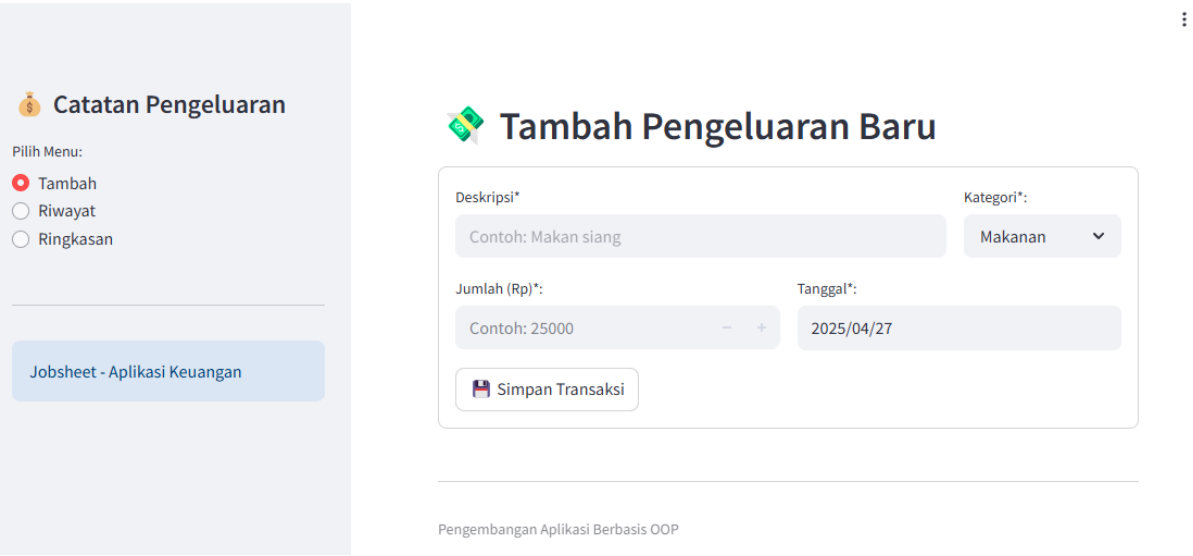
POLITEKNIK NEGERI SEMARANG
JURUSAN TEKNIK ELEKTRO
PROGRAM STUDI STR TEKNOLOGI REKAYASA KOMPUTER

JOBSHEET 11:
INTEGRASI OOP DALAM APLIKASI PENGELUARAN SEDERHANA



Dosen:
Ir. Prayitno, S.ST., M.T., Ph.D.

Nama Mahasiswa : _____
NIM Mahasiswa : _____

Tahun Akademik 2025

A. Tujuan Instruksional Khusus

Setelah menyelesaikan praktikum ini, mahasiswa diharapkan mampu:

1. Merancang dan mengimplementasikan kelas-kelas (Transaksi, AnggaranHarian) menggunakan prinsip OOP (Enkapsulasi, Komposisi) untuk mengelola data pengeluaran secara terstruktur.
2. Membuat dan mengelola skema database sederhana menggunakan SQLite untuk persistensi data transaksi.
3. Mengimplementasikan operasi dasar penyimpanan (Create) dan pengambilan (Read) data pada database SQLite melalui metode kelas menggunakan modul sqlite3 Python.
4. Menggunakan library Pandas untuk membaca data dari database SQLite dan menyiapkannya untuk ditampilkan.
5. Membangun antarmuka pengguna (UI) berbasis web yang interaktif menggunakan library Streamlit, termasuk form input dan elemen display data (tabel, metrik, grafik).
6. Mengintegrasikan komponen backend (logika OOP dan akses database) dengan frontend (antarmuka Streamlit) untuk membuat aplikasi pencatat pengeluaran harian yang fungsional.

B. Alat dan Bahan

- **Perangkat Lunak:**
  - Sistem operasi Windows/Linux/macOS.
  - Python 3.x (versi terbaru yang stabil direkomendasikan).
  - IDE atau Code Editor (misalnya, VS Code, PyCharm, Sublime Text, Jupyter Notebook/Lab, atau IDLE).
  - Library Python:
    - `streamlit`: Untuk membangun antarmuka pengguna web.
    - `pandas`: Untuk manipulasi data dan menampilkan tabel.
    - Perintah instalasi (jika belum terinstal, jalankan di terminal/command prompt):
      Bash
      ```
      pip install streamlit pandas
      ```
  - Modul `sqlite3`: Untuk interaksi database (sudah termasuk dalam instalasi standar Python).
  - Web Browser (Chrome, Firefox, Edge, dll.) untuk melihat aplikasi Streamlit.
- **Perangkat Keras:**
  - Komputer/Laptop dengan spesifikasi yang memadai untuk menjalankan Python dan Streamlit.
- **Bahan Pendukung:**
  - Jobsheet ini.
  - Modul/slide perkuliahan terkait Konsep OOP, SQLite, dan dasar Python.
  - Koneksi internet (terutama untuk instalasi library dan kadang saat menjalankan Streamlit).
  - Dokumentasi Streamlit: https://docs.streamlit.io/
  - Dokumentasi Pandas: https://pandas.pydata.org/docs/
  - Dokumentasi `sqlite3` Python: https://docs.python.org/3/library/sqlite3.html

C. Dasar Teori

Aplikasi pencatat pengeluaran harian ini merupakan contoh praktis bagaimana berbagai teknologi dan konsep pemrograman dapat diintegrasikan untuk menciptakan solusi fungsional. Kita akan menggabungkan prinsip Pemrograman Berorientasi Objek (OOP) untuk

menstrukturkan data dan logika, database SQLite untuk penyimpanan data yang persisten, library Pandas untuk mempermudah manipulasi dan penyajian data, serta Streamlit untuk membangun antarmuka pengguna berbasis web yang interaktif.

1. Konsep OOP dalam Aplikasi Pengeluaran

OOP membantu kita memodelkan entitas dunia nyata (dalam kasus ini, transaksi pengeluaran) dan mengelola operasinya secara terstruktur.

- **Kelas `Transaksi`**: Kelas ini bertindak sebagai *blueprint* untuk setiap catatan pengeluaran individual.
  - **Atribut**: Menyimpan data spesifik seperti `deskripsi` (misalnya, "Makan Siang"), `jumlah` (misalnya, 25000.0), `kategori` (misalnya, "Makanan"), dan `tanggal` (misalnya, objek `datetime.date`).
  - **Enkapsulasi**: Setiap objek `Transaksi` membungkus semua informasi yang relevan untuk satu pengeluaran, menyembunyikan detail internalnya.
- **Kelas `AnggaranHarian` (atau `ManajerTransaksi`)**: Kelas ini bertanggung jawab untuk mengelola keseluruhan data transaksi.
  - **Metode**: Menyediakan fungsi untuk berinteraksi dengan data, seperti `tambah_transaksi(transaksi_baru)`, `get_semua_transaksi()`, `hitung_total_pengeluaran(filter_tanggal=None)`, `get_pengeluaran_per_kategori(filter_tanggal=None)`. Metode ini berisi logika bisnis aplikasi.
  - **Komposisi**: Secara konseptual, objek `AnggaranHarian` *memiliki* atau *mengelola* kumpulan data `Transaksi`. Dalam implementasi kita, ia tidak menyimpan list objek secara langsung di memori, melainkan berinteraksi dengan database sebagai representasi kumpulan data tersebut. Ini adalah bentuk komposisi di mana `AnggaranHarian` bergantung pada (dan menggunakan) data `Transaksi`.
  - **Pemisahan Tanggung Jawab**: Kelas ini memisahkan logika bisnis (bagaimana data diolah dan dihitung) dari detail penyimpanan data (yang didelegasikan ke modul database) dan dari tampilan (yang ditangani oleh Streamlit). Ini membuatnya mirip dengan **Repository Pattern** sederhana.

2. Penyimpanan Data dengan SQLite (`sqlite3`)

Untuk menyimpan data pengeluaran agar tidak hilang saat aplikasi ditutup, kita menggunakan SQLite. SQLite adalah sistem manajemen database relasional (RDBMS) yang ringan, berbasis file (database disimpan dalam satu file `.db`), dan sudah termasuk dalam library standar Python (`sqlite3`), sehingga tidak memerlukan instalasi server database terpisah.

- **Tabel `transaksi`**: Kita akan membuat tabel ini di dalam file database (misalnya `pengeluaran_harian.db`). Kolom-kolomnya akan mencerminkan atribut `Transaksi`:
  - `id`: Kunci utama (Primary Key), integer, otomatis bertambah (Auto Increment).
  - `deskripsi`: Teks, tidak boleh kosong (NOT NULL).
  - `jumlah`: Angka real (bisa desimal), tidak boleh kosong, sebaiknya positif (REAL NOT NULL CHECK(jumlah > 0)).
  - `kategori`: Teks (bisa kosong/NULL).
  - `tanggal`: Tanggal (DATE NOT NULL).
- **Operasi SQL Dasar**: Modul `sqlite3` memungkinkan kita menjalankan perintah SQL:
  - `INSERT INTO transaksi (...) VALUES (?, ?, ?, ?)`: Menambah data baru. Penggunaan `?` (placeholder) penting untuk mencegah *SQL Injection*.
  - `SELECT * FROM transaksi ORDER BY ...`: Mengambil semua data.
  - `SELECT SUM(jumlah) FROM transaksi WHERE ...`: Menghitung total.

    - `SELECT kategori, SUM(jumlah) FROM transaksi WHERE ... GROUP BY kategori`: Menghitung total per kategori.
- **Koneksi & Cursor**: Interaksi dilakukan melalui objek `Connection` (menghubungkan ke file DB) dan `Cursor` (mengeksekusi perintah SQL). Penting untuk menutup koneksi (`conn.close()`) setelah selesai, atau lebih baik lagi menggunakan `with sqlite3.connect(...) as conn:` yang menanganinya secara otomatis (meskipun dalam contoh modular kita, kita menggunakan fungsi helper untuk manajemen koneksi).

3. Antarmuka Web dengan Streamlit

Streamlit adalah framework Python open-source yang memungkinkan pembuatan aplikasi web interaktif untuk data science dan aplikasi lainnya dengan cepat, hanya menggunakan skrip Python.

- **Widget Input**: Streamlit menyediakan berbagai widget untuk input pengguna: `st.text_input`, `st.number_input`, `st.selectbox`, `st.date_input`, `st.button`, `st.form`, dll. Ini memudahkan pembuatan form untuk menambah data transaksi.
- **Widget Output/Display**: Untuk menampilkan hasil:
    - `st.write()`: Menampilkan teks, angka, list, dictionary, dll.
    - `st.dataframe()`: Menampilkan DataFrame Pandas sebagai tabel interaktif yang bagus.
    - `st.table()`: Menampilkan data (seperti DataFrame atau list of lists) sebagai tabel statis.
    - `st.metric()`: Menampilkan satu angka penting (metrik) dengan label dan bisa menunjukkan perubahan (delta). Cocok untuk total pengeluaran.
    - `st.bar_chart()`, `st.line_chart()`, dll.: Untuk visualisasi data sederhana.
    - `st.success()`, `st.error()`, `st.warning()`, `st.info()`: Menampilkan pesan status dengan ikon dan warna yang sesuai.
    - `st.spinner()`: Menampilkan indikator loading saat proses backend berjalan.
- **Struktur Aplikasi**: Widget dapat diatur tata letaknya menggunakan `st.columns`, dikelompokkan dalam `st.expander`, atau dipisahkan menjadi halaman/bagian menggunakan `st.sidebar`, `st.tabs`, atau struktur aplikasi multi-halaman Streamlit.
- **Model Eksekusi**: Penting untuk diingat bahwa Streamlit menjalankan ulang skrip dari atas ke bawah setiap kali ada interaksi pengguna. Oleh karena itu, state aplikasi (seperti instance `AnggaranHarian` atau status filter) perlu dikelola dengan benar menggunakan mekanisme seperti `st.session_state` atau caching (`@st.cache_data`, `@st.cache_resource`).

4. Pandas untuk Integrasi Data

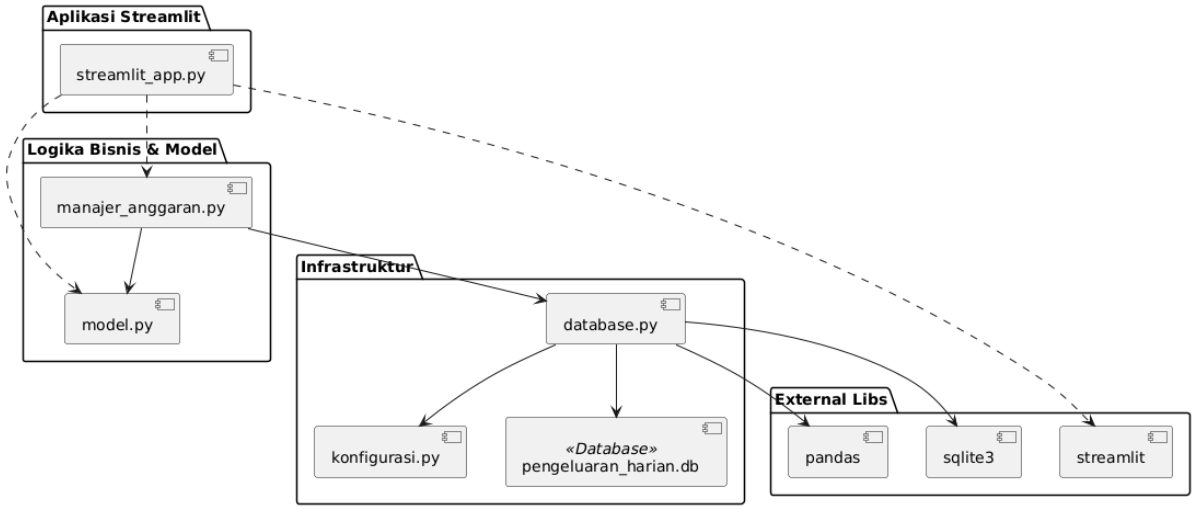
Meskipun tidak wajib, Pandas sering digunakan bersama Streamlit dan SQLite karena:

- `pd.read_sql_query(sql, conn)`: Memudahkan pembacaan hasil query SELECT langsung menjadi DataFrame.
- `st.dataframe(df)`: Cara paling mudah dan interaktif untuk menampilkan data tabular di Streamlit.
- Manipulasi Data: Pandas menyediakan banyak fungsi untuk membersihkan, mengubah format, atau menganalisis data sebelum ditampilkan jika diperlukan.

Dengan menggabungkan keempat pilar ini (OOP, SQLite, Streamlit, Pandas), kita dapat membangun aplikasi web pencatat pengeluaran yang terstruktur, persisten, interaktif, dan mudah dikembangkan lebih lanjut.

D. Langkah Praktikum

Praktikum ini akan memandu Anda membuat aplikasi pencatat pengeluaran harian dari awal hingga akhir, menggunakan struktur modular, kelas OOP, database SQLite, dan antarmuka web Streamlit.



Penjelasan Kelas Diagram (Diagram Komponen):
Diagram ini menunjukkan struktur akhir aplikasi yang modular:

- **`streamlit_app.py`**: Lapisan presentasi (UI) yang menggunakan Streamlit dan memanggil lapisan logika bisnis.
- **`manajer_anggaran.py`**: Lapisan logika bisnis/repository, menggunakan model dan utilitas database.
- **`model.py`**: Lapisan data, mendefinisikan struktur `Transaksi`.
- **`database.py`**: Lapisan akses data, menangani interaksi SQLite.
- **`konfigurasi.py`**: Menyimpan konstanta.
- **`pengeluaran_harian.db`**: Penyimpanan data fisik.
- **External Libs**: Library yang digunakan.

Panah menunjukkan dependensi antar modul. Ini adalah arsitektur yang umum digunakan untuk memisahkan tanggung jawab dalam sebuah aplikasi.

Langkah 1: Persiapan Awal (Konfigurasi & Setup Database)

Tujuan: Menyiapkan file konfigurasi dasar dan skrip untuk membuat database SQLite beserta tabelnya.

- **File 1: `konfigurasi.py`**

```
1: # konfigurasi.py
2: import os
3:
4: BASE_DIR = os.path.dirname(os.path.abspath(__file__))
5: NAMA_DB = 'pengeluaran_harian.db'
6: DB_PATH = os.path.join(BASE_DIR, NAMA_DB)
7: KATEGORI_PENGELUARAN = ["Makanan", "Transportasi", "Hiburan", "Tagihan",
"Belanja", "Kesehatan", "Pendidikan", "Lainnya"]
8: KATEGORI_DEFAULT = "Lainnya"
```

- **File 2: `setup_db_pengeluaran.py`** Buat file ini untuk setup database awal. **Jalankan file ini sekali saja** dari terminal (`python setup_db_pengeluaran.py`).

```
# setup_db_pengeluaran.py
import sqlite3
import os
from konfigurasi import DB_PATH # Ambil path dari konfigurasi
```

```
def setup_database():
    print(f"Memeriksa/membuat database di: {DB_PATH}")
    conn = None
    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        sql_create_table = """
        CREATE TABLE IF NOT EXISTS transaksi (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            deskripsi TEXT NOT NULL,
            jumlah REAL NOT NULL CHECK(jumlah > 0),
            kategori TEXT,
            tanggal DATE NOT NULL
        );"""
        print(" Membuat tabel 'transaksi' (jika belum ada)...")
        cursor.execute(sql_create_table)
        conn.commit(); print(" -> Tabel 'transaksi' siap.")
        return True
    except sqlite3.Error as e: print(f" -> Error SQLite saat setup: {e}");
return False
    finally:
        if conn: conn.close(); print(" -> Koneksi DB setup ditutup.")

if __name__ == "__main__":
    print("--- Memulai Setup Database Pengeluaran ---")
    if          setup_database():          print(f"\nSetup          database
'{os.path.basename(DB_PATH)}' selesai.")
    else: print(f"\nSetup database GAGAL.")
    print("--- Setup Database Selesai ---")
```

Observasi: Setelah menjalankan `setup_db_pengeluaran.py`, Anda akan memiliki file database `pengeluaran_harian.db` dengan tabel `transaksi` di direktori proyek Anda.

Langkah 2: Modul Akses Database (`database.py`)

Tujuan: Membuat modul terpisah yang berisi fungsi-fungsi untuk berinteraksi dengan database SQLite (koneksi, eksekusi query, fetch data).

```
File: database.py
01: # database.py
02: import sqlite3
03: import pandas as pd
04: from konfigurasi import DB_PATH # Gunakan path dari konfigurasi
05:
06: def get_db_connection() -> sqlite3.Connection | None:
07:     """Membuka dan mengembalikan koneksi baru ke database SQLite."""
08:     try:
09:                         conn   =   sqlite3.connect(DB_PATH,   timeout=10,
detect_types=sqlite3.PARSE_DECLTYPES)
10:         conn.row_factory = sqlite3.Row # Akses kolom by name
11:         return conn
12:      except sqlite3.Error as e: print(f"ERROR [database.py] Koneksi DB
gagal: {e}"); return None
13:
14: def execute_query(query: str, params: tuple = None):
15:        """Menjalankan  query  non-SELECT.  Mengembalikan  lastrowid  jika
INSERT."""
16:     conn = get_db_connection();
17:     if not conn: return None
18:     last_id = None
```

```
19:     try:
20:         cursor = conn.cursor();
21:         if params: cursor.execute(query, params)
22:         else: cursor.execute(query)
23:         conn.commit(); last_id = cursor.lastrowid; return last_id
24:     except sqlite3.Error as e: print(f"ERROR [database.py] Query gagal:
{e} | Query: {query[:60]}"); conn.rollback(); return None
25:     finally:
26:         if conn: conn.close()
27:
28: def fetch_query(query: str, params: tuple = None, fetch_all: bool =
True):
29:     """Menjalankan query SELECT dan mengembalikan hasil."""
30:     conn = get_db_connection();
31:     if not conn: return None
32:     try:
33:         cursor = conn.cursor();
34:         if params: cursor.execute(query, params)
35:         else: cursor.execute(query)
36:         result = cursor.fetchall() if fetch_all else cursor.fetchone();
return result
37:     except sqlite3.Error as e: print(f"ERROR [database.py] Fetch gagal:
{e} | Query: {query[:60]}"); return None
38:     finally:
39:         if conn: conn.close()
40:
41: def get_dataframe(query: str, params: tuple = None) -> pd.DataFrame:
42:      """Menjalankan query SELECT dan mengembalikan DataFrame Pandas."""
43:      conn = get_db_connection();
44:      if not conn: return pd.DataFrame()
45:      try: df = pd.read_sql_query(query, conn, params=params); return df
46:       except Exception as e: print(f"ERROR [database.py] Gagal baca ke
DataFrame: {e}"); return pd.DataFrame()
47:      finally:
48:         if conn: conn.close()
49:
50: def setup_database_initial(): # Fungsi setup dipindah ke sini juga
(opsional)
51:       """Memastikan tabel transaksi ada (dipanggil oleh AnggaranHarian
jika perlu)."""
52:        print(f"Memeriksa/membuat tabel di database (via database.py):
{DB_PATH}")
53:     conn = get_db_connection();
54:     if not conn: return False
55:     try:
56:         cursor = conn.cursor();
57:         sql_create_table = """
58:         CREATE TABLE IF NOT EXISTS transaksi (
59:               id INTEGER PRIMARY KEY AUTOINCREMENT, deskripsi TEXT NOT
NULL,
60:                jumlah REAL NOT NULL CHECK(jumlah > 0), kategori TEXT,
tanggal DATE NOT NULL );"""
61:        cursor.execute(sql_create_table); conn.commit(); print(" -> Tabel
'transaksi' siap."); return True
62:      except sqlite3.Error as e: print(f"Error SQLite saat setup tabel:
{e}"); return False
63:     finally:
64:         if conn: conn.close()
```

Observasi: Modul ini berisi fungsi-fungsi inti untuk semua operasi database. Fungsi `setup_database_initial` juga disertakan di sini agar bisa dipanggil oleh modul lain jika diperlukan (misalnya, saat `AnggaranHarian` pertama kali dibuat).

Langkah 3: Modul Model Data (`model.py`)

Tujuan: Mendefinisikan kelas `Transaksi` yang merepresentasikan struktur data untuk satu record pengeluaran.

```
01: # model.py
02: import datetime
03:
04: class Transaksi:
05:         """Merepresentasikan satu entitas transaksi pengeluaran (Data
Class)."""
06:     def __init__(self, deskripsi: str, jumlah: float, kategori: str,
07:                  tanggal: datetime.date | str, id_transaksi: int | None
= None):
08:         self.id = id_transaksi
09:       self.deskripsi = str(deskripsi) if deskripsi else "Tanpa Deskripsi"
10:         try:
11:             jumlah_float = float(jumlah)
12:             self.jumlah = jumlah_float if jumlah_float > 0 else 0.0
13:             if jumlah_float <= 0: print(f"Peringatan: Jumlah '{jumlah}'
harus positif.")
14:                 except (ValueError, TypeError): self.jumlah = 0.0;
print(f"Peringatan: Jumlah '{jumlah}' tidak valid.")
15:         self.kategori = str(kategori) if kategori else "Lainnya"
16:         if isinstance(tanggal, datetime.date): self.tanggal = tanggal
17:         elif isinstance(tanggal, str):
18:            try: self.tanggal = datetime.datetime.strptime(tanggal, "%Y-
%m-%d").date()
19:                except ValueError: self.tanggal = datetime.date.today();
print(f"Peringatan: Format tgl '{tanggal}' salah.")
20:         else: self.tanggal = datetime.date.today(); print(f"Peringatan:
Tipe tgl '{type(tanggal)}' tidak valid.")
21:
22:     def __repr__(self) -> str:
23:         try:
24:              import locale; locale.setlocale(locale.LC_ALL, 'id_ID.UTF-
8')
25:                   jml_str = locale.format_string("%.0f", self.jumlah,
grouping=True)
26:         except: jml_str = f"{self.jumlah:.0f}"
27:        return f"Transaksi(ID:{self.id}, Tgl:{self.tanggal.strftime('%Y-
%m-%d')}, Jml:{jml_str}, Kat:'{self.kategori}', Desc:'{self.deskripsi}')"
28:
29:     def to_dict(self) -> dict:
30:             return {"deskripsi": self.deskripsi, "jumlah": self.jumlah,
"kategori": self.kategori, "tanggal": self.tanggal.strftime("%Y-%m-%d")}
```

Observasi: Kelas ini fokus pada struktur data dan validasi dasar saat objek dibuat.

Langkah 4: Modul Manajer Anggaran (`manajer_anggaran.py`)

Tujuan: Membuat kelas `AnggaranHarian` yang berisi logika bisnis aplikasi dan menggunakan modul `database.py` untuk akses data.

```
01: # manajer_anggaran.py
02: import datetime
03: import pandas as pd
```

```
04: from model import Transaksi
05: import database # Impor modul database kita
06:
07: class AnggaranHarian:
08:          """Mengelola logika bisnis pengeluaran harian (Repository
Pattern)."""
09:      _db_setup_done = False # Flag untuk memastikan setup DB hanya dicek
sekali per sesi
10:
11:     def __init__(self):
12:         if not AnggaranHarian._db_setup_done:
13:             print("[AnggaranHarian] Melakukan pengecekan/setup database
awal...")
14:             if database.setup_database_initial(): # Panggil fungsi setup
dari database.py
15:                 AnggaranHarian._db_setup_done = True
16:                 print("[AnggaranHarian] Database siap.")
17:             else:
18:                   print("[AnggaranHarian] KRITICAL: Setup database awal
GAGAL!")
19:
20:     def tambah_transaksi(self, transaksi: Transaksi) -> bool:
21:          if not isinstance(transaksi, Transaksi) or transaksi.jumlah <=
0: return False
22:            sql = "INSERT INTO transaksi (deskripsi, jumlah, kategori,
tanggal) VALUES (?, ?, ?, ?)"
23:                    params = (transaksi.deskripsi, transaksi.jumlah,
transaksi.kategori, transaksi.tanggal.strftime("%Y-%m-%d"))
24:         last_id = database.execute_query(sql, params)
25:         if last_id is not None: transaksi.id = last_id; return True
26:         return False
27:
28:     def get_semua_transaksi_obj(self) -> list[Transaksi]:
29:           sql = "SELECT id, deskripsi, jumlah, kategori, tanggal FROM
transaksi ORDER BY tanggal DESC, id DESC"
30:         rows = database.fetch_query(sql, fetch_all=True)
31:         transaksi_list = []
32:         if rows:
33:             for row in rows:
34:                 transaksi_list.append(Transaksi(id_transaksi=row['id'],
deskripsi=row['deskripsi'], jumlah=row['jumlah'], kategori=row['kategori'],
tanggal=row['tanggal']))
35:         return transaksi_list
36:
37:      def get_dataframe_transaksi(self, filter_tanggal: datetime.date |
None = None) -> pd.DataFrame:
38:             query = "SELECT tanggal, kategori, deskripsi, jumlah FROM
transaksi"
39:          params = None
40:            if filter_tanggal: query += " WHERE tanggal = ?"; params =
(filter_tanggal.strftime("%Y-%m-%d"),)
41:          query += " ORDER BY tanggal DESC, id DESC"
42:          df = database.get_dataframe(query, params=params)
43:          if not df.empty:
44:               try:
45:                         import locale; locale.setlocale(locale.LC_ALL,
'id_ID.UTF-8')
46:                        df['Jumlah (Rp)'] = df['jumlah'].map(lambda x:
locale.currency(x or 0, grouping=True, symbol='Rp ')[:-3])
47:                 except: df['Jumlah (Rp)'] = df['jumlah'].map(lambda x:
f"Rp {x or 0:,.0f}".replace(",", "."))
```

```
48:                   df = df[['tanggal', 'kategori', 'deskripsi', 'Jumlah
(Rp)']]
49:           return df
50:
51:      def hitung_total_pengeluaran(self, tanggal: datetime.date | None =
None) -> float:
52:          sql = "SELECT SUM(jumlah) FROM transaksi"; params = None
53:                if tanggal: sql += " WHERE tanggal = ?"; params =
(tanggal.strftime("%Y-%m-%d"),)
54:        result = database.fetch_query(sql, params=params, fetch_all=False)
55:          if result and result[0] is not None: return float(result[0])
56:          return 0.0
57:
58:     def get_pengeluaran_per_kategori(self, tanggal: datetime.date | None
= None) -> dict:
59:          hasil = {}; sql = "SELECT kategori, SUM(jumlah) FROM transaksi";
params = []
60:                       if tanggal: sql += " WHERE tanggal = ?";
params.append(tanggal.strftime("%Y-%m-%d"))
61:            sql += " GROUP BY kategori HAVING SUM(jumlah) > 0 ORDER BY
SUM(jumlah) DESC"
62:          rows = database.fetch_query(sql, params=tuple(params) if params
else None, fetch_all=True)
63:          if rows:
64:              for row in rows:
65:                       kategori = row['kategori'] if row['kategori'] else
"Lainnya"; jumlah = float(row[1]) if row[1] is not None else 0.0
66:                   hasil[kategori] = jumlah
67:          return hasil
```

Observasi: Kelas ini mengimpor `Transaksi` dari `model.py` dan semua fungsi dari `database.py`. `__init__`-nya memanggil `database.setup_database_initial()` untuk memastikan tabel siap. Metode-metode lain fokus pada logika bisnis dan mendelegasikan operasi DB ke modul `database`.

Langkah 5: Aplikasi Utama Streamlit (`main_app.py`)

Tujuan: Membuat antarmuka pengguna web interaktif menggunakan Streamlit yang mengintegrasikan semua komponen backend.

```
001: # main_app.py
002: import streamlit as st
003: import datetime
004: import pandas as pd
005: import locale
006:
007: try: locale.setlocale(locale.LC_ALL, 'id_ID.UTF-8')
008: except locale.Error:
009:     try: locale.setlocale(locale.LC_ALL, 'Indonesian_Indonesia.1252')
010:     except: print("Locale id_ID/Indonesian tidak tersedia.")
011: def format_rp(angka):
012:     try: return locale.currency(angka or 0, grouping=True, symbol='Rp
')[:-3]
013:     except: return f"Rp {angka or 0:,.0f}".replace(",",".")
014:
015: try:
016:     from model import Transaksi
017:     from manajer_anggaran import AnggaranHarian
018:     from konfigurasi import KATEGORI_PENGELUARAN # Ambil list kategori
019: except ImportError as e:
020:    st.error(f"Gagal mengimpor modul: {e}. Pastikan file .py lain ada.")
```

```
021:     st.stop()
022:
023:  st.set_page_config(page_title="Catatan  Pengeluaran",  layout="wide",
initial_sidebar_state="expanded")
024:
025: # --- Inisialisasi Pengelola Anggaran (Gunakan Cache) ---
026: @st.cache_resource
027: def get_anggaran_manager():
028:           print(">>>  STREAMLIT:  (Cache  Resource)  Menginisialisasi
AnggaranHarian...")
029:     return AnggaranHarian() # Ini akan memicu cek DB/Tabel di __init__
030: anggaran = get_anggaran_manager()
031:
032: # --- Fungsi Halaman/UI ---
033: def halaman_input(anggaran: AnggaranHarian):
034:     st.header("💸 Tambah Pengeluaran Baru")
035:     with st.form("form_transaksi_baru", clear_on_submit=True):
036:         col1, col2 = st.columns([3, 1]);
037:                  with  col1:  deskripsi  =  st.text_input("Deskripsi*",
placeholder="Contoh: Makan siang");
038:                   with  col2:  kategori  =  st.selectbox("Kategori*:",
KATEGORI_PENGELUARAN, index=0);
039:         col3, col4 = st.columns([1, 1]);
040:                with  col3:  jumlah  =  st.number_input("Jumlah  (Rp)*:",
min_value=0.01, step=1000.0, format="%.0f", value=None, placeholder="Contoh:
25000");
041:                   with  col4:  tanggal  =  st.date_input("Tanggal*:",
value=datetime.date.today());
042:         submitted = st.form_submit_button("💾 Simpan Transaksi");
043:         if submitted:
044:            if not deskripsi: st.warning("Deskripsi wajib!", icon="⚠️")
045:               elif jumlah is None or jumlah <= 0: st.warning("Jumlah
wajib!", icon="⚠️")
046:             else:
047:                 with st.spinner("Menyimpan..."):
048:                     tx = Transaksi(deskripsi, float(jumlah), kategori,
tanggal)
049:                     if anggaran.tambah_transaksi(tx): st.success(f"OK!
Simpan.", icon="✅"); st.cache_data.clear(); st.rerun()
050:                     else: st.error("Gagal simpan.", icon="❌")
051:
052: def halaman_riwayat(anggaran: AnggaranHarian):
053:     st.subheader("Detail Semua Transaksi")
054:         if  st.button("🔄  Refresh  Riwayat"):  st.cache_data.clear();
st.rerun()
055:             with  st.spinner("Memuat  riwayat..."):  df_transaksi  =
anggaran.get_dataframe_transaksi()
056:     if df_transaksi is None: st.error("Gagal ambil riwayat.")
057:     elif df_transaksi.empty: st.info("Belum ada transaksi.")
058:           else:  st.dataframe(df_transaksi,  use_container_width=True,
hide_index=True)
059:
060: def halaman_ringkasan(anggaran: AnggaranHarian):
061:     st.subheader("Ringkasan Pengeluaran")
062:     col_filter1, col_filter2 = st.columns([1, 2])
063:     with col_filter1:
064:       pilihan_periode = st.selectbox("Filter Periode:", ["Semua Waktu",
"Hari  Ini",  "Pilih  Tanggal"],  key="filter_periode",  on_change=lambda:
st.cache_data.clear())
065:     tanggal_filter = None; label_periode = "(Semua Waktu)"
```

```
066:             if  pilihan_periode  ==  "Hari  Ini":  tanggal_filter  =
datetime.date.today();  label_periode  =  f"({tanggal_filter.strftime('%d
%b')})"
067:     elif pilihan_periode == "Pilih Tanggal Tertentu":
068:                 if  'tanggal_pilihan_state'  not  in  st.session_state:
st.session_state.tanggal_pilihan_state = datetime.date.today()
069:                     tanggal_filter  =  st.date_input("Pilih  Tanggal:",
value=st.session_state.tanggal_pilihan_state,        key="tanggal_pilihan",
on_change=lambda:    setattr(st.session_state,    'tanggal_pilihan_state',
st.session_state.tanggal_pilihan) or st.cache_data.clear())
070:         label_periode = f"({tanggal_filter.strftime('%d %b %Y')})"
071:
072:     with col_filter2:
073:         @st.cache_data(ttl=300) # Cache hasil total
074:                         def  hitung_total_cached(tgl_filter):  return
anggaran.hitung_total_pengeluaran(tanggal=tgl_filter)
075:         total_pengeluaran = hitung_total_cached(tanggal_filter)
076:                 st.metric(label=f"Total  Pengeluaran  {label_periode}",
value=format_rp(total_pengeluaran))
077:
078:     st.divider()
079:     st.subheader(f"Pengeluaran per Kategori {label_periode}")
080:     @st.cache_data(ttl=300) # Cache hasil kategori
081:                     def    get_kategori_cached(tgl_filter):    return
anggaran.get_pengeluaran_per_kategori(tanggal=tgl_filter)
082:     with st.spinner(f"Memuat ringkasan kategori..."): dict_per_kategori
= get_kategori_cached(tanggal_filter)
083:
084:      if not dict_per_kategori: st.info(f"Tidak ada data untuk periode
ini.")
085:     else:
086:         try:
087:              data_kategori = [{"Kategori": kat, "Total": jml} for kat,
jml in dict_per_kategori.items()]
088:                                                         df_kategori    =
pd.DataFrame(data_kategori).sort_values(by="Total",
ascending=False).reset_index(drop=True)
089:                                             df_kategori['Total   (Rp)']   =
df_kategori['Total'].apply(format_rp)
090:             col_kat1, col_kat2 = st.columns(2)
091:                                     with   col_kat1:   st.write("Tabel:");
st.dataframe(df_kategori[['Kategori',   'Total   (Rp)']],   hide_index=True,
use_container_width=True)
092:                                   with   col_kat2:   st.write("Grafik:");
st.bar_chart(df_kategori.set_index('Kategori')['Total'],
use_container_width=True)
093:          except Exception as e: st.error(f"Gagal tampilkan ringkasan:
{e}")
094:
095: # --- Fungsi Utama Aplikasi Streamlit ---
096: def main():
097:     st.sidebar.title("💰 Catatan Pengeluaran")
098:    menu_pilihan = st.sidebar.radio("Pilih Menu:", ["Tambah", "Riwayat",
"Ringkasan"], key="menu_utama")
099:     st.sidebar.markdown("---")
100:     st.sidebar.info("Jobsheet - Aplikasi Keuangan")
101:     manajer_anggaran = get_anggaran_manager()
102:     if menu_pilihan == "Tambah": halaman_input(manajer_anggaran)
103:     elif menu_pilihan == "Riwayat": halaman_riwayat(manajer_anggaran)
104:                           elif     menu_pilihan     ==     "Ringkasan":
halaman_ringkasan(manajer_anggaran)
```

```
105:     st.markdown("---"); st.caption("Pengembangan Aplikasi Berbasis OOP")
106:
107: if __name__ == "__main__":
108:       main() # Jalankan fungsi utama
```

Observasi:

- File ini sekarang menjadi inti dari aplikasi, mengimpor kelas dan fungsi dari modul lain.
- `@st.cache_resource` digunakan untuk `AnggaranHarian` agar tidak selalu membuat koneksi DB baru. `@st.cache_data` digunakan pada fungsi yang mengambil data dari `AnggaranHarian` untuk ditampilkan di UI, agar query DB tidak dijalankan terus menerus jika filter tidak berubah.
- UI dibagi menjadi beberapa fungsi halaman (`halaman_input`, `halaman_riwayat`, `halaman_ringkasan`).
- Navigasi menggunakan `st.sidebar.radio`.
- Setiap halaman memanggil metode yang sesuai dari instance `AnggaranHarian` untuk mendapatkan atau menyimpan data.

Langkah 6: Menjalankan dan Menguji Aplikasi Modular

Tujuan: Memastikan semua modul bekerja sama dengan baik dan aplikasi Streamlit berfungsi sesuai harapan.

- **Langkah:**
    1. Pastikan semua file (`konfigurasi.py`, `database.py`, `model.py`, `manajer_anggaran.py`, `streamlit_app.py`) berada dalam satu direktori.
    2. Pastikan database `pengeluaran_harian.db` sudah dibuat (jalankan `setup_db_pengeluaran.py` jika belum).
    3. Buka terminal di direktori tersebut.
    4. Jalankan aplikasi: `streamlit run streamlit_app.py`
    5. Lakukan pengujian menyeluruh seperti pada Praktikum 6 di versi non-modular sebelumnya:
        - Uji tambah data (valid dan invalid).
        - Uji lihat riwayat (pastikan data baru muncul setelah refresh/rerun).
        - Uji lihat total (pastikan terupdate).
        - Uji lihat ringkasan kategori (tabel dan grafik).
        - Uji filter tanggal pada ringkasan.
        - Periksa apakah ada error di terminal atau di halaman web.
    6. **(Opsional)** Periksa kembali isi file database `.db`.

Observasi: Aplikasi modular seharusnya berfungsi sama seperti versi gabungan, tetapi struktur kodenya lebih rapi dan terorganisir. Perhatikan pesan inisialisasi `AnggaranHarian` yang muncul di terminal (seharusnya hanya sekali karena cache).

E. Hasil Praktikum

Lengkapi hasil tabel praktikum berikut:

No.	Nama Praktikum	Hasil Praktikum
1	Langkah 1: Persiapan Awal (Konfigurasi & Setup Database)	
2	Langkah 2: Modul Akses Database (database.py)	
3	Langkah 3: Modul Model Data (model.py)	

4	Langkah 4: Modul Manajer Anggaran (manajer_anggaran.py)	
5	Langkah 5: Aplikasi Utama Streamlit (main_app.py)	
6	Langkah 6: Menjalankan dan Menguji Aplikasi Modular	

F. Penugasan

1. Kumpulkan Laporan Praktikum dari jobsheet ini dalam bentuk Microsoft word sesuai dengan format jobsheet praktikum dan dikumpulkan di web LMS. (JANGAN DALAM BENTUK PDF)
2. Kumpulkan luaran kode praktikum dalam bentuk ipynb yang sudah diunggah pada akun github masing-masing. Lampirkan tautan github yang sudah di unggah melalui laman LMS.
3. **Tambahkan kode Program** fungsionalitas Hapus Transaksi yang sudah dibuat.
   Langkah-langkah Pengembangan:
   1. **Backend (`manajer_anggaran.py`):**
   - Tambahkan metode baru `hapus_transaksi(self, id_transaksi: int) -> bool` pada kelas `AnggaranHarian`.
   - Metode ini harus menggunakan fungsi `database.execute_query()` untuk menjalankan perintah SQL `DELETE FROM transaksi WHERE id = ?` dengan `id_transaksi` yang diberikan.
   - Metode ini mengembalikan `True` jika penghapusan berhasil (query dieksekusi tanpa error SQLite), `False` jika gagal.
   2. **Frontend (`streamlit_app.py`):**
   - Modifikasi tampilan pada tab "Riwayat Lengkap". Anda perlu cara untuk memilih atau menargetkan transaksi yang ingin dihapus. Beberapa ide (pilih salah satu atau cari cara lain):
     - **Tombol per Baris:** Jika memungkinkan dengan `st.dataframe` atau library tabel lain (`streamlit-aggrid`), tambahkan kolom baru berisi tombol "Hapus" untuk setiap baris.
     - **Input ID & Tombol Hapus:** Tambahkan `st.number_input("ID Transaksi Hapus:", min_value=1, step=1)` dan tombol `st.button("Hapus Transaksi Terpilih")` di dekat tabel riwayat.
   - Implementasikan logika saat tombol Hapus ditekan:
     - Tampilkan konfirmasi kepada pengguna (misalnya menggunakan `st.warning` dan `st.button("Konfirmasi Hapus")`).
     - Jika dikonfirmasi, panggil metode `anggaran.hapus_transaksi(id_yang_dipilih)`.
     - Tampilkan pesan sukses atau gagal menggunakan `st.success` atau `st.error`.
     - Pastikan data pada tabel riwayat dan ringkasan diperbarui setelah penghapusan (gunakan `st.cache_data.clear()` dan `st.rerun()`).

G. Kesimpulan

Jobsheet ini telah membawa mahasiswa pada pengalaman mengintegrasikan berbagai teknologi dan konsep pemrograman modern untuk membangun aplikasi yang cukup kompleks, yaitu sistem presensi berbasis deteksi wajah. Melalui perancangan kelas-kelas OOP, mahasiswa belajar menstrukturkan komponen sistem seperti data pengguna, manajemen

database (dengan pola Singleton), dan logika pemrosesan utama. Penggunaan library eksternal seperti MediaPipe untuk computer vision (deteksi wajah) dan `sqlite3` untuk persistensi data menunjukkan bagaimana OOP dapat menjadi perekat yang mengelola interaksi antar modul berbeda. Meskipun aspek pengenalan wajah disimulasikan, project ini memberikan fondasi pemahaman tentang alur kerja sistem biometrik sederhana dan tantangan dalam mengintegrasikan input real-time (webcam), pemrosesan data (MediaPipe), penyimpanan data (SQLite), dan penanganan kesalahan (exception handling, logging) dalam satu kerangka kerja OOP yang kohesif dan terstruktur.

H. Daftar Pustaka

1. Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.
2. Guttag, J. V. (2016). Introduction to Computation and Programming Using Python (With Application to Understanding Data, 2nd ed.). MIT Press.
3. Python Software Foundation. Python 3 Documentation. Diakses dari https://docs.python.org/3/
4. Python Software Foundation. The Python Tutorial - Classes. Diakses dari https://docs.python.org/3/tutorial/classes.
5. Python Software Foundation. Built-in Exceptions. Diakses dari https://docs.python.org/3/library/exceptions.html
6. Python Software Foundation. abc — Abstract Base Classes. Diakses dari https://docs.python.org/3/library/abc.html
7. Das, B. N. (2018). Learn Object-Oriented Programming in Python. Packt Publishing.