UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Analiza codului-sursă PHP. Un instrument Web pentru detectarea vulnerabilităților

propusă de

Maxim Andrei

Sesiunea: *Iulie, 2019*

Coordonator științific

Conferențiar, Dr. Sabin Corneliu Buraga

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Analiza codului-sursă PHP. Un instrument Web pentru detectarea vulnerabilităților

Maxim Andrei

Sesiunea: *Iulie, 2019*

Coordonator științific

Conferențiar, Dr. Sabin Corneliu Buraga

Avizat,

Îndrumător Lucrare de Licenţă

Titlul, Numele şi prenumele

_____

Data _____ Semnătura

_____

DECLARAŢIE privind originalitatea conţinutului lucrării de licenţă

Subsemnatul(a) ……

domiciliul în ……..

născut(ă) la data de ……..…., identificat prin CNP ……..……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …….. specializarea ……, promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 şi 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student……..

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Analiza codului-sursă PHP. Un instrument Web pentru detectarea vulnerabilităților*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *27.06.2019*

Absolvent Maxim Andrei

_____

Cuprins

1 Introducere

"Complexity kills. It sucks the life out of developers, it makes products difficult to plan, build and test, it introduces security challenges and it causes end-user and administrator frustration."

Ray Ozzie

Într-o lume a sistemelor *software* în care vulnerabilitățile pot apărea de nicăieri și pot fi exploatate de oricine, a devenit din ce în ce mai necesar ca dezvoltatorii să-și ia toate măsurile de precauție atunci când dezvoltă o aplicație *software*. Atunci când se apropie termenul limită și trebuie livrat un cod de calitate având anumite standarde implementate greșeala nu este permisă și poate avea efecte fatale. Atunci când vine vorba de dezvoltarea unor proiecte mari devine din ce în ce mai dificil pentru programatori să întrețină codul, având în vedere că fiecare are propriul său stil și moduri diferite de a aborda o problemă. Astfel, analiza statică a codului vine ca o soluție pentru această problemă deoarece poate forța anumite standarde comune și poate detecta diferite erori sau greșeli din implementare încă din faza de dezvoltare, înainte ca proiectul să fie rulat într-un mediu de producție.

Multe aplicații web scrise în PHP suferă de vulnerabilități de tip *injection* iar analiza statică a codului este capabilă să expună aceste probleme înainte ca ele să ajungă pe web. Odată cu creșterea numărului de aplicații web, a crescut și numărul vulnerabilităților de securitate iar impactul acestora poate fi catastrofal. *Code review*-urile consumă destul timp, sunt predispuse la greșeala umană și de aceea este nevoie de o soluție care să automatizeze analizarea codului sursă.

Aplicația "**Analiza codului-sursă PHP. Un instrument Web pentru detectarea vulnerabilităților. (ACSP)**" este o soluție unică pe piață întrucât oferă pentru limbajul PHP atât o modalitate de analizare statică a codului, cât și o analizare dinamică printr-un mod eficient de *deployment*. Pentru limbajul PHP, o posibilă alternativă deja existentă în piață al acestei aplicații ar fi *Exakat,* diferența fiind faptul că ea este specializată doar în analizarea statică a codului iar aplicația ACSP oferă în plus posibilitatea unei analize dinamice dar și posibilitatea de *deployment* a mai multor proiecte.

1.1 Contribuții

Contribuțiile mele aduse în această aplicație sunt următoarele:

- Integrarea unor instrumente de analiză statică a codului într-o singură aplicație
- Parsarea și expunerea rezultatelor analizei statice a codului într-un format comun și ușor de vizualizat de către client
- Oferirea unei modalități de *deployment* a mai multor aplicații ce vor rula fiecare într-un mediu complet izolat, astfel punând la dispoziție clientului un mijloc de a analiza dinamic codul
- Activarea *profiler*-ului extensiei de PHP Xdebug ce este capabil să analizeze în mod dinamic o aplicație PHP
- Integrarea unui server extern specializat în oferirea unui mod eficient de vizualizare în timp real a metricilor întoarse de extensia PHP Xdebug. Poate expune clientului chiar și fișierele care au probleme, cu bucățile de cod vulnerabile marcate corespunzător
- Înglobarea tuturor acestor funcționalități într-o interfață web cât mai transparentă și cât mai eficientă

1.2 Motivație

Motivația alegerii acestei teme de licență constă în următoarele idei:

- PHP este unul dintre cele mai vulnerabile limbaje de programare când vine vorba de atacuri de tip *injection* dar și de probleme legate de permisiuni, acces al controlului sau validări ale datelor primite de la utilizator.
- Lipsa pe piață a unei soluții pentru limbajul PHP care să înglobeze atât detectarea vulnerabilităților printr-o analiză statică a codului dar și una dinamică, având posibilitatea de *deployment* a mai multor aplicații.
- Dorința de prevenire a greșelilor atunci cand se implementează un proiect dar și îmbunatățirea detecției vulnerabilităților înainte ca acestea să fie exploatate.
- Dorința de a automatiza procesul de descoperire a vulnerabilităților.

2 Fundamente

Mai departe, vor fi descrise cele mai importante tehnologii folosite, împreună cu rolul fiecăreia în arhitectura aplicației.

2.1 Calitatea *software*

Calitatea *software* este un concept abstract iar lipsa acesteia iese în evidență și poate fi observată cu ușurință. Doi factori stau la baza calității *software*:

- Produsul să fie conform cu cerințele
  - Cerințele trebuie să fie explicate clar și produsul să se conformeze
  - Orice deviație este considerată defect
  - Un produs de bună calitate conține mai puține defecte
- Produsul să fie bun pentru utilizare
  - Să se ridice la așteptările utilizatorilor
  - Un produs de bună calitate trebuie să ofere o satisfacție ridicată în utilizare

[1]

2.2 Metrici de complexitate

Complexitatea ciclomatică este o metrică prin care se măsoară complexitatea unui program. Aceasta reprezintă de fapt numărul de drumuri independente din codul sursă al unui program. Ea poate fi calculată cu respect la clasele, metodele sau modulele unui program.

LOC sau *lines of code* este o metrică prin care se măsoară dimensiunea unui program prin numărarea liniilor de cod.

LLOC sau *logical lines of code* este o metrică prin care se măsoară de asemenea dimensiunea unui program doar că aici se numără doar instrucțiunile. De exemplu „*i = 5*” va fi numărată, dar parantezele sau acoladele aflate pe câte o linie nu vor conta. [10]

2.3 **Atacuri**

SQL Injection este un tip de atac ce constă în inserarea sau injectarea unei comenzi SQL prin date sau informații primite direct de la client către aplicație. O injectare SQL executată cu succes poate rezulta în expunerea unor date sensibile sau modificarea lor din baza de date. [2]

Cross-Site Scripting (XSS) este un tip de atac ce constă în inserarea unor scripturi malițioase în aplicații de încredere. Acesta apare atunci când un atacator se folosește de o aplicație web pentru a trimite cod malițios, de obicei sub forma unui script ce va fi rulat de navigator, către alt utilizator. Greșelile care permit acest tip de atac sunt destul de răspândite și pot apărea în orice aplicație web care oferă utilizatorilor formulare dar nu validează corespunzător datele primite. [3]

2.4 **Docker & Compose**

Docker este o platformă *software* care are la bază conceptul de container. Containerele permit dezvoltatorului să-și împacheteze aplicația împreună cu dependințele necesare acesteia și să o ruleze într-un mediu izolat. Se aseamănă cu mașina virtuală dar funcționează diferit deoarece containerele virtualizează sistemul de operare, nu resursele *hardware*. Containerele sunt mai portabile și mai eficiente. Ele sunt create din imagini de docker. O imagine de docker este un *template* folosit pentru crearea unui container și poate fi obținută fie din *Docker Hub* [1], fie construită manual de către client. [11]

În Figura 1 putem vedea diferența dintre Docker și Mașina virtuală, referitoare la modul de virtualizare a resurselor.

[1] un repozitoriu public, similar cu GitHub

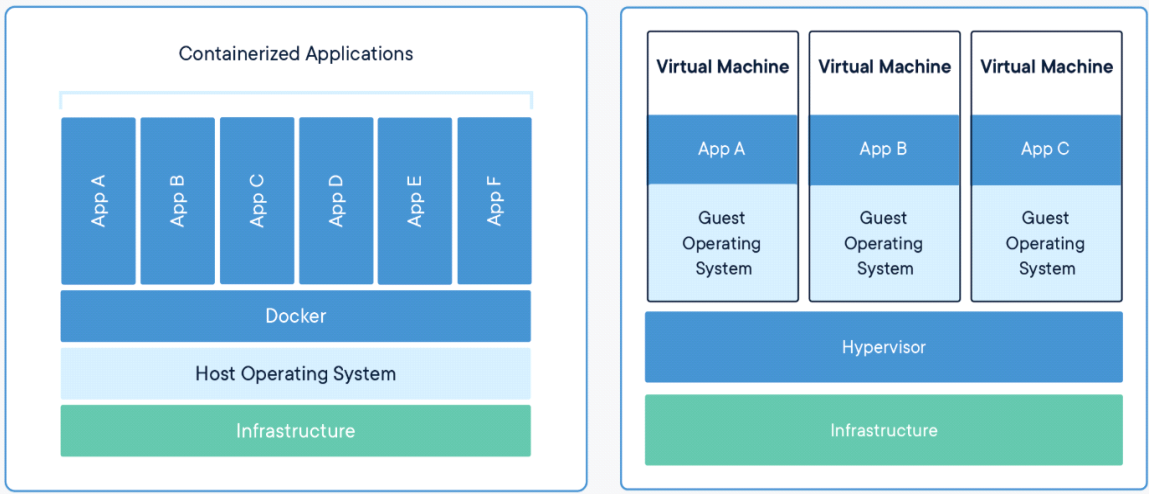

Figura 1 – docker vs mașina virtuală

Compose este un instrument construit cu scopul definirii și rulării mai multor containere. Este folosit un fișier de configurare YAML [2] unde sunt definite toate serviciile necesare unei aplicații. Apoi, printr-o singură comandă sunt create și pornite toate serviciile definite în acel fișier de configurare. [12]

2.5 **Nginx**

Nginx este un server web care poate fi folosit drept *load balancer*, *reverse proxy*, *HTTP cache* sau *proxy mail server*. A fost dezvoltat pentru a oferi un consum redus de memorie iar cererile sunt tratate în mod asincron într-un singur fir de execuție.

2.6 **Vue.js (Vue)**

Vue este un *framework open-source* scris în Javascript ce oferă multiple funcționalități pentru crearea de interfețe pentru utilizatori și aplicații *single-page*. Printre avantajele care au dus la utilizarea acestui *framework* față de altele precum Angular sau React sunt dimensiunea sa redusă de aproximativ 18-21 KB, ușurința cu care poate fi folosit și înțeles datorită structurii sale simple și o documentație detaliată astfel încât orice dezvoltator care este familiar cu Javascript și HTML își poate crea propria aplicație.

[2] ce reprezintă un *human-readable dată-serialization language*

De asemenea, Vue poate fi structurat pe componente ce extind elemente de HTML, astfel încapsuland cod reutilizabil. [13]

2.7 **Vuetify**

Vuetify este un *framework* pentru componente de Vue bazat pe Material Design [3]. Suportă toate navigatoarele moderne și oferă un mod de reutilizare a componentelor, ce dau aplicației ACSP un design modern, standardizat dar și o experiență dinamică a utilizatorului.

2.8 **MongoDB**

MongoDB este o bază de date NoSQL *open-source* orientată pe documente. Această bază de date beneficiază de suport din partea companiei 10gen. Diferența principală constă în faptul că stocarea datelor nu se face folosind tabele precum într-o bază de date relațională, MongoDB stochează datele sub formă de documente JSON [4] cu scheme dinamice. MongoDB este o bază de date scrisă în C++. Aceasta poate conține mai multe baze de date, colecții și indecși. În unele cazuri (baze de date și colecții) aceste obiecte pot fi create implicit. Odată create, ele se găsesc în catalogul sistemului *db.systems.collection, db.system.indexes*. Colecțiile conțin documente (BSON [5]). Aceste documente conțin la rândul lor mai multe câmpuri. În MongoDB nu există câmpuri predefinite spre deosebire de bazele de date relaționale, unde există coloanele care sunt definite în momentul în care tabelele sunt create. Nu există schemă pentru câmpurile dintr-un document, acestea precum și tipurile lor pot varia. Astfel nu există operația de "*alter table*" pentru adăugare de coloane. În practică este obișnuit ca o colecție să aibă o structura omogenă, deși nu este o cerință, colecțiile putând avea structuri diferite. Această flexibilitate presupune ușurință în migrarea și modificarea imaginii de ansamblu asupra datelor. [4] [14]

În cadrul aplicației ACSP a fost utilizat MongoDB deoarece nu sunt necesare relații între entități, crearea de documente se face *on the fly* iar schema sa dinamică permite aplicației să salveze rezultatele analizei statice a codului în orice format. În subcapitolul 4.4 sunt descrise mai detaliat colecțiile folosite cât și structura fiecăreia.

[3] *design language* dezvoltat de Google în 2014
[4] *Javascript Object Notation*
[5] *Binary Javascript Object Notation*

2.9 **PHP Composer**

Composer este un *package manager* la nivel de aplicație care oferă o metodă standard de a manipula dependințele PHP necesare unei aplicații. Funcționează la linia de comandă și permite utilizatorilor să-și instaleze biblioteci de PHP disponibile pe "Packagist" [6].Utilizarea comenzii "*composer require nume_pachet*" va instala pachetul și îl va adăuga în fișierul "composer.json". [5]

2.10 **Analiza codului sursă**

Analiza codului sursă poate fi efectuată fie în mod static, fie în mod dinamic. Analiza statică a codului este o metodă de detectare a greșelilor ce pot duce la vulnerabilități, prin examinarea automată a codului sursă înainte ca acesta să fie rulat. Alternativa ar fi pur și simplu examinarea manuală sau prin *code review*-uri dar aceasta ar fi mult mai lentă și nu la fel de eficientă precum cea automată.

Beneficiile folosirii unor instrumente de analiză statică sunt urmatoarele:

- Viteza de execuție. Dezvoltatorilor le-ar lua mai mult timp să efectueze manual *code review*-uri, așadar un analizator static ar aduce un plus în acest caz dar și în faptul că dezvoltatorului îi este specificată exact linia de cod cu probleme iar timpul de fixare al posibilelor vulnerabilități s-ar reduce considerabil.
- Adâncimea analizei. Un analizator static trece prin toate liniile unui cod sursă, încercând să detecteze potențiale breșe.
- Acuratețea analizei. O analiză umană a codului este mult mai predispusă la greșeală decât una automată. Din această cauză se preferă recurgerea la instrumente de analiză statică pentru identificarea problemelor și aducerea codului sursă la anumite standarde.

Analiza dinamică a codului este de asemenea o metodă de a detecta greșeli în implementarea unui proiect doar că aceasta este efectuată dupa ce codul sursă este rulat.

Pentru limbajul PHP, baza unei analize statice a codului este reprezentată de analizarea structurilor folosite pentru a primi date de la utilizator și a anumitor funcții sensibile ce pot cauza diferite atacuri. De exemplu se pot analiza structuri precum *$_GET,*

[6] repozitoriul principal de pachete PHP

$_POST, $_COOKIE, $_REQUEST, $_FILES, $_SERVER folosite necorespunzător în următoarele contexte:

- Împreună cu funcția *print* poate provoca un atac de tip *Cross-Site Scripting*
- Împreună cu funcția *mysql_*query poate provoca un atac de tip *SQL Injection*
- Împreună cu funcția *include* poate provoca un atac de tip *File Inclusion*
- Împreună cu funcția *eval* poate provoca un atac de tip *Code Execution*
- Împreună cu funcția *system* poate provoca un atac de tip *Command Execution*

Pentru o analiză statică eficientă a codului scris în PHP, se pot recurge la următoarele instrumente descrise în continuare pe scurt.

- *Phploc (phploc/phploc)* - măsoară și analizează structura unui proiect PHP și oferă statistici precum numărul total de linii de cod, complexitatea ciclomatică, numărul de variabile statice sau constante și multe altele. Acesta ajută utilizatorul în comparații cu alte proiecte sau în găsirea unor probleme.
- *Psalm (vimeo/psalm)* - utilizat pentru a detecta erori în aplicații. Este un instrument configurabil și este capabil să detecteze un număr ridicat de probleme într-o aplicație. Analiza lui se efectuează pe baza unui arbore abstract de sintaxă construit din codul sursă al proiectului. Acesta are două moduri de scanare a unui fișier: primul este "*shallow scan*" unde sunt urmărite signaturile funcțiilor, tipurile returnate, constantele și moștenirile între clase iar al doilea este "*deep scan*" unde sunt analizate dependințele invocate în urma unor apeluri de funcții.
- *The Parse (psecio/parse)* - utilizat pentru a detecta vulnerabilități pe partea de securitate. Acesta parsează toate fișierele sursă și avertizează utilizatorul atunci când sunt detectați anumiți factori ce pot afecta securitatea proiectului. În continuare, vor fi prezentați cei mai importanți factori care reprezintă vulnerabilități și care sunt detectați de acest instrument:
    - Utilizarea funcției *eval*. Această funcție primește la parametru un *string* care va fi evaluat precum un cod PHP. Din această cauza ea este vulnerabilă la un atac de tipul *Remote Code Execution* ce este constituit dintr-un script care nu validează în mod corespunzător datele trimise de catre utilizator. Astfel, un atacator poate introduce un cod malițios (incluzând și comenzi pentru sistemul de operare) și acesta va fi executat cu aceleași permisiuni precum ale serviciului web corespunzător. Următorul exemplu de cod PHP

descrie acest tip de atac. În acest caz, un atacator poate trimite *input*-ul "*10; system(cât /etc/passwd)*" și astfel va fi printat conținutul fișierului passwd.

- Utilizarea lui *$_REQUEST* (*request object*) pentru a obține *input*-ul utilizatorului. *$_REQUEST* reprezintă un obiect compus din parametrii trimiși în urma cererii HTTP de tip GET (din *$_GET*) și POST (din *$_POST*) dar și din obiectul *$_COOKIE* unde se află *cookie*-urile trimise în cerere. Prezența lui *$_COOKIE* face ca utilizarea obiectului *$_REQUEST* să fie de evitat în acest caz deoarece *cookie*-urile vor suprascrie parametrii trimiși de utilizator care au același nume cu ele, lucru nedorit de niciun dezvoltator pe partea de server.
- Utilizarea funcției *mysql_real_escape_string*. Aceasta are rolul de a face *escaping* pe caractere speciale precum: \x00, \n, \r, \, ', " și \x, dintr-un string ce va fi introdus într-o comandă SQL, pentru a preveni atacuri de tip *SQL Injection*. Problema acestei funcții este că funcționează doar dacă variabila ce conține string-ul pentru *query* este împachetată în ghilimele. Astfel, următorul cod PHP este vulnerabil la un atac de tip *SQL Injection*.

```
<?php

$bid = mysql_real_escape_string( $_GET['id'] );
$query = 'SELECT title FROM books WHERE id = ' . $bid;

?>
```

[6]

- Introducerea în cod a unor valori sensibile (*tokeni*, parole, etc.). De exemplu următorul cod PHP va genera două probleme și anume folosirea celor două valori *hard-codate*.

```
<?php
 $username = 'mySecretUsername';
 $password = 'mySecretPassword';
?>
```

2.11 **Xdebug & Webgrind**

Xdebug este o extensie pentru PHP care oferă dezvoltatorului funcționalitatea de *debugging* dar și modalități de *profiling*. *Profilerul* Xdebug este un instrument puternic care analizează codul PHP și poate determina părți ale codului care sunt lente și care ar putea fi îmbunătățite de către dezvoltator. Poate fi activat din fișierul *php.ini* prin setarea *xdebug.profiler_enable* = 1 care îl instruiește să scrie rezultatele în directorul configurat în *xdebug.profiler_output_dir*. Rezultatul întors de *profiler* constă în fișiere de tip Cachegrind [7] care pot fi interpretate de către o soluție web numită Webgrind. [7]

Webgrind oferă un server HTTP capabil să transforme fișierele de tip Cachegrind aflate în directorul din *xdebug.profiler_output_dir* într-un format plăcut utilizatorului. În Figura 2 putem vedea un exemplu cu pagina principală a serverului Webgrind. Cele patru coloane din tabel: "*Functions*", "*Invocation Count*", "*Total Self Cost",* "*Total Inclusive Cost*" reprezintă în ordine: numele funcției apelate în program, numărul apelurilor funcției respective, raportul timpului de execuție dar fără alte funcții externe folosite respectiv raportul timpului de execuție cu tot cu funcțiile externe folosite. În acest exemplu observăm că funcția "*call*" este cea mai costisitoare, cu un raport de 80.56 (din 100), cu mult peste restul.

De asemenea, putem vedea exact fișierul și linia de cod de unde această funcție este apelată dar și cauza pentru care este așa costisitoare, adică o problemă în fișierul "/*modals/notification.php*". În Figura 3 a fost deschisă fereastra (prin *click* pe iconița din dreapta coloanei "*Total call cost*") unde ne este ilustrată exact funcția pe care o analizăm, fișierul din care face parte iar linia de cod cea mai lentă este hașurată cu altă culoare. Webgrind mai oferă și un graf al apelurilor unde putem inspecta ordinea în care funcțiile sunt apelate. Nodurile sunt reprezentate de funcții iar o muchie între două noduri reprezintă de fapt un apel. Pentru fiecare muchie este prezentat și timpul necesar acelui apel, raportat la timpul total. Semnificația culorilor atât din acest graf, cât și din pagina principală din Figura 4 este în felul următor: albastru pentru funcții PHP interne, verde pentru metodele ce aparțin unei clase, portocaliu pentru funcții procedurale și gri pentru timpul de execuție a unor importuri de fișiere PHP. Graful apelurilor poate fi vizualizat în urma apăsării butonului "*Show Call Graph*" din dreapta sus iar graful asociat exemplului de mai jos este ilustrat în Figura 4.

[7] fișiere ce vor începe mereu cu prefixul "cachegrind.out." și se pot termina cu PID-ul proceselor PHP

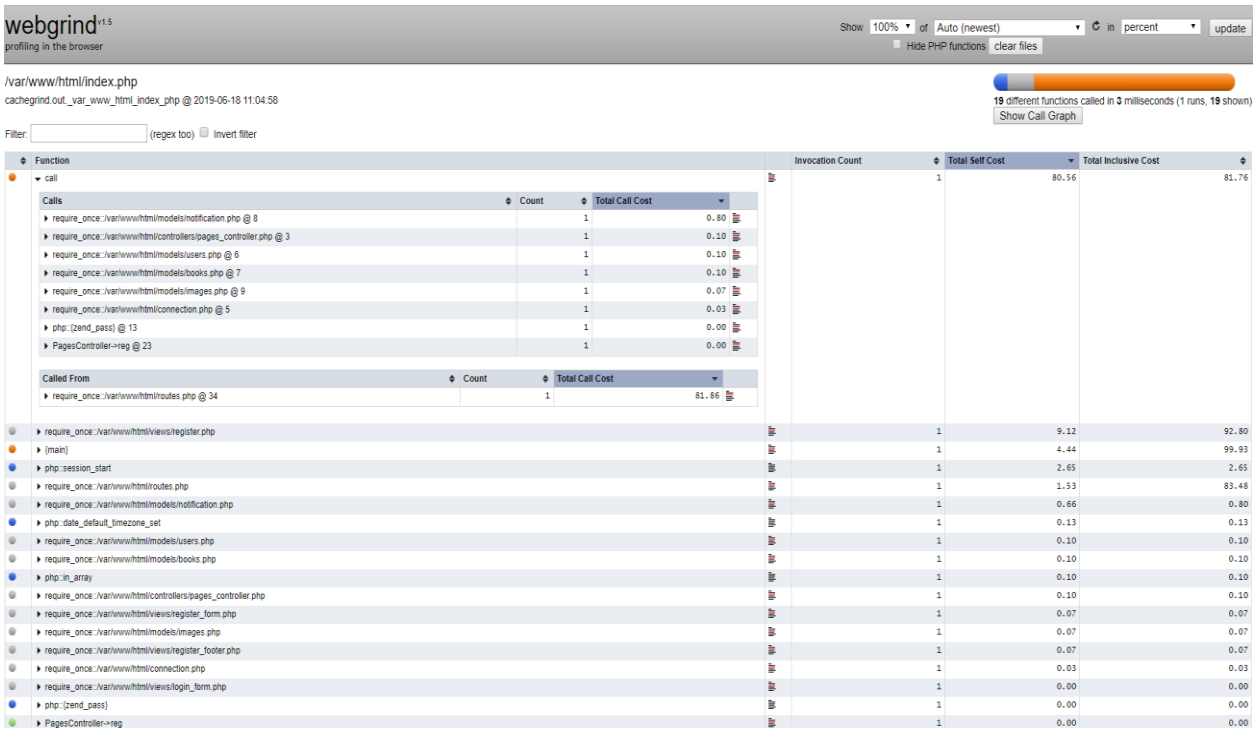

Figura 2 – webgrind pagina principală

Figura 3 – webgrind, vizualizarea codului lent

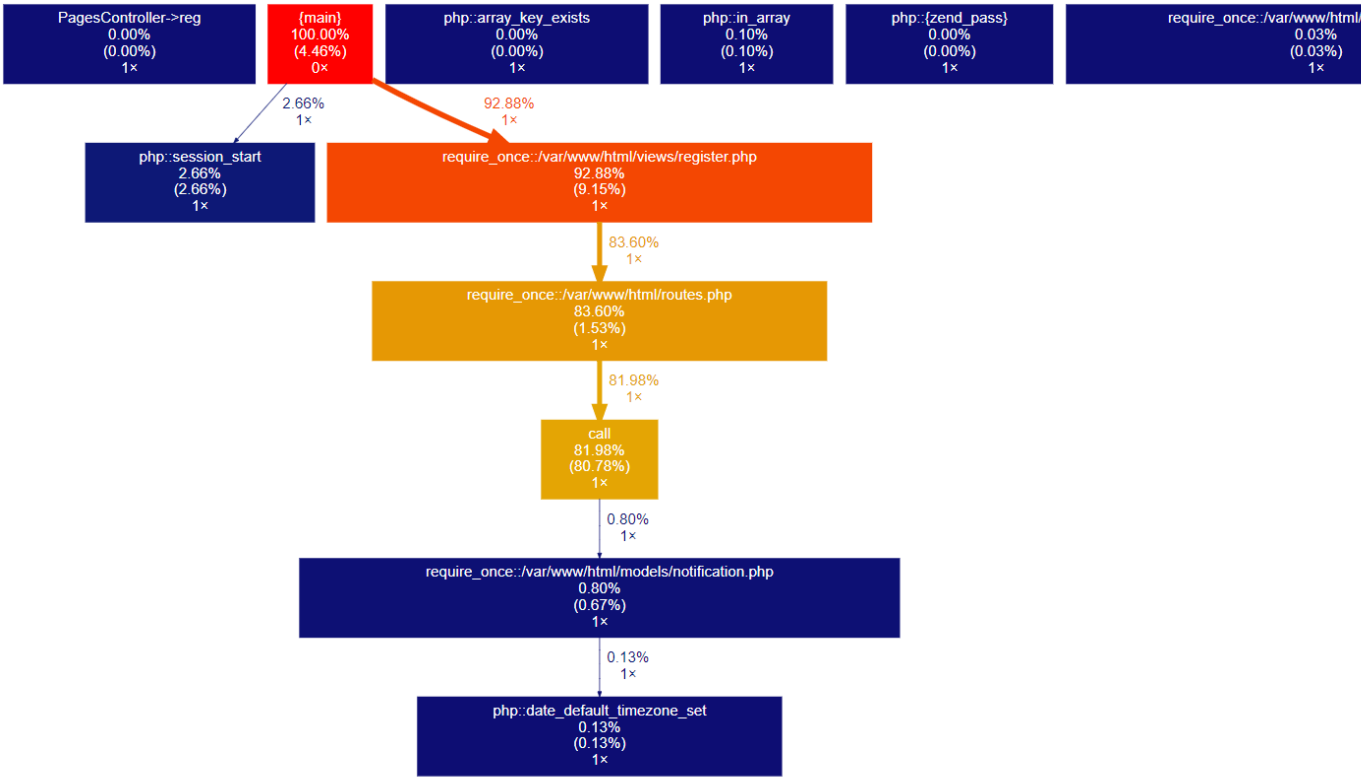

Figura 4 – webgrind, graful apelurilor

3 Scopurile si cerințele aplicației

Aplicația este compusă din multiple tehnologii, biblioteci și *framework-uri* și poate fi accesată de către orice navigator modern.

În primul rând, *backend-ul* aplicației este constituit dintr-un sistem de stocare nerela țional (MongoDB), un set restrâns de instrumente care analizează static codul PHP împreună cu un set de algoritmi capabili să gestioneze și să interpreteze rezultatele acelor instrumente dar și de servicii web scrise în Node.Js ce joacă roluri vitale în ciclul de viață al aplicației precum comunicarea cu Docker dar și *port management* sau *CRUD operations* cu baza de date.

În al doilea rând, partea de *front-end* este reprezentată de o aplicație scrisă în Javascript utilizând *framework-uri* precum Vue.js (sau Vue) și Vuetify (*framework* pentru aplicații scrise în Vue). Pentru cea mai bună experiență a utilizatorului, paginile web au fost dezvoltate într-un mod *responsive* prin intermediul tehnologiilor HTML5 și CSS3. Sunt ușor de folosit și de vizualizat, indiferent de ecran sau de tipul dizpozitivului: desktop, tabletă sau telefon. Apelurile către *back-end* se efectuează cu ajutorul bibliotecii *vue-axios,* care joacă rolul unui *wrapper* ce integrează *axios* (client HTTP) în Vue. De asemenea, interfața este servită de către Nginx (server web) care joacă rolul de *reverse proxy*.

Nu în ultimul rând, Docker joacă un rol esențial deoarece oferă un mod de a rula o aplicație într-un mediu izolat.

Mai departe vom discuta despre necesitatea unei astfel de aplicații cât și despre modul de rezolvare al acesteia.

3.1 Necesitatea aplicației

Aplicația ACSP vine în ajutorul tuturor dezvoltatorilor de aplicații PHP (în special al aplicațiilor web ce se ocupă cu date sensibile dar nu numai) care doresc să beneficieze de un nivel în plus de siguranță. Aceasta integrează modalități de detectare a vulnerabilităților de securitate (*SQL injection*, *XSS attack*, etc.), o analiză detaliată asupra volumului de cod (lucru ce ar putea indica dezvoltatorului o potențială nevoie de refactorizare), detecție de eventuale greșeli ce ar putea produce erori în momentul rulării dar și un mod eficient (prin Docker) și rapid de *deployment* orientat pe nevoile dezvoltatorului care își va putea simula aplicația într-un mediu complet izolat. Toate aceste funcționalități sunt integrate într-o singură platforma unde clientul își poate face managementul proiectelor sale.

În concluzie, necesitatea aplicației provine din capabilitatea acesteia de prevenire a unor evenimente ce ar putea cauza pierderi financiare sau pierderea de utilizatori.

3.2 **Modul de rezolvare a problemei**

În prima fază, aplicația pune la dispoziție diferite instrumente de analiză statică a codului și vizualizare ale acestor metrici într-un format comun dar și posibilitatea de a vedea exact liniile de cod și fișierele unde au fost detectate probleme. În plus, pe partea de analiză din punct de vedere al securității, pe lângă detectarea vulnerabilităților, clientul primește și unele recomandări (*best practices,* precum evitarea *hard*-codării valorilor sensibile sau evitarea folosirii anumitor funcții care sunt vulnerabile la atacuri de tip injecție pentru a-i fi de folos în fixarea problemelor și a nu se mai repeta pe viitor).

De asemenea, aplicația poate detecta eventuale greșeli din cod înainte ca aceasta să fie rulată, aici fiind incluse erori la compilare dar și cele la *run-time* (un exemplu ar fi utilizarea anumitor metode dintr-o variabilă nedefinită). În felul acesta clientul ar putea preveni breșe în securitatea proiectului său (vulnerabilitate la *SQL Injection* de exemplu) dar și greșeli din cod nevăzute de către dezvoltator, înainte de a face *deploy*-ul într-un mediu de producție unde ar putea avea de suferit.

În faza a doua, în condițiile în care clientul a vizualizat și fixat eventualele probleme detectate, îi este oferită funcționalitatea de *deployment* al proiectului într-o zonă complet izolată unde poate simula unele scenarii de producție.

4 Analiză si proiectare

În cadrul aplicației ACSP s-a pus accentul pe următoarele aspecte:

- Utilizarea unui format comun pentru instrumentele de analiza statică a codului si vizualizarea acestora într-un mod plăcut
- Automatizarea procesului de detectare al vulnerabilităților
- O modalitate de *deployment* cât mai eficientă si rapidă
- O experiență plăcută a utilizatorului pe platformă

Următoarea figură descrie o arhitectură generală a aplicației pe server.

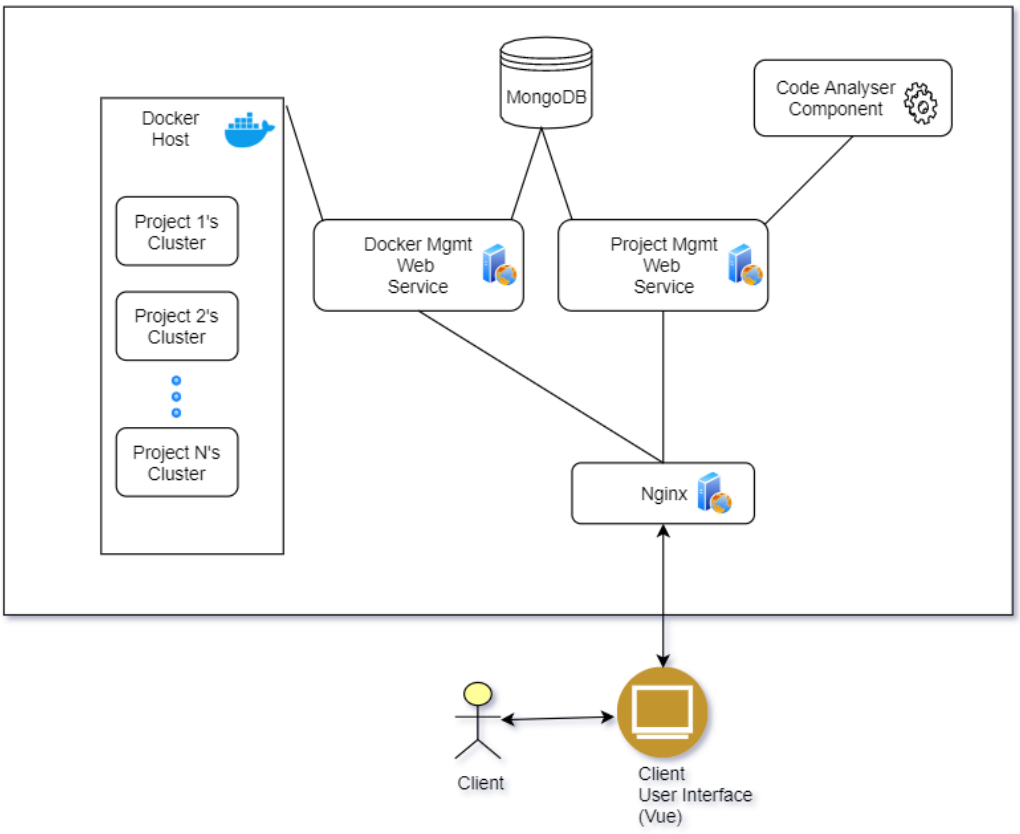

Figura 5 – arhitectura generală a aplicației

În Figura 6 este descrisă arhitectura unui asemenea *cluster*.

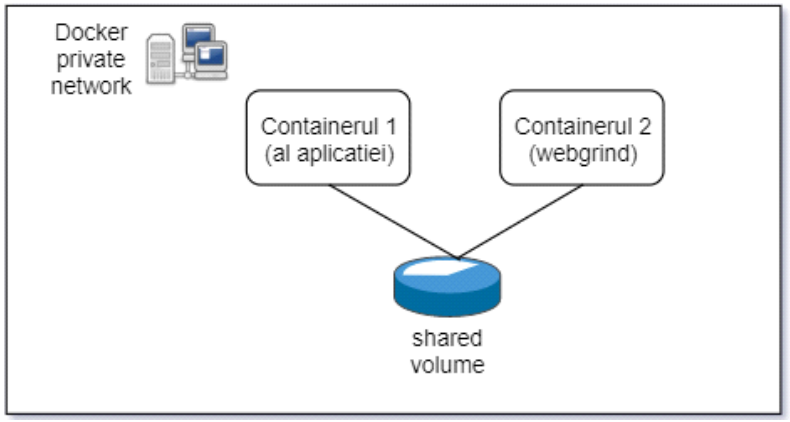

Figura 6 – arhitectura unui cluster

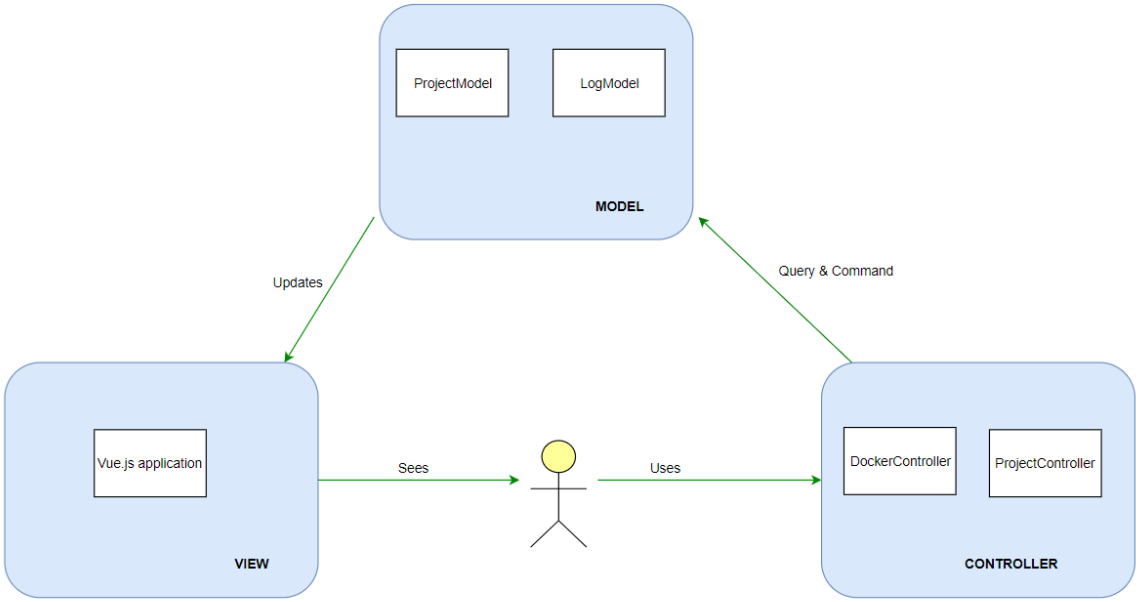

Figura 7 – arhitectura Model-View-Controller

4.1 Serverul *Docker Mgmt Web Service*

Acest server se ocupă cu managementul mediului de *sandbox* al unui proiect încărcat în sistem de către un utilizator. La acest pas, proiectul este deja salvat pe disc de către *Project Mgmt Web Service* care i-a adăugat și cele trei fișierele de configurare necesare descrise la punctul următor.

Clientul trimite serverului un identificator pentru proiectul pentru care se dorește o acțiune împreună cu metoda corespunzătoare.

Serverul pe baza acelui identificator, preia din baza de date toate informațiile acelui proiect și rulează diferite comenzi docker.

Sunt expuse clientului următoarele metode:

- *start* - se rulează comanda *docker-compose up* care pornește cele două containere (aplicația clientului împreună cu serverul Webgrind) definite în fișierul de configurare aflat în directorul sursă. În urma acestei operații, *status*-ul proiectului va deveni *running***.**
- *restart* - se rulează comanda *docker-compose restart* care va executa o operație de restart pe cele două containere.
- *pause* - se rulează comanda *docker-compose pause* care va opri temporar cele două containere, astfel clientul nu-și va mai putea accesa aplicația până se apelează iar metoda *start*. În urma acestei operații, *status*-ul proiectului va deveni *not running.*
- *remove* - se rulează comanda *docker-compose down -v* care va opri și șterge cele două containere împreună cu imaginile și volumele (prin utilizarea opțiunii "*-v*") acestora. În urma acestei operații, *status*-ul proiectului va deveni *not running***.**

In Figura 8 putem vedea diagrama de clase a acestui serviciu iar în continuare vom explica rolul fiecărei componente.

- *RequestExecuter* – acesta reprezinta nivelul cel mai ridicat, ce interacționează cu clientul și este responsabil cu procesarea tuturor comenzilor.
- *DockerExecuter* – această componentă este folosită de *RequestExecuter* și are rolul de a pregăti și configura comenzile de Docker pentru a fi executate de *ProcessExec*.
- *ProcessExec* – are rolul unui *wrapper* peste o bibliotecă specializată în executarea de comenzi la terminal și facilitează interacțiunea cu aceasta.
- *ProjectDAO* – are rolul unui *Data Access Object* și separă operațiile de nivel scăzut cu baza de date, de nivelul ridicat reprezentat de clasa *RequestExecuter*.
- *Project* – această componentă semnifică modelul unui proiect încărcat de un utilizator pe platformă și este reprezentată de stările si acțiunile descrise în imagine.

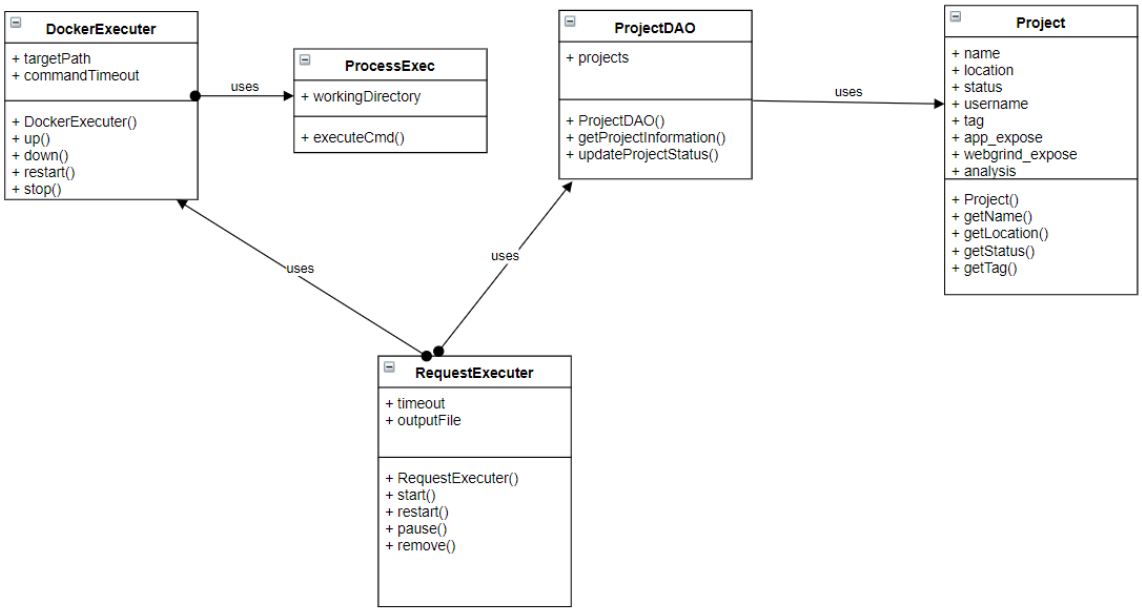

Figura 8 – diagrama de clase, Docker Mgmt

4.2 Serverul *Project Mgmt Web Service*

Acest server expune clientului următoarele metode:

- *fileUpload* – oferă posibilitatea încărcării unui proiect în sistem, împreună cu toate informațiile asociate precum nume sau tipurile de analize dorite. Fișierul încărcat trebuie să fie în format *.zip* și trebuie să conțină directorul în care se află proiectul. Acesta va fi dezarhivat într-o locație specială unde vor fi adăugate și cele două fișiere de configurare pentru Docker: *Dockerfile* și *docker-compose.yml*. Următorul pas va fi executarea componentei de analiză statică a codului sursă în funcție de preferințele clientului, urmat de adăugarea proiectului împreună cu toate rezultatele obținute în baza de date.
- *listProjects* – oferă posibilitatea listării tuturor proiectelor incărcate de un client pe platformă, împreună cu toate informațiile necesare.
- *listLogs* – oferă posibilitatea listării tuturor informațiilor necesare în pagina de istoric.
- *deleteProject* – oferă posibilitatea ștergerii unui proiect de pe platformă.

De asemenea, în momentul încărcării unui proiect serverul este responsabil cu asignarea acestuia a două port-uri necesare pentru rularea aplicației clientului dar și pentru serverul

HTTP Webgrind. Algoritmul constă în alegerea primelor două valori nefolosite dintr-un interval prestabilit și marcarea acestora în baza de date.

În Figura 9 putem vedea diagrama de clase a acestui serviciu și mai departe vom explica rolul fiecărei componente din imagine.

- *RequestExecuter* – acesta reprezinta nivelul cel mai ridicat, ce interacționează cu clientul și este responsabil cu procesarea tuturor comenzilor.
- *StaticAnalyser* – această componentă este folosită de *RequestExecuter* și are rolul de a pregăti instrumentele de analiză statică și de a le rula pe un proiect incărcat pe platformă.
- *Sandbox* – este folosit de *RequestExecuter* și se ocupă cu pregătirea configurațiilor de *deployment* pentru un proiect.
- *FileHandler* – are rolul unui *wrapper* peste o biblitecă specializată în operații cu fișiere și facilitează interacțiunea cu aceasta prin predefinirea anumitor parametri și înglobarea a mai multe functionalități într-o metodă.
- *ProjectDAO* și *LogDAO*– au rolul unui *Data Access Object* și separă operațiile de nivel scăzut cu baza de date, de nivelul ridicat reprezentat de clasa *RequestExecuter*.
- *Project* și *Log* – aceaste componente semnifică modele pentru proiectele încărcate de utilizatori pe platformă, respectiv pentru operațiile efectuate pe platformă și sunt reprezentate de stările si acțiunile descrise în imagine.

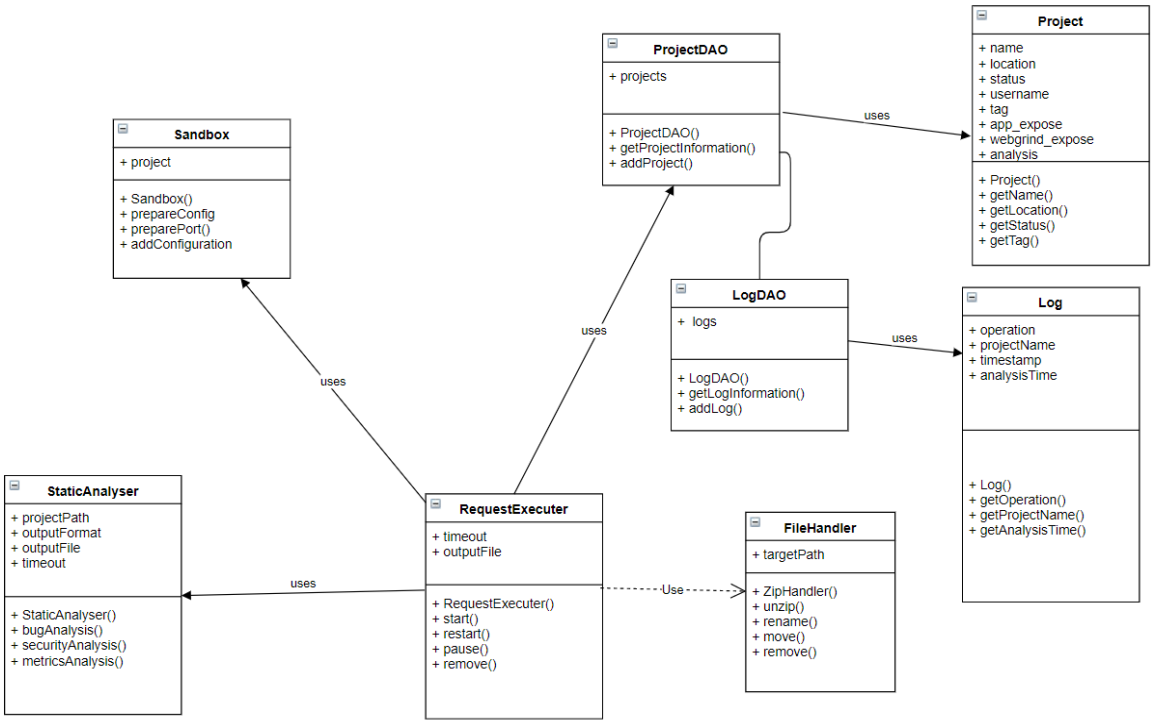

Figura 9 – diagrama de clase, Project Mgmt

4.3 Aplicația *front-end*

Arhitectura aplicației *front-end* este compusă din diferite componente de Vue, fiecare având un rol bine stabilit.

- Componenta "*MainMenu*" este o poartă către următoarele două componente descrise și este reprezentată vizual de cele două butoane: "*MY PROJECTS*" și "*HISTORY*".

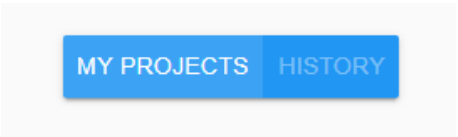

Figura 10 – componenta MainMenu

- Componenta "*MyProjects*" este utilizată în momentul în care utilizatorul apasă pe butonul "*MY PROJECTS*" și este compusă dintr-un tabel ce conține toate proiectele încărcate de client împreună cu funcționalitățile oferite de aplicația ACSP dar și de

butonul “*New project*” ce activează componenta “*NewProject*” ce va fi descrisă în cele ce urmează. Tabelul este format din următoarele coloane:

- “*Project name*” reprezintă numele unic ales de utilizator pentru proiectul respectiv.
- “*Latest analysis*” este reprezentat de o iconiță albastră cu caracter informativ si care atunci cand este apăsată redirectează utilizatorul către o pagină dedicată unde pot fi vizualizate rezultatele instrumentelor de analiză statică a codului pentru proiectul corespunzator.
- “*Status*” indică prin valoarea “*running*” faptul că aplicația respectivă rulează în acel moment intr-un *sandbox* sau “*not running*” altfel.
- “*Actions*” este compus din patru butoane ce oferă funcționalitățile urmatoare: restart, pornirea sau rularea aplicației într-un *sandbox*, oprirea temporara a aplicației, ștergerea unei aplicații.

De asemenea, atunci când un proiect este în status “*running*” rândul său poate fi expandat, astfel permițând accesul la cele două *endpoint*-uri unde rulează aplicația clientului într-un mediu izolat dar și serverul Webgrind ce expune metricile în timp real. În Figura 11 poate fi vizualizat un exemplu pentru o aplicație al cărui rând este expandat.

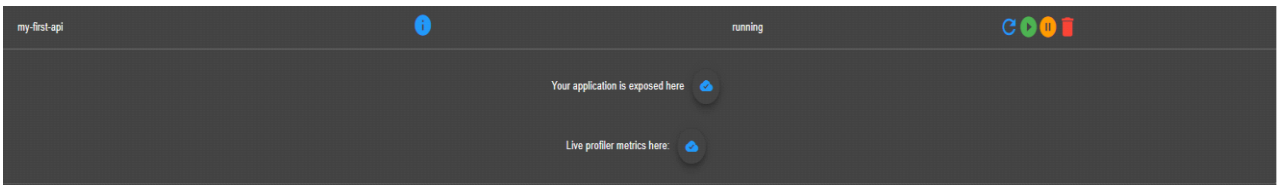

Figura 11- componenta MyProjects, proiect expandat

În Figura 12 putem vedea o privire de ansamblu al acestei componente.

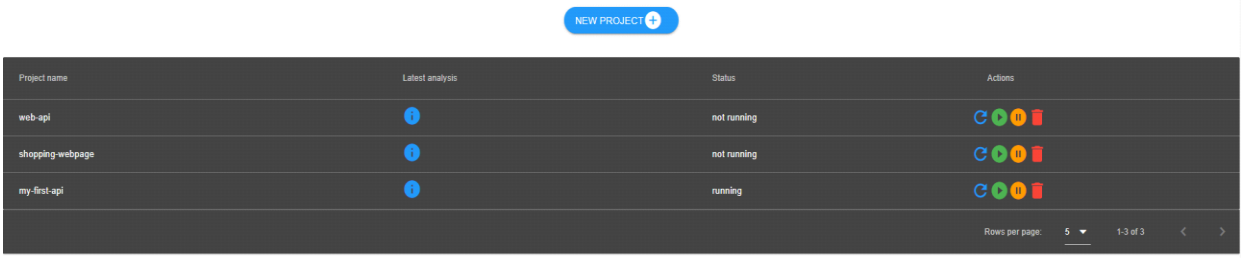

Figura 12 – componenta MyProjects, privire generală

- Componenta "*StaticInfo*" oferă utilizatorului un mod de a vizualiza rezultatele instrumentelor de analiză statică alese. Această componentă este activată atunci cand utilizatorul apasă pe butonul albastru din coloana "*Latest analysis*" din dreptul unui anumit proiect.

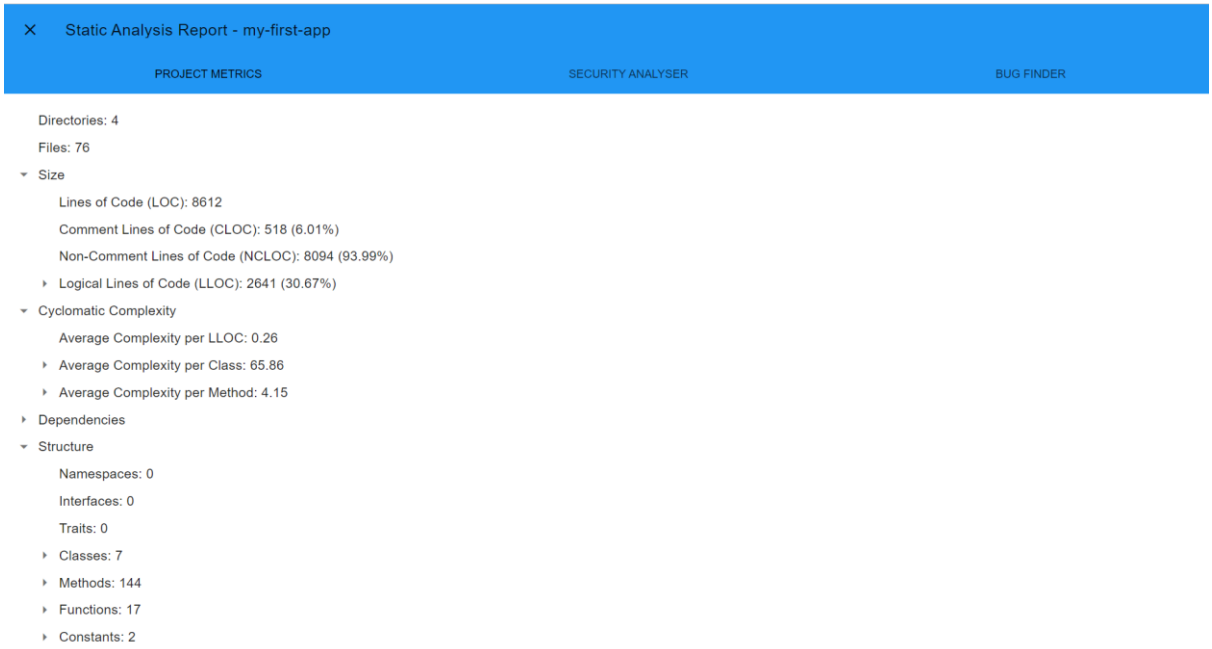

Figura 13 – raportul analizei statice a codului. Project metrics

În Figura 13 putem vedea un exemplu de raport al instrumentului de analiză statică *"Project metrics"*. Se pot inspecta metrici despre complexitatea ciclomatică a codului, informații despre dependințele utilizate, modul de structurare pe fișiere, directoare, clase etc. și informații despre mărimea proiectului.

În Figura 14 putem vedea un exemplu de raport al instrumentului de analiză statică *"Security analyser"*. Pentru fiecare vulnerabilitate găsită este specificat fișierul din care face parte împreuna cu numărul liniei dar și codul de pe linia respectivă cu o scurtă descriere a problemei.

În Figura 15 putem vedea un exemplu de raport al instrumentului de analiză statică *"Bug finder"*. Modul de vizualizare este asemanator cu cel al instrumentului *"Security analyser"*.

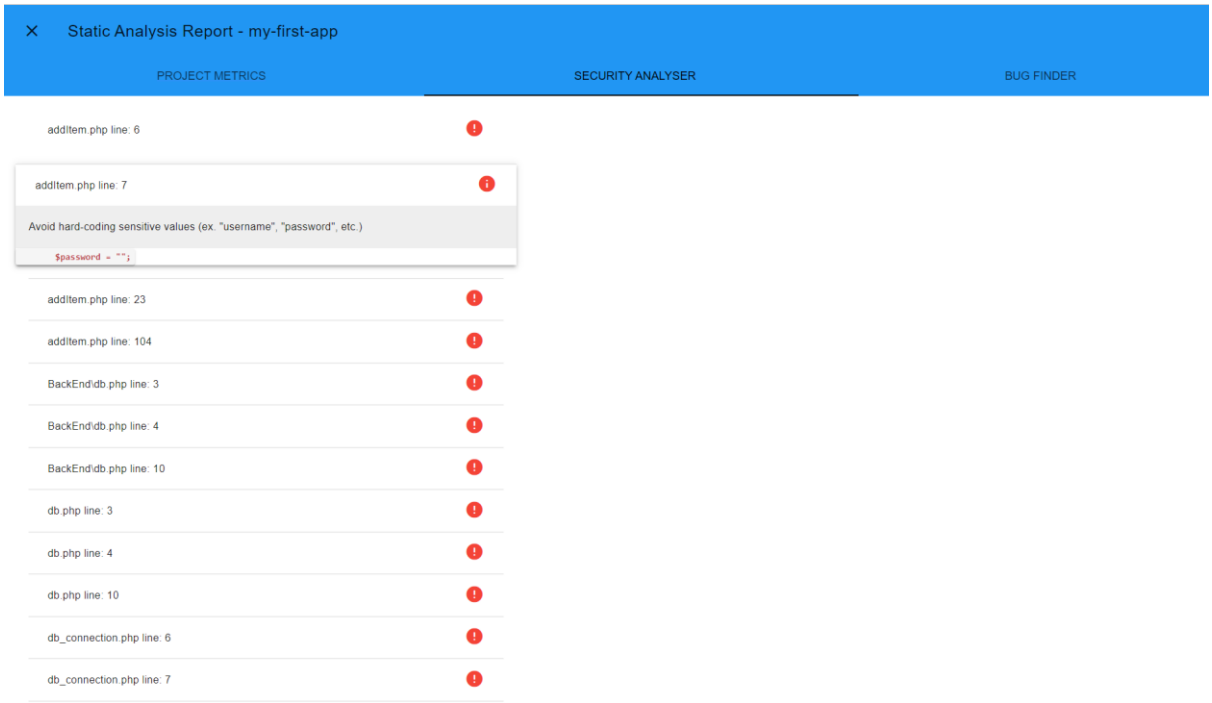

Figura 14 - raportul analizei statice a codului. Security analyser

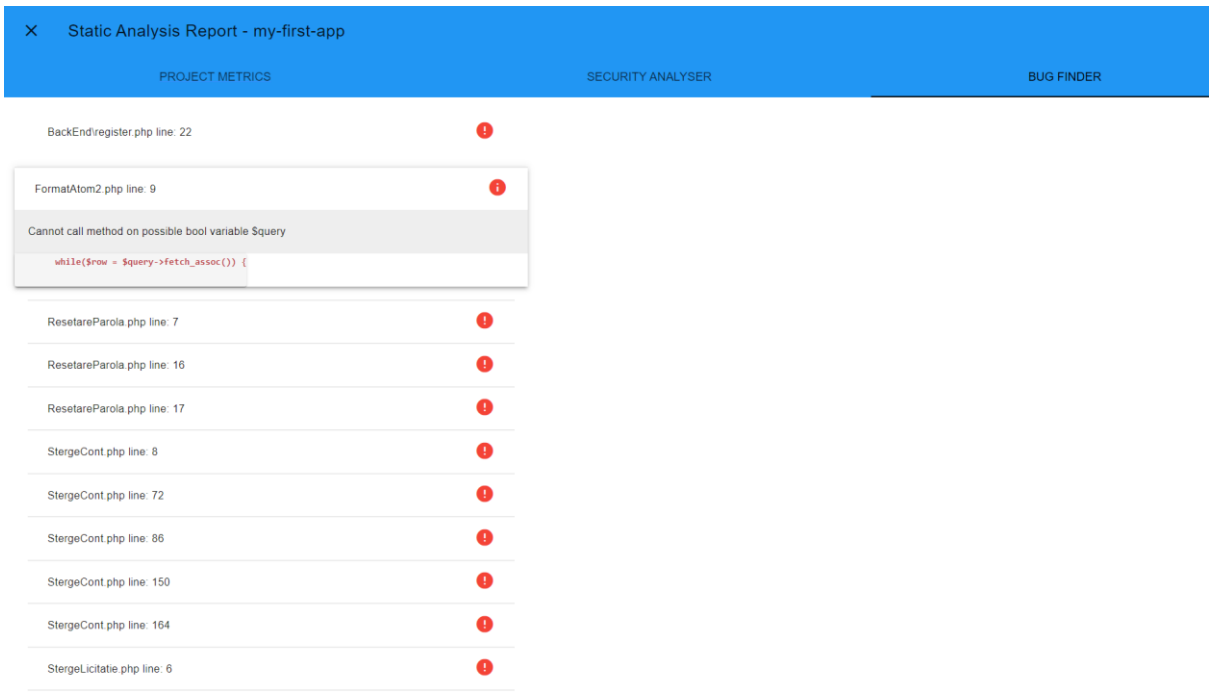

Figura 15 - raportul analizei statice a codului. Bug finder

- Componenta "*History*" oferă utilizatorului un istoric al operațiilor de *create* si *remove* efectuate pe aplicația ACSP. De asemenea, sunt afișați utilizatorului timpii de execuție pentru fiecare instrument ales.

Figura 16 – componenta History

- Componenta "*NewProject*" permite utilizatorului să își adauge un proiect nou. Aceasta este activată cand butonul "*New project*" este apăsat, moment în care va apărea *dialog*-ul din Figura 17.

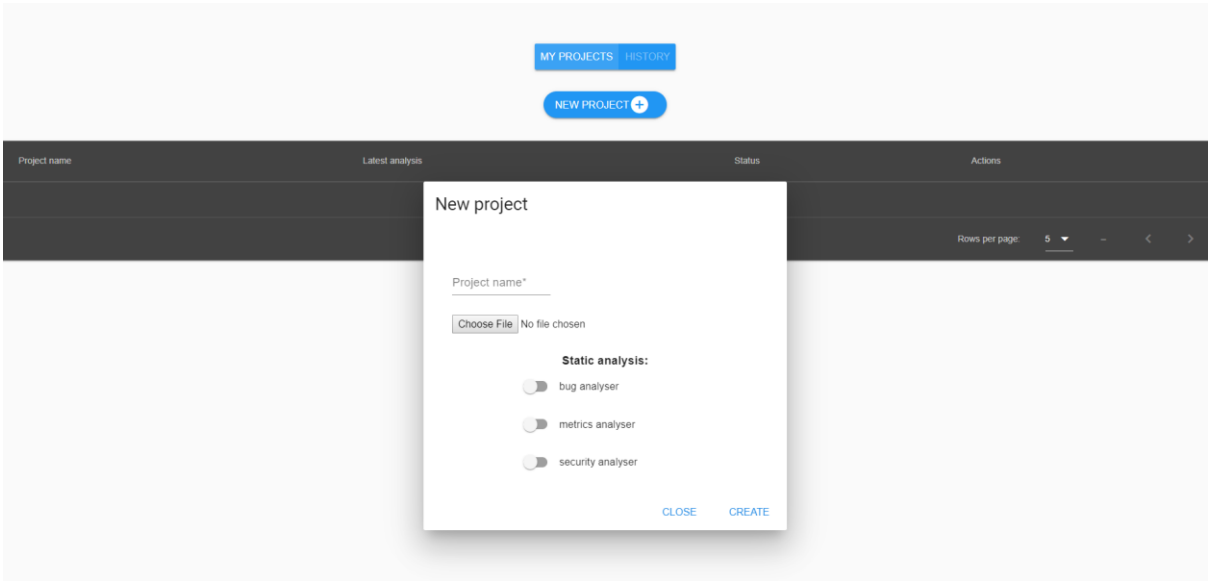

Figura 17 – componenta NewProject

Comunicarea cu serverul din *back-end* "*Docker Mgmt Web Service*" este realizată din componenta "*MyProjects*" ca urmare a utilizării celor patru acțiuni de către client astfel:

- În urma unei acțiuni de *restart* pe o anumită aplicație, o cerere AJAX [8] către calea "*/dockerMmgt/restart/tag=${tag}"* va fi efectuată și componenta "*MyProjects*" va fi actualizată corespunzător pentru ca utilizatorul să poată vizualiza rezultatul produs în urma acțiunii alese.
- În urma unei acțiuni de *start* pe o anumită aplicație, o cerere AJAX către calea "/*dockerMmgt/start/tag=${tag}"* va fi efectuată și componenta "*MyProjects*" va fi actualizată corespunzător pentru ca utilizatorul să poată vizualiza rezultatul produs în urma acțiunii alese.

- În urma unei acțiuni de *pause* pe o anumită aplicație, o cerere AJAX către calea "*/dockerMmgt/restart/pause=${tag}*" va fi efectuată și componenta "*MyProjects*" va fi actualizată corespunzător pentru ca utilizatorul să poată vizualiza rezultatul produs în urma acțiunii alese.
- În urma unei acțiuni de *remove* pe o anumită aplicație, o cerere AJAX către calea "/*dockerMmgt/remove/tag=${tag}*" va fi efectuată și componenta "*MyProjects*" va fi actualizată corespunzător pentru ca utilizatorul să poată vizualiza rezultatul produs în urma acțiunii alese.

4.4 **Schema bazei de date**

Baza de date constă în două colecții:

- *applications* – aici sunt stocate în documente informații referitoare la aplicațiile încărcate pe platformă. Următoarele informații sunt stocate într-un document:
    - *name* – numele proiectului
    - *path* – locația pe disc a proiectului
    - *status* – reprezintă starea aplicației
    - *username* – numele utilizatorului
    - *tag* – reprezintă un identificator pentru acest proiect.
    - *app_port* – *port*-ul la care pornește aplicația
    - *webgrind_port* – *port*-ul la care pornește serverul Webgrind
    - *app_expose* – URL-ul la care este disponibilă aplicația
    - *webgrind_expose* – URL-ul la care este disponibil serverul Webgrind

[8] cerere HTTP asincronă

    - *analysis* – rezultatele obținute în urma analizei statice a codului sursă
- *logs* – aici sunt stocate în documente informații referitoare la operațiile efectuate de un utilizator pe un proiect. Următoarele informații sunt stocate într-un document:
    - *operation* – tipul operației efectuate de utilizator
    - *name* – numele proiectului
    - *user* – numele utilizatorului
    - *time* – data efectuării operației
    - *additionalInfo* – timpul de execuție a operației

5 Implementare

În acest capitol este descrisă implementarea aplicației ACSP.

5.1 Serverul *Docker Mgmt Web Service*

Această entitate constă într-un server HTTP creat prin intermediul pachetului *http* și oferă clientului următoarele rute:

- *'POST /dockerMgmt/start/tag=${tag}'*
- *'POST /dockerMgmt/restart/tag=${tag}'*
- *'POST /dockerMgmt/remove/tag=${tag}'*
- *'POST /dockerMgmt/pause/tag=${tag}'*

Tag-ul unui proiect este reprezentat de concatenarea prin *underscore* a numelui utilizatorului cu numele proiectului.

Se utilizează biblioteca *child_process* pentru a se putea executa comenzi de terminal precum *docker-compose up* sau *docker-compose down.*

5.2 Serverul *Project Mgmt Web Service*

Acest server este creat tot prin intermediul pachetului *http* și oferă clientului următoarele rute:

- *'POST /projectMgmt/fileUpload'* – folosită pentru încărcarea în sistem a unui fișier în format *.zip.* Informațiile suplimentare legate de proiectul încărcat vor fi trimise ca headere. Headerele trimise sunt următoarele:
  - *Content-Type: 'multipart/form-data'* – specificarea formatului conținutului
  - *projectName: <nume proiect>* - specificarea numelui proiectului
  - *userName: <nume utilizator>* - specificarea numelui utilizatorului
  - *psalm: <true / false>* - se specifică dacă o analiză de tip *psalm* va fi efectuată
  - *psecio: <true / false>* - se specifică dacă o analiză de tip *psecio* va fi efectuată
  - *phploc: <true / false>* - se specifică dacă o analiză de tip *phploc* va fi efectuată
- *'POST /projectMgmt/deleteProject'* – folosită pentru ștergerea unui proiect din sistem. Conținutul cererii este în format JSON si conține următorii parametri:

    - *userName: <nume proiect>* - numele utilizatorului
    - *projectName: <nume proiect>* - numele proiectului care se dorește șters
- *'GET /projectMgmt/listProjects'* – folosită pentru listarea proiectelor. În headere trebuie specificat următorul câmp:
    - *userName: <nume utilizator>* - numele utilizatorului
- *'GET /projectMgmt/listLogs'* – folosită pentru listarea *log*-urilor. În headere trebuie specificat următorul camp:
    - *userName: <nume utilizator>* - numele utilizatorului

De asemenea, pentru diferite funcționalități s-au ales urmatoarele biblioteci:

- *path* – pentru manipularea căilor fișierelor
- *extract-zip* – pentru dezarhivarea fișierelor *.zip*
- *fs* – pentru diferite operații cu fișiere

5.3 *Deployment* & Configurări Docker

După cum am văzut în Figura 6, în momentul în care utilizatorul dorește să efectueze un *deploy* al aplicației sale, este creat un *cluster* format din următoarele două componente:

- Containerul de bază unde rulează efectiv aplicația clientului. În acest container rulează o imagine de PHP 7.2 cu extensia *XDEBUG* activată. În *background*, rulează *profiler*-ul care analizează execuția codului PHP și scrie în fișiere de tip Cachegrind într-un anumit director la care este configurat să asculte și componenta urmatoare.
- Containerul Webgrind este expus public și oferă clientului un server web unde pot fi vizualizate metricile întoarse de *profiler*-ul PHP activat pe aplicație. Acesta este configurat să asculte la directorul menționat în prima componentă.

Următoarele fișiere sunt generate și adăugate de către *Project Mgmt Web Service* la fiecare încărcare a unui proiect pe platformă:

- Dockerfile – conține toate comenzile necesare creării unui container unde să ruleze aplicația clientului. Prima linie definește imaginea de bază care este *mobtitude/php-xdebug:7.2-apache* si constă într-un mediu virtual cu urmatoarele funcționalități:
    - Limbajul PHP 7.2 instalat
    - Extensia de PHP Xdebug activată
    - Serverul HTTP Apache este instalat

Următoarele două linii din fișier reprezintă instrucțiuni de copiere a unor fișiere sursă dintr-o anumită locație în imaginea de Docker. Urmează două instrucțiuni ce oferă imaginii toate permisiunile pe cele două directoare. Penultima instrucțiune activează modulul *mod_rewrite* necesar serverului de Apache. În final, se specifică *port*-ul la care va porni containerul în rețeaua internă Docker.

```
FROM mobtitude/php-xdebug:7.2-apache

COPY ./xdebug.ini /usr/local/etc/php/conf.d/xdebug.ini
COPY . /var/www/html/

RUN chmod 777 /tmp
RUN chmod 777 /var

RUN a2enmod rewrite

EXPOSE 80
```

- xdebug.ini – acesta este fișierul specificat la linia doi din Dockerfile și constă în diferite setări pentru extensia Xdebug.

```
xdebug.profiler_enable = 1
xdebug.profiler_output_dir = /tmp
xdebug.profiler_output_name = cachegrind.out.%s
xdebug.profiler_append=0
xdebug.profiler_enable_trigger = 0
xdebug.trace_output_dir = /tmp
```

- docker-compose.yml – acesta este un fișier de configurare ce pornește cele două containere ale unui proiect dintr-un singur pas. Acest fișier este generat dinamic în funcție de *port*-urile asignate unui proiect. În continuare putem vedea un exemplu al acestui fișier pentru un proiect care are asignat *port*-ul 9000 pentru aplicație si 9001 pentru serverul Webgrind. Se observă că imaginea dupa care este creat serverul Webgrind este *tbfisher/webgrind* și că cele două containere create vor împărți un set de

volume tocmai pentru ca containerul Webgrind să aibă acces la fișierele Cachegrind generate din celălalt container, de către extensia Xdebug.

```
version: '3'
services:
 php_9000:
  container_name: "php_9000"
  build: .
  ports:
   - "9000:80"
  expose:
   - "9000"
  volumes:
   - stfiles9000:/tmp
   - srcfiles9000:/var/www/html
 webgrind_9001:
  container_name: "webgrind_9001"
  image: tbfisher/webgrind
  links:
   - php_9000
  volumes:
   - stfiles9000:/tmp
   - srcfiles9000:/var/www/html
  ports:
   - 9001:8080
  environment:
   XDEBUG_OUTPUT_DIR: /tmp
   WEBGRIND_STORAGE_DIR: /tmp
```

5.4 **Aplicația *front-end***

Aplicația a fost dezvoltată utilizând *framework*-ul Vue.js și este compusă din diferite componente care au rolul de a oferi o structură modulară și ușor de urmărit. Cele mai importante biblioteci folosite sunt următoarele:

- *vue-router* – pentru rutarea internă a componentelor
- *vue-axios* – pentru trimiterea de cereri AJAX în *back-end*
- *vue-highlightjs* – pentru afișarea de cod PHP într-un format plăcut

De asemenea, pagina principală a aplicației este expusă pe ruta “/”.

6 Manual de utilizare

În acest capitol va fi prezentat un studiu de caz efectuat pe douăzeci de proiecte de la materia Tehnologii Web a Facultății de Informatică, Iași. Au fost urmărite rezultatele instrumentelor de analiză statică a codului și putem observa următoarele lucruri:

- Următorul tabel prezintă valorile medii rezultate în urma unei analize statice a dimensiunii și structurii proiectelor pentru cele douăzeci de cazuri.

Numărul de directoarelor	13
Numărul de fișiere	105
Numărul liniilor de cod	14205
Numărul liniilor de cod comentate	5706
Numărul liniilor de cod logice	2292
Numărul de clase	69
Numărul de funcții	10
Numărul entităților ce nu sunt clase sau funcții	785
Numărul variabilelor globale	2
Numărul variabilelor statice	30
Numărul apelurilor de funcții	637
Numărul de interfețe	5
Numărul de clase abstracte	1
Numărul de clase concrete	68
Numărul de metode	534
Numărul de metode publice	480
Numărul de metode private	54
Numărul de funcții anonime	3
Numărul de constante	22
Complexitatea ciclomatică per clasă	33
Maximul complexității ciclomatice înregistrată per clasă	110
Complexitatea ciclomatică per metodă	10

Maximul complexității ciclomatice înregistrată per metodă	25

- Pe baza unei analize statice capabile să detecteze greșeli în cod înainte ca acesta să fie rulat, putem observa următoarele aspecte.

Numărul de proiecte cu cel puțin o greșeală gravă în cod	16/20
Numărul de proiecte cu cel puțin o greșeală dar care nu este foarte gravă din cod	13/20
Numărul maxim de greșeli gasite într-un proiect	283
Cele mai frecvente greșeli	*Cannot access array value on non-array variable*
	Possibly undefined global variable
	Cannot concatenate with a possibly null string
	Expected argument of type int but float was provided
	Cannot find file to include
Numărul total de greșeli identificate	627

- Pe partea de vulnerabilități de securitate, analiza statică a dus la identificarea următoarelor metrici:

Numărul de proiecte cu cel puțin o vulnerabilitate de securitate	20/20
Numărul maxim de vulnerabilități detectate într-un proiect	121
Numărul total de vulnerabilități detectate pe toate proiectele	398

Cele mai frecvente vulnerabilități detectate	Folosirea operatorilor *AND* și *OR* în defavoarea operatorilor *&&* și *||* (*AND* și *OR* au precedență mai slabă).
	Hard-codarea valorilor sensibile precum credențialele unei baze de date.
	Se folosește funcția *in_array (*verifică daca un element se găsește într-un vector) fără ultimul parametru *TRUE*, pentru verificarea și a tipului valorii de căutat. PHP este un limbaj slab tipizat și de aceea pentru a fi în siguranță se recomandă verificarea tipului valorii de căutat.
	Se folosește funcția *mysql_real_escape* care este vulnerabilă la *SQL Injection*. Mult mai puternice sunt *prepared statement*-urile.
	Datele primite de la utilizator sunt extrase cu *$_REQUEST*. Recomandarea este să se folosească *$_GET* sau *$_POST* pentru a ști mereu de unde se extrag datele.
	Se folosesc funcții precum *exec*, *system* sau *echo* direct cu datele primite de la utilizator. Acestea sunt vulnerabile la *Code Injection* și ar trebui folosite cu atenție.

7 Concluzii

Acest proiect a fost realizat din lipsa existenței în piață a unei soluții pentru limbajul PHP care să ofere clientului atât detectarea vulnerabilităților printr-o analiză statică a codului dar și posibilitatea efectuării unei analize dinamice printr-un mod eficient de *deployment* a mai multor aplicații.

Au fost acoperite următoarele aspecte:

- Integrarea a trei instrumente pentru analizarea statică a codului, având următoarele roluri:
  - Extragerea unor metrici legate de dimensiunea și structura unui cod sursă.
  - Detectarea unor greșeli în implementare înainte de rularea codului de către client.
  - Detectarea unor vulnerabilități de securitate în codul sursă.
- Expunerea rezultatelor analizei statice a codului într-un format comun.
- O modalitate de *deployment* a mai multor aplicații și rularea fiecăreia într-un mediu complet izolat.
- Activarea extensiei de PHP, Xdebug și a *profiler*-ului care analizează dinamic o aplicație PHP.
- Integrarea unui server extern (Webgrind), ce oferă clientului o vizualizare de ansamblu a rezultatelor analizei dinamice a codului.
- Oferirea unei interfețe web ce înglobeaza toate funcționalitățile acestei aplicații.

Ca direcții de viitor pentru această aplicație ne putem gândi la următoarele aspecte:

- Punerea la dispozitie a unui server SQL, integrat într-un mediu virtual pentru a putea fi accesibil aplicațiilor unui client.
- Extinderea instrumentelor de analiza statică a codului atât pentru detecțiile curente cât și pentru unele noi, cum ar fi analizarea anumitor *design pattern*-uri implementate sau analizarea nivelului de implementare a diferitor standarde de dezvoltare.
- Adăugarea unor detecții dinamice pentru diferite atacuri precum *Brute Force* sau *(Distributed) Denial Of Service*

8 Bibliografie

- https://profs.info.uaic.ro/~dlucanu/cursuri/css/resurse/Saptamana1.pdf [1]
- CLARKE-SALT, Justin. *SQL injection attacks and defense*. Elsevier, 2009. [2]
- GROSSMAN, Jeremiah, et al. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007. [3]
- https://ro.wikipedia.org/wiki/MongoDB [4]
- PEACOCK, Alan T.; WEIR, Ronald; BRIGGS, Asa. *The composer in the market place*. London: Faber Music, 1975. [5]
- https://www.owasp.org/index.php/CRV2_SQLInjPHP [6]
- https://xdebug.org/docs/profiler [7]
- https://www.owasp.org/index.php/Direct_Dynamic_Code_Evaluation_(%27Eval_Injection%27) [8]
- CHESS, Brian; MCGRAW, Gary. Static analysis for security. *IEEE security & privacy*, 2004, 2.6: 76-79. [9]
- MCCABE, Thomas J. A complexity measure. *IEEE Transactions on software Engineering*, 1976, 4: 308-320. [10]
- ANDERSON, Charles. Docker [software engineering]. *IEEE Software*, 2015, 32.3: 102-c3. [11]
- PARAISO, Fawaz, et al. Model-driven management of docker containers. In: *2016 IEEE 9th International Conference on cloud Computing (CLOUD)*. IEEE, 2016. p. 718-725. [12]
- KYRIAKIDIS, Alex; MANIATIS, Kostas. *The Majesty of Vue. js*. Packt Publishing Ltd, 2016. [13]
- MEMBREY, Peter; PLUGGE, Eelco; HAWKINS, DUPTim. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress, 2011. [14]