `WicketPath`. To se provede přidáním následujícího řádku do aplikační třídy, tj. do třídy `EEGDataBaseApplication.java`.

[18] Unikátní identifikátor

```
getDebugSettings().setOutputComponentPath(true);
```

Výpis 1: Příkaz po povolení WicketPath.

Konfigurace Selenia byla provedena tak, že v souborovém systému na pevném disku byla vytvořena složka `wicketPathLocatorBuilder` a v ní textový soubor `user-extension.js.wicketPathLocatorBuilder`, který obsahoval následující kód.

```
LocatorBuilders.add('wicketpath', function(e) {
  this.log.debug("wicketpath: e=" + e);
  if (e.attributes && e.hasAttribute("wicketpath")) {
    this.log.info("found attribute " +
      e.getAttribute("wicketpath"));
    return "//" +
      this.xpathHtmlElement(e.nodeName.toLowerCase()) +
      "[@wicketpath=" +
      this.attributeValue(e.getAttribute("wicketpath")) + "]";
  }
  return null;
});
LocatorBuilders.order.unshift(LocatorBuilders.order.pop());
```

Výpis 2: Obsah konfiguračního souboru pro Selenium IDE.

Po importu konfiguračního souboru do Selenia IDE a jeho restartování je možné začít vytvářet a spouštět systémové testy [8].

Veškeré testy, které byly v rámci systémového testování vytvořeny, se rozdělovaly to dvou skupin.

- Testy splněním
- Testy selháním

Testy splněním lze označit takové testy, u nichž musí být provedená akce úspěšně splněna. Příkladem může být úspěšné přihlášení do systému či vytvoření nového

uživatele. Naopak testy selháním jsou vyhodnoceny jako úspěšné pouze v případě, že se nějaká akce nepovede. Příkladem je neúspěšné přihlášení uživatele po zadání nesprávného přihlašovacího jména a hesla. Takto vytvořené testovací scénáře jsou pro přehlednost označeny jako `NS`[19] na konci testovacího souboru.

Pro testování byly vytvořeny čtyři emailové adresy a s nimi čtyři uživatelské účty, kde každý z účtů měl jinou uživatelskou roli. Díky nim bylo možné vytvářet typově stejné testy pro všechny uživatelské role a odhalit tak chyby v definovaných oprávnění. Tabulka 1 ukazuje seznam vytvořených emailových adres, potažmo uživatelských účtů.

Email	**Uživatelská role**
testaccountforeeg@seznam.cz	ROLE_ADMIN
testaccountforeeg2@seznam.cz	ROLE_USER
testaccountforeeg3@seznam.cz	ROLE_EXPERIMENTER
testaccountforeeg4@seznam.cz	ROLE_READER

Tabulka 1: Testovací uživatelské účty

Před samotným testováním byla celá aplikace rozdělena na malé funkční celky tak, aby bylo možné integrační a částěčně jednotkové testování spojit s testy systémovými. Tím došlo ke značné úspoře testů, které by se z větší části překrývaly a duplicitně kontrolovaly stejný kód. Většína funčních celků, které obsahovaly formuláře, byly pokryty testy splněním i selháním, aby se ověřila validace vstupních dat. Příkladem takovéto části byl přihlašovací formulář. Více o testování přihlašovacího formuláře v kapitole 6.4.1.

Při testování byla kontrolována samotná funkcionalita, tj. zdali se provedlo co se provést mělo, a činnost AJAXu, který se stará o vypisování chybových hlášení, částečnou validaci obsahu formulářů a zobrazování některých dialogových oken.

EEG/ERP portál je aplikace navržená tak, aby její prioritní funkcí bylo ukládání a spravování dat. Protože většina dat, se kterými portál pracuje, je potřeba uchovávat po celou dobu vývoje v oblasti EEG experimentů, nebylo vytvořeno pro některé položky ani GUI umožňující mazání záznamů. Tento fakt měl však za následek problematické vytváření automatických testů, které by testovaly vytváření nových

[19] Zkratka z anglického Not Successful.

uživatelů či výzkumných skupin. Proto je nutné vytvořené testy spouštět na speciální testovací databázi, která bude po ukončení testů smazána či obnovena. Tím dojde k odstranění nově vzniklých objektů a relací a bude je tedy možné vytvořit znovu.

- Vytváření nových uživatelů, kdy je potřeba opsat bezpečnostní CAPTCHU[20] a následně otevřít odkaz zaslaný na emailovou adresu nového uživatele
- Vytvoření nové výzkumné skupiny, protože skupinu nelze smazat
- Vytvoření nového testovacího scénáře, protože scénář nelze smazat
- Vytvoření nového experimentu, protože experiment nelze smazat

V době, kdy vznikaly testy, procházel portál rozsáhlými změnami a úpravami. U některých částí nebyla specifikována konečná podoba, a tak byly tyto stránky z testování vynechány. Protože nebyla dokončena specifikace, nebylo možné vytvořit ani testy založené na principu TDD neboli vývoj řízený testy, kdy jsou nejprve napsány testy a až poté samotný kód. [4]. V následujícím seznamu jsou uvedeny všechny části vynechané z testování.

- Vyhledávání v rámci celého portálu
- Vyhledávání experimentů
- Vyhledávání scénářů
- Vyhledávání osob
- Zobrazení scénářů jejichž vlastníkem je aktuálně přihlášený uživatel
- Rezervace laboratoří pro EEG/ERP experimenty
- Historie provedených změn
- Nákupní košík a nakupování experimentů

[20]CAPTCHA (completely automated public Turing test to tell computers and humans apart) je test pro odlišení počítače a lidí. Slouží především jako bezpečnostní prvek proti elektronickým robotům. CAPTCHA má podobu obrázku, obsahující deformovaný text, který musí uživatel opsat do příslušného formulářového pole

6.4.1 Testovací scénáře

Pro otestování přihlašovacího formuláře do portálu bylo vytvořeno následujících pět systémových testů, které pokrývají všechny varianty přihlášení pomocí uživatelského jména (emailové adresy) a hesla.

1. Přihlášení se do portálu jako uživatel s oprávněním ADMIN
2. Přihlášení se do portálu jako uživatel s oprávněním USER
3. Přihlášení se do portálu jako uživatel s oprávněním EXPERIMENTER
4. Přihlášení se do portálu jako uživatel s oprávněním READER
5. Neúspěšný pokus o přihlášení se do portálu

Testovací scénář pro přihlášení je pro první čtyři testy totožný, liší se pouze vstupní hodnoty dle tabulky 1. Samotný scénář se skládá z následujících kroků:

1. Načtení hlavní stránky portálu s dialogem pro přihlášení a kontrola, jestli je uživatel odhlášen
2. Zadání uživatelského jména
3. Zadání uživatelského hesla
4. Přihlášení se do portálu a kontrola úspěšnosti
5. Odhlášení se z portálu

Poslední z testů je výrazně delší, protože se jedná o test selháním a obsahuje všechny varianty neúspěšného pokusu o přihlášení. Jednotlivé testovací případy nebyly separovány do samostatných souborů z důvodu přehlednosti. Obsah celého scénáře je následující (blok přípazů je brán jako jeden bod v seznamu):

1. Načtení hlavní stránky portálu s dialogem pro přihlášení a kontrola, jestli je uživatel odhlášen
2. Pokus o přihlášení pro případ, kdy není vyplněné uživatelské jméno a ani heslo a následná kontrola chybových hlášení

3. Pokus o přihlášení pro případ, kdy je vyplněné heslo, ale ne uživatelské jméno a následná kontrola chybových hlášení
4. Pokus o přihlášení pro případ, kdy je vyplněné uživatelské jméno, ale ne heslo a následná kontrola chybových hlášení
5. Pokus o přihlášení pro případ, kdy je vyplněné správné uživatelské jméno, ale chybné heslo a následná kontrola chybových hlášení

Takovýmto způsobem byly koncipovány všechny zbylé testy. Do skupiny testování formuláře lze zahrnout testy uvedené v následujícím seznamu.

- Změna hesla přihlášeného uživatele
- Pokus o registraci nového uživatele na přihlašovací stránce (test selháním)
- Pokus o registraci nového uživatele v portálu (test selháním)
- Pokus o vyresetování starého hesla a zaslání nového (test selháním)
- Změna uživatelské role
- Pokus o vytvoření nové uživatelské skupiny (test selháním)
- Vytvoření nového parametru pro hardware
- Vytvoření nového parametru pro testovanou osobu
- Vytvoření nového parametru pro metadata
- Vytvoření nového parametru pro počasí
- Vytvoření nového parametru typu artefakt

Pro všechny z výše uvedených byly vytvořeny testy pro všechny uživatelské role, které k tomu mají oprávnění. V případě, že uživatel příslušná oprávnění nemá, je provedena kontrola že je skutečně nemůže provést. U testů, u kterých je uvedena poznámka `test selháním` se vyskytují automatizované testy pouze tohoto typu. U všech ostatních existují testy splněním i selháním. U testů splněním je však vytvořen nový objekt, který je nutné následně ihned smazat a scénář tak končí smazáním vytvořeného objektu. Pokud by test proběhl korektně, ale data ve skutečnosti smazána nebyla, bude opětovné spuštění testu vyhodnoceno jako `FAIL`[21],

[21] Jako FAIL jsou vyhodnoceny testy, které neprošly. Ty co jsou splněny jsou označeny jako PASS.

protože se v systému vyskytnou dvě identické položky. To je však nepřípustné a tudíž validované jak v aplikační, tak i datové vrstvě.

6.4.2 Výsledky

Systémovým testům byla v portálu věnována největší pozornost a také vedly k odhalení velkého množství chyb, které byly po reportování opraveny. Portál byl pokrýván systémovými testy systematicky, aby nedošlo k vynechání některé z jeho částí. První velkou část tvořily stránky portálu vyskytující se před přihlášením. Nejvýznamnější nalezené chyby byly nalezeny v přihlašovacím dialogu, kde bylo okno pro zadání emailové adresy Case sensitive, a tudíž se nebylo možné i se správnou emailovou adresou do portálu přihlásit, stejně jako nefunkční přihlášení pomocí sociální sítě LinkedIn. Další významnou chybou byla nesprávná validace formulářových dat při registraci nového uživatele, kdy bylo možné registrovat uživatele s neplatným jménem a datem narození. Druhou velkou částí při testování portálu, byl portál za přihlašovacím dialogem, kdy docházelo ke kontrole vytváření nových objektů a prováděných změn (například změna hesla). Zde byly nejvýznamnější chyby nalezeny v oprávnění uživatelů, kdy uživatel typu `ROLE_READER` neměl vůbec oprávnění přihlásit se do systému. Dále bylo zjištěno, že uživatel s oprávněním READER má v portálu větší pravomoce nežli definuje specifikace (např. mohl zakládat nové skupiny a oprávnění měl téměř identické jako uživatel s rolí `ROLE_USER`).

Pro testy ověřující funkčnost vytvoření nové výzkumné skupiny a scénáře je nutné vytvořit prázdnou testovací databázi a naplnit ji čtyřmi testovacími účty (blíže v tabulce 1). Ta však musí být po dokončení testů smazána, aby bylo možné opětovně vytvořit stejné objekty. Z časových důvodů však nebyla tato databáze vytvořena. Test pro přidání nového experimentu se nepodařilo ve stávající vývojovém stavu portálu vytvořit.

V rámci portálu bylo pomocí systémových testů odhaleno celkem 21 chyb, které byly po reportování opraveny.

6.5 Zátěžové testy

Nutnost vytvoření výkonových testů plyne z potřeby zjistit, zdali je vyvíjená portálová aplikace dostatečně výkonná a optimalizovaná pro provoz v reálných podmínkách. Je nepřípustné, aby byla aplikace při běžném, případně nadprůměrném zatížení nestabilní. Vzhledem k úspěšné registraci EEG/ERP portálu mezi zdroje NIF [11] se předpokládá zvýšení návštěvnosti portálu, protože obsahuje data z neuroinformatických experimentů, která jsou sdílená mezi dalšími výzkumnými skupinami. V důsledku vyššího vytížení serveru je třeba specifikovat chování aplikace při zátěži.

Na počátku testování bylo definováno, jaké je očekávané průměrné a maximální vytížení portálové aplikace. Protože je EEG/ERP portál určen pro relativně malé množství zasvěcených lidí, očekává se průměrné vytížení v řádu desítek současně pracujících osob, maximální vytížení pak v řádech stovek. Na tomto základě byly vytvořeny scénáře, které se spouštěly v 50 vláknech, což odpovídá průměrnému vytížení portálu. Z naměřených hodnot se dá predikovat chování aplikace při maximálním vytížení, případně při krátkodobém extrémním zatížení[22].

Aby se dalo z naměřených hodnot objektivně předpovídat chování systému při extrémním zatížení, byly zjištěny nejnáročnější příkazy (požadavky), které systém výpočetně nejvíce zatěžují. To bylo provedeno pomocí programu BadBoy (Kapitola 5.3), který jako jednu ze svých funkcí přehledně zobrazuje časy odezvy jednotlivých stránek. Dle těchto údajů byly vytvořeny odpovídající testy, které představovaly nejhorší možné scénáře chování uživatele.

Další snahou při vytváření objektivních testů bylo spuštění veškerých testů mimo počítač, na kterém je webový portál spuštěn. Tím na serveru nedošlo k alokaci velkého množství paměti potřebného pro vytvoření testovacích vláken[23] a zabrání procesorového času, který mohl být naopak využit pro účely portálu. Měření tak odpovídalo reálným podmínkám, při kterých bude spuštěn ostrý server.

Nemalou úlohu při testech tvoří zpoždění v rámci přenosu požadavků v síti. Pokud je zařízení, které posílá požadavky na server v jiné síti a u jiného poskytovatele

[22] Testy extrémního zatížení nebyly provedeny pro nedostatek hardwarových prostředků.

[23] Na serveru bylo ušetřeno přibližně 270MB potřebných pro inicializaci testů a spuštění 50 vláken.

URL	%	ms	Size	Sequence
... /lists-hardware	3.4%	583	23k	583 ms
... /files/images/footer-bg.gif	0.4%	66	0.2k	66 ms
... /files/images/action-box-bg.gif	0.4%	69	0.1k	69 ms
... /files/images/datatable-header-bg.gif	0.4%	70	0.2k	70 ms
... /files/images/menu-item-bg.gif	0.4%	69	0.2k	69 ms
... /files/images/header.gif	0.8%	134	37.2k	134 ms
... /scenarios-page	8.2%	1402	16.2k	1402 ms
... /files/images/footer-bg.gif	0.4%	60	0.2k	60 ms
... /files/images/datatable-header-bg.gif	0.4%	73	0.2k	73 ms
... /files/images/menu-item-bg.gif	0.4%	72	0.2k	72 ms
... /files/images/header.gif	0.8%	139	37.2k	139 ms
... /files/images/main-menu-selected-bg.gif	0.7%	116	0.2k	116 ms
... /experiments-list	20.9%	3567	21k	3567 ms

Obrázek 2: Ukázka doby potřebné pro načtení celého obsahu stránek

internetového připojení, může putovat paket s požadavkem na testovaný server přes několik vzdálených serverů. S každým přesměrováním na vzdáleném serveru získává paket zpoždění až v řádu jednotek milisekund. Aby se tomuto zpoždění předešlo, a tím také ke zkreslení výsledků testů, je vhodné spouštět testy na počítači, který se nachází ve stejné síti. Tím je z přenosu paketu odbourána cesta přes vzdálené servery a paket směřuje přímo k testovanému serveru.

Na ukázce délky pracování a přenosu dat (Obrázek 2) je vidět, že nejdéle trvá zpracování seznamů parametrů, scénářů a experimentů. Důvodem je řazení dat získaných z databáze na úrovni serveru, což má za následek vysokou výpočetní a paměťovou náročnost. Z toho lze usuzovat, že získávání těchno seznamů je nejnáročnější úlohou a je možné na nich postavit zátěžové testy.

6.5.1 Testovací scénáře

V systému se nachází čtyři typy uživatelských rolí (tabulka 1) s různým oprávněním. Aby bylo testování důkladné, byly vytvořeny testovací scénáře pro každou z těchto rolí. Tím dochází k ověření, že je stránka dostupná pro danou uživatelskou roli

(pokud by nebyla, skončil by test chybou) a navíc k pokrytí většího množství kódu a rozhodovacích hran.

Pomocí programu BadBoy byly zjištěny v portálu nejnáročnější operace (obrázek 2), Těmito operacemi bylo zobrazení seznamů testovacích experimentů, scénářů, výzkumných skupin a parametrů. Z tohoto důvodu byly zátěžové testy navrženy tak, aby procházely položky menu a především všechny zmíněné seznamy, v důsledku čehož bylo dosaženo nejvyššího vytížení.

Testovací scénáře jsou rozděleny dle položek hlavního menu, kdy každému testovacímu scénáři odpovídá jedna z položek hlavního menu. Poté, co je vybrána příslušná položka, dochází k cyklickému procházení všech položek menu druhé úrovně. Tímto strukturováním zátěžových testů je dosaženo rozdělení portálu na logické části a otestování každé z těchto částí zvlášť. Na základě měření lze určit, které části jsou výpočetně a paměťově nejnáročnější, případně určit úzká místa portálu a ty optimalizovat.

Příklad struktury testu pro položku hlavního menu `Lists` obsahující seznamy konfigurovatelných parametrů vypadá následovně:

- Zobrazení hlavní stránky s přihlašovacím formulářem
- Přihlášení uživatele s odpovídající rolí
- Výběr položky `Lists` v hlavním menu
- Procházení položek menu druhé úrovně
  - `Hardware definitions`
  - `Optional parameters for people`
  - `Optional parameters for experiments`
  - `File metadata definitions`
  - `Weather definitions`
  - `Artifact definitions`
- Odhlášení uživatele

Z tohoto testovacího scénáře jsou vytvořeny testovací případy s uživateli mající odpovídající uživatelskou roli v rámci portálu.

Další variantou zátěžového testu je procházení všech položek hlavního menu (bez procházení menu druhé úrovně). Tento test se více přibližuje činnosti běžného uživatele a test nerozděluje aplikaci na funkční bloky, ale testuje ji jako celek. Struktura testu je jednoduchá, ale z funkčního hlediska značně důležitá.

- Zobrazení hlavní stránky s přihlašovacím formulářem
- Přihlášení uživatele s odpovídající rolí
- Výběr položek v hlavním menu
  - `Home`
  - `Articles`
  - `Search`
  - `Experiments`
  - `Scenarios`
  - `Groups`
  - `People`
  - `Lists`
- Odhlášení uživatele

Seznam položek menu se mění společně s oprávněním uživatele, který je do systému přihlášen. Jedná se o bezpečnostní opatření provedené v rámci portálu pro omezení práv daných uživatelských skupin. Proto je pro každý typ uživatelského účtu jiný testovací případ.

Jako obecnější varianta testu pro zátěžové testování bylo zvoleno procházení celého hlavního menu společně s částečným procházením menu druhé úrovně včetně procházení a řazení seznamů. Procházení menu není systematické a test se podobá činnosti běžného uživatele mnohem více než zátěžové testy logických celků. Test byl navržen především pro výkonostní testování portálu jako celku a je ho vhodné použít pro simulaci reálného provozu na serveru, tj. pro testování nevyužívající pouze nejnáročnější činnosti. Zjednodušená struktura obecného testu vypadá takto:

- Zobrazení hlavní stránky s přihlašovacím formulářem
- Přihlášení uživatele s odpovídající rolí
- Procházení položek menu první a druhé úrovně
- Odhlášení uživatele

V rámci měření byl proveden test skládající se z kombinace předchozích testů. Test byl složen z následujících testovacích scénářů:

- Procházení všech položek menu první úrovně
- Procházení všech položek menu druhé úrovně pro `Experiments` v hlavním menu
- Procházení všech položek menu druhé úrovně pro `Scenarios` v hlavním menu
- Procházení všech položek menu druhé úrovně pro `People` v hlavním menu
- Procházení všech položek menu druhé úrovně pro `Lists` v hlavním menu

Každý z těchto testovacích scénářů byl spuštěn v 10 vláknech, čímž bylo dosaženo simulace 50 uživatelů pracujících s portálem najednou, přecemž bylo k testům přidáno několik sledovacích funkcí, které zaznamenávaly dobu odezvy a reprezentovaly jí pomocí tabulek a grafů [9]. Zároveň byl na testovaném serveru spuštěn nástroj pro monitorování systému a využití dostupných zdrojů. Jednalo se o nástroj `Sledování výkonu`, který je standardně integrován do systému MS Windows a poskytuje přehledné uživatelské rozhraní s možností výběru sledovaných veličin, včetně jejich reprezentace v grafech. Pro sledování vytížení byly klíčové čtyři základní hodnoty. Ačkoli by bylo možné sledovat i další, pro testy důležité údaje, byly zvoleny pouze následující čtyři pro udržení přehlednosti grafu.

- Procentuální vyjádření časů procesoru
- Procentuální vyjádření využití paměti
- Počet přijatých datagramů za vteřinu
- Počet odeslaných datagramů za vteřinu

6.5.2 Výsledky

Výsledky testu a naměřené hodnoty lze najít v příloze C. Celkově lze hodnotit měření jako úspěšné a systém dostatečně stabilní. Se zatížením 50 uživatelů neměl server problém a stále disponoval značnou rezervou. Během měření se využití operační paměti pohybovalo v rozmezí 36-50% (průměrná hodnota 44%), přičemž při běžném vytížení (do 10 současně pracujících osob) se pohybuje využití paměti mezi 33 až 38%. Také procesorový čas nebyl využit na maximum, ale průměrně na 60%.

Pokud by byly brány v potaz paměťové nároky pro 50 uživatelů, tak pro nejhoší případ (rozdíl mezi spodní hranicí využití paměti pro systém bez zátěži a horní hranici pro systém při zátěži) tvoří rozdíl využití 17%. Při 4GB paměťového prostoru činí 17% 696MB, což je hodnota značně nadsazená, protože oněch 50% bylo využito pouze krátkodobě (než byl spuštěn Garbage Collector, který uvolnil pamět v řádu jednotek procent) a rozdíl naměřených hodnot je nejhorší možný. Přesto volná pamět poskytuje prostor pro další 2,6 násobek dalších uživatelů. Portál je tedy schopen obsloužit minimálně 180 současně pracujích uživatelů.

6.6 Testy použitelnosti

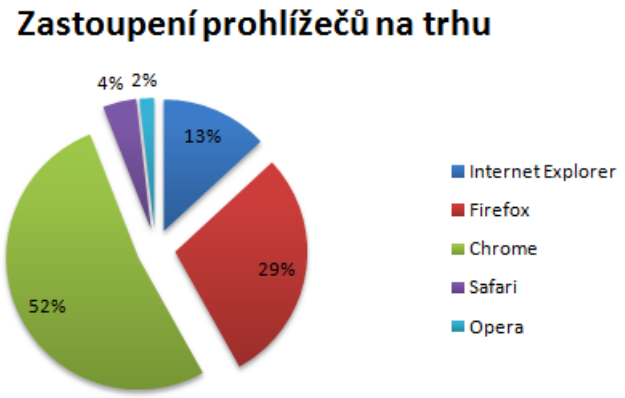
Mezi testy použitelnosti patří optimalizace intuitivnosti webové aplikace, kontrola všech odkazů, zdali jsou funkční a směřují na správné místo či se aplikace zobrazuje korektně v různých webových prohlížečích. Ačkoli se většinou nejedná o zásadní vady, zvláště pak intuitivnost aplikace, je důležité věnovat jim náležitou pozornost.

V rámci portálu byl vzhled již definován a vytvořen a nemělo význam se jím zabývat. Co již však bylo nutné otestovat, byla kompatibilita zobrazení napříč webovými prohlížeči. Webové prohlížeče sice dodržují standardy zobrazení HTML a CSS[24], ale ne všechny podporují všechny příkazy a funkce, případně jsou interpretovány trochu jinak, díky čemuž se stejná stránka zobrazuje v různých prohlížečích totožně.

Testování proběhlo v nejpoužívanějších prohlížečích v různých verzích. Jako ukazatel

[24]CSS (Cascading Style Sheets) je jazyk popisující způsob zobrazení obsahu webové stránky. O standardizaci se stará konsorcium W3C.

zastoupení webových prohlížečů na trhu byly použity statistiky konsorcia W3C. Dle zveřejněných statistik zaujímá vedoucí postavení pět webových prohlížečů (Obrázek 3). Další prohlížeče tedy nebyly brány v úvahu, protože jejich rozšířenost je zanedbatelně malá a případná optimalizace zbytečná.



Obrázek 3: Zastoupení vedoucích webových prohlížečů na trhu

Každý z těchto prohlížečů má několik verzí, které se na trhu vyskytují současně. Opět bylo využito statistik konsorcia W3C, které společně s prohlížeči sleduje jejich verze. Testovány byly vždy dvě verze s největším výskytem, zpravidla poslední dvě. Bližší informace o četnosti výskytu verzí ukazuje tabulka 2, kde jsou zobrazeny verze prohlížečů s četností výskytu nad 1%. V případě prohlížeče Chrome byly zvoleny verze `C 26` a `C 25`. Ačkoli má verze `C 24` stejné procentuální zastoupení jako `C 26`, předpokládá se do budoucna se zastoupením stále nižším, protože se jedná o starší verzi, a proto byla tato verze zavržena.

Prohlížeč	**Verze prohlížeče**	**Zastoupení na trhu**
Internet Explorer	IE 10	1,3%
	IE 9	5,2%
	IE 8	5,5%
Mozila Firefox	FF19	19,0%
	FF 18	1,4%
Chrome	C 26	2,1%
	C 25	44,2%
	C 24	2,1%
Safari	S 6	2,8%
	S 5	1,2%
Opera	O 12	1,1%

Tabulka 2: Zastoupení jednotlivých verzí webových prohlížečů na trhu [7]

Protože testy použitelnosti buď nelze automatizovat vůbec nebo velice špatně a omezeně, bylo toto testování provedeno manuálně. Sledováno bylo především CSS formátování, kdy bylo nejčastějším problémem špatné pozicování prvku a zalomování textu. Největší pozornost byla věnována hlavní stránce a menu, které tvoří nejdůležitější část portálu. Testování bylo provedeno v následujícím pořadí.

- Kontrola přihlašovací stránky
- Kontrola domovské stránky portálu (stránka po přihlášení)
- Kontrola hlavního menu
- Kontrola menu druhé úrovně
- Kontrola seznamů
  - Experimenty
  - Scénáře
  - Skupiny
  - Seznamy parametrů
- Kontrola stránky s informacemi o uživatelském účtu
- Kontrola zbylých stránek pro vytváření nových objektů (parametrů, skupin, scénářů).

U každé z výše uvedených stránek byla pozornost soustředěna na hlavičku, obsah a patičku stránky. Pro testování zobrazení portálu v prohlížeči Internet Explorer byl použit nástroj `IETester`, který je blíže popsán v kapitole 5.5.

Společně s kontrolou CSS formátování byla provedena také kontrola funkčnosti všech odkazů, které se na stránce vyskytovaly. Zde se testovalo, zdali odkazy směřují na existující stránku[25], případně jestli se jedná o správnou stránku. Ačkoli se na trhu vyskytuje mnoho nekomerčních aplikací, které kontrolují odkazy automaticky, nebyl nalezen žádný, který umí pracovat s celým EEG/ERP portálem. Problém spočívá v nutném počátečním přihlášení, bez kterého není možné do systému přistoupit. Ačkoli některé nástroje podporují možnost autentizace, v případě EEG/ERP portálu byly

[25] V případě, že odkaz ukazuje na neexistující stránku, je zobrazeno chybové hlášení `Error 404`.

odkazy pro odeslání přihlašovacích údajů označené jako nepodporované[26]. Proto bylo nutné přistoupit ke komerčním nástrojům, které tento typ odkazů podporují.

Pro ověření funčnosti odkazů byl vybrán použit nástroj Fast Link Checker (kapitola 5.7). Po provedení testu se ve výsledném reportu nacházely tři typy chybových a stavových HTML kódů.

- **200 (OK)** - Stránka byla bez problémů nalezena a je plně funkční.
- **302 (moved temporarily)** - Stránka byla dočasně přesunuta na jiné místo. V rámci portálu se tento typ kódu vyskytovat velice často, protože s každým načtením stránky se zvyšuje její ID, což je bráno jako přesunutí stránky na novou adresu.
- **410 (gone)** - Chybový stav oznamující, že stránka existovala, ale byla přesunuta na jiné místo a již není dále přístupná. V portálu se tato chyba vyskytovala celkem na 10 místech. Důvodem této chyby je logika vytváření seznamu dostupných odkazů, které se na stránce vyskytují. Po načtení stránky, jsou všechny nově nalezené odkazy přesunuty do fronty odkazů čekajících na zpracování. V případě, že je otevřeno okno s možností volby mezi dvěma hodnotami, a jedna z nich je vybrána, je odkaz na druhou hodnotu zařazen do seznamu, přičemž se stává neplatným.

Jiný typ HTML kódu nebyl zaznemenán a je možné říci, že všechny otestované stránky obsahují validní kódy odkazy. Podrobnou zprávu z testování je možné vidět na přiloženém CD, kde se v souborech HTML a CSV nachází export testovaných dat (zkontrolovaných odkazů).

Pro kontrolu odkazů v portálu, kde není potřeba registrace, byl použit nástroj Xenu's Link Sleuth (kapitola 5.6), který podrobně kontroluje odkazy na webové stránce společně s CSS soubory. Díky tomu byly nalezeny dvě chybné reference v souborech CSS na neexistující obrázek.

[26]Například nástroj **Web List Validator** (http://www.relsoftware.com/wlv/)

Cesta k chybějícímu obrázku	Soubor s chybnou referencí
http://147.228.64.172:8080/ files/imgs/sky_blue_sel_tree.png	http://147.228.64.172:8080/ files/dhtmlxtree.css
http://147.228.64.172:8080/ files/arrow.gif	http://147.228.64.172:8080/ files/colorPicker.css

Tabulka 3: Nalezené neplatné odkazy

6.6.1 Výsledky

Při testech použitelnosti byly nalezeny dvě chybné reference (tabulka 3). Ostatní z kontrolovaných odkazů byly funkční a chovaly se korektně. Pro část portálu vyžadující přihlášení byla využita TRIAL verze programu Fast Link Checker (kapitola 5.7), která podporuje maximálně 2500 zkontrolovaných odkazů. Proto byl portál testován pouze v tomto rozsahu. Krom kontroly odkazů došlo také ke kontrole zobrazení portálu mezi nejpoužívanějšími prohlížeči. Zde nebyly nalezeny žádné odchylky[27] a systém se zobrazoval korektně. Pro ověření a kontrolu nadpisů vyskytujících se v portále bylo využito nástroje Seleniu, kdy jsou procházeny všechny položky hlavního menu a menu druhé úrovně. U každé z těchto položek je porovnáván popis v menu s nadpisem dané stránky. Během těchto testů byly nalezeny 2 odchylky týkající se nesprávných nadpisů stránky. Poslední částí při testování použitelnosti byla kontrola chybových hlášení u formulářů. V tomto případě byly nalezeny chyby ve formuláři pro registraci nového uživatele, kdy docházelo k zobrazování nesprávných chybových hlášení (hlášení neodpovídalo vzniklé chybě). Všechny nalezené chyby byly opraveny. Zprávu z automatického testování odkazů je možné najít na přiloženém CD ve formátu CSV.

[27] Platnost k březnu 2013

7 Dosažené výsledky

Po dekomponování EEG/ERP portálu byly pro jednotlivé části a funkční celky navrženy a implementovány testy, které odhalily existující chyby. V rámci datové vrstvy nebyly nalezeny žádné chyby, ale byla ověřena správnost ORM mapování a funkčnost DAO objektů, díky čemuž vznikla množina automatických testů validující datovou vrstvu aplikace. Takto vytvořené testy je tedy možné zakomponovat do automatických testů spouštěných s každým buildem aplikace.

V aplikační vrstvě portálu, která byla testována systémovými testy, byly nalezeny závažné i méně závažné chyby, které byly po reportování opraveny a opětovně zkontrolovány. Mezi nalezené závažné chyby patřilo nesprávné nastavení oprávnění uživatelských rolí či problémy s přihlášením do portálu pomocí účtu na sociálních sítích. Mezi méně závažné chyby patřilo nesprávné validování dat ve formulářích a problémy s editací existujících záznamů.

Při testování prezentační vrstvy byly kontrolovány odkazy vyskytující se na internetových stránkách, jejich funkčnost a relevantnost. Při těchto testech byly v CSS souborech nalezeny odkazy na již neexistující obrázky. Dalším typem testů prezentační vrstvy byla kontrola chybových hlášek ve formulářích, kdy bylo zjištěno, že docházelo k zobrazování nesprávných upozornění a varování. Poslední část testů prezentační vrstvy tvořila kontrola napříč webovými prohlížeči, kdy bylo zobrazení vždy korektní, a tudíž nebyly nalezeny žádné chyby.

Pro ověření funkce portálu jako celku bylo provedeno zátěžové testování, které mělo ověřit způsobilost systému pracovat s očekávaným počtem uživatelů a zjistit jeho chování při nadstandardní zátěži. Během testů bylo zjištěno, že je potřeba zvýšit operační paměť serveru ze, na kterém portál běží a to ze dvou na čtyři GB. Po navýšení této paměti prošel portál zátěžovými testy se značnou rezervou. Je tedy možné říci, že je dimenzován tak, aby zvládl očekávané zatížení.

8 Závěr

V práci bylo nutné se nejprve seznámit s účelem a funkčností EEG/ERP portálu. Dalším krokem bylo zjistění technologií, které portál pro svou činnost využívá a možnosti jejich testování. Jednalo se především o technologie Hibernate a Apache Wicket. Na základě těchto informací byly pro celou aplikaci navrženy a následně i implementovány automatické testy. Pro případy, kdy nešlo testy automatizovat, bylo navrženo a provedeno manuální testování.

Po ověření základní funkčnosti portálu bylo přikročeno k výkonnostním testům, které ověřovaly chování aplikace při běžně očekávaném provozu a při vysoké zátěži. Po zjištění nejužších míst aplikace byly navrženy a implementovány testy. Ty simulovaly činnost uživatelů, kteří pracují s portálem současně a využívají jeho nejnáročnější operace. Po vyhodnocení těchto testů je možné říci, že na předpokládanou zátěž je portál i server připraven.

Další možná rozšíření portálu spočívají ve vytvoření kompletního testovacího prostředí, které by bylo možné spouštět s každou automatickou kompilací projektu[28] a udržovat takportál v konzistentním stavu. Dále by bylo vhodné, v případě nalezení chyby, vytvořit report, který by byl odeslán správci portálu a vývojáři, který dělal poslední aktualizaci (commit). Díky tomu by byla chyba rychle nalezena a opravena.

[28] Příkladem může být program Maven

Seznam zkratek

AJAX Asynchronous JavaScript and XML - technologie postavená na JavaScriptu určená pro interaktivní tvorbu aplikací.

CSS Cascading Style Sheets - jazyk popisující způsob zobrazení obsahu webové stránky.

CSV Comma-separated values - testový soubor, v němž jsou jednotlivé hodnoty na řádku odděleny čárkou.

DAO Data Access Object - objekt v datové vrstvě sloužící pro přístup k datům.

EEG elektroencefalogram - měření mozkové aktivity pomocí elektrod přiložených na lebce.

ERP Event-Related Potential - evokovaný potenciál je odezva vyvolaná smyslovým nebo motorickým podnětem.

HTML HyperText Markup Language - značkovací jazyk pro hypertext.

JMX JMeter File - přípona souboru s JMeter testem.

MVC Model-View-Controller - softwarová architektura dělící aplikaci do tří komponent, jimiž jsou datový model, uživatelské rozhraní a řídicí logiku.

ORM Object Relational Mapping - mapování objektových dat na relační databázi.

POJO Plain Old Java Object - objekty sloužící k ORM mapování dat

TDD Test Driven Development - typ extrémního programování, kdy jsou nejprve implementovány testy a až poté vyvíjený kód.

XML Extensible Markup Language - značkovací jazyk vyvinutý a standardizovaný W3C konsorciem.

Reference

[1] **Patton, Ron:** *Testování software*. CPress. 2002. ISBN 80-7226-636-5.

[2] **Johnston, Ken; Page, Alan; Rollison, Bj:** *How We Test Software at Microsoft*. Microsoft Press. 2008. ISBN 978-07-356-2425-2.

[3] **Stephens, Matt; Rosenberg, Doug:** *Design-Driven Testing*. Apress. 2010. ISBN 978-1-4302-2943-8.

[4] **Beck, Kent:** *Test Driven Development: By example*. Perason Education. 2003. ISBN 03-2114-653-0.

[5] **Borovcová, Anna:** *Testování webových aplikací*. Diplomová práce. MFF UK. 2008

[6] **Hartman, František:** *Testování portálových aplikací*. Diplomová práce. FI MU. 2011

[7] Web Statistics and Trends. *W3schools* [Online] 3, 2013. <http://www.w3schools.com/browsers/browsers_stats.asp>.

[8] Selenium Tips. *Apache Wiki* [Online] 5, 2013. <https://cwiki.apache.org/WICKET/selenium-tips.html>.

[9] JMeter User's Manual. *Apache JMeter* [Online] 5, 2013. <http://jmeter.apache.org/usermanual/index.html>.

[10] Unit testing Wicket pages. *Wicket testing* [Online] 5, 2013. <http://www.ibm.com/developerworks/web/library/wa-aj-wicket/>.

[11] NIF *About NIF* [Online] 5, 2013. <http://www.neuinfo.org/about/>.

[12] Wicket: A simplified framework for building and testing dynamic Web pages *IBM* [Online] 5, 2013. <http://www.ibm.com/developerworks/web/library/wa-aj-wicket/>.

[13] Problémy s pokrytím kódu *Testování softwaru* [Online] 5, 2013. <http://testovanisoftwaru.blogspot.cz/2009/10/problemy-s-pokrytim-kodu.html>.

Seznam obrázků

Seznam tabulek

A Ukázka jednotkového testu pro výzkumnou skupinu

```
package cz.zcu.kiv.eegdatabase.data.dao;

import cz.zcu.kiv.eegdatabase.data.AbstractDataAccessTest;
import cz.zcu.kiv.eegdatabase.data.pojo.Person;
import cz.zcu.kiv.eegdatabase.data.pojo.ResearchGroup;
import org.junit.Before;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.transaction.annotation.Transactional;

import static net.sf.ezmorph.test.ArrayAssertions.assertEquals;

public class ResearchGroupDaoTest extends AbstractDataAccessTest
    {

  @Autowired
  protected ResearchGroupDao researchGroupDao;
  @Autowired
  protected PersonDao personDao;

  protected ResearchGroup researchGroup;
  protected Person person;

  @Before
  public void setUp() throws Exception {
    person = personDao.getPerson("testaccountforeeg@seznam.cz");
        // ROLE_ADMIN

    researchGroup = new ResearchGroup();
    researchGroup.setDescription("test-description");
    researchGroup.setTitle("test-title");
    researchGroup.setPerson(person);
  }

  @Test
  @Transactional
  public void testCreateResearchGroup(){
    int count = researchGroupDao.getCountRecords();
    researchGroupDao.create(researchGroup);
    assertEquals(count + 1, researchGroupDao.getCountRecords());
  }

  @Test
  @Transactional
  public void testGetResearchGroupTitle() throws Exception {
```

```
    researchGroupDao.create(researchGroup);
    assertEquals("test-title",
      researchGroupDao.getResearchGroupTitle(
        researchGroup.getResearchGroupId()));
  }


  @Test
  @Transactional
  public void testGetListOfGroupMembers() throws Exception {
    researchGroupDao.create(researchGroup);
    assertEquals(0,
      researchGroupDao.getListOfGroupMembers(
        researchGroup.getResearchGroupId()).size());
  }
}
```

B Selenium test pro ověření přihlášení uživatele s oprávněním ADMIN, varianta XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http
    ://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="
    en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case
    ">
<meta http-equiv="Content-Type" content="text/html; charset=UTF
    -8" />
<link rel="selenium.base" href="http://localhost:8080/" />
<title>Login</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">Login</td></tr>
</thead><tbody>
<tr>
        <td>open</td>
        <td>/home-page?1</td>
        <td></td>
</tr>
<tr>
        <td>assertTextPresent</td>
        <td>No user logged</td>
        <td></td>
</tr>
<tr>
        <td>type</td>
        <td>//input[@wicketpath='login_userName']</td>
        <td>testAccountForEEG@seznam.cz</td>
</tr>
<tr>
        <td>type</td>
        <td>//input[@wicketpath='login_password']</td>
        <td>123456</td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//input[@wicketpath='login_submit']</td>
        <td></td>
</tr>
<tr>
        <td>verifyTextPresent</td>
        <td>Logged user: testaccountforeeg@seznam.cz</td>
```

```
        <td></td>
</tr>
<tr>
        <td>clickAndWait</td>
        <td>//a[@wicketpath='logout']</td>
        <td></td>
</tr>
<tr>
        <td>verifyTextPresent</td>
        <td>No user logged</td>
        <td></td>
</tr>
</tbody></table>
</body>
</html>
```

C Selenium test pro ověření přihlášení uživatele s oprávněním ADMIN, varianta Java

```
package com.example.tests;

import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class LoginAdmin {
  private WebDriver driver;
  private String baseUrl;
  private boolean acceptNextAlert = true;
  private StringBuffer verificationErrors = new StringBuffer();

  @Before
  public void setUp() throws Exception {
    driver = new FirefoxDriver();
    baseUrl = "http://147.228.64.172:8080/";
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.
       SECONDS);
  }

  @Test
  public void testLoginAdmin() throws Exception {
    driver.get(baseUrl + "/home-page?1");
    assertTrue(driver.findElement(By.cssSelector("BODY")).
       getText().matches("^[\\s\\S]*No user logged[\\s\\S]*$"));
    driver.findElement(By.xpath("//input[@wicketpath='
       login_userName']")).clear();
    driver.findElement(By.xpath("//input[@wicketpath='
       login_userName']")).sendKeys("testAccountForEEG@seznam.cz
       ");
    driver.findElement(By.xpath("//input[@wicketpath='
       login_password']")).clear();
    driver.findElement(By.xpath("//input[@wicketpath='
       login_password']")).sendKeys("123456");
    driver.findElement(By.xpath("//input[@wicketpath='
       login_submit']")).click();
    try {
      assertTrue(driver.findElement(By.cssSelector("BODY")).
         getText().matches("^[\\s\\S]*Logged user:
         testaccountforeeg@seznam\\.cz[\\s\\S]*$"));
```

```
    } catch (Error e) {
      verificationErrors.append(e.toString());
    }
    driver.findElement(By.xpath("//a[@wicketpath='logout']")).
       click();
    try {
      assertTrue(driver.findElement(By.cssSelector("BODY")).
         getText().matches("^[\\s\\S]*No user logged[\\s\\S]*$")
         );
    } catch (Error e) {
      verificationErrors.append(e.toString());
    }
  }

  @After
  public void tearDown() throws Exception {
    driver.quit();
    String verificationErrorString = verificationErrors.toString
       ();
    if (!"".equals(verificationErrorString)) {
      fail(verificationErrorString);
    }
  }

  private boolean isElementPresent(By by) {
    try {
      driver.findElement(by);
      return true;
    } catch (NoSuchElementException e) {
      return false;
    }
  }

  private String closeAlertAndGetItsText() {
    try {
      Alert alert = driver.switchTo().alert();
      if (acceptNextAlert) {
        alert.accept();
      } else {
        alert.dismiss();
      }
      return alert.getText();
    } finally {
      acceptNextAlert = true;
    }
  }
}
```

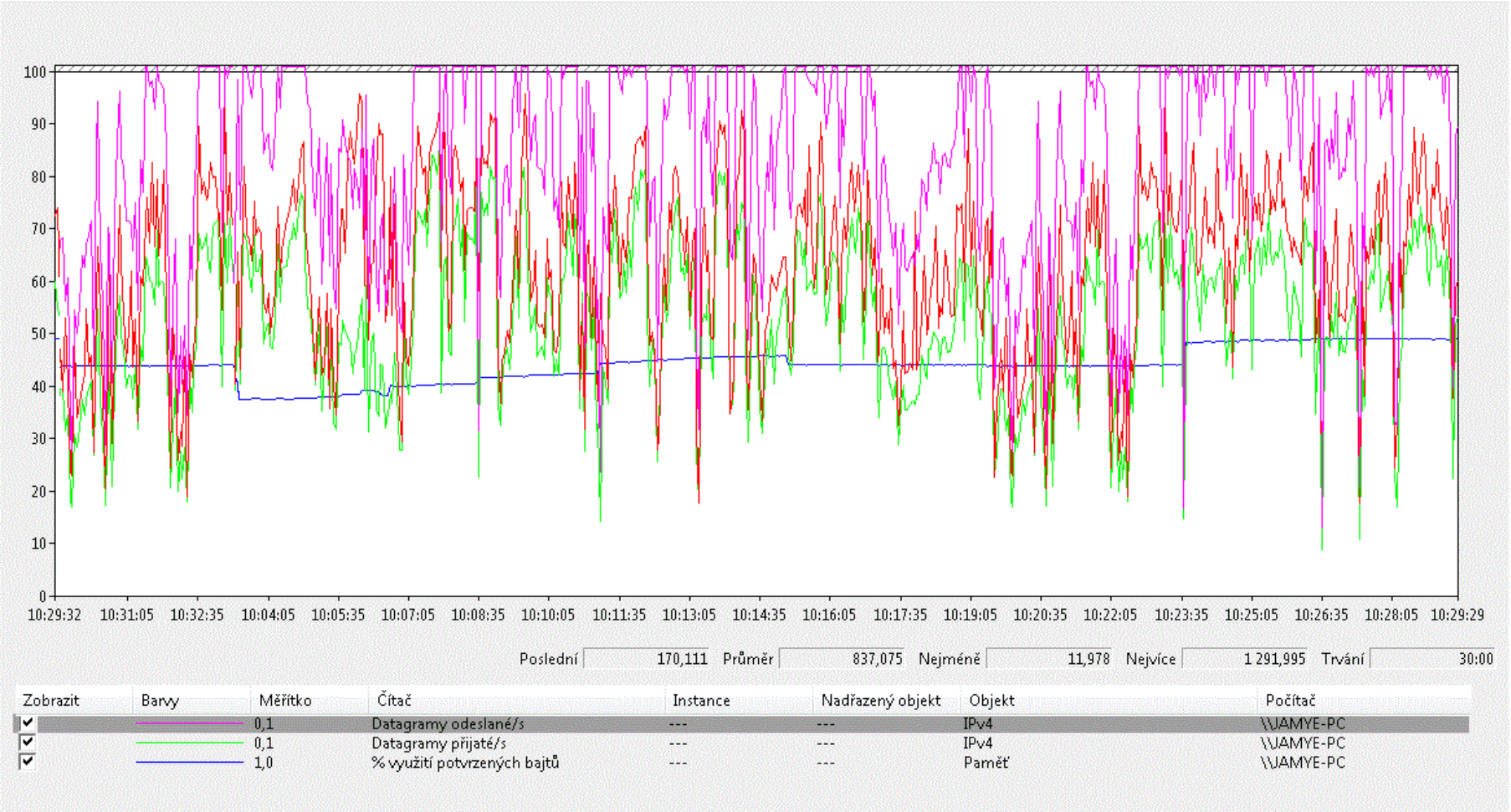
D Hodnoty a grafy naměřené při zátěžových testech

Sledovaná hodnota	**Barva grafu**	**Přepočtový koeficient**
Procentuální vyjádření časů procesoru	Červená	1.0
Procentuální vyjádření využití paměti	Modrá	1.0
Počet přijatých datagramů za vteřinu	Zelená	0.1
Počet odeslaných datagramů za vteřinu	Fialová	0.1

Tabulka 4: Legenda pro graf znárorňující chování systému při zátěži (obrázek 4)

Měřená veličina	**Význam**
Label	Testovaná stránka
# Samples	Počet požadavků zaslaných na danou stránku
Average	Průměrná doba odpovědi stránky na zaslaný požadavek. Vypočítáno pomocí aritmetického průměru.
Median	Medián doby odezvy stránky.
90% line	Fialová
Min	Minimální doba odezvy
Max	Maximální doba odezvy
Error %	Chybovost
Throughput	Propustnost linky. Udává počet obsloužených požadavků za jednotku času.
Bandwidth	Rychlost přenášených dat

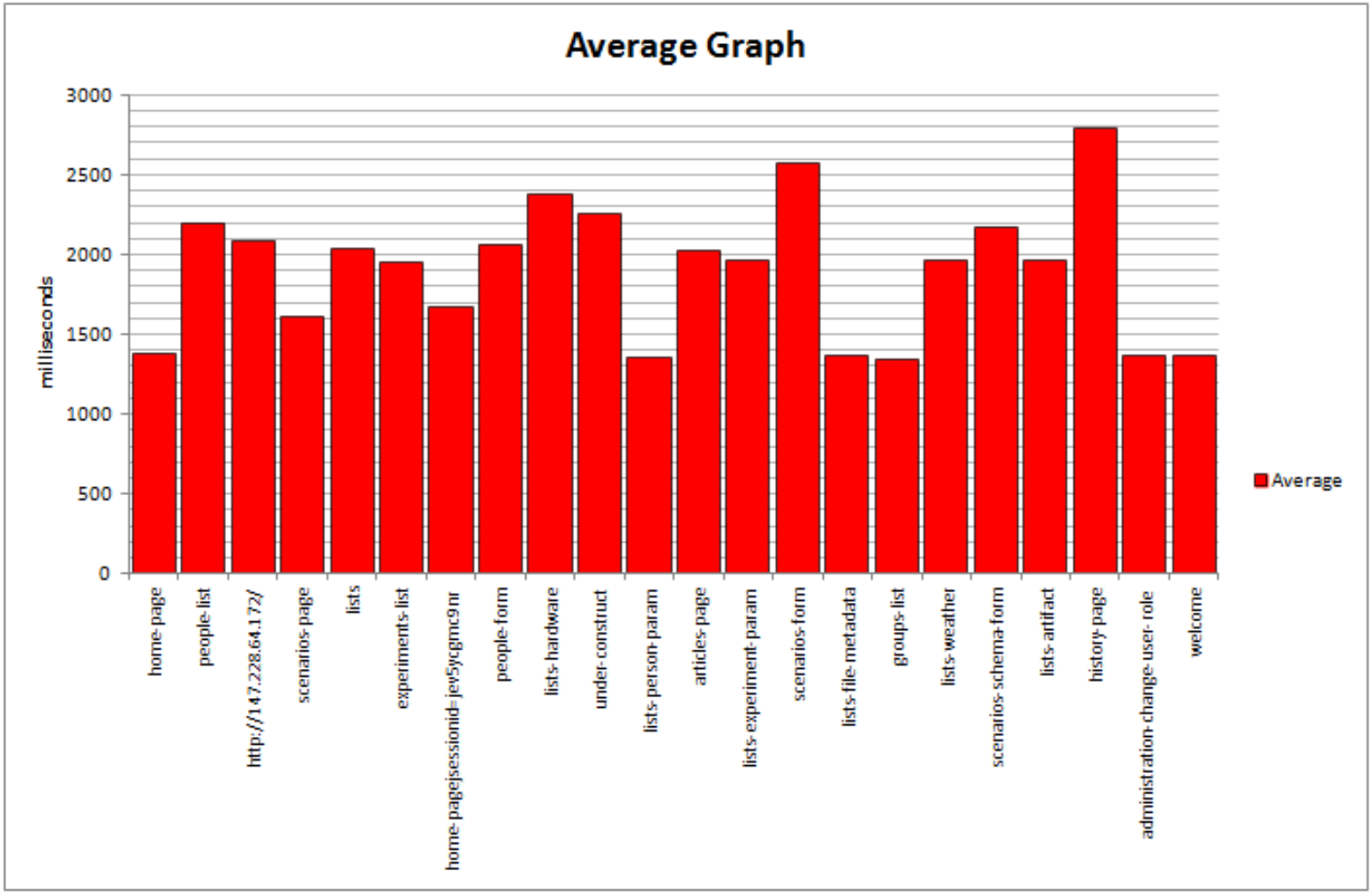
Tabulka 5: Legenda pro tabulku s průměrnými naměřenými hodnotami při zátěžových testech (obrázek 5)



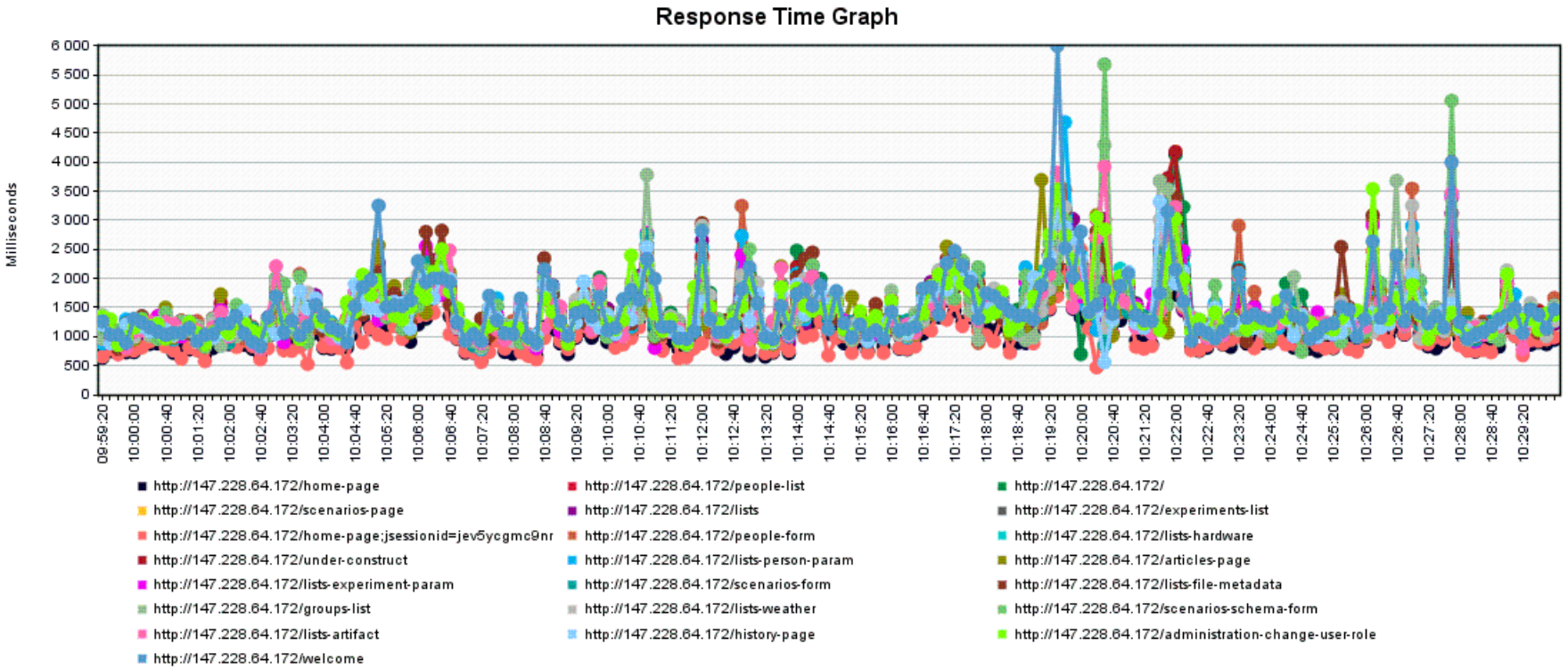
Obrázek 4: Chování testovaného systému v čase

Label	# Samples	Average	Median	90% line	Min	Max	Error %	Throughput	Bandwidth
http://147.228.64.172/home-page	6219	1380	900	1436	40	870358	0,05%	2,3/sec	21,89
http://147.228.64.172/people-list	11323	2189	1235	1963	57	870666	0,10%	4,2/sec	39,84
http://147.228.64.172/	1194	2089	1238	1997	51	869976	0,08%	26,4/min	4,20
http://147.228.64.172/scenarios-page	9720	1613	1235	1958	69	870250	0,03%	3,6/sec	34,22
http://147.228.64.172/lists	2591	2032	1239	1967	140	871009	0,08%	57,2/min	9,12
http://147.228.64.172/experiments-list	11356	1956	1230	1959	45	870628	0,07%	4,2/sec	39,97
http://147.228.64.172/home-pagejsessionid=jev5ycgmc9nr	1193	1671	896	1369	77	869970	0,08%	26,3/min	4,20
http://147.228.64.172/people-form	1270	2055	1230	1999	93	870621	0,08%	28,0/min	4,47
http://147.228.64.172/lists-hardware	4199	2373	1234	1928	80	870972	0,12%	1,5/sec	14,78
http://147.228.64.172/under-construct	4880	2251	1237	1971	68	870631	0,10%	1,8/sec	17,18
http://147.228.64.172/lists-person-param	1402	1352	1233	1943	72	5954	0,00%	45,6/min	7,27
http://147.228.64.172/articles-page	1192	2030	1208	1890	106	869977	0,08%	26,3/min	4,20
http://147.228.64.172/lists-experiment-param	1402	1961	1228	1986	147	869937	0,07%	31,0/min	4,94
http://147.228.64.172/scenarios-form	2134	2570	1258	1937	102	870925	0,14%	47,2/min	7,52
http://147.228.64.172/lists-file-metadata	1401	1362	1251	1983	162	6119	0,00%	45,6/min	7,27
http://147.228.64.172/groups-list	1189	1345	1227	1915	59	10565	0,00%	38,7/min	6,17
http://147.228.64.172/lists-weather	1401	1960	1223	2016	143	869643	0,07%	31,0/min	4,94
http://147.228.64.172/scenarios-schema-form	1066	2171	1235	2019	84	870817	0,09%	23,6/min	3,76
http://147.228.64.172/lists-artifact	1400	1967	1229	1952	158	870328	0,07%	31,0/min	4,94
http://147.228.64.172/history-page	1186	2794	1235	1941	212	870889	0,17%	26,2/min	4,18
http://147.228.64.172/administration-change-user-role	1184	1368	1236	1971	289	11518	0,00%	38,6/min	6,16
http://147.228.64.172/welcome	1184	1371	1223	1995	302	7336	0,00%	38,6/min	6,16
TOTAL	70086	1926	1204	1915	40	871009	0,07%	25,8/sec	246,62

Obrázek 5: Tabulka s naměřenými hodnotami při zátěžových testech (JMeter)



Obrázek 6: Graf s průměrnými časy odezvy při zátěžových testech (JMeter)



Obrázek 7: Graf s časy odezvy testovaných stránek (JMeter)