LAPORAN

TUGAS AKHIR

Mata Kuliah Pemrograman Berorientasi Objek

Disusun Oleh:

1. Alfin Afriza M (2213020173)
2. Mohammad Ubaidillah Ridlo (2213020215)
3. Raya Osgibran (2213020139)

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK DAN ILMU KOMPUTER
UNIVERSITAS NUSANTARA PGRI KEDIRI

TAHUN 2023

Kata Pengantar

Dengan rasa syukur dan penuh kegembiraan, penulis menyampaikan puji syukur kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya yang telah melimpah selama proses pembuatan tugas akhir ini. Tugas akhir ini merupakan hasil perjuangan dan dedikasi penulis dalam mengembangkan aplikasi sederhana berbasis PyQt ini yang bertujuan untuk memudahkan manajemen acara.

Tidak lupa, penulis mengucapkan terima kasih kepada dosen pembimbing, teman-teman sejawat, dan semua pihak yang telah memberikan dukungan, bimbingan, serta masukan konstruktif selama perjalanan pembuatan tugas akhir ini.

Pada kesempatan ini, penulis ingin menegaskan bahwa tugas akhir ini bukanlah hasil akhir yang sempurna. Oleh karena itu, setiap kritik, saran, dan masukan membangun dari pembaca akan sangat dihargai untuk pengembangan selanjutnya.

Semoga tugas akhir ini dapat memberikan manfaat, inspirasi, dan kontribusi positif terhadap dunia pemrograman berorientasi objek, khususnya dalam pengembangan aplikasi menggunakan PyQt.

Akhir kata, penulis berharap semoga tugas akhir ini dapat menjadi pijakan awal untuk mengeksplorasi dan mengembangkan lebih lanjut di masa mendatang. Terima kasih.

Daftar Isi

Kata Pengantar..... i
Daftar Isi.....…... ii

BAB I PENDAHULUAN……. 1
1.1 Latar Belakang **1**

BAB II PEMBAHASAN…….. 2
2.1 Flowchart Sistem……..... **2**
2.2 Class Diagram..... **5**
2.3 Source code……..... **7**
2.4 Hasil Program Dan Penjelasan…….. 27

BAB III PENUTUP……... 29
3.1 Kesimpulan..... **29**

Daftar Pustaka…….. 30

BAB I
PENDAHULUAN

1.1 Latar Belakang

Dalam era perkembangan teknologi informasi yang begitu pesat, penggunaan aplikasi perangkat lunak semakin meluas dan mendalam ke berbagai sektor kehidupan. Seiring dengan itu, tuntutan akan aplikasi yang efisien, user-friendly, dan dapat memenuhi kebutuhan pengguna menjadi suatu keharusan. Dalam konteks tersebut, pemrograman berorientasi objek (PBO) menjadi pendekatan yang relevan dan banyak digunakan untuk merancang aplikasi yang modular dan mudah dimengerti.

Dalam mata kuliah pemrograman berorientasi objek, penulis merasa perlu untuk mengembangkan suatu aplikasi sebagai bentuk implementasi dari konsep-konsep yang telah dipelajari. Dalam hal ini, pemilihan PyQt sebagai framework pengembangan aplikasi GUI (Graphical User Interface) memberikan keleluasaan dan kemudahan dalam merancang antarmuka pengguna yang menarik.

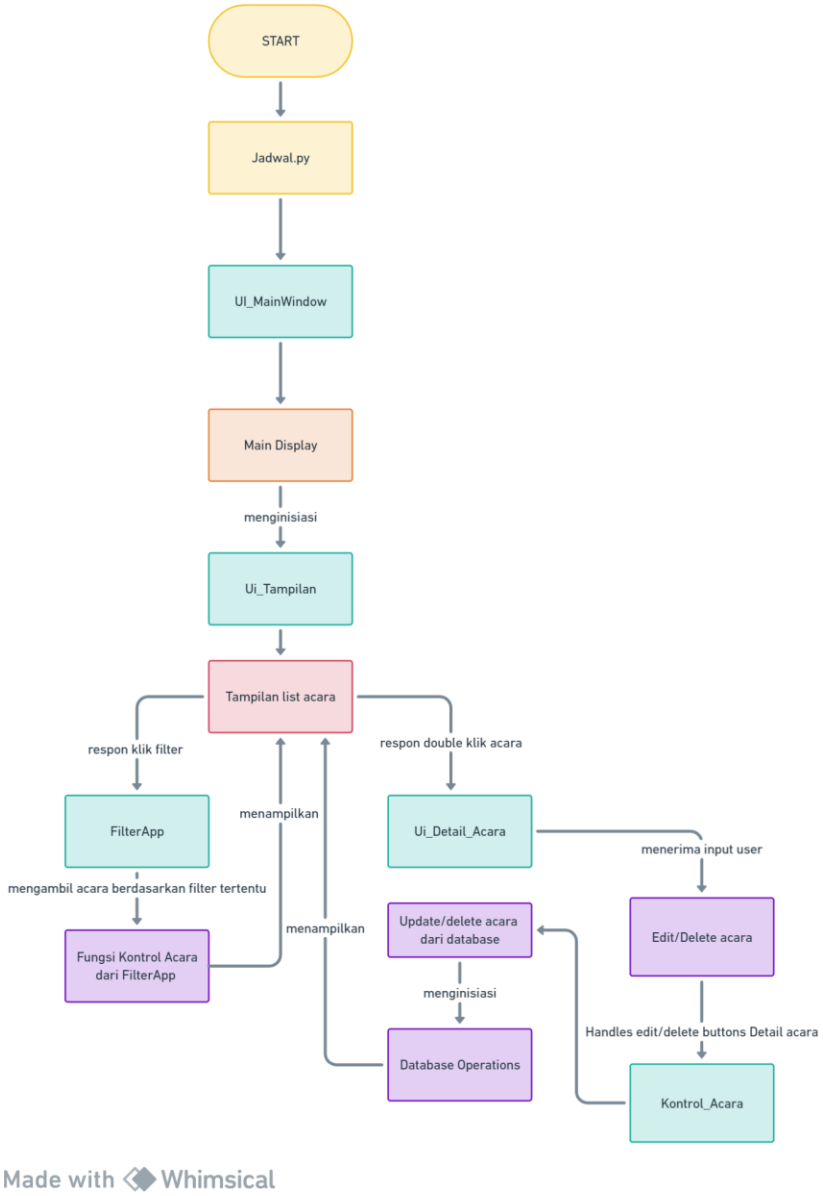
Latar belakang pembuatan aplikasi ini didasarkan pada kebutuhan untuk menyederhanakan manajemen acara, baik acara pribadi maupun acara di tingkat organisasi. Dengan adanya aplikasi ini, diharapkan pengguna dapat dengan mudah melihat, mengedit, dan menghapus informasi terkait acara tanpa harus bersusah payah melakukan proses manual yang cenderung rumit dan memakan waktu.

Melalui tugas akhir ini, penulis berharap dapat memberikan kontribusi kecil dalam pengembangan aplikasi berbasis PyQt dan membuka peluang untuk eksplorasi lebih lanjut terkait pemrograman berorientasi objek. Selain itu, aplikasi ini juga diharapkan dapat memberikan manfaat nyata bagi pengguna dalam memudahkan pengelolaan jadwal acara sehari-hari.

Dengan latar belakang tersebut, tugas akhir ini bertujuan untuk memberikan solusi praktis melalui implementasi konsep-konsep PBO dalam pembuatan aplikasi manajemen acara berbasis desktop.

BAB II
PEMBAHASAN

2.1 Flowchart Sistem

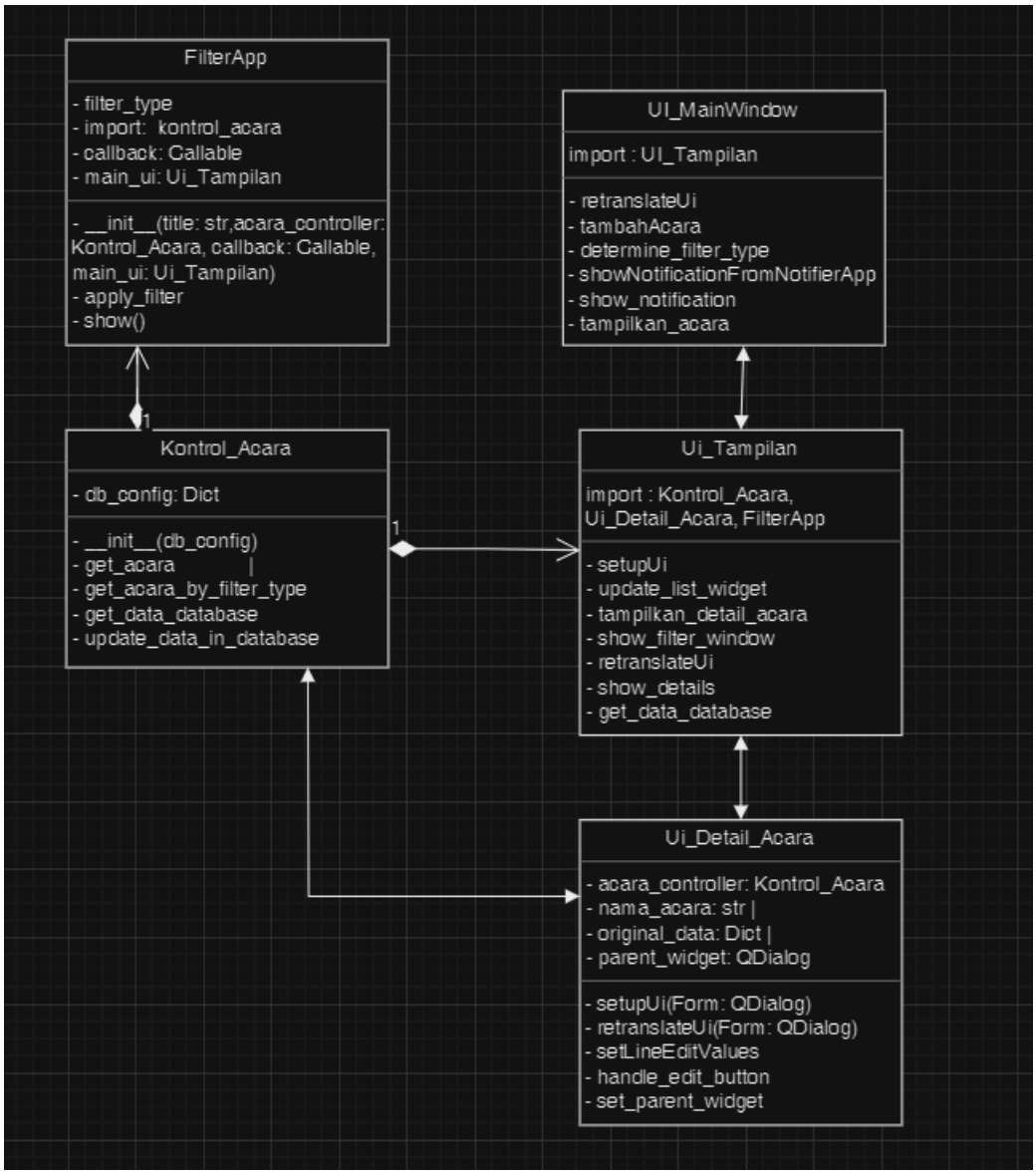


1. **Main Program:**
    - **Start:**
        - Program dimulai dari file **main_program.py**.
        - File ini akan membuat instance dari kelas **Ui_Tampilan** dan menampilkan antarmuka pengguna utama.
        - Menginisialisasi instance **Kontrol_Acara** dan **FilterApp**.
        - **Event Loop:**
            - Program terus berjalan dan menanggapi interaksi pengguna.
            - Menampilkan daftar acara menggunakan antarmuka pengguna utama.
            - Pengguna dapat mengklik tombol filter untuk memfilter acara.
            - Pengguna juga dapat mengklik acara untuk melihat detailnya.
2. **Tampilan Utama (Ui_Tampilan):**
    - **Inisialisasi:**
        - Membuat instance dari **Ui_Tampilan**.
        - Menginisialisasi elemen antarmuka pengguna.
        - Membuat instance dari **Kontrol_Acara** dan **FilterApp**.
    - **Menampilkan Daftar Acara:**
        - Menggunakan **Kontrol_Acara** untuk mendapatkan data acara.
        - Menampilkan data acara dalam QListWidget.
    - **Menanggapi Klik Filter:**
        - Memanggil fungsi **show_filter_window** dari **FilterApp**.
    - **Menanggapi Klik Acara:**
        - Memanggil fungsi **show_details** saat pengguna mengklik acara.
        - Menampilkan detail acara menggunakan **Ui_Detail_Acara**.
3. **Kontrol Acara (Kontrol_Acara):**
    - **Inisialisasi:**
        - Menerima konfigurasi database saat dibuat.
        - Membuat koneksi ke database.
    - **Operasi Database:**
        - Menyediakan fungsi untuk melakukan operasi pada tabel acara.

- Menyediakan metode untuk mendapatkan acara berdasarkan filter tertentu.
- Menyediakan metode untuk mendapatkan data acara dari database.
- Menyediakan metode untuk mengupdate dan menghapus data acara.

4. **FilterApp (Filter.py):**
    - **Inisialisasi:**
        - Menerima judul, kontrol acara, fungsi pembaruan, dan antarmuka pengguna utama.
        - Membuat elemen-elemen antarmuka pengguna untuk filter.
    - **Filter Acara:**
        - Memanggil metode **get_acara_by_filter_type** dari **Kontrol_Acara**.
        - Mengupdate antarmuka pengguna utama menggunakan fungsi pembaruan.
5. **Detail Acara (Ui_Detail_Acara):**
    - **Inisialisasi:**
        - Menerima kontrol acara dan nama acara saat dibuat.
        - Menginisialisasi elemen-elemen antarmuka pengguna untuk detail acara.
    - **Menampilkan Detail Acara:**
        - Memanggil fungsi **setLineEditValues** untuk mengisi elemen antarmuka pengguna dengan data acara.
        - Menerima input pengguna untuk mengedit atau menghapus acara.
        - Menangani tombol edit dan hapus untuk memperbarui atau menghapus acara dari database.

2.2 Class Diagram



1. **Main Program (jadwal.py):**
   - Classes:
     - UI_MainWindow
   - Associations:
     - **UI_Tampilan** (Association)

2. **Ui_Tampilan (Tampilan.py):**
    - Classes:
        - **Ui_Tampilan**
    - Associations:
        - **Kontrol_Acara** (Composition)
        - **FilterApp** (Composition)
        - **Ui_Detail_Acara** (Association)
3. **Kontrol Acara (kontrol_acara.py):**
    - Classes:
        - **Kontrol_Acara**
    - Associations:
        - Tidak ada dalam file ini.
4. **FilterApp (filter.py):**
    - Classes:
        - **FilterApp**
    - Associations:
        - **Kontrol_Acara** (Composition)
        - **Ui_Tampilan** (Association)
5. **Ui_Detail_Acara (detail_acara.py):**
    - Classes:
        - **Ui_Detail_Acara**
    - Associations:
        - **Kontrol_Acara** (Association)

2.3 Source code

source code disini kita bagi menjadi 5 fileyang berbeda: yaitu jadwal.py, kontrol_acara.py, detail_acara.py, tampilan.py, filter.py. berikut adalah full dari kode nya:

A). Jadwal.py

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QLabel,
QLineEdit, QPushButton, QVBoxLayout, QDateTimeEdit
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import QTimer
from plyer import notification
from datetime import datetime, timedelta
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QApplication, QMessageBox
from Tampilan import Ui_Tampilan
import calendar
import mysql.connector
import sys

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(754, 602)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(40, 30, 331, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(390, 30, 321, 41))
        self.lineEdit.setObjectName("lineEdit")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(40, 100, 201, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
```

```
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(40, 170, 201, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.dateEdit = QtWidgets.QDateEdit(self.centralwidget)
        self.dateEdit.setGeometry(QtCore.QRect(390, 100, 161, 41))
        self.dateEdit.setObjectName("dateEdit")
        self.timeEdit = QtWidgets.QTimeEdit(self.centralwidget)
        self.timeEdit.setGeometry(QtCore.QRect(390, 170, 161, 41))
        self.timeEdit.setObjectName("timeEdit")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(140, 220, 161, 31))
        font = QtGui.QFont()
        font.setPointSize(24)
        font.setBold(True)
        font.setWeight(75)
        self.label_4.setFont(font)
        self.label_4.setText("")
        self.label_4.setObjectName("label_4")
        self.label_6 = QtWidgets.QLabel(self.centralwidget)
        self.label_6.setGeometry(QtCore.QRect(40, 310, 331, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_6.setFont(font)
        self.label_6.setObjectName("label_6")
        self.textEdit = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit.setGeometry(QtCore.QRect(390, 310, 321, 141))
        self.textEdit.setUndoRedoEnabled(True)
        self.textEdit.setObjectName("textEdit")
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(40, 240, 201, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_5.setFont(font)
        self.label_5.setObjectName("label_5")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(390, 240, 81, 41))
```

```
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(500, 240, 81, 41))
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setGeometry(QtCore.QRect(170, 480, 151, 61))
        self.pushButton_4.setObjectName("pushButton_4")
        self.pushButton_6 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_6.setGeometry(QtCore.QRect(430, 480, 151, 61))
        self.pushButton_6.setObjectName("pushButton_6")
        self.pushButton_4.clicked.connect(self.tambahAcara)
        self.pushButton_6.clicked.connect(self.tampilkan_acara)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 754, 18))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.notification_timer = QTimer(self.centralwidget)
        self.notification_timer.timeout.connect(self.show_notification)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate(
            "MainWindow", "Penjadwalan Acara"))
        self.label.setText(_translate("MainWindow", "Nama Acara:"))
        self.label_2.setText(_translate("MainWindow", "Tanggal:"))
        self.label_3.setText(_translate("MainWindow", "Waktu:"))
        self.label_6.setText(_translate("MainWindow", "Deskripsi Acara:"))
        self.label_5.setText(_translate("MainWindow", "Pengingat:"))
        self.pushButton.setText(_translate("MainWindow", "Ya"))
        self.pushButton_2.setText(_translate("MainWindow", "Tidak"))
        self.pushButton_4.setText(_translate("MainWindow", "Tambah Acara"))
        self.pushButton_6.setText(_translate("MainWindow", "Tampilkan Acara"))

    def tambahAcara(self):
        # Ambil data dari UI
        event_name = self.lineEdit.text()
        event_date = self.dateEdit.date().toString("yyyy-MM-dd")
```

```
        event_time = self.timeEdit.time().toString("hh:mm:ss")
        event_description = self.textEdit.toPlainText()

        # Validasi data
        if not event_date or not event_time:
            QMessageBox.critical(self.centralwidget, "Error", "Tanggal dan Waktu
harus diisi.")
            return

        try:
            # Konfigurasi koneksi ke database MySQL
            db_config = {
                'host': 'localhost',
                'user': 'root',
                'password': 'valhalla123',
                'database': 'jadwal_acara',
            }

            # Membuat koneksi
            connection = mysql.connector.connect(**db_config)

            # Membuat kursor
            cursor = connection.cursor()

            # Tentukan Filter Type berdasarkan tanggal acara
            filter_type = self.determine_filter_type(event_date)

            # Query untuk menyimpan data
            query = "INSERT INTO acara (Nama_Acara, Tanggal, Waktu,
Deskripsi_Acara, Filter_Type) VALUES (%s, %s, %s, %s, %s)"
            values = (event_name, event_date, event_time, event_description,
filter_type)

            # Eksekusi query
            cursor.execute(query, values)

            # Commit perubahan ke database
            connection.commit()

            # Menutup kursor dan koneksi
            cursor.close()
            connection.close()

            # Mengatur nilai default setelah acara ditambahkan
            self.lineEdit.clear()
```

```
            self.dateEdit.setDate(QtCore.QDate.currentDate())
            self.timeEdit.setTime(QtCore.QTime.currentTime())
            self.textEdit.clear()

            # Memberikan notifikasi bahwa acara berhasil ditambahkan
            QMessageBox.information(
                self.centralwidget, "Sukses", "Acara berhasil ditambahkan ke
database.")
        except Exception as e:
            # Menampilkan pesan error jika terjadi masalah
            QMessageBox.critical(self.centralwidget, "Error", f"Terjadi
kesalahan: {str(e)}")

    def determine_filter_type(self, event_date):
        # Tentukan Filter Type berdasarkan tanggal acara
        selected_date = datetime.strptime(event_date, "%Y-%m-%d").date()
        today = datetime.now().date()

        _, last_day_of_month = calendar.monthrange(today.year, today.month)
        _, last_day_of_selected_month = calendar.monthrange(selected_date.year,
selected_date.month)

        end_of_week = today + timedelta(days=(calendar.SATURDAY -
today.weekday()))
        end_of_month = datetime(today.year, today.month,
last_day_of_month).date()

        if selected_date == today:
            return "Harian"
        elif today <= selected_date <= end_of_week:
            return "Mingguan"
        elif today <= selected_date <= end_of_month:
            return "Bulanan"
        else:
            return "Semua"

    def showNotificationFromNotifierApp(self):
        print("Entering showNotificationFromNotifierApp")

        # Extract relevant data from Ui_MainWindow
        event_name = self.lineEdit.text()
        event_date = self.dateEdit.date().toString("yyyy-MM-dd")
        event_time = self.timeEdit.time().toString("hh:mm:ss")
        event_description = self.textEdit.toPlainText()
```

```
        # Validasi: Pastikan pengingat dijadwalkan dengan acara
        if not (event_date and event_time):
            print("Pengingat dijadwalkan tanpa acara")
            self.notification_timer.start(0)  # Menjalankan timer tanpa waktu
tunggu
            return

        # Validasi: Pastikan nama acara diisi
        if not event_name:
            print("Nama Acara harus diisi")
            QMessageBox.critical(self.centralwidget, "Error", "Nama Acara harus
diisi.")
            return

        # Validasi: Jangan tampilkan notifikasi jika waktu yang dipilih sudah
berlalu
        selected_datetime_str = f"{event_date} {event_time}"
        selected_datetime = datetime.strptime(selected_datetime_str, "%Y-%m-%d
%H:%M:%S")
        current_datetime = datetime.now()

        if selected_datetime <= current_datetime:
            print("Waktu yang dipilih sudah berlalu")
            QMessageBox.critical(self.centralwidget, "Error", "Waktu yang dipilih
sudah berlalu.")
            return

        # Validasi: Pastikan deskripsi acara diisi
        if not event_description:
            print("Deskripsi Acara harus diisi")
            QMessageBox.critical(self.centralwidget, "Error", "Deskripsi Acara
harus diisi.")
            return

        # Calculate time difference in seconds
        time_difference = (selected_datetime - current_datetime).total_seconds()

        # Schedule the notification using a QTimer
        self.notification_timer.start(int(time_difference * 1000))

        print("Notification scheduled")

    def show_notification(self):
        # Stop the timer to avoid repeated notifications
        self.notification_timer.stop()
```

```
        # Extract relevant data from Ui_MainWindow
        event_name = self.lineEdit.text()
        event_description = self.textEdit.toPlainText()

        # Set data for notification
        get_title = event_name
        get_msg = event_description

        # Use plyer's notification with a longer timeout
        try:
            notification.notify(
                title=get_title,
                message=get_msg,
                app_name="Notifier",
                timeout=20
            )
            print("Notification shown")
        except Exception as e:
            print(f"Error showing notification: {e}")

    def tampilkan_acara(self):
        # Membuat instance dari QWidget untuk menampilkan antarmuka pengguna
        self.form_widget = QtWidgets.QWidget()

        # Membuat instance dari Ui_Form dan menampilkan form
        ui_form = Ui_Tampilan()
        ui_form.setupUi(self.form_widget)

        # Menyimpan rujukan ke Ui_Form
        self.form_ui = ui_form

        # Mengambil data dari database
        try:
            # Konfigurasi koneksi ke database MySQL
            db_config = {
                'host': 'localhost',
                'user': 'root',
                'password': 'valhalla123',
                'database': 'jadwal_acara',
            }

            # Membuat koneksi
            connection = mysql.connector.connect(**db_config)
```

```
            # Membuat kursor
            cursor = connection.cursor()

            # Query untuk mengambil semua data acara
            query = "SELECT Nama_Acara FROM acara"
            cursor.execute(query)

            # Mendapatkan semua data
            all_events = cursor.fetchall()

            # Menutup kursor dan koneksi
            cursor.close()
            connection.close()

            # Menampilkan nama acara pada QListWidget
            for event in all_events:
                ui_form.listWidget.addItem(event[0])

            # Tampilkan form
            self.form_widget.show()
        except Exception as e:
            # Menampilkan pesan error jika terjadi masalah
            QMessageBox.critical(self.centralwidget, "Error",f"Terjadi kesalahan:
{str(e)}")

if __name__ == "__main__":
    app = QApplication([])
    MainWindow = QMainWindow()
    ui_main = Ui_MainWindow()
    ui_main.setupUi(MainWindow)

    # Hanya membuat koneksi di __main__
    ui_main.pushButton.clicked.connect(ui_main.showNotificationFromNotifierApp)
    ui_main.pushButton_6.clicked.connect(ui_main.tampilkan_acara)

    MainWindow.show()
    sys.exit(app.exec())
```

B). Tampilan.py

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QMessageBox
```

```
from datetime import datetime, timedelta, date
from Detail_Acara import Ui_Detail_Acara
from Kontrol_Acara import Kontrol_Acara
from Filter import FilterApp
import mysql.connector

class Ui_Tampilan(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(481, 358)

        self.label = QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(50, 20, 301, 41))
        font = QtGui.QFont()
        font.setPointSize(18)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")

        self.listWidget = QtWidgets.QListWidget(Form)
        self.listWidget.setGeometry(QtCore.QRect(50, 80, 381, 251))
        self.listWidget.setObjectName("listWidget")

        self.pushButton = QtWidgets.QPushButton(Form)
        self.pushButton.setGeometry(QtCore.QRect(340, 20, 91, 41))
        self.pushButton.setObjectName("pushButton")
        self.pushButton.clicked.connect(self.show_filter_window)
        # Menghubungkan sinyal doubleClicked ke fungsi show_details
        self.listWidget.doubleClicked.connect(self.show_details)

        self.retranslateUi(Form)

        # Buat instance AcaraController dengan konfigurasi database yang sesuai
        self.acara_controller = Kontrol_Acara({
            'host': 'localhost',
            'user': 'root',
            'password': 'valhalla123',
            'database': 'jadwal_acara',
        })

        # Buat instance FilterApp dengan kontrol_acara dan callback yang sesuai
        self.filter_app = FilterApp("Filter Acara", self.acara_controller,
self.update_list_widget, self)
```

```
        self.filter_app.main_ui = self  # Atur atribut main_ui pada objek
FilterApp



    def filter_acara(self):
        # Mendapatkan tipe filter dari ComboBox
        filter_type = self.comboBox.currentText()

        # Set filter type di objek FilterApp
        self.filter_app.filter_type = filter_type

        # Panggil metode apply_filter dari objek FilterApp
        self.filter_app.apply_filter()

    def update_list_widget(self, data):
        # Hapus semua item di listWidget sebelum memperbarui
        self.listWidget.clear()

        # Check if filter_type is set
        if hasattr(self.filter_app, 'filter_type') and
self.filter_app.filter_type is not None:
            # Filter data based on filter type
            filtered_data = [item for item in data if item[1] ==
self.filter_app.filter_type]
            for item in filtered_data:
                self.listWidget.addItem(str(item[1]))
        else:
            # Display all data if no filter type is selected
            for item in data:
                self.listWidget.addItem(str(item[1]))

    def tampilkan_detail_acara(self, original_data):
        detail_acara_dialog = QtWidgets.QDialog()
        detail_acara_ui = Ui_Detail_Acara(self.acara_controller, original_data)
        detail_acara_ui.setupUi(detail_acara_dialog)
        result = detail_acara_dialog.exec_()

    def show_filter_window(self):
        # Panggil metode show_filter_window dari objek FilterApp
        self.filter_app.show()

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
```

```
        self.label.setText(_translate("Form", "Daftar Acara:"))
        self.pushButton.setText(_translate("Form", "Filter"))

    def show_details(self):
        # Mendapatkan item yang dipilih dari QListWidget
        selected_item = self.listWidget.currentItem()

        # Pernyataan print tambahan
        print("Nama Acara yang Dipilih:", selected_item.text())

        # Memastikan ada item yang dipilih sebelum melanjutkan
        if selected_item:
            # Menyimpan data awal sebelum diedit
            data_item =
self.acara_controller.get_data_database(selected_item.text())

            # Pernyataan print tambahan
            print("Data Item dari Database:", data_item)

            # Memastikan data_item tidak None dan memiliki panjang yang cukup
            if data_item is not None and len(data_item) > 0:
                data_item = data_item[0]  # Ambil elemen pertama dari tuple

                self.original_data = {
                    "Nama Acara": data_item[1],
                    "Tanggal": QtCore.QDate.fromString(str(data_item[2]), "yyyy-
MM-dd"),
                    "Waktu": QtCore.QTime.fromString(str(data_item[3]),
"hh:mm:ss"),
                    "Deskripsi Acara": data_item[4],
                }

                # Membuat instance dari dialog Detail_Acara
                detail_acara_dialog = QtWidgets.QDialog()
                detail_acara_ui = Ui_Detail_Acara()

                # Creating an instance of Ui_Detail_Acara and initializing it
                detail_acara_ui.initialize(self.acara_controller,
selected_item.text())

                # Setting up the UI for the detail view
                detail_acara_ui.setupUi(detail_acara_dialog)

                # Menampilkan data pada UI Detail_Acara
                detail_acara_ui.setLineEditValues(self.original_data)
```

```
                # Menampilkan dialog Detail_Acara
                if detail_acara_dialog.exec_() == QtWidgets.QDialog.Accepted:
                    # Update QListWidget
                    self.update_list_widget(self.acara_controller.get_acara_by_fi
lter_type(self.filter_app.filter_type))
            else:
                # Menampilkan pesan kesalahan jika data_item tidak lengkap atau
tidak tersedia
                error_message = f"Data untuk item dengan nama acara
'{selected_item.text()}' tidak tersedia atau tidak lengkap."
                QtWidgets.QMessageBox.critical(self.listWidget, "Error",
error_message)
                print(error_message)  # Tambahkan ini untuk mencetak informasi
tambahan ke terminal
        else:
            # Menampilkan pesan kesalahan jika tidak ada item yang dipilih
            QtWidgets.QMessageBox.critical(self.listWidget, "Error", "Tidak ada
item yang dipilih.")

    def get_data_database(self, nama_acara):
        # Assuming 'acara_controller' is an instance of Kontrol_Acara
        return self.acara_controller.get_data_database(nama_acara)

    def tampilkan_detail_acara(self, original_data):
        detail_acara_dialog = QtWidgets.QDialog()
        detail_acara_ui = Ui_Detail_Acara()
        detail_acara_ui.initialize(self.acara_controller, original_data)
        detail_acara_ui.setupUi(detail_acara_dialog)
        result = detail_acara_dialog.exec_()
```

C). Kontrol_acara.py

```
import mysql.connector

class Kontrol_Acara:
    def __init__(self, db_config):
        self.db_config = db_config
        self.connection = None

    def connect_to_database(self):
        try:
            self.connection = mysql.connector.connect(**self.db_config)
```

```
        except mysql.connector.Error as err:
            print(f"Error connecting to the database: {err}")

    def close_database_connection(self):
        if self.connection:
            self.connection.close()

    def execute_query(self, query, params=None):
        try:
            if not self.connection or not self.connection.is_connected():
                self.connect_to_database()

            with self.connection.cursor() as cursor:
                if params is not None:
                    cursor.execute(query, params)
                else:
                    cursor.execute(query)

                data = cursor.fetchall()
                if not data:
                    print("No data matching the filter criteria.")
                return data

        except mysql.connector.Error as err:
            print(f"MySQL error: {err}")
            print(f"Failed query: {query}")
            return None
        except Exception as e:
            print(f"Error: {str(e)}")
            return None
        finally:
            self.close_database_connection()

    def get_acara_by_filter_type(self, filter_type):
        if filter_type == "Harian":
            query = "SELECT * FROM acara WHERE Tanggal = CURDATE()"
        elif filter_type == "Mingguan":
            query = "SELECT * FROM acara WHERE Tanggal BETWEEN CURDATE() AND
CURDATE() + INTERVAL 6 DAY"
        elif filter_type == "Bulanan":
            query = "SELECT * FROM acara WHERE Tanggal BETWEEN CURDATE() AND
CURDATE() + INTERVAL 29 DAY"
        else:
            query = "SELECT * FROM acara"
```

```
        return self.execute_query(query)

    def get_data_database(self, nama_acara):
        query = "SELECT * FROM acara WHERE Nama_Acara = %s"
        return self.execute_query(query, (str(nama_acara),))

    def update_data_in_database(self, nama_acara, updated_data):
        query = "UPDATE acara SET Nama_Acara = %s, Tanggal = %s, Waktu = %s,
Deskripsi_Acara = %s WHERE Nama_Acara = %s"

        # Ensure that the values are properly formatted for SQL
        updated_values = (
            updated_data.get("Nama_Acara", ""),
            updated_data.get("Tanggal", ""),
            updated_data.get("Waktu", ""),
            updated_data.get("Deskripsi_Acara", "")
        )
        # Print the data before the update
        select_query = "SELECT * FROM acara WHERE Nama_Acara = %s"
        selected_data = self.get_data_from_database(select_query,
(str(nama_acara),))
        print(f"Selected data before update: {selected_data}")

        try:
            if not self.connection or not self.connection.is_connected():
                self.connect_to_database()

            with self.connection.cursor() as cursor:
                print(f"Executing query: {query} with values:
{updated_values!r}")
                cursor.execute(query, updated_values + (str(nama_acara),))  #
Concatenate the nama_acara to the tuple
                self.connection.commit()
                print("Data updated successfully.")
        except mysql.connector.Error as err:
            print(f"MySQL error: {err}")
            print(f"Failed query: {query} with values: {updated_values!r}")
            raise  # Reraise the exception to see the full traceback
        finally:
            # Always close the connection in the finally block
            self.close_database_connection()

    def delete_data_in_database(self, nama_acara):
        query = "DELETE FROM acara WHERE Nama_Acara = %s"
```

```
        try:
            if not self.connection or not self.connection.is_connected():
                self.connect_to_database()

            with self.connection.cursor() as cursor:
                cursor.execute(query, (str(nama_acara),))
            self.connection.commit()
            print("Data deleted successfully.")
        except mysql.connector.Error as err:
            print(f"MySQL error: {err}")
            print(f"Failed query: {query} with value: {nama_acara!r}")
            raise  # Reraise the exception to see the full traceback
        finally:
            self.close_database_connection()

    def get_data_for_detail_acara(self, event_name):
        # Fungsi khusus untuk mendapatkan data untuk Detail Acara
        query = "SELECT * FROM acara WHERE Nama_Acara = %s"
        return self.execute_query(query, (event_name,))

    def get_data_from_database(self, query, query_params=None):
        try:
            if not self.connection or not self.connection.is_connected():
                self.connect_to_database()

            with self.connection.cursor() as cursor:
                print(f"Executing query: {query} with params: {query_params}")
                if query_params is not None:
                    cursor.execute(query, query_params)
                else:
                    cursor.execute(query)

                data = cursor.fetchall()
                if not data:
                    print("No data matching the filter criteria.")
                return data

        except mysql.connector.Error as err:
            print(f"MySQL error: {err}")
            print(f"Failed query: {query}")
            return None
        except Exception as e:
            print(f"Error: {str(e)}")
            return None
        finally:
```

```
            self.close_database_connection()
```

D). Filter.py

```
from PyQt5.QtWidgets import QDialog, QLabel, QPushButton, QVBoxLayout,
QFormLayout, QComboBox
from Kontrol_Acara import Kontrol_Acara
import mysql.connector

class FilterApp(QDialog):
    def __init__(self, title, kontrol_acara, update_list_widget_callback,
main_ui):
        super().__init__()
        self.setWindowTitle(title)
        self.kontrol_acara = kontrol_acara
        self.update_list_widget_callback = update_list_widget_callback
        self.filter_type = None  # Add a new attribute for filter type
        self.main_ui = main_ui  # Set the main_ui attribute

        self.create_widgets()

    def create_widgets(self):
        layout = QFormLayout()

        # Combo Box for filter types
        self.filter_type_combo = QComboBox()
        self.filter_type_combo.addItems(["Harian", "Mingguan", "Bulanan"])
        layout.addRow(QLabel("Filter Type:"), self.filter_type_combo)

        # Filter Button
        filter_button = QPushButton("Filter")
        filter_button.clicked.connect(self.apply_filter)
        layout.addRow(filter_button)

        # Status Label
        self.status_label = QLabel()
        layout.addRow(self.status_label)

        self.setLayout(layout)

        # Dictionary to store filter values
        self.filters = {"Filter Type": None}

    def apply_filter(self):
```

```
        # Set filter values based on user input
        filter_type = self.filter_type_combo.currentText()
        self.filters["Filter Type"] = filter_type
        self.main_ui.filter_type = filter_type  # Set filter type in the main UI

        # Call the get_acara_by_filter_type function with the set filter
        result = self.kontrol_acara.get_acara_by_filter_type(filter_type)

        # Update the view if there are filter results
        if result is not None:
            self.update_list_widget_callback(result)

        self.status_label.setText(f"Filter Applied: {self.filters}")
```

E). Detail_acara.py

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QLineEdit, QDateEdit, QTimeEdit, QTextEdit

class Ui_Detail_Acara(object):
    def initialize(self, acara_controller, nama_acara):
        self.acara_controller = acara_controller
        self.nama_acara = nama_acara
        self.original_data = None

    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(731, 531)
        self.label = QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(30, 20, 311, 41))
        font = QtGui.QFont()
        font.setPointSize(20)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.lineEdit_2 = QtWidgets.QLineEdit(Form)
        self.lineEdit_2.setGeometry(QtCore.QRect(380, 80, 321, 41))
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.label_7 = QtWidgets.QLabel(Form)
        self.label_7.setGeometry(QtCore.QRect(30, 80, 331, 41))
```

```
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_7.setFont(font)
        self.label_7.setObjectName("label_7")
        self.dateEdit_2 = QtWidgets.QDateEdit(Form)
        self.dateEdit_2.setGeometry(QtCore.QRect(380, 150, 161, 41))
        self.dateEdit_2.setObjectName("dateEdit_2")
        self.label_8 = QtWidgets.QLabel(Form)
        self.label_8.setGeometry(QtCore.QRect(30, 150, 201, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_8.setFont(font)
        self.label_8.setObjectName("label_8")
        self.timeEdit_2 = QtWidgets.QTimeEdit(Form)
        self.timeEdit_2.setGeometry(QtCore.QRect(380, 220, 161, 41))
        self.timeEdit_2.setObjectName("timeEdit_2")
        self.label_4 = QtWidgets.QLabel(Form)
        self.label_4.setGeometry(QtCore.QRect(30, 220, 201, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_4.setFont(font)
        self.label_4.setObjectName("label_4")
        self.label_9 = QtWidgets.QLabel(Form)
        self.label_9.setGeometry(QtCore.QRect(30, 290, 331, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label_9.setFont(font)
        self.label_9.setObjectName("label_9")
        self.textEdit_2 = QtWidgets.QTextEdit(Form)
        self.textEdit_2.setGeometry(QtCore.QRect(380, 290, 321, 141))
        self.textEdit_2.setUndoRedoEnabled(True)
        self.textEdit_2.setObjectName("textEdit_2")
        self.pushButton = QtWidgets.QPushButton(Form)
        self.pushButton.setGeometry(QtCore.QRect(150, 460, 141, 51))
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(Form)
        self.pushButton_2.setGeometry(QtCore.QRect(440, 460, 141, 51))
```

```
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton.clicked.connect(self.handle_edit_button)
        self.pushButton_2.clicked.connect(self.handle_delete_button)
        print("LineEdit_2 found:", hasattr(self, 'lineEdit_2'))
        print("DateEdit_2 found:", hasattr(self, 'dateEdit_2'))
        print("TimeEdit_2 found:", hasattr(self, 'timeEdit_2'))
        print("TextEdit_2 found:", hasattr(self, 'textEdit_2'))

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def initialize(self, acara_controller, nama_acara):
        self.acara_controller = acara_controller
        self.nama_acara = nama_acara
        self.original_data = None

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Detail Acara"))
        self.label.setText(_translate("Form", "Detail Acara:"))
        self.label_7.setText(_translate("Form", "Nama Acara:"))
        self.label_8.setText(_translate("Form", "Tanggal:"))
        self.label_4.setText(_translate("Form", "Waktu:"))
        self.label_9.setText(_translate("Form", "Deskripsi Acara:"))
        self.pushButton.setText(_translate("Form", "Edit Acara"))
        self.pushButton_2.setText(_translate("Form", "Hapus Acara"))

    # Perubahan pada metode setLineEditValues
    def setLineEditValues(self, data):
        print("Data diatur ke Detail_Acara:", data)  # Debug line

        self.lineEdit_2.setText(data.get("Nama Acara", ""))
        self.dateEdit_2.setDate(data.get("Tanggal", QtCore.QDate.currentDate()))
        self.timeEdit_2.setTime(data.get("Waktu", QtCore.QTime.currentTime()))
        self.textEdit_2.setPlainText(data.get("Deskripsi Acara", ""))

    # Tambahkan fungsi untuk menangani tombol Edit Acara
    def handle_edit_button(self):
        # Mengambil nilai dari elemen UI
        updated_data = {
            "Nama_Acara": self.lineEdit_2.text(),
            "Tanggal": self.dateEdit_2.date().toString("yyyy-MM-dd"),
            "Waktu": self.timeEdit_2.time().toString("hh:mm:ss"),
            "Deskripsi_Acara": self.textEdit_2.toPlainText(),
        }
```

```
        # Update data di database menggunakan AcaraController
        self.acara_controller.update_data_in_database(self.nama_acara,
updated_data)

        # Tutup dialog setelah selesai mengedit (if applicable)
        if hasattr(self, 'parent_widget') and self.parent_widget:
            self.parent_widget.accept()

    def set_parent_widget(self, parent_widget):
        self.parent_widget = parent_widget

    # New method for handling the "Hapus Acara" button click
    def handle_delete_button(self):
        # Confirmation dialog before deleting the event
        reply = QtWidgets.QMessageBox.question(
            None,
            'Hapus Acara',
            'Apakah Anda yakin ingin menghapus acara ini?',
            QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No,
            QtWidgets.QMessageBox.No
        )

        if reply == QtWidgets.QMessageBox.Yes:
            # Call the delete method from the controller
            self.acara_controller.delete_data_in_database(self.nama_acara)

            # Close the detail window or perform any other actions needed
            if hasattr(self, 'parent_widget') and self.parent_widget:
                self.parent_widget.accept()
```

2.4 Hasil Program dan Penjelasan

Alur Program:

1. **Pembukaan Aplikasi:**
    - Program dimulai dengan eksekusi file **jadwal.py**.
    - File tersebut menyertakan definisi kelas dan berfungsi sebagai titik masuk utama.
2. **Inisialisasi UI dan Koneksi Database:**
    - Program membuat instance dari **Ui_Tampilan** (kelas yang bertanggung jawab untuk tampilan utama) dan menginisialisasi antarmuka pengguna (UI) menggunakan PyQt5.
    - **Ui_Tampilan** melakukan inisialisasi objek **Kontrol_Acara** yang bertindak sebagai penghubung antara antarmuka pengguna dan database MySQL.
3. **Tampilan Utama dan Filter:**
    - UI menampilkan daftar acara pada **listWidget**.
    - Pengguna dapat memfilter acara berdasarkan tipe (Harian, Mingguan, Bulanan) dengan mengklik tombol "Filter".
    - Saat tombol "Filter" ditekan, muncul jendela filter (**FilterApp**) yang memungkinkan pengguna memilih tipe filter.
4. **Pembaruan Tampilan Berdasarkan Filter:**
    - Hasil filter diterapkan pada daftar acara, dan UI diperbarui untuk mencerminkan hasil tersebut.
    - **FilterApp** mengubah filter type pada instance **Ui_Tampilan** dan memanggil **update_list_widget** untuk memperbarui tampilan.
5. **Menampilkan Detail Acara:**
    - Pengguna dapat melihat detail acara dengan mengklik dua kali pada salah satu acara dalam daftar.
    - **Ui_Tampilan** menanggapi sinyal double-click dengan memanggil metode **show_details**.
6. **Jendela Detail Acara (Ui_Detail_Acara):**
    - Jendela detail acara menampilkan informasi lengkap tentang acara yang dipilih.
    - Pengguna memiliki opsi untuk mengedit atau menghapus acara.
7. **Edit Acara:**
    - Pengguna dapat mengklik tombol "Edit Acara" di jendela detail acara.

- **Ui_Detail_Acara** menanggapi klik tombol dan memanggil metode **handle_edit_button**.

8. **Memperbarui Database:**
    - Data acara yang diubah dimasukkan kembali ke database melalui **Kontrol_Acara**.
9. **Hapus Acara:**
    - Pengguna dapat mengklik tombol "Hapus Acara" di jendela detail acara.
    - **Ui_Detail_Acara** menanggapi klik tombol dan memanggil metode **handle_delete_button**.
10. **Menghapus Data dari Database:**
    - Data acara dihapus dari database melalui **Kontrol_Acara**.
11. **Penutupan Aplikasi:**
    - Aplikasi berjalan sampai pengguna memutuskan untuk menutupnya.

Penjelasan:

- Aplikasi ini menggunakan konsep MVC (Model-View-Controller), di mana **Kontrol_Acara** berperan sebagai pengontrol logika bisnis dan interaksi dengan database, **Ui_Tampilan** sebagai antarmuka pengguna, dan **FilterApp** sebagai bagian dari antarmuka pengguna yang mengelola filter.
- **Ui_Detail_Acara** mengelola tampilan dan interaksi pada jendela detail acara.
- Data diambil dari database melalui **Kontrol_Acara** dan ditampilkan dalam UI menggunakan PyQt5.
- Pengguna dapat memanipulasi data acara melalui antarmuka pengguna, dan perubahan tersebut diterapkan kembali ke database melalui **Kontrol_Acara**.
- Program ini memanfaatkan modularitas dengan memisahkan fungsi-fungsi ke dalam kelas-kelas terpisah untuk memudahkan pemeliharaan dan pengembangan.

BAB III
PENUTUP

3.1 Kesimpulan

Dalam pengembangan aplikasi reminder ini, kami dapat menyimpulkan beberapa hal penting. Pertama, dari latar belakang yang telah dijelaskan pada Bab I Pendahuluan, aplikasi reminder ini dibangun untuk menjawab kebutuhan akan alat pengingat yang dapat membantu pengguna dalam mengelola waktu dan tugas sehari-hari.

Pembahasan di Bab II mengenai flowchart sistem memberikan pemahaman visual terhadap langkah-langkah kerja aplikasi. Class diagram menunjukkan struktur kelas dan hubungan antar kelas yang mendasari implementasi program. Dengan adanya diagram ini, kami dapat memastikan bahwa desain aplikasi reminder ini telah dikerjakan secara sistematis dan efisien.

Hasil program yang berhasil dijelaskan dalam bagian 2.3 menunjukkan bahwa aplikasi reminder ini dapat diimplementasikan dengan baik dan sesuai dengan kebutuhan pengguna. Fitur-fitur yang dihadirkan, seperti pengaturan waktu, notifikasi, dan manajemen tugas, memberikan solusi yang praktis dan efektif dalam membantu pengguna menjalani kegiatan sehari-hari.

Melalui praktik ini, kami mendapatkan pengalaman berharga dalam pengembangan perangkat lunak, mulai dari perencanaan, desain, hingga implementasi. Meskipun terdapat beberapa tantangan, seperti pengoptimalan performa dan penanganan perubahan kebutuhan, namun kami berhasil mengatasinya dengan mengandalkan kerjasama tim dan kreativitas.

Secara keseluruhan, aplikasi reminder ini diharapkan dapat memberikan nilai tambah bagi penggunanya dan menjadi solusi yang efisien dalam manajemen waktu. Ke depannya, kami akan terus mempertimbangkan masukan dan feedback dari pengguna untuk meningkatkan kualitas dan fungsionalitas aplikasi ini.

Demikianlah kesimpulan dari laporan praktik pengembangan aplikasi reminder ini. Terima kasih dan maaf jika ada kurangnya.

DAFTAR PUSTAKA

Membuat Class diagram. (n.d). Class Diagram. https://app.diagrams.net/

Membuat Flowchart. (n.d). Flowchart Sistem. https://whimsical.com/

PyQt5 Tutorial. (n.d). tutorialspoint https://www.tutorialspoint.com/pyqt5/index.htm

Panduan Class Diagram. (n.d.). Class Diagram Relationships. https://creately.com/guides/class-diagram-relationships/