UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

Departamento Ingeniería Informática y Ciencias de la Computación

Análisis Estático

Desarrollado por:

Rodrigo Bascuñán León

Francisco Cifuentes Ramírez

Joseph Matamala Sepúlveda

Marcelo Vergara Fierro

Concepción, martes 14 de octubre del 2025

1. Introducción

El presente informe presenta el análisis estático del proyecto Workouts UdeC.
El análisis estático nos permite revisar el código fuente sin ejecutarlo, con el objetivo de detectar errores, malas prácticas o vulnerabilidades. Nos ayuda a mantener el código más limpio, legible, seguro y fácil de mantener, ya que permite corregir problemas tempranos antes que causen errores en tiempo de ejecución

2. Herramientas de Análisis Utilizadas

2.1 Pylint

Es una herramienta de análisis estático que examina el código fuente de Python con la finalidad de detectar problemas de estilo, estructura y mantenibilidad.

Evalúa si el código sigue convenciones PEP8, siendo esto la guía oficial del estilo de Python, ayudando a mantener una base de código coherente y legible. Busca errores comunes como; variables no utilizadas o mal nombradas, código duplicado o innecesariamente complejo, errores de sintaxis o malas prácticas de importación, ausencia de docstrings en módulos, clases o funciones, desviaciones de estilo (indentación, espaciado, longitudes de línea), etc.

Al final su ejecución, Pylint genera un puntaje del 0 al 10 que refleja la calidad del código, junto con advertencias y recomendaciones sobre los aspectos a mejorar

2.2 Bandit

Es una herramienta de análisis estático enfocada en la seguridad del código de Python.

Su función principal es detectar vulnerabilidades comunes y configuraciones inseguras antes de que el programa se ejecute. Analiza el código completo, línea por línea para identificar patrones que podrían presentar riesgos, como contraseñas codificadas (tokens, claves API hardocdeadas,etc.), funciones peligrosas (eval, exec, pickle), inyecciones (comandos SQL construidos por concatenación), configuraciones de red inseguras y practicas criptográficas débiles (como hashes inseguros, algoritmos obsoletos)

2.3 Radon

Es una herramienta de análisis estático de código que calcula métricas de código fuente: medidas cuantitativas que describen propiedades del sofware (como complejidad,

mantenibilidad, tamaño, etc) a partir de su código. Entre las principales métricas se encuentran la siguientes: complejidad ciclomática, metricas “raw”

Finalmente, estas métricas permiten evaluar la calidad del código, detectar posibles puntos de refactorización y mejorar su mantenibilidad y legibilidad antes de realizar pruebas o ejecutar el software.

3. Resultados de las herramientas aplicadas

3.1 Pylint

En el reporte de Pylint, los códigos más comunes encontrados en el análisis, ordenados por tipo, son:

- Convenciones
    - C0103: Invalid naming style (3 ocurrencias)
    - C0114: Missing module docstring (13 ocurrencias)
    - C0115: Missing class docstring (48 ocurrencias)
    - C0116: Missing function or method docstring (27 ocurrencias)
    - C0121: Singleton comparison (3 ocurrencias)
    - C0303: Trailing whitespace (41 ocurrencias)
    - C0304: Final newline missing (16 ocurrencias)
    - C0301: Line too long (15 ocurrencias)
    - C0411: Wrong import order (4 ocurrencias)
    - C0413: Wrong import position (1 ocurrencia)
    - C0415: Import outside toplevel (11 ocurrencias)

- Warnings
    - W0221: Arguments differ (1 ocurrencia)
    - W0404: Reimport (10 ocurrencias)
    - W0611: Unused import (3 ocurrencias)
    - W0613: Unused argument (2 ocurrencias)
    - W0621: Redefining name from outer scope (6 ocurrencias)
    - W0622: Redefining built-in (4 ocurrencias)

- Errores

    E1102: Function not callable (5 ocurrencias)

    E0213: No self argument (1 ocurrencia)

- Refactor

    R0903: Too few public methods (10 ocurrencias)

    R0913: Too many arguments (1 ocurrencia)

Puntaje Global: 5.3/10

Dado este análisis se pueden extraer las siguientes conclusiones.

La falta de docstrings representa casi un 40% de los códigos entregados por Pylint, por lo tanto, es el área con potencial de mejora más grande. Para solucionarlo se debe agregar descripciones para las funciones públicas, clases y archivos principales.

El segundo inciso más frecuente son los problemas de formato, los cuales representan otro 30% de los hallazgos, pero son fáciles de corregir automáticamente. Para corregir estos problemas, se puede utilizar un formateador automático como Black, que se asegure de quitar los espacios al final de las líneas, añadir una línea vacía al final del archivo y ajustar las líneas que exceden los 100 caracteres.

Otra de las posibles mejoras que encontró Pylint es del apartado de los imports y nombres, donde se encuentran imports que no se usan, más de un import para el mismo módulo, imports en un orden que no sigue la convención, y variables redefinidas. Para el caso de este proyecto, las partes del código que provocan las advertencias descritas en este apartado son necesarias para el buen funcionamiento del sistema, por lo que se pueden omitir en esta instancia.

Con respecto al R0903, indica que hay clases demasiado simples, sin embargo, en el caso de este proyecto, se puede ignorar, ya que las clases son solo contenedores de datos. El R0913 que señala funciones con demasiados argumentos, se puede solucionar con objetos para agrupar parámetros relacionados o fusionando funciones que pasen demasiados datos.

Por último, uno de los códigos de error encontrados se puede solucionar añadiendo un self a la función que lo requiere, sin embargo, los demás son un falso positivo, ya que Pylint no entiende la metaprogramación que hace el código, el cual está creando un objeto

sqlalchemy.sql.functions.Function que representa la función SQL NOW(), por lo que estos errores se pueden ignorar.

3.2 Bandit

En el análisis de seguridad que realiza Bandit solamente se encuentra un *issue*, el cual es descrito a continuación:

Issue: [B104:hardcoded_bind_all_interfaces] Possible binding to all interfaces. Severity: Medium Confidence: Medium CWE: CWE-605 (https://cwe.mitre.org/data/definitions/605.html) More Info: https://bandit.readthedocs.io/en/1.7.9/plugins/b104_hardcoded_bind_all_interfaces.html Location: app/main.py:28:26

El código entregado por el análisis indica que encuentra un riesgo de exponer servicios innecesariamente, este tiene severidad y confianza medias, lo cual no implica un peligro inminente, sino que su gravedad depende netamente de si está desplegada públicamente o no. La confianza media indica que Bandit no está seguro completamente de que este problema sea real (podría ser intencional), pero es lo bastante sospechoso como para creer que lo es.

Dado que el proyecto se ejecuta dentro de Docker, este no es un problema real, porque dentro del contenedor se decide si su puerto se publica o no al exterior, por ende, el acceso externo está controlado por Docker, no por el código. El hecho de que el proyecto no se despliegue públicamente y sin el uso de Docker, el problema de seguridad es irrelevante y se puede considerar como falso positivo.

3.3 Radon

El análisis ejecutado arrojó un muy buen comportamiento del proyecto respecto a la complejidad ciclomática de sus funciones y clases. Todas las funciones y clases presentan un nivel de complejidad “A”, lo que indica un código con lógica simple. Para haber logrado esta nota, el rango de complejidad deberá estar entre 1-5, sin embargo, los que tienen mayor complejidad pueden revisarse, y en caso de ser difícil de leer, probar o mantener, podemos realizar refactoring.

Los resultados con complejidad más alta fueron las siguientes:

- Complejidad 5

app/api/endpoints/workouts.py
F 56:0 create_workout_from_template - A (5)
F 222:0 add_set_to_exercise - A (5)
F 265:0 delete_exercise_set - A (5)

- Complejidad 4

app/api/endpoints/workouts.py
F 42:0 read_workout_template - A (4)
F 136:0 complete_workout - A (4)
F 158:0 cancel_workout - A (4)
F 204:0 add_exercise_to_workout - A (4)
F 244:0 update_exercise_set - A (4)
F 324:0 update_exercise_notes - A (4)

app/crud/crud_workout.py
M 156:4 CRUDWorkout.create_from_template - A (4)
M 31:4 CRUDBase.update - A (4)

app/crud/base.py
M 32:4 CRUDBase.update - A (4)

- Complejidad 3

app/api/endpoints/admin.py
F 31:0 create_user - A (3)
F 160:0 remove_exercise_from_template - A (3)

app/api/endpoints/workouts.py
F 26:0 read_workout_templates - A (3)
F 120:0 update_workout - A (3)
F 189:0 read_workout - A (3)
F 307:0 update_workout_notes - A (3)

app/api/endpoints/users.py
F 30:0 create_user_open - A (3)

app/api/dependencies.py
F 23:0 get_current_user - A (3)

app/api/endpoints/auth.py
F 16:0 login_access_token - A (3)

app/crud/crud_workout.py
M 80:4 CRUDWorkoutTemplate.update_template_exercises - A (3)
M 195:4 CRUDWorkout.cancel_workout - A (3)
M 275:4 CRUDWorkout.update_exercise_set - A (3)
M 309:4 CRUDWorkout.get_exercise_progression - A (3)
C 114:0 CRUDWorkout - A (3)

app/crud/crud_user.py
M 28:4 CRUDUser.update - A (3)
M 41:4 CRUDUser.authenticate - A (3)

Bloques analizados: 134
Complejidad promedio: A (1.72)

4. Conclusión

El análisis estático del proyecto WorkoutsUdeC realizado con Pylint, Bandit y Radon, se encontró que la implementación del proyecto tiene espacio para mejoras, especialmente en el área de convenciones de sintaxis. En cuanto a la complejidad ciclomática el proyecto si resulto tener muy buenos resultados.

En retrospectiva el uso de estas herramientas es muy útiles especialmente para casos como estos, en el que se deben hacer proyectos con usos prácticos y que afectan directamente al cliente, la utilidad reside en el poder detectar problemas de forma rápida, por ejemplo, con Pylint se pueden detectar todos los problemas relacionados con la sintaxis; con Bandit, problemas de seguridad que pueden comprometer los datos personales de los clientes; y con Radon, se puede revisar la complejidad ciclomática.

El grupo puede concluir que definitivamente estas herramientas son necesarias y potencialmente indispensables dentro del desarrollo, esto debido a que muchas veces los programadores a veces pasan por alto detalles como lo pueden ser de sintaxis, o no se dan cuenta de que algún método está creando un potencial problema de seguridad, complejidad ciclomática, entre otros.