



# 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

# WEBOVÁ PLATFORMA PRO PODPORU PENETRAČNÍHO TESTOVÁNÍ

WEB PLATFORM TO SUPPORT PENETRATION TESTING

## BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

### AUTOR PRÁCE

AUTHOR

Willi Lazarov

### VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Zdeněk Martinásek, Ph.D.

BRNO 2022



# Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

**Student:** Willi Lazarov

**ID:** 221556

**Ročník:** 3

**Akademický rok:** 2021/22

## NÁZEV TÉMATU:

### Webová platforma pro podporu penetračního testování

#### POKYNY PRO VYPRACOVÁNÍ:

Hlavním cílem práce je návrh a implementace webové platformy pro podporu bezpečnostního testování poskytující testerovi základní funkcionality prostřednictvím webové aplikace. Webová platforma se bude skládat z backendu (API, DB a datové úložiště) a frontendu (Single-page aplikace). Základní funkcionalita aplikace bude testerovi umožňovat zakládání nových projektů, informací o testovaném prostředí a pomocí kontrolních seznamů následně procházet bezpečnostním testem. Tester bude mít k dispozici možnost zaznamenávat jednotlivé nálezy, na jejichž základě aplikace vygeneruje závěrečnou zprávu. Dílčím cílem je zaměření se při vývoji na modularitu výsledné aplikace a použití některého z moderních JavaScript frameworků. Požadavkem je možnost dalšího rozšiřování aplikace pomocí modulů bez nutnosti zásahu do jejího jádra. Součástí platformy bude také API umožňující napojení dalších externích nástrojů pro automatizované testování s možností automatického zaznamenávání vrácených výsledků. Student v rámci práce dále nastuduje libovolnou zranitelnost a vytvoří pro její otestování skript, který bude kompatibilní s implementovaným API rozhraním. Součástí výsledku bude také rozšiřující modul pro hodnocení závažnosti zranitelností podle různých metodologií, např. CVSS.

#### DOPORUČENÁ LITERATURA:

- [1] JADHAV, Madhuri A.; SAWANT, Balkrishna R.; DESHMUKH, Anushree. Single page application using angularjs. International Journal of Computer Science and Information Technologies, 2015, 6.3: 2876-2879.
- [2] ALISHEROV, Farkhod; SATTAROVA, Feruza. Methodology for penetration testing. In: International Journal of Grid and Distributed Computing. 2009.

**Termín zadání:** 7.2.2022

**Termín odevzdání:** 31.5.2022

**Vedoucí práce:** Ing. Zdeněk Martinásek, Ph.D.

**doc. Ing. Jan Hajný, Ph.D.**  
předseda rady studijního programu

#### UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

## **ABSTRAKT**

Bakalářská práce se zabývá návrhem, vývojem a implementací webové platformy pro podporu penetračního testování. Práce se v teoretické části věnuje popisu penetračního testování a způsobu hodnocení závažnosti zranitelností. Dále jsou popsány jednotlivé technologie, které byly použity při vývoji výsledného řešení. V praktické části následuje popis postupného řešení dílčích požadavků webové platformy. Jednotlivé kapitoly shrnují problematiku, návrh a implementaci vlastního řešení. Nejprve je popsán návrh vysoce škálovatelného modelu, kterým je řešena hlavní problematika zadání práce. Následuje návrh platformy, její zasazení do navrženého modelu a vývoj modulární webové aplikace. Dále je popsán vlastní vývoj aplikační části, její propojení s relační databází, nástroji pro automatizované penetrační testování a generátorem reportu. Závěrem je popsáno testování výsledného řešení v produkčním prostředí a srovnání relevantních nástrojů pro správu a realizaci penetračního testování. Výsledkem této práce je webová platforma, jejímž hlavním účelem je zvýšit efekt penetračního testování do takové míry, že časová náročnost, složitost a práce nutná k úspěšnému dokončení celého testu bude značně nižší než při použití současných relevantních dostupných nástrojů.

## **KLÍČOVÁ SLOVA**

Penterep, penetrační testování, odhalování zranitelností, webová aplikace, modularita

## **ABSTRACT**

The bachelor thesis deals with the design, development, and implementation of a web platform to support penetration testing. The theoretical part of the thesis is devoted to the description of penetration testing and vulnerability severity assessment. Next, the technologies used in the development of the final solution are described. The practical part describes the gradual solution of partial requirements of the web platform. The individual chapters summarize the problem, design, and implementation of the solution. The practical part starts with the design of a highly scalable model that addresses the main problem of the assignment of this thesis. Next, the design of the platform, its embedding in the proposed model, and the development of a modular web application. Furthermore, the actual development of the application part is described, specifically, its connection with the relational database, tools for automated penetration testing, and the report generator. In the next chapter, the testing of the platform in a production environment is described. The last chapter compares relevant tools for penetration testing. The result of the work is a web platform with the main purpose of increasing the effect of penetration testing to such an extent that the time, complexity, and work required to successfully complete the entire test will be considerably lower than using currently relevant available tools.

## **KEYWORDS**

Penterep, penetration testing, vulnerability assessment, web application, modularity

LAZAROV, Willi. *Webová platforma pro podporu penetračního testování*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telemunikací, 2021, 102 s. Bakalářská práce. Vedoucí práce: doc. Ing. Zdeněk Martinásek, Ph.D.

# Prohlášení autora o původnosti díla

<b>Jméno a příjmení autora:</b>	Willi Lazarov
<b>VUT ID autora:</b>	221556
<b>Typ práce:</b>	Bakalářská práce
<b>Akademický rok:</b>	2021/2022
<b>Téma závěrečné práce:</b>	Webová platforma pro podporu penetračního testování

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom 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\*

---

\*Autor podepisuje pouze v tištěné verzi.

## PODĚKOVÁNÍ

Rád bych poděkoval panu doc. Ing. Zdeňku Martináskovi, Ph.D. za odborné vedení mé bakalářské práce, konzultace, motivaci a veškerou neocenitelnou podporu během mého studia, realizace této práce a všech dalších výzkumných a vývojových aktivit. Dále bych velmi rád poděkoval odbornému konzultantovi panu Romanu Kümmelovi za rozsáhlou spolupráci, podnětné návrhy a poskytnuté profesní znalosti při řešení bakalářské práce. Také bych chtěl poděkovat mým kolegům z Ústavu telekomunikací FEKT VUT v Brně za jejich cenné rady, zájem a přínosné připomínky k této práci. V neposlední řadě patří poděkování mé rodině za trpělivost, pochopení a veškerou podporu.

T A  
Č R

Výzkum a vývoj popsaný v této práci byl spolufinancován se státní podporou Technologické agentury ČR v rámci Programu Zéta (č. projektu TJ04000456).

# Obsah

<b>Úvod</b>	<b>14</b>
<b>1 Teoretická část</b>	<b>15</b>
1.1 Penetrační testování . . . . .	15
1.1.1 Metodologie penetračního testování . . . . .	15
1.1.2 Typy penetračního testování . . . . .	17
1.2 Hodnocení závažnosti zranitelností . . . . .	18
1.2.1 Common Vulnerability Scoring System . . . . .	19
1.2.2 Výpočet závažnosti podle standardu CVSS . . . . .	20
1.3 Frontendové technologie . . . . .	23
1.3.1 JavaScript . . . . .	23
1.3.2 Vue.js . . . . .	24
1.3.3 TypeScript . . . . .	26
1.4 Backendové technologie . . . . .	26
1.4.1 PHP . . . . .	27
1.4.2 Laravel . . . . .	27
1.4.3 REST API . . . . .	28
1.4.4 MySQL . . . . .	29
<b>2 Praktická část</b>	<b>30</b>
2.1 Vlastní návrh webové platformy . . . . .	30
2.1.1 Řešená problematika . . . . .	30
2.1.2 Návrh vysoce škálovatelného modelu . . . . .	31
2.1.3 Zasazení webové platformy do navrženého modelu . . . . .	32
2.1.4 Zasazení webové platformy do produkčního prostředí . . . . .	33
2.2 Vlastní návrh a vývoj frontendové části . . . . .	34
2.2.1 Návrh modulární webové aplikace . . . . .	34
2.2.2 Vývoj modulární webové aplikace . . . . .	37
2.2.3 Návrh a implementace lokálního rozhraní . . . . .	40
2.3 Vlastní návrh a vývoj backendové části . . . . .	43
2.3.1 Návrh a vývoj REST API . . . . .	44
2.3.2 Autentizace klienta . . . . .	48
2.3.3 Model uzlu . . . . .	49
2.3.4 Strom uzlů . . . . .	51
2.3.5 Generátor reportu . . . . .	56
2.3.6 Automatizované testování . . . . .	59
2.4 Nástroj pro vyhledávání známých zranitelností . . . . .	63

<b>3</b>	<b>Testování v produkčním prostředí</b>	<b>64</b>
3.1	Nasazení platformy . . . . .	64
3.2	Testování platformy . . . . .	65
<b>4</b>	<b>Porovnání současných řešení</b>	<b>66</b>
	<b>Závěr</b>	<b>69</b>
	<b>Literatura</b>	<b>72</b>
	<b>Seznam symbolů a zkratek</b>	<b>76</b>
	<b>Seznam příloh</b>	<b>79</b>
<b>A</b>	<b>Požadavky webové platformy</b>	<b>80</b>
<b>B</b>	<b>Metriky a hodnoty CVSS</b>	<b>81</b>
<b>C</b>	<b>Velikost výsledného řešení</b>	<b>85</b>
<b>D</b>	<b>Ukázka uživatelského rozhraní</b>	<b>86</b>
<b>E</b>	<b>Ukázka nástroje pro vyhledávání známých zranitelností</b>	<b>99</b>
<b>F</b>	<b>Ukázka webové prezentace platformy</b>	<b>101</b>
<b>G</b>	<b>Obsah elektronické přílohy</b>	<b>102</b>

# Seznam obrázků

1.1	Proces penetračního testování . . . . .	15
1.2	Metriky pro hodnocení závažnosti zranitelnosti . . . . .	19
1.3	Komunikace Vue.js komponenty zasazená do MVVM . . . . .	24
1.4	Stromová struktura komponent . . . . .	25
1.5	Architektura Model-View-Controller . . . . .	28
1.6	Příklad HTTP požadavku pro API volání . . . . .	29
2.1	Návrh vysoce škálovatelného modelu . . . . .	31
2.2	Produkční prostředí platformy . . . . .	33
2.3	Stromová struktura komponent . . . . .	36
2.4	Rozmístění komponent do základní struktury . . . . .	38
2.5	Ukázka uživatelského rozhraní . . . . .	39
2.6	Komponenta CVSS kalkulátoru . . . . .	40
2.7	Návrh lokálního rozhraní SPA API . . . . .	42
2.8	Implementace lokálního rozhraní SPA API . . . . .	43
2.9	Ukázka cesty pro metodu GET . . . . .	45
2.10	Ukázka cesty s parametrem pro metodu GET . . . . .	45
2.11	Implementace a komunikace REST API . . . . .	46
2.12	Dokumentace REST API . . . . .	47
2.13	Průběh úspěšné autentizace . . . . .	48
2.14	Vztah mezi testovatelnými uzly . . . . .	49
2.15	Vztah mezi netestovatelnými uzly . . . . .	50
2.16	Obecný model vlastnosti uzlu . . . . .	50
2.17	Obecný model propojených vlastností uzlu . . . . .	51
2.18	Strom uzlů . . . . .	51
2.19	Operace přidání a odstranění uzlu . . . . .	54
2.20	Ukázka vygenerovaného reportu . . . . .	58
2.21	Proces aktivace automatizovaného testování . . . . .	59
2.22	Proces automatizovaného testování . . . . .	61
2.23	Vyhledávání známých zranitelností . . . . .	63
3.1	Komunikace mezi Docker hostem a Docker registrem . . . . .	64
3.2	Sekvence procesů na platformě GitHub . . . . .	65
D.1	Hlavní nástěnka s projekty . . . . .	86
D.2	Založení nového projektu . . . . .	86
D.3	Pozvání uživatelů do projektu . . . . .	87
D.4	Výběr kontrolních seznamů . . . . .	88
D.5	Uzly projektu . . . . .	88
D.6	Manuální přidání nového uzlu . . . . .	89

D.7	Testování uzlů . . . . .	90
D.8	Uzel testu . . . . .	91
D.9	Uzel zranitelnosti . . . . .	92
D.10	Strom uzlů . . . . .	93
D.11	Filtrování stromu uzlů . . . . .	93
D.12	Poznámky projektu . . . . .	94
D.13	Nahrávání příloh . . . . .	95
D.14	Uzel logu . . . . .	96
D.15	Uzel šablony reportu . . . . .	97
D.16	Uzel reportu . . . . .	98
E.1	Rozhraní nástroje ptvulnsearcher . . . . .	99
E.2	Výsledky vyhledávání nástroje ptvulnsearcher . . . . .	100
F.1	Webová prezentace platformy . . . . .	101

# **Seznam tabulek**

1.1	Závažnosti zranitelnosti . . . . .	22
2.1	Seznam komponent SPA . . . . .	35
2.2	Seznam balíčků produkční a vývojové verze . . . . .	37
4.1	Porovnání relevantních nástrojů s výsledným řešením . . . . .	67
B.1	Metriky základního skóre . . . . .	82
B.2	Metriky dočasného skóre . . . . .	83
B.3	Metriky skóre prostředí . . . . .	84
C.1	Velikost implementace praktické části . . . . .	85

# **Seznam výpisů**

2.1	Výpis stromu uzlů ve formátu JSON . . . . .	55
2.2	Příklad zdrojového kódu šablony reportu . . . . .	57
2.3	Výsledek automatizovaného testu . . . . .	62

# Úvod

Stále více organizací a uživatelů využívá online služby. S tímto nárůstem stoupá i počet serverů, na kterých jsou tyto služby provozovány a počet technologií, které služby využívají. Servery a aplikace jsou z důvodu jejich snadné dostupnosti častým cílem kybernetických útoků.<sup>1</sup> Pokud je takový útok úspěšný, může způsobit škody velkého rozsahu od přerušení služeb až po krádež nebo ztrátu citlivých dat. Útočeno je také na další prvky spadající pod oblast informačních a komunikačních technologií, čímž se množství rizik a rozsah dopadu ještě dále navyšuje. Nárůst těchto hrozob podnítil vznik nástrojů pro penetrační testování, které jako jedna z prevencí proti kybernetickým útokům, odhaluje bezpečnostní rizika testovaných cílů. Penetrační testování je náročný proces a na trhu chybí komplexní řešení, které by umožnilo jeho plnou správu a poloautomatizaci prostřednictvím uživatelského rozhraní spolu s týmovou spoluprací, správou projektů a sofistikovanou vizualizací.

Tato práce se zabývá vývojem webové platformy, jakožto podpůrným řešením uvedené problematiky s odlišujícím se přístupem od dostupných nástrojů. Hlavním cílem práce je vlastní návrh, vývoj a nasazení rozsáhlého prostředí pro efektivní realizaci a správu komplexního penetračního testování, které by testerům umožnilo spolupracovat jak mezi sebou, tak i s manažery, vývojáři, správci systému a dalšími relevantními osobami testovaného subjektu. Při realizaci práce bude kladen důraz na vysokou modularitu, škálovatelnost a optimalizaci všech dílcích řešení. Cíleno bude také na uživatelskou přívětivost a použitelnost klientské části webové platformy.

Bakalářská práce se v teoretické části věnuje penetračnímu testování a způsobu hodnocení závažnosti zranitelností. Dále jsou popsány frontendové a backendové technologie, které byly použity při vývoji webové platformy. V praktické části je následně popsáno postupné řešení dílcích požadavků webové platformy. Jednotlivé sekce shrnují problematiku, návrh a implementaci vlastního řešení. Praktická část je rozdělena na návrh celé platformy, následně popisuje vývoj vysoce modulární SPA (Single-page application) a jejích vybraných dílcích částí, které dohromady tvoří frontend. Dále je popsán návrh a vývoj backendu, zejména API (Application Programming Interface), jednotlivé procesy a spojení s relační databází. Praktickou část zakončuje kapitola pojednávající o integraci nástrojů pro automatizované penetrační testování a vývoji vlastního nástroje pro vyhledávání známých zranitelností. Následně je popsáno nasazení platformy do produkčního prostředí a procesy testování výsledného řešení. V poslední kapitole jsou s výsledky této práce srovnány vybrané relevantní nástroje pro správu a realizaci penetračního testování.

---

<sup>1</sup>Tento fakt potvrzuje například rozsáhlá analýza společnosti SiteLock, která ukazuje, že do roku 2019 mohlo být napadeno škodlivým kódem celkem 17,6 milionů webových stránek. Oproti předchozímu roku jde o 59% nárůst s průměrem 62 útoku za den [1].

# 1 Teoretická část

## 1.1 Penetrační testování

Penetrační testování je rozsáhlá metoda hodnocení úrovně bezpečnosti informačních a komunikačních systémů. Předmětem testování může být téměř cokoliv v této oblasti, nejčastěji však síťová infrastruktura, služby a protokoly aplikační vrstvy, webové aplikace nebo koncoví uživatelé. Hlavním cílem penetračních testů je preventivně odhalit, ověřit a popsat zranitelnosti testovaného cíle, aby mohla být sjednána náprava v podobě odstranění nalezených zranitelností. Tento způsob bezpečnostního testování spadá pod oblast etického hackingu a provádí se zcela legitimně s předchozím souhlasem testovaného subjektu, nejčastěji externí stranou na základě smlouvy o dílo.<sup>1</sup> Rozsah a průběh penetračního testování se může lišit v závislosti na použité metodologii, testovaném prostředí a způsobu jeho provedení.

### 1.1.1 Metodologie penetračního testování

Jakým způsobem postupovat při penetračním testování určují různé metodologie. Mezi ty nejčastěji používané metodologie patří OWASP (Open Web Application Security Project), OSSTMM (Open Source Security Testing Methodology Manual), ISSAF (Information Systems Security Assessment Framework) a PTES (Penetration Testing Execution Standard). Jednotlivé fáze penetračního testování jsou znázorněny na obrázku 1.1 a popsány v následujících podkapitolách [2].



Obr. 1.1: Proces penetračního testování

#### Plánování testování

Před samotným testem je důležité určit rozsah penetračního testování a jakých cílů má být dosaženo. Testování může být prováděno jednotlivci nebo týmem testerů, přičemž rozhodující jsou zejména přidělené finanční prostředky a časové možnosti obou stran. Penetrační testování by tedy mělo probíhat pouze v rozsahu, na kterém se zadavatel a zpracovatel smluvně dohodnou [3].

<sup>1</sup>Penetrační testování může být prováděno také interním týmem testovaného subjektu nebo CERT a CSIRT týmy, jako jsou v České republice vládní GovCERT.CZ a národní CSIRT.CZ.

## **Průzkum prostředí**

Cílem této fáze je získat co nejvíce informací o testovaném prostředí, kterými jsou například síťová infrastruktura, zařízení, služby nebo použité technologie a jejich parametry (např. operační systém, software a jeho verze). Pro urychlení a usnadnění sběru informací se používají automatizované nástroje, skenery, doplňky webových prohlížečů nebo také netechnické metody, jako je sociální inženýrství [2]. Všechny získané informace následně vstupují do další fáze odhalování zranitelností.

## **Odhalování zranitelností**

Na základě informací získaných v předchozí fázi testování zahrnuje proces odhalování zranitelností dvě hlavní části: hledání zranitelností v databázích známých zranitelností a dílčí testování prostředí na přítomnost konkrétní zranitelnosti (např. chybná konfigurace). Stejně jako v předchozí fázi, tak i zde je možné využít nástroje, které hledání zranitelností provádí automatizovaně, nebo je postupováno manuálně podle konkrétní metodologie. Výsledkem této fáze je určení zranitelných míst, která jsou následně v rámci testování využita pro další postup [2].

## **Zneužití zranitelnosti**

Pouhé odhalení zranitelnosti jako výsledek penetračního testování nestačí. Nalezená rizika je nutné ověřit zneužitím zranitelnosti, tzv. exploitací. Pro urychlení této fáze lze použít již existující exploity<sup>2</sup>, pomocí kterých je možné cíl snadno kompromitovat. Po zneužití zranitelnosti může dojít také k odhalení předem nepřístupné části testovaného prostředí (např. administrace webové aplikace) a je třeba předchozí dvě fáze zopakovat [2]. Postupným odhalováním a zneužíváním zranitelností je praktická část penetračního testování dokončena.

## **Závěrečná zpráva**

Veškeré zranitelnosti a další nedostatky zjištěné v předchozích fázích testování musí být předány zadavateli v podobě závěrečné zprávy (reportu). Hlavním významem této zprávy je popsat odhalená rizika, stanovit jejich závažnost a vydat doporučení, na základě kterého by měl zadavatel správně provést patřičná opatření (např. úprava zdrojového kódu, aktualizace systému apod.). Závěrečná zpráva je citlivý dokument, který by měl být doručen zadavateli přes zašifrované spojení. Stejně tak by mělo být od počátku přistupováno k zabezpečení veškeré komunikace mezi zpracovatelem a zadavatelem penetračního testování [4].

---

<sup>2</sup>Exploit je program, skript nebo postup, který dokáže zneužít konkrétní zranitelnost. Existují také veřejně přístupné databáze exploitů (např. Exploit Database a Rapid7).

## 1.1.2 Typy penetračního testování

Penetrační testování lze dále posuzovat:

- podle strany provedení,
- podle úrovně znalosti,
- podle způsobu provedení.

### Testování podle strany provedení

Penetrační testování se podle pozice umístění testera dělí na:

- **Externí testování**, které je prováděno bez oprávnění a přístupu do vnitřní části testovaného prostředí. Podle fází uvedených v kapitole 1.1.1 se tester snaží cíl kompromitovat a získat nebo pozměnit chráněná aktiva. Hlavním cílem externích testů je tedy prověřit míru zabezpečení před hrozby z vnější strany testovaného cíle (nejčastěji internetu) [5].
- **Interní testování**, kdy má tester již od začátku přístup do vnitřní části testovaného prostředí. Tester v tomto případě představuje legitimního uživatele (např. zaměstnance testovaného subjektu), nebo útočníka, který se do vnitřní strany dostal neoprávněně. Při interním testování se hodnotí zejména bezpečnostní mechanismy, systémové politiky a řízení přístupu [5, 6].

### Testování podle úrovně znalosti

Podle úrovně znalostí jsou testy rozděleny na:

- **Black-box testy** prováděné bez počáteční znalosti vnitřní části testovaného cíle. Testování spočívá v postupném odhalování jednotlivých součástí a hledání jejich zranitelností. Příkladem takové zranitelnosti může být zastaralá verze operačního systému, chybná konfigurace webového serveru nebo neošetřené uživatelské vstupy testované aplikace.
- **White-box testy**, které spočívají v kompletní znalosti testovaného prostředí. Tester má k dispozici plný přístup ke všem součástem systému, zdrojovým kódům, databázemi apod. U white-box testů se zpravidla provádí hlubší analýza. V případě testování aplikací dochází navíc ke statické analýze, během které se hledají bezpečnostní chyby ve zdrojovém kódu [6].
- **Gray-box testy** složené kombinací white-box a black-box testů. Za účelem snížení časové náročnosti má tester k dispozici informace usnadňující nalezení a ověření zranitelností v testovaném prostředí (např. zdrojový kód, struktura databáze nebo přístupové údaje). Na rozdíl od white-box testů se neprovádí kompletní statická analýza zdrojového kódu. Informace bývají využívány pouze k lepšímu pochopení funkčnosti testovaného prostředí na rozdíl od black-box testů, během kterých musí tyto informace získat sám tester [3].

## **Testování podle způsobu provedení**

Penetrační testování lze provádět následujícími způsoby:

- **Manuálním testováním**, při kterém má penetrační tester plnou kontrolu nad procesem testování a dosažené výsledky jsou tak zcela závislé na jeho postupu a schopnostech. Výhodou manuálního testování je zejména přesnost a důslednost v odhalování zranitelností. Naopak nevýhodou tohoto způsobu je časová náročnost a nutnost disponovat rozsáhlými znalostmi [7].
- **Automatickým testováním** s použitím automatizovaných nástrojů, které vyhledávají zranitelnosti zadaného cíle. Nástroje používají sadu testovacích řetězců, databáze známých zranitelností apod. Testerovi potom předkládají výsledky s možnými nálezy. Oproti manuálnímu testování je tento způsob mnohem rychlejší a jeho provedení vyžaduje pouze znalost práce s danými nástroji [3]. Nevýhodou může být naopak nižší úspěšnost v hledání zranitelností, falešně pozitivní nálezy nebo nemožnost otestovat některé zranitelnosti (např. pokud se zranitelnost projevuje na jiném místě, než se nachází vstup, nebo je potřeba logického uvažování).
- **Poloautomatickým testováním**, které kombinuje manuální a automatické testy, přičemž využívá výhody a zmírňuje nevýhody obou typů. Pro snížení časové náročnosti se fáze průzkumu a identifikace prostředí provádí z velké části automatizovanými nástroji, zatímco potenciálně zranitelná místa a možnost jejich zneužití se testují manuálně.

## **1.2 Hodnocení závažnosti zranitelností**

Mezi hlavní fáze penetračního testování patří odhalení a zneužití zranitelností. Pokud penetrační tester přítomnost zranitelnosti úspěšně ověří, tak by měl správně určit i její závažnost, která potom vstupuje do závěrečného reportu. Pro hodnocení závažnosti zranitelností existují různé metodiky. Jednou z těch nejpoužívanějších je otevřený standard CVSS (Common Vulnerability Scoring System) verze 3.1.<sup>3</sup> Kromě CVSS existují i další známé metody, jako je například OWASP RRM (Risk Rating Methodology) pro hodnocení závažnosti zranitelností webových aplikací [8, 9].

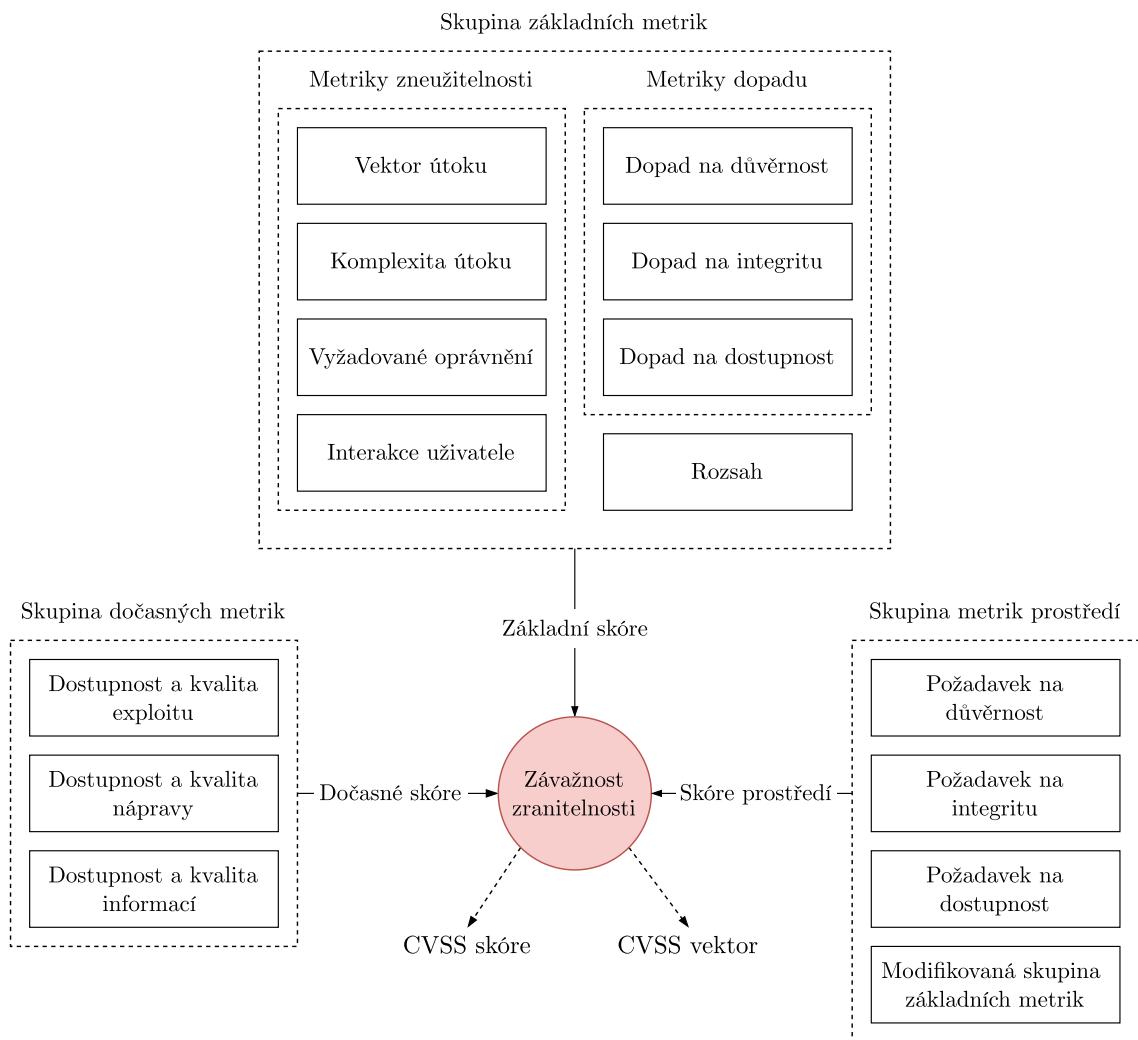
Jedním z dílčích cílů této práce je vývoj kalkulátoru, který by prostřednictvím webové platformy umožnil penetračním testerům určit závažnost zranitelností bez nutnosti použití externích nástrojů. Za tímto účelem byl k implementaci komponenty použit standard CVSS, který je díky jeho komplexnosti přesný, a z toho důvodu i vhodný pro podporu manuálního penetračního testování.

---

<sup>3</sup>Standard CVSS je spravován mezinárodní organizací FIRST (Forum of Incident Response and Security Teams) a jedná se o jeden z nejpoužívanějších standardů pro hodnocení zranitelností.

### 1.2.1 Common Vulnerability Scoring System

Pomocí standardu CVSS se závažnost zranitelnosti hodnotí podle skóre vypočteného z rovnic skupiny základních metrik, skupiny dočasných metrik a skupiny metrik prostředí. Pro stanovení dopadu se využívá CIA triáda<sup>4</sup>, která se skládá z Confidentiality (důvěrnosti), Integrity (integrity) a Availability (dostupnosti). Výsledkem hodnocení je CVSS skóre (stupnice hodnocení 0–10) a CVSS vektor, který vyjadřuje přesnou kalkulaci závažnosti. Jakým způsobem jednotlivé metriky vstupují do výsledného hodnocení je znázorněno na obrázku 1.2 [8]. Rovnice pro výpočet skóre jednotlivých skupin a formát výsledné závažnosti uvádí následující kapitola 1.2.2.



Obr. 1.2: Metriky pro hodnocení závažnosti zranitelnosti

<sup>4</sup>CIA triáda je model pro stanovení základních principů kybernetické bezpečnosti.

## 1.2.2 Výpočet závažnosti podle standardu CVSS

Standard CVSS hodnotí celkovou závažnost zranitelnosti podle metrik základního skóre  $BS$  (Base Score), dočasného skóre  $TS$  (Temporal Score) a skóre prostředí  $ES$  (Environmental Score). Pro jejich výpočet se používá funkce `Roundup`, která vrací číslo s přesností na jedno desetinné místo směrem k vyššímu číslu. Dále se při výpočtu využívá funkce `Min`, která vrací menší ze dvou vstupních argumentů této funkce. Rovnice uvedené níže vycházejí ze standardu CVSS verze 3.1 a možné hodnoty jednotlivých metrik jsou uvedeny spolu s jejich popisem v příloze B [10].

### Základní skóre

Základní skóre  $BS$  se počítá přímo pro závažnost konkrétní zranitelnosti bez dalších vnějších vlivů, které by závažnost zranitelnosti ovlivnily. Základní skóre je neměnné a na rozdíl od dočasného skóre a skóre prostředí je nezávisle na čase a prostředí.

Rovnice pro výpočet základního skóre počítá s hodnotou dopadu  $I$  (Impact), hodnotou dílčího skóre dopadu  $ISS$  (Impact Sub-Score) a hodnotou zneužitelnosti  $E$  (Exploitability). Pro hodnotu základního skóre platí, že pokud  $I \leq 0$ , tak  $BS = 0$ . Způsob výpočtu ovlivňuje rozsah, který určuje dopad zranitelnosti na jiný prvek než ten, který byl určen jako zranitelný. Pokud se rozsah nezměnil a zároveň platí, že  $I > 0$ , tak se základní skóre vypočte rovnicí:

$$BS = \text{Roundup}(\text{Min}[(I + E), 10]), \quad (1.1)$$

nebo v opačném případě při změně rozsahu:

$$BS = \text{Roundup}(\text{Min}[1, 08 \cdot (I + E), 10]). \quad (1.2)$$

Pro výpočet dopadu je nejprve nutné vypočítat dílčího skóre dopadu  $ISS$ , které je definováno následující rovnicí:

$$ISS = 1 - [(1 - I_C) \cdot (1 - I_I) \cdot (1 - I_A)], \quad (1.3)$$

kde  $I_C$  je parametrem důvěrnosti,  $I_I$  integrity a  $I_A$  dostupnosti. Výpočet dopadu se stejně jako u základního skóre liší podle změny rozsahu. Pokud se rozsah nezměnil, tak se dopad  $I$  vypočte rovnicí:

$$I = 6,42 \cdot ISS, \quad (1.4)$$

nebo v případě změny rozsahu:

$$I = 7,52 \cdot (ISS - 0,029) - 3,25 \cdot (ISS - 0,02)^{15}. \quad (1.5)$$

Výpočet zneužitelnosti  $E$  se provede následující rovnicí:

$$E = 8,22 \cdot AV \cdot AC \cdot PR \cdot UI, \quad (1.6)$$

kde  $AV$  (Attack Vector) je vektor útoku,  $AC$  (Attack Complexity) komplexita útoku (obtížnost zneužití zranitelnosti),  $PR$  (Privileges Required) vyžadované oprávnění a  $UI$  (User Interaction) interakce uživatele.

### **Dočasné skóre**

Dočasné skóre  $TS$  doplňuje základní skóre zranitelnosti o vnější vlivy, kterými jsou dostupnost a kvalita exploitu, nápravy a informace o zranitelnosti. Výpočet  $TS$  je definován bez dalších podmínek rovnicí:

$$TS = \text{Roundup}(BS \cdot ECM \cdot RL \cdot RC), \quad (1.7)$$

kde  $BS$  je základní skóre,  $ECM$  (Exploit Code Maturity) určuje pravděpodobnost zneužitelnosti zranitelnosti,  $RL$  (Remediation Level) dostupnost nápravy a hodnota  $RC$  (Report Confidence) představuje dostupnost informací o zranitelnosti.

### **Skóre prostředí**

Skóre prostředí je závislé na výskytu dané zranitelnosti. Výpočet skóre prostředí ovlivňuje modifikovaný dopad  $MI$  (Modified Impact), přičemž pro  $MI \leq 0$  platí  $ES = 0$ . Pro případ, kdy se parametr rozsahu nezměnil a  $MI > 0$ , tak se rovnice pro výpočet skóre prostředí definuje jako:

$$ES = \text{Roundup}(\text{Roundup}[\text{Min}([MI + ME], 10)] \cdot ECM \cdot RL \cdot RC), \quad (1.8)$$

kde  $MI$  značí modifikovaný dopad,  $ME$  (Modified Exploitability) modifikovanou zneužitelnost,  $ECM$  pravděpodobnost zneužitelnosti,  $RL$  dostupnost nápravy a  $RC$  dostupnost informací o zranitelnosti. Pokud se v opačném případě rozsah změnil, tak je rovnice upravena na:

$$ES = \text{Roundup}(\text{Roundup}[\text{Min}(1,08 \cdot [MI + ME], 10)] \cdot ECM \cdot RL \cdot RC). \quad (1.9)$$

Pro výpočet  $MI$  je nejprve nutné vypočítat dílčí skóre modifikovaného dopadu  $MISS$  (Modified Impact Sub-Score), které je definováno rovnicí:

$$MISS = \text{Min}([1 - (1 - CR \cdot M_C) \cdot (1 - IR \cdot M_I) \cdot (1 - AR \cdot M_A)], 0,915), \quad (1.10)$$

kde  $CR$  (Confidentiality Requirement) je požadavek důvěrnosti,  $M_C$  modifikovaný dopad na důvěrnost,  $IR$  (Integrity Requirement) požadavek na integritu,  $M_I$  modifikovaný dopad na integritu,  $AR$  (Availability Requirement) požadavek na dostupnost a  $M_A$  modifikovaný dopad na dostupnost.

Výpočet modifikované dopadu  $MI$  je ovlivněn změnou rozsahu. Pokud se rozsah nezměnil, tak se pro výpočet použije následující rovnice:

$$MI = 6,42 \cdot MISS, \quad (1.11)$$

pokud se naopak rozsah změnil, tak se modifikovaný dopad vypočte rovnicí:

$$MI = 7,52 \cdot (MISS - 0,029) - 3,25 \cdot (MISS \cdot 0,9731 - 0,02)^{13}. \quad (1.12)$$

Výpočet modifikované zneužitelnosti  $ME$  se vypočte hodnotami modifikované skupiny základních metrik následující rovnicí:

$$ME = 8,22 \cdot M_{AV} \cdot M_{AC} \cdot M_{PR} \cdot M_{UI}, \quad (1.13)$$

kde modifikovanými metrikami jsou vektor útoku  $M_{AV}$ , komplexita útoku  $M_{AC}$ , vyžadované oprávnění  $M_{PR}$  a interakce uživatele  $M_{UI}$ .

### **Výsledné hodnocení závažnosti zranitelnosti**

Podle hodnoty kalkulace jsou dle standardu CVSS stanovena číselná rozmezí, kterým je přiřazena závažnost (viz následující tabulka 1.1).

Tab. 1.1: Závažnosti zranitelnosti

Závažnost	CVSS skóre
Žádná	0,0
Nízká	0,1–3,9
Střední	4,0–6,9
Vysoká	7,0–8,9
Kritická	9,0–10,0

Dalším výsledkem závažnosti je CVSS vektor. Jedná se o textový řetězec, který přesněji popisuje závažnost dané zranitelnosti pomocí hodnot jednotlivých metrik. Obecně lze formát CVSS vektoru zapsat jako:

$$\text{CVSS3Vektor} = \text{CVSS:3.1}/M_1:H_1/M_2:H_2/\dots, M_i:H_i,$$

kde CVSS:3.1 značí verzi standardu,  $M_1, M_2, \dots, M_i$  jsou metriky a  $H_1, H_2, \dots, H_i$  představují jejich hodnoty. Příklad CVSS vektoru může být následující řetězec:

$$\text{CVSS3Vektor} = \text{CVSS:3.1}/\text{AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H},$$

kde vektor útoku AV je síťový N (Network), komplexita útoku AC je nízká L (Low), vyžadované oprávnění PR je označeno jako žádné N (None), rozsah S (Scope) byl změněn C (Changed) a jednotlivé dopady na důvěrnost C (Confidentiality), integritu I (Integrity) a dostupnost A (Availability) jsou vysoké H (High).

## 1.3 Frontendové technologie

Frontendovou část výsledného řešení tvoří vysoce modulární jednostránková webová aplikace SPA, se kterou uživatel interaguje. Princip SPA je postaven na dynamickém přepisování svého obsahu a server tak žádá pouze o nová data. Tím se výrazně liší od běžných webových aplikací, kde je pro změnu obsahu nutné žádat server o kompletní novou stránku. Díky tomu dokáže SPA rychleji reagovat na interakci uživatele, snižuje zátěž serveru a pomocí cachování dat dokáže fungovat i offline. Nevýhodou je naopak vzhledem k existenci pouze jedné stránky s aplikací horší optimalizace pro vyhledávače, čímž se indexování obsahu velmi stěžuje.<sup>5</sup> SPA je v této práci naprogramována v jazyce TypeScript za použití JavaScript frameworku Vue.js. Jedná se o jedny z nejpoužívanějších technologií v oblasti vývoje webových aplikací a pro jejich bližší seznámení jim bude věnován v této části prostor.

### 1.3.1 JavaScript

JavaScript je vysokoúrovňový interpretovaný skriptovací programovací jazyk. Jedná se zároveň o jeden z nejpopulárnějších programovacích jazyků na světě [11, 12, 13] a jeho uplatnění lze nalézt převážně při vývoji webových stránek a aplikací. Využívá se také na straně serveru jako backendová technologie. JavaScript může být použit jako objektově orientovaný nebo procedurální programovací jazyk. Jelikož se jedná o interpretovaný jazyk, tak se jeho kód nekompliluje, ale překládá a spouští až při běhu programu. To je prováděno nejčastěji na straně klienta ve webovém prohlížeči nebo na straně serveru [14]. JavaScript bývá často označován společně s verzí ES6+ (ECMAScript 6 a vyšší). ECMAScript 6 je skriptovací jazyk standardizovaný mezinárodní organizací Ecma International pod označením ECMA-262. Standard bývá každoročně rozšiřován o novou edici, díky čemuž jsou implementace jazyka ECMAScript pravidelně vylepšovány [15].

Na straně klienta se JavaScript používá primárně pro pokročilé dynamické funkce webových aplikací, jako jsou například reakce na konkrétní události (např. kliknutí na tlačítko), změna obsahu elementů webové stránky, komunikace s API, správa úložiště webového prohlížeče nebo cookies. Naopak JavaScript na straně serveru může být použit k vývoji aplikací fungujících v reálném čase (např. streamovácí nebo chatovací aplikace), webových služeb nebo také k práci s databázemi. Vývoj v JavaScriptu je nejčastěji realizován za použití některé z jeho nadstaveb v podobě frameworku nebo knihovny. Mezi ty nejpoužívanější patří za rok 2021 knihovna React, framework Vue.js a framework Angular [16].

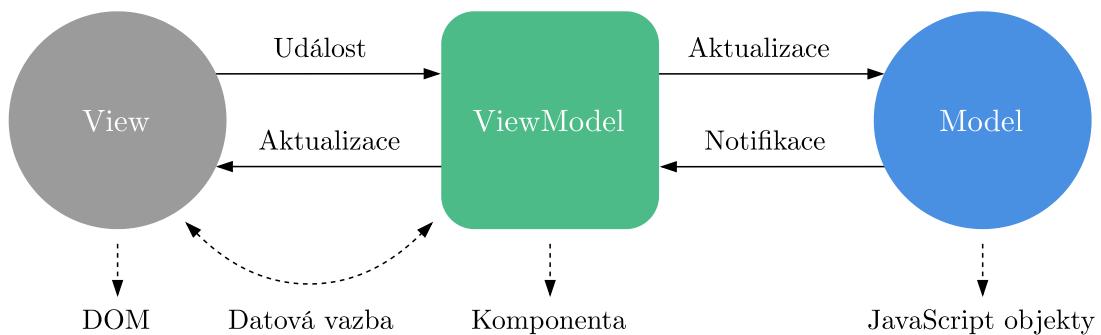
---

<sup>5</sup>Z toho důvodu se SPA používají primárně k vývoji interních aplikací, které není třeba indexovat a jejich obsah je dostupný zpravidla až po přihlášení uživatele.

### 1.3.2 Vue.js

Vue.js je jedním z předních progresivních frameworků pro jazyk JavaScript. Používá se jak k vývoji jednoduchých uživatelských rozhraní, tak i komplexních webových aplikací. Jeho předností oproti ostatním frameworkům je jednotnost ve vytváření komponent, kde se HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) a JavaScript nachází dohromady v jednom souboru [17]. Celý framework je od verze 3 napsán v jazyce TypeScript, díky čemuž poskytuje nativní podporu tohoto jazyka. Framework Vue.js byl pro vývoj SPA v této práci vybrán na základě vstupní analýzy, která byla součástí návrhu webové platformy. Samotný návrh je blíže popsán v praktické části, ale ve stručnosti byl Vue.js upřednostněn oproti knihovně React a frameworku Angular hlavně z důvodu cílení na vysokou modularitu SPA a dalších požadavků vázajících se k frontendové části uvedených v příloze A.

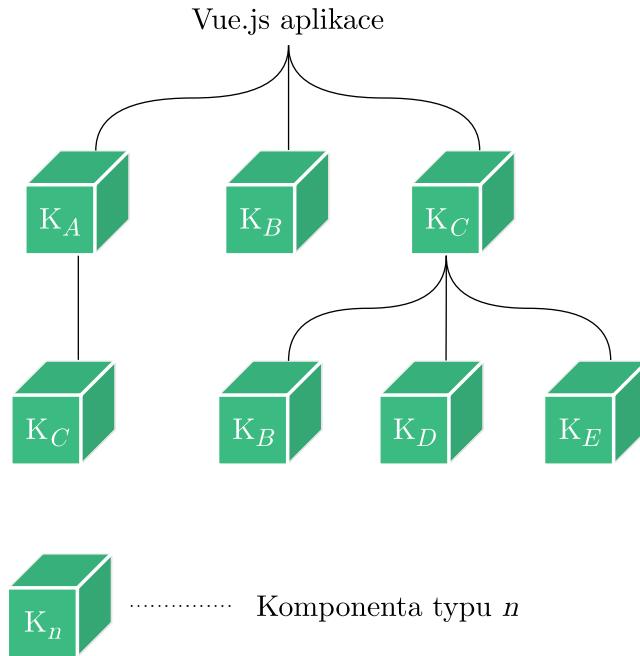
Fungování Vue.js je inspirováno architekturou MVVM (Model-View-ViewModel). MVVM odděluje část Model (logika aplikace) od View (uživatelské rozhraní) pomocí instance ViewModel (komponenta), která zajišťuje synchronizaci těchto dvou částí. Model ve Vue.js představuje JavaScript objekty jednotlivých komponent. Objekty jsou reaktivní, díky čemuž může komponenta na základě konkrétní události nebo změny stavu aplikace okamžitě aktualizovat část pohledu bez nutnosti svého zničení a opakovaného vytvoření. View je potom pohled představující uživatelské rozhraní, se kterým uživatel interaguje. Vue.js k vytváření pohledů používá šablony založené na DOM (Document Object Model) elementech, které prostřednictvím datové vazby (data binding) mapuje na Model [17]. Princip vnitřní komunikace Vue.js komponent zasazený do architektury MVVM je znázorněn na obrázku 1.3.



Obr. 1.3: Komunikace Vue.js komponenty zasazená do MVVM

Všechny instance aplikace jsou reprezentovány komponentami, přičemž každá z nich obsahuje svoji vlastní šablonu, logiku a vnitřní stav. Tato vlastnost dělá z Vue.js progresivní framework, protože lze jednotlivé části vyvíjet nezávisle na sobě. Komponenty je také možné opakovat použít, aniž by došlo k narušení jejich stavu.

Díky tomu je možné snadno zamezit vzniku duplicit a aplikace tak může být snadno škálovatelná. Komponenty v sobě mohou zároveň obsahovat další komponenty, čímž mezi nimi vzniká vztah rodiče a potomka [17]. Rodičovská komponenta registruje své potomky a tvoří se tak stromová struktura celé vnitřní části Vue.js aplikace, jako je znázorněno na obrázku 1.4.



Obr. 1.4: Stromová struktura komponent

Poslední a jednou z nejdůležitějších částí frameworku je životní cyklus každé z komponent, v rámci kterého je komponenta vytvořena, aktualizována a zničena. Životní cyklus se skládá celkem z 8 hlavních částí [17]:

1. **beforeCreate** – komponenta je inicializována,
2. **created** – komponenta je vytvořena,
3. **beforeMount** – komponenta bude vyrenderována,
4. **mounted** – komponenta byla vyrenderována,
5. **beforeUpdate** – data byla změněna,
6. **updated** – data se projeví změnou DOM elementů,
7. **beforeDestroyed** – komponenta bude zničena,
8. **destroyed** – komponenta je zničena.

Životní cyklus umožňuje rozdělit logiku aplikace do jednotlivých fází, které budou volány podle aktuálního stavu komponenty. Díky tomu lze efektivně reagovat na jednotlivé události a pomocí reaktivnosti ovlivňovat konkrétní DOM elementy nebo celé komponenty. Tento mechanismus je obsažen v každé z komponent, které jsou svým vnitřním stavem na sobě nezávislé.

### 1.3.3 TypeScript

TypeScript je striktně typovaný programovací jazyk vytvořený společností Microsoft. TypeScript vznikl za účelem zavedení typové kontroly pro JavaScript, kterému jako interpretovanému jazyku tato vlastnost chyběla. Díky tomu lze odhalit chyby ve zdrojovém kódu ještě před jeho spuštěním a také celý kód zpřehlednit pomocí typového systému [18]. Veškerý kód se následně konvertuje do validního JavaScript kódu. Mezi hlavní vlastnosti TypeScriptu patří [19]:

- **anotace typů** – přiřazení datových typů proměnným, funkcím atd.,
- **třídy** – plná podpora tříd podle specifikace ECMAScript,
- **rozhraní** – podpora rozhraní pro kontrolu vlastností a typů,
- **mixiny** – dílčí třídy pro kombinaci a opakované použití metod,
- **moduly** – organizace kódu do interních a externích modulů,
- **jmenné prostory** – prostředky pro organizaci a distribuci kódu,
- **iterátory a generátory** – pokročilé vykonávání a pozastavení funkcí,
- **výčtové typy** – množiny neměnných hodnot (konstant),
- **výchozí hodnoty parametrů** – podpora podle specifikace ECMAScript.

Pro modulární přístup k vývoji je použití TypeScriptu velmi výhodné. Kód lze rozšířit o typovou kontrolu, vytvářet přístupnější bloky kódu a přesně definovat funkcím, jaké parametry mají přijímat či vracet. Pokud některá část nemá určené datové typy, tak je použito základní dynamické typování jazyka JavaScript. Výhoda TypeScriptu se projeví také při pozdější úpravě kódu, kde je přesněji definováno schéma jeho jednotlivých částí. Tím je ve výsledku celý kód lépe strukturovaný, čitelnější a není tak nutné psát podrobnější dokumentaci.

## 1.4 Backendové technologie

Backendovou část této práce spravuje Docker<sup>6</sup>, který z obrazů vytváří a spouští kontejner webové aplikace poskytující REST (Representational State Transfer) API, webového serveru Nginx sloužící jako reverzní proxy a kontejner MySQL databáze. Hlavní úlohou backendu je klientovi po úspěšné autentizaci předat SPA a následně autorizovat a zpracovat veškeré jeho požadavky. Komunikaci s SPA zajišťuje API, které pracuje s datovým úložištěm a komunikuje s relační databází MySQL. Webová aplikace byla naprogramována v jazyce PHP za použití frameworku Laravel.

---

<sup>6</sup>Docker je jedním z nejvíce rozšířených nástrojů pro kontejnerizaci. Na rozdíl od virtuálních strojů provádí Docker kontejnerovou virtualizaci na úrovni operačního systému a využívá tak jeho jádro, díky čemuž klade menší paměťové nároky. Mezi další výhody patří izolace, škálovatelnost a portabilita Docker kontejnerů [20].

### **1.4.1 PHP**

PHP (rekurzivně Hypertext Preprocessor) je skriptovací programovací jazyk určený převážně k vývoji webových aplikací. Nejčastěji se nachází na straně serveru, kde jsou skripty tohoto jazyka vykonávány interpretem. PHP je dynamicky typovaný jazyk a jeho kód se tudíž nekompliluje.<sup>7</sup> Základem jazyka je open source skriptovací engine Zend Engine. PHP je multiplatformní a jeho vykonání závisí pouze na interpretu a konfiguraci jazyka [21]. Stejně jako mnoho dalších jazyků používaných pro vývoj webových aplikací, tak samotná nativní verze jazyka PHP není příliš dostačující a je vhodné ho rozšířit použitím některého z jeho frameworků. Mezi ty dlouhodobě nejpoužívanější patří framework Laravel, Symfony a CodeIgniter [22].

### **1.4.2 Laravel**

Laravel je progresivní framework pro vývoj webových aplikací v jazyce PHP. Jako nadstavba programovacího jazyka přináší sadu vlastních nástrojů a další rozšíření, díky kterým lze aplikace vyvíjet rychleji, optimálněji a bezpečněji. Laravel funguje na základě architektury MVC (Model-View-Controller). Dalším významným prvkem je vestavěný konzolový nástroj pro automatizovanou práci s prostředím frameworku a kontrolou vývojového serveru. Laravel je také velmi škálovatelný a díky stabilní komunitě existuje velké množství rozšiřujících knihoven. Framework Laravel přináší zejména následující možnosti [23]:

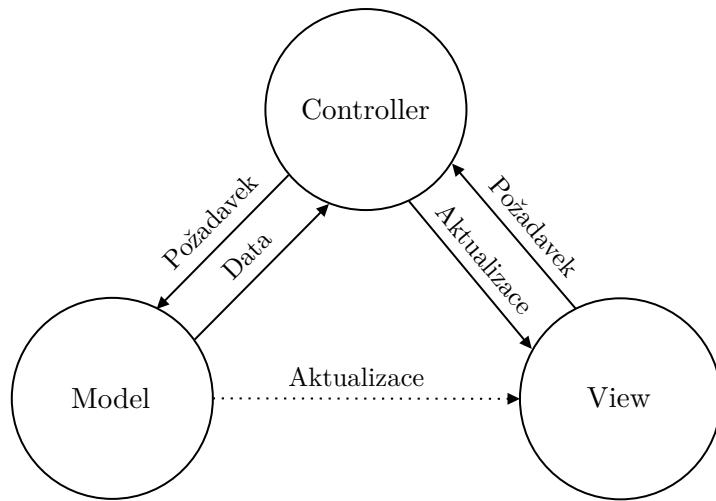
- šablonovací systém,
- cachovací systém,
- pomocné funkce (helpers),
- správu front a plánování,
- bezpečnostní mechanismy,
- rozhraní pro dotazování databáze,
- objektově relační mapování,
- migrační strukturu databází,
- unit testing (testování jednotek).

Jak již bylo zmíněno, tak základem frameworku je architektura MVC, podle které jsou aplikace v Laravelu vyvíjeny. MVC se dělí na tři hlavní části – Model (model), View (pohled) a Controller (řadič). Rozdělení MVC je znázorněno na obrázku 1.5. Model v architektuře představuje data a logiku aplikace. Na základě požadavků zpracovává data, komunikuje s databází a zajišťuje logické funkce, jako je například autentizace uživatele nebo validace přijatých dat [24].

---

<sup>7</sup>Výjimkou mohou být desktopové aplikace, pro které existuje komplikovaná verze jazyka.

View tvoří uživatelské rozhraní aplikace, ve kterém jsou data reprezentována. Prezentace a logika dat jsou zcela odděleny, takže View nikdy nezpracovává data. Prostřednictvím pohledu uživatel posílá požadavky na Controller, který je řídící jednotkou celé aplikace a obsluhuje veškeré požadavky, které dál předává Modelu ke zpracování. Následně vykreslí View s příslušnými daty a celý tento pohled předá zpět uživateli. Další možností je také aktualizovat View přímo Modelem bez další komunikace. Tento případ se může lišit podle použitého frameworku [25].



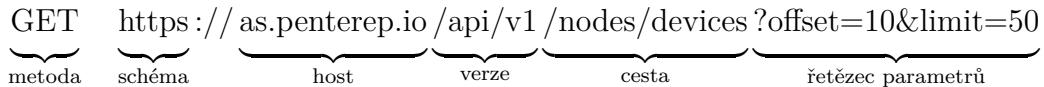
Obr. 1.5: Architektura Model-View-Controller

### 1.4.3 REST API

REST API je aplikační programové rozhraní, které vychází z architektury REST. Rozhraní slouží ke zpracování operací CRUD (Create, Read, Update, Delete) za použití aplikačního protokolu HTTP (Hypertext Transfer Protocol). Tím, že je HTTP bezstavový protokol, tak neudržuje trvalé spojení, a proto musí být v jeho hlavičce obsaženo vše potřebné k jeho vykonání. Pokud je vyžadované požadavek autorizovat, tak by měl obsahovat navíc autorizační token [26]. Veškeré požadavky posílané na REST API se označují jako API volání v podobě URL (Uniform Resource Locator), které se skládají z následujících částí:

- **metoda** – HTTP metoda,
- **schéma** – použitý protokol,
- **host** – doména nebo IP adresa,
- **verze** – verze REST API,
- **cesta** – cesta pro konkrétní API volání,
- **řetězec parametrů** – parametry a hodnoty.

Příklad formátu HTTP požadavku pro API volání je znázorněn na obrázku 1.6.



Obr. 1.6: Příklad HTTP požadavku pro API volání

#### 1.4.4 MySQL

Stěžejním prvkem výsledného řešení je komunikace s relační databází. MySQL je jeden z nejpoužívanějších<sup>8</sup> databázových systémů vlastněný a spravovaný společností Oracle Corporation. Jelikož se jedná o relační databázový systém, tak se struktura databáze dělí na tabulky, mezi kterými mohou existovat relace (vztahy). Jednotlivé tabulky pak obsahují atributy v podobě sloupců, které mají určité vlastnosti, jako je datový typ, délka, unikátnost apod. Relace jsou v MySQL databázi tvořeny cizími klíči, které představují nejčastěji primární klíče jiné tabulky. Vytvořením atributu cizího klíče vznikne v tabulce relace, díky které lze data lépe organizovat a provádět nad nimi efektivně určité operace. S relační databází MySQL a jejími daty se pracuje prostřednictvím následující skupiny jazyků [27]:

- **DDL (Data Definition Language)** – definice struktury a schéma databáze, tabulek, sloupců (atributů), jejich datových typů, vlastností a relací.
- **DML (Data Manipulation Language)** – manipulace s daty v tabulkách databáze (výběr, vytvoření, úprava a smazání dat).
- **DCL (Data Control Language)** – řízení přístupu (autorizace) k databázi a jejím jednotlivým částem včetně uložených dat.
- **TCL (Transaction Control Language)<sup>9</sup>** — databázové transakce, přesněji nedělitelné, konzistentní a izolované provedení DML operací.

Operace DDL, DML, DCL a TCL jsou prováděny pomocí dotazovacího jazyka SQL (Structured Query Language). Všechny relační databázové systémy jazyk SQL podporují a jeho syntaxi podle potřeby rozšiřují o vlastní příkazy. To může způsobit komplikace při migraci databáze do jiného systému, nicméně základní operace jazyka zůstávají zachovány. Standardy jazyka SQL podporuje také většina programovacích jazyků spolu s jejich frameworky a knihovnami, čímž se značně zvyšuje kompatibilita relačních databází napříč různými aplikacemi.

<sup>8</sup>Popularitu databázových systémů lze sledovat pomocí pravidelně aktualizovaného indexu DB-Engines Ranking. Žebříček je dostupný na adrese <https://db-engines.com/en/ranking>.

<sup>9</sup>Označován také jako TCS (Transaction Control Statements).

## 2 Praktická část

### 2.1 Vlastní návrh webové platformy

Hlavní požadavky webové platformy jsou uvedeny v příloze A. Cílem této práce je vyvinout řešení pro komplexní realizaci penetračního testování, které by zvýšilo celkové pokrytí testované oblasti s cílem podpořit manuální testování. Zároveň je potřeba zajistit, aby bylo možné platformu snadno rozšiřovat bez nutnosti zásahu do již dokončených částí. Podle požadavků kladených na webovou platformu jsou pro prvotní návrh řešení důležité následující body:

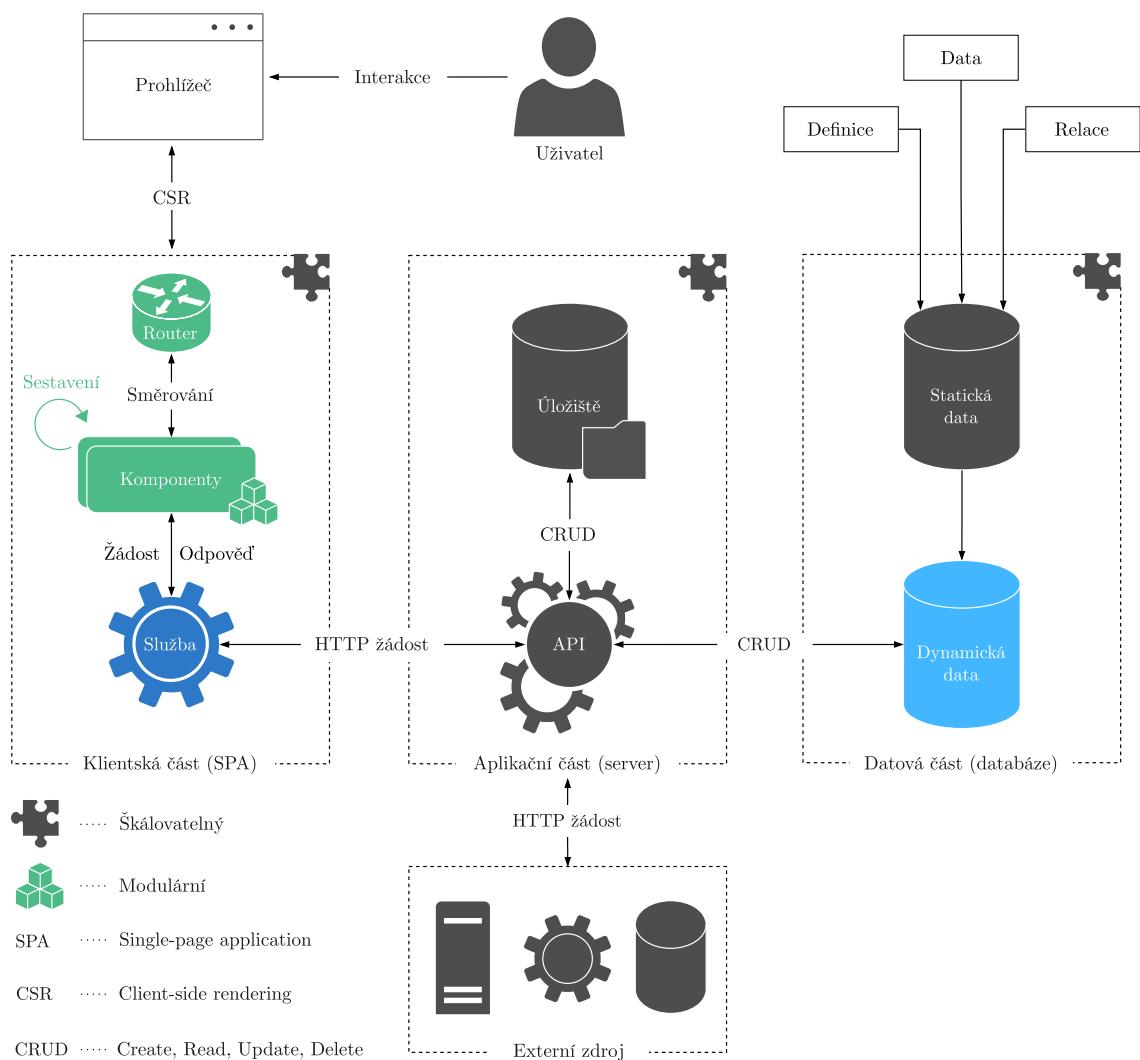
- **modularita** – opakované použití komponent a zamezení duplicit,
- **škálovatelnost** – rozšíření bez nutnosti zásahu do hlavní struktury,
- **minimum statického obsahu** – oddělení obsahu od zdrojového kódu,
- **vysoká optimalizace** – maximalizace výkonu všech dílčích částí,
- **verzování kódu** – ukládání změn s možností jejich navrácení,
- **údržba kódu** – snadná pozdější úprava zdrojového kódu,
- **testování a ladění** – testování vývojové a produkční verze.

#### 2.1.1 Řešená problematika

Postup penetračního testování se liší podle metodologií a testovaného prostředí. Webová platforma musí pro pokrytí této oblasti poskytnout strukturu pro maximální počet typů testů, přičemž schéma nelze jednoduše zobecnit [28]. Příkladem může být testování síťové infrastruktury v porovnání s testováním webové aplikace. V případě počítačové sítě bude tester pracovat zejména s typem, maskou a výchozí bránou sítě. Odhalovat při tom bude síťová a koncová zařízení, která se mohou lišit typem (např. router, server, firewall atd.) a podle toho i síťových služeb s rozdílnými parametry. Naopak při penetračním testování webové aplikace může tester odhalit použité programovací jazyky, frameworky a další webové stránky, které mohou mít odlišné uživatelské vstupy. Uvážíme-li každý možný cíl penetračního testování, tak by pro pokrytí celé oblasti bylo nutné vyvinout postupně sekci pro každý typ testu. Vývoj by v takovém rozsahu vyžadoval velké množství času, financí a lidských zdrojů. Pozdější úprava může být z důvodu takto rozsáhlého celku značně problematická. Další nevýhodou je náchylnost ke vzniku duplicitního a nepřehledného zdrojového kódu. Vzhledem k velkému množství dílčích částí se stěžuje i proces testování a optimalizace celého řešení. Uvedené nevýhody a nedostatky znesezdňují postupný vývoj takto komplexního nástroje, zejména s cílem splnit body uvedené v úvodu praktické části, a proto je problematika v této práci řešena jiným způsobem, který je podrobně popsán v následující kapitole 2.1.2.

## 2.1.2 Návrh vysoko škálovatelného modelu

Řešením uvedené problematiky z předchozí kapitoly je návrh vysoko škálovatelného modelu, podle kterého byla platforma vyvíjena a dále efektivně rozšiřována. Vlastní návrh modelu je zobrazen na obrázku 2.1 a následující text postupně vysvětuje jeho jednotlivé části. Model se skládá celkem ze tří hlavních částí – datové, aplikační a klientské. Schéma užití platformy vzniká v datové části, kde dochází k definování vstupních dat a vytváření relací – vztahu mezi daty. Výstupem tohoto procesu jsou statická data představující datovou strukturu celé platformy. Statická data následně určují, jakým způsobem budou reprezentována dynamická data, která do databáze vstupují až při používání platformy v produkčním prostředí a práce s nimi je tak zcela oddělena od první fáze tvorby hlavní struktury. Obě tyto části se nacházejí v databázi, se kterou komunikuje aplikační část navrženého modelu.



Obr. 2.1: Návrh vysoko škálovatelného modelu

Druhou částí modelu je aplikační strana, představující jeden nebo více serverů, které se starají o logiku navrženého modelu a prostřednictvím vlastního rozhraní komunikují s klientskou a datovou částí. Aplikační server dále zajišťuje autentizaci uživatelů a autorizaci všech požadavků. Jeho součástí může být také datové úložiště pro práci se soubory. Hlavním prvkem je API, které na základě HTTP požadavku vrací z databáze strukturovaná statická data pro klientskou část, nebo zpracovává příslušná dynamická data. Aplikační server je možné dodatečně rozšířit o další API určené ke komunikaci s externí stranou, kterou může být např. databáze, cloudové úložiště nebo jiná vzdálená služba. Aplikační část navrženého modelu lze rozdělit do více služeb s vlastním rozhraním, které mohou po vzoru architektury mikroslužeb fungovat na odlišných technologiích a to zcela nezávisle na sobě [29].

Třetí částí modelu je klientská strana s jednostránkovou aplikací SPA, se kterou uživatel interaguje a posílá jejím prostřednictvím požadavky na aplikační server. Jakmile se klient úspěšně autentizuje, tak mu aplikační server zašle SPA, která se následně načte do jeho webového prohlížeče. SPA funguje na principu dynamického vykreslování CSR (Client-side rendering) a veškeré změny tak provádí na straně klienta a server žádá podle potřeby o nová data. SPA je v navrženém modelu vysoko modulární a na základě požadavků klienta a příchozích statických dat se sama sestaví ze svých komponent, které následně zažádají o potřebná dynamická data, přičemž v sobě mohou zahrnovat další komponenty a celý tento proces tak lze opakovat.

Navržený model je použitelný pro širokou škálu případů užití.<sup>1</sup> Jeho hlavním cílem je definovat architekturu modulárního přístupu pro vývoj webové platformy při současném zachování nezávislosti použitých technologií. Důležité je dodržovat zásady pro zajištění vysoké modularity, škálovatelnosti, optimalizace, automatizace, údržby zdrojového kódu a možnosti každou část plně testovat.

### 2.1.3 Zasazení webové platformy do navrženého modelu

Navržený model plně vyhovuje zadání této práce. Počáteční fází je podle navrženého modelu definování schémat penetračního testování. Tento proces je velmi rychlý a je tak možné v krátkém čase vytvořit strukturu pro jednotlivé typy testů, prostředí a všechny související části. Veškerá dynamická data vytvořená nebo získaná během testování představují uzly, které jsou následně popsány a rozdeleny statickými daty. Uzly tvoří vstupní data a rozlišují se podle typu (např. webová aplikace, síťové zařízení, log, uživatel atd.). Všechna data se ukládají do MySQL databáze, ve které jsou od sebe oddělena statická a dynamická data. Pro zajištění požadavku vysoké optimalizace se datová a aplikační část nachází na stejném hostiteli.

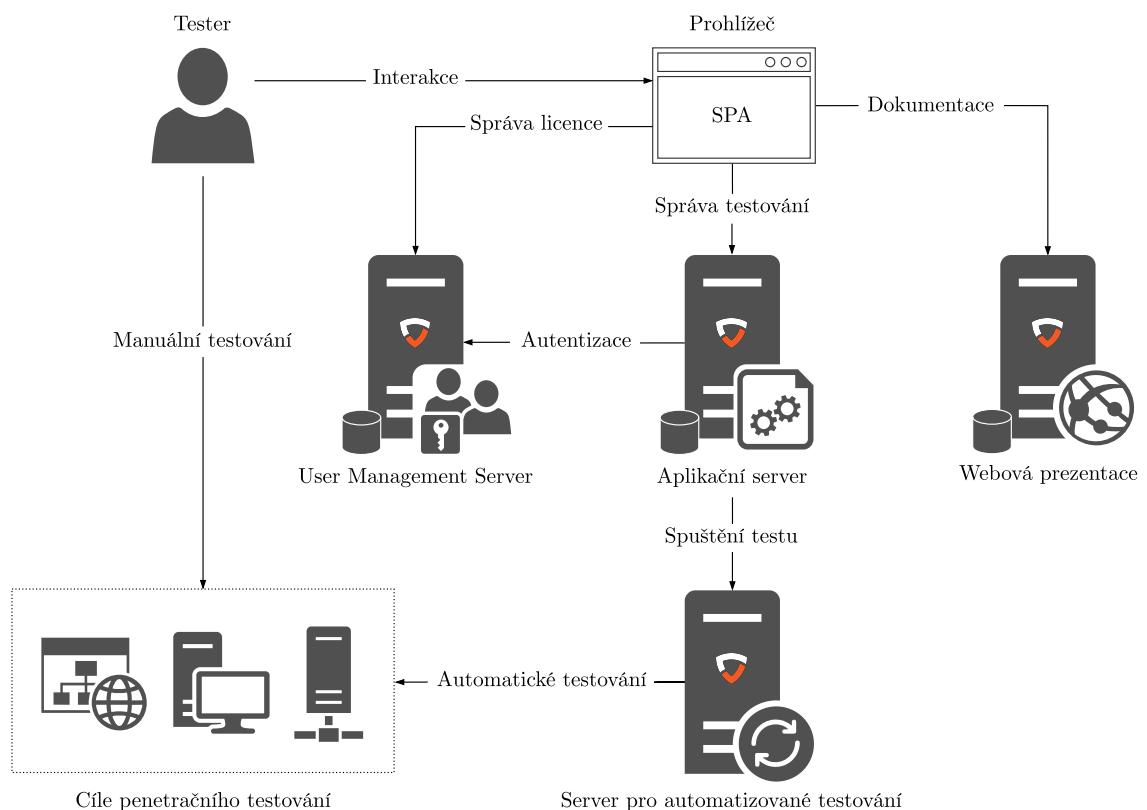
---

<sup>1</sup>Platformu lze použít například pro testování software nebo auditorskou činnost. Stačí pouze definovat strukturu a kontrolní seznamy v prvotní fázi před produkčním nasazením.

Aplikační server a relační databáze byly prostřednictvím Docker kontejnerizace nasazeny na VPS (Virtual Private Server) s operačním systémem Ubuntu. Jednotlivé požadavky obsluhují příslušná API, která komunikují s SPA, serverem pro automatizované penetrační testování, datovým úložištěm a autentizačním serverem. Celé aplikační rozhraní bylo vyvinuto PHP frameworkem Laravel. Na klientské straně je SPA technicky řešena jako modulární Vue.js aplikace naprogramována v jazyce TypeScript, pomocí kterého je zároveň zajištěna striktní typová kontrola zdrojového kódu. SPA obsluhuje požadavky klienta a předává je dále ke zpracování aplikačnímu serveru prostřednictvím svých služeb.

#### 2.1.4 Zasazení webové platformy do produkčního prostředí

Webová platforma komunikuje při produkčním nasazení s dalšími částmi, které jsou součástí projektu výzkumu a vývoje, jehož je tato práce součástí. Celé produkční prostředí je znázorněno na obrázku 2.2.



Obr. 2.2: Produkční prostředí platformy

Uživatel interaguje s platformou prostřednictvím webového prohlížeče, ve kterém se nachází SPA. Předtím, než je SPA uživateli poskytnuta aplikačním serverem (AS), tak se musí autentizovat vůči UMS (User Management Server), kam je přesměrován.

Jakmile se uživatel úspěšně autentizuje, tak je navrácen zpět na AS s přiděleným tokenem od UMS. AS ověří token vůči UMS a pokud je validní, tak je mu vráceno SPA a to je následně načteno do jeho webového prohlížeče. Od této chvíle už veškeré požadavky odesílá SPA na základě interakce uživatele a dalších událostí aplikace. Pro proces autentizace všech nasazených aplikačních serverů je určeno jedno UMS. AS může být hostováno v cloudu, na klientských stanicích, VPS, dedikovaných nebo vlastních serverech klientů. Uživatelé mohou také podle potřeby využít podrobnější návody, seznamy s popisem zranitelností, články a další informační stránky, které jsou veřejně dostupné z oficiální webové prezentace.<sup>2</sup>

Poslední součástí produkčního prostředí je server pro automatizované testování (SAT). Uživatelé provádí testování buď manuálně a výsledky nahrávají do webové platformy, nebo spustí automatizované testy, které provede SAT a výsledky uloží přes AS do databáze platformy. Cílem automatického testu může být téměř cokoliv, pokud SAT disponuje vhodným integrovaným nástrojem pro jeho testování. Webová platforma, s cílem podpořit penetrační testování v maximální možné míře, umožňuje testování realizovat nejen penetračními testery, ale i dalšími uživateli, jako jsou například vývojáři, kteří nalezené zranitelnosti odstraní a tester následně přítomnost zranitelnosti znova ověří. Klíčovou funkcí je generování reportu celého testování, které by při manuálním použití textového editoru bylo časově náročné.

## 2.2 Vlastní návrh a vývoj frontendové části

Frontendovou část této práce tvoří SPA, pro jejíž vývoj byl zvolen framework Vue.js a programovací jazyk TypeScript. Hlavní požadavky kladené na SPA jsou podle návrhu platformy vysoká modularita komponent a škálovatelnost celé aplikace. Tato kapitola blíže popisuje strukturu aplikace a návrh komponent, ze kterých se SPA dynamicky sestavuje. Dále je popsán vlastní vývoj dílčích částí SPA spolu s ukázkou uživatelského rozhraní. Kapitola v závěru popisuje návrh a implementaci lokálního rozhraní, které je zásadní pro komunikaci vnitřní části aplikace.

### 2.2.1 Návrh modulární webové aplikace

Pro snazší zajištění modularity jsou komponenty, služby a konfigurační soubory rozděleny do podadresářů s vlastní dokumentací, aby mohly být implementovány odděleně a podle potřeby opakovaně v aplikaci používány. Cílem tohoto rozdělení je koncipovat vývoj tak, aby byla maximalizována modularita dílčích částí SPA a nedocházelo k vytváření duplicitního zdrojového kódu.

---

<sup>2</sup>Oficiální webové stránky Penterep dostupné na adrese <https://www.penterep.com/>.

Komponenty pracují převážně s daty, která obdrží z vnější strany, kterou může být uživatel, aplikační server nebo jiný externí zdroj. SPA proto poskytuje služby pro API volání, správu webového úložiště LocalStorage a naslouchání WebSocket<sup>3</sup> notifikacím. Dále jsou pro komponenty definovány datové typy a rozhraní, které jsou určené k bližší specifikaci entit aplikace (např. funkce, parametry a návratový typ). Dalším mechanismem je směrování vnitřní části aplikace pomocí routera, kterým je zajištěna dynamická změna obsahu podle parametrů obsažených v URL. Důležitým prvkem jsou také neměnné konstanty a lokalizace pro načtení jazykové mutace podle nastavení uživatele. Poslední součástí SPA jsou prvky ovlivňující vizuální podobu aplikace – vektorová grafika a definice kaskádových stylů.

## Komponenty

Na základě zadání této práce byly vyvinuty komponenty uvedené v tabulce 2.1. Unikátní komponenty mohou být v aplikaci použity pouze jednou, zatímco ostatní opakovaně bez omezení. Dále se mohou lišit v tom, zda uchovávají pouze svůj vnitřní stav nebo ho navíc sdílí v rámci celé aplikace a ostatní komponenty k němu mohou v případě potřeby přistupovat.

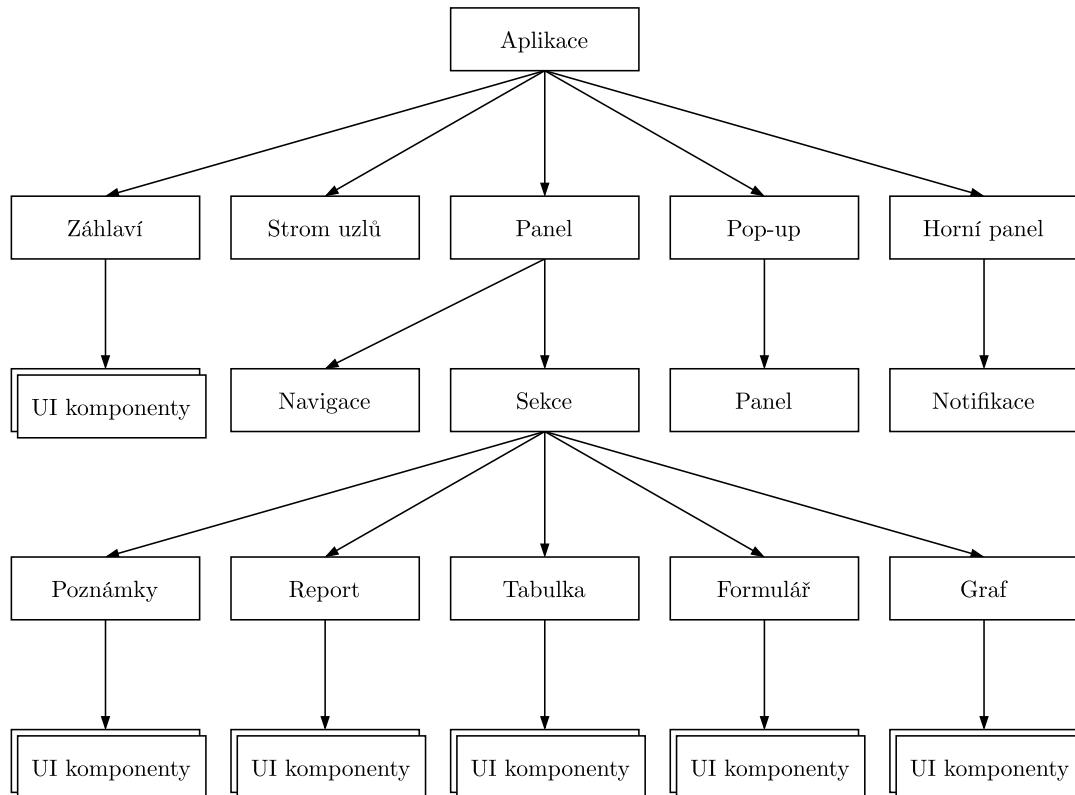
Tab. 2.1: Seznam komponent SPA

Název	Unikátní	Sdílený stav	Popis
Aplikace	✓	✓	Kořenová komponenta
Záhlaví	✓	✗	Záhlaví s logem a navigací
Strom uzlů	✓	✓	Strom s uzly projektu
Panel	✗	✗	Hlavní panel s obsahem
Horní panel	✓	✓	Horní panel pro notifikace
Pop-up	✗	✗	Vyskakovací okno s panelem
Navigace	✗	✗	Výběr sekcí s komponentami
Sekce	✓	✗	Sekce obsahující komponenty
Notifikace	✓	✓	Správce notifikací
Poznámky	✗	✗	Editor pro poznámky
Report	✗	✗	Editor pro tvorbu reportů
Tabulka	✗	✓	Tabulková komponenta
Formulář	✗	✓	Formulářová komponenta
Graf	✗	✗	Komponenta pro graf

---

<sup>3</sup>Protokol WebSocket slouží pro stanovení stálého duplexního spojení se vzdáleným serverem, pomocí kterého může aplikace reagovat na vnější změny bez nutnosti se opakovaně dotazovat.

Komponenty se do sebe postupně zanořují v podobě stromové struktury. Kořenem SPA je komponenta **Aplikace**, jejíž větvení lze vidět na obrázku 2.3. Stromová struktura mezi komponentami vytváří vztah rodiče a potomka, kteří si předávají data napřímo nebo pomocí lokálního rozhraní (více popsáno v kapitole 2.2.3).



Obr. 2.3: Stromová struktura komponent

Ze stromové struktury je patrné, že komponenta **Panel** obsažená v **Pop-up** okně obsahuje podstrom v podobě všech potomků komponenty **Sekce** a **Navigace**. To se při použití aplikace může projevit například otevřením několika **Pop-up** oken, ve kterých bude uživatel pracovat s dalšími instancemi **Panelu**, přičemž veškerá logika a další metody budou vycházet z jediné komponenty. Každá z uvedených komponent může podle potřeby využívat UI (User Interface) komponenty, jako jsou například textové vstupy, tlačítka nebo checkboxy. Přidání nové komponenty není v rámci modularity nikterak problematické a její začlenění tak nenaruší strukturu aplikace. Komponenty pak mohou v rámci svého zanoření vyžadovat od svého rodiče povinné nebo volitelné parametry, jejichž vlastnosti a typ si sami určují. Každá komponenta uchovává svůj vnitřní stav a podle potřeby může ovlivnit svého rodiče, potomka nebo sdílený stav celé aplikace. Kromě dat mohou být mezi komponentami předávány také různé události, jako je například kliknutí myší nebo stisknutí klávesy.

## 2.2.2 Vývoj modulární webové aplikace

### Instalace závislostí

SPA se během vývoje a následně pro produkční build<sup>4</sup> sestavuje prostřednictvím balíčkovacího systému NPM (Node Package Manager), který se zároveň používá pro instalaci a aktualizaci potřebných balíčků. Seznam použitých balíčků pro vývojovou a produkční verzi je spolu s jejich licencí uveden v tabulce 2.2.<sup>5</sup>

Tab. 2.2: Seznam balíčků produkční a vývojové verze

(a) Produkční verze		(b) Vývojová verze	
Název balíčku	Licence	Název balíčku	Licence
mdi-vue	MIT	axios	MIT
moment	MIT	babel/core	MIT
primeicons	MIT	babel-loader	MIT
primevue	MIT	lodash	MIT
qs	BSD-3	storybook	MIT
sass	BSD-3	prettier	MIT
sass-loader	Apache-2.0	ts-loader	MIT
vee-validate	MIT	typescript	MIT
vue-quill	MIT	vite	MIT
vue-resize-observer	MIT	vite-plugin-laravel	MIT
vue-router	MIT	vue/compiler-sfc	MIT
vue3-markdown-it	MIT	vue-loader	MIT
vuedraggable	MIT	vue	MIT
vuex	MIT		
chart.js	MIT		

Pro příklad instalace frameworku Vue.js se z repozitáře balíčků NPM provede příkazem `npm install vue@next`. Po stažení balíčku dojde k jeho zápisu do souboru `package.json`. Tento postup je pro ostatní balíčky z tabulky 2.2 stejný a všechny se tak spolu s jejich verzí nachází v tomto souboru. Při každém dalším spuštění příkazu `npm update` se v souboru `package.json` zkонтrolují verze všech balíčku a bude-li to možné, tak se zastaralé balíčky aktualizují včetně jejich závislostí.

<sup>4</sup>Verze aplikace sestavená pro nasazení do produkčního prostředí.

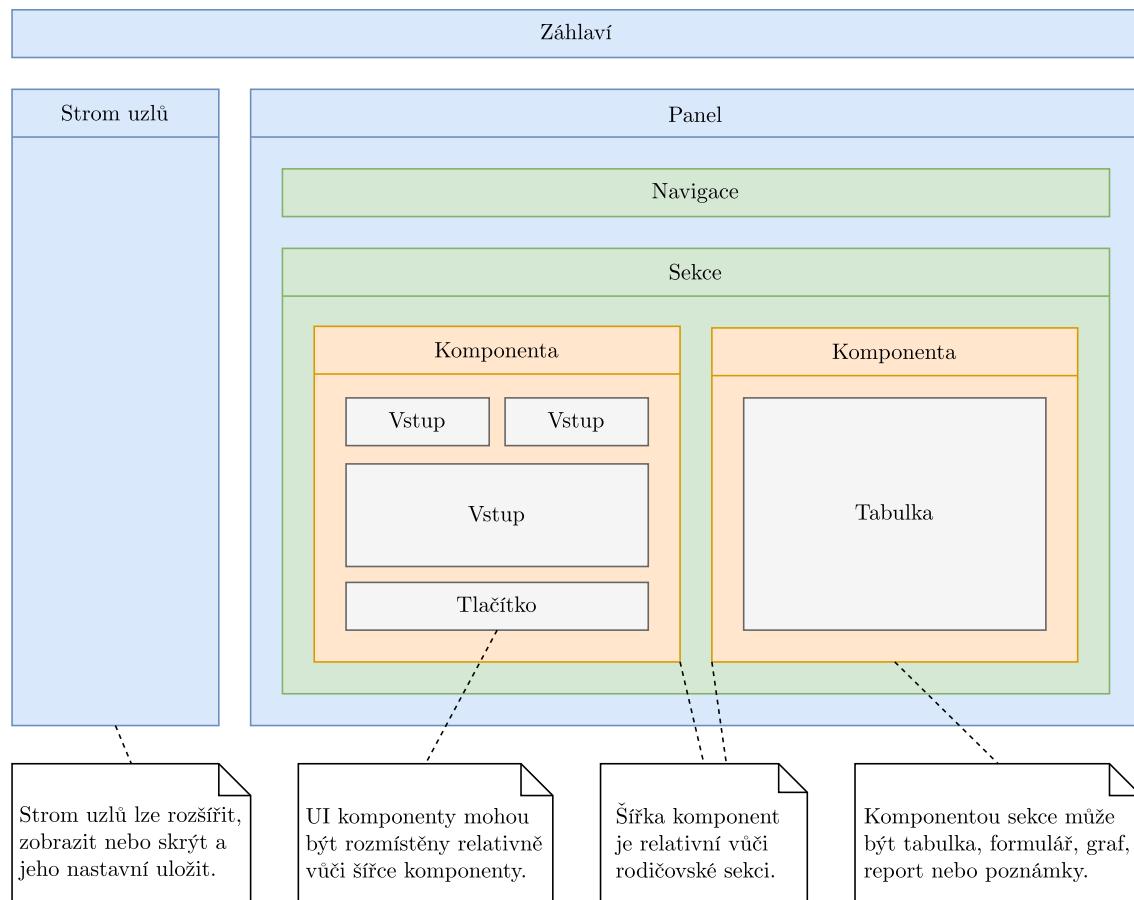
<sup>5</sup>Seznam všech použitých balíčků se vztahuje k datu 30. 4. 2022 a v průběhu dalšího vývoje se může lišit (např. odebráním stávajícího nebo přidáním nového balíčku).

## Vývoj komponent

Pro maximalizaci modularity SPA a zajištění konzistentního postupu vývoje byla pro komponenty stanovena následující pravidla:

- žádná z komponent nebude obsahovat statický obsah,
- pro neměnné hodnoty budou vytvořeny konstanty,
- funkce, služby a komponenty nebudou duplicitní,
- každá dílčí část bude stručně dokumentována,
- typování bude v maximální míře striktní,
- použité jmenné konvence budou konzistentní,
- aktualizovat se budou pouze dotčené elementy,
- komponenty mohou a nemusí sdílet svůj stav.

Na základě bodů výše byly komponenty postupně vyvíjeny a umístovány podle návrhu základní struktury, který je znázorněn na obrázku 2.4. Základní strukturu SPA tvoří záhlaví, strom uzlů a panel obsahující navigaci pro výběr sekcí, ve kterých se mohou nacházet další komponenty, jako je například tabulka nebo formulář.



Obr. 2.4: Rozmístění komponent do základní struktury

Na obrázku 2.5 je zachycena implementovaná základní struktura uživatelského rozhraní, které SPA klientovi sestavilo podle přijatých dat. V levé části se nachází strom všech testovaných uzelů a jejich potomků. Pro každý uzel stromu se v moment jeho výběru sestaví panel s příslušnými sekczemi (např. tabulky, testy, zranitelnosti, přílohy atd.), se kterými může uživatel dále pracovat. V případě této ukázky SPA sestavilo uzel webové stránky s tabulkou penetračních testů.

The screenshot shows the penterep application interface. On the left, there is a tree view of the test results for the 'Brno University of Technology' node. The tree includes branches for 'cyberarena.utko.feec.vutbr.cz', 'API rozhraní', 'Autentizace' (selected), 'SMS kód (druhý faktor)', 'Lokální úložiště', 'PHPSESSID', 'lang', 'Session management', 'Software' (selected), 'Laravel', 'Vue.js', 'Ubuntu', 'Zdroje', '/robots.txt', '/index.php', 'query', '/images/', 'Test', and 'Websockety'. The 'Autentizace' and 'Software' nodes are expanded. The right side of the screen displays a detailed report for the 'Jméinem a heslem' node under the 'Autentizace' category. The report title is 'Jméinem a heslem'. Below it is a 'Informace' button and several icons for navigation and editing. The main content area is titled 'Testy' and contains a table with the following data:

Označení	Úkol	Stav	Nález	Automat.
PTL-WEB-AUTH...	Není omezena délka vstupního pole pro heslo?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Není zakázáno použít znak mezery v hesle?	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Nejsou z hesla vypouštěny mezery?	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Nedochází ke slučování více po sobě jdoucích ...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Nedochází k ořezávání bílých znaků na začátku ...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Jsou mezery v hesle při přenosu správné kódov...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Je povoleno použít v hesle UNICODE znaky?	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Jsou UNICODE znaky interpretovány jako jeden ...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Neznemožní UNICODE znaky možnost ověření ...	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
PTL-WEB-AUTH...	Může si uživatel změnit své heslo?	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

At the bottom of the table, there are navigation buttons for page 1, 2, 3, 4, and so on, with page 2 highlighted in orange. There are also buttons for '10' rows per page and a dropdown menu.

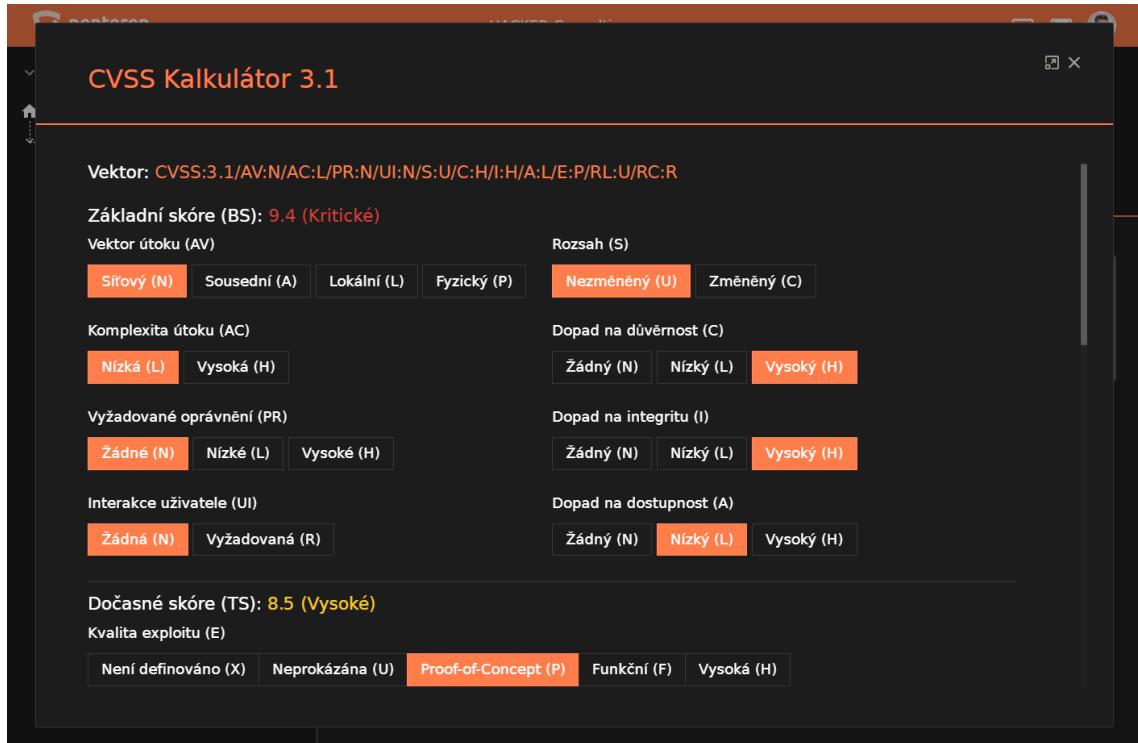
Obr. 2.5: Ukázka uživatelského rozhraní

## Vývoj kalkulátoru pro hodnocení zranitelností

Jedním z dílčích cílů této práce je vývoj rozšiřujícího modulu v podobě kalkulátoru pro hodnocení závažnosti zranitelností. Pro tento účel byl vybrán standard CVSS verze 3.1, přičemž vývoj CVSS kalkulátoru probíhal odděleně a po jeho dokončení byl v rámci modularity přidán jako potomek komponenty **Sekce** ze stromové struktury znázorněné v předchozím obrázku 2.3. Komponenta byla vyvinuta podle metrik a rovnic uvedených v kapitole 1.2.2 a pro zajištění kompatibility je technicky řešena stejně jako všechny ostatní komponenty SPA, tzn. zdrojový kód je napsán v jazyce TypeScript za využití frameworku Vue.js.

CVSS kalkulátor provádí kalkulace na straně klienta a výsledné CVSS skóre spolu s vektorem odesílá pro další zpracování na AS, který obdržené hodnoty uloží do relační databáze. Pokud zranitelnost již CVSS vektor obsahuje, tak se podle jeho hodnoty načte konfigurace kalkulátoru pro možnost změnit jakoukoliv hodnotu a závažnost dané zranitelnosti tak znovu přepočítat.

Podle zvolených hodnot jednotlivých metrik komponenta vypočítá základní skóre, dočasné skóre a skóre prostředí, na jejichž základě je stanoveno výsledné CVSS skóre a vygeneruje se CVSS vektor [10]. Komponenta zobrazuje výsledky hned po volbě metrik a uživatel má tak okamžitou představu o závažnosti dané zranitelnosti. Jakmile je výsledné hodnocení závažnosti potvrzeno ze strany uživatele, tak je skóre s vektorem odesláno k uložení v relaci se zranitelností, pro kterou byla závažnost vypočtena. Komponenta CVSS kalkulátoru je zachycena na obrázku 2.6.



Obr. 2.6: Komponenta CVSS kalkulátoru

## 2.2.3 Návrh a implementace lokálního rozhraní

### Řešená problematika

Přímá komunikace komponent ve vztahu rodiče a potomka je efektivní, nicméně při hlubším zanoření se začíná stávat problematickou. Příkladem takového případu může být zanoření komponenty o 4 úrovně. Pokud by potřebovala nejhlobubejší zanořená komponenta komunikovat s komponentou na 1. úrovni, tak musí využít svého rodiče a takto pokračovat dál až ke konečné rodičovské komponentě. Takové řešení je sice funkční, ale značně neefektivní. Pro implementaci je nutné upravit komponenty na všech úrovních zanoření a stejným způsobem postupovat v případě pozdějších změn. Tím je narušen základní princip modularity, protože komunikace závisí bez další funkcionality na více prvcích než je zdroj a cíl.

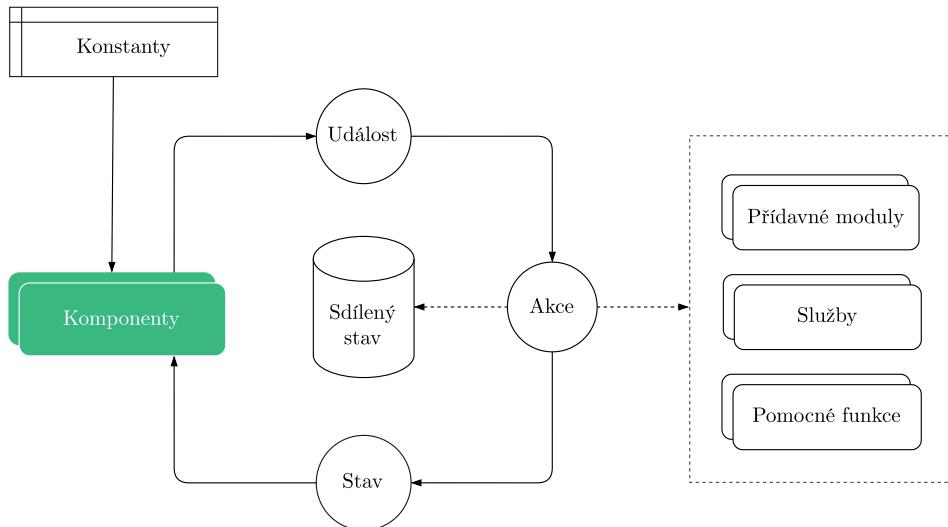
## Návrh SPA API

Splnění požadavku vysoké modularity a škálovatelnosti webové platformy vyžaduje navíc také zpřístupnění komunikace komponent se službami, úložištěm webového prohlížeče a protokolem WebSocket. Jednotlivé komponenty proto musí být schopny přistupovat ke všem těmto částem mimo vlastní strukturu bez nadbytečné komunikace. Pro zajištění uvedených požadavků bylo navrženo lokální API, které tyto části propojuje. Podstatou SPA API je poskytnout rozhraní, ve kterém bude možné zajistit komunikaci komponent bez ohledu na jejich zanoření. Návrh rozhraní je znázorněn na obrázku 2.7 a skládá se z následujících částí:

- **Sdílený stav** představuje řešení pro zajištění dostupnosti datových objektů aplikace napříč komponentami bez ohledu na jejich zanoření. Více komponent může být při běhu ovlivňováno stejnými daty a na jejich změnu musí patřičně reagovat. Model SPA API vychází z architektury Flux, která se v knihovně React používá k průtoku dat a skládá se z části Store – stav a logika aplikace, Action – pomocné metody k přenosu dat, Dispatcher – distributor přenosu dat a View – komponenty přistupující ke Store [30].
- **Konstanty** slouží pro minimalizaci statického obsahu, uložení a snazší úpravu konfiguračních parametrů nebo jazykových mutací. Díky tomuto rozdělení lze při pozdější úpravě přímo přistoupit ke konkrétním konstantám bez nutnosti hledat hodnoty ve zdrojovém kódu a zároveň tak zamezit vzniku duplicitního obsahu. Konstanty lze dále pro lepší přehlednost seskupovat podle jejich účelu, datového typu a komponent, ve kterých jsou použity. Pro konstanty platí, že jsou neměnné a jejich hodnoty tak nelze za běhu programu změnit.
- **Události a akce** jsou základními prvky životního cyklu SPA API. Události jsou vyvolané samotnými komponentami na základě jejich metod, které mohou být obsluhovány automaticky podle vnitřního stavu dané komponenty nebo mohou být vyvolány přímou interakcí ze strany uživatele. Akce potom pracuje s výstupy ostatních prvků (např. se sdíleným stavem aplikace nebo příchozími daty z vnějšího API), které předává cílovým komponentám. Komponenty na tyto akce patřičně reagují a to buď svým opakováním sestavením, změnou stavu, nebo vyvoláním nové události.
- **Služby** jsou v modelu SPA API určeny ke komunikaci s vnější stranou, kterou jsou nejčastěji webové služby<sup>6</sup>. Komponenty prostřednictvím služeb odesílají HTTP požadavky, jejichž odpověď je jim navrácena s odpovídající strukturou (data, návratová zpráva, status kód atd.). Požadavek se skládá z hlavičky, autorizačních údajů (např. token nebo přihlašovací údaje) a těla požadavku, nejčastěji ve formátu JSON (JavaScript Object Notation).

---

<sup>6</sup>REST API, GraphQL API, WebSocket API nebo SOAP (Simple Object Access Protocol).



Obr. 2.7: Návrh lokálního rozhraní SPA API

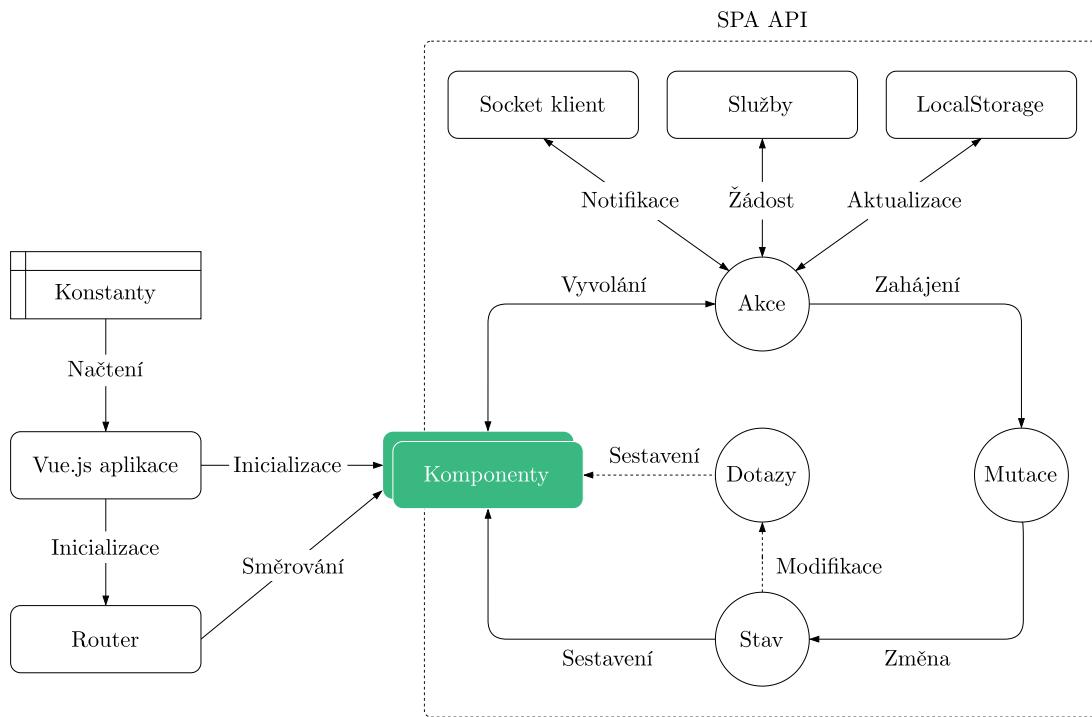
- **Pomocné funkce** umožňují využívat logické metody v rámci celé SPA. Pokud aplikace komunikuje s externími webovými službami z více komponent, tak může nastat situace, kdy je potřeba provést stejnou operaci nad více různými návratovými hodnotami. S modulárním přístupem by bylo nevhodné opakováně implementovat stejnou funkci ve všech dílčích komponentách, a proto jsou takové funkce sdílené a dostupné všem částem aplikace.
- **Přídavné moduly** v obecném modelu SPA API představují taková rozšíření aplikace, která vstupují do komunikace s další funkcionalitou. Jejich implementace je ovlivněna použitými technologiemi a může se jednat buď o vlastní moduly, nebo řešení třetích stran. Hlavním cílem přídavných modulů je zajistit škálovatelnost aplikace bez nutnosti zásahu do její hlavní struktury.

## Implementace SPA API

Hlavní část rozhraní pro sdílení a správu stavu komponent byla implementována za použití technologie Vuex<sup>7</sup>. Konstanty jsou neměnné a do aplikace se načítají při jejím vytvoření, přičemž komponentám jsou poskytnuty ještě před inicializací. Komponenty se v moment jejich vytvoření stávají součástí životního cyklu celé Vue.js aplikace a přes Akce mohou získat nebo modifikovat data z vnějšího zdroje prostřednictvím Služeb. Dále mohou pracovat s daty klienta z úložiště webového prohlížeče LocalStorage. Prostřednictvím Akce lze vyvolat Mutace a změnit tak sdílený Stav. Data mohou být volitelně navrácena přes Dotazy, které fungují jako modifikátory a vyžádaný sdílený stav aplikace před navrácením příslušné komponentě dodatečně upravují (např. pomocí filtrovacích nebo statistických funkcí).

<sup>7</sup>Vuex je knihovna pro správu a centralizaci stavu všech komponent Vue.js aplikace.

Vzhledem k tomu, že SPA funguje na principu CSR a vykresluje se na straně klienta, tak jsou komponenty vnitřně směrovány za účelem načtení příslušné části podle URL a zadaných parametrů. Směrování včetně uchovávání jeho historie zajišťuje Router. Poslední součást rozhraní je Socket klient, který udržuje trvalé spojení se serverem a aktivně naslouchá příchozím notifikacím (např. oznámení o přidání nového záznamu). Notifikace mohou navíc obsahovat pokyn k aktualizaci komponenty. Výsledná implementace SPA API je znázorněna na obrázku 2.8.



Obr. 2.8: Implementace lokálního rozhraní SPA API

## 2.3 Vlastní návrh a vývoj backendové části

Stěžejním bodem mezi klientskou a datovou částí platformy je backend aplikačního serveru, pro jehož vývoj byl zvolen PHP framework Laravel. Ten tvoří z hlavní části REST API, prostřednictvím kterého se SPA dotazuje na nová data, nebo odesílá požadavky k jejich zpracování. Zdrojem dat je pro backend relační databáze MySQL, výsledky automatických penetračních testů nebo uživatelské vstupy z SPA. Soubory se ukládají přímo na server nebo vzdáleně na cloudové úložiště. Veškeré požadavky a vzniklé chyby jsou automaticky logovány. Pro každou operaci směřovanou na aplikační server musí být uživatel autentizován a zároveň autorizován daný požadavek vykonat. Tato kapitola blíže popisuje proces fungování aplikačního serveru včetně praktických příkladů a ukázek jednotlivých částí.

### 2.3.1 Návrh a vývoj REST API

Jakmile se klient autentizuje vůči UMS, tak už další komunikaci provádí pouze prostřednictvím SPA, která na základě událostí a požadavků klienta může žádat o nová data pro aktualizaci svého obsahu. Všechny tyto akce mají podobu API volání směrovaného na AS. REST API všechny požadavky obsluhuje, přičemž pracuje se statickými a dynamickými daty relační databáze MySQL. Vzhledem ke komplexnosti celé databáze, která čítá 52 tabulek, se návrh a vývoj REST API řídil opakováně následujícími body:

1. určení možného požadavku SPA,
2. zvolení HTTP metody a cesty pro API volání,
3. stanovení povinných a nepovinných parametrů,
4. praktická implementace obsluhy API volání,
5. tvorba technické dokumentace.

#### Určení možného požadavku SPA

Vzhledem k minimalizaci statického obsahu na straně SPA musí každá komponenta požádat AS o data pro dokončení procesu svého sestavení. Požadovaná statická data ovlivňují tzv. uzly, které představují dynamická data MySQL databáze. Uzly jsou určitého typu, mají svého rodiče, potomky a další vlastnosti. Každý uzel má své unikátní identifikační číslo, tzv. GUID (Globally Unique Identifier), které je důležité pro identifikaci uzlu v požadavku odeslaného ze strany SPA na REST API. Dále je v rámci struktury SPA stanoveno, jaká data bude konkrétní komponenta potřebovat a jaká bude struktura návratové hodnoty (např. klíče JSON objektu).

Každá cesta pro API volání je definována pod skupinou určenou podle třídy, která daný požadavek obsluhuje. Všechny skupiny jsou zároveň kontrolovány, zda je zdroj požadavku oprávněn API volání vykonat, čímž je zajištěna autorizace celého rozhraní. Pokud uživateli vyprší relace (session), jejíž platnost je nastavena na dvě hodiny, tak je přesměrován na UMS, kde se musí opětovně autentizovat. Pro obsluhu požadavků byly implementovány následující třídy:

- **NodeController** – zpracování uzlů a dat komponent,
- **NotesController** – operace s poznámkami uživatelů,
- **TreeController** – skládání a přepočítávání stromu uzlů,
- **TestController** – správa penetračních testů všech uzlů,
- **ChecklistController** – kontrolní seznamy a jejich kategorie,
- **VulnerabilityController** – správa nalezených zranitelností,
- **ReportController** – generování závěrečné zprávy a správa šablon,
- **FileController** – správa nahraných příloh (souborů),
- **InvitationController** – obsluha pozvánek pro připojení do platformy.

## Zvolení HTTP metody a cesty pro API volání

Požadavky se kromě obsahu přenášené zprávy mohou lišit v HTTP metodě, kterou volají. Data mohou být buď žádána (GET), vytvářena (POST), upravována (PUT) nebo odstraňována (DELETE). Práce s dynamickými daty (uzly) zahrnuje všechny tyto metody, zatímco v případě statických dat je použita pouze HTTP metoda GET. Cílem SPA není měnit schéma penetračního testování, a proto je tento proces zahrnut pouze v počáteční fázi definování dat. HTTP požadavky potom obsluhují třídy, které byly uvedeny v předchozí podkapitole. Příklad cesty pro API volání metodou GET je znázorněn na obrázku 2.9.

GET https://as.penterep.io/api/v1/checklists/categories

Obr. 2.9: Ukázka cesty pro metodu GET

## **Stanovení povinných a nepovinných parametrů**

Cesta API volání v sobě zahrnuje parametry funkce, která dané volání obsluhuje. Na základě požadavku potom funkce provede sekvenci SQL dotazů, přičemž může použít parametry obsažené v samotném dotazu (např. unikátní identifikátor uzlu). API volání může být zpracováno více způsoby a předané parametry tak mohou být v některých případech volitelné. Přestože je PHP dynamicky typovaný jazyk, tak je pro všechny parametry a návratové hodnoty specifikován datový typ, aby mohly být snáze odhaleny chyby při běhu programu.

Ukázku existující cesty pro API volání metody GET lze vidět na obrázku 2.10. Cesta je stejná jako v předchozím případě, ale obsahuje navíc parametr, jehož hodnota se předává obsluhující funkci. Každá taková funkce může zároveň přistupovat k objektu HTTP požadavku, konkrétně hlavičky, která obsahuje například IP adresu zdroje požadavku, nebo payload<sup>8</sup> v případě metody POST a PUT.

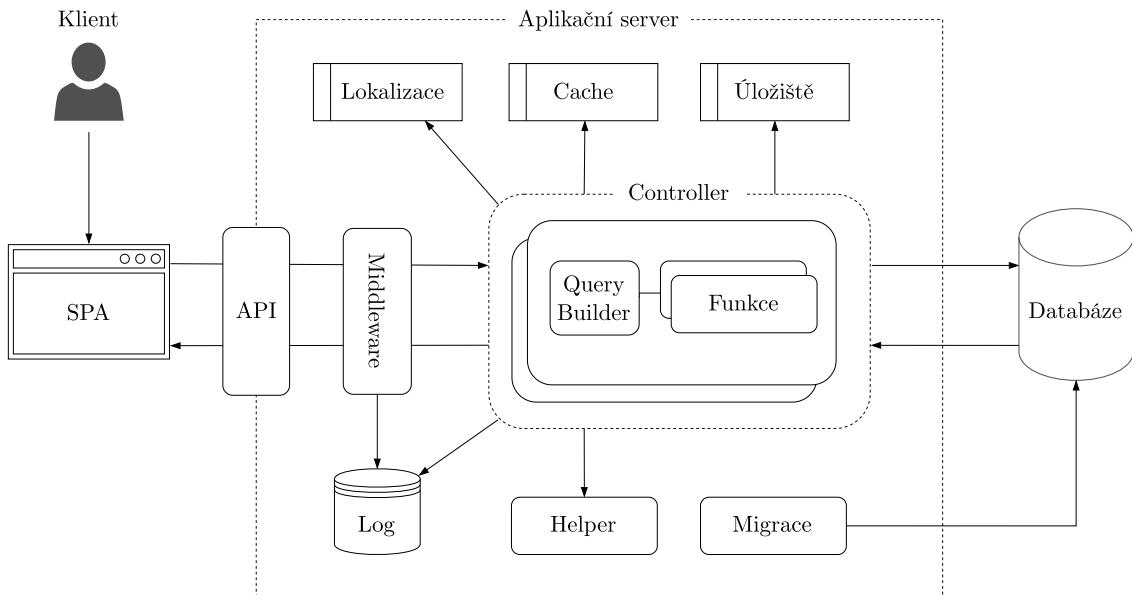
GET <https://as.penterep.io/api/v1/checklists/categories/{guid}>  
  └── guid  
      └── parameter

Obr. 2.10: Ukázka cesty s parametrem pro metodu GET

<sup>8</sup>Tělo požadavku obsahující data, která byla do požadavku přidána zdrojem, který ho odeslal. Častým příkladem HTTP payloadu jsou data z HTML formuláře odeslaná metodou POST.

## Praktická implementace obsluhy API volání

Implementace obsluhy požadavků na REST API a vnitřní struktura aplikační části je znázorněna na obrázku 2.11. Každé API volání je kontrolované třídou **Middleware**, která HTTP žádost autorizuje a zaznamenává do **Logu**. Po úspěšné autorizaci je požadavek předán příslušnému **Controlleru** pro další zpracování. **Controller** komunikuje s relační databází přes rozhraní **Query Builder**, prostřednictvím kterého lze efektivně skládat SQL dotazy včetně jejich zabezpečení a optimalizace. Dotazy se skládají pomocí statických metod třídy, která využívá aktivní spojení s databází. Všechny metody se na sebe řetězí a překládají do SQL dotazu, který se vykonává v databázi. Struktura databáze se jednorázově vytváří pomocí **Migrací** (skriptů) při prvním spuštění aplikačního serveru. Pokud dotaz vrací data, tak je možné na konci řetězce použít pomocné metody třídy **Helper** pro úpravu návratových dat do specifického formátu (např. datový typ pole nebo objekt JSON).



Obr. 2.11: Implementace a komunikace REST API

**Controller** může také využít **Cache** úložiště pro rychlý přístup k přednáčteným datům (např. názvy vlastností). Webová platforma je vícejazyčná, takže nepoužívá statické návratové zprávy, ale podle nastavení uživatele si načte příslušnou jazykovou **Lokalizaci**. Pokud uživatelé pracují s mediálními soubory, jako je například video, tak se soubory ukládají na serverové nebo cloudové úložiště, přičemž reference na soubor je s dalšími informacemi (např. formát souboru) uložena v databázi. Veškeré chyby, které při zpracování požadavku nastanou, jsou zaznamenány do **Logu**.

## Dokumentace REST API

Velké množství požadavků vyžaduje udržovat přehled nad celým REST API. Za účelem testování a možné pozdější úpravy jsou všechna implementovaná API volání patřičně dokumentována. K tomu byl použit open source nástroj Swagger, který je jedním z nejpoužívanějších řešení pro návrh, tvorbu, dokumentaci a testování REST API [31]. Dokumentace není určena pouze pro čtení, ale zároveň umožňuje jednotlivá volání testovat včetně možnosti zadávání parametrů. Ukázka dokumentace API volání pro získání uzlu je zachycena na obrázku 2.12.

The screenshot shows the Swagger UI interface for a REST API endpoint. At the top, it specifies a **GET** request to **/nodes/{GUID}** with the description "Find node by GUID". Below this, it states "Returns a specific node according to the specified GUID".  
**Parameters:** A table with one row labeled "GUID \* required" and "string (path)". The value input field contains "21e1c1ee-ea02-4bef-916f-8b205a41e8a8".  
**Responses:** A table with one row for "Code" (200) and "Details". The "Details" section shows the "Response body" as a JSON object:

```
{  "success": true,  "message": "Node content has been successfully found",  "data": {    "name": "Firma - infrastruktura",    "breadcrumb": [      {        "guid": "21e1c1ee-ea02-4bef-916f-8b205a41e8a8",        "name": "Firma - infrastruktura",        "icon": "chart-bar"      }    ],    "tabs": [      {        "name": "Project-info",        "title": "Informace",        "icon": "information-outline",        "type": "L_TAB_TYPE_STANDARD",        "order": 10      }    ]}}
```

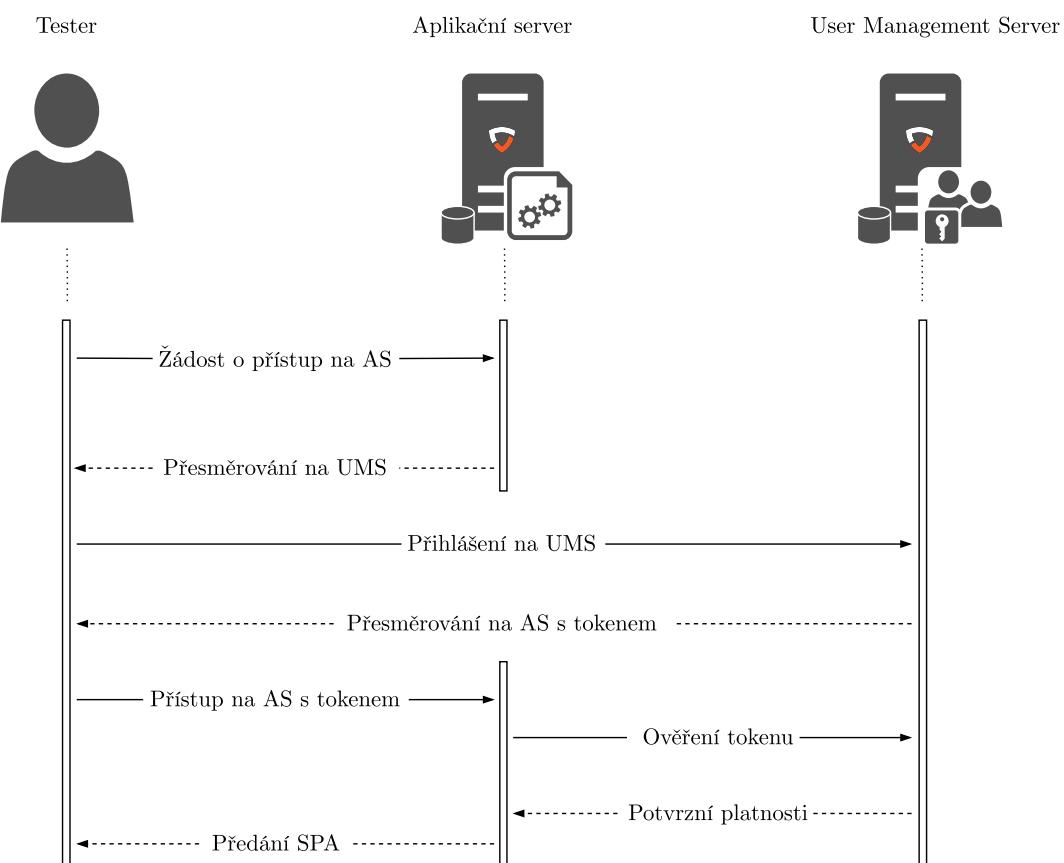
Below the response body are "Copy" and "Download" buttons.  
The "Response headers" section lists standard HTTP headers: cache-control, connection, content-security-policy, content-type, date, feature-policy, keep-alive, referer-policy, server, x-content-type-options, x-frame-options, x-ratelimit-limit, x-ratelimit-remaining, and x-xss-protection.

Obr. 2.12: Dokumentace REST API

### 2.3.2 Autentizace klienta

Veškeré uživatelské účty a jejich licence jsou registrovány a spravovány na UMS.<sup>9</sup> Pro přístup na AS se uživatel musí nejprve autentizovat vůči UMS, načež je navrácen zpět s přiděleným tokenem. Celý proces autentizace je znázorněn na obrázku 2.13 a skládá se postupně z následujících kroků:

1. klient bez předchozí autentizace přistoupí na adresu AS,
2. AS přesměruje klienta s parametrem své adresy na UMS,
3. klient se svými přihlašovacími údaji autentizuje vůči UMS,
4. po přihlášení je klient přesměrován s tokenem zpět na AS,
5. AS ověří obdržený autentizační token klienta vůči UMS,
6. je-li token platný, tak je klientovi předána SPA.



Obr. 2.13: Průběh úspěšné autentizace

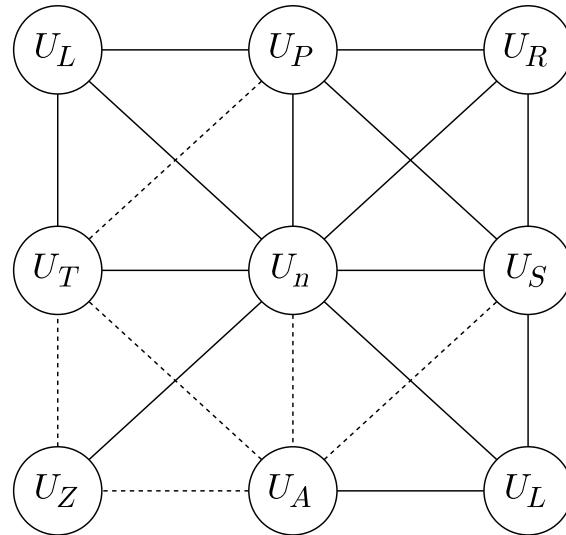
Po úspěšném dokončení procesu autentizace se aktivuje relace s platností dvou hodin od poslední provedené akce. Každý požadavek klienta a všechna API volání jsou kontrolována, a pokud není klient v době jejich vykonání autentizován, nebo relace klienta vyprší, tak je celý proces autentizace opakován.

<sup>9</sup>Všechna AS se v produkčním prostředí autentizují vůči jednomu stejnemu UMS.

Další přidanou ochranou je nastavení techniky Cross-origin Resource Sharing (CORS) na hodnotu „same-origin policy“, která povoluje komunikaci zdroje a cíle pouze v rámci stejné domény. V takovém případě by bez přidaných výjimek nemohlo SPA přistupovat ke zdrojům jiného AS než toho, které předalo aplikaci klientovi.

### 2.3.3 Model uzlu

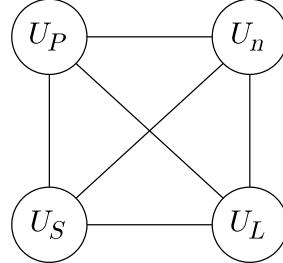
Veškerá dynamická data, se kterými platforma pracuje, jsou řešena jako uzly. Každý uzel je určitého typu a vůči jiným uzlům má definovaný vztah. Stejně jako v případě komponent SPA, tak i zde jsou uzly ve vztahu rodiče a potomka. Webová platforma obsahuje vždy jeden globální uzel, jehož potomci jsou uzly typu projekt, uživatel, oprávnění, šablona reportu nebo log pro zaznamenání každé provedené operace nad uzlem. Uzly projektu se dále dělí na testovatelné a netestovatelné uzly. Testovatelné mají zpravidla více vztahů s jinými uzly, zatímco netestovatelné méně. Příkladem testovatelného uzlu je webová aplikace, jejímž potomkem může být uzel typu webová stránka. Kromě uzlu určitého typu může mít také každý testovatelný uzel potomky typu soubor<sup>10</sup>, zranitelnost, test, automatický test a log. Každý testovatelný uzel může být zároveň zahrnut v reportu, který je také uzlem. Všechny možné vztahy testovatelného uzlu projektu jsou znázorněny na obrázku 2.14.



Obr. 2.14: Vztah mezi testovatelnými uzly, kde  $U_P$  je uzel projektu,  $U_n$  uzel typu  $n$ ,  $U_R$  uzel reportu,  $U_T$  uzel testu,  $U_S$  uzel souboru,  $U_Z$  uzel zranitelnosti,  $U_A$  uzel automatického testu a  $U_L$  uzel logu. Plná čára značí přímou relaci mezi uzly, zatímco přerušovaná volitelnou, kdy vztah nemusí mezi danými uzly vždy existovat.

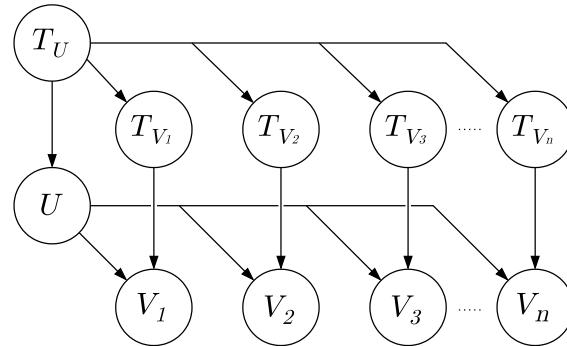
<sup>10</sup>Uzel typu soubor představuje nejčastěji nahranou přílohu k rodičovskému uzlu.

Ne vždy je uzel projektu určen k testování, a proto existují uzly projektu, pro které lze nahrávat pouze přílohy, přičemž veškeré operace nad uzlem se stejně jako v případě testovatelného uzlu zaznamenávají do uzlu typu log. Vztah takového uzlu je vyjádřen na obrázku 2.15, kde existují pouze přímé relace. Příkladem takových uzlů jsou uživatelé, reporty nebo poznámky.



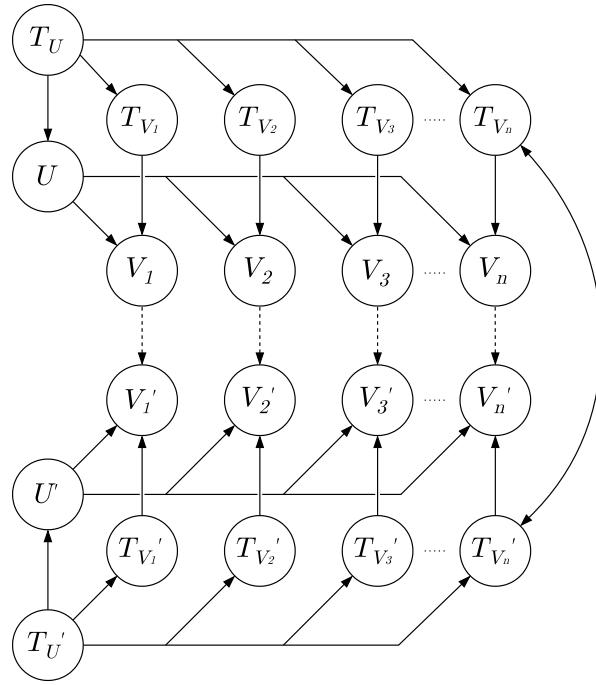
Obr. 2.15: Vztah mezi netestovatelnými uzly projektu, kde  $U_P$  je uzel projektu,  $U_n$  uzel typu  $n$ ,  $U_S$  uzel souboru a  $U_L$  uzel logu.

Pro každý uzel může existovat vlastnost, která byla k typu uzlu přiřazena během fáze tvorby schémat penetracního testování (proces tvorby statických dat popsaný v kapitole 2.1.2). Při vytvoření vlastnosti automaticky vzniká relace k uzlu a definici, které ji naleží. Obecný model vlastnosti uzlu je znázorněn na obrázku 2.16.



Obr. 2.16: Obecný model vlastnosti uzlu, kde  $U$  je obecným uzlem,  $T_U$  je typ uzlu,  $T_{V_1}, T_{V_2}, \dots, T_{V_n}$  jsou typy vlastností a  $V_1, V_2, \dots, V_n$  jsou vlastnostmi uzlu.

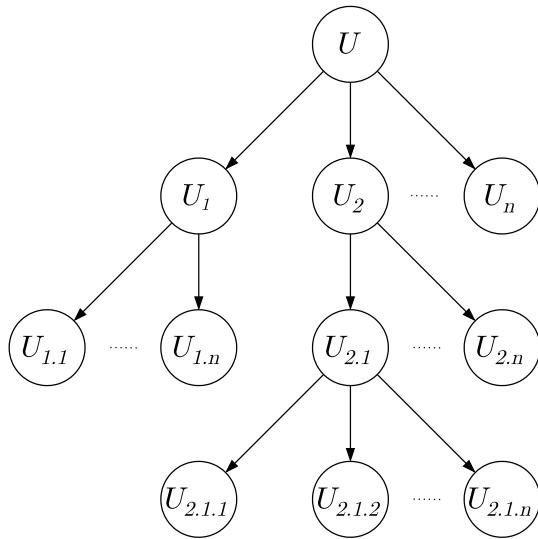
Pro určité typy vlastností může existovat speciální případ, kterým je propojení vlastností mezi více uzly. V takovém případě je pro typ vlastnosti vytvořena relace, která určuje jaké vlastnosti mohou být mezi sebou propojeny. Pro takovou vlastnost se neukládá vlastní hodnota, jako tomu bylo u předchozího modelu, ale hodnotu přebírá z propojené vlastnosti uzlu. Obecný model je znázorněn na obrázku 2.17.



Obr. 2.17: Obecný model propojených vlastností uzlu, kde  $T_U$  je typem uzlu,  $T_{V_1}, T_{V_2}, \dots, T_{V_n}$  jsou typy vlastností a  $V_1, V_2, \dots, V_n$  jsou vlastnostmi uzlu. Plná čára představuje přímou relaci a přerušovaná volitelnou.

### 2.3.4 Strom uzlů

Dalším důležitým prvkem platformy je stromové zobrazení uzlů v rámci vybraného projektu. Každý projekt je zároveň také uzlem, přičemž obsahuje potomky v podobě poduzlů, které se stejným způsobem mohou dále zanořovat. Obecně je tato stromová struktura znázorněna na obrázku 2.18.



Obr. 2.18: Strom uzlů, kde  $U_1, U_2, \dots, U_n$  jsou uzly stromu.

## Řešená problematika

Veškeré uzly a jejich hodnoty se ukládají jako řádky do relační databáze, kterou v navrženém modelu představuje dynamická databáze. Samotné založení nového uzlu není nijak problematické, protože stačí v uzlu vytvořit relaci na jeho rodiče. Problémem je nicméně kompletní výpis všech uzelů a jejich potomků do stromové struktury. Řešením problematiky by mohla být rekurzivní funkce do doby, kdy uzel obsahuje potomky, jako je ukázáno v algoritmu 1.

---

### Algoritmus 1 Rekurzivní získávání potomků uzlu

---

**Input:** uzel  $n$

```
1: function ZISKATPOTOMKY( $n$ )
2:   if  $n.potomci$  then
3:     for all  $u \in n.potomci$  do
4:        $u.potomci \leftarrow ZiskatPotomky(u)$             $\triangleright$  rekurzivní volání funkce
5:     end for
6:   else
7:     return  $n$                                  $\triangleright$  vrací uzel se všemi potomky
8:   end if
9: end function
```

---

Přestože lze tímto způsobem všechny uzly a jejich potomky vypsat, tak je řešení z důvodů časové složitosti značně neefektivní. Náročnost výběru uzelů projektu a všech jejich potomků lze vyjádřit pomocí časové složitosti algoritmu využívající rekurzi. Ta bude počítána podle délky stromu od kořenového uzlu (projektu) po nejhлouběji zanořené uzly vynásobené jejich počtem. Rekurzivní funkce navíc v každém svém vykonání iteruje svými potomky. Ve výsledku má taková operace kvadratickou složitost  $\mathcal{O}(n^2)$ , která se nemění ani v případě procházení podstromu.

## Traverzování kolem stromu

Optimálním řešením pro ukládání a práci se stromovou strukturou v relační databázi je tzv. traverzování kolem stromu. Relační databáze nicméně neumožňují ukládat pokročilé datové struktury, jako je strom uzelů. V relačním modelu je tedy nutné stromovou strukturu ukládat jako řádky do tabulek a atributy zvolit tak, aby bylo možné výběrem dat v aplikaci strom sestavit. Tabulka pro ukládání uzelů stromu bude pro tento účel obsahovat atribut cizího klíče uzlu, levou pozici, pravou pozici a hloubku uzlu. Počet stromů je roven počtu projektů, takže pro každý uzel existuje relace na uzel, který je projektem a tedy i kořenem stromu. Strom nebo podstrom musí být při každé operaci zároveň aktualizován jeho přepočítáním [32].

Náročnost traverzování kolem stromu je logaritmická  $\mathcal{O}(\log_2 n)$ . Žádné rekurzivní volání funkce zde není potřeba a veškeré operace nad stromem uzelů pro vybraný projekt lze provádět efektivněji pouze pomocí SQL dotazování.

## Přidání a odstranění uzlu

Základní operací je přidání nového uzlu do stromu, při kterém se všechny poduzly podle levé a pravé pozice rodičovského uzlu inkrementují o 2. Následně se nový uzel vloží s hloubkou a pravou pozicí vyšší o jednu pozici a levou pozicí podle pravé pozice jeho rodiče. Podrobně je tento postup vyjádřen algoritmem 2.

---

### Algoritmus 2 Operace přidání uzlu

---

**Input:** strom uzelů  $s$ , rodičovský uzel  $r$  a nový uzel  $n$

```

1: for all  $u \in s.uzly$  do
2:   if  $u.leva > r.prava$  then            $\triangleright$  přepočítá levé pozice uzelů stromu
3:      $u.leva \leftarrow u.leva + 2$ 
4:   end if
5:   if  $u.prava > r.prava$  then            $\triangleright$  přepočítá pravé pozice uzelů stromu
6:      $u.prava \leftarrow u.prava + 2$ 
7:   end if
8: end for
9:  $n.leva \leftarrow r.prava$ 
10:  $n.prava \leftarrow r.prava + 1$ 
11:  $n.hloubka \leftarrow r.hloubka + 1$ 

```

---

Podobně je to i v případě odstranění. Rozdíl je v tom, že se nemaže pouze uzel, ale celý jeho podstrom, takže je třeba smazat s uzlem zároveň všechny jeho potomky a následně strom opět přepočítat. Tato operace je vyjádřena algoritmem 3.

---

### Algoritmus 3 Operace odstranění uzlu

---

**Input:** strom uzelů  $s$  a mazaný uzel  $n$

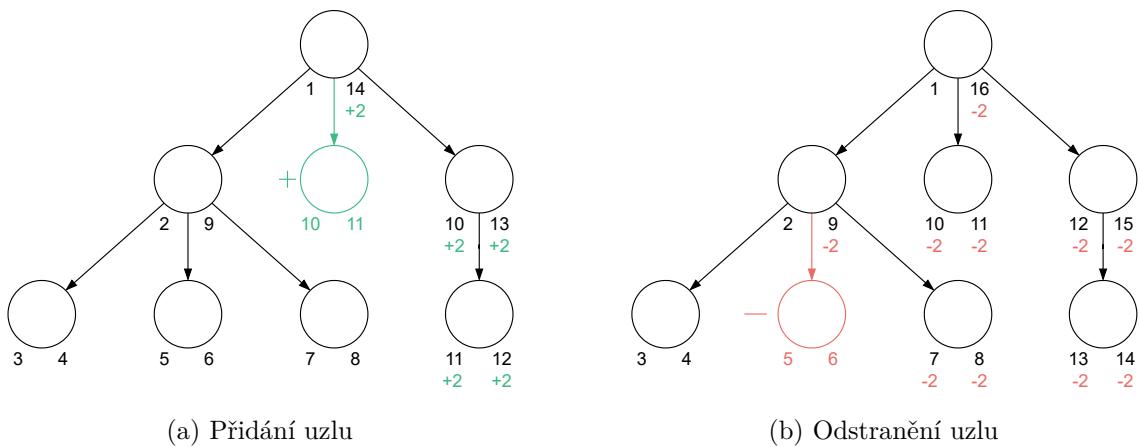
```

1: for all  $u \in s.uzly$  do
2:   if  $u.leva \geq n.leva \& u.prava \leq n.prava$  then
3:     delete  $u$             $\triangleright$  odstraní uzel podstromu mazaného uzlu
4:   else if  $u.leva > n.prava$  then            $\triangleright$  přepočítá levé pozice uzelů stromu
5:      $u.leva \leftarrow n.prava - n.leva + 1$ 
6:   else if  $u.prava > n.prava$  then            $\triangleright$  přepočítá pravé pozice uzelů stromu
7:      $u.prava \leftarrow n.prava - n.leva + 1$ 
8:   end if
9: end for

```

---

Operaci přidání a odstranění lze pro lepší představu vyjádřit i graficky. Názorný příklad obou operací je znázorněn na obrázku 2.19.



Obr. 2.19: Operace přidání a odstranění uzlu

## Klonování uzlu

Speciální operací nad stromem je klonování uzlu. Pokud je uzel klonován včetně jeho potomků, tak se vybírá celý podstrom uzlu, jehož potomci jsou postupně vytvořeni a následně se celý podstromu přepočítá (viz algoritmus 4).

---

**Algoritmus 4** Operace klonování uzlu

---

**Input:** strom uzlů  $s$ , rodičovský uzel  $r$ , klonovaný uzel  $n$  a původní uzel  $p$

- ```

1:  $n \leftarrow p$ 
2:  $n.nazev \leftarrow p.nazev + 1$                                 ▷ inkrementuje názvu původního uzlu
3: for all  $u \in s.uzly$  do
4:   if  $u.leva > r.prava$  then                                ▷ přepočítá levé pozice uzlů stromu
5:      $u.leva \leftarrow u.leva + 2$ 
6:   end if
7:   if  $u.prava > r.prava$  then                                ▷ přepočítá pravé pozice uzlů stromu
8:      $u.prava \leftarrow u.prava + 2$ 
9:   end if
10: end for
11:  $n.leva \leftarrow r.prava$ 
12:  $n.prava \leftarrow r.prava + 1$ 
13:  $n.hloubka \leftarrow r.hloubka + 1$ 

```

Režim bez potomků je totožný s operací přidání nového uzlu. Hlavní rozdíl je zde v tom, že se při této operaci pro každý klonovaný uzel duplikují jeho hodnoty, což dělá operaci náročnější z důvodu kontroly integrity názvu uzlu, který musí být vždy unikátní pro přímou množinu poduzlů rodičovského uzlu, ve které se uzel klonuje. Během klonování se proto automaticky inkrementuje název nového uzlu.

## Výpis stromu uzlů

Nejpoužívanější operací nad stromem uzlů je výpis všech jeho potomků do stromové struktury. Pomocí traverzování se pro výpis stromu projektu provede jeden SQL dotaz, který vybere všechny uzly projektu seřazené postupně podle levé strany. Vybraný strom je následně převeden do JSON objektu a odeslán SPA, která ho vykreslí. Ukázku návratového JSON objektu s kořenovým uzlem, dvěma potomky a jejich vlastnostmi, lze vidět ve výpisu 2.1.

Výpis 2.1: Výpis stromu uzlů ve formátu JSON

---

```
1  {
2      "key": "08c2345c-aac2-4601-8185-17c2960567d5",
3      "title": "Brno University of Technology",
4      "tests": 100,
5      "progress": 74.75,
6      "vulnerable": 1,
7      "icon": "chart-bar",
8      "children": [
9          {
10             "key": "16217d72-aa6d-447e-978a-571d855414be",
11             "title": "cyberarena.utko.feec.vutbr.cz",
12             "tests": 45,
13             "progress": 69.47,
14             "vulnerable": 0,
15             "icon": "web"
16         },
17         {
18             "key": "53791b32-bb3h-227k-469b-39nb1454321e",
19             "title": "openstack.utko.feec.vutbr.cz",
20             "tests": 55,
21             "progress": 80.00,
22             "vulnerable": 1,
23             "icon": "web"
24         }
25     ]
26 }
```

---

## 2.3.5 Generátor reportu

### Řešená problematika

Každý penetrační test by měl být správně zakončen závěrečnou zprávou – reportem. Report představuje výsledek a shrnutí penetračního testování, jehož obsahem je stav závažnosti a všechny nalezené zranitelnosti testovaného cíle. Obsahem reportu mohou být také reference na provedené testy, doporučení k nápravě nebo poznámky pro vývojáře. Na základě reportu by měl správně subjekt<sup>11</sup>, kterému byla zpráva doručena, sjednat nápravu v podobě odstranění nalezených zranitelností a vydání dalších příslušných opatření [33].

Vzhledem k důležitosti obsahu reportu představuje jeho tvorba časově náročný proces a v případě velkého množství nalezených zranitelností se může doba potřebná pro vypracování reportu blížit až k délce samotného penetračního testování. Délka tvorby reportu je závislá na mnoha faktorech, jako je například velikost testovaného prostředí, způsob testování nebo samotný cíl testování. Tato práce si klade za cíl zefektivnit penetrační testování mimo jiné tak, aby byla jeho časová náročnost co nejnižší, aniž by došlo ke snížení důkladnosti a přesnosti testování. Proto byla za tímto účelem tato činnost zautomatizována při současném zachováním možnosti úpravy jakékoli části šablony reportu nebo tvorby vlastních šablon.

### Vlastní řešení

Řešením uvedené problematiky výše je generátor reportu, který využívá model uzlů popsaný v podkapitole 2.3.3 a operace stromové struktury z podkapitoly 2.3.4. Vlastní řešení procesu tvorby reportu bylo během implementace rozděleno na dvě dílkové části – šablona reportu a generování reportu.

Šablónu reportu je možné vytvořit a upravovat jako uzel, který je potomkem globálního uzlu celé platformy. Struktura reportu je řešena HTML kódem s podporou kaskádových stylů pro možnost definovat vizuální vlastnosti, jako je např. barva textu, velikost písma, odsazení textu atd. Pomocí vytvořených šablon lze následně v rámci každého projektu vygenerovat report. Jak šablona reportu může vypadat lze vidět na ukázkovém zdrojovém kódu ve výpisu 2.2.

Generování reportu je možné provést v projektu, kdy je report vytvořen jako uzel, který je jeho přímým potomkem. Proces generování představuje komplikaci šablony do výsledného HTML kódu, který již kromě struktury obsahuje i samotná data reportu. Kompilace je řešena pomocí šablonovacího systému Latte<sup>12</sup> a s pomocí jeho syntaxe je struktura šablony naplněna vlastními daty.

<sup>11</sup>Příjemcem reportu bývá nejčastěji objednatel nebo jiný zadavatel penetračního testu.

<sup>12</sup>Tvorba šablony reportu plně podporuje syntaxi šablonovacího systému Latte, která je dostupná z oficiální dokumentace na <https://latte.nette.org/>.

---

Výpis 2.2: Příklad zdrojového kódu šablony reportu

---

```
1 <h1>Závěrečná zpráva</h1>
2 <div>Vystavil: {$user}</div>
3 <div>Společnost: {$company}</div>
4 <div>Zákazník: {$customer}</div>
5 <h2>1. Obsah</h2>
6 <div>{$contents}</div>
7 <h2>2. Manažerské shrnutí</h2>
8 <table>
9   <th>
10    <td>Oblast</td>
11    <td>Označení</td>
12    <td>Hodnocení</td>
13    <td>Poznámky</td>
14  </th>
15  <tr>
16    <td>Autentizace</td>
17    <td>{$auth_score_icon}</td>
18    <td>{$auth_score_text}</td>
19    <td>{$auth_score_note}</td>
20  </tr>
21  <tr>
22    <td>Řízení přístupu</td>
23    <td>{$access_score_icon}</td>
24    <td>{$access_score_text}</td>
25    <td>{$access_score_note}</td>
26  </tr>
27 </table>
28 <h2>3. Shrnutí</h2>
29 <p>{$summary}</p>
```

---

Vygenerovaný report je možné stáhnout jako soubor ve formátu HTML nebo PDF. Další přidanou funkcionalitou je stažení archivu s přílohami všech uzlů, které byly označeny příznakem pro zahrnutí přílohy do reportu. Tvorba archivu je řešena opět za pomoci stromu uzlů, kdy jsou vybrány všechny uzly typu souboru, které byly k zahrnutí do reportu označeny. Každý takový uzel obsahuje informaci o místě uložení souboru, ke kterému se uzel váže. Na začátku této operace je archiv vytvořen a pojmenován podle názvu reportu. Následně jsou postupně do archivu přidány všechny příslušné přílohy. Uzel reportu obsahuje informace pouze z doby jeho vzniku. Pokud by došlo v rámci projektu ke změnám, tak je nutné vygenerovat nový report, čímž vznikne jeho nejaktuálnější verze v podobě nového uzlu. Ukázku jedné stránky vygenerovaného reportu lze vidět na obrázku 2.20.

#### 4.3.2 PŘETĚŽOVÁNÍ



**Očekávané chování:** pokud jeden uživatel (IP adresa) odešle na server během určité doby velké množství požadavků, měl by být požádán o opsání CAPTCHA. Mělo by se tím zamezit například automatické registraci nových uživatelských účtů s pomocí skriptu, odesílání nekonečného množství smyšlených objednávek, odesílání velkého množství dotazů na podporu, výše uvedeným enumeračním útokům apod.

**Zjištěné chování:** Aplikace nijak neomezuje množství requestů, které jsou jí zaslány.

**Doporučení:** Doporučujeme nasadit omezení v počtu požadavků, které mohou přijít z jedné IP adresy za určité časové období. Byl bych opatrný na tvrdou blokaci IP adresy, protože by tak útočník mohl vyvolat DoS na uživatele, pokud by zneužil jeho prohlížeče k odesílání requestů.

**Výskyt:** Nastavení účtu, odesílání objednávek, odesílání dotazů na podporu, registrace nových uživatelů, odesílání náhradního hesla apod. Nejproblematičtější je například opakování zasílání náhradního hesla, při kterém může útočník zcela zahltit e-mailovou schránku uživatele.

**Reference:** V3.13, V7.8 OWASP ASVS 2013

#### 4.3.3 CROSS-SITE REQUEST FORGERY (CSRF)



**Očekávané chování:** Uživatel nemůže bez svého vědomí odeslat pod svou identitou objednávku, dotaz, změnit si nastavení účtu, změnit si heslo, odstranit si účet apod. Při změně hesla a změně kontaktních údajů, na které je zasíláno náhradní heslo, aplikace vyžaduje zadání aktuálního hesla.

**Zjištěné chování:** Není kontrolováno, zda uživatel skutečně zasílá requesty na server úmyslně. Útočník může snadno zneužít důvěry serveru v uživatele a uživateli tak například odstranit nebo ukrást účet. Aplikace při změně hesla nevyžaduje prokázání znalosti aktuálního hesla.

**Doporučení:** Doporučujeme přidat ke všem akcím, které jsou spojeny s identitou uživatele, autorizační CSRF tokeny. V případě funkčních odkazů přidat tyto tokeny jako parametry do URL, u formulářů je pak přidat do skrytých polí. U kritických akcí, jako je změna hesla, nebo změna kontaktních údajů, na které je zasíláno náhradní heslo, vyžadovat od uživatelů zadání aktuálního hesla.

**Výskyt:**

Odhlášení:

<https://www.penterepmail.com/qf/cs/ramjet/performLogout>

<https://dev.penterepmail.com/qf/cs/ramjet/performLogout>

Odstranění účtu:

<https://www.penterepmail.com/qf/cs/ramjet/reg/edit?formId=&eventName%3AremoveUser=removeSubject>

<https://dev.penterepmail.com/qf/cs/ramjet/reg/edit?formId=&eventName%3AremoveUser=removeSubject>

Provedení změn v nastavení účtu:

<https://www.penterepmail.com/qf/cs/ramjet/registration/profileDialog/editor>

<https://dev.penterepmail.com/qf/cs/ramjet/registration/profileDialog/editor>

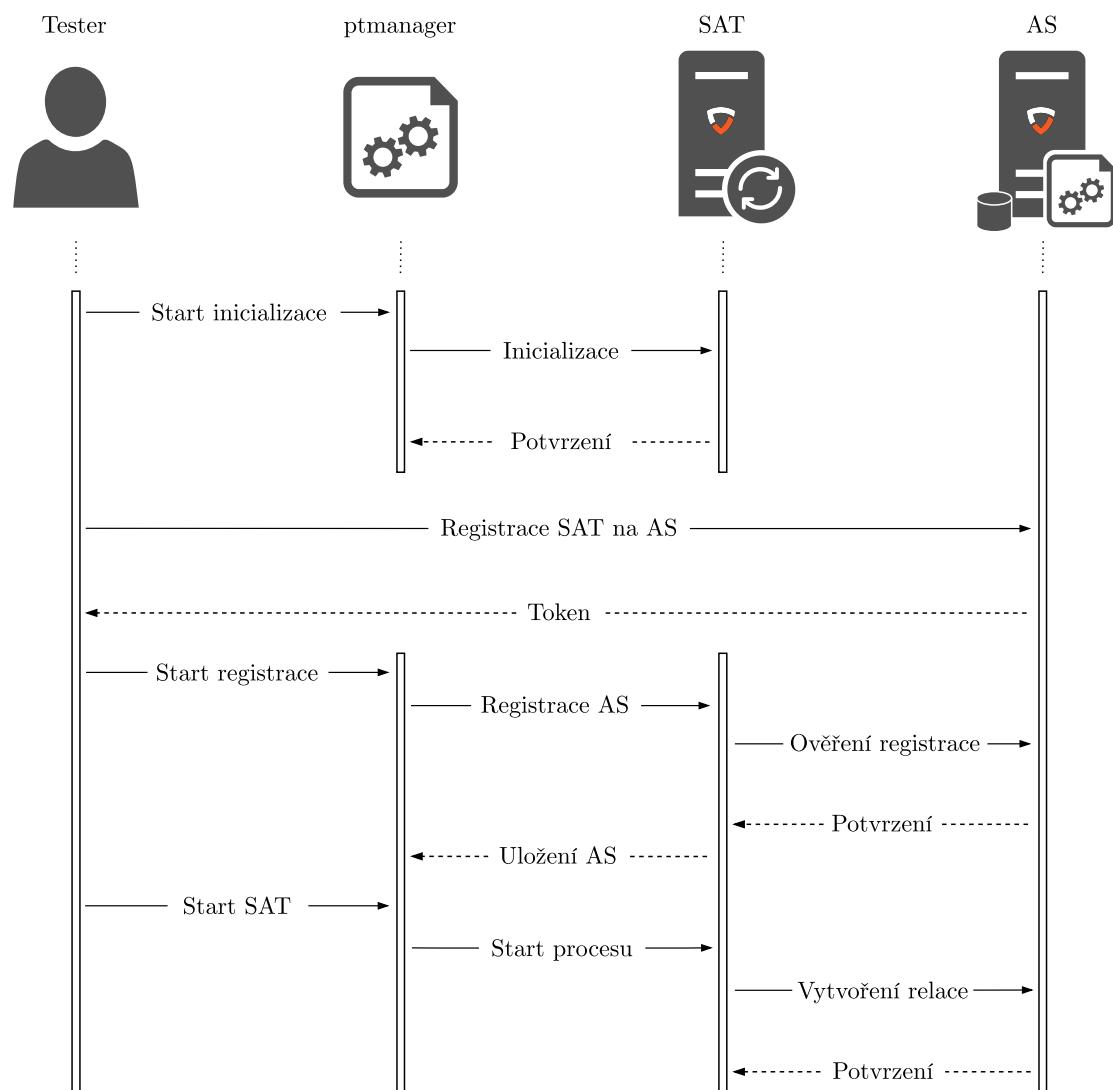
**Reference:** V3.12 OWASP ASVS 2013

Obr. 2.20: Ukázka vygenerovaného reportu

### 2.3.6 Automatizované testování

Jednou z hlavních funkcí webové platformy je možnost spouštět automatizované penetrační testování vůči zadanému cíli. Testy vykonává SAT za pomocí nástroje ptmanager<sup>13</sup> a jejich výsledky zpracovává AS. Zjednodušený proces komunikace mezi jednotlivými částmi je znázorněn na obrázku 2.21.

#### Registrace SAT



Obr. 2.21: Proces aktivace automatizovaného testování, kde SAT představuje server pro automatizované testování a AS je aplikativním serverem, který je přímou součástí webové platformy pro podporu penetračního testování.

<sup>13</sup>Nástroj ptmanager byl pro možnost integrace automatizovaných penetračních nástrojů vyvinut v programovacím jazyce Python a jeho vlastní implementace je mimo rozsah této práce.

Pro správu automatizovaného testování je nutné nejprve nainstalovat nástroj `ptmanager` na SAT, přičemž se nemusí jednat vždy o server, ale jako SAT může být použita i lokální stanice klienta. Aktivace SAT se provede příkazem `ptmanager -i`, který vygeneruje autentizační identifikátor SATID. Následně uživatel zaregistrouje SAT ve webové platformě pomocí SATID, čímž AS vygeneruje autorizační Token a pro server založí uzel se stavem „žádost“. Registrace musí být provedena i na straně SAT a to příkazem `ptmanager -np -T <URL SAT> -a <Token>`. Spuštěním příkazu se odešle na stranu AS registrační žádost, která obsahuje Token a SATID. AS následně zkонтroluje autorizační Token, uloží si k registrovanému uzlu SAT jeho SATID a přepne stav uzlu na „aktivní“. V závěru SAT obdrží od AS název webové platformy, který si uloží do lokálního souboru `config.json`<sup>14</sup>. Seznam registrací a jejich pořadových čísel je možné vypsat příkazem `ptmanager -lp`.

Po aktivaci SAT je nutné nastartovat proces, který se bude starat o zpracování testů ve frontě. Při startování procesu přečte `ptmanager` ze souboru `config.json` hodnotu SATID a URL platformy. Tento proces se spustí příkazem `ptmanager -sp <číslo registrace>`, čímž se odešle požadavek na AS. Po obdržení požadavku provede AS kontrolu porovnáním uloženého SATID s hodnotou, kterou odeslal SAT. Pokud se jeho hodnota shoduje s vlastností některého z uzlů typu SAT, zašle zpět Session ID (identifikátor relace) prostřednictvím HTTP hlavičky Set-Cookie, čímž dojde k aktivaci autorizované relace mezi SAT a AS.

### Zadávání testů do fronty na AS

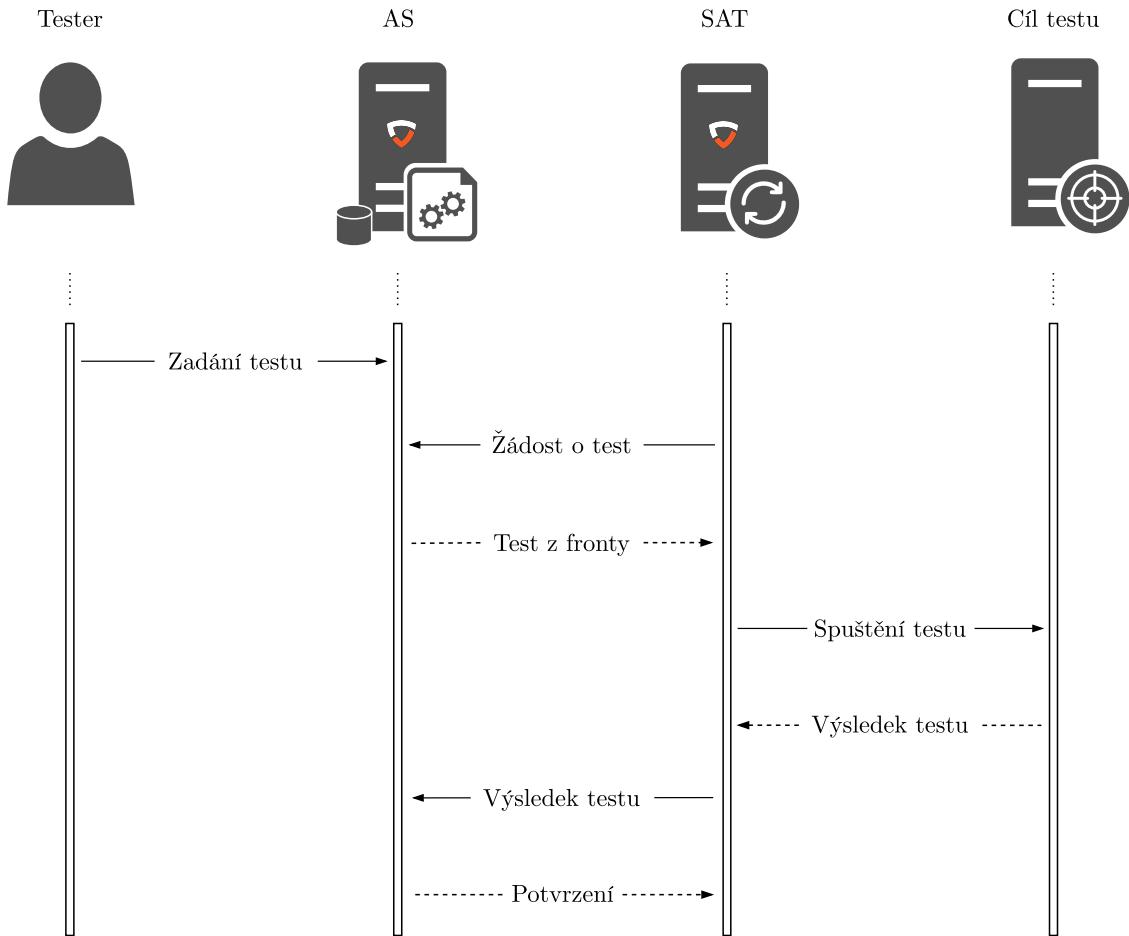
Každý typ automatického testu je definován jako uzel s vlastnostmi, které se liší podle vybraného nástroje. Při přidávání nového testu do fronty se uživateli zobrazí okno s formulářem obsahující konfigurační vstupy s výchozími hodnotami. Formulář obsahuje také speciální pole zobrazující výsledný příkaz, který se odešle na SAT. Při změně jakékoliv hodnoty vlastnosti se příkaz automaticky přizpůsobí s možností jeho manuální úpravy. Jakmile je uzel testu vytvořen, nastaví se stav na stav „čekající“. Pokud se u běžícího uzlu testu změní stav na „zrušený“, odešle se prostřednictvím protokolu WebSocket na stranu SAT notifikace, nebo dojde k automatické aktualizaci při žádosti o další úkol z fronty.

### Vyžádání úkolu z fronty na AS ze strany TS

Proces SAT si v jednotlivých uvolněných vláknech vyžádá od AS další test v řadě čekající ve frontě pro odpovídající SAT. Žádost provede SAT prostřednictvím HTTP požadavku, na který AS odpoví v JSON formátu s testem z fronty, načež AS změní stav uzlu testu na „běžící“. Celý tento proces je znázorněn na obrázku 2.22.

---

<sup>14</sup>Soubor `config.json` obsahuje adresy registrovaných platform a seznam integrovaných nástrojů.



Obr. 2.22: Proces automatizovaného testování, kde SAT provádí automatizované testování vůči zadanému cíli a výsledky vrací na stranu AS pro další zpracování.

### Odeslání výsledků testu ze SAT na AS

Po dokončení testu odešle SAT odpověď ve formátu JSON s výsledkem testu na AS. Nálezy testu se ukládají jako nové uzly určitého typu (např. zranitelnost) nebo jako vlastnosti uzlu. Pokud se výsledek testu vztahuje přímo na konkrétní uzel, tak může dojít k jeho aktualizaci nebo vytvoření nových vlastností (např. identifikovaný operační systém testovaného serveru). Při zakládání nových uzlů, a tedy i potomků již existujícího uzlu (např. webové stránky, které jsou potomkem webové aplikace), se zkонтroluje, zda může typ zakládaného uzlu existovat jako potomek rodičovského uzlu. Stejně tak se kontroluje, zda nová vlastnost může pro daný typ uzlu existovat. Pro identifikaci testu se vždy používá GUID uzlu. Příklad úspěšně provedeného automatizovaného testu lze vidět v JSON formátu ve výpisu 2.3, ve kterém se nachází pokyn k založení nového uzlu webové stránky spolu se dvěma vlastnostmi, kterými je název a typ webové stránky.

### Výpis 2.3: Výsledek automatizovaného testu

---

```
1  {
2      "task_guid": "5aa81391-386f-4b1d-b53d-9907f9ff9c61",
3      "status": "finished",
4      "result": [
5          {
6              "node_type": "webpage",
7              "action": "add_node",
8              "properties": [
9                  {
10                     "name": "Contact",
11                     "type": "HTML"
12                 }
13             ]
14         }
15     ]
16 }
```

---

## Nástroje pro penetrační testování

Pomocí nástroje `ptmanager` lze snadno SAT rozšířit o další automatizované nástroje, o jejichž návratové hodnoty se již stará `ptmanager` a není tak dálé nutné zasahovat do samotných integrovaných nástrojů. Jako první byly integrovány následující nástroje zaměřené převážně na penetrační testování webových aplikací [34]:

- `ptaxfr` sloužící k testu DNS zone transfer,
- `ptcookie` sloužící k podrobné analýze cookie,
- `ptcrossd` pro testování souboru `crossdomain.xml`,
- `ptdos` pro testování odolnosti útoku DoS (Denial of Service),
- `ptgetpage` k extrakci webových stránek obsahu a hlavičky HTTP,
- `pthost` hledání zranitelností přes Host HTTP header,
- `ptiistild` pro identifikaci zranitelnosti IIS Tilde Enumeration,
- `ptinsearcher` určený k extrakci informací z webových zdrojů,
- `ptmethods` pro testování HTTP metod,
- `ptmultiviews` pro ověření zranitelnosti MultiViews,
- `ptprssi` k testu zranitelnosti Path-relative style sheet import,
- `ptsamesite` k testu zranitelnosti Same Site Scripting,
- `ptsearchsploit` pro exploitaci využívající SearchSploit,
- `ptsecurixt` pro hledání souboru `security.txt`,
- `ptvulnsearcher` pro vyhledávání známých zranitelností,
- `ptwebdiscover` pro mapování webových aplikací.

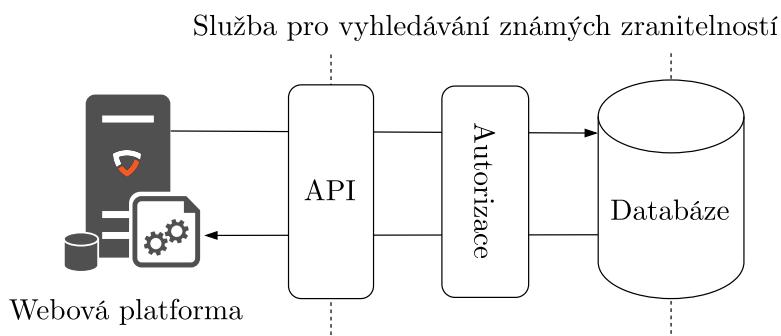
Všechny uvedené nástroje jsou součástí balíčku PenterepTools, který je společně s výstupem této práce hlavním výsledkem projektu výzkumu a vývoje řešeného se státní podporou Technologické agentury České republiky. Jednotlivé nástroje je také možné vhodně kombinovat pro dosažení optimálních výsledků testování.

## 2.4 Nástroj pro vyhledávání známých zranitelností

Pro zvýšení podpory penetračního testování byl v rámci této práce vyvinut nástroj pro vyhledávání známých zranitelností. Nástroj cílí na zranitelné a neaktualizované části testovaného cíle. Tato zranitelnost bývá často důvodem pro eskalaci dalších útoků a dlouhodobě se umisťuje v žebříčku OWASP Top 10, ve kterém společnost OWASP Foundation každoročně zveřejňuje 10 nejčastějších zranitelností webových aplikací. V roce 2021 se tato zranitelnost pod označením „A06:2021-Vulnerable and Outdated Component“ umístila v žebříčku na 6. místě [35].

Nástroj využívá veřejně přístupná data z databází známých zranitelností, kdy se na základě dostupných informací získaných z penetračního testování dotazuje na existující zranitelnosti CVE (Common Vulnerabilities and Exposures). Pro zajištění vyšší kontroly nad obsluhou požadavků byla vytvořena vlastní relační databáze, která je pravidelně plněna daty z veřejně dostupných zdrojů. Konkrétně jsou data čerpána z databází CVE® Program, CVE Details a NVD (National Vulnerability Database). Pro možnost komunikace s vytvořenou databází známých zranitelností bylo implementováno REST API, na které se webová platforma dotazuje.

Proces integrace a automatizace nástroje je stejný jako v kapitole 2.3.6. Během procesu penetračního testování lze zranitelnosti vyhledávat podle dynamických dat testovatelných uzelů. V moment spuštění nástroje jsou nástroji na vstupu předány dostupné informace o testovaném uzlu, jako je například název software, jeho verze a další odhalené technologie. Nástroj tyto informace porovná s dostupnými daty známých zranitelností, a pokud je nalezena shoda, předá webové platformě pokyn k vytvoření nového uzlu zranitelnosti spolu se získanými informacemi. Celý tento proces je znázorněn na obrázku 2.23.



Obr. 2.23: Vyhledávání známých zranitelností

Pro možnost širšího využití byla vyvinuta i samostatná verze nástroje s názvem ptvulnsearcher pod otevřenou licencí GPLv3.<sup>15</sup>

<sup>15</sup>Nástroj je volně dostupný ke stažení na <https://pypi.org/project/ptvulnsearcher/>.

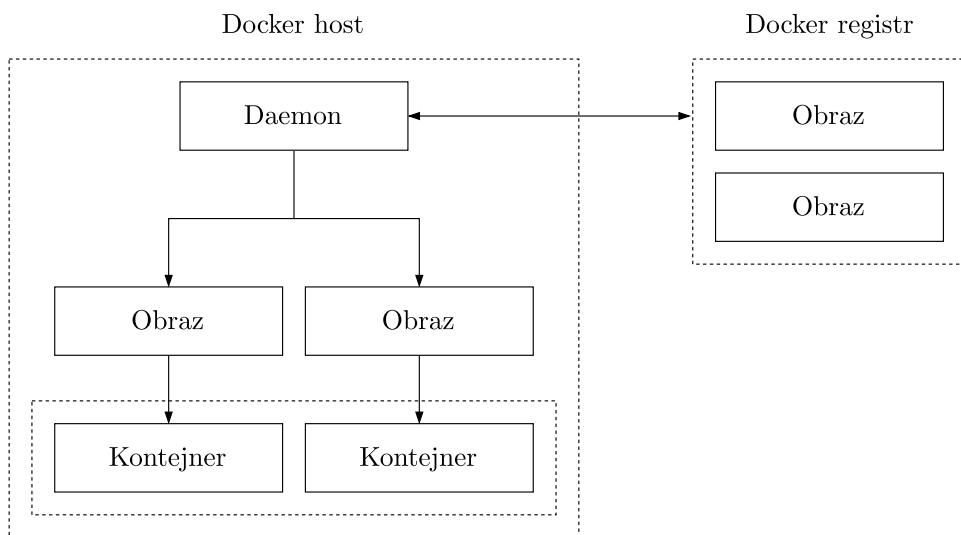
# 3 Testování v produkčním prostředí

## 3.1 Nasazení platformy

Pro testování v reálných podmínkách byla platforma nasazena na dedikovaný VPS s operačním systémem Ubuntu. Aplikační server a relační databáze byly nasazeny jako Docker kontejnery, díky čemuž lze celý proces snadno automatizovat bez další závislosti na hostované stanici. Nutná je pouze přítomnost Dockeru. Kontejnerizací celé platformy je možné také zajistit distribuci změn a oprav vůči všem nasazeným platformám v produkčním prostředí. Proces nasazení byl automatizován, a pokud v některé fázi selže, tak je nutné provést patřičnou nápravu a proces zopakovat. Celý tento proces probíhá v následujících krocích:

1. host stáhne obrazy platformy z Docker registru<sup>1</sup>,
2. podle instrukcí obrazů se vytvoří kontejnery,
3. spustí se aplikační server a relační databáze,
4. zahájí se migrace databáze (tvorba struktury),
5. databáze se naplní inicializačními daty,
6. postupně se provedou automatické testy,
7. platforma je připravena k použití.

Jakým způsobem komunikuje Docker host s Docker registrum při produkčním nasazení je ve zjednodušené formě znázorněno na obrázku 3.1.

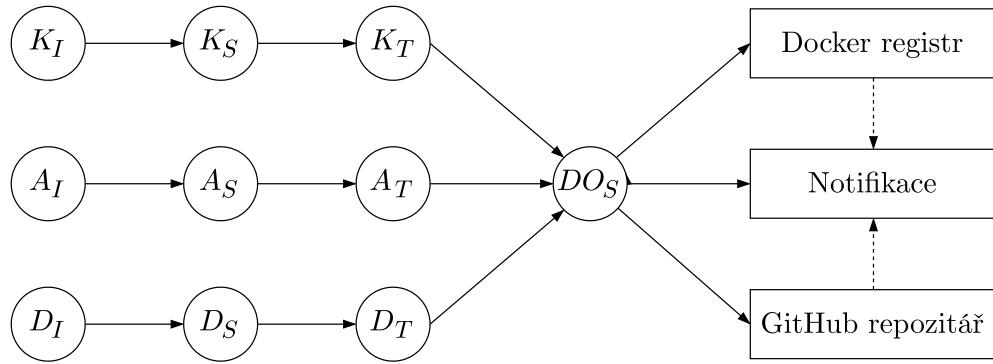


Obr. 3.1: Komunikace mezi Docker hostem a Docker registrum, ve které Daemon obsluhuje vnější požadavky a spravuje kontejnery podle instrukcí obrazů.

<sup>1</sup>Docker registr je bezstavová vysoce škálovatelná aplikace na straně serveru, která ukládá a umožňuje efektivně distribuovat Docker obrazy (image) [36].

## 3.2 Testování platformy

Před každou změnou produkční verze je automaticky spuštěna sekvence procesů na platformě GitHub. Tyto automatické procesy zahrnují kromě testů také instalaci balíčků a sestavení Docker obrazu. Po každém vytvoření Pull Requestu<sup>2</sup> v repozitáři projektu jsou nejprve spuštěny procesy GitHub Actions, a proběhnou-li úspěšně, tak se změny zahrnou do produkční větve a sestavený Docker obraz se nahraje na Docker registr, ze kterého si následně změny stáhnou všechny nasazené platformy v produkčním prostředí. Tento proces je znázorněn na obrázku 3.2.



Obr. 3.2: Sekvence procesů na platformě GitHub, kde jsou skupiny procesů značeny jako  $K$  (klientská část),  $A$  (aplikační část),  $D$  (datová část) a  $DO$  (Docker obraz). Procesy skupin jsou postupně  $I$  (instalace),  $S$  (sestavení) a  $T$  (testy). Pokud každý proces proběhne úspěšně, tak je aktualizován Docker registr a produkční (hlavní) větev v repozitáři GitHub. Následně je notifikován správce projektu.

V průběhu realizace této práce byla platforma pravidelně testována ve vývojovém a produkčním prostředí, přičemž při jejím testování bylo uloženo přibližně 5 000 000 záznamů, aniž by to mělo větší vliv na celkový výkon. Cílením na modularitu byly minimalizovány duplicity ve zdrojovém kódu s možností opakování použití jeho dílčích částí. Statická databáze určující hlavní strukturu celé platformy je tvořena 14 000 záznamy, velikostně čítá zdrojový kód přibližně 15 000 řádků a pro obsluhu všech požadavků bylo implementováno 433 funkcí.

Vzhledem ke komplexnosti výsledného řešení byly v průběhu vývoje prováděny také manuální testy. Konkrétně byla průběžně testována dostupnost API endpointů, obsluhující funkce REST API, WebSocket komunikace, správce lokálního úložiště a uživatelské rozhraní SPA. Kromě testů, které vyžadují uživatelskou interakci, byla většina těchto testů zahrnuta do automatizovaných procesů z obrázku 3.2.

---

<sup>2</sup>Pull Request představuje žádost o změnu v repozitáři projektu. Provádí se v etapách při aplikaci změn z jedné větve repozitáře do jiné (např. z vývojové do produkční).

## 4 Porovnání současných řešení

Na trhu existuje celá řada různých nástrojů pro penetrační testování. Ty lze dále rozlišovat podle oblasti, na kterou cílí (např. počítačová síť, webová aplikace, mobilní aplikace apod.). Naopak méně početnou skupinou jsou řešení pro samotnou správu nebo realizaci penetračních testů využívající automatizované nástroje. Tato práce do obou skupin patří, a proto bude porovnána s těmi nejrelevantnějšími. Porovnány mezi sebou budou následující řešení:

- **Burp Suite** – softwarový nástroj pro penetrační testování webových aplikací,
- **w3af** – software pro skenování a exploitaci nalezených zranitelností,
- **Acunetix** – webová aplikace pro vyhledávání a testování zranitelností,
- **AF (AttackForge)** – webová platforma pro správu penetračního testování,
- **Penterep<sup>1</sup>** – webová platforma pro podporu penetračního testování, která je výsledným řešením této bakalářské práce.

Přestože mají společný cíl, tak se svými vlastnostmi a funkcemi liší, kvůli čemuž obsahují nedostatky, na základě kterých může být jejich použití na specifické testy omezené. Nástroje budou porovnány v následujících vlastnostech:

- **podpora automatického testování** – testování automatizovanými skripty,
- **podpora manuálního testování** – návody a podpůrné mechanismy,
- **testování specifických oblastí** – rozlišení podle testovaného prostředí,
- **forenzní analýza** – zaměření na digitální forenzní analýzu,
- **plánování testů** – testy je možné plánovat a řadit do fronty,
- **konfigurace testů** – automatizované testy lze konfigurovat,
- **vizualizace prostředí** – vizualizace testovaného prostředí,
- **vlastní schéma testování** – možnost definovat vlastní metodologii testování,
- **dokumentace/nápověda** – dostupná dokumentace či nápověda k ovládání,
- **evidence nálezů** – výsledky lze kategorizovat, filtrovat a uložit,
- **ukazatel závažnosti** – vizualizace zranitelnosti podle závažnosti,
- **kalkulátor závažnosti** – možnost hodnocení závažnosti zranitelností,
- **grafické uživatelské rozhraní** – interakce pomocí uživatelského rozhraní,
- **podpora mobilního zařízení** – přizpůsobení mobilnímu zařízení,
- **import/export výsledků** – výsledky lze importovat a exportovat,
- **správa projektů** – lze zakládat a upravovat projekty pro testování,
- **tvorba reportu** – možnost vytvářet a exportovat reporty,
- **podpora více uživatelů** – na testování může pracovat více uživatelů,
- **verze licence zdarma** – existuje neplacená verze licence,
- **integrace vlastních nástrojů** – možnost integrovat nástroje pro testování,
- **integrace třetích stran** – možnost propojení s externími systémy.

---

<sup>1</sup>Název platformy „Penterep“ vznikl pro pojmenování platformy v produkčním prostředí.

Vlastnosti byly vybrány na základě fází penetračního testování z kapitoly 1.1.1 a požadavků z přílohy A. Výsledky srovnání jsou uvedeny v tabulce 4.1.

Tab. 4.1: Porovnání relevantních nástrojů s výsledným řešením

| Vlastnost/funkcionalita       | Burp Suite | Acunetix | w3af | AF | Penterep |
|-------------------------------|------------|----------|------|----|----------|
| Podpora automat. testování    | ✓          | ✓        | ✓    | ✗  | ✓        |
| Podpora manuál. testování     | ✗          | ✗        | ✗    | ✓  | ✓        |
| Testování webových aplikací   | ✓          | ✓        | ✓    | ✗  | ✓        |
| Testování mobilních aplikací  | ✗          | ✗        | ✗    | ✗  | ✓!       |
| Testování síť. infrastruktury | ✗          | ✓        | ✗    | ✗  | ✓!       |
| Digitální forenzní analýza    | ✗          | ✗        | ✗    | ✗  | ✓!       |
| Plánování testů               | ✓          | ✓        | ✗    | ✓  | ✓        |
| Konfigurace testů             | ✓          | ✗        | ✓    | ✗  | ✓        |
| Vizualizace prostředí         | ✗          | ✗        | ✗    | ✗  | ✓        |
| Vlastní schéma testování      | ✗          | ✗        | ✗    | ✗  | ✓        |
| Dokumentace/nápověda          | ✓          | ✓        | ✓    | ✓  | ✓        |
| Evidence nálezů               | ✓          | ✓        | ✗    | ✓  | ✓        |
| Ukazatel závažnosti           | ✓          | ✓        | ✗    | ✓  | ✓        |
| Kalkulátor závažnosti         | ✗          | ✗        | ✗    | ✗  | ✓        |
| Grafické uživatelské rozhraní | ✓          | ✓        | ✓    | ✓  | ✓        |
| Podpora mobilního zařízení    | ✗          | ✗        | ✗    | ✓  | ✓        |
| Import výsledků               | ✓          | ✗        | ✗    | ✓  | ✓        |
| Export výsledků               | ✓          | ✓        | ✓    | ✓  | ✓        |
| Správa projektů               | ✗          | ✗        | ✗    | ✓  | ✓        |
| Tvorba reportu                | ✓          | ✓        | ✗    | ✓  | ✓        |
| Podpora více uživatelů        | ✗          | ✓        | ✗    | ✓  | ✓        |
| Verze licence zdarma          | ✓          | ✗        | ✓    | ✓  | ✓        |
| Integrace vlastních nástrojů  | ✓          | ✓        | ✓    | ✗  | ✓        |
| Integrace třetích stran       | ✓          | ✓        | ✓    | ✓  | ✓!       |

Pozn.: ✓ – implementováno, ✗ – neimplementováno, ✓! – plánováno v plné verzi.

Z výsledku porovnání je patrné, že webová platforma Penterep řeší problematiku penetračního testování s cílem maximalizovat možnosti testování, podpořit manuální testování a zvýšit celkové pokrytí testované oblasti. Tím se výrazně liší od ostatních nástrojů, které se specializují pouze na konkrétní podoblasti informačních a komunikačních technologií. Tato práce se k problematice staví s novým přístupem, kterým řeší nedostatky ostatních řešení a to s hlavním cílem poskytnout kompletní podporu pro penetrační testování.

Kromě platformy Penterep a AttackForge, jsou ostatní porovnávané nástroje omezeny pouze na automatizované testování. Takovým nástrojem je w3af, který umožňuje více konfigurovat automatizované testy, ale postrádá možnost spravovat projekty a jeho rozsah užití pro penetrační testování je proto značně omezený [37]. Naopak funkce pro správu a podporu manuálního penetračního testování nabízí platforma AttackForge, která však postrádá možnosti automatizovaných testů [38]. Webová aplikace Acunetix staví testování primárně na svých skenerech, přičemž konfiguraci a náhled celého průběhu automatického testování umožňuje pouze v malé míře nebo vůbec. Z důvodu tohoto omezení se Acunetix používá více jako dohledový systém pro sledování a audit bezpečnosti webových aplikací nežli jako nástroj pro podporu manuálního penetračního testování [39]. Nástroj Burp Suite naopak nabízí skrze své moduly velké možnosti automatizovaného testování včetně jeho konfigurace a integrace komunitních rozšíření, ale jeho použití je omezeno pouze na penetrační testování webových aplikací [40]. Všechny tyto nástroje je možné do platformy Penterep integrovat a využít je stejně jako nástroje pro automatizované penetrační testování uvedené v kapitole 2.3.6.

Podotknout je zde třeba, že se jedná o dlouho vyvíjené nástroje. Burp Suite vznikl již v roce 2003 a aplikace Acunetix se na trhu nachází od roku 2005, přičemž v obou případech se jedná o komerční řešení velkých společností. Nástroj w3af je vyvíjen od roku 2013 jako open source a na jeho vývoji se tak podílí komunita. Novějším oproti ostatním je platforma AttackForge se vstupem na trh v roce 2018 a cílením na malé týmy i velké společnosti. Samotný vývoj webové platformy Penterep začal v květnu roku 2021 a první PoC (Proof of Concept) byl dokončen v červnu téhož roku. Díky navrženému modelu, cílení na modularitu a použité metodice se podařilo v krátkém čase vytvořit rozsáhlé prostředí pro komplexní realizaci penetračního testování.

# Závěr

Na základě stanovených cílů bakalářské práce byl pro vlastní vývoj webové platformy navržen vysoce škálovatelný model, který se ukázal jako plně vyhovující pro splnění požadavků kladených na platformu a v průběhu vývoje došlo k ověření jeho efektivity a možnostem širšího využití. Z toho důvodu byl navržený model nejprve popsán v obecné rovině a následně do něj byla platforma zasazena. Díky tomu bylo možné v krátkém čase vytvořit kompletní schéma pro penetrační testování webových aplikací, které jsou připraveny k použití v produkčním prostředí.

Při realizaci praktické části této práce byla navržena a následně implementována klientská část platformy – modulární webová aplikace, která se skládá z komponent a lokálního rozhraní. Podle standardu CVSS byl vyvinut kalkulátor pro hodnocení závažnosti zranitelností, který byl vložen jako rozšiřující modul do SPA. Dále byla vyvinuta aplikační část – REST API, obsluhující funkce, proces autentizace, strom uzlů, generátor reportu, rozhraní pro automatizované testování a spojení s datovou částí – relační databází. Výstupem bakalářské práce je také nástroj pro vyhledávání známých zranitelností na základě vstupních parametrů. Tento nástroj byl integrován do aplikační části jako rozšiřující služba a dodatečně byla vyvinuta samostatná verze nástroje pro podporu penetračního testování. Webová platforma byla za využití Docker kontejnerizace nasazena na VPS pro testování v produkčním prostředí.

Od začátku vývoje webové platformy byl kladen důraz na optimalizaci a údržbu zdrojového kódu, díky čemuž se operace na klientské straně, aplikační straně a nad databází provádějí rychle a efektivně využívají implementované pomocné funkce. Veškeré dílčí změny byly během vývoje verzovány v repozitárii projektu na platformě GitHub. V průběhu realizace této práce byla platforma pravidelně testována ve vývojovém a produkčním prostředí. Cílením na modularitu byly minimalizovány duplicity ve zdrojovém kódu s možností opakovaného použití jeho dílčích částí. S tímto vším byly veškeré stanovené cíle bakalářské práce splněny.

## Vedlejší výsledky

Dosažené výsledky bakalářské práce byly prezentovány na studentské konferenci STUDENT EEICT 2022, do jejíhož sborníku byla zařazena publikace pojednávající o webové platformě pro komplexní penetrační testování s názvem „Web Platform for Comprehensive Penetration Testing“. V soutěži studentské tvůrčí činnosti obsadila práce 1. místo v kategorii „B7 – Communication Systems and Network Security“, čímž byla zařazena do mezinárodní citační databáze Web of Science. Prezentovaná práce byla také v rámci konference oceněna zvláštními cenami sponzorů ŠKODA AUTO, NXP Semiconductors Czech Republic, E.ON a CROSS Zlín.

Dalším výsledkem, který vychází z této práce, je konferenční článek s názvem „PTWEBDISCOVER: Nástroj pro efektivní mapování webových aplikací během penetračního testování“, který byl přijat a prezentován na konferenci EurOpen.CZ. Součástí této prezentace byla také ukázka webové platformy.

Pro účely veřejné prezentace byly nad rámec této práce vyvinuty webové stránky, které jsou dostupné na adrese <https://www.penterep.io> a lze na nich dohledat základní informace o platformě, časovou osu a reference aplikovaného výzkumu, jehož je tato bakalářská práce výstupem.

## **Vlastní přínos práce**

Webová platforma byla v poslední kapitole srovnána s dalšími relevantními nástroji, které se používají k penetračnímu testování. Na základě výsledku srovnání a všech dokončených dílčích částí webové platformy lze výstup této práce považovat jako přínosný pro řešení problematiky komplexního penetračního testování. Mezi hlavní výhody výsledného řešení patří podpora manuálního a automatického testování, rozsáhlá správa projektů, možnost týmové spolupráce, vizualizace testovaného prostředí, kontrolní seznamy a automatizace tvorby reportu. Výsledná práce zvyšuje efekt penetračního testování do takové míry, že časová náročnost, složitost a práce nutná k úspěšnému dokončení celého testu je značně nižší než při použití dostupných nástrojů, které se běžně používají pro správu a realizaci penetračního testování.

Díky navrženému vysoce škálovatelnému modelu a metodice použité při vývoji lze webovou platformu podle potřeby dále rozšiřovat, aniž by bylo nutné zasahovat do již implementovaných částí. Takové rozšíření zároveň není omezeno na použité technologie, protože ho lze v rámci modelu integrovat jako autonomní službu, která bude prostřednictvím API komunikovat s ostatními službami nebo s jinou externí stranou. Další značnou výhodou současného řešení je možnost cílení na libovolnou oblast testování a integrovat do platformy další automatizované nástroje. V takovém případě lze platformu rozšířit o zcela odlišné oblasti od penetračního testování, jako jsou např. kontrolní procesy, audit GDPR nebo postupy při vyšetřování trestních činů a dalších souvisejících činností orgánů činných v trestním řízení.

## **Další využití a možné rozšíření práce**

Výsledné řešení této práce bude navíc využíváno pro edukativní účely, konkrétně v laboratorních úlohách pro předměty Bezpečnost ICT2 a Bezpečnost ICT3, které se vyučují v oboru Informační bezpečnost na FEKT VUT v Brně. Obdobně bude platforma použita pro virtualizované kybernetické scénáře zaměřené na penetrační testování v Kybernetické aréně.

Dalším uplatněním této práce v oblasti vzdělávání kybernetické bezpečnosti je využití platformy pro školící kurzy zaměřené na zranitelnost a testování bezpečnosti webových aplikací v počítačové škole GOPAS, která je největším poskytovatelem školení v oblasti informačních technologií na českém a slovenském trhu.

Budoucí rozšíření platformy v sobě zahrnují nové modulární komponenty, funkce a další vylepšení uživatelského rozhraní platformy. Jmenovitě se jedná o interaktivní zobrazení testovaného prostředí, konfigurátor slovníkového útoku, diskuze mezi členy týmu, návrhář útočných scénářů a integrace nástrojů třetích stran (Jira, Trello, GitHub, GitLab, Microsoft Teams aj.). Dalším plánovaným rozšířením jsou moduly pro testování síťové infrastruktury, digitální forenzní analýzu a statickou analýzu zdrojového kódu. Současně bude s těmito moduly realizován vývoj automatizovaných nástrojů pro testování příslušné cílené oblasti, které budou do webové platformy následně integrovány. Veškerá tato rozšíření bude možné realizovat v rámci řešení dalších projektů aplikovaného výzkumu nebo vypsáním bakalářských a diplomových prací na Vysokém učením technickém v Brně.

# Literatura

- [1] SiteLock Cybersecurity And Website Report. *SiteLock* [online]. [cit. 2021-10-03]. Dostupné z: <<https://www.sitelock.com/resources/security-report/>>
- [2] BACUDIO, A. G.; YUAN, X. A.; CHU, B. T. B.; JONES, M. An Overview of Penetration Testing. In *International Journal of Network Security & Its Applications*. [online]. 2011, **3**(6), 19–38 [cit. 2021-10-10]. ISSN 0974-9330.
- [3] SHAH, S.; MEHTRE, B. M.; CHU, B. T. B.; JONES, M. An overview of vulnerability assessment and penetration testing techniques. In *Journal of Computer Virology and Hacking Techniques*. [online]. 2015, **11**(1), 27–49 [cit. 2021-10-10]. ISSN 2263-8733.
- [4] ZAKARIA, M. N.; PHIN, P. A.; MOHMAD, N.; ISMAIL, S. A.; KAMA, M. N.; YUSOP, O. A Review of Standardization for Penetration Testing Reports and Documents. In *2019 6th International Conference on Research and Innovation in Information Systems (ICRIIS)*. 2019, s. 1–5. ISBN 978-1-7281-6726-8.
- [5] VATS, P.; MANDOT, M.; GOSAIN., A. A Comprehensive Literature Review of Penetration Testing & Its Applications. In *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 2020, s. 674–680. ISBN 978-1-7281-7016-9.
- [6] SCARFONE, K.; SOUPPAYA, M.; CODY, A.; OREBAUGH, A. Technical Guide to Information Security Testing and Assessment. *National Institute of Standards and Technology*. [online]. 2008, s. 2–5. [cit. 2021-10-30]. Dostupné z: <<https://doi.org/10.6028/NIST.SP.800-115>>
- [7] NAGPURE, S; KURKURE, S. Vulnerability assessment and penetration testing of Web application. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. 2017, s. 1–6. ISBN 978-1-5386-4008-1.
- [8] Common Vulnerability Scoring System version 3.1: Specification Document. *Forum of Incident Response and Security Teams* [online]. [cit. 2022-05-07]. Dostupné z: <<https://www.first.org/cvss/specification-document>>
- [9] WILLIAMS, J. OWASP Risk Rating Methodology. *OWASP Foundation* [online]. [cit. 2022-05-07]. Dostupné z: <[https://owasp.org/www-community/OWASP\\_Risk\\_Rating\\_Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)>

- [10] Common Vulnerability Scoring System v3.1: Calculator Use & Design. *Forum of Incident Response and Security Teams* [online]. [cit. 2022-05-07]. Dostupné z: <<https://www.first.org/cvss/v3.1/use-design>>
- [11] Stack Overflow Developer Survey 2021. *Stack Overflow* [online]. [cit. 2021-10-19]. Dostupné z: <<https://insights.stackoverflow.com/survey/2021>>
- [12] PYPL PopularitY of Programming Language. *PYPL Index* [online]. [cit. 2021-10-19]. Dostupné z: <<https://pypl.github.io/PYPL.html>>
- [13] The State of Developer Ecosystem 2021. *JetBrains* [online]. [cit. 2021-10-19]. Dostupné z: <<https://www.jetbrains.com/lp/devcosystem-2021/>>
- [14] JavaScript. *MDN Web Docs* [online]. [cit. 2021-10-20]. Dostupné z: <<https://developer.mozilla.org/en-US/docs/Web/JavaScript>>
- [15] ECMAScript® 2022 Language Specification. *Ecma TC39* [online]. [cit. 2021-10-20]. Dostupné z: <<https://tc39.es/ecma262/>>
- [16] Front-end Frameworks. *State of JS 2020* [online]. [cit. 2022-04-20]. Dostupné z: <<https://2021.stateofjs.com/en-US/libraries/front-end-frameworks>>
- [17] Introduction. *Vue.js* [online]. [cit. 2021-10-24]. Dostupné z: <<https://v3.vuejs.org/guide/introduction.html>>
- [18] GAO, Z.; BIRD, C; BARR, E. T. To type or not to type: quantifying detectable bugs in JavaScript. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, s. 758–769. ISBN 978-1-5386-3868-2.
- [19] TypeScript Documentation. *TypeScript* [online]. [cit. 2021-10-24]. Dostupné z: <<https://www.typescriptlang.org/docs/>>
- [20] RAD, B. B.; BHATTI, H. J.; AHMADI, M. An introduction to docker and analysis of its performance. In *International Journal of Computer Science and Network Security (IJCSNS)*. 2017, **17**(3), 228–235. ISSN 1738-7906.
- [21] PHP Manual. *php* [online]. [cit. 2021-10-30]. Dostupné z: <<https://www.php.net/manual/>>
- [22] LAAIRI, M.; BENMOUSSA, K; KHOULJI, S.; KERKEB, M L. A Comparative study of PHP frameworks performance. In *12th International Conference Interdisciplinarity in Engineering*. 2019, s. 864–871. ISSN 2351-9789.

- [23] The PHP Framework for Web Artisans. *Laravel* [online]. [cit. 2021-10-30]. Dostupné z: <<https://laravel.com/>>
- [24] POP, D. P.; ALTAR, A. Designing an MVC model for rapid web application development. *Procedia Engineering*. 2014, s. 1172–1179. ISSN 1877-7058.
- [25] SINGH, S.; IYER, J. Comparative Study of MVC (Model View Controller) Architecture with respect to Struts Framework and PHP. In *International Journal of Computer Science Engineering (IJCSE)*. 2016, s. 142–143. ISSN 2319-7323.
- [26] FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. [online] Irvine, 2000. Disertace. University of California, Irvine. Dostupné z: <<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>
- [27] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database system concepts*. Seventh Edition. New York, NY: McGraw-Hill, 2020, s. 12–15. ISBN 978-0-0780-2215-9.
- [28] DENIS, M.; ZENA, C.; HAYAJNEH, T. Penetration testing: Concepts, attack methods, and defense strategies. In *IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. 2016, pp. 1–6. ISBN 978-1-4673-8490-2.
- [29] SILL, A. The Design and Architecture of Microservices. *IEEE Cloud Computing*. 2016, **3**(5), 76–80. ISSN 2325-6095.
- [30] Flux. *Facebook Open Source* [online]. [cit. 2021-11-28]. Dostupné z: <<https://facebook.github.io/flux/>>
- [31] API Design. *Swagger* [online]. [cit. 2021-11-30]. Dostupné z: <<https://swagger.io/solutions/api-design/>>
- [32] GYORODI, C.; MOLDOVAN-DUSE, R.; GYORODI, R.; PECHERLE, G. Improve Query Performance On Hierarchical Data. Adjacency List Model Vs. Nested Set Model. In *International Journal of Advanced Computer Science and Applications (IJACSA)*. 2016, **7**(4), 272-278. ISSN 2156-5570.
- [33] ALGHAMDI, A. A. Effective Penetration Testing Report Writing. In *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 2021, pp. 1–5. ISBN 978-1-6654-1262-9.
- [34] Penterep. *PyPi* [online]. [cit. 2022-04-08]. Dostupné z: <<https://pypi.org/user/penterep/>>

- [35] OWASP Top 10:2021. *OWASP Foundation* [online]. [cit. 2022-04-10]. Dostupné z: <<https://owasp.org/Top10/>>
- [36] Docker Registry. *Docker Documentation* [online]. [cit. 2022-05-08]. Dostupné z: <<https://docs.docker.com/registry/>>
- [37] Understanding the basics. *w3af* [online]. [cit. 2021-10-12]. Dostupné z: <<https://w3af.org/howtos/understanding-the-basics>>
- [38] Penetration Testing Workflow Management & Collaboration Tools. *AttackForge* [online]. [cit. 2021-10-12]. Dostupné z: <<https://attackforge.com/>>
- [39] Acunetix Vulnerability Scanner. *Acunetix* [online]. [cit. 2021-10-12]. Dostupné z: <<https://www.acunetix.com/vulnerability-scanner/>>
- [40] Burp Suite Enterprise Edition plan. *PortSwigger* [online]. [cit. 2021-10-12]. Dostupné z: <<https://portswigger.net/burp/enterprise/pricing>>

# Seznam symbolů a zkrátek

|              |                                               |
|--------------|-----------------------------------------------|
| <b>AC</b>    | Attack Complexity                             |
| <b>AF</b>    | AttackForge                                   |
| <b>AR</b>    | Availability Requirement                      |
| <b>API</b>   | Application Programming Interface             |
| <b>AS</b>    | Aplikační server                              |
| <b>AV</b>    | Attack Vector                                 |
| <b>BS</b>    | Base Score                                    |
| <b>CORS</b>  | Cross-Origin Resource Sharing                 |
| <b>CR</b>    | Confidentiality Requirement                   |
| <b>CRUD</b>  | Create, Read, Update, Delete                  |
| <b>CSR</b>   | Client-side rendering                         |
| <b>CSS</b>   | Cascading Style Sheets                        |
| <b>CVE</b>   | Common Vulnerabilities and Exposures          |
| <b>CVSS</b>  | Common Vulnerability Scoring System           |
| <b>DCL</b>   | Data Control Language                         |
| <b>DDL</b>   | Data Definition Language                      |
| <b>DML</b>   | Data Manipulation Language                    |
| <b>DOM</b>   | Document Object Model                         |
| <b>DOS</b>   | Denial of Service                             |
| <b>ECM</b>   | Exploit Code Maturity                         |
| <b>ES</b>    | Environmental Score                           |
| <b>ES6+</b>  | ECMAScript 6 a vyšší                          |
| <b>FIRST</b> | Forum of Incident Response and Security Teams |
| <b>GUID</b>  | Globally Unique Identifier                    |

|               |                                                   |
|---------------|---------------------------------------------------|
| <b>HTML</b>   | Hypertext Markup Language                         |
| <b>HTTP</b>   | Hypertext Transfer Protocol                       |
| <b>IR</b>     | Integrity Requirement                             |
| <b>ISS</b>    | Impact Sub-Score                                  |
| <b>ISSAF</b>  | Information Systems Security Assessment Framework |
| <b>JSON</b>   | JavaScript Object Notation                        |
| <b>ME</b>     | Modified Exploitability                           |
| <b>MI</b>     | Modified Impact                                   |
| <b>MISS</b>   | Modified Impact Sub-Score                         |
| <b>MVC</b>    | Model-View-Controller                             |
| <b>MVVM</b>   | Model-View-ViewModel                              |
| <b>NPM</b>    | Node Package Manager                              |
| <b>NVD</b>    | National Vulnerability Database                   |
| <b>OSSTMM</b> | Open Source Security Testing Methodology Manual   |
| <b>OWASP</b>  | Open Web Application Security Project             |
| <b>PoC</b>    | Proof of Concept                                  |
| <b>PR</b>     | Privileges Required                               |
| <b>PTES</b>   | Penetration Testing Execution Standard            |
| <b>RC</b>     | Report Confidence                                 |
| <b>REST</b>   | Representational State Transfer                   |
| <b>RL</b>     | Remediation Level                                 |
| <b>RRM</b>    | Risk Rating Methodology                           |
| <b>SAT</b>    | Server pro automatizované testování               |
| <b>SOAP</b>   | Simple Object Access Protocol                     |
| <b>SPA</b>    | Single-page application                           |

|            |                                |
|------------|--------------------------------|
| <b>SQL</b> | Structured Query Language      |
| <b>TCL</b> | Transaction Control Language   |
| <b>TCS</b> | Transaction Control Statements |
| <b>TS</b>  | Temporal Score                 |
| <b>UI</b>  | User Interaction               |
| <b>UI</b>  | User Interface                 |
| <b>UMS</b> | User Management Server         |
| <b>URL</b> | Uniform Resource Locator       |
| <b>VPS</b> | Virtual Private Server         |

# **Seznam příloh**

|                                                         |     |
|---------------------------------------------------------|-----|
| A Požadavky webové platformy                            | 80  |
| B Metriky a hodnoty CVSS                                | 81  |
| C Velikost výsledného řešení                            | 85  |
| D Ukázka uživatelského rozhraní                         | 86  |
| E Ukázka nástroje pro vyhledávání známých zranitelností | 99  |
| F Ukázka webové prezentace platformy                    | 101 |
| G Obsah elektronické přílohy                            | 102 |

## A Požadavky webové platformy

- Modulární komponenty (např. tabulky, formuláře, grafy).
- Textový editor pro poznámky.
- Vlastnosti a validace uživatelských vstupů.
- Logování akcí, ná povědy a notifikace.
- Rozdělení obsahu do dynamických sekcí, widgetů a panelů.
- Automatická změna sekcí na základě jejich vlastnosti.
- Strom uzlů, jeho automatické přepočítávání a propojení s tabulkami.
- Drobečková navigace a pop-up okno pro zobrazení obsahu uzlu.
- Vytváření, klonování, kopírování, mazání a přesun uzlů.
- Aliasy uzlů a jejich provázání se stromem uzlů.
- Přílohy, export a import dat tabulek.
- Evidence nálezů a zranitelností, jejich export a import.
- CVSS kalkulátor a ukazatel závažnosti u nalezených zranitelností.
- Pokročilé uživatelské rozhraní (možnost přizpůsobení a nastavení).
- Kontextové menu jednotlivých komponent.
- Oprávnění komponent a uživatelských rolí.
- Ukládání nastavení rozhraní do LocalStorage.
- Uživatelské role a diskuze mezi členy týmu.
- Import struktury REST API a sitemap.
- Globální vyhledávání ve všech uzlech.
- WebSocket komunikace pro notifikace a diskuze.
- Procházení penetračního testu pomocí kontrolních seznamů.
- Tvorba šablon pro reporty, export a evidence reportů.
- Vyhledávání známých zranitelností a exploitů.
- Import nalezených zranitelností do reportu.
- Rozhraní pro automatizované penetrační testování.
- Konfigurace automatických testů.
- Sledování bezpečnosti v čase.
- Responzivní verze pro mobilní zařízení.
- Migrační struktura relační databáze.
- Intergrace systémů třetí strany (Jira, Trello, GitHub, GitLab aj.).
- Možnost rozšíření o další moduly bez zásahu do hlavní struktury.

## B Metriky a hodnoty CVSS

Příloha obsahuje tabulky následujících metrik:

- Tabulka B.1 – metriky základního skóre:
  - vektor útoku,
  - komplexita útoku,
  - vyžadované oprávnění,
  - interakce uživatele,
  - dopad na důvěrnost,
  - dopad na integritu,
  - dopad na dostupnost,
  - rozsah.
- Tabulka B.2 – metriky dočasného skóre:
  - dostupnost a kvalita exploitu,
  - dostupnost a kvalita nápravy,
  - dostupnost a kvalita informací.
- Tabulka B.3 – metriky základního skóre:
  - požadavek na důvěrnost,
  - požadavek na integritu,
  - požadavek na dostupnost,
  - modifikovaná skupina základních metrik.

Tab. B.1: Metriky základního skóre

| Název metriky            | Popis metriky        | Možné hodnoty | Popis hodnoty | Číselná hodnota |
|--------------------------|----------------------|---------------|---------------|-----------------|
| AV – Attack Vector       | Vektor útoku         | N – Network   | Sítový        | 0,85            |
|                          |                      | A – Adjacent  | Sousední      | 0,62            |
|                          |                      | L – Local     | Lokální       | 0,55            |
|                          |                      | P – Physical  | Fyzický       | 0,20            |
| AC – Attack Complexity   | Komplexita útoku     | L – Low       | Nízká         | 0,77            |
|                          |                      | H – High      | Vysoká        | 0,44            |
| PR – Privileged Required | Vyžadované oprávnění | N – None      | Žádné         | 0,85            |
|                          |                      | L – Low       | Nízké         | 0,62 (0,68)*    |
|                          |                      | H – High      | Vysoké        | 0,27 (0,50)*    |
| UI – User Interaction    | Interakce uživatele  | N – None      | Žádná         | 0,85            |
|                          |                      | R – Required  | Vyžadovaná    | 0,62            |
| C – Confidentiality      | Dopad na důvěrnost   | N – None      | Žádný         | 0               |
|                          |                      | L – Low       | Nízký         | 0,22            |
|                          |                      | H – High      | Vysoký        | 0,56            |
| I – Integrity            | Dopad na integritu   | N – None      | Žádný         | 0               |
|                          |                      | L – Low       | Nízký         | 0,22            |
|                          |                      | H – High      | Vysoký        | 0,56            |
| A – Availability         | Dopad na dostupnost  | N – None      | Žádný         | 0               |
|                          |                      | L – Low       | Nízký         | 0,22            |
|                          |                      | H – High      | Vysoký        | 0,56            |
| S – Scope                | Rozsah               | U – Unchanged | Nezměněný     | -               |
|                          |                      | C – Changed   | Změněný       | -               |

\*V případě změny rozsahu se použije hodnota v závorce.

Tab. B.2: Metriky dočasného skóre

| Název metriky               | Popis metriky                  | Možné hodnoty        | Popis hodnoty   | Číselná hodnota |
|-----------------------------|--------------------------------|----------------------|-----------------|-----------------|
| ECM – Exploit Code Maturity | Dostupnost a kvalita exploitu  | X – Not Defined      | Není definováno | 1               |
|                             |                                | H – High             | Vysoká          | 1               |
|                             |                                | F – Functional       | Funkční         | 0,97            |
|                             |                                | P – Proof-of-Concept | Demonstrována   | 0,94            |
|                             |                                | U – Unproven         | Neprokázána     | 0,91            |
| RL – Remediation Level      | Dostupnost a kvalita nápravy   | X – Not Defined      | Není definováno | 1               |
|                             |                                | U – Unavailable      | Nedostupná      | 1               |
|                             |                                | W – Workaround       | Neoficiální     | 0,97            |
|                             |                                | T – Temporary Fix    | Dočasná         | 0,96            |
|                             |                                | O – Official Fix     | Oficiální       | 0,95            |
| RC – Report Confidence      | Dostupnost a kvalita informací | X – Not Defined      | Není definováno | 1               |
|                             |                                | C – Confirmed        | Potvrzená       | 1               |
|                             |                                | R – Reasonable       | Odůvodněná      | 0,96            |
|                             |                                | U – Unknown          | Neznámá         | 0,92            |

Tab. B.3: Metriky skóre prostředí

| Název metriky                    | Popis metriky                   | Možné hodnoty                                      | Popis hodnoty   | Číselná hodnota |
|----------------------------------|---------------------------------|----------------------------------------------------|-----------------|-----------------|
| CR – Confidentiality Requirement | Požadavek na důvěrnost          | X – Not Defined                                    | Není definováno | 1               |
|                                  |                                 | L – Low                                            | Nízký           | 0,5             |
|                                  |                                 | M – Medium                                         | Střední         | 1               |
|                                  |                                 | H – High                                           | Vysoký          | 1,5             |
| IR – Integrity Requirement       | Požadavek na integritu          | X – Not Defined                                    | Není definováno | 1               |
|                                  |                                 | L – Low                                            | Nízký           | 0,5             |
|                                  |                                 | M – Medium                                         | Střední         | 1               |
|                                  |                                 | H – High                                           | Vysoký          | 1,5             |
| AR – Availability Requirement    | Požadavek na dostupnost         | X – Not Defined                                    | Není definováno | 1               |
|                                  |                                 | L – Low                                            | Nízký           | 0,5             |
|                                  |                                 | M – Medium                                         | Střední         | 1               |
|                                  |                                 | H – High                                           | Vysoký          | 1,5             |
| MBM – Modified Base Metrics      | Modifikované metriky zák. skóre | Stejné jako metriky základního skóre (tabulka B.1) |                 |                 |

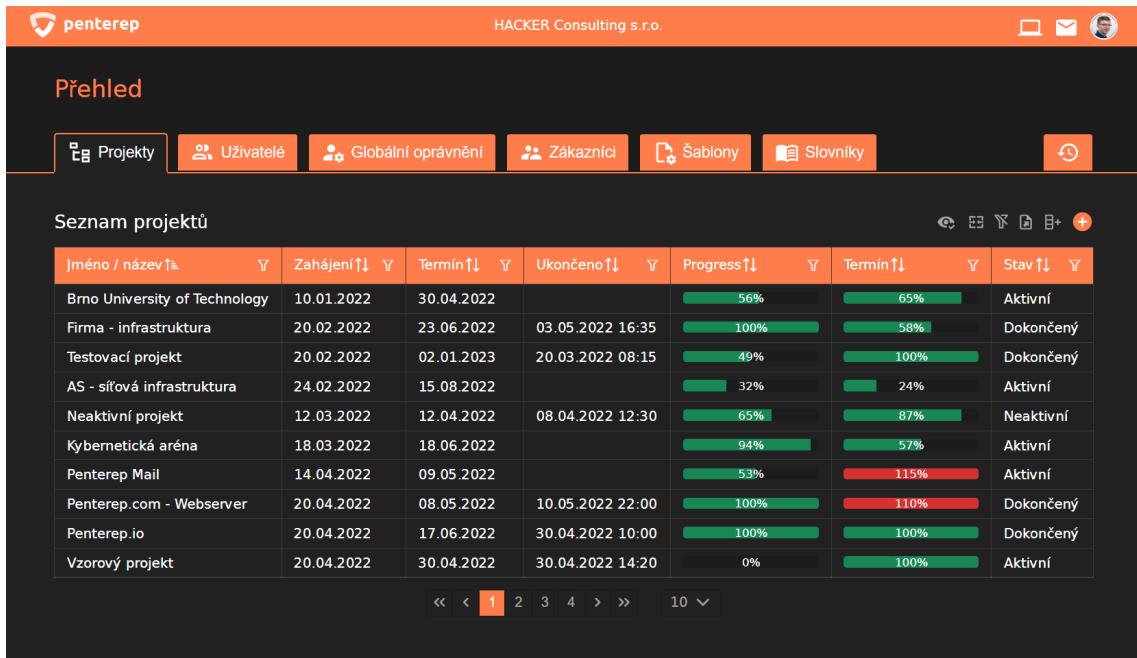
## C Velikost výsledného řešení

Tab. C.1: Velikost implementace praktické části

| Část platformy        | Velikost          |
|-----------------------|-------------------|
| Zdrojový kód          | 15 000 řádků      |
| Repozitář (GitHub)    | 1 000 commitů     |
| Implementované funkce | 430 funkcí        |
| Relační databáze      | 14 000 záznamů    |
| Testovací data        | 5 000 000 záznamů |

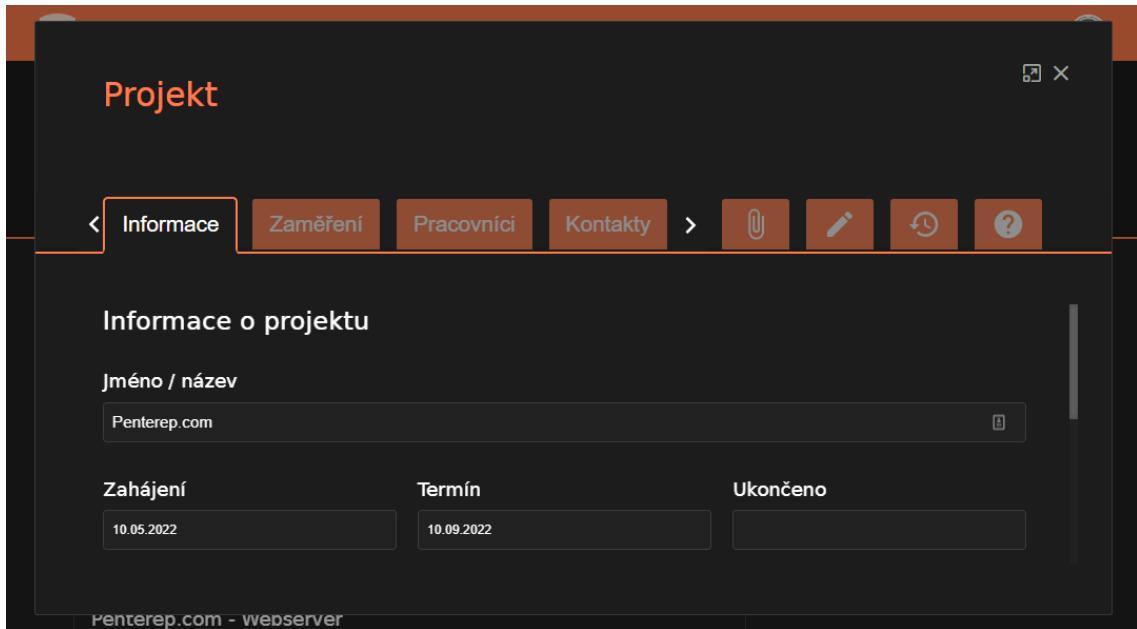
Pozn.: Číselné hodnoty byly zaokrouhleny dolů.

## D Ukázka uživatelského rozhraní



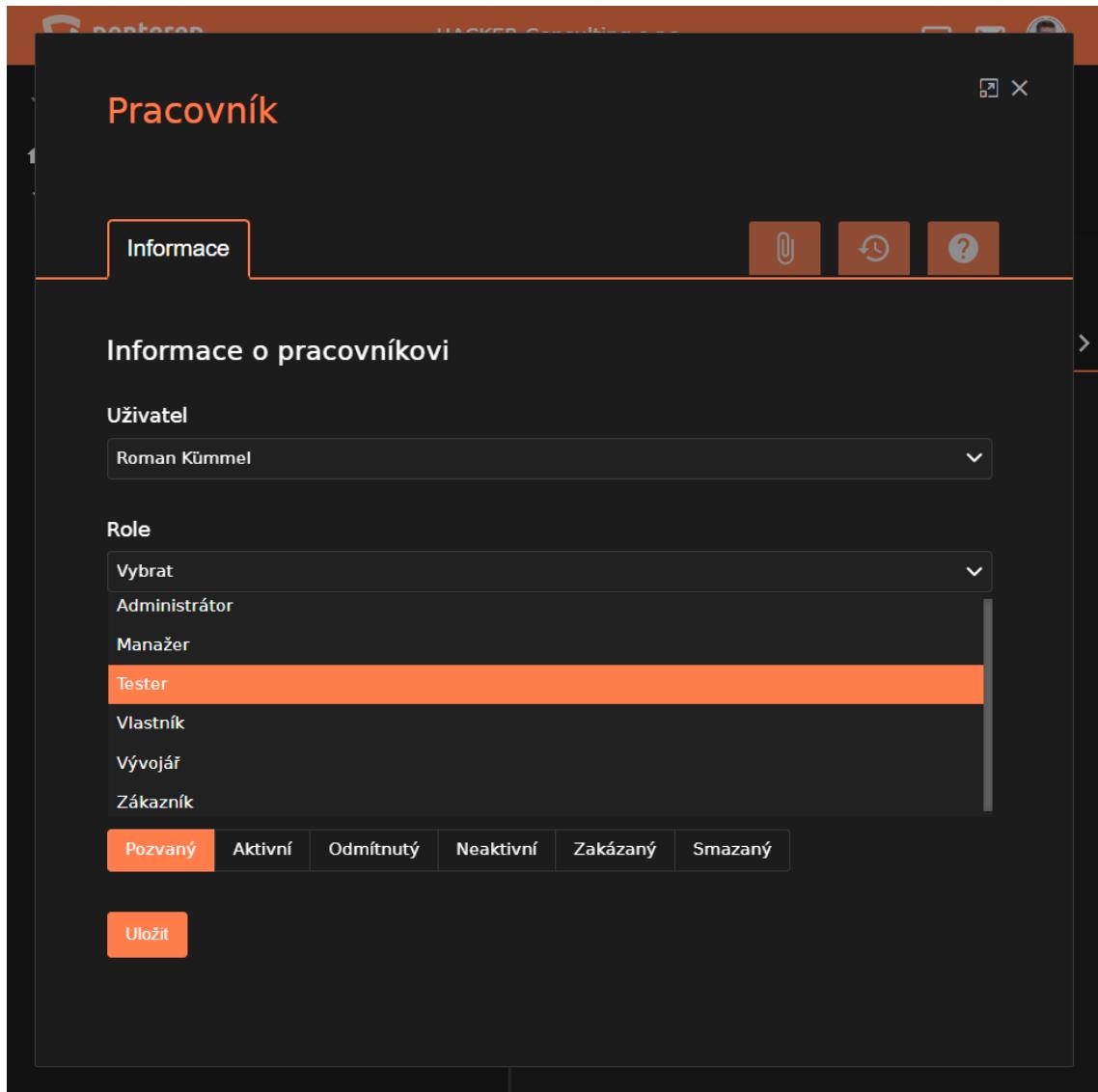
The screenshot shows the main dashboard of the Penterep platform. At the top, there's a header with the logo 'penterep' and the company name 'HACKER Consulting s.r.o.'. On the right side of the header are icons for a laptop, email, and user profile. Below the header is a navigation bar with tabs: 'Projekty' (selected), 'Uživatelé', 'Globální oprávnění', 'Zákazníci', 'Šablony', and 'Slovniky'. There's also a refresh icon and a '+' button. The main content area is titled 'Přehled' (Overview) and contains a section titled 'Seznam projektů' (List of projects). This section displays a table with columns: 'Jméno / název', 'Zahájení', 'Termín', 'Ukončeno', 'Progress', 'Termín', and 'Stav'. The table lists several projects with their respective details and progress bars. At the bottom of the table is a pagination control with buttons for '1' (selected), '2', '3', '4', '...', '10', and a dropdown menu.

Obr. D.1: Hlavní nástěnka, kde uživatel vidí projekty, do kterých byl pozván. Pokud je uživatel administrátorem, tak vidí další jinak nepřístupné sekce.



The screenshot shows a modal window titled 'Projekt' (Project). The window has a dark background with orange highlights. At the top, there's a navigation bar with tabs: 'Informace' (selected), 'Zaměření', 'Pracovníci', 'Kontakty', and buttons for 'Úloha', 'Upravit', 'Znovu', and '?'. Below the tabs, the title 'Informace o projektu' (Information about the project) is displayed. Underneath, there are input fields for 'Jméno / název' (Name / Name) containing 'Penterep.com', 'Zahájení' (Start date) with value '10.05.2022', 'Termín' (Deadline) with value '10.09.2022', and 'Ukončeno' (Completed) which is empty. At the bottom of the modal, the text 'Penterep.com - webserver' is visible.

Obr. D.2: Založení nového projektu se provádí z hlavní nástěnky. Pro vytvoření projektu musí uživatel mít přidělenou roli administrátora nebo vlastníka platformy.



Obr. D.3: Pozvání uživatelů do projektu se provádí přidáním nového uzlu typu „Pracovník“. Další uživatele může do projektu přidat administrátor, přičemž při tvorbě pozvánky volí i jejich příslušnou roli.

Obr. D.4: Výběr kontrolních seznamů se provádí po založení projektu a na základě jejich výběru se automaticky novým uzelům v projektu přiřadí příslušné testy podle typu uzlu (např. testy autentizace webové aplikace, lokálního úložiště apod.).

| Jméno / název | Typ stránky | URL                                             | HTTP ... | Citlivý obsah                       |
|---------------|-------------|-------------------------------------------------|----------|-------------------------------------|
| /images/      | Adresář     | https://cyberarena.utko.feec.vutbr.cz/images    | GET      | <input checked="" type="checkbox"/> |
| /sitemap.xml  | Sitemap.xml | https://cyberarena.utko.feec.vutbr.cz/sitem...  | GET      | <input type="checkbox"/>            |
| /index.php    | Javascript  | https://cyberarena.utko.feec.vutbr.cz/index.... | GET      | <input type="checkbox"/>            |
| /robots.txt   | Robots.txt  | https://cyberarena.utko.feec.vutbr/robots.txt   | GET      | <input checked="" type="checkbox"/> |

Obr. D.5: Uzly projektu se načítají spolu s ostatními uzly stejného typu do tabulky příslušné sekce rodičovského uzlu. Výběrem kontrolních seznamů dojde také k určení toho, co bude cílem testu (např. webová aplikace). Uzly projektu potom obsahují potomky, které lze přidat manuálně nebo pomocí automatizovaných nástrojů.

The screenshot shows a web-based configuration interface for a node named '/robots.txt'. The interface includes the following fields:

- Jméno / název:** /robots.txt
- Typ stránky:** Robots.txt
- URL:** https://cyberarena.utko.feec.vutbr.cz/robots.txt
- HTTP metoda:** GET
- Status kód:** 200
- Autorizace:** Public
- Citlivý obsah:** checked
- Popis:** (empty text area)

At the bottom left is a red "Uložit" (Save) button.

Obr. D.6: Manuální přidání nového uzlu se provádí prostřednictvím formuláře, který podle daného typu uzlu sestaví webová aplikace. Vlastnosti uzlu jsou určitého typu (např. textové pole, rozbalovací seznam, kalendář atd.) a podle toho mohou mít definovaná vlastní pravidla (např. povinné pole, interval hodnot, formát apod.).

| Označení            | Úkol                                                                              | Stav | Nález | Automatický test |
|---------------------|-----------------------------------------------------------------------------------|------|-------|------------------|
| PTL-WEB-INFO-OSSEN  | Není možné identifikovat OS pomocí case sensitivity?                              | ✓    | ✗     | 🔗                |
| PTL-WEB-INFO-WSICO  | Není dostupný alias na adresář /icons?                                            | ✓    | □     | 🔗                |
| PTL-WEB-INFO-OSLNK  | Neprozrazuje přístup k LPT1, COM1 použití OS Windows?                             | ✓    | □     | 🔗                |
| PTL-WEB-INFO-TEFPD  | Není možné dohledat FPD prostřednictvím webových vyhledávacích motorů?            | □    | □     |                  |
| PTL-WEB-INFO-OSTCP  | Není možné identifikovat OS pomocí odchylek v implementaci TCP?                   | □    | □     |                  |
| PTL-WEB-INFO-WSU... | Není možné identifikovat webový server prodlužováním URL?                         | □    | □     | 🔗                |
| PTL-WEB-INFO-WSR... | Není možné identifikovat webový server použitím neplatného portu?                 | ✓    | ✗     | 🔗                |
| PTL-WEB-INFO-WSR... | Není možné identifikovat webový server zvětšováním obsahu?                        | ✓    | ✗     | 🔗                |
| PTL-WEB-INFO-WSR... | Není možné identifikovat webový server na základě pořadí HTML?                    | ⚠    | □     | 🔗                |
| PTL-WEB-INFO-TEDPA  | Není dostupná defaultní uvítací stránka serveru?                                  | ✓    | □     | 🔗                |
| PTL-WEB-INFO-WSPAT  | Není možné identifikovat webový server Apache pokusem o připojení?                | ✓    | □     | 🔗                |
| PTL-WEB-INFO-LNGEX  | Není možné identifikovat programovací jazyk na základě přípony souborů?           | ✓    | □     | 🔗                |
| PTL-WEB-INFO-LNGHP  | Není možné identifikovat programovací jazyk přístupem k součástem stránek?        | ✓    | □     |                  |
| PTL-WEB-INFO-LNGSE  | Není možné identifikovat programovací jazyk prostřednictvím souborného rozšíření? | ✓    | □     |                  |
| PTL-WEB-INFO-LNG... | Není možné identifikovat programovací jazyk PHP použitím názvu souboru?           | ⚠    | □     | 🔗                |
| PTL-WEB-INFO-ANM... | Není možné identifikovat autora aplikace na základě obsahu stránek?               | □    | □     | 🔗                |
| PTL-WEB-INFO-ANC... | Není možné identifikovat autora aplikace na základě názvu souboru?                | ✓    | ✗     |                  |
| PTL-WEB-INFO-ANW... | Není možné identifikovat autora aplikace na základě obsahu stránek?               | ⚠    | □     | 🔗                |

Obr. D.7: Uzly určené k testování obsahují příslušné testy podle svého typu, hodnot vlastností (např. operační systém) a vybraných kontrolních seznamů projektu. Test uzlu se provede buď manuálně podle návodu v něm obsaženého, nebo se pro jeho otestování spustí automatizovaný nástroj.

**PTL-WEB-AUTH-PWMIN**

**Úkol**

Nemůže si uživatel nastavit heslo kratší než 12 znaků?

| Obtížnost | Časová náročnost | Automatický test |
|-----------|------------------|------------------|
| Snadné    | 10               |                  |

**Popis**

Úkolem tohoto testu je ověřit, zda aplikace zabraňuje uživateli v nastavení hesla kratšího než 12 znaků. Zjišťuje se také minimální délka hesla, kterou je aplikace ochotna akceptovat. Důvodem pro vynucení minimální délky hesla je zamezit útočníkům v hánání hesel hrubou silou, při nichž by bylo možné v relativně krátkém čase uhodnout správné heslo zkoušením jednotlivých kombinací znaků.

**Jak na to**

Během testu ověřte, zda si uživatel nemůže nastavit krátké heslo. Postupně použijte hesla s různou délkou, přičemž začněte prázdným řetězcem a následně heslo prodlužujte na 1, 2, 3, 4, 5, ... znaků. Tímto způsobem zjistěte minimální délku hesla, kterou je aplikace ochotna akceptovat. V případě, že aplikace vyžaduje použití komplexních hesel, je nutné tomuto požadavku přizpůsobit tvar testovacího řetězce. Musíte tedy použít hesla jako A1, Ab1, Ab1\$, Ab1\$23, apd.

**Výsledek**

Minimální možná délka hesla je / není 12 a více znaků  
Minimální možná délka hesla

**Reference**

ASVS 4.0 2.1.1  
WSTG-ATHN-07  
NIST 800-63B 5.1.1.2

| Stav       | Nález |
|------------|-------|
| Otestováno |       |

Obr. D.8: Uzly testu obsahují návod a další informace k provedení manuálního testu.

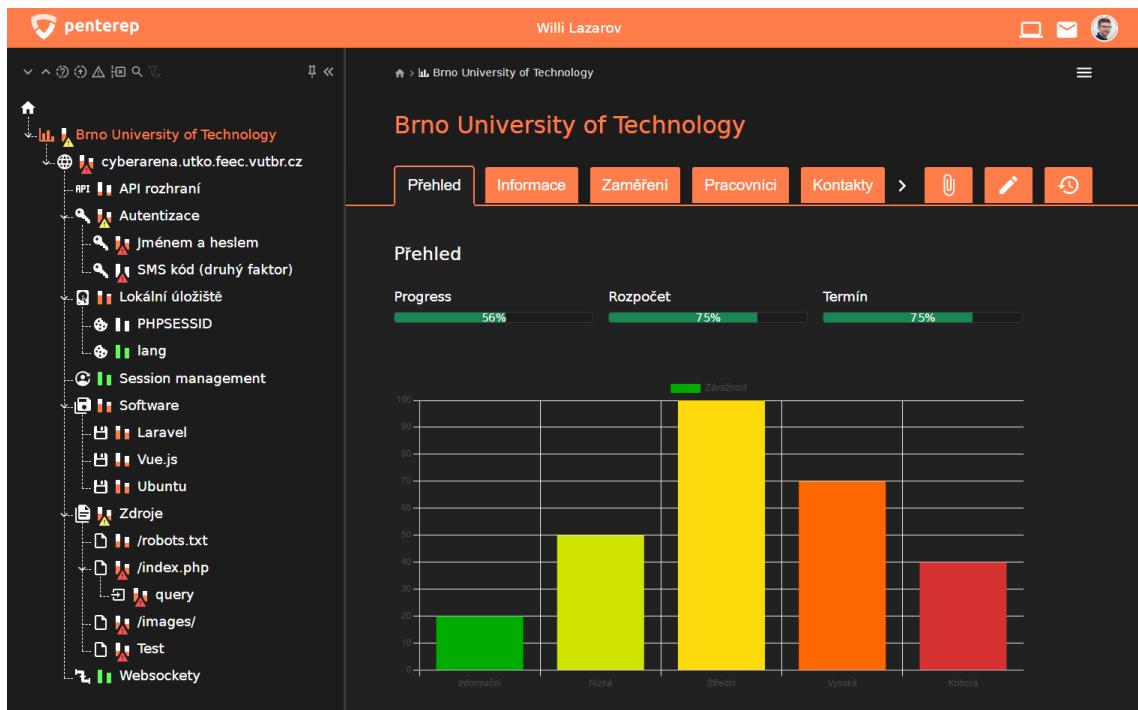
The screenshot shows a web-based application interface for security testing. At the top, there's a header with the logo 'penterep' and the text 'HACKER Consulting s.r.o.' followed by icons for a laptop, email, and user profile. Below the header, the main title 'Zranitelnost' is displayed. There are two tabs: 'Informace' (Information) and 'Diskuze' (Discussions), with 'Diskuze' currently selected. On the right side of the header are three orange action buttons: a clipboard icon, a pencil icon, and a circular arrow icon.

The main content area contains several sections:

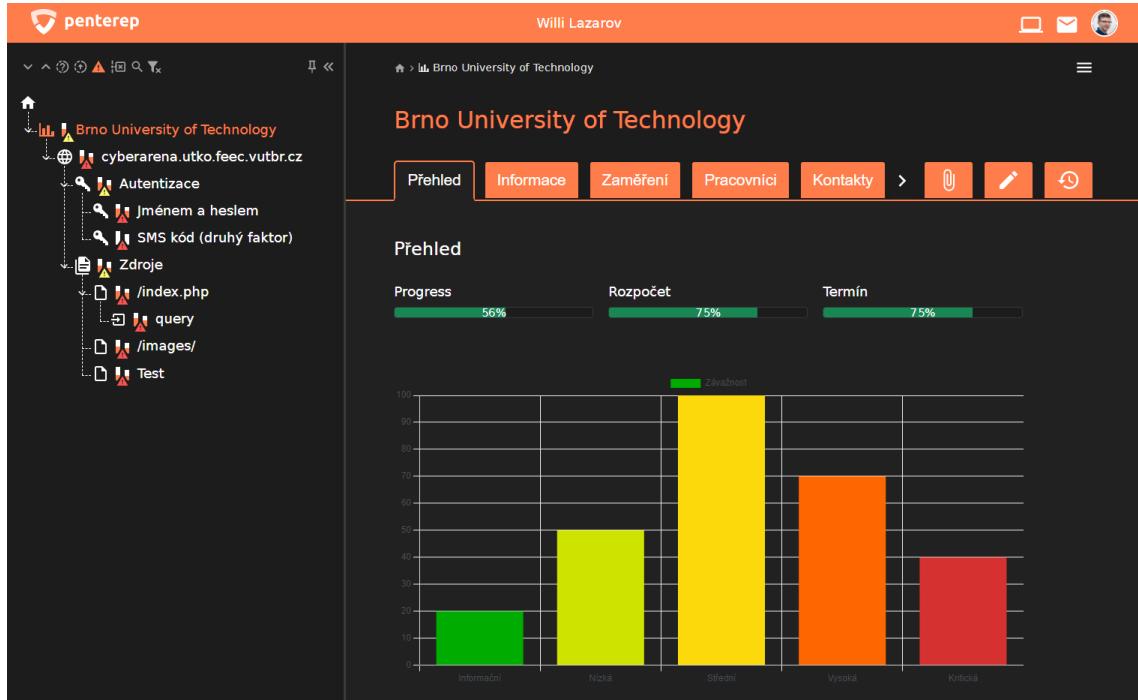
- Název ↗**: 'Uživatelům je povoleno použití krátkého hesla'
- Popis zranitelnosti ↗**: 'Uživatelům je povoleno použít hesel kratších než 12 znaků, což neodpovídá aktuálním bezpečnostním standardům.'
- Příčina ↗**: 'Příčinou je chybějící kontrola délky hesla při jeho nastavování, nebo příliš benevolentní nastavení, které uživatelům umožňuje nastavení hesel kratších než 12 znaků.'
- Projevy ↗**: 'Zranitelnost se projevuje možností nastavit si heslo kratší než 12 znaků.'
- Dopady ↗**: 'Útočník může provést různé typy útoků směřující k uhodnutí uživatelského hesla. Pokud jsou mu k dispozici uložené hashy hesel, může provést také různé útoky směřující k prolomení tétohashů. V případě použití krátkého hesla je pravděpodobnější, že se útočníkovi útok podaří a on tak získá přístupové údaje k cizímu účtu.'
- Doporučení ↗**: 'Náprava spočívá v přidání podmínky ověřující minimální délku hesla, nebo v úpravě stávajícího nastavení tak, aby heslo kratší než 12 znaků nebylo možné použít.'

Below these sections, there's a 'Závažnost' (Severity) section with five buttons: 'Informační', 'Nízká', 'Střední' (selected), 'Vysoká' (highlighted in orange), and 'Kritická'. Further down, under 'Reference', it lists 'ASVS 4.0 1.1.3'.

Obr. D.9: Při testu může dojít k nalezení zranitelnosti, která se stejně jako ostatní uzly založí manuálně, nebo automaticky v případě spuštění automatizovaného testu. U nalezených zranitelností je možné nastavit jeden z následujících stavů: falešně pozitivní, neověřené, potvrzené, k retestu, nebo opraveno. Každou z předvyplněných hodnot vlastnosti zranitelnosti lze dodatečně upravit.



Obr. D.10: Celkový postup testování konkrétního projektu lze vidět ve stromě uzelů, který pro každý uzel vizualizuje stav dokončených testů a grafickou indikaci, zda se v uzlu nebo některém z jeho potomků nachází zranitelnost.



Obr. D.11: Ve stromě uzelů lze filtrovat nedokončené nebo zranitelné uzly. Možné je také vyhledat konkrétní uzly zadáním jejich názvu.

The screenshot shows a web-based application interface for managing notes or snippets. At the top, there's a header with the logo 'penterep' and the text 'HACKER Consulting s.r.o.'. Below the header, the URL is displayed as 'Brno University of Technology > cyberarena.utko.feec.vutbr.cz > Zdroje > /index.php'. The main content area shows a dark-themed browser window with the title '/index.php'. Inside the browser window, there's a heading 'Debugovací režim' (Developer mode). Below it, a message says 'Výpis Console vypisuje následující chybové zprávy:' (Console output displays the following error messages:).

```

    • TypeError: Cannot set properties of null (setting '__draggable_context')
      at mA [main.487e91e6.js:47:38553]
      at main.487e91e6.js:47:39054
      at Array.forEach (<anonymous>)
      at HA.updated [main.487e91e6.js:47:39038]
      at Proxy.updated [main.487e91e6.js:47:41586]
      at L5 [main.487e91e6.js:4:656]
      at H1 [main.487e91e6.js:4:735]
      at Array.Jo.t._wen.t._wen [main.487e91e6.js:4:18459]
      at so [main.487e91e6.js:4:2174]
      at V8 [main.487e91e6.js:4:2400]

    • Failed to create chart: can't acquire context from the given item
    • Uncaught (in promise) Error: Canvas is already in use. Chart with ID '0' must be destroyed before the canvas can be reused.
      at new zi [auto.esm.15d52109.js:16:79067]
      at main.487e91e6.js:1648:3752

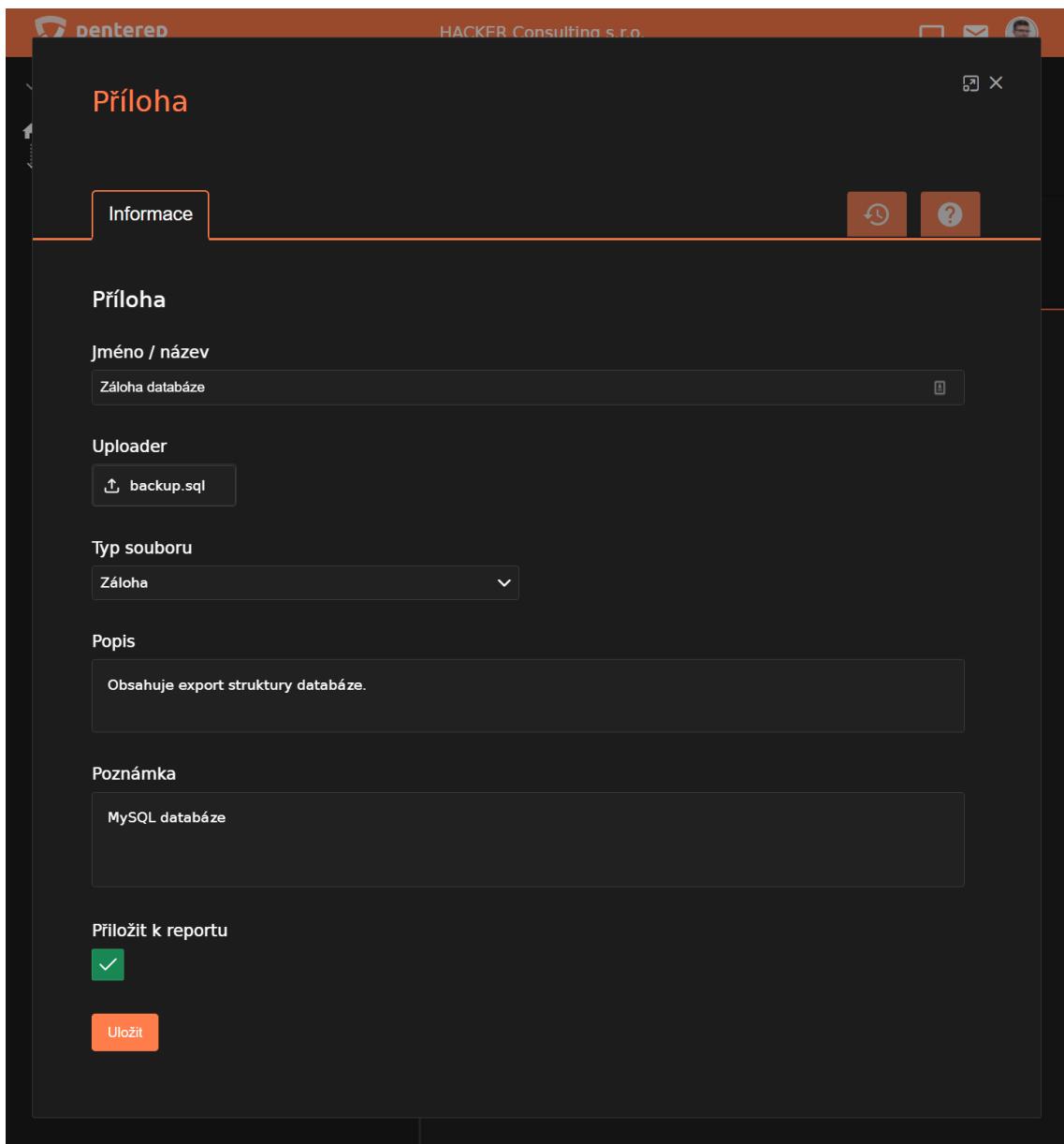
    • TypeError: Cannot set properties of null (setting '__draggable_context')
      at mA [main.487e91e6.js:47:38553]
      at main.487e91e6.js:47:39054
      at Array.forEach (<anonymous>)
      at HA.updated [main.487e91e6.js:47:39038]
      at Proxy.updated [main.487e91e6.js:47:41586]
      at L5 [main.487e91e6.js:4:656]
      at H1 [main.487e91e6.js:4:735]
      at Array.Jo.t._wen.t._wen [main.487e91e6.js:4:18459]
      at so [main.487e91e6.js:4:2174]
      at V8 [main.487e91e6.js:4:2400]

```

Below the error messages, there's a section titled 'Screenshot' containing a redacted screenshot of a browser developer tools console.

At the bottom left, there's a 'Debugging:' section with some code and a callout pointing to a variable 'undefined'.

Obr. D.12: Každý uživatel může k uzlu, ke kterému má přístup, vkládat vlastní poznámky. Komponenta pro editaci poznámek nabízí základní funkce formátování textu (např. tučné písma, kurziva, nadpisy atd.) a možnost vkládat obrázky.



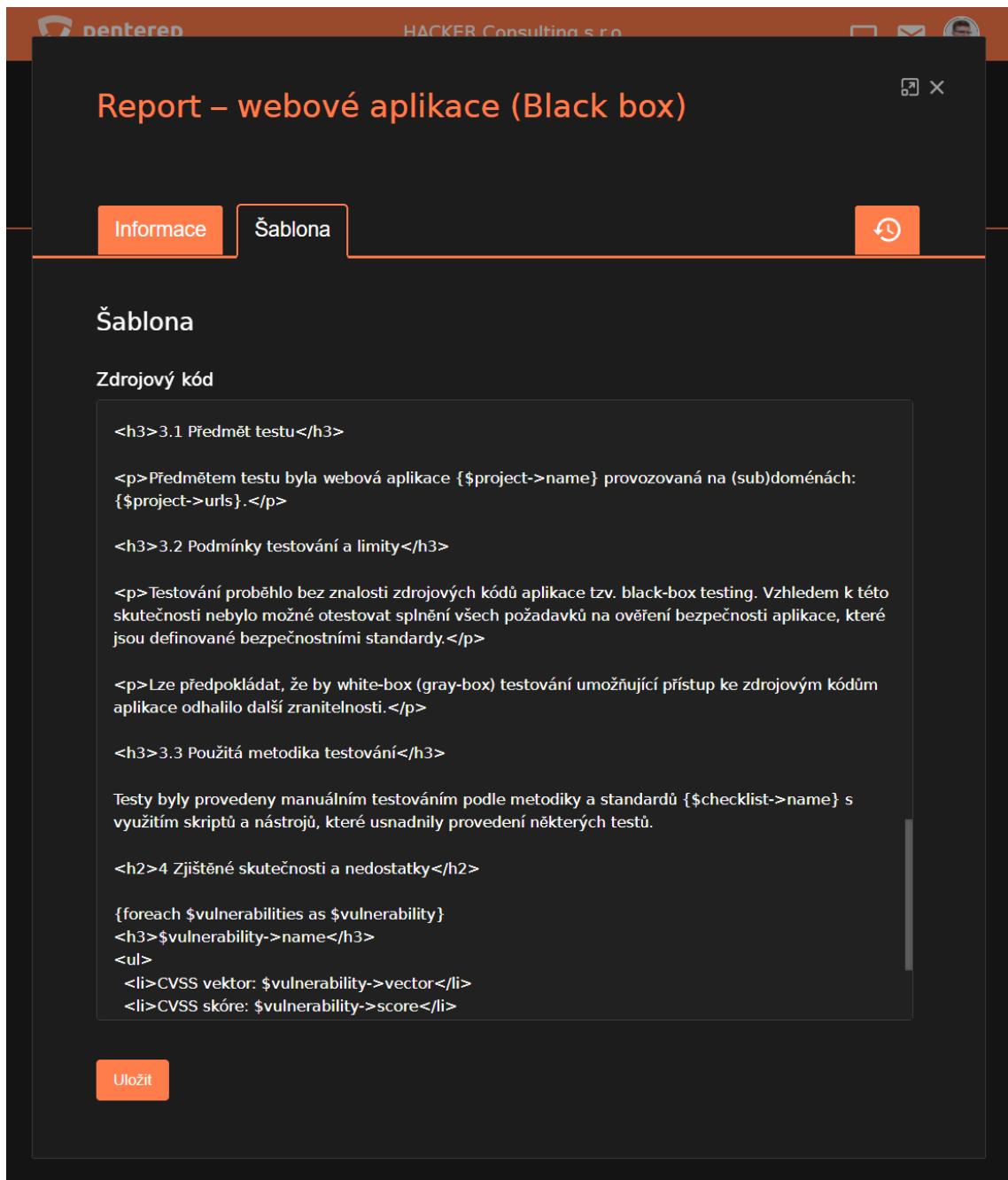
Obr. D.13: V platformě lze ke každému uzlu nahrávat přílohy (např. nalezené zálohy v průběhu testování), které mohou být připojeny k výslednému reportu. Přílohy lze označit s příznakem pro připojení k reportu, aby mohly být automaticky při jeho vytvoření k reportu připojeny v podobě archivovaného souboru.

The screenshot shows a web application interface for managing network nodes. At the top, there's a header with the logo 'penterep' and the text 'HACKER Consulting s.r.o.'. Below the header, the URL 'Brno University of Technology' is visible. The main content area is titled 'Brno University of Technology'. A navigation bar below the title includes links for 'Přehled', 'Informace', 'Zaměření', 'Pracovníci', 'Kontakty', 'Reporty', and several icons for search, refresh, and user profile.

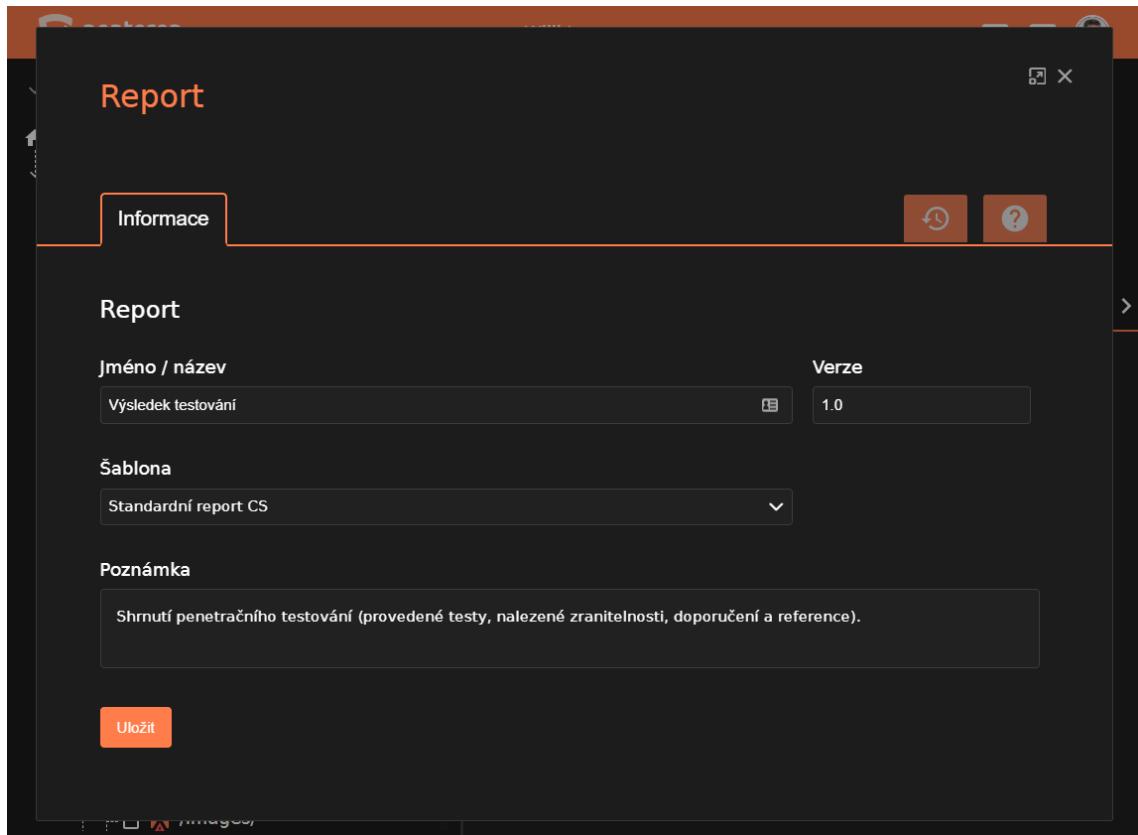
The central part of the screen is a table titled 'Logy' (Logs). The table has columns for 'Datum a čas' (Date and time), 'Typ logu' (Log type), 'Uživatel' (User), 'IP adresa' (IP address), 'Typ uzlu' (Node type), and 'Název uzlu' (Node name). The table contains 20 rows of log entries, mostly from 'Willi Lazarov' at IP 147.223.146.148, detailing various actions like 'Úprava' (Edit) and 'Založení' (Creation) on different node types such as 'Projekt' and 'Report'.

| Datum a čas        | Typ logu   | Uživatel      | IP adresa       | Typ uzlu  | Název uzlu                    |
|--------------------|------------|---------------|-----------------|-----------|-------------------------------|
| 12. 05. 2022 08:39 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 12. 05. 2022 08:39 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 12. 05. 2022 08:29 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 12. 05. 2022 08:28 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 12. 05. 2022 08:25 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 12. 05. 2022 08:25 | Úprava     | Willi Lazarov | 147.223.146.148 | Projekt   | Brno University of Technology |
| 19. 04. 2022 19:44 | Úprava     | Willi Lazarov | 83.142.32.11    | Projekt   | Brno University of Technology |
| 18. 04. 2022 14:20 | Úprava     | Willi Lazarov | 83.142.32.11    | Projekt   | Brno University of Technology |
| 18. 04. 2022 12:35 | Úprava     | Willi Lazarov | 83.142.32.11    | Projekt   | Brno University of Technology |
| 24. 03. 2022 19:28 | Založení   | Willi Lazarov | 172.18.0.1      | Report    | Report + přílohy              |
| 16. 03. 2022 13:11 | Úprava     | Willi Lazarov | 147.223.117.34  | Projekt   | Brno University of Technology |
| 15. 03. 2022 09:56 | Založení   | Willi Lazarov | 147.223.146.148 | Report    | Report v3                     |
| 15. 03. 2022 09:56 | Odstranění | Willi Lazarov | 147.223.146.148 | Report    | Report v2                     |
| 15. 03. 2022 09:54 | Založení   | Willi Lazarov | 147.223.146.148 | Report    | Report v1                     |
| 11. 03. 2022 21:45 | Úprava     | Willi Lazarov | 83.142.32.11    | Pracovník | Roman Kummel                  |
| 11. 03. 2022 21:09 | Založení   | Willi Lazarov | 83.142.32.11    | Příloha   | Dokumentace REST API          |
| 10. 03. 2022 16:40 | Založení   | Willi Lazarov | 147.223.148.102 | Příloha   | Technické parametry           |
| 10. 03. 2022 13:52 | Založení   | Willi Lazarov | 147.223.148.102 | Příloha   | Zadávací dokument             |
| 10. 03. 2022 13:51 | Odstranění | Willi Lazarov | 147.223.148.102 | Kontakt   | Správce sítě                  |

Obr. D.14: Každá akce provedená nad konkrétním uzlem se ukládá do logu, který je potomkem dotčeného uzlu nebo také rodičovského uzlu, pokud se ho změna týká. Například při založení nového uzlu se vytvoří log této události v rodičovském uzlu a zároveň log jako potomek nového uzlu se všemi vytvořenými vlastnostmi.



Obr. D.15: Report projektu se generuje podle předem definované šablony, která je naplněna daty podle uzlů daného projektu. Šablona reportu se vytváří jako uzel, který je potomkem globálního uzlu. Zdrojový kód šablony je psán v jazyce HTML.



Obr. D.16: Výsledný report projektu se vytváří podle vybrané šablony jako nový uzel, který je přímý potomek uzlu projektu. Vygenerovaný report je možné stáhnout jako HTML nebo PDF soubor. Stáhnout lze i archiv s přílohami všech uzelů, které byly označeny příznakem pro zahrnutí přílohy do reportu.

## E Ukázka nástroje pro vyhledávání známých zranitelností

Jedním z cílů bakalářské práce bylo vyvinout nástroj pro podporu penetračního testování. Výsledným řešením je nástroj `ptvulnsearcher`, který byl při řešení této práce naprogramován v jazyce Python a je volně dostupný ke stažení pod otevřenou licencí GPLv3 na adrese <https://pypi.org/project/ptvulnsearcher/>. Nástroj `ptvulnsearcher` slouží pro vyhledávání známých zranitelností, k čemuž využívá služby a data z veřejně přístupných databází popsané v kapitole 2.4.

Vyhledávání známých zranitelností lze provést zadáním klíčových slov příkazem `ptvulnsearcher -s <klíčová slova>`. Vyhledat lze také konkrétní zranitelnost zadáním příslušného označení CVE pomocí příkazu `ptvulnsearcher -cve <cve>`. Ukázkou rozhraní nástroje lze vidět na obrázku E.1 a ukázka nalezených zranitelností podle zadaných klíčových slov je zachycena na obrázku E.2.



```
ptvulnsearcher v0.0.1
https://www.penterep.com

Description:
    Tool for searching CVE (Common Vulnerabilities and Exposures)

Usage:
    ptvulnsearcher <options>

Usage example:
    ptvulnsearcher -s Apache v2.2

Options:
    -s    --search    <search>    Search keywords
    -cve --cve       <cve>      Search specific CVE
    -j    --json       Output in JSON format
    -v    --version     Show script version and exit
    -h    --help        Show this help message and exit
```

Obr. E.1: Rozhraní nástroje `ptvulnsearcher`

Obr. E.2: Výsledky vyhledávání nástroje ptvulnsearcher

## F Ukázka webové prezentace platformy

V průběhu řešení bakalářské práce byla vyvinuta webová prezentace platformy, která je v anglickém jazyce veřejně dostupná na adrese <https://www.penterep.io>. Jejím obsahem je kromě základního popisu platformy také časová osa a reference projektu aplikovaného výzkumu, při jehož řešení platforma vznikla. Webová prezentace je vedlejším výsledkem a byla vyvinuta nad rámec zadání bakalářské práce. Ukázku obsahu webové prezentace lze vidět na obrázku F.1.

The screenshot displays the Penterep web application. At the top, there's a navigation bar with the logo, a search bar, and links for Features, Timeline, Team, and PenterepMail, along with social media icons for Facebook and LinkedIn. Below the navigation, there are two columns of bullet points under the heading 'Features':

- ❖ Manual and automated testing
- ❖ Project management with many options
- ❖ Checklists and guides
- ❖ Tests for specific targets
- ❖ Team collaboration
- ❖ Automated customizable reporting
- ❖ and much more...

- ❖ High modularity, scalability, and optimization
- ❖ Visualization of the tested environment
- ❖ Advanced and responsive user interface
- ❖ Charts, calculators, and many UI components
- ❖ Secure platform environment
- ❖ Advanced logging system
- ❖ and much more...

In the center, there's a section titled 'Tests' with a table showing test results for various OSes (Windows, Linux, Mac OS X, Solaris). The table includes columns for 'OS' and 'Status' (Green, Red, Yellow). Below the table, there are sections for 'Statistics' (Progress, Budget, Deadline) and 'Report' (with a detailed log of the penetration test).

Penterep is a web platform for comprehensive penetration testing. From the beginning, the platform was developed with a primary focus on modularity, scalability, and optimization. We created a considerably large environment in the limited time of one year, which is gradually being extended by other parts and its possibilities of use are thus increasing over time. The main contribution of the platform is the unification of all penetration tester tasks into one place, automation of testing processes, and team collaboration between testers and other users. The result is a solution that improves the effect of penetration testing to such an extent that the time, complexity, and work required to successfully complete the entire test will be significantly lower than using the currently available tools.

Obr. F.1: Webová prezentace platformy

## G Obsah elektronické přílohy

Obsahem elektronické přílohy je tato bakalářská práce uložená ve formátu PDF. Dále je součástí přílohy demonstrační video a textový soubor s odkazy pro další informace. Příloha obsahuje ve vlastním adresáři článek a prezentaci z konference STUDENT EEICT 2022, na které byly výsledky bakalářské práce prezentovány. Zdrojové kódy webové platformy pro podporu penetračního testování a schéma relační databáze jsou součástí obchodního tajemství společnosti Hacker Consulting s.r.o., v případě zájmu kontaktujte info@hacker-consulting.cz.

```
/ ..... kořenový adresář přiloženého archivu
├── 2022_BP_Lazarov.pdf ..... bakalářská práce
├── STUDENT_EEICT_2022 ..... adresář s dokumenty z konference
│   ├── 2022_EEICT_Lazarov_clanek.pdf
│   └── 2022_EEICT_Lazarov_prezentace.pdf
└── Videoukazka.mp4 ..... demonstrační video výsledného řešení
└── Odkazy.txt ..... soubor s odkazy pro další informace
```