PROJEK AKHIR
PEMROGRAMAN VISUAL
AIRAWARE - APLIKASI PEMANTAUAN KUALITAS UDARA

Disusun oleh:
Raizul Furkon (F1D022024)

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MATARAM
2024/2025

A. Deskripsi Aplikasi

AirAware merupakan sebuah aplikasi *dekstop* interaktif yang dikembangkan menggunakan bahasa *python* dengan berbasis pada PyQt5. Aplikasi ini dirancang untuk memberikan pengguna pengalaman baru dalam memantau *Air Quality Indeks* (AQI) atau kualitas udara di suatu tempat secara *realtime* karena memanfaatkan API eksternal. Data yang ditampilkan mencakup parameter polusi seperti PM2.5, PM10, CO, SO_2, NO_2, dan O_3, lengkap dengan warna indikator yang berubah sesuai dengan tingkat AQI. Pengguna dapat melakukan pencarian spesifik terhadap kota yang ingin diketahui AQI-nya. Selain itu aplikasi ini dapat menampilkan secara langsung daftar kota-kota di dunia lengkap dengan AQI saat itu juga. Aplikasi ini juga dapat melakukan ekspor data hasil pencarian ke CSV atau PDF untuk keperluan penelitian lebih lanjut.

B. Langkah Pembuatan Aplikasi

1. Merancang *User Interface*

   Pada aplikasi ini *User Interface* (UI) dirancang menggunakan aplikasi Qt Designer. Dengan menggunakan Qt Designer, komponen aplikasi seperti layout, widget, button, line edit, label sudah tersedia dan dapat langsung digunakan dengan cara *drag and drop*. Adapun tahapan dalam pembentukan UI pada aplikasi ini adalah sebagai berikut:

   a. Membuat Halaman Dashboard

      Pada halaman ini dirancang dengan menambahkan elemen-elemen seperti QLineEdit untuk input nama kota, QPushButton untuk tombol pencarian, serta beberapa QLabel untuk menampilkan informasi indeks kualitas udara (AQI) dari berbagai polutan (CO, NO_2, O_3, SO_2, PM2.5, PM10).

   b. Membuat Halaman All Places

      Pada halaman ini menampilkan data kualitas udara dari berbagai kota dalam bentuk tabel. Komponen yang digunakan antara lain QTableWidget yang dikonfigurasi agar dapat menampilkan data AQI secara baris dan kolom.

   c. Membuat Halaman History

      Halaman History untuk mencatat setiap pencarian yang dilakukan pengguna. Tabel riwayat pencarian menggunakan QTableWidget, dan dilengkapi dengan tombol ekspor seperti QPushButton untuk menyimpan data ke dalam format CSV atau PDF serta tombol *Delete* untuk menghapus *history* yang dipilih

   d. Membuat Window About App

      Jendela "About App" dibuat secara terpisah dalam file berbeda, yaitu desc_UI.ui. Jendela ini berisi QLabel untuk menampilkan deskripsi aplikasi dan tabel

QTableWidget yang menjelaskan kategori tingkat kualitas udara (AQI) dari level "Good" hingga "Hazardous

2. Membuat Logika Utama

   Pada tahapan ini, UI yang telah dibuat dikonversi menjadi *file* python menggunakan pyuic5 dan digunakan pada kelas utama sehingga dapat ditampilkan kepada pengguna. Selanjutnya membuat navigasi halaman dengan *button* pada *side bar* sehingga dapat beralih antar halaman. Beberapa fungsi utama dibuat pada bagian ini yaitu:

   a. Mengambil data dari API

      Proses mengambil data dari API yaitu menggunakan url API yang disediakan oleh pihak penyedia menggunakan *secret key* yang didapatkan dari login. Ketika pengguna mencari data pada QLineEdit maka aplikasi akan melakukan request data kepada API.

   b. Menampilkan hasil API

      Hasil dari *request* akan diterima dalam bentuk .json yang kemudian disinkronisasikan ke dalam QLabel untuk di*update* nilainya.

   c. Membuat *database*

      Hasil dari pencarian akan dimasukkan ke database yang bertipe Sqlite.

   d. Mengekspor data ke CSV atau PDF

      Hasil *database* yang telah terisi data dapat dikonversikan menjadi file CSV atau PDF.

3. Melakukan Pengujian

   Proses pengujian dilakukan untuk mengetahui apakah aplikasi telah berjalan lancar dengan tidak adanya *bug*, semua *button* telah berfungsi dengan lancar, API dapat digunakan mengambil data, dan menu bar dapat melakukan fungsinya.

C. Penjelasan Fungsi Utama

1. `get_aqi_data`

```
import requests
API_KEY = 'zVNMpUbzEkmGSl99fZA3QA==JqRGYA6DFqBJsjay'
def get_aqi_data(city):
    api_url                  =                  f'https://api.api-
ninjas.com/v1/airquality?city={city}'
    headers = {'X-Api-Key': API_KEY}
    response = requests.get(api_url, headers=headers)
    if response.status_code == requests.codes.ok:
        return response.json()
    else:
        return None
```

Kode di atas berfungsi untuk mengambil data AQI dari suatu kota menggunaakn API dari API ninja. Cara kerjanya adalah fungsi akan menerima parameter berupa nama kota dan kemudian membuat URL API berdasarkan nama kota tersebut. Selanjutnya kode akan mengirim permintaan GET ke API menggunakan request.get() yang berisi URL API dan headers yang berisi API_KEY agar sever mengenali siapa yang membuat permintaan. Jika permintaan berhasil maka akan mengembalikan hasil berupa data dalam bentuk JSON

2. `AQIworker`

```
from get_aqi_api import get_aqi_data
from PyQt5.QtCore import QObject, pyqtSignal
class AQIWorker(QObject):
    dataFetched = pyqtSignal(str, dict)
    finished = pyqtSignal()
    def __init__(self, city_list):
        super().__init__()
        self.city_list = city_list
    def run(self):
        for city in self.city_list:
            data = get_aqi_data(city)
            if data:
                self.dataFetched.emit(city, data)
        self.finished.emit()
```

Kode di atas berasal dari file AQIworker.py. file tersebut digunakan untuk mengambil data AQI dalam latar belakang. Pada file ini dipanggil pula fungsi get_aqi_api untuk mengambil data dari API. Ketika metode run() dijalankan, program akan melakukan iterasi ke setiap kota dalam daftar tersebut. Untuk setiap kota, data AQI akan diambil melalui fungsi get_aqi_data(city). Jika data berhasil diambil, maka sinyal dataFetched dikirim ke antarmuka utama dengan data tersebut. Setelah semua kota diproses, sinyal finished dikirim sebagai penanda bahwa pekerjaan selesai. Kode ini nantinya akan digunakan dan dijalankan dalam thread terpisah menggunakan QThread, sehingga proses pengambilan data dari internet tidak membekukan tampilan

aplikasi (UI tetap responsif) dan digunakan untuk mengambil data dari banyak kota sekaligus.

3. Databasemanager

```
import sqlite3
DB_NAME = "aqi_data.db"

class DatabaseManager:
    def __init__(self):
        self.conn = sqlite3.connect(DB_NAME)
    def create_table(self):
        cursor = self.conn.cursor()
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS history (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                city TEXT,
                co REAL,
                no2 REAL,
                o3 REAL,
                so2 REAL,
                pm25 REAL,
                pm10 REAL,
                overall_aqi INTEGER,
                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
            )
        """)
        self.conn.commit()
    def insert_history(self, city, data):
        cursor = self.conn.cursor()
        cursor.execute("""
            INSERT INTO history (city, co, no2, o3, so2, pm25, pm10,
overall_aqi)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            city,
            data["CO"]["aqi"],
            data["NO2"]["aqi"],
            data["O3"]["aqi"],
            data["SO2"]["aqi"],
            data["PM2.5"]["aqi"],
            data["PM10"]["aqi"],
            data["overall_aqi"]
        ))
        self.conn.commit()

    def fetch_all_history(self):
        cursor = self.conn.cursor()
        cursor.execute("""
            SELECT city, co, no2, o3, so2, pm25, pm10, overall_aqi,
timestamp
            FROM history
            ORDER BY id DESC
        """)
        return cursor.fetchall()
    def close(self):
        self.conn.close()
    def delete_history(self, city, timestamp):
        self.conn.execute("DELETE FROM history WHERE city = ? AND
timestamp = ?", (city, timestamp))
```

```
        self.conn.commit()
```

Kode di atas merupakan kelas yang digunakan untuk mengelola database. Database yang dibuat dalam bentuk SQLite dan bernama aqi_data.db. aat objek kelas ini dibuat, koneksi ke database langsung dibuka. Salah satu metode utamanya adalah create_table(), yang bertugas membuat tabel bernama history jika belum ada, dengan kolom-kolom seperti nama kota, nilai AQI untuk berbagai jenis polutan (CO, NO2, O3, SO2, PM2.5, PM10), nilai AQI keseluruhan, serta waktu pencatatan (timestamp) yang secara otomatis terisi saat data disimpan. Terdapat juga fungsi untuk memasukkan data ke database yaitu insert_history, fungsi mengambil seluruh data yaitu fetch_all_history, dan metode delete_history(city, timestamp) yang memungkinkan pengguna menghapus data riwayat tertentu berdasarkan nama kota dan waktu pencatatannya.

4. get_city_list

```
def get_city_list():
    city_list = [
            "London",
            "Paris",
            "Berlin",
             .
             .
             .
           ]
    return city_list
```

Kode di atas merupakan fungsi yang digunakan untuk menyimpan data berupa nama kota yang akan dilakukan AQI pencarian secara otomatis. Fungsi ini dibuat karena API tidak menyediakan endpoint untuk menampilkan keseluruhan data dalam API nya.

5. get_aqi_color

```
def get_aqi_color(aqi, as_hex=False):
    if aqi <= 50: color = QColor("green")
    elif aqi <= 100: color = QColor("yellow")
    elif aqi <= 150: color = QColor("orange")
    elif aqi <= 200: color = QColor("red")
    elif aqi <= 300: color = QColor("purple")
    else: color = QColor("maroon")
    return color.name() if as_hex else color
```

Kode di atas digunakan untuk menentukan warna yang sesuai berdasarkan nilai AQI agar dapat ditampilkan secara visual dalam antarmuka pengguna (misalnya mewarnai sel tabel atau label). Jika as_hex=True, maka fungsi akan mengembalikan kode warna dalam format heksadesimal (misalnya #ff0000), dan jika False, maka akan mengembalikan objek QColor langsung. Hal ini berguna karena dalam

pengaplikasiannya pada program utama, terdapat fungsi yang menerima heksadesimal yaitu pada stylesheets dan menerima QColor langsung.

6. connect_button

```
def connect_btn(self):
        self.ui.btn_page_1.clicked.connect(self.page_1)
        self.ui.btn_page_2.clicked.connect(self.page_2)
        self.ui.btn_page_3.clicked.connect(self.page_3)
        self.ui.btnSearch.clicked.connect(self.search_city_aqi)
        self.ui.actionExit.triggered.connect(self.closeWindow)

self.ui.actionRefresh.triggered.connect(self.refresh_program)

self.ui.btnDelete.clicked.connect(self.delete_selected_history)

self.ui.actionExport_to_CSV.triggered.connect(self.export_to_csv_
via_menu)

self.ui.actionExport_to_PDF.triggered.connect(self.export_to_pdf_
via_menu)
        self.ui.btnSaveAs.clicked.connect(self.save_as)

self.ui.actionAbout.triggered.connect(self.show_description)
```

Kode di atas berfungsi untuk menghubungkan tombol-tombol dan menu aksi pada antarmuka pengguna (UI) dengan fungsi-fungsi yang akan dijalankan saat tombol tersebut diklik atau menu dipilih.

7. search_city_aqi

```
def search_city_aqi(self):

        city = self.ui.searchBox.text()
        if not city:
            QMessageBox.warning(
                self,
                "Input Kosong",
                "Silakan masukkan nama kota terlebih dahulu."
            )
            return

        data = get_aqi_data(city)
        if data:
            self.ui.labelPlace.setText(city.capitalize())


self.ui.co_cons.setText(str(data['CO']['concentration']))
            self.ui.co_aqi.setText(str(data['CO']['aqi']))
            self.ui.co_aqi.setStyleSheet(f"color:
{get_aqi_color(data['CO']['aqi'], as_hex=True)}")


self.ui.no2_cons.setText(str(data['NO2']['concentration']))
            self.ui.no2_aqi.setText(str(data['NO2']['aqi']))
            self.ui.no2_aqi.setStyleSheet(f"color:
{get_aqi_color(data['NO2']['aqi'], as_hex=True)}")


self.ui.o3_cons.setText(str(data['O3']['concentration']))
```

```
            self.ui.o3_aqi.setText(str(data['O3']['aqi']))
            self.ui.o3_aqi.setStyleSheet(f"color:
{get_aqi_color(data['O3']['aqi'], as_hex=True)}")


self.ui.so_cons.setText(str(data['SO2']['concentration']))
            self.ui.so2_aqi.setText(str(data['SO2']['aqi']))
            self.ui.so2_aqi.setStyleSheet(f"color:
{get_aqi_color(data['SO2']['aqi'], as_hex=True)}")


self.ui.pm25_cons.setText(str(data['PM2.5']['concentration']))
            self.ui.pm25_aqi.setText(str(data['PM2.5']['aqi']))
            self.ui.pm25_aqi.setStyleSheet(f"color:
{get_aqi_color(data['PM2.5']['aqi'], as_hex=True)}")


self.ui.pm10_cons.setText(str(data['PM10']['concentration']))
            self.ui.pm10_aqi.setText(str(data['PM10']['aqi']))
            self.ui.pm10_aqi.setStyleSheet(f"color:
{get_aqi_color(data['PM10']['aqi'], as_hex=True)}")

            overall_aqi = data['overall_aqi']
            self.ui.overall_aqi_value.setText(str(overall_aqi))
            self.ui.overall_aqi_value.setStyleSheet(f"color:
{get_aqi_color(overall_aqi, as_hex=True)}")

            if overall_aqi <= 50:
                status = "Good"
            elif overall_aqi <= 100:
                status = "Moderate"
            elif overall_aqi <= 150:
                status = "Unhealthy for Sensitive Groups"
            elif overall_aqi <= 200:
                status = "Unhealthy"
            elif overall_aqi <= 300:
                status = "Very Unhealthy"
            else:
                status = "Hazardous"
            self.ui.status_aqi.setText(status)
            self.ui.status_aqi.setStyleSheet(f"color:
{get_aqi_color(overall_aqi, as_hex=True)}")
            self.db.insert_history(city, data)
        else:
            QMessageBox.warning(
                self,
                "Kota Tidak Ditemukan",
                f"Kota '{city}' tidak ditemukan atau tidak tersedia
dalam data AQI."
            )

            self.ui.labelPlace.setText("-")
            self.ui.status_aqi.setText("-")
            self.ui.overall_aqi_value.setText("-")
            self.ui.co_cons.setText("-")
            self.ui.co_aqi.setText("-")
            self.ui.no2_cons.setText("-")
            self.ui.no2_aqi.setText("-")
            self.ui.o3_cons.setText("-")
            self.ui.o3_aqi.setText("-")
            self.ui.so_cons.setText("-")
```

```
            self.ui.so2_aqi.setText("-")
            self.ui.pm25_cons.setText("-")
            self.ui.pm25_aqi.setText("-")
            self.ui.pm10_cons.setText("-")
            self.ui.pm10_aqi.setText("-")
```

Kode tersebut merupakan method yang digunakan untuk melakukan fitur pencarian AQI dalam aplikasi ini. Fungsi ini menangani proses pengambilan dan penampilan data kualitas udara berdasarkan nama kota yang dimasukkan pengguna ke dalam kotak pencarian.

Saat tombol cari diklik, fungsi akan mengambil teks dari searchBox. Jika kotak pencarian kosong, maka akan muncul pesan peringatan (QMessageBox) yang meminta pengguna memasukkan nama kota. Namun jika ada input dan input benar, aplikasi akan memanggil fungsi get_aqi_data(city) untuk mengambil data kualitas udara kota tersebut melalui API. Adapun yang dilakukan selanjutnya adalah:

a. Nama kota ditampilkan sebagai judulnya.
b. Data konsentrasi dan nilai AQI untuk setiap polutan (CO, NO2, O3, SO2, PM2.5, PM10) ditampilkan di masing-masing label.
c. Warna teks AQI setiap polutan diubah sesuai tingkat bahayanya menggunakan fungsi get_aqi_color().
d. Nilai AQI keseluruhan (overall_aqi) ditampilkan dan diberi warna yang sesuai.
e. Status kualitas udara (seperti “Good”, “Moderate”, hingga “Hazardous”) ditentukan berdasarkan nilai overall_aqi, lalu ditampilkan di label status_aqi dengan warna yang juga disesuaikan.
f. Data pencarian kota dan hasilnya disimpan ke database menggunakan self.db.insert_history(city, data).

Apabila kota telah diisi namun tidak terdapat di database, bisa karena data tidak ada atau kesalahan pengetikkan, maka kan muncul peringatan bahwa kota tidak tersedia. Selain itu, semua label hasil pencarian akan direset ke simbol "-" agar antarmuka tidak menampilkan informasi yang salah.

8. load_all_city

```
def load_all_city(self):
        self.city_list = get_city_list()
        self.worker_thread = QThread()
        self.worker = AQIWorker(self.city_list)
        self.worker.moveToThread(self.worker_thread)
        self.worker_thread.started.connect(self.worker.run)
        self.worker.dataFetched.connect(self.add_row_to_table)
        self.worker.finished.connect(self.worker_thread.quit)
        self.worker.finished.connect(self.worker.deleteLater)
```

```
self.worker_thread.finished.connect(self.worker_thread.deleteLate
r)
        self.worker_thread.start()
```

Kode di atas digunakan untuk memuat data kualitas udara (AQI) dari banyak kota secara bersamaan tanpa mengganggu antarmuka pengguna dengan memanfaatkan *threading* di PyQt5. Fungsi ini mengambil daftar kota melalui get_city_list(), lalu membuat objek AQIWorker yang akan berjalan di thread terpisah (QThread). Saat thread dimulai, fungsi run() akan dipanggil untuk mengambil data AQI setiap kota. Setiap data yang berhasil diambil akan langsung ditampilkan ke tabel melalui sinyal dataFetched, dan setelah semua kota selesai diproses, thread akan dihentikan dan sumber daya dibersihkan. Dengan cara ini, proses pengambilan data dilakukan secara asynchronous.

9. `add_row_to_table`

```
def add_row_to_table(self, city, data):
        row = self.ui.tableWidget.rowCount()
        self.ui.tableWidget.insertRow(row)

        def add_item(col, val):
            item = QTableWidgetItem(str(val))
            item.setForeground(QBrush(get_aqi_color(float(val))))
            self.ui.tableWidget.setItem(row, col, item)

        self.ui.tableWidget.setItem(row,                          0,
QTableWidgetItem(city.capitalize()))
        add_item(1, data['CO']['aqi'])
        add_item(2, data['NO2']['aqi'])
        add_item(3, data['O3']['aqi'])
        add_item(4, data['SO2']['aqi'])
        add_item(5, data['PM2.5']['aqi'])
        add_item(6, data['PM10']['aqi'])
        add_item(7, data['overall_aqi'])
```

Kode di atas berfungsi untuk menambahkan satu baris data kualitas udara (AQI) dari suatu kota ke dalam QTableWidget pada aplikasi. Pertama, fungsi menentukan jumlah baris saat ini dan menambahkan baris baru ke tabel. Fungsi lokal add_item(col, val) digunakan untuk menyisipkan data ke kolom tertentu dengan memberi pewarnaan teks berdasarkan nilai AQI menggunakan get_aqi_color(). Nama kota ditampilkan di kolom pertama, kemudian nilai AQI dari masing-masing polutan (CO, NO2, O3, SO2, PM2.5, PM10), serta nilai AQI keseluruhan (overall_aqi) ditambahkan ke kolom-kolom berikutnya. Dengan cara ini, data kota ditampilkan secara rapi dan informatif dalam tabel, termasuk pewarnaan sesuai tingkat polusi.

10. `load_history`

```
def load_history(self):
        rows = self.db.fetch_all_history()
```

```
        self.ui.tableHistory.setRowCount(0)

        for row_num, row_data in enumerate(rows):
            self.ui.tableHistory.insertRow(row_num)
            for col_num, col_data in enumerate(row_data):
                item = QTableWidgetItem(str(col_data))

                if 1 <= col_num <= 7:
                    try:
                        aqi_val = float(col_data)

item.setForeground(QBrush(get_aqi_color(aqi_val)))
                    except ValueError:
                        pass

                self.ui.tableHistory.setItem(row_num,     col_num,
item)
```

Kode di atas digunakan untuk menampilkan riwayat data kualitas udara (AQI) yang telah disimpan sebelumnya di database ke dalam tabel tableHistory. Fungsi ini pertama-tama mengambil seluruh data riwayat dari database menggunakan fetch_all_history() lalu mengosongkan isi tabel. Selanjutnya, fungsi melakukan iterasi terhadap setiap baris data dan menambahkannya ke tabel. Untuk kolom yang berisi nilai AQI (kolom 1 sampai 7), nilai tersebut diwarnai menggunakan get_aqi_color() agar tampilannya mencerminkan tingkat polusi udara. Pewarnaan ini dilakukan dengan setForeground() sehingga pengguna dapat dengan mudah mengidentifikasi tingkat bahaya udara dari warna yang ditampilkan.

11. delete_selected_history

```
def delete_selected_history(self):
        selected_row = self.ui.tableHistory.currentRow()
        if selected_row == -1:
            QMessageBox.warning(self, "Tidak Ada Data", "Silakan
pilih salah satu data yang ingin dihapus.")
            return

        city_item = self.ui.tableHistory.item(selected_row, 0)
        timestamp_item  =  self.ui.tableHistory.item(selected_row,
8)

        if not city_item or not timestamp_item:
            QMessageBox.warning(self, "Data Tidak Lengkap", "Data
yang dipilih tidak valid.")
            return

        city = city_item.text()
        timestamp = timestamp_item.text()

        confirm = QMessageBox.question(
            self,
            "Konfirmasi Hapus",
            f"Apakah Anda yakin ingin menghapus data riwayat untuk
kota '{city}' pada '{timestamp}'?",
            QMessageBox.Yes | QMessageBox.No,
            QMessageBox.No
```

```
        )

        if confirm == QMessageBox.Yes:
            self.db.delete_history(city, timestamp)
            self.ui.tableHistory.removeRow(selected_row)
            QMessageBox.information(self, "Sukses", "Data berhasil
dihapus.")
```

Kode di atas berfungsi untuk menghapus satu baris entri dari database riwayat data AQI. Fungsinya pertama-tama mengecek apakah ada baris yang dipilih, jika tidak ada maka akan menampilkan pesan error. Jika ada baris yang dipilih maka aplikasi akan memunculkan dialog konfirmasi. Jika pengguna setuju maka data yang dipilih akan dihapus dari database.

12. `export_to_csv`

```
def export_to_csv(self, path_csv):
        try:
            with     open(path_csv,     mode='w',     newline='',
encoding='utf-8') as file:
                writer = csv.writer(file)

                headers = []
                for                        i                        in
range(self.ui.tableHistory.columnCount()):
                    header_item                                      =
self.ui.tableHistory.horizontalHeaderItem(i)
                    headers.append(header_item.text()            if
header_item else f"Kolom {i+1}")
                writer.writerow(headers)

                for row in range(self.ui.tableHistory.rowCount()):
                    row_data = []
                    for                    col                    in
range(self.ui.tableHistory.columnCount()):
                        item = self.ui.tableHistory.item(row, col)
                        row_data.append(item.text()  if  item  else
"")
                    writer.writerow(row_data)

            QMessageBox.information(self,     "Sukses",     f"Data
berhasil diekspor ke CSV:\n{path_csv}")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Gagal mengekspor
CSV:\n{str(e)}")
```

Kode di atas digunakan untuk mengekspor seluruh data dari tabel riwayat (tableHistory) ke dalam file CSV yang disimpan di lokasi path_csv. Fungsi ini membuka file dalam mode tulis dengan encoding UTF-8, lalu menggunakan csv.writer untuk menuliskan data. Pertama, fungsi mengambil dan menuliskan nama-nama kolom sebagai baris header. Setelah itu, setiap baris data dalam tabel dibaca dan dituliskan ke file CSV secara berurutan.

13. export_to_pdf

```
def export_to_pdf(self, path_pdf):
        try:
            c = canvas.Canvas(path_pdf, pagesize=A4)
            _, height = A4
            y = height - 50

            headers = []
            for i in range(self.ui.tableHistory.columnCount()):
                item                                                    =
self.ui.tableHistory.horizontalHeaderItem(i)
                headers.append(item.text()  if  item  else  f"Kolom
{i+1}")

            x = 40
            for i, header in enumerate(headers):
                c.drawString(x + i * 65, y, header)
            y -= 20

            for row in range(self.ui.tableHistory.rowCount()):
                for                     col                     in
range(self.ui.tableHistory.columnCount()):
                    item = self.ui.tableHistory.item(row, col)
                    if item:
                        c.drawString(x + col * 65, y, item.text())
                y -= 20
                if y < 50:
                    c.showPage()
                    y = height - 50

            c.save()
            QMessageBox.information(self,    "Berhasil",    f"Data
berhasil diekspor ke PDF.\n{path_pdf}")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Gagal mengekspor
PDF:\n{str(e)}")
```

Kode di atas digunakan untuk mengekspor data riwayat kualitas udara dari tableHistory ke dalam file PDF dengan menggunakan pustaka reportlab. Fungsi ini pertama-tama membuat kanvas PDF dengan ukuran halaman A4, lalu menentukan posisi awal teks di bagian atas halaman. Nama-nama kolom diambil dan dituliskan sebagai header di bagian atas tabel PDF. Setelah itu, fungsi membaca isi tabel baris demi baris dan mencetak setiap nilai ke dalam posisi yang sesuai di PDF. Jika posisi y mendekati batas bawah halaman, halaman baru akan dibuat secara otomatis. Setelah seluruh data dituliskan, file PDF disimpan

14. save_as

```
def save_as(self):
        path, selected_filter = QFileDialog.getSaveFileName(
            self,
            "Simpan Sebagai",
            "",
            "PDF Files (*.pdf);;CSV Files (*.csv)"
        )
```

```
        if path:
            if selected_filter == "CSV Files (*.csv)" or
path.endswith(".csv"):
                self.export_to_csv(path)
            elif selected_filter == "PDF Files (*.pdf)" or
path.endswith(".pdf"):
                self.export_to_pdf(path)
            else:
                QMessageBox.warning(self, "Format Tidak Didukung",
"Hanya mendukung file PDF atau CSV.")
```

Kode di atas memungkinkan pengguna untuk menyimpan data riwayat dalam format CSV atau PDF melalui dialog penyimpanan file. Fungsi ini menampilkan QFileDialog.getSaveFileName() yang memungkinkan pengguna memilih lokasi penyimpanan dan format file. Setelah pengguna memilih file dan format, fungsi memeriksa ekstensi file atau filter yang dipilih. Jika pengguna memilih CSV atau nama file berakhiran .csv, maka fungsi export_to_csv() akan dipanggil. Jika memilih PDF atau nama file berakhiran .pdf, maka fungsi export_to_pdf() dijalankan. Jika format file tidak sesuai, akan ditampilkan peringatan bahwa hanya format PDF dan CSV yang didukung.

15. `export_to_csv_via_menu`

```
def export_to_csv_via_menu(self):
        path_csv, _ = QtWidgets.QFileDialog.getSaveFileName(self,
"Simpan CSV", "", "CSV Files (*.csv)")
        if path_csv:
            self.export_to_csv(path_csv)
```

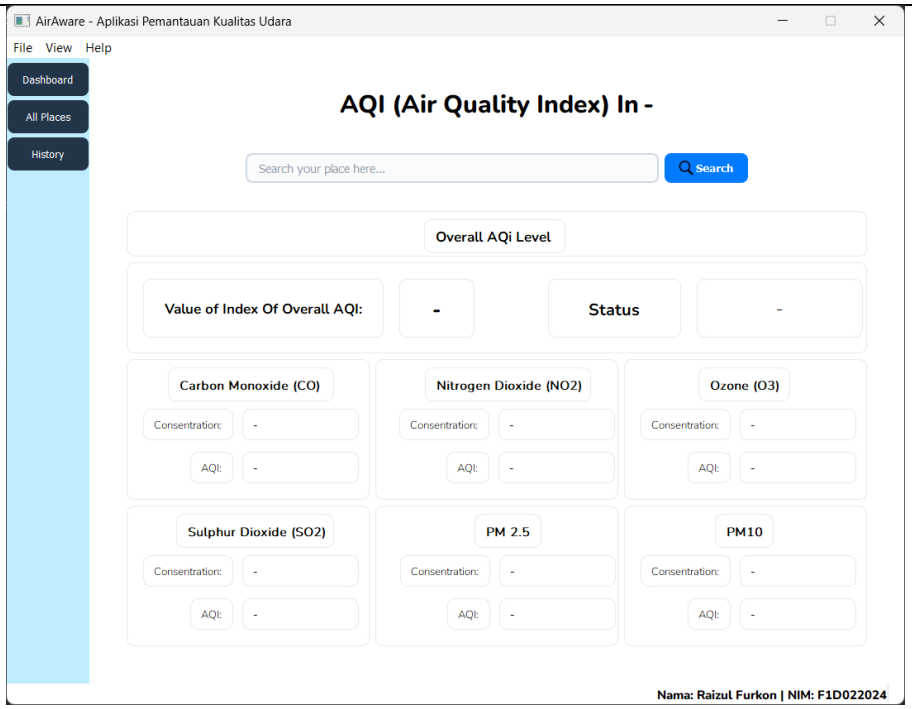
Kode di atas digunakan untuk mengekspor data riwayat ke file CSV melalui opsi menu aplikasi.

16. `export_to_pdf_via_menu`

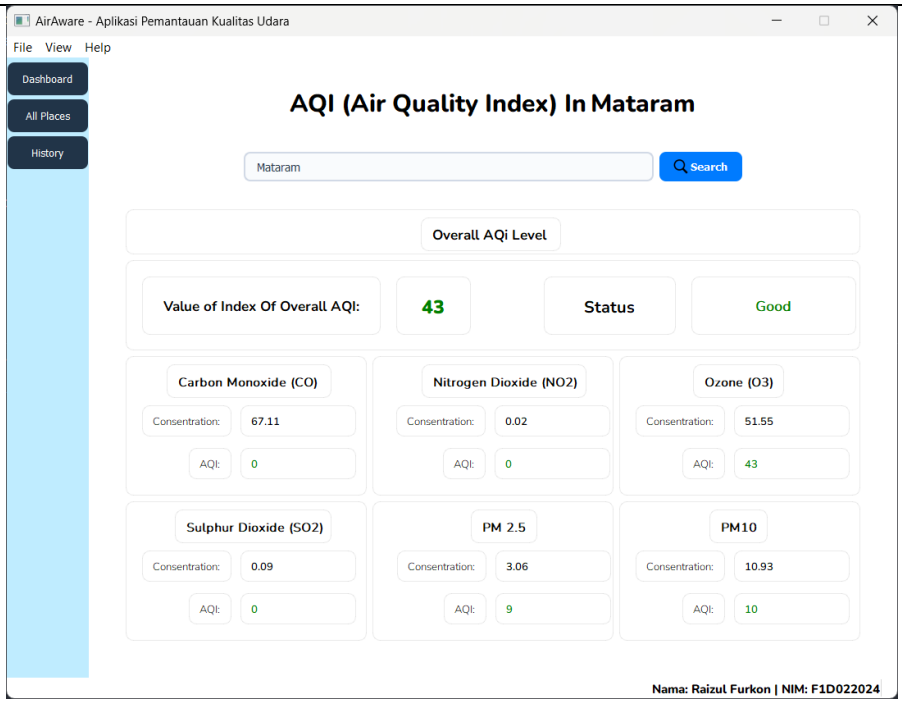
```
def export_to_pdf_via_menu(self):
        path_pdf, _ = QtWidgets.QFileDialog.getSaveFileName(self,
"Simpan PDF", "", "PDF Files (*.pdf)")
        if path_pdf:
            self.export_to_pdf(path_pdf)
```

Kode di atas digunakan untuk mengekspor data ke dalam format PDF melalui menu aplikasi.

D. Hasil Aplikasi (*Screenshot*)

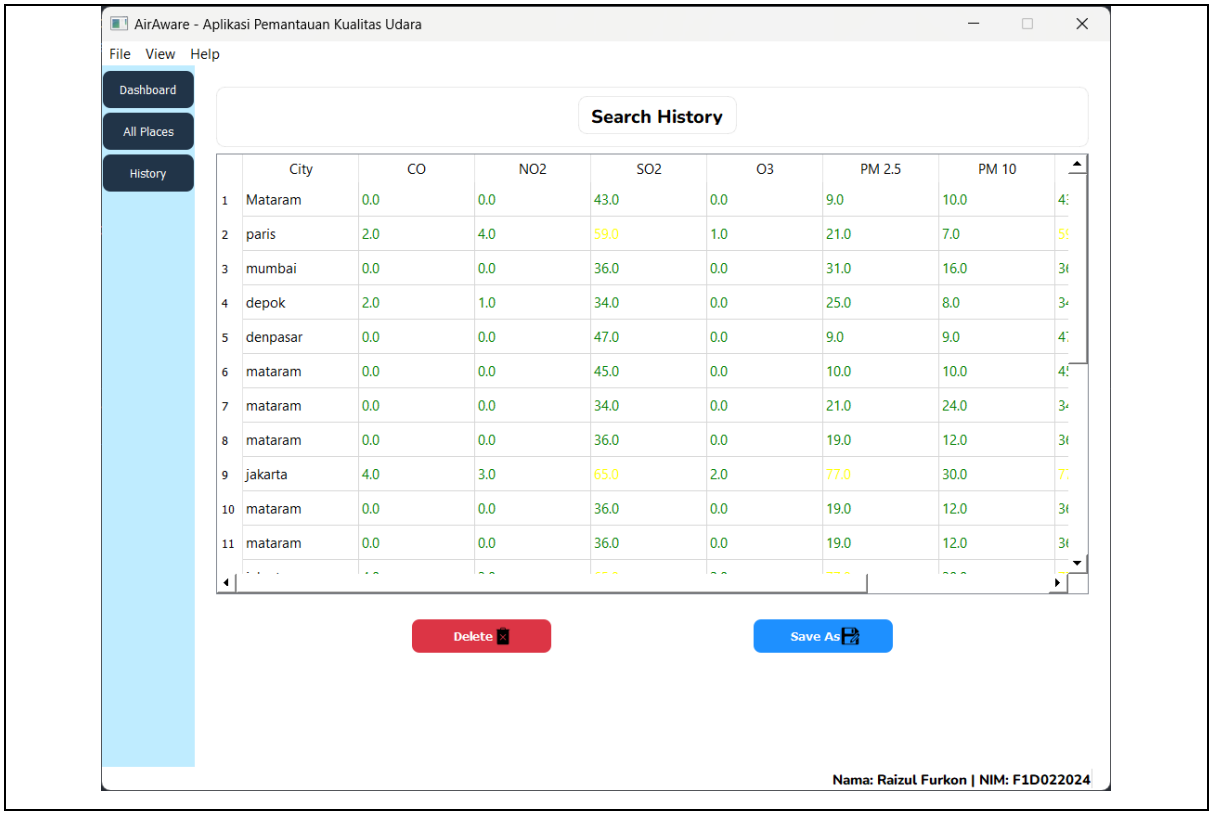


Gambar 1. Tampilan Dashboard Aplikasi

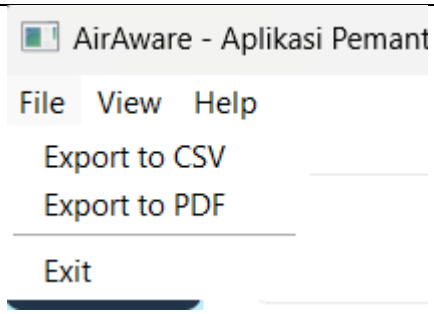


Gambar 2. Tampilan Aplikasi Saat Menampilkan Data AQI di Mataram

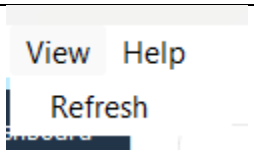
Gambar 3. Tampilan Halaman All Places



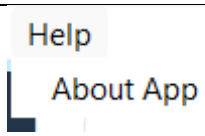
Gambar 4. Tampilan Halaman Search History



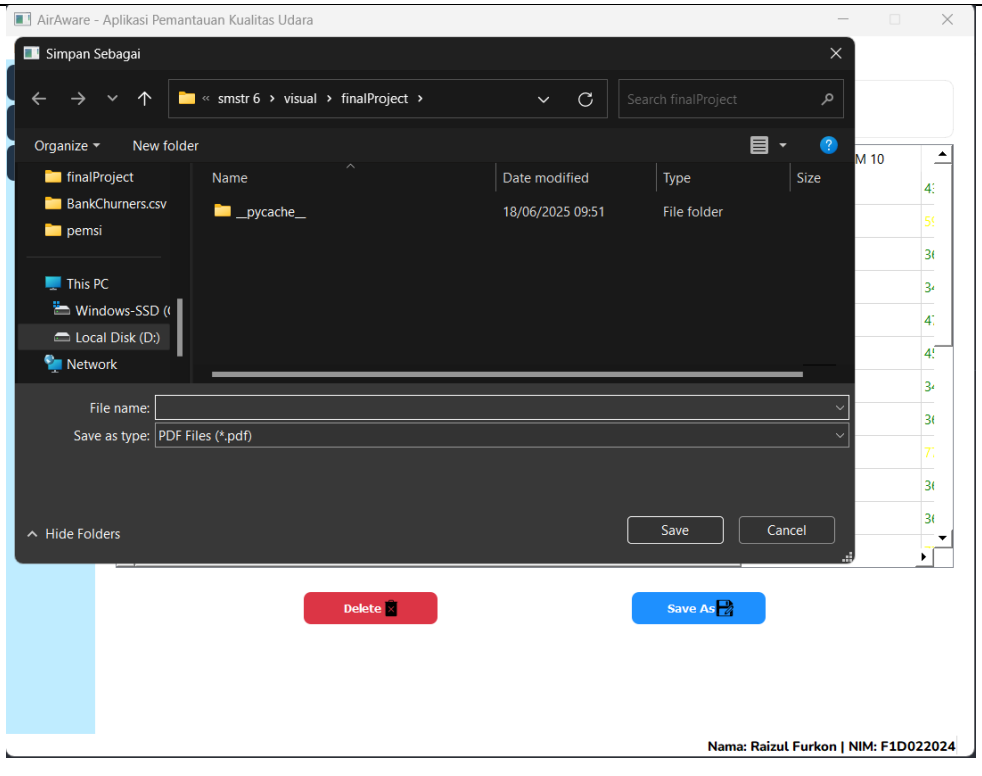
Gambar 5. Tampilan Menu Bar File



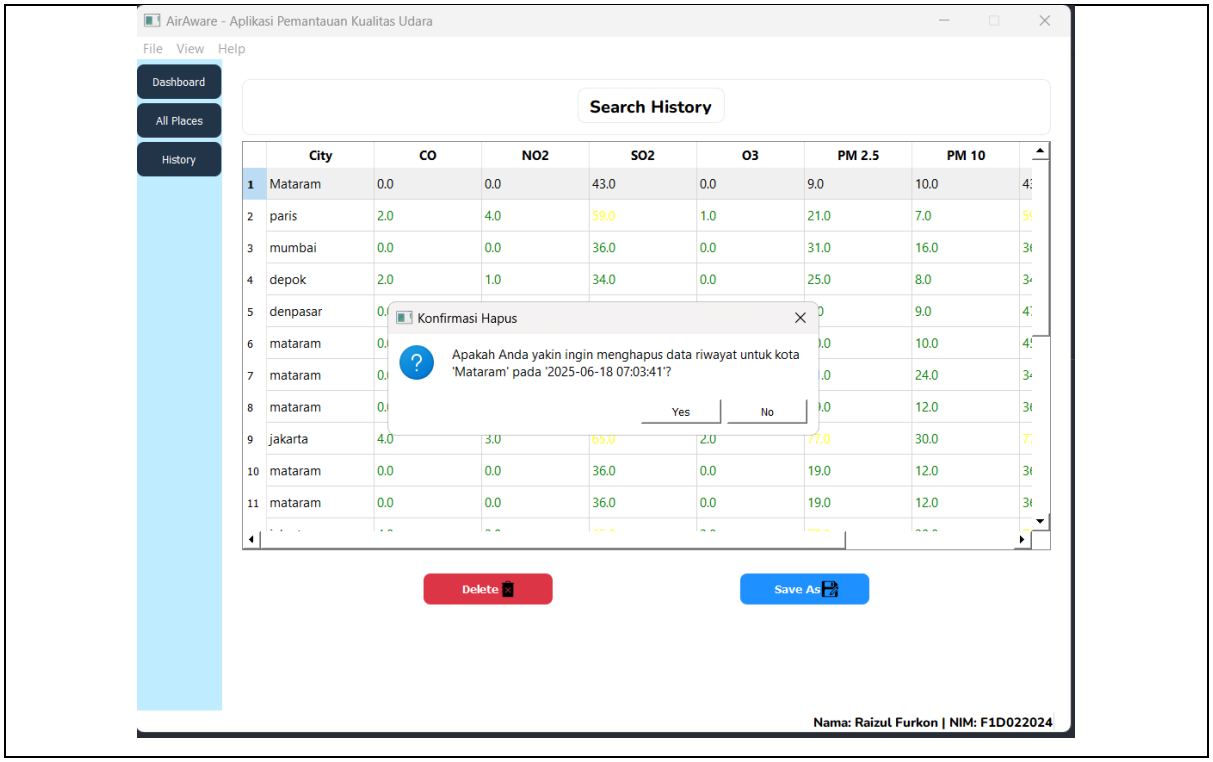
Gambar 5. Tampilan Menu Bar View



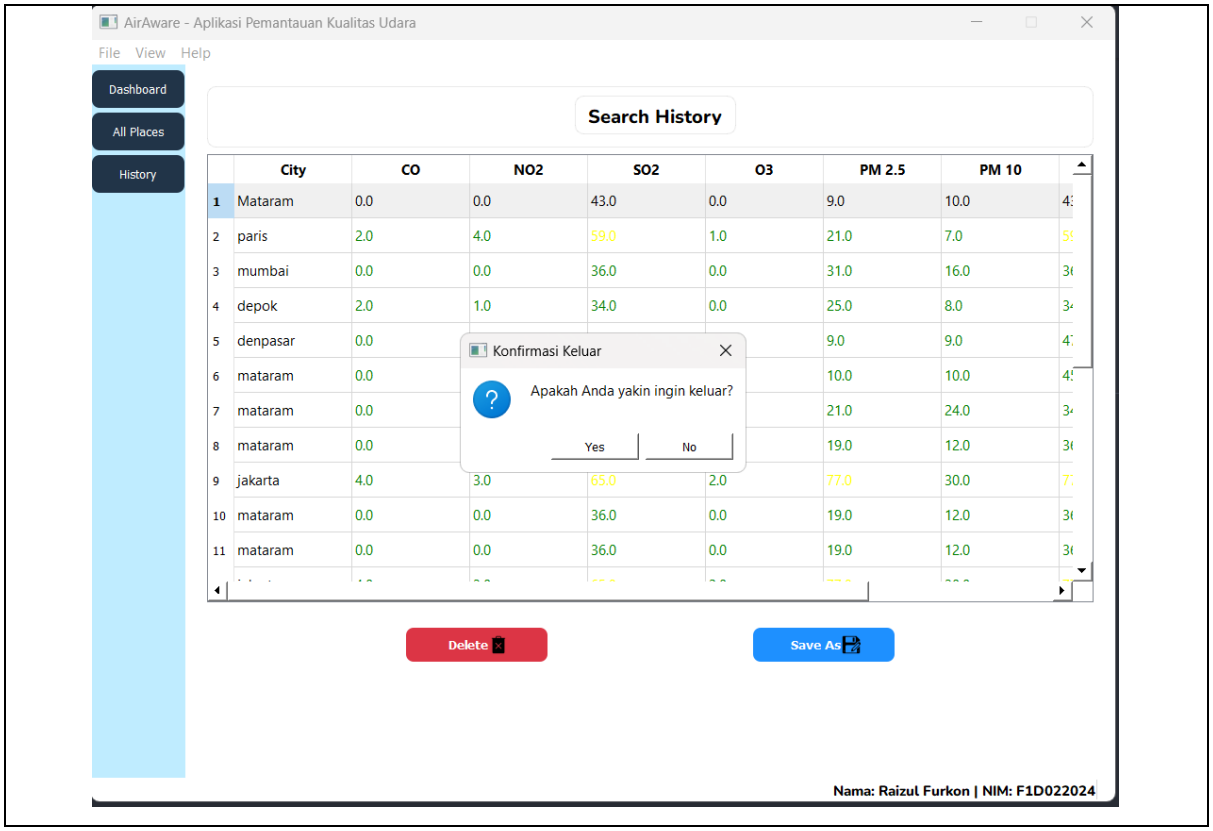
Gambar 6. Tampilan Menu Bar Help



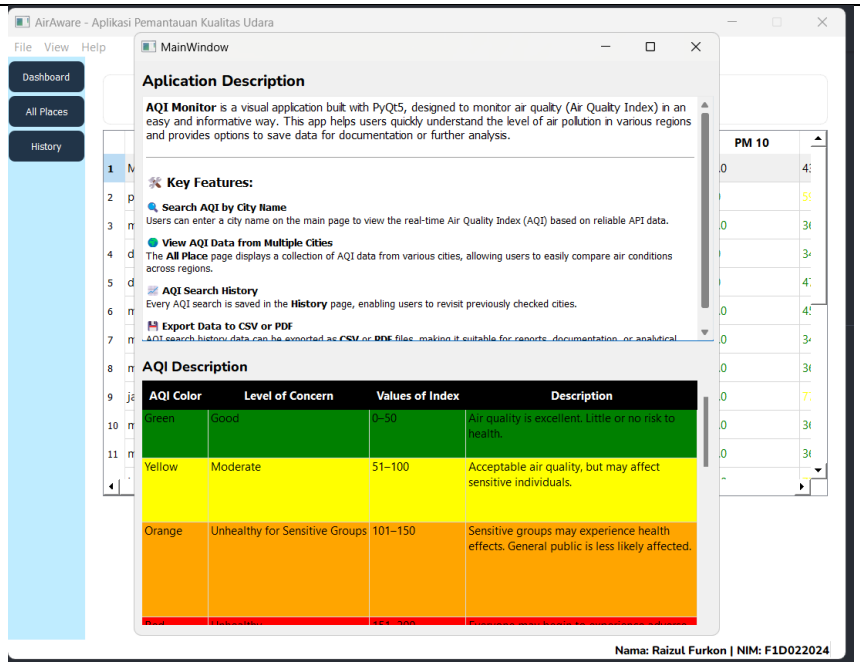
Gambar 7. Tampilan Saat Save History ke CSV atau PDF



Gambar 8. Tampilan Konfirmasi Delete Data History



Gambar 9. Tampilan Konfirmasi Keluar Aplikasi



Gambar 10. Tampilan Window About App