UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

Departamento Ingeniería Informática y Ciencias de la Computación

Refactoring

Desarrollado por:

Rodrigo Bascuñán León

Francisco Cifuentes Ramírez

Joseph Matamala Sepúlveda

Marcelo Vergara Fierro

Concepción, martes 14 de octubre del 2025

1. Contexto de las refactorizaciones

Durante el análisis estático del proyecto se encontraron diversas oportunidades de mejora en cuanto a estructura, legibilidad y mantenibilidad del código. Aunque los resultados generales fueron bastante positivos, pudimos detectar ciertos métodos con posibilidades de mejora, como fue el análisis de Radon donde a pesar de todas las funciones, clases y métodos obtener una A en el análisis, existieron algunos que arrojaban complejidad ciclomática de 5, generando posibilidades de descender a B si se escala el código, esto motivaron a un revisión más detallada de las funciones, buscando simplificar estructuras y centralizar lógicas repetidas para mejorar la mantenibilidad.

Por otro lado, el análisis realizado con pylint nos entregó un puntaje global de 5.3/10, reflejando una base de código que funciona, pero que tiene muchas oportunidades de mejoras, un 40% es relacionado a falta de docstrings lo que evidencia la necesidad de mejorar en la documentación, además se detectaron problemas de formato e importaciones duplicadas, así como nombres pocos descriptivos y un error (E0213) vinculado a la ausencia del parámetro self en un método de clase.

Finalmente, algunas oportunidades de mejora no vinieron específicamente del análisis estático, sino de una revisión detallada y manual del código con su comportamiento hacia la base de datos, en este proceso se identificó la necesidad de implementar una eliminación en cascada (cascade delete) en SQL, con el fin de asegurar la integridad referencial y evitar inconsistencias al eliminar registros relacionados.

2. Motivos de las refactorizaciones

El refactor del formateo con la herramienta black viene por la necesidad de mantener un código limpio y lo más estandarizado posible para que su posterior revisión y o revisión sea más fácil.

En cuanto a la eliminación en cascada, este se vio pertinente ya que el que estaba anteriormente era una especie de borrado en cascada artificial en el cual había que declarar a mano todas las relaciones de la base de datos, esto genera un gran problema posteriormente ya que, ante una expansión del producto, con muchas más relaciones, cada vez serían más líneas de código y se tendría que cuidar siempre que borrar primero para lograr el borrado en cascada, es por ello que se refactorizo para utilizar el borrado en cascada de la librería SQLalchemy directamente

La centralización de validaciones en validate_workout_accessf ue motivada por la detección de duplicación de lógica en distintas funciones de workouts.py, Radon mostró que varias de estas funciones presentaban complejidad entre 4 a 5, con la refactorización se eliminó código duplicado y se logró un flujo más claro. Otro método que poseía una complejidad alta fue

update(), ya que presentaba estructuras condicionales innecesarias y varios retornos intermedios que complicaban la lectura y el seguimiento del flujo, la refactorización permitió reducir su complejidad a 1 y dejó el método en formato más línea y legible.

3. Soluciones propuestas y tipos de refactorización

Se aplicaron distintas estrategias de refactorización, continuación se detalla cada una de las refactorizaciones realizadas:

3.1. Refactor 1: Centralización de validación y reducción de complejidad en workouts.py

Tipo de refactorización: Extract Method / Reduce Cyclomatic Complexity

Commit: a58fef6

El archivo workouts.py pertece a los endpoints del backend, su función principal es gestionar las operaciones CRUD relacionadas con los entrenamientos, permite crear, actualizar, completar o cancelar una rutina. Este módulo recibe las peticiones HTTP del cliente y coordina las acciones con los controladores y las funciones de la base de datos. Antes del refactor, cada función realizaba manualmente sus propias validaciones de acceso, verificando si el usuario tenía permisos o si el entrenamiento existía, esto generaba una duplicación de validaciones, lo que desencadenaba en un aumento en el nivel de complejidad ciclomática, dificultaba el mantenimiento y podía provocar inconsistencias si alguna validación quedaba desactualizada.

La solución aplicada fue crear una función validate_workout_access() la cual se encarga de realizar todas las comprobaciones de acceso a los entrenamientos, luego fue llamada desde los distintos endpoints, así reduciendo el número de condiciones por función y eliminando código repetido.

3.2. Refactor 2: Corrección de E0213 de Pylint en base_class.py

Tipo de refactorización:

Commit: a106458

Se añadió el argumento self en la función __tablename__ de la clase Base en el archivo base_class.py, para solucionar el código E0213 de Pylint “No self argument”.

Es importante que el primer argumento de todo método debe ser la instancia sobre la que se llama, la cual por convención se denomina self. Si se omite este parámetro, el método no puede acceder a los atributos del objeto, o puede llegar a experimentar un comportamiento distinto al esperado.

3.3. Refactor 3: Formateo con Black

Tipo de refactorización: Formateo de sintaxis

Commit: c41a2d0

Se formateo todo el backend con Black mediante el comando '**black .\workouts_udec_backend**',

3.4. Refactor 3: Simplificados metodo update() en CRUDBase

Tipo de refactorización: Extract Method + Simplify Conditional / Reduce Cyclomatic Complexity

Commit: ce25fa1

El archive base.py contiene la clase CRUDBase, esta es responsable de gestionar operaciones genéricas de base de datos (CRUD) utilizadas por los diferentes modelos del sistema. Dentro de esta clase, el método update() presentaba estructuras condicionales innecesarias y múltiples puntos de retorno, lo que aumentaba su complejidad ciclomatica a un valor de 4 según Radon.

La refactorización consistió en dividir las responsabilidades del método principal mediante la extracción de funciones auxiliares privadas, se creó _get_update_data() para aislar la lógica de obtención de datos desde obj_in, se creó _apply_updates() para manejar de forma centralizada la asignación de nuevos valores a los atributos del objeto (setattr), finalmente se creó _save() para encapsular las operaciones de persistencia(add, commit, refresh) reutilizables tanto en créate() como en update(). Gracias a todo esto el método update() quedó reducido a tres llamadas secuenciales, reduciendo la complejidad ciclomatica de 4 a 1.

3.5. Refactor 5: Implementación de eliminación en cascada con SQLalchemy

Tipo de refactorización: Reestructuración de metodo

Commit: 7a34df1

Se remplazó toda la función de **delete_with_cascade** en una sola llamada para borrar el usuario y dejando que SQLAlchemy haga el resto, esto se logró añadiendo directamente en las relaciones de los modelos, el parámetro '**cascade="all, delete-orphan"**' lo que logra el efecto declarado anteriormente.

3.5 Refactor 6: Docstring

Tipo de refactorización: Documentación pertinente

Commit: a5dad32

Se agregó documentación Docstring a todos los métodos y clases en las que el reporte de Pylint encontró ausentes, es importante realizar este cambio dado que simplifica la comprensión del código, su legibilidad y mantenimiento, también acerca al proyecto a que se cumplan los estándares de calidad.

4. Resultados y aportes de las refactorizaciones

Tras aplicar las refactorizaciones:

El rating de pylint subió desde un 5.20 a un 8.22

El refactor del borrado en cascada logra que el código sea más mantenible y facilita el trabajo a futuro a la hora de quere trabajar en la base de datos, ya sea reestructurándola, ampliándola, o reduciéndola.

El refactor Black logra un código más estándar y legible, así facilitando su comprensión logrando así que el trabajo a futuro sea más efectivo y se pierda menos tiempo tratando de descifrar lo escrito.

Los refactor de reducción de complejidad lograron mejorar la mantenibilidad del código, reduciendo la complejidad promedio del proyecto desde A(1.72) a A(1.58), si bien ambas calificaciones corresponden al nivel óptimo según Radon, la disminución refleja una simplificación real en las estructuras de control.