

**LAPORAN PRAKTIKUM TUGAS BESAR
PROYEK STRUKTUR DATA & ALGORITMA**



Disusun Oleh:

Kelompok 1

- | | |
|---------------------------|-------------|
| 1. Yovanza Villareal | (G1A023054) |
| 2. Robi Septian Subhan | (G1A023060) |
| 3. Rahma Hidayati Fitrah | (G1A023074) |
| 4. Khalisa Rizgita Amanda | (G1A023080) |

Kelas : 2B

Nama Asisten Dosen :

- | | |
|----------------------------|-------------|
| 1. Davi Sulaiman | (G1A022001) |
| 2. Atiyya Dianti Fadli | (G1A022002) |
| 3. Abdi Agung Kurniawan | (G1A022011) |
| 4. Sophina Shafa Salsabila | (G1A022021) |
| 5. Evelyn Funiqe Aritonang | (G1A022024) |
| 6. Diodo Arrahman | (G1A022027) |
| 7. Sinta Ezra Wati Gulo | (G1A022040) |
| 8. Wahyu Ozorah Manurung | (G1A022060) |
| 9. Alif Nurhidayat | (G1A022073) |
| 10. Ahmad Radesta | (G1A022086) |

Dosen Pengampu :

1. Arie Vatesia, S.T, M.TI, P.hD
2. Mochammad Yusa, S.kom., M.kom.

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS BENGKULU

2024

KATA PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh

Segala puji bagi Allah SWT yang telah memberikan kami kemudahan sehingga kami dapat menyelesaikan Laporan Praktikum ini dengan tepat waktu. Tanpa pertolonganNya tentunya kami tidak akan sanggup untuk menyelesaikan laporan ini dengan baik. Shalawat serta salam semoga terlimpah curahkan kepada baginda tercinta kita yaitu Nabi Muhammad SAW yang kita nanti-nantikan syafa'atnya di akhirat nanti. Penulis mengucapkan syukur kepada Allah SWT atas limpahan nikmat sehat-Nya, baik itu berupa sehat fisik maupun akal pikiran, sehingga kami mampu untuk menyelesaikan pembuatan Laporan Praktikum sebagai tugas besar membuat Sistem Informasi Perpustakaan dari mata kuliah Proyek Struktur Data dan Algoritma, prodi Informatika, Universitas Bengkulu.

Kami tentu menyadari bahwa Laporan ini masih jauh dari kata sempurna dan masih banyak terdapat kesalahan serta kekurangan di dalamnya. Untuk itu, kami mengharapkan kritik serta saran dari pembaca untuk Laporan ini, supaya Laporan ini nantinya dapat menjadi lebih baik lagi. Demikian, dan apabila terdapat banyak kesalahan pada makalah ini penulis mohon maaf yang sebesar-besarnya. Kami juga mengucapkan terima kasih kepada semua pihak khususnya kepada Dosen Pengampu, Ibu Arie Vatesia, S.T, M.TI, P.hD dan Pak Mochammad Yusa, S.kom., M.kom. Serta juga kepada Asisten Dosen Mba Atiyya Dianti Fadli (G1A022002) dan Abang Alif Nurhidayat (G1A022073) yang telah membimbing kami dalam menulis Laporan Praktikum ini. Demikian, semoga laporan ini dapat bermanfaat. Terima kasih.

Bengkulu, 6 Mei 2024

Kelompok 1

DAFTAR ISI

KATA PENGANTAR	i
BAB 1 PENDAHULUAN	1
A. Latar Belakang.....	1
B. Rumusan Masalah	1
C. Tujuan Dan Manfaat.....	1
BAB 2 LANDASAN TEORI	2
A. Pendeskripsian Sistem Informasi	2
B. Pendeskripsian C++ dan QT	2
BAB 3 PEMBAHASAN	4
BAB 4 KESIMPULAN DAN SARAN	47
A. Kesimpulan.....	47
B. Saran	47
DAFTAR PUSTAKA	48
LAMPIRAN.....	49

BAB I

PENDAHULUAN

A. Latar Belakang

Berkaitan dengan akhir semester kedua untuk Program Studi Informatika, Fakultas Teknik, Universitas Bengkulu. Kami diwajibkan untuk melaksanakan Tugas Besar pada mata kuliah Proyek Struktur Data dan Algoritma yang mana sudah menjadi pilihan kami diawal saat pengisian KRS, dimana proses secara teorinya yang telah diserap dan dipelajari dan senantiasa dapat diterapkan dalam pembuatan Sistem Informasi, salah satunya Sistem Informasi Perpustakaan menggunakan Binary Search ini. Saat ini perkembangann teknologi sudah sangat pesat, sehingga banyak perpustakaan yang sudah memanfaatkan Sistem Informasi berbasis komputer di dalam mendata buku di perpustakaannya. Karena dengan adanya Sistem Informasi kegiatan-kegiatan yang sebelumnya dilakukan secara manual dan membutuhkan waktu yang tidak sedikit, sekarang dapat dilakukan menggunakan komputer dengan waktu yang lebih singkat. Komputerisasi menggunakan Sistem Informasi bukanlah hal yang sulit, karena Sistem Informasi dibuat dengan tujuan untuk memudahkan pekerjaan. Sehingga desain antarmuka suatu Sistem Informasi dirancang agar para pengguna dapat mengerti dan tidak mengalami kesulitan dalam menggunakan Sistem Informasi tersebut.

B. Rumusan Masalah

1. Bagaimana membuat Sistem Informasi Perpustakaan?
2. Apakah sistem informasi dapat dijalankan?

C. Tujuan dan Manfaat

Tujuan dari pembuatan Sistem Informasi Perpustakaan ini adalah

1. Supaya mampu menerapkan hal yang telah dipelajari dalam perkuliahan dalam penggunaan di masyarakat.
2. Agar penulis dapat menambah wawasan dalam pekerjaan yang diaplikasikan antara teori dan praktek.
3. Untuk membantu dalam mengecek data buku perpustakaan

BAB II

LANDASAN TEORI

A. Pendeskripsian Sistem Informasi

Sistem informasi adalah kombinasi dari berbagai komponen yaitu: manusia, fasilitas atau alat teknologi, media, prosedur dan pengendalian yang ditujukan untuk mengatur jaringan komunikasi yang penting, proses transaksi tertentu dan rutin, membantu manajemen dan pemakai internal maupun eksternal serta menyediakan dasar untuk pengambilan keputusan yang tepat (Nash & Martin, 1984).

Penerapan sistem informasi di dalam suatu organisasi bertujuan untuk memberikan dukungan informasi yang dibutuhkan, khususnya oleh para pengguna informasi dari berbagai tingkatan manajemen. Sistem informasi merupakan kombinasi dari berbagai tingkatan manajemen, organisasi, dan teknologi yaitu (Seddon, Staples, Patnayakuni & Bowtell, 1999):

1. *Dimensi Manajemen*, sistem informasi meliputi: kepemimpinan, strategi, dan perilaku manajemen.
2. *Dimensi Teknologi*, terdiri dari: peranti keras, peranti lunak komputer, teknologi manajemen data dan teknologi jaringan atau telekomunikasi (termasuk internet).
3. *Dimensi Organisasi*, merupakan sistem informasi melibatkan hirarki organisasi, keahlian fungsional, proses bisnis budaya, dan kelompok politisi. (Taty, 2016).

B. Pendeskripsian C++ dan QT

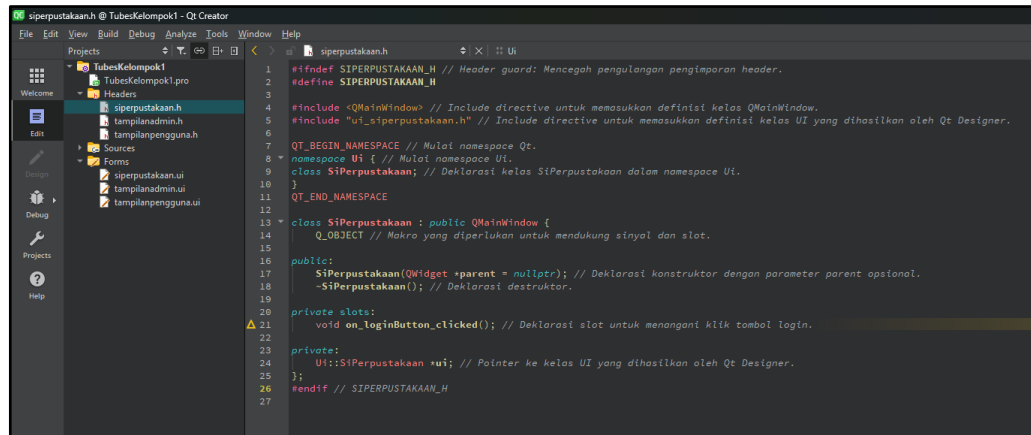
Bahasa Pemrograman C++ merupakan Bahasa yang sudah lama dan populer di dalam pembelajaran pada computer dan Programmer yang ingin mengembangkan system software ataupun membuat game. Oleh sebab itu pemahaman akan bahasa pemrograman C++ ini akan sangat penting terlebih lagi ketika mahasiswa pada jurusan – jurusan yang berkaitan dengan teknologi ataupun programming. Pembelajaran Bahasa Pemrograman C++ secara teori maupun praktikum ini mempunyai peranan penting untuk meningkatkan

pengetahuan mahasiswa tentang bahasa pemrograman disamping mahasiswa belajar bahasa pemrograman mahasiswa secara otomatis mempelajari tata cara penulisan Bahasa pemrograman dimana salahsatu dari tipe data dari Bahasa pemrograman tersebut yaitu integer (atau bilangan bulat). Bilangan bulat bertanda ini dapat mepresentasikan nilai bilangan bulat negatif sedangkan yang tidak bertanda adakah kebalikan dari bilangan bulat yang bertanda. Bilangan bulat bertanda (positif) di dalam computer adalah rantaiian bit, yg dimana rantaiian ini menggunakan bilangan biner. Urutan pada bit-bit yang muncul pun bervariasi tergantung dari jumlah bit – bit yang di presentasikan atau yang keluar.(Aziz & Adwitya, n.d.)

Struktur proyek dalam Qt Creator terdiri dari beberapa elemen utama, termasuk file proyek, direktori sumber, dan file konfigurasi. File proyek merupakan file konfigurasi proyek utama yang berisi informasi tentang sumber daya proyek, pengaturan kompilasi, dan dependensi. Direktori sumber berisi semua file sumber yang digunakan dalam proyek, termasuk file header, implementasi, dan sumber daya. File konfigurasi seperti file .pro digunakan untuk mengonfigurasi proyek dan mendefinisikan properti seperti dependensi eksternal dan opsi kompilasi. Antarmuka pengguna Qt Creator dirancang untuk memberikan pengalaman pengembangan yang intuitif dan produktif. Antarmuka pengguna Qt Creator terdiri dari beberapa komponen utama, termasuk editor kode, alat desain UI, navigator proyek, dan debugger. Editor kode menyediakan lingkungan penulisan kode yang kuat, sedangkan alat desain UI memungkinkan pengembang untuk merancang antarmuka pengguna secara visual. Navigator proyek menyediakan akses cepat ke semua file dan direktori proyek, sementara debugger memungkinkan analisis dan debug yang efisien dari kode. Salah satu keunggulan Qt Creator adalah integrasinya yang erat dengan Qt framework. Qt Creator menyediakan dukungan penuh untuk semua fitur dan fungsionalitas Qt, termasuk manajemen objek, manajemen memori, dan tata letak GUI. Ini memungkinkan pengembang untuk mengembangkan aplikasi Qt dengan cepat dan efisien tanpa perlu bergantung pada alat pengembangan eksternal.

BAB III

PEMBAHASAN



Gambar 1 Source code siperpustakaan.h

Source Code:

#ifndef SIPERPUSTAKAAN_H // Header guard: Mencegah pengulangan pengimporan header.

#define SIPERPUSTAKAAN_H

#include <QMainWindow> // Include directive untuk memasukkan definisi kelas QMainWindow.

#include "ui_siperpustakaan.h" // Include directive untuk memasukkan definisi kelas UI yang dihasilkan oleh Qt Designer.

QT_BEGIN_NAMESPACE // Mulai namespace Qt.

namespace Ui { // Mulai namespace Ui.

class SiPerpustakaan; // Deklarasi kelas SiPerpustakaan dalam namespace Ui.
}

QT_END_NAMESPACE

class SiPerpustakaan : public QMainWindow {

Q_OBJECT // Makro yang diperlukan untuk mendukung sinyal dan slot.

```

public:

    SiPerpustakaan(QWidget *parent = nullptr); // Deklarasi konstruktor dengan
parameter parent opsional.

    ~SiPerpustakaan(); // Deklarasi destruktork.

private slots:

    void on_loginButton_clicked(); // Deklarasi slot untuk menangani klik
tombol login.

private:

    Ui::SiPerpustakaan *ui; // Pointer ke kelas UI yang dihasilkan oleh Qt
Designer.
};

#endif // SIPERPUSTAKAAN_H

```

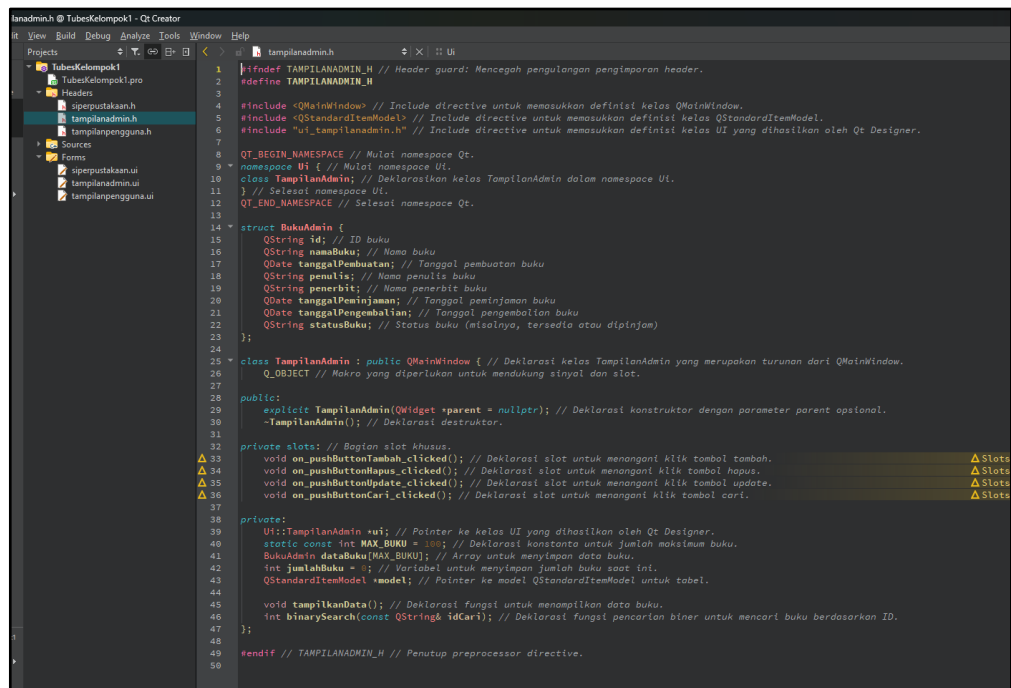
Penjelasan Code:

Kode di atas merupakan bagian dari sebuah header file (**siperpustakaan.h**) dalam sebuah proyek aplikasi menggunakan Qt, sebuah framework untuk pengembangan aplikasi GUI. Pertama-tama, terdapat penggunaan header guard (**#ifndef SIPERPUSTAKAAN_H** dan **#define SIPERPUSTAKAAN_H**) untuk mencegah pengulangan pengimporan header. Ini penting agar definisi kelas dan fungsi yang ada dalam header tidak diimpor secara berulang kali dalam satu kompilasi, yang dapat mengakibatkan kesalahan kompilasi.

Kemudian, header ini mengimpor definisi kelas **QMainWindow** dari Qt (**<QMainWindow>**) yang akan digunakan sebagai kelas utama aplikasi. Selain itu, terdapat juga pengimporan dari file header **"ui_siperpustakaan.h"**, yang merupakan definisi dari kelas UI yang dihasilkan oleh Qt Designer. Ini menunjukkan bahwa proyek menggunakan Qt Designer untuk merancang antarmuka pengguna.

Selanjutnya, terdapat definisi dari kelas **SiPerpustakaan** yang merupakan turunan dari kelas **QMainWindow**. Di dalamnya terdapat **konstruktor** dan **destruktork**, serta **slot (private slots)** yang akan menangani sinyal dari antarmuka

pengguna, seperti klik tombol login. Selain itu, terdapat pointer ke kelas UI yang dihasilkan oleh Qt Designer (`Ui::SiPerpustakaan *ui`), yang akan digunakan untuk mengakses elemen antarmuka pengguna yang telah dirancang. Dengan demikian, header ini menyediakan kerangka kerja dasar untuk aplikasi siperpustakaan yang menggunakan Qt.



Gambar 2 Source code tampilanadmin.h

Source Code:

```

#ifndef TAMPILANADMIN_H // Header guard: Mencegah pengulangan
pengimporan header.
#define TAMPILANADMIN_H

#include <QMainWindow> // Include directive untuk memasukkan definisi
kelas QMainWindow.
#include <QStandardItemModel> // Include directive untuk memasukkan
definisi kelas QStandardItemModel.
#include "ui_tampilanadmin.h" // Include directive untuk memasukkan definisi
kelas UI yang dihasilkan oleh Qt Designer.

QT_BEGIN_NAMESPACE // Mulai namespace Qt.

```

```

namespace Ui { // Mulai namespace Ui.
class TampilanAdmin; // Deklarasikan kelas TampilanAdmin dalam namespace
Ui.
} // Selesai namespace Ui.
QT_END_NAMESPACE // Selesai namespace Qt.

struct BukuAdmin {
    QString id; // ID buku
    QString namaBuku; // Nama buku
    QDate tanggalPembuatan; // Tanggal pembuatan buku
    QString penulis; // Nama penulis buku
    QString penerbit; // Nama penerbit buku
    QDate tanggalPeminjaman; // Tanggal peminjaman buku
    QDate tanggalPengembalian; // Tanggal pengembalian buku
    QString statusBuku; // Status buku (misalnya, tersedia atau dipinjam)
};

class TampilanAdmin : public QMainWindow { // Deklarasi kelas
TampilanAdmin yang merupakan turunan dari QMainWindow.
    Q_OBJECT // Makro yang diperlukan untuk mendukung sinyal dan slot.

public:
    explicit TampilanAdmin(QWidget *parent = nullptr); // Deklarasi
konstruktor dengan parameter parent opsional.
    ~TampilanAdmin(); // Deklarasi destruktur.

private slots: // Bagian slot khusus.
    void on_pushButtonTambah_clicked(); // Deklarasi slot untuk menangani
klik tombol tambah.
    void on_pushButtonHapus_clicked(); // Deklarasi slot untuk menangani klik
tombol hapus.

```

```

    void on_pushButtonUpdate_clicked(); // Deklarasi slot untuk menangani klik
tombol update.

    void on_pushButtonCari_clicked(); // Deklarasi slot untuk menangani klik
tombol cari.

private:
    Ui::TampilanAdmin *ui; // Pointer ke kelas UI yang dihasilkan oleh Qt
Designer.

    static const int MAX_BUKU = 100; // Deklarasi konstanta untuk jumlah
maksimum buku.

    BukuAdmin dataBuku[MAX_BUKU]; // Array untuk menyimpan data buku.
    int jumlahBuku = 0; // Variabel untuk menyimpan jumlah buku saat ini.
    QStandardItemModel *model; // Pointer ke model QStandardItemModel
untuk tabel.

    void tampilkanData(); // Deklarasi fungsi untuk menampilkan data buku.
    int binarySearch(const QString& idCari); // Deklarasi fungsi pencarian biner
untuk mencari buku berdasarkan ID.
};

#endif // TAMPILANADMIN_H // Penutup preprocessor directive.

```

Penjelasan Code:

Kode di atas adalah bagian dari sebuah header file (**tampilanadmin.h**) dalam proyek aplikasi Qt yang berfungsi sebagai tampilan administrasi. Pertama-tama, digunakan header guard (**#ifndef TAMPILANADMIN_H** dan **#define TAMPILANADMIN_H**) untuk mencegah pengulangan pengimporan header. Ini adalah praktik yang umum dilakukan dalam pemrograman C++ untuk menghindari konflik definisi.

Header tersebut mengimpor beberapa definisi kelas dari Qt, termasuk **QMainWindow** dan **QStandardItemModel**, yang akan digunakan dalam tampilan administrasi. Selain itu, juga diimpor definisi kelas **UI** yang dihasilkan oleh Qt

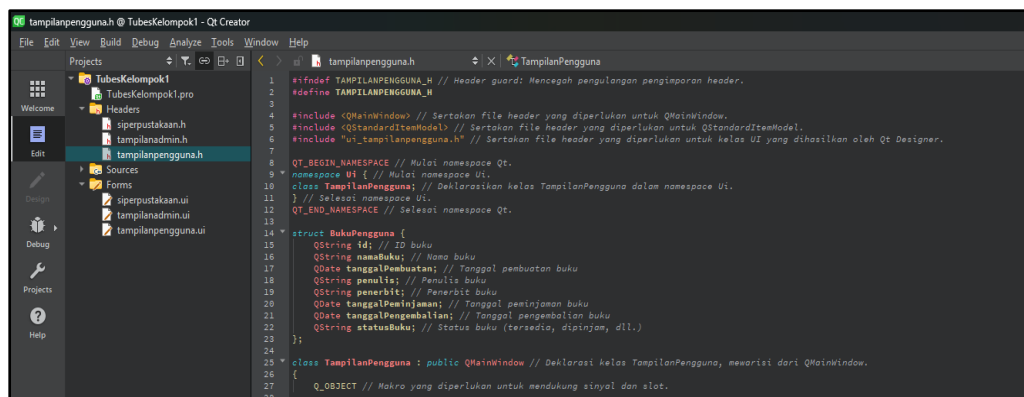
Designer melalui file header "`ui_tampilanadmin.h`". Hal ini menunjukkan bahwa tampilan admin diimplementasikan menggunakan Qt Designer.

Selanjutnya, terdapat definisi struktur '`BukuAdmin`', yang merupakan representasi data untuk buku dalam tampilan admin. Struktur ini menyimpan informasi seperti ID buku, nama buku, penulis, penerbit, dan tanggal peminjaman serta pengembalian. Ini memberikan kerangka kerja untuk menyimpan dan mengelola informasi buku dalam aplikasi.

Kelas '`TampilanAdmin`' merupakan turunan dari '`QMainWindow`' dan menyediakan fungsionalitas utama untuk tampilan administrasi. Di dalamnya, terdapat beberapa slot (private slots) yang akan menangani sinyal dari antarmuka pengguna, seperti klik tombol tambah, hapus, update, dan cari. Hal ini memungkinkan interaksi pengguna dengan data buku yang ditampilkan dalam antarmuka.

Selain itu, kelas ini juga memiliki beberapa variabel dan metode tambahan, seperti array '`dataBuku`' untuk menyimpan informasi buku, variabel '`jumlahBuku`' untuk melacak jumlah buku saat ini, serta model '`QStandardItemModel`' untuk menampilkan data dalam tabel. Metode '`tampilkanData()`' digunakan untuk menampilkan data buku dalam antarmuka pengguna, sementara metode '`binarySearch()`' merupakan implementasi pencarian biner untuk mencari buku berdasarkan ID.

Akhirnya, penutup preprocessor directive (`#endif`) menandakan akhir dari definisi dalam header file. Ini menyelesaikan penggunaan header guard dan menutup file header secara aman. Dengan demikian, header ini menyediakan kerangka kerja yang lengkap untuk tampilan administrasi dalam aplikasi Qt yang memungkinkan pengelolaan data buku.



```
1 #ifndef TAMPILANPENGGUNA_H // Header guard: Hencegah pengulangan pengimporan header.
2 #define TAMPILANPENGGUNA_H
3
4 #include <QMainWindow> // Sertakan file header yang diperlukan untuk QMainWindow.
5 #include <QStandardItemModel> // Sertakan file header yang diperlukan untuk QStandardItemModel.
6 #include "ui_tampilanpengguna.h" // Sertakan file header yang diperlukan untuk kelas UI yang dihasilkan oleh Qt Designer.
7
8 QT_BEGIN_NAMESPACE // Mulai namespace Qt.
9 namespace Ui { // Mulai namespace Ui.
10     class TampilanPengguna; // Deklarasikan kelas TampilanPengguna dalam namespace Ui.
11 } // Selesai namespace Ui.
12 QT_END_NAMESPACE // Selesai namespace Qt.
13
14 struct BukuPengguna {
15     QString id; // ID buku
16     QString namaBuku; // Nama buku
17     QDate tanggalPembuatan; // Tanggal pembuatan buku
18     QString penulis; // Penulis buku
19     QString penerbit; // Penerbit buku
20     QDate tanggalPeminjaman; // Tanggal peminjaman buku
21     QDate tanggalPengembalian; // Tanggal pengembalian buku
22     QString statusBuku; // Status buku (tersedia, dipinjam, dll.)
23 };
24
25 class TampilanPengguna : public QMainWindow // Deklarasi kelas TampilanPengguna, mewarisi dari QMainWindow.
26 {
27     Q_OBJECT // Makro yang diperlukan untuk mendukung sinyal dan slot.
28 }
```

```

27 Q_OBJECT // Makro yang diperlukan untuk mendukung sinyal dan slot.
28
29 public:
30     explicit TampilanPengguna(QWidget *parent = nullptr); // Deklarasi konstruktor dengan parameter parent opsional.
31     ~TampilanPengguna(); // Deklarasi destruktur.
32
33     // Fungsi-fungsi untuk akses ke data buku.
34     BukuPengguna getBuku(int index) const; // Mendapatkan informasi buku berdasarkan indeks.
35     int getJumlahBuku() const; // Mendapatkan jumlah total buku.
36
37     // Fungsi-fungsi untuk operasi-operasi pada data buku.
38     void tambahBuku(const BukuPengguna& buku); // Menambahkan buku baru.
39     void hapusBuku(int index); // Menghapus buku berdasarkan indeks.
40     void updateBuku(int index, const BukuPengguna& buku); // Memperbarui informasi buku.
41     int cariBuku(const QString& idCari) const; // Mencari buku berdasarkan ID.
42
43     // Fungsi untuk menampilkan data.
44     QStandardItemModel* tampilkanData() const; // Menampilkan data buku dalam tabel.
45
46 private slots: // Bagian slot khusus.
47     void on_pushButtonTambah_clicked(); // Deklarasi slot untuk menangani klik tombol tambah.
48     void on_pushButtonHapus_clicked(); // Deklarasi slot untuk menangani klik tombol hapus.
49     void on_pushButtonUpdate_clicked(); // Deklarasi slot untuk menangani klik tombol update.
50     void on_pushButtonCari_clicked(); // Deklarasi slot untuk menangani klik tombol cari.
51
52 private:
53     Ui::TampilanPengguna *ui; // Pointer ke kelas UI yang dihasilkan oleh Qt Designer.
54     static const int MAX_BUKU = 100; // Batasan jumlah maksimum buku.
55     BukuPengguna dataBuku[MAX_BUKU]; // Array untuk menyimpan data buku.
56     int jumlahBuku = 0; // Jumlah buku saat ini.
57     QStandardItemModel *model; // Pointer ke model QStandardItemModel untuk tabel.
58
59     void tampilkanData(); // Menampilkan data buku dalam tabel.
60     int binarySearch(const QString& idCari); // Pencarian biner untuk mencari buku berdasarkan ID.
61 };
62
63 #endif // TAMPILANPENGGUNA_H // Penutup preprocessor directive.
64

```

Gambar 3 Source code tampilanpengguna.h

Source Code:

#ifndef TAMPILANPENGGUNA_H // Header guard: Mencegah pengulangan pengimporan header.

#define TAMPILANPENGGUNA_H

#include <QMainWindow> // Sertakan file header yang diperlukan untuk QMainWindow.

#include <QStandardItemModel> // Sertakan file header yang diperlukan untuk QStandardItemModel.

#include "ui_tampilanpengguna.h" // Sertakan file header yang diperlukan untuk kelas UI yang dihasilkan oleh Qt Designer.

QT_BEGIN_NAMESPACE // Mulai namespace Qt.

namespace Ui { // Mulai namespace Ui.

class TampilanPengguna; // Deklarasikan kelas TampilanPengguna dalam namespace Ui.

} // Selesai namespace Ui.

QT_END_NAMESPACE // Selesai namespace Qt.

```

struct BukuPengguna {
    QString id; // ID buku
    QString namaBuku; // Nama buku
    QDate tanggalPembuatan; // Tanggal pembuatan buku
    QString penulis; // Penulis buku
    QString penerbit; // Penerbit buku
    QDate tanggalPeminjaman; // Tanggal peminjaman buku
    QDate tanggalPengembalian; // Tanggal pengembalian buku
    QString statusBuku; // Status buku (tersedia, dipinjam, dll.)
};

class TampilanPengguna : public QMainWindow // Deklarasi kelas
TampilanPengguna, mewarisi dari QMainWindow.
{
    Q_OBJECT // Makro yang diperlukan untuk mendukung sinyal dan slot.

public:
    explicit TampilanPengguna(QWidget *parent = nullptr); // Deklarasi
konstruktor dengan parameter parent opsional.
    ~TampilanPengguna(); // Deklarasi destruktork.

    // Fungsi-fungsi untuk akses ke data buku.
    BukuPengguna getBuku(int index) const; // Mendapatkan informasi buku
berdasarkan indeks.
    int getJumlahBuku() const; // Mendapatkan jumlah total buku.

    // Fungsi-fungsi untuk operasi-operasi pada data buku.
    void tambahBuku(const BukuPengguna& buku); // Menambahkan buku baru.
    void hapusBuku(int index); // Menghapus buku berdasarkan indeks.
    void updateBuku(int index, const BukuPengguna& buku); // Memperbarui
informasi buku.

```

```

    int cariBuku(const QString& idCari) const; // Mencari buku berdasarkan ID.

    // Fungsi untuk menampilkan data.
    QStandardItemModel* tampilkanData() const; // Menampilkan data buku
    dalam tabel.

private slots: // Bagian slot khusus.
    void on_pushButtonTambah_clicked(); // Deklarasi slot untuk menangani
    klik tombol tambah.
    void on_pushButtonHapus_clicked(); // Deklarasi slot untuk menangani klik
    tombol hapus.
    void on_pushButtonUpdate_clicked(); // Deklarasi slot untuk menangani klik
    tombol update.
    void on_pushButtonCari_clicked(); // Deklarasi slot untuk menangani klik
    tombol cari.

private:
    Ui::TampilanPengguna *ui; // Pointer ke kelas UI yang dihasilkan oleh Qt
    Designer.
    static const int MAX_BUKU = 100; // Batasan jumlah maksimum buku.
    BukuPengguna dataBuku[MAX_BUKU]; // Array untuk menyimpan data
    buku.
    int jumlahBuku = 0; // Jumlah buku saat ini.
    QStandardItemModel *model; // Pointer ke model QStandardItemModel
    untuk tabel.

    void tampilkanData(); // Menampilkan data buku dalam tabel.
    int binarySearch(const QString& idCari); // Pencarian biner untuk mencari
    buku berdasarkan ID.
};

#endif // TAMPILANPENGGUNA_H // Penutup preprocessor directive.

```

Penjelasan Code:

Kode di atas adalah bagian dari sebuah header file (**tampilanpengguna.h**) dalam proyek aplikasi Qt yang bertujuan sebagai tampilan untuk pengguna. Seperti pada header file sebelumnya, penggunaan header guard (**#ifndef TAMPILANPENGGUNA_H** dan **#define TAMPILANPENGGUNA_H**) juga digunakan di sini untuk mencegah pengulangan pengimporan header yang sama.

Header tersebut mengimpor beberapa definisi kelas dari Qt yang diperlukan, seperti **QMainWindow** dan **QStandardItemModel**, serta mengimpor definisi kelas UI yang dihasilkan oleh Qt Designer melalui file header **"ui_tampilanpengguna.h"**.

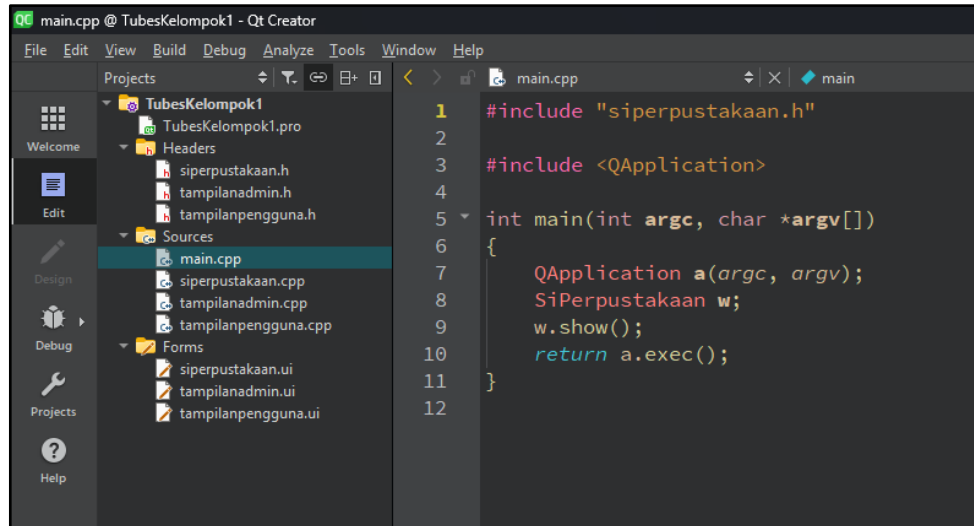
Selanjutnya, terdapat definisi struktur **'BukuPengguna'** yang digunakan untuk merepresentasikan data buku dalam tampilan pengguna. Struktur ini memiliki atribut-atribut seperti **ID buku**, **nama buku**, **penulis**, **penerbit**, **tanggal peminjaman** dan **pengembalian**, serta **status buku**. Ini memungkinkan pengelolaan informasi buku dalam aplikasi dengan lebih terstruktur.

Kelas **'TampilanPengguna'** merupakan turunan dari **'QMainWindow'** dan menyediakan fungsi-fungsi untuk mengakses dan mengelola data buku dalam tampilan pengguna. Ada fungsi-fungsi untuk mendapatkan buku, menambahkan buku baru, menghapus buku, memperbarui informasi buku, dan mencari buku berdasarkan **ID**.

Selain itu, terdapat slot (**private slots**) yang akan menangani sinyal dari antarmuka pengguna, seperti klik tombol **tambah**, **hapus**, **update**, dan **cari**. Ini memungkinkan interaksi pengguna dengan data buku yang ditampilkan dalam antarmuka.

Kelas ini juga memiliki beberapa variabel dan metode tambahan, seperti array **'dataBuku'** untuk menyimpan informasi buku, variabel **'jumlahBuku'** untuk melacak jumlah buku saat ini, serta model **'QStandardItemModel'** untuk menampilkan data dalam tabel. Metode **'tampilkanData()'** digunakan untuk menampilkan data buku dalam antarmuka pengguna, sementara metode **'binarySearch()'** merupakan implementasi pencarian biner untuk mencari buku berdasarkan **ID**.

Akhirnya, penutup preprocessor directive (**#endif**) menandakan akhir dari definisi dalam header file. Ini menyelesaikan penggunaan header guard dan menutup file header secara aman. Dengan demikian, header ini menyediakan kerangka kerja yang lengkap untuk tampilan pengguna dalam aplikasi Qt yang memungkinkan pengelolaan data buku dengan berbagai operasi.



Gambar 4 Source code main.cpp

Source Code:

```
#include "siperpustakaan.h"
```

```
#include <QApplication>
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    QApplication a(argc, argv);
```

```
    SiPerpustakaan w;
```

```
    w.show();
```

```
    return a.exec();
```

```
}
```

Penjelasan Code:

Kode di atas adalah bagian dari fungsi **'main()'** dalam aplikasi yang menggunakan Qt. Pertama-tama, dilakukan inklusi dari file header **"siperpustakaan.h"**, yang merupakan header file dari kelas utama aplikasi, **'SiPerpustakaan'**. Kemudian, dilakukan inklusi dari header file **<QApplication>**

yang merupakan bagian dari Qt dan diperlukan untuk menjalankan aplikasi GUI. Dalam fungsi `main()`, sebuah objek `QApplication` ('a') dibuat dengan parameter `argc` dan `argv` yang diteruskan dari baris perintah. Selanjutnya, sebuah objek `SiPerpustakaan` ('w') dibuat, yang merupakan jendela utama aplikasi, dan ditampilkan menggunakan metode `show()`. Terakhir, fungsi `exec()` dari objek `QApplication` ('a') dipanggil untuk menjalankan event loop aplikasi, yang menangani event dan interaksi pengguna.

```

siperpustakaan.cpp @ TubesKelompok1 - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
Projects
  TubesKelompok1
    TubesKelompok1.pro
    Headers
      siperpustakaan.h
      tampilanadmin.h
      tampilanpengguna.h
    Sources
      main.cpp
      siperpustakaan.cpp
      tampilanadmin.cpp
      tampilanpengguna.cpp
    Forms
      siperpustakaan.ui
      tampilanadmin.ui
      tampilanpengguna.ui
  Design
  Welcome
  Edit
  Debug
  Projects
  Help

siperpustakaan.cpp
1 #include "siperpustakaan.h" // Mengimpor file header SiPerpustakaan.h yang berisi deklarasi kelas SiPerpustakaan.
2 #include "tampilanadmin.h" // Mengimpor file header tampilanadmin.h yang berisi deklarasi kelas TampilanAdmin.
3 #include "tampilanpengguna.h" // Mengimpor file header tampilanpengguna.h yang berisi deklarasi kelas TampilanPengguna.
4 #include "ui_siperpustakaan.h" // Mengimpor file header ui_SiPerpustakaan.h yang berisi deklarasi kelas Ui::SiPerpustakaan.
5 #include <QMessageBox> // Mengimpor header file QMessageBox yang diperlukan untuk menampilkan pesan dialog.
6
7 // Implementasi konstruktor SiPerpustakaan
8 SiPerpustakaan::SiPerpustakaan(QWidget *parent)
9     : QMainWindow(parent), ui(new Ui::SiPerpustakaan)
10 {
11     ui->setupUi(this); // Menyalin antarmuka pengguna yang dihasilkan oleh Qt Designer.
12 }
13
14 // Implementasi destruktur SiPerpustakaan
15 SiPerpustakaan::~SiPerpustakaan()
16 {
17     delete ui; // Menghapus antarmuka pengguna yang dihasilkan oleh Qt Designer.
18 }
19
20 // Implementasi slot on_loginbutton_clicked
21 void SiPerpustakaan::on_loginbutton_clicked()
22 {
23     // Mendapatkan username dan password dari inputan pengguna
24     QString username = ui->usernameLineEdit->text();
25     QString password = ui->passwordLineEdit->text();
26
27     // Memeriksa apakah username dan password adalah "admin"
28     if (username == "admin" && password == "admin") {
29         // Menampilkan pesan informasi jika login berhasil sebagai admin
30         QMessageBox::information(this, "Login", "Login sebagai admin berhasil!");
31
32         // Membuat instance baru dari TampilanAdmin dan menampilkannya
33         TampilanAdmin *tampilanAdmin = new TampilanAdmin();
34         tampilanAdmin->show();
35
36         // Menutup jendela login
37         this->close();
38     }
39     // Memeriksa apakah username dan password adalah "user"
40     else if (username == "user" && password == "user") {
41         // Menampilkan pesan informasi jika login berhasil sebagai user
42         QMessageBox::information(this, "Login", "Login sebagai user berhasil!");
43
44         // Membuat instance baru dari TampilanPengguna dan menampilkannya
45         TampilanPengguna *tampilanPengguna = new TampilanPengguna();
46         tampilanPengguna->show();
47
48         // Menutup jendela login
49         this->close();
50     }
51     // Memeriksa jika username atau password tidak sesuai
52     else {
53         // Menampilkan pesan peringatan jika login gagal
54         QMessageBox::warning(this, "Login", "Username atau password salah!");
55     }
56 }
57

```

Gambar 5 Source code siperpustakaan.cpp

Source Code:

```

#include "siperpustakaan.h" // Mengimpor file header SiPerpustakaan.h yang
berisi deklarasi kelas SiPerpustakaan.
#include "tampilanadmin.h" // Mengimpor file header tampilanadmin.h yang
berisi deklarasi kelas TampilanAdmin.
#include "tampilanpengguna.h" // Mengimpor file header tampilanpengguna.h
yang berisi deklarasi kelas TampilanPengguna.
#include "ui_siperpustakaan.h" // Mengimpor file header ui_SiPerpustakaan.h
yang berisi deklarasi kelas Ui::SiPerpustakaan.
#include <QMessageBox> // Mengimpor header file QMessageBox yang
diperlukan untuk menampilkan pesan dialog.

```

```

// Implementasi konstruktor SiPerpustakaan
SiPerpustakaan::SiPerpustakaan(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::SiPerpustakaan)
{
    ui->setupUi(this); // Menyiapkan antarmuka pengguna yang dihasilkan oleh
Qt Designer.
}

// Implementasi destruktur SiPerpustakaan
SiPerpustakaan::~SiPerpustakaan()
{
    delete ui; // Menghapus antarmuka pengguna yang dihasilkan oleh Qt
Designer.
}

// Implementasi slot on_loginButton_clicked
void SiPerpustakaan::on_loginButton_clicked()
{
    // Mendapatkan username dan password dari inputan pengguna
    QString username = ui->usernameLineEdit->text();
    QString password = ui->passwordLineEdit->text();

    // Memeriksa apakah username dan password adalah "admin"
    if (username == "admin" && password == "admin") {
        // Menampilkan pesan informasi jika login berhasil sebagai admin
        QMessageBox::information(this, "Login", "Login sebagai admin
berhasil!");

        // Membuat instance baru dari TampilanAdmin dan menampilkannya
        TampilanAdmin *tampilanAdmin = new TampilanAdmin();
        tampilanAdmin->show();
    }
}

```

```

        // Menutup jendela login
        this->close();
    }
    // Memeriksa apakah username dan password adalah "user"
    else if (username == "user" && password == "user") {
        // Menampilkan pesan informasi jika login berhasil sebagai user
        QMessageBox::information(this, "Login", "Login sebagai user berhasil!");

        // Membuat instance baru dari TampilanPengguna dan menampilkannya
        TampilanPengguna *tampilanPengguna = new TampilanPengguna();
        tampilanPengguna->show();

        // Menutup jendela login
        this->close();
    }
    // Memeriksa jika username atau password tidak sesuai
    else {
        // Menampilkan pesan peringatan jika login gagal
        QMessageBox::warning(this, "Login", "Username atau password salah!");
    }
}

```

Penjelasan Code:

Kode di atas mengimplementasikan fungsionalitas dari kelas `'SiPerpustakaan'`, yang merupakan bagian dari aplikasi manajemen perpustakaan. Pertama-tama, dilakukan inklusi dari beberapa file header yang diperlukan, termasuk header dari kelas utama `'SiPerpustakaan'` serta header dari kelas `'TampilanAdmin'` dan `'TampilanPengguna'`, yang akan ditampilkan tergantung pada jenis login. Selain itu, juga diimpor header file `<QMessageBox>` yang diperlukan untuk menampilkan pesan dialog dalam aplikasi.

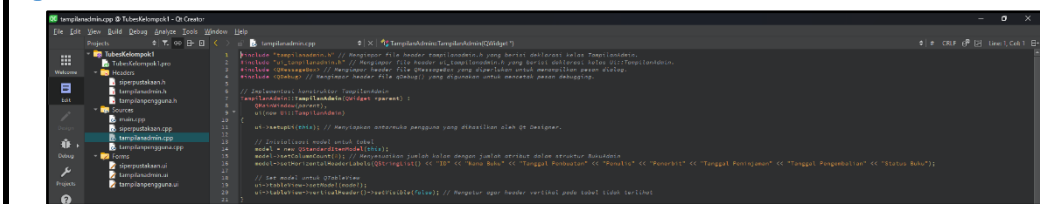
Selanjutnya, dilakukan implementasi dari konstruktor `'SiPerpustakaan'`, di mana antarmuka pengguna yang telah dirancang menggunakan Qt Designer

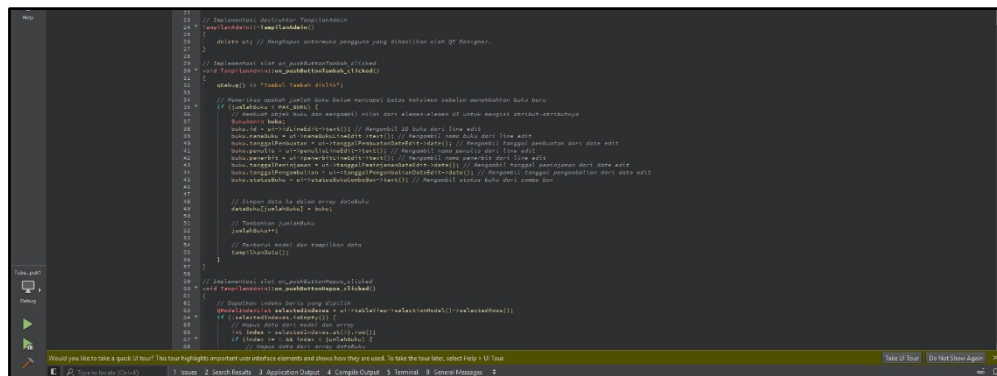
disiapkan menggunakan metode `'setupUi(this)'`. Metode ini membantu menginisialisasi komponen-komponen antarmuka pengguna yang telah ditentukan sebelumnya.

Kemudian, terdapat implementasi dari slot `'on_loginButton_clicked()'`, yang akan dipanggil ketika tombol login ditekan. Di dalam slot ini, username dan password yang dimasukkan oleh pengguna diambil dari elemen antarmuka pengguna. Kemudian, dilakukan pengecekan apakah username dan password sesuai dengan `"admin"` atau `"user"`. Jika sesuai, maka jendela baru `'TampilanAdmin'` atau `'TampilanPengguna'` akan dibuat dan ditampilkan, sesuai dengan jenis login yang berhasil. Jendela login (`'SiPerpustakaan'`) akan ditutup setelah login berhasil

Terakhir, jika username atau password tidak sesuai dengan yang diharapkan, maka pesan peringatan akan ditampilkan menggunakan `'QMessageBox::warning()'`. Pesan ini memberi informasi kepada pengguna bahwa login gagal karena username atau password tidak valid. Dengan demikian, pengguna diberikan umpan balik yang sesuai terkait status login mereka.

Selain menangani proses login, slot `'on_loginButton_clicked()'` juga berfungsi sebagai penghubung antara antarmuka pengguna dengan logika aplikasi. Setiap kali pengguna melakukan login, slot ini memproses informasi yang dimasukkan pengguna dan menentukan akses yang tepat sesuai dengan jenis pengguna. Dengan demikian, penggunaan pesan dialog seperti `'QMessageBox'` untuk memberikan umpan balik kepada pengguna menjadi penting dalam memberikan informasi terkait hasil proses login. Selain itu, pembuatan dan penampilan jendela baru seperti `'TampilanAdmin'` atau `'TampilanPengguna'` menunjukkan penggunaan pola desain Model-View-Controller (MVC) dalam pengembangan aplikasi, di mana antarmuka pengguna (View) dipisahkan dari logika aplikasi (Controller) dan data (Model). Hal ini memungkinkan untuk manajemen dan pengembangan aplikasi yang lebih terstruktur dan mudah dipelihara.





Gambar 6 Source code tampilanadmin.cpp

Source Code:

```
#include "tampilanadmin.h" // Mengimpor file header tampilanadmin.h yang
// berisi deklarasi kelas TampilanAdmin.

#include "ui_tampilanadmin.h" // Mengimpor file header ui_tampilanadmin.h
// yang berisi deklarasi kelas Ui::TampilanAdmin.

#include <QMessageBox> // Mengimpor header file QMessageBox yang
// diperlukan untuk menampilkan pesan dialog.

#include <QDebug> // Mengimpor header file qDebug() yang digunakan untuk
// mencetak pesan debugging.

// Implementasi konstruktor TampilanAdmin
TampilanAdmin::TampilanAdmin(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::TampilanAdmin)
{
    ui->setupUi(this); // Menyiapkan antarmuka pengguna yang dihasilkan oleh
    // Qt Designer.

    // Inisialisasi model untuk tabel
    model = new QStandardItemModel(this);
    model->setColumnCount(8); // Menyesuaikan jumlah kolom dengan jumlah
    // atribut dalam struktur BukuAdmin
```

```

        model->setHorizontalHeaderLabels(QStringList() << "ID" << "Nama Buku"
<< "Tanggal Pembuatan" << "Penulis" << "Penerbit" << "Tanggal
Peminjaman" << "Tanggal Pengembalian" << "Status Buku");

        // Set model untuk QTableView
        ui->tableView->setModel(model);
        ui->tableView->verticalHeader()->setVisible(false); // Mengatur agar header
vertikal pada tabel tidak terlihat
    }

// Implementasi destruktur TampilanAdmin
TampilanAdmin::~TampilanAdmin()
{
    delete ui; // Menghapus antarmuka pengguna yang dihasilkan oleh Qt
Designer.
}

// Implementasi slot on_pushButtonTambah_clicked
void TampilanAdmin::on_pushButtonTambah_clicked()
{
    qDebug() << "Tombol Tambah diklik";

    // Memeriksa apakah jumlah buku belum mencapai batas maksimum sebelum
menambahkan buku baru
    if (jumlahBuku < MAX_BUKU) {
        // Membuat objek buku dan mengambil nilai dari elemen-elemen UI untuk
mengisi atribut-atributnya
        BukuAdmin buku;
        buku.id = ui->idLineEdit->text(); // Mengambil ID buku dari line edit
        buku.namaBuku = ui->namaBukuLineEdit->text(); // Mengambil nama
buku dari line edit
    }
}

```

```

        buku.tanggalPembuatan = ui->tanggalPembuatanDateEdit->date(); //
Mengambil tanggal pembuatan dari date edit
        buku.penulis = ui->penulisLineEdit->text(); // Mengambil nama penulis
dari line edit
        buku.penerbit = ui->penerbitLineEdit->text(); // Mengambil nama penerbit
dari line edit
        buku.tanggalPeminjaman = ui->tanggalPeminjamanDateEdit->date(); //
Mengambil tanggal peminjaman dari date edit
        buku.tanggalPengembalian = ui->tanggalPengembalianDateEdit->date(); //
Mengambil tanggal pengembalian dari date edit
        buku.statusBuku = ui->statusBukuComboBox->text(); // Mengambil status
buku dari combo box

// Simpan data ke dalam array dataBuku
dataBuku[jumlahBuku] = buku;

// Tambahkan jumlahBuku
jumlahBuku++;

// Perbarui model dan tampilkan data
tampilkanData();
    }
}

// Implementasi slot on_pushButtonHapus_clicked
void TampilanAdmin::on_pushButtonHapus_clicked()
{
    // Dapatkan indeks baris yang dipilih
    QModelIndexList selectedIndexes = ui->tableView->selectionModel()-
>selectedRows();
    if (!selectedIndexes.isEmpty()) {

```



```

// Hapus data dari model dan array
int index = selectedIndexes.at(0).row();
if (index >= 0 && index < jumlahBuku) {
    // Hapus data dari array dataBuku
    for (int i = index; i < jumlahBuku - 1; ++i) {
        dataBuku[i] = dataBuku[i + 1];
    }

    // Kurangi jumlahBuku karena satu data telah dihapus
    jumlahBuku--;

    // Tampilkan data yang telah diupdate di dalam QTableView
    tampilkanData();
}
}
}

```

Penjelasan Code:

Kode di atas adalah implementasi dari kelas `'TampilanAdmin'`, yang bertanggung jawab untuk menampilkan tampilan administrasi dalam aplikasi perpustakaan. Pertama-tama, terdapat konstruktor `'TampilanAdmin'` yang melakukan inisialisasi antarmuka pengguna yang telah dirancang menggunakan Qt Designer dengan memanggil metode `'setUpUi(this)'`. Selain itu, dilakukan juga inisialisasi model untuk tabel yang akan menampilkan data buku. Model tersebut diatur untuk memiliki delapan kolom sesuai dengan jumlah atribut dalam struktur `'BukuAdmin'`, dan label-header horizontal ditetapkan menggunakan metode `'setHorizontalHeaderLabels()'`.

Selanjutnya, terdapat implementasi dari destruktur `'TampilanAdmin'` yang bertugas untuk menghapus antarmuka pengguna yang telah dibuat untuk mencegah kebocoran memori.

Kemudian, ada slot `'on_pushButtonTambah_clicked()'` yang dipanggil ketika tombol `"Tambah"` di klik. Di dalam slot ini, terlebih dahulu pesan debugging menggunakan `'QDebug()'` dicetak untuk memantau jalannya aplikasi.

Selanjutnya, terdapat implementasi dari slot `on_pushButtonHapus_clicked()` yang dipanggil ketika tombol "Hapus" di klik. Slot ini bertugas untuk menghapus buku dari model dan array berdasarkan baris yang dipilih dalam `QTableView`. Jika baris dipilih, indeks baris yang dipilih diambil dan data buku pada indeks tersebut dihapus dari array `dataBuku`. Selanjutnya, jumlah buku dikurangi dan model diperbarui untuk mencerminkan perubahan yang terjadi.



```
void TampilanAdmin::on_pushButtonUpdate_clicked() {
    // Dapatkan indeks baris yang dipilih
```

```

QModelIndexList selectedIndexes = ui->tableView->selectionModel()-
>selectedRows();

if (!selectedIndexes.isEmpty()) {
    int index = selectedIndexes.at(0).row();

    // Ambil data dari input teks yang ingin diupdate

    QString id = ui->idLineEdit->text(); // Mengambil ID dari QLineEdit
    QString namaBuku = ui->namaBukuLineEdit->text(); // Mengambil Nama
Buku dari QLineEdit

    QDate tanggalPembuatan = ui->tanggalPembuatanDateEdit->date(); //
Mengambil Tanggal Pembuatan dari QDateEdit

    QString penulis = ui->penulisLineEdit->text(); // Mengambil Penulis dari
QLineEdit

    QString penerbit = ui->penerbitLineEdit->text(); // Mengambil Penerbit
dari QLineEdit

    QDate tanggalPeminjaman = ui->tanggalPeminjamanDateEdit->date(); //
Mengambil Tanggal Peminjaman dari QDateEdit

    QDate tanggalPengembalian = ui->tanggalPengembalianDateEdit->date();
// Mengambil Tanggal Pengembalian dari QDateEdit

    QString statusBuku = ui->statusBukuComboBox->text(); // Mengambil
Status Buku dari QComboBox

    // Update data di dalam array dataBuku
    if (index >= 0 && index < jumlahBuku) {
        BukuAdmin buku;

        buku.id = id;

        buku.namaBuku = namaBuku;

        buku.tanggalPembuatan = tanggalPembuatan;

        buku.penulis = penulis;

        buku.penerbit = penerbit;

        buku.tanggalPeminjaman = tanggalPeminjaman;

        buku.tanggalPengembalian = tanggalPengembalian;

        buku.statusBuku = statusBuku;

        dataBuku[index] = buku;
    }
}

```

```

        // Tampilkan data yang telah diupdate di dalam QTableView
        tampilkanData();
    }
}

// Implementasi fungsi tampilkanData untuk menampilkan data dalam
// QTableView
void TampilanAdmin::tampilkanData() {
    // Bersihkan model sebelum menambahkan data baru
    model->clear();

    // Set header untuk tabel
    model->setHorizontalHeaderLabels(QStringList() << "ID" << "Nama Buku"
    << "Tanggal Pembuatan" << "Penulis" << "Penerbit" << "Tanggal
    Peminjaman" << "Tanggal Pengembalian" << "Status Buku");

    // Melakukan loop melalui setiap data buku dalam array dataBuku
    for (int i = 0; i < jumlahBuku; ++i) {
        // Membuat QList untuk menyimpan pointer QStandardItem untuk setiap
        baris
        QList<QStandardItem *> rowItems;

        // Menambahkan QStandardItem untuk ID buku
        rowItems << new QStandardItem(dataBuku[i].id);

        // Menambahkan QStandardItem untuk nama buku
        rowItems << new QStandardItem(dataBuku[i].namaBuku);

        // Menambahkan QStandardItem untuk tanggal pembuatan buku diformat
        sebagai "dd-MM-yyyy"
        rowItems << new
        QStandardItem(dataBuku[i].tanggalPembuatan.toString("dd-MM-yyyy"));
    }
}

```

```

// Menambahkan QStandardItem untuk penulis buku
rowItems << new QStandardItem(dataBuku[i].penulis);

// Menambahkan QStandardItem untuk penerbit buku
rowItems << new QStandardItem(dataBuku[i].penerbit);

// Menambahkan QStandardItem untuk tanggal peminjaman buku diformat
sebagai "dd-MM-yyyy"
rowItems << new
QStandardItem(dataBuku[i].tanggalPeminjaman.toString("dd-MM-yyyy"));

// Menambahkan QStandardItem untuk tanggal pengembalian buku
diformat sebagai "dd-MM-yyyy"
rowItems << new
QStandardItem(dataBuku[i].tanggalPengembalian.toString("dd-MM-yyyy"));

// Menambahkan QStandardItem untuk status buku
rowItems << new QStandardItem(dataBuku[i].statusBuku);

// Menambahkan rowItems ke model, mewakili satu baris data buku dalam
QTableView
model->appendRow(rowItems);
}
}

// Implementasi fungsi binarySearch untuk pencarian biner
int TampilanAdmin::binarySearch(const QString& idCari) {
    int left = 0;
    int right = jumlahBuku - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;

```

```

// Periksa apakah idCari berada di tengah array
if (dataBuku[mid].id == idCari)
    return mid;
// Jika idCari lebih kecil, cari di sebelah kiri
if (dataBuku[mid].id < idCari)
    left = mid + 1;
// Jika idCari lebih besar, cari di sebelah kanan
else
    right = mid - 1;
}
// Jika tidak ditemukan, kembalikan -1
return -1;
}

// Implementasi slot on_pushButtonCari_clicked untuk pencarian buku
void TampilanAdmin::on_pushButtonCari_clicked()
{
    // Ambil ID buku yang akan dicari dari QLineEdit
    QString idCari = ui->idLineEdit->text();

    // Lakukan pencarian biner
    int index = binarySearch(idCari);

    // Periksa apakah ID buku ditemukan atau tidak
    if (index != -1) {
        // Jika ditemukan, tampilkan pesan informasi
        QMessageBox::information(this, "Pencarian Berhasil", QString("Buku
dengan ID '%1' ditemukan di indeks %2").arg(idCari).arg(index));

        // Pilih baris yang sesuai dengan hasil pencarian di QTableView
        ui->tableView->selectRow(index);
    } else {

```

```

// Jika tidak ditemukan, tampilkan pesan peringatan
QMessageBox::warning(this, "Pencarian Gagal", QString("Buku dengan
ID '%1' tidak ditemukan.").arg(idCari));
}
}

```

Penjelasan Code:

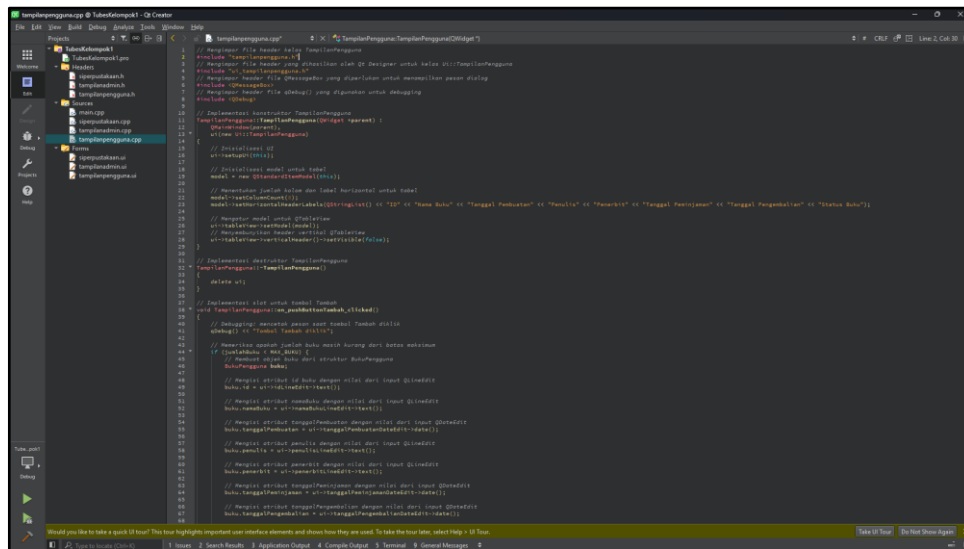
Fungsi ``on_pushButtonUpdate_clicked()`` adalah slot yang dipanggil saat tombol "Update" ditekan dalam antarmuka pengguna. Pada implementasinya, terlebih dahulu indeks baris yang dipilih dalam ``QTableView`` diambil. Kemudian, data buku yang akan diupdate diambil dari elemen-elemen antarmuka pengguna seperti ID, nama buku, tanggal pembuatan, penulis, penerbit, tanggal peminjaman, tanggal pengembalian, dan status buku. Data tersebut kemudian disimpan dalam objek ``buku`` dari struktur ``BukuAdmin`` dan diperbarui dalam array ``dataBuku``. Setelah itu, dipanggil fungsi ``tampilkanData()`` untuk memperbarui tampilan data dalam ``QTableView``.

Fungsi ``tampilkanData()`` bertanggung jawab untuk menampilkan data buku dalam ``QTableView``. Pertama-tama, model tabel dibersihkan dengan memanggil ``clear()`` untuk menghapus data yang telah ada sebelumnya. Selanjutnya, label-header horizontal ditetapkan dan loop dilakukan melalui setiap data buku dalam array ``dataBuku``. Pada setiap iterasi, sebuah `QList` dibuat untuk menyimpan pointer ``QStandardItem`` untuk setiap kolom dalam satu baris data buku. Data buku kemudian ditambahkan ke ``rowItems`` sesuai dengan kolom yang sesuai, dan selanjutnya ``rowItems`` ditambahkan ke model untuk merepresentasikan satu baris data dalam ``QTableView``.

Fungsi ``binarySearch()`` adalah implementasi algoritma pencarian biner untuk mencari buku berdasarkan ID-nya dalam array ``dataBuku``. Pada implementasinya, batas kiri dan kanan ditentukan, dan pencarian dilakukan sampai batas kiri lebih kecil atau sama dengan batas kanan. Jika ID yang dicari ditemukan di tengah array, maka indeksnya dikembalikan; jika tidak, fungsi mengembalikan -1.

Terakhir, fungsi ``on_pushButtonCari_clicked()`` adalah slot yang dipanggil ketika tombol "Cari" ditekan dalam antarmuka pengguna. Pada

implementasinya, ID buku yang akan dicari diambil dari elemen 'QLineEdit', kemudian fungsi 'binarySearch()' dipanggil untuk melakukan pencarian. Jika ID buku ditemukan, maka pesan informasi ditampilkan dengan indeks tempat buku ditemukan dan baris di 'QTableView' dipilih sesuai hasil pencarian. Jika tidak ditemukan, pesan peringatan ditampilkan bahwa buku dengan ID tersebut tidak ditemukan. Dengan demikian, fungsi ini memungkinkan pengguna untuk dengan cepat mencari buku berdasarkan ID-nya dalam antarmuka pengguna.



Gambar 8 Source code tampilanpengguna.cpp

Source Code:

```
// Mengimpor file header kelas TampilanPengguna
#include "tampilanpengguna.h"

// Mengimpor file header yang dihasilkan oleh Qt Designer untuk kelas
Ui::TampilanPengguna
#include "ui_tampilanpengguna.h"

// Mengimpor header file QMessageBox yang diperlukan untuk menampilkan
pesan dialog
#include <QMessageBox>

// Mengimpor header file qDebug() yang digunakan untuk debugging
#include <QDebug>

// Implementasi konstruktor TampilanPengguna
TampilanPengguna::TampilanPengguna(QWidget *parent) :
```



```

QMainWindow(parent),
ui(new Ui::TampilanPengguna)
{
    // Inisialisasi UI
    ui->setupUi(this);

    // Inisialisasi model untuk tabel
    model = new QStandardItemModel(this);

    // Menentukan jumlah kolom dan label horizontal untuk tabel
    model->setColumnCount(8);
    model->setHorizontalHeaderLabels(QStringList() << "ID" << "Nama Buku"
    << "Tanggal Pembuatan" << "Penulis" << "Penerbit" << "Tanggal
    Peminjaman" << "Tanggal Pengembalian" << "Status Buku");

    // Mengatur model untuk QTableView
    ui->tableView->setModel(model);
    // Menyembunyikan header vertikal QTableView
    ui->tableView->verticalHeader()->setVisible(false);
}

// Implementasi destruktur TampilanPengguna
TampilanPengguna::~~TampilanPengguna()
{
    delete ui;
}

// Implementasi slot untuk tombol Tambah
void TampilanPengguna::on_pushButtonTambah_clicked()
{
    // Debugging: mencetak pesan saat tombol Tambah diklik
    qDebug() << "Tombol Tambah diklik";
}

```

```

// Memeriksa apakah jumlah buku masih kurang dari batas maksimum
if (jumlahBuku < MAX_BUKU) {
    // Membuat objek buku dari struktur BukuPengguna
    BukuPengguna buku;

    // Mengisi atribut id buku dengan nilai dari input QLineEdit
    buku.id = ui->idLineEdit->text();

    // Mengisi atribut namaBuku dengan nilai dari input QLineEdit
    buku.namaBuku = ui->namaBukuLineEdit->text();

    // Mengisi atribut tanggalPembuatan dengan nilai dari input QDateEdit
    buku.tanggalPembuatan = ui->tanggalPembuatanDateEdit->date();

    // Mengisi atribut penulis dengan nilai dari input QLineEdit
    buku.penulis = ui->penulisLineEdit->text();

    // Mengisi atribut penerbit dengan nilai dari input QLineEdit
    buku.penerbit = ui->penerbitLineEdit->text();

    // Mengisi atribut tanggalPeminjaman dengan nilai dari input QDateEdit
    buku.tanggalPeminjaman = ui->tanggalPeminjamanDateEdit->date();

    // Mengisi atribut tanggalPengembalian dengan nilai dari input QDateEdit
    buku.tanggalPengembalian = ui->tanggalPengembalianDateEdit->date();

    // Mengisi atribut statusBuku dengan nilai dari input QComboBox
    buku.statusBuku = ui->statusBukuComboBox->text();

    // Simpan data ke dalam array dataBuku

```

```

        dataBuku[jumlahBuku] = buku;

        // Tambahkan jumlahBuku
        jumlahBuku++;

        // Perbarui model dan tampilkan data
        tampilkanData();
    }
}

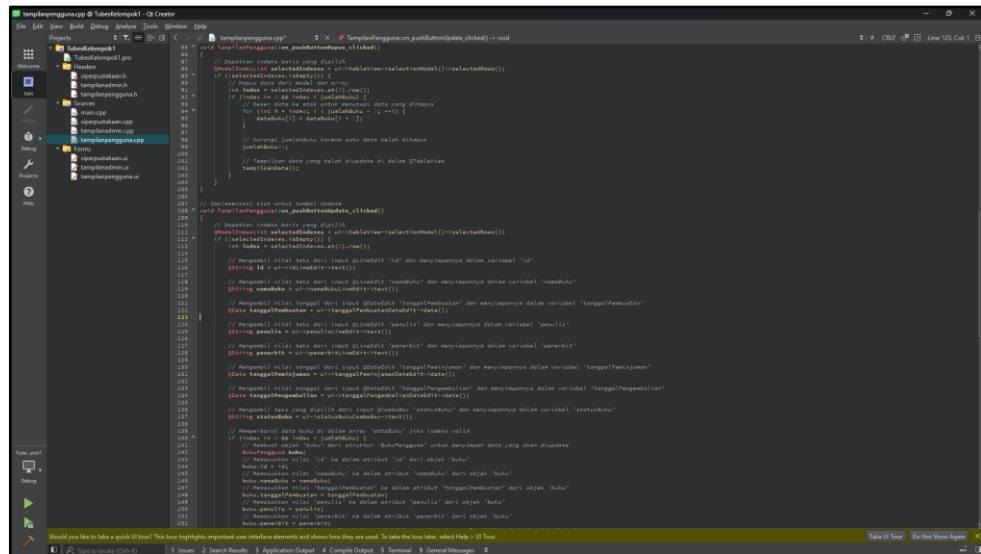
```

Penjelasan Code:

Kode di atas merupakan implementasi dari kelas `'TampilanPengguna'` dalam proyek yang menggunakan Qt untuk pembuatan antarmuka pengguna. Konstruktor `'TampilanPengguna'` digunakan untuk menginisialisasi antarmuka pengguna yang telah dibuat menggunakan Qt Designer. Setelah itu, model untuk tabel yang akan menampilkan data buku diinisialisasi dengan menentukan jumlah kolom dan label horizontalnya. Model ini kemudian diatur untuk digunakan dalam `'QTableView'` yang ada dalam antarmuka pengguna, dengan menyembunyikan header vertikalnya.

Fungsi `'on_pushButtonTambah_clicked()'` merupakan slot yang akan dipanggil ketika tombol **"Tambah"** pada antarmuka pengguna ditekan. Di dalamnya, terlebih dahulu dilakukan pengecekan apakah jumlah buku masih kurang dari batas maksimum yang telah ditentukan. Jika iya, data buku yang diinputkan oleh pengguna diambil dari elemen-elemen antarmuka seperti **ID**, **nama buku**, **tanggal pembuatan**, **penulis**, **penerbit**, **tanggal peminjaman**, **tanggal pengembalian**, dan **status buku**. Data tersebut kemudian disimpan dalam objek `'BukuPengguna'` dan ditambahkan ke dalam array `'dataBuku'`. Setelah itu, dipanggil fungsi `'tampilkanData()'` untuk memperbarui tampilan data dalam `'QTableView'`. Kelas `'TampilanPengguna'` juga memiliki destruktur yang digunakan untuk menghapus memori yang dialokasikan untuk antarmuka pengguna yang telah dibuat. Pada implementasi di atas, terdapat juga beberapa baris kode yang digunakan untuk debugging, seperti pencetakan pesan

menggunakan `qDebug()`, yang membantu dalam melacak proses jalannya program dan memudahkan dalam mengidentifikasi masalah ketika terjadi.



Gambar 9 Source code tampilanpengguna.cpp

Source Code:

```
void TampilanPengguna::on_pushButtonHapus_clicked()
```

```
{
```

```
    // Dapatkan indeks baris yang dipilih
```

```
    QModelIndexList selectedIndexes = ui->tableView->selectionModel()-
```

```
>selectedRows();
```

```
    if (!selectedIndexes.isEmpty()) {
```

```
        // Hapus data dari model dan array
```

```
        int index = selectedIndexes.at(0).row();
```

```
        if (index >= 0 && index < jumlahBuku) {
```

```
            // Geser data ke atas untuk menutupi data yang dihapus
```

```
            for (int i = index; i < jumlahBuku - 1; ++i) {
```

```
                dataBuku[i] = dataBuku[i + 1];
```

```
            }
```

```
            // Kurangi jumlahBuku karena satu data telah dihapus
```

```
            jumlahBuku--;
```

```
            // Tampilkan data yang telah diupdate di dalam QTableView
```

```

        tampilkanData();
    }
}

// Implementasi slot untuk tombol Update
void TampilanPengguna::on_pushButtonUpdate_clicked()
{
    // Dapatkan indeks baris yang dipilih
    QModelIndexList selectedIndexes = ui->tableView->selectionModel()-
>selectedRows();
    if (!selectedIndexes.isEmpty()) {
        int index = selectedIndexes.at(0).row();

        // Mengambil nilai teks dari input QLineEdit 'id' dan menyimpannya
dalam variabel 'id'
        QString id = ui->idLineEdit->text();

        // Mengambil nilai teks dari input QLineEdit 'namaBuku' dan
menyimpannya dalam variabel 'namaBuku'
        QString namaBuku = ui->namaBukuLineEdit->text();

        // Mengambil nilai tanggal dari input QDateEdit 'tanggalPembuatan' dan
menyimpannya dalam variabel 'tanggalPembuatan'
        QDate tanggalPembuatan = ui->tanggalPembuatanDateEdit->date();

        // Mengambil nilai teks dari input QLineEdit 'penulis' dan menyimpannya
dalam variabel 'penulis'
        QString penulis = ui->penulisLineEdit->text();

        // Mengambil nilai teks dari input QLineEdit 'penerbit' dan menyimpannya
dalam variabel 'penerbit'

```

```

QString penerbit = ui->penerbitLineEdit->text();

// Mengambil nilai tanggal dari input QDateTime 'tanggalPeminjaman' dan
menyimpannya dalam variabel 'tanggalPeminjaman'
QDate tanggalPeminjaman = ui->tanggalPeminjamanDateEdit->date();

// Mengambil nilai tanggal dari input QDateTime 'tanggalPengembalian'
dan menyimpannya dalam variabel 'tanggalPengembalian'
QDate tanggalPengembalian = ui->tanggalPengembalianDateEdit->date();

// Mengambil teks yang dipilih dari input QComboBox 'statusBuku' dan
menyimpannya dalam variabel 'statusBuku'
QString statusBuku = ui->statusBukuComboBox->text();

// Memperbarui data buku di dalam array 'dataBuku' jika indeks valid
if (index >= 0 && index < jumlahBuku) {
    // Membuat objek 'buku' dari struktur 'BukuPengguna' untuk
menyimpan data yang akan diupdate
    BukuPengguna buku;

    // Memasukkan nilai 'id' ke dalam atribut 'id' dari objek 'buku'
    buku.id = id;

    // Memasukkan nilai 'namaBuku' ke dalam atribut 'namaBuku' dari
objek 'buku'
    buku.namaBuku = namaBuku;

    // Memasukkan nilai 'tanggalPembuatan' ke dalam atribut
'tanggalPembuatan' dari objek 'buku'
    buku.tanggalPembuatan = tanggalPembuatan;

    // Memasukkan nilai 'penulis' ke dalam atribut 'penulis' dari objek 'buku'
    buku.penulis = penulis;

    // Memasukkan nilai 'penerbit' ke dalam atribut 'penerbit' dari objek
'buku'
    buku.penerbit = penerbit;

```

```

        // Memasukkan nilai 'tanggalPeminjaman' ke dalam atribut
        'tanggalPeminjaman' dari objek 'buku'
        buku.tanggalPeminjaman = tanggalPeminjaman;
        // Memasukkan nilai 'tanggalPengembalian' ke dalam atribut
        'tanggalPengembalian' dari objek 'buku'
        buku.tanggalPengembalian = tanggalPengembalian;
        // Memasukkan nilai 'statusBuku' ke dalam atribut 'statusBuku' dari
        objek 'buku'
        buku.statusBuku = statusBuku;

        // Memperbarui data buku di dalam array 'dataBuku' dengan objek
        'buku' yang sudah diupdate
        dataBuku[index] = buku;
    }
}
}

```

Penjelasan Code:

Dua slot yang didefinisikan dalam kelas `TampilanPengguna` tersebut bertanggung jawab atas operasi penghapusan dan pembaruan data buku dalam antarmuka pengguna.

Pertama, fungsi `on_pushButtonHapus_clicked()` digunakan untuk menghapus data buku yang dipilih oleh pengguna dari antarmuka pengguna. Saat tombol "Hapus" ditekan, indeks baris yang dipilih dalam `QTableView` diambil. Jika ada baris yang dipilih, data dihapus dari model dan array `dataBuku`. Selanjutnya, data di dalam array akan digeser ke atas untuk menutupi data yang dihapus. Setelah itu, jumlah buku dikurangi satu dan antarmuka pengguna diperbarui untuk menampilkan data yang telah dihapus.

Kedua, fungsi `on_pushButtonUpdate_clicked()` digunakan untuk memperbarui data buku yang dipilih dalam antarmuka pengguna. Ketika tombol "Update" ditekan, indeks baris yang dipilih dalam `QTableView` diambil. Kemudian, nilai-nilai dari elemen-elemen antarmuka seperti ID, nama buku, tanggal pembuatan, penulis, penerbit, tanggal peminjaman, tanggal

pengembalian, dan status buku diambil. Data buku pada indeks yang dipilih diupdate dengan nilai-nilai yang baru. Dalam implementasi ini, pembaruan dilakukan dengan membuat objek **'BukuPengguna'** baru dan menggantikan data buku yang lama pada indeks tertentu dengan objek yang baru.

Kedua fungsi tersebut digunakan dalam kelas **'TampilanPengguna'** untuk memungkinkan pengguna mengelola data buku, baik dengan menghapus entri yang tidak diperlukan maupun memperbarui informasi yang sudah ada. Dengan menggunakan slot-slot ini, antarmuka pengguna menjadi lebih interaktif dan memudahkan pengguna dalam melakukan operasi terkait data buku.

```

185 // Implementasi fungsi untuk menampilkan data dalam QTableView
186 void TampilanPengguna::tampilkanData()
187 {
188     // Bersihkan model sebelum menambahkan data baru
189     model->clear();
190
191     // Set header untuk tabel
192     QStringList headerLabels(QStringList() << "ID" << "Nama Buku" << "Tanggal Pemusnahan" << "Penulis" << "Penerbit" << "Tanggal Pengembalian" << "Status Buku");
193     model->setHeaderLabels(headerLabels);
194
195     // Siapkan array sumber data untuk dimasukkan ke dalam model QTableView
196     QList<Buku> data;
197     // Buat daftar untuk menyimpan informasi QTableView dalam satu baris
198     QStringList row;
199     // Menambahkan item QTableView untuk setiap 'buku' ke dalam QTableView
200     foreach (Buku buku in bukuList)
201     {
202         row.clear();
203         row.append(buku.getId());
204         row.append(buku.getNamaBuku());
205         row.append(buku.getTanggalPemusnahan());
206         row.append(buku.getPenulis());
207         row.append(buku.getPenerbit());
208         row.append(buku.getTanggalPengembalian());
209         row.append(buku.getStatusBuku());
210         data.append(Buku(buku.getId(), buku.getNamaBuku(), buku.getTanggalPemusnahan(), buku.getPenulis(), buku.getPenerbit(), buku.getTanggalPengembalian(), buku.getStatusBuku()));
211     }
212     model->insertRows(0, data.count(), QModelIndex());
213 }
214
215 // Implementasi fungsi pencarian buku
216 void TampilanPengguna::lakukanPencarian(QString kata)
217 {
218     int left = 0;
219     int right = data.count() - 1;
220
221     // Melakukan pencarian buku di dalam array sumber data
222     while (left < right)
223     {
224         int mid = (left + right) / 2;
225
226         // Jika kata ditemukan di tengah array, kembalikan indeksnya
227         if (data[mid].getId() == kata)
228             return mid;
229
230         // Jika kata lebih kecil, cari di sebelah kanan
231         if (data[mid].getNamaBuku() < kata)
232             left = mid + 1;
233         // Jika kata lebih besar, cari di sebelah kiri
234         else
235             right = mid - 1;
236     }
237     // Jika tidak ditemukan, kembalikan -1
238     return -1;
239 }
240
241 // Implementasi slot untuk tombol Cari
242 void TampilanPengguna::lakukanPencarian()
243 {
244     // Ambil ID buku yang akan dicari dari QLineEdit
245     QString kata = ui->lineEditPencarian->text();
246
247     // Lakukan pencarian buku
248     int index = lakukanPencarian(kata);
249
250     // Periksa apakah ID buku ditemukan atau tidak
251     if (index != -1)
252     {
253         // Jika ditemukan, tampilkan pesan informasi
254         QMessageBox::information(this, "Pesan ke Berhasil", QString("Buku dengan ID '%0' ditemukan di indeks %1").arg(index).arg(index));
255     }
256     else
257     {
258         // Jika tidak ditemukan, tampilkan pesan peringatan
259         QMessageBox::warning(this, "Pesan ke Gagal", QString("Buku dengan ID '%0' tidak ditemukan").arg(kata));
260     }
261 }

```

Gambar 10 Source code tampilanpengguna.cpp

Source Code:

```

// Implementasi fungsi untuk menampilkan data dalam QTableView
void TampilanPengguna::tampilkanData()
{
    // Bersihkan model sebelum menambahkan data baru
    model->clear();

    // Set header untuk tabel

```



```

    model->setHorizontalHeaderLabels(QStringList() << "ID" << "Nama Buku"
<< "Tanggal Pembuatan" << "Penulis" << "Penerbit" << "Tanggal
Peminjaman" << "Tanggal Pengembalian" << "Status Buku");

    // Iterasi melalui array 'dataBuku' untuk menambahkan data ke dalam model
QStandardItemModel
    for (int i = 0; i < jumlahBuku; ++i) {
        // Membuat QList untuk menyimpan item-item QStandardItem dalam satu
baris
        QList<QStandardItem *> rowItems;
        // Menambahkan item QStandardItem untuk atribut 'id' buku ke dalam
QList
        rowItems << new QStandardItem(dataBuku[i].id);
        // Menambahkan item QStandardItem untuk atribut 'namaBuku' ke dalam
QList
        rowItems << new QStandardItem(dataBuku[i].namaBuku);
        // Menambahkan item QStandardItem untuk atribut 'tanggalPembuatan'
(format tanggal) ke dalam QList
        rowItems << new
QStandardItem(dataBuku[i].tanggalPembuatan.toString("dd-MM-yyyy"));
        // Menambahkan item QStandardItem untuk atribut 'penulis' buku ke
dalam QList
        rowItems << new QStandardItem(dataBuku[i].penulis);
        // Menambahkan item QStandardItem untuk atribut 'penerbit' buku ke
dalam QList
        rowItems << new QStandardItem(dataBuku[i].penerbit);
        // Menambahkan item QStandardItem untuk atribut 'tanggalPeminjaman'
(format tanggal) ke dalam QList
        rowItems << new
QStandardItem(dataBuku[i].tanggalPeminjaman.toString("dd-MM-yyyy"));
        // Menambahkan item QStandardItem untuk atribut 'tanggalPengembalian'
(format tanggal) ke dalam QList

```

```

        rowItems << new
QStandardItem(dataBuku[i].tanggalPengembalian.toString("dd-MM-yyyy"));
        // Menambahkan item QStandardItem untuk atribut 'statusBuku' ke dalam
QList
        rowItems << new QStandardItem(dataBuku[i].statusBuku);
        // Menambahkan baris dengan item-item QStandardItem ke dalam model
        model->appendRow(rowItems);
    }
}

// Implementasi fungsi pencarian biner
int TampilanPengguna::binarySearch(const QString& idCari)
{
    int left = 0;
    int right = jumlahBuku - 1;

    // Melakukan pencarian biner di dalam array dataBuku
    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Jika idCari ditemukan di tengah array, kembalikan indeksnya
        if (dataBuku[mid].id == idCari)
            return mid;

        // Jika idCari lebih kecil, cari di sebelah kanan
        if (dataBuku[mid].id < idCari)
            left = mid + 1;

        // Jika idCari lebih besar, cari di sebelah kiri
        else
            right = mid - 1;
    }
}

```

```

// Jika tidak ditemukan, kembalikan -1
return -1;
}

// Implementasi slot untuk tombol Cari
void TampilanPengguna::on_pushButtonCari_clicked()
{
    // Ambil ID buku yang akan dicari dari QLineEdit
    QString idCari = ui->idLineEdit->text();

    // Lakukan pencarian biner
    int index = binarySearch(idCari);

    // Periksa apakah ID buku ditemukan atau tidak
    if (index != -1) {
        // Jika ditemukan, tampilkan pesan informasi
        QMessageBox::information(this, "Pencarian Berhasil", QString("Buku
dengan ID '%1' ditemukan di indeks %2").arg(idCari).arg(index));

        // Pilih baris yang sesuai dengan hasil pencarian di QTableView
        ui->tableView->selectRow(index);
    } else {
        // Jika tidak ditemukan, tampilkan pesan peringatan
        QMessageBox::warning(this, "Pencarian Gagal", QString("Buku dengan
ID '%1' tidak ditemukan.").arg(idCari));
    }
}

```

Penjelasan Code:

Fungsi `tampilkanData()` digunakan untuk menampilkan data buku dalam `QTableView` pada antarmuka pengguna. Pertama, model yang terkait dengan `QTableView` dibersihkan dengan memanggil fungsi `clear()`.

Kemudian, header untuk tabel ditentukan menggunakan fungsi `'setHorizontalHeaderLabels()'`. Setelah itu, dilakukan iterasi melalui array `'dataBuku'` untuk menambahkan data ke dalam model `'QStandardItemModel'`. Setiap baris data direpresentasikan sebagai sebuah `'QList'` yang berisi `'QStandardItem'` untuk setiap atribut buku. Baris ini kemudian ditambahkan ke dalam model menggunakan `'appendRow()'`.

Fungsi `'binarySearch()'` digunakan untuk melakukan pencarian biner dalam array `'dataBuku'` berdasarkan ID buku. Pencarian dimulai dengan menentukan indeks awal dan akhir dalam array, kemudian dilakukan iterasi hingga indeks awal tidak lagi lebih kecil dari indeks akhir. Pada setiap iterasi, indeks tengah dihitung dan nilai ID di tengah array dibandingkan dengan ID yang dicari. Jika ID ditemukan, indeksnya dikembalikan; jika tidak, pencarian dilanjutkan ke setengah array yang sesuai.

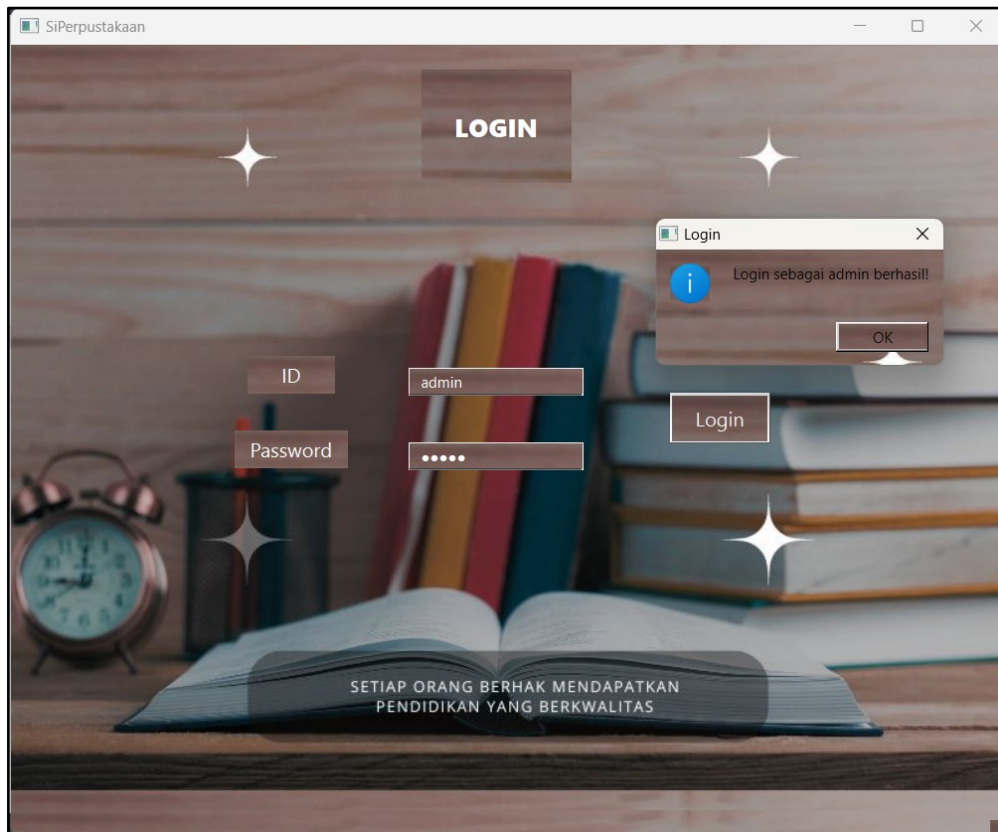
Selanjutnya, slot `'on_pushButtonCari_clicked()'` dipicu saat pengguna menekan tombol "Cari" pada antarmuka pengguna. ID buku yang dicari diambil dari `'QLineEdit'`, dan pencarian biner dilakukan dengan memanggil fungsi `'binarySearch()'`. Hasil pencarian kemudian digunakan untuk menampilkan pesan informasi jika buku ditemukan bersama dengan indeksnya, atau pesan peringatan jika buku tidak ditemukan.

Kedua fungsi tersebut memungkinkan pengguna untuk mencari buku berdasarkan ID dan menampilkan hasil pencarian dalam `'QTableView'`. Dengan adanya fitur pencarian ini, pengguna dapat dengan mudah menemukan buku yang diinginkan dalam daftar yang mungkin besar.

Fungsi `'on_pushButtonHapus_clicked()'` diimplementasikan untuk menghapus data buku dari antarmuka pengguna berdasarkan baris yang dipilih dalam `'QTableView'`. Pertama, indeks baris yang dipilih diperoleh menggunakan `'selectedRows()'`. Jika ada baris yang dipilih, data pada model dan array `'dataBuku'` dihapus dengan menggeser data di atas baris yang dihapus untuk menutupi data yang dihapus. Setelah itu, jumlah buku dikurangi satu dan tampilan data diperbarui dengan memanggil `'tampilkanData()'`.

Selanjutnya, fungsi `'on_pushButtonUpdate_clicked()'` diimplementasikan untuk memperbarui data buku berdasarkan baris yang dipilih dalam

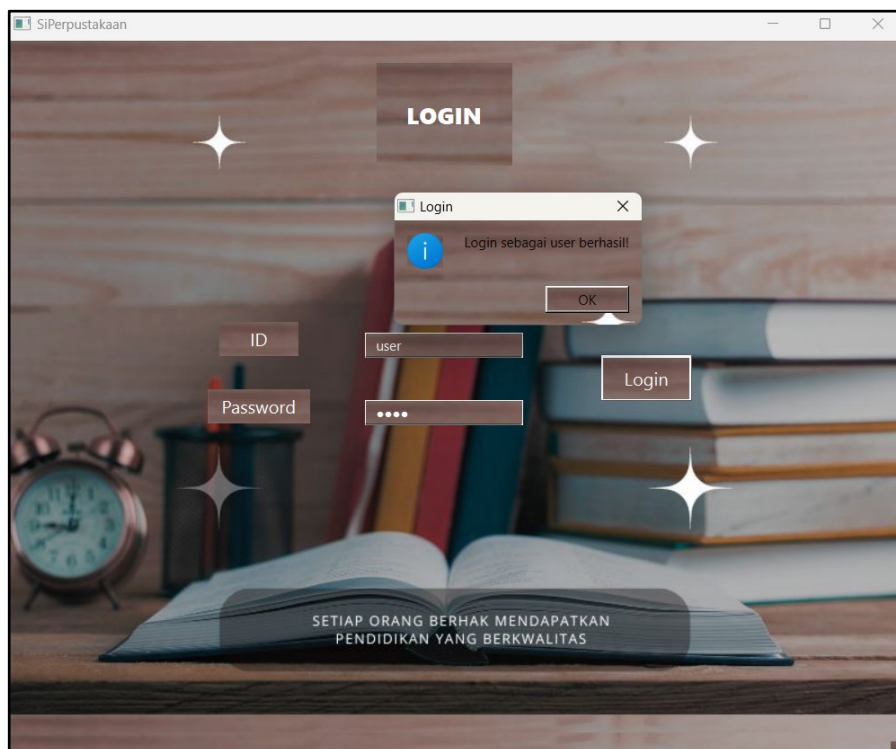
'QTableView'. Jika ada baris yang dipilih, data dari inputan pengguna diambil dan disimpan dalam variabel. Data buku yang ada di array 'dataBuku' diperbarui dengan data yang baru. Ini memungkinkan pengguna untuk memperbarui informasi buku yang telah dimasukkan sebelumnya. Dengan demikian, pengguna dapat mengelola data buku dengan lebih fleksibel, baik untuk menambahkan, menghapus, atau memperbarui informasi buku sesuai kebutuhan.



Gambar 11 Output Login Admin

Penjelasan Output:

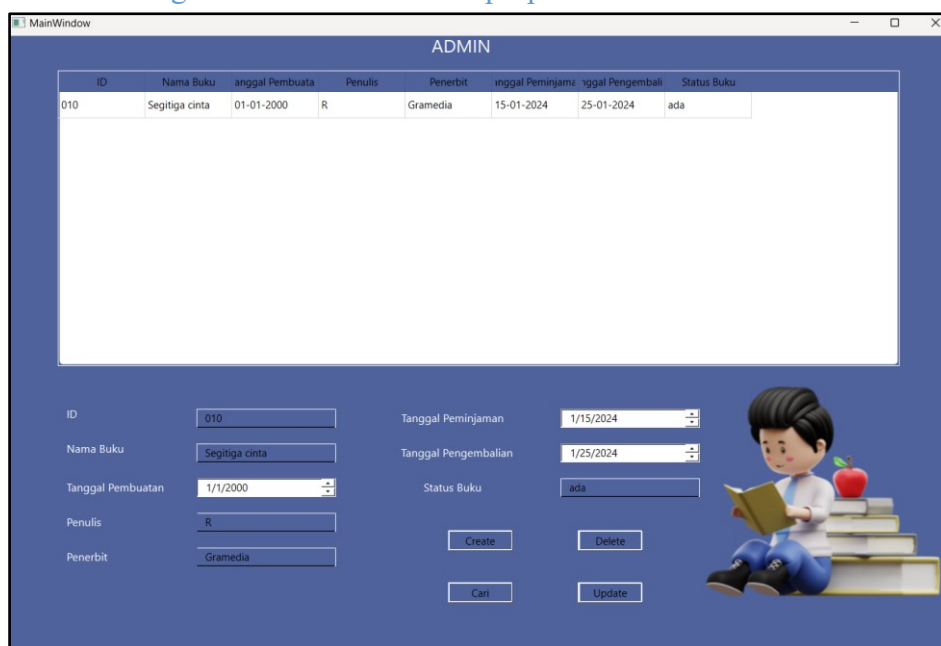
Output ini menunjukkan bahwa ketika login berhasil sebagai admin, pengguna akan diarahkan ke antarmuka admin untuk mengelola data perpustakaan. Hal ini memungkinkan admin untuk melakukan operasi seperti menambah, menghapus, atau memperbarui informasi buku dalam perpustakaan. Selain itu, keseluruhan proses login dan navigasi antarmuka pengguna menjadi lebih intuitif dan mudah digunakan.



Gambar 12 Output Login User

Penjelasan Output:

Ketika pengguna berhasil login sebagai "user", mereka akan dialihkan ke antarmuka TampilanPengguna, yang memungkinkan mereka untuk mengakses fungsi-fungsi tertentu seperti menambahkan, menghapus, atau memperbarui data buku. Ini memberikan pengguna akses terbatas, dengan kemampuan untuk melihat dan mengelola data buku dalam perpustakaan.



Gambar 13 Output Halaman Admin

Penjelasan Output:

Pada tampilan di atas, terdapat implementasi kelas `'TampilanAdmin'` yang bertanggung jawab untuk mengatur antarmuka pengguna untuk admin dalam sistem perpustakaan. Konstruktor `'TampilanAdmin'` digunakan untuk menyiapkan antarmuka pengguna yang telah dibuat menggunakan Qt Designer. Setelah itu, model untuk tabel diinisialisasi dan disesuaikan dengan jumlah kolom yang sesuai dengan atribut dalam struktur `'BukuAdmin'`, lalu diatur sebagai model untuk `'QTableView'`.

Kemudian, terdapat implementasi slot `'on_pushButtonTambah_clicked'` yang akan diaktifkan ketika tombol "Tambah" diklik. Fungsi ini bertugas untuk menambahkan buku baru ke dalam sistem. Jika jumlah buku belum mencapai batas maksimum, data buku diambil dari elemen-elemen antarmuka pengguna dan disimpan dalam array `'dataBuku'`. Jumlah buku kemudian diperbarui dan tampilan data diperbarui dengan memanggil fungsi `'tampilkanData()'`.

Selanjutnya, terdapat implementasi slot `'on_pushButtonHapus_clicked'` yang akan dipanggil saat tombol "Hapus" diklik. Fungsi ini bertanggung jawab untuk menghapus data buku yang dipilih oleh admin dari antarmuka pengguna. Jika ada baris yang dipilih di `'QTableView'`, data buku akan dihapus dari model dan array `'dataBuku'`. Selanjutnya, jumlah buku dikurangi dan tampilan data diperbarui dengan memanggil `'tampilkanData()'`.

Selain itu, terdapat implementasi slot `'on_pushButtonUpdate_clicked'` yang akan diaktifkan saat tombol "Update" diklik. Fungsi ini digunakan untuk memperbarui data buku yang dipilih oleh admin. Data baru diambil dari elemen-elemen antarmuka pengguna, kemudian data buku yang sesuai di array `'dataBuku'` diperbarui. Setelah itu, tampilan data diperbarui dengan memanggil `'tampilkanData()'`.

ID	Nama Buku	tanggal Pembuatan	Penulis	Penerbit	tanggal Peminjaman	tanggal Pengembalian	Status Buku
03	bumi manusia	01-02-2005	herry	erlangga	02-01-2024	06-01-2024	pinjam
010	segitiga cinta	01-01-2000	r	gramedia	15-01-2024	25-01-2024	pinjam

ID:
 Tanggal Peminjaman:

Nama Buku:
 Tanggal Pengembalian:

Tanggal Pembuatan:
 Status Buku:

Penulis:

Penerbit:

Create Cari Update

Gambar 14 Output Halaman User

Tampilan di atas mengimplementasikan kelas `'TampilanPengguna'`, yang bertanggung jawab untuk mengatur antarmuka pengguna bagi pengguna dalam sistem perpustakaan. Konstruktor `'TampilanPengguna'` digunakan untuk menginisialisasi antarmuka pengguna yang telah dibuat menggunakan Qt Designer. Model untuk tabel diinisialisasi dan disesuaikan dengan jumlah kolom yang sesuai dengan atribut dalam struktur `'BukuPengguna'`, lalu diatur sebagai model untuk `'QTableView'`.

Selanjutnya, terdapat implementasi slot `'on_pushButtonTambah_clicked'` yang akan diaktifkan ketika tombol **"Tambah"** diklik oleh pengguna. Fungsi ini bertugas untuk menambahkan buku baru ke dalam sistem. Jika jumlah buku belum mencapai batas maksimum, data buku diambil dari elemen-elemen antarmuka pengguna dan disimpan dalam array `'dataBuku'`. Jumlah buku kemudian diperbarui dan tampilan data diperbarui dengan memanggil fungsi `'tampilkanData()'`.

Selain itu, terdapat implementasi slot `'on_pushButtonHapus_clicked'` yang akan dipanggil saat tombol **"Hapus"** diklik. Fungsi ini bertanggung jawab untuk menghapus data buku yang dipilih oleh pengguna dari antarmuka pengguna. Jika ada baris yang dipilih di `'QTableView'`, data buku akan dihapus

dari model dan array `'dataBuku'`. Selanjutnya, jumlah buku dikurangi dan tampilan data diperbarui dengan memanggil `'tampilkanData()'`.

Kemudian, terdapat implementasi slot `'on_pushButtonUpdate_clicked'` yang akan diaktifkan saat tombol "Update" diklik oleh pengguna. Fungsi ini digunakan untuk memperbarui data buku yang dipilih oleh pengguna. Data baru diambil dari elemen-elemen antarmuka pengguna, kemudian data buku yang sesuai di array `'dataBuku'` diperbarui. Setelah itu, tampilan data diperbarui dengan memanggil `'tampilkanData()'`.

Terakhir, terdapat implementasi fungsi `'tampilkanData()'` yang bertugas untuk menampilkan data dalam `'QTableView'`. Fungsi ini membersihkan model sebelum menambahkan data baru, kemudian iterasi melalui array `'dataBuku'` untuk menambahkan data ke dalam model `'QStandardItemModel'`. Setiap baris data buku direpresentasikan sebagai QList dari item-item `'QStandardItem'`, yang kemudian ditambahkan ke model.

Selain itu, terdapat implementasi fungsi `'binarySearch'` yang digunakan untuk melakukan pencarian biner dalam array `'dataBuku'` berdasarkan ID buku. Fungsi ini menerima parameter berupa ID buku yang akan dicari dan mengembalikan indeks di mana ID buku ditemukan dalam array `'dataBuku'`, atau -1 jika tidak ditemukan. Pencarian biner dilakukan dengan membagi array menjadi dua bagian secara berulang hingga ID buku yang dicari ditemukan atau seluruh array telah dijelajahi. Jika ID buku ditemukan, pesan informasi ditampilkan kepada pengguna bersama dengan indeks di mana buku ditemukan, dan baris yang sesuai dipilih dalam `'QTableView'`.

BAB IV

KESIMPULAN DAN SARAN

A. Kesimpulan

Sistem informasi adalah kombinasi dari berbagai komponen yaitu: manusia, fasilitas atau alat teknologi, media, prosedur dan pengendalian yang ditujukan untuk mengatur jaringan komunikasi yang penting, proses transaksi tertentu dan rutin, membantu manajemen dan pemakai internal maupun eksternal serta menyediakan dasar untuk pengambilan keputusan yang tepat.

Bahasa Pemrograman C++ merupakan Bahasa yang sudah lama dan populer di dalam pembelajaran pada computer dan Programmer yang ingin mengembangkan system software ataupun membuat game. Oleh sebab itu pemahaman akan bahasa pemrograman C++ ini akan sangat penting terlebih lagi ketika mahasiswa pada jurusan – jurusan yang berkaitan dengan teknologi ataupun programming. Pembelajaran Bahasa Pemrograman C++ secara teori maupun praktikum ini mempunyai peranan penting untuk meningkatkan pengetahuan mahasiswa tentang bahasa pemrograman, mahasiswa secara otomatis mempelajari tata cara penulisan Bahasa pemrograman dimana salah satu dari integer (atau bilangan bulat).

B. Saran

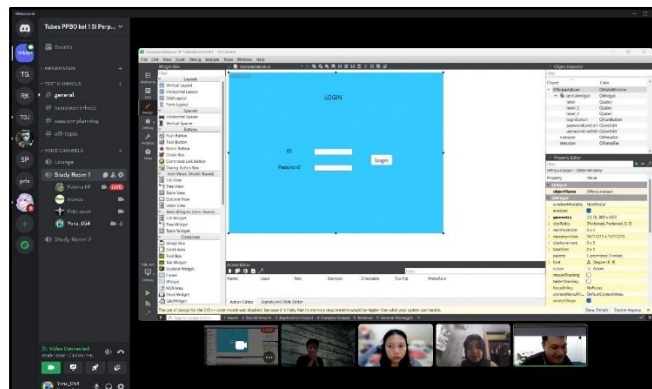
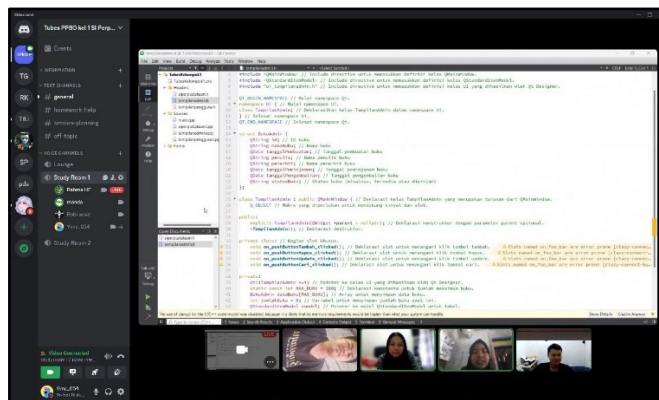
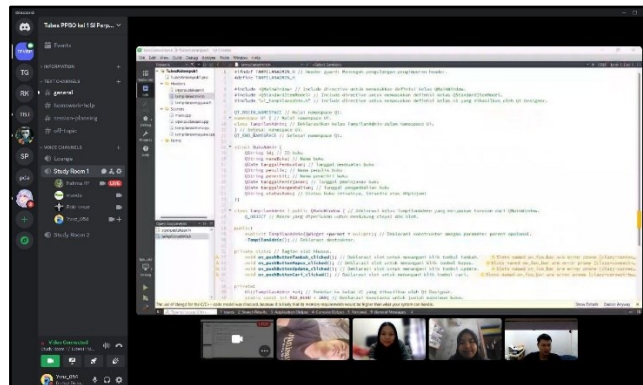
Untuk meningkatkan kinerja dan kehandalan aplikasi Anda, ada beberapa saran yang dapat Anda pertimbangkan. Pertama, pastikan untuk melakukan pengujian menyeluruh terhadap aplikasi setelah melakukan perbaikan. Pengujian ini harus mencakup skenario yang beragam, termasuk situasi di mana ID yang dicari berada di berbagai posisi dalam array. Dengan melakukan pengujian ini, Anda dapat memastikan bahwa aplikasi dapat menghasilkan hasil pencarian yang konsisten dan akurat.

Selanjutnya, lakukan pemeliharaan terus-menerus terhadap kode Anda. Hal ini meliputi pemantauan kinerja aplikasi dan pembaruan reguler untuk meningkatkan efisiensi dan kehandalan. Selain itu, pertimbangkan untuk mengoptimalkan algoritma pencarian biner jika diperlukan, terutama jika ukuran array `dataBuku` menjadi sangat besar.

DAFTAR PUSTAKA

- Rahmadhani, Andri, et al. "Prediksi pergerakan kurva harga saham dengan metode simple moving average menggunakan C++ dan Qt Creator." dalam Prosiding Seminar Kontribusi Fisika. 2011.
- Yansen, Yansen, and Agus Prijono. "Pengembangan Aplikasi Mobile dengan Qt Sdk: Studi Kasus Monitoring Ruangan Menggunakan Kamera." ComTech: Computer, Mathematics and Engineering Applications 5.1 (2014): 473-484.
- Garing, Lendri A., Stenly C. Takarendehang, and Abraham Kamal. "Sistem pendukung keputusan kelayakan akreditasi puskesmas dengan metode analytic hierarchy process (AHP) pada dinas kesehatan kabupaten sangihe." Jurnal Ilmiah Behongang 1.1 (2018): 16-21.

LAMPIRAN





**KEMENTERIAN PENDIDIKAN, KEBUDAYAAN
RISET DAN TEKNOLOGI
UNIVERSITAS BENGKULU
FAKULTAS TEKNIK
PROGRAM STUDI INFORMATIKA**

Jl. Wr. Supratman Kandang Limun, Bengkulu Bengkulu
38371 A Telp: (0736) 344087, 22105 - 227

LEMBAR ASISTENSI

PROYEK STRUKTUR DATA & ALGORITMA

Nama Kelompok	: 1. Yovanza Villareal (G1A023054) 2. Robi Septian Subhan (G1A023060) 3. Rahma Hidayati Fitrah (G1A023074) 4. Khalisa Rizgita Amanda (G1A023080)
Dosen	: 1. Arie Vatesia, S.T, M.TI, P.hD 2. Mochammad Yusa, S. Kom, M. Kom.
Asisten Dosen	:1. Davi Sulaiman (G1A022001) 2. Atiyya Dianti Fadli (G1A022002) 3. Abdi Agung Kurniawan (G1A022011) 4. Sophina Shafa Salsabila (G1A022021) 5. Evelyn Funike Aritonang (G1A022024) 6. Diodo Arrahman (G1A022027) 7. Sinta Ezra Wati Gulo (G1A022040) 8. Wahyu Ozorah Manurung (G1A022060) 9. Alif Nurhidayat (G1A022073) 10. Ahmad Radesta (G1A022086)

Laporan Praktikum

Catatan dan Tanda Tangan

Laporan Tugas Besar

