

Laporan Tugas Proyek Pemrograman Berorientasi Objek Sistem Pengelolaan Event Organizer



Disusun Oleh :

11323010	Frangklyn Aldo Ignatia Lumbantoruan
11323020	Marshanda Kasih Simangunsong
11323055	Febyanti Hutahaeen

**Institut Teknologi Del
Fakultas Vokasi
DIII- Teknologi Informasi**

**Laguboti
2024**

DAFTAR ISI

DAFTAR ISI	2
DAFTAR GAMBAR.....	3
DAFTAR TABEL	4
1 Pendahuluan	5
1.1 Latar belakang	5
1.2 Rumusan masalah.....	5
1.3 Tujuan	5
1.4 Manfaat	6
1.4.1 Manfaat bagi EO:.....	6
1.4.2 Manfaat bagi mahasiswa pengembang:	6
2 Desain dan Struktur Sistem	7
2.1 Deskripsi proyek	7
2.2 Fitur utama	7
2.3 ER-Diagram	7
2.4 Bisnis Proses Sistem.....	8
2.5 Struktur kelas.....	8
2.5.1 Struktur ClassLoginContoller	8
2.5.2 Struktur Class RegisterController	9
2.5.3 Struktur Class ClientContreller.....	11
2.5.4 Struktur Class EventController	12
2.5.5 Struktur Class ScheduleController.....	12
2.5.6 Struktur Class MainController.....	13
2.5.7 Struktur Class Client.....	14
2.5.8 Struktur Class Event	15
2.5.9 Struktur Class Schedule	16
3 Penerapan Konsep OOP.....	18
3.1 Penerapan OOP pada Class clientContreller	18
3.2 Penerapan OOP pada Class cventContreller	20
3.3 Penerapan OOP pada Class loginContreller	22
3.4 Penerapan OOP Pada cLass MainContreeler	23
3.5 Penerapan OOP pada class register controller.....	25
3.6 Penerapan oop pada classs ScheduleController.....	27
3.7 Penerapan oop pada class client	29
3.8 Penerapan oop pada class event	30
3.9 Penerapan oop pada class schedule	31
4 Hasil Implementasi	32
4.1 Hasil implementasi register	32
4.2 Hasil Implementasi Login	33
4.3 Hasil Implementasi dashboard	34
4.4 Hasil Implementasi client.....	35
4.4.1 Hasil implementasi tambah client.....	36
4.4.2 Hasil implementasi edit client.....	37
4.4.3 Hasil implementasi hapus client	38
4.5 Hasil Implementasi Event	39
4.6 Hasil Implementasi Schedule	40
5 Kesimpulan Dan Saran.....	42
5.1 Kesimpulan	42
6 Link YT Presentasi.....	43

DAFTAR GAMBAR

Gambar 1. Er-Diagram Sistem Pengelolaan Event Organizer	7
Gambar 2. Bisni Proses Sistem Pengelolaan Event Organizer	8
Gambar 3 Abstraction Pada Class Clientcontreller	18
Gambar 4 Encapsulation Pada Class Clientcontreller	19
Gambar 5 Inheritance Class Clientcontreller	19
Gambar 6 Polumorphism Class Clientcontreller	19
Gambar 7 Exception Handling Pada Class Clientcontreller	20
Gambar 8 Abstraction Pada Class Eventcontreller	20
Gambar 9 Encapsulation Pada Class Eventcontreller	21
Gambar 10 Inheritance Pada Class Ecentcontreller.....	21
Gambar 11 Polymorphism Pada Class Eventcontreller	22
Gambar 12 Encapsulation Pada Class Logincontroller	22
Gambar 13 Polymorphism Pada Class Logincontroller.....	23
Gambar 14 Abstraction Pada Class Logincontroller	23
Gambar 15 Encapsulation Pada Maincontrllor	24
Gambar 16 Abstraction Pada Cllass Maincontroller	24
Gambar 17 Exception Handling Pada Class Maincontroller	25
Gambar 18 Encapsulation Pada Class Registercontroller.....	26
Gambar 19 Abstraction Pada Class Registercontroller.....	26
Gambar 20 Exception Handling Pada Class Reistercontroller	26
Gambar 21 Encapsulation Pada Class Schedulecontoller.....	27
Gambar 22 Polymorphism Pada Class Schedulecontroller	27
Gambar 23 Abstraction Pada Class Schedulecontroller	28
Gambar 24 Interface Pada Class Schedulecontroller	28
Gambar 25 Encapsulation Pada Class Client.....	29
Gambar 26 Encapsulation Pada Class Event	30
Gambar 27 Encapsulation Pada Class Schedule.....	31
Gambar 28 Hasil Implementasi Register	32
Gambar 29 Hasil Implemetasi Register Jika Tidak Seusal Kriteria	32
Gambar 30 Hasil Implementasi Register Jika Suda Berhasil Buat Akun.....	33
Gambar 31 Hasil Implementasi Login	33
Gambar 32 Hasil Implementasi Login Ketika Berhasil Masuk	34
Gambar 33 Hasil Implementasi Dashboard Jika Semua Event Selesai	34
Gambar 34 Hasil Implementasi Dashboard Jika Sebagian Event Belum Selesai.....	35
Gambar 35 Hasial Implementasi Client	35
Gambar 36 Hasil Implementasi Tambah Client.....	36
Gambar 37 Hasil Implementasi Jika Client Berhasil Ditambah	36
Gambar 38 Hasil Implementasi Edit Client	37
Gambar 39 Hasil Implementasi Jika Client Berhasil Di Edit	37
Gambar 40 Hasil Implementasi Hapus Client	38
Gambar 41 Hasil Implementasi Hapus Client	38
Gambar 42 Implementasi Tambah Event	39
Gambar 43 Hasil Implementasi Tambah Event	39
Gambar 44 Implementasi Tambah Schedule	40
Gambar 45 Hasil Impementasi Tambah Schedule	40
Gambar 46 Implementasi Jika Tidak Ada Lagi Event Yang Ingin Ditambahkan	41

DAFTAR TABEL

tabel 1 method ClassLoginContoller	9
tabel 2 Method RegisterController.....	11
tabel 3 Method Class ClientContreller	11
tabel 4 method class EventController	12
tabel 5 method ScheduleController.....	13
tabel 6 method MainController	14
tabel 7 method Class Client	15
tabel 8 method Class Event.....	16
tabel 9 method Class Schedule	17

1 Pendahuluan

1.1 Latar belakang

Event Organizer (EO) memiliki peran penting dalam mengelola berbagai jenis acara, mulai dari perencanaan hingga pelaksanaan. Acara seperti pernikahan, seminar, konser, dan konferensi sering kali melibatkan banyak pihak yang memerlukan koordinasi yang baik agar acara berjalan lancar. Proses pengelolaan ini semakin kompleks seiring dengan bertambahnya jumlah klien dan acara yang harus ditangani secara bersamaan.

Berbagai aspek seperti pengelolaan data klien, informasi acara, hingga penjadwalan membutuhkan perhatian khusus untuk menghindari kesalahan yang dapat memengaruhi kualitas layanan EO. Oleh karena itu, sistem pengelolaan berbasis aplikasi menjadi solusi yang dapat Ru perhatian khusus untuk menghindari kesalahan yang dapat memengaruhi kualitas layanan EO. Oleh karena itu, sistem pengelolaan berbasis aplikasi menjadi solusi yang dapat membantu EO dalam meningkatkan efisiensi, akurasi, dan efektivitas operasional mereka.

1.2 Rumusan masalah

Rumusan masalah berikut dirancang untuk mengidentifikasi tantangan utama yang dihadapi dalam pengelolaan data dan operasional Event Organizer. Beberapa permasalahan yang perlu diselesaikan adalah sebagai berikut:

1. Bagaimana cara mempermudah pengelolaan data klien oleh Event Organizer?
2. Bagaimana memastikan informasi acara yang terorganisir dan mudah diakses?
3. Bagaimana membuat jadwal acara yang dapat dipantau statusnya (aktif/selesai)?

1.3 Tujuan

Dalam pengembangan sistem ini, terdapat beberapa tujuan utama yang ingin dicapai untuk mendukung operasional Event Organizer secara lebih efektif. Tujuan tersebut meliputi :

1. Mengembangkan aplikasi berbasis teknologi untuk membantu EO dalam mengelola data klien, informasi acara, dan jadwal secara terpusat.
2. Menyediakan fitur yang intuitif dan mudah digunakan untuk meningkatkan pengalaman pengguna.
3. Meningkatkan efisiensi dan akurasi dalam proses pengelolaan data EO.
4. Mempermudah pemantauan status acara (aktif/selesai) secara real-time.

1.4 Manfaat

Pengembangan sistem ini memberikan manfaat tidak hanya bagi Event Organizer (EO) sebagai pengguna utama, tetapi juga bagi mahasiswa sebagai pengembang sistem. Berikut adalah manfaat yang diharapkan:

1.4.1 Manfaat bagi EO:

1. Memudahkan EO dalam menyimpan dan mengakses informasi klien serta acara tanpa kendala.
2. Mengurangi potensi kesalahan dalam pengelolaan data yang dilakukan secara manual, sehingga meningkatkan keandalan sistem.
3. Meningkatkan produktivitas tim EO dengan menyediakan sistem yang terintegrasi dan dapat diakses kapan saja.
4. Meningkatkan kepuasan klien dengan layanan yang lebih cepat, terorganisir, dan profesional.
5. Memberikan alat bantu analitik untuk memahami kebutuhan klien dan tren acara, yang dapat membantu pengambilan keputusan strategis.

1.4.2 Manfaat bagi mahasiswa pengembang:

1. Memberikan pengalaman langsung dalam menerapkan konsep Pemrograman Berorientasi Objek (PBO) ke dalam pengembangan aplikasi nyata.
2. Meningkatkan pemahaman terhadap siklus pengembangan perangkat lunak, mulai dari analisis kebutuhan hingga implementasi.
3. Melatih keterampilan teknis dalam merancang dan mengintegrasikan sistem berbasis teknologi untuk menyelesaikan permasalahan dunia nyata.
4. Menumbuhkan kemampuan kolaborasi dan manajemen proyek, karena pengembangan melibatkan diskusi dan pembagian tugas.
5. Menjadi portofolio berharga sebagai bukti kemampuan mahasiswa dalam mengembangkan solusi inovatif yang aplikatif.

2 Desain dan Struktur Sistem

2.1 Deskripsi proyek

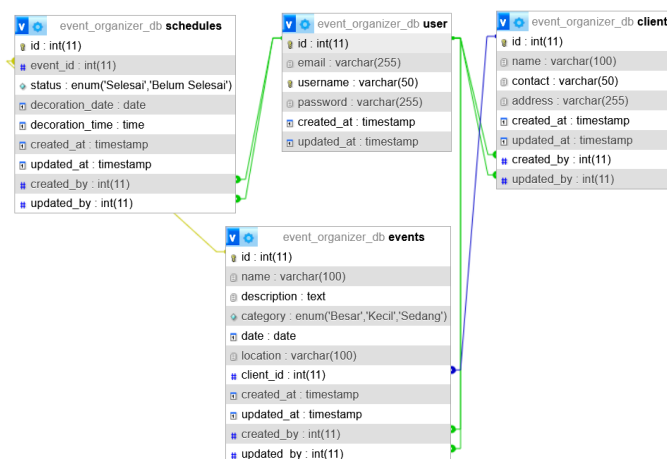
Sistem Pengelolaan Event Organizer (EO) adalah sistem yang dirancang untuk mempermudah pengelolaan data terkait klien, acara, dan jadwal acara secara terpusat. Sistem ini akan dilengkapi dengan antarmuka yang user-friendly dan fitur yang komprehensif untuk mendukung kebutuhan operasional EO.

2.2 Fitur utama

Berikut adalah fitur utama yang dikembangkan dalam sistem :

1. Register: Fitur yang memungkinkan pengguna baru untuk membuat akun dan mendaftar ke dalam sistem dengan memasukkan informasi yang diperlukan.
2. Login: Memungkinkan pengguna yang sudah terdaftar untuk masuk ke dalam sistem dengan menggunakan kredensial yang valid (username dan password) guna mengakses fitur-fitur yang tersedia.
3. Manajemen Klien: Memungkinkan EO untuk menambahkan, mengubah, dan menghapus data klien, serta menyimpan detail acara yang terkait dengan masing-masing klien.
4. Manajemen Acara: Mengelola informasi acara seperti jenis acara, tanggal pelaksanaan, dan lokasi secara efisien.
5. Pengelolaan Jadwal Acara: Memberikan kemampuan untuk memantau status acara (aktif atau selesai) dan mencari acara berdasarkan tanggal tertentu.

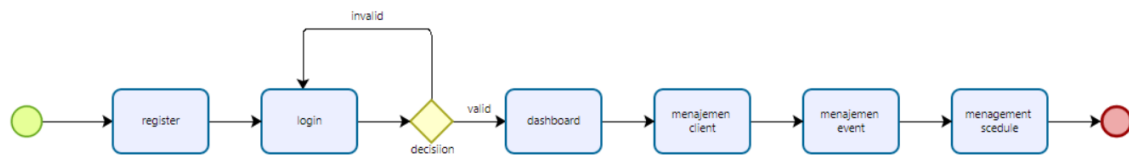
2.3 ER-Diagram



Gambar 1. ER-Diagram Sistem Pengelolaan Event Organizer

Pada ER-Diagram ini, terlihat hubungan antara beberapa entitas yang membentuk sistem pengelolaan Event Organizer. Entitas **user** menyimpan data pengguna yang mengelola sistem, dengan relasi ke entitas **clients**, **events**, dan **schedules** melalui atribut **created_by** dan **updated_by**, yang menunjukkan siapa yang membuat atau memperbarui data tersebut. Entitas **clients** menyimpan data klien, yang memiliki hubungan **one-to-many** dengan entitas **events**, dimana satu klien dapat memiliki banyak acara. Setiap acara memiliki relasi dengan entitas **schedules**, yang menyimpan jadwal terkait dengan acara tersebut, termasuk status dan waktu dekorasi. Dengan struktur ini, sistem dapat secara efektif mengelola data klien, acara, dan jadwal secara terpusat, serta memastikan interaksi antar entitas berjalan dengan baik.

2.4 Bisnis Proses Sistem



Gambar 2. Bisni Proses Sistem Pengelolaan Event Organizer

Gambar 2 menggambarkan alur bisnis proses sistem pengolahan event organizer yang dimulai dengan registrasi pengguna, diikuti dengan proses login untuk validasi akses. Jika login berhasil, pengguna diarahkan ke dashboard sebagai pusat pengelolaan, di mana mereka dapat mengelola data klien dan event secara terstruktur. Tahapan berikutnya melibatkan pengelolaan solusi manajemen untuk menyusun rencana dan eksekusi event secara menyeluruh. Proses ini memastikan semua aktivitas dari perencanaan hingga penyelesaian event dapat dilakukan dengan efisien dan terorganisir.

2.5 Struktur kelas

2.5.1 Struktur ClassLoginContoller

- Deskripsi

`LoginController` adalah **controller class** yang bertanggung jawab untuk mengatur logika proses login pada aplikasi berbasis JavaFX. Class ini bertindak sebagai penghubung antara tampilan (View) dan logika (Model) dalam pola arsitektur **MVC (Model-View-Controller)**. Class ini mengatur pengambilan data dari form login, memvalidasi input, memproses autentikasi dengan basis data, dan mengalihkan ke halaman yang sesuai (seperti halaman utama atau halaman registrasi).

- Fungsi:

1. **Menelola Proses Login:** Mengambil data dari input pengguna, memvalidasinya, dan mengecek ke basis data apakah username dan password cocok.
2. **Navigasi Halaman:** Berpindah dari halaman login ke halaman utama (MainView) atau halaman registrasi (Register).
3. **Memberikan Notifikasi:** Menampilkan pesan notifikasi sementara kepada pengguna, baik saat login berhasil, gagal, atau jika ada kesalahan input.

- Method

method	deskripsi	parameter
handleLogin()	Mengambil data dari form login, memvalidasi input, dan memeriksa ke basis data untuk proses autentikasi.	-
goToMainView()	Berpindah ke halaman utama <code>MainView.fxml</code> jika login berhasil.	-
goToRegister()	Berpindah ke halaman registrasi <code>Register.fxml</code> jika pengguna ingin mendaftar.	-
showTemporaryNotification(String message, String type, double durationInSeconds, Runnable onComplete)	Menampilkan notifikasi sementara dengan pesan tertentu, tipe notifikasi (sukses/gagal), dan durasi tampilan.	message, type, durationInSeconds, onComplete]

tabel 1 method ClassLoginContoller

2.5.2 Struktur Class RegisterController

- Deskripsi

Class `RegisterController` adalah bagian dari arsitektur **MVC** (Model-View-Controller). Class ini bertanggung jawab untuk menangani proses pendaftaran pengguna pada aplikasi Event Organizer. Proses ini mencakup pengambilan data input dari pengguna, validasi data input, pengecekan ketersediaan username, penyimpanan data ke database, serta navigasi ke halaman login setelah pendaftaran berhasil.

- Fungsi

Fungsi utama dari class RegisterController adalah sebagai berikut:

1. Mengelola Input Pendaftaran: Mengambil input dari TextField (username, email, dan password) dan memvalidasi input tersebut.
2. Validasi Data: Memastikan bahwa username unik, email valid (mengandung @gmail.com), dan password memenuhi kriteria minimal (6 karakter, mengandung huruf dan angka).
3. Menyimpan Data ke Database: Menyimpan informasi pengguna ke dalam tabel **user** di database jika semua validasi berhasil.
4. Navigasi Halaman: Mengarahkan pengguna ke halaman **Login** setelah pendaftaran berhasil.
5. Menampilkan Notifikasi dan Alert: Memberikan feedback kepada pengguna, baik dalam bentuk notifikasi sementara maupun alert dialog.

- Merhod

Nama Metode	parameter	deskripsi
handleRegister()	-	Memproses registrasi pengguna dengan memvalidasi input, memeriksa username, dan menyimpan ke database.
isUsernameExists(String username)	username: String	Mengecek apakah username sudah ada di database.
goToLogin()	-	Mengarahkan pengguna ke halaman Login.
showAlert()	type: AlertType, title: String, content: String	Menampilkan pesan kesalahan atau pemberitahuan ke pengguna.

showTemporaryNotification	message: String, durationInSeconds: double, onComplete: Runnable	Menampilkan notifikasi sementara dengan durasi tertentu dan tindakan setelah selesai.
---------------------------	---	---

tabel 2 Method RegisterController

2.5.3 Struktur Class ClientContreller

- Deskripsi:
Class ini berperan sebagai **pengendali (controller)** yang mengatur interaksi antara antarmuka pengguna (UI) dan logika aplikasi. Class ini bertanggung jawab menangani input dari pengguna dan mengatur data yang akan ditampilkan pada tabel.
- Fungsi:
 1. Menangani interaksi UI (klik tombol, pengisian form).
 2. Mengelola form popup untuk menambah data client baru.
 3. Menampilkan daftar client di dalam **TableView**.
- Method:

Nama Method	Parameter	
initialize()	-	Mengatur kolom, TableView , tombol, dan data awal.
btnAddClientTop.setOnAction()	Event action dari tombol	Membuka form tambah client saat tombol ditekan.
btnSaveClient.setOnAction()	Event action dari tombol	Menyimpan data client dari form ke ObservableList
btnCancelClient.setOnAction()	Event action dari tombol	Membatalkan form tambah client tanpa menyimpan data.

tabel 3 Method Class ClientContreller

2.5.4 Struktur Class EventController

- Deskripsi

Kelas ini adalah **controller** dalam aplikasi berbasis JavaFX. Kelas ini bertanggung jawab untuk mengelola interaksi antara tampilan (view) dan data (model) pada tabel event. Kelas ini mengatur data dari database MySQL ke tampilan tabel.
- Fungsi
 1. Mengelola data event dari database MySQL
 2. menampilkannya di UI (TableView). Selain itu, mengatur interaksi user seperti mengisi tabel secara otomatis dari database.
- Method

Nama Method	Parameter	Deskripsi
initialize()	-	Method ini dipanggil saat tampilan UI di-load. Mengatur kolom tabel agar sesuai dengan properti di kelas <code>Event</code> dan memuat data dari database.
loadDataFromDatabase()	-	Mengambil data event dari database MySQL menggunakan koneksi JDBC. Data event dimasukkan ke dalam <code>ObservableList</code> dan ditampilkan di tabel UI.
setCellValueFactory()	Properti dari <code>TableColumn</code>	Menautkan kolom-kolom pada UI dengan atribut dari kelas <code>Event</code> , sehingga data dari database dapat ditampilkan.

tabel 4 method class EventController

2.5.5 Struktur Class ScheduleController

- Deskripsi

Kelas ini adalah **controller** dalam aplikasi berbasis JavaFX. Kelas ini bertanggung jawab untuk mengatur interaksi antara tampilan (view) dan data.

- Fungsi
 1. Mengelola tampilan dan logika interaksi pengguna di halaman jadwal
 2. Mengatur tampilan popup dan tabel jadwal.

- Method

Nama Method	Parameter	DEskripsi
initialize()	-	Method ini dipanggil saat tampilan UI di-load. Digunakan untuk inisialisasi awal seperti pengaturan kolom tabel atau pengaturan UI lainnya.
showAddSchedulePopup()	-	Method ini membuat popup form untuk menambahkan jadwal menjadi terlihat (setVisible(true)).
hideAddSchedulePopup()	-	Method ini menyembunyikan popup form dengan mengatur visibilitas menjadi false (setVisible(false)).

tabel 5 method ScheduleController

2.5.6 Struktur Class MainController

- Deskripsi

Kelas ini bertanggung jawab untuk mengelola interaksi antara tampilan (view) dan logika aplikasi (business logic) terkait statistik dan data event.
- Fungsi
 1. Mengelola tampilan statistik jumlah pengguna, klien, dan event,
 2. menampilkan grafik status event dan data klien dalam tabel.
- Method

Nama Method	Parameter	Deskripsi
initialize()	-	Method yang dipanggil saat aplikasi dijalankan untuk menginisialisasi

		komponen UI dan memuat data dari database.
connectToDatabase()	-	Membuka koneksi ke database menggunakan utility yang sudah disiapkan.
loadTotalCounts()	-	Memuat data jumlah total pengguna, klien, dan event dari database dan menampilkannya di label yang sesuai.
loadEventStatusChart()	-	Memuat data status event dari database dan menampilkan data tersebut dalam bentuk grafik PieChart .
loadTableView()	-	Memuat daftar nama klien dari database dan menampilkannya dalam TableView .

tabel 6 method MainController

2.5.7 Struktur Class Client

- Deskripsi
Kelas ini bertanggung jawab untuk mengelola interaksi antara tampilan (view) dan logika aplikasi (business logic) terkait statistik dan data event.
- Fungsi
 1. Menyimpan data client (nama, kontak, dan alamat).
 2. Menyediakan **getter** dan **setter** agar data dapat diakses dan diperbarui.
 3. Mempermudah pengelolaan data client di **TableView** aplikasi.
- Method

Nama Method	Parameter	Deskripsi
-------------	-----------	-----------

Client(String name, String contact, String address)	name, contact, address	Constructor untuk membuat objek baru dari class Client .
getName()	-	Mengambil data nama client.
setName(String name)	name (String)	Mengatur data nama client.
getContact()	-	Mengambil data kontak client.
setContact(String contact)	contact (String)	Mengatur data kontak client.
getAddress()	-	Mengambil data alamat client.
setAddress(String address)	address (String)	Mengatur data alamat client.

tabel 7 method Class Client

2.5.8 Struktur Class Event

- Deskripsi
Kelas ini berfungsi untuk mengatur interaksi antara tampilan pengguna (UI) dan logika aplikasi terkait jadwal. Kelas ini menangani pengaturan tampilan tabel jadwal (TableView) dan popup untuk menambahkan jadwal baru.
- Fungsi
 1. Mengelola tampilan dan pengaturan elemen UI yang terkait dengan jadwal. Mengatur visibilitas popup yang digunakan untuk menambahkan jadwal.
 2. Mengelola data yang ditampilkan dalam TableView.
- Method

Nama Method	Parameter	Deskripsi
initialize()	-	Dipanggil setelah file FXML dimuat, digunakan untuk inisialisasi elemen UI.

showAddSchedulePopup()	-	Menampilkan popup untuk menambahkan jadwal baru.
hideAddSchedulePopup()	-	Menyembunyikan popup penambahan jadwal.
loadScheduleData(List<Schedule> scheduleList)	scheduleList (List<Schedule>)	Memuat data jadwal ke dalam TableView.

tabel 8 method Class Event

2.5.9 Struktur Class Schedule

- Deskripsi

Kelas `Schedule` digunakan untuk menyimpan jadwal yang terdiri dari jam dan kegiatan atau acara untuk setiap hari dalam sebulan. Setiap hari (dari hari 1 hingga hari 31) memiliki atribut sendiri yang berisi informasi terkait jadwal yang dilakukan pada hari tersebut. Kelas ini memiliki getter dan setter untuk setiap atribut, yang memungkinkan untuk mendapatkan dan mengatur jadwal berdasarkan jam dan hari tertentu.

- Fungsi:

- Menyimpan Jadwal:** Kelas ini berfungsi untuk menyimpan informasi jadwal berdasarkan jam dan hari dalam sebulan.
- Mengakses dan Memodifikasi Data Jadwal:** Dengan menggunakan getter dan setter, kelas ini memungkinkan kita untuk mengakses jadwal per hari atau mengubah jadwal yang telah ditentukan sebelumnya.

- Method

Nama Method	Parameter	Deskripsi
getHour()	-	Mengambil nilai jam.
setHour(int hour) ()	hour (int)	Menetapkan nilai jam
getDay1() ()	-	Mengambil nilai jadwal untuk hari 1.
setDay1(String day1)	day1 (String)	Menetapkan nilai untuk hari 1.

getDay2()		Mengambil nilai jadwal untuk hari 2
setDay2(String day2)	Day2 (String)	Menetapkan nilai untuk hari 2
getDay3()		Mengambil nilai jadwal untuk hari 3
setDay3(String day3)	Day3 (String)	Menetapkan nilai untuk hari 3
getDay31()		Mengambil nilai jadwal untuk hari 31
setDay31(String day31)	Day31 (String)	Menetapkan nilai untuk hari 31

tabel 9 method Class Schedule

3 Penerapan Konsep OOP

3.1 Penerapan OOP pada Class clientContreller

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu **Abstraction**, Encapsulation, Inheritance, Polymorphism, Interface, dan Exception Handling. Berikut penjelasannya:

- Abstraction

Abstraksi adalah konsep untuk menyembunyikan kompleksitas implementasi dan hanya menampilkan fungsionalitas yang diperlukan kepada pengguna. Pada kode ini, kelas `ClientController` menyembunyikan banyak detail implementasi terkait dengan database dan antarmuka pengguna. Misalnya, cara menyimpan, memperbarui, dan menghapus data client disembunyikan dalam metode-metode seperti `saveClient()`, `updateClient()`, dan `deleteClient()`

```
private void saveClient() {
    String nama = nameField.getText().trim();
    String kontak = contactField.getText().trim();
    String alamat = addressField.getText().trim();

    if (nama.isEmpty() || kontak.isEmpty() || alamat.isEmpty()) {
        showAlert("Validasi", "Harap isi semua field!");
        return;
    }

    try {
        String query = "INSERT INTO clients (name, contact, address) VALUES (?, ?, ?)";
        PreparedStatement statement = connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
        statement.setString(1, nama);
        statement.setString(2, kontak);
        statement.setString(3, alamat);

        int rowsInserted = statement.executeUpdate();
        if (rowsInserted > 0) {
            ResultSet generatedKeys = statement.getGeneratedKeys();
            if (generatedKeys.next()) {
                int id = generatedKeys.getInt(1);
                Client newClient = new Client(nama, kontak, alamat);
                newClient.setNumber(id);
                clientData.add(newClient);
            }
            clearFields();
            addClientPopup.setVisible(false);
        }
    } catch (SQLException e) {
        showAlert("Error Menyimpan", e.getMessage());
    }
}
```

Gambar 3 Abstraction pada Class clientContreller

- Encapsulation

Konsep encapsulation diterapkan dengan cara menyembunyikan data menggunakan modifier akses `private` pada atribut dan hanya memungkinkan akses melalui metode getter dan setter. contoh nya pada Atribut seperti `connection`, `clientData`, dan `selectedClient` dideklarasikan sebagai `private`, sehingga tidak dapat diakses secara langsung dari luar kelas. Akses hanya dapat dilakukan melalui metode atau proses internal kelas tersebut.

```
private ObservableList<Client> clientData = FXCollections.observableArrayList();
private Connection connection;
private Client selectedClient;
```

Gambar 4 encapsulation pada Class clientContreller

- Inheritance

ClientController mewarisi dari Initializable. Dengan mewarisi Initializable, kelas ini diwajibkan untuk mengimplementasikan metode `initialize()`. Metode ini dipanggil secara otomatis saat file FXML dimuat.

```
public class ClientController implements Initializable {
    @FXML
    // ...
}
```

Gambar 5 Inheritance Class clientContreller

- Polymorphism

Konsep polymorphism diterapkan dengan memungkinkan metode yang sama memiliki perilaku yang berbeda pada situasi tertentu. Listener pada TableView:

`clientTable.getSelectionModel().selectedItemProperty().addListener()` menggunakan Lambda Expression yang merupakan salah satu bentuk dari polimorfisme runtime. Saat baris pada tabel dipilih, objek Client yang dipilih dapat memicu perilaku berbeda tergantung data dari baris tersebut.

```
// Tambah event listener untuk seleksi baris
clientTable.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
    if (newSelection != null) {
        selectedClient = newSelection;
        populateClientFields(newSelection);
        btnSaveClient.setVisible(false);
        btnUpdateClient.setVisible(true);
        btnDeleteClient.setVisible(true);
        addClientPopup.setVisible(true);
    }
});
```

Gambar 6 polumorphism Class clientContreller

- Exception Handling

Pada metode `goToClient()`, proses pemuatan file FXML dapat gagal karena file mungkin hilang atau rusak. Untuk menghindari error yang tidak terduga, digunakan try-catch untuk menangkap dan menampilkan pesan kesalahan. Exception handling juga diterapkan pada operasi database, seperti insert, update, dan delete data klien, menggunakan try-catch untuk menangkap `SQLException`.

```
try {
    String query = "INSERT INTO clients (name, contact, address) VALUES (?, ?, ?)";
    PreparedStatement statement = connection.prepareStatement(query, Statement.RETURN_GENERATED_KEYS);
    statement.setString(1, nama);
    statement.setString(2, kontak);
    statement.setString(3, alamat);

    int rowsInserted = statement.executeUpdate();
    if (rowsInserted > 0) {
        ResultSet generatedKeys = statement.getGeneratedKeys();
        if (generatedKeys.next()) {
            int id = generatedKeys.getInt(1);
            Client newClient = new Client(nama, kontak, alamat);
            newClient.setNumber(id);
            clientData.add(newClient);
        }
        clearFields();
        addClientPopup.setVisible(false);
    }
} catch (SQLException e) {
    showAlert("Error Menyimpan", e.getMessage());
}
```

Gambar 7 Exception Handling pada Class clientContreller

3.2 Penerapan OOP pada Class cventContreller

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu **Abstraction**, Encapsulation, Inheritance, Polymorphism, . Berikut penjelasannya:

- Abstraction

Metode `showAlert()` adalah contoh abstraction. Metode ini menyembunyikan detail pembuatan dan pengaturan Alert dari pengguna. Pengguna hanya perlu memanggil `showAlert()` dengan judul dan pesan, tanpa perlu tahu bagaimana detail Alert tersebut dibuat.

```
private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}
```

Gambar 8 abstraction pada Class eventContreller

- Encapsulation

Encapsulation adalah pembungkusan data (atribut) dan metode dalam satu unit (kelas) serta membatasi aksesnya menggunakan akses modifier seperti `private`. Pada kode di atas, encapsulation diterapkan melalui atribut `connection` dan `eventData`, yang hanya dapat diakses melalui metode tertentu, bukan secara langsung.

```
public void initialize(URL location, ResourceBundle resources) {
    try {
        connection = DatabaseConnection.getDBConnection();
    } catch (SQLException e) {
        showAlert("Database Connection Error", e.getMessage());
        return;
    }
}
```

Gambar 9 Encapsulation pada class eventContreller

- Inheritance

Inheritance adalah konsep di mana sebuah kelas dapat mewarisi atribut dan metode dari kelas induknya. Dalam kode ini, `EventController` mewarisi dari `Initializable`, memungkinkan penggunaan metode `initialize()`.

```
public class EventController implements Initializable {

    @FXML private Button goToDashboard;
    @FXML private Button goToClient;
    @FXML private Button goToEvent;
    @FXML private Button goToSchedule;
    @FXML private Button btnAddEventTop;

    @FXML private TableView<Event> eventTable;
    @FXML private TableColumn<Event, Integer> numberColumn;
    @FXML private TableColumn<Event, String> nameColumn;
    @FXML private TableColumn<Event, String> descriptionColumn;
    @FXML private TableColumn<Event, String> categoryColumn;
    @FXML private TableColumn<Event, Date> dateColumn;
    @FXML private TableColumn<Event, String> locationColumn;
    @FXML private TableColumn<Event, String> clientNameColumn;

    @FXML private AnchorPane addEventPopup;
    @FXML private TextField nameField;
    @FXML private TextField descriptionField;
    @FXML private ComboBox<Category> categoryComboBox;
    @FXML private DatePicker eventDatePicker;
    @FXML private TextField locationField;
    @FXML private ComboBox<String> clientComboBox;

    @FXML private Button btnSaveEvent;
    @FXML private Button btnUpdateEvent;
    @FXML private Button btnCancelEvent;
    @FXML private Button btnDeleteEvent;

    private ObservableList<Event> eventData = FXCollections.observableArrayList();
    private Connection connection;
    private Event selectedEvent;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
```

Gambar 10 inheritance pada class eventcontroller

- Polymorphism

Polymorphism memungkinkan satu metode memiliki banyak bentuk. Dalam kode ini, polymorphism diterapkan melalui metode

`setCellValueFactory()`, yang dapat **mengambil** berbagai jenis data properti (seperti Integer, String, dan Date) di kolom tabel.

```
// Configure table columns
numberColumn.setCellValueFactory(cellData -> cellData.getValue().numberProperty().asObject());
nameColumn.setCellValueFactory(cellData -> cellData.getValue().nameProperty());
descriptionColumn.setCellValueFactory(cellData -> cellData.getValue().descriptionProperty());
categoryColumn.setCellValueFactory(cellData -> cellData.getValue().categoryProperty());
dateColumn.setCellValueFactory(cellData -> cellData.getValue().dateProperty());
locationColumn.setCellValueFactory(cellData -> cellData.getValue().locationProperty());
clientNameColumn.setCellValueFactory(cellData -> {
    String clientName = getClientNameById(cellData.getValue().getClientId());
    return new javafx.beans.property.SimpleStringProperty(clientName);
});
```

Gambar 11 polymorphism pada class eventContreller

3.3 Penerapan OOP pada Class loginContreller

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu Abstraction, Encapsulation, Polymorphism, Berikut penjelasannya:

- Encapsulation

Encapsulation adalah konsep yang menyembunyikan data internal dan hanya memperbolehkan akses melalui metode tertentu. Pada kode ini, penggunaan `usernameField` dan `passwordField` adalah contoh dari encapsulation, karena mereka bersifat `private` dan hanya bisa diakses atau dimodifikasi melalui metode yang disediakan.

```
private PasswordField passwordField;

public void handleLogin() {
    String username = usernameField.getText();
    String password = passwordField.getText();
}
```

Gambar 12 encapsulation pada class loginController

- Polymorphism

Polymorphism memungkinkan satu metode untuk bekerja dengan berbagai tipe objek. Pada kode ini, polymorphism tidak terlalu eksplisit, namun dapat dilihat dari metode `showTemporaryNotification()` yang menerima parameter `Runnable onComplete`, yang bisa berbeda-beda implementasinya, tergantung pada apa yang ingin dilakukan setelah notifikasi selesai.

```
if (rs.next()) {
    // Jika login berhasil, tampilkan notifikasi sukses dan pindah ke halaman utama
    showTemporaryNotification("Login Successful!", "success", 1.5, this::goToMainView);
} else {
```

Gambar 13 polymorphism pada class logincontroller

- Abstraction

Abstraction adalah konsep yang menyembunyikan implementasi internal dan hanya menyediakan interface yang diperlukan. Metode `showTemporaryNotification()` adalah contoh abstraction, di mana detail tentang bagaimana notifikasi ditampilkan disembunyikan dari pengguna, dan mereka hanya perlu memberikan parameter seperti pesan, tipe, durasi, dan aksi lanjutan.

```
// Metode untuk menampilkan notifikasi sementara dengan durasi tertentu
private void showTemporaryNotification(String message, String type, double durationInSeconds, Runnable onComplete) {
    Stage notificationStage = new Stage();
    notificationStage.initOwner(WindowUtils.getWindow().getOwner());
```

Gambar 14 abstraction pada class logincontroller

3.4 Penerapan OOP Pada cLass MainContreeler

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu Abstraction, Encapsulation, Polymorphism, resubality, Exception Handling , Berikut penjelasannya:

- Encapsulation

Metode `connectToDatabase()` dan `loadTotalCounts()` menyembunyikan detail implementasi terkait database dari kode lain. Hanya metode-metode ini yang mengakses data dan mengatur UI sesuai kebutuhan.

```

public class MainController {

@FXML
private Button goToDashboard;

@FXML
private Button goToClient;

@FXML
private Button goToEvent;

@FXML
private Button goToSchedule;

@FXML
private Label totalClientLabel;

@FXML
private Label totalEventLabel;

@FXML
private PieChart eventStatusChart;

private Connection connection;

// Initialize method called by FXML Loader
@FXML
public void initialize() {
    connection = connectToDatabase();
    loadTotalCounts();
    loadEventStatusChart();
}
}

```

Gambar 15 encapsulation pada maincontrllor

- Abstraction

loadTotalCounts() dan loadEventStatusChart() adalah contoh penggunaan abstraksi. Mereka mengabstraksi logika yang kompleks seperti query database dan pembaruan UI agar lebih mudah digunakan tanpa memikirkan detail implementasi.

```

// Load total counts for User, Clients, and Events
private void loadTotalCounts() {
    String totalClientsQuery = "SELECT COUNT(*) AS total FROM clients";
    String totalEventsQuery = "SELECT COUNT(*) AS total FROM EVENTS";

    try {
        PreparedStatement statement;
    }
}

```

Gambar 16 abstraction pada cclass maincontroller

- **Exception Handling:**

Dalam setiap metode yang melibatkan operasi database, error handling menggunakan try-catch untuk menangani potensi masalah dengan koneksi atau query, yang memperlihatkan bagaimana error dikelola dengan baik dalam OOP.

```
public void goToSchedule() {  
    try {  
  
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/Schedule.fxml"));  
        Parent clientView = loader.load();  
  
        // Get t-he current stage  
        Stage stage = (Stage) goToSchedule.getScene().getWindow();  
  
        // Create a new scene with the client view  
        Scene scene = new Scene(clientView);  
  
        // Set the new scene  
        stage.setScene(scene);  
        stage.show();  
    } catch (IOException e) {  
        System.out.println("Error loading Schedule view: " + e.getMessage());  
        e.printStackTrace();  
    }  
}
```

Gambar 17 Exception Handling pada class maincontroller

3.5 Penerapan OOP pada class register controller

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu Abstraction, Encapsulation, Polymorphism, resubality, Exception Handling , Berikut penjelasannya:

- **Encapsulation**

Enkapsulasi diterapkan di sini dengan cara menyembunyikan detail implementasi, seperti proses validasi dan interaksi dengan database, dan hanya menyediakan antarmuka (metode) untuk digunakan oleh pengguna kelas. Contohnya adalah penggunaan metode `handleRegister()`, yang menangani semua logika pendaftaran pengguna dan memvalidasi input tanpa mengekspose detail implementasi kepada pengguna kelas lainnya.

```
// Cek apakah username sudah ada di database
private boolean isUsernameExists(String username) {
    try (Connection connection = DatabaseConnection.getDBConnection()) {
        String sql = "SELECT COUNT(*) FROM user WHERE username = ?";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, username);

        ResultSet rs = stmt.executeQuery();
        if (rs.next() && rs.getInt(1) > 0) {
            return true;
        }
    } catch (Exception e) {
        showAlert(AlertType.ERROR, "Database Error", "Failed to check username: " + e.getMessage());
        e.printStackTrace();
    }
    return false;
}
```

Gambar 18 encapsulation pada class registercontroller

- Abstraction

Abstraksi diterapkan pada metode-metode seperti `handleRegister()` dan `goToLogin()`. Metode ini memberikan antarmuka yang jelas bagi pengguna kelas tanpa harus tahu bagaimana detail implementasinya.

```
public void handleRegister() {
    String username = usernameField.getText();
    String email = emailField.getText();
    String password = passwordField.getText();

    // TODO: Implement Register Logic
}
```

Gambar 19 abstraction pada class registercontroller

- Exception Handling

Penanganan pengecualian diterapkan di seluruh metode yang berinteraksi dengan database, seperti `isUsernameExists()` dan `handleRegister()`. Jika terjadi kesalahan (misalnya kegagalan koneksi ke database), pengecualian akan ditangani dengan blok `try-catch`, dan pesan kesalahan akan ditampilkan kepada pengguna.

```
// Cek apakah username sudah ada di database
private boolean isUsernameExists(String username) {
    try (Connection connection = DatabaseConnection.getDBConnection()) {
        String sql = "SELECT COUNT(*) FROM user WHERE username = ?";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, username);

        ResultSet rs = stmt.executeQuery();
        if (rs.next() && rs.getInt(1) > 0) {
            return true;
        }
    } catch (Exception e) {
        showAlert(AlertType.ERROR, "Database Error", "Failed to check username: " + e.getMessage());
        e.printStackTrace();
    }
    return false;
}
```

Gambar 20 Exception Handling pada class reistercontroller

3.6 Penerapan oop pada class ScheduleController

Pada class ini terdapat penerapan beberapa konsep OOP, yaitu Abstraction, Encapsulation, Polymorphism, interface, Berikut penjelasannya:

- Encapsulation

Encapsulation diterapkan dengan menyembunyikan detail implementasi elemen antarmuka pengguna seperti `Button`, `TableView`, dan `AnchorPane`. Elemen-elemen ini hanya diakses melalui metode di dalam kelas, menjadikan kode lebih terstruktur dan mudah dipelihara.

```
@FXML
private Button goToSchedule;

@FXML
private TableView<?> scheduleTable;
```

Gambar 21 encapsulation pada class scheduleContoller

- Polymorphism

Polymorphism diterapkan melalui metode navigasi yang berbeda untuk setiap tombol (`goToClient`, `goToDashboard`, dll.). Setiap metode memuat tampilan yang berbeda, namun mereka memiliki nama yang serupa, menunjukkan kesamaan dalam fungsionalitas meskipun implementasinya berbeda.

```
public void goToDashboard() {
    try {

        FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/MainView.fxml"));
        Parent clientView = loader.load();

        // Get the current stage
        Stage stage = (Stage) goToDashboard.getScene().getWindow();

        // Create a new scene with the client view
        Scene scene = new Scene(clientView);

        // Set the new scene
        stage.setScene(scene);
        stage.show();
    } catch (IOException e) {
        System.out.println("Error loading Dashboard view: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Gambar 22 polymorphism pada class schedulecontroller

- Abstraction

Dengan menggunakan FXML dan Scene, rincian implementasi tampilan disembunyikan dari kode kontroler. Ini memungkinkan fokus pada logika aplikasi dan membuat tampilan dapat diganti tanpa mengubah logika navigasi.

```
FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/Schedule.fxml"));
Parent clientView = loader.load();

// Get the current stage
Stage stage = (Stage) goToSchedule.getScene().getWindow();

// Create a new scene with the client view
Scene scene = new Scene(clientView);
```

Gambar 23 Abstraction pada class schedulecontroller

- Interface

elas ScheduleController bisa mengimplementasikan interface ini dan menyediakan implementasi untuk metode go to client. Ini memberikan fleksibilitas untuk menggunakan berbagai implementasi navigasi di berbagai controller.

```
public class ScheduleController {

    @FXML
    private Button goToDashboard;

    @FXML
    private Button goToClient;

    @FXML
    private Button goToEvent;

    @FXML
    private Button goToSchedule;

    @FXML
    private TableView<?> scheduleTable;

    @FXML
    private AnchorPane addSchedulePopup;

    @FXML
    private void initialize() {
        // Initialize TableView, and any other setup
    }

    @FXML
    private void showAddSchedulePopup() {
        addSchedulePopup.setVisible(true);
    }

    @FXML
    private void hideAddSchedulePopup() {
        addSchedulePopup.setVisible(false);
    }

    public void goToClient() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/Client.fxml"));
            Parent clientView = loader.load();

            // Get the current stage
            Stage stage = (Stage) goToClient.getScene().getWindow();

            // Create a new scene with the client view
            Scene scene = new Scene(clientView);
```

Gambar 24 Interface pada class scheduleController

3.7 Penerapan oop pada class client

Pada class ini terdapat penerapan konsep OOP, yaitu Encapsulation, Berikut penjelasannya:

- Encapsulation

Pada penerapan enkapsulasi, atribut dalam kelas `Client` disembunyikan dengan menggunakan modifier `private`, dan akses terhadap data hanya dapat dilakukan melalui metode **getter** dan **setter**. Dengan demikian, data hanya dapat dimodifikasi atau dibaca melalui cara yang telah ditentukan, menjaga integritas dan keamanan data.

```
public class Client {  
    private IntegerProperty number;  
    private IntegerProperty id;  
    private StringProperty name;  
    private StringProperty contact;  
    private StringProperty address;  
  
    public Client(String name, String contact, String address) {  
        this.number = new SimpleIntegerProperty();  
        this.id = new SimpleIntegerProperty();  
        this.name = new SimpleStringProperty(name);  
        this.contact = new SimpleStringProperty(contact);  
        this.address = new SimpleStringProperty(address);  
    }  
  
    // Getter untuk Property  
    public IntegerProperty numberProperty() {  
        return number;  
    }  
  
    public IntegerProperty idProperty() {  
        return id;  
    }  
  
    public StringProperty nameProperty() {  
        return name;  
    }  
  
    public StringProperty contactProperty() {  
        return contact;  
    }  
  
    public StringProperty addressProperty() {  
        return address;  
    }  
  
    // Getter dan Setter biasa  
    public int getNumber() {  
        return number.get();  
    }  
  
    public void setNumber(int number) {  
        this.number.set(number);  
    }  
  
    public int getId() {  
        return id.get();  
    }  
  
    public void setId(int id) {  
        this.id.set(id);  
    }  
  
    public String getName() {  
        return name.get();  
    }  
  
    public void setName(String name) {  
        this.name.set(name);  
    }  
}
```

Gambar 25 Encapsulation pada class client

3.8 Penerapan oop pada class event

Pada class ini terdapat penerapan konsep OOP, yaitu Encapsulation, Berikut penjelasannya

- Encapsulation

Enkapsulasi adalah prinsip OOP yang menyembunyikan atribut atau data dalam kelas dan memberikan akses melalui metode khusus (getter dan setter). Dalam kelas Event, atribut seperti number, id, name, description, category, date, location, clientId, createdBy, dan updatedBy disembunyikan dengan menggunakan private access modifier. Akses ke data ini hanya dapat dilakukan melalui getter dan setter, yang memungkinkan pengontrolan dan perlindungan data.

```
package model;

import javax.xml.bind.annotation.*;
import java.sql.Date;

public class Event {
    private IntegerProperty number;
    private IntegerProperty id;
    private StringProperty name;
    private StringProperty description;
    private StringProperty category;
    private ObjectProperty<Date> date;
    private StringProperty location;
    private IntegerProperty clientId;
    private IntegerProperty createdBy;
    private IntegerProperty updatedBy;

    public Event() {
        this.number = new SimpleIntegerProperty();
        this.id = new SimpleIntegerProperty();
        this.name = new SimpleStringProperty("");
        this.description = new SimpleStringProperty("");
        this.category = new SimpleStringProperty("");
        this.date = new SimpleObjectProperty<>();
        this.location = new SimpleStringProperty("");
        this.clientId = new SimpleIntegerProperty();
        this.createdBy = new SimpleIntegerProperty();
        this.updatedBy = new SimpleIntegerProperty();
    }

    // Constructor with parameters
    public Event(String name, String description, String category, Date date,
        String location, int clientId) {
        this.number = new SimpleIntegerProperty();
        this.id = new SimpleIntegerProperty();
        this.name = new SimpleStringProperty(name);
        this.description = new SimpleStringProperty(description);
        this.category = new SimpleStringProperty(category);
        this.date = new SimpleObjectProperty<>(date);
        this.location = new SimpleStringProperty(location);
        this.clientId = new SimpleIntegerProperty(clientId);
        this.createdBy = new SimpleIntegerProperty();
        this.updatedBy = new SimpleIntegerProperty();
    }

    // Property getters
    public IntegerProperty numberProperty() {
        return number;
    }

    public IntegerProperty idProperty() {
        return id;
    }

    public StringProperty nameProperty() {
        return name;
    }
}
```

Gambar 26 encapsulation pada class event

3.9 Penerapan oop pada class schedule

Pada class ini terdapat penerapan konsep OOP, yaitu Encapsulation, Berikut penjelasannya

- Encapsulation

Encapsulation adalah prinsip OOP yang menyarankan untuk menyembunyikan data internal (atribut) dan hanya memberikan akses ke data tersebut melalui metode tertentu (getter dan setter). Hal ini bertujuan agar data tersebut tidak bisa diakses sembarangan, memberikan kontrol lebih terhadap perubahan nilai atribut.

Pada kelas Schedule, atribut seperti day1, day2, ..., day31 dan hour di-private-kan (tidak dapat diakses langsung dari luar kelas) dan hanya bisa diubah atau diambil nilainya melalui metode getter dan setter yang disediakan:

```
package model;

public class Schedule {

    private int hour; // Represents the hour of the day (0 to 23)
    private String day1;
    private String day2;
    private String day3;
    private String day4;
    private String day5;
    private String day6;
    private String day7;
    private String day8;
    private String day9;
    private String day10;
    private String day11;
    private String day12;
    private String day13;
    private String day14;
    private String day15;
    private String day16;
    private String day17;
    private String day18;
    private String day19;
    private String day20;
    private String day21;
    private String day22;
    private String day23;
    private String day24;
    private String day25;
    private String day26;
    private String day27;
    private String day28;
    private String day29;
    private String day30;
    private String day31;

    // Getter and Setter for Hour
    public int getHour() {
        return hour;
    }

    public void setHour(int hour) {
        this.hour = hour;
    }

    // Getter and Setter for each day
    public String getDay1() { return day1; }
    public void setDay1(String day1) { this.day1 = day1; }

    public String getDay2() { return day2; }
    public void setDay2(String day2) { this.day2 = day2; }

    public String getDay3() { return day3; }
    public void setDay3(String day3) { this.day3 = day3; }

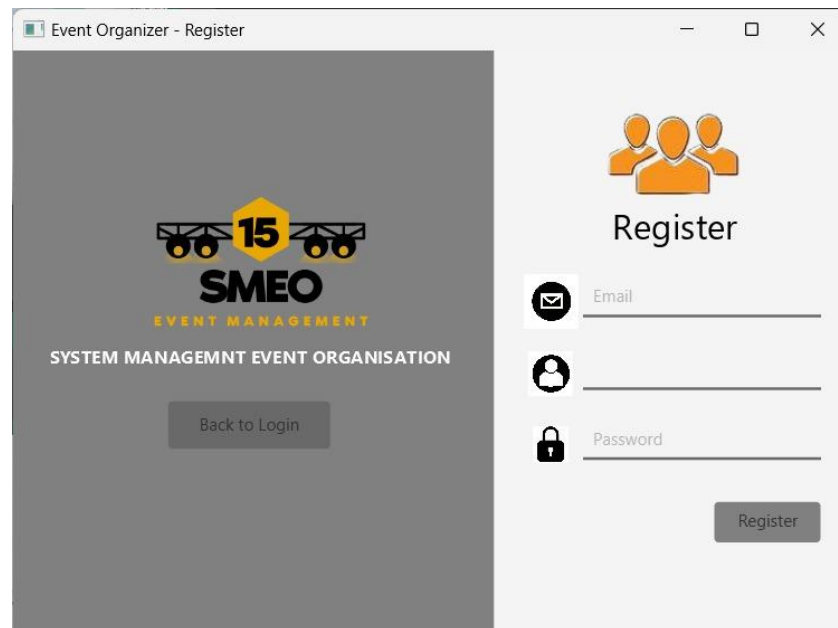
    public String getDay4() { return day4; }
    public void setDay4(String day4) { this.day4 = day4; }
```

Gambar 27 encapsulation pada class schedule

4 Hasil Implementasi

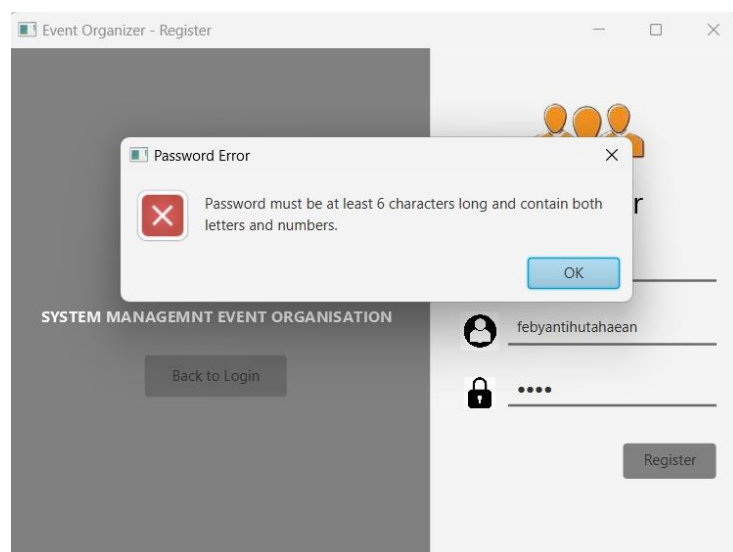
4.1 Hasil implementasi register

Pada halaman ini, pengguna harus mengisi form Email, username dan password, setelah itu klik button register. Akan tampil raise notice berisikan akun berhasil didaftarkan.



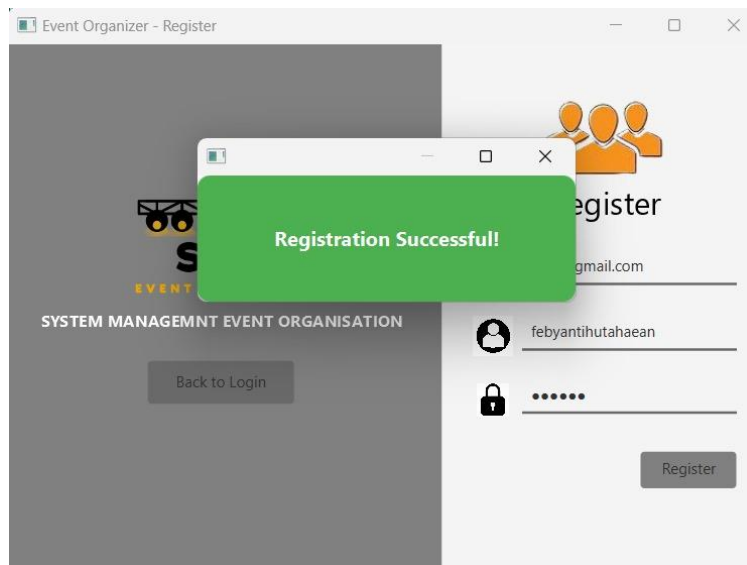
Gambar 28 hasil implementasi register

Jika pengguna memasukkan data yang tidak memenuhi kriteria, aplikasi akan menampilkan notifikasi seperti pada gambar berikut.



Gambar 29 hasil implemetasi register jika tidak seusal kriteria

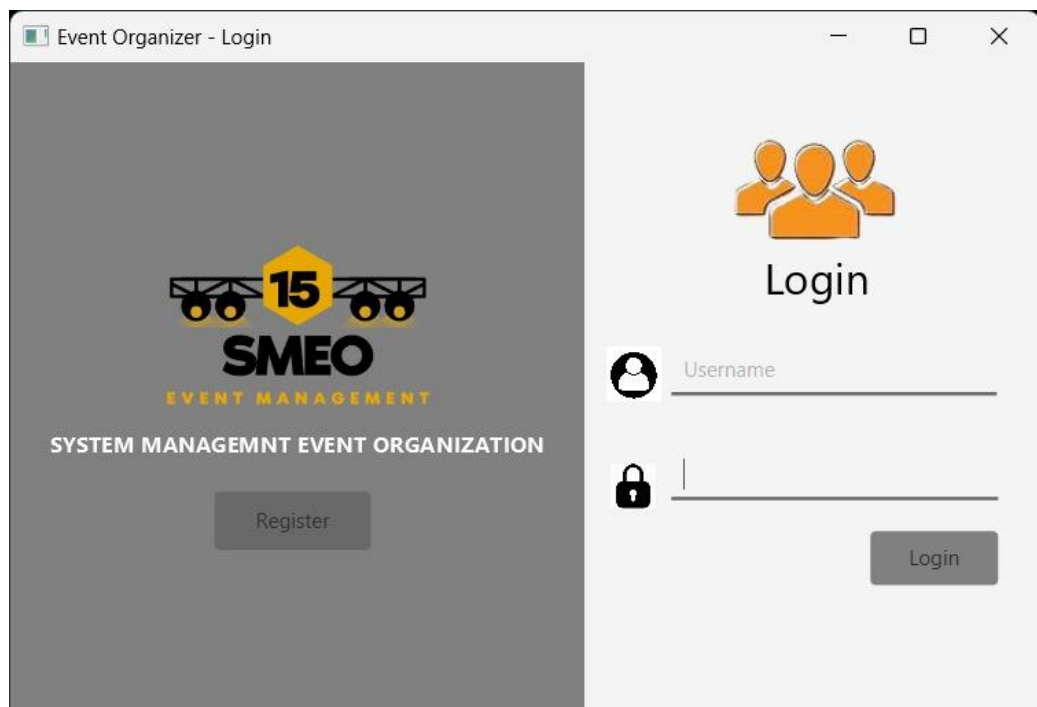
Namun, jika data yang dimasukkan benar, aplikasi akan menampilkan notifikasi seperti pada gambar berikut



Gambar 30 hasil implementasi register jika suda berhasil buat akun

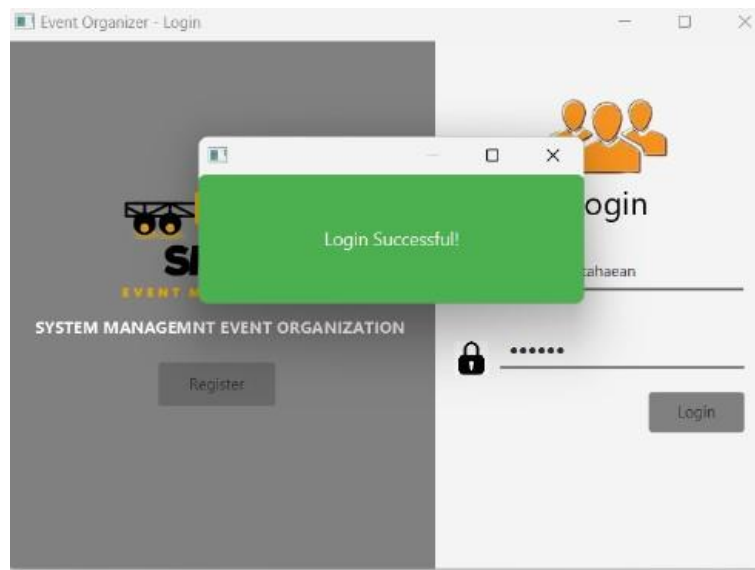
4.2 Hasil Implementasi Login

Pada halaman ini, pengguna harus mengisi form username password, setelah itu klik button register. Akan tampil raise notice berisikan berhasil login.



Gambar 31 hasil implementasi login

Jika anda sudah memasukkan username dan password dengan benar , dia akan menampilkan notifikasi seperti dibawah.

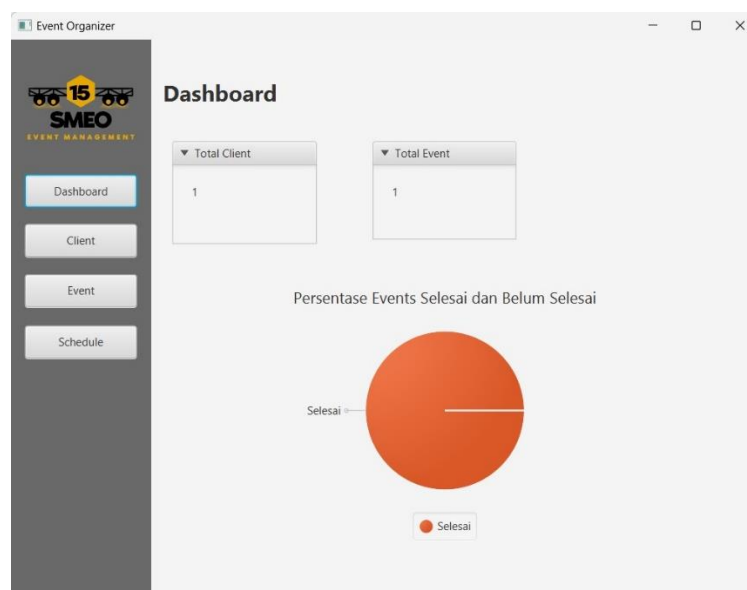


Gambar 32 hasil implementasi login ketika berhasil masuk

4.3 Hasil Implementasi dashboard

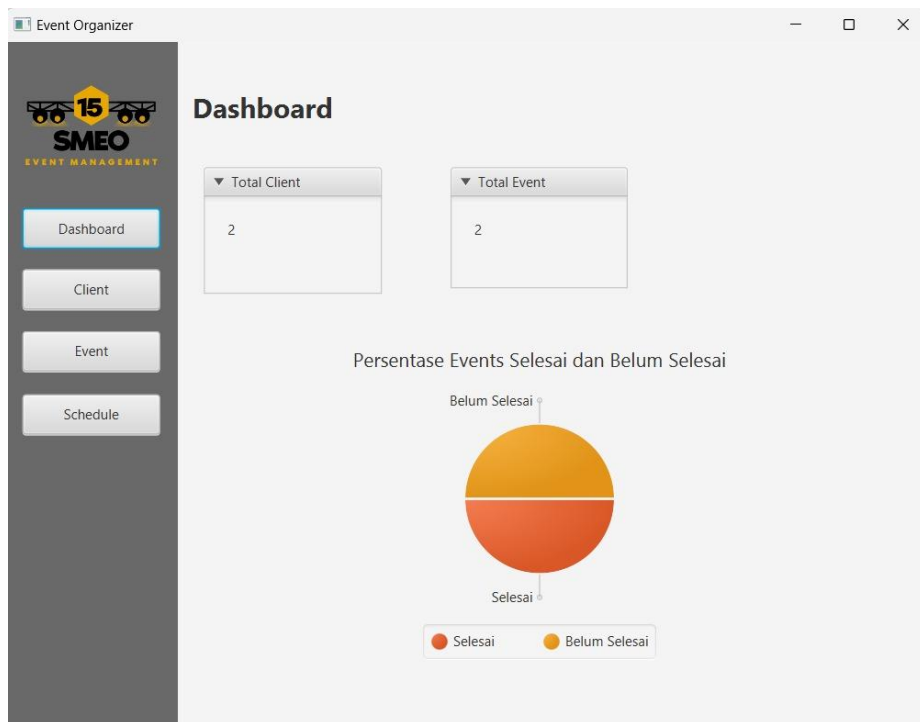
Pada halaman ini, pengguna dapat melihat jumlah client atau event , dan juga melihat presentasi event yang sudah selesai dan belum selesai. berikut ada lah hasil implementasi nya,

Berikut contoh dashboard ketika semua events selesai



Gambar 33 hasil implementasi dashboard jika semua event selesai

Berikut contoh dahshboard ketika ada event yang belum selesai



Gambar 34 hasil implementasi dashboard jika sebagian event belum selesai

4.4 Hasil Implementasi client

Pada halaman ini, pengguna dapat menambah client, mengedit client, dan menghapus client. Hasil implementasi dapat dilihat pada gambar dibawah

[illegible]

Gambar 35 hasial implementasi client

4.4.1 Hasil implementasi tambah client

"Klik tombol 'Tambah Client' di atas untuk menampilkan formulir seperti di bawah ini, yang kemudian dapat diisi sesuai kebutuhan."

Tambah Client

Gambar 36 hasil implementasi tambah client

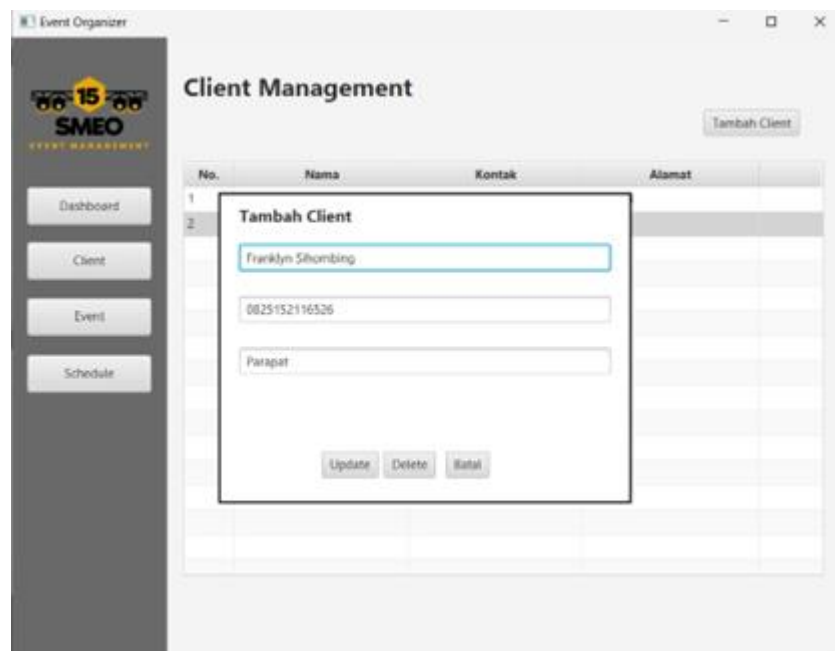
Data yang dimasukkan akan secara otomatis ditambahkan dan ditampilkan di dalam tabel

[illegible]

Gambar 37 hasil implementasi jika client berhasil ditambah

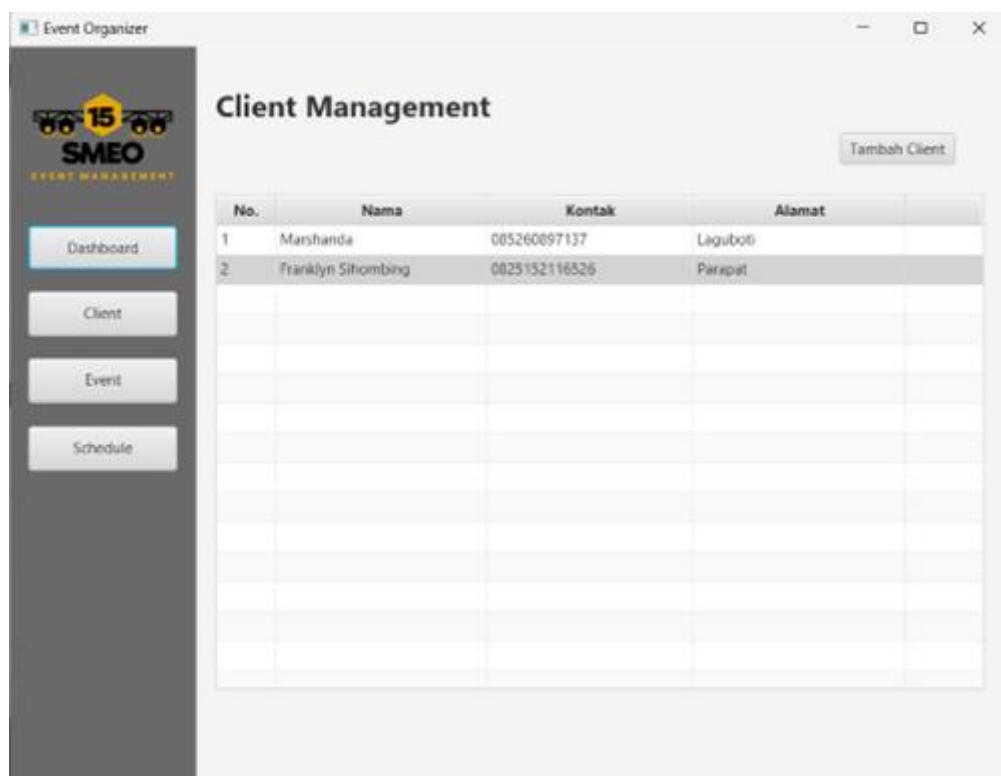
4.4.2 Hasil implementasi edit client

Untuk mengedit data client, klik pada client yang ingin diubah, lalu tekan tombol 'Update' untuk memperbarui informasi sesuai kebutuhan.



Gambar 38 hasil implementasi edit client

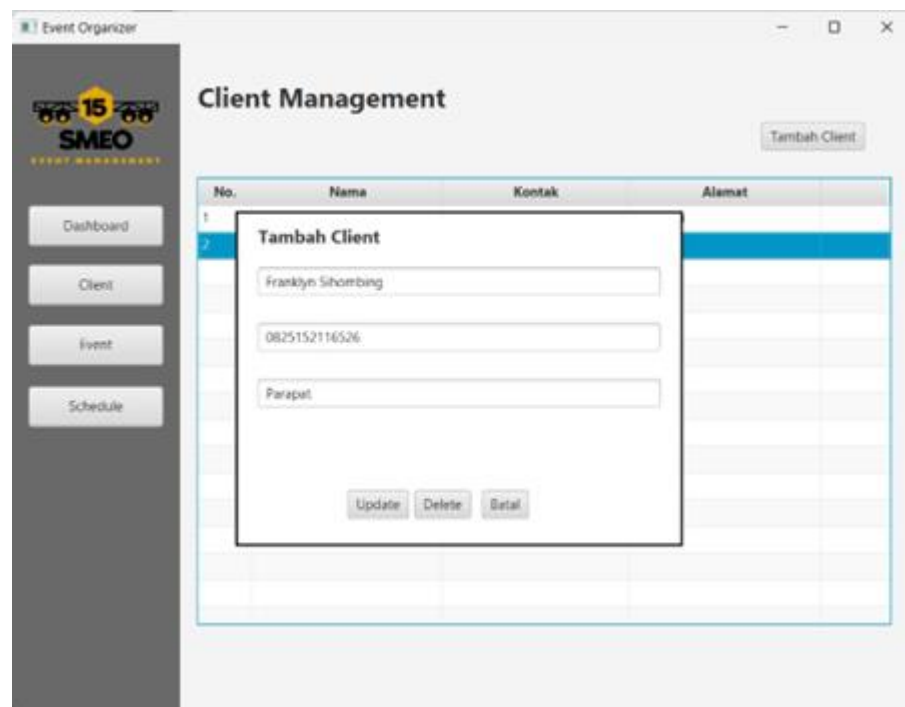
Berikut adalah hasil dari perubahan yang telah dilakukan



Gambar 39 hasil implementasi jika client berhasil di edit

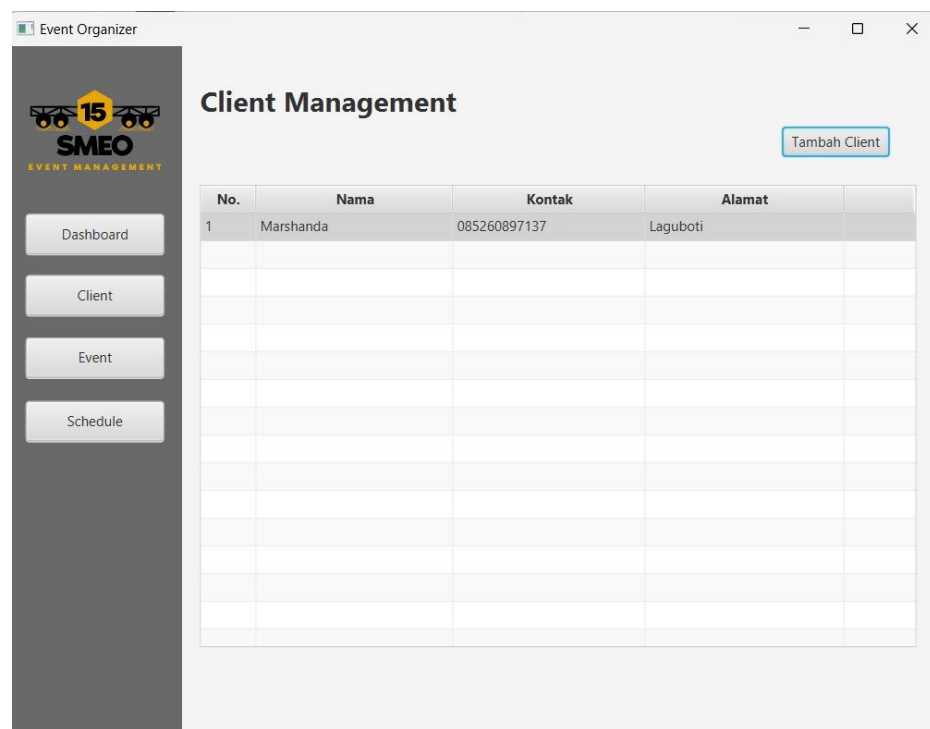
4.4.3 Hasil implementasi hapus client

Untuk menghapus data, klik pada data yang ingin dihapus, lalu tekan tombol 'Delete'



Gambar 40 hasil implementasi hapus client

Data tersebut akan terhapus secara otomatis



Gambar 41 hasil implementasi hapus client

4.5 Hasil Implementasi Event

kita dapat menambahkan acara dengan mengisi formulir di bawah ini. Pada bagian 'Client', hanya klien yang telah ditambahkan sebelumnya melalui menu 'Tambah Client' yang dapat dipilih setelah itu ktekan simpan

Event Form

Natal Sekolah Minggu

Tema Ungu

Sedang

12/26/2024

HKBP Sitoluama

Febyanti Hutahaeen

Simpan Batal

Tanggal	Lokasi	Nama C
2024-12-20	Gedung Haran...	Marshanda

Gambar 42 implementasi tambah event

Maka event akan otomatis muncul pada tabel seperti pada gambar dibawah

No.	Nama Event	Deskripsi	Kategori	Tanggal	Lokasi	Nama C
1	Ulang Tahun	Temanya itu M...	Kecil	2024-12-20	Gedung Haran...	Marshanda
2	Natal Sekolah ...	Tema Ungu	Sedang	2024-12-26	HKBP Sitoluama	Febyanti Hut

Gambar 43 hasil implementasi tambah event

4.6 Hasil Implementasi Schedule

Kami dapat menambahkan jadwal dengan mengisi formulir di bawah ini. Berikut adalah hasil implementasinya.

Tambah Schedule

Natal Sekolah Minggu

▼


Status

Belum Selesai

▼

Tanggal Dekorasi

12/25/2024



Jam Dekorasi

18:00

Simpan

Batal

Gambar 44 implementasi tambah schedule

Maka akan secara otomatis akan masuk pada tabel schedule

[illegible]

Gambar 45 hasil impementasi tambah schedule

Setiap acara hanya dapat dijadwalkan sekali. Karena setiap acara sudah pernah digunakan, maka acara tersebut tidak dapat dijadwalkan kembali. seperti dibawah

The screenshot displays the 'Event Organizer' application window. On the left is a dark sidebar with the 'SMEO EVENT MANAGEMENT' logo and four buttons: 'Dashboard', 'Client', 'Event', and 'Schedule'. The main area is titled 'Schedule' and contains a table with columns 'Event Name' and 'Time'. The table lists 'Ulang Tahun' and 'Natal Sekolah Ming'. A 'Tambah Schedule' button is in the top right. A modal form titled 'Tambah Schedule' is open, featuring a 'Nama Event' dropdown menu, a 'Status' dropdown menu, a 'Tanggal Dekorasi' date picker, and a 'Jam Dekorasi' time picker with a 'Format: HH:mm' hint. At the bottom of the modal are 'Simpan' and 'Batal' buttons.

Gambar 46 implementasi jika tidak ada lagi event yang ingin ditambahkan

5 Kesimpulan Dan Saran

5.1 Kesimpulan

Proyek "Sistem Pengelolaan Event Organizer" berhasil dikembangkan dengan menggunakan pendekatan Pemrograman Berorientasi Objek (PBO). Sistem ini dirancang untuk mempermudah pengelolaan data klien, acara, dan jadwal secara terpusat. Dengan fitur-fitur utama seperti registrasi, login, manajemen klien, manajemen acara, dan pengelolaan jadwal, sistem ini mampu meningkatkan efisiensi operasional dan akurasi dalam pengelolaan data EO. Selain itu, sistem ini menyediakan antarmuka pengguna yang intuitif dan fungsional, sehingga mempermudah pengguna dalam mengelola data dan memantau status acara secara real-time. Dari sisi pengembangan, proyek ini memberikan pengalaman langsung kepada mahasiswa dalam menerapkan konsep PBO, merancang struktur kelas, serta mengintegrasikan komponen MVC.

5.2 Saran

1. **Pengembangan Fitur Tambahan:** Untuk meningkatkan fungsionalitas, sistem dapat dikembangkan lebih lanjut dengan menambahkan fitur notifikasi otomatis, seperti pengingat jadwal acara kepada klien atau tim EO melalui email atau pesan singkat.
2. **Integrasi Sistem:** Disarankan untuk mengintegrasikan sistem dengan platform pembayaran online untuk mendukung transaksi yang terkait dengan layanan EO.
3. **Pengujian yang Lebih Mendalam:** Lakukan pengujian secara komprehensif terhadap sistem, termasuk uji beban dan uji keamanan, untuk memastikan performa sistem tetap optimal dalam skenario penggunaan yang kompleks.
4. **Dokumentasi Pengguna:** Tambahkan panduan pengguna (user manual) yang lebih rinci untuk membantu pengguna baru memahami cara kerja sistem secara keseluruhan.
5. **Evaluasi dan Perbaikan Berkelanjutan:** Adakan evaluasi berkala dengan pengguna EO untuk mengidentifikasi area perbaikan dan meningkatkan kepuasan pengguna terhadap sistem.
6. **Penyempurnaan Antarmuka:** Meningkatkan desain antarmuka pengguna agar lebih modern dan responsif untuk pengalaman pengguna yang lebih baik, termasuk pada perangkat seluler.

6 Link YT Presentasi

[LINK YT KELOMPOK 15 Sistem Pengelolaan Even Organizer](#)