Analisis Penerapan Closure Menggunakan Decorator pada Pengelolaan Akses Admin Dalam Suatu Aplikasi

Rahma Neliyana[1)], Eka Fidiya Putri[2)], Izza Lutfia[3)], Dea Mutia Risani[4)], Dinda Nababan[5)]

Program Studi Sains Data, Fakultas Sains, Institut Teknologi Sumatera

Email : rahma.122450036@student.itera.ac.id[1)], eka.122450045@student.itera.ac.id[2)], izza.122450090@student.itera.ac.id[3)], dea.122450099@student.itera.ac.id[4)], dinda.122450120@student.itera.ac.id[5)]

1.Pendahuluan

Pengelolaan akses admin dalam suatu aplikasi seringkali melibatkan berbagai aspek, mulai dari autentikasi pengguna hingga penentuan hak akses yang sesuai dengan peran dalam sistem. Dalam pengembangan suatu aplikasi, keamanan dan fleksibilitas dalam manajemen akses menjadi hal yang sangat penting. Salah satu teknik yang dapat digunakan untuk meningkatkan fleksibilitas dalam pengelolaan akses admin adalah dengan menerapkan closure menggunakan decorator.

Closure adalah sebuah fungsi yang dapat disimpan dalam variabel dan biasa dimanfaatkan untuk mewakili (*enclose*) sebuah proses pada blok tertentu. Variabel yang menyimpan *closure* memiliki sifat seperti fungsi yang disimpannya. Dengan menggunakan *closure*, fungsi dapat mengembalikan objek (atau fungsi lain) dan mengingat lingkungan tempat fungsi tersebut diinisiasi. Nilai yang ada pada outer *scope* masih diingat meskipun fungsi tersebut telah dihapus. *Closure* memiliki peran yang sangat penting dalam decorator. *Closure* digunakan untuk menyimpan nilai fungsi yang telah di dekorasi.

Dalam *python*, *decorator* adalah fungsi yang digunakan untuk mengubah perilaku fungsi atau metode lain tanpa mengubah kode sumber asli. *Decorator* dalam artikel ini digunakan untuk memeriksa apakah pengguna yang memanggil suatu fungsi adalah admin atau bukan agar dapat mengubah, menambahkan serta mengedit produk dalam suatu aplikasi. Hal ini mempermudah admin untuk menjaga produk menjadi lebih bersih dan mudah di pelihara.

2. Metode

2.1. Closure

Closure adalah sebuah fungsi yang dapat disimpan dalam variabel dan biasa dimanfaatkan untuk mewakili (*enclose*) sebuah proses pada blok tertentu. Variabel yang menyimpan *closure* memiliki sifat seperti fungsi yang disimpannya. Dengan menggunakan *closure*, fungsi dapat mengembalikan objek (atau fungsi lain) dan mengingat lingkungan tempat fungsi tersebut diinisiasi. Nilai yang ada pada outer scope masih diingat meskipun fungsi tersebut telah dihapus. *Closure* memiliki peran yang sangat penting dalam decorator. *Closure* digunakan untuk menyimpan nilai fungsi yang telah di dekorasi.

2.2. Decorator

Decorator adalah suatu konsep dalam bahasa pemrograman *python*. *Decorator* dapat memodifikasi atau melengkapi fungsi atau metode dengan cara yang bersih dan elegan. *Decorator* sangat berguna untuk mengorganisir dan mengelola kode yang di buat. *Decorator* juga adalah fungsi yang digunakan untuk mengubah perilaku fungsi atau metode lainnya tanpa mengubah kode sumber asli. *Decorator* bekerja sebagai lapisan ekstra yang membungkus fungsi yang ada dan memungkinkan untuk menambahkan fungsionalitas tambahan sebelum atau sesudah fungsi tersebut dijalankan. *Decorator*

berguna dalam berbagai kasus penggunaan seperti validasi input atau parameter fungsi, *logging* aktivitas, mengukur waktu eksekusi fungsi, mengelola izin atau otentikasi dan memproses serta memanipulasi hasil fungsi [1].

3. Pembahasan

3.1. Kode dan Hasil Pemrograman

Berikut merupakan kode dan hasil pemrograman yang telah kami buat. Secara keseluruhan kode ini menjelaskan tentang penerapan *closure* menggunakan *decorator* pada pengelolaan akses admin dalam suatu aplikasi.

```
2 def require_admin(func):
3     def wrapper(*args, **kwargs):
4         if is_user_admin():
5             return func(*args, **kwargs)
6         else:
7             return "Akses ditolak. Anda bukan admin toko."
8     return wrapper
```

Gambar 1. *Decorator Require Admin*

Gambar 1 menjelaskan mengenai sebuah *decorator* Python yang bernama require_admin. *Decorator* ini bertujuan untuk memeriksa apakah pengguna yang memanggil suatu fungsi adalah admin atau bukan. Jika pengguna tersebut adalah admin, maka fungsi yang didekorasi akan dijalankan seperti biasa dengan argumen dan *keyword* arguments yang diteruskan ke dalamnya. Namun, jika pengguna bukan admin, *decorator* akan mengembalikan pesan "Akses ditolak. Anda bukan admin toko." tanpa menjalankan fungsi yang didekorasi. Dengan menggunakan *decorator* ini, penggunaan kode untuk memeriksa peran admin dapat disederhanakan dengan menambahkan *decorator* *@require_admin* di atas definisi fungsi yang membutuhkan akses admin. Kode pada gambar 1 juga menggunakan prinsip *closure* yaitu pada fungsi *wrapper* yang berperan sebagai penutup (*closure*) yang menangkap lingkungan (*environment*) di sekitarnya. Kode ini memungkinkan kita untuk menambahkan fungsionalitas tambahan sebelum atau sesudah fungsi yang didekorasi dieksekusi.

```
@require_admin
def tambah_produk(nama_produk, harga):
    # Logika untuk menambahkan produk
    return f"Produk {nama_produk} berhasil ditambahkan dengan harga {harga}."
```

Gambar 2. Menambahkan produk dengan *decorator require admin*

Gambar 2 menjelaskan mengenai fungsi bernama tambah_produk yang memiliki dua parameter, yaitu nama_produk dan harga. Ada juga sebuah *decorator* *@require_admin*, yang mengindikasikan bahwa fungsi ini hanya dapat diakses oleh admin. Fungsi tersebut seharusnya mempunyai logika untuk menambahkan produk baru dengan nama dan harga yang diberikan, tetapi logika tersebut belum diimplementasikan dalam kode. Fungsi hanya mengembalikan sebuah string yang menyatakan bahwa produk berhasil ditambahkan beserta nama dan harga yang diberikan. Namun, perlu dicatat bahwa tanpa implementasi logika yang sesungguhnya, fungsi ini tidak akan melakukan apa pun dalam aplikasi yang sebenarnya.

```python
@require_admin
def edit_produk(id_produk, nama_baru, harga_baru):
    # Logika untuk mengedit produk
    return f"Produk dengan ID {id_produk} berhasil diubah menjadi {nama_baru} dengan harga {harga_baru}."
```

Gambar 3. Mengedit produk dengan *decorator require admin*

Gambar 3 menjelaskan mengenai fungsi bernama edit_produk yang memiliki tiga parameter: id_produk, nama_baru, dan harga_baru. Fungsi ini diberi *decorator @require_admin*, menunjukkan bahwa hanya admin yang dapat mengaksesnya. Namun, seperti fungsi sebelumnya, logika aktual untuk mengedit produk belum diimplementasikan dalam kode tersebut. Fungsi hanya mengembalikan string yang memberitahukan bahwa produk dengan ID tertentu berhasil diubah dengan nama dan harga baru yang diberikan. Meskipun demikian, tanpa implementasi logika yang sesungguhnya, fungsi ini tidak akan memiliki efek apapun dalam aplikasi yang sebenarnya.

```python
def is_user_admin():
    return True if input("Apakah Anda admin? (ya/tidak): ").lower() == 'ya' else False
```

Gambar 4. Fungsi untuk memeriksa apakah pengguna adalah admin

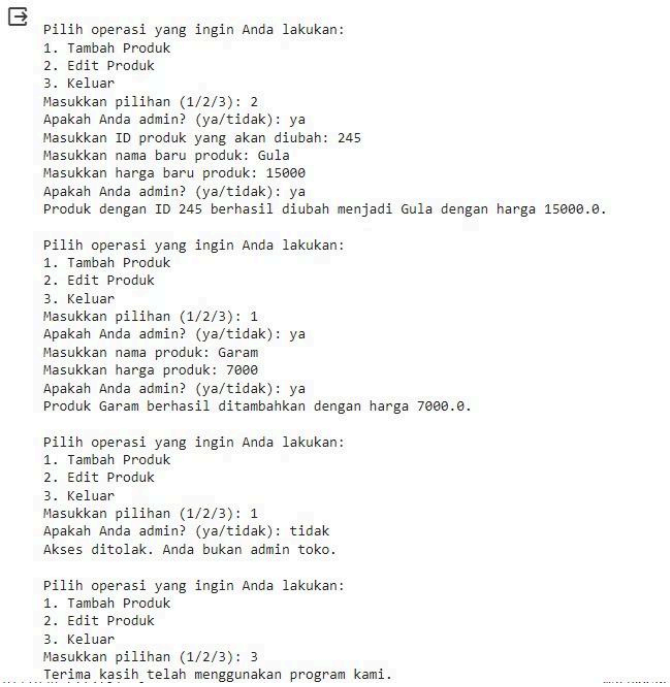
Gambar 4 menjelaskan mengenai fungsi bernama *is_user_admin* yang tidak memiliki parameter. Fungsi ini bertujuan untuk menanyakan kepada pengguna apakah mereka adalah admin dengan menggunakan fungsi input untuk menerima masukan. Jika pengguna menjawab 'ya', fungsi akan mengembalikan *True*, dan jika tidak, akan mengembalikan *False*. Namun, perlu dicatat bahwa pendekatan ini tergantung pada masukan pengguna dan mungkin tidak cocok untuk aplikasi yang membutuhkan verifikasi otomatis keanggotaan admin. Selain itu, penggunaan fungsi input ini akan membuatnya sulit untuk diuji secara otomatis atau dalam konteks penggunaan di luar lingkungan interaktif.

```python
# Interaksi dengan pengguna
while True:
    print("\nPilih operasi yang ingin Anda lakukan:")
    print("1. Tambah Produk")
    print("2. Edit Produk")
    print("3. Keluar")

    pilihan = input("Masukkan pilihan (1/2/3): ")

    if pilihan == '1':
        if is_user_admin():
            nama_produk = input("Masukkan nama produk: ")
            harga = float(input("Masukkan harga produk: "))
            print(tambah_produk(nama_produk, harga))
        else:
            print("Akses ditolak. Anda bukan admin toko.")
    elif pilihan == '2':
        if is_user_admin():
            id_produk = int(input("Masukkan ID produk yang akan diubah: "))
            nama_baru = input("Masukkan nama baru produk: ")
            harga_baru = float(input("Masukkan harga baru produk: "))
            print(edit_produk(id_produk, nama_baru, harga_baru))
        else:
            print("Akses ditolak. Anda bukan admin toko.")
    elif pilihan == '3':
        print("Terima kasih telah menggunakan program kami.")
        break
    else:
        print("Pilihan tidak valid. Silakan masukkan pilihan yang benar (1/2/3).")
```

Gambar 5. Interaksi dengan Pengguna

Gambar 5 menjelaskan bahwa program ini memiliki sebuah *loop* yang akan terus berjalan sampai pengguna memilih opsi "Keluar". Pada setiap iterasi, program akan

menampilkan menu dengan tiga opsi: "Tambah Produk", "Edit Produk", dan "Keluar". Pengguna dapat memilih salah satu opsi dengan memasukkan angka 1, 2, atau 3. Jika pengguna memilih opsi "Tambah Produk" atau "Edit Produk", program akan memeriksa apakah pengguna adalah admin menggunakan fungsi *is_super_admin*. Jika pengguna adalah admin, program akan menjalankan fungsi tambah produk atau edit_produk sesuai dengan pilihan pengguna. Jika pengguna bukan admin, program akan menampilkan pesan "Akses ditolak. Anda bukan admin toko.".



```
Pilih operasi yang ingin Anda lakukan:
1. Tambah Produk
2. Edit Produk
3. Keluar
Masukkan pilihan (1/2/3): 2
Apakah Anda admin? (ya/tidak): ya
Masukkan ID produk yang akan diubah: 245
Masukkan nama baru produk: Gula
Masukkan harga baru produk: 15000
Apakah Anda admin? (ya/tidak): ya
Produk dengan ID 245 berhasil diubah menjadi Gula dengan harga 15000.0.

Pilih operasi yang ingin Anda lakukan:
1. Tambah Produk
2. Edit Produk
3. Keluar
Masukkan pilihan (1/2/3): 1
Apakah Anda admin? (ya/tidak): ya
Masukkan nama produk: Garam
Masukkan harga produk: 7000
Apakah Anda admin? (ya/tidak): ya
Produk Garam berhasil ditambahkan dengan harga 7000.0.

Pilih operasi yang ingin Anda lakukan:
1. Tambah Produk
2. Edit Produk
3. Keluar
Masukkan pilihan (1/2/3): 1
Apakah Anda admin? (ya/tidak): tidak
Akses ditolak. Anda bukan admin toko.

Pilih operasi yang ingin Anda lakukan:
1. Tambah Produk
2. Edit Produk
3. Keluar
Masukkan pilihan (1/2/3): 3
Terima kasih telah menggunakan program kami.
```

Gambar 6. Hasil Kode

Gambar 6 menjelaskan bahwa pengguna memilih opsi "Edit Produk" dan menjawab "ya" pada pertanyaan "Apakah Anda admin?". Program akan menjalankan fungsi edit_produk dan menampilkan output "Produk dengan *ID* 245 berhasil diubah menjadi Gula dengan harga 15000.0.". Pengguna memilih opsi "Tambah Produk" dan menjawab "ya" pada pertanyaan "Apakah Anda admin?". Program akan menjalankan fungsi tambah_produk dan menampilkan output "Produk Garam berhasil ditambahkan dengan harga 7000.0.". Pengguna memilih opsi "Tambah Produk" dan menjawab "tidak" pada pertanyaan "Apakah Anda admin?". Program akan menampilkan pesan "Akses ditolak. Anda bukan admin toko.". Pengguna memilih opsi "Keluar". Program akan menampilkan pesan "Terima kasih telah menggunakan program kami." dan berhenti berjalan.

4. Kesimpulan

Dalam artikel kali ini dapat disimpulkan bahwa penerapan *closure* menggunakan *decorator* pada pengelolaan akses admin dalam suatu aplikasi dapat memberikan kemudahan dalam memastikan bahwa hanya admin yang dapat mengedit, menambahkan serta menghapus produk dalam aplikasi. Hal itu dilakukan agar keamanan dan kebersihan aplikasi dapat terjaga

dengan baik oleh admin. Implementasi *closure* dan *decorator* memberikan fleksibilitas dalam pengaturan kode tanpa perlu mengubah struktur dasar sumber kode.

Referensi

[1] F. Syaferi, "Memahami Decorators di Python," Universitas Mahakarya asia, 21 Oktober 2023. [Online]. Available: https://blog.unmaha.ac.id/memahami-decorators-di-python. [Accessed 2024].