VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. Adam Dostál


# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# NÁSTROJ PRO FUNKČNÍ TESTOVÁNÍ
SOFTWARE TOOL FOR FACTORY ACCEPTANCE TEST

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Adam Dostál**
AUTHOR

**VEDOUCÍ PRÁCE** — **Ing. Zdeněk Martinásek, Ph.D.**
SUPERVISOR


**BRNO 2018**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

***Student:*** Bc. Adam Dostál ***ID:*** 164804
***Ročník:*** 2 ***Akademický rok:*** 2017/18

**NÁZEV TÉMATU:**

## Nástroj pro funkční testování

**POKYNY PRO VYPRACOVÁNÍ:**

V teoretické části diplomové práce nastudujte oborový standard testů FAT (Factory Acceptance Test) a z jednotlivých skupin se zaměřte na testy autentizace/autorizace, výkonnostní, zátěžové, stability, rozhraní a bezpečnostní. Zvolené oblasti rozpracujte na úroveň úvodní analýzy tvorby testovacích skriptů. V rámci praktické části práce zvolte vhodný softwarový nástroj k automatizovanému vykonání testů a pro výše zmíněné testy navrhněte a implementujte testovací skript, který provede testování a jednoduché vyhodnocení.

**DOPORUČENÁ LITERATURA:**

[1] PALANI, Narayanan. Software Automation Testing Secrets Revealed. 1st edition. 330 pages. ISBN 978-9383952953.

[2] CRISPIN, Lisa. Agile Testing: A Practical Guide for Testers and Agile Teams. 1st edition. 576 pages. ISBN 978-0321534460.

***Termín zadání:*** 5.2.2018 ***Termín odevzdání:*** 21.5.2018

***Vedoucí práce:*** Ing. Zdeněk Martinásek, Ph.D.
***Konzultant:*** Ing. Radim Dodek

**prof. Ing. Jiří Mišurec, CSc.**
*předseda oborové rady*

## ABSTRAKT

Tato práce je zaměřena na realizaci funkčních testů v rámci softwarového projektu v oblasti energetiky ve společnosti *Unicorn*. Teoretická část obecně popisuje projektové metodiky vývoje software a metodiku *Rational Unified Process* (RUP). Dále jsou v práci zahrnuty způsoby testování a model pro řízení kvality FURPS+. Poslední část uvádí funkční testy včetně popisu těch, které jsou použity k otestování vyvíjené aplikace.
Praktickou část tvoří popis energetického projektu *Nemo Link*, vyvíjené aplikace *Nemo Link Dispatch System* (NDS), jednotlivých modulů aplikace, testovacího prostředí, použitých nástrojů a navržené metodologie pro testy. Na základě této metodologie jsou následně vykonány a vyhodnoceny jednotlivé zvolené testy.

## KLÍČOVÁ SLOVA

Funkční testování, metodologie, software, Apache Jmeter, GUI/autorizační test, výkonnostní test, zátěžový test, stability test, stress test, bezpečnostní test.

## ABSTRACT

This work is focused on the implementation of functional tests within the software project in the field of energy in the company *Unicorn*. The theoretical part describes in general the project methodology of software development and the methodology *Rational Unified Process* (RUP). In addition, the test methods are included, and the quality management model FURPS+. The last section introduces functional tests, including a description of those that are used to test the developed application.
The practical part consists of description of the energy project *Nemo Link*, developed application *Nemo Link Dispatch System* (NDS), individial application modules, test enviroment, used tools and designed test methodology. Based on this methodology, individual selected tests are performed and evaluated.

## KEYWORDS

Functional testing, methodology, software, Apache Jmeter, GUI/authorization test, performance test, load test, stability test, stress test, security test.

DOSTÁL, Adam *Nástroj pro funkční testování*: diplomová práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2018. 101 s. Vedoucí práce byl Ing. Zdeněk Martinásek, Ph.D.

# PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Nástroj pro funkční testování“ jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

BRNO ..... .....

podpis autora(-ky)

## PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Zdeňku Martináskovi, Ph.D. a Ing. Radimovi Dodkovi za konzultace, připomínky a rady, které mi v průběhu psaní této práce poskytli.


BRNO . . . . . .

. . . . . .

podpis autora(-ky)


Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

## PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

BRNO ..... .....

podpis autora(-ky)



# OBSAH

# SEZNAM OBRÁZKŮ

# ÚVOD

V minulosti se nepřikládala taková důležitost zajištění kvality (QA - Quality Assurance) v procesu vývoje programového vybavení (dále bude používán pojem „software“) tak jako v současnosti. V oblasti vývoje software se jedná o jednu z nejrychleji se vyvíjejících disciplín a v naprosté většině metodik, ať už klasických nebo agilních, patří dnes testování k velmi důležitým činnostem. Software znamená pro zákazníka obchod a prostřednictvím software dosahuje obchodních cílů. Pokud jsou v software chyby, představují pak pro zákazníka obchodní riziko. Z tohoto důvodu je nezbytné testováním eliminovat chyby v co největší míře.

Pro dodavatele rovněž představují chyby v software rizika neúspěchu celého projektu, zvyšování nákladů či poškození hodnoty společnosti na trhu. Eliminací chyb se tak riziko neúspěchu podstatně snižuje. Lze říci, že zajištění kvality ve vývoji software je velmi podstatné, jak pro zákazníka tak pro dodavatele a je v zájmu obou stran vyvíjet kvalitní software s minimálním počtem chyb.

Práce bude zaměřena na realizaci funkčních testů (FAT - Factory Acceptance Tests) v rámci softwarového projektu v oblasti energetiky. Použité metodiky, fáze, návrhy, implementace a postupy v této práci budou tedy zejména v praktické části zaměřeny na takové, které jsou použity v daném projektu.

První kapitola teoretické části popisuje projektové metodiky vývoje software, uvádí rozdíly mezi tradičními a agilními metodikami a podrobněji je uvedena metodika *Rational Unified Process* (RUP). Druhá kapitola je věnována způsobům testování. V této kapitole je rozebráno, podle čeho se určuje správná strategie testování, a jak se testy dělí. Podrobněji je popsán model FURPS+, který představuje východisko pro řízení kvality v metodice RUP. V poslední kapitole jsou obecně popsány funkční testy, kde jsou dále uvedeny testy použité v energetickém projektu. Každý test je následně rozepsán.

Cílem praktické části je seznámit čtenáře s energetickým projektem *Nemo Link* a systémem *Nemo Link Dispatch System* (NDS). Následně bude popsáno testovací prostředí, použité nástroje a navržená metodologie pro testy. Na základě metodologie budou popsány jednotlivé testy a vyhodnoceny, zda-li byly úspěšné nebo neúspěšné.

Vzhledem ke správné korekci česko-anglického překladu a lepší přehlednosti v textu či obrázcích bude práce v některých částech obsahovat anglické pojmy.

# 1 PROJEKTOVÉ METODIKY VÝVOJE SOFTWARE

Vývoj software je složitá a komplikovaná záležitost, a to jak z technického, tak organizačního hlediska. Největší problém v oblasti vývoje software představují jednotlivé procesy vývoje software [1]. Softwarové projekty jsou svým způsobem pokaždé jedinečné z čehož plyne, že projekt není opakující se záležitost a pokaždé pracujeme na něčem jiném. Z tohoto důvodu je obtížné projekt naplánovat, řídit a předvídat jeho vývoj.

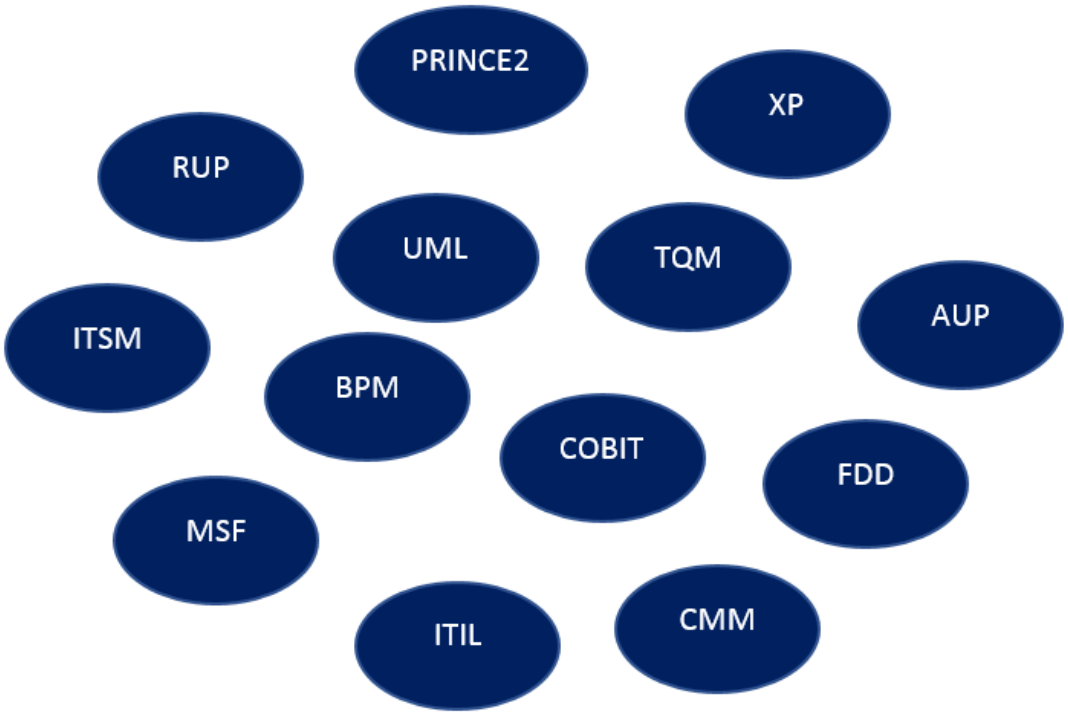
Základem je určitá znalost a používání vhodné metodiky vývoje software, která podstatně ulehčí projektový proces. K zajištění úspěchu celého procesu vývoje je důležité, aby tým zabývající se tvorbou software znal a používal určitou metodiku. Metodika [2] je soubor dohodnutých pravidel, principů a postupů ovlivňujících efektivnost a organizaci práce vedoucí k dosažení určeného cíle, tzn. úspěšného vývoje a dodání software zákazníkovi.

Nemůže existovat pouze jedna univerzální metodika, která by vyhovovala každému typu softwarového projektu a každému vývojovému týmu. Každý projekt je specifický, má jiná omezení, jinou technologii, vývojový tým, partnery atd. Z tohoto důvodu existují desítky různých metodik, viz obr. 1.1, zabývajících se problematikou vývoje software. Některé jsou podobné ostatním metodikám a některé jsou naopak relativně dost odlišné. Je však velice nepravděpodobné, že bychom potřebovali využít metodiku v celém rozsahu (obvykle se využívá pouze část dané metodiky) a žádná z existujících metodik nebude s největší pravděpodobností v původním stavu vyhovovat přesně našim potřebám. Metodiku je obvykle nutné omezit, upravit, případně rozšířit v závislosti na konkrétních potřebách projektu.

Dříve se používaly přístupy k vývoji software např. vodopádový model, spirálový model, atd., které však pokrývaly pouze určitou část procesu vývoje. V současné době existují metodiky pokrývající veškerý proces vývoje software, které rozlišujeme na tradiční (rigorózní) a agilní.

## 1.1 Inkrementální a iterativní vývoj

Inkrementální a iterativní vývoj [3] je v současné době základem všech moderních metodik vývoje software, jak tradičních, tak i agilních. Iterativní vývoj patří mezi nejlepší praktické způsoby při vývoji software. Nicméně iterativní vývoj je zároveň i inkrementální. Důvodem je, že u větších systémů nejsme schopni vytvořit a dodat najednou kompletní systém – velké projekty je vždy nutné rozdělovat na menší části, tzv. subsystémy. To znamená, že vývoj větších projektů je iterativní a zároveň




Obr. 1.1: Různé metodiky a přístupy (autor) [3]

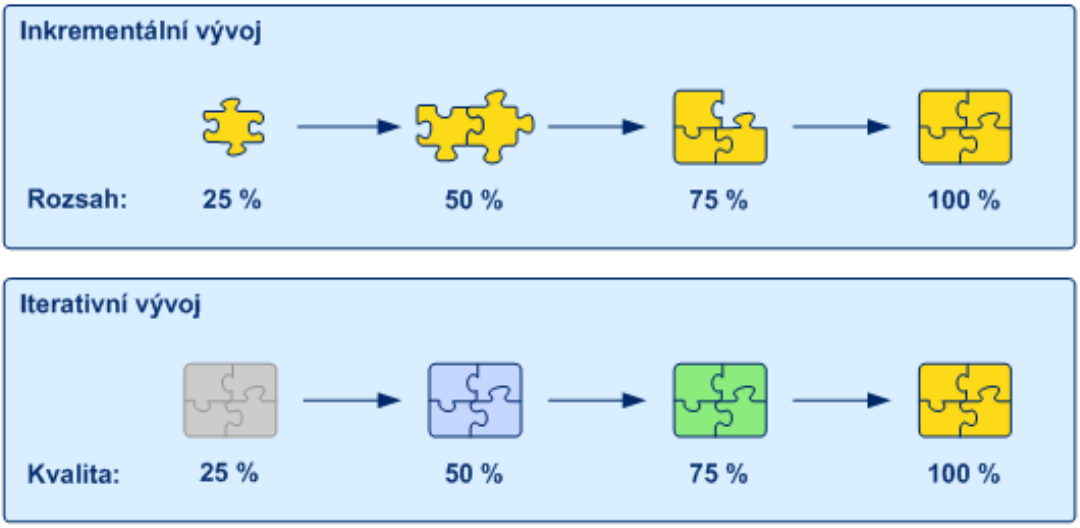
inkrementální.

## Inkrementální

Základním principem je, že postupně dodáváme zákazníkovi jednotlivé části systému, které mají vždy implementovanou veškerou požadovanou funkcionalitu. Dodáváme plně funkční iterace do té doby, dokud není systém kompletní. Celý systém je tedy dodáván po iteracích.

## Iterativní

Iterativní vývoj je založen na jiném principu. Vždy je k dispozici celý systém, ale s omezenou funkcionalitou. Postupně se provádí jednotlivé iterace, než je systém plně funkční. V každé iteraci je vylepšena funkcionalita celého systému. Systém je tedy vždy úplný. Rozdíl obou přístupů je znázorněn na obr. 1.2.




Obr. 1.2: Inkrementální a iterativní vývoj [3]

## 1.2 Tradiční a agilní metodiky

Ať už se jedná o tradiční nebo agilní metodiky, rozdělují se podle následujících kritérií [4], [5]:

- **Zaměření metodiky** - hodnocení, zda se jedná o metodiku v závislosti na jednom projektu nebo budování ICT celé organizace.
- **Rozsah metodiky** - je určen počtem fází životního cyklu informačního systému, které metodika pokrývá.
- **Váha metodiky** - vyjadřuje, jak podrobně se metodika daným tématem zabývá a jak přesně je zpracováno dané téma. Podle váhy se metodiky dělí na lehké a těžké (rigorózní).
- **Typ řešení** - rozlišuje vývoj nového řešení, integrace řešení, rozvoj a rozšíření implementace.
- **Doména** - oblast, pro kterou je systém vytvářen (Business Intelligence, management vztahu se zákazníky, obecný software atd.).
- **Přístup k řešení** - zohledňuje paradigma vývoje (strukturovaný vývoj, rychlý vývoj aplikací, objektový vývoj, komponentový vývoj, vývoj orientovaný na služby).

Na základě kritéria váha metodiky se tedy metodiky rozdělují na „těžké“ tradiční (rigorózní) a „lehké“ agilní.

Jelikož není cílem práce popisovat jednotlivé metodiky, tak následně budou pouze obecně popsány tradiční a agilní metodiky [5], [6], [7] a detailněji bude popsána metodika *Rational Unified Process* (RUP), která je použita v energetickém projektu, kterým se tato práce zabývá.

## Tradiční

Tradiční metodiky vycházejí z přesvědčení, že procesy při budování IS lze popsat, plánovat, řídit a měřit. Hlavní snahou je podrobně a přesně definovat procesy, postupy, fáze a činnosti, a z tohoto důvodu jsou také velmi objemné. Velké množství těchto metodik je založeno na vodopádovém modelu ve kterém jednotlivé fáze vývoje software navazují na sebe. Některé jsou založeny na iterativním vývoji, který rozkládá projekt na řadu iterací, přičemž každá iterace obsahuje všechny fáze vývoje. Metodiky je vhodné aplikovat na tradiční a větší projekty. Mezi nejznámější tradiční metodiky patří:

- *Rational Unified Process* (RUP)
- *Unified Process* (UP)
- *Enterprise Unified Process* (EUP)
- *Microsoft Solutions Framework* (MSF)
- *OPEN/OML*

## Agilní

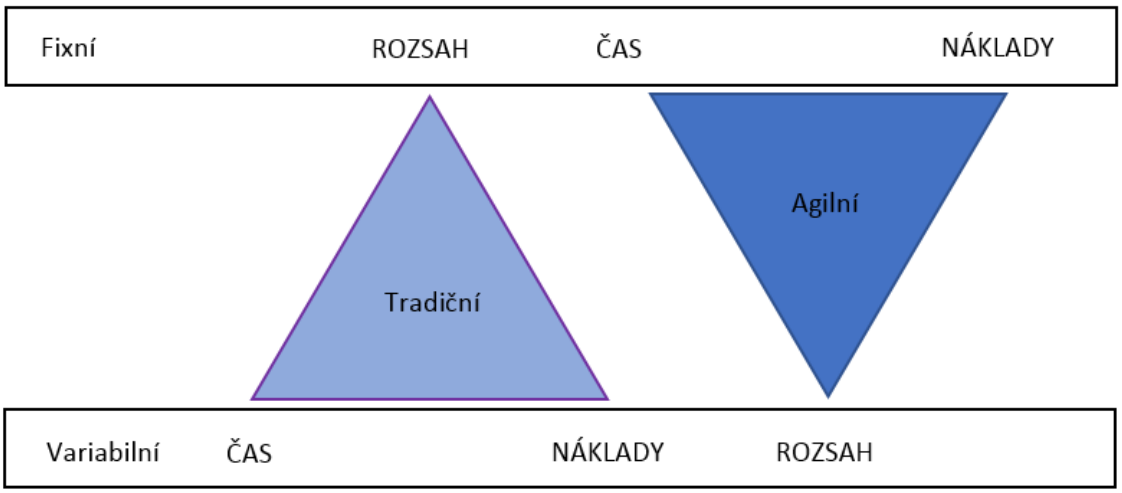
Agilní metodiky jsou opakem tradičních. Vycházejí z přesvědčení, že nejlepší možný způsob k dosažení úspěšné implementace je software co nejrychleji vyvinout, dodat zákazníkovi (i přes fakt, že bude s omezenou funkčností) a na základě zpětné vazby zákazníka software dále upravovat. Velký důraz je zde kladen na co nejkratší iteraci (i jeden den) a neustálou komunikaci se zákazníkem. Zákazník dostává průběžně nové verze software a má vývoj více pod kontrolou a v případě potřeby může pružněji reagovat. Důraz je kladen i na neustálé testování software vzhledem k velmi krátkým iteracím, které produkují velký počet nových verzí. Metodiky je vhodné aplikovat na projekty s nejasným nebo měnícím se zadáním. Mezi nejznámější agilní metodiky patří:

- *Extreme Programming* (XP)
- *Adaptive Software Development* (ASD)
- *Lean Development* (LD)
- *Dynamic System Development Method* (DSDM)
- *SCRUM*

Zásadní rozdíl mezi tradičními a agilními metodikami velmi dobře znázorňuje obr. 1.3, na kterém je vidět význam třech základních proměnných důležitých při vývoji software.

Tradiční přístupy považují za fixní (neměnný) rozsah. Podle rozsahu se poté odhaduje čas a náklady projektu (není to však pravidlo).

Naopak agilní přístupy považují čas a náklady za neměnné. Termín je nutné stanovit na začátku projektu, zdroje jsou pevně dané. Proměnný je zde rozsah, který se mění a přizpůsobuje potřebám zákazníka.




Obr. 1.3: Tradiční a agilní metodiky (autor)

## 1.3 Rational Unified Process

*Rational Unified Process* (RUP) [8], [9], [10], [11] je metodika vývoje software, kterou vytvořila americká společnosti *Rational Software*. V současné době je *Rational Software* součástí společnosti *IBM*. RUP vychází z obecnější metodiky *Unified Process* (UP), která je na rozdíl od RUP volně dostupná. Metodika UP vznikla již v roce 1967.

RUP představuje detailně propracovaný, zdokumentovaný, systematický a organizovaný přístup k vývoji software. Jedná se o poměrně rozsáhlou metodiku založenou na principu iterativního vývoje. Metodika je dodávaná jako rozsáhlá nápověda (znalostní báze) v podobě webových stránek. V podstatě jde o online instruktora, tzn. kdykoliv v průběhu životního cyklu vývoje software můžeme získat potřebné informace a zjistit, jaké kroky budou následovat. Metodika si klade za cíl vývoj kvalitního software, který splňuje požadavky zákazníka, má určitý harmonogram a nepřekročí stanovený rozpočet.

Na RUP je možné se dívat ve třech rovinách [9]:

1. **RUP jako proces softwarového inženýrství** - představuje pohled na RUP jako na systematický návod k přiřazování úkolů a odpovědností v týmech a

organizacích, které se zabývají vývojem software. Cílem je zajistit úspěch softwarových projektů.

2. **RUP jako produkt** - představuje pohled na RUP jako na softwarový produkt, který je v současné době vyvíjený a udržovaný společností *IBM*. Metodika je dodávaná jako rozsáhlá nápověda v podobě webových stránek. RUP je také integrován s dalšími produkty a nástroji pro podporu vývoje software.
3. **RUP jako procesní framework** - představuje pohled na RUP jako na rámec, obsahující předpřipravené procesní modely a návody. Lze jej rozšiřovat a přizpůsobovat tak, aby co nejlépe vyhovoval organizaci, která jej využívá.

Metodika RUP je velice rozsáhlá, protože se snaží pokrýt všechny možné situace vývoje software a popsat veškeré situace, které mohou nastat. Je však prakticky nemožné, aby jedna osoba znala detailně metodiku RUP v celé své šíři. Z těchto důvodů je nutné RUP upravit a přizpůsobit pro konkrétní potřeby projektu. Žádný projekt nemůže využít celý RUP, ale vždy jen jeho určité části. RUP tedy nelze nikdy použít ve své původní podobě, musíme jej vždy implementovat pro naše konkrétní potřeby.

Vybrané varianty RUP podle rozsahu a zaměření:

- *Classic RUP*
- *RUP for Small Projects*
- *RUP for .NET*
- *RUP for Java*

Metodika RUP je vhodná především pro střední a velké projekty s desítkami až stovkami funkčních požadavků, trvající déle než jeden měsíc a pro vývojové týmy s více než deseti členy. Také je vhodná v případě, kdy je projekt náročný po technické anebo organizační stránce.

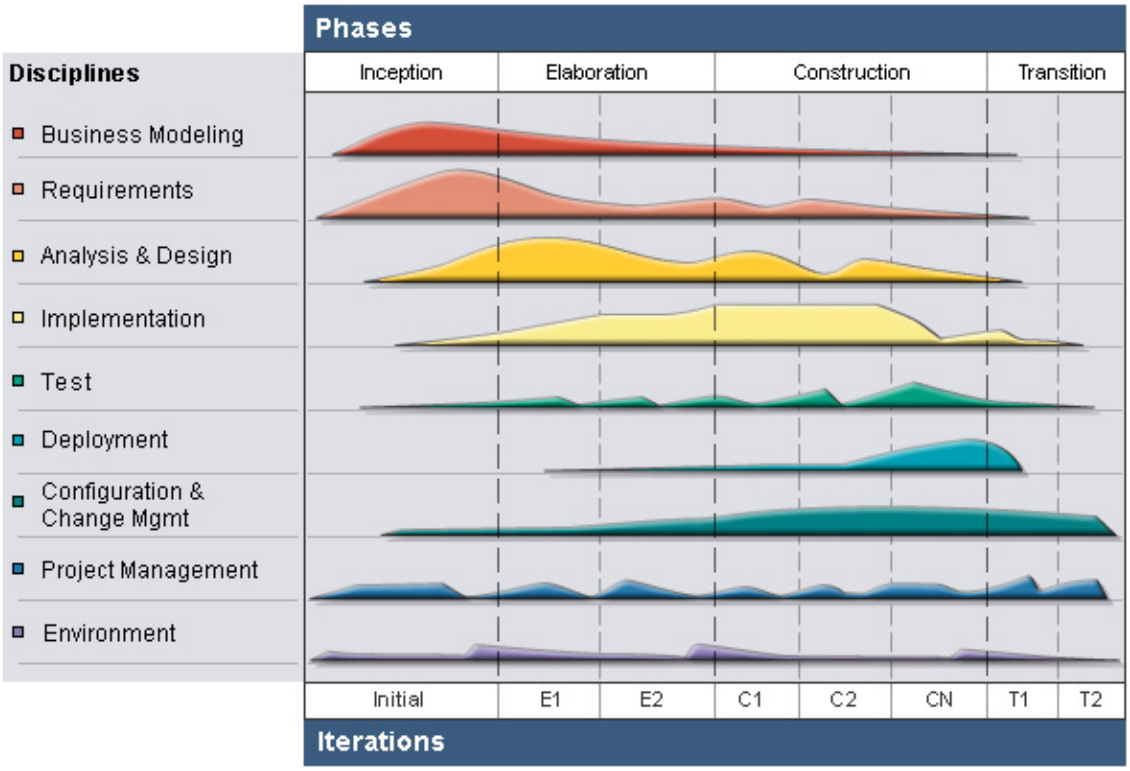
Naopak nevhodné použití metodiky RUP je pro malé projekty s malým počtem členů vývojového týmu. Vzhledem k tomu, že RUP jako produkt je drahý na pořízení, tak vyžaduje velké náklady na vyškolení členů týmu. Dalším důvodem je administrativní složitost, což může práci malého týmu značně brzdit. I přes existenci varianty *RUP for Small Projects* je lepší zvážit použití některé z agilních metodik vzhledem k jejich pružnosti.

V metodice RUP se rozlišují dvě základní dimenze, viz obr. 1.4:

1. **Věcná** - jednotlivé disciplíny (Disciplines)
2. **Časová** - jednotlivé fáze (Phases)

Jednotlivé disciplíny představuje osa Y a fáze osa X. U každé disciplíny je zároveň vidět, jak velké je vynaložené úsilí v různé fázi projektu. Například první dvě disciplíny Business modelování (Business Modeling) a Požadavky (Requirements) se nejvíce provádějí v počáteční (Inception) fázi projektu. Naopak testování a jeho příprava se provádí spíše v pozdních fázích projektu, ale částečně také již v druhé

fázi elaborace. Je však důležité začít s testováním pokud možno co nejdříve. Řízení projektu (Project management) musí zase nutně probíhat po celou dobu projektu(ve všech čtyřech hlavních fázích).




Obr. 1.4: Disciplíny a fáze RUP metodiky [9]

## Disciplíny RUP

V metodice RUP se rozlišují tyto hlavní disciplíny:

- Business modelování (**Business modeling**) - je předřazeno všem disciplínám a provádí se jako první disciplína při zahájení projektu. Účelem je popsání problémové domény zákazníka. Po porozumění problémové oblasti pak následně vyplývají požadavky zákazníka. Hlavním cílem je vytvoření jasné vize projektu, stanovení obchodních cílů a vytvoření business modelu řešené problematiky.
- Požadavky (**Requirements**) - účelem této disciplíny je analýza a správa požadavků zákazníka. Požadavky vycházejí z problémové oblasti a business modelu. Cílem je detailní analýza problémové oblasti, definice systému/požadavků a následně pak řízení všech pozdějších změn.
- Analýza a návrh (**Analysis & Design**) - cílem této disciplíny je transformovat požadavky do návrhu systému. Je potřeba vytvořit mnoho různých modelů,

které popisují budoucí systém z různých pohledů. Detailní návrh je pak podkladem pro následnou implementaci.

- Implementace (**Implementation**) - implementace probíhá přesně podle návrhu z předchozí disciplíny. Představuje transformaci požadavků zákazníka do kódu.
- Testování (**Test**) - účelem této disciplíny je eliminovat pokud možno co nejvíce chyb v software. Jde o vyvinuté úsilí ke zjištění toho, zda systém pracuje tak, jak je to popsáno v jeho specifikaci.
- Dodání (**Deployment**) - cílem je dodání a nasazení software do provozu u zákazníka. Informační systém je nasazen do produkčního prostředí a spuštěn nejprve v pilotním provozu na omezené skupině uživatelů a dat.
- Konfigurační a změnové řízení (**Configuration & Change management**) - cílem této disciplíny je sledovat a udržovat integritu mezi jednotlivými částmi projektu. V průběhu projektu postupně vzniká, mění se a zaniká velké množství meziproduktů. Tyto změny je nutné průběžně sledovat a řídit.
- Řízení projektu (**Project management**) - řízení musí probíhat po celou dobu projektu, samozřejmě není možné připustit, aby projekt nebyl v některé fázi dostatečně řízen. Cílem je řízení lidských zdrojů, fází, iterací, rizik, financí, atd.
- Prostředí (**Environment**) - Pro práci vývojového týmu je nutné zajistit odpovídající prostředí a vhodně zvolit potřebný hardware, software, vývojové prostředí, nástroje, zpřístupnit všechny potřebné zdroje informací atd. Současně bychom měli definovat standardy a vnitřní postupy zejména stylu psaní kódu, komentářů v kódu apod.

## Fáze RUP

Rup definuje čtyři základní fáze vývoje. Každá fáze sleduje určitý hlavní cíl, jednotlivé fáze jsou odděleny milníky (kontrolní bod v rámci daného projektu, kde se ověřuje, jaká část projektu byla dokončena). Základní fáze RUP jsou:

1. Zahájení (**Inception**) - tato fáze definuje účel projektu, rozsah projektu a jeho obchodní kontext. Řeší se zde, co máme vyrobit a zároveň co není předmětem projektu. Cílem je jasné vymezení oblasti, kterou budeme realizovat. Musíme od zákazníka získat popis hlavních a klíčových požadavků na systém. Výstupem je vize, která by měla stanovit základní představu projektu.
2. Projektování (**Elaboration**) - v této fázi probíhá analýza projektu a specifikace architektury software. Řeší se zde, jak máme software vyrobit. Fáze zahrnuje detailní popis informačního systému, vytvoření veškerých potřebných modelů, návrh softwarové architektury, případně prototyp a je zpracován detailní plán a rozpočet implementace. Výstupem této fáze je technický projekt.

3. Realizace (**Construction**) - v konstrukční fázi se zabýváme samotnou tvorbou software. V této fázi je informační systém vyroben přesně podle zadání z elaborační fáze. Realizace musí být rutinní proces, protože jen tak může probíhat podle plánu tzn. v požadované kvalitě, kvantitě, termínu a rozpočtu. Výhodou RUP metodiky je, že na realizační fázi projektu mohou pracovat i méně zkušení vývojáři, protože pracují přesně podle zadání, což je rozdíl oproti jiným metodikám, které vyžadují zkušenější vývojáře.
4. Zavedení (**Transition**) - v této fázi dochází k předání produktu zákazníkovi. Fáze zavedení probíhá v prostředí zákazníka. Informační systém je nasazen do produkčního prostředí a spuštěn nejprve ve zkušebním provozu na omezené skupině uživatelů a dat. Cílem zkušebního provozu je odladit poslední chyby v systému a ověřit jeho kvalitu v reálném provozu. Pokud je provoz úspěšný, můžeme informační systém nasadit do běžného provozu.

## Základní model RUP

Metodika RUP je postavená na čtyřech základních pojmech:

1. **Role** - role určují, kdo a co provádí za činnost. Zástupci těchto rolí jsou zaměstnanci. Jednu roli může postupně zastupovat až několik zaměstnanců, nikoliv však současně a naopak jeden zaměstnanec může být obsazen do různých rolí.
2. **Artefakty** - artefakty určují, co se má dělat. Artefakt představuje výsledek, který je použit jako vstup pro další činnost. Za jeho vytvoření a věcnou správnost je zodpovědná určitá role. Artefakty mohou mít různé podoby (dokument, spustitelnou aplikaci, zdrojový kód atd.) a dělí se do několika skupin (artefakty týkající se testování, implementace, vývojové prostředí, analýzy apod.).
3. **Činnosti** - činnosti určují, jak provádět jednotlivé úkoly. Každou činnost provádí zaměstnanec v dané roli.
4. **Procesy** - procesy určují, kdy provádět jednotlivé úkoly. Jedná se o logicky uspořádanou řadu činností, které vedou k dosažení určeného cíle.

Většina věcí v metodice RUP se odkazují na některé z výše zmíněných pojmů.

# 2 ZPŮSOBY TESTOVÁNÍ

Pro efektivní otestování software je nezbytné, aby dodavatel na začátku projektu stanovil správnou strategii testování, která zahrnuje přístupy a typy použitých testů, v jaké časové míře budou použity, popřípadě jaké nástroje budou využity [12]. Existuje velké množství různých testů a je nutné vybrat takové, které splní účel nejefektivnějšího otestování produktu. V případě nevhodné testovací strategie hrozí, že toto rozhodnutí může mít kritický dopad na cenu produktu nebo termín dodání.

První rozdělení je podle znalosti vnitřní struktury software:
**White box** (Bílá skříňka) testování [13] se vyskytuje také pod pojmy glass box, clear box, open box a předpokládá, že tester zná vnitřní strukturu aplikace. White box vyžaduje znalost vnitřních datových struktur, programových struktur a je nutné znát, jakým způsobem je aplikace implementována. Testerovi je předána veškerá dokumentace, binární i zdrojový kód aplikace. Úkolem testera je porozumět poskytnutému kódu a provést analýzu. White box testování může být provedeno ručně nebo automatizovaně. Pro analýzu existují dva typy nástrojů. První typ nástroje potřebuje zdrojový kód. Druhý typ provede dekompilaci binárního kódu, na jehož základě je automaticky po částech vytvořen zdrojový kód, který je analyzován. White box testování je používáno na začátku vývojového cyklu a je velice vhodné pro webové služby a také k otestování odolnosti proti neautorizovaným přístupům.

Mezi výhody patří včasné odhalování chyb na základě analýzy zdrojového kódu a odhalení nežádoucího kódu na základě znalosti binárního i zdrojového kódu (lze otestovat kód proti nežádoucím operacím).

Mezi nevýhody patří odborná znalost cílového systému, použitých nástrojů a programovacích jazyků. Další nevýhodou jsou vysoké náklady vzhledem k specializovaným nástrojům na analýzu kódu.

**Black box** (Černá skříňka) testování [14] se vyskytuje také pod pojmy opaque box, closed box a realizuje se bez znalosti vnitřní struktury aplikace. Předpokladem je, že tester nemá k dispozici žádné dokumentace, binární ani zdrojové kódy. Black box testování je založeno na základě testovacích scénářů, které mohou být testerovi poskytnuty nebo si je tester musí vytvořit sám. Podle vstupních dat a chování aplikace tester ví, jaké může očekávat výstupy, což vede k porovnání, zda-li byl test úspěšný či nikoliv. Black box testy se provádí ručně nebo automatizovaně pomocí vhodných nástrojů. Tento typ testu se používá u aplikací, kde jsou známé vstupy i výstupy.

Mezi výhody patří snadnost a rychlost vzhledem k tomu, že tester nemusí znát programovací jazyky. Další výhody představují např. transparentnost (test je pro

zákazníka srozumitelný), testovací scénáře mohou být napsány až po dokončení specifikace software, při změně programovacího jazyka nebo operačního systému není nutné měnit testovací scénáře atd.

Mezi nevýhody patří nižší kvalita kódu a nežádoucí chování aplikace, jelikož se netestuje kód ani jeho nežádoucí chování.

**Grey box** (Šedá skříňka) testování [15] se vyskytuje také pod pojmem translucent box, a předpokládá omezenou znalost interních datových a programových struktur k vytvoření vhodných testovacích scénářů. Principem je kombinace white box a black box testování. Nejde však ani o black box vzhledem k částečným znalostem testera vnitřní struktury, ale ani o white box, protože tester nemá podrobnou znalost vnitřní struktury. Koncept grey box testování je velice jednoduchý. Test je prováděn zvenku stejně jako u black box testu s tím rozdílem, že tester ví, jak jednotlivé komponenty aplikace fungují uvnitř a může tak navrhnout efektivní testovací scénáře. Způsobů využití grey box testu je mnoho. Typickým příkladem je použití white box testu k nalezení zranitelností a následnému využití black box testu ke zjištění, zda-li by bylo možné nalezené zranitelnosti využít k útoku.

Mezi hlavní výhodu patří kombinace výhod black box a white box testování. Další výhodou je, že tester nepotřebuje přístup k binárnímu ani zdrojovému kódu (stačí znalost funkční specifikace, rozhraní a architekturu aplikace).

Mezi nevýhody patří nemožnost otestovat veškeré datové toky a efektivnost binárního a zdrojového kódu.

Podle toho, zda je nutné k testování spuštění aplikace:
**Statické testy** [16], [17] nevyžadují běh aplikace, tudíž je možné je využívat v prvních fázích životního cyklu software, kdy je vytvořen první prototyp aplikace nebo před začátkem psaní kódu pro kontrolu specifikace požadavků. Využití nachází na složitě otestovatelné komponenty, u kterých by testeři museli složitě vytvářet dynamické cesty, což by vedlo k nedostačující kvalitě testování nebo by ji muselo být dosaženo na úkor vyšších nákladů. Statické testování je efektivní např. při procházení kódu a hledání určitých chyb.

**Dynamické testy** [16], [17] vyžadují běh aplikace a pro použití je potřeba mít spustitelnou verzi aplikace. Používají se v pozdějších fázích životního cyklu software a principem je vkládání různých vstupů, na jejichž základě se posuzují výstupy testované aplikace.

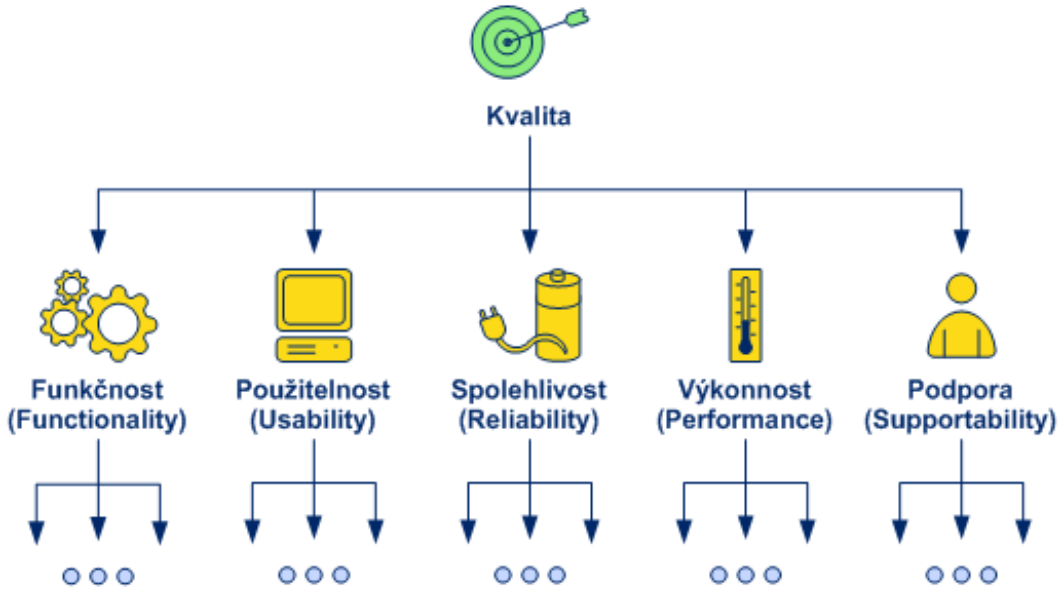
Podle provádění testů:

**Manuální** testování [18], [19] je prováděno „ručně“. Je vhodné ho používat v případech, kdy test vyžaduje lidský úsudek, popřípadě různé přístupy, které nemusíme zaznamenávat a pravidelně opakovat. K testování není potřeba žádný specializovaný nástroj, pouze daná aplikace. Za výhody lze tedy považovat jednoduchost, cenu (absence drahých nástrojů) a možnost testovat manuálně danou aplikaci méně zkušenými pracovníky. Mezi nevýhody patří časová náročnost, cena rostoucí lineárně s počtem testerů a nemožnost otestovat větší množství variant.

**Automatizované** testování [19], [20] spočívá v zaznamenání nebo vyvolání požadované činnosti v testované aplikaci pomocí určitého nástroje. Není zde nutný zásah testera. Zásah je nutný pouze v počáteční fázi a při tvorbě automatického testu a případně při jeho vyhodnocení. Samotný test probíhá automaticky. Mezi výhody patří nízké náklady na samotný provoz, jelikož veškeré testování obstarává samotný nástroj bez nutnosti zásahu testera. Dalšími výhodami je eliminace lidských chyb, které vznikají důsledkem lidské nepozornosti, možnost testovat velké množství vstupů a výstupů, opakovatelnost testů atd. Největší nevýhodou jsou vysoké počáteční náklady, časová a organizační náročnost celého řešení a potřeba zkušených specialistů.

Podle dimenze kvality modelu FURPS+:
**FURPS+** model [21], [22] je východiskem pro řízení kvality v metodice RUP. Model zahrnuje pět hlavních dimenzí kvality (funkcionálních), viz obr. 2.1, které lze dále rozdělit na další konkretní typy testů a libovolný počet vedlejších dimenzí (nefunkcionálních). Tyto dimenze by měly být brány v potaz při vývoji i testování software. Mezi dimenze patří:

1. Funkčnost (**F**unctionality)
2. Použitelnost (**U**sability)
3. Spolehlivost (**R**eliability)
4. Výkon (**P**erformance)
5. Podpora (**S**upportability)
6. Ostatní dimenze (**+**)




Obr. 2.1: Hlavní dimenze modelu FURPS+ [19]

## Funkčnost

Kategorie, do které patří testy zaměřené na testování funkčních požadavků:

- **Funkce** - účelem testu je ověření správné funkce celého systému i jeho dílčích částí.
- **Bezpečnost** - účelem testu je ověření, zda-li je systém dostatečně zabezpečen proti neoprávněnému použití.
- **Kapacita** - účelem testu je ověření, že systém je schopen obsluhovat velké množství vstupů a generovat velké množství výstupů, obsluhovat databázi atd.

## Použitelnost

Kategorie, do které jsou zahrnuty testy použitelnosti systému:

- **Uživatelské rozhraní** - test je určen ke kontrole konzistence uživatelského rozhraní (GUI), online nápovědy, uživatelské dokumentace a další materiály, které může běžný uživatel systému využít.
- **Uživatelská dokumentace a školící materiály** - test je určen ke zkoumání školících materiálů a uživatelské dokumentace, zda-li je jejich obsah kompletní a konzistentní podle požadavků, zda-li neobsahují věci, které by obsahovat neměly, dále je potřeba ověřit správnou terminologii atd. Nejefektivnějším způsobem testování uživatelské dokumentace je zapojení do testování kromě testerů i samotné uživatele.

## Spolehlivost

Kategorie, do které patří testy týkající se spolehlivosti systému:

- **Náchylnost k chybám** - účelem testu je ověření na jednotlivých částech systému odolnost proti selhání. Pro ověření se používají zátěžové testy (Stress tests), které mají za úkol ověřit, jak se bude systém chovat při extrémních podmínkách např. snížení operační paměti aplikačního serveru, omezené sdílení HW i SW prostředků, velké množství najednou připojených uživatelů apod.
- **Zotavitelnost systému** - účelem testu je ověření, jak se bude systém chovat v situaci, kdy dojde k jeho „spadnutí“ a je potřeba ho obnovit. Zálohování a případné obnovování dat je jedna z velmi důležitých činností systému, a proto by se neměl tento test opomíjet.
- **Předvídatelnost** - účelem testu je předvídat určité neobvyklé situace v chování systému a tyto situace simulovat a ověřovat, jaký dopad budou mít na systém.
- **Přesnost** - účelem testu je ověření, zda systém na základě daných vstupů a prováděných operací poskytuje přesné výstupy.

## Výkon

Kategorie, do které patří testy týkající se výkonnosti systému:

- **Zátěžový test** - jedná se o test, kde se sleduje doba odezvy systému při různé úrovni zátěže, popřípadě při různých konfiguracích. Pokud zákazník stanovil určité limity, tak musí být dodrženy.
- **Rychlost** - účelem testu je ověření rychlosti systému, která se nejčastěji měří pomocí tzv. srovnávacích testů (Benchmark tests). Principem testu je mít k dispozici dva systémy v podobných hardwarových nebo softwarových podmínkách, které se následně porovnávají.
- **Dostupnost** - v současné době z praktického hlediska neexistuje systém, který by nepřetržitě zachoval funkčnost po celou dobu své životnosti. Je nezbytné u každého systému předvídat výpadky způsobené závadami, zálohováním, údržbou atd. Každý systém je určen pro jiné nároky z hlediska dostupnosti a proto je nutné, aby byly tyto nároky vždy předem specifikovány.
- **Doba odezvy** - vlastnost systému, která je úzce spojena s jeho rychlostí. Pro optimalizaci je nutné sledovat dobu odezvy výkonnostních profilů systému (definice časů potřebné k vykonání určité činnosti). Optimalizací by se mělo začínat od nejpomalejších částí systému, protože jejich optimalizací se docílí největšího zrychlení systému.
- **Používání zdrojů** - další dopad na výkon systému má sdílení zdrojů, nejvíce datových. Optimalizaci sdílených zdrojů by se měla věnovat patřičná po-

zornost, vzhledem k tomu, že v některých situacích mohou velmi ovlivňovat výkonnost systému. Data, ke kterým se přistupuje současně, je vhodné simulovat automatizovaně pomocí testovacího nástroje, kdy v danou dobu provádí simulaci více uživatelů přistupujících ke stejným datům.

## Podporovatelnost

Kategorie, do které patří testy týkající se podpory systému:

- **Kompatibilita** - účelem testu je ověření kompatibility systému s různým software nebo hardware.
- **Rozšířitelnost** - rozšířitelnost je daná použitou architekturou. V závislosti na architektuře je možné předurčit, jak snadné bude rozšiřovat systém v budoucnosti. Je velmi důležité již na začátku projektu zvážit, zda bude potřeba v budoucnosti systém rozšiřovat či nikoliv.
- **Přizpůsobivost** - účelem testu je ověření, jak se bude systém chovat při změně prostředí, databáze, operačního systému atd.
- **Konfigurovatelnost** - účelem testu je ověřit, zda systém bude fungovat i na jiných softwarových nebo hardwarových konfiguracích, než jaké jsou nastaveny v současném testovacím prostředí.
- **Instalovatelnost** - test ověřuje korektní instalace na různé softwarové nebo hardwarové konfigurace, testy kompletní instalace, testy aktualizací apod.
- **Lokalizovatelnost** - v současné době je většina systémů vyvíjena pro mezinárodní trh s různými jazykovými prostředky. Lokalizovatelnost by měla být brána v potaz již na začátku projektu v závislosti na zákazníkovi. Při testování je nutné ověřit všechny překlady z hlediska jazykové správnosti včetně správného zobrazování.

## Ostatní dimenze

Kromě pěti hlavních dimenzí kvality je nutné brát do úvahy i ty, které nejsou viditelné na výsledném produktu. Tyto dimenze jsou označeny znakem „+“ a jde o takové, které jsou především důležité ve fázi vývoje produktu a jeho dalších úpravách. Patří mezi ně např. různé standarty při psaní kódu, architektura databáze atd.

# 3 FUNKČNÍ TESTY

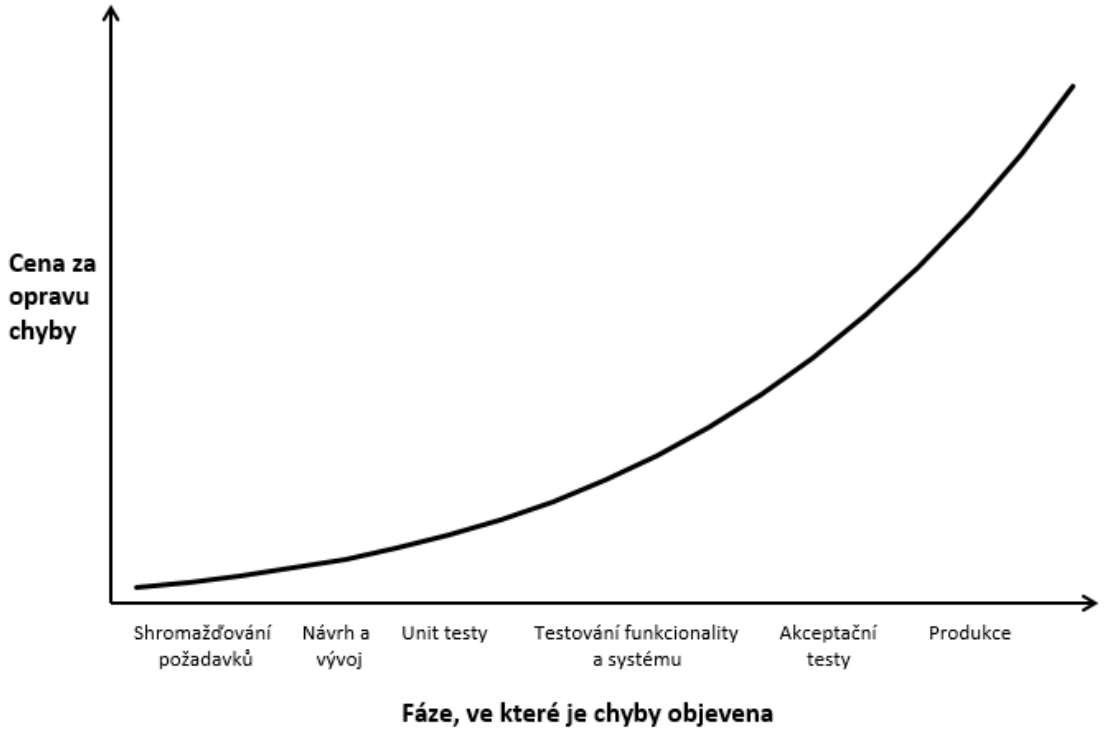
Funkční testy (FAT - Factory Acceptance Tests) [23], [24], [25] jsou testy prováděné na straně dodavatele a vykonávány jsou převážně funkční testy. Během této fáze nemusí být software testován na plně integrovaném prostředí. FAT jsou testy mezi fázemi unitního a systémového testování. Hlavním cílem je splnění akceptačních kritérií, které stanovuje a schvaluje zákazník a jde o nutnou podmínku pro vstup do následující fáze systémových testů (SIT - Site Integration Tests).

FAT jsou často prováděny podle připravených testovacích scénářů simulující situace, které mohou reálně nastat. Testovací scénáře má za úkol připravit testovací tým a jsou spouštěny na testovacím prostředí na straně dodavatele. Jednotlivé scénáře jsou vykonávány v několika kolech. Případné nalezené chyby jsou reportovány vývojovému týmu, který dané chyby opraví a v dalších kolech dochází k opětovnému testování. Chyby se dělí podle závažnosti (severity) na tři typy:

1. **Kritická typu A** - chyba, kvůli které nelze pokračovat podle testovacího scénáře (selhání systému, ztráta dat, úbytek paměti atd.).
2. **Hlavní typu B** - chyba, kterou lze vyřešit v průběhu testovacího scénáře.
3. **Drobná typu C** - chyba, která nemá vliv na správné chování systému (může se jednat o kosmetickou chybu jako např. nesprávná slova nebo nesprávně zarovnaný text apod.).

Testování aplikace probíhá tak dlouho, dokud nejsou splněné stanovené akceptační kritéria např. kritické chyby typu A mohou být pouze dvě, hlavních chyb typu B dvacet a drobných chyb typu C třicet pět. Je velice důležité, aby se ve fázi FAT testů našlo pokud možno co nejvíce chyb, které mají zásadní dopad na chování nebo funkčnost systému. V případě, že se nějaká závažná chyba opomene nebo nenajde, může to mít v pozdějších fázích testování fatální dopad na výslednou cenu a úspěch celého projektu, viz obr. 3.1.

Graf znázorňuje ve vertikálním směru cenu za opravu chyby a v horizontálním fázi, ve které je chyba objevena. Pokud je chyba objevena ve fázích shromažďování požadavků, návrhu, vývoje a unit testů, nejedná se o závažný problém, jelikož produkt je ještě ve fázi vývoje a oprava chyby bude nízká. V dalších fázích testování funkcionality a systému a akceptačních testů se jedná o závažnější problém vzhledem k tomu, že chyba se musí reportovat vývojovému týmu, kde musí být opravena a znovu testována ve všech fázích až do té, ve které byla nalezena. Poslední a nejzávažnější fází je produkce, kdy je produkt už v plném provozu u zákazníka. V případě nalezení závažné chyby se cena opravy může zvýšit až na stonásobek, což může vést k neúspěchu celého projektu.




Obr. 3.1: Znázornění ceny opravy chyby vztahující se k fázi nalezení chyby (autor) [26]

Testů, které mohou být ve FAT zahrnuty, je mnoho. Není však pevně dané, jaké testy musí obsahovat a každá společnost si sama určuje typy testů, obsažených ve FAT. Vzhledem k energetickému projektu, kterým se tato práce zabývá, budou následně obecně popsány testy, které byly stanovené v projektové dokumentace testovací strategie oboru FAT pro otestování vyvíjené aplikace. Jedná se o tyto testy:

- **GUI test**
- **Test autentizace a autorizace**
- **Test stability**
- **Test výkonu**
- **Stress test**
- **Test rozhraní**
- **Bezpečnostní test**

## 3.1 GUI test

GUI testování [27], [28] je proces, zajišťující správnou funkčnost grafického uživatelského rozhraní (GUI) pro danou aplikaci a ujištění, že je v souladu s jejími specifikacemi. Kromě ověření, zda aplikace funguje tak, jak má, se vyhodnocují jednotlivé prvky návrhu, např. rozvržení, barvy, písma, velikost písma, popisky, textová pole, formátování textu, tlačítka, ikony, odkazy apod. Procesy testování GUI mohou být buď automatické nebo manuální, je za ně odpovědný tester. Často jsou prováděny spíše třetími společnostmi (společnosti, které jsou nepřímo zapojeny) než vývojáři nebo koncovými uživateli, ale není to pravidlo a vše závisí na dohodě zákazníka s dodavatelem software, v jaké fázi začnou GUI testy.

Testování GUI může vyžadovat spoustu programování a je časově náročné, ať už manuální nebo automatické. Obvykle analytik napíše dokumentaci ohledně jednotlivých funkcí uživatelského grafického rozhraní, tudíž poté tester bude přesně vědět, jaké výsledky má očekávat. Testování GUI má také testovat určitá chování programu, které uživatelé očekávají např. zobrazení příslušné ikony, kdy je program zaneprázdněn, vyvolání nápovědy tlačítkem F1 a mnoha dalšími běžnými detaily.

Důležité je si uvědomit, že zákazník ke GUI přistupuje jinak než vývojáři. Vývojáři vnímají GUI jako prostředek k reprezentaci jejich práce a logika aplikace je schovaná, avšak u zákazníka to je první věc, kterou vidí. V případě, že si bude zákazník organizovat svoje testy aplikace před akceptací, pak se budou první hlášené chyby týkat právě GUI a bude jich mnoho. V praxi je poté také běžné, že zákazník vnímá hodnocení závažnosti chyb zcela jinak než dodavatel, což vede k tomu, že GUI chyby budou mít zákazníkem nastavenou nejvyšší prioritu. Je nezbytné, aby měl vývojový tým na paměti, že není vhodné odkládat otázky, týkající se GUI a je třeba je probírat se zákazníkem v průběhu celého vývoje. Výsledkem bude méně chyb GUI a tím i lepší dojem zákazníka, když poprvé spustí aplikaci.

## 3.2 Test autentizace a autorizace

Test autentizace [29] se provádí za účelem potvrzení někoho (nebo něčeho) jako autentického. Ověřování objektu znamená potvrzení jeho původu, zatímco autentizace osoby spočívá v ověření její identity. Autentizace závisí na jednom nebo více autentizačních faktorech. Typickým příkladem v oblasti software je přihlašovací proces, kdy uživatel odešle svou digitální identitu k ověření. Při testování autentizace se tester zaměřuje na ochranu uživatelských údajů při přenosu do systému a zpět, ověření, zda je možné použít nástroje hrubé síly („Brute-force“), použití různých metod, které se pokoušejí obejít autentizační schéma, testování zranitelných hesel a jejich resetování atd.

V případě autorizace se jedná o koncept umožňující přístup k prostředkům pouze osobám, které mají oprávněný přístup. Testování autorizace, znamená porozumění, jak autorizační proces funguje a použití těchto informací k obcházení tohoto procesu. Autorizace je proces, který nastává po úspěšné autentizaci, tedy tester autorizaci ověřuje až po obdržení příslušných platných pověření, spojené s definovanou sadou rolí a výsad. Během tohoto testu by mělo být ověřeno, zda je možné obejít autorizační schéma a najít zranitelnost nebo způsob, jak zvýšit oprávnění přiřazená testeru.

## 3.3 Test stability

Test stability [30] je nefunkční typ testu, který obvykle spadá pod výkonnostní testování. Hlavním zaměřením je určit stabilitu softwaru, když je vystaven běžnému zatížení, které je poté v produkci očekávané. Je také nazýváno jako „soak“ testování a je to časové testování, kde se výkon měří v průběhu času. Zátěž je aplikována na testovaný systém po dlouhou dobu, což pomáhá určit, jak je stabilní. Součástí stability testu je také často tzv. „stress“ test, který neověřuje normální chování, ale bod, kdy dojde k zahlcení softwaru, který je znám jako zlomový bod systému.

Pokud není splněn test stability testovaného systému, může poté docházet k určitým nedostatkům, např. zpomalování systému s velkým množstvím dat, bezdůvodné padání systému, výkonnostní výkyvy, abnormální chování systému apod. Abychom zmíněným problémům předešli, je nutné začít s testováním stability včas, vzhledem k tomu, že se jedná o testování založené na čase, je důležité znát časový limit požadovaný pro provedení celého testovacího procesu, a nepřekročit tak termín testování.

Úspěšné provádění testování stability testovaného systému vede k daným výhodám jako např. určení stability a robustnosti systému při zatížení, dodání důvěry výkonnosti systému, lepšímu uživatelskému zážitku apod. Testování lze provést pomocí automatizovaných nástrojů nebo ručně. Testování stability vyžaduje správné testovací prostředí s potřebnými nástroji a strukturovaným přístupem pro vysokou efektivitu. Pokud dojde k selhání systému při testování, doba potřebná k obnovení z havárie také určuje výkon, jak zacházet s negativními scénáři.

## 3.4 Stress test

Stress test [32] je proces, při kterém se určuje spolehlivost a stabilita systému za podmínek extrémní zátěže. Test pomůže také stanovit robustnost systému, a jak se systém chová v případě chyb, které vznikají extrémní zátěží. Nejdůležitějším

použitím stresového testování je stanovení limitu, při kterém nejsou splněny určité podmínky nebo systém selže.

U větších projektů může být zátěžové testování časově náročné a zdlouhavé. Je vhodné stanovit metriky, na které se tester zaměří při zkoumání výsledků např. kolik webových stránek za sekund bylo vyžádáno, propustnost dat, odezva aplikace, průměrný čas na načtení webové stránky, průměrný čas k načtení všech informací na webové stránce atd. Tento typ testu se velmi často provádí ve spojení s výkonnostním testováním.

## 3.5 Test výkonu

Test výkonu [31] je proces, který slouží k určení výkonu systému podle hlavních měřítek, validace nebo ověření kvality atributů systému, jako je odezva, rychlost, škálovatelnost a stabilita za různých podmínek zatížení. Systém je testován v závislosti na různých podmínkách zatížení a je kontrolován čas, který systém vyžaduje pro zpracování požadavků při různém pracovním zatížení. Cílem testování výkonu je nejen nalezení chyb v systému, ale také odstranění překážek výkonu ze systému. Testování výkonu se často provádí ve spojení se stresovými testy, zátěžovými testy, porovnávacími testy atd.

Pokud software předáme zákazníkovi bez provedení testování výkonu, může to způsobit problémy jako zpomalování systému při současném přístupu více uživatelů, špatná použitelnost, která pravděpodobně povede ke špatné pověsti dodavatele a přímo ovlivní očekávaný prodejní cíl. Testování výkonu zahrnuje celou řadu různých testů, které umožňují analýzu různých aspektů systému. Testování výkonu udává, co je potřeba opravit před předáním software zákazníkovi (především problémy, které se vyskytují v různých podmínkách zatížení). Testování se provádí pomocí výkonnostních nástrojů (Apache Jmeter, Load Runner, Web Load atd.).

## 3.6 Test rozhraní

Test rozhraní [33] je proces, který ověřuje, zda je komunikace mezi dvěma různými softwarovými systémy prováděna správně. Spojení, které integruje dvě součásti, se nazývá rozhraní. Testování těchto spojovacích služeb nebo rozhraní je označováno jako testování rozhraní.

Rozhraní je ve skutečnosti software, který se skládá ze sad příkazů, zpráv a dalších atributů, které umožňují komunikaci mezi zařízením a uživatelem. Testování rozhraní zahrnuje testování rozhraní webového serveru a aplikačního serveru a

rozhraní aplikačního serveru a databázového serveru. Testuje se, zda jsou servery nastavené správně, zda se chyby zpracovávají správně nebo se vracejí chybové zprávy pro všechny dotazy, které aplikace provedou, kontrola následků, pokud dojde ke ztrátě spojení s web serverem atd.

Testování rozhraní se provádí, aby se zajistilo, že koncoví uživatelé nebo zákazníci při používání určitého softwarového produktu neměli žádné potíže. Cílem je určení, které oblasti aplikace jsou zpravidla používány koncovými uživateli a zkontrolovat jejich uživatelskou přívětivost, ověření požadavků na zabezpečení během komunikace mezi systémy a zkontrolovat, zda je řešení schopné zvládnout selhání sítě mezi aplikačním serverem a webovou stránkou.

## 3.7 Bezpečnostní test

Bezpečnostní (penetrační) test [34] je proces určený k odhalení zranitelností informačního systému a zjištění, zda jsou data a zdroje dostatečně chráněny před případnými útočníky. Rozsah tohoto testu se liší v závislosti na požadavcích zákazníka, přidělených prostředcích (čas, finance, personál) a zkušenostech testera.

Existují 4 hlavní oblasti, které je třeba zvážit při testování bezpečnosti zejména u webových aplikací:

1. **Zabezpečení sítě** - zahrnuje hledání zranitelných míst v oblasti síťové infrastruktury.
2. **Zabezpečení systémového software** - jedná se o posouzení bezpečnosti především v oblasti operačního/databázového systému, popřípadě dalších systémů, se kterými aplikace pracuje.
3. **Bezpečnost na straně klienta** - zabývá se tím, že klienta (např. webový prohlížeč) není možné jakkoliv zmanipulovat.
4. **Bezpečnost na straně serveru** - zabývá se tím, aby kód serveru a technologie, které využívá, byly dostatečně robustní proti jakémukoliv narušení.

Nejzávažnější zranitelnosti v oblasti webových aplikací zveřejňuje každé tři roky bezpečnostní organizace *OWASP* (The Open Web Application Security Project). Mezi nejrizikovější zranitelnosti patří např. SQL injektování, chybná autentizace a správa relace, cross-site skriptování, nezabezpečené přímé odkazy na objekty, nezabezpečená konfigurace atd. Na tyto by se tester měl nejvíce zaměřit. Možností je použití i skenovacích nástrojů zranitelností (OWASP ZAP, Nessus, Vega, WebScarab apod.). Jedná se o automatizované nástroje, které analyzují cílovou aplikaci a dokáží detekovat případné zranitelnosti a nedostatky.

# 4 PRAKTICKÁ ČÁST

Cílem této kapitoly je představení energetického projektu *Nemo Link* a vyvíjeného systému *Nemo Link Dispatch System* (NDS), na který budou následně vykonány tyto testy:

- **GUI/Autorizační test**
- **Výkonnostní test**
- **Zátěžový, stability a stress test**
- **Bezpečnostní test**

Jednotlivé testy budou mít strukturu vycházející z vlastního návrhu metodologie, která je popsána v kapitole 4.4. Jejich realizace bude na testovacím prostředí prostřednictvím nástroje Apache Jmeter, kromě bezpečnostního testu, kde byl zvolen přístup skrze nástroje specializované na skenování zranitelností. Většina popisovaných funkcí a sekcí bude převážně čerpána z interních dokumentací projektu NDS.

## 4.1 Nemo Link

*Nemo Link* [35] je název projektu, který začal v roce 2015. Cílem je položení podmořského vysokonapěťového elektrického kabelu (HVDC - High Voltage Direct Current) a zvýšení kapacity přenosu elektrické energie mezi Velkou Británií a Evropou. Jedná se o společný projekt mezi společností *National Grid Interconnector Limited* a belgickou skupinou *Elia*.

*Nemo Link Dispatch System* je systém, který vyvíjí česká společnost *Unicorn Systems*, a který bude mít na starosti řízení přenosu energie po kabelu. Systém budou používat operátoři na dvou místech. Jedná se o místo v Belgii Herdersbrug (Elia) a Richborough (NGET) ve Velké Británii. Operátoři budou zodpovědní nejen za systém NDS, ale i za jiné provozní systémy a postupy související s provozem propojení *Nemo Link*. NDS bude na jednom místě aktivní, zatímco na druhém bude určené jako záložní v případě výpadku systému prvního.

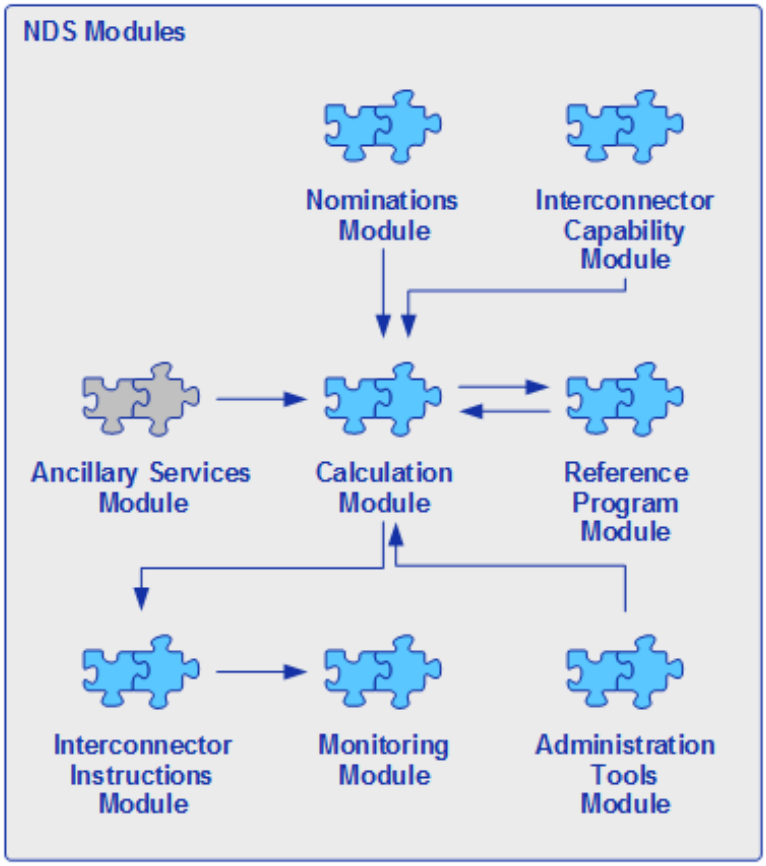
## 4.2 Moduly systému NDS

Systém NDS se skládá celkem z osmi modulů. Každý model má za úkol zajištění nezbytných funkcí k provozu NDS. Jednotlivé moduly a vazby mezi nimi jsou na obr. 4.1. Funkce každého modulu jsou následující:

1. **Nominations Module** - modul má úkol přijetí souboru obchodního plánu (Commercial Schedule) od regionální platformy (RNP - Regional Nomination Platform) obsahující agregované nominace (hodnoty udávající, kolik energie se

přenese v danou hodinu a v jakém směru) na daný i následující den. Modul také umožňuje operátorovi manuálně definovat obchodní plán (v případě výpadku mezi NDS a RNP), provedení validací u každého příchozího/definovaného plánu a uložení těchto souborů do systému.

2. **Interconnector Capability Module** - modul má za úkol generování a posílání souborů (Interconnector Capability) na obě provozní místa systému




Obr. 4.1: Vnitřní moduly a vazby systému NDS

NDS (NGET, Elia) a nominační platformě RNP, obsahující informace o momentální maximální přenosové kapacitě kabelu, popřípadě její redukce. Dále je modul zodpovědný za posílání informací o plánovaných/neplánovaných výpadcích, včetně provedení nutných validací a ukládání vygenerovaných souborů do systému.

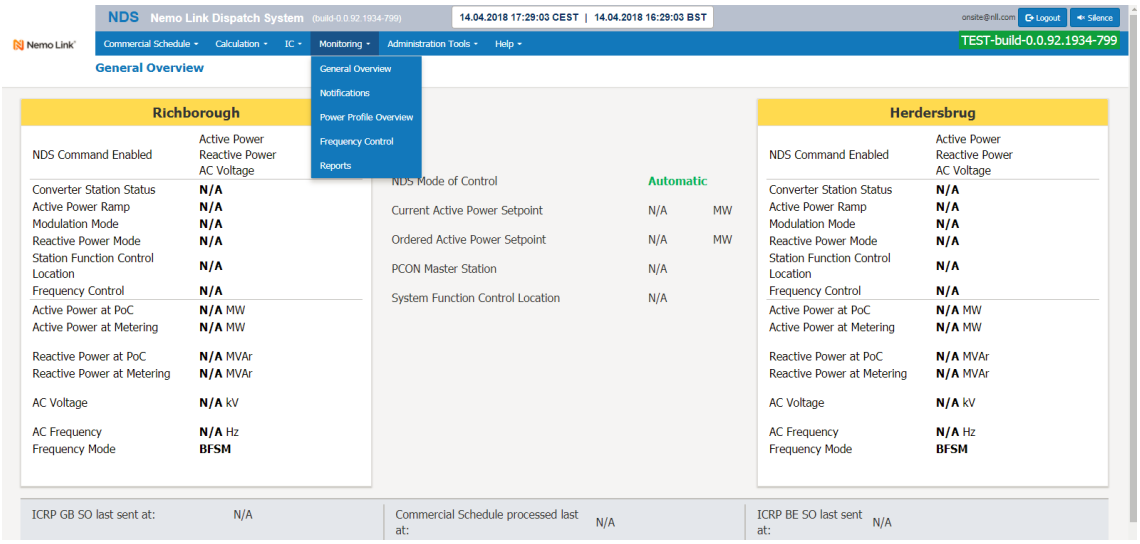
3. **Ancillary Services Module** - modul ovládající reaktivní energii a související doplňkové služby s úložištěm, validacemi, konfigurací a konfiguračními parametry.
4. **Calculation Module** - modul má za úkol převedení nominací ze souboru obchodního plánu na fyzický plán (Physical Schedule) a generování energetických

objednávek (Power Orders), obsahující energetické úrovně a směr přenosu energie pro určité naplánované hodiny. Mezi další funkce patří validace a uložení souborů energetických objednávek do systému.

5. **Reference Program Module** - hlavním vstupem jsou energetické objednávky generované „Calculation“ modulem. Výstupem tohoto modulu je převedení objednávek na finální soubor obsahující konkrétní hodnoty energie, které se budou v daných časech převádět po kabelu. Tento soubor je zaslán pro operátory v NGET a Elia. Dále je modul zodpovědný také za zpracování případných modifikací od operátorů, v případě potřebné změny množství přenesené energie po kabelu.
6. **Interconnector Instructions Module** - modul je zodpovědný za zprostředkování komunikace s vysokonapěťovým stejnosměrným řídícím ochranným systémem (HVDC C&P - High Voltage Direct Current Control and Protection) a integraci s komunikačním rozhraním (RCG - Remote Control Gateway) mezi systémy HVDC C&P a NDS. V podstatě hlavní funkcí je vkládání řídících instrukcí při energetických objednávkách a příjem/odesílání signálů pro monitorování stavu systému.
7. **Monitoring Module** - základní funkcí modulu je monitorování a zobrazení provozního stavu operátorům. Zobrazuje systémové notifikace, přehled informací o stavu propojení, všeobecný přehled s číselnými a textovými hodnotami (např. směr toku) a přehled energetického profilu (grafické zobrazení obchodního plánu a energetických objednávek).
8. **Administration Tools Module** - modul poskytuje prostředky pro správu systému. Mezi tyto prostředky patří přihlašovací proces uživatele, správa uživatelských účtů, konfigurace systému, řízení módu ovládání (Mode of Control - umožňuje operátorovi přepnutí systému z automatického módu do manuálního a naopak), zobrazení a procházení sledovaných záznamů systému, ukládání souboru do úložiště a manuální nahrávání souborů.

## 4.3 Testovací prostředí a nástroje

Testování bude realizováno v testovacím prostředí, viz obr. 4.2, které je zprovozněno na fyzických serverech ve společnosti *Unicorn*. Aby bylo zajištěno přesné a spolehlivé testování, tak parametry těchto serverů, viz obr. 4.3, jsou zcela totožné se servery, které budou v produkci zprovozněny u zákazníka.




Obr. 4.2: Testovací prostředí NDS

| Parametry | CPU | RAM | HDD |
|---|---|---|---|
| Server | [počet jader] | [GB] | [GB] |
| Aplikační server | 4 | 8 | 30 |
| Databázový server | 4 | 8 | 50 |
| Souborový server | 2 | 2 | 50 |

Obr. 4.3: Parametry serverů

Všechny servery používají operační systém *CentOS Linux*. Dále aplikační server používá webový server *Apache Tomcat*, databázový server používá relační databázi *MariaDB* a souborový server používá protokol pro vzdálený přístup k souborům přes počítačovou síť NFS (Network File System).

Dále testovací prostředí (aplikace NDS) používá architekturu REST (Representational State Transfer) [36], [37]. REST je založen na protokolu HTTP (Hypertext Transfer Protocol), a je orientován datově pro distribuovaná prostředí a určuje, jak se má přistupovat k datům. Rozhraní REST se používá pro jednotný přístup ke zdrojům. Zdrojem jsou myšlena data i stavy aplikace. Veškeré zdroje mají svůj identifikátor URI (Uniform Resource Identifier) a REST obsahuje 4 metody, jak k nim přistupovat (GET, POST, DELETE, PUT). Aplikace NDS spolu s RESTem využívá při výměně dat formát JSON (JavaScript Object Notation).

Důležité je, že rozhraní REST by se mělo udržovat bezstavové. Tím je myšleno, že např. autentizace uživatele by neměla záviset na relaci nebo „cookies“. Požadavky by měly obsahovat údaje, podle kterých REST pozná, kdo je připojen. Aplikace NDS používá pro autentizaci uživatele X-Auth-Token, který poskytuje uživateli autentizační token na základě správnosti údajů při přihlášení. Ten se poté přenáší s každým dalším požadavkem vyvolaným přihlášeným uživatelem. Kromě použité architektury aplikace podporuje webové prohlížeče Google Chrome, Mozilla Firefox, Internet Explorer a Microsoft Edge.

U zákazníka budou nasazena prostředí dvě, z něhož jedno bude určeno pro běžný provoz a druhé jako záložní v případě výpadku prvního.

### Apache JMeter

Apache JMeter [38] je nástroj vyvinutý společností *The Apache Software Foundation* s open-source kódem (volně dostupným). Celá aplikace je napsána v jazyce Java a je určena k zátěžovému testování, testování funkčního chování, výkonnostnímu testování ostatních aplikací a systémů apod. Původně byla navržena pro testování webových aplikací, ale od té doby se rozšířila i na další testovací funkce.

JMeter může být použit pro testování výkonu, jak na statických, tak na dynamických zdrojích a webových dynamických aplikacích. Může být použit k simulaci velké zátěže na serveru, skupině serverů, síti nebo objektu, aby bylo možné otestovat jejich sílu nebo analyzovat celkový výkon v různých typech zatížení. Mezi další funkce patří např. schopnost načíst a testovat výkonnost různých typů aplikací, serverů nebo protokolů (LDAP, SMTP, HTTP, TCP, shell skripty, objekty Java atd.), režim příkazového řádku, plně vybavené testovací prostředí IDE (Integrated Development Environment) umožňující rychlé nahrávání testovacího plánu (z prohlížečů nebo nativní aplikace), velkou škálu rozšíření pro ladění apod.

### WebDriver Sampler

WebDriver Sampler [39] je plugin do nástroje Apache Jmeter využívající Selenium a umožňující psaní skriptů/testování aplikací prostřednictvím tří metod (ovládání samotného prohlížeče, výběr webových prvků a pomůcek pro ladění). Klíčové metody jsou get (String), které se používají k načtení nové webové stránky, a různé další metody jako findElement, které se používají k nalezení webových elementů. Tento plugin může být použit pro jednodušší i složitější testování grafického uživatelského prostředí s kombinací výkonnostního testování, popřípadě ověření přístupových práv. Výhodou je, že uživatel při spuštění testu přesně vidí na obrazovce, jak systém reaguje na jednotlivé požadavky, což může vést k lepší identifikaci případných nedostatků.

### Oracle VM VirtualBox

Oracle VM VirtualBox [40], [41] je nástroj určený pro virtualizaci ve dvou licenčních podobách (PUEL, GPL). O vývoj a údržbu se stará organizace *Oracle*. Obsahuje a podporuje mnoho funkcí např. ukládání aktuálních snímků, sdílení složek, 3D virtualizaci, podporu více jazyků apod. Je navržen tak, aby mohl být spuštěn na operačních systémech Windows, Unix/Linux, MAC OS a Solaris.

### Kali Linux

Kali Linux [40], [42] je open-source linuxová distribuce určená pro penetrační testování a bezpečnost webových aplikací. Obsahem této distribuce je přes 300 penetračních nástrojů specializovaných k vyhledání bezpečnostních nedostatků v cílových systémech (Metasploit Framework, SQLmap, OWASP ZAP, Burp Suite atd.). Kali Linux byl vyvinut společností *Offensive Security*, která má na starosti údržbu a financování. Systém je k dispozici v 32/64 bitové verzi nebo ve formě obrazu (armel, armhf) založený na ARM architektuře. Nejčastěji se zprovozňuje prostřednictvím VMware software nebo Oracle VM VirtualBox jako virtuální systém, ale lze ho zprovoznit také pomocí externího zařízení (flash disk) bez instalace nebo instalací na pevný disk.

### Vega Vulnerability Scanner

Vega Vulnerability Scanner [43] je open-source nástroj určený ke skenování a testování bezpečnosti webových aplikací. Je napsán v programovacím jazyce Java, poskytuje grafické uživatelské rozhraní a podporu operační systémy Linux, OS X a Windows. Nástroj umožňuje provádět automatické skenování, obsahovou analýzu, zachycování proxy a je schopen najít případné zranitelnosti (SQL injektování, cross-site skriptování, neúmyslně zveřejněné důvěrné informace atd.).

### OWASP Zed Attack Proxy

OWASP Zed Attack Proxy (ZAP) [40], [44] je open-source nástroj určený pro penetrační testování a hledání případných zranitelností ve webových aplikacích. Jde o jeden z nejaktivnějších projektů společností *OWASP*, která patří mezi nejznámější organizace věnující se bezpečnosti software. Je napsán v programovacím jazyce Java, aby byla zajištěna funkčnost na libovolném OS a obsahuje různé moduly k automatickému/manuálnímu vyhledání zranitelností (web „crawlers“, pasivní skenery, automatické skenery, proxy server apod.).

### Nessus Vulnerability Scanner

Nessus Vulnerability Scanner [40], [45] je celosvětové nejrozšířenější skener zranitel-

ností. Byl vyvinut společností *Tenable Network Security* a je navržen tak, aby byl podporovaný různými OS a platformami. Pracuje na modelu klient-server, z čehož plyne, že uživatel se prostřednictvím webového prohlížeče může k serveru připojit z libovolné stanice v síti. Nástroj je zaměřen na vyhledání zranitelností jako např. nesprávná konfigurace v podobě chybějících aktualizací, výchozí hesla, běžná hesla, účty s chybějícími hesly, odmítání služeb (Denial of Service) proti TCP/IP apod. Testování je založeno na principu kontroly konkrétních chyb pomocí pluginů, které mají velmi rozsáhlé databáze a jsou denně aktualizované. Výsledky testů poté umožňuje zaznamenat ve formátech XML, HTML, LaTeX a PDF.

## 4.4 Navržená metodologie pro testy

Cílem této kapitoly je navrhnout obecnou metodologii pro testy, kterými se zabývá tato práce. Každé testování by mělo mít nějaký řád a nemělo by být prováděno bez stanovené strategie nebo kritérií. Proto by se tester měl držet tzv. životního cyklu testování software [46], [47], [48]. Tento cyklus se skládá ze skupin určitých aktivit, které je doporučené při testování software dodržovat.

Vlastní návrh takového cyklu se skládá ze čtyř následujících fází:

- **Analýza a návrh testů**
- **Implementace testů**
- **Analýza výsledků**
- **Report výsledků**

### Analýza a návrh testů

V této první fázi by se tester měl zaměřit na dokumentace k systému, který dostal za úkol otestovat. Na základě těchto dokumentací porozumět, jak celý systém funguje a identifikovat, co je testovatelné. Zkoumáním dokumentace by měl být schopen také určit rozsah testů, popřípadě akceptační kritéria. Tato fáze z velké části zahrnuje časté konverzace s vývojáři, analytiky a managementem k lepšímu a přesnějšímu určení celkové testovací strategii.

Po analýze požadavků provede tester návrh jednotlivých testů na základě dokumentace. Tato činnost patří mezi nejdůležitější a je velmi často podceňována. Tester musí určit, kolikrát je potřeba testy spustit, jaké jsou jejich podmínky, jak přistupovat ke každému testu, vytvořit testovací případy, zvolení vstupu a očekávaného výstupu atd. Součástí návrhu je i vhodná volba nástrojů a technologií pro automatizaci testů. Pokud tester neprovede kvalitní návrh testů, tak se v pozdějších fázích může celý testovací tým potýkat s různými problémy. Mezi ně patří např., že test není dostatečně efektivní, nepokrývá důležité části testované aplikace, výsledky

testu jsou nepřesné apod. To následně může vést k nedostatečnému otestování aplikace a nesplnění požadavků, které stanovil zákazník.

### Implementace testu

Jedná se o finální fázi příprav, kdy se jednotlivé navržené scénáře (testy) zadávají do zvoleného testovacího nástroje. Tester vytváří testovací skripty, které ověří, zda systém skutečně splňuje požadavky a funguje podle představ zákazníka. Stejně tak jako u předchozí fáze musí být pověřený tester velmi pozorný, jakým způsobem skripty implementuje. Pokud tomu tak není, může se stát, že testovací skript testuje jinou funkcionalitu systému, než by testovat měl nebo ji netestuje správně. Proto je zde důležité zohlednit testerovu technickou zdatnost, znalost a praktické zkušenosti.

Po implementaci tester spouští jednotlivé testy na předpřipraveném testovacím prostředí. Před spuštěním testů je vhodné jednotlivé testovací scénáře zkontrolovat, že je vše v pořádku a správně nastaveno. Při prvotním spuštění testu je nutné, aby tester sledoval testovací prostředí i testovací nástroj a zajistil tak, že se chovají podle očekávání.

### Analýza výsledků

Po skončení testu je předposlední fází analýza výsledků. Přestože tomu tak na první pohled být nemusí je tato poměrně náročná. Tester musí být schopen se orientovat v naměřených výsledcích a především jim rozumět. Má za úkol zjistit počet úspěšných a neúspěšných testů, analyzovat naměřené grafy a jednotlivé zjištěné závady, vyhledat příčinu proč vznikly, přiřazovat jim prioritu závažnosti atd. Pokud analýza není provedena důkladně mohou se poté opomenout jisté závady, které mohou mít zásadní dopad na celkovou kvalitu software. Stejně jako u implementace testů je nutné, aby tester měl dostatečné technické znalosti i praktické zkušenosti.

### Report výsledků

Po analýze výsledků tester vytvoří výsledné shrnutí testu, což je zároveň i poslední fází životního cyklu testování software. Tester je zodpovědný zaznamenat v přehledné a srozumitelné formě dosažené výsledky pro ostatní členy týmu. V praxi to znamená, že každá chyba, která byla objevena, je zaznamenána do nějakého nástroje pro evidenci chyb a problémů, např. JIRA. Každá chyba by měla být evidována zvlášť, popřípadě je nutné zmínit určitou souvislost s chybou jinou, aby byla zajištěna přehlednost a dobrá orientace v jednotlivých chybách. Tester by měl vzniklou chybu popsat, přiložit potřebné obrázky nebo soubory, a pokud je to v rozsahu jeho znalostí, tak doporučit, jak chybu odstranit.

V kapitole *Moduly systému NDS* jsou popsány jednotlivé moduly, ze kterých se systém NDS skládá. Ke každému modulu existuje interní dokumentace, kde je popsáno na základě požadavků od zákazníka, jak má daný modul fungovat, jaké jsou jeho vstupy/výstupy, závislosti na ostatních modulech, uživatelská práva, konfigurace systému atd.

Jelikož se jedná o mezinárodní projekt, je nutné zmínit, že veškeré dokumentace jsou v anglickém jazyce. To celý proces značně ztížilo a bylo potřeba důkladně číst každou informaci a provádět adekvátní překlad do českého jazyka. Na základě studia těchto dokumentací jsem provedl analýzu a navrhl provedení jednotlivých následujících testů. Každý test bude mít strukturu na základě navržené metodologie pro testy.

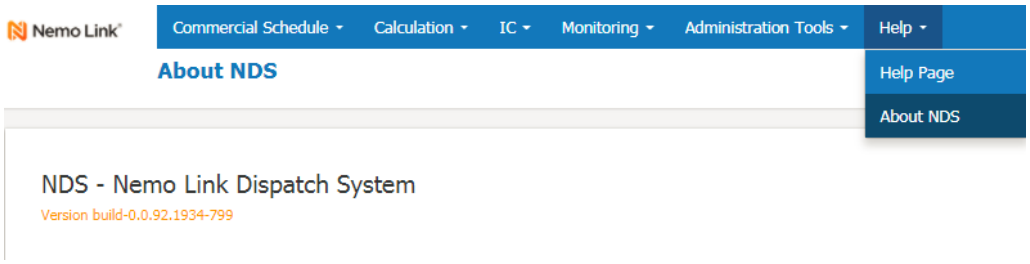
## 4.5 GUI/Autorizační test

Cílem je provést GUI a autorizační test na testovacím prostředí aplikace NDS. Na základě výsledků poté provést vyhodnocení, zda byl test úspěšný nebo neúspěšný.

### 4.5.1 Analýza a návrh testu

Návrh GUI testu byla poměrně komplexní záležitost. K automatizaci testů pro kompletní kontrolu uživatelského grafického rozhraní by bylo zapotřebí desítky až stovky hodin. Takový postup by byl z hlediska omezeného počtu hodin pro každý test neefektivní a nereálný. Hlavním důvodem je, že v dokumentaci jsou požadavky, u kterých by bylo potřeba kontrolovat názvy všech použitých elementů, tabulek, formulářů, jejich velikost, barva, písmo apod.

Z tohoto důvodu jsem zvolil postup, kde GUI test bude určen k ověření správnosti názvů jednotlivých kategorií v hlavní liště aplikace NDS, jejich podkategorií (názvy konkrétních obrazovek) a při zvolení dané podkategorie např. **About NDS** po načtení obrazovky její hlavní nadpis, viz obr. 4.4. Součástí GUI testu nástroj po načtení každé obrazovky provede snímek a uloží ho do souboru. Snímek bude k dispozici pro rychlou kontrolu grafického rozhraní. Zde vycházím z faktu, že tester má nastudované dokumentace a z pouhého snímku téměř okamžitě pozná, jestli je vše v pořádku podle požadavků či nikoliv.




Obr. 4.4: Názvy kategorií, podkategorie **Help** a obrazovka **About NDS**

Návrh autorizačního testu jsem provedl na základě požadovaného omezení, kdy je uživatel přihlášen za danou roli. V každé dokumentaci modulu je popsáno, k jaké obrazovce má uživatel v určité roli přístup, popřípadě oprávnění ke změnám v systému. Na základě těchto informací jsem vytvořil 2 tabulky, ze kterých jsou zřetelně vidět všechna oprávnění.

V systému NDS existuje 5 různých rolí:

1. **Systémový administrátor**
2. **Vedoucí operátor**
3. **Operátor na místě**
4. **Vzdálený operátor**
5. **Obchodní analytik**

Každá role má určité oprávnění ke změnám v systému, viz obr. 4.5 a pro přístup k obrazovkám, viz obr. 4.6, a to jak v automatickém, tak v manuálním módu.

| Práva u dané obrazovky | Automatický mód | | | | | Manuální mód | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SA | OM | OO | OR | CA | SA | OM | OO | OR | CA |
| **User Management – Create/Edit User Account** (Vytvoření/Editace Uživatelského Účtu) | ✓ | - | - | - | - | ✓ | - | - | - | - |
| **System Configuration –Edit Configuration** (Editace Konfigurace) | ✓ | ✓ | - | - | - | ✓ | ✓ | - | - | - |
| **Outage Overview – Create/Edit Outage** (Vytvoření/Editace Výpadku) | - | ✓ | ✓ | ✓ | - | - | ✓ | ✓ | ✓ | - |
| **Commercial Schedule Overview – Create Commercial Schedule** (Vytvoření Obchodního Plánu) | - | - | - | - | - | - | - | ✓ | ✓ | - |

SA = System Administrator (Systémový Administrátor), OM = Operator Manager (Vedoucí Operátor),
OO = Operator On-Site (Operátor na Místě), OR = Operator Remote (Vzdálený Operátor),
CA = Commercial Analyst (Obchodní Analytik), ✓ = Práva uděleny, "-" = Práva zamítnuty

Obr. 4.5: Práva ke změnám v systému u daných obrazovek

| Název obrazovky | Automatický mód | | | | | Manuální mód | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SA | OM | OO | OR | CA | SA | OM | OO | OR | CA |
| **Mode of Control** (Mód Ovládání) | ✓ | - | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | - |
| **File Storage** (Uložiště Souborů) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **User Management** (Správa Uživatelů) | ✓ | ✓ | - | - | - | ✓ | ✓ | - | - | - |
| **System Configuration** (Systémová Konfigurace) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Audit Log** (Záznam Auditu) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Password Change** (Změna Hesla) | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| **Manual File Upload** (Manuální Nahrání Souboru) | ✓ | - | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | - |
| **Help Page** (Stránka Napovědy) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **About** NDS (O NDS) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **General Overview** (Celkový Přehled) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Notifications** (Notifikace) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Power Profile Overview** (Přehled Energetického Profilu) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Reports** (Reporty) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **IC Overview** (Přehled Mezispojové Schopnosti) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Outage Overview** (Přehled Výpadků) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Power Orders Overview** (Přehled Energetických Objednávek) | - | ✓ | ✓ | ✓ | - | - | ✓ | ✓ | ✓ | - |
| **Commercial Schedule Overview** (Přehled Obchodního Plánu) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

SA = System Administrator (Systémový Administrátor), OM = Operator Manager (Vedoucí Operátor),

OO = Operator On-Site (Operátor na Místě), OR = Operator Remote (Vzdálený Operátor),

CA = Commercial Analyst (Obchodní Analytik), ✓ = Přístup k obrazovce, "-" = Bez přístupu k obrazovce

Obr. 4.6: Práva pro přístup k obrazovkám

Na základě výše uvedených informací byl navržen test, který provede GUI i autorizační test zároveň. Nejprve byl vytvořen GUI test v rámci jedné role, který byl později rozšířen o role zbývající. Pro vykonání testu jsem zvolil nástroj Apache Jmeter a plugin WebDriver Sampler.

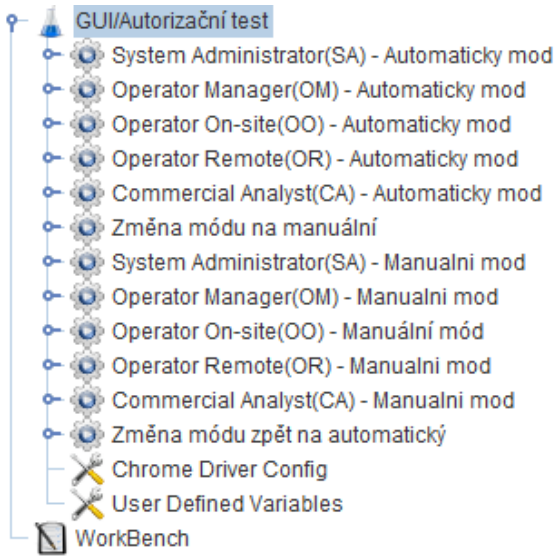
Bude provedena simulace uživatele, který se přihlásí prostřednictvím uživatelského účtu a ověří, zda-li jsou názvy kategorií, podkategorií a obrazovek správné, ale také odpovídající oprávnění k jednotlivých obrazovkám a změnám v systému. Kroky simulující uživatele z pohledu jedné role budou následující:

1. Spuštění webového prohlížeče Google Chrome, načtení webové adresy testovacího prostředí a pořízení snímku přihlašovací obrazovky aplikace.

2. Vyplnění příslušných přihlašovacích údajů k účtu a přihlášení uživatele v dané roli. Po přihlášení pořízení snímku úvodní obrazovky aplikace.
3. Postupné procházení příslušných obrazovek podle výše uvedených tabulek. Uživatel kontroluje správné názvy, práva pro přístup k obrazovkám a změnám v systému, ke kterým by měl mít práva v roli, za kterou je přihlášen. U každé z obrazovek pořídí snímek, uloží ho do souboru a zaznamená tuto činnost do souboru se záznamy.

### 4.5.2 Implementace testu

Kompletní testovací plán v nástroji Apache Jmeter je na obr. 4.7. Plán je definován jako **GUI/Autorizační test**. Níže jsou definovaná vlákna pro každou existující roli v systému. Nejprve se provede test pro všech pět rolí, když je systém v automatickém módu. Poté se spustí skript k přepnutí systému na manuální mód a opět nástroj provede testování pěti rolí. Ke konci se spustí skript k přepnutí systému zpět na automatický mód.




Obr. 4.7: Přehled testovacího plánu **GUI/Autorizační test**

Aby mohl být test spuštěn, je nutné přidat konfigurační ovladač (**Chrome Driver Config**), který propojí ovladač pro automatizované testy Selenium WebDriver a příslušný webový prohlížeč Google Chrome. K zachování správné funkčnosti bylo nutné zadání relativní cesty k ovladači webového prohlížeče. Ostatní nastavení bylo ponecháno v původním nastavení.

Pro zvýšení efektivity testování je do testu přidán konfigurační element pro definování proměnných (**User Defined Variables**) viz obr. 4.8. Z obrázku je patrné,

že zde byly definovány tyto proměnné:

- **WEB** - webová adresa cílového testovacího prostředí (serveru).
- **JMÉNO()** - uživatelské jméno pro přihlášení do systému za danou roli.
- **HESLO()** - uživatelské heslo pro přihlášení do systému za danou roli.
- **DATUM** - datum potřebný pro přepnutí obrazovky *Commercial Schedule Overview* o jeden den dopředu pro ověření příslušných práv.
- **SLOŽKA** - složka, do které se ukládají výsledky testů.

Definici proměnných je důležité u testů zvážit. Pokud používáme určitou proměnnou např. webovou adresu ve více, jak jednom vlákně (test v této práci obsahuje vláken 12), tak by bylo velice neefektivní tuto proměnnou nezadefinovat. V situaci, kdy by došlo ke změně webové adresy testovacího prostředí, musel by uživatel ve skriptu každého vlákna webovou adresu upravit. Z hlediska větších projektů by to byla velká ztráta času. Pokud má uživatel společnou proměnnou zadefinovanou, stačí pouze přepsat daný řádek a změna se projeví ve všech vláknech, která tuto proměnnou používají.

**User Defined Variables**

**Name:** User Defined Variables

**Comments:**

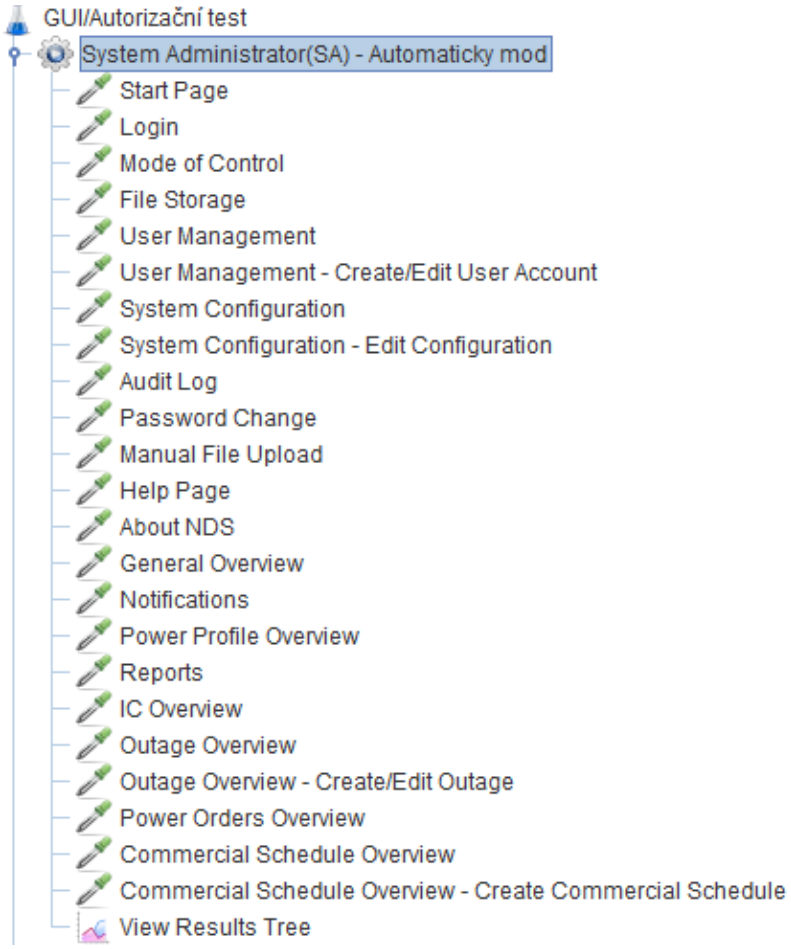
**User Defined Variables**

| Name: | Value |
|---|---|
| WEB | http://192.168.33.46/#!/login |
| JMÉNO(SA) | system@nll.com |
| HESLO(SA) | test |
| JMÉNO(OM) | manager@nll.com |
| HESLO(OM) | test |
| JMÉNO(OO) | onsite@nll.com |
| HESLO(OO) | test |
| JMÉNO(OR) | remote@nll.com |
| HESLO(OR) | test |
| JMÉNO(CA) | analyst@nll.com |
| HESLO(CA) | test |
| DATUM | 15.04.2018 |
| SLOŽKA | GUI,Autorizační test |

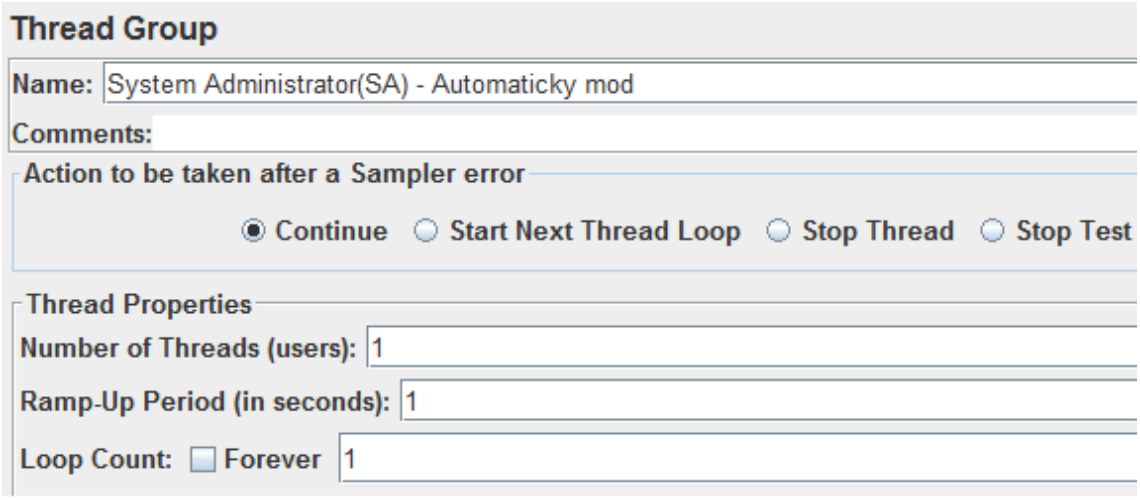
Obr. 4.8: Definované proměnné (**User Defined Variables**)

Po rozbalení vlákna systémového administrátora v automatickém módu (**Thread Group**), viz obrázek 4.9 lze vidět 23 testovacích skriptů a tzv. naslouchač (**View Results Tree**), kam se zapisují výsledky. Nastavení vlákna viz obr. 4.10 je následující:

- **Continue** - v situaci, kdy nástroj zaznamená chybu vzorku bude pokračovat v testování.
- **Number of Threads** - počet uživatelů simulace.




Obr. 4.9: Skripty vlákna **System Administrator(SA) - Automaticky mod**




Obr. 4.10: Nastavení vlákna **System Administrator(SA) - Automaticky mod**

- **Ramp-Up Period** - čas v sekundách udávající, za jakou dobu dojde k nahrání všech uživatelů simulace.
- **Loop Count** - udává počet kolikrát se všechny skripty vlákna zopakují.

Struktura skriptů vypadá pro všechny ostatní role totožně kromě vláken pro přepínání módů. Všechny skripty jsou napsané pro programovací jazyk JavaScript a jako vzorkovač je použit již zmíněný (**WebDriver Sampler**). Pro vyhledávání požadovaných elementů z formuláře HTML (HyperText Markup Language) je využita metoda XPath (XML Path Language), která umožňuje vyjádřit relativní cestu k danému elementu.

Jednotlivé kroky prvních tří skriptů každého vlákna jsou patřičně okomentovány (zbytek skriptů je již bez komentáře, jelikož se použité příkazy do jisté míry opakují). Z tohoto důvodu bude popsáno pouze vlákno **System Administrator(SA) - Automaticky mod** a jeho první tři skripty.

První skript má za úkol načtení přihlašovací webové stránky (**Start Page**), viz obr. 4.11 a skládá se z následujících pěti kroků:

1. Importování potřebných Selenium balíčků pro automatizaci testů a nastavení časového limitu čekání webového ovladače na 10 sekund v případě nedostupnosti elementu.
2. Spuštění webového prohlížeče Google Chrome a načtení webové adresy z proměnné *WEB*.
3. Vyčkání, dokud není element *username* načten v záhlaví formuláře, z čehož plyne, že po jeho načtení je přihlašovací stránka zobrazena.
4. Pořízení snímku obrazovky a přejmenování na *Start Page(SA).png*.
5. Zapsání záznamu o provedení snímku do soubor *jmeter.log*.

Script (see below for variables that are defined)

```
1  var driver = JavaImporter(org.openqa.selenium, org.openqa.selenium.support.ui)
2  //importování potřebných Selenium balíčků pro automatizování testů
3  var wait = new driver.WebDriverWait(WDS.browser, 10)
4  //nastavení časového limitu čekání na 10 sekund v případě nedostupnosti elementu
5
6
7  WDS.browser.get('${WEB}')
8  //jdi na webovou adresu uloženou v proměnné WEB
9  wait.until(driver.ExpectedConditions.elementToBeClickable(driver.By.xpath("//input[@id='username']")))
10 //vyčkej, dokud není element "username" načten v záhlaví formuláře
11
12
13 WDS.browser.getScreenshotAs(driver.OutputType.FILE).renameTo(new java.io.File('Start Page(SA).png'))
14 //udělej snímek obrazovky a proveď přejmenování na "Start Page(SA).png"
15 WDS.log.info(WDS.name + 'screenshot saved');
16 //proveď záznam o provedení snímku obrazovky do souboru jmeter.log
```

Obr. 4.11: Skript pro načtení přihlašovací webové stránky (**Start Page**)

Druhý použitý skript má za úkol přihlášení uživatele do systému (**Login**), viz obr. 4.12 a skládá se z pěti následujících kroků:

1. Importování potřebných Selenium balíčků pro automatizaci testů a nastavení časového limitu čekání webového ovladače na 5 sekund v případě nedostupnosti elementu.
2. Vyhledání elementu *username* a vložení řetězce znaků uložený v proměnné *JMÉNO(SA)*.
3. Vyhledání elementu *password* a vložení řetězce znaků uložený v proměnné *HESLO(SA)*.
4. Vyhledání elementu (tlačítka) pro přihlášení *loginButton* a kliknutí na něj.
5. Vyčkání, dokud není element *menu-heading ng-binding* načten v záhlaví formuláře, z čehož plyne, že po jeho načtení je úvodní stránka po přihlášení zobrazena.

Script (see below for variables that are defined)

```
var driver = JavaImporter(org.openqa.selenium, org.openqa.selenium.support.ui)
var wait = new driver.WebDriverWait(WDS.browser, 5)


WDS.browser.findElement(driver.By.xpath("//input[@id='username']")).sendKeys('${JMÉNO(SA)}')
//vyhledej element "username" a vlož řetězec znaků uložený v proměnné "JMÉNO(SA)"
WDS.browser.findElement(driver.By.xpath("//input[@id='password']")).sendKeys('${HESLO(SA)}')
//vyhledej element "password" a vlož řetězec znaků uložený v proměnné "HESLO(SA)"
WDS.browser.findElement(driver.By.xpath("//button[@id='loginButton']")).click()
//vyhledej element "loginButton" a klikni na něj

wait.until(driver.ExpectedConditions.elementToBeClickable(driver.By.xpath
("//h4[@class='menu-heading ng-binding']")))
//vyčkej, dokud není element "menu-heading ng-binding" načten v záhlaví formuláře
```

Obr. 4.12: Skript pro přihlášení uživatele (**Login**)

Třetí použitý skript má za úkol zkontrolovat názvy a ověřit práva módu ovládání (**Mode of Control**), viz obr. 4.13. Testovací skript se skládá ze sedmi následujících kroků:

Script (see below for variables that are defined)

```
var driver = JavaImporter(org.openqa.selenium, org.openqa.selenium.support.ui)
var wait = new driver.WebDriverWait(WDS.browser, 3)


WDS.browser.findElement(driver.By.linkText("Administration Tools")).click()
//vyhledej element "Administration Tools" a klikni na něj
wait.until(driver.ExpectedConditions.elementToBeClickable(driver.By.linkText("Mode of Control")))
//vyčkej, dokud není element "Mode of Control" načten v záhlaví formuláře

WDS.browser.findElement(driver.By.linkText("Mode of Control")).click()
//vyhledej element "Mode of Control" a klikni na něj
wait.until(driver.ExpectedConditions.elementToBeClickable(driver.By.xpath("//button/span[text()='Confirm']")))
//vyčkej, dokud není element "Confirm" načten v záhlaví formuláře

WDS.browser.getScreenshotAs(driver.OutputType.FILE).renameTo(new java.io.File('Mode of Control(SA).png'))
WDS.log.info(WDS.name + ' screenshot saved');
```

Obr. 4.13: Skript pro kontrolu módu ovládání (**Mode of Control**)

1. Importování potřebných Selenium balíčků pro automatizaci testů a nastavení

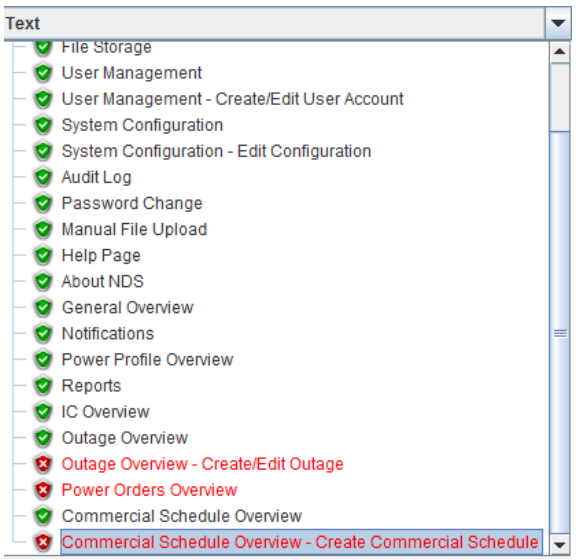
časového limitu čekání webového ovladače na 3 sekundy v případě nedostupnosti elementu.

2. Vyhledání elementu (kategorie z hlavní lišty) *Administration Tools* a a kliknutí na něj.
3. Vyčkání, dokud není element (podkategorie) *Mode Of Control* načten v záhlaví .formuláře (nástroj čeká než se načte požadovaná podkategorie)
4. Vyhledání elementu (podkategorie) *Mode of Control* a kliknutí na něj.
5. Vyčkání, dokud není element (tlačítko) *Confirm* načten v záhlaví formuláře (pokud je načtené v záhlaví formuláře tlačítko *Confirm* je zobrazena i daná obrazovka).
6. Pořízení snímku obrazovky a přejmenování na *Mode of Control(SA).png*.
7. Zapsání záznamu o provedení snímku do souboru *jmeter.log*.

Zbývajících 20 skriptů tohoto vlákna je téměř stejné, tak jako u ostatních vláken. Jiné jsou pouze v názvech elementů, relativní cestě k elementu XPath a použitých proměnných. Celý testovací plán bude k dispozici v příloze této práce.

### 4.5.3 Analýza výsledků

Po skončení testu se výsledky každého vlákna zapisují do naslouchače (**View Results Tree**), viz obr. 4.14.

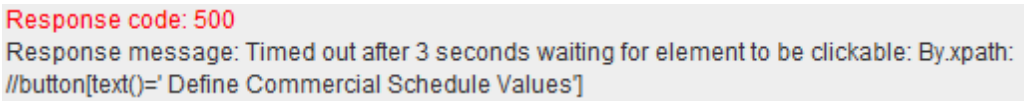



Obr. 4.14: Část výsledků testu pro systémového administrátora (**View Results Tree**)

Ten zobrazuje skripty v pořadí, v jakém byly prováděny v průběhu testu. Zeleně jsou označeny skripty, které byly bez problému a červeně jsou označeny chyby.

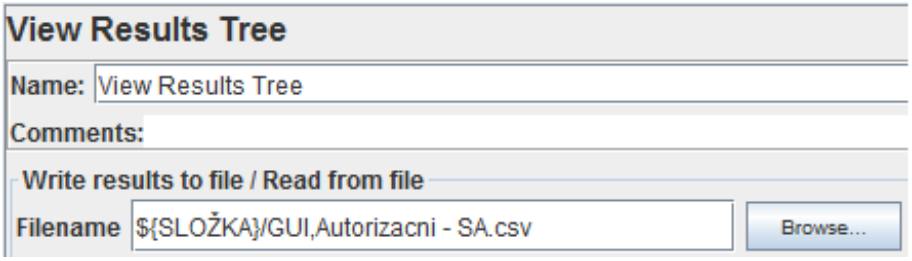
Nemusí se však jednat o chybu, ale naopak o chtěné ověření práv (pokud uživatel v dané roli nemá práva pro určitou činnost nebo přístup k obrazovce, nástroj element nenačte a tudíž ho označí červeně).

Jako příklad je uveden červeně označený skript pro manuální vytvoření obchodního plánu (**Commercial Schedule Overview - Create Commercial Schedule**), viz obr. 4.15. Z něho lze vyčíst, že čas čekání na načtení elementu pro definování nových hodnot obchodního plánu (**Define Commercial Schedule Values**) vypršel po 3 nastavených sekundách. Pokud se vrátíme k tabulce na obr. 4.5, tak zjistíme, že uživatel v roli systémového administrátora v automatickém módu práva k této činnosti mít neměl, a tudíž lze výsledek označit za platný a správně označený.




Obr. 4.15: Detail chyby u přístupu k obrazovce (**Commercial Schedule Overview - Create Commercial Schedule**)

Výsledky obou vláken pro danou roli v automatickém i manuálním módu se zapisují do souboru *GUI,Autorizační - SA.csv*, viz obr. 4.16. Pod proměnnou *SLOŽKA* je adresář *GUI,Autorizační test*, do kterého se mají výsledky ukládat. Zápis vykonaných testů je určen pro zpětnou kontrolu. Nevýhodou je, že Jmeter zapisuje výsledky do těchto souborů relativně nepřehledně a v případě větších projektů bych tento postup nedoporučoval. S tím souvisí i další nevýhoda, že zápis výsledků nezvládá českou diakritiku, která způsobí ještě větší nepřehlednost. Z tohoto důvodu jsou vlákna pojmenovány bez diakritiky.




Obr. 4.16: Zapisování výsledku do souboru *GUI,Autorizační - SA.csv*

Test byl vykonáván 12 minut a 17 sekund. Celkem bylo do testu zapojeno 12 vláken, spuštěno 236 skriptů, z toho 184 bylo označeno zeleně a 52 označeno červeně. Na základě výsledků byla vytvořena zjednodušená tabulka viz obr. 4.17. Zde jsou zobrazeny pouze ty obrazovky, ke kterým by na základě testu neměl mít uživatel za danou roli přístup nebo oprávnění ke změnám v systému (vynechány jsou obrazovky,

které byly označeny zeleně pro všechny role v obou módech). Pokud srovnáme výsledky s tabulkami na obr. 4.5 a 4.6, tak zjistíme, že podle požadavků by měl test (za předpokladu, že je vše v pořádku) označit červeně pouze 46 skriptů.

| Název/práva u obrazovky | Automatický mód | | | | | Manuální mód | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SA | OM | OO | OR | CA | SA | OM | OO | OR | CA |
| **Mode of Control** (Mód Ovládání) | ✓ | - | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | - |
| **User Management** (Správa Uživatelů) | ✓ | ✓ | - | - | - | ✓ | ✓ | - | - | - |
| **Password Change** (Změna Hesla) | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| **Manual File Upload** (Manuální Nahrání Souboru) | ✓ | - | ✓ | ✓ | - | ✓ | - | ✓ | ✓ | - |
| **Power Orders Overview** (Přehled Energetických Objednávek) | - | - | - | - | - | - | - | - | - | - |
| **User Management – Create/Edit User Account** (Vytvoření/Editace Uživatelského Účtu) | ✓ | - | - | - | - | ✓ | - | - | - | - |
| **System Configuration –Edit Configuration** (Editace Konfigurace) | ✓ | ✓ | - | - | - | ✓ | ✓ | - | - | - |
| **Outage Overview – Create/Edit Outage** (Vytvoření/Editace Výpadku) | - | ✓ | ✓ | ✓ | - | - | ✓ | ✓ | ✓ | - |
| **Commercial Schedule Overview – Create Commercial Schedule** (Vytvoření Obchodního Plánu) | - | - | - | - | - | - | - | ✓ | ✓ | - |

SA = System Administrator (Systémový Administrátor), OM = Operator Manager (Vedoucí Operátor),

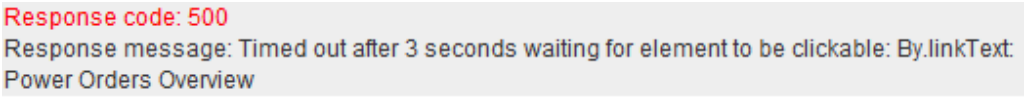
OO = Operator On-Site (Operátor na Místě), OR = Operator Remote (Vzdálený Operátor),

CA = Commercial Analyst (Obchodní Analytik), ✓ = Přístup/práva udělen/y, "-" = Přístup/práva zamítnut/y

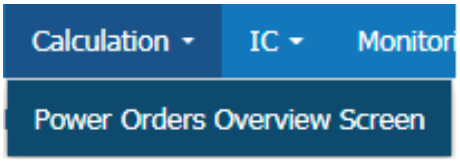
Obr. 4.17: Oprávnění uživatele za jednotlivé role podle výsledků testu

Na první pohled je z výsledků poznat, že problém nastal u obrazovky pro přehled energetických objednávek (**Power Orders Overview**). Přístup k této obrazovce by měl mít uživatel v roli vedoucího operátora, operátora na místě a vzdáleného operátora v obou módech systému systému.

Pokud se podíváme na detail výsledku tohoto skriptu u role vedoucího operátora, tak lze vidět chybu, viz obr. 4.18. Z té bylo zjištěno, že systém během 3 sekund nemohl najít element *Power Orders Overview*. Po manuální kontrole testovacího prostředí NDS byl identifikován problém, že podkategorie se jmenuje **Power Orders Overview Screen**, viz obr. 4.19. Tento název je podle požadavků nesprávný, a tudíž je nesprávný i název obrazovky.




Obr. 4.18: Detail chyby u přístupu k obrazovce (**Power Orders Overview**)

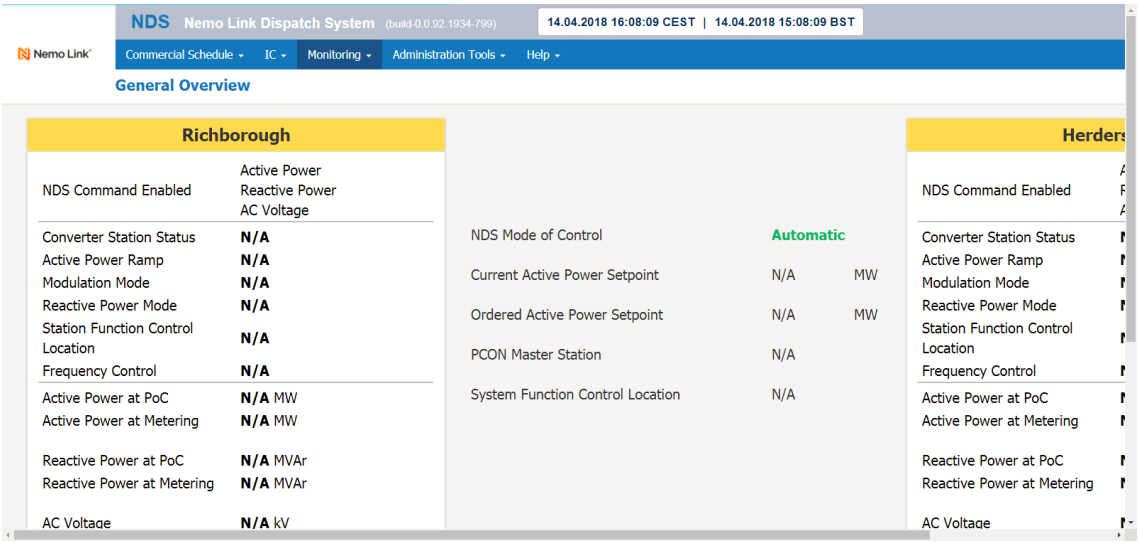



Obr. 4.19: Špatně nazvaná podkategorie (**Power Orders Overview Screen**)

### 4.5.4 Report výsledků

Z výsledků mohu udělat závěr, že test byl neúspěšný a zachytil jednu z chyb grafického rozhraní, která se projevuje u 3 systémových rolí v obou módech (celkem je tedy zaznamenáno 6 chyb).

Dále z pořízených snímků obrazovek bylo zjištěno, že vyvíjená aplikace NDS je ve výchozím nastavení webového prohlížeče Google Chrome zvětšená, viz obr. 4.20. To má za následek, že není vidět celá webová stránka do té doby, dokud si ji uživatel ručně nezmenší. I přes fakt, že snímky jsou kvůli popsané chybě zvětšeny, tak z nich po porovnání s dokumentací žádné závažnější chyby nebyly nalezeny.




Obr. 4.20: Pořízený snímek celkového přehledu (**General Overview**)

Z pohledu autorizačních práv žádný problém nenastal a byly úspěšně ověřeny. Nalezené chyby budou předány vývojovému týmu, který bude mít za úkol chyby

opravit do vydání nové verze aplikace NDS. Tento test bude spuštěn s každou nově vydanou verzí vyvíjené aplikace pro případné další zachycení chyb.

## 4.6 Výkonnostní test

Cílem výkonnostního testu je ověřit, zda aplikace NDS splňuje definované výkonnostní požadavky. Na základě výsledků poté provést vyhodnocení, zda-li byl test úspěšný nebo neúspěšný.

### 4.6.1 Analýza a návrh testu

K návrhu výkonnostního testu bylo nejprve zapotřebí zjistit, jaké jsou požadavky na výkon systému z pohledu uživatele. Na základě dokumentací byly identifikovány následující 3 výkonnostní požadavky:

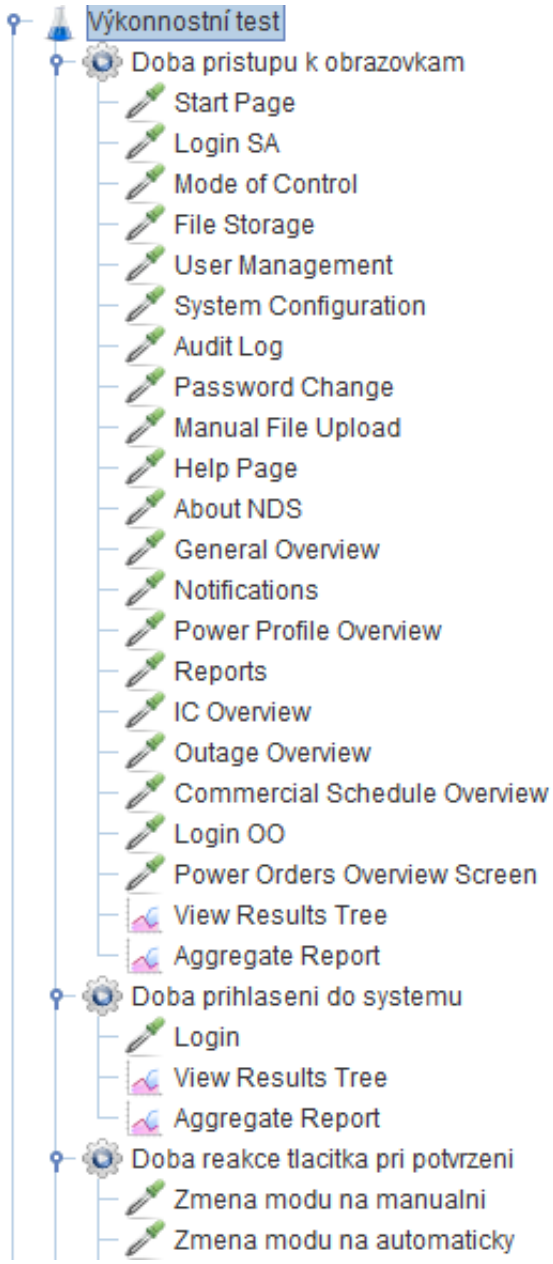
1. Odezva systému k zobrazení jednotlivých obrazovek uživatelem nesmí být vyšší než 2 sekundy.
2. Proces přihlášení uživatele nesmí být vyšší než 5 sekund.
3. Zaznamenání první vizuální změny od odeslání požadavku uživatele ke změně módu systému nesmí být vyšší než 1 sekunda (tímto je myšleno, za jak dlouho dojde k přepnutí módu systému po kliknutí na tlačítko k jeho změně).

Tyto požadavky musí být splněny jak z pohledu jednoho uživatele bez zátěže, tak i při průměrné očekávané zátěži 30. uživatelů (simulace zátěže více uživatelů bude popsána v zátěžovém testu).

Na základě těchto informací jsem navrhl využít opět plugin WebDriver Sampler a část již napsaných skriptů z GUI/autorizačního testu. Ten obsahuje definované přístupy do všech obrazovek, tudíž je potřeba provést pouze úpravu a rozšíření o měření odezvy. Plugin WebDriver Sampler umožňuje měření odezvy, za jak dlouho se určitá operace vykoná. Tato metoda je z pohledu uživatele a definovaných požadavků realistická a přesná. Testem se změří rychlost zpracování požadavků na straně klienta (aplikace NDS), nikoliv serveru. Zároveň slouží i jako vizuální kontrola, že systém skutečně zpracovává požadavky tak, jak by měl.

### 4.6.2 Implementace testu

Celý testovací plán včetně detailů vláken je na obr. 4.21.




Obr. 4.21: Přehled testovacího plánu **Výkonnostní test**

Vzhledem k tomu, že detailně byly jednotlivé použité prvky a skripty popsány v GUI/autorizačním testu, tak následně budou popsány pouze změny nebo nové prvky.

Struktura vlákna **Doba pristupu k obrazovkam** zůstala téměř stejná. Nutnou změnou bylo odstranit skripty pro kontrolu práv a upravit názvy ve skriptu *Power Orders Overview Screen* na základě výsledků z předchozího testu (nebylo by jinak možné změřit odezvu u této obrazovky vzhledem ke špatnému názvu). Skript *Login OO* pouze přihlásí jiného uživatele v roli (operátora na místě), který má pří-

stup k obrazovce přehledu energetických objednávek (**Power Orders Overview**). Simulace činnosti uživatele je tedy téměř totožná s předchozím testem. V celém testovacím plánu byly skripty rozšířeny o metodu k měření odezvy, viz obr. 4.22.

```
WDS.sampleResult.sampleStart()
//začátek zachycování času vzorkovače

WDS.browser.findElement(driver.By.linkText("Mode of Control")).click()
//vyhledej element "Mode of Control" a klikni na něj
wait.until(driver.ExpectedConditions.elementToBeClickable(driver.By.xpath("//button/span[text()='Confirm']")))
//vyčkej, dokud není element "Confirm" načten v záhlaví formuláře

WDS.sampleResult.sampleEnd()
//ukončení zachycování času vzorkovače
```

Obr. 4.22: Ukázka měření odezvy ve skriptu (**Mode of Control**)

Vlákno **Doba prihlaseni do systemu** obsahuje pouze jeden skript *Login*, který má za úkol spustit webový prohlížeč, načíst webovou adresu z proměnné a změřit dobu procesu přihlášení uživatele.

Vlákno **Doba reakce tlacitka pri potvrzeni** obsahuje skripty dva *Zmena modu na manualni* a *Zmena modu na automaticky*. První skript má za úkol spustit webový prohlížeč, načíst webovou adresu z proměnné, přihlásit uživatele, přejít na obrazovku ke změně módu (**Mode of Control**) a změřit dobu, za jakou se systém přepne do manuálního módu. Druhý skript měří dobu, za jakou se systém přepne zpět do módu automatického.

Nově přidanou položkou u všech vláken je naslouchač (**Aggregate Report**), který se používá k výkonnostnímu testování nejčastěji. Shromažďuje různé výsledky o časových odezvách, které jsou nezbytné pro určení případných nedostatků ve výkonu systému. U každého vlákna se budou zapisovat výsledky testu do souboru pro zpětnou analýzu výsledků. Naslouchač (**View Results Tree**) je určen pouze pro detailnější analýzu v případě výskytu chyb při testování.

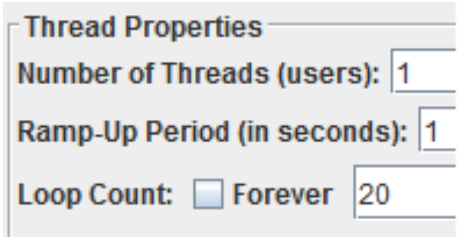
### 4.6.3 Analýza výsledků

Kapitola bude rozdělena na 2 části. První částí je analýza výsledků testu bez zátěže a druhá část je analýza výsledků testu se zátěží.

#### Test bez zátěže

U prvního testu jsou všechna 3 vlákna nastavena následovně, viz obr. 4.23:

- **Number of Threads** - počet uživatelů simulace.
- **Ramp-Up Period** - čas v sekundách udávající, za jakou dobu dojde k nahrání všech uživatelů simulace.
- **Loop Count** - udává počet kolikrát se všechny skripty vlákna zopakují.




Obr. 4.23: Nastavení vláken

První test byl dokončen za 9 minut a 58 sekund. Výsledky vlákna **Doba pristupu k obrazovkam** jsou zaznamenány v tabulce naslouchače (**Aggregate Report**),viz obr. 4.24.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Start Page | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Login SA | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mode of Control | 20 | 660 | 660 | 674 | 680 | 686 | 643 | 686 |
| File Storage | 20 | 659 | 661 | 671 | 674 | 674 | 639 | 674 |
| User Management | 20 | 183 | 183 | 205 | 224 | 246 | 153 | 246 |
| System Configuration | 20 | 660 | 652 | 684 | 693 | 720 | 641 | 720 |
| Audit Log | 20 | 754 | 751 | 775 | 785 | 836 | 721 | 836 |
| Password Change | 20 | 157 | 155 | 166 | 170 | 218 | 143 | 218 |
| Manual File Upload | 20 | 186 | 179 | 202 | 225 | 327 | 158 | 327 |
| Help Page | 20 | 677 | 663 | 698 | 744 | 771 | 655 | 771 |
| About NDS | 20 | 152 | 148 | 159 | 160 | 209 | 138 | 209 |
| General Overview | 20 | 124 | 124 | 138 | 140 | 142 | 103 | 142 |
| Notifications | 20 | 1174 | 1121 | 1373 | 1392 | 1691 | 944 | 1691 |
| Power Profile Overview | 20 | 686 | 682 | 703 | 706 | 725 | 666 | 725 |
| Reports | 20 | 169 | 169 | 182 | 199 | 199 | 148 | 199 |
| IC Overview | 20 | 658 | 659 | 670 | 671 | 677 | 642 | 677 |
| Outage Overview | 20 | 663 | 663 | 683 | 696 | 700 | 630 | 700 |
| Commercial Schedule Overview | 20 | 126 | 122 | 158 | 158 | 159 | 98 | 159 |
| Login OO | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Power Orders Overview Screen | 20 | 649 | 648 | 665 | 666 | 680 | 630 | 680 |
| TOTAL | 400 | 417 | 327 | 730 | 836 | 1328 | 0 | 1691 |

Obr. 4.24: Výsledky vlákna **Doba pristupu k obrazovkam**

Vzhledem k velikosti tabulky a dobré čitelnosti výsledků, je zobrazena pouze její nejdůležitější část (časové odezvy). V plné velikosti bude k dispozici v příloze. Z tabulky lze vyčíst:

- Vzorky (**Samples**) - celkový počet vzorků.
- Průměr (**Average**) - průměrný čas doby přístupu k obrazovce.
- Medián (**Median**) - znamená, že 50% vzorků netrvalo déle než uvedená doba a 50 % vzorků naopak trvalo uvedenou dobu nebo déle.
- 90% Hranice (**90% Line**) - znamená, že 90% vzorků netrvalo déle než uvedená doba a 10% vzorků naopak trvalo uvedenou dobu nebo déle.
- 95% Hranice (**95% Line**) - znamená, že 95% vzorků netrvalo déle než uvedená doba a 5% vzorků naopak trvalo uvedenou dobu nebo déle.
- 99% Hranice (**99% Line**) - znamená, že 99% vzorků netrvalo déle než uvedená doba a 1% vzorků naopak trvalo uvedenou dobu nebo déle.

- Minimum (**Min**) - minimální čas doby přístupu k obrazovce .
- Maximum (**Max**) - maximální čas doby přístupu k obrazovce .

Tabulka znázorňuje, že pro skripty *Start Page*, *Login SA* a *Login OO*, měření odezvy nebylo provedeno. To je z důvodu, že nejsou k zobrazení obrazovek určeny.

K hranici požadavku 2 sekund se nejvíce přibližovaly vzorky pro obrazovku notifikací (**Notifications**), kde nejvyšší byl 1691 ms. Jednoduchou matematikou s ohledem na parametr **99% Line** lze spočítat, že tato doba se týče pouze jednoho vzorku. Delší časové prodlevy jsou pravděpodobně způsobeny WebSocketem (technologie webových aplikací, kde klient navazuje obousměrné spojení se serverem a vyměňují si informace v reálném čase), který notifikace v aplikaci NDS využívají.

Z uvedené tabulky a průměrných i maximálních časů mohu udělat závěr, že požadavek na dobu zobrazení obrazovek do 2 sekund byl splněn.

Výsledky vlákna **Doba prihlaseni do systemu** jsou na obr. 4.25. Nejvyšší naměřený čas byl 2045 ms. Jedná se opět pouze o jediný vzorek.

Z průměrného a maximálního času mohu udělat závěr, že požadavek na dobu přihlášení uživatele do 5 sekund byl splněn.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Login | 20 | 1000 | 856 | 1487 | 1894 | 2045 | 803 | 2045 |
| TOTAL | 20 | 1000 | 856 | 1487 | 1894 | 2045 | 803 | 2045 |

Obr. 4.25: Výsledky vlákna **Doba prihlaseni do systemu**

Výsledky vlákna **Doba reakce tlacitka pri potvrzeni** jsou na obr. 4.26. U vzorku *Zmena modu na manualni* byl nejvyšší naměřený čas 1200 ms a u vzorku *Zmena modu na automaticky* 2196 ms. Při detailnější analýze ve vygenerovaném souboru bylo zjištěno, že 7 vzorků přesáhlo stanovený požadavek 1 sekundy. Průměrný čas obou vzorků (783 a 785 ms) je také relativně vysoký. Chyba je s největší pravděpodobností někde ve zdrojovém kódu, který budou muset vývojáři analyzovat a optimalizovat.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Zmena modu na manualni | 20 | 783 | 654 | 1171 | 1181 | 1200 | 633 | 1200 |
| Zmena modu na automaticky | 20 | 785 | 625 | 1155 | 1709 | 2196 | 603 | 2196 |
| TOTAL | 40 | 784 | 645 | 1171 | 1200 | 2196 | 603 | 2196 |

Obr. 4.26: Výsledky vlákna **Doba reakce tlacitka pri potvrzeni**

Z průměrné a maximální odezvy mohu udělat závěr, že požadavek pro zaznamenání první vizuální změny od odeslání požadavku uživatele ke změně módu systému do 1 sekundy splněn nebyl.

**Test se zátěží**

U druhého testu se simulovanou očekávanou zátěží 30. uživatelů zůstalo nastavení vláken stejné.

Druhý test byl dokončen za 10 minut a 3 sekundy. Výsledky vlákna **Doba pristupu k obrazovkam** jsou zaznamenány na obr. 4.27. Z tabulky je vidět, že maximální naměřený čas 2835 ms je u vzorku audit logu (**Audit Log**). Pokud vezmu do úvahy parametr **99% Line**, tak se tato doba týče pouze jednoho vzorku, což je rámci tolerance. V porovnání s tabulkou 4.24 je vidět, že průměrná doba přístupu většiny vzorků je nepatrně vyšší v řádech jednotek/desítek ms. To je však zanedbatelná doba vzhledem k simulované zátěži.

V tomto případě mohu udělat závěr, že požadavek na dobu zobrazení obrazovek do 2 sekund byl splněn i u testu druhého s očekávanou zátěží.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Start Page | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Login SA | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Mode of Control | 20 | 669 | 668 | 690 | 696 | 708 | 639 | 708 |
| File Storage | 20 | 664 | 662 | 684 | 689 | 715 | 636 | 715 |
| User Management | 20 | 202 | 198 | 223 | 246 | 261 | 161 | 261 |
| System Configuration | 20 | 678 | 666 | 704 | 706 | 882 | 641 | 882 |
| Audit Log | 20 | 909 | 760 | 917 | 1348 | 2835 | 732 | 2835 |
| Password Change | 20 | 174 | 172 | 199 | 202 | 204 | 152 | 204 |
| Manual File Upload | 20 | 202 | 202 | 214 | 237 | 239 | 176 | 239 |
| Help Page | 20 | 677 | 670 | 703 | 706 | 708 | 654 | 708 |
| About NDS | 20 | 172 | 168 | 196 | 199 | 202 | 141 | 202 |
| General Overview | 20 | 143 | 139 | 172 | 183 | 199 | 111 | 199 |
| Notifications | 20 | 908 | 888 | 1015 | 1029 | 1048 | 822 | 1048 |
| Power Profile Overview | 20 | 691 | 690 | 725 | 727 | 740 | 651 | 740 |
| Reports | 20 | 186 | 182 | 215 | 220 | 221 | 161 | 221 |
| IC Overview | 20 | 659 | 663 | 674 | 677 | 680 | 631 | 680 |
| Outage Overview | 20 | 657 | 656 | 676 | 678 | 684 | 634 | 684 |
| Commercial Schedule Overview | 20 | 125 | 120 | 147 | 166 | 173 | 106 | 173 |
| Login OO | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Power Orders Overview Screen | 20 | 659 | 659 | 665 | 677 | 684 | 645 | 684 |
| TOTAL | 400 | 419 | 261 | 734 | 848 | 1015 | 0 | 2835 |

Obr. 4.27: Výsledky vlákna **Doba pristupu k obrazovkam**

Výsledky vlákna **Doba prihlaseni do systemu** jsou na obr. 4.28. Nejvyšší naměřený čas vzorku byl 3972 ms (1 z 20). V porovnání s tabulkou 4.25 je průměrný čas o 350 ms vyšší (1350 ms). Z naměřených výsledků lze udělat závěr, že požadavek na dobu přihlášení uživatele do 5 sekund byl splněn.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Login | 20 | 1350 | 835 | 2419 | 2495 | 3972 | 791 | 3972 |
| TOTAL | 20 | 1350 | 835 | 2419 | 2495 | 3972 | 791 | 3972 |

Obr. 4.28: Výsledky vlákna **Doba prihlaseni do systemu**

Výsledky vlákna **Doba reakce tlacitka pri potvrzeni** jsou na obr. 4.29. U vzorku *Zmena modu na manualni* byl nejvyšší naměřený čas 2727 ms a u vzorku

*Zmena modu na automaticky* 2173 ms. Z vygenerovaného souboru bylo analyzováno, že 3 vzorky přesáhly požadavek 1 sekundy. V porovnání s tabulkou 4.26 jsou průměrné časy do jisté míry podobné.

Z výsledků mohu tedy udělat závěr, že tento požadavek nebyl splněn a stejně jako u testu bez zátěže bude potřeba provést optimalizaci zdrojového kódu.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Zmena modu na manualni | 20 | 787 | 658 | 721 | 1152 | 2727 | 633 | 2727 |
| Zmena modu na automaticky | 20 | 708 | 628 | 644 | 704 | 2173 | 603 | 2173 |
| TOTAL | 40 | 747 | 642 | 704 | 1152 | 2727 | 603 | 2727 |

Obr. 4.29: Výsledky vlákna **Doba reakce tlacitka pri potvrzeni**

### 4.6.4 Report výsledků

Z naměřených výsledků celého výkonnostního testu mohu udělat následující závěr:

- Požadavek k zobrazení všech obrazovek do 2 sekund byl splněn pro test bez zátěže i pro test se simulovanou zátěží 30. uživatelů.
- Požadavek o přihlášení uživatele do 5 sekund byl splněn pro test bez zátěže i pro test se simulovanou zátěží 30. uživatelů.
- Požadavek o zaznamenání vizuální změny od požadavku uživatele ke změně módu systému do 1 sekundy splněn nebyl pro oba testy a byl odhalen výkonnostní nedostatek.
- Za předpokladu, že za nesplněním 3. požadavku je neoptimalizovaný zdrojový kód (zátěž neměla téměř žádný vliv), tak lze usoudit, že aplikace NDS si dokáže zachovat funkčnost i pod průměrnou zátěží uživatelů.

Výsledek testu bude předán k analýze vývojovému týmu a následné optimalizaci aplikace. Stejně jako u GUI/autorizačního testu, bude i tento test spouštěn s každou nově vydanou verzí aplikace NDS. Všechny použité skripty, naměřené výsledky a pořízené obrázky budou v příloze této práce.

## 4.7 Zátěžový, stability a stress test

Cílem je provést zátěžový, stability a stress test. Zátěžový test ověří, jestli si aplikace NDS zachová funkcionalitu i při hraniční simulované zátěži. Stability test ověří, jestli si aplikace NDS zachová funkcionalitu při simulované zátěži po delší definovanou dobu. Cílem stress testu bude vyhledat tzv. bod nasycení („saturation point“). Jedná se o hraniční bod maximálního počtu uživatelů, které dokáže webový server obsloužit, aniž by docházelo k velké degradaci odezvy nebo využití procesoru. Na

základě výsledků každého testu bude provedeno vyhodnocení, zda-li byl úspěšný nebo neúspěšný.

### 4.7.1 Analýza a návrh testu

Vzhledem k tomu, že pro simulaci zátěžového, stability a stress testu je zapotřebí více uživatelů, byla by použitá metoda v předchozích testech velice neefektivní. Pokud by byl test spuštěn např. pro 30 uživatelů, spustil by se 30-krát webový prohlížeč, což by vedlo k velmi nepřesným výsledkům. V rámci testů v této kapitole je navíc hlavním účelem zjistit reakce serveru na zaslané požadavky, čehož bychom použitím předchozí metody nedocílili.

Zvolil jsem tedy metodu prostřednictvím tzv. HTTP požadavků (HTTP requests), jejichž použití je pro tyto typy testů nejrozšířenější a nejefektivnější. HTTP požadavek je forma zprávy, kterou posílá pokaždé webový prohlížeč webovému serveru s žádostí o určitý soubor nebo informace potřebné k zobrazení požadované webové stránky. Požadavek obsahuje v záhlaví potřebné informace pro server např. zdrojovou IP adresu, o jaký dokument žádáme, verzi webového prohlížeče, preferovaný jazyk, informace o relaci, použité kódování atd. Webový server odešle zpět odpověď s požadovanými informacemi. Tato komunikace je založena na nejrozšířenější architektuře typu klient-server.

K tomu, aby simulace činnosti uživatelů v systému byla pokud možno co nejpřesnější, je nutné použít časovače. Ty mají za úkol simulovat časovou prodlevu uživatele u dané činnosti např. prohlížení obchodního plánu, přihlášení do systému atd. Dále je nezbytné vědět, které funkcionality a obrazovky budou uživatelé využívat nejčastěji. Tyto informace však v dokumentaci stanoveny nebyly a bylo nutné zaslat e-mail zákazníkovi. E-mail obsahoval požadavek ohledně doplnění dokumentace o potřebné informace tzn. kolik průměrného času stráví uživatel na příslušné obrazovce, a které se budou využívat nejčastěji s ohledem na všechny systémové role.

Na základě odpovědi od zákazníka bude implementován testovací plán, který simuluje nejčastěji prováděnou činnost uživatele za systémové role vedoucího operátora, operátora na místě a vzdáleného operátora. Simulovaná činnost je složena z deseti následujících kroků (požadavků):

1. Požadavek pro zobrazení úvodní přihlašovací stránky.
2. Požadavek pro přihlášení uživatele.
3. Požadavek pro zobrazení celkového přehledu (**General Overview**).
4. Požadavek pro zobrazení notifikací (**Notifications**).
5. Požadavek pro zobrazení přehledu mezispojové schopnosti (**IC Overview**).
6. Požadavek pro zobrazení přehledu obchodního plánu pro určený den (**Commercial Schedule Overview**).

7. Požadavek pro zobrazení přehledu energetických objednávek pro určený den (**Power Orders Overview**).
8. Požadavek pro zobrazení přehledu energetického profilu pro určený den (**Power Profile Overview**).
9. Požadavek pro zobrazení systémové konfigurace (**System Configuration**).
10. Požadavek pro odhlášení uživatele.

Podmínky pro spuštění jednotlivých testů jsem navrhl následovně:

- Zátěžový test bude simulovat definovanou uživatelskou činnost s hraniční zátěží identifikovanou v požadavcích a to 50 uživatelů po dobu 20. minut.
- Stability test bude spuštěn pod zátěží totožnou jako zátěžový test, ale po dobu 2 hodin.
- Stress test bude spuštěn s velkým počtem uživatelů simulující činnost. Na základě naměřených grafů bude test analyzován a vyhledán bod nasycení. Test bude spuštěn po dobu 20 minut pod zátěží 250 uživatelů.

Všechny 3 zvolené testy spolu úzce souvisí, a proto jsou ve stejné kapitole. Stress test není v osnově práce, ale navrhl jsem ho zařadit mezi ostatní testy, jelikož by měl být součástí každého zátěžového testování. K vykonání testů je použit nástroj Apache Jmeter.

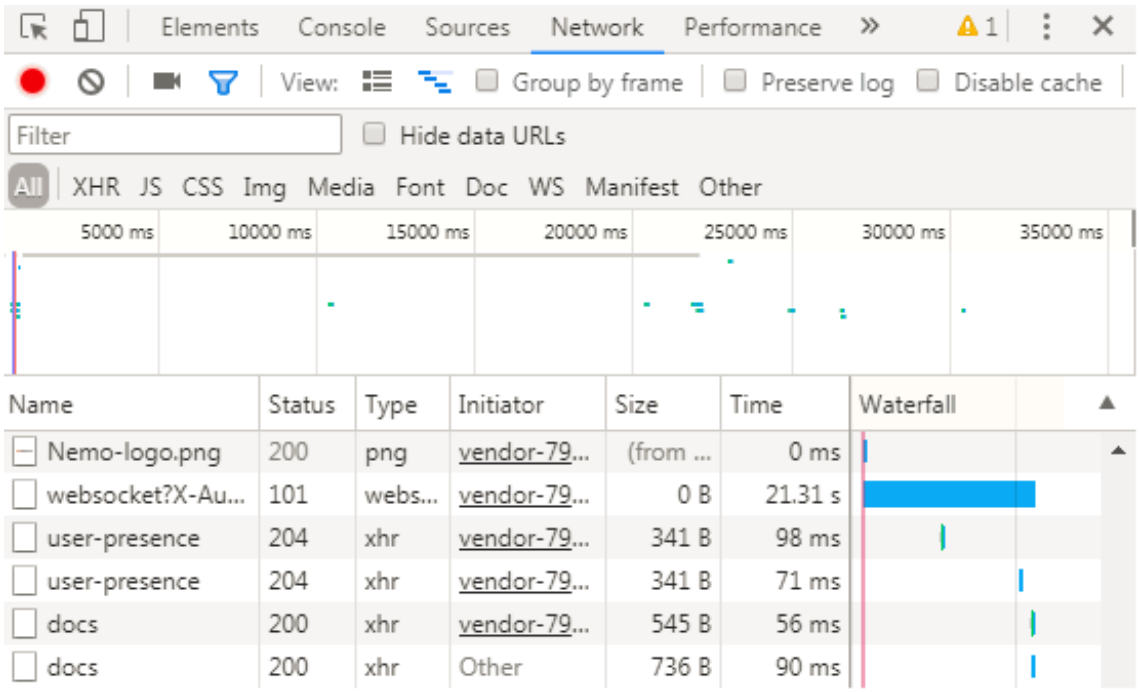
### 4.7.2 Implementace testu

K vytvoření testovacího plánu je potřeba nejprve zvolit způsob, jakým budou jednotlivé HTTP požadavky vytvářeny. Jednodušší, ale ne vždy správná metoda je nakonfigurovat Jmeter jako transparentní proxy server. Následně nakonfigurovat webový prohlížeč, aby směroval veškerý provoz přes Jmeter. Poté stačí v testované aplikaci pouze klikat (simulovat navržený testovací scénář) a jednotlivé HTTP požadavky bude zaznamenávat Jmeter. Poté je potřeba všechny zachycené požadavky upravit ručně podle potřeb. Ovšem tento způsob má i nevýhodu, protože ne vždy je žádoucí, aby byly zachyceny všechny požadavky. V případě delších scénářů může úprava zabrat více času než následující způsob.

Druhým způsobem je vytvářet HTTP požadavky ručně. Výhodou je, že testerovi dává plnou kontrolu nad tím, jaké požadavky budou volány a v jakém pořadí. Tato metoda je časově náročnější a je nutné, aby tester znal názvy formulářových polí, která se odesílají a uměl je následně vyplnit do vzorkovače (HTTP Request) v Jmeteru.

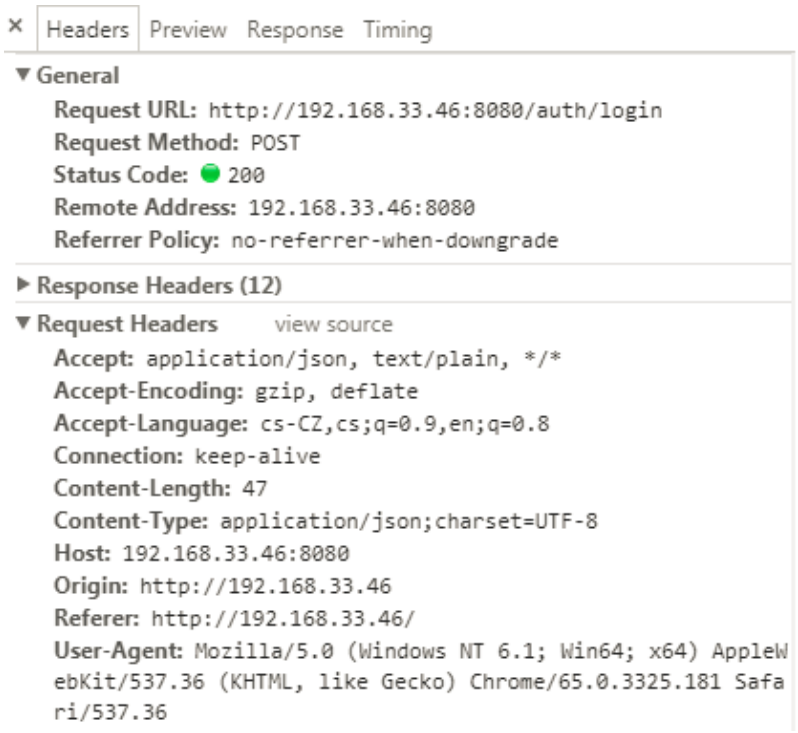
Na základě těchto informací jsem zvolil ruční metodu. Aplikace NDS nepoužívá větší počty formulářových polí v rámci požadavků a navržená simulace uživatele by neměla trvat delší dobu.

Pro zjištění formy jednotlivých HTTP požadavků jsem využil sadu zabudovaných nástrojů pro vývojáře (DevTools) ve webovém prohlížeči Google Chrome. Tuto sadu používají zejména vývojáři k ladění aplikace, získání informací pro optimalizaci kódu, vyhledat problémy s uspořádáním webové stránky apod. Pro spuštění stačí použít klávesovou zkratku *Ctrl+Shift+I* a nástroj se zpřístupní. Nejdůležitější je záložka **Network**, viz obr. 4.30, ve které se zaznamenává veškerá aktivita v síti a tedy i požadavky.




Obr. 4.30: Příklad zaznamenané aktivity v záložce **Network**

Po kliknutí na libovolný požadavek se otevře okno s detailními informacemi, viz obr. 4.31. Z toho lze vyčíst potřebné informace v záhlaví požadavku, které webový prohlížeč zasílá serveru ke zpracování. Tyto informace jsou nezbytné pro vytvoření jednotlivých HTTP požadavků v nástroji Jmeter.




Obr. 4.31: Detail požadavku pro přihlášení uživatele

Tímto způsobem byl vytvořen testovací plán, viz obr. 4.33. Vzhledem k velkému počtu nově použitých prvků na rozdíl od předchozích testů budou následně jednotlivě popsány. U každého prvku bude napsáno, k čemu slouží, a jak je nastaven. Pokud prvek bude použit v testu vícekrát, tak bude uveden pouze jeden jako příklad (toto se především týká HTTP požadavků).

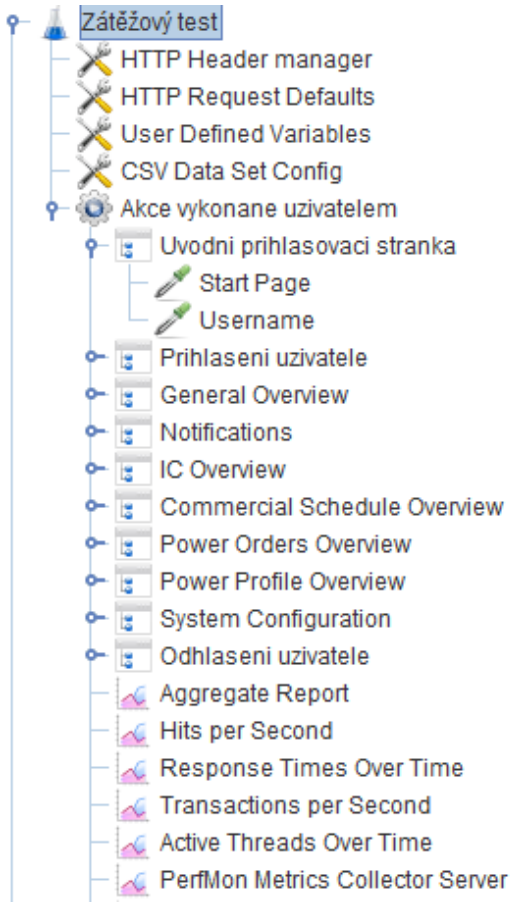
### HTTP Header Manager

HTTP Header Manager je správce záhlaví, který umožňuje přidat nebo přepisovat záhlaví HTTP požadavků. Používá se v případech, kdy se přenáší v požadavcích více stejných hodnot. V testu správce záhlaví obsahuje 6 položek, viz obr. 4.32, které je nutné přenášet v záhlaví každého definovaného HTTP požadavku:

**Headers Stored in the Header Manager**

| Name: | Value |
|---|---|
| User-Agent | Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro... |
| Accept-Language | cs-CZ,cs;q=0.9,en;q=0.8 |
| Accept-Encoding | gzip, deflate |
| Accept | application/json, text/plain, */* |
| Referer | http://192.168.33.46/ |

Obr. 4.32: Zadefinované hodnoty ve správci záhlaví (**HTTP Header Manager**)




Obr. 4.33: Přehled testovacího plánu **Zátěžový test**

1. **User-Agent** - umožňuje definovat typ klienta a přijímat verzi stránky, která bude správně v prohlížeči zobrazena.
2. **Accept-Language** - definuje jazyk stránky, kterou prohlížeč preferuje v přijaté odezvě.
3. **Accept-Encoding** - informuje server o způsobu kódování, který by měl zvolit před odesláním odpovědi klientovi.
4. **Accept** - označuje typ médií, které jsou klientem přijatelné.
5. **Referer** - adresa předchozí webové stránky, odkud klient následoval odkaz na aktuálně požadovanou stránku.

Některé HTTP požadavky mají v testu svého správce záhlaví, který definuje určitou hodnotu navíc:

- **X-Auth-Token** - autentizační token, který slouží k autentizaci uživatele.
- **Content-Type** - typ použitých médií v těle požadavku (používá se s požadavky typu POST).

Tyto hodnoty je potřeba přenášet pouze v některých požadavcích, a tudíž by bylo je zbytečné přenášet ve všech.

### HTTP Request Defaults

HTTP Request Defaults umožňuje nastavit výchozí hodnoty HTTP požadavků. To znamená hodnoty, které jsou použity v každém odeslaném požadavku. V testu je zadefinované pouze jediné pole tohoto elementu a to IP adresa serveru načtená přes proměnnou *WEB*.

### User Defined Variables

Konfigurační element pro definování proměnných, který byl popsán v GUI/Autorizačním testu. V tomto testu bylo zadefinováno 5 proměnných, viz obr. 4.34:

| User Defined Variables | |
|---|---|
| Name: | Value |
| WEB | 192.168.33.46 |
| SLOŽKA | Zátěžový test |
| DATUM | 2018-04-28T15:00:00.000Z |
| DATUM2 | 2018-04-28 |
| DATUM3 | 28.04.2018 |

Obr. 4.34: Zadefinované proměnné **User Defined Variables**

- **WEB** - webová adresa cílového testovacího prostředí. (serveru)
- **SLOŽKA** - složka, do které se ukládají výsledky testů.
- **DATUM** - první typ datumu, který využívá více HTTP požadavků.
- **DATUM2** - druhá typ datumu, který využívá více HTTP požadavků.
- **DATUM3** - třetí typ datumu, který využívá více HTTP požadavků.

### CSV Data Set Config

CSV Data Set Config se používá ke čtení řádků ze souboru a rozdělení řádků na proměnné. Pro tento test byl vytvořen soubor *Uzivatele.txt*, ve kterém je zadefinováno 6 uživatelů ve formě přihlašovacího jména a hesla, viz obr 4.35. Uživatelé používají role vedoucího operátora, operátora na místě a vzdáleného operátora. Jmeter tyto údaje čte v pořadí, v jakém jsou napsány a předává je HTTP požadavku pro přihlášení uživatele. Tím je zajištěno, že všechny následující požadavky po přihlášení budou simulovány s daným uživatelem.

```
onsite@nll.com,test
manager@nll.com,test
remote@nll.com,test
OpeMan@nll.com,test
OpeOn@nll.com,test
OpeRem@nll.com,test
```

Obr. 4.35: Zadefinovaní uživatelé v souboru *Uzivatele.txt*

### Ultimate Thread Group

Ultimate Thread Group (**Akce vykonane uzivatelem**) je plugin, který rozšiřuje nastavení vlákna Thread Group. Umožňuje nastavit mnohem realističtější průběh celého testu, což je pro zátěžové testy nezbytné. V tomto testu je vlákno nastaveno následovně, viz obr. 4.36:

Threads Schedule

| Start Threads Count | Initial Delay, sec | Startup Time, sec | Hold Load For, sec | Shutdown Time |
|---|---|---|---|---|
| 50 | 0 | 150 | 1200 | 150 |

Obr. 4.36: Nastavení vlákna **Akce vykonane uzivatelem**

- **Start Threads Count** - počet uživatelů simulace.
- **Initial Delay** - počáteční zpoždění.
- **Startup Time** - doba, za jakou dojde k nahrání všech uživatelů simulace (maximální nastavené zátěže).
- **Hold Load For** - označuje dobu, po kterou Jmeter udrží maximální zátěž.
- **Shutdown Time** - po skončení doby Hold Load For, Jmeter postupně ukončuje běh všech uživatelů za nastavenou dobu.

### Transaction Controller

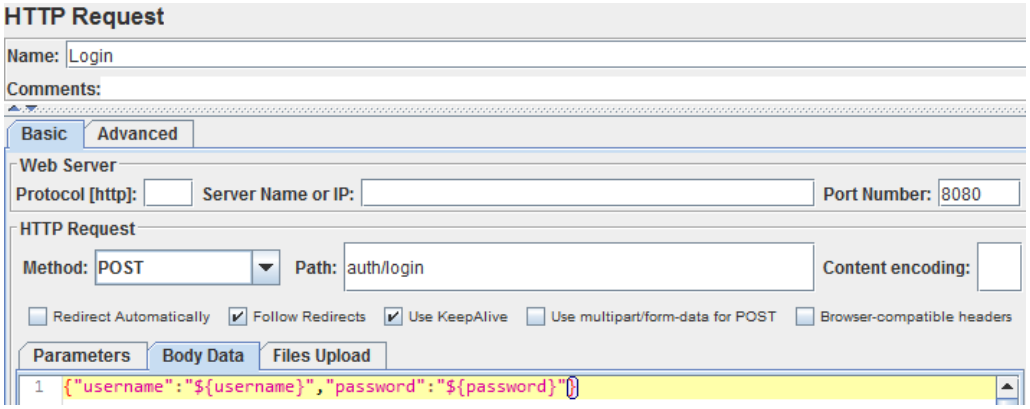
Transaction Controller umožňuje sjednotit všechny HTTP požadavky a změřit jejich celkovou odezvu. Aby zátěžové testy byly přesnější, je potřeba, aby nebyla měřena odezva pouze jednoho požadavku. Je nutné měřit odezvu všech zaslaných požadavků serveru, které jsou součástí nějaké vykonané akce např. zobrazení úvodní stránky. V testu je Transaction Controller použit u všech těchto akcí vykonaných uživatelem. V něm je zaškrtnuta pouze hodnota *Generate parent sample*. Tím se docílí, že Jmeter ve výsledcích vygeneruje pouze hlavní vzorek např. **Uvodni prihlasovaci stranka**, kde bude uvedena celková odezva. Také z hlediska přehlednosti v grafech a výsledcích je to nezbytné.

### HTTP Request

HTTP Request umožňuje odeslat HTTP požadavky webovému serveru. Nejčastěji používanými metodami je GET a POST. GET požaduje určitá data od webového serveru. POST předává data webovému serveru pro zápis. Jako příklad je uveden HTTP požadavek pro přihlášení uživatele, viz obr. 4.37. Všechny hodnoty byly ponechány ve výchozím nastavení kromě následujících:

- **Port Number** - číslo portu, na kterém webový server naslouchá.
- **Method** - určuje použitou metodu požadavku.

- **Path** - cesta ke zdroji dat.
- **Body Data** - řetězec dat, který je v požadavku předán webovému serveru.



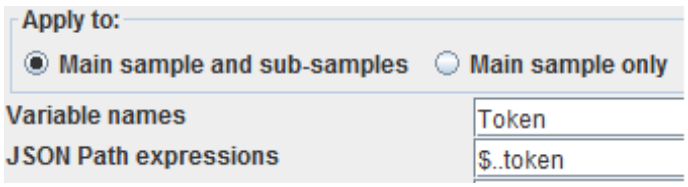

Obr. 4.37: Nastavení HTTP požadavku **Login**

Proměnné *${username}* a *${password}* označují uživatelské jméno a heslo. Tyto hodnoty Jmeter čte ze zmíněného souboru *Uzivatele.txt*.

### JSON Extractor

JSON Extractor umožňuje vytvářet výrazy, pomocí kterých Jmeter uloží vyžádaný řetězec znaků do proměnné. Používá se zejména u odpovědí typu JSON webového serveru na HTTP požadavky zaslané klientem. Příkladem je uložení autentizačního tokenu při přihlášení uživatele, viz obr. 4.38. Při odeslání požadavku pro přihlášení server pošle odpověď, ve které přidělí uživateli autentizační token. Tento token je nutné předávat všem následujícím HTTP požadavkům, protože označuje, že zadané uživatelské jméno a heslo byly správné. Pro zajištění této funkcionality je nastavení následovné:

- **Main sample and sub-samples** - zajistí, že token bude předán všem hlavním i dílčím vzorkům.
- **Variable names** - název proměnné.
- **JSON Path expressions** - výraz pro vyjmutí požadovaného řetězce znaků.




Obr. 4.38: Nastavení výrazu v **JSON Extractor**

**Uniform Random Timer**

Uniform Random Timer je časovač umožňující nastavit zpozdění (simulace přemýšlení) k napodobení reálného chování uživatele. Má 2 hodnoty k nastavení. Maximální velikost náhodně vygenerovaného zpoždění a konstantu, která se přičítá k vygenerované hodnotě. Tato zpoždění jsou v testu různá a byla sestavena na základě odpovědi od zákazníka.

**Použité naslouchače**

Následně budou popsány naslouchače, které zaznamenávají průběh testu a pomáhající s analýzou výsledků:

- **Aggregate Report** - vytváří tabulku, kde je zaznamenaný každý změřený požadavek. Zaznamenává různé informace o odezvě, počtu požadavků a chyb, propustnost atd. Podrobně byl popsán ve výkonnostním testu.
- **Hits per Second** - graf zaznamenávající celkový počet vygenerovaných HTTP požadavků cílených na webový server za sekundu.
- **Response Times Over Time** - graf zobrazuje průměrnou dobu odezvy každého vzorkovače.
- **Transactions per Second** - graf zobrazuje počet dokončených transakcí každého vzorkovače.
- **Active Threads Over Time** - graf zobrazuje průběh, jak se mění počet simulovaných uživatelů v průběhu testu.
- **Perfmon Metrics Collector** - umožňuje monitorovat stav serveru (využití CPU, sítě apod.).

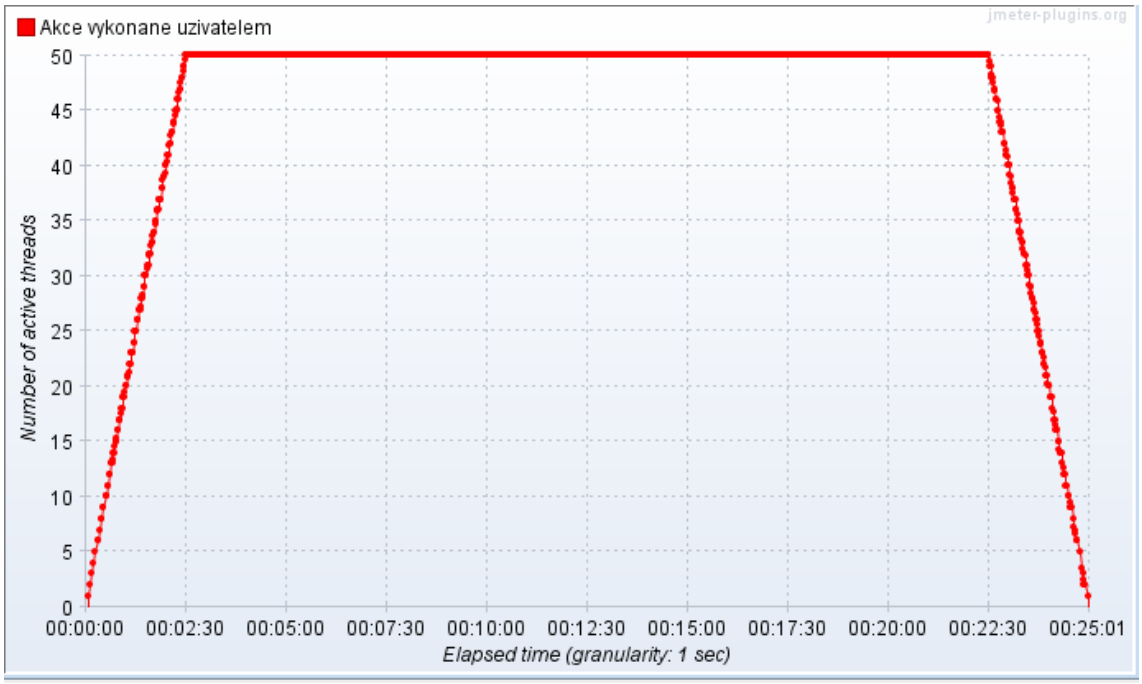
### 4.7.3 Analýza výsledků

Kapitola bude rozdělena na 3 části. První částí je analýza výsledků zátěžového testu, druhou analýza výsledků stability testu a třetí analýza výsledků stress testu. Výsledky naslouchače (**Aggregate Report**) budou stejně jako ve výkonnostním testu zkráceny o 4 sloupce z důvodu lepší čitelnosti. Naměřené grafy budou v analýze výsledků taktéž použity pouze ty, které byly nejpodstatnější pro daný test. Celé tabulky a všechny naměřené grafy budou k dispozici v příloze práce.

**Zátěžový test**

Zátěžový test byl dokončen za 25 minut a 1 sekundu. Na obr. 4.39 je vidět naměřený graf, který znázorňuje počet simulovaných uživatelů provádějících definovanou činnost v průběhu testu. Tento graf by měl potvrzovat, že průběh testu proběhl, tak jak byl nastaven ve vlákně **Akce vykonane uzivatelem**. Nástroj nejprve postupně

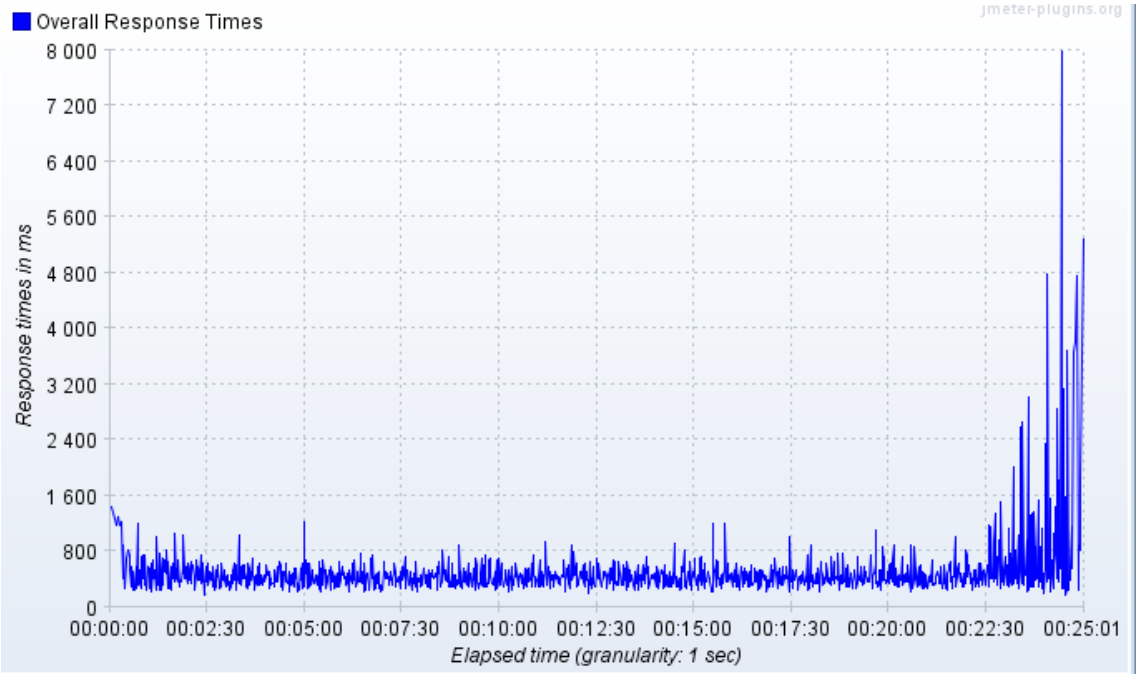
2 minuty a 30 sekund nahrával uživatele, dokud nedosáhl maximální zátěže 50 uživatelů. Poté 20 minut udržoval maximální zátěž. Po uplynutí této doby postupně během 2 minut a 30 sekund ukončoval simulaci všech uživatelů.



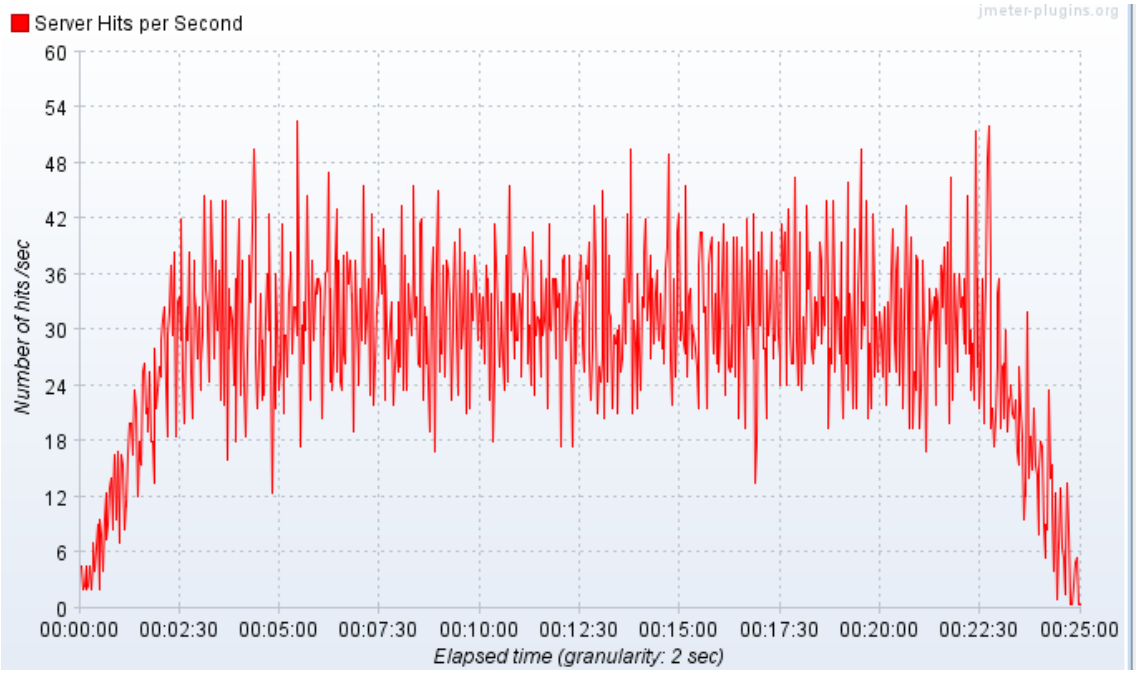

Obr. 4.39: Aktivní počet simulovaných uživatelů

Na obr. 4.40 je graf, který znázorňuje čas odezvy všech HTTP požadavků. Graf je v celém svém průběhu v pořádku až do času 22 minut a 30 sekund. V tento čas nástroj započal ukončovat simulaci všech uživatelů a odezva se prudce zvýšila. Tuto příčinu však pravděpodobně způsobuje nástroj Jmeter a nikoliv aplikace NDS. Při ukončování simulace uživatelů se některé HTTP požadavky nedokončí, což vede ke zvýšení celkové odezvy.

Na obr. 4.41 je graf, který znázorňuje celkový součet zpracovaných HTTP požadavků odeslaných serveru za sekundu. Tento graf by měl být do jisté míry podobný s grafem 4.39. Se zvyšujícím se počtem uživatelů se zvyšuje i počet HTTP požadavků zaslaných serveru. Pokud v určitém okamžiku nedochází k výraznému poklesu a tento pokles netrvá delší dobu, tak je vše v pořádku. V případě poklesu by to znamenalo, že testovaný systém je přetížen nebo nereaguje.




Obr. 4.40: Celkový čas odezvy HTTP požadavků




Obr. 4.41: Součet všech zpracovaných HTTP požadavků odeslaných serveru

Celkové výsledky jsou zaznamenány v tabulce naslouchače (**Aggregate Report**), viz obr. 4.42. Z tabulky lze vyčíst následující informace:

- Celkem (**TOTAL**) bylo vygenerováno 10653 HTTP požadavků.

- Maximální doba odezvy (**Max**) je velice vysoká u všech HTTP požadavků a byla způsobena v době, kdy nástroj započal postupně ukončovat simulaci uživatelů. S ohledem na parametr **99% Line** se však tato doba týkala pouze několika HTTP požadavků v řádu pár desítek.
- Nejvyšší průměrnou dobu odezvy (**Average**) měla transakce **Uvodni prihlasovaci stranka**. To je způsobeno tím, že při načtení úvodní stránky webový prohlížeč musí získat různé důležité zdroje (obrázky, skripty, soubory CSS apod.) potřebné ke správnému vykreslení webové stránky a její funkčnosti. Průměrná doba odezvy je 462 ms, což lze označit za velmi dobrý výsledek.
- Počet chyb (**Error**) v celém testu je 0%. To znamená, že všechny HTTP požadavky byly úspěšně zpracovány serverem i klientem.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max | Error ... |
|---|---|---|---|---|---|---|---|---|---|
| Uvodni prihlasovaci stranka | 1092 | 1361 | 1242 | 1834 | 1925 | 2066 | 1068 | 4709 | 0,00% |
| Prihlaseni uzivatele | 1091 | 560 | 440 | 550 | 761 | 2680 | 386 | 13951 | 0,00% |
| Zobrazeni General Overview | 1080 | 317 | 299 | 319 | 332 | 507 | 272 | 5296 | 0,00% |
| Zobrazeni Notifications | 1076 | 271 | 248 | 266 | 282 | 497 | 225 | 6631 | 0,00% |
| Zobrazeni IC Overview | 1068 | 281 | 247 | 263 | 274 | 577 | 223 | 6257 | 0,00% |
| Zobrazeni Commercial Schedule O... | 1058 | 469 | 438 | 469 | 491 | 756 | 403 | 9334 | 0,00% |
| Zobrazeni Power Orders Overview | 1051 | 592 | 581 | 608 | 622 | 850 | 536 | 3658 | 0,00% |
| Zobrazeni Power Profile Overview | 1049 | 204 | 195 | 208 | 216 | 293 | 172 | 3756 | 0,00% |
| Zobrazeni System Configuration | 1046 | 282 | 264 | 282 | 292 | 501 | 239 | 5841 | 0,00% |
| Odhlaseni uzivatele | 1042 | 257 | 222 | 321 | 470 | 1016 | 173 | 2418 | 0,00% |
| TOTAL | 10653 | 462 | 292 | 1152 | 1260 | 1919 | 172 | 13951 | 0,00% |

Obr. 4.42: Celkové výsledky zátěžového testu

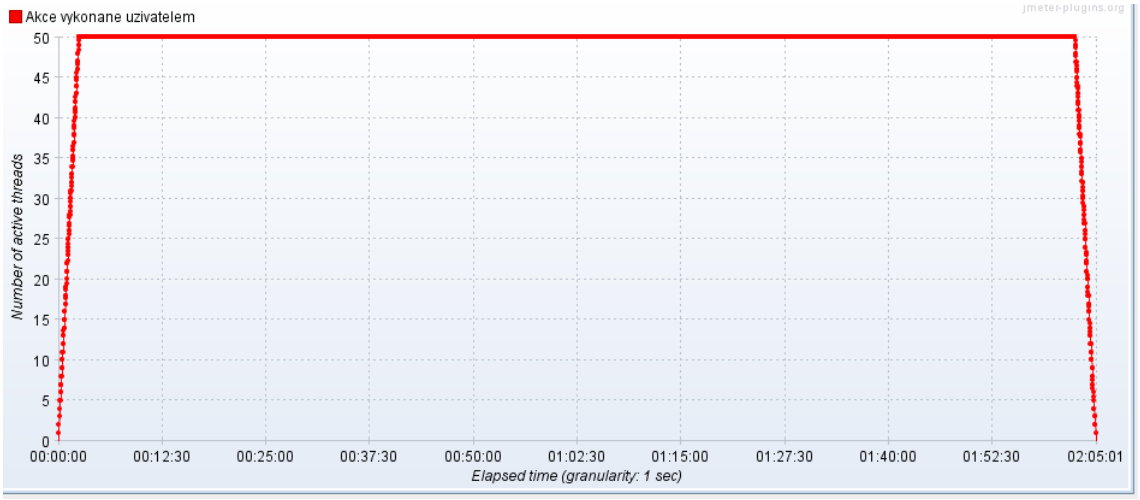
Využití procesoru (CPU) webovým serverem nevykazovalo žádnou neobvyklou činnost nebo náhlou zátěž a bylo v běžných pracovních hodnotách.

I přes skutečnost, že chybovost byla 0% a průměrná doba odezvy byla velmi dobrá (462 ms), musím označit tento test za neúspěšný. Důvodem je nepřijatelná doba odezvy ke konci testu. Ačkoliv příčinou je s největší pravděpodobností testovací nástroj Jmeter, tak by bylo vhodné daný problém diskutovat s vývojovým týmem a následně test opakovat.

### Stability test

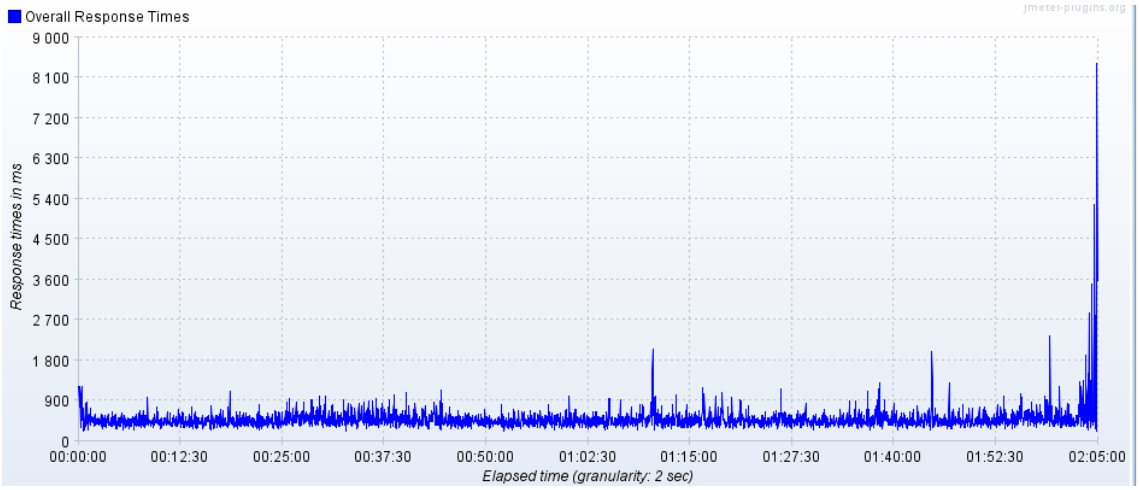
Stability test je ve výsledcích velice podobný testu zátěžovému. Test byl dokončen za 2 hodiny 5 minut a 1 sekundu. Na obr. 4.43 je vidět naměřený graf, který znázorňuje počet simulovaných uživatelů provádějící definovanou činnost v průběhu testu. Tento graf potvrzuje, že průběh testu proběhl tak, jak byl nastaven ve vlákně **Akce vykonane uzivatelem**. Nástroj nejprve postupně 2 minuty a 30 sekund nahrával uživatele, dokud nedosáhl maximální zátěže 50 uživatelů. Poté 2 hodiny udržoval

maximální zátěž. Po uplynutí této doby postupně během 2 minut a 30 sekund ukončoval simulaci všech uživatelů.



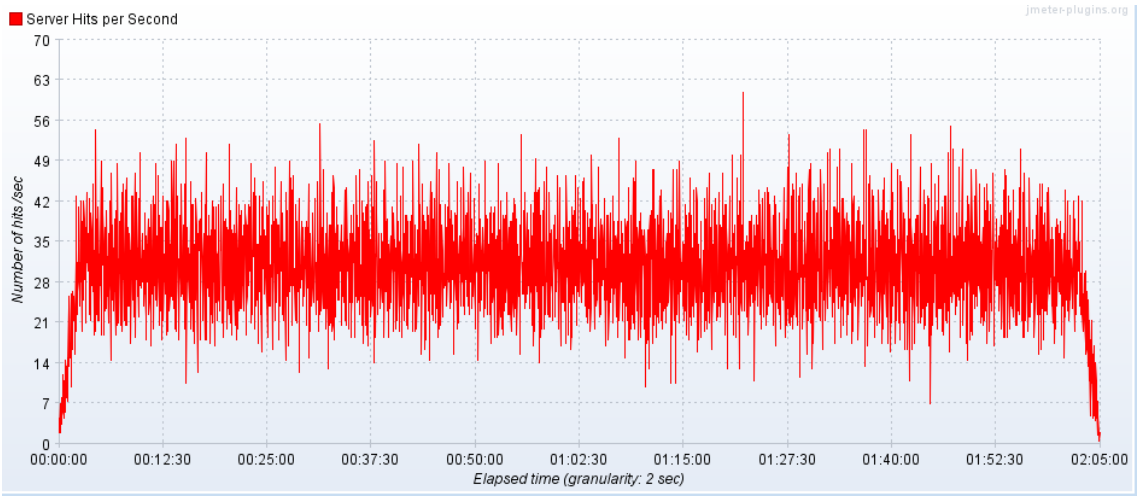

Obr. 4.43: Aktivní počet simulovaných uživatelů

Na obr. 4.44 je graf, který znázorňuje čas odezvy všech HTTP požadavků. Graf je v celém svém průběhu v pořádku a odezvy jsou v běžných hodnotách. Ovšem v čase 2 hodin, 2 minut a 30 sekund se odezva prudce zvýšila. Nastává zde tedy stejný problém jako u zátěžového testu, tudíž se nejednalo o jednorázový problém.




Obr. 4.44: Celkový čas odezvy HTTP požadavků

Na obr. 4.45 je graf, který znázorňuje celkový součet zpracovaných HTTP požadavků odeslaných serveru za sekundu. Graf je do jisté míry podobný grafu 4.43. Nedochází k výrazným poklesům, které by trvaly delší dobu, což znamená, že server nebyl v průběhu testu pod zátěží, kterou by nezvládl.




Obr. 4.45: Součet všech zpracovaných HTTP požadavků odeslaných serveru

Celkové výsledky jsou zaznamenány v tabulce naslouchače (**Aggregate Report**), viz obr. 4.46. Z tabulky lze vyčíst následující informace:

- Celkem (**TOTAL**) bylo vygenerováno 57132 HTTP požadavků.
- Maximální doba odezvy (**Max**) je velice vysoká u všech HTTP požadavků a byla způsobena v době, kdy nástroj započal postupně ukončovat simulaci uživatelů. S ohledem na parametr **99% Line** a velkému počtu HTTP požadavků se však tato doba týkala pouze několika požadavků v řádu stovek.
- Nejvyšší průměrnou dobu odezvy (**Average**) měla opět transakce **Uvodni prihlasovaci stranka** ze stejného důvodu jako u zátěžového testu. Průměrná doba odezvy je 482 ms, což je jen o 20 ms více než u testu zátěžového. Tento výsledek lze označit za velmi dobrý.
- Počet chyb (**Error**) v celém testu je 0%. To znamená, že všechny HTTP požadavky byly úspěšně zpracovány serverem i klientem.

| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max | Error % |
|---|---|---|---|---|---|---|---|---|---|
| Uvodni prihlasovaci stranka | 5738 | 1500 | 1265 | 1956 | 2485 | 4375 | 1052 | 7176 | 0,00% |
| Prihlaseni uzivatele | 5737 | 514 | 438 | 648 | 849 | 1608 | 325 | 20346 | 0,00% |
| Zobrazeni General Overview | 5727 | 329 | 299 | 329 | 445 | 856 | 272 | 8370 | 0,00% |
| Zobrazeni Notifications | 5717 | 274 | 248 | 275 | 372 | 725 | 224 | 7527 | 0,00% |
| Zobrazeni IC Overview | 5716 | 274 | 248 | 271 | 360 | 822 | 223 | 4815 | 0,00% |
| Zobrazeni Commercial Schedule O... | 5713 | 490 | 439 | 523 | 724 | 1615 | 402 | 13255 | 0,00% |
| Zobrazeni Power Orders Overview | 5706 | 641 | 582 | 722 | 889 | 1891 | 531 | 8870 | 0,00% |
| Zobrazeni Power Profile Overview | 5696 | 221 | 196 | 216 | 304 | 711 | 173 | 4607 | 0,00% |
| Zobrazeni System Configuration | 5693 | 292 | 263 | 289 | 406 | 852 | 234 | 7695 | 0,00% |
| Odhlaseni uzivatele | 5689 | 277 | 236 | 410 | 613 | 1176 | 62 | 3775 | 0,00% |
| TOTAL | 57132 | 482 | 297 | 1167 | 1319 | 2022 | 62 | 20346 | 0,00% |

Obr. 4.46: Celkové výsledky stability testu

Využití procesoru webovým serverem nevykazovalo žádnou neobvyklou činnost nebo náhlou zátěž a bylo v běžných pracovních hodnotách.

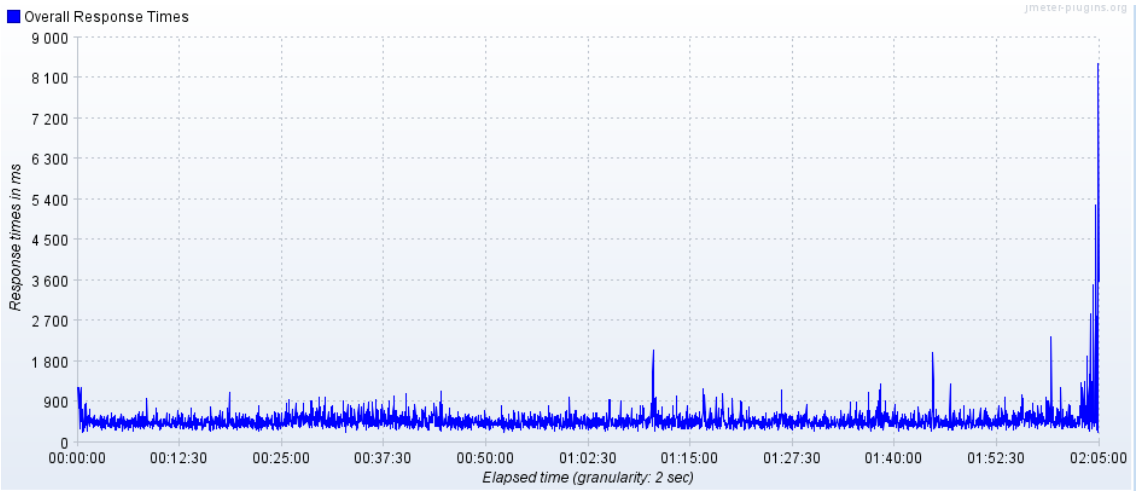
Z pohledu stability serveru, chybovosti 0% a průměrné odezvy 482 ms mohu tento test označit za úspěšný. Vysoké odezvy ke konci testu nejsou předmětem testování stability a tím pádem jsou pro tento typ testu zanedbatelné.

**Stress test**

Stress test byl dokončen za 45 minut a 6 sekund. Na obr. 4.47 je vidět naměřený graf, který znázorňuje počet simulovaných uživatelů provádějících definovanou činnost v průběhu testu. Graf potvrzuje, že průběh testu proběhl tak, jak byl nastaven ve vlákně **Akce vykonane uzivatelem**. Nástroj nejprve postupně 12 minut a 30 sekund nahrával uživatele, dokud nedosáhl maximální zátěže 250 uživatelů. Poté 20 minut udržoval maximální zátěž. Po uplynutí této doby postupně během 12 minut a 30 sekund ukončoval simulaci všech uživatelů. K nalezení bodu nasycení je důležité, aby nahrávání uživatelů bylo plynulé, a bylo tak možné v naměřených grafech stanovit bod nasycení.

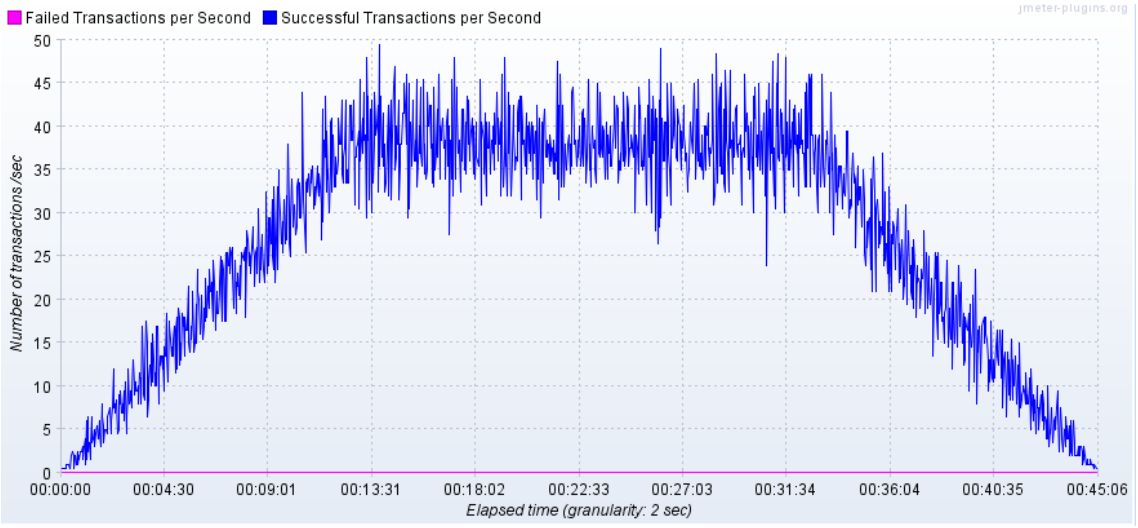

Obr. 4.47: Aktivní počet simulovaných uživatelů

Na obr. 4.48 je graf znázorňující čas odezvy všech HTTP požadavků. Graf je v celém svém průběhu v pořádku a odezvy jsou v běžných hodnotách. V čase 34. minut a 40. sekund se odezva začala postupně zvyšovat. I u tohoto testu tedy nastává stejný problém jako u zátěžového a stability testu.




Obr. 4.48: Celkový čas odezvy HTTP požadavků

Na obr. 4.49 je graf znázorňující počet zpracovaných transakcí za sekundu. Modrou barvou jsou označeny úspěšné transakce a růžovou neúspěšné. Průměrný počet úspěšných transakci je přibližně 38 a neúspěšné transakce nebyly zaznamenány žádné.




Obr. 4.49: Počet zpracovaných transakcí

Abychom zjistili, zda klient zvládá zpracovat tento počet transakcí, je potřeba prozkoumat propustnost (**Throughput**) v naslouchači **Aggregate Report**, viz obr. 4.50. Celková propustnost je 27,5 transakcí za sekundu. Je však nutné si uvědomit, že nástroj měřil propustnost i v době, kdy nebylo dosaženo nejvyšší zátěže 250 uživatelů. To znamená, že propustnost by byla v době nejvyšší zátěže mnohem vyšší. Pokud by propustnost nebyla dostatečná, byla by i mnohem vyšší odezva HTTP požadavků. Odezva však byla v době nejvyšší zátěže v přijatelných hodnotách.

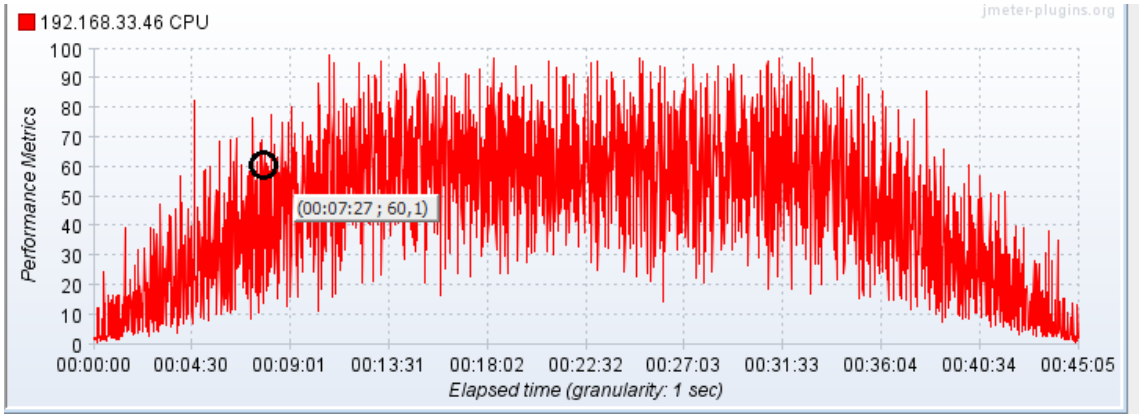
| Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Max | Error % | Throughput |
|---|---|---|---|---|---|---|---|---|---|---|
| Uvodni prihlasovaci stranka | 7552 | 2274 | 1958 | 3835 | 4457 | 5918 | 1075 | 10237 | 0,00% | 2,8/sec |
| Prihlaseni uzivatele | 7540 | 646 | 549 | 837 | 985 | 1554 | 356 | 18423 | 0,00% | 2,8/sec |
| Zobrazeni General Overview | 7501 | 398 | 348 | 524 | 633 | 867 | 278 | 9415 | 0,00% | 2,8/sec |
| Zobrazeni Notifications | 7479 | 329 | 286 | 426 | 519 | 759 | 124 | 8386 | 0,00% | 2,8/sec |
| Zobrazeni IC Overview | 7458 | 328 | 282 | 430 | 522 | 796 | 230 | 6646 | 0,00% | 2,8/sec |
| Zobrazeni Commercial Schedule... | 7435 | 598 | 509 | 758 | 898 | 1353 | 59 | 12928 | 0,00% | 2,8/sec |
| Zobrazeni Power Orders Overview | 7391 | 760 | 677 | 997 | 1158 | 1561 | 79 | 11193 | 0,00% | 2,8/sec |
| Zobrazeni Power Profile Overview | 7348 | 257 | 221 | 337 | 413 | 624 | 174 | 5162 | 0,00% | 2,8/sec |
| Zobrazeni System Configuration | 7332 | 347 | 300 | 449 | 545 | 872 | 241 | 8847 | 0,00% | 2,8/sec |
| Odhlaseni uzivatele | 7309 | 295 | 266 | 409 | 518 | 781 | 168 | 2028 | 0,00% | 2,8/sec |
| TOTAL | 74345 | 627 | 384 | 1245 | 2012 | 3997 | 59 | 18423 | 0,00% | 27,5/sec |

Obr. 4.50: Celkové výsledky stress testu

Ze zaznamenaných výsledků naslouchače (**Aggregate Report**) lze vyčíst následující informace:

- Celkem (**TOTAL**) bylo vygenerováno 74345 HTTP požadavků.
- Maximální doba odezvy (**Max**) je stejně jako u zátěžového a stability testu velice vysoká. Byla způsobena v době, kdy nástroj započal postupně ukončovat simulaci uživatelů. S ohledem na parametr **99% Line** a velkému počtu HTTP požadavků se tato doba týkala pouze několika požadavků v řádu stovek.
- Nejvyšší průměrnou dobu odezvy (**Average**) měla opět transakce **Uvodni prihlasovaci stranka** ze stejného důvodu jako u zátěžového a stability testu. Průměrná doba odezvy je 627 ms, což je vzhledem k velkému počtu uživatelů přijatelné a výsledek označit za velmi dobrý.
- Počet chyb (**Error**) v celém testu je 0%. To znamená, že všechny HTTP požadavky byly úspěšně zpracovány serverem i klientem.

Problém nastal ve využití procesoru serverem, viz obr 4.51.




Obr. 4.51: Využití procesoru [%] webovým serverem

Ve vyznačeném čase 7 minut a 27 sekund začalo být využití procesoru větší než 60%. Při srovnání s grafem 4.47 je čas roven přibližně 155 uživatelům. Využití

procesoru nadále roste se zvyšujícím se počtem uživatelů. Při dosažení maximálního počtu (250 uživatelů) dosahuje využití procesoru i přes 90%. Za předpokladu, že by mělo aplikaci NDS používat tolik uživatelů, tak je to nereálné z dlouhodobého hlediska. Aby byla zachována stabilita a spolehlivost systému, tak využití procesoru by mělo být v rozmezí 55% - 65%, což se rovná 155 uživatelům.

Stress test byl úspěšný a podařilo se vyhledat bod nasycení. Stanovit se ho podařilo až při analýze grafu, který znázorňuje využití procesoru serverem. Hranice maximálního počtu uživatelů, kteří by mohli využívat systém, aniž by docházelo k degradaci procesoru nebo vyšší odezvy, byla stanovena na 155 uživatelů.

### 4.7.4 Report výsledků

Z naměřených výsledků zátěžového, stability a stress testu mohu udělat následující závěr:

- **Zátěžový test** naměřil celkovou chybovost 0% a průměrná doba odezvy byla 462 ms, což je velmi dobrý výsledek. Test je však neúspěšný z důvodu nepřijatelné odezvy v době, kdy nástroj začal postupně ukončovat simulaci uživatelů.
- **Stability test** byl ve výsledcích velice podobný testu zátěžovému. Celkovou chybovost naměřil 0% a průměrná doba odezvy byla 482 ms. Nástroj naměřil vysoké odezvy i u tohoto testu při ukončování simulace uživatelů. Vysoké odezvy jsou však zanedbány, jelikož nejsou předmětem ověřování stability a test byl označen za úspěšný.
- **Stress test** pod zátěží 250 uživatelů naměřil celkovou chybovost 0%, průměrnou dobu odezvy 627 ms a propustnost 27,5 transakcí za sekundu, což jsou přijatelné hodnoty. Problém však nastal ve využití procesoru webovým serverem, které při 155 uživatelích přesahovalo 60%. Vyšší procento by mohlo být z dlouhodobého hlediska používání aplikace NDS rizikové. Test byl úspěšný a podařilo se tímto vyhledat bod nasycení, který byl stanoven na 155 uživatelů. Při tomto počtu uživatelů by měl systém zůstat stabilní a spolehlivý.

Výsledky testu budou zaznamenány a následně diskutovány s vývojovým týmem a konfiguračními manažery. Test bude spouštěn s každou nově vydanou verzí aplikace NDS jako u předchozích testů. Celý testovací plán, naměřené výsledky a grafy budou k dispozici v příloze této práce.

## 4.8 Bezpečnostní test

Cílem bezpečnostního testu je ověřit, že aplikace NDS neobsahuje nějaké závažné bezpečnostní zranitelnosti či nedostatky. Výsledky budou analyzovány a vyhodnoceny z hlediska závažnosti.

### 4.8.1 Analýza a návrh testu

Návrh bezpečnostního testu byl relativně komplikovaný. K návrhu dobrého a efektivního bezpečnostního testu je zapotřebí velmi zkušeného testera s mnohaletou praxí v oblasti bezpečnosti webových aplikací. Tester by měl také umět využít každou nalezenou zranitelnost a využít ji po další testování.

Aplikaci NDS bude v pozdější fázi testovat náš specializovaný bezpečnostní tým sídlící v Praze. Nicméně mě byl zadán úkol, že mám provést prvotní bezpečnostní testování aplikace. Vzhledem k tomu, že natolik rozsáhlé znalosti a zkušenosti nemám, poslal jsem e-mail bezpečnostnímu týmu. V e-mailu jsem sdělil informace o projektu, aplikaci NDS a svých zkušenostech s prosbou o zvolení nejvhodnějšího způsobu k otestování aplikace z pohledu bezpečnosti. Bylo mi doporučeno zvolit automatizované skenovací nástroje zranitelností, kde není nutná natolik hluboká znalost v oblasti bezpečnosti. Tímto způsobem budu schopen z určité části aplikaci NDS efektivně otestovat proti závažným zranitelnostem webových aplikací.

Na základě vlastních zkušeností jsem vybral 3 nekomerční nástroje (Vega Vulnerability Scanner, OWASP Zed Attack Proxy a Nessus Vulnerability Scanner), které jsem vyhodnotil jako nejvhodnější a nejpřesnější. Výsledky jednotlivých nástrojů budou analyzovány a vyhodnoceny podle závažnosti. Součástí vyhodnocení bude i doporučení k opravě nalezených zranitelností. Výsledky budou poté poslány bezpečnostnímu týmu, aby mohl navrhnout na jejich základě svůj bezpečnostní test.

Z interní dokumentace bylo také zjištěno, že aplikace NDS bude používat v produkci dvoufaktorovou autentizaci. Jedná se metodu, která obsahuje 2 nezávislé způsoby, jak ověřit totožnost uživatele, který se chce přihlásit do systému. Aplikace NDS bude využívat kombinaci USB tokenu (něco, co uživatel vlastní) a hesla (něco, co uživatel zná). Všichni uživatele budou mít vlastní přihlašovací jméno, heslo a USB token s nainstalovaným certifikátem pro přístup do aplikace. Certifikát bude využíván pro podpis zabezpečené komunikace HTTPS (Hypertext Transfer Protocol Secure) mezi uživatelským počítačem a aplikací NDS. Bez certifikátu se uživateli nezobrazí úvodní přihlašovací stránka.

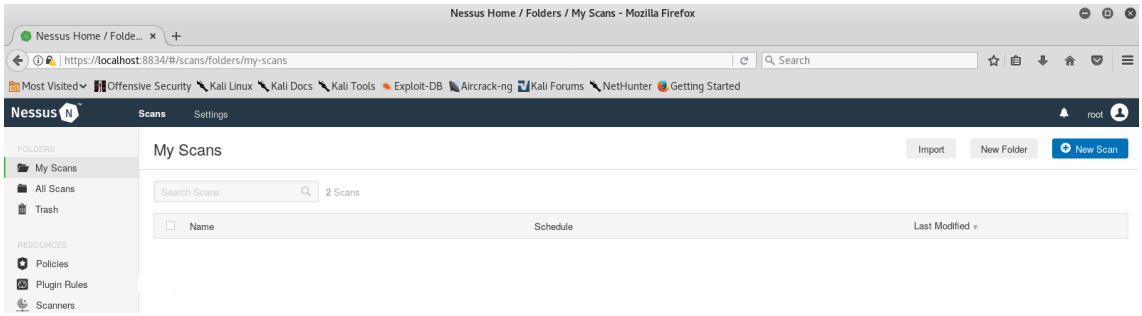
### 4.8.2 Implementace testu

Samotná implementace tohoto testu není příliš složitá. Nutné je nainstalovat nástroj Oracle VM VirtualBox a prostřednictvím něho zprovoznit distribuci Kali Linux, která je určená pro bezpečnostní testy. V Kali Linux je již nástroj OWASP ZAP nainstalován a bylo potřeba doinstalovat nástroje Nessus a Vega. Nástroj Vega jsem byl nakonec nucen zprovoznit na operačním systému Windows, jelikož se mi nepodařilo nainstalovat ho na Kali Linux. Instalace nástroje Nessus proběhla bez

problému. Vzhledem k tomu, že na produkci má aplikace NDS používat zabezpečenou komunikaci HTTPS, budou všechny 3 testy zaměřeny na testovací prostředí, kde je zkušebně tato komunikace aplikována.

### Nessus Vulnerability Scanner

Nessus se spouští přes rozhraní příkazového řádku příkazem */etc/init.d/nessusd start*. Poté stačí zadat do URL webovou adresu *https://localhost:8834/*. Po přihlášení se zobrazí úvodní stránka, viz obr. 4.52.

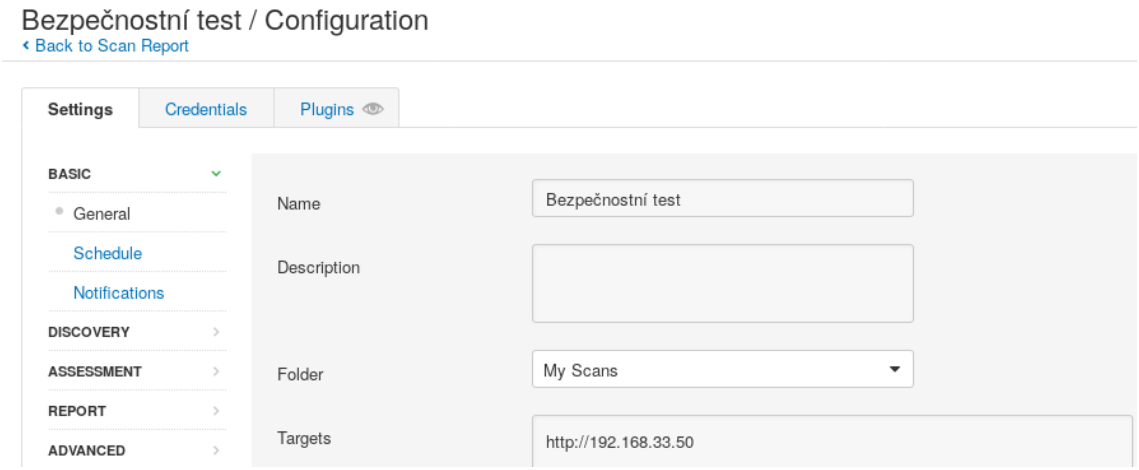



Obr. 4.52: Úvodní stránka nástroje **Nessus**

Pro vytvoření nového skenu stačí kliknout na tlačítko *New Scan*. Dále zvolit typ skenu *Basic Network Scan*, který provede kompletní testování cílového systému. Následně se zobrazí konfigurace testu, viz obr. 4.53, která byla vyplněna následovně:

- **Name** - název testu.
- **Folder** - název složky pro ukládání výsledků.
- **Targets** - IP adresa cílového systému.
- **Discovery -> Scan Type** - nastavení týkající se skenování portů. Typ skenu byl zvolen *Port scan (common ports)*. To znamená, že Nessus bude skenovat přibližně 4 790 běžně používaných portů.
- **Assesment -> Scan Type** - nastavení týkající se způsobu, jakým budou zranitelnosti identifikovány a jaké zranitelnosti budou identifikovány. Typ skenu byl zvolen *Scan for all web vulnerabilities (complex)*. To znamená, že Nessus bude používat způsoby k identifikaci všech nejrizikovějších zranitelností webových aplikací.

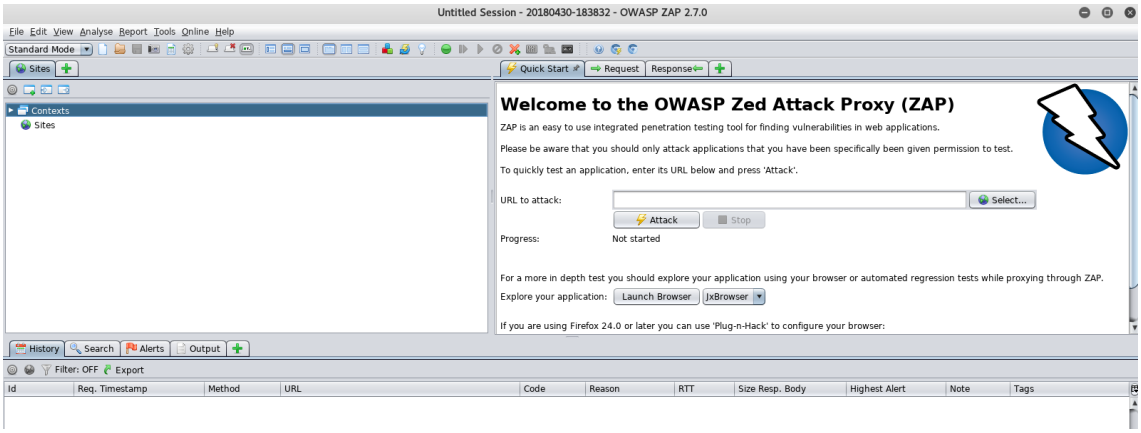
Ostatní hodnoty byly ponechány ve výchozím nastavení. Tímto je test připraven ke spuštění.




Obr. 4.53: Konfigurace bezpečnostního testu

### OWASP Zed Attack Proxy

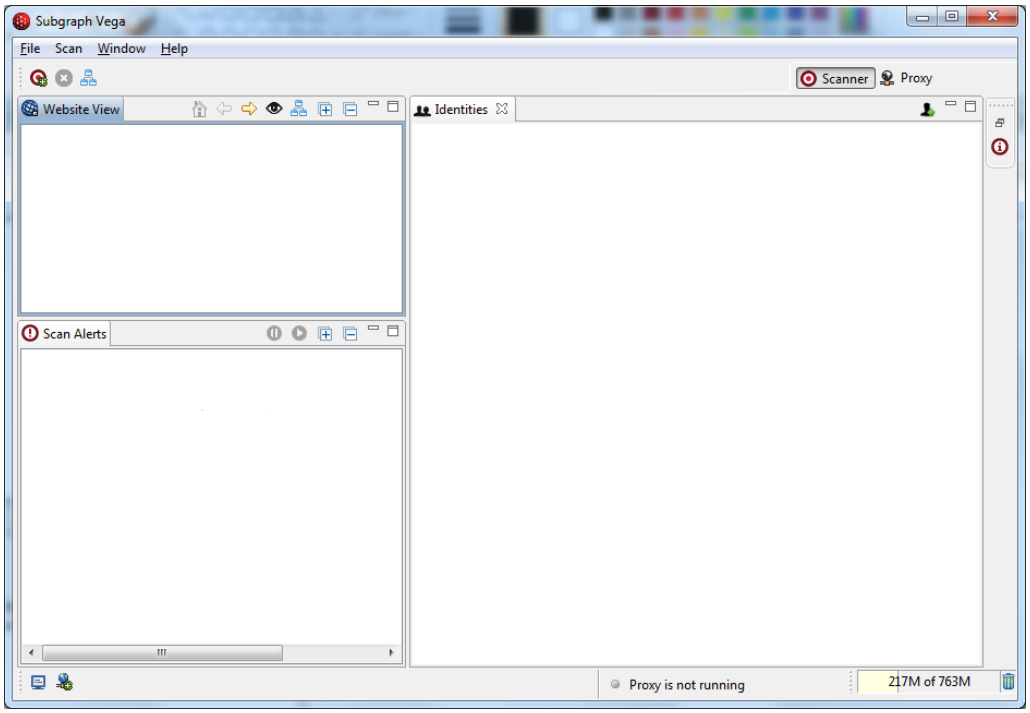
OWASP ZAP se spouští přes menu v hlavní liště *Applications -> Web Application Analysis -> OWASP ZAP*. Po spuštění se zobrazí úvodní obrazovka, viz obr. 4.54. Zde stačí vyplnit webovou adresu prostředí *https://192.168.33.50* („URL to attack“), na které bude vykonán útok. Dále přepnout vlevo nahoře mód testu na *Attack Mode*. To znamená, že pokud nástroj vyhledá nějaké další odkazy mimo výše definovaný rozsah např. *https://192.168.33.50/robots.txt*, tak je započítá do testu. Tímto je test připraven ke spuštění.




Obr. 4.54: Úvodní obrazovka nástroje **OWASP ZAP**

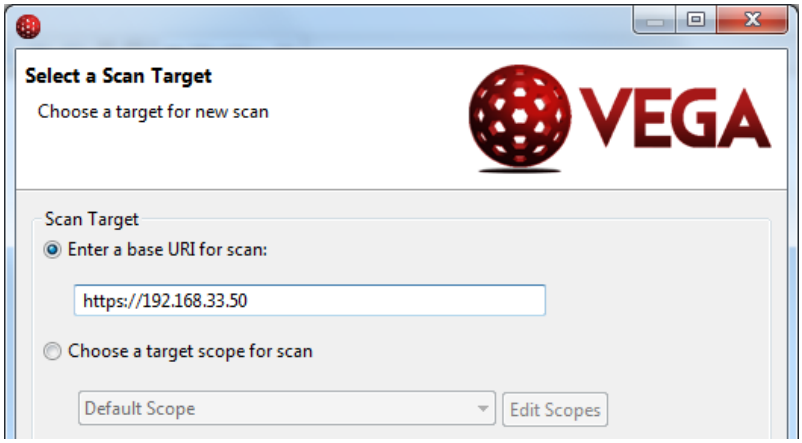
### Vega Vulnerability Scanner

Vega Vulnerability Scanner se spouští běžným způsobem přes spustitelný soubor *Vega.exe*. Po spuštění je zobrazena úvodní obrazovka, viz obr. 4.55.



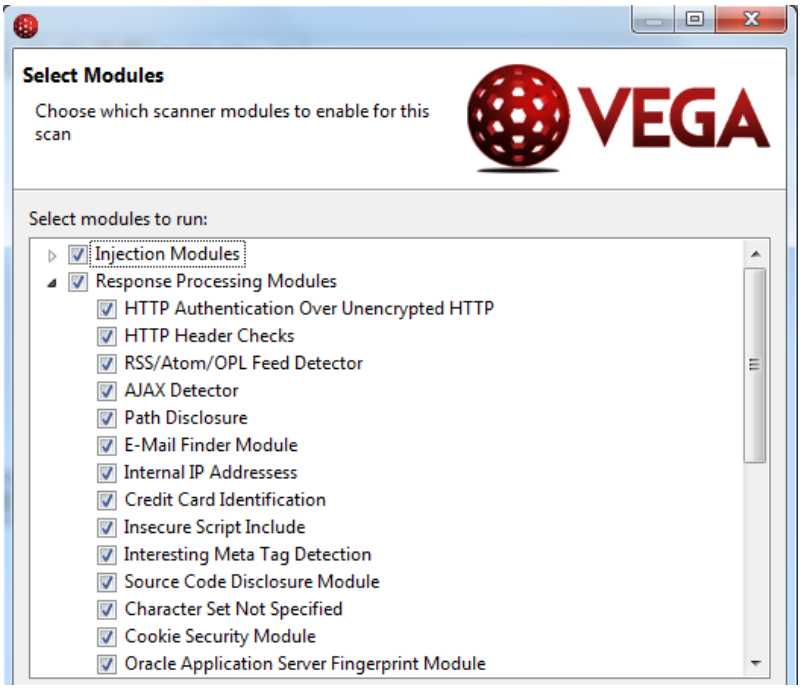

Obr. 4.55: Úvodní obrazovka nástroje **Vega Vulnerability Scanner**

Pro vytvoření nového skenu je potřeba zvolit *Scan -> Start New Scan*. Otevře se nastavení testu, kde je nutné zadat URI, tedy webovou adresu cíle, viz obr. 4.56.




Obr. 4.56: Konfigurace bezpečnostního testu

Posledním krokem je výběr zranitelností, které má nástroj započítat do testování, viz obr. 4.57. Zde byly zvoleny všechny zranitelnosti. Po kliknutí na *Finish* je nastavení u konce a test je připraven ke spuštění.




Obr. 4.57: Specifikování zranitelností

### 4.8.3 Analýza výsledků

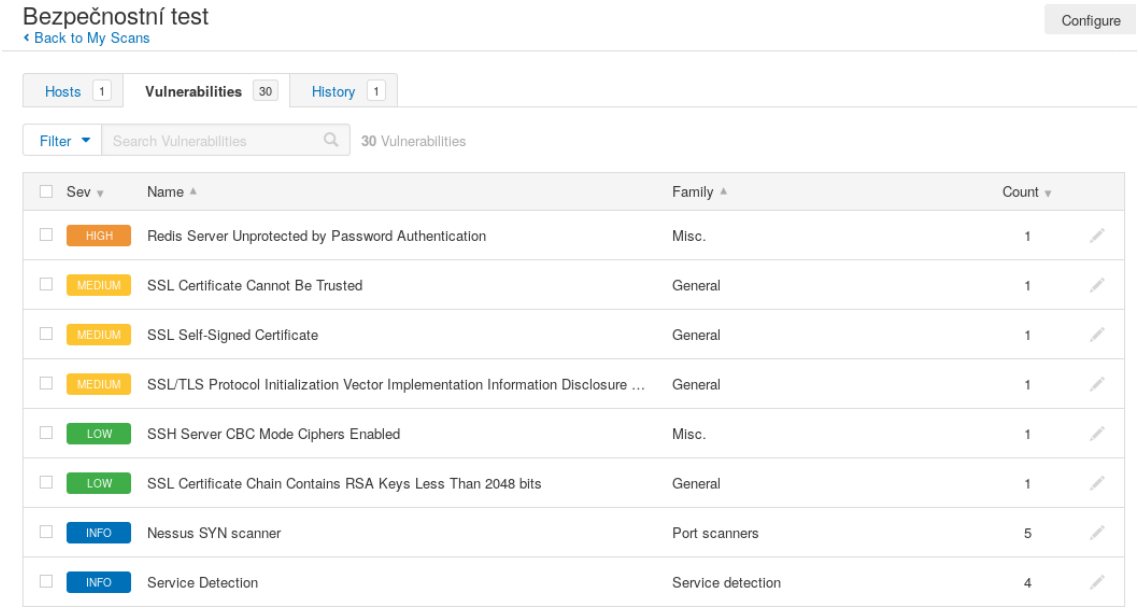
Tato kapitola bude rozdělena na 3 části. První část bude analýza výsledků nástroje Nessus, druhá analýza výsledků nástroje OWASP ZAP a třetí analýza výsledků nástroje Vega.

**Nessus Vulnerability Scanner**

Nástroj Nessus detekoval celkem 30 (40) zranitelností, viz obr. 4.58. Ty jsou rozděleny podle závažnosti následovně:

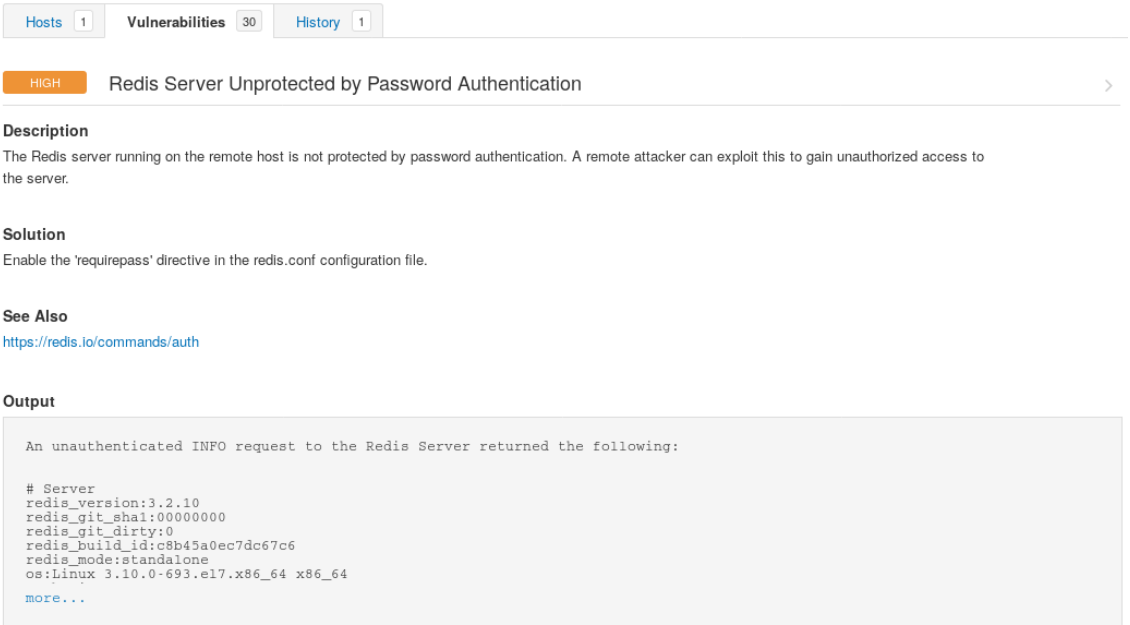
- Vysoká (**High**) - detekován 1 typ zranitelnosti
- Střední (**Medium**) - detekovány 3 typy zranitelností
- Nízká (**Low**) - detekovány 2 typy zranitelností
- Informativní (**Info**) - detekováno 24 typů zranitelností

Zranitelnosti, které jsou stejného typu např. **Nessus SYN scanner**, jsou započítány jako jedna zranitelnost. Na první pohled se může zdát, že se jedná o hodně zranitelností, ale je třeba si uvědomit, že zranitelnost typu *Informativní* ve své podstatě není zranitelnost. Jsou to informace (otevřené porty, verze systému/serveru, port na jakém servery naslouchají atd.), které mohou být užitečné při mnohem rozsáhlejším a detailnějším bezpečnostním testu. Tyto informace je také velice těžké zneužít, pokud útočník není specialista na bezpečnost.




Obr. 4.58: Část výpisu detekovaných zranitelností nástrojem **Nessus**

Při zvolení libovolné zranitelnosti např. *Redis Server Unprotected by Password Authentication* se zobrazí detailnější popis, viz obr. 4.59.




Obr. 4.59: Detail zranitelnosti **Redis Server Unprotected by Password Authentication**

Z obrázku lze zjistit na základě popisu o jakou zranitelnost se jedná, jaké by mohlo být řešení k opravě a výstup, prostřednictvím kterého byla zranitelnost detekována.

Analyzovány budou zranitelnosti všechny kromě informativních. Vzhledem k velkému počtu informativních jsem vybral pouze jednu, která by na produkci mohla způsobit problémy. Podrobný výpis všech zranitelností bude ve vygenerovaném souboru nástrojem Nessus v příloze. Na základě výsledků jsem vytvořil seznam zranitelností, kde je každá zranitelnost popsána a vyhodnocena:

- **Redis Server Unprotected by Password Authentication -> Závažnost -> Vysoká** - Nástroj detekoval otevřený port 6379, na kterém je zprovozněna Redis Cache (mezipamět serveru Redis) a přístup do ní není chráněný heslem. Jediný dostupný port z vnější sítě na produkci bude zabezpečený 443 – SSL (Secure Sockets Layer), zbytek bude chráněn firewallem. V praxi to znamená, že na produkci tuto zranitelnost nástroj nenajde. Řešením by bylo v Redis serveru zapnout autentizaci heslem.
- **SSL Certificate Cannot Be Trusted -> Závažnost -> Střední** - Nástroj detekoval nedůvěryhodný certifikát. Jedná se však o testovací certifikát, na produkčním prostředí bude použit certifikát podepsaný certifikační autoritou.
- **SSL Self-Signed Certificate -> Závažnost -> Střední** - Nástroj detekoval, že certifikát sám sebe podepíše. Souvisí se zranitelností uvedenou výše. Zranitelnost byla nalezena, protože se jedná dočasně o testovací certifikát.
- **SSL/TLS Protocol Initialization Vector Implementation Information Disclosure Vulnerability -> Závažnost -> Střední** - Nástroj detekoval, že webový server (Nginx) používá starou verzi (1.0) protokolu TLS (Transport Layer Security), který má bezpečnostní nedostatky. Řešením je webový server překonfigurovat, aby používal novější verzi TLS 1.2.
- **SSH Server CBC Mode Ciphers Enabled -> Závažnost -> Nízká** - Nástroj detekoval, že SSH (Secure Shell) server podporuje šifrovací algoritmus CBC (Cipher Block Chaining) na portu 22. Tento algoritmus nemá příliš vysokou bezpečnost, jelikož útočník by mohl přenášené informace zpětně detekovat ve formě nezašifrovaného textu. SSH spojení bude na produkci povoleno pouze v rámci vnitřní sítě, jediný dostupný port pro uživatele bude 443 (SSL), zbytek bude chráněn firewallem.
- **SSL Certificate Chain Contains RSA Keys Less Than 2048 bits -> Závažnost -> Nízká** - Nástroj detekoval, že certifikační klíč RSA obsahuje méně než 2048 bitů. Zranitelnost opět souvisí s testovacím certifikátem, na produkčním prostředí bude použit certifikát s RSA klíčem o minimálně délce 2048 bitů.
- **Nessus SYN Scanner -> Závažnost -> Informativní** - Nástroj detekoval, že je možné identifikovat, které TCP porty jsou otevřené. Jediný dostupný port z vnější sítě na produkci bude 443 – SSL, zbytek bude chráněn firewallem. Nástrojem doporučený IP filtr asi nebude řešením, protože nebudeme znát

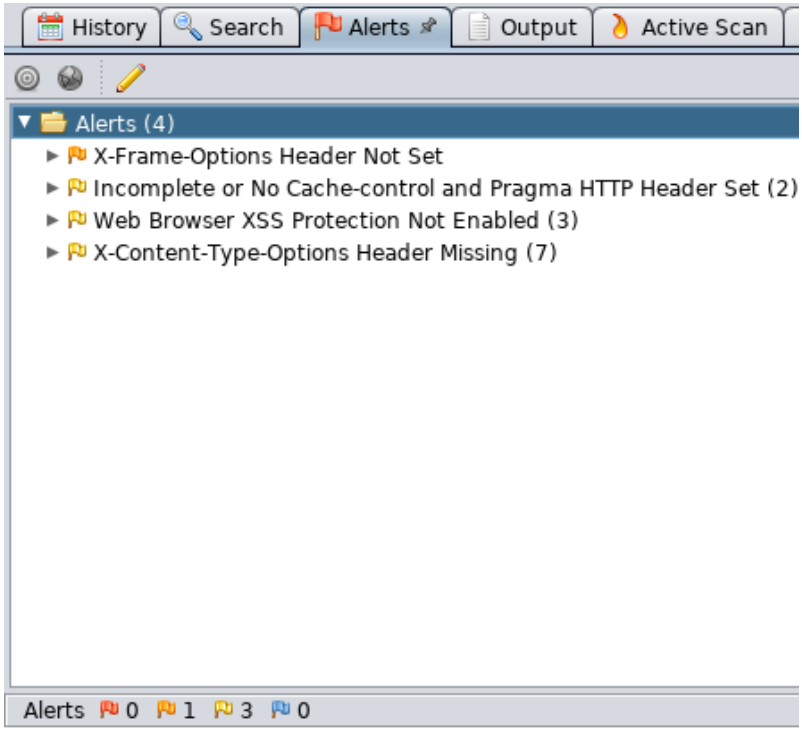
IP adresy všech uživatelů. Tento problém bude diskutován se zákazníkem a popřípadě se IP filtr vytvoří v rámci VPN.

**OWASP Zed Attack Proxy**

Nástroj OWASP ZAP detekoval celkem 4 (13) typy zranitelností, viz obr. 4.60. Ty jsou rozděleny podle závažnosti následovně:

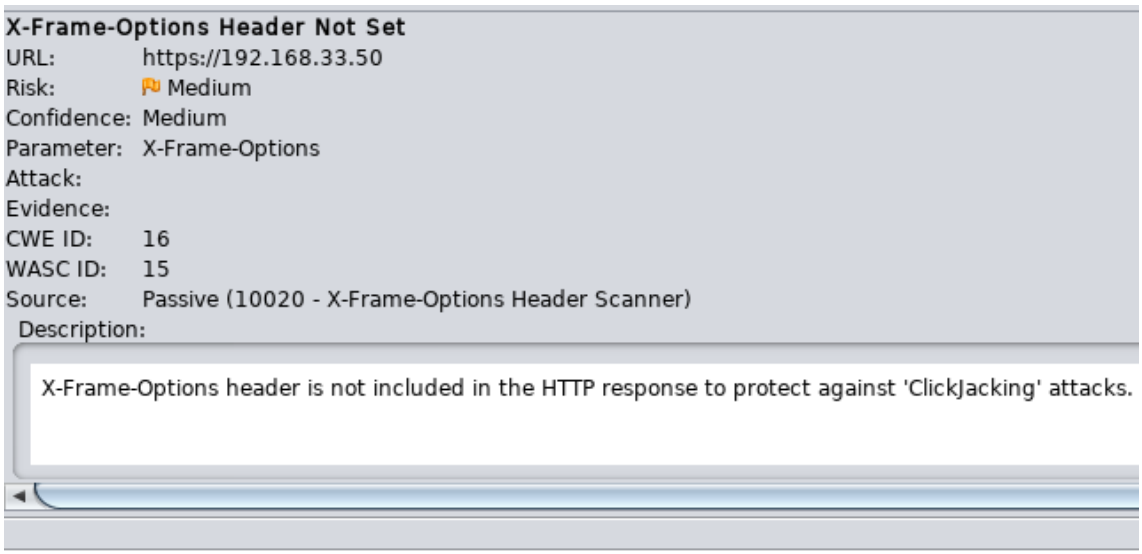
- Střední (**Medium**) - detekován 1 typ zranitelnosti
- Nízká (**Low**) - detekovány 3 typy zranitelností

Platí pravidlo jako u nástroje Nessus, že zranitelnost, která je stejného typu je započítána jako jedna zranitelnost. Čísla v závorkách určují kolika požadavků se zranitelnost týká.




Obr. 4.60: Detekované zranitelnosti nástrojem **OWASP ZAP**

Nástroj OWASP ZAP detekoval méně zranitelností než Nessus. Po zvolení zranitelnosti *X-Frame-Options Header Not Set* se zobrazí detailnější popis, viz obr. 4.61. Z toho lze zjistit o jakou zranitelnost se jedná, jaké by mohlo být řešení k opravě a URL odkazy na dokumentaci společnosti *OWASP*, kde je zranitelnost vysvětlena detailněji.




Obr. 4.61: Část detailu zranitelnosti **X-Frame-Options Header Not Set**

Podrobný výpis všech zranitelností bude ve vygenerovaném souboru nástroje OWASP ZAP v příloze. Následně budou jednotlivé typy zranitelností popsány a vyhodnoceny:

- **X-Frame-Options Header Not Set -> Závažnost -> Střední** - Nástroj detekoval, že zdroj HTTP odpovědi nenastavil hlavičku X-Frame-Options. Tato hlavička pomáhá s nastavením pravidel ohledně domén a zmírňuje riziko útoku „clickjacking“. Řešením je přidání X-Frame-Options hlavičky do záhlaví všech nalezených HTTP odpovědí.
- **Incomplete or No Cache-control and Pragma HTTP Header Set (2) -> Závažnost -> Nízká** - Nástroj detekoval, že hlavička pro kontrolu mezipaměti (cache-control) a pragma nebyla správně nastavena nebo chybí u HTTP požadavku. Toto umožňuje prohlížeči a proxy serverům ukládat do mezipaměti obsah. Řešením je přidání těchto hlaviček do záhlaví HTTP požadavků.
- **Web Browser XSS Protection Not Enabled (3) -> Závažnost -> Nízká** - Nástroj detekoval, že ochrana proti XSS (Cross-site scripting) není povolena nebo je zakázána u HTTP odpovědí. Jedná se metodu narušení webových stránek využitím bezpečnostních chyb ve skriptech. Řešením je přidání X-XSS-Protection hlavičky do záhlaví HTTP odpovědí.
- **X-Content-Type-Options Header Missing (7) -> Závažnost -> Nízká** - Nástroj detekoval, že hlavička X-Content-Type-Options nebyla nastavena na hodnotu „nosniff“. Pokud tato hodnota není nastavena, tak umožňuje útočníkovi přepisovat typ obsahu HTTP odpovědi. To znamená, že s hodnotou „nosniff“ webový prohlížeč vykreslí obsah tak, jak určil server např. text/html a žádný jiný. Řešením je přidání X-Content-Type-Options hlavičky do záhlaví

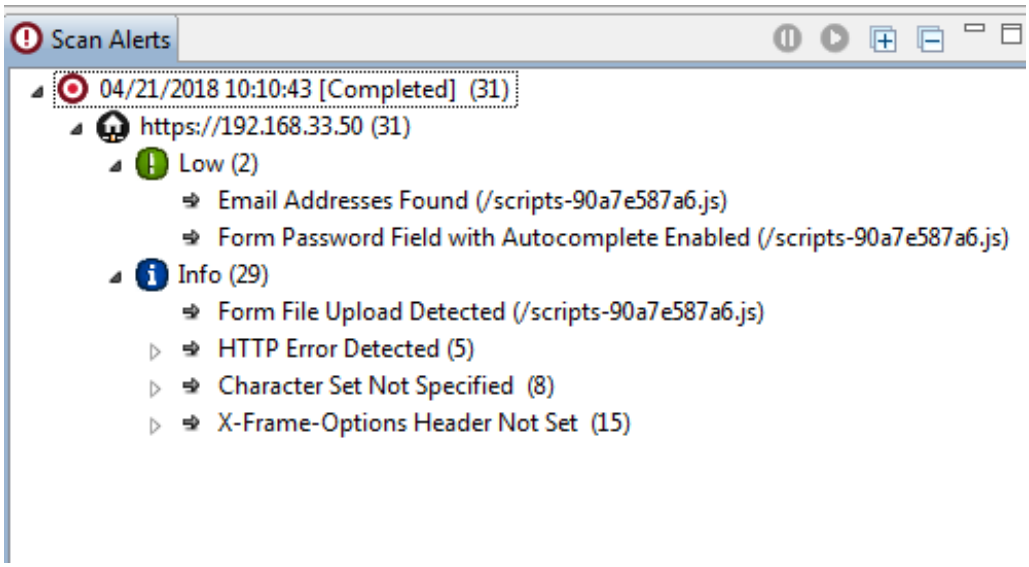
HTTP odpovědí.

**Vega Vulnerability Scanner**

Nástroj Vega detekoval celkem 6 (31) typů zranitelností, viz obr. 4.62. Ty jsou rozděleny podle závažnosti následovně:

- Nízká (**Low**) - detekovány 2 typy zranitelností
- Informativní (**Info**) - detekovány 4 typy zranitelností

Opět platí pravidlo jako u nástrojů Nessus a OWASP ZAP, že zranitelnost, která je stejného typu je započítána jako jedna zranitelnost. Čísla v závorkách určují kolika požadavků se zranitelnost týká.

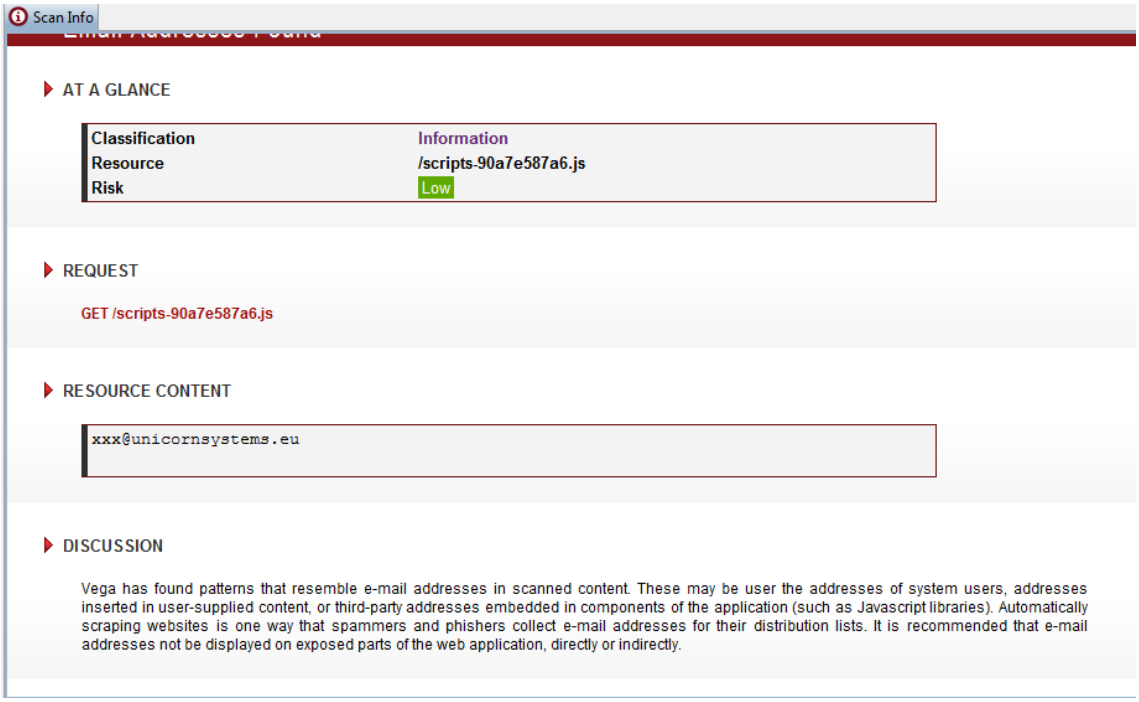



Obr. 4.62: Detekované zranitelnosti nástrojem **Vega**

Nástroj Vega detekoval podobné množství typů zranitelností jako nástroj OWASP ZAP. Zranitelnost *X-Frame-Options Header Not Set* detekovaly oba nástroje totožně. Po zvolení zranitelnosti *Email Addresses Found* se zobrazí detailnější popis, viz obr. 4.61. Z toho lze zjistit typ HTTP požadavku, obsah zdroje, o jakou zranitelnost se jedná, jaké riziko zranitelnost představuje, jaké by mohlo být řešení k opravě a URL odkazy na dokumentace s detailnějšími informacemi. Ačkoliv je Vega velmi efektivní nástroj, má jednu velkou nevýhodu, jelikož nepodporuje vytváření reportů. Z tohoto důvodu budou v příloze obrázky od každého typu detekované zranitelnosti nástrojem Vega. Jednotlivé typy zranitelností budou následně popsány a vyhodnoceny:

- **Email Addresses Found -> Závažnost -> Nízká** - Nástroj detekoval v aplikaci NDS e-mailovou adresu. Zobrazení e-mailové adresy na hlavní stránce

aplikace je vyžadováno zákazníkem z důvodu systémové podpory. Aplikace nebude volně dostupná z internetu, pouze při použití USB tokenu. To znamená, že útočníkem by musel být pouze někdo z vnitřní infrastruktury zákazníka.




Obr. 4.63: Část detailu zranitelnosti **Email Addresses Found**

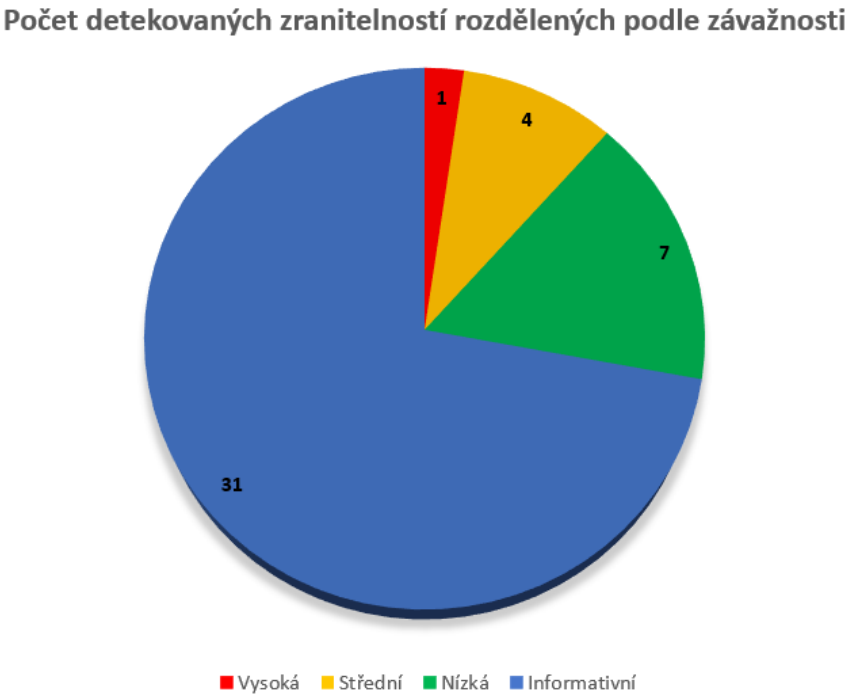
- **Form Password Field with Autocomplete Enabled -> Závažnost -> Nízká** - Nástroj detekoval, že automatické vyplnění hesla při zadání přihlašovacího jména není vypnuto. To znamená, že webové prohlížeče ukládají hesla do paměti, které může následně útočník získat. Aplikace NDS nebude volně dostupná z internetu, pouze při použití USB tokenu. Pokud by s tím nebyl zákazník spokojen, je možností tuto funkcionalitu i přesto vypnout. Toto řešení však může způsobit další problémy, jelikož každý typ webového prohlížeče reaguje poté svým vlastním způsobem při vypnutí této funkce.
- **Form File Upload Detected -> Závažnost -> Informativní** - Nástroj detekoval formulář pro nahrávání souborů. V aplikaci NDS se jedná o formulář nahrávání souborů v minifikovaném JavaScript souboru (účelem minifikace je optimalizovat velikost výsledného souboru obsahujícího zdrojový kód) – zdrojový kód frontend (část webu, kterou vidí návštěvník stránek) aplikace. Tento formulář nebude bez použití USB tokenu dostupný.
- **HTTP Error Detected (5) -> Závažnost -> Informativní** - Nástroj detekoval, že odpověď na poslaný HTTP požadavek byl ve formě chyby *<title>403 Forbidden</title>*. Ve všech případech se jedná o správně nastavenou funkci-

onalitu při požadavku na dané URL adresy.

- **Character Set Not Specified (8) -> Závažnost -> Informativní** - Nástroj detekoval, že zdroj HTTP odpovědi nespecifikoval znakovou sadu. Při nespecifikování znakové sady může webový prohlížeč vytvářet své předpoklady, jaká sada je použita na základě obsahu odpovědi. Toho může využít útočník a cíleně přinutit prohlížeč k zobrazení škodlivého obsahu. Řešením je přidání informace o znakové sadě do záhlaví všech nalezených HTTP odpovědí.
- **X-Frame-Options Header Not Set (15) -> Závažnost -> Informativní** - Nástroj detekoval, že zdroj HTTP odpovědi nenastavil hlavičku X-Frame-Options. Tato hlavička pomáhá s nastavením pravidel ohledně domén (odkud odpověď přišla) a zmírňuje riziko útoku „clickjacking“. Řešením je přidání X-Frame-Options hlavičky do záhlaví všech nalezených HTTP odpovědí.

### 4.8.4 Report výsledků

Na základě výsledků nástrojů Nessus, OWASP ZAP a Vega byl zhotoven graf, viz obr. 4.64. V grafu je znázorněno kolik je celkem typů zranitelností v aplikaci NDS. Pokud bych vzal do úvahy, že informativní zranitelnosti nejsou natolik podstatné, tak mohu prohlásit, že v současné době je téměř 75% aplikace z pohledu bezpečnosti v pořádku.




Obr. 4.64: Znázornění celkového počtu zranitelností (Autor)

Nástroj Nessus detekoval celkem 30 typů zranitelností, Vega 6 a OWASP ZAP 4. Nessus však označoval i za zranitelnosti informace, které jsou více informativní než zneužitelné a útočník by jich bez dalších informací velmi složitě využil. Odlišnost nástrojů v detekci zranitelností, a jakým způsobem je vyhodnocují jsou celkově relativně rozdílné. Nástroj Nessus označuje za zranitelnosti i do jisté míry nepodstatné informace. Nástroj Vega označil za informativní zranitelnosti i takové, kterých by zkušenější útočník mohl zneužít. Nástroj OWASP ZAP za informativní neoznačil žádné a těm, kterým by Vega přiřadil prioritu informativní, tak je OWASP vyhodnotil s nízkou prioritou. Z toho plyne, že je potřeba důkladně kontrolovat všechny informace poskytnuté skenerem zranitelností. I takové, které podle nástrojem přiřazené priority nevypadají závažně.

Vzhledem k faktu, že se jednalo o první bezpečnostní test mohu označit tento výsledek za úspěšný. Kromě nejzávažnější zranitelnosti (nezabezpečená mezipaměť serveru Redis) se jednalo o relativně drobné bezpečnostní nedostatky. Ty vznikaly zejména testovacím certifikátem a nebo chybějícími hlavičkami v záhlaví HTTP odpovědí od serveru. Nutné je mít také na paměti, že aplikace NDS bude v produkci využívat dvoufaktorovou autentizaci (USB token a heslo). To znamená, že potencionální útočník z vnější strany by nejprve musel vlastnit USB token. USB Token by musel buď ukrást a nebo by se muselo jednat o útok z vnitřní infrastruktury zákazníka (útočníkem by byl zaměstnanec), což je nepravděpodobné.

# 5 ZÁVĚR

V první kapitole jsou obecně popsány projektové metodiky vývoje software, co je inkrementální a iterativní vývoj, a jaké jsou rozdíly mezi tradičními a agilními metodikami. V závěru první části je rozvedena metodika *Rational Unified Process* (RUP).

Druhá kapitola je věnována způsobům testování. Je zde rozebráno, podle čeho se určuje správná strategie testování, a jaké existuje obecné rozdělení testů. Podrobněji je popsán model FURPS+, který představuje východisko pro řízení kvality v metodice RUP a zahrnuje pět hlavních dimenzí kvality a jednu vedlejší.

V poslední kapitole teoretické části jsou popsány akceptační funkční testy, v jaké fázi testování se používají, a jaké jsou kritéria jejich splnění. Následně jsou uvedeny testy, které budou aplikovány k otestování systému *Nemo Link Dispatch System* (NDS). Každý test je následně obecně popsán k čemu slouží, a jak se používá.

Praktickou část tvoří nejprve úvod do problematiky energetického projektu *Nemo Link* a systému NDS. Popsáno je především osm jednotlivých modulů NDS, které jsou nezbytné k zajištění požadované funkčnosti celého systému.

Další část tvoří popis testovacího prostředí. Dále jsou uvedeny nástroje, které jsou v této práci využity k vykonání vybraných testů. Poté je navržena vlastní metodologie pro testy na základě životního cyklu testování. Struktura metodologie je složena ze 4 částí a tuto strukturu poté dodržují i jednotlivé testy. Na základě této metodologie byly navrženy, implementovány, realizovány a vyhodnoceny celkem 4 testy.

Prvním testem je GUI a autorizační test. GUI test byl označen za neúspěšný, jelikož zachytil více chyb uživatelského grafického rozhraní. U autorizačního testu nenastaly žádné problémy a úspěšně ověřil práva na základě dokumentací.

Druhým testem je výkonnostní test. Test ověřil 3 definované požadavky na základě dokumentace. Požadavek ke změně módu systému do 1. sekundy však splněn nebyl, a tudíž byl test označen za neúspěšný.

Třetím testem je zátěžový, stability a stress test. Zátěžový test byl označen za neúspěšný vzhledem k velmi vysokým odezvám ke konci testu. Stability test byl označen za úspěšný, jelikož po celou dobu zátěže nebyly identifikovány žádné vyšší odezvy ani nedostatky. Stress test byl označen za úspěšný, jelikož se povedlo vyhledat bod nasycení. Bod nasycení byl stanoven na počet 155 uživatelů. Při tomto počtu by systém měl zůstat stabilní a spolehlivý.

Čtvrtým testem je bezpečnostní test. Ten byl vykonán prostřednictvím 3. skenovacích nástrojů zranitelností. I přes skutečnost, že nástroje detekovaly dohromady 43 typů zranitelností, tak se kromě jedné nejedná o kritické bezpečnostní nedostatky. Vzhledem k první kontrole bezpečnosti aplikace NDS a poměru závažnosti jednotlivých typů zranitelností byl test označen za úspěšný.

Práce může poskytnout čtenářům přehledný a poměrně rozsáhlý návod pro vykonávání systémových, integračních nebo akceptačních testů na základě požadavků zákazníka. Z jednotlivých částí, ze kterých se každý test skládá v praktické části mohou čtenáři porozumět tomu, na co si dát při testování pozor. To znamená provedení podrobné analýzy dokumentací systému, kvalitní návrh a implementace testů, vhodně zvolené nástroje, rozumět tomu co výsledky testů znázorňují a provést report výsledků.

# LITERATURA


[1] KOVÁŘ, V. *Znalosti při vývoji software* 3. 10. 2007. [cit. 14. 10. 2017]. Dostupné z: [Interní materiál společnosti UNICORN.]

[2] *Metodika vývoje softwaru* [online]. 30. 5. 2017. [cit. 14. 10. 2017]. Dostupné z: <https://cs.wikipedia.org/wiki/Metodika_v%C3%BDvoje_softwaru>.

[3] KOVÁŘ, V. *Znalosti při vývoji software* 3. 10. 2007. [cit. 14. 10. 2017]. Dostupné z: [Interní materiál společnosti UNICORN.]

[4] PEJCHAL, J. *Agilní a tradiční metodiky v projektovém řízení* Masarykova Univerzita Brno, 2015. 69 s.

[5] BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů.* 1. vyd. Praha: Grada, 2005. 163 s. ISBN 80-247-1075-7.

[6] FARRELL, A. *Selecting a Software Development Methodology based on Organizational Characteristics* [online]. 11. 10. 2007. [cit. 14. 10. 2017]. Dostupné z: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.614.9644&rep=rep1&type=pdf>.

[7] CHAROUS, P. *Metodiky projektového riadenia a ich implementácia v praxi* Bankovní institut vysoká škola Praha, 2014. 117 s.

[8] MANCIN, E. *The IBM Rational Unified Process for System z.* 1. vyd., 2007. 270 s. ISBN 073848900X.

[9] MANCIN, E. *Rational Unified Process* 14. 2. 2012. [cit. 14. 10. 2017]. Dostupné z: [Interní projektová dokumentace RUP.]

[10] RANDOVÁ, L. *Metodika RUP a testování* Vysoká škola ekonomická v Praze, 207. 44 s.

[11] JULINEK, P. *Použití RUP pro malé SW projekty* Masarykova Univerzita Brno, 2008. 81 s.

[12] SZOTKOWSKI, J. *Automatické testování webových aplikací společnosti Unicorn* Masarykova univerzita Brno, 2016. 84 s.

[13] ČERMÁK, M. *White box test* [online]. 30. 10. 2010. [cit. 12. 11. 2017]. Dostupné z: <http://www.cleverandsmart.cz/white-box-test/>.

[14] ČERMÁK, M. *Black box test* [online]. 30. 10. 2010. [cit. 12. 11. 2017]. Dostupné z: <http://www.cleverandsmart.cz/black-box-test/>.

[15] ČERMÁK, M. *Grey box test* [online]. 30. 10. 2010. [cit. 12. 11. 2017]. Dostupné z: <http://www.cleverandsmart.cz/grey-box-test/>.

[16] BOROVCOVÁ, A. *Testování webových aplikací* Univerzita Karlova v Praze, 2008. 140 s.

[17] HLAVA, T. *Statické a dynamické testy* [online]. 12. 8. 2011. [cit. 12. 11. 2017]. Dostupné z: <http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/staticke-a-dynamicke-testy/>.

[18] SCHWAGER, M. *Automatizace funkčních testů* Masarykova Univerzita Brno, 2010. 51 s.

[19] BLAŽOVÁ, R. *Manuální a automatizované testy* 12. 3. 2016. [cit. 12. 11. 2017]. Dostupné z: [Interní materiál společnosti UNICORN.]

[20] HLAVA, T. *Automatizované testování* [online]. 11. 8. 2011. [cit. 12. 11. 2017]. Dostupné z: <http://testovanisoftwaru.cz/automatizovane-testovani/>.

[21] Eeles, P. *FURPS* [online]. 27. 5. 2017. [cit. 12. 11. 2017]. Dostupné z: <https://en.wikipedia.org/wiki/FURPS>.

[22] MICHÁLEK, L. *Použití metod testování softwaru v praxi* Vysoká škola ekonomická v Praze, 2006. 53 s.

[23] HLAVA, T. *Fáze a úrovně provádění testů* [online]. 21. 8. 2011. [cit. 18. 11. 2017]. Dostupné z: <http://testovanisoftwaru.cz/tag/factory-acceptance-test/#fat>.

[24] HLAVA, T. *Funkční a nefunkční testy* [online]. 20. 8. 2011. [cit. 18. 11. 2017]. Dostupné z: <http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/funkcni-a-nefunkcni-testy/>.

[25] FRONĚK, J. *TestStrategy* 10. 11. 2017. [cit. 18. 11. 2017]. Dostupné z: [Interní materiál společnosti UNICORN.]

[26] COATES, G. *Testing, Fixing and Costs* [online]. 10. 3. 2009. [cit. 18. 11. 2017]. Dostupné z: <https://www.chromosphere.co.uk/2009/03/10/testing-fixing-and-costs/>.

[27] ROUSE, M. *GUI testing (graphical user interface testing)* [online]. 2. 1. 2015. [cit. 20. 11. 2017]. Dostupné z: <http://whatis.techtarget.com/definition/GUI-testing-graphical-user-interface-testing>.

[28] JOOMLA *Testování GUI - malé nakousnutí velkého tématu* [online]. 15. 7. 2016. [cit. 20. 11. 2017]. Dostupné z: <http://test.swtestovani.cz/index.php?option=com_content&view=article&id=68:testovani-gui-male-nakousnuti-velkeho-tematu&catid=3:zaklady&Itemid=11>.

[29] BELLUCCI, D *OWASP TESTING GUIDE* [online]. 15. 9. 2008. [cit. 20. 11. 2017]. Dostupné z: <https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf>.

[30] *Stability Testing - What Why How?* [online]. 13. 6. 2017. [cit. 20. 11. 2017]. Dostupné z: <http://www.softwaretestingclass.com/stability-testing-what-why-how/>.

[31] ROUSE, M *Performance testing* [online]. 16. 8. 2014. [cit. 20. 11. 2017]. Dostupné z: <http://searchsoftwarequality.techtarget.com/definition/performance-testing>.

[32] *What is STRESS Testing in Software Testing: Tools, Need and Types* [online]. 18. 10. 2016. [cit. 20. 11. 2017]. Dostupné z: <https://www.guru99.com/stress-testing-tutorial.html>.

[33] *Interface Testing Tutorial: Types, Strategy & Example* [online]. 25. 9. 2015. [cit. 20. 11. 2017]. Dostupné z: <https://www.guru99.com/interface-testing.html>.

[34] *OWASP Testing Guide v4* [online]. 3. 9. 2015. [cit. 20. 2. 2018]. Dostupné z: <https://www.owasp.org/index.php/OWASP_Testing_Project>.

[35] *Nemo Link* [online]. [cit. 28. 11. 2017]. Dostupné z: <http://www.nemo-link.com>.

[36] *Representational state transfer* [online]. 13. 4. 2018. [cit. 28. 3. 2018]. Dostupné z: <https://en.wikipedia.org/wiki/Representational_state_transfer>.

[37] *Stopařův průvodce REST API* [online]. 2013. [cit. 28. 3. 2018]. Dostupné z: <https://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api>.

[38] *Apache Jmeter* [online]. 1998. [cit. 28. 11. 2017]. Dostupné z: <http://jmeter.apache.org/>.

[39] *Web Driver Sampler* [online]. [cit. 28. 11. 2017]. Dostupné z: <https://jmeter-plugins.org/wiki/WebDriverSampler/>.

[40] DOSTÁL, A. *Laboratorní úloha seznamující studenty se síťovými útoky* Vysoké učení technické v Brně, 2016. 80 s.

[41] *Oracle VM VirtualBox* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://www.virtualbox.org/>.

[42] *Kali Linux* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://docs.kali.org/introduction/what-is-kali-linux>.

[43] *Vega Vulnerability scanner* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://subgraph.com/vega/>.

[44] *OWASP Zed Attack Proxy* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>.

[45] *Nessus Vulnerability scanner* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://docs.tenable.com/nessus/>.

[46] *Software Testing Life Cycle STLC* [online]. [cit. 20. 3. 2018]. Dostupné z: <http://www.softwaretestingclass.com/software-testing-life-cycle-stlc/>.

[47] *STLC - Software Testing Life Cycle* [online]. [cit. 20. 3. 2018]. Dostupné z: <https://www.guru99.com/software-testing-life-cycle.html>.

[48] JÁCHYM, J. *Zátěžové testování webových aplikací* Vysoká škola ekonomická v Praze, 2012. 77 s.

[49] FARRELL, A. *Selecting a Software Development Methodology based on Organizational Characteristics* [online]. 11. 10. 2007. [cit. 14. 10. 2017]. Dostupné z: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.614.9644&rep=rep1&type=pdf>.

[50] IBM staff *Rational Unified Process Best Practices for Software Development Teams* [online]. 11. 1. 2005. [cit. 14. 10. 2017]. Dostupné z: <https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

| | |
|---|---|
| RUP | Rational Unified Process |
| NDS | Nemo Link Dispatch System |
| GUI | Graphical User Interface |
| QA | Quality Assurance |
| FAT | Factory Acceptance Tests |
| IS | Information System |
| ICT | Information and Communication Technology |
| OML | Open Methodology License |
| UP | Unified Process |
| EUP | Enterprise Unified Process |
| MSF | Microsoft Solutions Framework |
| XP | Extreme Programming |
| ASD | Adaptive Software Development |
| LD | Lean Development |
| DSDM | Dynamic System Development Method |
| SIT | Site Integration Tests |
| OAT | Operational Acceptance Tests |
| OWASP | The Open Web Application Security Project |
| SQL | Structured Query Language |
| HVDC | High Voltage Direct Current |
| RNP | Regional Nomination Platform |
| RCG | Remote Control Gateway |
| NFS | Network File System |
| REST | Representational State Transfer |

| | |
|---|---|
| HTTP | Hypertext Transfer Protocol |
| URI | Uniform Resource Identifier |
| JSON | JavaScript Object Notation |
| CSS | Cascading Style Sheets |
| LDAP | Lightweight Directory Access Protocol |
| SMTP | Simple Mail Transfer Protocol |
| TCP | Transmission Control Protocol |
| IDE | Integrated Development Enviroment |
| ARM | Advanced RISC Machine |
| OS | Operational System |
| XML | Extensible Markup Language |
| HTML | HyperText Markup Language |
| HTTPS | Hypertext Transfer Protocol Secure |
| URL | Uniform Resource Locator |
| SSL | Secure Sockets Layer |
| TLS | Transport Layer Security |
| SSH | Secure Shell |
| VPN | Virtual Private Network |
| USB | Universal Serial Bus |

# SEZNAM PŘÍLOH

# A OBSAH CD

- **Elektronická verze práce**
- **Nástroj Apache Jmeter s nainstalovanými pluginy**
- **GUI/Autorizační test** - Skript nástroje Apache Jmeter, snímky obrazovek, zaznamenané výsledky nástrojem v souborech.
- **Výkonnostní test** - Skript nástroje Apache Jmeter, zaznamenané výsledky nástrojem v souborech, obrázky tabulek vytvořených naslouchačem Aggregate Report.
- **Zátěžový, stability a stress test** - Skripty nástroje Apache Jmeter, zaznamenané výsledky nástrojem v souborech, grafy a tabulky vytvořené příslušnými naslouchači.
- **Bezpečnostní test** - Vygenerované reporty nástrojem OWASP ZAP a Nessus Vulnerability Scanner, obrázky typů zranitelností detekované nástrojem Vega Vulnerability Scanner.

# B INSTALACE PLUGINŮ DO NÁSTROJE APACHE JMETER

Vytvořené skripty nástroje Apache Jmeter obsahují rozšiřující pluginy, které základní verze nástroje neobsahuje a je nutné je před otevřením skriptů nainstalovat.

Postup ke správné instalaci:

1. Stáhnout nástroj Apache Jmeter z odkazu <`http://mirror.hosting90.cz/apache//jmeter/binaries/apache-jmeter-4.0.zip`>.
2. Stáhnout Plugin Manager z odkazu <`https://jmeter-plugins.org/get/`>.
3. Umístit Plugin Manager do adresáře *Apache Jmeter/lib/ext*.
4. Spustit nástroj Jmeter *bin/ApacheJmeter.jar*.
5. Kliknout na *Options -> Plugin Manager -> Available Plugins*.
6. Zvolit pluginy **3 Basics Graphs**, **5 Additional Graphs**, **Custom Threads Group**, **JSON Plugins**, **PerfMon** a **Selenium/WebDriver Support**.
7. Kliknout na *Apply Changes and Restart Jmeter*.

Nyní je vše připraveno a vytvořené skripty v příloze práce mohou být otevřeny.