LAPORAN PROYEK UAS: RANCANG BANGUN APLIKASI HABIT TRACKER BERBASIS PYTHON DENGAN PENDEKATAN OBJECT-ORIENTED PROGRAMMING DAN ANTARMUKA TKINTER

Mata Kuliah

Pemrograman Berorientasi Objek

Dosen Pengampu

M. Adamu Islam Mashuri, S.Tr.T., M.Tr.Kom.

Oleh

El Syifa Sawitri

24091397120

2024D

PROGRAM STUDI MANAJEMEN INFORMATIKA

FAKULTAS VOKASI

UNIVERSITAS NEGERI SURABAYA

2025

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan laporan Ujian Akhir Semester dengan judul "Laporan Proyek UAS: Rancang Bangun Aplikasi Habit Tracker Berbasis Python Dengan Pendekatan Object-Oriented Programming Dan Antarmuka Tkinter" dengan baik dan tepat waktu.

Laporan ini disusun sebagai salah satu syarat penilaian pada mata kuliah Pemrograman Berorientasi Objek (PBO). Melalui penyusunan laporan ini, penulis berupaya merancang merancang dan mengimplementasikan sebuah aplikasi Habit Tracker berbasis Python yang menerapkan konsep Object-Oriented Programming secara nyata, mulai dari pemodelan kelas, pengelolaan logika bisnis, hingga integrasi dengan antarmuka pengguna menggunakan Tkinter.

Ucapan terima kasih disampaikan kepada Bapak Dosen Pengampu mata kuliah Pemrograman Berorientasi Objek yang telah memberikan bimbingan selama proses penyusunan laporan ini.

Penulis menyadari bahwa laporan ini masih memiliki kekurangan. Oleh karena itu, saran dan masukan dari berbagai pihak sangat diharapkan untuk penyempurnaan di masa mendatang.

Semoga laporan ini dapat memberikan manfaat dan menambah wawasan bagi pembaca, khususnya dalam memahami penerapan konsep Object-Oriented Programming dalam pengembangan aplikasi perangkat lunak.

Surabaya, 1 Desember 2025

Penulis

DAFTAR ISI

BAB I
PENDAHULUAN

1.1. Latar Belakang

Kebiasaan (habit) merupakan aktivitas yang dilakukan secara berulang dan berperan penting dalam membentuk pola perilaku seseorang. Kebiasaan yang positif, seperti belajar secara rutin, berolahraga, atau menulis jurnal harian, dapat memberikan dampak signifikan terhadap peningkatan produktivitas, kedisiplinan, serta kualitas hidup secara keseluruhan. Namun demikian, membangun dan mempertahankan kebiasaan positif bukanlah hal yang mudah, karena sering kali seseorang mengalami penurunan motivasi atau lupa untuk menjalankan kebiasaan tersebut secara konsisten.

Perkembangan teknologi informasi memberikan peluang untuk mengatasi permasalahan tersebut melalui pemanfaatan perangkat lunak. Salah satu solusi yang dapat digunakan adalah aplikasi Habit Tracker, yaitu aplikasi yang berfungsi untuk membantu pengguna dalam mencatat, memantau, dan mengevaluasi kebiasaan yang dilakukan setiap hari. Dengan adanya sistem pencatatan yang terstruktur, pengguna dapat melihat perkembangan kebiasaan mereka dalam jangka waktu tertentu, sehingga dapat meningkatkan kesadaran dan motivasi untuk tetap konsisten.

Dalam konteks pengembangan perangkat lunak, penerapan Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) menjadi sangat relevan. OOP merupakan paradigma pemrograman yang menekankan pada pemodelan objek yang merepresentasikan entitas nyata beserta perilaku dan atributnya. Dengan pendekatan OOP, sebuah aplikasi dapat dibangun secara modular, terstruktur, serta mudah dikembangkan dan dipelihara. Hal ini sangat penting, terutama dalam pengembangan aplikasi yang memiliki berbagai fitur dan logika bisnis yang kompleks, seperti perhitungan streak, freeze, dan rekap data berkala.

Bahasa pemrograman Python dipilih dalam proyek ini karena memiliki sintaks yang sederhana, mudah dipahami, serta mendukung paradigma OOP secara penuh. Selain itu, Python menyediakan pustaka standar Tkinter yang dapat digunakan untuk membangun antarmuka grafis (GUI) tanpa memerlukan framework tambahan. Dengan menggunakan Tkinter, aplikasi Habit Tracker dapat dijalankan sebagai aplikasi desktop yang interaktif dan mudah digunakan oleh pengguna.

Berdasarkan latar belakang tersebut, proyek Ujian Akhir Semester ini bertujuan untuk merancang dan membangun sebuah aplikasi Habit Tracker berbasis Python dengan menerapkan konsep Object-Oriented Programming secara nyata. Aplikasi ini diharapkan tidak hanya berfungsi sebagai alat bantu pengelolaan kebiasaan, tetapi juga sebagai media pembelajaran dalam memahami penerapan OOP dan arsitektur perangkat lunak yang baik.

1.2. Rumusan Masalah

Berdasarkan latar belakang tersebut, maka rumusan masalah dalam proyek ini adalah sebagai berikut:

1. Bagaimana menerapkan konsep Object-Oriented Programming dalam pengembangan aplikasi Habit Tracker?

2. Bagaimana merancang arsitektur aplikasi yang memisahkan antara logika bisnis, antarmuka pengguna, dan penyimpanan data?
3. Bagaimana membangun aplikasi Habit Tracker dengan fitur streak, freeze, dan rekap mingguan menggunakan Python?

1.3. Tujuan Proyek

Tujuan dari pembuatan proyek ini adalah:

1. Menerapkan konsep OOP secara nyata dalam pengembangan aplikasi.
2. Mengembangkan aplikasi Habit Tracker berbasis Python dengan antarmuka grafis.
3. Mengimplementasikan fitur manajemen habit, checklist harian, streak, dan laporan mingguan.
4. Melatih kemampuan perancangan sistem perangkat lunak yang terstruktur

1.4. Manfaat Proyek

Manfaat dari proyek ini antara lain:

1. Sebagai media pembelajaran penerapan OOP dalam pengembangan aplikasi.
2. Memberikan contoh implementasi arsitektur perangkat lunak yang baik.
3. Membantu pengguna dalam mengelola kebiasaan harian secara konsisten

1.5. Batasan Masalah

Agar pembahasan lebih terfokus, maka batasan masalah dalam proyek ini adalah:

1. Aplikasi berbasis desktop menggunakan Python dan Tkinter.
2. Penyimpanan data menggunakan file JSON.
3. Tidak menggunakan database atau framework tambahan.
4. Aplikasi ditujukan untuk penggunaan personal (single user).

BAB II
LANDASAN TEORI

2.1. Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) merupakan paradigma pemrograman yang berfokus pada penggunaan objek sebagai komponen utama dalam pengembangan perangkat lunak. Objek merupakan representasi dari entitas dunia nyata yang memiliki atribut (data) dan metode (perilaku). OOP bertujuan untuk meningkatkan keteraturan kode, kemudahan pemeliharaan, serta fleksibilitas dalam pengembangan sistem.

Konsep utama dalam OOP meliputi encapsulation, inheritance, polymorphism, dan abstraction. Encapsulation digunakan untuk membatasi akses langsung terhadap data internal objek, sehingga data hanya dapat dimodifikasi melalui metode tertentu. Inheritance memungkinkan sebuah kelas untuk mewarisi atribut dan metode dari kelas lain, sehingga dapat mengurangi duplikasi kode. Polymorphism memungkinkan metode yang sama memiliki perilaku berbeda tergantung pada objek yang memanggilnya. Sementara itu, abstraction digunakan untuk menyederhanakan kompleksitas sistem dengan menampilkan hanya fitur yang relevan kepada pengguna.

Dalam aplikasi Habit Tracker ini, konsep OOP diterapkan mulai dari pemodelan entitas habit, pengelolaan logika streak dan freeze, hingga pemisahan antara logika bisnis dan antarmuka pengguna.

2.2. Tkinter

Tkinter merupakan pustaka standar Python yang digunakan untuk membangun antarmuka grafis (Graphical User Interface/GUI). Tkinter menyediakan berbagai komponen antarmuka seperti tombol (button), label, kotak input, tabel, dan dialog, yang dapat digunakan untuk membangun aplikasi desktop yang interaktif.

Keunggulan Tkinter terletak pada kemudahannya dalam penggunaan serta integrasinya langsung dengan Python tanpa perlu instalasi pustaka tambahan. Dalam proyek ini, Tkinter digunakan untuk membangun antarmuka aplikasi Habit Tracker yang terdiri dari panel checklist, manajemen habit, serta ringkasan progres mingguan. Dengan antarmuka grafis ini, pengguna dapat berinteraksi dengan aplikasi secara lebih intuitif dibandingkan aplikasi berbasis teks.

2.3. JSON sebagai Media Penyimpanan Data

JSON (JavaScript Object Notation) merupakan format pertukaran data yang ringan, mudah dibaca oleh manusia, dan mudah diproses oleh mesin. JSON sering digunakan sebagai media penyimpanan data sederhana karena strukturnya yang fleksibel dan berbasis pasangan key-value.

Dalam aplikasi Habit Tracker ini, JSON digunakan sebagai media penyimpanan data sementara untuk menyimpan informasi habit, seperti nama habit, tanggal penyelesaian, dan data freeze. Penggunaan JSON memungkinkan aplikasi untuk menyimpan data tanpa menggunakan sistem

database yang kompleks. Selain itu, struktur JSON yang sederhana memudahkan proses pengembangan serta memungkinkan migrasi ke database di masa mendatang tanpa mengubah logika utama aplikasi.

2.4. Konsep Habit Tracker

Habit Tracker adalah sebuah sistem yang digunakan untuk membantu pengguna mencatat dan memantau kebiasaan yang dilakukan secara berkala. Konsep utama dalam Habit Tracker meliputi pencatatan aktivitas harian, perhitungan konsistensi (streak), serta evaluasi progres dalam periode tertentu, seperti mingguan atau bulanan.

Dalam aplikasi ini, konsep Habit Tracker dikembangkan lebih lanjut dengan menambahkan fitur streak freeze, yaitu mekanisme yang memungkinkan streak tetap terjaga meskipun pengguna melewatkan satu hari, dengan batasan tertentu. Fitur ini dirancang untuk mendukung aspek psikologis pengguna agar tidak kehilangan motivasi hanya karena satu kali kelalaian. Selain itu, aplikasi juga menyediakan badge sebagai bentuk penghargaan visual untuk meningkatkan motivasi pengguna dalam menjaga konsistensi kebiasaan.

BAB III
PERANCANGAN SISTEM

3.1. Analisis Kebutuhan Sistem

Analisis kebutuhan sistem dilakukan untuk menentukan fungsi-fungsi utama yang harus dimiliki oleh aplikasi Habit Tracker agar dapat berjalan sesuai dengan tujuan yang diharapkan. Kebutuhan sistem dibagi menjadi kebutuhan fungsional dan kebutuhan non-fungsional.

3.1.1. Kebutuhan Fungsional

Kebutuhan fungsional dalam aplikasi ini meliputi:

1. Sistem dapat menambahkan habit baru dengan nama yang ditentukan pengguna.
2. Sistem dapat mengedit dan menghapus habit yang telah dibuat.
3. Sistem dapat menampilkan checklist habit berdasarkan tanggal tertentu.
4. Sistem dapat mencatat penyelesaian habit harian.
5. Sistem dapat menghitung streak harian secara otomatis.
6. Sistem menyediakan fitur streak freeze untuk menjaga streak ketika satu hari terlewat.
7. Sistem menampilkan ringkasan progres mingguan dan badge motivasi.
8. Sistem dapat mengekspor data progres mingguan ke dalam format CSV

3.1.2. Kebutuhan Non-Fungsional

Kebutuhan non-fungsional aplikasi meliputi:

1. Aplikasi mudah digunakan (user-friendly).
2. Aplikasi berjalan secara responsif pada sistem desktop.
3. Struktur kode mudah dipahami dan dikembangkan.
4. Aplikasi tidak bergantung pada database eksternal.

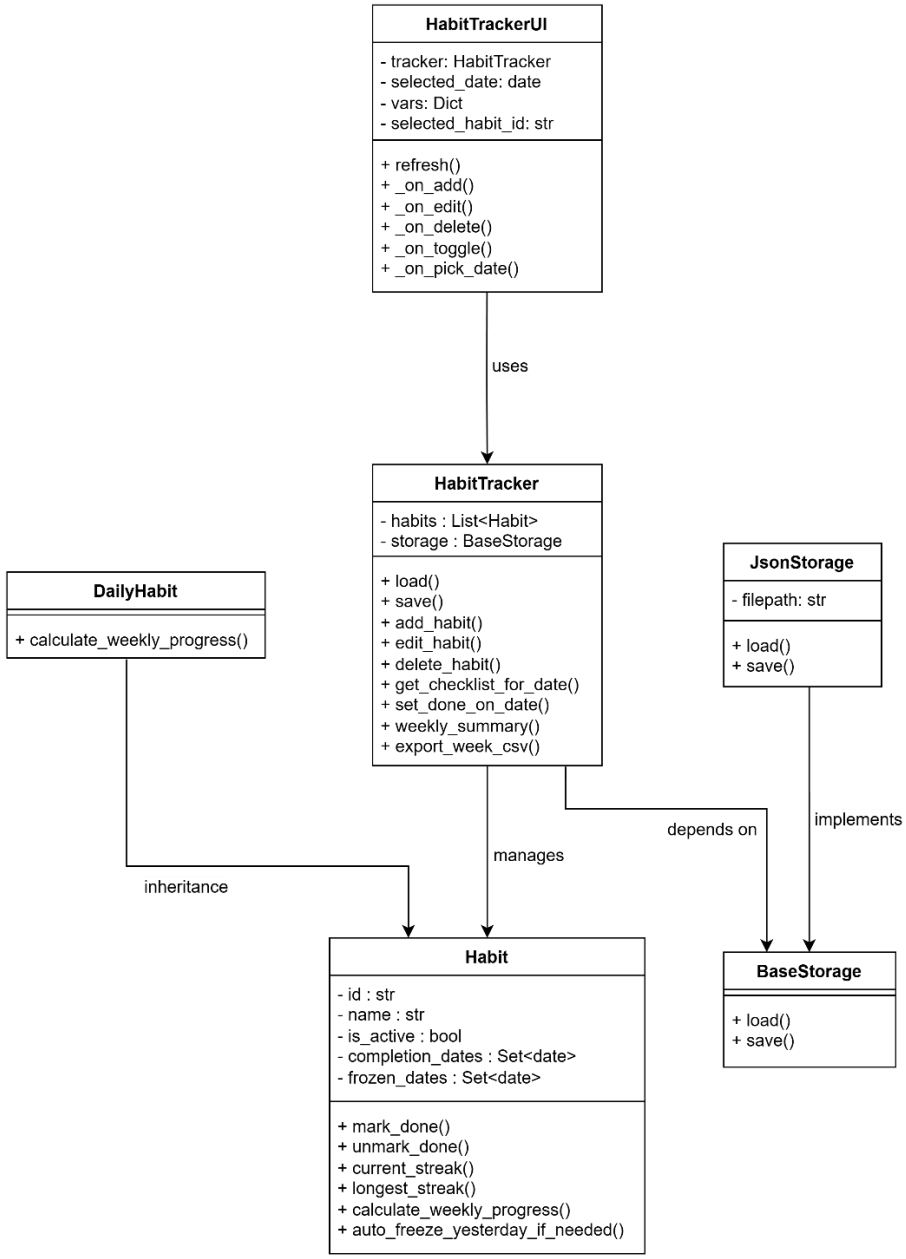
3.2. Arsitektur Sistem

Aplikasi Habit Tracker dirancang menggunakan arsitektur berlapis (layered architecture) yang memisahkan tanggung jawab setiap komponen. Arsitektur ini terdiri dari:

1. Presentation Layer (UI Layer)
   Bertugas menampilkan antarmuka pengguna dan menerima input dari pengguna.
2. Application Service Layer
   Bertugas mengatur alur logika aplikasi dan menjadi penghubung antara UI dan domain model.
3. Domain Model Layer
   Bertugas menangani seluruh aturan bisnis seperti streak, freeze, dan progres mingguan.
4. Persistence Layer
   Bertugas menyimpan dan memuat data dari media penyimpanan.

Pendekatan ini bertujuan untuk menjaga keterpisahan logika, sehingga perubahan pada satu layer tidak berdampak besar pada layer lainnya.

3.3. Diagram Class



Class diagram pada aplikasi Habit Tracker berbasis Python Tkinter dirancang untuk merepresentasikan struktur objek, relasi antar kelas, serta pemisahan tanggung jawab di dalam sistem. Diagram diatas menunjukkan bahwa aplikasi tidak dibangun secara prosedural, melainkan menggunakan pendekatan Object-Oriented Programming (OOP) dengan pemisahan yang jelas antara antarmuka pengguna, logika aplikasi, model domain, dan penyimpanan data.

Struktur class diagram secara umum terbagi menjadi empat lapisan utama:

1. UI Layer
2. Application Service Layer
3. Domain Model Layer
4. Persistence Layer

Pembagian ini ialah untuk menjaga keteraturan kode, kemudahan pengembangan, serta kesesuaian dengan prinsip rekayasa perangkat lunak.

1. Class HabitTrackerUI (UI Layer)

   Masuk ke analisis diagram diatas, mulai dari class HabitTrackerUI yangmerupakan representasi dari lapisan antarmuka pengguna yang dibangun menggunakan Tkinter, dengan mewarisi class ttk.Frame.

   Pewarisan ini membuat class tersebut berfungsi sebagai satu komponen UI yang utuh dan terintegrasi langsung dengan event loop Tkinter.

   Secara konsep, HabitTrackerUI berperan sebagai:

   - Penghubung antara pengguna dan sistem
   - Pengirim perintah ke lapisan logika aplikasi
   - Penampil hasil pemrosesan data

   Atribut seperti tracker, selected date, dan vars menunjukkan penerapan encapsulation, di mana state UI disimpan secara internal dan tidak diakses oleh class lain. UI hanya berinteraksi dengan sistem melalui method publik yang disediakan oleh class HabitTracker. Relasi antara HabitTrackerUI dan HabitTracker bersifat asosiasi satu arah, di mana UI bergantung pada tracker sebagai penyedia layanan logika, tetapi tracker tidak mengetahui keberadaan UI. Ini penting untuk menjaga independensi logika aplikasi dari teknologi antarmuka yang digunakan.

2. Class HabitTracker (Application Service Layer)

   Class HabitTracker berfungsi sebagai pusat orkestrasi logika aplikasi. Class ini menjadi satu-satunya pintu masuk UI untuk berinteraksi dengan domain model dan data persistence.

   Pada class diagram, HabitTracker memiliki relasi:

   - Asosiasi dengan Habit
   - Dependen**cy** terhadap BaseStorage

   Atribut habits yang bersifat private menunjukkan penerapan encapsulation, di mana daftar habit tidak dapat dimodifikasi langsung oleh UI. Seluruh manipulasi data harus melalui method seperti add_habit(), edit_habit(), dan set_done_on_date().

   Dalam konteks Tkinter, setiap event (klik tombol, checklist, pemilihan tanggal) akan memanggil method tertentu di HabitTracker, sehingga class ini berperan sebagai pengendali logika berbasis event-driven architecture.

3. Class Habit dan DailyHabit (Domain Model Layer)

   Class Habit merupakan model domain inti yang menyimpan dan mengelola data serta aturan bisnis terkait habit, termasuk: Completion harian, Streak, Freeze, dan Progress mingguan.

   Atribut seperti id, name, dan completion dates menunjukkan pembagian akses antara private dan protected, yang menegaskan bahwa data domain tidak boleh diakses langsung oleh UI maupun storage.

   Class DailyHabit merupakan turunan dari Habit, yang memperluas fungsionalitas dengan menambahkan logika badge mingguan. Relasi ini merupakan contoh penerapan inheritance, di mana perilaku dasar diwariskan, lalu dimodifikasi atau diperluas pada class turunan.

   Polymorphism terlihat pada method calculate_weekly_progress(), di mana:

   - HabitTracker memanggil method ini tanpa mengetahui tipe konkret objek
   - Objek DailyHabit menjalankan implementasi versinya sendiri

Dalam sistem Tkinter, hasil polymorphism ini ditampilkan langsung di UI dalam bentuk badge mingguan, tanpa UI perlu memahami bagaimana badge tersebut dihitung.

4. Class BaseStorage dan JsonStorage (Persistence Layer)
   Class BaseStorage berperan sebagai abstraksi lapisan penyimpanan data, yang mendefinisikan kontrak method load() dan save(). Class ini tidak memiliki implementasi konkret, melainkan menjadi acuan bagi class turunan.
   Class JsonStorage merupakan implementasi konkret dari BaseStorage yang menggunakan file JSON sebagai media penyimpanan. Relasi ini merupakan bentuk inheritance dan polymorphism, di mana sistem hanya bergantung pada interface BaseStorage, bukan pada implementasi spesifik.
   Dalam konteks aplikasi Tkinter, persistence layer ini bekerja secara transparan:
   - Pengguna tidak menyadari proses load dan save
   - Data tetap konsisten antar sesi penggunaan aplikasi

Hubungan antara class diagram dan sistem Tkinter dapat dirangkum sebagai berikut:

- Tkinter digunakan hanya pada UI Layer
- Event GUI diterjemahkan menjadi pemanggilan method pada HabitTracker
- HabitTracker mengatur alur logika dan komunikasi antar objek domain
- Domain model (Habit, DailyHabit) menangani seluruh aturan bisnis
- Storage bekerja di belakang layar tanpa interaksi langsung dengan UI

Pendekatan ini memastikan bahwa perubahan pada antarmuka (misalnya migrasi dari Tkinter ke web) tidak memerlukan perubahan pada domain logic maupun storage.

3.4. Perancangan Struktur Data dan Antarmuka Pengguna

Struktur data dirancang agar sederhana namun fleksibel. Data habit disimpan dalam bentuk file JSON yang berisi daftar habit, di mana setiap habit memiliki atribut:

- ID
- Nama Habit
- Tanggal Pembuatan
- Status Aktif
- Daftar Tanggal Penyelesaian
- Daftar Tanggal Freeze

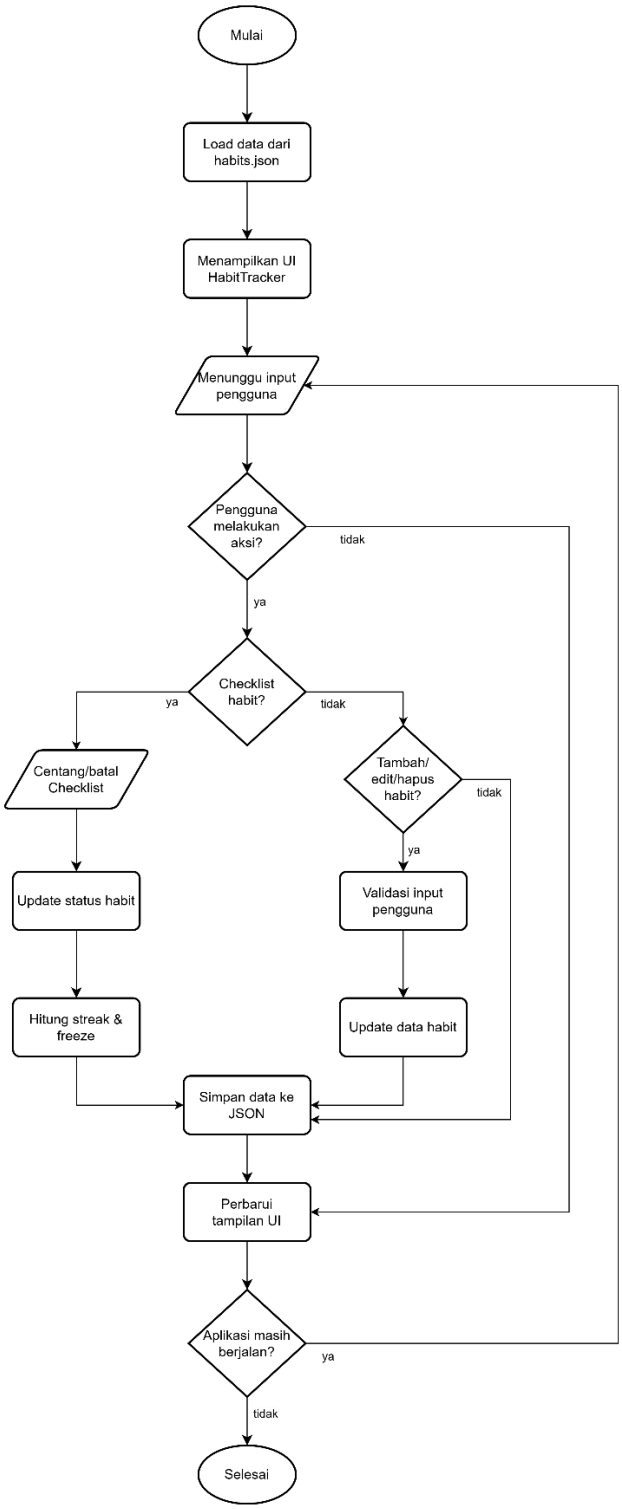
Antarmuka pengguna dirancang dengan konsep dua panel utama, yaitu:

- Panel kiri untuk checklist harian dan pengelolaan habit
- Panel kanan untuk ringkasan mingguan, streak, badge, dan statistik

Dibagi menjadi 2 panel agar memberikan pengalaman pengguna yang sederhana dan mudah digunakan.

3.5. Alur Aplikasi

Flowchart alur aplikasi menggambarkan proses kerja sistem Habit Tracker secara terstruktur dan sistematis sesuai dengan kaidah flowchart pada umumnya. Dibawah ini diagram flowchart sistem Habit Tracker:



Proses dimulai dengan inisialisasi aplikasi, di mana sistem memuat data habit dari media penyimpanan menggunakan lapisan persistence. Setelah data berhasil dimuat, sistem menampilkan antarmuka pengguna berbasis Tkinter dan menunggu interaksi dari pengguna. Setiap aksi yang dilakukan pengguna, seperti mencentang checklist harian atau melakukan manajemen habit (tambah, edit, dan hapus), akan diproses melalui lapisan logika aplikasi sebelum disimpan kembali ke dalam file JSON. Proses ini berlangsung secara berulang selama aplikasi berjalan, sehingga setiap perubahan data selalu tersimpan dan tampilan antarmuka selalu diperbarui secara langsung/real-time.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Sistem

Aplikasi Habit Tracker diimplementasikan menggunakan bahasa pemrograman Python dengan paradigma Object-Oriented Programming. Struktur file dibagi menjadi beberapa modul untuk memisahkan tanggung jawab setiap bagian sistem.
Struktur file utama sistem adalah sebagai berikut:

- main.py
- ui.py
- tracker.py
- habit.py
- storage.py
- habits.json

4.2. Analisis Implementasi Fitur Utama

Analisis implementasi pada bagian ini membahas secara rinci penerapan fitur-fitur utama dalam aplikasi Habit Tracker. Pembahasan mencakup analisis kode pada masing-masing file program, mekanisme penerapan fitur utama di dalam sistem, serta kesesuaian implementasi tersebut dengan perancangan yang telah dilakukan sebelumnya.
Fokus utama analisis meliputi penerapan konsep Object-Oriented Programming (OOP), yang terdiri dari:

1. Encapsulation, yaitu penggunaan atribut dan metode dengan pengaturan hak akses (private, protected, dan public) untuk menjaga integritas data dan membatasi akses langsung terhadap logika internal objek.
2. Inheritance, yaitu penerapan pewarisan antar kelas untuk memperluas fungsionalitas tanpa melakukan duplikasi kode.
3. Polymorphism, yaitu penggunaan metode dengan nama yang sama namun memiliki perilaku yang berbeda pada kelas turunan.

Selain itu, analisis juga mencakup penggunaan struktur data dan algoritma pendukung, seperti list dan set, yang dimanfaatkan untuk menyimpan dan mengelola data habit, tanggal penyelesaian, serta perhitungan streak dan progres mingguan sesuai dengan kebutuhan aplikasi.
Selanjutnya, dibahas pula implementasi interaksi pengguna yang intuitif melalui antarmuka grafis berbasis Tkinter, di mana pengguna dapat dengan mudah melakukan pengelolaan habit, mengisi checklist harian, serta melihat ringkasan progres dan statistik secara visual.

4.2.1. File main.py

File main.py berfungsi sebagai entry point/titik masuk dan composition root aplikasi. Di file ini, seluruh komponen utama aplikasi disusun dan dihubungkan, tanpa mengandung logika bisnis maupun logika antarmuka secara detail. Main.py juga bertanggung jawab untuk menentukan konfigurasi awal aplikasi, menginisialisasi storage, menginisialisasi controller

(HabitTracker), dan menjalankan antarmuka pengguna (Tkinter). Singkatnya, peran Main.py sebagai penghubung antar layer, bukan pelaksana logika aplikasi.

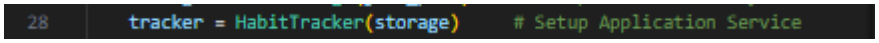
Alur kerja program pada main.py dapat dijelaskan sebagai berikut:

1. Menentukan lokasi file habits.json
2. Membuat objek JsonStorage
3. Membuat objek HabitTracker dengan dependency storage
4. Memuat data habit dari file JSON
5. Menginisialisasi window Tkinter
6. Menjalankan UI melalui HabitTrackerUI
7. Menjalankan event loop aplikasi

Alur kerja diatas, menunjukkan bahwa logika bisnis tidak bercampur dengan UI dan Storage tidak diakses langsung oleh UI.

Masuk ke inti Main.py, berikut analisis keseluruhan kodenya:

Bahasan ke-	Analisis
1.	`import os` `import tkinter as tk` `from storage import JsonStorage` `from tracker import HabitTracker` `from ui import HabitTrackerUI` Bagian awal file ini berisi import modul yang dibutuhkan untuk menjalankan aplikasi. Modul os digunakan untuk mengelola path file secara portabel, sehingga aplikasi dapat dijalankan di berbagai sistem operasi tanpa perubahan kode. tkinter diimpor untuk menyediakan root window bagi antarmuka pengguna. Import JsonStorage, HabitTracker, dan HabitTrackerUI menunjukkan dengan jelas bahwa main.py adalah satu-satunya tempat di mana seluruh layer sistem bertemu. UI tidak membuat storage, storage tidak membuat tracker, dan tracker tidak membuat UI. Semua hubungan antar-objek dibentuk di sini, sesuai prinsip desain OOP yang baik. `def main() -> None:` Fungsi main() didefinisikan sebagai titik awal eksekusi aplikasi. Docstring di dalamnya menjelaskan peran fungsi ini secara eksplisit, yaitu menentukan konfigurasi, menghubungkan seluruh layer, dan memulai event loop UI. Pernyataan ini penting karena menegaskan bahwa main() tidak boleh berisi logika domain atau manipulasi data.
2.	`# Tentukan lokasi file data` `base_dir = os.path.dirname(os.path.abspath(__file__))` `json_path = os.path.join(base_dir, "habits.json")` Bagian ini bertujuan untuk menentukan lokasi absolut file habits.json, yaitu file penyimpanan data habit. • os.path.abspath() memastikan path bersifat absolut • os.path.join() menjaga kompatibilitas lintas sistem operasi • file digunakan untuk mendapatkan lokasi file main.py Poin poin diatas ini membuat aplikasi tidak bergantung pada working directory, sehingga lebih stabil saat dijalankan dari berbagai lingkungan.
3.	`storage = JsonStorage(json_path)  # Setup Persistence Layer`

	Pada tahap ini, aplikasi menginisialisasi layer penyimpanan data menggunakan class JsonStorage. • JsonStorage merupakan implementasi dari BaseStorage • main.py secara eksplisit memilih jenis storage yang digunakan • Layer lain (Tracker dan UI) tidak mengetahui detail penyimpanan Pendekatan ini menerapkan prinsip Dependency Inversion, di mana aplikasi bergantung pada abstraksi, bukan implementasi konkret.
4.	 Class HabitTracker berperan sebagai application service atau business logic orchestrator. • HabitTracker menerima storage melalui dependency injection • Tracker menjadi satu-satunya pintu masuk UI ke domain • UI tidak berinteraksi langsung dengan Habit atau JsonStorage Ini menjaga pemisahan tanggung jawab antara: UI (presentasi), Logic aplikasi, dan Penyimpanan data.
5.	 Pemanggilan method load() bertujuan untuk: • mengambil data mentah dari storage, • mengonversinya menjadi object domain (Habit), • dan menyiapkan state awal aplikasi. Dari poin diatas, dapat dijabarkan abhwa: • Storage hanya mengembalikan data berbentuk dictionary • Proses konversi data → objek dilakukan oleh domain (Habit.from_dict) • main.py tidak terlibat dalam proses parsing atau validasi data
6.	 Bagian ini bertanggung jawab untuk: membuat window utama Tkinter (root) dan menghubungkan UI dengan application service (HabitTracker). Dari kode diatas dapat dijabarkan bahwa: • UI menerima tracker sebagai dependency • UI tidak mengetahui keberadaan storage • Semua aksi UI diteruskan ke tracker melalui method resmi Pendekatan ini memastikan UI hanya berfungsi sebagai presentation layer.
7.	 Pemanggilan mainloop() digunakan untuk: • menjalankan event loop Tkinter • menjaga aplikasi tetap aktif • merespons input pengguna Seluruh interaksi pengguna (klik, input, checklist) akan diproses melalui mekanisme ini
8.	 Bagian ini merupakan entry guard yang bertujuan untuk: • mencegah eksekusi otomatis saat file di-*import* • memastikan main() hanya dijalankan saat file dieksekusi langsung Ini merupakan praktik standar Python yang baik dan profesional.

4.2.2. File ui.py

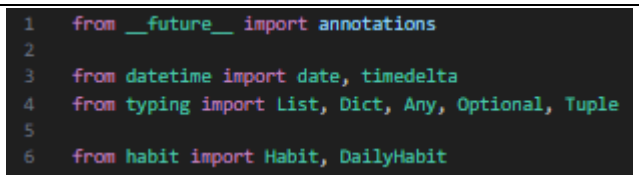
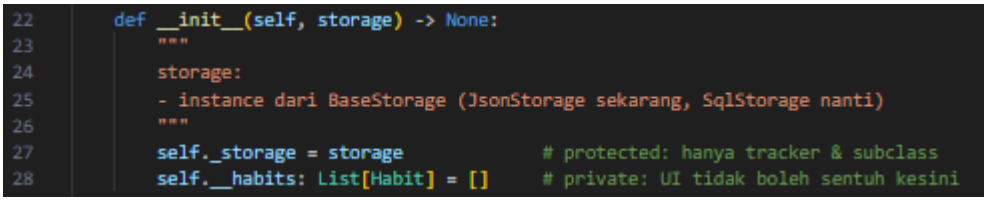
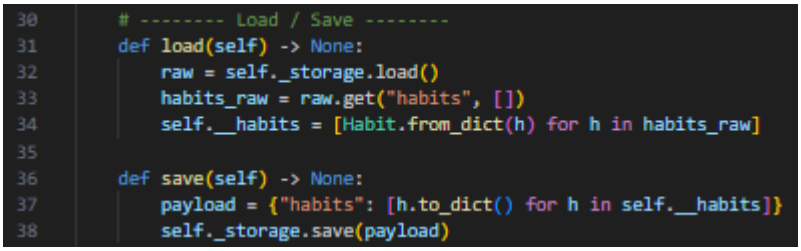
File ui.py berfungsi sebagai lapisan antarmuka pengguna (User Interface Layer) dalam aplikasi Habit Tracker. File ini dirancang dengan prinsip separation of concerns, di mana UI dipisahkan secara tegas dari logika bisnis dan pengolahan data. Jadi lebih jelasnya, UI tidak menyimpan atau memproses data mentah, melainkan hanya memanggil metode yang telah disediakan oleh lapisan logika aplikasi (business logic). Tak lupa, bahwa file ini ertanggung jawab untuk: menampilkan data kepada pengguna, menerima input dari pengguna, dan meneruskan input tersebut ke HabitTracker sebagai business logic handler.

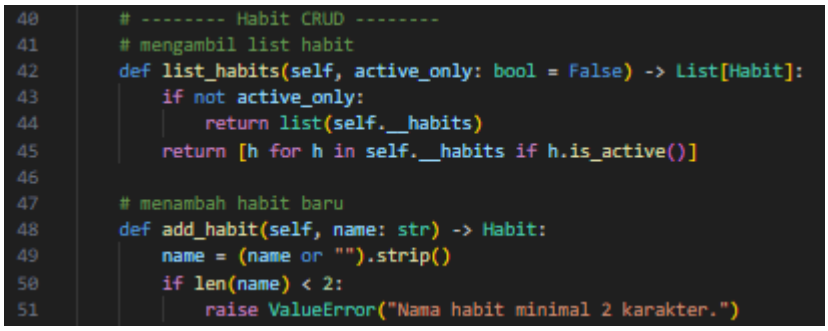
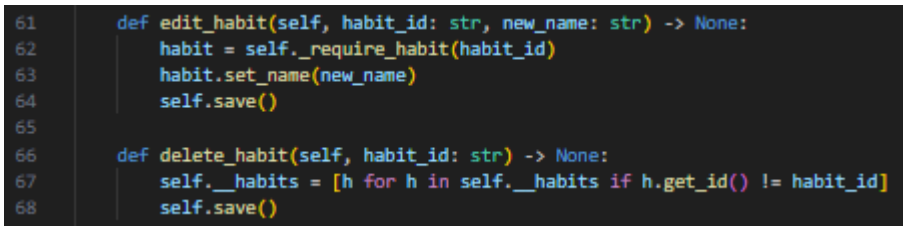
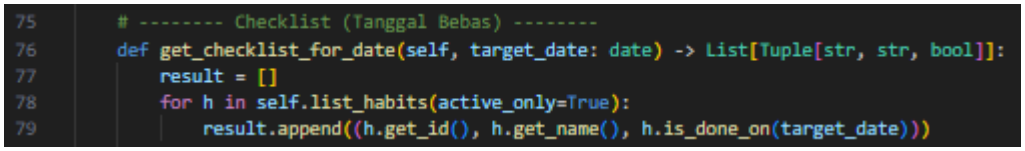
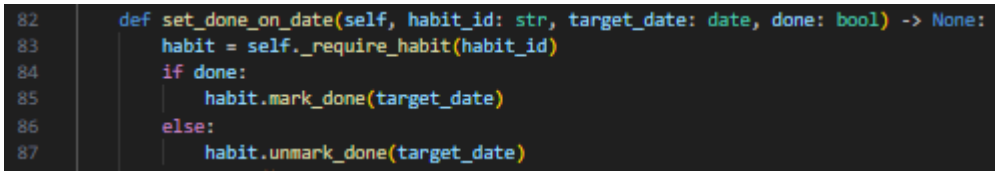
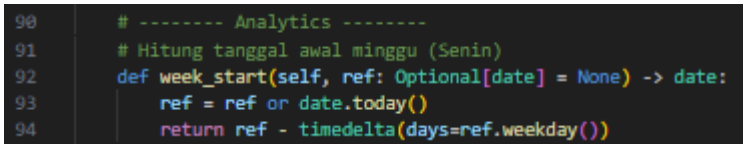
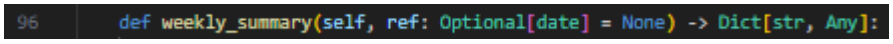
Bahasan ke-	Analisis
1.	```1 from __future__ import annotations 2 3 import tkinter as tk 4 from tkinter import ttk, messagebox, filedialog, simpledialog 5 from typing import Dict 6 from datetime import date 7 8 from tracker import HabitTracker``` Awalan ui.py diisi dengan import library dan dependency yang dibutuhkan oleh antarmuka pengguna • tkinter dan ttk digunakan untuk membangun GUI aplikasi. • messagebox, filedialog, dan simpledialog digunakan untuk interaksi pengguna berbasis dialog. • Dict dari modul typing digunakan untuk type hinting, yang meningkatkan keterbacaan dan keamanan kode. • date digunakan untuk mengelola tanggal checklist habit. • HabitTracker diimpor sebagai lapisan logika bisnis, menandakan bahwa UI tidak menangani data secara langsung.
2.	```11 class HabitTrackerUI(ttk.Frame):``` Class utama dalam ui.py ini adalah class HabitTrackerUI, yang juga merupakan turunan dari ttk.Frame, yang menunjukkan bahwa: • UI dibangun menggunakan inheritance dari komponen Tkinter • Aplikasi memanfaatkan kembali komponen GUI bawaan Python Inheritance ini membuat class UI dapat: menggunakan sistem layout Tkinter dan terintegrasi langsung ke event loop GUI.
3.	```20 def __init__(self, master: tk.Tk, tracker: HabitTracker) -> None: 21 super().__init__(master, padding=12)``` Bagian ini adalah Konstruktor, yang menerima 2 dependency: master dari root tkinter, dan tacker dari objek HabitTracker. Dengan ini, UI tidak membuat tracker sendiri, melainkan menerimanya dari luar (dari main.py). Pemanggilan super() menandakan bahwa konstruktor kelas induk (Frame) tetap dijalankan.
4.	```23 self._tracker = tracker 24 self._vars: Dict[str, tk.BooleanVar] = {} 25 self._selected_date = date.today()``` Pada atribut diatas, terlihat bahwa masing maisng atribut menggunakan awalan uderscore _, yang menandakan protected attribute. Berikut implementasi OOP Encapsulationnya:

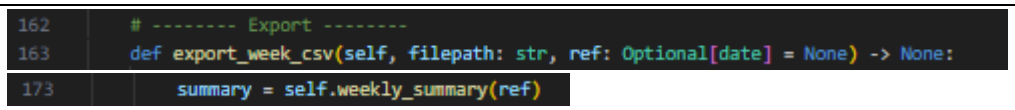
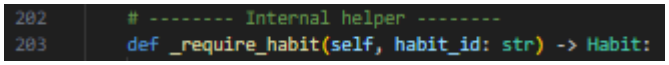
	• _tracker untuk referensi ke bussiness logic • _vars untuk menyimpan state checkbox • _selected_date untuk tanggal aktif
5.	```27 # selection state (MUST exist before refresh) 28 self._selected_habit_id: str | None = None 29 self._habit_id_map: list[str] = []``` Dua atribut ini digunakan untuk mengelola seleksi habit di sisi UI. _habit_id_map berfungsi sebagai jembatan antara indeks visual pada Listbox dan ID habit sebenarnya. Pendekatan ini penting karena UI tidak boleh bergantung pada nama habit sebagai identitas, melainkan tetap menggunakan ID domain. Encapsulation di sini memastikan bahwa detail seleksi hanya diketahui oleh UI.
6.	```31 self._build_layout() 32 self.refresh()``` Pemanggilan _build_layout() membangun struktur visual aplikasi, sedangkan refresh() langsung menyinkronkan UI dengan data dari tracker. Pemisahan ini memperlihatkan desain yang rapi: membangun tampilan dan mengisi data dilakukan oleh dua tahap berbeda.
7.	```34 # ---------- UI build ---------- 35 def _build_layout(self) -> None:``` Metode ini sepenuhnya berisi kode untuk menyusun tampilan: pengaturan window, panel kiri dan kanan, label, tombol, listbox, checklist, dan treeview. Tidak ada satu pun logika domain di sini. Sebagai contoh, permilihan habit diatur melalui event berikut: ```60 self._habit_listbox.bind("<<ListboxSelect>>", self._on_select_habit)``` Di sini terlihat bahwa UI hanya **menangkap aksi pengguna**, lalu meneruskannya ke handler internal.
8.	```79 ttk.Button(btns, text="Tambah", command=self._on_add).grid(row=0, column=0, padx=3) 80 ttk.Button(btns, text="Edit", command=self._on_edit).grid(row=0, column=1, padx=3) 81 ttk.Button(btns, text="Hapus", command=self._on_delete).grid(row=0, column=2, padx=3) 82 ttk.Button(btns, text="Pilih Tanggal", command=self._on_pick_date).grid(row=0, column=3, padx=3) 83 ttk.Button(btns, text="Hari Ini", command=self._on_today).grid(row=0, column=4, padx=3)``` Tiap button/tombol diatas memiliki masing masing 1 tanggung jawab dan akan langsung memanggil metode handler. Interaksi intuitif button-button diatas, saat sistem dijalankan, pengguna langsung memahami fungsi tiap buttonnya.

9.	```130 # ---------- Refresh ---------- 131 def refresh(self) -> None:``` Metode refresh diatas ini adalah pusat dari sinkronisasi UI. ```146 for h in self._tracker.list_habits(active_only=True): 147 self._habit_listbox.insert(tk.END, h.get_name()) 148 self._habit_id_map.append(h.get_id())``` UI hanya mengambil nama dan ID, tanpa menyentuh struktur internal habit. Ini menunjukkan kepatuhan UI terhadap batas domain. ```153 self._vars.clear() 154 155 for i, (hid, name, done) in enumerate(self._tracker.get_checklist_for_date(self._selected_date)): 156 var = tk.BooleanVar(value=done) 157 self._vars[hid] = var``` Penggunakan struktur data (dictionary) diatas ini ialah untuk memetakan ID habit ke status checkbox. Dirancang seperti diatas agar mempermudah pembacaan dan update data. ```182 self._tree.insert(183 "", 184 "end", 185 iid=h["id"], 186 values=(``` Treeview digunakan sebagai struktur data tabel untuk menampilkan ringkasan mingguan. Dan walaupun bukan tree algoritmik murni, Treeview diatas ini sesuai untuk data hierarkis/tabular.
10.	```167 # ----- weekly summary ----- 168 summary = self._tracker.weekly_summary()``` UI memanggil weekly_summary() dan menerima data yang sudah siap ditampilkan. Treeview hanya bertugas menampilkan hasil, termasuk badge, freeze, dan streak. ```198 self._tree.tag_configure(tag, foreground=self._streak_color(h["current_streak"]))``` Pemilihan warna streak adalah contoh logika presentasi murni. Ini sengaja diletakkan di UI karena tidak memengaruhi perhitungan streak itu sendiri.
11.	```209 # ---------- Actions ---------- 210 def _on_toggle(self, habit_id: str) -> None: 211 self._tracker.set_done_on_date(habit_id, 212 self._selected_date, 213 bool(self._vars[habit_id].get())) 214 self.refresh()``` Setiap handler memiliki pola yang konsisten: UI mengirim perintah ke tracker, lalu memanggil refresh() untuk memperbarui tampilan. UI tidak menyimpan hasil perhitungan; ia selalu meminta ulang dari tracker. Handler lain seperti _on_add, _on_edit, _on_delete, dan _on_export mengikuti prinsip yang sama. Dialog digunakan untuk input, tracker untuk logika, dan UI untuk tampilan.
12.	```252 # ---------- UI helper ---------- 253 def _streak_color(self, streak: int) -> str: 254 if streak >= 7: 255 return "#c9a227" 256 if streak >= 3: 257 return "#2e8b57" 258 if streak == 0: 259 return "#888888" 260 return "#000000"``` Method ini memetakan nilai streak ke warna teks. Ini adalah logika presentasi murni yang tidak memiliki makna bisnis. Dengan memisahkannya ke helper khusus, kode menjadi lebih terbaca dan mudah diubah tanpa memengaruhi domain.

4.2.3. File tracker.py

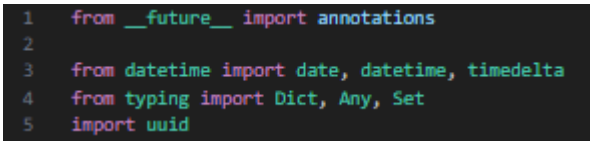
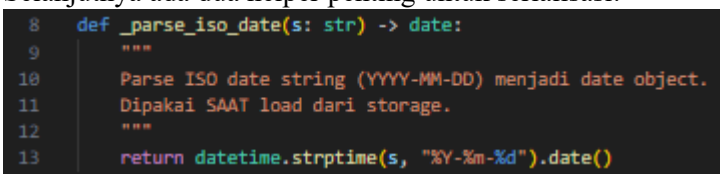
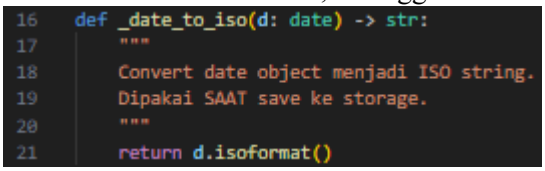
Bahasan ke-	Analisis
1.	 Bagian awal file tracker.py ini berisi import modul yang mendukung logika inti aplikasi. • date dan timedelta digunakan untuk perhitungan tanggal, streak, dan minggu aktif. • typing digunakan untuk type hinting (List, Dict, Tuple, dll.), yang meningkatkan keterbacaan dan kejelasan kontrak metode. • Habit dan DailyHabit merupakan domain entity, yaitu representasi objek habit beserta logikanya. Import Habit dan DailyHabit menunjukkan bahwa HabitTracker tidak menyimpan logika habit secara langsung, melainkan mengorkestrasi domain entity.
2.	 Kelas HabitTracker ini berperan sebagai pusat orkestrasi logika aplikasi. Tidak ada inheritance langsung di sini, karena kelas ini dirancang sebagai service layer / application layer, bukan turunan dari kelas lain.
3.	 Bagian ini adalah Konstruktor, yang didalamnya berisi encapsulation. Storage yang protected attribute dan habits yang private attribute.
4.	Kode dibawah ini adalah load dan save data (orkestrasi storage) Dimana pada load data: • Tracker memanggil storage • Data mentah diubah menjadi objek habit • List dipakai untuk menyimpan kumpulan objek habit Dan pada save data tracker bertanggung jawab penuh atas konversi objek dengan data mentah 

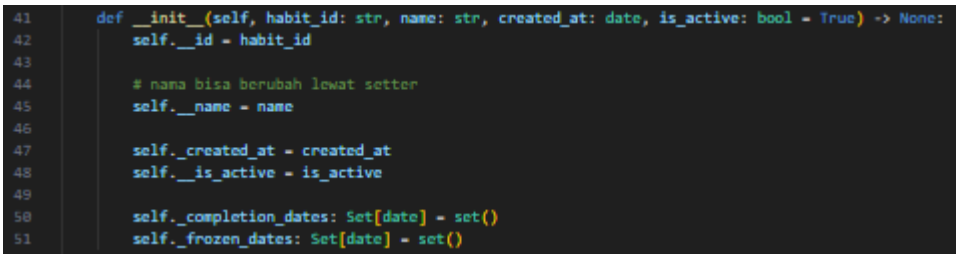
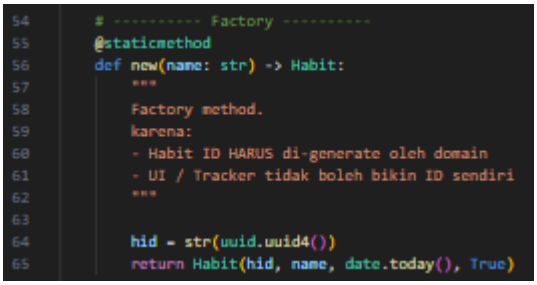
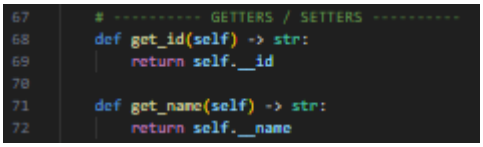
5.	 CRUD habit (manajemen data inti) Pada mengambil daftar habit, list digunakan untuk menyimpan dan memfilter objek habit. Sedangkan pada menambah habit baru, ada implementasi OOP berupa polymorphism.
6.	 Pada edit dan habus habit diatas, diterapkan OOP berupa encapsulation. Jadi UI tidak bisa mengedit langsung dari objek habit.
7.	 Pada mengabil checklist dari checklist berdasarkan tanggal diatas, tuple digunakan untuk mnegirim data ringan ke UI dan UI tidak menerima objek Habit langsung.  Pada menandai habit selesai/tidak diatas, tracker menjaga agar perubahan state selalu diikuti save().
8.	 Pada perhitungan awal minggu diatas, diterapkan algoritma sederhana untuk menentukan awal minggu (Senin).  Pada weekly summary diatas ini ada dict untuk ringkasan dan list of dict untuk data per habitnya. Pada bagian ini berfungsi untuk mengatur urutan eksekusi logika, menghitung progress, streak, freeze, dan menentukan best habit.

9.	 Pada aplikasi HabitTracker, ada fitur export CSV, dimana CSV ini hanyalah representasi tampilan data, bukan logika baru.
10.	 Helper internal diatas ini menerapkan encapsulation (internal guard) yang mana digunakan untuk validasi internal dan method protected.

4.2.4. File habit.py

Pada sistem HabitTracker, file habit.py adalah inti dari domain layer: tempat definisi "habit" sebagai objek nyata dalam aplikasi beserta aturan-aturan perilakunya. Jika ui.py bertugas menampilkan dan menerima input, dan tracker.py bertugas mengorkestrasi alur serta penyimpanan, maka habit.py bertugas mendefinisikan apa itu habit dan bagaimana aturan streak, checklist, serta freeze bekerja. Dengan kata lain, file ini adalah "jantung logika" domain: UI dan storage boleh berubah, tetapi aturan domain di sini seharusnya tetap stabil.

Bahasan ke-	**Analisis**
1.	 Import di atas menunjukkan bahwa objek habit membutuhkan operasi tanggal yang cukup intensif (date, timedelta) karena seluruh konsep streak dan weekly progress bergantung pada urutan hari. Set dari typing penting karena penyimpanan tanggal checklist lebih tepat menggunakan himpunan (unik, cepat untuk pengecekan keanggotaan). uuid dipakai agar ID habit selalu unik dan tidak bergantung pada UI atau storage. Selanjutnya ada dua helper penting untuk serialisasi:  Fungsi ini menjadi jembatan saat load dari storage: storage menyimpan string, domain membutuhkan date. Penggunaan datetime.strptime memastikan format yang diterima konsisten YYYY-MM-DD, sehingga domain tidak menerima data tanggal sembarangan.  Sebaliknya, fungsi ini dipakai saat save: domain menghasilkan date, storage membutuhkan string. Dua helper ini memperkuat pemisahan tanggung jawab: domain mengontrol bagaimana tanggal dipahami dan diserialisasi, sementara storage hanya menyimpan hasilnya.

2.	`class Habit:` Deklarasi Class Habit ini menegaskan Habit sebagai entity domain, artinya objek yang punya identitas (id) dan perilaku (aturan done, freeze, streak). Poin pentingnya: kelas ini tidak mengetahui UI, tidak mengetahui storage, dan tidak mengetahui tracker. Inilah penerapan desain OOP yang benar: logic domain tidak boleh "bocor" ke UI atau penyimpanan. Di sistemmu, HabitTracker memanggil method-method di Habit, tetapi Habit tidak bergantung pada HabitTracker.
3.	`FREEZE_MAX_PER_WEEK = 1` Konstanta kelas ini adalah contoh domain rule yang eksplisit. Freeze token adalah aturan bisnis yang melekat pada definisi habit (bukan pada UI atau storage). Dengan menaruhnya di kelas domain, perubahan aturan freeze (misalnya dari 1 jadi 2) cukup dilakukan di satu tempat tanpa mengubah UI ataupun tracker.
4.	 Di sini terlihat penerapan encapsulation yang kuat. Atribut id, name, dan is active dibuat private (double underscore) agar tidak bisa diubah sembarangan dari luar kelas, karena id adalah identitas dan tidak boleh diganti, nama harus melewati validasi setter, dan status aktif memengaruhi boleh/tidaknya habit dicentang. Sementara itu created at, completion dates, dan frozen dates dibuat protected (single underscore) karena kelas turunan (DailyHabit) dan proses rebuild (from dict) butuh akses. Dua set _completion_dates dan _frozen_dates adalah penggunaan struktur data yang sangat sesuai: himpunan membuat operasi in dan add menjadi cepat dan menjamin tidak ada tanggal duplikat. Dalam sistem HabitTracker, dua set ini adalah "rekam jejak" utama: tracker dan UI tidak menyimpan histori detail checklist, mereka hanya meminta informasi dari entity ini.
5.	 Method new() adalah penerapan OOP yang penting karena menjadikan domain sebagai satu-satunya pihak yang berhak membuat Habit dengan ID valid. UI atau tracker tidak membuat ID sendiri, sehingga integritas identitas objek terjaga. Ini juga memudahkan scaling, misalnya jika nanti aturan pembuatan ID berubah, hanya domain yang diubah.
6.	 Getter ini memberi akses aman ke ID tanpa membuka peluang mengubahnya. Begitupun nama yang diberikan melalui getter.

	``python def set_name(self, new_name: str) -> None: new_name = (new_name or "").strip() if len(new_name) < 2: raise ValueError("Nama habit minimal 2 karakter.") self.__name = new_name ``` Setter ini adalah inti dari encapsulation: perubahan nama wajib melewati validasi minimal 2 karakter. Dengan begini, UI tidak perlu mengulang validasi yang sama; UI cukup menangkap error jika ada.
	``python def is_active(self) -> bool: return self.__is_active def set_active(self, active: bool) -> None: self.__is_active = bool(active) ``` Status aktif dibuat formal: domain mengubahnya dan menjamin bertipe boolean. Dalam sistem, fitur “habit non-aktif tidak bisa dicentang” nanti ditegakkan oleh method mark_done
7.	```python # ---------- Completion ---------- # tndai habit selesai di tanggal tertentu def mark_done(self, on_date: date) -> None: if not self.is_active(): raise ValueError("Habit non-aktif tidak bisa dicentang.") self._completion_dates.add(on_date) ``` Di sini terlihat fungsi domain entity yang paling nyata, dimana aturan bahwa habit non-aktif tidak boleh dicentang dijaga oleh domain. UI tidak perlu menebak atau memblokir; UI cukup memanggil tracker → tracker memanggil entity, dan entity yang menolak jika tidak valid.
	``python # Batalkan checklist pada tanggal tertentu def unmark_done(self, on_date: date) -> None: self._completion_dates.discard(on_date) ``` discard sengaja dipakai karena aman: jika tanggal tidak ada, tidak error. Ini membuat operasi “batal checklist” idempotent dan stabil.
	``python def is_done_on(self, d: date) -> bool: return d in self._completion_dates ``` Pengecekan membership pada set sangat efisien, mendukung UI ketika membangun checklist harian.
8.	```python # ---------- Freeze logic ---------- def is_frozen_on(self, d: date) -> bool: return d in self._frozen_dates ``` Ini adalah pasangan dari is_done_on, tapi untuk status freeze.
	``python # hitung sisa freeze token dalam minggu ref_date def freeze_remaining_for_week(self, ref: date) -> int: ws = self._week_start(ref) used = sum(1 for fd in self._frozen_dates if self._week_start(fd) == ws) return max(0, self.FREEZE_MAX_PER_WEEK - used) ``` Method ini mendefinisikan aturan penting: freeze dihitung per minggu. Cara menghitungnya cukup cerdas: ia mencari minggu awal dari ref, lalu menghitung berapa frozen_date yang berada pada minggu yang sama. Ini menjadikan freeze benar-benar “token mingguan”, bukan token global.

```python
    def try_freeze_date(self, missed_date: date) -> bool:
        """
        Coba apply freeze ke tanggal yang terlewat.
        Return:
        - True  → freeze berhasil / sudah ada
        - False → tidak ada token tersisa
        """
        # sudah frozen → aman, tidak consume token lagi
        if missed_date in self._frozen_dates:
            return True

        # cek token minggu tersebut
        if self.freeze_remaining_for_week(missed_date) > 0:
            self._frozen_dates.add(missed_date)
            return True

        return False
```

Method ini penting karena idempotent: jika tanggal sudah frozen, return True tanpa mengurangi token lagi. Ini membuat pemanggilan berulang aman. Jika token masih ada, tanggal ditambahkan ke frozen_dates. Jika token habis, ia menolak (False). Keputusan token dipegang domain, bukan UI atau tracker.

```python
    def auto_freeze_yesterday_if_needed(self, ref_date: date) -> bool:
        """
        Auto-freeze (streak psychology):
        Jika:
        - hari ini done
        - kemarin bolong
        - kemarin belum frozen
        Maka: coba freeze kemarin (jika token ada)

        Tracker yang memanggil method ini,
        bukan UI
        """
        yesterday = ref_date - timedelta(days=1)
        if (
            self.is_done_on(ref_date)
            and not self.is_done_on(yesterday)
            and not self.is_frozen_on(yesterday)
        ):
            return self.try_freeze_date(yesterday)
        return False
```

Method ini adalah contoh kuat bahwa domain mendefinisikan “psikologi streak”: jika hari ini done, kemarin bolong, maka kemarin dicoba dibekukan untuk menjaga streak (selama token tersedia). Tracker yang memanggil method ini untuk menjaga urutan eksekusi (freeze dulu baru hitung streak). UI tidak perlu tahu detailnya.

Dalam sistem habit tracker, auto-freeze ini membuat pengalaman pengguna terasa lebih “memotivasi” karena streak tidak langsung putus akibat lupa sehari, tetapi tetap dibatasi aturan token mingguan agar tidak “curang”.

9.

```python
    # ---------- Streak ----------
    def current_streak(self, ref_date: date) -> int:
        streak = 0
        d = ref_date

        while self.is_done_on(d) or self.is_frozen_on(d):
            streak += 1
            d -= timedelta(days=1)

        return streak
```

Algoritma current streak berjalan mundur dari ref_date sampai menemukan hari yang tidak done dan tidak frozen. Ini sederhana, namun sangat tepat untuk definisi streak harian. Karena membership set cepat, loop ini juga efisien.

```python
    def longest_streak(self) -> int:
        if not self._completion_dates and not self._frozen_dates:
            return 0

        all_days = sorted(self._completion_dates | self._frozen_dates)
        best = run = 1

        for i in range(1, len(all_days)):
            if all_days[i] == all_days[i - 1] + timedelta(days=1):
                run += 1
                best = max(best, run)
            else:
                run = 1

        return best
```

Longest streak menggabungkan completion dan frozen menjadi satu himpunan, lalu disortir. Setelah itu, ia menghitung panjang run berurutan (konsep seperti “consecutive days”). Ini adalah algoritma yang tepat karena longest streak membutuhkan histori total, bukan hanya sampai hari ini. Penggunaan union set (|) menguatkan bahwa frozen dianggap bagian dari streak.

Dalam sistem, HabitTracker.weekly_summary() memakai current_streak dan longest_streak untuk menampilkan statistik ke UI.

10.

```python
    # ---------- Weekly progress ----------
    def calculate_weekly_progress(self, week_start: date) -> Dict[str, Any]:
        days = [week_start + timedelta(days=i) for i in range(7)]
        done = sum(1 for d in days if self.is_done_on(d))
        rate = (done / 7) * 100
        return {
            "done_days": done,
            "target_days": 7,
            "completion_rate": round(rate, 2),
        }
```

Method ini menghitung progress mingguan generik. Catatan di docstring (“OVERRIDABLE”) adalah kunci: desain ini memang dibuat agar subclass dapat mengganti perilakunya. Di sinilah polymorphism disiapkan: tracker memanggil h.calculate_weekly_progress(...) tanpa peduli h itu Habit atau DailyHabit. Hasilnya bisa berbeda sesuai implementasi class masing-masing, tetapi interface method-nya tetap sama.

11.

```python
    # ---------- Serialization ----------
    def to_dict(self) -> Dict[str, Any]:
        return {
            "type": "Habit",
            "id": self.get_id(),
            "name": self.get_name(),
            "created_at": _date_to_iso(self._created_at),
            "is_active": self.is_active(),
            "completion_dates": sorted(_date_to_iso(d) for d in self._completion_dates),
            "frozen_dates": sorted(_date_to_iso(d) for d in self._frozen_dates),
        }
```

Serialization: Mengubah Objek Domain Menjadi Data Mentah

Method to_dict() membuat objek bisa disimpan oleh storage tanpa storage perlu memahami kelas Habit. Ia mengubah set tanggal menjadi list string yang terurut. Ini penting untuk konsistensi output dan memudahkan debugging.

	``python @staticmethod def from_dict(data: Dict[str, Any]) -> Habit: # Rebuild domain habit_type = data.get("type", "Habit") habit_id = data["id"] name = data["name"] created_at = _parse_iso_date(data["created_at"]) is_active = bool(data.get("is_active", True)) # polymorphism saat load if habit_type == "DailyHabit": habit = DailyHabit(habit_id, name, created_at, is_active) else: habit = Habit(habit_id, name, created_at, is_active) for s in data.get("completion_dates", []): habit._completion_dates.add(_parse_iso_date(s)) for s in data.get("frozen_dates", []): habit._frozen_dates.add(_parse_iso_date(s)) return habit ``` Di sini polymorphism benar-benar terjadi saat load. Berdasarkan field "type", domain membuat objek kelas yang sesuai. Ini membuat sistem extensible: jika nanti ada WeeklyHabit, kamu cukup menambahkan kondisi tipe tanpa mengubah UI sama sekali. Setelah objek dibuat, completion_dates dan frozen_dates diisi ulang ke set internal.
12.	```python # ---------- internal Helper ---------- def _week_start(self, d: date) -> date: return d - timedelta(days=d.weekday()) ``` Helper ini menjadi fondasi perhitungan mingguan (freeze token per minggu), yang diletakkan sebagai method internal karena hanya relevan untuk domain, tidak perlu diekspos ke UI.
13.	```python class DailyHabit(Habit): ``` Baris ini adalah implementasi inheritance: DailyHabit adalah turunan dari Habit. Dalam sistem, DailyHabit tetap sebuah habit (punya id, nama, streak, freeze), tetapi menambahkan perilaku khusus berupa “badge”. ```python def calculate_weekly_progress(self, week_start: date) -> Dict[str, Any]: base = super().calculate_weekly_progress(week_start) done = base["done_days"] ``` Di sini ia memanggil implementasi parent (super()), lalu memperkaya hasilnya. ```python # streak psychology via badge if done == 7: badge = "Perfect Week 🏆" elif done >= 5: badge = "Great Week ⭐" elif done >= 3: badge = "Good Momentum 👍" else: badge = "Keep Going 💪" base["badge"] = badge return base ``` Ini adalah polymorphism dalam bentuk paling jelas: method dengan nama yang sama (calculate_weekly_progress) menghasilkan output yang berbeda pada subclass. Tracker cukup memanggil method yang sama, dan UI mendapat tambahan field badge tanpa perlu tahu bahwa habit tersebut adalah DailyHabit. Dalam sistem HabitTracker, badge ini kemudian ditampilkan di UI pada treeview summary.

4.2.5. File storage.py

Dalam arsitektur aplikasi HabitTracker, file storage.py berperan sebagai lapisan persistence (penyimpanan data). Jika habit.py mendefinisikan aturan domain dan tracker.py mengorkestrasi alur logika, maka storage.py bertugas menyediakan mekanisme penyimpanan data yang dapat diganti tanpa memengaruhi UI maupun domain. File ini menjadi fondasi fleksibilitas sistem: hari ini data disimpan di JSON, besok bisa berpindah ke database atau API tanpa mengubah kode lain.

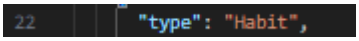
Bahasan ke-	Analisis
1.	`from __future__ import annotations` `from abc import ABC, abstractmethod` `from typing import Dict, Any` `import json` `import os` Bagian import menunjukkan bahwa file ini tidak berurusan dengan logika tanggal, streak, atau habit sama sekali. Fokusnya murni pada abstraction dan I/O. Modul abc digunakan untuk mendefinisikan kelas abstrak, yang menandakan bahwa storage memang dirancang sebagai kontrak, bukan implementasi tunggal. json dan os hanya dipakai oleh implementasi konkret (JsonStorage), bukan oleh kontraknya.
2.	`class BaseStorage(ABC):` Deklarasi ini memperkenalkan BaseStorage sebagai abstract base class. Dalam konteks sistem HabitTracker, kelas ini berfungsi sebagai jembatan kontrak antara application layer (HabitTracker) dan dunia luar (file, database, cloud).
3.	`@abstractmethod` `def load(self) -> Dict[str, Any]:` `raise NotImplementedError` Method load didefinisikan sebagai abstrak, artinya setiap subclass wajib mengimplementasikannya. Method ini bertugas mengambil data mentah dari media penyimpanan dan mengembalikannya dalam bentuk dictionary. Perlu dicatat bahwa data yang dikembalikan belum berbentuk objek domain; konversi tersebut adalah tanggung jawab HabitTracker dan Habit. `@abstractmethod` `def save(self, data: Dict[str, Any]) -> None:` `raise NotImplementedError` Method save juga abstrak dan menerima snapshot data mentah. Storage tidak tahu dan tidak peduli isi data tersebut mewakili habit, streak, atau apapun. Ia hanya menyimpan apa yang diberikan. Di sinilah terlihat dengan jelas pemisahan tanggung jawab: storage tidak mengandung logika bisnis sama sekali.
4.	`class JsonStorage(BaseStorage):` JsonStorage adalah subclass dari BaseStorage, yang berarti ia memenuhi kontrak persistence yang telah didefinisikan. Dengan inheritance ini, JsonStorage dapat digunakan di mana pun BaseStorage diharapkan, termasuk di dalam HabitTracker. Dalam sistem HabitTracker, ini berarti tracker bisa menerima JsonStorage, SqlStorage, atau storage lain tanpa perubahan kode.

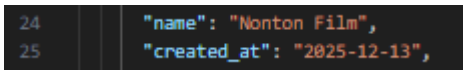
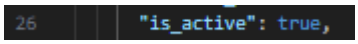
| **5.** | ```
50      def __init__(self, filepath: str) -> None:
51          self._filepath = filepath  # protected
```<br>Atribut _filepath dibuat protected, bukan public. Ini menandakan bahwa path file adalah detail internal storage dan tidak boleh dimodifikasi langsung oleh UI atau tracker. Tracker hanya berhak memanggil load() dan save(), bukan mengatur bagaimana file disimpan. Encapsulation di sini sederhana, tetapi tepat: storage menyembunyikan detail lokasi dan format file. |
| **6.** | ```
53      def load(self) -> Dict[str, Any]:
54          if not os.path.exists(self._filepath):
55              return {"habits": []}
```<br>Jika file tidak ada, storage tidak melempar error, melainkan mengembalikan struktur data kosong yang valid. Keputusan ini penting secara desain: aplikasi tetap bisa berjalan walaupun file belum pernah dibuat. HabitTracker kemudian akan memuat list habit kosong tanpa crash. |
| | ```
58              with open(self._filepath, "r", encoding="utf-8") as f:
59                  raw = json.load(f)
```<br>Storage hanya membaca file dan mem-parsing JSON. Tidak ada validasi habit, tidak ada pengecekan streak, dan tidak ada aturan domain di sini. |
| | ```
60              if not isinstance(raw, dict):
61                  return {"habits": []}
62              raw.setdefault("habits", [])
63              return raw
```<br>Bagian ini menunjukkan bahwa storage menjaga struktur minimum data, bukan kebenaran isinya. Selama data berbentuk dictionary dan memiliki key habits, storage menganggapnya valid. Jika format rusak, fallback tetap aman.. |
| | ```
64          except (json.JSONDecodeError, OSError):
65              return {"habits": []}
```<br>Exception handling di sini menunjukkan filosofi desain yang kuat: storage error tidak boleh menjatuhkan aplikasi. HabitTracker akan selalu menerima data dalam bentuk yang bisa diproses. Ini membuat aplikasi tahan terhadap file rusak/terhaps. |
| **7.** | Sebelum menyimpan file, storage memastikan direktori tersedia. Ini adalah detail teknis yang sepenuhnya disembunyikan dari tracker dan UI.<br>```
67      # simpan data ke file JSON
68      def save(self, data: Dict[str, Any]) -> None:
69          os.makedirs(os.path.dirname(self._filepath) or ".", exist_ok=True)
```<br>Storage menulis data apa adanya ke file JSON dengan format yang rapi. Perlu ditekankan bahwa data di sini sudah diproses oleh tracker dan domain. Storage tidak menambah, mengurangi, atau menginterpretasi isi data. Yang mana ini memperkuat peran storage.py sebagai passive persistence layer. |
| | ```
70          with open(self._filepath, "w", encoding="utf-8") as f:
71              json.dump(data, f, ensure_ascii=False, indent=2)
```<br>Storage menulis data apa adanya ke file JSON dengan format yang rapi. Perlu ditekankan bahwa data di sini sudah diproses oleh tracker dan domain. Storage tidak menambah, mengurangi, atau menginterpretasi isi data. Yang mana ini memperkuat peran storage.py sebagai passive persistence layer. |

4.2.6. File habits.json

Dalam sistem HabitTracker, file habits.json berfungsi sebagai media penyimpanan data persisten yang merepresentasikan keadaan (state) aplikasi pada waktu terakhir dijalankan. File ini bukan bagian dari logika program, melainkan hasil serialisasi dari objek-objek domain (Habit dan turunannya) yang dilakukan oleh HabitTracker melalui JsonStorage. Dengan
```

demikian, habits.json dapat dipahami sebagai snapshot data, bukan sebagai sumber aturan atau perilaku.

Bahasan ke-	Analisis
1.	 Struktur paling luar dari file ini adalah sebuah objek JSON ({}) yang memiliki satu key utama, yaitu "habits". Nilai dari key ini adalah sebuah array (list) yang berisi kumpulan habit. Struktur ini sesuai dengan kontrak yang disepakati antara HabitTracker dan JsonStorage, di mana tracker selalu menyimpan data dalam bentuk dictionary dengan key "habits". Setiap elemen di dalam array "habits" merepresentasikan satu objek habit yang telah diserialisasi dari kelas Habit (atau subclass-nya). Struktur ini adalah hasil langsung dari method Habit.to_dict() pada domain entity. Artinya, setiap field di sini memiliki padanan langsung dengan atribut dan state di objek Habit, bukan hasil interpretasi storage atau UI.
2.	 Field "type" memiliki peran penting dalam konteks OOP dan desain sistem. Nilai ini digunakan saat proses load oleh method Habit.from_dict() untuk menentukan kelas apa yang harus diinstansiasi kembali. Jika suatu saat terdapat subclass seperti "DailyHabit" atau tipe habit lain, field ini memungkinkan sistem melakukan polymorphism saat deserialisasi. Dengan adanya field ini, habits.json menjadi fleksibel dan future-proof. UI dan tracker tidak perlu mengetahui tipe konkret habit; mereka cukup memanggil method domain, dan domain akan membangun objek yang sesuai berdasarkan nilai "type".
3.	 Field "id" menyimpan UUID yang dihasilkan oleh domain melalui factory method Habit.new(). ID ini bersifat unik dan stabil, serta menjadi identitas utama habit di seluruh sistem. Tracker, UI, dan storage semuanya merujuk ke habit berdasarkan ID ini, bukan berdasarkan nama. Dalam OOP, ini mencerminkan konsep entity: meskipun nama habit bisa berubah, identitasnya tetap sama.

4.	 Field "name" adalah representasi langsung dari atribut privat __name pada objek Habit. Perubahan nilai ini hanya dapat dilakukan melalui setter domain (set_name), lalu disimpan kembali ke JSON. Hal ini menunjukkan bahwa JSON tidak pernah memvalidasi atau memodifikasi nama, melainkan hanya menyimpan hasil keputusan domain. Field "created_at" menyimpan tanggal pembuatan habit dalam format ISO (YYYY-MM-DD). Format ini konsisten dengan helper _date_to_iso() dan _parse_iso_date() di domain, sehingga konversi antara objek date dan string berjalan aman dan terkontrol.
5.	 Field ini merepresentasikan status aktif atau non-aktif suatu habit. Dalam sistem HabitTracker, status ini memengaruhi apakah habit boleh dicentang atau tidak. Namun, JSON hanya menyimpan nilainya; aturan “habit non-aktif tidak bisa dicentang” sepenuhnya ditegakkan oleh domain (mark_done).
6.	 Field ini menyimpan daftar tanggal di mana habit ditandai selesai. Di dalam domain, data ini sebenarnya disimpan sebagai Set[date] untuk efisiensi dan keunikan. Saat diserialisasi ke JSON, set tersebut diubah menjadi list string yang terurut. Riwayat ini menjadi dasar bagi: • perhitungan streak, • weekly progress, • dan statistik lain di HabitTracker. Namun, JSON sendiri tidak menghitung apa pun; ia hanya menyimpan histori mentah.
7.	 Field "frozen_dates" menyimpan tanggal-tanggal yang dibekukan (freeze) oleh domain sebagai kompensasi hari yang terlewat. Dalam contoh di atas, tanggal 2025-12-12 difreeze karena habit dilakukan lagi keesokan harinya dan token freeze masih tersedia. Adanya field ini menunjukkan bahwa konsep freeze adalah state domain yang persisten, bukan sekadar efek sementara. Ketika aplikasi ditutup dan dibuka kembali, domain dapat melanjutkan perhitungan streak dengan konsisten karena data freeze tersimpan di sini.

8.	 Kumpulan habit dalam satu array menunjukkan bahwa habits.json adalah single source of persistence untuk seluruh aplikasi. Setiap habit berdiri sebagai entitas terpisah, tetapi semuanya dikelola oleh satu tracker dan satu storage. Ini memudahkan load/save secara atomik dan menjaga konsistensi data.

4.3. Pengujian Sistem

Pengujian aplikasi dilakukan untuk memastikan bahwa sistem Habit Tracker yang dikembangkan dapat berjalan sesuai dengan kebutuhan fungsional yang telah dirancang. Pengujian ini juga bertujuan untuk memverifikasi bahwa setiap fitur utama bekerja dengan benar, stabil, serta mampu menangani interaksi pengguna secara tepat. Selain itu, pada bagian ini juga dijelaskan cara penggunaan aplikasi sebagai bagian dari validasi bahwa sistem dapat digunakan secara intuitif oleh pengguna.

Metode pengujian yang digunakan pada aplikasi ini adalah pengujian fungsional (functional testing) dengan pendekatan black-box testing, di mana pengujian difokuskan pada input dan output sistem tanpa melihat implementasi kode secara internal.

4.3.1. Lingkungan pengujian

Pengujian aplikasi dilakukan dengan spesifikasi lingkungan sebagai berikut:

- Sistem Operasi: Windows
- Bahasa Pemrograman: Python
- Library GUI: Tkinter
- Media Penyimpanan Data: File JSON (habits.json)
- Editor / Interpreter: Python 3.11

Aplikasi dijalankan melalui file main.py sebagai entry point utama sistem.

4.3.2. Cara Menjalankan dan Menggunakan Aplikasi

Aplikasi Habit Tracker dijalankan dengan mengeksekusi file main.py. Saat aplikasi dijalankan, sistem akan secara otomatis memuat data habit terakhir yang tersimpan di file habits.json. Jika file tersebut belum ada atau kosong, aplikasi tetap dapat berjalan dengan kondisi awal tanpa habit.

Setelah aplikasi terbuka, pengguna akan melihat antarmuka utama yang terbagi menjadi dua bagian, yaitu panel kiri dan panel kanan. Pada panel kiri, pengguna dapat melakukan pengelolaan habit serta checklist harian. Pengguna dapat menambahkan habit baru dengan menekan tombol Tambah, kemudian memasukkan nama habit melalui dialog input yang muncul. Habit yang berhasil ditambahkan akan langsung ditampilkan pada daftar habit dan checklist harian.

Pengguna juga dapat memilih salah satu habit pada daftar untuk melakukan aksi Edit atau Hapus. Aksi edit memungkinkan pengguna mengganti nama habit, sedangkan aksi hapus akan menghapus habit setelah pengguna menyetujui dialog konfirmasi yang ditampilkan sistem.

Checklist harian ditampilkan dalam bentuk checkbox. Pengguna dapat mencentang atau membatalkan centang pada habit untuk menandai apakah habit tersebut telah dilakukan pada tanggal yang aktif. Sistem akan secara otomatis menyimpan perubahan ini dan memperbarui tampilan tanpa perlu tindakan tambahan dari pengguna.

Pengguna juga dapat mengganti tanggal aktif menggunakan tombol Pilih Tanggal dengan format YYYY-MM-DD, atau kembali ke tanggal hari ini dengan menekan tombol Hari Ini. Checklist dan data yang ditampilkan akan menyesuaikan dengan tanggal yang dipilih.
Pada panel kanan, pengguna dapat melihat ringkasan mingguan (weekly summary) yang berisi informasi seperti:

- Rentang tanggal minggu aktif
- Persentase penyelesaian keseluruhan
- Habit dengan streak aktif terbaik
- Habit dengan streak terpanjang sepanjang waktu

Selain itu, panel ini juga menampilkan tabel ringkasan per habit yang mencakup jumlah hari selesai, tingkat penyelesaian, badge mingguan, sisa freeze, streak saat ini, dan streak terpanjang. Pengguna dapat mengekspor ringkasan mingguan ke dalam file CSV dengan menekan tombol Export CSV.

4.3.3. Pengujian Fungsional Fitur Utama

a. Pengujian Penambahan Habit
   Pengujian dilakukan dengan menekan tombol Tambah dan memasukkan nama habit. Hasil pengujian menunjukkan bahwa sistem berhasil menambahkan habit baru, menyimpannya ke dalam file JSON, dan langsung menampilkan habit tersebut pada UI. Jika nama habit kurang dari dua karakter, sistem menolak input dan menampilkan pesan kesalahan.
b. Pengujian Pengeditan dan Penghapusan Habit
   Pengujian pengeditan dilakukan dengan memilih habit dari daftar, lalu menekan tombol Edit. Sistem berhasil memperbarui nama habit dan menyimpan perubahan tersebut. Pengujian penghapusan dilakukan dengan menekan tombol Hapus, dan sistem menampilkan dialog konfirmasi sebelum benar-benar menghapus habit. Hal ini

menunjukkan bahwa mekanisme validasi dan keamanan interaksi pengguna berjalan dengan baik.

c. Pengujian Checklist Harian
   Pengujian checklist dilakukan dengan mencentang dan membatalkan centang habit pada tanggal tertentu. Sistem berhasil memperbarui status habit, menghitung ulang streak, serta menyimpan perubahan secara otomatis. Checklist juga berubah sesuai dengan tanggal yang dipilih, menunjukkan bahwa sistem mampu menangani data berbasis tanggal dengan benar.
d. Pengujian Perhitungan Streak dan Freeze
   Pengujian dilakukan dengan mensimulasikan hari terlewat dan melakukan habit kembali di hari berikutnya. Sistem berhasil menerapkan mekanisme freeze otomatis sesuai dengan aturan domain, serta menghitung streak aktif dan streak terpanjang dengan benar. Hasil perhitungan ditampilkan secara konsisten pada ringkasan mingguan.
e. Pengujian Ringkasan Mingguan dan Export Data
   Pengujian ringkasan mingguan menunjukkan bahwa sistem mampu menampilkan statistik mingguan secara akurat, termasuk badge dan persentase penyelesaian. Pengujian fitur Export CSV berhasil menghasilkan file CSV yang berisi data ringkasan mingguan sesuai dengan tampilan aplikasi..

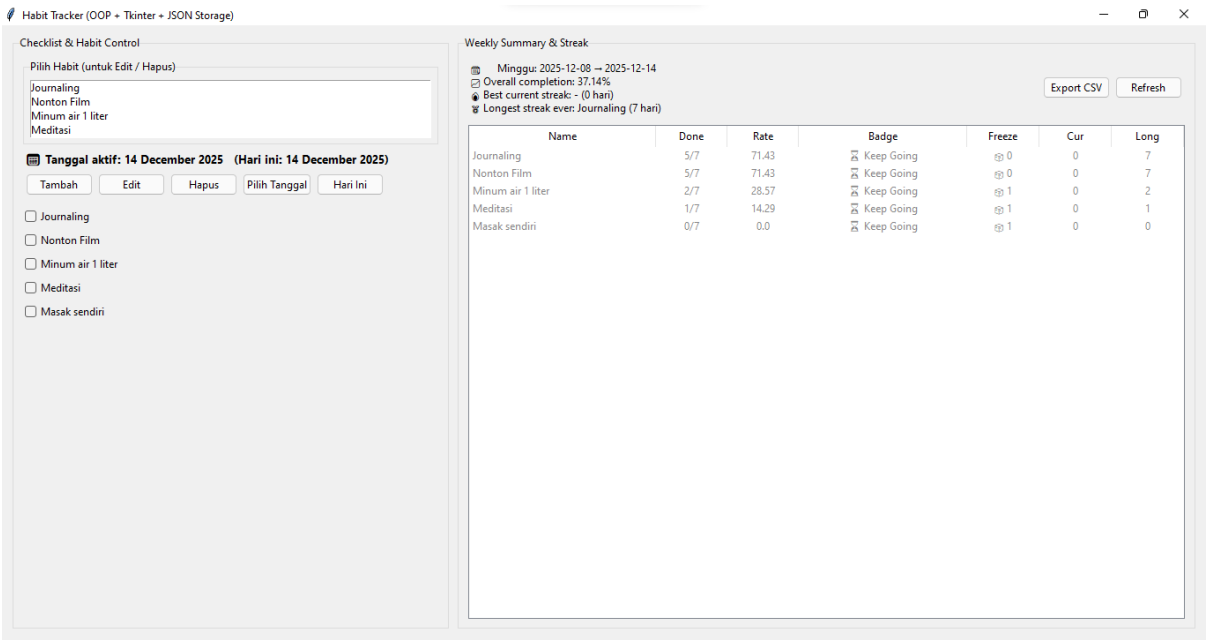
4.3.4. Hasil Pengujian

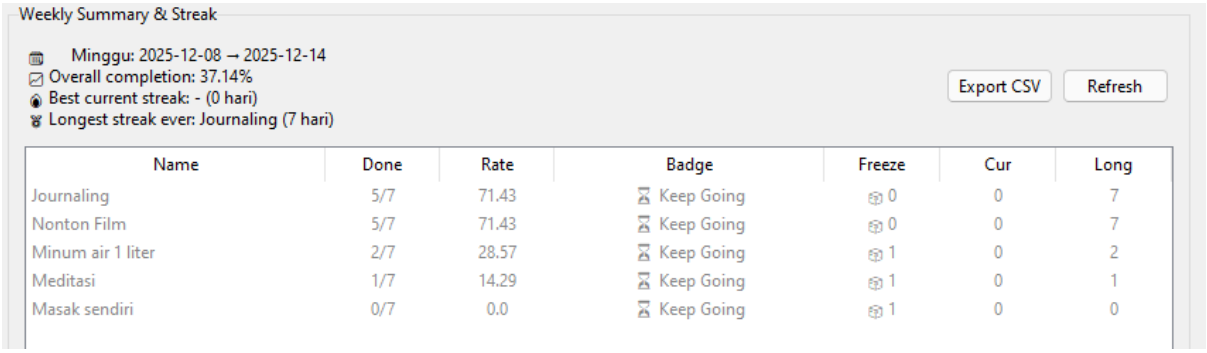
Berdasarkan seluruh pengujian yang telah dilakukan, dapat disimpulkan bahwa:

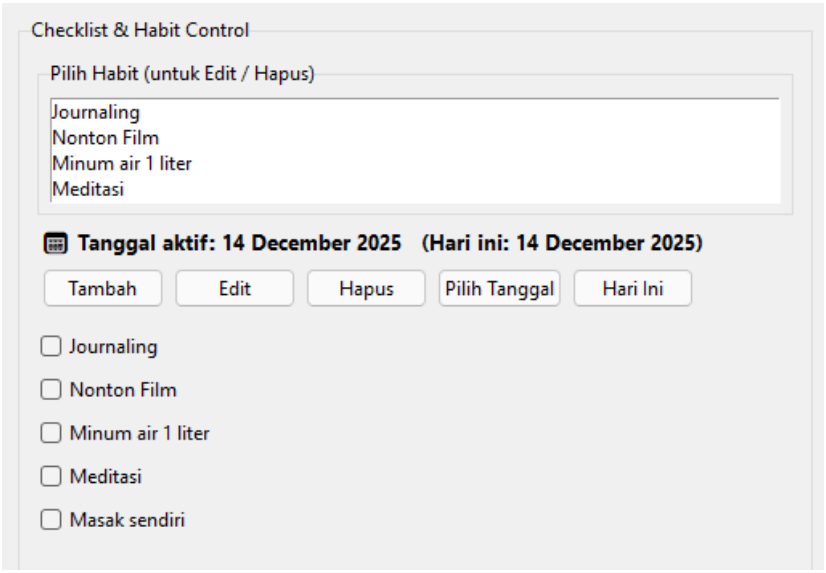
- Seluruh fitur utama aplikasi berjalan sesuai dengan spesifikasi
- Tidak ditemukan kesalahan fatal (runtime error) selama pengujian
- Interaksi pengguna bersifat intuitif dan responsif
- Data tersimpan dan dimuat kembali dengan konsisten

Dengan demikian, aplikasi Habit Tracker dinyatakan berhasil diimplementasikan dan siap digunakan, serta telah memenuhi tujuan pengembangan yang ditetapkan pada tahap perancangan.

4.4. Tampilan Aplikasi







BAB V
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilakukan, dapat disimpulkan bahwa aplikasi Habit Tracker berhasil dikembangkan dengan menerapkan prinsip Pemrograman Berorientasi Objek (OOP) secara konsisten dan terstruktur. Aplikasi ini mampu membantu pengguna dalam mencatat, memantau, dan mengevaluasi kebiasaan harian melalui fitur checklist berbasis tanggal, perhitungan streak, serta ringkasan mingguan. Seluruh fungsi utama berjalan dengan baik dan sesuai dengan tujuan awal pengembangan sistem.

Penerapan konsep OOP pada aplikasi ini terlihat jelas melalui pemisahan tanggung jawab antar kelas dan file, di mana kelas Habit berperan sebagai domain entity yang menyimpan aturan dan perilaku habit, HabitTracker bertindak sebagai application service yang mengorkestrasi alur logika, storage menangani persistensi data, dan ui berfungsi sebagai antarmuka pengguna. Pemisahan ini membuat sistem lebih mudah dipahami, dikembangkan, dan dipelihara, sekaligus mengurangi ketergantungan antar komponen.

Selain itu, penggunaan struktur data yang tepat, seperti list, dictionary, dan set, serta penerapan inheritance dan polymorphism, memungkinkan sistem untuk menangani data habit secara efisien dan fleksibel. Hasil pengujian menunjukkan bahwa aplikasi dapat menyimpan data dengan konsisten, menampilkan informasi secara akurat, serta memberikan interaksi pengguna yang intuitif. Dengan demikian, aplikasi Habit Tracker dapat dikatakan telah memenuhi kriteria sebagai aplikasi berbasis OOP yang fungsional dan layak digunakan.

5.2. Saran

Meskipun aplikasi Habit Tracker telah berjalan dengan baik, masih terdapat beberapa peluang pengembangan yang dapat dilakukan di masa mendatang. Salah satu pengembangan yang disarankan adalah mengganti atau menambahkan mekanisme penyimpanan data berbasis database SQL. Dengan menggunakan database relasional, pengelolaan data habit dapat menjadi lebih terstruktur, aman, dan scalable, terutama jika jumlah data semakin besar atau aplikasi dikembangkan untuk banyak pengguna.

Selain itu, pada fitur pemilihan tanggal, aplikasi saat ini masih mengharuskan pengguna memasukkan tanggal secara manual dengan format YYYY-MM-DD. Ke depannya, fitur ini dapat dikembangkan dengan menambahkan komponen kalender (date picker) pada antarmuka pengguna, sehingga pengguna dapat memilih tanggal secara visual dan lebih intuitif tanpa harus mengetik format tanggal secara manual.

Pengembangan lain yang dapat dipertimbangkan adalah penambahan fitur seperti statistik bulanan, grafik visualisasi progres habit, sistem notifikasi atau pengingat harian, serta dukungan multi-user. Dengan pengembangan-pengembangan tersebut, aplikasi Habit Tracker diharapkan dapat menjadi lebih lengkap, menarik, dan bermanfaat bagi pengguna dalam membangun kebiasaan positif secara berkelanjutan.