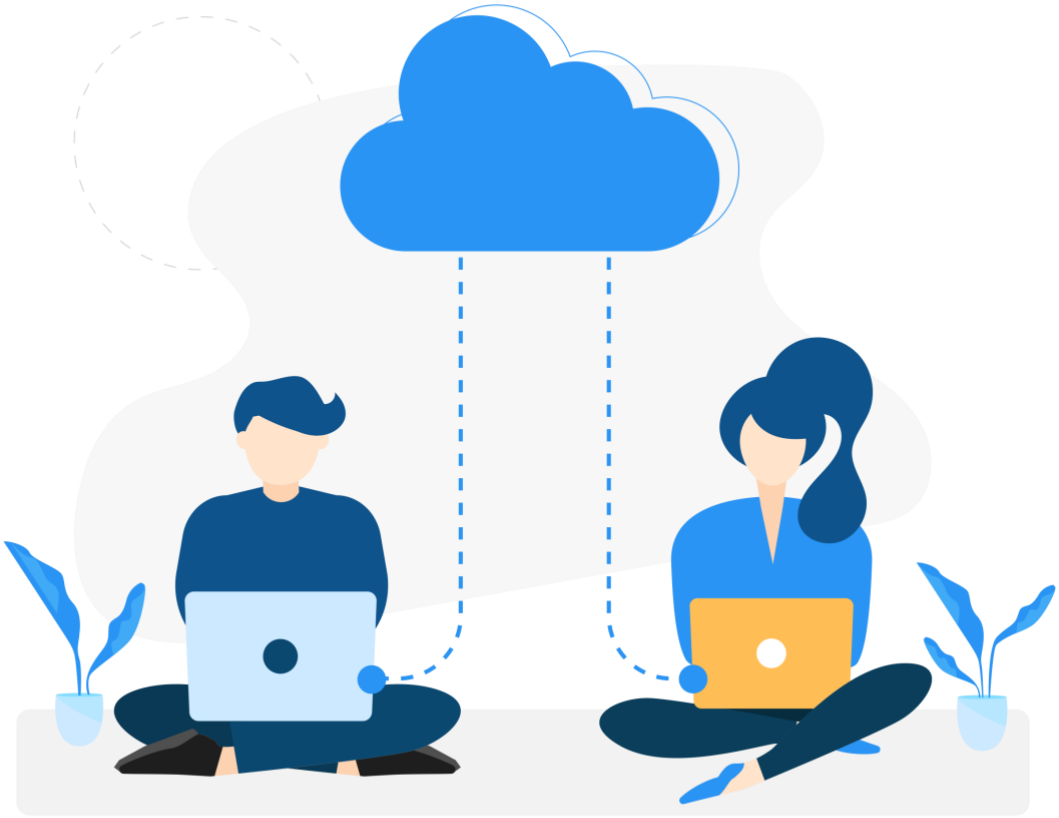
BAB 9

Form dan Database



9.1 Tujuan

1. Dapat memahami pembuatan form dengan PyQt5
2. Dapat mengimplementasikan operasi database dengan PyQt5

9.2 Pengantar

9.2.1 Form

Dalam konteks pengembangan aplikasi GUI (Graphical User Interface), form adalah jendela atau layar yang berisi elemen-elemen GUI seperti tombol, kotak teks, label, dan lainnya. Form digunakan untuk berinteraksi dengan pengguna, misalnya untuk menerima input dari pengguna atau menampilkan informasi kepada pengguna. Dalam PyQt5, kita dapat membuat form dengan menggunakan class QWidget atau class turunannya seperti QMainWindow dan QDialog. Secara umum, kita akan menggunakan QWidget untuk membuat widget kustom atau form sederhana, QMainWindow untuk jendela utama aplikasi, dan QDialog untuk dialog dan kotak pesan.

9.2.1.1 Qwidget

QWidget adalah kelas dasar untuk semua objek antarmuka pengguna, atau widget, dalam PyQt5. QWidget adalah kelas yang sangat fleksibel yang dapat digunakan sebagai dasar untuk membuat berbagai jenis widget, termasuk form dan dialog. QWidget dapat berisi widget lain, dan kita dapat menentukan layout untuk menata widget-widget tersebut.

```
1. from PyQt5.QtWidgets import QApplication, QWidget
2. 
3. class MyForm(QWidget):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle('My Form')
7. 
8. app = QApplication([])
9. form = MyForm()
10.form.show()
11.app.exec_()
```

9.2.1.2 QMainWindow

QMainWindow adalah kelas khusus yang diturunkan dari QWidget dan dirancang untuk menjadi jendela utama aplikasi Anda. QMainWindow memiliki struktur yang telah ditentukan sebelumnya yang mencakup area konten utama dan beberapa panel tambahan, seperti toolbar, menu, dan status bar. Kita biasanya akan menambahkan widget lain ke area konten utama QMainWindow.

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. class MainWindow(QMainWindow):
3.     def __init__(self):
4.         super().__init__()
5.         self.setWindowTitle('Main Window')
6. app = QApplication([])
7. main_window = MainWindow()
8. main_window.show()
9. app.exec_()
```

9.2.1.3 QDialog

QDialog adalah kelas lain yang diturunkan dari QWidget dan dirancang untuk digunakan sebagai dialog. Dialog adalah jendela sekunder yang digunakan untuk berinteraksi dengan pengguna, biasanya untuk mendapatkan input atau memberikan informasi. QDialog memiliki beberapa fitur tambahan yang dirancang khusus untuk penggunaan ini, seperti kemampuan untuk menampilkan dialog secara modal (yaitu, mencegah pengguna berinteraksi dengan jendela lain sampai dialog ditutup).

```
1. from PyQt5.QtWidgets import QApplication, QDialog
2.
3. class MyDialog(QDialog):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle('My Dialog')
7.
8. app = QApplication([])
9. dialog = MyDialog()
10.dialog.show()
```

```
11. app.exec_()
12.
```

9.2.1.4 Menghubungkan dua form atau lebih

Berikut ini contoh kode jika kita ingin menghubungkan dua form dalam PyQt5. Dalam kode ini, kita membuat dua class: MainWindow dan DialogWindow. MainWindow adalah subclass dari QMainWindow dan DialogWindow adalah subclass dari QDialog. MainWindow memiliki QPushButton, dan ketika tombol ini diklik, sebuah instance dari DialogWindow dibuat dan ditampilkan.

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton,
    QDialog
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle('Main Window')
7.
8.         self.button = QPushButton("Open Dialog", self)
9.         self.button.clicked.connect(self.open_dialog)
10.
11.    def open_dialog(self):
12.        self.dialog = DialogWindow()
13.        self.dialog.show()
14.
15.class DialogWindow(QDialog):
16.    def __init__(self):
17.        super().__init__()
18.        self.setWindowTitle('Dialog Window')
19.
20.if __name__ == "__main__":
21.    app = QApplication([])
22.    main_window = MainWindow()
23.    main_window.show()
24.    app.exec_()
```

9.2.2 Database

Database adalah kumpulan data yang terorganisir. Dalam konteks aplikasi PyQt5, database biasanya digunakan untuk menyimpan data yang dihasilkan atau digunakan oleh aplikasi, seperti data pengguna, data transaksi, dan lainnya. PyQt5 mendukung berbagai jenis database melalui modul QtSql, termasuk SQLite, PostgreSQL, MySQL, dan lainnya. kita dapat melakukan operasi database seperti membuat tabel, memasukkan data, mengambil data, memperbarui data, dan menghapus data dari database.

9.2.2.1 SQLite

SQLite adalah sistem manajemen database relasional (RDBMS) yang disimpan dalam satu file di disk. Ini adalah perangkat lunak open source dan salah satu sistem database yang paling banyak digunakan di dunia. Berikut adalah beberapa fitur utama SQLite:

- Serverless: SQLite tidak menggunakan arsitektur server-client seperti kebanyakan sistem manajemen database lainnya. Sebaliknya, SQLite membaca dan menulis langsung ke file disk.
- Zero Configuration: SQLite tidak memerlukan konfigurasi atau instalasi terpisah untuk mulai bekerja. Ini membuat SQLite menjadi pilihan yang baik untuk pengembangan aplikasi yang cepat dan prototyping.
- Portable: Database SQLite adalah file disk tunggal yang dapat dengan mudah ditransfer antara sistem.
- Transaksi ACID: SQLite mendukung transaksi ACID (Atomicity, Consistency, Isolation, Durability), yang berarti bahwa semua operasi database adalah atomik dan konsisten, dan mereka tetap demikian bahkan dalam kondisi kegagalan sistem atau crash.

- Mendukung SQL Standar: SQLite mendukung sebagian besar SQL standar, yang memungkinkan pengguna untuk melakukan operasi database kompleks seperti join dan subquery.

SQLite biasanya digunakan dalam aplikasi yang tidak memerlukan skala besar dan di mana kecepatan dan simplicitas lebih penting daripada fitur-fitur lanjutan seperti replikasi dan partisi. Contoh penggunaan SQLite termasuk aplikasi desktop, aplikasi mobile, dan aplikasi IoT.

9.2.2.2 Koneksi ke Database

Untuk melakukan koneksi ke database kita dapat menggunakan beberapa class yang sudah disediakan oleh PyQt5 seperti QSqlDatabase dan QsqlQuery. Berikut ini merupakan contoh kode koneksi PyQt5 ke database SQLite

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. from PyQt5.QtSql import QSqlDatabase, QSqlQuery
3. 
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.         self.setWindowTitle('Main Window')
8. 
9.         # Membuat koneksi ke database SQLite
10.         self.db = QSqlDatabase.addDatabase("QSQLITE")
11.         self.db.setDatabaseName("database.sqlite")  # Database dis
    impan dalam file disk
12. 
13.         # Membuka koneksi ke database
14.         if not self.db.open():
15.             print("Failed to open database")
16. 
17.         # Membuat tabel dalam database
18.         query = QSqlQuery()
19.         query.exec_("CREATE TABLE people (id INTEGER PRIMARY KEY,
    name TEXT)")
20. 
21.         # Menambahkan data ke tabel
22.         query.exec_("INSERT INTO people (name) VALUES ('Setiawan A
    rif')")
23. 
```

```
        # Mengambil data dari tabel
        query.exec_("SELECT * FROM people")
        while query.next():
            print(query.value(1))

if __name__ == "__main__":
    app = QApplication([])
    main_window = MainWindow()
    main_window.show()
    app.exec_()
```

Dalam kode ini, kita membuat class MainWindow yang merupakan subclass dari QMainWindow. Di dalam constructor MainWindow, kita membuat koneksi ke database SQLite dengan menggunakan class QSqlDatabase. Kita kemudian membuka koneksi ke database bernama database.sqlite, membuat tabel dalam database, menambahkan data ke tabel, dan mengambil data dari tabel. Semua operasi database ini dilakukan dengan menggunakan class QSqlQuery.

9.2.2.3 Operasi CRUD

CRUD adalah singkatan dari Create, Read, Update, dan Delete. Ini adalah empat operasi dasar yang dapat dilakukan pada data dalam database.

Create: Membuat atau menambahkan data baru ke dalam database.

```
INSERT INTO people (name) VALUES ('John Doe');
```

Read: Membaca atau mengambil data dari database.

```
SELECT * FROM people;
```

Update: Memperbarui atau mengubah data yang sudah ada dalam database.

```
UPDATE people SET name = 'Jane Doe' WHERE id = 1;
```

Delete: Menghapus data dari database.

```
DELETE FROM people WHERE id = 1;
```

Berikut ini merupakan contoh aplikasi CRUD menggunakan PyQT5

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow
2. from PyQt5.QtSql import QSqlDatabase, QSqlQuery
3. 
4. class MainWindow(QMainWindow):
5.     def __init__(self):
6.         super().__init__()
7.         self.setWindowTitle('Main Window')
8. 
9.         # Membuat koneksi ke database SQLite
10.         self.db = QSqlDatabase.addDatabase("QSQLITE")
11.         self.db.setDatabaseName("database.sqlite")
12. 
13.         # Membuka koneksi ke database
14.         if not self.db.open():
15.             print("Failed to open database")
16. 
17.         # Membuat tabel dalam database (Create)
18.         query = QSqlQuery()
19.         query.exec_("CREATE TABLE people (id INTEGER PRIMARY KEY,
    name TEXT)")
20. 
21.         # Menambahkan data ke tabel (Create)
22.         query.exec_("INSERT INTO people (name) VALUES ('Setiawan
   Arif')")
23. 
24.         # Mengambil data dari tabel (Read)
25.         query.exec_("SELECT * FROM people")
26.         while query.next():
27.             for i in range(2): #range(2) karena terdapat 2 kolom
   dalam database
28.                 print(query.value(i))
29. 
30.         # Memperbarui data dalam tabel (Update)
31.         query.exec_("UPDATE people SET name = 'Jane Doe' WHERE id
    = 1")
32. 
33.         # Menghapus data dari tabel (Delete)
34.         query.exec_("DELETE FROM people WHERE id = 1")
35. 
36. if __name__ == "__main__":
37.     app = QApplication([])
38.     main_window = MainWindow()
```

```
    main_window.show()
    app.exec_()

```

9.2.3 Widget dalam operasi CRUD

Seiring dengan meningkatnya kompleksitas aplikasi yang dibuat, maka terdapat beberapa widget tambahan yang perlu kita ketahui untuk memudahkan dalam membangun aplikasi CRUD

9.2.3.1 QTableWidget

QTableView adalah widget yang digunakan untuk menampilkan data dalam format tabel. QTableView menggunakan model untuk mengatur dan mengakses data. Model yang paling sering digunakan adalah QStandardItemModel dan QSqlTableModel. Berikut adalah contoh penggunaan QTableView:

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableView
from PyQt5.QtGui import QStandardItemModel, QStandardItem

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # Membuat model
        self.model = QStandardItemModel()
        self.model.setHorizontalHeaderLabels(['Name', 'Age'])

        # Menambahkan data ke model
        names = ['Alice', 'Bob', 'Charlie']
        ages = [25, 32, 18]
        for i in range(3):
            name_item = QStandardItem(names[i])
            age_item = QStandardItem(str(ages[i]))
            self.model.appendRow([name_item, age_item])

        # Membuat QTableView dan mengatur modelnya
        self.table = QTableView()
        self.table.setModel(self.model)

        self.setCentralWidget(self.table)
```

```
25.
26.if __name__ == "__main__":
27.    app = QApplication([])
28.    window = MainWindow()
29.    window.show()
30.    app.exec_()
       31.
```

9.2.3.2 QListWidget

QListWidget adalah widget yang digunakan untuk menampilkan data dalam format daftar. Berbeda dengan QTableView, QListWidget tidak menggunakan model dan lebih sederhana untuk digunakan. Berikut adalah contoh penggunaan QlistWidget :

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QListWidget
2.
3. class MainWindow(QMainWindow):
4.     def __init__(self):
5.         super().__init__()
6.
7.         # Membuat QListWidget
8.         self.list_widget = QListWidget()
9.
10.        # Menambahkan item ke QListWidget
11.        self.list_widget.addItem('Alice')
12.        self.list_widget.addItem('Bob')
13.        self.list_widget.addItem('Charlie')
14.
15.        self.setCentralWidget(self.list_widget)
16.
17.if __name__ == "__main__":
18.    app = QApplication([])
19.    window = MainWindow()
20.    window.show()
21.    app.exec_()
```

9.2.3.3 QDialog

QDialog adalah kelas dasar untuk dialog. Dialog adalah jendela independen yang digunakan untuk tugas-tugas sementara yang memerlukan interaksi pengguna. Berikut adalah contoh penggunaan QDialog:

```
1. from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog, QPu
   shButton, QVBoxLayout
2. 
3. class CustomDialog(QDialog):
4.     def __init__(self):
5.         super().__init__()
6.         self.setWindowTitle("Hello!")
7.         layout = QVBoxLayout()
8.         button = QPushButton("Close")
9.         button.clicked.connect(self.close)
10.         layout.addWidget(button)
11.         self.setLayout(layout)
12. 
13.class MainWindow(QMainWindow):
14.     def __init__(self):
15.         super().__init__()
16.         button = QPushButton("Open Dialog")
17.         button.clicked.connect(self.open_dialog)
18.         self.setCentralWidget(button)
19. 
20.     def open_dialog(self):
21.         dialog = CustomDialog()
22.         dialog.exec_()
23. 
24.if __name__ == "__main__":
25.     app = QApplication([])
26.     window = MainWindow()
27.     window.show()
28.     app.exec_()
```

9.2.3.4 QMessageBox

QMessageBox adalah dialog modal yang menyediakan pesan standar. Ini digunakan untuk menampilkan pesan, pertanyaan, atau meminta pengguna untuk memasukkan informasi. Berikut adalah contoh penggunaan QMessageBox:

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox
, QPushButton

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        button = QPushButton("Show Message")
        button.clicked.connect(self.show_message)
        self.setCentralWidget(button)

    def show_message(self):
        QMessageBox.information(self, "Hello!", "This is a message
.")

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec_()
```

9.2.3.5 QDateTimeEdit

QDateTimeEdit adalah widget yang memungkinkan pengguna untuk memilih dan memasukkan tanggal dan waktu. Berikut adalah contoh penggunaan QDateTimeEdit:

```
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QDa
teTimeEdit
from PyQt5.QtCore import QDateTime


class DateTimeEditExample(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QDateTimeEdit Example')

        layout = QVBoxLayout()

        datetime_edit = QDateTimeEdit()
        datetime_edit.setDateTime(QDateTime.currentDateTime())  # M
engatur tanggal dan waktu awal ke tanggal dan waktu saat ini
```

```
        layout.addWidget(datetime_edit)

        self.setLayout(layout)


if __name__ == "__main__":
    app = QApplication([])
    window = DateTimeEditExample()
    window.show()
    app.exec()
```

9.3 Kegiatan Praktikum

9.3.1 Kegiatan 1 : Data dan Form

1. Buat sebuah file program baru kemudian tulis kode program berikut ini

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Main Window')

        # Membuat widget utama
        widget = QWidget()
        self.setCentralWidget(widget)

        # Membuat layout utama
        layout = QVBoxLayout()
        widget.setLayout(layout)

        # Membuat input nama
        self.name_input = QLineEdit()
        layout.addWidget(self.name_input)

        # Membuat tombol kirim
        send_button = QPushButton('Kirim')
        send_button.clicked.connect(self.send_data)
        layout.addWidget(send_button)

        # Membuat instance widget
```

```
        self.widget = Widget()

    def send_data(self):
        # Mengambil data nama dari input
        name = self.name_input.text()
        # Mengirim data nama ke widget
        self.widget.receive_data(name)
        # Menampilkan widget
        self.widget.show()


class Widget(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Widget')

        # Membuat layout
        layout = QVBoxLayout()
        self.setLayout(layout)

        # Membuat label untuk menampilkan data nama
        self.name_label = QLabel()
        layout.addWidget(self.name_label)

    def receive_data(self, name):
        # Menampilkan data nama pada label
        self.name_label.setText(f'Halo, {name}!')


if __name__ == '__main__':
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec()
```

2. Amati hasilnya kemudian tulis analisis singkat mengenai kegiatan ini

9.3.2 Kegiatan 2 : Studi Kasus Aplikasi Kontak

1. Buat beberapa file berikut dalam satu folder

Nama File : **main.py**

```
import sys
from PyQt5.QtWidgets import QApplication
from contact_app import ContactApp

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = ContactApp()
    window.show()
    sys.exit(app.exec_())
```

Nama File : **contact.py**

```
class Contact:
    def __init__(self, name, phone, email):
        self.name = name
        self.phone = phone
        self.email = email
```

Nama File : **contact_app.py**

```
import sys
from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout, QHB
oxLayout, QLabel, QLineEdit, QPushButton, QMessageBox, QListWidget
from PyQt5.QtSql import QSqlDatabase, QSqlQuery
from contact import Contact

class ContactApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Daftar Kontak')
        self.setGeometry(200, 200, 400, 300)

        # Membuat koneksi database SQLite
        self.db = QSqlDatabase.addDatabase('QSQLITE')
        self.db.setDatabaseName('contact.sqlite')
        if not self.db.open():
            QMessageBox.critical(self, 'Error', 'Tidak dapat terhu
bung ke database.')
            sys.exit(1)

        # Membuat tabel kontak jika belum ada
```

```
        query = QSqlQuery()
        query.exec_('CREATE TABLE IF NOT EXISTS contacts (name TEXT, phone TEXT, email TEXT)')

        # Membuat widget utama
        widget = QWidget()
        self.setCentralWidget(widget)

        # Membuat layout utama
        layout = QVBoxLayout()
        widget.setLayout(layout)

        # Membuat label dan inputan
        name_label = QLabel('Nama:')
        self.name_input = QLineEdit()
        phone_label = QLabel('Nomor Telepon:')
        self.phone_input = QLineEdit()
        email_label = QLabel('Email:')
        self.email_input = QLineEdit()

        # Membuat tombol-tombol
        button_layout = QHBoxLayout()
        add_button = QPushButton('Tambah')
        add_button.clicked.connect(self.add_contact)
        update_button = QPushButton('Perbarui')
        update_button.clicked.connect(self.update_contact)
        delete_button = QPushButton('Hapus')
        delete_button.clicked.connect(self.delete_contact)
        button_layout.addWidget(add_button)
        button_layout.addWidget(update_button)
        button_layout.addWidget(delete_button)

        # Menambahkan widget ke dalam layout utama
        layout.addWidget(name_label)
        layout.addWidget(self.name_input)
        layout.addWidget(phone_label)
        layout.addWidget(self.phone_input)
        layout.addWidget(email_label)
        layout.addWidget(self.email_input)
        layout.addLayout(button_layout)

        # Menambahkan daftar kontak
        self.contact_list = QListWidget()
        self.contact_list.itemClicked.connect(self.show_contact)
        layout.addWidget(self.contact_list)

```

```
65.         # Memuat daftar kontak dari database
66.         self.load_contacts()
67.
68.     def load_contacts(self):
69.         # Mengambil daftar kontak dari database dan menampilkannya
70.         query = QSqlQuery()
71.         query.exec_('SELECT name FROM contacts')
72.         self.contact_list.clear()
73.         while query.next():
74.             name = query.value(0)
75.             self.contact_list.addItem(name)
76.
77.     def add_contact(self):
78.         # Menambahkan kontak baru ke database
79.         name = self.name_input.text()
80.         phone = self.phone_input.text()
81.         email = self.email_input.text()
82.         contact = Contact(name, phone, email)
83.
84.         query = QSqlQuery()
85.         query.prepare('INSERT INTO contacts (name, phone, email) V
    ALUES (?, ?, ?)')
86.         query.bindValue(0, name)
87.         query.bindValue(1, phone)
88.         query.bindValue(2, email)
89.
90.         if query.exec_():
91.             self.load_contacts()
92.             self.clear_inputs()
93.             QMessageBox.information(self, 'Info', 'Kontak berhasil
     ditambahkan.')
94.         else:
95.             QMessageBox.warning(self, 'Peringatan', 'Gagal menamba
    hkan kontak.')
96.
97.     def update_contact(self):
98.         # Mengupdate kontak yang dipilih di database
99.         current_item = self.contact_list.currentItem()
100.            if current_item:
101.                selected_name = current_item.text()
102.                name = self.name_input.text()
103.                phone = self.phone_input.text()
104.                email = self.email_input.text()
105.                contact = Contact(name, phone, email)
106.
107.                query = QSqlQuery()
```

```
108.                 query.prepare('UPDATE contacts SET name=?, phone=?,
    email=? WHERE name=?')
109.                 query.bindValue(0, name)
110.                 query.bindValue(1, phone)
111.                 query.bindValue(2, email)
112.                 query.bindValue(3, selected_name)
113.
114.                 if query.exec_():
115.                     self.load_contacts()
116.                     self.clear_inputs()
117.                     QMessageBox.information(self, 'Info', 'Kontak b
   erhasil diperbarui.')
118.                 else:
119.                     QMessageBox.warning(self, 'Peringatan', 'Gagal
   memperbarui kontak.')
120.             else:
121.                 QMessageBox.warning(self, 'Peringatan', 'Pilih kont
   ak terlebih dahulu.')
122.
123.         def delete_contact(self):
124.             # Menghapus kontak yang dipilih dari database
125.             current_item = self.contact_list.currentItem()
126.             if current_item:
127.                 selected_name = current_item.text()
128.                 confirm = QMessageBox.question(self, 'Konfirmasi',
   f'Anda yakin ingin menghapus kontak {selected_name}?', QMessageBox
   .Yes | QMessageBox.No)
129.                 if confirm == QMessageBox.Yes:
130.                     query = QSqlQuery()
131.                     query.prepare('DELETE FROM contacts WHERE name=
   ?')
132.                     query.bindValue(0, selected_name)
133.
134.                     if query.exec_():
135.                         self.load_contacts()
136.                         self.clear_inputs()
137.                         QMessageBox.information(self, 'Info', 'Kont
   ak berhasil dihapus.')
138.                     else:
139.                         QMessageBox.warning(self, 'Peringatan', 'Ga
   gal menghapus kontak.')
140.             else:
141.                 QMessageBox.warning(self, 'Peringatan', 'Pilih kont
   ak terlebih dahulu.')
142.
143.         def show_contact(self, item):
```

```
144.             # Menampilkan informasi kontak saat diklik
145.             selected_name = item.text()
146.             query = QSqlQuery()
147.             query.prepare('SELECT * FROM contacts WHERE name=?')
148.             query.bindValue(0, selected_name)
149.
150.             if query.exec_() and query.next():
151.                 self.name_input.setText(query.value(0))
152.                 self.phone_input.setText(query.value(1))
153.                 self.email_input.setText(query.value(2))
154.
155.         def clear_inputs(self):
156.             # Mengosongkan inputan
157.             self.name_input.clear()
158.             self.phone_input.clear()
159.             self.email_input.clear()
160.
```

2. Klik Run pada main.py untuk menjalankan aplikasi ini

9.4 Tugas

Berdasarkan pada Praktikum Kegiatan 3, Identifikasi dan jelaskan widget, modul PyQt5 class yang digunakan dalam membangun aplikasi daftar kontak