Auditoria Tokens

"Goal Stadium"

Miguel Ángel Mora Tudela
CIBINAR – CIBERSEGURIDAD E INTELIGENCIA ARITIFICAL

CONTRATO INTELIGENTE DE CRIPTOMONEDAS

INFORME DE ANÁLISIS DE CÓDIGO Y ANÁLISIS DE VULNERABILIDADES

Cliente: GoalStadium
Realizado en fecha: 11 de noviembre de 2021
Plataforma: GoalStadium and Smart Contract
Lenguaje: Solidity

ÍNDICE DE CONTENIDOS

1. Integridad de los contratos

cc662d46eddc44b9a9be8bd874745ab3	GoaL.sol	MD5
5451d69f18990a55c3b6e6b92390a8fadd6c2308	GoaL.sol	SHA1

Se incluye la huella MD5 y la huella SHA1 del fichero analizado para validar la integridad del mismo. Así mismo, si hubiera algún cambio en éste, las huellas se modificarían y el resultado de esta auditoría quedaría invalidado.

2. Resumen de auditoría y características

La auditoría de contrato inteligente es un proceso de estudio cuidadoso del código, es decir, se lleva a cabo el descubrimiento de errores, vulnerabilidades y riesgos antes de implementarlo y usarlo. De esta manera, el código no puede volver a modificarse una vez se ha publicado.

Para ello, se han utilizado las siguientes herramientas: Myth, Slither y RemixIDE. Más adelante, se presentarán y se comentarán las principales características.

La auditoría del proyecto GoalStadium es un proceso de estudio de la calidad del código mediante el análisis estático de éste. Así, podemos obtener una serie de puntos a mejorar basados en métricas establecidas en Sonarqube.

Más adelante, se presentará y se comentará las principales características.

3. Presentación de las herramientas utilizadas

3.1 Presentación de las herramientas usadas en el contrato inteligente

Para verificar la seguridad del contrato usaremos Myth, Slither y RemixIDE. Los cuáles son frameworks con dependencias con el compilador de Solidity y tienen como finalidad proveer una serie de métricas para detectar diferentes tipos de vulnerabilidades. Las principales funcionalidades de estas tres herramientas son:

Myth	**Slither**
- Detecta vulnerabilidades de seguridad en contratos para Ethereum, Hedera, Quorum, Vechain, Roostock y Tron - Utiliza ejecución simbólica, resolución SMT y análisis de contaminación - Se utiliza en combinación con otras herramientas y técnicas en MythX	- Detecta los falsos positivos - Localiza de manera precisa los errores del código - Cubre el 99,9 % del código de Solidity - La media de ejecución es rápida

RemixIDE
- Proporciona una manera fácil de leer el AST de contrato inteligente escrito en Solidity - Cuenta con un conjunto de bibliotecas que pueden ser usadas en distintos módulos con el objetivo de llevar a cabo el análisis - Proporciona la posibilidad de importar un contrato inteligente desde distintoa repositortios

3.2 Presentación de la herramienta usada en el proyecto

Sonarqube permite llevar a cabo la evaluación de la calidad del código fuente del proyecto, es decir, muestra una serie de puntos a mejorar en el código fuente a través de un conjunto de métricas establecidas.

Por tanto, en el dashboard de Sonarqube nos encontramos con los apartados que describen los bugs, vulnerabilidades, duplicidad de código, etc. En cada caso se especifican las medidas a tomar para intentar evitar que estos errores puedan ocasionar una vulnerabilidad en el proyecto.

4. Vulnerabilidades encontradas en el contrato inteligente

El alcance del contrato inteligente es el análisis de la calidad del código, que permite descubrir posibles errores, vulnerabilidades y riesgos. Este alcance está definido en las siguientes categorías: Security, Gas & Economy, ERC y Miscellaneous.

RemixIDE

Security

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp to a certain degree to change the outcome of a transaction in the mined block.

Los mineros de blockchain pueden utilizar el "block.timstamp" para influenciarlo hasta cierto punto.

Esto quiere decir que los mineros pueden "elegir" que el "block.timstamp" cambie el resultado (hasta cierto punto) de una transacción del bloque minado.

```
    modifier not_locked {
        if(msg.sender == _addresses.presaleAddress) {
            require( block.timestamp > _locks.presaleLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.sponsorAddress) {
            require( block.timestamp > _locks.sponsorLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.teamAddress) {
            require( block.timestamp > _locks.teamLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.stakingAddress) {
            require( block.timestamp > _locks.stakingLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.reservesAddress) {
            require( block.timestamp > _locks.reservesLock, "GoaL : The tokens still
locked");
        }
```

```
        if(msg.sender == _addresses.marketingAddress) {
            require( block.timestamp > _locks.marketingLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.farmingAddress) {
            require( block.timestamp > _locks.farmingLock, "GoaL : The tokens still
locked");
        }
        if(msg.sender == _addresses.playToEarnAddress) {
            require( block.timestamp > _locks.playToEarnLock, "GoaL : The tokens still
locked");
        }
        _;
    }
```

Gas costs:

Gas requirement of function GoaL name is infinite: If the gas requirement of a function is higher than the block gas limit it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage).

El gas necesario de la función GoaL es infinito: Si el gas necesario de una función es mayor al límite del bloque de gas, no podrá ejecutarse.

Evite el uso de bucles en las funciones o acciones que modifiquen las áreas de almacenamiento (storage), incluyendo la limpieza o la copia de matrices en el almacenamiento.

```
// Contract Info Functions
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    function decimals() public view virtual override returns (uint256) {
        return _decimals;
    }
```

```
    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view virtual override returns
(uint256) {
        return _balances[account];
    }
```

ERC:

ERC20 contract's "decimals" function should have "uint8" as return type.

La función "decimales" del ERC20 debería tener asignado "uint8" como tipo de retorno.

```
// Contract Info Functions
    function name() public view virtual override returns (string memory) {
        return _name;
    }

    function symbol() public view virtual override returns (string memory) {
        return _symbol;
    }

    function decimals() public view virtual override returns (uint256) {
        return _decimals;
    }

    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view virtual override returns
(uint256) {
        return _balances[account];
    }

    // Transfer Function
    function transfer(address recipient, uint256 amount) public virtual override
whitelisted_both(msg.sender, recipient) not_locked returns (bool) {
```

```
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address _owner, address spender) public view virtual override
returns (uint256) {
        return _allowances[_owner][spender];
    }
```

Miscellaneous - Constant/View/Pure functions:

IERC20.transfer(address.uint256): Potentially should be constant/view/pure but is not.

Note: Modifiers are currently not considered by this static analysis.

IERC20.transfer(address.uint256): debería ser constante/visualización/pura, pero no lo es.

Nota: el presente análisis estático no ha considerado los modificadores.

```
pragma solidity ^0.8.0;

/* GoaL Token BEP-20
/* V.1.0
/* Last update 10/11/21*/

interface IERC20{

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns
(uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}
```

Similar variable names:

GoaL.(string,string,uint256,uint256,structGoaL.Addresses,struct GoaL.Locks,struct Goal.Supplies): Variables have very similar names "_decimals" and "decimals_".

Note: Modifiers are currently not considered by this static analysis.

Nombres de variables similares:
GoaL.(string,string,uint256,uint256,structGoaL.Addresses,struct GoaL.Locks,struct Goal.Supplies): las variables tienen nombres muy similares "_decimals" and "decimals_".

Nota: el presente análisis estático no ha tenido considerado los modificadores.

```
    // Constructor
    constructor (string memory name_, string memory symbol_,
      uint256 initialBalance_,uint256 decimals_,
      Addresses memory addresses_, Locks memory locks_, Supplies memory
supplies_) {

       // Token values asignation
        _name = name_;
        _symbol = symbol_;
        _totalSupply = initialBalance_* 10**decimals_;
        _decimals = decimals_;

        // Supply values asignation
        _supplies = Supplies(
                    supplies_.presaleSupply*10**decimals_,
                    supplies_.sponsorSupply*10**decimals_,
                    supplies_.teamSupply*10**decimals_,
                    supplies_.stakingSupply*10**decimals_,
                    supplies_.reservesSupply*10**decimals_,
                    supplies_.marketingSupply*10**decimals_,
                    supplies_.farmingSupply*10**decimals_,
                    supplies_.playToEarnSupply*10**decimals_
                    );


        // Wallets asignation
        _addresses = Addresses(
                    addresses_.presaleAddress,
                    addresses_.sponsorAddress,
                    addresses_.teamAddress,
                    addresses_.stakingAddress,
```

```
            addresses_.reservesAddress,
            addresses_.marketingAddress,
            addresses_.farmingAddress,
            addresses_.playToEarnAddress,
            msg.sender,
            addresses_.betsAddress
        );
```

No return:

IERC20.balanceOf(address): Defines a return type but never explicitly returns a value.

Sin retorno:

IERC20.balanceOf(address): define un tipo de retorno, pero nunca devuelve un valor de forma explícita.

```
pragma solidity ^0.8.0;

/* GoaL Token BEP-20
/* V.1.0
/* Last update 10/11/21*/

interface IERC20{

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

Condiciones de incógnitas:

Utiliza "assert(x)" si quieres que X nunca (jamás) sea falsa, bajo ninguna circunstancia (a excepción de que aparezca un bug en el código). Utiliza "require(x)" si X puede ser falsa, por razones de input incorrecto o no válido o fallos de componentes externos, entre otras.

Locks private _locks;

```
struct Supplies {
    uint256 presaleSupply;
    uint256 sponsorSupply;
    uint256 teamSupply;
    uint256 stakingSupply;
    uint256 reservesSupply;
    uint256 marketingSupply;
    uint256 farmingSupply;
    uint256 playToEarnSupply;
}

Supplies private _supplies;


// Modifiers

modifier whitelisted_both(address _from, address _to) {
   require( _whitelist[_from] == _from, "GoaL : You are not whitelisted" );
   require( _whitelist[_to] == _to, "GoaL : The receptor is not whitelisted" );
   _;
}

modifier onlyOwner {
    require(msg.sender == _addresses.ownerAddress);
    _;
}
```

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

La división de valores enteros vuelve a dar como resultado otro valor entero. Por ejemplo, 10 / 100 = 0 en vez de 0,1, ya que el resultado aparecerá como número entero (sin decimales). Este ejemplo no se aplica a la división de (únicamente) valores literales, ya que estos producen constantes racionales.

```
    // Aux Functions
    function _transfer(address sender, address recipient, uint256 amount) internal
virtual {
        require(sender != address(0), "GoaL: transfer from the zero address");
        require(recipient != address(0), "GoaL: transfer to the zero address");


        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "BEP20: transfer amount exceeds balance");
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += amount;

        emit Transfer(sender, _addresses.reservesAddress, amount / 100 * 2);
        emit Transfer(sender, _addresses.betsAddress, amount / 100 * 3);
        emit Transfer(sender, recipient, amount / 100 * 95);
    }

    function _approve(address _owner, address spender, uint256 amount) internal
virtual {
        require(_owner != address(0), "GoaL: approve from the zero address");
        require(spender != address(0), "GoaL: approve to the zero address");

        _allowances[_owner][spender] = amount;
        emit Approval(_owner, spender, amount);
    }
```

Slither

A continuación, se adjunta parte del código del contrato inteligente y se describen los fallos destacables.

En primer lugar, aparece una advertencia en la función "Context. _msgData()" en la cual se señala que esta nunca llega a usarse y debe ser eliminada.

En segundo lugar, aparecen alertas referidas a "Pragma version" en las cuáles se indica que se necesita una versión anterior para ser confiable y recomienda instalar la versión 0.6.12/0.7.6.

En tercer lugar, se indica que el nombre de las funciones debe declararse externas y no locales.

--> GoaLF.sol

Context._msgData() (GoaLF.sol#36-39) is never used and should be removed

Pragma version^0.8.0 (GoaLF.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (GoaLF.sol#20) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (GoaLF.sol#29) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (GoaLF.sol#43) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.7 is not recommended for deployment

Parameter GoaL.allowance(address,address)._owner (GoaLF.sol#255) is not in mixedCase
Modifier GoaL.whitelisted_both(address,address) (GoaLF.sol#106-110) is not in mixedCase
Modifier GoaL.not_locked() (GoaLF.sol#117-143) is not in mixedCase

Redundant expression "this (GoaLF.sol#37)" inContext (GoaLF.sol#31-40)
Reference:

GoaL.addToBlackList(address) (GoaLF.sol#318-320) uses literals with too many digits:
- _whitelist[address_] = 0x0000000000000000000000000000000000000000 (GoaLF.sol#319)

name() should be declared external:
- GoaL.name() (GoaLF.sol#229-231)
symbol() should be declared external:

- GoaL.symbol() (GoaLF.sol#233-235)
decimals() should be declared external:
- GoaL.decimals() (GoaLF.sol#237-239)
totalSupply() should be declared external:
- GoaL.totalSupply() (GoaLF.sol#241-243)
balanceOf(address) should be declared external:
- GoaL.balanceOf(address) (GoaLF.sol#245-247)
transfer(address,uint256) should be declared external:
- GoaL.transfer(address,uint256) (GoaLF.sol#250-253)
allowance(address,address) should be declared external:
- GoaL.allowance(address,address) (GoaLF.sol#255-257)
approve(address,uint256) should be declared external:
- GoaL.approve(address,uint256) (GoaLF.sol#259-262)
transferFrom(address,address,uint256) should be declared external:
- GoaL.transferFrom(address,address,uint256) (GoaLF.sol#264-272)
increaseAllowance(address,uint256) should be declared external:
- GoaL.increaseAllowance(address,uint256) (GoaLF.sol#274-277)
decreaseAllowance(address,uint256) should be declared external:
- GoaL.decreaseAllowance(address,uint256) (GoaLF.sol#280-286)
addToWhiteList(address) should be declared external:
- GoaL.addToWhiteList(address) (GoaLF.sol#314-316)
addToBlackList(address) should be declared external:
- GoaL.addToBlackList(address) (GoaLF.sol#318-320)

GoaLF.sol analyzed (4 contracts with 75 detectors), 28 result(s) found

Myth

Por un lado, mediante esta herramienta se ha ejecutado un comando que analiza vulnerabilidades en el código del contrato inteligente. Los resultados arrojan que no se han encontrado fallas en el código.

```
# ls
 GoaL.sol            systemd-private-c8732d3a79414b58a63fad2de56c8808-
ModemManager.service-S4qs8i
 hsperfdata_root             systemd-private-c8732d3a79414b58a63fad2de56c8808-
systemd-logind.service-bBBiei
 mozilla_parrot0             systemd-private-c8732d3a79414b58a63fad2de56c8808-
upower.service-y9o68h
 nvimXbt6iH          Temp-8b06e3dc-50e7-4ca9-b65a-b871cb92c132
 qtsingleapp-zoom-3e8   Temp-fc639a95-e345-4ec0-ba8d-eb4b4799aa1c
```

 # myth analyze GoaL.sol
The analysis was completed successfully. No issues were detected.

Por otro lado, con otro comando se ha analizado el estado de seguridad de las funciones del contrato inteligente y los resultados muestran que las funciones se consideran seguras.

#myth safe-functions GoaL.sol Contract GoaL:

13 functions are deemed safe in this contract: name(), addToBlackList(address), decimals(), symbol(), transfer(address,uint256), totalSupply(), transferFrom(address,address,uint256), addToWhiteList(address), approve(address,uint256), increaseAllowance(address,uint256), allowance(address,address), decreaseAllowance(address,uint256), balanceOf(address)ç

5. Vulnerabilidades encontradas en el proyecto

5.1 Vulnerabilidades serias encontradas en el proyecto

Categoría: Malas prácticas

Las variables deben declararse explícitamente
La variable *scope* de JavaScript puede ser particularmente difícil de entender y usar. La situación empeora cuando se considera la creación accidental de variables globales, que es lo que sucede cuando se declara una variable dentro de una función o la cláusula *for* de un bucle *for* (for-loop) sin usar las palabras clave let, const o var.

let y *const* se introdujeron en ECMAScript 2015 y ahora son las palabras clave preferidas para la declaración de variables.

Ejemplo sin declarar variables

```
function f(){
  i = 1;        // Noncompliant; i is global

  for (j = 0; j < array.length; j++) {  // Noncompliant; j is global now too
    // ...
  }

}
```

Ejemplo con variables declaradas explícitamente

```
function f(){
  var i = 1;

  for (let j = 0; j < array.length; j++) {
    // ...
  }

}
```

Corregir
GoalStadium/public/assets/js/metamaskconnector.js
GoalStadium/public/assets/js/metamaskconnector_backup.js

Las salidas de las funciones no deben ser invariantes

Cuando una función está diseñada para devolver un valor invariante, puede ser un diseño deficiente, pero no debería afectar negativamente el resultado de su programa. Sin embargo, cuando sucede en todos los caminos a través de la lógica, es probable que sea un error.

Hay una función con varias declaraciones return que devuelve el mismo valor.

Corregir
GoalStadium/public/js/app.js

Switch case sin "break" incondicional

Los switch cases deben terminar con una declaración de "break" incondicional. Cuando la ejecución no finaliza explícitamente al final de un switch case, continúa ejecutando las declaraciones del case siguiente. Si bien esto a veces es intencional, a menudo es un error que conduce a un comportamiento inesperado.

Ejemplo
Código con switch case sin break incondicional

```
switch (myVariable) {
  case 1:
    foo();
    break;
  case 2:  // 'hazAlgo()' y 'hazAlgoMas()' seran ejecutados
    hazAlgo();
```

```
  default:
    hazAlgoMas();
    break;

}
```

Código con switch case y break incondicional

```
switch (myVariable) {
  case 1:
    foo();
    break;
  case 2:
    doSomething();
    break;
  default:
    doSomethingElse();
    break;

}
```

Corregir
GoalStadium/public/js/app.js

Cadenas literales duplicadas

Las cadenas literales (string literals) duplicadas hacen que el proceso de refactorización sea propenso a errores, ya que hay que asegurarse de actualizar todas las ocurrencias, a diferencia de las constantes que pueden ser referenciadas desde muchos lugares, pero sólo necesitan ser actualizadas en un único lugar.

Ejemplo:
Código con cadenas literales duplicadas

```
function run() {
  prepare('esto es un duplicado');
  execute('esto es un duplicado');
  release('esto es un duplicado');

}
```

Código corregido

```
MESSAGE = 'esto es un duplicado';

function run() {
  prepare(MESSAGE);
  execute(MESSAGE);
  release(MESSAGE);

}
```

Corregir
GoalStadium/config/logging.php
GoalStadium/resources/lang/en/validation.php
GoalStadium/tests/Feature/Auth/AuthenticationTest.php
GoalStadium/tests/Feature/Auth/PasswordConfirmationTest.php
GoalStadium/tests/Feature/Auth/PasswordResetTest.php

La complejidad cognitiva de las funciones no debe ser demasiado alta

La Complejidad Cognitiva es una medida de cuán difícil es comprender el flujo de control de una función. Las funciones con alta complejidad cognitiva serán difíciles de mantener.

Corregir
GoalStadium/public/js/app.js

Las funciones no deben estar vacías

Hay varias razones para que una función puede estar vacía:

- Es una omisión involuntaria y debe corregirse para evitar un comportamiento inesperado en la producción.
- Aún no se ha admitido, o nunca lo será. En este caso, se debe lanzar una excepción en los idiomas donde ese mecanismo está disponible.
- El método es una anulación intencionalmente en blanco. En este caso, un comentario anidado debería explicar el motivo de la anulación en blanco.

Corregir
GoalStadium/public/js/app.js

Los contadores de bucle no deben asignarse desde dentro del cuerpo del bucle

Los contadores de bucle no deben modificarse en el cuerpo del bucle. Sin embargo, otras variables de control de bucle que representan valores lógicos pueden modificarse en el bucle, por ejemplo, una bandera para indicar que algo se ha completado, que luego se prueba en la declaración for.

Corregir
GoalStadium/public/js/app.js

Categoría: Bugs

Las propiedades CSS deben ser válidas

Las especificaciones del W3C definen las propiedades CSS válidas. Solo se deben usar las propiedades oficiales y específicas del navegador para obtener el impacto esperado en el renderizado final.

Se ignoran:
Sintaxis de las variables $sass, @less, y var(--custom-property).
propiedades con prefijo de proveedor (por ejemplo, -moz-align-self, -webkit-align-self).

Corregir
Laboratory/.../src/__tests__/fixtures/counter-style.css
Laboratory/.../src/__tests__/fixtures/counter-style.post.css

Deben evitarse las propiedades de taquigrafía shorthand que anulan las propiedades de taquigrafía longhand relacionadas

Una propiedad de taquigrafía shorthand definida después de una propiedad de taquigrafía longhand anulará por completo el valor definido en la propiedad de taquigrafía longhand, haciendo que la de taquigrafía longhand sea inútil. El código debe refactorizarse para considerar taquigrafía longhand o eliminarla por completo.

Ejemplo
Incorrecto

```
a {
  padding-left: 10px;
  padding: 20px; /* Noncompliant; padding is overriding padding-left making it useless */

}
```

Correcto

```
a {
  padding: 10px; /* Compliant; padding is defining a general behaviour and padding-left, just after, is precising the left case */
  padding-left: 20px;

}
```

Corregir
GoalStadium/.../loadjs/test/vendor/mocha-4.1.0/mocha.css

5.2 Vulnerabilidades medias encontradas en el proyecto

Categoría: Seguridad

Expresión regular vulnerable al tiempo de ejecución superlineal debido al retroceso

Se debe asegurar que la expresión regular utilizada aquí, que es vulnerable al tiempo de ejecución superlineal debido al retroceso, no pueda conducir a la denegación de servicio. Puede suponer una Denegación de servicio (DoS).

La mayoría de los motores de expresión regular utilizan el retroceso para probar todas las posibles rutas de ejecución de la expresión regular al evaluar una entrada; en algunos casos, puede causar problemas de rendimiento, denominados situaciones catastróficas de retroceso. En el peor de los casos, la complejidad de la expresión regular es exponencial en el tamaño de la entrada, esto significa que una pequeña entrada cuidadosamente diseñada (como 20 caracteres) puede desencadenar un retroceso catastrófico y provocar una denegación de servicio de la aplicación. La complejidad de las expresiones regulares superlineales también puede provocar el mismo impacto con, en este caso, una gran entrada cuidadosamente diseñada (miles de caracteres).

Esta regla determina la complejidad del tiempo de ejecución de una expresión regular y le informa si no es lineal.

Es un riesgo si:

- La entrada está controlada por el usuario.
- El tamaño de entrada no está restringido a una pequeña cantidad de caracteres.
- No hay tiempo de espera para limitar el tiempo de evaluación de expresiones regulares.

Para evitar situaciones problemáticas de retroceso, hay que asegurarse que ninguna de las siguientes condiciones se aplique a su expresión regular.

En todos los casos siguientes, el retroceso solo puede suceder si la parte problemática de la expresión regular es seguida por un patrón que puede fallar, lo que hace que el retroceso realmente suceda.

- Si hay una repetición r* o r*?, tal que la expresión regular r podría producir diferentes coincidencias posibles (de longitudes posiblemente diferentes) en la misma entrada, el peor tiempo de coincidencia puede ser exponencial. Este puede ser el caso si r contiene partes opcionales, alternancias o repeticiones adicionales (pero no si la repetición está escrita de tal manera que solo hay una forma de igualarla).

- Si existen varias repeticiones que pueden coincidir con el mismo contenido y son consecutivas o solo están separadas por un separador opcional o un separador que puede coincidir con ambas repeticiones, el peor tiempo de coincidencia puede ser polinómico (O(n^c) donde c es el número de repeticiones problemáticas). Por ejemplo, a*b* no es un problema porque a * y b * coinciden con cosas diferentes y a * _a * no es un problema porque las repeticiones están separadas por un '_' y no pueden coincidir con ese '_'. Sin embargo, a*a* y.*_. * tienen tiempo de ejecución cuadrático.

- Si la expresión regular no está anclada al principio de la cadena, el tiempo de ejecución cuadrático es especialmente difícil de evitar porque cada vez que falla una coincidencia, el motor de expresiones regulares volverá a intentarlo comenzando en el siguiente índice. Esto significa que cualquier repetición ilimitada, si va seguida de un patrón que puede fallar, puede provocar un tiempo de ejecución cuadrático en algunas entradas. Por ejemplo, str.split(/\s*,/) se ejecutará en tiempo cuadrático en cadenas que constan completamente de espacios (o al menos contienen grandes secuencias de espacios, no seguidas de una coma).

Para reescribir la expresión regular sin estos patrones, hay que considerar las siguientes estrategias:

- Si corresponde, se debe definir un número máximo de repeticiones esperadas utilizando los cuantificadores acotados, como {1,5} en lugar de +, por ejemplo.

- Se debe refactorizar los cuantificadores anidados para limitar el número de formas en que el cuantificador externo puede hacer coincidir el grupo interno; por ejemplo, esta situación del cuantificador anidado (ba+)+ no causa problemas de rendimiento; de hecho, el grupo interno solo se puede emparejar si existe exactamente un b char por repetición del grupo.

- Se debe optimizar las expresiones regulares emulando cuantificadores posesivos y agrupaciones atómicas.

- Se recomienda utilizar clases de caracteres negadas en lugar de. para excluir separadores cuando corresponda. Por ejemplo, la expresión regular cuadrática. *_. * puede hacerse lineal cambiándola a [^_]* _.*

A veces no es posible reescribir la expresión regular para que sea lineal y al mismo tiempo coincida con lo que desea que coincida. Especialmente cuando la expresión regular no está anclada al principio de la cadena, por lo que es bastante difícil evitar tiempos de ejecución cuadráticos. En esos casos, hay que considerar los siguientes enfoques:

- Se debe resolver el problema sin expresiones regulares.

- Se recomienda usar implementaciones alternativas de expresiones regulares sin retroceso, como Google's RE2 o node-re2.

- Se debe utilizar varias pasadas. Esto podría significar preprocesar y/o posprocesar la cadena manualmente antes/después de aplicarle la expresión regular o usar varias expresiones regulares. Un ejemplo de esto sería reemplazar str.split(/\s*,\s*/) con str.split (",") y luego recortar los espacios de las cadenas como segundo paso.

- A menudo es posible hacer que la expresión regular sea infalible haciendo que todas las partes que podrían fallar sean opcionales, lo que evitará el retroceso. Por supuesto, esto significa que aceptará más cadenas de las previstas, pero esto se puede manejar mediante el uso de grupos de captura para verificar si las partes opcionales coincidieron o no y luego ignorar la coincidencia si no lo fueron. Por ejemplo, la expresión regular x*y podría reemplazarse con x*(y)? y luego la llamada a str.match(regex) podría reemplazarse con matched = str.match(regex) y matched[1] !== undefined.

Corregir
GoalStadium/public/js/app.js

Inyección dinámica o ejecución de código

Hay que asegurarse que esta inyección dinámica o ejecución de código sea segura.
La ejecución de código dinámicamente es sensible a la seguridad. Ha dado lugar en el pasado a las siguientes vulnerabilidades:

- CVE-2017-9807
- CVE-2017-9802

Algunas API permiten la ejecución de código dinámico proporcionándolo como cadenas en tiempo de ejecución. Estas API pueden resultar útiles en algunos casos de uso de meta programación muy específicos. Sin embargo, la mayoría de las veces su uso está mal visto, ya que también aumentan el riesgo de código inyectado. Dichos ataques pueden ejecutarse en el servidor o en el cliente (ejemplo: ataque XSS) y tener un gran impacto en la seguridad de una aplicación.

Esta regla plantea problemas en las llamadas a los constructores eval y Function. Esta regla no detecta inyecciones de código. Solo destaca el uso de API que debe usarse con moderación y con mucho cuidado. El objetivo es orientar las revisiones del código de seguridad.

Esta regla no planteará ningún problema cuando el argumento de eval o Function es una cadena literal, ya que es razonablemente seguro.

Hay un riesgo si:

- el código ejecutado puede provenir de una fuente que no es de confianza y no se ha desinfectado.
- realmente necesitas ejecutar código dinámicamente.

Con respecto a la ejecución de código desconocido, la mejor solución es no ejecutar código proporcionado por una fuente que no sea de confianza. Si se requiere se puede ejecutar el código en un entorno de espacio aislado. Hay que utilizar jails, firewalls y cualquier medio que le proporcione su sistema operativo y lenguaje de programación (ejemplo: Security Managers en java, iframes y política del mismo origen para javascript en un navegador web).
Se debe evitar crear una lista negra de código peligroso. Es imposible cubrir todos los ataques de esa manera.

Evítese el uso de API de código dinámico siempre que sea posible. El código codificado siempre es más seguro.

Corregir
GoalStadium/public/js/app.js

Criptografía débil

No se deben emplear algoritmos hash débiles para contener información sensible.

Los algoritmos hash criptográficos como MD2, MD4, MD5, MD6, HAVAL-128, HMAC-MD5, DSA (que usa SHA-1), RIPEMD, RIPEMD-128, RIPEMD-160, HMACRIPEMD160 y SHA-1 ya no se consideran seguros. Estos pueden ser vulnerados (con poco esfuerzo computacional es suficiente para encontrar dos o más entradas diferentes que produzcan el mismo hash).

Se recomiendan alternativas más seguras, como SHA-256, SHA-512, SHA-3, y para el hash de contraseñas, es incluso mejor usar algoritmos que no calculan demasiado "rápido", como bcrypt, scrypt, argon2 o pbkdf2 porque ralentiza hacia abajo los ataques de fuerza bruta.

Corregir
GoalStadium/tests/Feature/Auth/EmailVerificationTest.php

Fragmento de código mejorable

```
Event::fake();
$verificationUrl = URL::temporarySignedRoute (
    'verification.verify',
    now()->addMinutes(60),
    ['id' => $user->id, 'hash' => sha1($user->email)]
);
$response = $this->actingAs($user)->get($verificationUrl);
Event::assertDispatched(Verified::class);
```

```
]);
$verificationUrl = URL::temporarySignedRoute (
    'verification.verify',
    now()->addMinutes(60),
    ['id' => $user->id, 'hash' => sha1('wrong-email')]
);
$this->actingAs($user)->get($verificationUrl);
$this->assertFalse($user->fresh()->hasVerifiedEmail());
```

Categoría: Malas prácticas

Secciones de código comentadas

Comentar el código abulta los programas y reduce la legibilidad. El código no utilizado debe eliminarse. Si luego se quiere recuperar puede hacerse a través de un historial de control.

Corregir
GoalStadium/app/Console/Kernel.php
GoalStadium/app/Providers/RouteServiceProvider.php
GoalStadium/phpunit.xml
GoalStadium/public/vendor/url-polyfill/tests/test-url-class.html
GoalStadium/resources/views/stake/partials/table.blade.php

Parámetros sin usar

Los parámetros no utilizados pueden ser confusos y no afectan al comportamiento de la aplicación.Si realmente el parámetro no aporta nada es mejor quitarlo.

Corregir
GoalStadium/app/Exceptions/Handler.php
GoalStadium/database/factories/UserFactory.php

Los bloques de código anidados no deben dejarse vacíos

La mayoría de las veces, un bloque de código está vacío cuando realmente falta un fragmento de código. Por lo tanto, dicho bloque vacío debe llenarse o eliminarse.

Corregir
GoalStadium/public/js/app.js

Dos ramas en una estructura condicional no deberían tener exactamente la misma implementación

Tener dos cases en una declaración switch o dos ramas en una cadena if con la misma implementación es, en el mejor de los casos, un código duplicado y, en el peor, un error de codificación. Si realmente se necesita la misma lógica para ambas instancias, entonces en una cadena if deberían combinarse, o para un cambio, una debería pasar a la otra.

Corregir
GoalStadium/public/js/app.js

Las funciones siempre deben devolver el mismo tipo

A diferencia de los lenguajes fuertemente tipados, JavaScript no impone un tipo de retorno en una función. Esto significa que diferentes rutas a través de una función pueden devolver diferentes tipos de valores, lo que puede resultar muy confuso para el usuario y significativamente más difícil de mantener.

Corregir
GoalStadium/public/js/app.js

Las expresiones regulares no deberían ser demasiado complicadas

Las expresiones regulares demasiado complicadas son difíciles de leer y mantener y pueden causar fácilmente errores difíciles de encontrar. Si una expresión regular es demasiado complicada, debería considerar reemplazarla partes de ella con código regular o dividirla en varios patrones al menos.

La complejidad de una expresión regular se determina de la siguiente manera:

Cada uno de los siguientes operadores aumenta la complejidad en una cantidad igual al nivel de anidación actual y también aumenta el nivel de anidación actual en uno para sus argumentos:

- | - cuando es múltiple operadores | se utilizan juntos, los siguientes solo aumentan la complejidad en 1

- Cuantificadores (*, +, ?, {n,m}, {n,} or {n})
- Aserciones de búsqueda anticipada y retrospectiva

- Además, cada uso de las siguientes características aumenta la complejidad en 1 independientemente del anidamiento:

  ✓ clases de caracteres

  ✓ referencias posteriores

Si una expresión regular se divide entre varias variables, la complejidad se calcula para cada variable individualmente, no para toda la expresión regular.

Corregir
GoalStadium/public/js/app.js

Bloques duplicados

Los archivos de origen no deben tener bloques duplicados. Se crea un problema en un archivo tan pronto como hay al menos un bloque de código duplicado en este archivo

Corregir
GoalStadium/database/seeders/SettingSeeder.php
GoalStadium/resources/lang/en/validation.php

Los atributos obsoletos en HTML5 no deben usarse

Con la llegada de HTML5, muchos atributos antiguos quedaron obsoletos. Para garantizar la mejor experiencia de usuario, no se deben utilizar atributos obsoletos. En su lugar debe utilizarse el equivalente en CSS.

Corregir
GoalStadium/resources/views/index/partials/carousel.blade.php (atributo frameborder)
GoalStadium/resources/views/index/partials/restingvideo.blade.php (atributo frameborder)
GoalStadium/resources/views/index/partials/slider.blade.php (atributo frameborder)
GoalStadium/resources/views/index/partials/trainingvideo.blade.php (atributo frameborder)
GoalStadium/resources/views/index/partials/video.blade.php (atributo frameborder)

Es aconsejable que los videos tengan subtítulos

Para que su sitio llegue a más personas se deben proporcionar subtítulos para los videos. Esta regla plantea un problema cuando *video* no incluye al menos una etiqueta <track/> con el atributo *kind* establecido en *captions*, *descriptions* o al menos en *subtitles*. Se recomienda que *subtitles* no esté destinado a la accesibilidad sino a la traducción. Los *captions* se dirigen a personas con discapacidad auditiva y *descriptions* se dirigen a personas con discapacidad visual.

Ejemplo
Código sin *captions*

```
<video id="video" controls preload="metadata">
  <source src="resources/myvideo.mp4" type="video/mp4">
  <source src="resources/myvideo.webm" type="video/webm">

</video>
```

Código con *captions*

```
<video id="video" controls preload="metadata">
  <source src="resources/myvideo.mp4" type="video/mp4">
  <source src="resources/myvideo.webm" type="video/webm">
  <track label="English" kind="captions" srclang="en" src="resources/myvideo-en.vtt" default>
  <track label="Deutsch" kind="captions" srclang="de" src="resources/myvideo-de.vtt">
  <track label="Español" kind="captions" srclang="es" src="resources/myvideo-es.vtt">

</video>
```

Corregir
GoalStadium/resources/views/index/partials/characters.blade.php
Laboratory/Slider1.html
Laboratory/Slider2.html

Los selectores no deben duplicarse

La duplicación de selectores puede indicar un error de copiar y pegar. La regla detecta los siguientes tipos de duplicaciones:

- dentro de una lista de selectores en un solo conjunto de reglas
- para selectores duplicados en diferentes conjuntos de reglas dentro de una sola hoja de estilo.

Corregir
GoalStadium/public/assets/css/main.css
GoalStadium/public/css/app.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap-reboot.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap-reboot.rtl.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap.css

GoalStadium/.../vendor/bootstrap/dist/css/bootstrap.rtl.css
GoalStadium/.../owl.carousel/dist/assets/owl.carousel.css
GoalStadium/.../docs/assets/owlcarousel/assets/owl.carousel.css
GoalStadium/.../docs_src/assets/scss/custom.scss
GoalStadium/.../docs_src/assets/scss/docs.scss
GoalStadium/public/vendor/swiper/swiper-bundle.css
Laboratory/.../posthtml-parser/coverage/base.css
Laboratory/.../coverage/lcov-report/base.css
Laboratory/node_modules/swiper/swiper-bundle.css

Las asignaciones no deben realizarse desde sub-expresiones

Las asignaciones dentro de las subexpresiones son difíciles de detectar y, por lo tanto, hacen que el código sea menos legible. Idealmente, las sub-expresiones no deberían tener efectos secundarios.

Corregir
GoalStadium/public/js/app.js

Las asignaciones no utilizadas deben eliminarse

Un almacenamiento muerto ocurre cuando a una variable local se le asigna un valor que no es leído por ninguna instrucción posterior. Calcular o recuperar un valor solo para luego sobrescribirlo o desecharlo, podría indicar un error grave en el código. Incluso si no es un error, en el mejor de los casos es una pérdida de recursos. Por tanto, deben utilizarse todos los valores calculados.

Corregir
GoalStadium/public/js/app.js

Las funciones no deben tener demasiados parámetros

Una lista de parámetros larga puede indicar que se debe crear una nueva estructura para ajustar los numerosos parámetros o que la función está haciendo demasiadas cosas.

Corregir
GoalStadium/public/js/app.js

Las variables no deben sombrearse

Reemplazar o sombrear una variable declarada en un ámbito externo puede afectar fuertemente la legibilidad y, por lo tanto, la capacidad de mantenimiento de un fragmento de código. Además, podría llevar a los encargados de mantenimiento a introducir errores porque creen que están usando una variable, pero en realidad están usando otra.

Corregir
GoalStadium/public/js/app.js

Los operadores ternarios no deben estar anidados

El hecho de que pueda hacer algo no significa que deba hacerlo, y ese es el caso de las operaciones ternarias anidadas. Anidar operadores ternarios da como resultado el tipo de código que puede parecer claro como el día cuando lo escribe, pero seis meses después dejará a los mantenedores (o peor aún, a usted en el futuro) rascándose la cabeza y maldiciendo.

En su lugar, se debe optar por el lado de la claridad y use otra línea para expresar la operación anidada como una declaración separada.

Corregir
GoalStadium/public/js/app.js

No se deben usar labels

Los labels no se usan comúnmente y muchos desarrolladores no entienden cómo funcionan. Además, su uso dificulta el seguimiento del flujo de control, lo que reduce la legibilidad del código.

Corregir
GoalStadium/public/js/app.js

Categoría: Bugs

El elemento "<html>" debe tener un atributo de idioma

El elemento <html> debe proporcionar el atributo lang y/o xml: lang para identificar el idioma predeterminado de un documento. Permite que las tecnologías de asistencia, como los lectores de pantalla, proporcionen una experiencia de lectura cómoda adaptando la pronunciación y el acento al idioma. También ayuda a softwares de traducción braille, por ejemplo, indicándole que cambie los códigos de control para caracteres acentuados.

Otros beneficios de indicar el idioma son:

- Ayudar a los agentes de usuario a proporcionar definiciones de diccionario o ayudar a los usuarios a beneficiarse de las herramientas de traducción.

- Mejorar la clasificación de los motores de búsqueda.

Los atributos lang y xml:lang sólo pueden tomar un valor.

Ejemplo:

Código sin atributo de idioma

```
<!DOCTYPE html>
<html>
        <head>
                <title>Una página escrita en ingles</title>
                <meta content="text/html; charset=utf-8" />
        </head>


        <body>
        ...
        </body>

</html>
```

Código con atributo de idioma

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <title>Una página escrita en ingles</title>
                <meta content="text/html; charset=utf-8" />
```

```
        </head>


        <body>
        ...
        </body>
</html>
```

Corregir
GoalStadium/public/vendor/loadjs/examples/after-window-onload.html
GoalStadium/public/vendor/loadjs/examples/before-domcontentloaded.html
GoalStadium/public/vendor/loadjs/examples/before-window-onload.html
GoalStadium/public/vendor/loadjs/examples/crossdomaincss.html
GoalStadium/public/vendor/loadjs/examples/css-testonerror.html
GoalStadium/public/vendor/loadjs/examples/css-testpreload.html
GoalStadium/public/vendor/loadjs/examples/css-testpreloadperformance.html
GoalStadium/public/vendor/loadjs/examples/examples.html
GoalStadium/public/vendor/loadjs/examples/image.html
GoalStadium/public/vendor/loadjs/test/index.html
GoalStadium/public/vendor/owl.carousel/test/index.html
Laboratory/.../console-browserify/test/static/index.html
Laboratory/.../crypto-browserify/example/index.html
Laboratory/.../node_modules/terser/tools/props.html
Laboratory/.../node_modules/postcss/docs/api/index.html
Laboratory/node_modules/sprintf-js/demo/angular.html
Laboratory/node_modules/terser/tools/props.html
Laboratory/.../vm-browserify/example/run/index.html

El tag "<title>" debe estar presente en todas las páginas

Los títulos son importantes porque se muestran en los resultados del motor de búsqueda, así como en la barra de herramientas del navegador. Esta regla verifica que la etiqueta <head> contiene una etiqueta <title> y la etiqueta <html> una <head>.

Ejemplo
Código sin <title>

```
<html>

<body>
...
</body>


</html>
```

Código con <title>

```
<html>

<head>
<title>Algún título relevante</title>
</head>

<body>
...
</body>


</html>
```

Corregir
GoalStadium/public/vendor/loadjs/examples/after-window-onload.html
GoalStadium/public/vendor/loadjs/examples/before-domcontentloaded.html
GoalStadium/public/vendor/loadjs/examples/before-window-onload.html
GoalStadium/public/vendor/loadjs/examples/crossdomaincss.html
GoalStadium/public/vendor/loadjs/examples/css-testonerror.html
GoalStadium/public/vendor/loadjs/examples/css-testpreload.html
GoalStadium/public/vendor/loadjs/examples/css-testpreloadperformance.html
GoalStadium/public/vendor/loadjs/examples/examples.html
GoalStadium/public/vendor/loadjs/examples/image.html

Laboratory/.../crypto-browserify/example/index.html
Laboratory/.../node_modules/terser/tools/props.html
Laboratory/node_modules/sprintf-js/demo/angular.html
Laboratory/node_modules/terser/tools/props.html
Laboratory/.../vm-browserify/example/run/index.html

¡Las declaraciones “<!DOCTYPE>" deben aparecer antes de las etiquetas "<html>"

¡La declaración <!DOCTYPE> le indica al navegador web qué versión (X)HTML se está utilizando en la página y, por lo tanto, cómo interpretar los distintos elementos. Los validadores también lo tienen en cuenta para saber en qué reglas se debe forzar su cumplimiento. Siempre debe preceder a la etiqueta <html>.

Ejemplo
¡Código sin <!DOCTYPE>

```
<html>
...

</html>
```

¡Código con <!DOCTYPE>

```
<!DOCTYPE html>
<html>
...

</html>
```

Corregir
Laboratory/.../node_modules/terser/tools/props.html
Laboratory/node_modules/terser/tools/props.html
Laboratory/.../vm-browserify/example/run/index.html

Las etiquetas "<th>" deben tener los atributos "id" o "scope"

La asociación de encabezados <table>, es decir, elementos <th>, con sus celdas <td> permite a los lectores de pantalla anunciar el encabezado antes de los datos. Esto aumenta considerablemente la accesibilidad de las tablas a los usuarios con discapacidad visual.

Hay dos formas de hacerlo:

- Agregar un atributo *scope* a los encabezados <th>.
- Agregar un atributo *id* a los encabezados <th> y un atributo de encabezados a cada elemento <td>.

Se recomienda agregar atributos *scope* a los encabezados <th> siempre que sea posible. Use <th id="..."> y <td headers="..."> solo cuando <th scope="..."> no sea capaz de asociar celdas a sus encabezados. Esto sucede para tablas muy complejas que tienen encabezados que dividen los datos en múltiples subtablas. Puede consultar tutoriales de accesibilidad web W3C WAI para obtener más información.

Se debe tener en cuenta que las tablas complejas a menudo se pueden dividir en varias tablas más pequeñas, lo que mejora la experiencia del usuario.

Ejemplo
Código sin los atributos id o scope en la etiqueta <th>

```
<table border="1">
  <caption>Contact Information</caption>
  <tr>
    <td></td>
    <th>Name</th>                          <!-- Non-Compliant -->
    <th>Phone#</th>                        <!-- Non-Compliant -->
    <th>City</th>                        <!-- Non-Compliant -->
  </tr>
  <tr>
    <td>1.</td>
    <th>Joel Garner</th>                     <!-- Non-Compliant -->
    <td>412-212-5421</td>
    <td>Pittsburgh</td>
  </tr>
  <tr>
    <td>2.</td>
    <th>Clive Lloyd</th>                     <!-- Non-Compliant -->
    <td>410-306-1420</td>
    <td>Baltimore</td>
```

```
  </tr>

</table>
```

Código con el atributo scope en la etiqueta <th>

```
<table border="1">
  <caption>Contact Information</caption>
  <tr>
    <td></td>
    <th scope="col">Name</th>                    <!-- Compliant -->
    <th scope="col">Phone#</th>                   <!-- Compliant -->
    <th scope="col">City</th>                  <!-- Compliant -->
  </tr>
  <tr>
    <td>1.</td>
    <th scope="row">Joel Garner</th>              <!-- Compliant -->
    <td>412-212-5421</td>
    <td>Pittsburgh</td>
  </tr>
  <tr>
    <td>2.</td>
    <th scope="row">Clive Lloyd</th>              <!-- Compliant -->
    <td>410-306-1420</td>
    <td>Baltimore</td>
  </tr>

</table>
```

Corregir
Laboratory/.../posthtml-parser/coverage/index.html
Laboratory/.../coverage/lcov-report/index.html
Laboratory/.../lcov-report/posthtml-parser/index.html
Laboratory/.../coverage/posthtml-parser/index.html
Laboratory/.../node_modules/postcss/docs/api/index.html

Las tablas deben tener encabezados

Las tecnologías de asistencia, como los lectores de pantalla, utilizan encabezados <th> para proporcionar algo de contexto cuando los usuarios navegan por una tabla. Sin él, el usuario se pierde rápidamente en el flujo de datos. Los encabezados deben asociarse correctamente con las celdas <td> correspondientes mediante el uso de un atributo *scope* o encabezados y atributos *id*. Consulte Tutoriales de accesibilidad web W3C WAI para obtener más información.

No se planteará ningún problema en <table> utilizado con fines de diseño, es decir, cuando contiene un atributo role establecido como "presentation" o "none". Se recomienda tener en cuenta que utilizar <table> con fines de diseño es una mala práctica.

Ejemplo
Código sin <th>

```
<table> <!-- Noncompliant -->
  <tr>
    <td>Name</td>
    <td>Age</td>
  </tr>
  <tr>
    <td>John Doe</td>
    <td>24</td>
  </tr>
  <tr>
    <td>Alice Doe</td>
    <td>54</td>
  </tr>

</table>
```

Código con <th>

```
<table>
  <tr>
    <th scope="col">Name</th>
    <th scope="col">Age</th>
  </tr>
  <tr>
    <td>John Doe</td>
    <td>24</td>
  </tr>
  <tr>
```

```
  <td>Alice Doe</td>
  <td>54</td>
 </tr>

</table>
```

Corregir
Laboratory/.../lcov-report/posthtml-parser/index.js.html
Laboratory/.../coverage/posthtml-parser/index.js.html

Las propiedades no deben duplicarse

CSS permite nombres de propiedad duplicados, pero solo la última instancia de un nombre duplicado determina el valor real que se utilizará para él. Por lo tanto, cambiar los valores de otras apariciones de un nombre duplicado no tendrá ningún efecto y puede causar malentendidos y errores.

Esta regla ignora las sintaxis de las variables $sass, @less, y var(--custom-property).

Corregir
GoalStadium/public/assets/css/main.css
GoalStadium/.../owl.carousel/dist/assets/owl.theme.default.css
GoalStadium/.../owl.carousel/dist/assets/owl.theme.green.css
GoalStadium/.../docs/assets/owlcarousel/assets/owl.theme.default.css
GoalStadium/.../docs/assets/owlcarousel/assets/owl.theme.green.css
GoalStadium/.../docs_src/assets/scss/custom.scss
GoalStadium/.../vendor/owl.carousel/src/scss/_theme.scss
GoalStadium/.../swiper/modules/navigation/navigation.less
GoalStadium/.../swiper/modules/navigation/navigation.scss
GoalStadium/public/vendor/swiper/swiper-bundle.css

Las declaraciones no vacías deben cambiar el flujo de control o tener al menos un efecto secundario

Cualquier declaración (que no sea una declaración nula, que significa una declaración que contenga solo un punto y coma;) que no tenga efectos secundarios y no produzca un cambio de flujo de control, normalmente indicará un error de programación y, por lo tanto, debe refactorizarse.

Corregir
GoalStadium/public/assets/js/index.js
GoalStadium/public/assets/js/index2.js
GoalStadium/public/js/app.js

Los nombres de propiedad no deben duplicarse dentro de una clase o un objeto literal

JavaScript permite nombres de propiedad duplicados en clases y objetos literales, pero solo la última instancia de un nombre duplicado determina el valor real que se utilizará para él. Por lo tanto, cambiar los valores de otras apariciones de un nombre duplicado no tendrá ningún efecto y puede causar malentendidos y errores.

Definir un class con un constructor duplicado generará un error.

Antes de ECMAScript 2015, el uso de nombres duplicados generaba un error en el código de modo estricto de JavaScript.

Corregir
GoalStadium/public/assets/js/index.js

Las alternativas en expresiones regulares deben agruparse cuando se usan con anchors

En expresiones regulares, los anclajes (anchors) ^ y $ tienen mayor precedencia que el operador | . Entonces, en una expresión regular como ^alt1 | alt2 | alt3$, alt1 estaría anclado al principio, alt3 al final y alt2 no estaría anclado en absoluto. Por lo general, el comportamiento previsto es que todas las alternativas estén ancladas en ambos extremos. Para lograr esto, se debe utilizar un grupo no capturador en torno a las alternativas.

En los casos en los que se pretende que los anclajes solo se apliquen a una alternativa cada uno, agregar (no capturar) grupos alrededor de los anclajes y las partes a las que se aplican hará explícito qué partes están ancladas y evitará que los lectores malinterpreten la precedencia o la cambien porque asumen erróneamente que la precedencia no fue intencionada.

Corregir
GoalStadium/public/js/app.js

Los valores de retorno de funciones sin efectos secundarios no deben ignorarse

Cuando la llamada a una función no tiene efectos secundarios, ¿de qué sirve hacer la llamada si se ignoran los resultados? En tal caso, la llamada a la función es inútil y debe descartarse o el código fuente no se comporta como se esperaba.

Para evitar generar falsos positivos, esta regla desencadena problemas solo en una lista predefinida de objetos y funciones conocidos.

Corregir
GoalStadium/public/js/app.js

Los identificadores especiales no deben estar vinculados ni asignados

JavaScript tiene identificadores especiales que, aunque no están reservados, no deben usarse como identificadores. Incluyen:

- eval - evalúa una cadena como código JavaScript
- arguments: se utiliza para acceder a los argumentos de la función a través de propiedades indexadas.
- undefined: devuelto para valores y propiedades que aún no se han asignado
- NaN: no es un número; devuelto cuando fallan las funciones matemáticas.
- Infinity: cuando un número excede el límite superior de los números de coma flotante

Estas palabras no deben estar vinculadas ni asignadas, ya que al hacerlo sobrescribiría las definiciones originales de estos identificadores. Es más, asignar o vincular algunos de estos nombres generará un error en el código de modo estricto de JavaScript.

Corregir
GoalStadium/public/js/app.js

No se deben utilizar expresiones idénticas en ambos lados de un operador binario

Usar el mismo valor en ambos lados de un operador binario es casi siempre un error. En el caso de los operadores lógicos, es un error de copiar/pegar y, por lo tanto, un error, o simplemente es un código desperdiciado y debe simplificarse. En el caso de los operadores bit a bit y la mayoría de los operadores matemáticos binarios, tener el mismo valor en ambos lados de un operador produce resultados predecibles y debe simplificarse.

Corregir
GoalStadium/public/js/app.js

Las expresiones regulares no deben contener caracteres de control

Las entradas en la tabla ASCII debajo del código 32 se conocen como caracteres de control o caracteres no imprimibles. Como no son comunes en las cadenas de JavaScript, lo más probable es que el uso de estos caracteres invisibles en expresiones regulares sea un error.

Corregir
GoalStadium/public/js/app.js

Los clústeres de grafemas Unicode deben evitarse dentro de las clases de caracteres regex

Al colocar grupos de grafemas Unicode (caracteres que deben codificarse en varios puntos de código) dentro de una clase de caracteres de una expresión regular, esto probablemente dará lugar a un comportamiento no deseado.

Por ejemplo, el grupo de grafemas c̈ requiere dos puntos de código: uno para 'c', seguido de otro para el modificador de diéresis '\u{0308}'. Si se coloca dentro de una clase de carácter, como [c̈], la expresión regular considerará que la clase de carácter es la enumeración [c\u{0308}] en su lugar. Por lo tanto, coincidirá con todas las 'c' y todas las diéresis que no se expresen como un solo punto de código, lo que es muy poco probable que sea el comportamiento previsto.

Esta regla plantea un problema cada vez que se utilizan clústeres de grafemas Unicode dentro de una clase de caracteres de una expresión regular.

Corregir
GoalStadium/public/js/app.js

Los selectores de pseudoelementos deben ser válidos

Las especificaciones del W3C definen los selectores de pseudoelementos válidos. Solo se deben usar los selectores de pseudoelementos oficiales y específicos del navegador para obtener el impacto esperado en la representación final.

Corregir
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap-reboot.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap-reboot.rtl.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap.css
GoalStadium/.../vendor/bootstrap/dist/css/bootstrap.rtl.css
GoalStadium/public/vendor/bootstrap/scss/_reboot.scss
GoalStadium/.../vendor/bootstrap/scss/forms/_form-control.scss

5.3 Vulnerabilidades de baja gravedad encontradas en el proyecto

Categoría: Seguridad

Política permisiva de Cross-Origin Resource Sharing (CORS)

Asegúrese de que la política permisiva de CORS aplicada no pueda llegar a comprometer la seguridad.

En el pasado ha dado lugar a las siguientes vulnerabilidades:

- CVE-2018-0269
- CVE-2017-14460

La política Same origin en los navegadores evita, de forma predeterminada y por razones de seguridad, una interfaz de JavaScript para realizar una solicitud HTTP de origen cruzado a un recurso que tiene un origen diferente (dominio, protocolo o puerto) del suyo. El destinatario que recibe la solicitud puede agregar encabezados HTTP adicionales en la respuesta, llamados CORS, que actúan como directivas para el navegador y cambia la política de control de acceso.

Puede ser un riesgo para la seguridad si:

- no confía en el origen especificado, por ejemplo: Access-Control-Allow-Origin: web_no_confiable.com.
- La política de control de acceso está completamente deshabitada: Access-Control-Allow-Origin: *
- Su política de control de acceso se define dinámicamente mediante una entrada controlada por el usuario, como el encabezado de origen.

El encabezado *Access-Control-Allow-Origin* debe configurarse solo para un origen confiable y para recursos específicos. Permita solo dominios de confianza y seleccionados en el encabezado Access-Control-Allow-Origin. Es preferible incluir dominios en listas blancas en lugar de usar listas negras.
No se debe usar comodines (*) ni devolver el contenido del encabezado de Origin sin ninguna verificación.

Revisar
GoalStadium/config/cors.php

Fragmento de código a revisar

```
'paths' => ['api/*', 'sanctum/csrf-cookie'],
'allowed_methods' => ['*'],
'allowed_origins' => ['*'],
'allowed_origins_patterns' => [],
'allowed_headers' => ['*'],
```

Ausencia de función de integridad de los recursos

Hay que asegurarse de no utilizar la función de integridad de los recursos es seguro aquí. Obtener recursos externos, por ejemplo, de una CDN, sin verificar su integridad podría afectar la seguridad de una aplicación si la CDN se ve comprometida y los recursos son reemplazados por maliciosos. La función de integridad de recursos bloqueará la inclusión de recursos en una aplicación si el resumen precalculado del recurso esperado no coincide con el resumen del recurso recuperado. Es un riesgo si los recursos se obtienen de CDN externos.

Se recomienda implementar comprobaciones de integridad de recursos para todos los recursos estáticos (donde "estático" significa que el contenido del recurso no cambia dinámicamente según el navegador).
También utilizar recursos versionados en lugar de utilizar la versión "más reciente" (latest) de los recursos.

Ejemplo

```
<script src="https://cdnexample.com/script.js" integrity="sha384-
oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
>
</script>
```

Corregir
GoalStadium/public/vendor/rangetouch/docs/index.html
(<script src="https://cdn.shr.one/1.0.1/shr.js"></script>)
GoalStadium/resources/views/index/index.blade.php
(<script src="https://unpkg.com/swiper/swiper-bundle.min.js"></script>)
Laboratory/Slider1.html
(<script src="https://unpkg.com/swiper/swiper-bundle.min.js"></script>)
Laboratory/Slider2.html
(<script src="https://unpkg.com/swiper/swiper-bundle.min.js"></script>)
Laboratory/node_modules/sprintf-js/demo/angular.html

(<scriptsrc="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-rc.3/angular.min.js"></script>)

Ausencia de rel="noopener"

Cuando un enlace abre una URL en una nueva pestaña con target="_blank", es muy sencillo que la página abierta cambie la *location* de la página original porque la variable de JavaScript window.opener no es nula (null) y, por lo tanto, window.opener.location se puede configurar mediante la página abierta. Esto expone al usuario a ataques de phishing.

Imagine un enlace publicado en un comentario de un sitio web popular que abre una nueva pestaña cambiando la URL de la página original. En la nueva pestaña accedes a una página de inicio de sesión falsa similar a la del sitio web verdadero pero controlada por el atacante y le pide al usuario que vuelva a iniciar sesión, fingiendo que la sesión acaba de expirar.

Para evitar que las páginas abusen de window.opener, use rel=noopener en <a href=> para forzar que su valor sea nulo en las páginas abiertas.

En Chrome 88+, Firefox 79+ o Safari 12.1+ se habilita la protección rel=noopener por defecto en links que usan target = _blank.

Como excepción, no se generará ningún problema cuando *href* contenga una URL relativa codificada, ya que allí tiene menos posibilidades de ser vulnerable. Una URL se considera codificada y relativa si no comienza con http:// o https://, y si no contiene ninguno de los caracteres {}$()[]

Puede ser un problema si:

- La aplicación abre una URL externa que no es de confianza en una nueva pestaña usando target=_blank.

- La aplicación es compatible con navegadores web anteriores a Chrome 88, Firefox 79 o Safari 12.1.

Se recomienda el uso de rel=noopener en <a href=> para evitar que las páginas que no son de confianza abusen de window.opener.

Corregir
GoalStadium/.../vendor/owl.carousel/docs/demos/animate.html
GoalStadium/.../vendor/owl.carousel/docs/demos/autoheight.html
GoalStadium/.../vendor/owl.carousel/docs/demos/autoplay.html
GoalStadium/.../vendor/owl.carousel/docs/demos/autowidth.html
GoalStadium/.../vendor/owl.carousel/docs/demos/basic.html
GoalStadium/.../vendor/owl.carousel/docs/demos/center.html

GoalStadium/.../vendor/owl.carousel/docs/demos/demos.html
GoalStadium/.../vendor/owl.carousel/docs/demos/events.html
GoalStadium/.../vendor/owl.carousel/docs/demos/lazyLoad.html
GoalStadium/.../vendor/owl.carousel/docs/demos/merge.html
GoalStadium/.../vendor/owl.carousel/docs/demos/mousewheel.html
GoalStadium/.../vendor/owl.carousel/docs/demos/responsive.html
GoalStadium/.../vendor/owl.carousel/docs/demos/rtl.html
GoalStadium/.../vendor/owl.carousel/docs/demos/stagepadding.html
GoalStadium/.../vendor/owl.carousel/docs/demos/test.html
GoalStadium/.../vendor/owl.carousel/docs/demos/urlhashnav.html
GoalStadium/.../vendor/owl.carousel/docs/demos/video.html
GoalStadium/.../vendor/owl.carousel/docs/docs/api-classes.html
GoalStadium/.../vendor/owl.carousel/docs/docs/api-events.html
GoalStadium/.../vendor/owl.carousel/docs/docs/api-options.html
GoalStadium/.../vendor/owl.carousel/docs/docs/dev-buildin-plugins.html
GoalStadium/.../vendor/owl.carousel/docs/docs/dev-external.html
GoalStadium/.../vendor/owl.carousel/docs/docs/dev-plugin-api.html
GoalStadium/.../vendor/owl.carousel/docs/docs/dev-styles.html
GoalStadium/.../vendor/owl.carousel/docs/docs/started-faq.html
GoalStadium/.../vendor/owl.carousel/docs/docs/started-installation.html
GoalStadium/.../vendor/owl.carousel/docs/docs/started-welcome.html
GoalStadium/.../vendor/owl.carousel/docs/docs/support-changelog.html
GoalStadium/.../vendor/owl.carousel/docs/docs/support-contact.html
GoalStadium/.../vendor/owl.carousel/docs/docs/support-contributing.html
GoalStadium/public/vendor/owl.carousel/docs/index.html
GoalStadium/public/vendor/rangetouch/docs/index.html
GoalStadium/public/vendor/rangetouch/docs/index.html
Laboratory/node_modules/posthtml-parser/coverage/index.html
Laboratory/.../posthtml-parser/coverage/lcov-report/index.html
Laboratory/.../coverage/lcov-report/posthtml-parser/index.html
Laboratory/.../coverage/lcov-report/posthtml-parser/index.js.html
Laboratory/.../posthtml-parser/coverage/posthtml-parser/index.html
Laboratory/.../posthtml-parser/coverage/posthtml-parser/index.js.html

Categoría: Bugs

Imagen, área o botón con etiquetas *image* deben tener un atributo "alt".

El atributo *alt* proporciona una alternativa textual a una imagen. Se utiliza siempre que no se pueda renderizar la imagen real.

Razones para usar *alt*:

- Para usuarios con discapacidad visual que utilizan un software lector de pantalla.
- Para usuarios con la carga de imágenes deshabilitada para reducir el consumo de datos en teléfonos móviles.
- Para situaciones en las que no se puede encontrar la imagen.

Sin embargo, no se debe establecer un atributo *alt* en un item no informativo. Imágenes que no dan ninguna información al usuario no necesitan el atributo alt. Para estos casos, como imágenes decorativas, es mejor utilizar una imagen de fondo CSS en lugar de una etiqueta <img>. Si no es posible utilizar una imagen de fondo CSS, se tolera un alt ="" vacío.

Revisar
GoalStadium/resources/views/index/partials/carousel.blade.php
GoalStadium/resources/views/index/partials/characters.blade.php
Laboratory/Slider1.html
Laboratory/Slider2.html

La etiqueta "<frames>" debe tener un atributo "title"

Los marcos permiten juntar diferentes páginas web en el mismo espacio visual. Los usuarios sin discapacidades pueden escanear fácilmente el contenido de todos los marcos a la vez. Sin embargo, los usuarios con discapacidad visual que utilizan lectores de pantalla escuchan el contenido de la página de forma lineal.

El atributo title se utiliza para enumerar todos los marcos de la página, lo que permite a esos usuarios navegar fácilmente entre ellos. Por lo tanto, las etiquetas <frame> y <iframe> siempre deben tener un atributo title.

Ejemplo
Código sin atributo title en la etiqueta frames

```
<frame src="index.php?p=menu">

<frame src="index.php?p=home" name="contents">
```

Código con atributo title en la etiqueta frames

```
<frame src="index.php?p=menu" title="Navigation menu">

<frame src="index.php?p=home" title="Main content" name="contents">
```

Corregir
GoalStadium/resources/views/index/partials/carousel.blade.php (etiqueta iframe)

Deben usarse las etiquetas "" y ""

Las etiquetas y equivalen a <b> y <i> en la mayoría de los navegadores web, pero hay una diferencia fundamental entre ellas: y tienen un significado semántico mientras que <b > y <i> solo transmiten información de estilo como CSS.

Si bien <b> simplemente no puede tener ningún efecto en algunos dispositivos con pantalla limitada o cuando una persona ciega utiliza un software de lectura de pantalla, sí:

- Muestra el texto en negrita en navegadores normales
- Habla en un tono más bajo cuando se utiliza un lector de pantalla como Jaws

En consecuencia:

- para transmitir semántica, las etiquetas <b> y <i> nunca deben usarse.
- para transmitir información de estilo, se deben evitar <b> y <i> y se debe usar CSS en su lugar.

Corregir
Laboratory/node_modules/brfs/example/robot.html
Laboratory/node_modules/brfs/test/files/robot.html

Las etiquetas "<table>" deben tener una descripción

Para que los usuarios con discapacidad visual puedan acceder a ellas, es importante que las tablas proporcionen una descripción de su contenido antes de acceder a los datos.

La forma más sencilla de hacerlo, y también la recomendada por WCAG2, es agregar un elemento <caption> dentro de <table>.

Corregir
Laboratory/.../posthtml-parser/coverage/index.html
Laboratory/.../coverage/lcov-report/index.html
Laboratory/.../lcov-report/posthtml-parser/index.html
Laboratory/.../lcov-report/posthtml-parser/index.js.html
Laboratory/.../coverage/posthtml-parser/index.html
Laboratory/.../coverage/posthtml-parser/index.js.html
Laboratory/.../node_modules/postcss/docs/api/index.html

Los parámetros de función, las excepciones capturadas y los valores iniciales de las variables foreach no deben ignorarse

Si bien es técnicamente correcto asignar parámetros desde dentro de los cuerpos de las funciones, reduce la legibilidad del código porque los desarrolladores no podrán saber si se está accediendo al parámetro original o alguna variable temporal sin pasar por toda la función. Además, algunos desarrolladores también pueden esperar que las asignaciones de parámetros de función sean visibles para las personas que llaman, lo que no es el caso, y esta falta de visibilidad podría confundirlos. En su lugar, todos los parámetros, las excepciones capturadas y los parámetros foreach deben tratarse como constantes.

Corregir
GoalStadium/public/js/app.js

Categoría: Malas prácticas

Seguimiento de las etiquetas "TODO"

Las etiquetas TODO se usan comúnmente para marcar lugares donde se requiere más código, pero que el desarrollador desea implementar más adelante.

Se deben revisar el código con estas etiquetas y garantizar que no pasan desapercibidas.
Revisar
Laboratory/node_modules/json-schema/draft-zyp-json-schema-04.xml

Los cuantificadores de expresión regular y las clases de caracteres deben usarse de forma concisa

Con la sintaxis de expresiones regulares, es posible expresar lo mismo de muchas formas. Por ejemplo, para hacer coincidir un número de dos dígitos, se podría escribir [0-9]{2,2} o \d{2}. Luego no solo es más corto en términos de longitud de expresión, sino que también es más fácil de leer y, por lo tanto, de mantener. Esta regla recomienda reemplazar algunos cuantificadores voluminosos y clases de caracteres con equivalentes más concisos:

- \d para [0-9] y \D para [^0-9]
- \w para [A-Za-z0-9_] y \W para [^A-Za-z0-9_]
- . para clases de caracteres que coincidan con todo (por ejemplo, [\w\W], [\d\D] o [\s\S] con la bandera s)
- x? para x{0,1}, x* para x{0,}, x+ para x{1,}, x{N} para x{N,N}

Corregir
GoalStadium/public/js/app.js

Deben eliminarse las variables y funciones locales no utilizadas

Si se declara una variable local o una función local pero no se usa, es un código muerto y debe eliminarse. Si lo hace, mejorará la capacidad de mantenimiento porque los desarrolladores no se preguntarán para qué se utiliza la variable o función.

Corregir
GoalStadium/public/js/app.js

"for of" debe usarse con Iterables

Si tiene un iterable, como una matriz, conjunto o lista, su mejor opción para recorrer sus valores es la sintaxis for of. Se debe usar un contador y obtener el comportamiento correcto, pero su código no es tan limpio o claro.

Corregir
GoalStadium/public/js/app.js

Se deben quitar los puntos y comas adicionales

Los puntos y comas adicionales (;) generalmente se introducen por error, por ejemplo, porque:

- Estaba destinado a ser reemplazado por una declaración real, pero esto se olvidó.
- Hubo un error tipográfico que hizo que el punto y coma se duplicase, es decir;;.
- Hubo un malentendido acerca de dónde se requieren o son útiles los puntos y comas.

Corregir
GoalStadium/public/js/app.js

Los literales booleanos no deben usarse en comparaciones

Deben evitarse los literales booleanos en las expresiones de comparación == y != para mejorar la legibilidad del código.

Esta regla también informa sobre operaciones booleanas redundantes.

Corregir
GoalStadium/public/js/app.js

6. Resultados

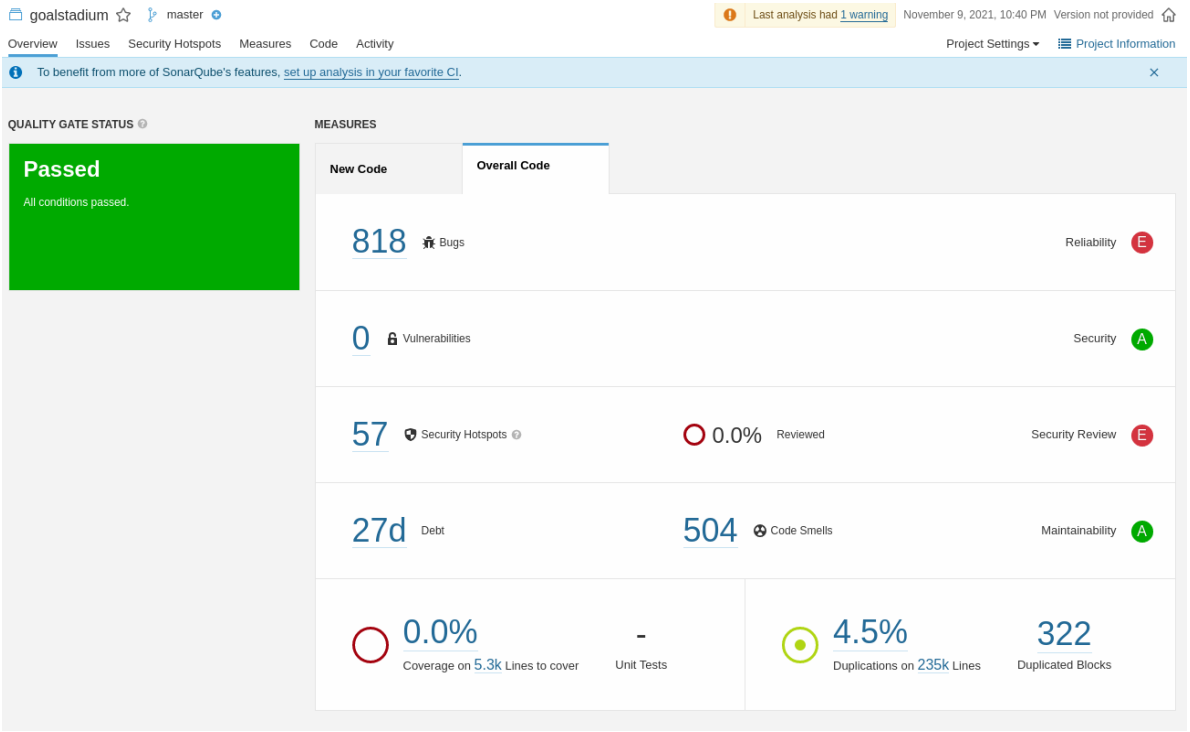
Por un lado, el contrato inteligente tiene un archivo de 314 líneas de código. Todos los comentarios de función y variable de estado siguen el formato de descripción estándar para escribir lo que puede ayudarnos a comprender rápidamente cómo funciona el programa.

Se ha realizado un correcto uso de la librería ERC20 y esto hace que el código generado sea pequeño, conciso y sencillo de seguir.

En cuanto a la calidad del código no se ha encontrado ninguna mejora a realizar para el objetivo del contrato inteligente. Sólo se han informado de posibles mejoras en el código que no tienen ningún impacto en la usabilidad del contrato.

Por tanto, no se han encontrado vulnerabilidades de ningún tipo de gravedad y las recomendaciones de reparación deben ser examinadas por los desarrolladores y el éxito de la misma es un proceso continuo antes de hacerse público.

Por otro lado, el análisis del proyecto general, realizado con Sonarqube, concluye que no se han producido vulnerabilidades en el código. En cambio, se han encontrado posibles mejoras como se puede observar en la siguiente evidencia:



7. Descargo de responsabilidad

La auditoría realizada garantiza que no se han encontrado vulnerabilidades conocidas en la totalidad del código, pero no exime de que posteriormente puedan aparecer algunas de estas. Por ello, desde Ciberseguridad e Inteligencia Artificial se recomienda hacer un seguimiento continuo del proyecto desarrollado mediante auditorías de token. A lo largo de estas auditorías se puede analizar y evaluar el estado de este código pudiendo realizar comparaciones con las auditorías anteriores.

El seguimiento continuo es imprescindible para asegurar que no se produzcan posibles fugas de información o activos tanto de la empresa como de los usuarios, evitando así fallos de seguridad que afecten al software.

El presente documento se basa en proveer seguridad en el código asumiendo que la empresa desarrolladora ha tenido en cuenta la normativa vigente. Es decir, el objetivo es más bien realizar un análisis exhaustivo sobre el código para intentar encontrar vulnerabilidades o minimizar el riesgo de fallos de seguridad en el proyecto. Sin embargo, no es un documento legalmente vinculante de la seguridad del código, la aplicabilidad del modelo comercial, el sistema regulatorio del modelo comercial o cualquier otra declaración de un estado libre de errores.