VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Simona Holovová

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

APLIKACE NA PODPORU TESTOVÁNÍ BEZPEČNOSTI WEBOVÝCH APLIKACÍ
APPLICATION THAT SUPPORTS PENETRATION TESTS OF WEB APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Simona Holovová**
AUTHOR

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

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Informační bezpečnost**

Ústav telekomunikací

Studentka: Bc. Simona Holovová ***ID:*** 185927
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Aplikace na podporu testování bezpečnosti webových aplikací

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit webovou aplikaci na podporu manuálního penetračního testování. Vytvořená aplikace bude sbírat vstupy od testera a vytvářet evidenci testování a zpřístupňovat specifické kontrolní seznamy nutných úkonů testování. Aplikace umožní testerovi vytvářet nové projekty. Každý projekt bude obsahovat kontrolní seznam testovacích případů, které je potřeba provést jednou v rámci celé aplikace (průzkum prostředí, zranitelnosti serveru apd.). V každém projektu následně tester vkládá seznam všech nalezených stránek. U každé stránky bude k dispozici kontrolní seznam, který je potřeba provést pro každou stránku. Ke každé stránce lze přidávat formuláře se seznamy úkonů, které je nutné provést u každého formuláře. Ke každému formuláři bude možné přidávat seznam jednotlivých vstupů, včetně informace o datovém typu vstupu (řetězec, číslo, soubor, apd.). U každého z testovacích případů by měla být uvedena nápověda obsahující krátký popis jak daný test provést. Vytvořená webová aplikace bude přehledně zobrazovat postup a dosažené výsledky penetračního testu. Aplikace by měla být navržena a implementována modulárně z důvodu snadné rozšiřitelnosti. Kontrolní seznamy a nápovědy budou vycházet z metodologie OWASP (OWASP Application Security Verification Standard – ASVS a OWASP Testing Guide).

DOPORUČENÁ LITERATURA:

[1] CUTHBERT, Daniel. Application Security Verification Standard. online https://www.owasp.org, 3 2019.

[2] KEARY, Eoin. Owasp testing guide. Online https://www.owasp.org/images/1/19/OTGv4.pdf, 2019.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

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

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

ABSTRAKT

Diplomová práca je o bezpečnosti webových aplikácií a o penetračných testoch. Hlavným cieľom je oboznámenie sa s metodológiou OWASP Testing Guide a ASVS a implementovať tieto poznatky do webovej aplikácie na podporu manuálneho penetračného testovania. Teoretická časť rozoberá už spomínané metodológie a webové technológie použité pri vývoji aplikácií. Samotná praktická časť sa venuje návrhu aplikácie podľa zadania, jej implementovaniu a následne zabezpečeniu.

KĽÚČOVÉ SLOVÁ

Penetračné testovanie, ASVS, OWASP, ZAP, Laravel, PHP, bezpečnosť

ABSTRACT

This master´s thesis is about the security of web applications and penetration testing. The main goal is to gain knowledge about testing methodologies OWASP Testing Guide and ASVS and to implement this knowledge into a web application to assist during manual penetration testing. The theoretical part of the thesis describes both methodologies and web technologies used during the development of the application. The practical part of the thesis is about the design of the application based on the specification, its implementation, and security hardening.

KEYWORDS

Penetration testing, ASVS, OWASP, ZAP, Laravel, PHP, security

HOLOVOVÁ, Simona. *Aplikace na podporu testování bezpečnosti webových aplikací*. Brno, 2020, 64 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Zdeněk Martinásek, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Aplikace na podporu testování bezpečnosti webových aplikací“ som vypracovala samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autorka uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušila autorské práva tretích osôb, najmä som nezasiahla nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomá následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autorky

POĎAKOVANIE

Rada by som poďakovala vedúcemu diplomovej práce pánovi Ing. Zdeňkovi Martináskovi, Ph.D. za odborné vedenie, konzultácie a trpezlivosť. Taktiež by som rada poďakovala pánovi Romanovi Kümmelovi za jeho spoluprácu na tejto práci, konzultácie a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov

Úvod

Komunikácia, práca, vzdelávanie. Tieto a mnohé ďalšie činnosti je dnes možné realizovať prostredníctvom internetu a s tým spojenou službou World Wide Web. Webové stránky sa stali významným zdrojom informácií aj miestom pre zábavu a uľahčujú nám každodenný život. V súvislosti s týmto trendom môžeme hovoriť o tzv. webových aplikáciach. Používanie webových aplikácií umožňuje od jednoduchej komunikácie napríklad medzi obchodnými partnermi, cez sledovanie zábavných videí, či spravovaní vlastných financií, až po rýchle a pohodlné platby z pohodlia domova.

Avšak, okrem týchto pozitív webové aplikácie prinášajú určité riziká. V dôsledku masového rozvoja týchto technológií je otázka bezpečnosti veľmi aktuálna. Činnosť každej aplikácie je spojená so spracovávaním veľkého množstva dát, ktoré je potrebné zabezpečiť. Bezpečnosť by mala byť v tomto prípade prioritou, no nie vždy je tomu tak. Bezpečnostné chyby vznikajú z rôznych príčin, môže ísť o ľudský faktor v dôsledku nedostatočnej informovanosti programátora alebo napríklad aj kvôli časovému tlaku na vydanie aplikácie. Jednou z možností v otázke bezpečnosti webových aplikácií je penetračné testovanie.

Penetračný test je v princípe bezpečnostný test aplikácie, ktorého cieľom je odhalenie bezpečnostných chýb v aplikácií. Tester sa v tomto prípade správa ako reálny útočník a snaží sa aplikáciu napadnúť. Jediný rozdiel medzi testerom a reálnym útočníkom je ten, že tester má povolenie od majiteľa/správcu danej aplikácie na vykonanie testu.

Cieľom tejto práce je vytvorenie aplikácie na podporu penetračného testovania. Tester bude môcť zadávať svoje nálezy do aplikácie, kde prehľadne uvidí, aké testy už boli vykonané a ktoré je nutné ešte vykonať. Do aplikácie bude možné zadať jednotlivé objavené vstupy a aplikácia sama vygeneruje kontrolný zoznam testov, ktoré je nutné pre daný typ vstupu vykonať.

Účelom teoretickej časti tejto práce je priblížiť metodológie penetračného testovania OWASP Testing Guide, ktoré slúžia ako hlavný zdroj počas penetračného testovania. Jednotlivé podkapitoly popisujú, aké časti aplikácie sa testujú.

Praktická časť práce popisuje vlastný návrh celej aplikácie na základe špecifikovaných požiadaviek, vrátane návrhu databázy a popisuje použité technológie pri vývoji aplikácie, ako je PHP framework Laravel alebo návrhový vzor Model-View-Controller. Znalosti z tejto kapitoly sú využité pri implementácií aplikácie. Návrh je graficky zdokumentovaný pomocou UML diagramov. Treťou kapitolou práce je samotná vlastná implementácia aplikácie v jazyku PHP. Vychádza z návrhu a popisuje postup práce s frameworkom Laravel. Na ukážkach je demonštrovaný princíp, na ktorom aplikácia pracuje a postup implementácie jednotlivých modulov je popísaný

krok po kroku. Kapitola sa zaoberá aj kontrolnými zoznamami, spôsobom, akým boli vytvorené a ako sa s nimi pracuje. Uvedený je aj postup zabalenia aplikácie do Docker kontajnera, ktorý slúži na jednoduché nasadenie a distribúciu aplikácie.

Posledná kapitola je venovaná bezpečnosti samotnej aplikácie. Vlastné testovanie zahŕňalo manuálne zabezpečovanie aplikácie ako aj automatizovaný sken pomocou aplikácie OWASP ZAP.

1 OWASP

Open Web Application Security Project (OWASP) je nezisková organizácia zaoberajúca sa bezpečnosťou softwaru hlavne v oblasti webu. OWASP Foundation vznikla 21.4.2004 v USA, no už od 1.12.2001 sú na stránke www.owasp.org dostupné materiály ohľadom tohto projektu [1]. Iniciátormi a zakladateľmi projektu sú Mark Curphey a Dennis Groves. Hlavným cieľom organizácie je zlepšiť bezpečnosť a zabezpečenie webových aplikácií a šíriť osvetu v tejto oblasti. Zaoberá sa tvorbou bezplatne dostupných materiálov, dokumentov, metodológií či nástrojov v tejto oblasti.

1.1 OWASP Testing Guide

Projekt Testing Guide vznikol v roku 2003. Cieľom projektu je pomôcť ľuďom pochopiť princípy testovania webových aplikácií [1]. Či už ide o laickú verejnosť so záľubou v technológiách alebo o bezpečnostných profesionálov, Testing Guide je považovaný za štandard pri testovaní webových aplikácií. Mnohé firmy vykonávajú penetračné testy práve podľa metodológie OWASP Testing Guide.

Na úvod uvádza Testing Guide základné poznatky o životnom cykle softwaru [1] (SDLC – Software Development Life Cycle) a zdôrazňuje, že na bezpečnosť je nutné myslieť už pri prvotnom návrhu aplikácie. Taktiež sú spomenuté základy penetračného testovania. Táto práca sa zameriava najmä na samotné bezpečnostné testy webových aplikácií. Cieľom tejto kapitoly je predstaviť Testing Guide a časti, ktoré obsahuje. Účelom nie je vytvoriť detailný popis jednotlivých sekcií, preto budú uvedené podkapitoly zamerané iba na hlavné aspekty a princípy. OWASP delí bezpečnostné testovanie do jedenástich kategórií.

Zber informácií (Information Gathering)

Na začiatku každého testu je vykonávaný zber informácií. Zámerom je zistiť čo najviac informácií o testovanej aplikácií [1]. Využíva sa samotná aplikácia (prejdenie celej funkcionality, jej pochopenie) ale aj aplikácie tretích strán ako napríklad vyhľadávače. Pomocou vyhľadávačov je možné nájsť napríklad produkčnú verziu aplikácie, prípadne testovacie verzie, rôzne administratívne rozhrania a podobne. Webové vyhľadávače môžu tiež nájsť citlivé informácie o webovej aplikácii, napríklad zabudnuté zdrojové kódy, mená vývojárov (môžu byť využité pre sociálne inžinierstvo) a podobne.

Podstatnou časťou zberu informácií je zistenie webového servera. Názov (a prípadne aj verzia) webového servera najčastejšie uniká prostredníctvom HTTP (Hy-

pertext Transfer Protocol) hlavičiek v odpovediach servera (response header). HTTP je protokol na siedmej vrstve ISO/OSI modelu. Ide o komunikačný protokol primárne určený na komunikáciu medzi prehliadačom (klientom) a webovým serverom. Vo výpise 1 je možné vidieť názornú odpoveď servera, ktorá obsahuje názov aj verziu servera. Takáto odpoveď by napríklad vznikla pomocou príkazu *curl http://localhost/* (podobne zvyšné ukážky HTTP odpovedí predpokladajú štandardnú HTTP GET požiadavku, napr. pomocou curl).

```
1 HTTP/1.1 200 OK
2 Server: Apache/1.3.23
```

Výpis 1: Odpoveď servera obsahuje jeho názov a verziu

Citlivosť verzie a názvu webového servera je daná tým, že útočníkovi umožňuje cieliť útok na úzko definovanú oblasť [1]. Webové servery, podobne ako iné aplikácie, majú bezpečnostné chyby, ktoré sú zverejnené a dajú sa vyhľadať pod uvedeným CVE (Common Vulnerabilities and Exposures – publikovaným zraniteľnostiam je pridelený unikátny identifikátor) identifikátorom[1]. Viacero dostupných zraniteľností má verejne dostupný exploit (spustiteľný kód na zneužitie zraniteľnosti). Ak útočník vie presnú verziu servera, nie je preňho problém nájsť dostupné zraniteľnosti a vyskúšať ich. V prípade ak verziu nevie, útok je zložitejší a vyžaduje omnoho viac času (nakoľko je potrebné prejsť veľké množstvo zraniteľností pre všetky servery). Penetračný tester sa správa ako útočník, preto aj on hľadá verziu servera na jej možné zneužitie.

Z aplikácie môžu unikať (relatívne) citlivé informácie aj formou komentárov v zdrojovom kóde. Komentáre môžu obsahovať reťazce ako „TODO:“, ktoré môžu naznačiť, že niektorá časť aplikácie nie je v dokončenom stave, a teda môže obsahovať zraniteľnosti [1]. Komentáre často obsahujú verzie jednotlivých knižníc, ktoré aplikácia využíva (ide o podobný problém ako pri verzií servera).

Ďalším krokom pri zbere informácií je skúmanie použitých technológií v aplikácií. Pokiaľ ide o aplikáciu postavenú na známom riešení (napr. PHP Laravel, WordPress a podobne, často open-source), môžu byť na danú verziu známe zraniteľnosti. Aplikácie, ktoré sú „na mieru“ majú zvyčajne uzavretý kód a nájsť voľne dostupné zraniteľnosti je takmer nemožné.

Všetky získané informácie sú pre testera/útočníka podstatné a môžu sa hodiť v neskoršej fáze testu/útoku.

[1] https://cve.mitre.org/

Management konfigurácie a nasadenia (Configuration and Deployment Management)

Táto kapitola sa prioritne zameriava na konfiguráciu infraštruktúry (configuration and deployment management), na ktorej je aplikácia spustená. Najčastejšie sa využíva sken otvorených portov. Správna konfigurácia servera má otvorené iba tie porty, ktoré nevyhnutne potrebuje [1]. Zvyčajne ide o porty 80 (HTTP) a 443 (HTTPS). Administratívne porty (ako napríklad port 20 – SSH) by nemali byť prístupné z internetu (ideálne obmedzenie na špecifické IP adresy a podobne). Ak by bol takýto port voľne prístupný, bolo by možné realizovať rôzne útoky ako brute-force na prelomenie užívateľského účtu a podobne.

Tester musí overiť, či sa na serveri nenachádzajú staré/zabudnuté súbory záloh alebo iné citlivé dokumenty, ktoré aplikácia nevyužíva, prípadne ktoré by nemali byť prístupné.

Aplikácie môžu obsahovať administratívne rozhranie (napríklad phpMyAdmin). Tieto rozhrania (podobne ako SSH port) by nemali byť prístupné z internetu. Prístup by mal byť obmedzený na špecifické IP adresy alebo podobným riešením.

Management identity (Identity Management)

Aplikácie obsahujú často viaceré roly. Napríklad rola „admin“ alebo „užívateľ“. Je na posúdení testera, či sú roly dobre definované. Platí princíp najmenších privilégií [1] (užívateľ by nemal mať vyššie práva než také, ktoré nevyhnutne potrebuje).

Ďalšou časťou kapitoly je registračný proces. Aplikácia by nemala prezradiť, či existuje zadaný účet (v prípade ak takéto informácie poskytuje, je možné získať zoznam užívateľských účtov a uskutočniť útok hrubou silou pre prelomenie hesla).

Najmä vo firemnej sfére sú užívateľské účty predpovedateľné. Napríklad pri využívaní platformy Active Directory (Windows) môže ísť o špecifický formát ako napríklad meno.priezvisko alebo [prvé písmeno krstného mena].priezvisko a podobne. Tieto schémy sú náchylné na útoky hrubou silou či slovníkové útoky. Mená zamestnancov je jednoduché nájsť napríklad na sieti Linkedin.com.

Autentifikácia (Authentication)

Sekcia autentifikácie rozoberá bezpečnosť prihlasovania. Pre prihlásenie je nevyhnutné použiť šifrovaný kanál (HTTPS), prihlasovacie údaje zaslané prostredníctvom HTTP protokolu nesmú byť overované. Užívatelia by taktiež mali zmeniť svoje počiatočné heslá po prvom prihlásení.

Tester sa snaží obísť autentifikačnú schému a vniknúť do chránenej sekcie aplikácie bez znalosti užívateľského mena a hesla (alebo inej kombinácie prihlasovacích

údajov). Využívajú sa metódy ako SQL injection [1] alebo modifikácia parametrov zaslaných v HTTP požiadavke (napr. isAdmin=true).

Ďalšou časťou je problematika ukladania stránok do cache pamäte. V prípade nesprávnej konfigurácie nastáva situácia, kedy prehliadač ukladá do cache pamäte navštívené stránky. Táto funkcionalita sa využíva na zrýchlenie načítavania statického obsahu ako knižnice, obrázky a podobne, avšak môže dôjsť ku „cacheovaniu“ aj citlivých stránok [1] (napr. stránka v internet bankingu s informáciami ako IBAN (International Bank Account Number – medzinárodné číslo bankového účtu), zostatok na účte, vykonané transakcie a podobne). Pre zabránenie ukladania stránky do cache pamäte musí server v odpovedi správne nastaviť hlavičky Cache-Control, ako je vidieť vo výpise 30.

```
1 HTTP/1.1 200 OK
2 Cache-Control: no-cache, no-store
```

Výpis 2: Zabránenie ukladania odpovede do cache pamäte

Autorizácia (Authorization)

Autorizácia umožňuje užívateľom vykonávať špecifikované úlohy a neumožní vykonať úlohy, ktoré daný užívateľ nemá právo vykonať (nie je autorizovaný). Napríklad užívateľ U má právo vykonať akciu A (napr. napísať komentár), avšak nemá právo vykonať akciu B (napr. modifikovať cudzí komentár). Úlohou testera je zistiť, či neexistuje spôsob, ako môže užívateľ U vykonať akciu B, na ktorú nie je oprávnený. Medzi testovacie metódy patrí modifikácia parametrov (napr. zmena ID komentáru pre úpravu) alebo priamy prístup ku zdrojom (napríklad ak je známe alebo predikovateľné číslo faktúry).

Tester sa snaží eskalovať svoje práva na vyššiu rolu, napríklad z užívateľa na administrátora aplikácie. Jednotlivé metódy sú špecifické pre danú aplikáciu a závisia na jej implementácií.

Management relácií (Session Management)

HTTP je bezstavový protokol. Pre správne fungovanie webových aplikácií sa využíva „relácia“, ktorá býva označovaná parametrom *session ID*. Najčastejšie sa ukladá do cookie a prehliadač ju posiela na server pri každej požiadavke. Server týmto spôsobom vie, od ktorého užívateľa prišla požiadavka a vie overiť autorizáciu užívateľa (či má oprávnenia na vykonanie danej akcie). Pokiaľ by sa útočníkovi podarilo odchytiť session ID parameter, vedel by aplikáciu využívať bez prihlásenia dovtedy, kým

sa užívateľ neodhlási (za predpokladu, že odhlásenie funguje). Je teda nevyhnutné, aby session ID parameter bol generovaný náhodne, nebolo ho možné uhádnuť ani odchytiť (nesmie sa zasielať v GET parametroch ani nešifrovane) [1]. Chybné zasielanie session ID parametra je vidieť vo výpise 3. Problém GET parametrov je ten, že sú viditeľné v podobe otvoreného textu napríklad v histórií prehliadača, môžu byť uložené v logoch firewallu/IPS systému, prípadne na cache serveroch a proxy. Attack-surface (v tomto prípade počet spôsobov ako získať platné session ID) je tak značne zvýšený, čím sa zvyšuje aj riziko zneužitia.

```
1 GET /hello.php?session=1A45Sdasd4587Aglojki HTTP/1.1
2 Host: test.example
```

Výpis 3: Chybné zasielanie session ID parametru v GET požiadavke

Správny spôsob zasielania session ID parametru je pomocou cookie, prípadne v tele POST požiadavky (výpis 4).

```
1 GET /hello.php HTTP/1.1
2 Host: test.example
3 Cookie: sessionId=1A45Sdasd4587Aglojki;
```

Výpis 4: Správne zasielanie session ID parametru pomocou cookie

Pri prihlásení je tiež nevyhnutné, aby aplikácia vygenerovala novú reláciu s novým session ID aj v prípade, že užívateľ zadal vlastnú hodnotu session ID. Aplikácia nesmie zobrať hodnotu session ID od užívateľa a pri prihlásení ju vyzdvihnúť na platnú session ID. Ide o *session fixation* [1] útok, pri ktorom môže útočník podvrhnúť obeti cookie s dlhou platnosťou. Keďže útočník pozná hodnotu cookie a server negeneruje novú hodnotu cookie pri prihlásení, môže využiť cookie pre prihlásenie do aplikácie vždy, keď je prihlásená obeť (najmä na stránkach, kde nedôjde k automatickému odhláseniu môže ísť o veľký problém).

Odhlásenie užívateľa je taktiež podstatným prvkom aplikácie. Užívateľ by mal mať vždy možnosť odhlásenia sa z aplikácie. Týmto krokom musí byť zneplatnená relácia (najmä na strane servera). Nesmie dôjsť iba k odstráneniu cookie na strane klienta.

Validácia vstupov (Input Validation)

Validácia vstupov je najrozsiahlejšia kapitola v Testing Guide. Obsahuje návod pre testovanie viacerých kategórií vstupov. Od vstupov, ktoré sa reflektujú na stránke

(*cross-site scripting* – XSS, ide o útok zaslaním spustiteľného kódu, zvyčajne javascript, ktorý sa vykoná u používateľa), cez vstupy do databázy (*SQL injection*) až po rôzne XML (*eXtensible Markup Language*) vstupy (*XML external entity* – XXE) alebo zadávanie príkazov pre operačný systém (*remote code execution* – RCE).

Podľa dát spoločnosti Imperva [2], sú najčastejšími zraniteľnosťami webových aplikácií práve nesprávne ošetrené vstupy. Až 19 % všetkých zraniteľností sú rôzne formy injection. Prekvapujúco, väčšinu z nich netvorí *SQL injection*, ale práve kritický *remote code execution* (RCE), ktorý umožňuje vykonať spustiteľný kód na strane servera. Zo všetkých zraniteľností za rok 2018 išlo o *RCE* v 11,5 % prípadoch, *SQL injection* bol nájdený iba v 8 % prípadoch.

Druhou najčastejšou chybou je zraniteľnosť *Cross-Site Scripting* (XSS) [1, 2]. Ide o zraniteľnosť, pri ktorej server nekontroluje užívateľský vstup, ktorý obsahuje spustiteľný javascript kód. Vzorovým príkladom XSS je napríklad vloženie kódu vo výpise 5 do formulára. Ak sa údaje z formulára zobrazia v inej časti aplikácie (ideálne v administratívnom rozhraní, kam majú prístup admini), tak je možné získať *session ID* užívateľa a odoslať ho na vlastný server. Kód vytvorí v štruktúre stránky nový obrázok s umiestnením na serveri mojserver.sk, ktorý je pod správou útočníka. Prehliadač odošle požiadavku na tento obrázok a pridá hodnotu cookie (document.cookie) do URL (*Uniform Resource Locator* – označuje zdroj na internete). Útočník si na svojom serveri „vyzdvihne“ hodnotu cookie a môže využívať aplikáciu pod reláciou obete. Týmto spôsobom je možné kompromitovať užívateľské účty. Útok v tejto forme by fungoval iba ak sú parametre cookie nesprávne nastavené (chýba HttpOnly/secure parameter).

```
1 <script>
2 new Image().src="http://mojserver.sk/cookie.php?cookie="
3         +document.cookie;
4 </script>
```

Výpis 5: Odoslanie hodnoty cookie na vlastný server

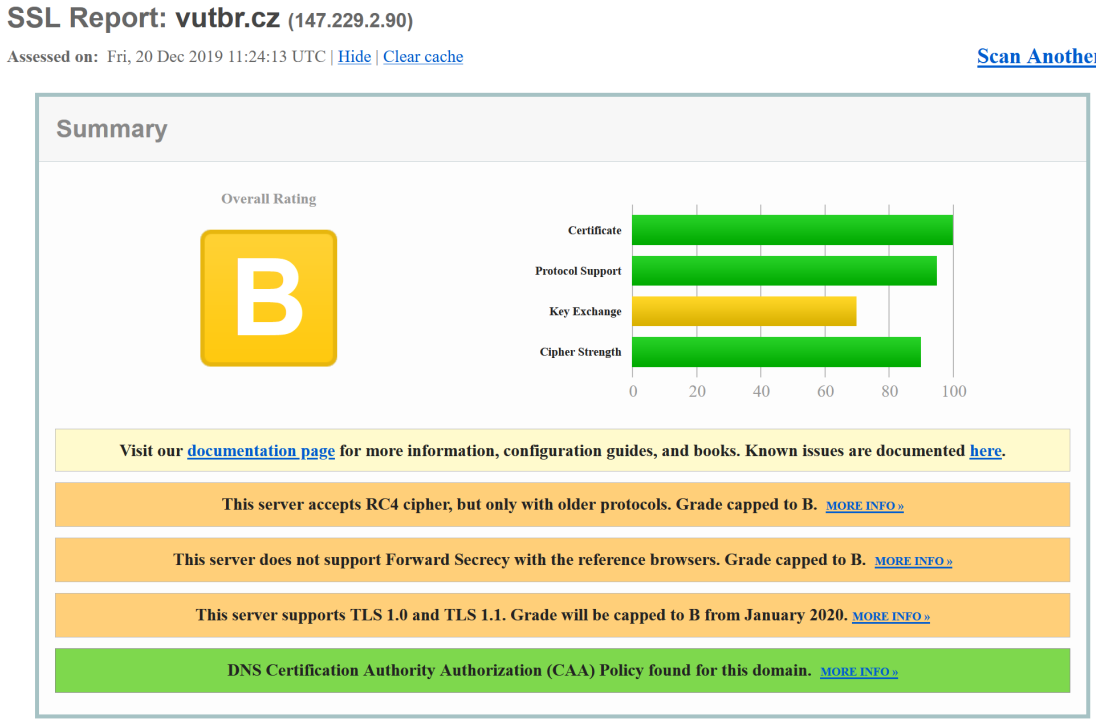
Ošetrenie chýb (Error Handling)

Kapitola sa skladá z dvoch častí, a to analýza chybových správ a analýza stack-trace (chybový výpis, ktorý obsahuje informácie ako volané metódy, cesty k jednotlivým súborom a podobne). V praxi platí, že aplikácia by nemala obsahovať príliš detailné chybové hlásenia. Takéto hlásenie nesmie obsahovať názvy používaných produktov a ich verzie, podobne nesmie obsahovať informácie o cestách k súborom (podľa ciest

je možné zistiť typ operačného systému a umiestnenie aplikácie). Aplikácia by vôbec nemala zobrazovať stack-trace výpisy.

Kryptografia (Cryptography)

Nesprávne nasadenie TLS certifikátu môže spôsobiť problémy pre aplikáciu. Podpora oslabených kryptografických systémov alebo zastaraných verzií TLS (1.0, 1.1) [3] môže ohroziť užívateľov aplikácie a tým aj samotnú aplikáciu. Pre overenie správnej konfigurácie TLS je možné použiť napríklad online nástroj Qualys SSL Labs[2]. Vzorový výstup z domény vutbr.cz je možné vidieť na obrázku 1.1. Z obrázku je zjavné, že TLS konfigurácia na vutbr.cz obsahuje chyby. Napríklad tu je podporovaná šifra RC4 a TLS verzie 1.0 a 1.1.



Obr. 1.1: SSL/TLS test webového servera vutbr.cz

Biznis logika (Business Logic)

Kapitola sa zameriava na bezpečnostné chyby pri biznis logike aplikácie. Napríklad pri eshope by sa kontrolovalo, či nie je možné tovar objednať bez zaplatenia, či sa nedá jednorázový zľavový kupón využiť viackrát. Podobne sa testujú možnosti preskočenia niektorých krokov v aplikácií. V prípade webového fóra by mohlo ísť o preskočenie nutnosti schválenia príspevku moderátorom [1]. Testovanie biznis

[2]https://www.ssllabs.com/ssltest/

logiky je špecifické pre danú aplikáciu a tester sa musí vynájsť pri metódach jej testovania. Aplikácia by tiež mala overovať zadané dáta a pracovať s nimi podľa svojho návrhu. Pre názornosť, predpokladajme online kalkulačku na výpočet mesačných splátok hypotéky. Zadaním špecifických parametrov ako je dĺžka splácania, preferovaná mesačná splátka či veľkosť hypotéky by mohol byť systém, ktorý cenu počíta uvedený do špecifického stavu, kedy by uviedol cenu hypotéky za nulovú alebo cenu neprimerane malú.

Testovanie strany klienta (Client Side Testing)

Testovanie strany klient (Client-side testing) sa zaoberá možnosťami spúšťania kódu na strane klienta. Môže ísť o HTML/CSS/Javascript kód a podobne. Pod túto kapitolu tiež spadá testovanie odchytávania klikov klienta (Clickjacking), pomocou ktorého môžeme obeť prinútiť klikať na špecifické miesta v aplikácií, a tak oklamať obeť pre realizovanie nevyžiadanej akcie. Aplikácia by v tomto prípade bola prekrytá vrstvou útočníka (napríklad forma hry alebo podobne) a jednotlivé tlačidlá by boli umiestnené na špecifických miestach tak, aby korešpondovali s tlačidlami v aplikácií. Obeť si myslí, že kliká na hru, avšak reálne sú kliknutia presmerované do aplikácie a nevyžiadaná akcia sa zrealizuje.

Správa (Report)

Správa o vykonanom penetračnom teste je záverom samotného testu. Dobrá správa musí obsahovať zadanie testu (scope, dodané materiály, čas začiatku/ukončenia, všetky dohodnuté informácie s klientom a podobne). Nasleduje manažérske zhrnutie, v ktorom sú jednotlivé nálezy popísane „ľudskou rečou“ s vysvetleným popisom a hrozbou, ktorú daná zraniteľnosť predstavuje pre samotnú aplikáciu, klientov a prevádzkovateľa. Poslednú časť tvorí samotný popis nálezov s detailným postupom ako bol nález objavený, jeho klasifikácia podľa závažnosti (informačná až kritická) a odporúčanie na odstránenie zraniteľnosti. Táto časť správy je najrozsiahlejšia a je písaná odborným štýlom s dôrazom na proof-of-concept ukážky s vysvetlením, akú hrozbu daná zraniteľnosť predstavuje.

1.2 Application Security Verification Standard

Ide o samostatný projekt OWASP. ASVS nie je súčasťou Testing Guide, no do značnej miery z neho vychádza. Testing Guide je možné chápať ako rozsiahly návod, zatiaľčo ASVS je iba kontrolný checklist [4] s testami, ktoré je nutné vykonať pre daný vstup alebo overenie danej zraniteľnosti.

ASVS poskytuje vývojárom návod, aké kontroly majú do svojich aplikácií implementovať. Podobne je to aj návod pre bezpečnostných špecialistov, ktorí penetračné testy realizujú, aby nezabudli na žiaden potrebný test a mohli tak dôkladne vykonať testovanie podľa zadania. ASVS rozdeľuje aplikácie na tri úrovne [4]:

- Level 1 – poskytuje nízku mieru uistenia o bezpečnosti aplikácie
- Level 2 – odporúčaná úroveň pre väčšinu aplikácií (najmä tie, ktoré obsahujú citlivé dáta)
- Level 3 – určené pre najkritickejšie aplikácie, ktoré realizujú transakcie alebo obsahujú zdravotné dáta

Výhodou ASVS (a jeho úrovní) je, že objednávateľ penetračného testu tak aj tester presne vedia, ako detailne bude aplikácia otestovaná a ktoré testy sa budú uskutočňovať. Vývojári používajú ASVS pri bezpečnom vývoji a udržiavaní aplikácií. Ukážka z ASVS je zobrazená na obrázku 1.2.

#	Description	L1	L2	L3	CWE	NIST §
2.1.1	Verify that user set passwords are at least 12 characters in length. (C6)	✓	✓	✓	521	5.1.1.2
2.1.2	Verify that passwords 64 characters or longer are permitted. (C6)	✓	✓	✓	521	5.1.1.2
2.1.3	Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. (C6)	✓	✓	✓	521	5.1.1.2
2.1.4	Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.	✓	✓	✓	521	5.1.1.2

Obr. 1.2: Ukážka z ASVS dokumentu

1.3 Manuálny a automatizovaný penetračný test

V praxi platí, že automatizované skenovanie na zraniteľnosti nie je veľmi presné a viaceré komplexnejšie zraniteľnosti, ktoré by človek odhalil, automat nenájde [1]. Automat taktiež nevie logicky spájať viacero zraniteľností dokopy tak ako to dokáže človek. Pre automatizované testovanie je však k dispozícii viacero nástrojov ako napríklad OWASP ZAP (bezplatný) alebo Burp Suite Professional (platený). Tieto nástroje (najmä ZAP) je možné integrovať do vývojového procesu a spúšťať napríklad sken s každou aktualizáciou aplikácie, prípadne integrovať do CI/CD procesu (Continuous Integration/Continuous Delivery). Keďže ide o bezplatný nástroj, jeho využitie je výhodne aj ak nedokáže nájsť všetky zraniteľnosti. Aj drobné zraniteľnosti (napríklad nesprávne nastavené hlavičky) je lepšie odhaliť počas vývoja než pri produkčnom nasadení.

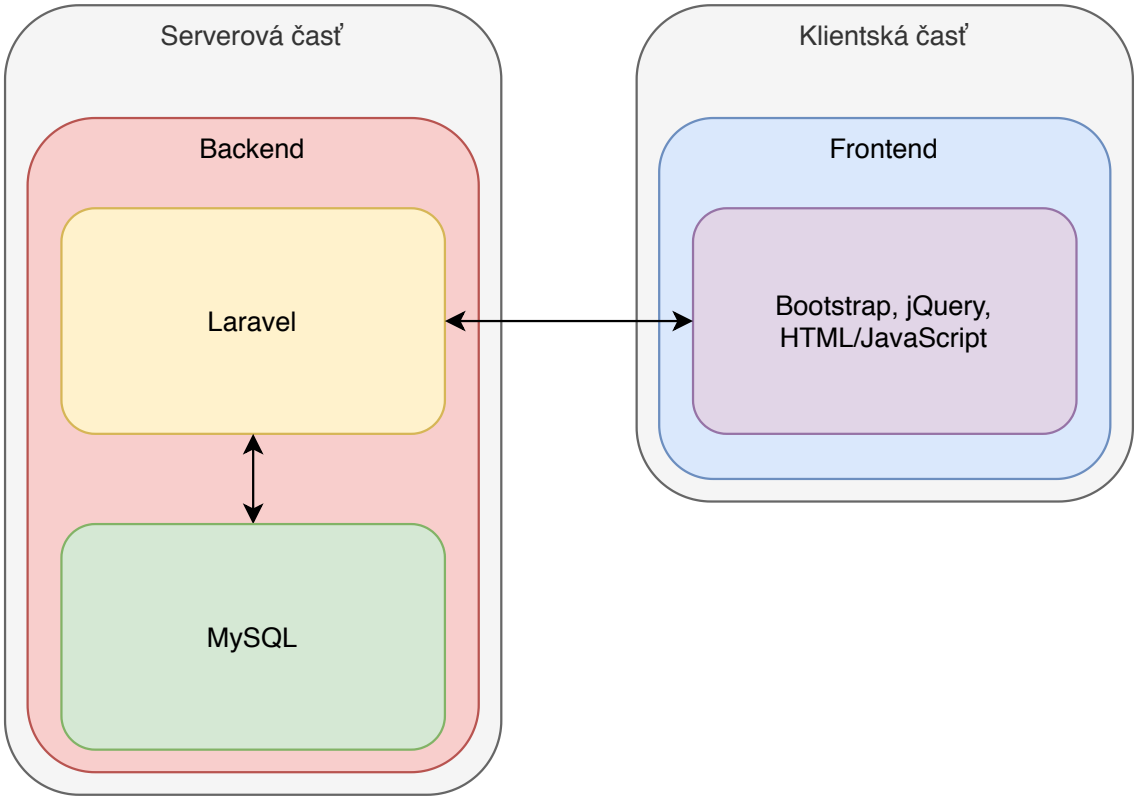
Na druhej strane manuálny test vykonáva človek. Tým pádom je značne drahší a často je realizovaný externou firmou. Preto je nutné zabezpečiť komunikáciu a spoluprácu medzi oboma subjektmi. Skúsený tester odhalí väčšinu (v ideálnom stave všetky) zraniteľností. Test však trvá dlhšie a nie je možné ho vykonať na počkanie.

2 Vlastný návrh webovej aplikácie

Táto kapitola sa venuje návrhu architektúry tak aj funkcionality aplikácie. V tomto prípade je návrh vytvorený abstraktne, nakoľko by bolo veľmi nepravdepodobné, že by bol pri implementácií návrh dodržaný. Až pri samotnej implementácií sa bude dať zistiť, čo presne je nevyhnutné pri jednotlivých objektoch. Aj smerovanie projektu sa s najväčšou pravdepodobnosťou bude počas práce meniť.

2.1 Použité technológie

Na praktickú implementáciu webovej aplikácie na podporu manuálneho testovania webových stránok boli použité viaceré moderné technológie ako framework Laravel, či databáza MySQL a ďalšie. Tieto technológie môžeme rozdeliť do dvoch základných a všeobecne známych kategórii, a to konkrétne časť Frontend (na strane klienta) a časť Backend (strana servera). Na obrázku 2.1 je toto rozdelenie vidieť, spolu s použitými technológiami.



Obr. 2.1: Rozdelenie aplikácie na serverovú a klientskú časť

2.1.1 Frontend

Pod anglickým názvom *Frontend* je možné rozumieť časť webovej aplikácie či stránky viditeľnej pre používateľa, teda všetky prvky stránky, s ktorými má používateľ možnosť akýmkoľvek spôsobom interagovať [5]. Príkladom môže byť vyhľadávacie menu, prihlasovací formulár, či akýkoľvek iný viditeľný (prípadne v danom momente aj skrytý) prvok stránky. Dôležitým prvkom viditeľnej časti webovej aplikácie je jej responzívnosť, teda schopnosť prispôsobiť sa zobrazovanej ploche a čo najefektívnejšie ju využiť. V praxi to obvykle znamená odlišné zobrazovanie webovej stránky na malom telefóne, stredne veľkom tablete či veľkom monitore notebooku, či stolného počítača. Pre splnenie tohto kritéria je v praktickej časti práce použitý framework Bootstrap.

Bootstrap

Základom Frontendu, teda pre používateľa viditeľnej časti aplikácie, je framework Bootstrap [5, 6]. Samotný framework je postavený na najnovšom štandarde HTML5, ktorý zabezpečuje vizuálne prvky, kaskádových štýlov CSS3, ktoré sa starajú o štýly jednotlivých prvkov a skriptov v jazyku Javascript, respektíve v knižnici jQuery, ktoré prispôsobujú vzhľad stránky veľkosti zobrazovacej plochy. Bootstrap bol pôvodne vyvinutý firmou Twitter už v roku 2010, momentálne je však vo verzii 4 ako open source projekt a je možné do neho prispievať svojimi nápadmi.

jQuery

V aplikácií sa využíva knižnica jQuery [7] pre asynchrónne volania. Ide o volania, pri ktorých nie je nutné znova načítavať stránku. Volania sa dejú „na pozadí“. jQuery umožnuje reagovať na udalosti v aplikácií, napríklad stlačenie tlačidla alebo odkazu, a vykonať tak vlastnú funkciu, napríklad umožniť uloženie formulára ak sa niektoré pole zmení.

2.1.2 Backend

Opakom Frontendu je anglický termín *Backend* [8]. Ten v sebe zahŕňa serverovú časť stránky či aplikácie, ktorú používateľ danej aplikácie, iným slovom aj klient, nevidí a nemôže s ňou priamo interagovať. Zahŕňa v sebe použitie rôznych API (aplikačné rozhranie – *Application Programming Inteface*), knižníc a prepojení medzi jednotlivými službami, napr. obsluhu prijatých požiadaviek a komunikáciu s databázou. Medzi najviac používané programovacie jazyky pre oblasť Backendu webových aplikácií možno zaradiť jazyky PHP, Python, Java, C++ či JavaScript (ako framework Node.js). Pre spomínanú komunikáciu s databázou sú dva najčastejšie používané

prístupy a to tzv. priamy prístup, pri ktorom developer aplikácie komunikuje s databázou posielaním databázových príkazov, alebo pomocou tzv. *ORM* nástrojov (*Object Relational Mapping*), ktoré zabezpečujú vrstvu komunikácie medzi aplikáciou a samotnou databázou.

Laravel

Základným prvkom Backendu vyvíjanej aplikácie je framework jazyka PHP nazývaný Laravel [9]. Z dôvodov pre výber tohto frameworku možno spomenúť fakt, že Laravel patrí medzi najpoužívanejšie frameworky na tvorbu webových aplikácií, podporuje jednoduchý a rýchly vývoj vďaka množstvu už hotových komponentov, je open source a poskytuje ORM nástroj pre komunikáciu so zvolenou SQL databázou. Framework stavia na overenom návrhovom vzore Model-View-Controller, skrátene MVC.

MySQL

Jednou z najznámenších a najrozšírenejších SQL (*Structured Query Language*) databáz je databáza MySQL [10]. Jej popularita je založená na otvorenej licencii vhodnej pre rôzne typy projektov, jednoduché nasadenie i použitie a taktiež aj pre prispievanie veľkých technologických spoločností, využívajúcich túto databázu (napr. Facebook, Paypal, LinkedIn a iné). Podporuje štandardný jazyk SQL a na jednoduchšie spravovanie a náhľad nad dátami uloženými v databáze možno použiť viacero grafických editorov, napr. phpMyAdmin či MySQL Workbench.

Návrhový vzor

Keďže podľa zadania má samotná aplikácia podporovať zobrazovanie dát používateľovi, ukladanie do databázy a vytváranie reportov, je vhodné použiť štandardný návrhový vzor pre vývoj takejto aplikácie. Tento fakt sa odrazil už pri výbere frameworku pre výber backendu tak, aby zvolený framework podporoval najčastejšie používaný, a pre túto aplikáciu vhodný, už spomenutý návrhový vzor MVC [11]. Príbuzné návrhové vzory, ktoré je tiež možné použiť, sú napríklad Model-View-Adapter (MVA) či Model-View-Presenter (MVP). Ich výhodou je presnejšia definícia, ktorá je menej závislá od konkrétnej implementácie v danom frameworku. Nejedná sa však o zložitú aplikáciu, v ktorej by bolo nutné uplatniť výhody bezstavového prístupu týchto príbuzných návrhových vzorov.

Kombinácia zvoleného návrhového vzoru podporovaného vybraným frameworkom naznačuje spôsob fungovania aplikácie. Z modelu MVC vyplývajú vzťahy reprezentované medzi kontrolérom a ďalšími dvomi prvkami modelu [12]. Príkladom

môže byť popis situácie, v ktorej si používateľ aplikácie vyžiada od aplikácie report o svojom testovaní. Pohľad, angl. view (klientská časť aplikácie), na ktorej používateľ zadal požiadavku na report, odošle popis tejto požiadavky s potrebnými dátami kontroléru (serverová časť aplikácie). Kontrolér spracuje požiadavku a vytvorí požiadavku na databázový model, od ktorého následne dostáva dáta. Prijaté dáta kontrolér zasiela klientskej časti aplikácie a používateľ dostáva dáta vo forme reportu.

2.1.3 Docker

Docker[1] je platforma pre vývoj, spúšťanie a zdieľanie tzv. „kontajnerov“, v ktorých beží aplikácia alebo iná služba [13]. Kontajnery majú viacero výhod. Sú dostatočne flexibilné, aby v nich mohla bežať aj komplikovaná služba a je ich možno použiť takmer vždy. Na rozdiel od virtuálneho zariadenia sú kontajnery efektívnejšie vo využívaní systémových zdrojov a nezaberajú toľko miesta na disku a pamäte, ako klasické virtuálne zariadenie. Na platforme Linux dokáže Docker kontajner zdieľať kernel medzi hostiteľom a kontajnerom.

Zdieľanie kontajnerov je tiež veľký benefit. Často je nutné vytvorenú aplikáciu odoslať inej osobe za účelom napríklad kontroly alebo podobne. Táto PHP aplikácia, ktorá je cieľom tejto práce vyžaduje pre svoju funkcionalitu viacero služieb ako PHP, webový server, databázový server a podobne. Samotná aplikácia sa skladá z viacerých priečinkov a súborov, nejde teda o jednoduchý .exe súbor. Ak takúto aplikáciu vložíme do kontajnera, ten je možné ľahko zdieľať (buď fyzickým odoslaním kontajnera zabaleného v archíve alebo pomocou Docker Hubu[2], ktorý slúži na zdieľanie kontajnerov medzi užívateľmi). Druhá osoba tak nemusí mať vlastné vývojové prostredie a rovnaké verzie PHP, webového servera a podobne. Stačí spustiť kontajner, v ktorom beží daná aplikácia, vrátane všetkých služieb, ktoré potrebuje. Obe strany tak majú rovnaké prostredie a je možné vyhnúť sa problémom typu „na mojom stroji to nefunguje.“

Pri použití v dátových centrách alebo pri masívnom nasadení jednej aplikácie na viacero fyzických strojov (ak je nutné aby aplikácia bežala na viacerých zariadeniach zároveň) poskytujú kontajnery dobrú škálovateľnosť, podobne ako virtuálne zariadenia. Jednotlivé kontajnery sú oddelené a izolované od seba.

Platforma Docker bola vybraná pre tento projekt preto, lebo poskytuje ľahkú distribúciu výslednej aplikácie pomocou Docker Hubu a oproti virtuálnemu zariadeniu je menej náročná na systémové zdroje. Taktiež inštalácia samotnej aplikácia je

[1] https://www.docker.com/
[2] https://hub.docker.com/

značne jednoduchšia a je možné ju jednoducho a rýchlo nasadiť na veľké množstvo zariadení.

2.2 Návrh backendu

Ide o aplikáciu na podporu manuálneho penetračného testovania webových aplikácií. Aplikácia musí zvládať vytvárať projekty a priradiť k nim kontrolný zoznam testovacích prípadov (podľa metodológie OWASP Testing Guide v4 a OWASP ASVS). Do projektu bude možné zadávať zoznam nájdených stránok. Podobne tak bude možné vkladať formuláre. K formulárom aj k projektu bude automaticky vygenerovaný kontrolný zoznam podľa metodológie ASVS. Aplikácia bude testerovi zobrazovať postup a dosiahnuté výsledky, ktoré budú na záver testovania prezentované formou vygenerovaného reportu.

Projekt

Projekt bude základným prvkom aplikácie. Bude obsahovať informácie o samotnej testovanej aplikácii, o klientovi, pre ktorého je test realizovaný, aj informácie ohľadom projektu (dátum začatia/ukončenia testovania, počet predpokladaných dní,...). K projektu budú viazané jednotlivé nájdené stránky, kontrolné zoznamy a všetko, s čím bude aplikácia pracovať.

Projekt bude obsahovať svoj vlastný kontrolný zoznam. Pôjde o testy, ktoré nespadajú pod žiaden špecifický vstup alebo formulár, ale sú pre celú aplikáciu globálne (najmä autentizácia a manažment relácií).

Aplikácia umožní vytvoriť viacero užívateľov. Každý projekt bude viazaný na konkrétneho užívateľa. Podobne bude možné priradiť k projektu klienta. V praxi je bežné, že tester vykonáva viacero projektov pre jedného klienta, preto by mal byť vytvorený objekt *Klient*.

Stránka

Stránka sa viaže k projektu a slúži na reprezentáciu jednej stránky v aplikácií. K stránke sa budú viazať jednotlivé vstupy a formuláre, ktoré obsahujú kontrolné zoznamy. Stránka sama o sebe obsahuje parametre URL a názov.

Formulár

Formulár je viazaný na stránku a obsahuje kontrolný zoznam aj dodatočné vstupy. Obsahuje tiež názov, cieľové URL a popis.

Vstup

Formulár obsahuje viacero vstupov, každý z nich musí obsahovať názov a typ (číslo, dátum, atď.). Vstupy obsahujú samotné testy, ktoré sa na nich musia vykonať. Musí teda existovať mapovanie, ktoré podľa typu vstupu určí, ktoré testy je nutné realizovať. Táto časť je automatizovaná. Štruktúra testu by mala zodpovedať štruktúre OWASP ASVS, niektoré testy sú však vynechané, nakoľko ide o auditné testy, pri ktorých nie je možné ich vykonanie na diaľku (napr. je nutná súčinnosť vývojárov).

Kontrolný zoznam

Ide o zoskupenie testov, ktoré boli priradené ku konkrétnemu vstupu. Tento zoznam je vytváraný automaticky pri vytvorení vstupu, projektu alebo formulára. Test sa priradzuje (Testing Guide, ASVS) ku konkrétnemu vstupu pokiaľ má so vstupom súvis. Toto mapovanie bude predvytvorené v aplikácií a uložené v databáze. Test obsahuje mieru zraniteľnosti (informačná, nízka,...). Tester má taktiež možnosť k testu napísať vlastný popis, napríklad ako zraniteľnosť opraviť alebo ako bola objavená.

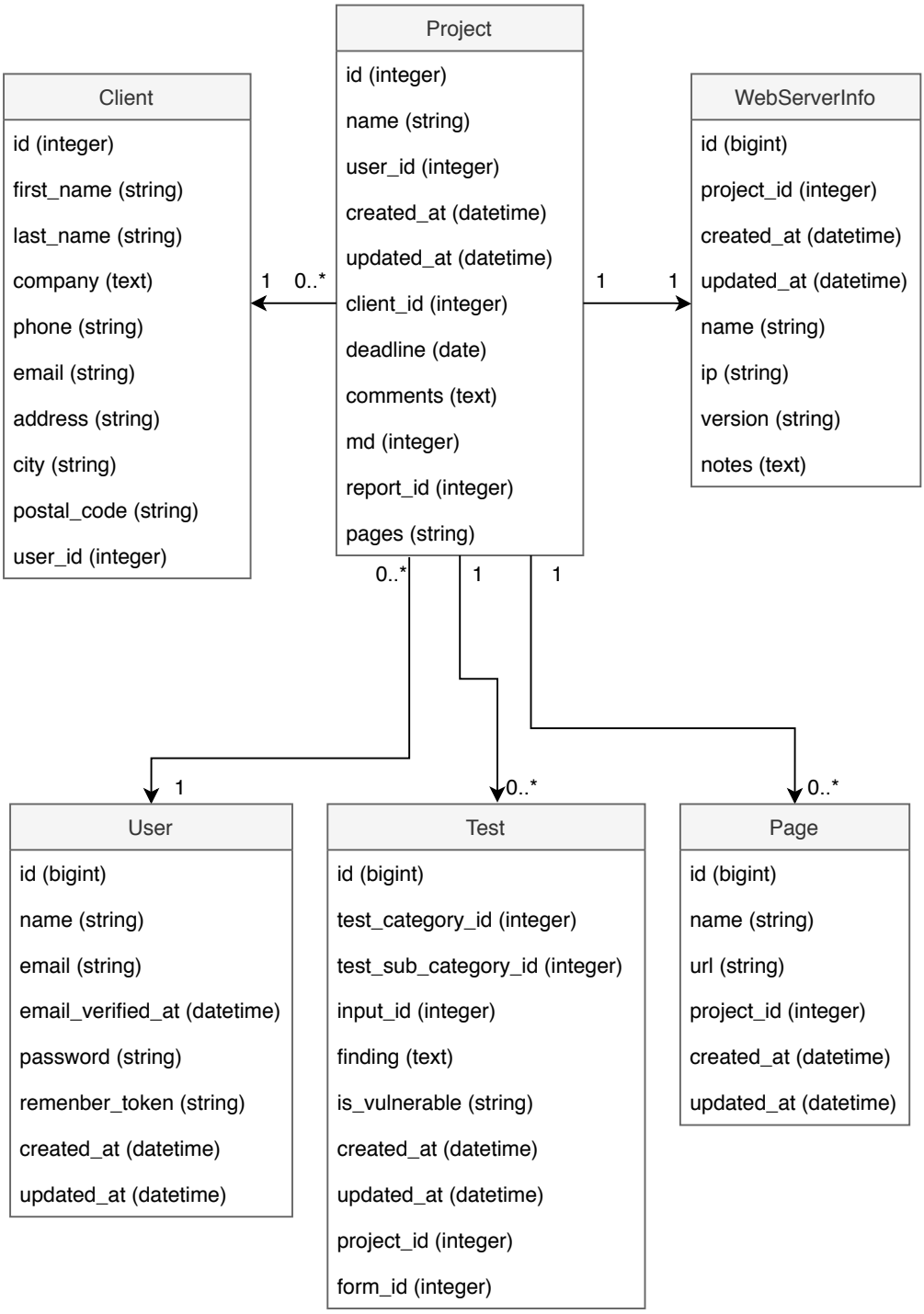
2.3 Databáza

Ako databáza sa používa MySQL. Pre implementáciu celkovej funkcionality je nutné vytvoriť viacero tabuliek, niektoré budú slúžiť ako pomocné tabuľky, iné budú primárne využívané aplikáciou. Návrh databázy je možné vidieť na obrázkoch 2.2 a 2.3. Tento dizajn umožňuje (pomocou vzťahov) pohybovanie sa medzi objektami (tzn. z objektu *Page* je možné sa dostať k jednotlivým formulárom a následne k testom pre konkrétny formulár a podobne).

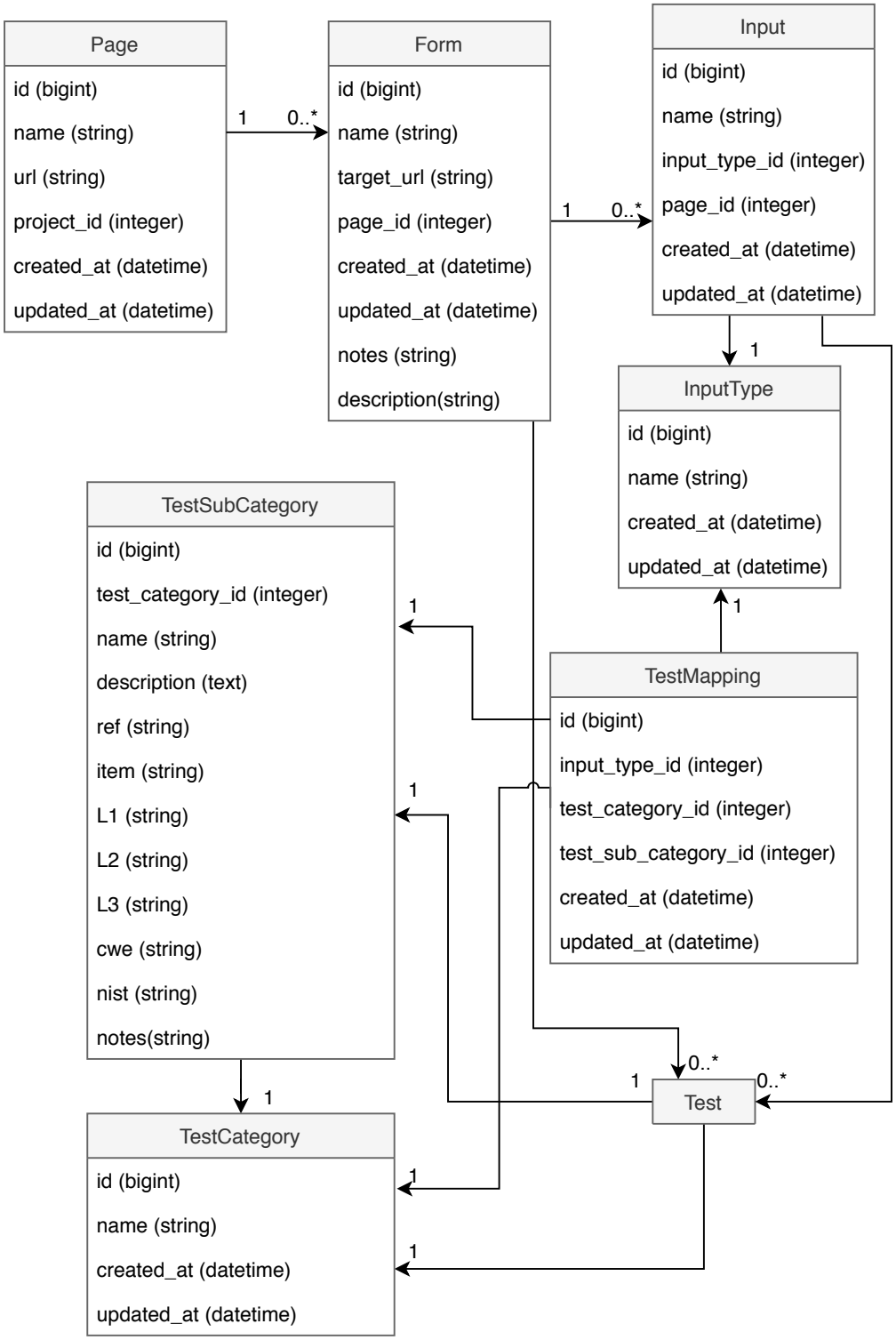
Väzby medzi objektami zabezpečuje modul Object-relational-mapping (ORM) priamo v Laraveli (Eloquent). Vďaka nemu je možné vytvárať väzby jednoducho. Návrh databázy značne využíva funkcionalitu cudzích kľúčov (foreign-key). Nie je však vynútená na úrovni databázy (neexistuje žiadne obmedzenie *constraint*, funkcionalitu cudzích kľúčov zabezpečuje priamo Eloquent).

Druhou možnosťou (bez využitia ORM) je použitie klasických SQL príkazov pre prácu s databázou. Toto riešenie však nie je vhodné, keďže využitie ORM je bezpečnejšie, jednoduchšie aj praktickejšie do budúcnosti.

Jednotlivé objekty (modely) obsahujú pomocné funkcie pre prácu s nimi. Napríklad model Projekt vie zistiť štatistiku všetkých testov, ktoré pod projekt patria.



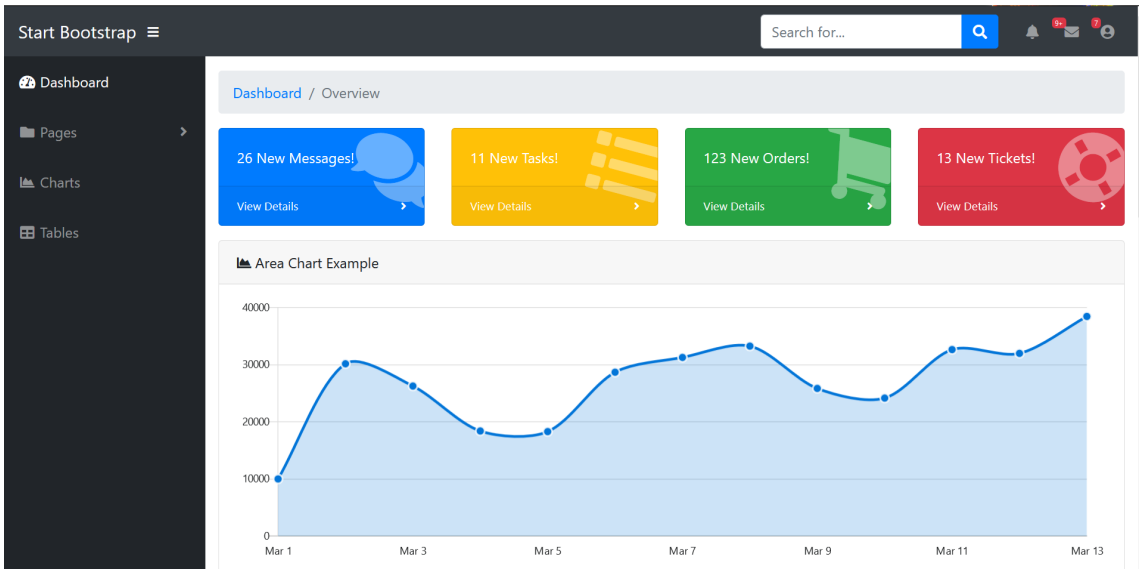
Obr. 2.2: UML diagram



Obr. 2.3: UML diagram – pokračovanie

2.4 Návrh frontendu

Grafické rozhranie (GUI) je inšpirované bezplatne dostupným riešením „SB Admin“[3]. Pre využitie v Laravel projekte bude nevyhnutné jednotlivé časti rozdeliť na menšie celky, pomocou ktorých bude GUI poskladané. Prevzaté časti budú najmä skripty (bootstrap aj skripty prispôsobené pre danú tému) a navigačná lišta. Samotný vzhľad obsahu bude prispôsobený na mieru aplikácii. Vzor GUI (bez úprav) je vidieť na obrázku 2.4. Centrálna oblasť (kde sú aktuálne vidieť karty a graf) bude upravená pre potreby aplikácie. Zvyšná časť bude prevziatá, ale taktiež prispôsobená aplikácií. Odstráni sa vyhľadávanie a zmenia sa navigačné položky v menu.



Obr. 2.4: Vzor GUI

Laravel využíva „blade“ súbory (nazov.blade.php). Rozdelením GUI na dve časti bude získaná modularita dizajnu. Prvou časťou bude hlavná navigačná lišta (menu v ľavej časti obrazovky) a horná lišta, ktorá zabezpečuje spravovanie užívateľa (napr. odhlásenie). Druhou časťou bude samotný obsah, ktorý bude závisieť priamo na aplikácií. Bude použitý Bootstrap s dôrazom na prehľadnosť. Využité budú uzatváracie prvky (collapsible) pre zlepšeniu prehľadnosti. Vďaka využitiu Bootstrapu bude aplikácia plne responzívna a graficky prívetivá.

[3]https://startbootstrap.com/templates/sb-admin/

3 Vlastná implementácia webovej aplikácie

V kapitole o implementácii bude prezentované, akým spôsobom bola dosiahnutá funkcionalita jednotlivých častí a následne spojená do jedného funkčného celku.

Príprava prostredia

Pred začatím samotných prác na aplikácií je potrebné pripraviť prostredie. V tomto prípade ide o inštaláciu najnovšej vezie PHP, MySQL a Apache. Použitá bude platforma Windows 10, preto je vhodné použiť program XAMPP, ktorý obsahuje všetky potrebné aplikácie. Doinštalovať bude potrebné aj nástroj *composer*. Inštalácia, tak aj konfigurácia jednotlivých programov je štandardná, nie je potrebné robiť žiadne špecifické úkony, preto detailne popísaná nebude.

3.1 Implementácia backendu

Vytvorenie projektu

Laravel využíva na vytvorenie projektu program *composer*. Pomocou neho je možné vytvoriť vzorový Laravel projekt s predpripravenou štruktúrou a priečinkami. Vzorový projekt je vytvorený príkazom zobrazeným vo výpise 6.

```
1 composer create-project --prefer-dist laravel/laravel blog
```

Výpis 6: Vytvorenie počiatočného Laravel projektu

Na samotnom projekte nezáleží, funkcionalita bude odstránená, avšak štruktúra súborov a ich konfigurácia je vhodným počiatočným bodom práce na aplikácií.

Štruktúra Laravel projektu obsahuje veľké množstvo priečinkov a súborov. Podstatné súbory a priečinky sú:

- app/ – východiskové miesto pre PHP triedy/objekty
- app/Http/Controllers – obsahuje triedy (kontroléry – podľa návrhového vzoru Model-View-Controller), ktoré určujú aká funkcionalita sa má vykonať na konkrétnej URL
- config/ – konfigurácia aplikácie, napr. nastavenie mena a hesla pre pripojenie do databázy
- database/migrations – priestor pre migrácie
- public/ – verejne dostupný priestor, slúži na uloženie potrebných CSS a JS súborov (prípadne obrázkov) a podobne

- routes/web.php – v tomto súbore sa konfigurujú cesty v aplikácií a priraďuje sa im funkcionalita alebo kontrolér
- resources/view – priečinok pre súbory GUI (views)

Model

Model je v princípe objekt z pohľadu aplikácie. V tomto prípade bude ako model každá trieda, ktorá reprezentuje funkcionalitu v aplikácií (napr. Projekt, Stránka, Test,...).

Pre vytváranie modelov (tried/objektov) je vhodné použiť nástroj *artisan*, ktorý vytvorí daný model, umiestni ho do správnej zložky a dá mu správnu štruktúru. Model môže byť vytvorený ako *resource*. V prípade že model bude vytvorený ako resource, môže mu byť predpripravená štruktúra kontroléru a aj mapovanie ciest. Východiskový stav vygeneruje metódy pre:

- zobrazenie zoznamu modelu (napríklad stránka, ktorá zobrazuje všetky projekty)
- zobrazenie formulára pre vytvorenie nového objektu (nový projekt)
- samotné vytvorenie nového objektu, po odoslaní formulára
- zobrazenie konkrétneho objektu
- zobrazenie formulára pre úpravu objektu
- aktualizácia objektu (odoslaním formulára s úpravou objektu)
- zmazanie objektu

Všetky metódy budú mať správny tvar (budú obsahovať potrebné parametre ako *id* alebo objekty typu *Request* pri práci s HTTP POST metódou). Ide teda o zjednodušenie práce. Mapovanie resource objektov je taktiež jednoduché. Nie je potrebné mapovať každú URL na špecifickú metódu ale stačí použiť volanie Route::resource(), ktoré automaticky namapuje potrebné URL na predpripravený kontrolér. Vzor takejto konfigurácie je viditeľný vo výpise 7. Táto konfigurácia vytvorí mapovanie zobrazené v tabuľke 3.1 (parameter *project* označuje ID konkrétneho projektu).

```
1 Route::resource("projects", "ProjectsController");
```

Výpis 7: Mapovanie resource modelu na kontrolér

3.2 Migrácie v databáze

Laravel umožňuje v databáze vytváranie tabuliek aj ich editovanie pomocou migrácie. Samotná migrácia obsahuje dvojakú funkcionalitu. Prvá funkcionalita sa spustí

Tab. 3.1: Automaticky vygenerované cesty

HTTP metóda	**URL**	**Vysvetlenie**
POST	projects	Vytvorenie nového projektu
GET/HEAD	projects	Zobrazenie všetkých projektov
GET/HEAD	projects/create	Zobrazenie formulára pre vytvorenie nového projektu
GET/HEAD	projects/{project}	Zobrazenie konkrétneho projektu (pomocou id – {project})
PUT/PATCH	projects/{project}	Vytvorenie nového projektu
DELETE	projects/{project}	Zmazanie projektu
GET/HEAD	projects/{project}/edit	Zobrazenie formulára pre úpravu projektu

pri vykonaní migrácie, druhá pri jej odstránení (rollback). Vzor migrácie pre vytvorenie tabuľky *projects* je vidieť vo výpise 8 (samotnú migráciu je možné vytvoriť prípazom „php artisan make:migration nazov“). Spustenie všetkých (doposiaľ nespustených) migrácií je možné príkazom „php artisan migrate“. Pomocou migrácií sa aj do tabuliek dopĺňajú dodatočné stĺpce, na ktoré sa pri vytváraní tabuľky zabudlo.

3.3 Implementácia frontendu

Implementácia grafického rozhrania začala stiahnutím šablóny a umiestnením statických súborov do správnej zložky (public). Počiatočne bol vytvorený súbor hlavného rozloženia (layout.blade.php), ktorý obsahuje všetky potrebné CSS a JS súbory (buď mapované zo zložky public alebo cez CDN). Tento súbor obsahuje navigačné menu.

Jednotlivá funkcionalita aplikácie obsahuje vlastné súbory (napr. view súbor projects.blade.php), ktorý obsahuje iba kód špecifický pre danú funkcionalitu. Tento súbor „dedí“ zo súboru layout.blade.php a vypĺňa iba jeho obsah. Hlavný súbor layout.blade.php má definovanú sekciu, kam sa má vložiť súbor s obsahom (pomo-

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateProjectsTable extends Migration
{
    public function up()
    {
        Schema::create('projects', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->integer("user_id");
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('projects');
    }
}
```

Výpis 8: Vzorová migrácia pre vytvorenie tabuľky „projects“

cou direktívy *@yield("content")*. Vzorový súbor pre zobrazenie všetkých projektov projects.blade.php je možné vidieť vo výpise 31 (príloha A). Podstatnou časťou sú direktívy ako *@foreach* a využívanie premenných (ktoré sú do view zaslané z kontroléra).

3.4 Vzorová implementácia funkcionality

Pre ukážku implementácie funkcionality bola vybraná funkcionalita stránok (Page), teda objektu, ktorý reprezentuje jednu stránku v testovanej aplikácií. Ostatné objekty sú implementované podobne.

Na začiatku sa vytvorí model (Page.php) pomocou artisan príkazu „php artisan make:model -a“ (parameter -a vytvorí model ako resource, vytvorí kontrolér aj migráciu. Prvým krokom je naplnenie migrácie, do ktorej vložíme potrebné stĺpce (výpis 9).

Po spustení migrácie je v databáze vytvorená tabuľka „pages“ so zadanými stĺpcami. V prípade potreby je možné vytvoriť dodatočnú migráciu, ktorá by doplnila potrebné stĺpce do tabuľky, prípadne by ich odstránila alebo zmenila. V migrácií je

```php
public function up(){
        Schema::create('pages', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('name');
        $table->string('url');
        $table->unsignedInteger('project_id')->nullable(true);
        $table->timestamps();
    });
}
```

Výpis 9: Migrácie pre vytvorenie tabuľky „pages“

špecifikovaný stĺpec „project_id“, ktorý bude slúžiť na spárovanie stránky s konkrétnym projektom.

Samotný model, ktorý bol vytvorený, je prázdny a neobsahuje žiadnu funkcionalitu. Pre naše potreby musíme vytvoriť vzťah medzi stránkou a vstupom. Z pohľadu aplikácie sú vstupy (Input) viazané na formulár (tabuľka „forms“ obsahuje stĺpec „form_id“, ktorý slúži ako cudzí kľúč, názvoslovie vychádza z východiskovej konfigurácie ORM modulu Eloquent). V modeli Page musíme vytvoriť samotný vzťah aby bolo možné cez premennú typu Page pristúpiť ku všetkým formulárom (Form), ktoré sú viazané k danej stránke. Podstatným faktom je, že jedna stránka má viacero formulárov. Z pohľadu Eloquent pôjde o vzťah „hasMany“. Implementácia vzťahu je veľmi jednoduchá a stačí pridať verejnú metódu do objektu Page (výpis 10).

```php
public function forms() {
    return $this->hasMany(Form::class);
}
```

Výpis 10: Implementácia vzťahu medzi objektami Page a Form

Ďalším krokom je namapovanie URL adries na vytvorený kontrolér. Keďže stránka patrí pod projekty, malo by to byť reflektované aj v adrese URL. Do súboru routes/web.php sa vloží mapovanie „Route::resource('projects.pages', 'PagesController');“. Vďaka tomu budú URL adresy namapované na správne metódy v kontroléri a budú hierarchicky pod projektami (napríklad URL projects/project/pages). Podstatným poznatkom je, že v tejto konfigurácií je nutné pracovať aj s parametrom project (teda ID projektu). Toto je nutné zohľadniť v kontroléri (pridaním premennej do jednotlivých metód).

Implementácia kontroléru spočíva v doplnení kódu k jednotlivým metódam,

ktorý vykoná potrebnú funkcionalitu a navráti grafické rozhranie (view). Prvou metódou na implementovanie je „index“, ktorá má za úlohu zobraziť všetky stránky v projekte (výpis 11). Do funkcie vstupuje ako premenná $project_id, ktorá obsahuje ID aktuálneho projektu, pomocou neho vytiahneme daný projekt z databázy (využitý bude v GUI). Pomocou ID projektu tiež získame všetky stránky (Pages), ktoré patria k projektu (riadok 4). Na záver je vrátené „view“ s premennými *project* a *pages*, ktoré budú prístupne priamo v showPages.blade.php súbore.

```
1  public function index($project_id){
2      $project = Project::find($project_id);
3      $pages = Page::where("project_id", $project_id)->orderBy("url",
       ↪  "DESC")->get();
4
5      return view("showPages")->with("project", $project)->with("pages", $pages);
6  }
```

Výpis 11: Implementácie metódy index

Metódy „create“, „show“ a „edit“ sú obsahovo podobné, ich cieľom je zobraziť daný objekt alebo formulár (vytvorenie/úprava). Ich funkcionalita je podobná index metóde. Ďalšou metódou bude metóda „store“ (výpis 12), ktorá spracuje formulár pre vytvorenie nového projektu. Do metódy vstupuje premenná *request* typu *Request*. Okrem iného obsahuje parametre tela POST požiadavky, zaslanej z front-endu na back-end. V tejto metóde sa vytvorí nová inštancia objektu Page a priradia sa jej hodnoty z formulára. Volaním metódy *save()* na objekte sa samotný objekt uloží do databázy. Výsledkom metódy je presmerovanie na zobrazenie všetkých stránok.

```
1  public function store($project_id, Request $request){
2      $page = new Page();
3      $page->name = $request->name;
4      $page->url = $request->url;
5      $page->project_id = $project_id;
6      $page->save();
7      return redirect("/projects/$project_id/pages");
8  }
```

Výpis 12: Implementácie metódy store

Predposledne spomenutou je metóda „update“, ktorá spracuje formulár po zmene objektu (edit). Jej implementácia je zhodná s metódou store, až na hodnotu *project_id*, ktorá sa pri úprave nemení (nedá sa zmeniť z front-endu). Metóda „destroy“

(pre zmazanie objektu) je tiež veľmi jednoduchá. Stačí nájsť danú stránku podľa jej ID a zavolať metódu *delete()* na daný objekt. Tým dôjde k odstráneniu objektu z databázy.

Implementácia grafického rozhrania spočíva v návrhu užívateľsky prívetivého dizajnu, ktorý vhodne prezentuje jednotlivé objekty a uľahčí užívateľovi prácu s nimi. Jednotlivé metódy v kontroléri majú vlastné grafické zobrazenie (súbory .blade.php). Ich obsahom je prevažne HTML štruktúra (prípadne formulár), ktorá zobrazuje daný objekt. Pre prácu s formulármi sa využíva balík „HTML“ od Laravel Collective[1]. Ide o bývalú implementáciu formulárov, avšak v najnovšej verzií už nie je zakomponovaný priamo v Laraveli a je nutné ju pomocou nástroja composer doinštalovať manuálne.

3.4.1 Zobrazenie stránky (Page)

Zobrazenie stránky (model Page) obsahuje najviac funkcionality, nakoľko je zobrazená samotná stránka, formuláre k stránke viazané a ich testy, a vstupy a ich testy. V hornej časti obrazovky sú zobrazené informácie o samotnej stránke a je možné ich aktualizovať

Vytvorenie nového formulára prebieha kliknutím na tlačidlo „New Form“. Otvorí sa modálne (pop-up) okno s formulárom pre vytvorenie nového formulára. Vyplnením formulára a jeho odoslaním sa vytvorí samotný formulár a zároveň sa vygenerujú testy pre daný formulár.

Zaujímavejšou časťou je však vytváranie vstupov (Input). Podobne ako pri vytvorení formulára sa po kliknutí na tlačidlo pridania vstupu otvorí modálne okno s formulárom, podobne ako pri vytváraní formulára. Problém nastáva pri priradení vstupu k formuláru (vstup je viazaný na formulár, preto je nutné pri vytváraní vstupu zaslať na server aj ID formulára, ku ktorému bude patriť). Formulár je iba jeden, nemôže teda mať staticky nastavený ID parameter pre všetky formuláre. Riešením je využiť skrytý parameter. Po kliknutí tlačidla, ktoré obsahuje ako parameter ID formulára, ku ktorému patrí, sa aktualizuje skrytý parameter s názvom _formId. Ten je súčasťou modálneho formulára. Vo východiskovom nastavení je hodnota parametra _formId prázdna.

Tento princíp je znázornený vo výpise 13. Tlačidlo obsahuje parameter „data-id“, ktorý je definovaný hodnotou „$form->id“. Ide o zápis v blade šablóne a táto hodnota bude nahradená ID daného formulára, ku ktorému tlačidlo patrí. Vo východiskovom nastavení je hodnota parametra _formId prázdna (riadok 5). Tlačidlo obsahuje triedu „new-input-button“, táto trieda má definovanú funkciu na udalosť

[1] https://laravelcollective.com/docs/6.0/html

„click“ pomocou jQuery skriptu (riadok 9-10). Táto funkcia sa realizuje pri každom stlačení každého tlačidla s touto triedou. Funkcia vezme voliteľný parameter „id“ z tlačidla (označený ako „data-id“) a nastaví ňou hodnotu parametra _formId pomocou ID modálneho formulára, ktorý je „newInputForm“. Táto štruktúra zabezpečí, že pri vytváraní nového vstupu sa vždy aktualizuje hodnota _formId na správnu, podľa toho, ktoré tlačidlo bolo stlačené. Každé tlačidlo obsahuje hodnotu data-id podľa formulára, ku ktorému patrí (všetky formuláre sa prechádzajú pomocou foreach funkcie).

```
1  <button type="button" class="new-input-button" data-id="{{$form->id}}">
2      <i class="fa fa-plus"></i>
3  </button>
4  ...
5  <input name="_formId" type="hidden" value="">
6  ...
7  <script>
8  $(document).ready(function() {
9      $('.new-input-button').on('click', function () {
10         $('#newInputForm').find('input[name="_formId"]').val($(this).data('id'))
11     });
12 });
13 </script>
```

Výpis 13: Aktualizácia _formId parametru.

Samotné vytvorenie vstupu je asynchrónne a vykonáva sa pomocou jQuery volania (AJAX). Odoslanie modálneho formulára je viazané na jQuery funkciu (výpis 14). Ako prvé (riadok 2) je zablokovaná východisková funkcionalita tlačidla. Ide o preventívne zablokovanie, aby sa spustila táto funkcia a nie odoslanie formulára, ak by mal nastavenú akciu. Ďalej je nastavená potrebná hlavička (X-XSRF-Token), ktorá obsahuje hodnotu anti-CSRF tokenu, ktorý generuje Laravel. Nasleduje konfigurácia ajaxového volania (riadok 8). Táto konfigurácia obsahuje URL, na ktorú má volanie smerovať, HTTP metódu (v tomto prípade PUT), a dáta v tele požiadavky (ide o dáta z formulára). Na záver sú definované dve funkcie, prvá (success) sa vykoná v prípade, ak bude požiadavka úspešne vykonaná a druhá (error), ak dôjde k chybe z ľubovoľného dôvodu.

Ak volanie dopadne úspešne, bude zobrazená potvrdzujúca hláška (plugin toastr[2]) a zároveň je vstup automaticky vložený na začiatok k formuláru (riadok 15-17). Toto vloženie je však nepraktické, nakoľko musí byť v jQuery metóde vložený

[2]https://github.com/CodeSeven/toastr

```
1  $('#newInputForm').on('submit',function(event){
2      event.preventDefault();
3      $.ajaxSetup({
4          headers: {
5              'X-XSRF-Token': "{{ csrf_token() }}"
6          }
7      });
8      $.ajax({
9          url: "{{route('projects.pages.inputs.storeAjax', [$project->id,
           ↪  $page->id])}}",
10         type:"PUT",
11         data: $("#newInputForm").serialize(),
12         success:function(response){
13             toastr.success('Success');
14             var $iId = response.id;
15             $("#collapseForm" + $('#newInputForm')
16                 .find('input[name="_formId"]').val())
17                 .prepend("<form>...");
18             $("#collapseForm" + $('#newInputForm')
19                 .find('input[name="_formId"]').val() + " select")
20                 .first().val($('#newInputForm')
21                 .find('select[name="type"]').val()).change();
22             $("#closeModal").trigger("click");
23         },
24         error:function(response){
25             toastr.error('Error!');
26         },
27     });
28 });
```

Výpis 14: Asynchrónne pridanie vstupu pomocou jQuery.

všetok potrebný HTML kód, ktorý často vzniká aj na strane servera pomocou blade šablóny (tento kód nie je vo výpise, nakoľko je príliš dlhý). Vo výsledku sa tak kombinuje javascript kód s PHP kódom a nie je to praktické ani prehľadné. Práve pre tento dôvod nie je zvyšok funkcionality pridania stránok, formulárov, vstupov, atď. asynchrónny. Riadok 18-21 zabezpečí aby mal novo pridaný vstup správnu hodnotu jeho typu a riadok 22 zatvorí modálne okno (formulár). Pri neúspešnom volaní sa užívateľovi zobrazí chybová hláška.

Funkcie jQuery sa tiež využívajú pri inej funkcionalite. Napríklad umožnenie uloženia vstupu. Tlačidlo uloženia je vo východiskovom stave zablokované a odblokuje sa v prípade, ak by bola detegovaná zmena niektorého z polí. Táto funkcionalita je

viditeľná vo výpise 15. V prípade, že je detegovaná zmena (udalosť „keyup“ alebo „change“), odblokuje sa tlačidlo na uloženie daného vstupu/formulára.

```
$('.formCheck').on('keyup', function () {
    $('#formSave' + $(this).data('id')).prop('disabled', false);
});

$('.inputForm').on('change keyup', function () {
    $('#inputSaveButton' + $(this).data('id')).prop('disabled', false);
});
```

Výpis 15: Kontrola na zmenu údajov formulára/vstupu.

3.4.2 Monitorovanie stavu postupu (progress bar)

Pri globálnych testoch (zobrazenie projektu) sa zobrazuje progress bar s graficky znázorneným postupom pri testoch. Pri tomto zobrazení sa berú do úvahy všetky testy, ktoré spadajú pod projekt, vrátane testov pri jednotlivých formulároch a vstupoch. Jednotlivé úrovne zraniteľností sú farebne odlíšené. Ukážka kódu je vo výpise 16.

```
<div class="progress">
<div class="progress-bar ..." role="progressbar" style="width:
↪  {{$project->testStats(0)}}%" aria-valuenow="{{$project->testStats(0)}}"
↪  aria-valuemin="0" aria-valuemax="100">N {{$project->testStats(0)}}%</div>
...
</div>
```

Výpis 16: Progress bar.

Vďaka farebnému progress baru tester vždy vie, koľko percent testov už absolvoval, vidí ich graficky zatriedené podľa závažnosti a tiež je vidieť, koľko percent testov ešte chýba dotestovať. Ukážka je zobrazená na obŕazku 3.1.

Obr. 3.1: Ukážka progress bar.

3.4.3 Mazanie a čistenie objektov

Tým, že jednotlivé objekty obsahujú viacero podobjektov (napr. formulár obsahuje viacero vstupov alebo testov, vstupy znova obsahujú testy apod.) dochádza k stavu, kedy pri zmazaní rodiča sa musia zmazať aj deti (potomkovia). Napríklad, pri zmazaní projektu je nutné zmazať všetky stránky, formuláre, testy k formulárom, vstupy a testy k týmto vstupom. Ak by sa tak nestalo, v databáze by zostávali „zombie“ záznamy, ktoré už neexistujú ale zaberajú miesto a zdroje.

Riešenie je možné v dvoch úrovniach. Prvá možnosť je na úrovni MySQL databázy s využitím funkcie „ON DELETE CASCADE“. Pre použitie tejto funkcie je však nutné mať správne nakonfigurované cudzie kľúče na úrovni databázy. Laravel túto konfiguráciu nevytvára automaticky. Z toho dôvodu bolo zvolené druhé riešenie a tým je zmazanie detí na úrovni Laravel aplikácie v jednotlivých modeloch.

Pre zmazanie potomkov sa využíva špeciálna metóda „boot()“, ktorá je volaná Laravel frameworkom po načítaní všetkých služieb a udalostí. V tejto metóde je možné vytvoriť funkcie, ktoré budú reagovať na danú udalosť. V tomto prípade pôjde o udalosť zmazania objektu („deleted())“). Kód je možné vidieť vo výpise 17.

```
1  public static function boot()
2  {
3      parent::boot();
4      static::deleted(function($project)
5      {
6          $project->pages()->delete();
7          $project->webInfo()->delete();
8          $project->tests()->delete();
9      });
10 }
```

Výpis 17: Zmazanie potomkov pri modely Project

Na začiatku sa zavolá boot metóda nadradeného objektu. Následne sa nastavuje vlastná funkcia, ktorá sa vykoná ak nastane udalosť „zmazanie“ (tá nastáva pri zmazaní objektu). Táto funkcia zmaže všetky stránky, ktoré patria k projektu aj všetky testy, viazané na projekt (testy viazané na formuláre alebo vstupy táto funkcia nezmaže). Odstránený je tiež objekt WebServerInfo.

Pre zmazanie formulárov, vstupov a ďalších objektov, ktoré spadajú pod projekt, sa implementuje podobná forma tejto funkcie do všetkých objektov. Pri zmazaní objektu sa zmažú stránky, čím sa spustí táto funkcia v modeli Page, zmažú sa formuláre atď. Každý objekt teda obsahuje takúto funkciu, ktorá reaguje na udalosť zmazania a podľa potreby zmaže svoje vlastné objekty. Týmto spôsobom je zabezpečená

integrita databázy, aby nevznikali „zombie“ záznamy.

3.5 Kontrolné zoznamy

Generované testy vychádzajú z metodológie ASVS, ktorá obsahuje približne 280 testov rozdelených do viacero kategórií. Pre funkcionalitu testov slúžia štyri databázové tabuľky v aplikácií. Prvou je „test_categories“, ktorá obsahuje ID a názov kategórie tak, ako ich špecifikuje ASVS (napríklad „Authentication Verification Requirements“). Druhou tabuľkou je „test_sub_categories“, ktorá obsahuje jednotlivé testy, tie sú zároveň namapované na kategóriu. V tejto tabuľke je vložených 226 testov z celkových približne 280. Jednotlivé testy obsahujú všetky informácie, ktoré ASVS poskytuje, teda názov (ktorý je často aj jeho popisom), úroveň (L1 až L3), odkaz na CWE[3] (Common Weakness Enumeration) a sekciu NIST normy. K jednotlivým testom nebol vytvorený návod, ako test vykonať. Je to z dôvodu veľkého množstva testov a vyžadovaná odbornosť pre vytvorenie takéhoto návodu je nad rámec tejto práce. Viacero testov by malo iný postup v závislosti na aplikácií a nie je možné vytvoriť všeobecný postup.

Treťou tabuľkou je „test_mappings“. Ide o najpodstatnejšiu tabuľku, ktorá mapuje testy na vstupy/formuláre/projekt. Určuje, ktoré testy budú vygenerované na základe toho, či ide o projekt, formulár alebo vstup. V prípade vstupu sa ešte posudzuje o aký druh vstupu ide[4] (file, date, number,...). Tieto druhy sú uložené v oddelenej tabuľke „input_types“. Posledná tabuľka („tests“) obsahuje zoznam všetkých vygenerovaných testov. Obsahuje ID prvku, pre ktorý bol vytvorený, teda ID formulára, projektu alebo vstupu a ID testu (z tabuľky test_sub_categories).

3.5.1 Vytvorenie dát pre testy

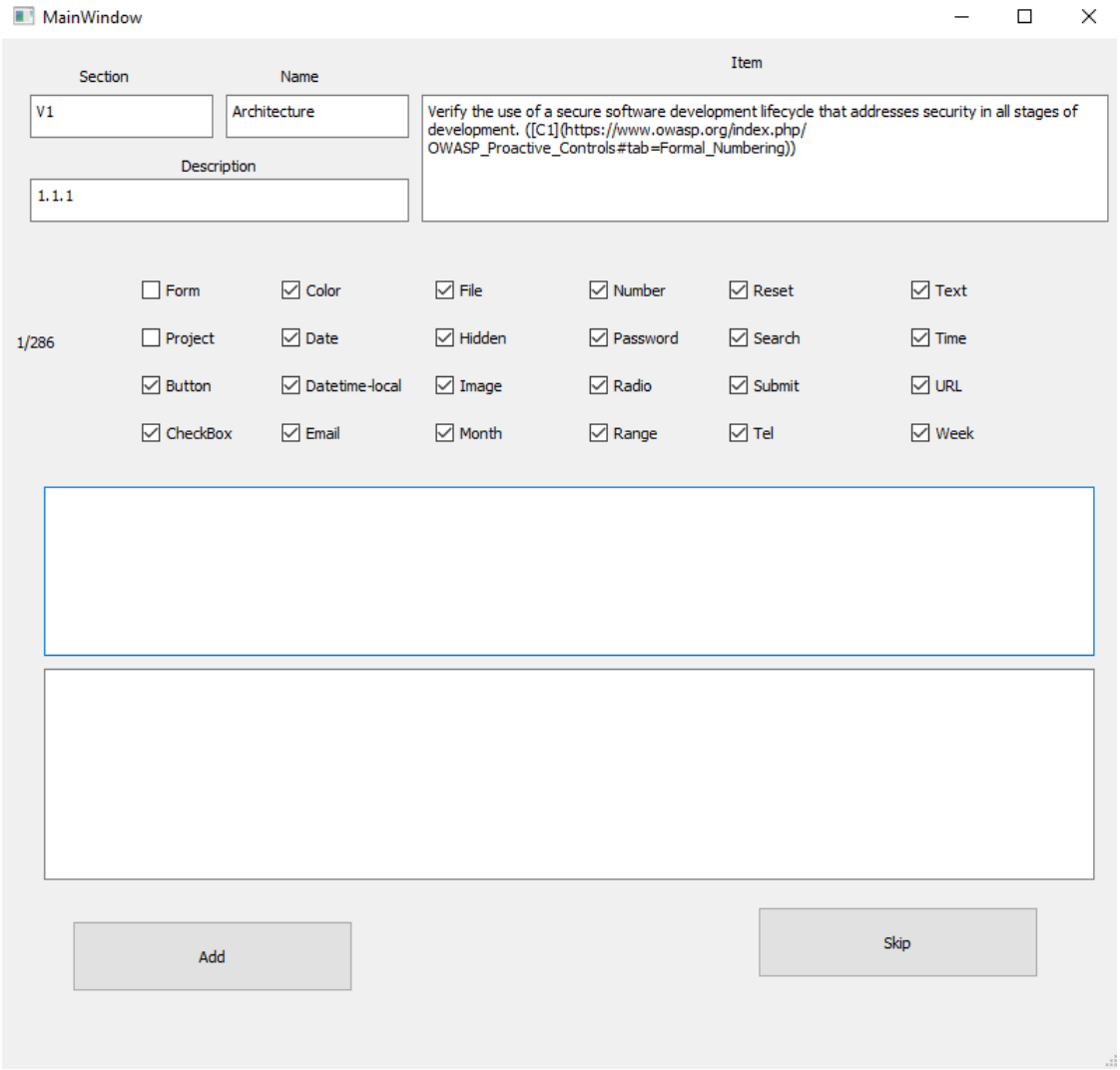
ASVS je možné získať formou CSV (hodnoty oddelené čiarkou) súboru, ktorý je možné strojovo spracovať. Pre potreby vytvorenia mapovania, teda ktorý test sa má priradiť ku ktorému typu vstupu, bola vytvorená dodatočná aplikácia pomocou jazyka Python a frameworku PyQt5[5] (pre grafické rozhranie). Externá aplikácia bola vytvorená preto, lebo táto funkcionalita (teda vytvorenie testov) sama o sebe nebola dostačujúca na to aby PHP aplikácia obsahovala administratívne rozhranie, ktoré musí byť špecificky zabezpečené a oddelené od štandardnej verzie aplikácie. Tento program je možné vidieť na obrázku 3.2. Aplikácia prechádza CSV súbor

[3] https://cwe.mitre.org/

[4] Pre účely tejto práce sa ako druh vstupu používajú druhy, ktoré sú v HTML štandarde: https://www.w3schools.com/html/html_form_input_types.asp

[5] https://pypi.org/project/PyQt5/

s ASVS a každý test vypíše do GUI (jednotlivé položky ako popis testu, kategória a podobne). Užívateľ pomocou checkboxov vyberie, pre ktoré typy vstupov má byť test priradený (formulár a projekt je tiež braný ako typ vstupu). Taktiež je možné pripísať poznámku k testu. Kliknutím na tlačidlo „Skip“ sa test preskočí a nebude vložený do databázy. Tlačidlo „Add“ vygeneruje SQL príkazy, potrebné na vloženie samotného testu do databázy aj na jeho mapovanie. Generované SQL sa zobrazuje užívateľovi v spodnom textovom okne a zároveň sa zapisuje do súboru.



Obr. 3.2: GUI pre parsovanie ASVS

Časť vygenerovaného SQL je zobrazená vo výpise 18. Po spracovaní celého ASVS bolo vygenerovaných 1018 riadkov SQL kódu, ktorý je možné vložiť do databázy. Je však potrebné, aby tabuľky test_sub_categories a test_mappings boli prázdne, a automatické inkrementovanie bolo obnovené na pôvodnú hodnotu (ID sa musia navzájom zhodovať). Toto je možné docieliť príkazom „ALTER TABLE test_mappings

AUTO_INCREMENT = 1“. Oproti manuálnemu vkladaniu testov a mapovaní je práca s touto aplikáciou rýchlejšia a menej frustrujúca (manuálne vkladať 1000 záznamov do databázy je časovo náročné). Toto generovanie dát pre testy stačí spustiť iba raz, a následne je možné aplikáciu distribuovať s predpripravenou databázou.

```
1 INSERT INTO test_sub_categories VALUES (NULL, 5, 'Validation', '5.2.7', 'Verify
  ↪ that the application sanitizes, disables, or sandboxes user-supplied SVG
  ↪ scriptable content, especially as they relate to XSS resulting from inline
  ↪ scripts, and foreignObject.', 'X', 'X', 'X', '159', '', '', NULL, NULL);
2 INSERT INTO test_mappings VALUES (NULL, 12, 5, 97, NULL, NULL);
3 INSERT INTO test_mappings VALUES (NULL, 14, 5, 97, NULL, NULL);
4 INSERT INTO test_mappings VALUES (NULL, 24, 5, 97, NULL, NULL);
5 INSERT INTO test_sub_categories VALUES (NULL, 5, 'Validation', '5.2.8', 'Verify
  ↪ that the application sanitizes, disables, or sandboxes user-supplied
  ↪ scriptable or expression template language content, such as Markdown, CSS or
  ↪ XSL stylesheets, BBCode, or similar.', 'X', 'X', 'X', '94', '', '', NULL,
  ↪ NULL);
6 INSERT INTO test_mappings VALUES (NULL, 12, 5, 98, NULL, NULL);
7 INSERT INTO test_mappings VALUES (NULL, 14, 5, 98, NULL, NULL);
```

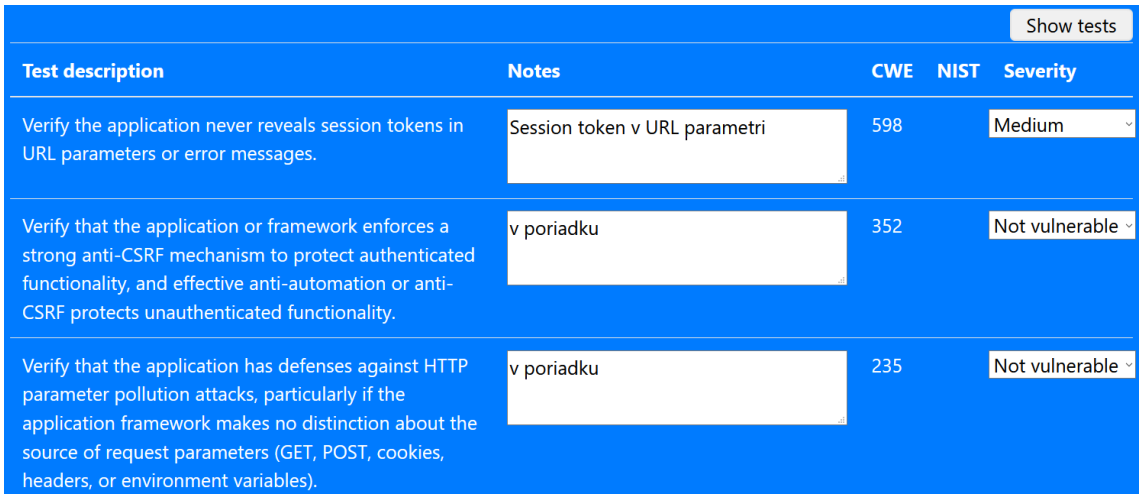
Výpis 18: Vygenerovaný SQL kód.

3.5.2 Práca s testami

Testy sa v aplikácií zobrazujú na dvoch miestach. Pri zobrazení projektu (/projects/{id}) sa zobrazujú globálne testy pre celý projekt. Pri zobrazení stránky sa zobrazia testy pre jednotlivé formuláre a vstupy. Testy sú zobrazené ako tabuľka. Toto zobrazenie je vidieť na obrázku 3.3. V celej aplikácií platí, že testy sú „zabalené“, je teda nutné stlačiť tlačidlo pre rozbalenie sekcie. Podobne sú riešené aj vstupy, ktoré sú zabalené tiež. Vďaka takémuto zabaľovaniu je aplikácia prehľadnejšia a nepotrebné sekcie je možné skryť.

Každý test obsahuje mieru závažnosti (severity), ktorý je reprezentovaný pomocou rozbaľovacieho menu (dropdown). Pri zmene závažnosti sa automaticky pomocou asynchrónneho volania (jQuery) zmena uloží na server (viď výpis 32 v prílohe B).

Všetky závažnosti (HTML element „<select>“) majú nastavenú triedu s názvom „setVulnerable“. JQuery obsahuje funkciu, ktorá pri zmene stavu (udalosť „change“) odošle nový stav závažnosti na server (na URL /tests/changeState pomocou POST metódy, ktorá obsahuje ID testu a nový stav).



Obr. 3.3: Zobrazenie testov.

Na podobnom princípe funguje aj popis nálezu (element „<textarea>“). Pri zmene popisu je automaticky a asynchrónne zaslaná požiadavka na server s ID testu a popisom, ktorý sa uloží. Nie je nutné znova načítavať stránku.

CWE identifikátor je spracovaný tak, aby automaticky obsahoval link na stránku s popisom daného CWE. CWE môže obsahovať informácie o zraniteľnosti, popis ako ju zneužiť/opraviť aj aký môže mať dopad.

3.5.3 Report

Pre vytvorenie PDF reportu sa využíva PHP knižnica mPDF[6]. Táto knižnica generuje PDF súbor z HTML kódu. Pre vytvorenie PDF sa vytvorí HTML kód so všetkými dátami, ktoré majú byť v reporte obsiahnuté a o zvyšok sa postará knižnica. Report obsahuje obálku, stručné zhrnutie projektu, kde je zobrazený názov, kto projekt testoval, koľko testov bolo vygenerovaných a štatistika závažnosti. Posledná časť reportu obsahuje jednotlivé testy. Tieto testy sú zoradené od najzávažnejších zistení a sú zoskupené podľa ASVS testu. Jeden ASVS test môže byť vygenerovaný pre viacero vstupov a teda je vhodné ich v reporte zoskupiť do jednej kategórie, aby užívateľ videl, že daná zraniteľnosť sa nachádza v aplikácii na viacerých miestach. Ukážka reportu je zobrazená na obrázku 3.4.

Zmyslom tohoto reportu nie je úplné nahradenie reportu, ktorý vytvára človek, nakoľko takýto report je komplexný, obsahuje obrázky, ukážky a podobné výstupy. Štandardný report tiež nepostupuje podľa ASVS štandardu ale podľa Testing Guide. Cieľ tohto reportu je podobný ako u iných automatických nástrojov, ako napríklad

[6]https://mpdf.github.io/

Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the "null" origin.

Family: Config

Name	Type	URL	Severity
	Global		Medium

Verify the application never reveals session tokens in URL parameters or error messages.

Family: Session

Name	Type	URL	Severity
asda	Form (asda)	asdf	Medium
Finding: Session token v URL parametri			

Obr. 3.4: Ukážka reportu.

OWASP ZAP. Report je súhrnom zapísaných informácií a poskytuje prehľad o vykonaných testoch.

3.6 Vytvorenie Docker kontajnera

Vytvorenie kontajnera prebiehalo podľa doporučeného návodu[14]. Kontajner sa bude skladať celkovo z troch služieb. Prvou službou je samotná aplikácia, ktorá bude vyžadovať PHP a dodatočné nástroje ako napríklad composer. Druhá služba je MySQL databáza, ku ktorej sa aplikácia bude pripájať a poslednou službou je Nginx webový server. Celkovo bude tento kontajner obsahovať tri moduly. MySQL služba bude vytvorená pomocou predpripraveného obrazu (image), ktoré je k dispozícií prostredníctvom Docker Hub. Pre aplikáciu a Nginx je nutné vytvoriť vlastné obrazy. Prvým krokom pre vytvorenie obrazu modulu aplikácie je vytvorenie konfiguračného súboru pre nový obraz, tzv. „Dockerfile“. Vytvorený Dockerfile pre aplikáciu je možné vidieť vo výpise 19.

V prvom riadku sa nastavuje východiskový obraz, z ktorého bude tento obraz vychádzať, ide o oficiálny PHP 7.4 obraz dostupný na Docker Hub. Riadky 2 a 3

```
1  FROM php:7.4-fpm
2  ARG user
3  ARG uid
4  RUN apt-get update && apt-get install -y git curl libpng-dev \
5      libonig-dev libxml2-dev zip unzip
6  RUN apt-get clean && rm -rf /var/lib/apt/lists/*
7  RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd
8  COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
9  ADD ./ /var/www
10 RUN chmod -R 777 /var/www/storage/
11 RUN chmod -R 777 /var/www/vendor/mpdf/mpdf
12 RUN useradd -G www-data,root -u $uid -d /home/$user $user
13 RUN mkdir -p /home/$user/.composer && \
14     chown -R $user:$user /home/$user
15 WORKDIR /var/www
16 USER $user
```

Výpis 19: Dockerfile pre PHP službu.

obsahujú argumenty, ktoré budú definované v inom súbore (docker-compose.yml). Riadky 4 a 5 spustia aktualizáciu operačného systému a nainštalujú dodatočné nástroje. Nainštalujú sa tiež doplnky pre PHP (riadok 7), composer (riadok 8, skopíruje sa najaktuálnejšia verzia composer do /usr/bin/composer aby bol príkaz „composer“ k dispozícií) a presunie sa samotná aplikácia z aktuálnej zložky (na hostiteľskom zariadení) do zložky /var/www v kontajneri (riadok 9). Je však nutné upraviť práva k niektorým zložkám aby mohla samotná aplikácia do nich zapisovať (riadky 10 a 11). vytvorí sa nový užívateľ, na ktorého účet sa zároveň prepne. Ku kontajneru sa tak pristupuje ako bežný užívateľ bez root právomoci. Ako východiskový adresár je zvolený /var/www, kam je umiestnená aj samotná aplikácia. Pri ďalšom konfigurovaní bude nutné spúšťať príkazy pomocou composer a artisan nástrojov. Prepnutie adresára zabezpečí, že príkazy sa vykonajú v správnom priečinku. Treba podotknúť, že samotný Dockerfile sa nachádza v koreňovom priečinku aplikácie.

Vytvorenie obrazu pre Nginx službu je jednoduchšie, nakoľko nie sú vyžadované veľké zmeny do predpripraveného obrazu. Jediná podstatná vec je nakopírovanie zložky public do adresára /var/www/public v Nginx kontajneri. Takto vytvorený Dockerfile je vidieť vo výpise 20.

Ďalším krokom je vytvoriť konfiguráciu pre Nginx server. Ide o štandardný .conf súbor, ktorý nastavuje parametre ako port, adresár s aplikáciou a podobne. Pri vytvorení kontajnera sa použije tento konfiguračný súbor namiesto pôvodného. Vytvorený súbor nginx.conf je zobrazený vo výpise 21. Nginx bude bežať na porte

```
FROM nginx:alpine
ADD --chown=nginx:nginx ./public /var/www/public/
```

Výpis 20: Dockerfile pre Nginx službu.

80 (bez TLS), pre produkčné nasadenie je možné pridať aj zabezpečenú verziu pre port 443 s TLS a so správnym nastavením certifikátov. Pre prácu s PHP súbormi je nastavená cesta na „app:9000“, app je názov služby/modulu, na ktorom beží PHP (ide o modul, pre ktorý bol vytvorený Dockerfile) a 9000 je číslo portu.

```
server {
    listen 80;
    index index.php index.html;
    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/public;
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
    location / {
        try_files $uri $uri/ /index.php?$query_string;
        gzip_static on;
    }
}
```

Výpis 21: Konfiguračný súbor pre Nginx.

Ďalším krokom je zvolenie predpripravenej databázy, ktorá bude použitá ako východisková. Keďže aplikácia ma špecifickú štruktúru a niektoré tabuľky sú predpripravené, je vhodné distribuovať kontajner s fungujúcou a pripravenou databázou. Vyexportovanú databázu vo formáte .sql (tzv „SQL dump“) vložíme do priečinka docker-compose/mysql s názvom init_db.sql. Databázu je možné manuálne vyexportovať pomocou administratívneho rozhrania ako phpMyAdmin, manuálne z príkazového riadku alebo podobne. Databázu je možné tiež manuálne importovať, avšak je vhodné zvoliť toto riešenie, nakoľko je automatizované.

Posledným krokom je konfigurácia jednotlivých docker služieb, ktorá sa vykonáva v súbore docker-compose.yml. V princípe bude každá služba (aplikácia, nginx, databáza) bežať vo vlastnom kontajneri, avšak budú vystupovať ako jedna aplikácia. Vo výpise 22 je vidieť základnú štruktúru tohto konfiguračného súboru. Prvý riadok obsahuje verziu použitého nástroja docker-compose (na OS Windows funguje verzia 3.8 v poriadku, avšak na OS Linux dochádza k chybe a je nutné verziu znížiť na 3.7). Nasleduje popis služieb, ktorý bude obsahovať spomínané 3 služby pre aplikáciu, databázu a webový server. Tieto služby budú podrobne popísané nižšie. Na záver je definovaná zdieľaná sieť pre všetky služby s názvom „dp“ a je nastavená na bridge mód.

```
1 version: "3.8"
2 services:
3         . . .
4 networks:
5   dp:
6     driver: bridge
```

Výpis 22: Východiskové nastavenie súboru docker-compose.yml.

Prvou službou bude samotná aplikácia s PHP a jej názov je „app“. Bude vytvorená zostavením kontajnera pomocou definovaného Dockerfile (riadok 7). Konfiguráciu tejto služby je vidieť vo výpise 23. V tejto službe je definovaný jej názov (app), parametre pre Dockerfile (user a uid, z ktorých sa vytvorí nový užívateľ), cesta ku Dockerfile (nachádza sa v rovnakom priečinku), názov obrazu, ktorý sa vytvorí (sim0nah/dp a značka verzie „latest“), názov kontajnera (dp-app), parameter „restart“, ktorý zabezpečí aby bola služba automaticky reštartovaná, pokiaľ nebola manuálne zastavená a nastavenie adresárov a siete. Kontajner pri nasadení na OS Linux zdieľa kernel medzi kontajnerom a hostiteľským systémom, preto je nutné voliť UID užívateľa rozumne, keďže toto UID bude zároveň zdieľané na úrovni hostiteľského kernelu a môže dôjsť ku kolízii.

Nasledujúcou službou bude databáza s názvom „db“ (výpis 24). Bude vytvorená z predpripraveného oficiálneho obrazu MySQL vo verzií 5.7 (distribuovaný prostredníctvom Docker Hub) a jednotlivé parametre ako názov databázy, meno a heslo užívateľa budú priradené z .env súboru, ktorý je zároveň konfiguračným súborom pre Laravel aplikáciu (a je teda priamo obsiahnutý v aplikácií, nie je nutné ho dodatočne vytvárať). Toto zaručí, že aplikácia má správne nastavené prístupové údaje. Riadok 13 zabezpečí vloženie predpripraveného SQL kódu s vyexportovanou databázou do databázy v kontajneri. MySQL služba automaticky tento súbor načíta

```
app:
  build:
    args:
      user: user
      uid: 1000
    context: ./
    dockerfile: Dockerfile
  image: sim0nah/dp:latest
  container_name: dp-app
  restart: unless-stopped
  working_dir: /var/www/
  networks:
    - dp
```

Výpis 23: Nastavenie služby „app“.

a vloží do svojej databázy, čím vytvorí nakonfigurovanú štruktúru a predpripravené dáta. Nastavenie siete a parameter „restart“ sú zhodné ako pri službe app.

```
db:
  image: mysql:5.7
  container_name: dp-db
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: ${DB_DATABASE}
    MYSQL_ROOT_PASSWORD: ${DB_PASSWORD}
    MYSQL_PASSWORD: ${DB_PASSWORD}
    MYSQL_USER: ${DB_USERNAME}
    SERVICE_TAGS: dev
    SERVICE_NAME: mysql
  volumes:
    - ./docker-compose/mysql:/docker-entrypoint-initdb.d
  networks:
    - dp
```

Výpis 24: Nastavenie služby „db“.

Posledná služba je samotný webový server Nginx (výpis 25). Táto služba vychádza z obrazu Alpine Linux a Nginx vo verzií 1.17. Ide o odľahčenú verziu OS Linux, aby nezaberala veľa systémových zdrojov. Jedná sa o oficiálnu distribúciu tohoto obrazu pre Docker. Nginx je populárny webový server, ktorý je podobný Apache serveru, navyše je rýchlejší a efektívnejší [15], preto bol zvolený ako webový

server. Riadok 8 až 9 obsahuje presmerovanie portov. Port 8000 na hostiteľskom zariadení bude presmerovaný na port 80 tejto služby (teda na samotný Nginx). Riadok 11 umožní kontajneru využiť prístup k hostiteľskej zložke a tá je zdieľaná medzi hostiteľom a kontajnerom. Týmto spôsobom je konfigurácia ľahko prístupná pre užívateľa v prípade nevyhnutných zmien. Nastavenie siete je zhodné s predchádzajúcimi službami, všetky služby musia mať rovnakú sieť, keďže spolu komunikujú. Týmto krokom je konfigurácia dokončená.

```
1  nginx:
2    build:
3      context: .
4      dockerfile: ./docker-compose/nginx/Dockerfile
5    image: sim0nah/dp-nginx:latest
6    container_name: dp-nginx
7    restart: unless-stopped
8    ports:
9      - 8000:80
10   volumes:
11     - ./docker-compose/nginx:/etc/nginx/conf.d/
12   networks:
13     - dp
```

Výpis 25: Nastavenie služby „nginx“.

Nasledujúcim krokom je spustenie procesu „docker-compose“, ktorý vytvorí nami definované obrazy a následne kontajner. Spustením série príkazov vo výpise 26 sa vykonajú obe tieto akcie a zároveň sa spustí vytvorený kontajner.

```
1  docker-compose build app
2  docker-compose build nginx
3  docker-compose up -d
```

Výpis 26: Vytvorenie obrazu, kontajneru a jeho spustenie.

Kontajner však nie je plne funkčný. Laravel aplikácia obsahuje viaceré knižnice tretích strán, ktoré boli použité. Tie však zatiaľ nie sú v kontajneri nainštalované. Ich inštalácia je možná pomocou nástroja composer. Ďalej je nutné vytvoriť náhodný kľúč, ktorý Laravel využíva pri šifrovaní dát akými sú relácie alebo citlivé dáta. Oba tieto príkazy je možné vidieť vo výpise 27 (príkazy sú spúšťané prostredníctvom docker-compose). Týmto je kontajner hotový a funkčný. Pre spustenie aplikácie je možné z hostiteľa prejsť na URL „localhost:8000“.

```
docker-compose exec app composer install
docker-compose exec app php artisan key:generate
```

Výpis 27: Nainštalovanie dodatočných knižníc a generovanie kľúča pre Laravel.

Takto pripravený kontejner je možné distribuovať ako archív, prípadne pomocou platformy Docker Hub. Táto aplikácia bude vo finálnom stave dostupná aj ako archív (docker aj klasická štruktúra) a aj prostredníctvom Docker Hub[7][8]. Potrebné súbory pre docker-compose budú taktiež zverejnené[9].

[7] https://hub.docker.com/r/sim0nah/dp
[8] https://hub.docker.com/r/sim0nah/dp-nginx
[9] https://github.com/sim0nah/DP_WebApp

4 Testovanie bezpečnosti

Aplikácia bola budovaná v zmysle štandardov OWASP Testing Guide a dodržuje stanovené „best-practice“ postupy.

4.1 Zvýšenie bezpečnosti aplikácie

Autentifikácia

Celá aplikácia spadá pod autentifikáciu, s výnimkou prihlásenia a registrácie. Zvyšok aplikácie (hlavná časť) vyžaduje prihlásenie užívateľa. Autentifikácia je riešená priamo pomocou Laravelu. Vďaka tomu je možné využiť „middleware“ dodávaný priamo s Laravelom. Middleware je v podstate proxy, cez ktorú prechádzajú všetky požiadavky na server a odpovede na ne. Prostredníctvom „auth“ middleware je možné implementovať autentifikáciu, keďže všetky požiadavky budú automaticky kontrolované, či obsahujú platnú reláciu (cookie s názvom „laravel_session“). Pridaním tohto middleware do jednotlivých controllerov, pre ktoré je vyžadovaná autentifikácia (všetky okrem controllerov, ktoré obstarávajú autentifikáciu ako prihlásenie či registrácia), sa zabezpečí vyžadovanie platnej relácie (session), a tým sa zabezpečí autentifikácia v aplikácii. Pridanie middleware je možné pomocou kódu viditeľného vo výpise 28.

```
1 public function __construct()
2 {
3     $this->middleware('auth');
4 }
```

Výpis 28: Laravel middleware pre autentifikáciu.

Heslá užívateľov nie sú ukladané v otvorenom formáte, využíva sa funkcia bcrypt, ktorá je odporúčaná v štandardoch OWASP[16]. Použitá je aj soľ a ako algoritmus sa využíva blowfish. Formát hesla uloženého v databáze je vidieť vo výpise 29. Prvá časť hesla („$2y$“) značí použitie blowfish algoritmu. Nasledujúce číslo (10) značí tzv. „work factor“ alebo „cost factor“. Ide o počet iterácií, ktoré sú aplikované na heslo. V tomto prípade využíva aplikácia 10 iterácií. Nasledujúcich 128 bitov slúži pre soľ, zvyšok je samotný výstup bcrypt funkcie.

Táto aplikácia sa dá považovať za aplikáciu, ktorá pracuje s citlivými údajmi. Preto bola znížená doba platnosti užívateľskej relácie na 60 minút (pôvodná hodnota bola 120 minút).

```
1 $2y$10$HUOKQUyMR1sg3AUWg9XsieT8deNevE2iPOC4DFpKWAbJKefsZHOjm
```

Výpis 29: Heslo uložené v databáze.

Autorizácia

Všetky funkcie v aplikácií sú chránené voči útokom zameraným na autorizáciu. Jednotlivé funkcie v controlleroch overujú údaje zaslané v požiadavkách. Táto kontrola je zameraná na platnosť všetkých identifikátorov vzhľadom k užívateľovi. Napríklad sa kontroluje, či stránka, ku ktorej sa užívateľ snaží pristúpiť, patrí danému užívateľovi. Toto je možné vďaka vzťahom medzi jednotlivými objektami. V aplikácií platí postupnosť Projekt→Stránka→Formulár→Vstup (Test môže byť viazaný na projekt, formulár alebo vstup). Je teda možné zo vstupu zistiť projekt, ku ktorému patrí. Podobne to platí pri stránke alebo formulári. Pri každej funkcií sa preto kontroluje, či všetky ID s ktorými daná funkcia pracuje, patria danému užívateľovi. Táto kontrola zabraňuje situácií, kedy by útočník s platnými prihlasovacími údajmi mohol meniť údaje iného užívateľa, napríklad vytvárať projekt, alebo zmeniť hodnotu niektorého testu.

Hlavičky

Laravel pri východiskovom nastavení nenastavuje správne všetky hlavičky pri odpovediach zo strany servera. Prvou nesprávne nastavenou hlavičkou je „Cache-Control“, ktorá určuje, či má prehliadač obsah ukladať do medzipamäte (cache) alebo nie. Cache je vhodný pre statický obsah ako obrázky (ak neobsahujú citlivé dáta), CSS alebo JS súbory a podobne. Keďže v tejto aplikácií sa pracuje s citlivými dátami, je nevhodné aby sa údaje o testoch a zraniteľnostiach ukladali na disk používateľa. Východiskové nastavenie v Laravel je nastavené na „no-cache, private“. Toto nastavenie nie je bezpečné[1], nakoľko je stále prehliadaču umožnené uložiť dáta na disk. Správne nastavenie musí obsahovať možnosti no-cache a no-store.

Pre nastavenie tejto hlavičky bol vytvorený nový middleware, cez ktorý prechádzajú všetky požiadavky aj odpovede. Táto trieda je viditeľná vo výpise 30. Middleware je následne zaregistrovaný v Kernel.php súbore.

Druhou podstatnou hlavičkou je „X-FRAME-OPTIONS“, ktorá vo východiskovom nastavení nie je použitá. Táto hlavička bráni tomu, aby bola aplikácia načítaná v <iframe> HTML tagu. Ak by mohla byť načítaná, hrozilo by riziko Clickjacking útoku[1], pri ktorom by útočník mohol nalákať obeť na podvodnú stránku. Táto stránka by na pozadí načítala iframe s aplikáciou, no prekryla by ju netransparentnou vrstvou, aby ju obeť nevidela. Obeť by si myslela, že kliká na úplne inú neškodnú

```
class CacheControl
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);
        $response->header('Cache-Control', 'no-cache, no-store');
        return $response;
    }
```

Výpis 30: Nastavenie Cache-Control hlavičky.

stránku no v skutočnosti by sa kliknutia vykonávali na tejto aplikácií.

Laravel obsahuje middleware pre nastavenie tejto hlavičky s názvom FrameGuard. Stačí ho zaregistrovať v súbore Kernel.php a hlavička bude pridaná k odpovediam serveru. Východiskové nastavenie má hodnotu „sameorigin“, takže stránku je možné načítať pomocou iframe, avšak iba z rovnakého hostiteľa, na ktorom sa nachádza samotná aplikácia. Keďže pre túto aplikáciu nie je vyžadované načítavanie pomocou iframe, hodnota bola v triede FrameGuard zmenená na „deny“ (nie je tak možné stránku načítavať v iframe a clickjacking nie je možný).

Práca so vstupmi

V aplikácií sa nepoužívajú priame SQL príkazy, ktoré by mohli byť náchylné na SQL injection. Pre prácu s databázou sa využívajú štandardné nástroje dodávané s Laravelom.

Útoky typu XSS (Cross Site Scripting)[1] sú tiež zablokované, a to vďaka automatického HTML kódovaniu všetkých zobrazovaných vstupov. Ak by aj vstup obsahoval škodlivý kód, ten by sa nevykonal, keďže by prešiel HTML kódovaním. V Laraveli sa pre tento princíp využíva zápis „{{ ... }}“, ktorým automaticky HTML kóduje zadaný vstup.

4.2 Bezpečnostný test webovej aplikácie – OWASP ZAP

Nástroj OWASP ZAP[1] je bezplatná webová proxy, vrátane aktívneho a pasívneho skenera zraniteľností, s otvoreným kódom. Vývoj prebieha priamo pod záštitou organizácie OWASP. Pre potreby tejto aplikácie bol zrealizovaný automatizovaný test zraniteľností na celej stránke.

[1] https://www.zaproxy.org/

Najprv bol spustený pavúk (spider), ktorý prehľadal celú stránku a objavil všetky podstránky. Následne bol spustený automatický sken, s najtvrdšími nastaveniami (úroveň hlásenia zraniteľností od „Low“ a intenzita útoku na „Insane“). Sken aj spider bežali v autentifikovanom móde. ZAP mal tak plný prístup k aplikácií pod prihláseným užívateľom a k dispozícií vzorové dáta. Výsledok aktívneho skenu je možné vidieť v tabuľke 4.1.

Tab. 4.1: Výsledok aktívneho skenu pomocou nástroja OWASP ZAP.

	Strength	Reqs	Alerts	Status
Analyser		23		
Plugin				
Path Traversal	Insane	4479	24	Completed
Remote File Inclusion	Insane	3080	0	Completed
Source Code Disclosure (/WEB-INF folder)	Insane	0	0	Skipped
Server Side Include	Insane	352	0	Completed
Cross Site Scripting (Reflected)	Insane	259	5	Completed
Cross Site Scripting (Persistent)	Insane	88	0	Completed
Cross Site Scripting (Persistent)	Insane	95	0	Completed
Cross Site Scripting (Persistent)	Insane	0	0	Completed
SQL Injection	Insane	5701	5	Completed
Server Side Code Injection	Insane	704	0	Completed
Remote OS Command Injection	Insane	5808	0	Completed
Directory Browsing	Insane	95	0	Completed
External Redirect	Insane	968	0	Completed
Buffer Overflow	Insane	82	0	Completed
Format String Error	Insane	246	0	Completed
CRLF Injection	Insane	616	0	Completed
Parameter Tampering	Insane	130	0	Completed
Script Active Scan Rules	Insane	0	0	Skipped
Totals		23078	34	

Nájdené zraniteľnosti boli Path Traversal, XSS (Reflected) a SQL injection. Po manuálnom prezretí jednotlivých zraniteľností bolo usúdené, že všetky sú tzv. „false positive“ a teda reálnu aplikácia nie je zraniteľná. Ako Path Traversal bola vyhodnotená situácia, kedy v texte ASVS testu bolo zadané slovo „etc“ (v slovenčine atď.).

ZAP vyhodnotil túto skutočnosť ako priečinok /etc a označil ako zraniteľnosť.

XSS bolo objavené pri vytváraní nového projektu. Ak bolo zadané zlé ID klienta, aplikácia vyhodila chybu, v ktorej bol zobrazený text od užívateľa (v tomto prípade <script>alert(1)</script> kód). Pri zobrazení chybovej hlášky sa však kód nespustil, a tak nejde o reálnu zraniteľnosť. Tento nález však poukázal na nedostatok pri validácií, ktorý bol opravený. Pri vytváraní projektu sa teda kontroluje, či je ID klienta zadané v správnom tvare, ak nie, aplikácia má ošetrenú výnimku. Samotná chybová hláška nie je chyba, nakoľko aplikácia pracuje v debug režime, aby bolo možné opravovať chyby, ktoré sa pri vývoji vyskytnú.

ZAP vyhodnotil ako SQL injection stav, kedy aplikácia využíva skrytý parameter „_method“ v tele POST požiadavky, aby bolo možné využívať HTTP metódy PUT a DELETE. ZAP pridal k tomuto parametru znak „%“, ktorý však nemal vplyv na samotnú požiadavku, keďže všetky požiadavky boli vyhodnotené ako HTTP 404 Not Found (v tom čase už boli zdroje, na ktoré ZAP pristupoval zmazané, keďže ich počas skenu sám zmazal). Opäť tak nejde o zraniteľnosť.

Pasívny skener odhalil zopár problémov s hlavičkami, avšak opäť ide o false positive nálezy. Napríklad ZAP vyhodnotil, že aplikácia neobsahuje anti-CSRF tokeny. ZAP však kontroluje iba špecifické názvy anti-CSRF tokenov a štandardný názov pre Laravel (_token) medzi ne nepatrí. Aplikácia obsahuje anti-CSRF tokeny pri všetkých formulároch, kde sú potrebné, ZAP ich iba neobjavil.

Celkovo tento sken neodhalil žiadne zásadné zraniteľnosti. Bola objavená iba chyba pri validácií poľa „client“ a tento problém bol na základe nálezu odstránený.

Záver

Teoretická časť práce rozoberá metodológiu bezpečnostného testovania webových aplikácií OWASP Testing Guide. Popísané sú jednotlivé kapitoly, ktoré vysvetľujú priebeh a okruhy testovania. Spomenutá je aj metodológia ASVS, ktorá je založená na Testing Guide ale je formulovaná ako checklist.

Kapitola Návrh je prvou kapitolou praktickej časti a zameriava sa na návrh samotnej aplikácie. Sú v nej popísané technológie, ktoré boli použité pri vývoji aplikácie na podporu penetračného testovania. Medzi spomenutými technológiami sú PHP framework Laravel alebo napríklad knižnica Bootstrap. Popísaný je aj návrhový vzor MVC, ktorý je využitý aj pri samotnej implementácií. Ďalej bol vytvorený UML diagram databázy s popisom jednotlivých častí. Špecifikované boli tiež požiadavky na aplikáciu. Kapitolu uzatvára návrh grafického užívateľského rozhrania aplikácie.

Tretia kapitola sa zameriava na samotnú implementáciu aplikácie. Popis, krok po kroku, ozrejmuje prácu s frameworkom Laravel. V texte sú popísané jednotlivé funkcionality a spôsob, akým boli dosiahnuté. Celý postup implementácie je ukážkovo popísaný pri implementovaní jednej z častí aplikácie, a takisto krok po kroku. Ďalej nasleduje popis práce s testami. Pre vygenerovanie testov bola vytvorená vlastná aplikácia, aby sa tak uľahčila a zrýchlila práca s ASVS. Testy boli implementované na úrovni projektu, formulárov a vstupov. Aplikácia obsahuje aj možnosť vygenerovať report zo všetkých testov pre jednotlivé projekty.

Posledná kapitola bola venovaná bezpečnosti aplikácie. Pomocou automatizovaného skenera zraniteľností bol odhalený jeden nedostatok pri validácií dát, ktorý bol následne odstránený. Aplikácia je špecificky chránená voči útokom proti autorizácií, hlavičky sú nastavené tak, aby spĺňali platné štandardy a celá aplikácia by mala byť bezpečná.

Aplikácia podporuje všetky zadané funkcie, ktoré má mať. Je možné vytvárať projekty, zadávať k nim stránky, formuláre a vstupy. Testy pre jednotlivé prvky sú generované automaticky na základe štandardu ASVS. Prehľadné GUI uľahčuje prácu pri používaní aplikácie. Aplikácia umožňuje vytvárať viacero užívateľov, ktorí sú od seba navzájom oddelení. Zo zozbieraných údajov je možné vygenerovať report. Z aplikácie bol vytvorený Docker kontajner pre jednoduchšie zdieľanie a nasadenie.

Literatúra

[1] MEUCCI, Matteo a Andrew MULLER, 2019. *OWASP Testing Guide v4* [online]. [cit. 2019-12-02]. Dostupný z URL: <https://www.owasp.org/images/1/19/OTGv4.pdf>.

[2] AVITAL, Nadav, 2019. *The State of Web Application Vulnerabilities in 2018* [online]. [cit. 2019-12-02]. Dostupné z URL: <https://www.imperva.com/blog/the-state-of-web-application-vulnerabilities-in-2018/>.

[3] *SSL and TLS Deployment Best Practices* [online], 2017. [cit. 2019-12-03]. Dostupné z URL: <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>.

[4] CUTHBERT, Daniel, Andrew STOCK a Jim MANICO, 2019. *OWASP Application Security Verification Standard Project* [online]. In: . [cit. 2019-12-03]. Dostupné z URL: <https://github.com/OWASP/ASVS/tree/master/4.0>.

[5] LINDLEY, Cody *Front-end Developer Handbook 2019* [online]. [cit. 2019-12-05]. Dostupné z URL: <https://frontendmasters.com/books/front-end-handbook/2019/>.

[6] Bootstrap Documentation *About Bootstrap* [online]. [cit. 2019-12-05]. Dostupné z URL: <https://getbootstrap.com/docs/4.4/about/overview/>.

[7] *JQuery API* [online]. [cit. 2020-05-25]. Dostupné z: https://api.jquery.com/

[8] GeeksForGeeks *Frontend vs Backend* [online]. [cit. 2019-12-7]. Dostupné z URL: <https://www.geeksforgeeks.org/frontend-vs-backend/>.

[9] *What is Laravel and Why You Should Learn it?* [online]. [cit. 2019-12-09]. Dostupné z URL: <https://www.larashout.com/what-is-laravel-and-why-you-should-learn-it>.

[10] *MySQL Homepage* [online]. [cit 2019-12-12]. Dostupné z URL: <https://www.mysql.com/>.

[11] FERRARA, Anthony*Alternatives To MVC* [online]. [cit. 2019-12-12]. Dostupné z URL: <https://blog.ircmaxell.com/2014/11/alternatives-to-mvc.html>.

[12] IGHODARO, Neo *How Laravel implements MVC and how to use it effectively* [online]. [cit. 2019-12-15]. Dostupné z URL: <https://blog.pusher.com/laravel-mvc-use/>.

[13] *Docker Documentation* [online]. [cit. 2020-05-26]. Dostupné z: https://docs.docker.com/get-started/

[14] HEIDI, Erika. *How To Containerize a Laravel Application for Development with Docker Compose on Ubuntu 18.04* [online]. 23.1.2020 [cit. 2020-05-26]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-containerize-a-laravel-application-for-development-with-docker-compose-on-ubuntu-18-04

[15] PLESKY, Elvis, 2018. *Nginx vs Apache — which is the best web server?* [online]. [cit. 2020-05-26]. Dostupné z: https://www.plesk.com/blog/various/nginx-vs-apache-which-is-the-best-web-server/

[16] *Password Storage Cheat Sheet* [online]. [cit. 2020-05-25]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Zoznam príloh

A Vzorový view súbor

```
@extends("layout") //dedenie z hlavného súboru
@section("content") //umiestnenie súboru do sekcie "content" v hlavnom súbore
<div id="content-wrapper">
    <div class="container-fluid">
    <div class="row">
    @foreach ($projects as $project) //prejdenie zoznamu projektov, pre každý
    ↪ projekt sa vykoná sekcia
        <div class="col-xl-3 col-sm-6 mb-3">
        <a class="card-body text-white clearfix mr-5"
        ↪ href="/project/{{$project->id}}">
        <div class="card text-white bg-dark o-hidden h-100">
        <div class="card-body">
        //blade.php formát umožnuje pristupovať k premenným ak sú umiestnené v
        ↪ zátvorkách {{}}
        <span class="float-left">{{$project->name}}</span>
        <span class="float-right"></span>
        </div>
        <div>
        <a href="/projects/{{$project->id}}/edit">
        <button class="float-right btn btn-primary">Edit</button>
        </a>
        <a href="/projects/{{$project->id}}">
        <button class="float-right btn btn-primary"
        ↪ style="margin-right:5px">View</button></a>
        </div>
        </div>
        </a>
    @endforeach
    </div>
    </div>
    </div>
@endsection
```

Výpis 31: Vzorový *view* súbor pre zobrazenie všetkých projektov

B Asynchrónna zmena závažnosti

```
<select id="vuln{{$pTest->id}}" class="setVulnerable" data-id="{{$pTest->id}}">
...
</select>
<script>
$(document).ready(function() {
  $('.setVulnerable').on('change', function () {
      var $id = $(this).data('id');
      var $value = this.value;
      $.ajaxSetup({
          headers: {
              'X-CSRF-Token': "{{ csrf_token() }}"
          }
      });
      $.ajax({
          url: "/tests/changeState",
          type:"POST",
          data: "test=" + $id + "&state=" + $value,
          success:function(response){
              toastr.success('Success');
          },
          error:function(response){
              toastr.error('Error');
          },
      });
  });
});
</script>
```

Výpis 32: Asynchrónna zmena závažnosti.