Informe de Calidad de Código y Vulnerabilidades

# Proyecto: Conejo Furioso

Fecha: [29/09/2024]

Herramientas de Análisis: SonarQube, SonarScanner

Tecnología del Proyecto: Python/Django

# 1. Resumen Ejecutivo

## 1.1 Objetivo del Informe

Este informe tiene como objetivo identificar y documentar las vulnerabilidades presentes en el código fuente del proyecto desarrollado en Python/Django. El análisis realizado con SonarQube y SonarScanner evalúa la calidad del código, destacando las áreas críticas que requieren atención para garantizar un producto final más robusto y eficiente.

## 1.2 Resumen de Resultados de Scan

- Líneas analizadas: 1.400

- Bugs: 51 issues de confiabilidad

- Vulnerabilidades: 1 vulnerabilidad de alta severidad, 43 security hotspots que requieren revisión

- Code Smells: 72 issues relacionados con la mantenibilidad

# 2. Resultados Detallados

## 2.1 Bugs

Descripción:

La mayoría de los bugs encontrados están relacionados con la accesibilidad y el uso de prácticas HTML obsoletas. Estos incluyen:

* Falta de la etiqueta `<title>` en vistas individuales que dependen de un template base, lo que resulta en un título genérico en todas las páginas.
* Uso de etiquetas `<th>` sin `id` o `scope`, lo cual afecta la accesibilidad de tablas.
* Etiquetas `<label>` no asociadas correctamente a controles de formulario.
* Elementos `<center>` obsoletos.
* Uso de etiquetas `<a>` como botones para ejecutar acciones, lo que no es semánticamente correcto.

Estos problemas afectan principalmente la accesibilidad y usabilidad del sitio, dificultando la interacción de usuarios que dependen de tecnologías asistivas, como los lectores de pantalla. Esto puede generar confusión, aumentar el esfuerzo necesario para la navegación y limitar la experiencia para usuarios con discapacidades.

Severidad: bajo

Archivo afectado: múltiples vistas en app/templates/app

Acciones recomendadas:

* Definir un `<title>` específico para cada vista que refleje el contenido o la función de la página.
* Agregar un atributo `id` o `scope` a las etiquetas `<th>` para mejorar la accesibilidad de las tablas.
* Asociar adecuadamente las etiquetas `<label>` a los controles de formulario mediante el atributo `for`.
* Eliminar los elementos `<center>` y usar CSS para centrar el contenido de manera moderna.
* Reemplazar los elementos `<a>` que ejecutan acciones con botones (`<button>`) y aplicar CSS para mantener la apariencia deseada.

## 2.2 Vulnerabilidades

Django Secret Key Exposed

Descripción:

La clave secreta de Django está expuesta en el archivo `settings.py`. En un entorno de producción, esto representa un riesgo de seguridad significativo, ya que los atacantes podrían aprovechar esta brecha para:

* Falsificar cookies de sesión o tokens, comprometiendo la autenticación de usuarios.
* Realizar ataques de CSRF y otros tipos de manipulación de solicitudes.

Severidad: Alta

Archivo afectado: ecommerce/settings.py

Acciones recomendadas:

Definir el secret key en las variables de entorno del sistema operativo en la máquina host del proyecto para posteriormente obtener el secret key a través de esta y así no setearlo explícitamente en el código fuente del proyecto.

## 2.3 Code Smells

**Descripción:**

Uno de los code smells más críticos que se ha detectado es la duplicación de código, afectando al 5.1% de las líneas totales. La duplicación de código incrementa el riesgo de inconsistencias a medida que el proyecto crezca. Los cambios futuros en una sección duplicada deberán aplicarse manualmente a todas las copias, lo que aumenta la deuda técnica y la posibilidad de errores.

Se ha detectado una duplicación de código significativa en las vistas (`views.py`), especialmente en las llamadas a las APIs. Esta duplicación podría volverse difícil de mantener a medida que se realicen más llamadas a otras APIs.

Severidad: Media

Archivo afectado: app/views.py

Acciones recomendadas:

* Refactorizar llamadas a la API: Definir rutas base en un archivo de configuración o crear funciones reutilizables para realizar las llamadas a las APIs, eliminando la duplicación de código.
* Centralizar la lógica de llamadas a la API: Implementar una función que maneje todas las llamadas GET a la API y permita un manejo centralizado de excepciones y parámetros.

# 3. Deuda Técnica y Refactorización

## 3.1 Análisis de la Deuda Técnica

El análisis ha revelado una deuda técnica significativa, derivada principalmente de la duplicación de código, problemas de accesibilidad, problemas de seguridad y el uso de prácticas obsoletas. Esta deuda técnica dificulta el mantenimiento del proyecto y eleva el costo de futuros desarrollos o cambios, lo que afecta la capacidad del equipo para escalar el código de manera eficiente.

## 3.2 Sugerencias para la Refactorización

Las áreas prioritarias que requieren atención para reducir la deuda técnica incluyen:

Métodos en views.py sin la protección necesaria.

Duplicación de código en las vistas (views.py), principalmente en las llamadas a la API.

Problemas de accesibilidad en el uso de etiquetas <th>, <label>, <a>, y la falta de etiquetas <title> específicas en las vistas.

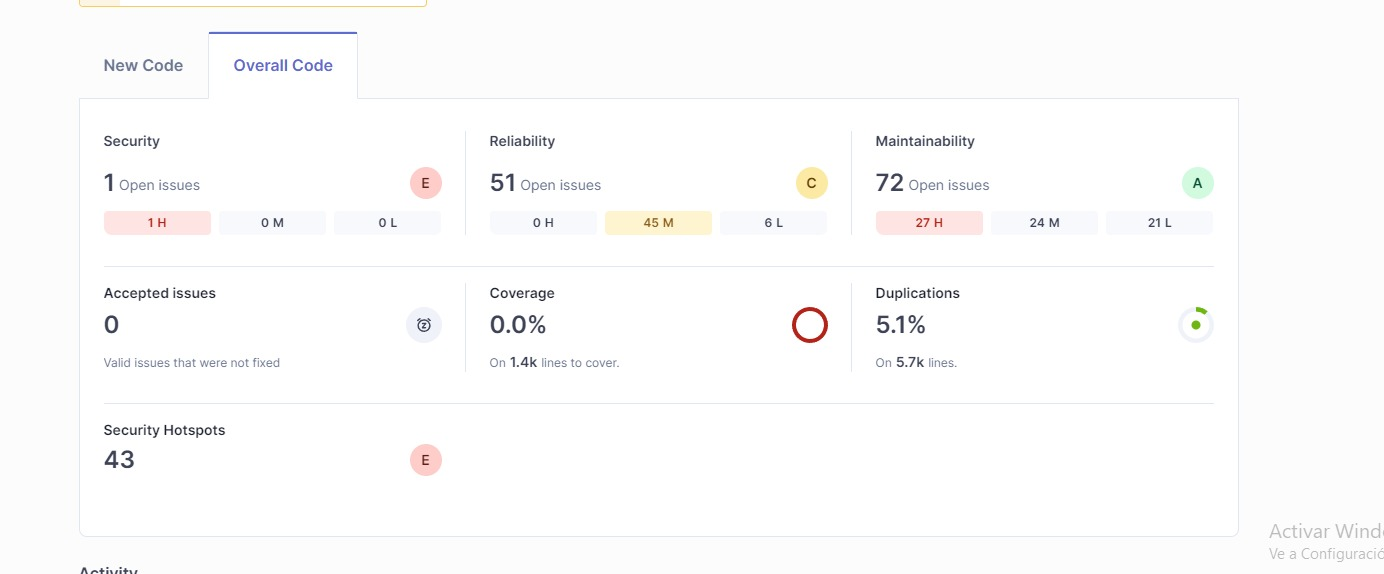
Eliminación de elementos obsoletos como <center> y reemplazo por soluciones CSS modernas.

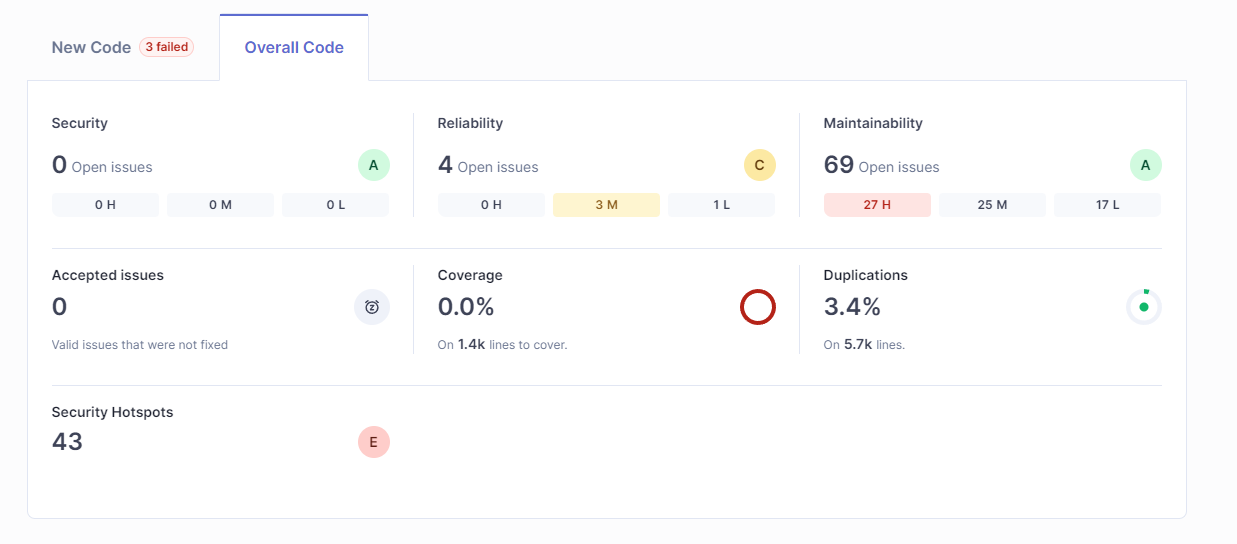
# 

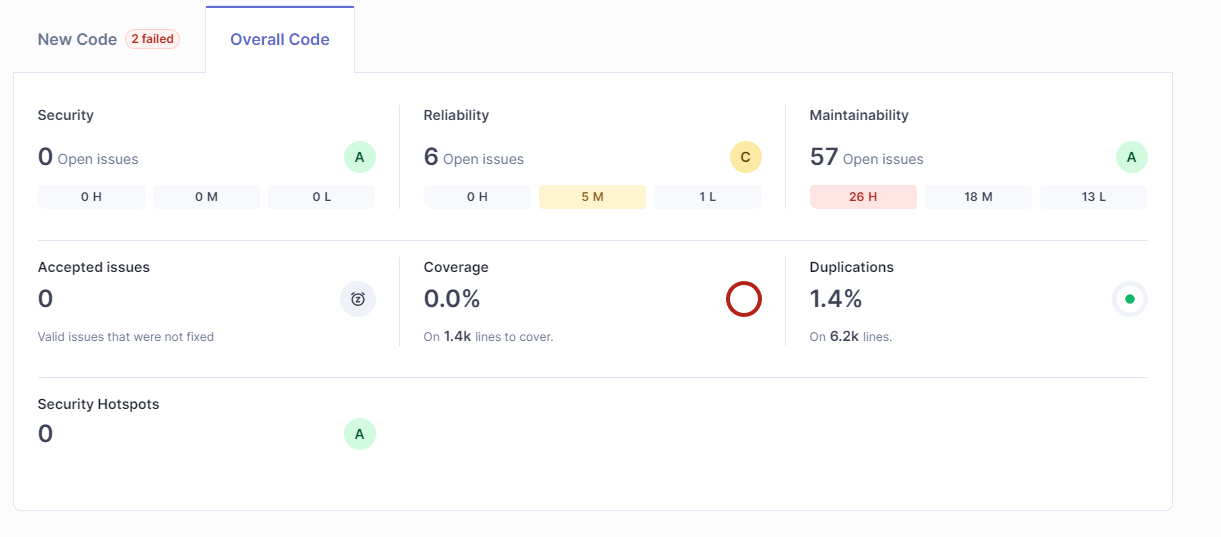
# 

# 4. Evidencias

Primer Scan realizado en SonarQube







Durante el proceso, se llevaron a cabo una serie de mejoras en el proyecto con el objetivo de mejorar la accesibilidad, seguridad, semántica, y la estructura del código HTML. A continuación, se detallan las modificaciones realizadas:

**Mejoras de accesibilidad y semántica en HTML:**

Etiquetas <th>: Se añadió el atributo scope="col" a las etiquetas <th> en las tablas para mejorar la accesibilidad para los usuarios que dependen de lectores de pantalla.

Etiquetas <label>: Se corrigieron las asociaciones incorrectas entre las etiquetas <label> y los campos <input>, asegurando que el atributo for coincida con el id del campo de formulario correspondiente.

Reemplazo de <center>: Se eliminó el uso obsoleto de la etiqueta <center> en varios lugares del proyecto y se reemplazó con el uso de la propiedad text-align: center; en CSS, utilizando tanto clases reutilizables como estilos en línea.

Botones (<button>) en lugar de enlaces (<a>): Se sustituyeron varios enlaces <a> que contenían botones <button> por botones <button> con eventos onclick, lo que mejora la semántica y mantiene una mejor estructura del código.

**Refactorización y mejora del código:**

Se eliminaron elementos obsoletos y se ajustaron los estilos duplicados o mal organizados en el código para mejorar la mantenibilidad y la claridad del mismo.

Se optimizó la semántica del código al separar la lógica de los enlaces de acción de botones que ejecutan funciones, logrando un diseño más coherente y de fácil mantenimiento.

**Seguridad:**

Se agregan los token csrf a todos los formularios y se agrega el método de django require\_http\_methods para proteger con GET y/o Post según sea el caso.

# 5. Conclusión

El análisis de calidad de código ha permitido identificar áreas críticas que requieren atención inmediata, como la exposición de la `SECRET\_KEY`, y áreas de mejora relacionadas con la accesibilidad, seguridad y la duplicación de código. Si bien los resultados generales son positivos, es fundamental corregir estos problemas para garantizar la escalabilidad, mantenibilidad, usabilidad y seguridad del proyecto a largo plazo. Se recomienda priorizar las acciones de refactorización y solucionar las vulnerabilidades identificadas para mejorar la calidad del producto final.