Reporte análisis estático de código

**Informe detallado de resultados: dvwa**

Control de cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Apartado** | **Resumen de cambios** |
| 1.0 | 02/06/2021 | N/A | Versión inicial |
|  |  |  |  |
|  |  |  |  |

Índice

[1. Introducción 5](#_Toc532203416)

[2. Información de la aplicación 6](#_Toc532203417)

[2.1. Datos generales de la aplicación 6](#_Toc532203418)

[2.2. Datos de calidad 6](#_Toc532203419)

[3. Nivel y modelo de calidad 7](#_Toc532203420)

[3.1. Listado de reglas incumplidas 8](#_Toc532203421)

[4. Tendencias e histórico 9](#_Toc532203422)

[5. Análisis de la aplicación 10](#_Toc532203423)

[5.1. Vulnerabilidades 10](#_Toc532203424)

[5.2. Bugs 10](#_Toc532203425)

[5.3. Code Smells 10](#_Toc532203426)

[5.4. Duplicidad 10](#_Toc532203427)

[6. Siguientes pasos 11](#_Toc532203428)

1. Introducción

Los resultados expuestos en este documento tratan de reflejar el estado de la aplicación dvwa analizada desde el punto de vista de la calidad de su código fuente.

El modelo de calidad aplicado responde a las necesidades planteadas y trata de minimizar el impacto que supone la implantación de un servicio de análisis estático de código en un equipo de desarrollo. La herramienta en la que se apoya el servicio, SonarQube, establece una clasificación en el tipo de problemas detectados atendiendo a la característica de calidad con la que están relacionados y este informe se referirá a ellos en sus mismos términos. De esta forma, surgen los siguientes tipos de problemas:

* ***Vulnerability***. Problemas relacionados con la seguridad de la aplicación y que pueden suponer un riesgo para la integridad de la misma.
* ***Bug***. Problemas relacionados con la fiabilidad de la aplicación, su capacidad para recuperarse de errores inesperados y mantener el servicio.
* ***Code smell.*** Este tipo de problema se relaciona con la mantenibilidad del código fuente, su capacidad para cambiar de una forma eficiente y efectiva.

Así mismo, SonarQube emplea una terminología propia para ciertos aspectos internos que se explica a continuación:

* ***Quality profile.*** Conjunto de reglas de calidad de código que se aplican a un lenguaje en concreto.
* ***Quality gate.*** Métrica o conjunto de métricas asociadas a ciertos valores límite en los cuales se basa SonarQube para valorar el resultado de un escaneo.

De forma adicional a estos conceptos, durante el informe se hará mención a otros aspectos técnicos de la calidad de código que se explican a continuación:

* ***Problema***. Un problema hace referencia a un componente (proyecto, modulo, archivo, bloque de código, etc.) que no cumple con la especificación de una regla de calidad.
* ***Regla cumplida.*** Hace referencia a reglas de calidad de código que pertenecen al modelo de calidad y que no generan ningún problema en el análisis actual.
* ***Regla incumplida.*** Son reglas de calidad de código pertenecientes al modelo de calidad de código aplicado y que generan al menos un problema.
* ***Duplicidad.*** Métrica de calidad de código que recoge el porcentaje de código duplicado que se ha encontrado al analizar la aplicación.
* ***Bloques duplicados.*** Un bloque duplicado es un conjunto de líneas de código que se encuentran al menos dos veces escritos en la aplicación.

1. Información de la aplicación

Esta sección muestra la información de partida para conocer la aplicación que se está valorando, su tamaño y los lenguajes implicados en la misma.

* 1. Datos generales de la aplicación

|  |  |  |
| --- | --- | --- |
| Nombre del proyecto | [dvwa](http://localhost:9000/dashboard?id=dvwa) | |
| Análisis base | 02-06-2021 22:50 | |
| Líneas de código | 52708 LOC | |
| Lenguajes | CSS | 967 LOC (1,83 %) |
| JavaScript | 562 LOC (1,07 %) |
| PHP | 18.789 LOC (35,65 %) |
| Python | 34 LOC (0,06 %) |
| HTML | 31.607 LOC (59,97 %) |
| XML | 749 LOC (1,42 %) |
| Nivel reglas SonarQube | Nivel 1 | |
| Quality gates | No se han establecido | |

* 1. Datos de calidad

A continuación, se muestran las métricas de calidad más relevantes.

|  |  |
| --- | --- |
| Vulnerabilities | 145 |
| Hotspots | 103 |
| Bugs | 1.568 |
| Code smells | 7.126 |
| Duplicidad | 13,1 % |
| Bloques duplicados | 687 |

1. Nivel y modelo de calidad

dvwa se encuentra actualmente en el **Nivel 1 del modelo de calidad** acordado entre los equipos de desarrollo y calidad. Dentro de este nivel, se muestran a continuación la relación de reglas cumplidas e incumplidas (reglas que generan al menos un problema):

Además, se muestra a continuación los problemas encontrados categorizados por tipo y lenguaje.

* 1. Listado de reglas incumplidas

Las reglas incumplidas y el número de problemas que ocasiona cada una, separadas por tipo, son las siguientes:

**Security Hotspots**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| sql-injection | Make sure that formatting this SQL query is safe here. | 2 |
| xss | Set the "session.cookie\_httponly" property to "true" if needed. | 5 |
| dos | Make sure that using a regular expression is safe here. | 39 |
| rce | Make sure that this dynamic injection or execution of code is safe. | 3 |
| weak-cryptography | Make sure that hashing data is safe here. | 33 |
| insecure-conf | Make sure creating this cookie without the "secure" flag is safe here. | 14 |
| log-injection | Make sure that this logger's configuration is safe. | 1 |
| object-injection | Make sure deserializing objects is safe here. | 6 |
| Total |  | 103 |

**Vulnerability**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| HTML | Links with "target=\_blank" should prevent phishing attacks | 42 |
| Total |  | 42 |

**Bug**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| CSS | Font declarations should contain at least one generic font family | 2 |
| CSS | Properties should not be duplicated | 1 |
| CSS | Shorthand properties that override related longhand properties should be avoided | 1 |
| JavaScript | "with" statements should not be used | 2 |
| PHP | All branches in a conditional structure should not have exactly the same implementation | 28 |
| PHP | Objects should not be created to be dropped immediately without being used | 3 |
| PHP | Variables should be initialized before use | 2 |
| PHP | All code should be reachable | 2 |
| PHP | Related "if/else if" statements and "cases" in a "switch" should not have the same condition | 1 |
| HTML | "strong" and "em" tags should be used | 650 |
| HTML | "table" tags should have a description | 392 |
| HTML | Tables should have headers | 371 |
| HTML | "html" element should have a language attribute | 47 |
| HTML | "th" tags should have "id" or "scope" attributes | 42 |
| HTML | Elements deprecated in HTML5 should not be used | 20 |
| HTML | "frames" should have a "title" attribute | 3 |
| HTML | "!DOCTYPE" declarations should appear before "html" tags | 1 |
| Total |  | 1.568 |

**Code Smell**

|  |  |  |
| --- | --- | --- |
| Lenguaje | Regla | Problemas |
| CSS | Empty blocks should be removed | 21 |
| JavaScript | Variables should not be shadowed | 3 |
| JavaScript | Variables should be declared explicitly | 2 |
| JavaScript | Multiline blocks should be enclosed in curly braces | 1 |
| PHP | Control structures should use curly braces | 648 |
| PHP | String literals should not be duplicated | 301 |
| PHP | Class names should comply with a naming convention | 233 |
| PHP | Cognitive Complexity of functions should not be too high | 74 |
| PHP | Boolean literals should not be redundant | 60 |
| PHP | Ternary operators should not be nested | 52 |
| PHP | Functions should not contain too many return statements | 52 |
| PHP | Track uses of "TODO" tags | 51 |
| PHP | Sections of code should not be commented out | 39 |
| PHP | Local variables should not be declared and then immediately returned or thrown | 39 |
| PHP | Local variables should not have the same name as class fields | 39 |
| PHP | Source files should not have any duplicated blocks | 32 |
| PHP | Unused function parameters should be removed | 29 |
| PHP | Collapsible "if" statements should be merged | 24 |
| PHP | Generic exceptions ErrorException, RuntimeException and Exception should not be thrown | 24 |
| PHP | "for" loop stop conditions should be invariant | 23 |
| PHP | Unused "private" methods should be removed | 23 |
| PHP | Unused local variables should be removed | 18 |
| PHP | "switch" statements should have "default" clauses | 13 |
| PHP | "" and "||" should be used | 10 |
| PHP | Functions should use "return" consistently | 7 |
| PHP | Nested blocks of code should not be left empty | 6 |
| PHP | Functions should not have too many lines of code | 5 |
| PHP | Jump statements should not be redundant | 5 |
| PHP | "switch" statements should have at least 3 "case" clauses | 5 |
| PHP | Classes should not have too many methods | 4 |
| PHP | Unused assignments should be removed | 4 |
| PHP | Two branches in a conditional structure should not have exactly the same implementation | 4 |
| PHP | "switch" statements should not have too many "case" clauses | 2 |
| PHP | Redundant pairs of parentheses should be removed | 2 |
| PHP | Variable variables should not be used | 2 |
| PHP | Constant names should comply with a naming convention | 2 |
| PHP | Interface names should comply with a naming convention | 2 |
| PHP | Methods should not have identical implementations | 1 |
| PHP | A close curly brace should be located at the beginning of a line | 1 |
| PHP | Overriding methods should do more than simply call the same method in the super class | 1 |
| PHP | Unused "private" fields should be removed | 1 |
| PHP | Track uses of "FIXME" tags | 1 |
| HTML | Attributes deprecated in HTML5 should not be used | 5.259 |
| HTML | Sections of code should not be commented out | 1 |
| Total |  | 7.126 |

1. Tendencias e histórico

No aplica debido a que es el primer análisis realizado de dvwa y no existen datos suficientes para realizar la comparativa. En esta sección se realizará un análisis de la evolución de las métricas más destacadas y una valoración de la calidad a nivel histórico, hitos planteados e hitos conseguidos.

1. Análisis de la aplicación

Después de someter la aplicación dvwa a un análisis más detallado basado en los resultados obtenidos por SonarQube y teniendo en cuenta que este informe, por ser el primero, tiene como objetivo presentar una primera radiografía de la aplicación y establecer un punto de partida sobre el que comenzar a trabajar en la mejora de la calidad del código fuente, se exponen las siguientes conclusiones.

* 1. Vulnerabilidades
  2. Bugs
  3. Code Smells
  4. Duplicidad

1. Siguientes pasos

Se recomiendan las siguientes tareas:

* Centrar las actividades del equipo de desarrollo en los problemas de tipo vulnerability.
* Analizar y confirmar los falsos positivos sugeridos por el equipo de CALIDAD en cuanto a bugs se refiere.
* Comprobar y confirmar que las contraseñas encontradas por el analizador no suponen un riesgo para la integridad de los sistemas.

Así mismo el Equipo de Calidad propone:

* Mantener a la aplicación dvwa en el Nivel 1 del modelo de calidad hasta que se realicen nuevos informes detallados y se analice la evolución derivada de los mismos.
* Mantener los análisis planificados y desatendidos en SonarQube de forma semanal. Tanto el Equipo de Calidad como el Equipo de Desarrollo podrán acceder a los datos actualizados por cada análisis a través de [SonarQube](http://srv-analiza/sonar).
* Realizar el siguiente análisis detallado y generación de informes durante el mes de \_ o en el momento en que se libere una nueva versión si es anterior a esa fecha.
* La instalación del complemento SonarLint en los equipos de los desarrolladores de la aplicación dvwa. De esta manera se puedan realizar análisis del código fuente en fases tempranas del desarrollo por parte de los integrantes del equipo.

Sobre ESIIAB

Para más información acceda a: [Escuela Superior de Ingeniería Informática (uclm.es)](https://www.esiiab.uclm.es/)

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |