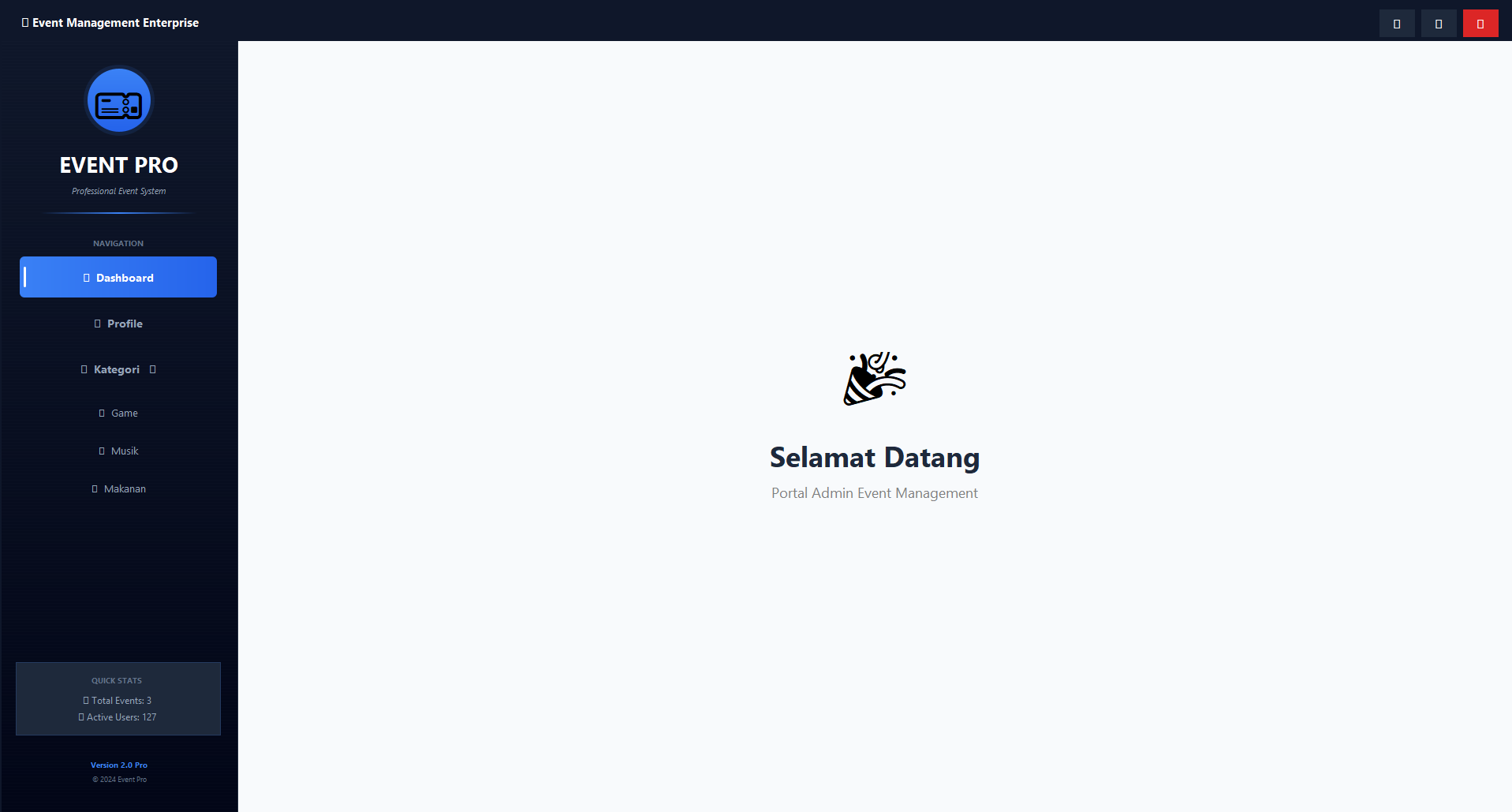
**PORTOFOLIO PROYEK PEMROGRAMAN BERORIENTASI OBJEK (TP-PBO2025)**

**a. Identitas Proyek**

* **NIM**: 2400018013
* **Nama**: Zulhan Hafiz Fadhilahlutfi
* **Judul Projek**: Sistem Manajemen Reservasi Event Berbasis Java GUI
* **Link Projek**: <https://github.com/zulhanhafiz/pbo-Sistem-Reservasi-Tiket/tree/main>
* **Capture Tampilan Utama**:



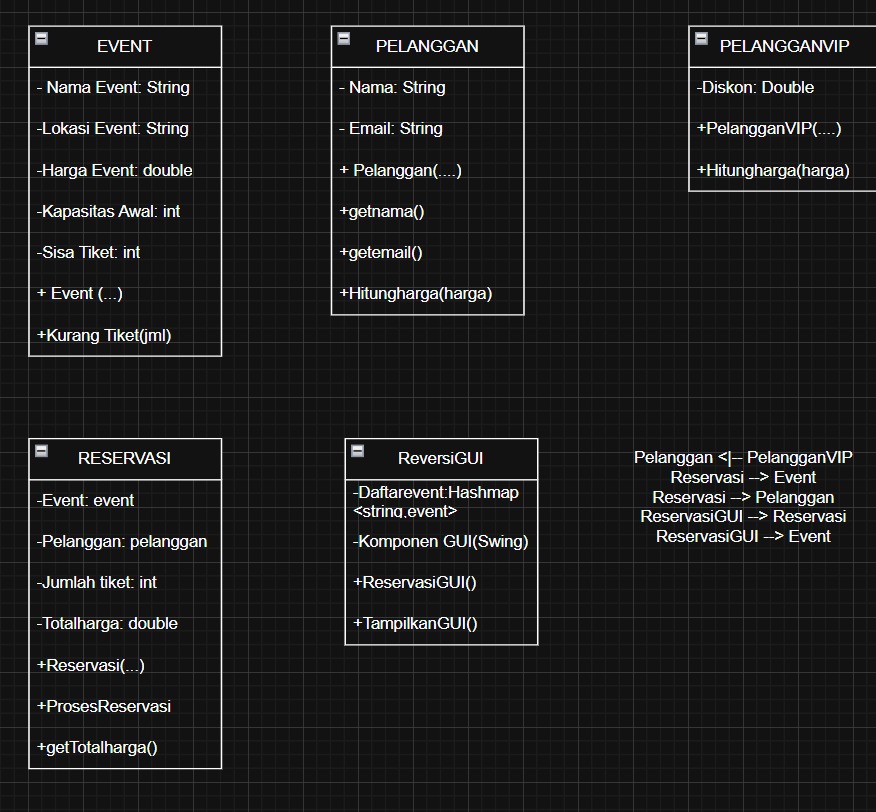
**b. Persoalan Bisnis dan Deskripsi Proyek**

* **Persoalan Bisnis**: Pengelolaan reservasi acara seringkali sulit dilakukan secara manual, terutama dalam membedakan fasilitas antara pelanggan biasa dan pelanggan VIP serta mengelola data event secara terorganisir.
* **Deskripsi Proyek**: Aplikasi ini adalah sistem manajemen reservasi yang memungkinkan admin/pengguna untuk mendata event, mengelola data pelanggan, dan memproses reservasi secara digital menggunakan antarmuka grafis (GUI).

**c. Daftar Seluruh Spesifikasi Aplikasi**

1. **Manajemen Event**: Menyimpan informasi detail mengenai acara.
2. **Manajemen Pelanggan**: Mengelola data identitas pelanggan.
3. **Sistem Membership**: Membedakan antara Pelanggan Reguler dan Pelanggan VIP (Inheritance).
4. **Proses Reservasi**: Menghubungkan data pelanggan dengan event yang dipilih.
5. **Interface Berbasis GUI**: Interaksi pengguna menggunakan form input dan tombol aksi (Swing/AWT).

**d. Rancangan Model Diagram UML**

****

* **Class Parent**: Pelanggan.
* **Class Child**: PelangganVIP (extends Pelanggan).
* **Class Model**: Event, Reservasi.
* **Class Boundary**: ReservasiGUI.

**e. Rancangan Antar Muka berbasis GUI**

Untuk bagian tampilan, saya membuat aplikasi **Event Management Enterprise** ini dengan konsep yang lebih modern supaya nggak terlihat kaku seperti aplikasi Java jaman dulu. Fokus utamanya adalah kenyamanan pengguna (*user experience*) dan navigasi yang simpel.

**1. Pembagian Layout Utama**

Saya membagi layar jadi tiga bagian utama supaya rapi:

* **Bagian Atas (Title Bar Kustom)**: Di sini saya nggak pakai *title bar* bawaan Windows yang standar. Saya bikin sendiri supaya warnanya senada dengan aplikasi, lengkap dengan tombol *close*, *maximize*, dan *minimize* yang bisa berubah warna kalau disentuh kursor.
* **Menu Samping (Sidebar)**: Di sisi kiri, saya buatkan menu navigasi dengan warna gelap (biru tua/navy). Di situ ada tombol Dashboard, Profil, dan yang paling penting adalah menu **Kategori**. Menu kategori ini bisa diklik dan bakal muncul pilihan (dropdown) seperti Game, Musik, dan Makanan.
* **Halaman Konten (Main Card)**: Bagian tengah ini fungsinya dinamis. Jadi kalau kita klik menu di samping, isinya akan berubah tanpa harus membuka jendela (frame) baru.

**2. Detail dan Estetika**

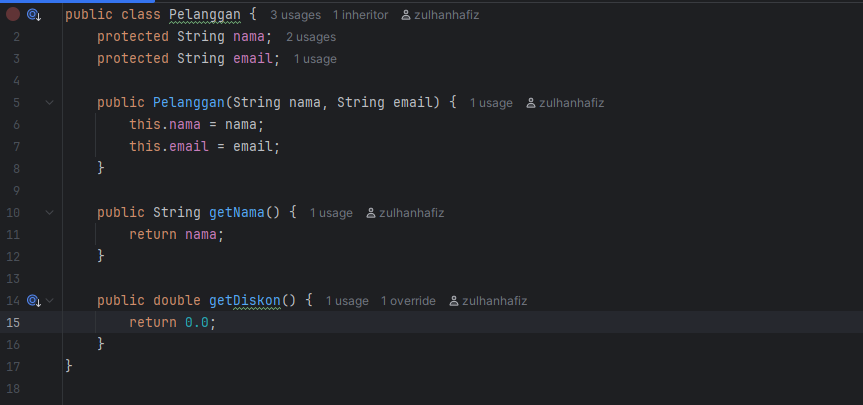
* **Pewarnaan**: Saya sengaja pakai perpaduan warna *Dark Mode* untuk navigasi dan warna putih bersih untuk area kerja supaya mata nggak cepat lelah.
* **Kartu Informasi (Stat Cards)**: Begitu kategori dipilih, informasi event akan muncul dalam bentuk kotak-kotak kartu (mirip dashboard web modern). Isinya ada nama event, lokasi, sampai harganya.
* **Indikator Stok (Progress Bar)**: Saya tambahkan bar warna di bagian bawah inputan tiket. Kalau tiketnya masih banyak warnanya hijau, tapi kalau sudah mau habis, otomatis bar-nya berubah jadi kuning terus merah. Ini buat pengingat visual biar admin tahu stoknya kritis.

**3. Cara Kerja Form**

Inputan di bagian reservasi saya buat cukup interaktif. Contohnya, kotak tulisan (*TextField*) bakal berubah warna garis pinggirnya kalau lagi diklik. Selain itu, saya tambahkan logika validasi: kalau emailnya bukan @gmail.com atau jumlah tiketnya ngawur (minus), aplikasi bakal kasih tahu lewat notifikasi yang bakal hilang sendiri dalam 2 detik. Semua riwayat pendaftaran juga langsung tercatat di kotak "Transaction History" biar bisa dicek ulang.

**f.** Skrip Program dan Penjelasannya

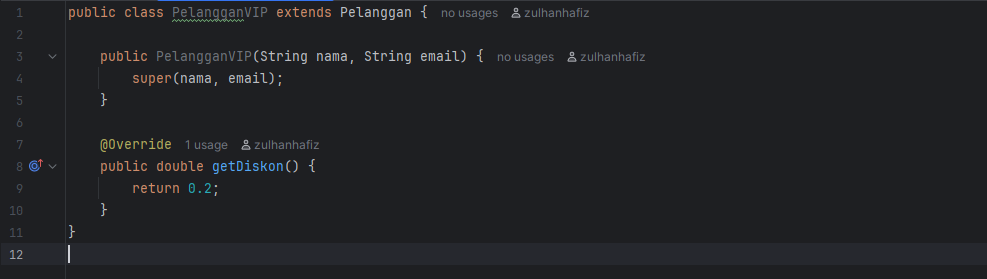
- Class Pelanggan



Class ini merupakan kerangka dasar untuk semua jenis pembeli tiket.

* **public class Pelanggan {**: Baris pembuka untuk membuat class bernama Pelanggan.
* **protected String nama;**: Variabel untuk menyimpan nama, pakai protected supaya anak *class*-nya (VIP) bisa langsung pakai.
* **protected String email;**: Sama seperti nama, ini untuk menyimpan alamat email.
* **public Pelanggan(String nama, String email) {**: Ini *constructor*, fungsinya untuk "menangkap" data saat kita membuat objek pelanggan baru.
* **this.nama = nama;**: Memasukkan nama dari inputan ke dalam variabel class.
* **this.email = email;**: Memasukkan email dari inputan ke dalam variabel class.
* **public String getNama() { return nama; }**: Fungsi simpel untuk mengambil kembali nama pelanggan.
* **public double getDiskon() { return 0.0; }**: Fungsi pemberi diskon. Di sini diatur 0.0 karena pelanggan biasa tidak dapat potongan.

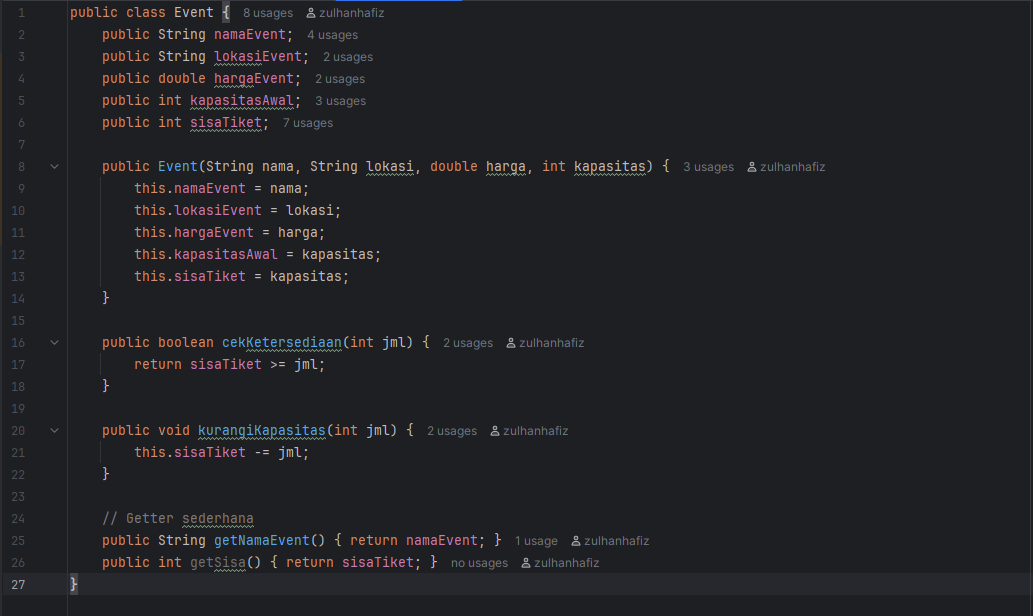
- Class PelangganVIP



Class ini merupakan versi Pewarisan dari Class pelanggan

* **public class PelangganVIP extends Pelanggan {**: Baris ini artinya PelangganVIP adalah "anak" dari Pelanggan (*inheritance*).
* **public PelangganVIP(String nama, String email) {**: *Constructor* untuk pelanggan VIP.
* **super(nama, email);**: Perintah untuk mengirim data nama dan email ke class bapaknya (Pelanggan) agar diproses di sana.
* **@Override**: Tanda bahwa kita mengubah fungsi yang sudah ada di class bapaknya.
* **public double getDiskon() { return 0.2; }**: Di sini kita ubah nilainya jadi 0.2. Jadi setiap yang bertipe VIP otomatis dapat diskon 20%.

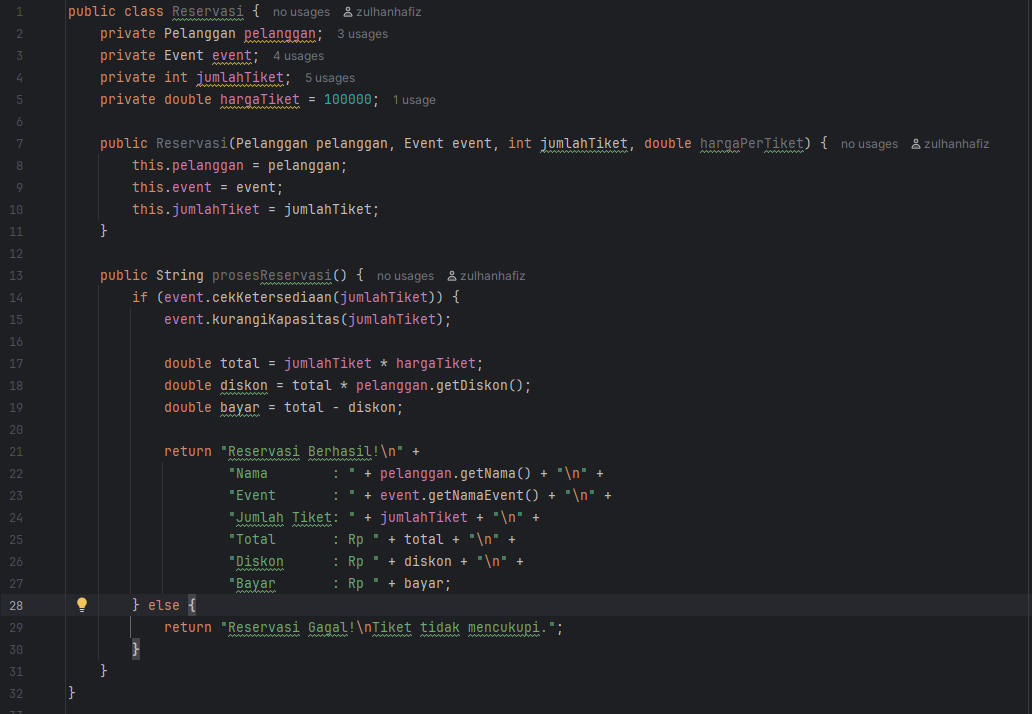
- Class Event



Class ini merupakan class yang mengatur data acara dan stok tiketnya.

* **public String namaEvent, lokasiEvent;**: Variabel untuk menampung info dasar acara.
* **public int kapasitasAwal, sisaTiket;**: Untuk mencatat berapa total kursi dan berapa sisa yang masih bisa dibeli.
* **public Event(...) { ... }**: *Constructor* untuk mengisi data awal sebuah event.
* **this.sisaTiket = kapasitas;**: Saat event baru dibuat, sisa tiket pastinya masih sama dengan kapasitas totalnya.
* **public boolean cekKetersediaan(int jml) { return sisaTiket >= jml; }**: Logika untuk mengecek: "Masih cukup nggak tiketnya kalau dibeli sebanyak jml?".
* **public void kurangiKapasitas(int jml) { this.sisaTiket -= jml; }**: Jika transaksi berhasil, baris ini bertugas memotong jumlah tiket yang ada di stok.

- Class Reservasi

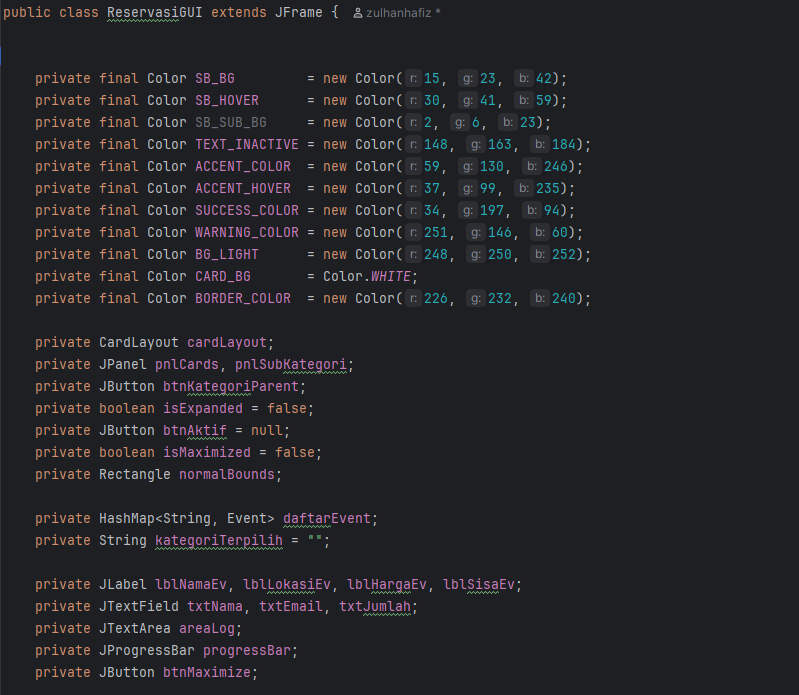


Class ini merupakan Class yang menghitung transaksi.

* **private Pelanggan pelanggan;**: Variabel untuk menampung siapa yang beli.
* **private Event event;**: Variabel untuk menampung event apa yang dipilih.
* **public String prosesReservasi() {**: Fungsi utama untuk menjalankan pendaftaran.
* **if (event.cekKetersediaan(jumlahTiket)) {**: Cek dulu ke class Event, tiketnya masih ada atau tidak.
* **double total = jumlahTiket \* hargaTiket;**: Menghitung harga kotor (jumlah tiket kali harga per lembar).
* **double diskon = total \* pelanggan.getDiskon();**: Menghitung potongan. Di sinilah PBO bekerja: kalau pelanggannya VIP, getDiskon() akan mengambil nilai 20%, kalau biasa ya 0%.
* **double bayar = total - diskon;**: Harga akhir yang harus dibayar setelah dipotong diskon.
* **return "Reservasi Berhasil! ... ";**: Mengirim laporan lengkap berupa teks untuk ditampilkan di layar.

- Class ReservasiGUI

1. Deklarasi Warna dan Variabel Utama



 **public class ReservasiGUI extends JFrame {**: Baris ini mendefinisikan bahwa class ini adalah jendela utama aplikasi yang mewarisi sifat-sifat JFrame.

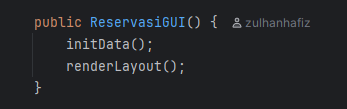
 **private final Color SB\_BG = new Color(15, 23, 42);**: Baris ini (dan baris warna di bawahnya) menetapkan palet warna khusus agar tampilan aplikasi konsisten, menggunakan format kode RGB untuk mendapatkan tema *dark mode* yang modern.

 **private CardLayout cardLayout;**: Variabel ini digunakan untuk mengatur perpindahan antar halaman (seperti Dashboard ke Detail Event) di dalam satu bingkai yang sama.

 **private HashMap<String, Event> daftarEvent;**: Saya menggunakan HashMap untuk menyimpan koleksi data event, di mana "Nama Kategori" menjadi kuncinya untuk memudahkan pemanggilan data.

 **private JProgressBar progressBar;**: Komponen ini disiapkan untuk menampilkan visualisasi sisa stok tiket secara dinamis.

2. Konstruktor dan Inisialisasi Data



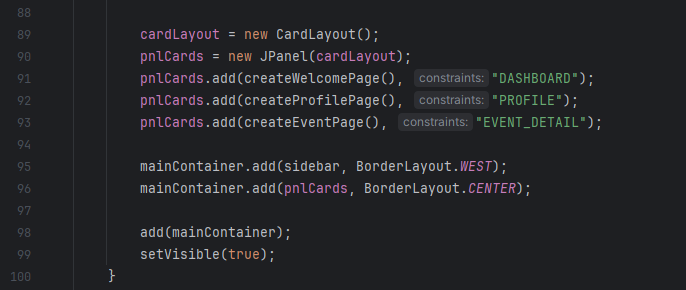
 **public ReservasiGUI() { ... }**: Ini adalah fungsi yang pertama kali dijalankan. Di dalamnya saya memanggil initData() dan renderLayout().

 **daftarEvent = new HashMap<>();**: Baris ini membuat objek penyimpanan data baru di memori.

 **daftarEvent.put("Game", new Event(...));**: Baris-baris ini memasukkan data mentah acara (Nama, Lokasi, Harga, Stok) ke dalam sistem agar siap dipilih oleh pengguna.

3. Pengaturan Tampilan (renderLayout)





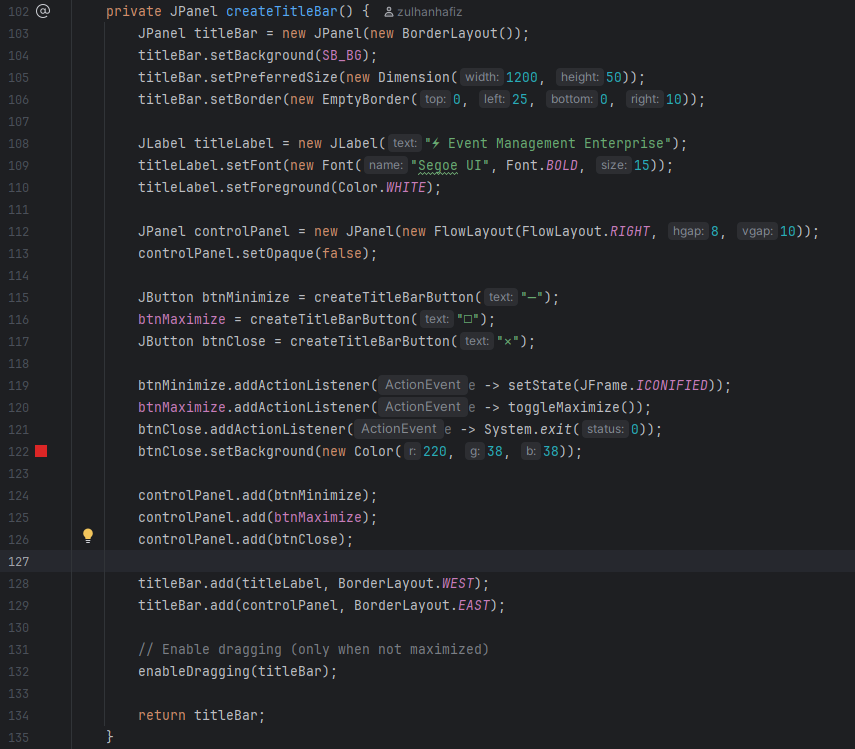
 **Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();**: Kode ini mengambil ukuran monitor pengguna agar aplikasi bisa tampil proporsional di layar mana pun.

 **setSize(width, height);**: Menentukan ukuran jendela aplikasi sebesar 85% dari lebar layar dan 90% dari tinggi layar.

 **setUndecorated(true);**: Perintah ini sangat penting karena berfungsi menghilangkan bingkai standar Windows agar saya bisa mendesain *title bar* sendiri yang lebih estetik.

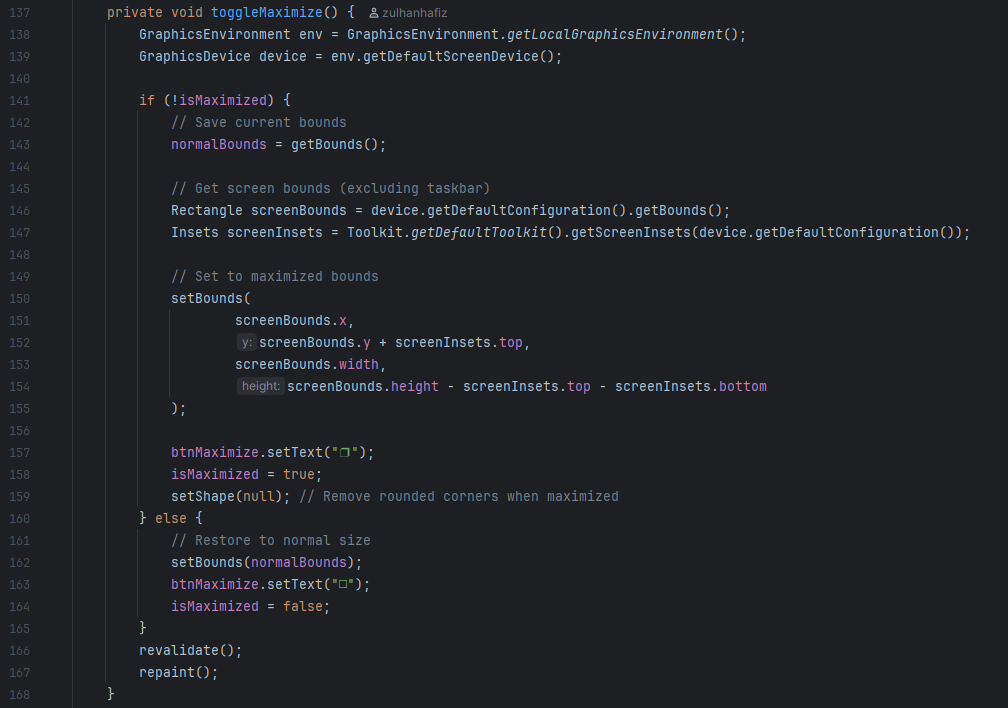
 **mainContainer.add(sidebar, BorderLayout.WEST);**: Meletakkan panel menu navigasi di sisi kiri jendela.

4.Pembuatan Title Bar Kustom



* **JPanel titleBar = new JPanel(new BorderLayout());**: Membuat panel khusus di bagian atas untuk menggantikan tempat judul jendela yang dihilangkan tadi.
* **btnMinimize.addActionListener(e -> setState(JFrame.ICONIFIED));**: Logika untuk meminimalkan jendela saat tombol minus diklik.
* **btnClose.addActionListener(e -> System.exit(0));**: Logika untuk menutup total aplikasi saat tombol silang (X) diklik.
* **enableDragging(titleBar);**: Memanggil fungsi agar jendela aplikasi tetap bisa digeser dengan mouse meskipun tidak memiliki bingkai standar.

**5. Fitur Toggle Maximize (Layar Penuh)**



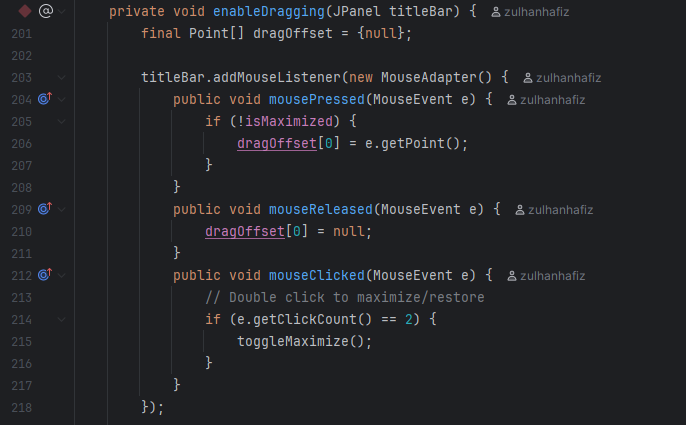
* **if (!isMaximized) { ... }**: Pengecekan kondisi; jika jendela sedang kecil, maka jalankan perintah untuk memperbesar.
* **normalBounds = getBounds();**: Menyimpan posisi dan ukuran jendela terakhir sebelum diperbesar, supaya saat dikecilkan lagi posisinya tidak berubah.
* **Rectangle screenBounds = device.getDefaultConfiguration().getBounds();**: Mengambil koordinat layar penuh (tanpa menutupi *taskbar* Windows).
* **setBounds(screenBounds.x, ...);**: Memaksa jendela untuk memenuhi seluruh area layar monitor.
* **revalidate(); repaint();**: Perintah wajib untuk menyegarkan tampilan GUI agar perubahan ukuran langsung terlihat oleh pengguna.

**6. Pembuatan Tombol Judul (createTitleBarButton)**



* **JButton btn = new JButton(text);**: Membuat tombol baru dengan teks yang ditentukan (seperti "✕" untuk keluar).
* **btn.setBorderPainted(false);**: Menghilangkan garis tepi kotak tombol agar terlihat menyatu dengan barisan judul.
* **btn.setCursor(new Cursor(Cursor.HAND\_CURSOR));**: Mengubah kursor menjadi ikon tangan saat berada di atas tombol, memberi tahu pengguna bahwa bagian ini bisa diklik.
* **public void mouseEntered(MouseEvent e) { ... }**: Logika interaksi saat kursor masuk ke area tombol. Jika itu tombol "✕", warnanya berubah menjadi merah terang; jika bukan, warnanya berubah menjadi abu-abu gelap.
* **public void mouseExited(MouseEvent e) { ... }**: Mengembalikan warna tombol ke semula saat kursor pergi, sehingga aplikasi terasa sangat responsif terhadap gerakan mouse.

**7. Fitur Geser Jendela (enableDragging)**



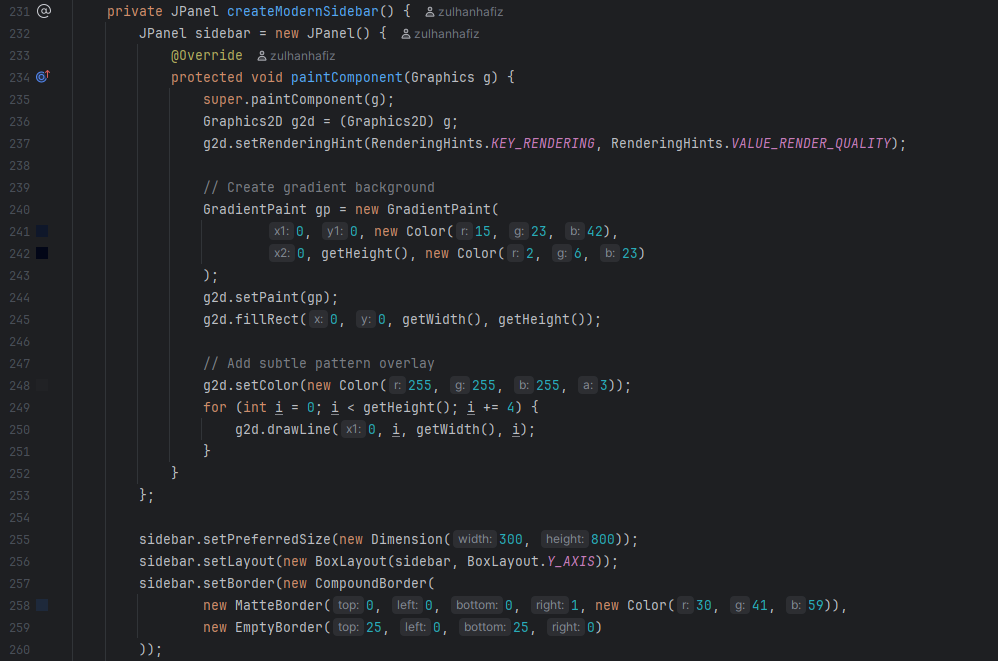
 **final Point[] dragOffset = {null};**: Menyiapkan variabel untuk mencatat koordinat awal saat pengguna menekan mouse di area judul.

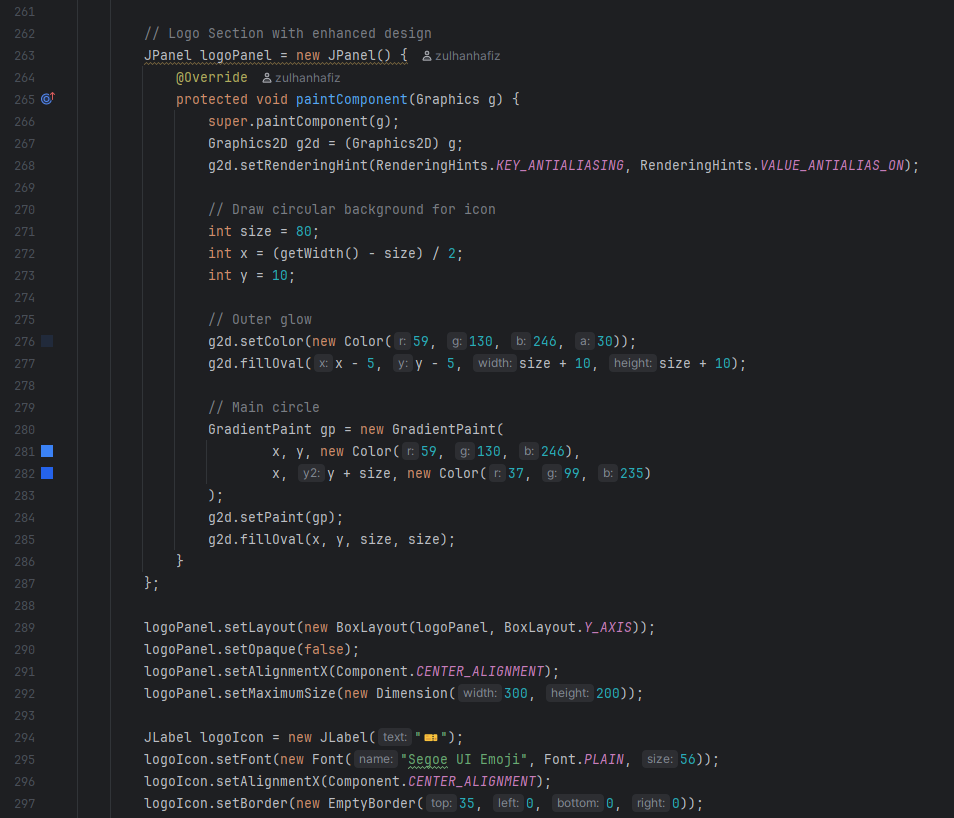
 **dragOffset[0] = e.getPoint();**: Mencatat posisi mouse tepat saat ditekan, selama jendela tidak dalam kondisi layar penuh (*full screen*).

 **if (e.getClickCount() == 2) { toggleMaximize(); }**: Menambahkan fitur kenyamanan; jika area judul diklik dua kali, jendela otomatis berubah antara mode layar penuh atau mode normal.

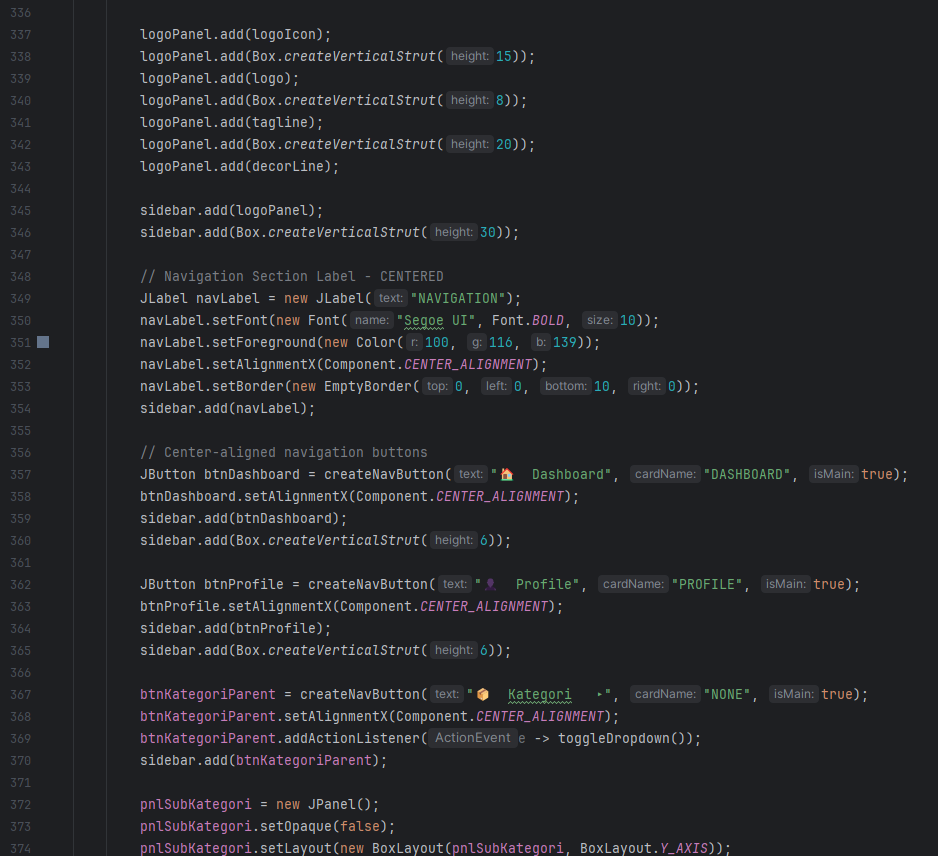
 **setLocation(location.x + e.getX() - dragOffset[0].x, ...);**: Rumus matematika untuk menggerakkan jendela mengikuti gerakan mouse secara halus dengan menghitung selisih koordinat awal dan akhir.

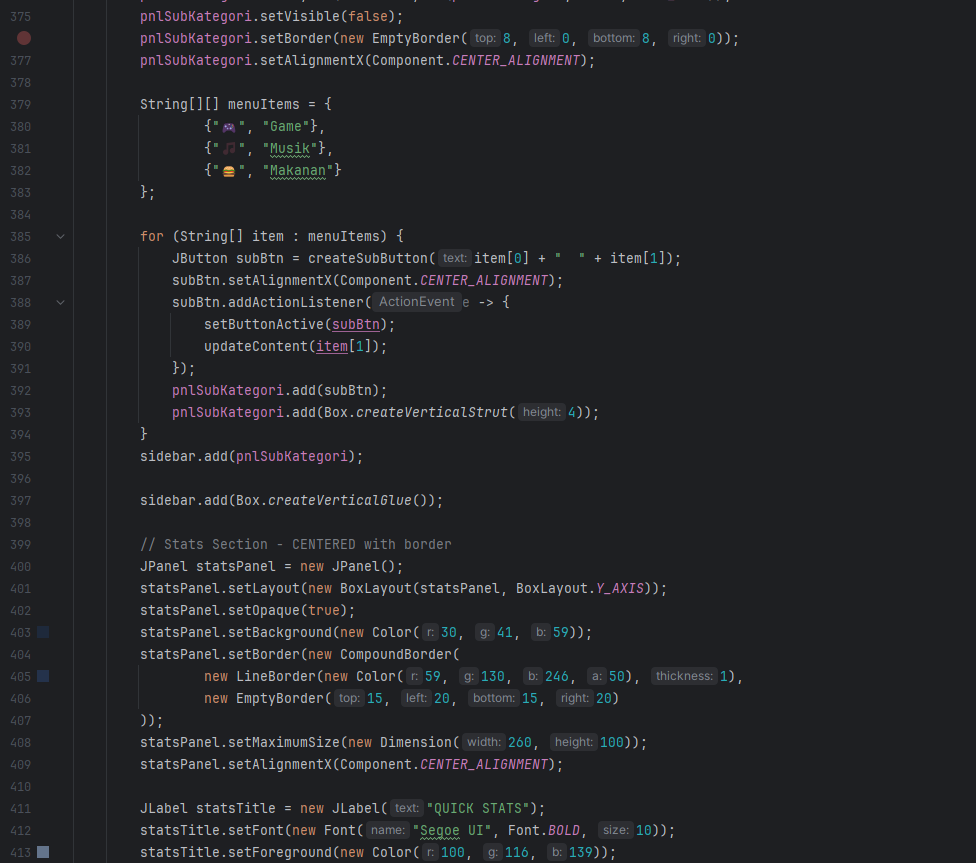
**8. Desain Sidebar Modern (createModernSidebar)**

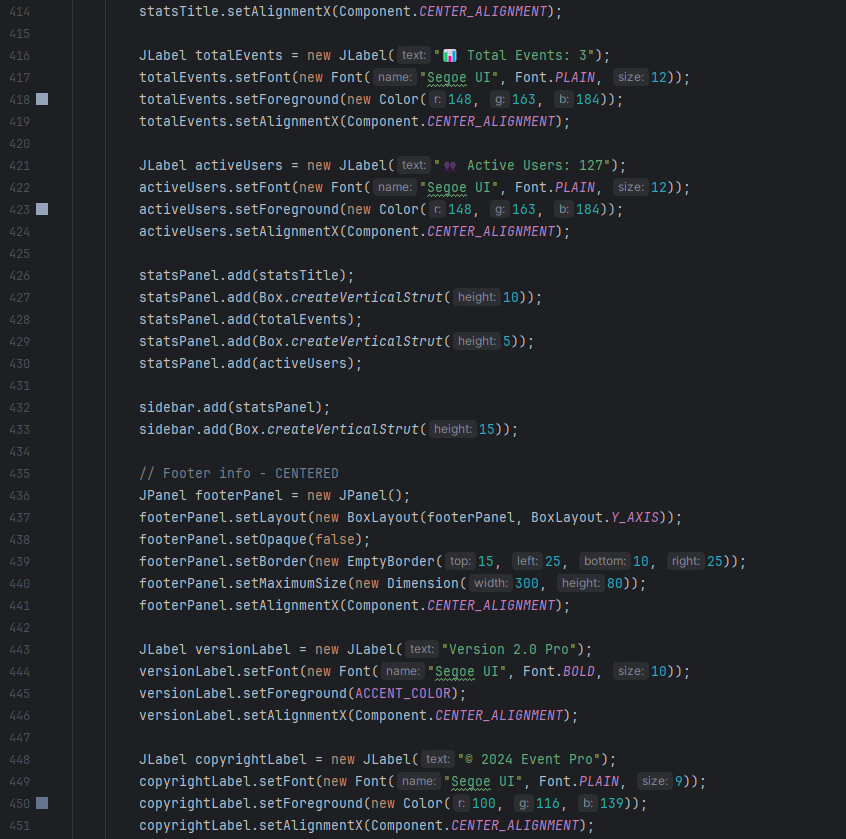


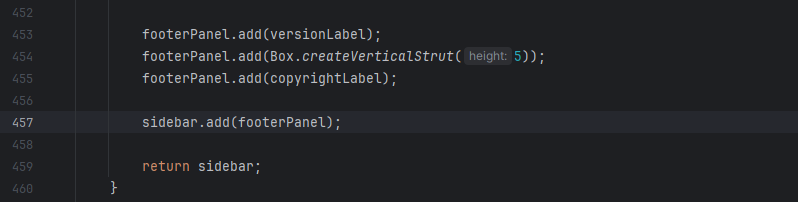








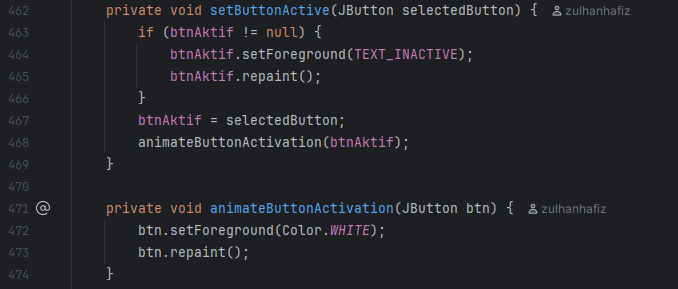




Bagian ini adalah salah satu fitur paling kompleks dari sisi visual karena saya melakukan modifikasi langsung pada cara Java menggambar komponen (*Custom Rendering*).

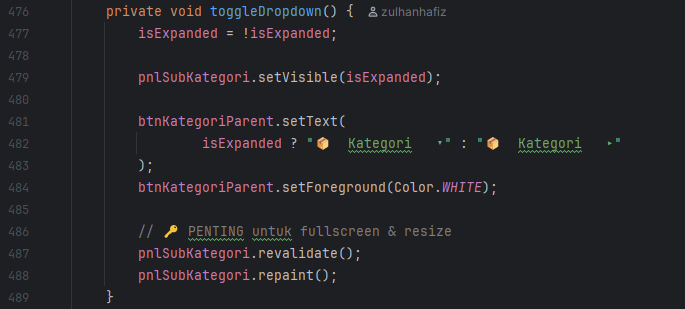
* **protected void paintComponent(Graphics g) { ... }**: Saya melakukan *override* pada metode internal Java Swing ini. Tujuannya adalah untuk mengambil kendali penuh atas bagaimana latar belakang sidebar digambar sebelum komponen lain (seperti tombol) diletakkan di atasnya.
* **Graphics2D g2d = (Graphics2D) g;**: Saya mengubah objek Graphics standar menjadi Graphics2D. Hal ini diperlukan agar saya bisa mengakses fitur grafis yang lebih canggih, seperti pengaturan kualitas gambar (*anti-aliasing*) dan pembuatan warna gradien.
* **g2d.setRenderingHint(RenderingHints.KEY\_RENDERING, ...);**: Baris ini saya gunakan untuk menginstruksikan sistem agar memprioritaskan kualitas visual (*Render Quality*). Ini memastikan bahwa perpindahan warna gradien terlihat halus dan tidak pecah atau bergaris-garis (*banding*).
* **GradientPaint gp = new GradientPaint(0, 0, new Color(15, 23, 42), 0, getHeight(), new Color(2, 6, 23));**: Di sini saya menciptakan efek pencahayaan vertikal. Warna di bagian atas sidebar dibuat sedikit lebih terang (#0F172A), kemudian secara perlahan menggelap hingga ke bagian bawah (#020617). Teknik ini memberikan kesan kedalaman (*depth*) dan dimensi, sehingga aplikasi tidak terlihat "datar" atau membosankan.
* **g2d.setPaint(gp); g2d.fillRect(0, 0, getWidth(), getHeight());**: Perintah ini bertugas mengecat seluruh area sidebar dengan warna gradien yang sudah dibuat tadi.
* **g2d.setColor(new Color(255, 255, 255, 3));**: Saya menambahkan detail halus berupa pola garis-garis tipis. Saya menggunakan warna putih dengan tingkat transparansi (*Alpha*) yang sangat tinggi (hanya 3 dari 255), sehingga garis tersebut hampir tidak terlihat namun memberikan tekstur yang elegan.
* **for (int i = 0; i < getHeight(); i += 4) { g2d.drawLine(0, i, getWidth(), i); }**: Saya menggunakan perulangan (*looping*) untuk menggambar garis horizontal setiap 4 piksel sekali. Teknik ini sering digunakan dalam desain antarmuka modern (disebut *scanline pattern*) untuk memberikan kesan teknologi tinggi atau futuristik pada aplikasi.
* **sidebar.setLayout(new BoxLayout(sidebar, BoxLayout.Y\_AXIS));**: Untuk menyusun menu, saya menggunakan BoxLayout dengan sumbu Y. Ini memastikan bahwa semua elemen seperti logo, navigasi, dan statistik tersusun rapi secara vertikal dari atas ke bawah, mengikuti standar tata letak aplikasi *dashboard* profesional.
* **new MatteBorder(0, 0, 0, 1, new Color(30, 41, 59))**: Terakhir, saya menambahkan garis pembatas (*border*) setebal 1 piksel hanya di sisi kanan sidebar. Garis ini berfungsi sebagai pemisah visual yang tegas antara area menu navigasi dengan area konten utama, sehingga struktur aplikasi terlihat lebih terorganisir.
* **g2d.fillOval(x, y, size, size);**: Menggambar lingkaran dengan warna gradien biru sebagai latar belakang ikon logo.
* **JLabel logoIcon = new JLabel("🎫");**: Menggunakan simbol emoji sebagai ikon aplikasi agar terlihat kekinian tanpa perlu file gambar eksternal.
* **logoPanel.add(decorLine);**: Menambahkan garis pembatas dekoratif di bawah logo untuk memisahkan area identitas dengan area navigasi menu.
* **pnlSubKategori.setVisible(false);**: Secara standar, menu pilihan (Game, Musik, Makanan) disembunyikan terlebih dahulu agar sidebar tetap rapi.
* **for (String[] item : menuItems) { ... }**: Menggunakan perulangan (*looping*) untuk membuat tombol-tombol kategori secara otomatis berdasarkan daftar yang sudah disiapkan.
* **setButtonActive(subBtn); updateContent(item[1]);**: Saat sebuah kategori diklik, tombol tersebut akan ditandai sebagai "aktif" dan isi halaman tengah aplikasi langsung diperbarui sesuai pilihan.

9. Manajemen Status Tombol (setButtonActive)



* **btnAktif.setForeground(TEXT\_INACTIVE);**: Mengubah warna tulisan tombol yang sebelumnya aktif menjadi pudar, menandakan tombol tersebut sudah tidak dipilih lagi.
* **animateButtonActivation(btnAktif);**: Memberikan efek perubahan warna menjadi putih terang pada tombol yang baru saja diklik, sehingga pengguna tahu di halaman mana mereka berada.

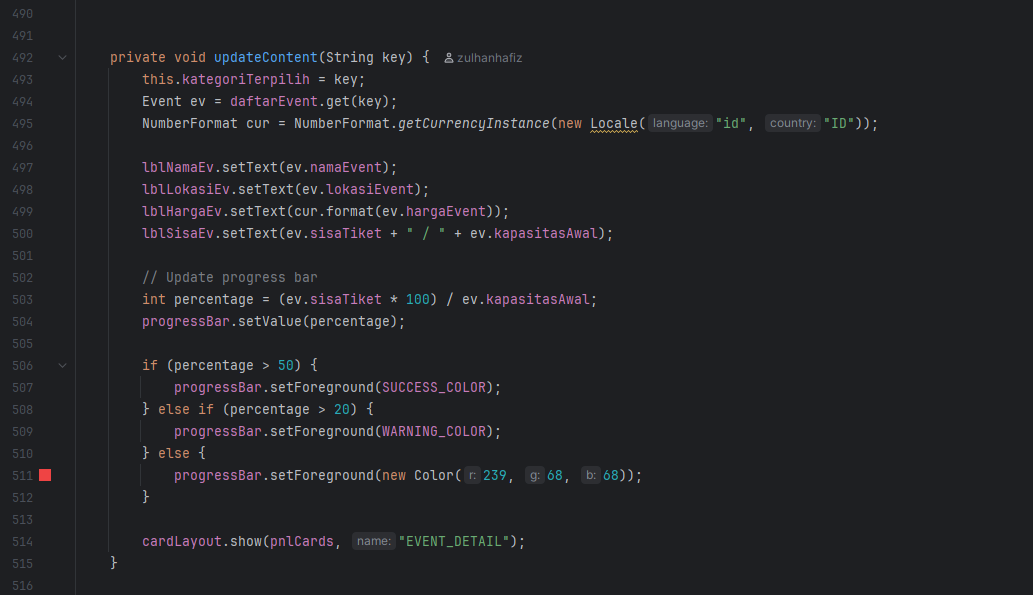
**10. Fungsi Mekanisme Dropdown (toggleDropdown)**

****

Fungsi ini bertugas mengontrol menu "Kategori" di sidebar agar bisa terbuka dan tertutup (seperti menu *accordion* pada website modern).

* **isExpanded = !isExpanded;**: Ini adalah logika *toggle*. Setiap kali tombol diklik, statusnya akan dibalik; jika sebelumnya tertutup (false) maka jadi terbuka (true), dan sebaliknya.
* **pnlSubKategori.setVisible(isExpanded);**: Baris ini secara langsung mengatur muncul atau hilangnya panel yang berisi daftar Game, Musik, dan Makanan berdasarkan status *expanded* tadi.
* **isExpanded ? "📦 Kategori ▾" : "📦 Kategori ▸"**: Saya menggunakan operator *ternary* untuk mengubah ikon panah pada teks tombol. Panah bawah (▾) saat terbuka dan panah samping (▸) saat tertutup, memberikan petunjuk visual yang jelas kepada pengguna.
* **revalidate() dan repaint()**: Ini adalah bagian paling krusial. Karena sidebar bisa berubah ukurannya saat menu terbuka, saya memaksa Java untuk menghitung ulang tata letak (*layout*) dan menggambar ulang komponennya. Tanpa ini, tampilan menu bisa terlihat berantakan atau "patah" saat aplikasi sedang dalam mode *fullscreen*.

**11. Fungsi Pembaruan Konten (updateContent)**



Fungsi ini adalah "jembatan" yang mengambil data dari memori (objek Event) dan menampilkannya ke layar GUI.

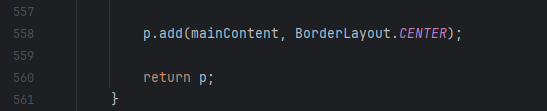
* **daftarEvent.get(key);**: Program mencari data event di dalam HashMap berdasarkan kategori yang diklik pengguna (misalnya "Musik").
* **NumberFormat.getCurrencyInstance(...)**: Saya menggunakan pustaka Java untuk memformat angka harga menjadi format mata uang Rupiah secara otomatis. Jadi, angka 2500000 akan tampil cantik sebagai Rp2.500.000,00 sesuai standar lokal Indonesia.
* **Pembaruan Label**: Baris lblNamaEv.setText(ev.namaEvent) dan seterusnya bertugas mencetak detail acara ke dalam komponen label yang ada di halaman utama.

Di dalam updateContent, saya juga menyisipkan logika untuk memberikan peringatan visual mengenai ketersediaan tiket:

* **Perhitungan Persentase**: Program menghitung sisa tiket dibandingkan kapasitas awal untuk mendapatkan angka persentase.
* **Perubahan Warna Otomatis**:
  + Jika tiket di atas 50%, bar berwarna **Hijau** (aman).
  + Jika tiket di bawah 50% tapi di atas 20%, bar berubah menjadi **Oranye/Kuning** (peringatan).
  + Jika tiket sudah sangat tipis (di bawah 20%), bar berubah menjadi **Merah** (kritis).
* **cardLayout.show(...)**: Baris terakhir ini berfungsi sebagai perintah untuk memindahkan "kartu" tampilan dari halaman Dashboard/Profil langsung ke halaman Detail Event yang baru saja diperbarui datanya.

**12.** Halaman Utama Event (createEventPage)

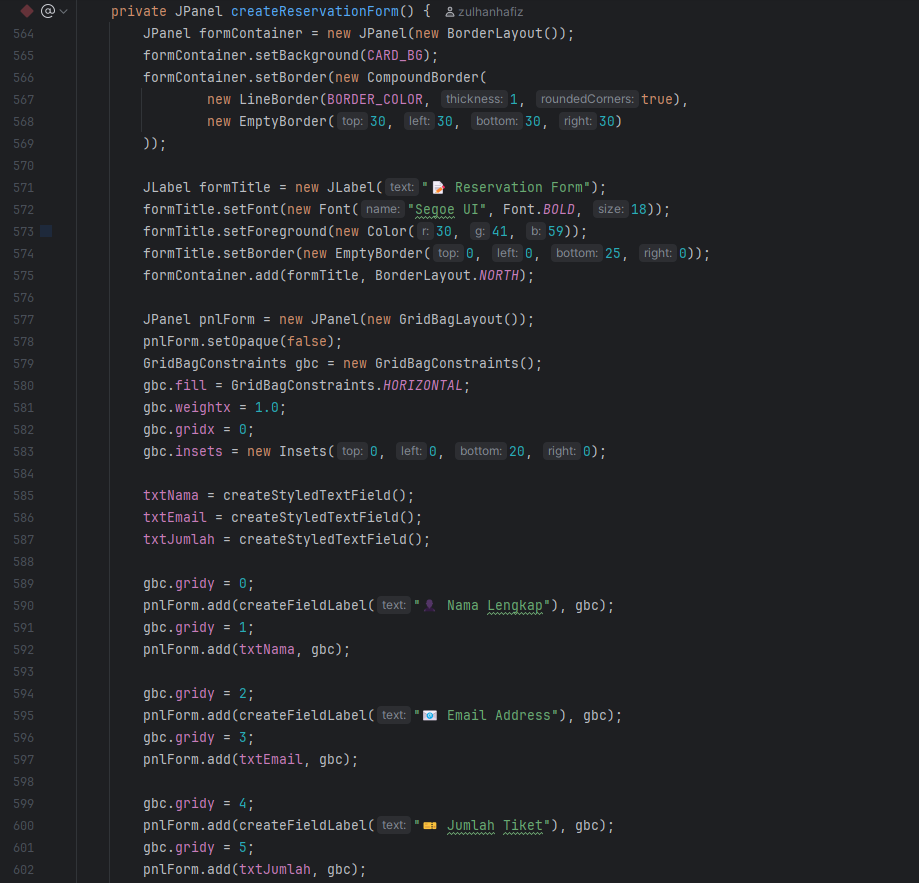


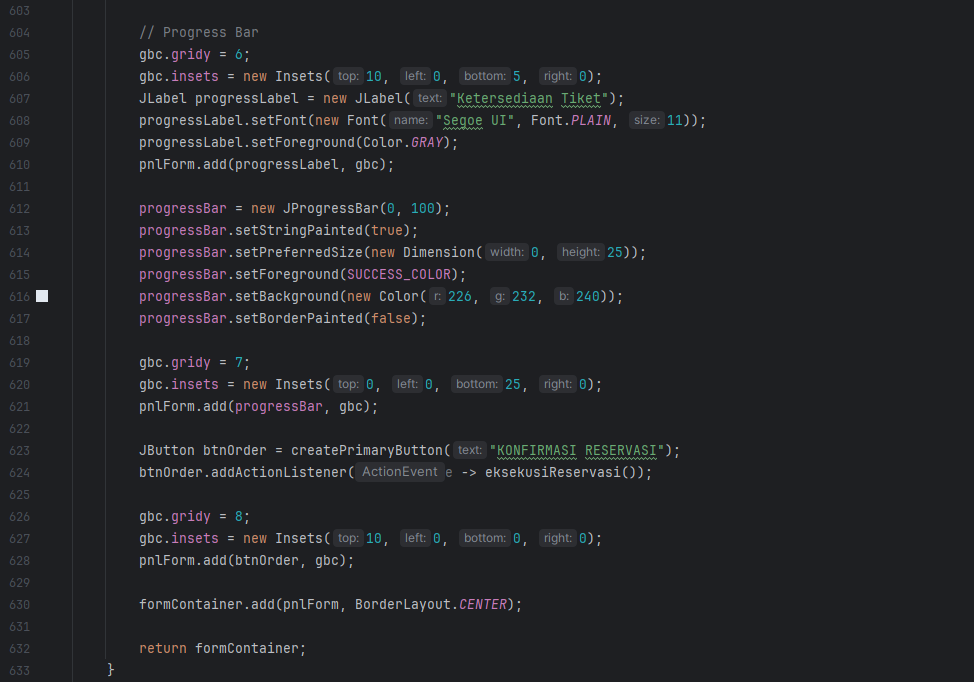


Metode ini membangun "panggung" utama tempat informasi event ditampilkan.

* **Layouting**: Saya menggunakan BorderLayout dengan jarak (*gap*) sebesar 25 piksel agar antar komponen tidak terlihat berhimpitan. Seluruh halaman diberi warna BG\_LIGHT yang lembut di mata.
* **Stat Cards (Header)**: Di bagian atas, saya membuat barisan kartu informasi menggunakan GridLayout(1, 4). Empat kartu ini menampilkan Nama Event, Lokasi, Harga, dan Sisa Tiket secara instan. Ini memberikan ringkasan cepat kepada pengguna tanpa harus membaca teks yang panjang.
* **Pembagian Area**: Di bawah header, layar dibagi menjadi dua kolom besar (GridLayout 1, 2). Sebelah kiri berisi formulir pendaftaran, dan sebelah kanan berisi riwayat transaksi.

**13.** Formulir Reservasi (createReservationForm)





Ini adalah area interaktif tempat pengguna memasukkan data mereka.

* **Wadah (Container)**: Saya menggunakan CompoundBorder yang menggabungkan garis tipis dengan ruang kosong (*padding*) di dalamnya, sehingga form terlihat seperti kartu putih yang mengapung dan bersih.
* **GridBagLayout**: Untuk menyusun kolom input, saya menggunakan GridBagLayout. Ini adalah pengatur tata letak paling fleksibel di Java, yang memungkinkan saya mengatur posisi label dan kotak input secara presisi satu per satu dari atas ke bawah.
* **Visual Indikator (ProgressBar)**: Saya menyisipkan JProgressBar tepat di atas tombol konfirmasi. Tujuannya sebagai indikator visual ketersediaan tiket; warnanya akan berubah otomatis (hijau, kuning, atau merah) sesuai dengan sisa stok yang ada.

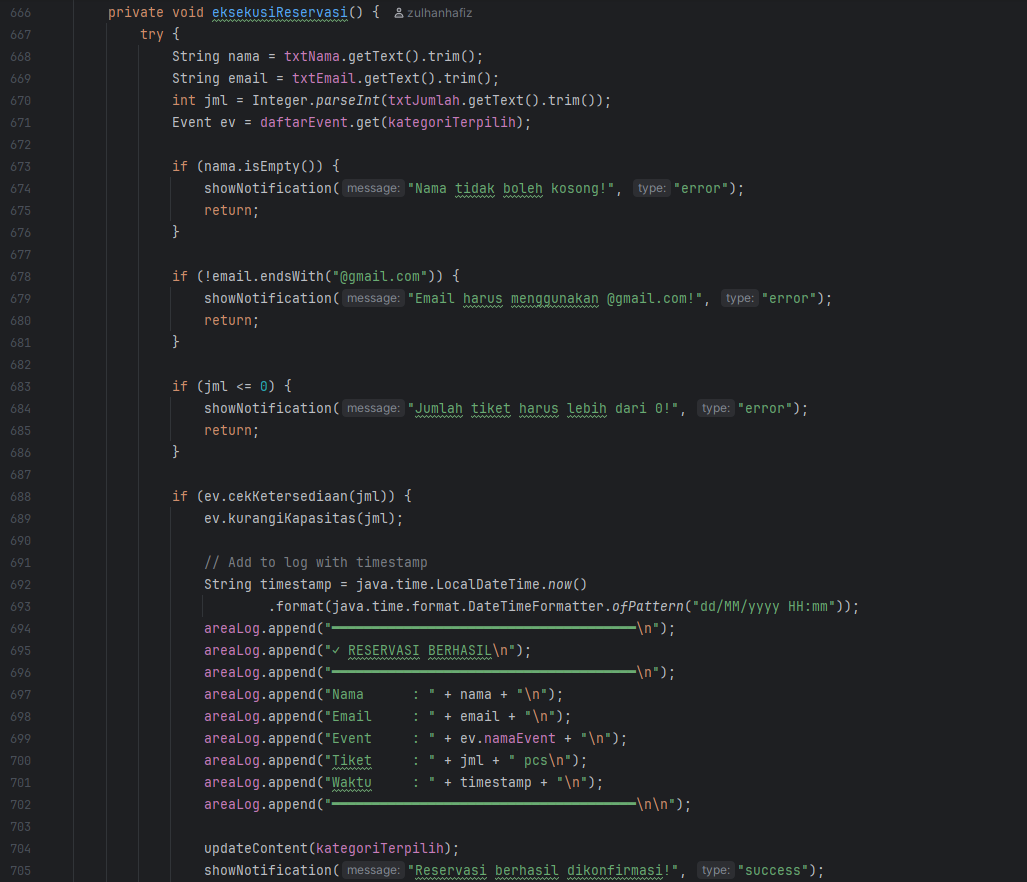
14. Riwayat Transaksi (createTransactionLog)



Bagian ini berfungsi sebagai "struk digital" sementara.

* **JTextArea**: Saya menggunakan kotak teks yang diatur menjadi setEditable(false) agar pengguna tidak bisa mengubah riwayat yang sudah tercatat.
* **JScrollPane**: Karena riwayat bisa menjadi sangat panjang, saya membungkusnya dalam fungsi *scroll*. Saya juga mengatur kecepatan geser (*unit increment*) agar pergerakan layar saat di-*scroll* terasa lebih halus.

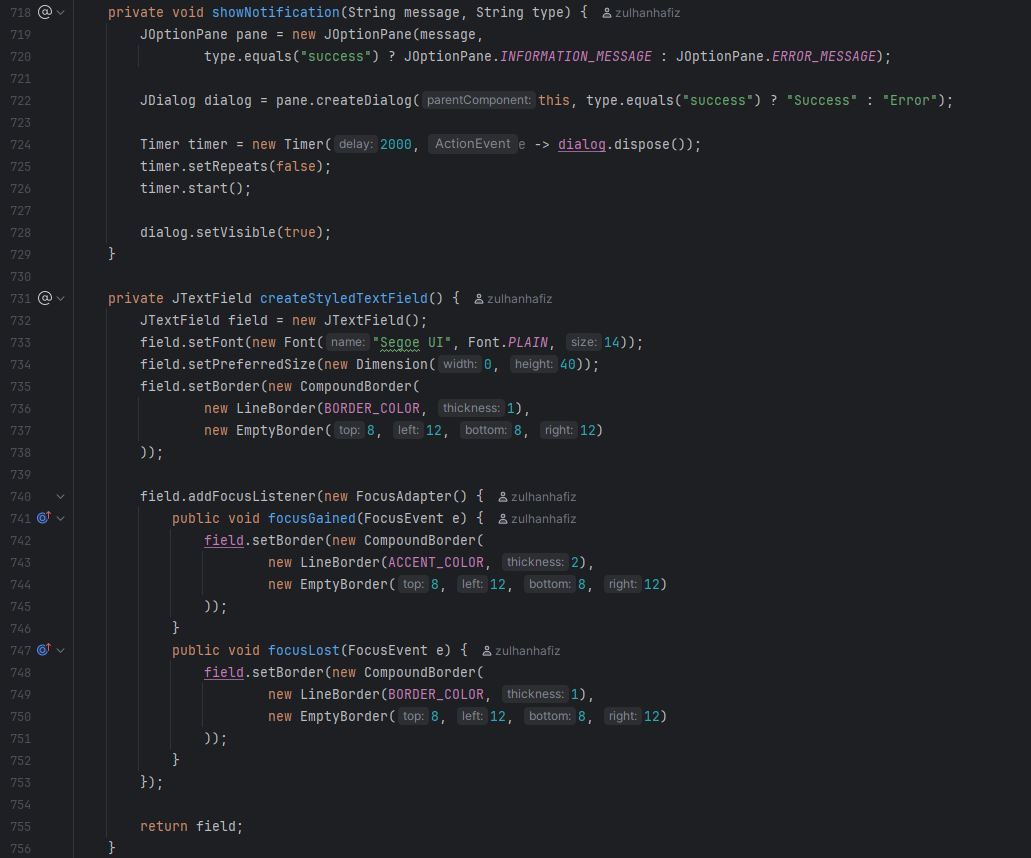
15. Logika Eksekusi (eksekusiReservasi)



Ini adalah bagian terpenting karena di sinilah semua data diproses secara teknis.

* **Validasi Data**: Sebelum data diproses, program melakukan tiga pengecekan: nama tidak boleh kosong, email harus diakhiri dengan @gmail.com, dan jumlah tiket harus berupa angka positif. Jika salah satu gagal, fungsi showNotification akan muncul.
* **Sinkronisasi Stok**: Jika validasi lolos, program memanggil metode ev.cekKetersediaan(jml). Jika tiket masih ada, stok di objek Event akan langsung dipotong.
* **Timestamping**: Saya menambahkan kode untuk mengambil waktu sistem secara otomatis menggunakan LocalDateTime. Jadi, setiap struk yang muncul di histori akan memiliki catatan jam dan tanggal yang akurat.
* **Reset Form**: Setelah sukses, semua kolom input akan dibersihkan kembali secara otomatis agar admin bisa langsung menginput data pelanggan berikutnya tanpa perlu menghapus teks lama secara manual.

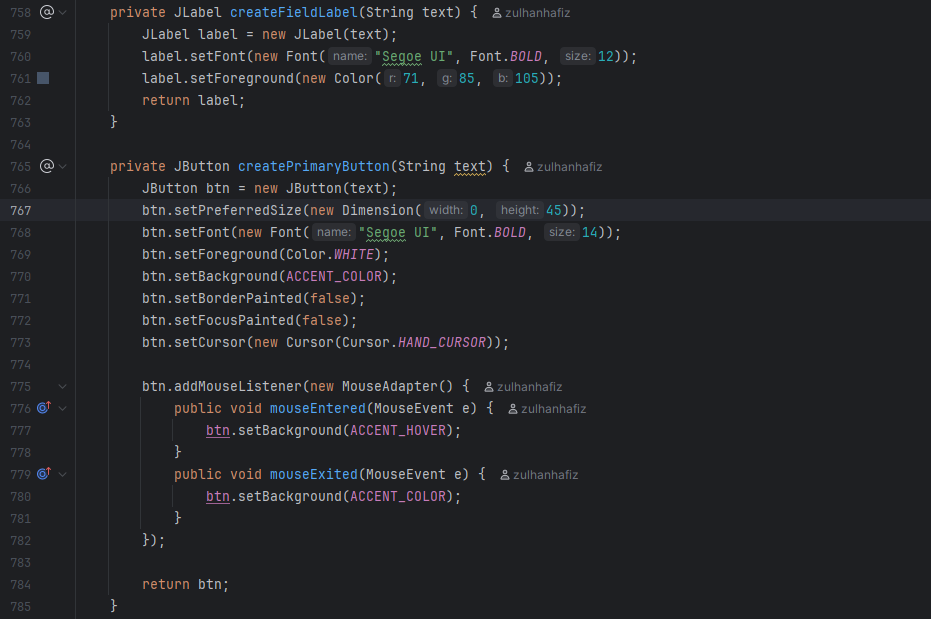
**16**. Notifikasi & Styling Input (showNotification & createStyledTextField)



 **Notifikasi Pintar**: Fungsi showNotification tidak hanya menampilkan pesan, tapi saya pasang Timer selama 2000ms (2 detik). Jadi, jendela pesan akan menutup sendiri secara otomatis tanpa perlu diklik "OK", sangat menghemat waktu pengguna.

 **Efek Fokus**: Pada kolom input, saya menambahkan FocusListener. Ketika pengguna mengklik kotak input, garis pinggirnya akan menebal dan berubah warna menjadi biru (ACCENT\_COLOR). Ini adalah standar desain modern agar pengguna tahu persis kolom mana yang sedang aktif mereka ketik.

17. Label Form & Tombol Utama

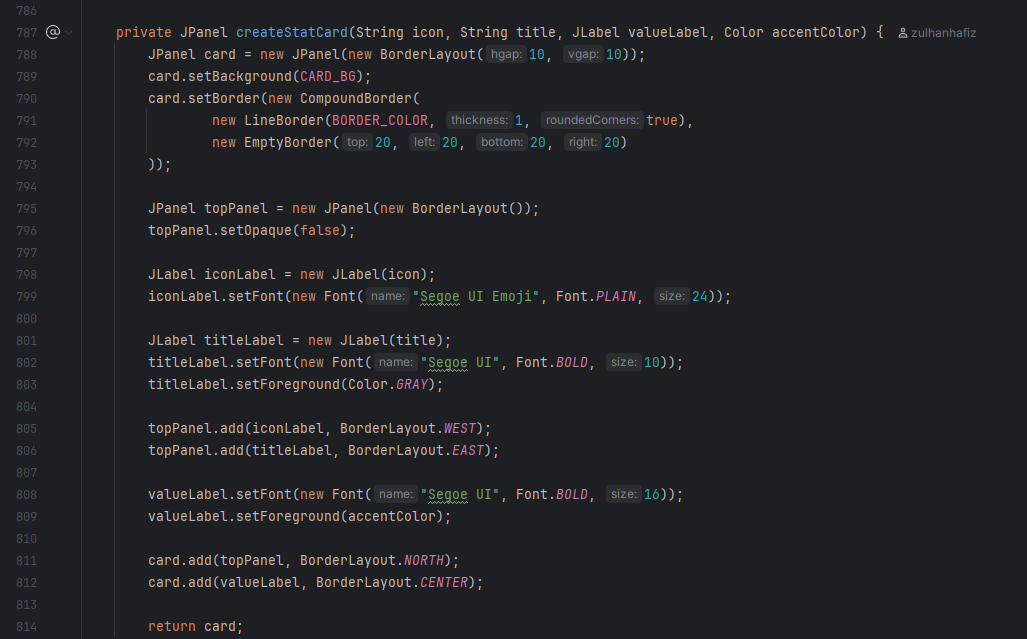


 **createFieldLabel**: Fungsi ini digunakan untuk membuat judul di atas kotak input (seperti "Nama Lengkap"). Saya mengaturnya dengan font **Segoe UI BOLD** dan warna abu-abu gelap agar terlihat profesional dan mudah dibaca.

 **createPrimaryButton**: Ini adalah pabrik pembuat tombol konfirmasi.

* Saya menggunakan **MouseListener** untuk memberikan efek *feedback*. Saat kursor masuk (mouseEntered), warna tombol akan berubah sedikit lebih gelap (ACCENT\_HOVER), dan kembali normal saat kursor keluar. Ini penting agar pengguna tahu bahwa tombol tersebut aktif dan bisa diklik.

18. Kartu Statistik (createStatCard)

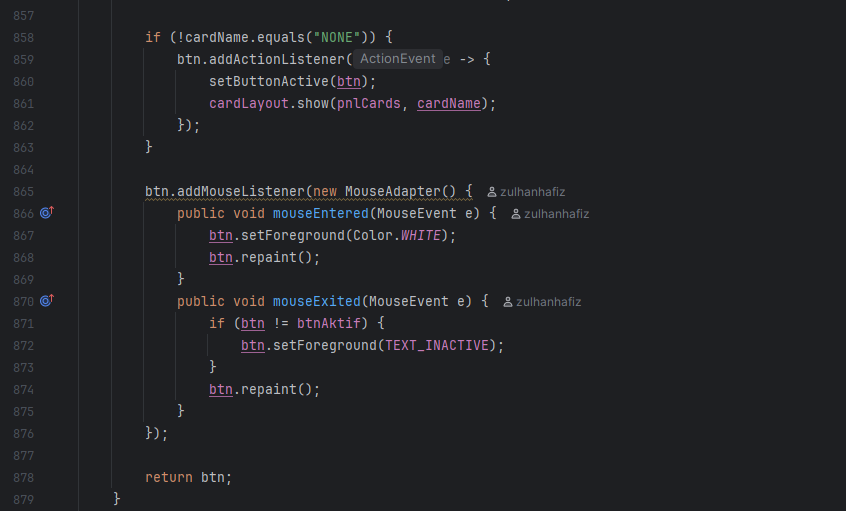


Metode ini menghasilkan kotak informasi yang ada di bagian atas halaman detail event.

* **Struktur Layout**: Menggunakan BorderLayout di dalam panel kecil. Bagian atas (NORTH) diisi oleh ikon dan judul kecil, sementara bagian tengah (CENTER) diisi oleh nilai datanya.
* **Desain Kartu**: Menggunakan CompoundBorder untuk menciptakan efek kartu yang rapi dengan garis tepi tipis dan jarak dalam (*padding*) yang pas (20 piksel di setiap sisi).
* **Fleksibilitas**: Fungsi ini menerima parameter accentColor, sehingga setiap kartu (Nama, Lokasi, Harga, Stok) bisa memiliki warna teks yang berbeda-beda sesuai fungsinya.

19. Tombol Navigasi Utama (createNavButton)

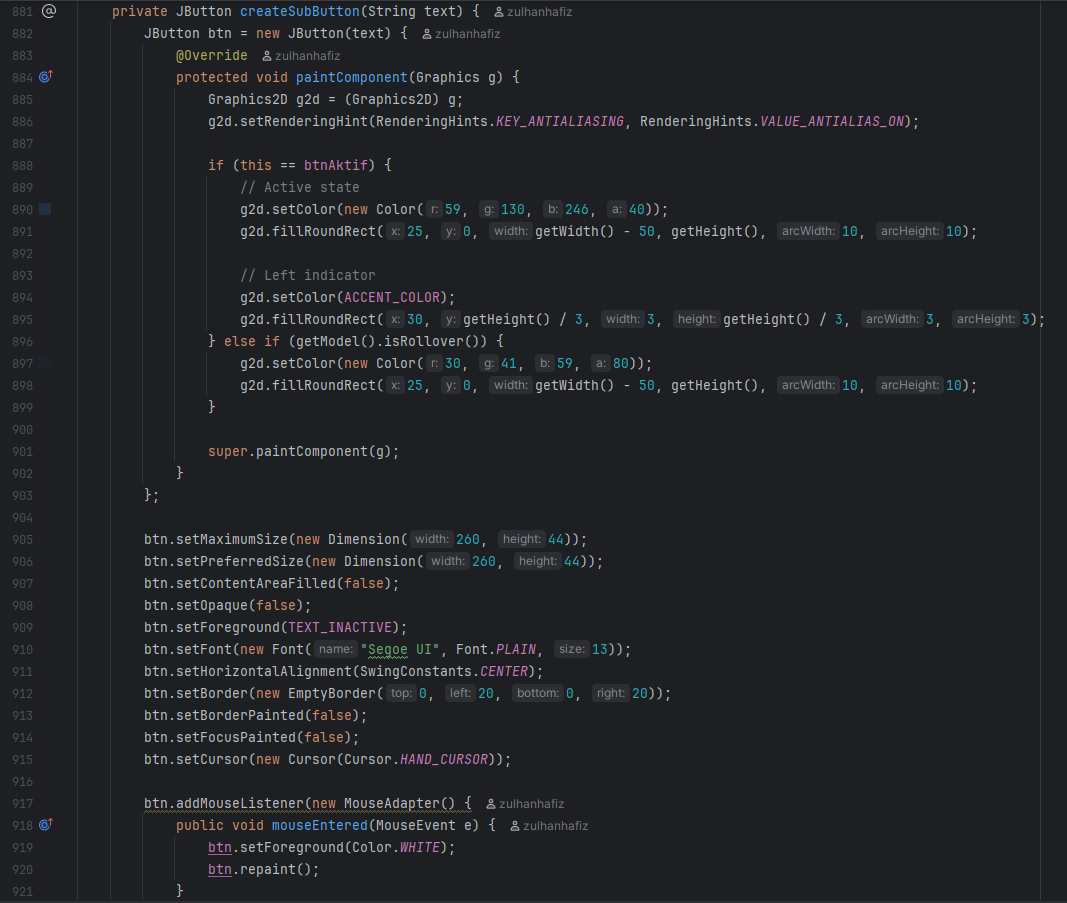


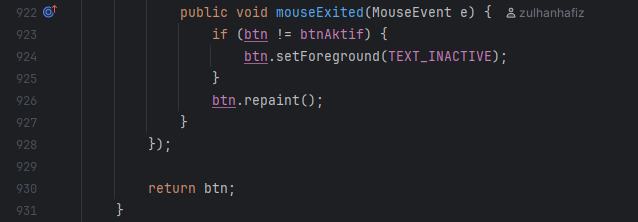


Ini adalah salah satu bagian tersulit karena saya melakukan *custom painting* pada tombol navigasi di sidebar.

* **paintComponent**: Saya menulis ulang cara tombol digambar. Jika tombol tersebut sedang aktif (this == btnAktif), program akan menggambar latar belakang gradien biru dan sebuah garis indikator putih kecil di sisi kiri.
* **fillRoundRect**: Saya tidak menggunakan kotak kaku, melainkan bentuk persegi panjang dengan sudut membulat (*rounded*) agar terlihat seperti aplikasi modern zaman sekarang.
* **cardLayout.show**: Tombol ini juga berfungsi sebagai pemicu perpindahan halaman. Begitu diklik, layar utama akan langsung berganti sesuai nama halaman yang dituju.

**20**. Tombol Sub-Menu (createSubButton)





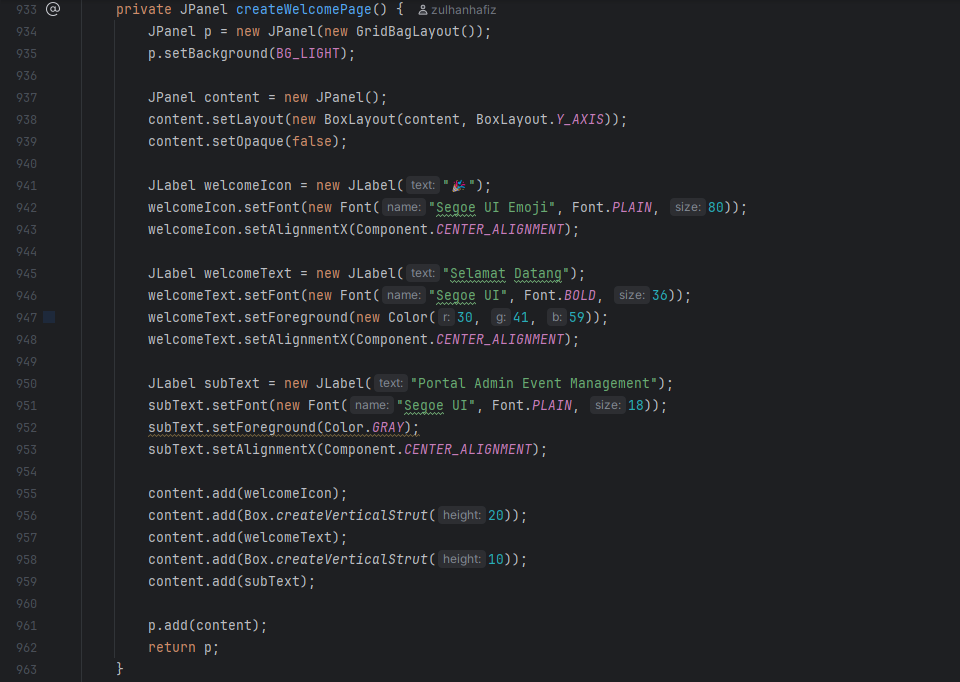
Mirip dengan tombol navigasi, tetapi desainnya dibuat lebih minimalis karena ini merupakan menu turunan (seperti pilihan Game atau Musik).

* **Indikator Tipis**: Saat dipilih, tombol ini tidak memberikan warna penuh, melainkan hanya garis indikator kecil dan latar belakang biru yang sangat transparan (Alpha: 40).
* **Penyesuaian Ukuran**: Ukurannya dibuat sedikit lebih kecil dari menu utama agar secara visual pengguna tahu bahwa ini adalah bagian dari menu "Kategori".

Pada kedua jenis tombol navigasi di atas, saya menambahkan logika pembaruan warna teks:

* Tulisan akan berubah menjadi **Putih Terang** saat kursor menyentuh tombol, dan kembali menjadi **Abu-abu Pudar** saat kursor menjauh (kecuali jika tombol tersebut sedang aktif).
* **btn.repaint()**: Perintah ini sangat penting untuk memicu pembaruan visual secara instan setiap kali ada interaksi mouse, sehingga transisi warna terlihat halus.

21. Halaman Selamat Datang (createWelcomePage)

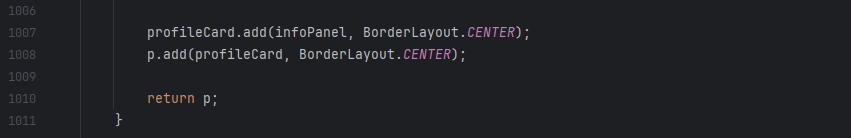


Metode ini membuat tampilan pertama yang dilihat pengguna saat aplikasi dibuka (Dashboard).

* **new GridBagLayout()**: Saya menggunakan layout ini pada panel utama agar semua konten di dalamnya otomatis berada tepat di tengah-tengah layar, baik secara vertikal maupun horizontal.
* **BoxLayout Y\_AXIS**: Di dalam panel konten, saya menyusun elemen secara vertikal dari atas ke bawah. Ini mencakup ikon emoji "🎉", teks judul, dan sub-judul.
* **Box.createVerticalStrut(20)**: Baris ini berfungsi untuk memberikan jarak kosong yang tetap (spacer) antar teks, sehingga tampilan tidak terlihat menumpuk dan lebih nyaman dipandang.
* **Alignment**: Semua elemen diatur ke CENTER\_ALIGNMENT agar teks dan ikon benar-benar sejajar di tengah.

**22.** Halaman Profil (createProfilePage)

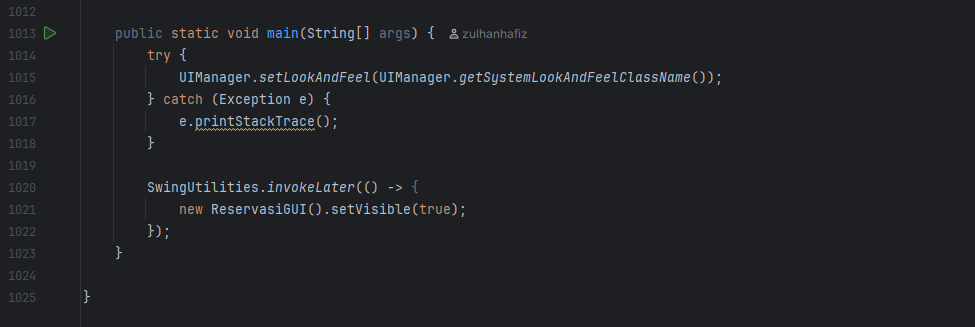




Halaman ini dirancang untuk menampilkan informasi admin atau pengguna sistem.

* **BorderLayout & Border**: Panel luar menggunakan BorderLayout dan saya memberikan EmptyBorder yang cukup lebar (40-50 piksel) di sekelilingnya agar konten profil tidak menyentuh pinggir jendela.
* **Profile Card**: Identitas user dibungkus di dalam sebuah panel bernama profileCard. Sama seperti desain kartu lainnya, saya menggunakan kombinasi LineBorder untuk garis tepi dan warna latar belakang putih bersih (CARD\_BG) agar terlihat menonjol dari latar belakang aplikasi.
* **Informasi Admin**: Di sini saya menempatkan avatar (👤), nama admin, dan peran sistemnya (*System Administrator*). Penggunaan font **Segoe UI BOLD** ukuran 24 memberikan penegasan pada nama pengguna.

23. Metode Utama (main method)

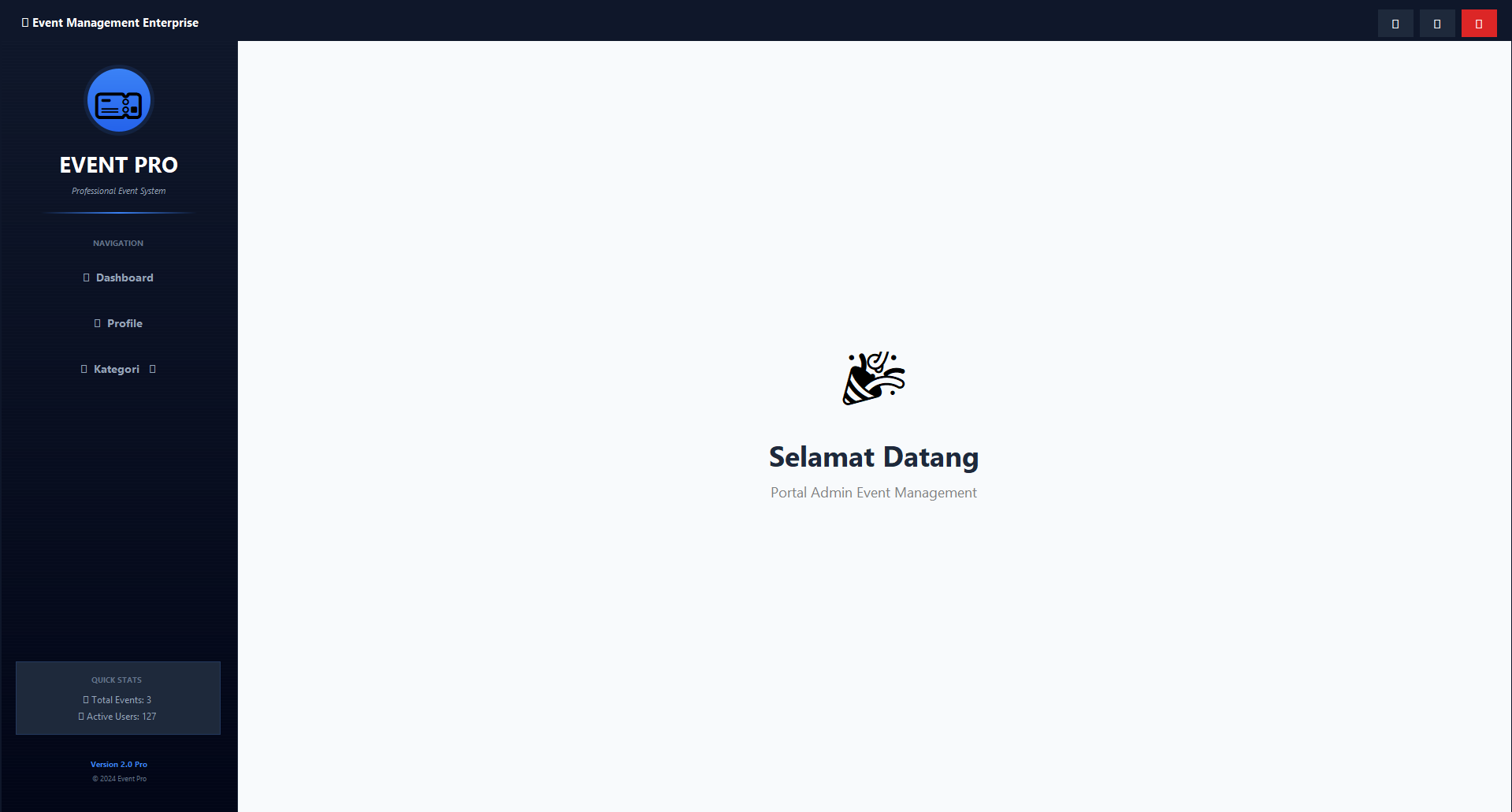


Ini adalah pintu masuk utama agar program bisa berjalan di komputer.

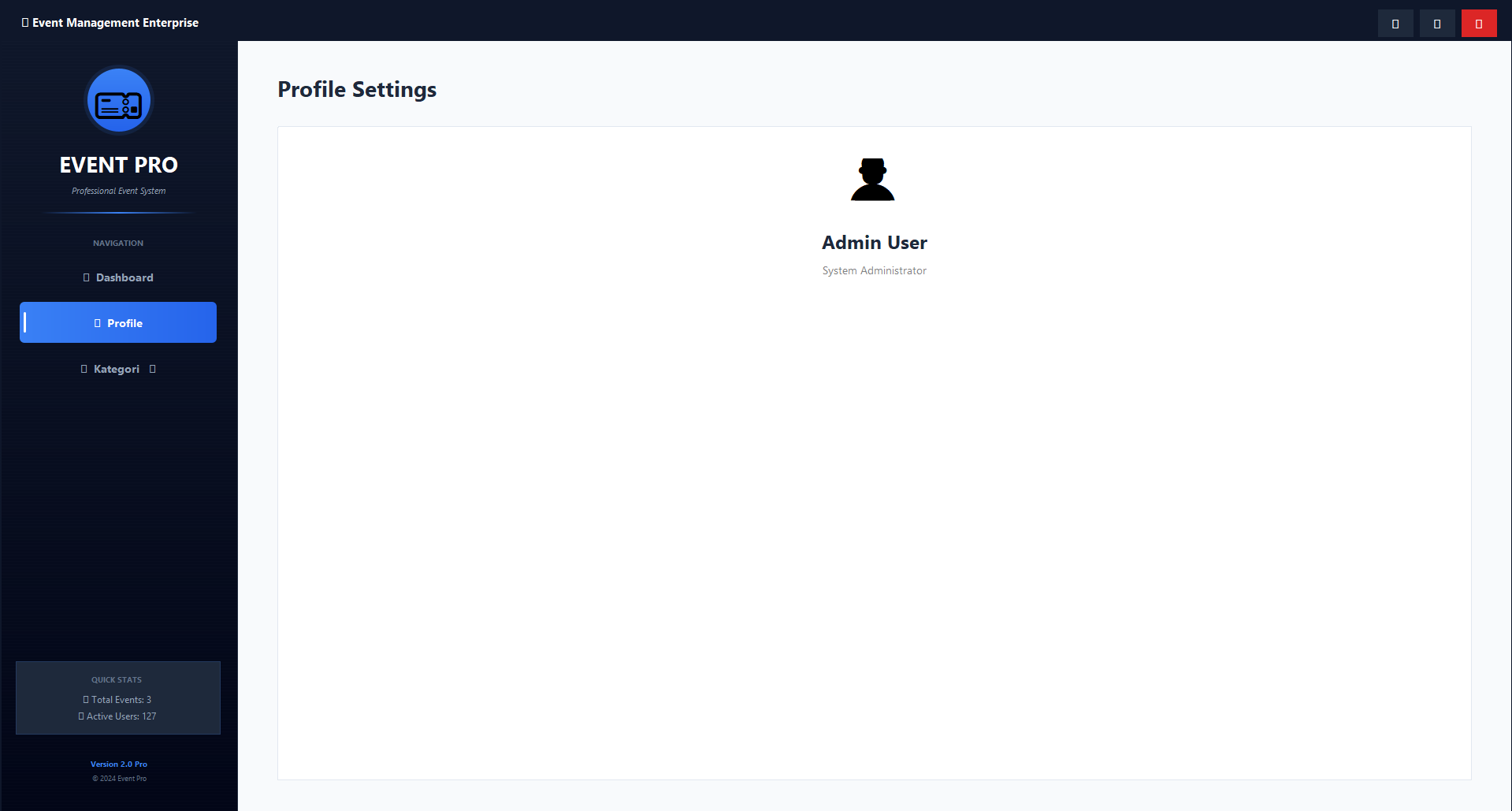
* **UIManager.setLookAndFeel(...)**: Baris ini sangat penting secara estetika. Fungsinya adalah memerintahkan Java agar menggunakan gaya tampilan (Look and Feel) asli dari sistem operasi (misalnya gaya Windows atau Mac), sehingga tombol dan kolom input tidak terlihat kaku seperti gaya bawaan Java lama.
* **Try-Catch**: Saya membungkus pengaturan *Look and Feel* dalam blok ini untuk mencegah aplikasi *crash* jika sistem operasi tidak mendukung gaya yang diminta.
* **SwingUtilities.invokeLater(...)**: Ini adalah standar keamanan dalam pemrograman GUI Java. Kode ini memastikan bahwa pembuatan jendela aplikasi dilakukan di dalam *Event Dispatch Thread* (EDT), yang artinya antarmuka akan berjalan lebih stabil dan tidak mudah membeku (*freeze*).
* **new ReservasiGUI().setVisible(true)**: Baris terakhir ini akhirnya memanggil konstruktor class yang sudah kita buat sebelumnya dan memunculkan jendela aplikasi ke layar pengguna.

**g.** Penjelasan Screenshot Tampilan Aplikasi

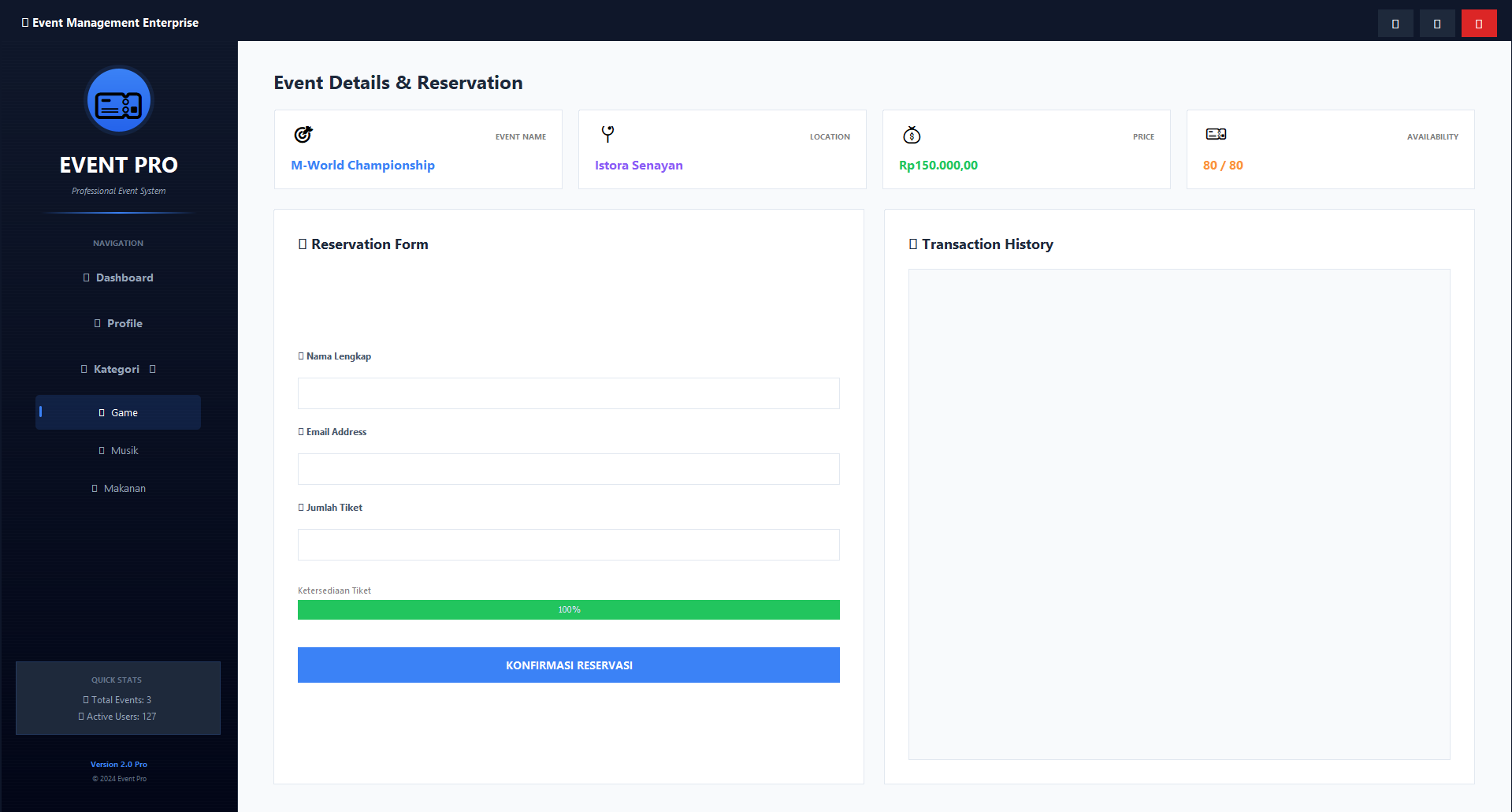
**1. Halaman Utama (Dashboard)**

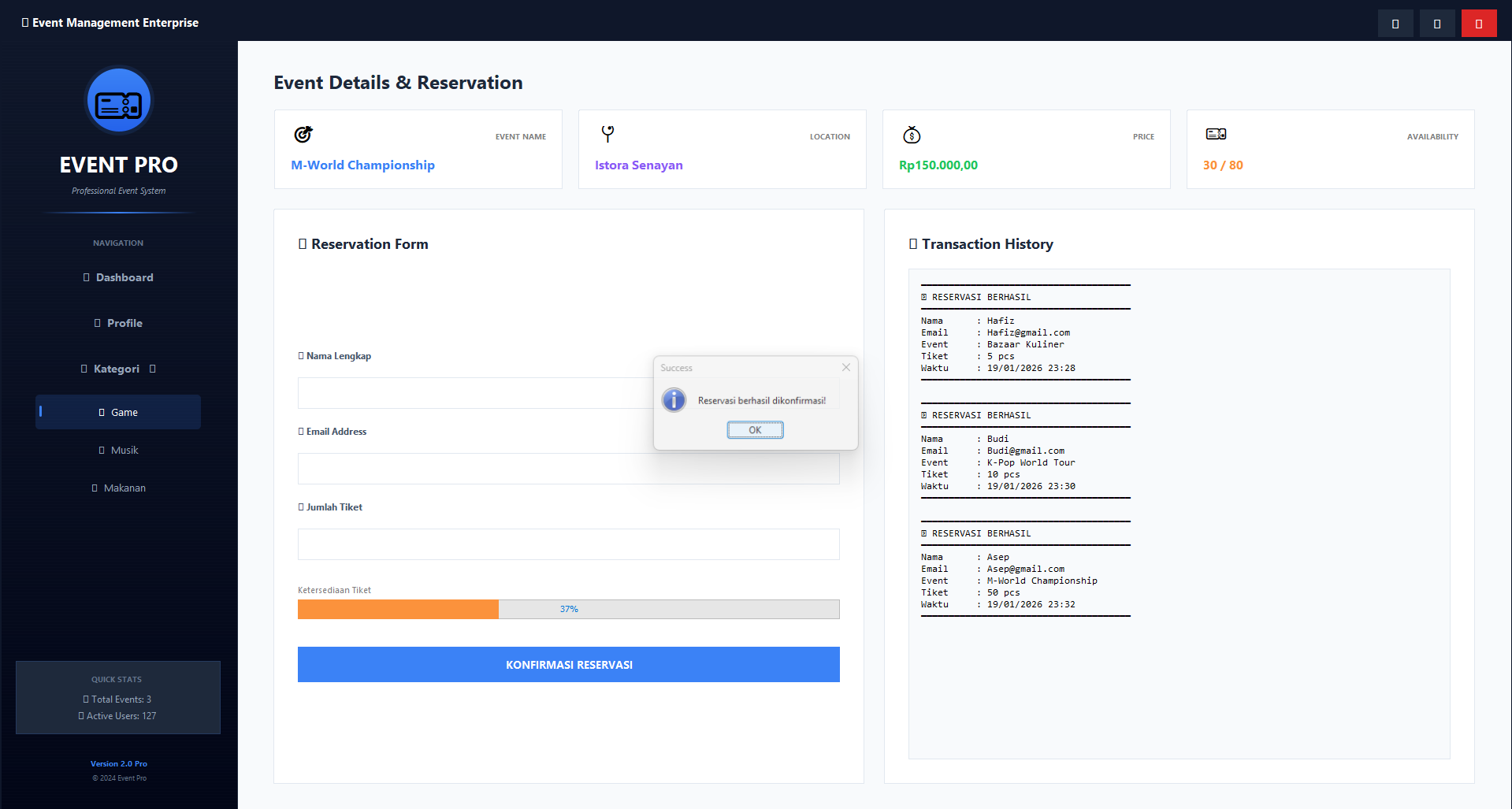
****

**2. Halama Profile**

****

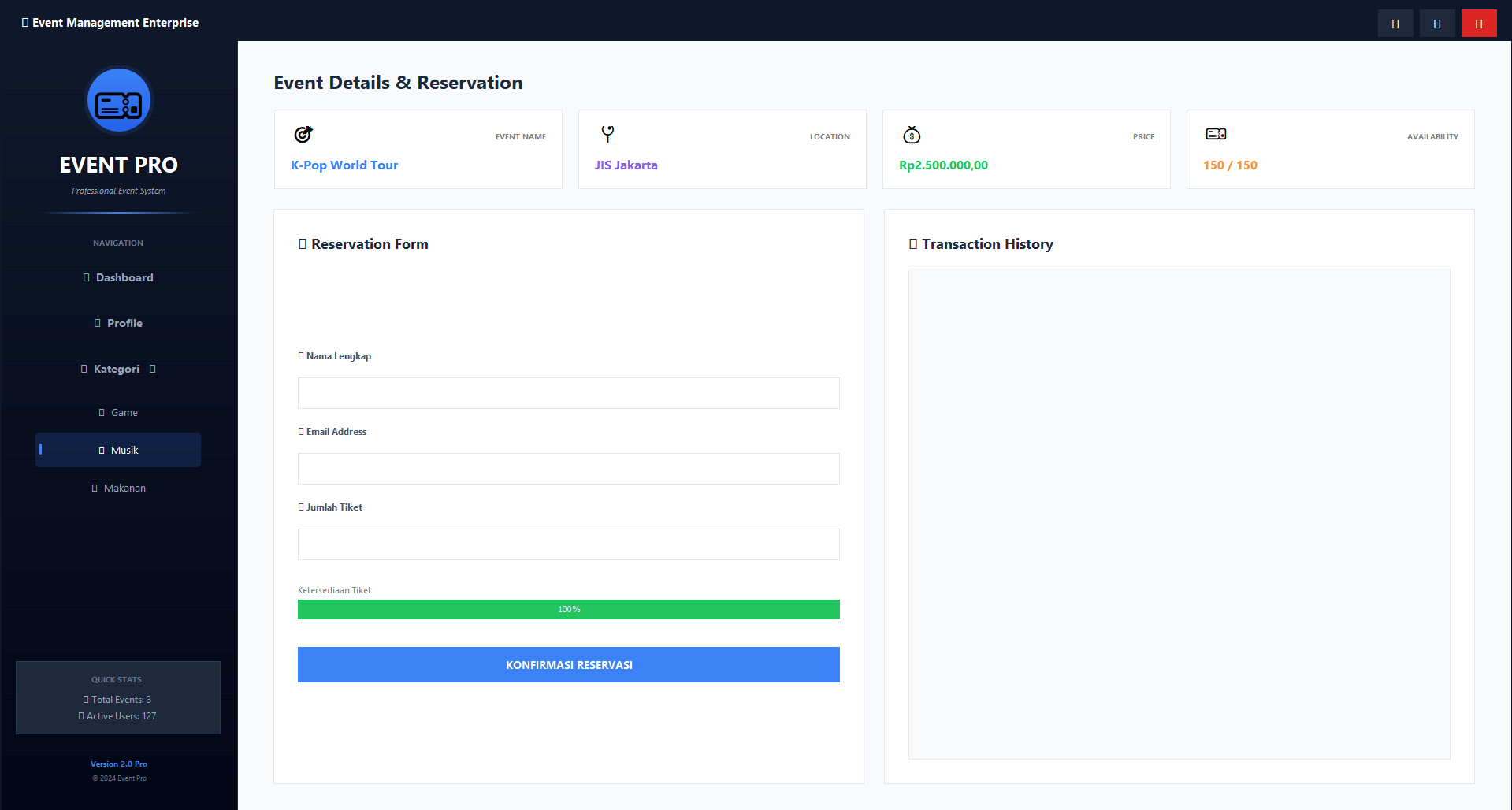
**3. Halaman Kategori ( Kategori Game )**

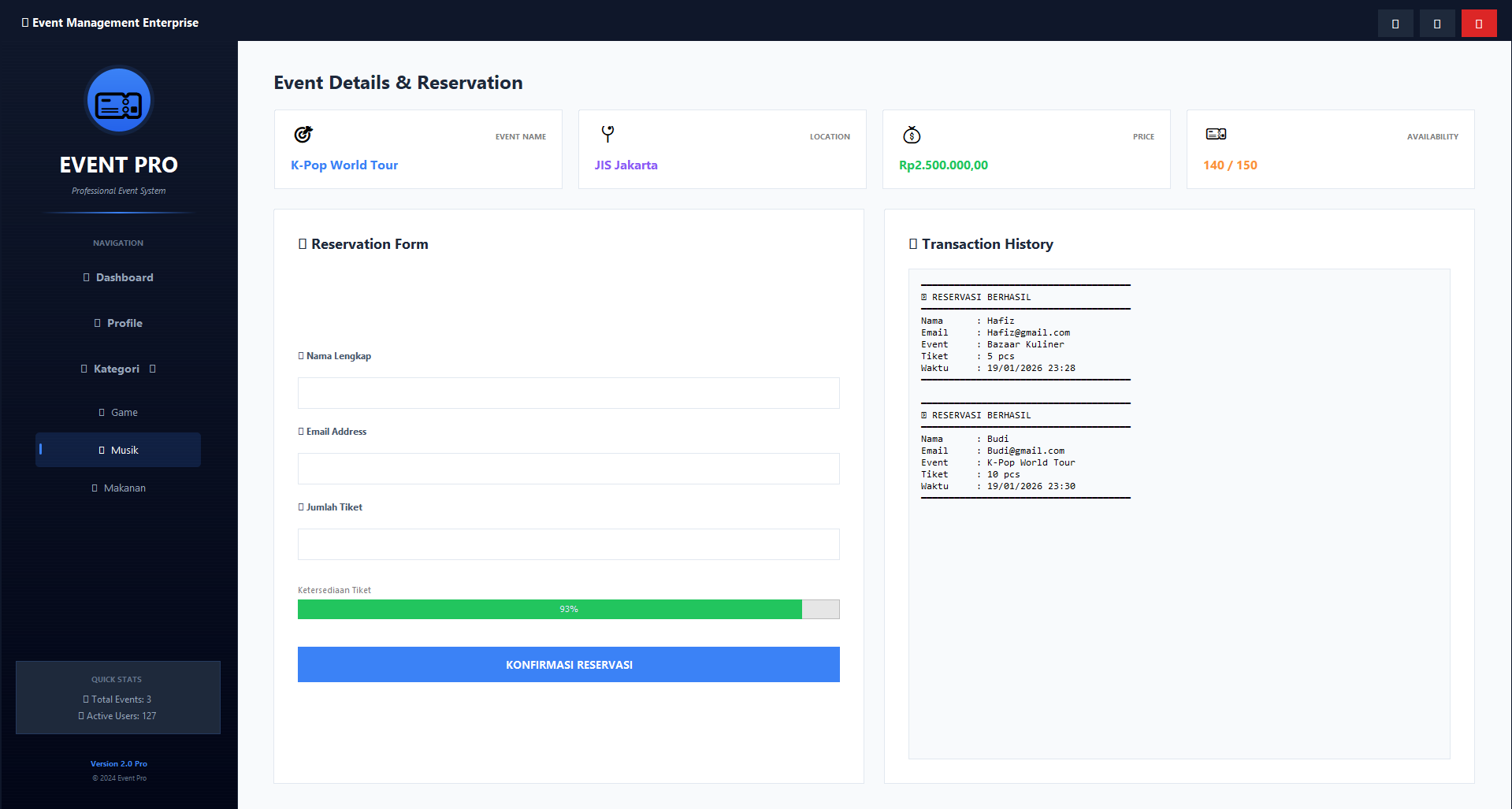
****

****

Gambar diatas itu Ketika sudah Menginput kan Nilai maka yang terjadi Pada Event Detail & Reservation Jumlah Tiketnya Berubah dan progress bar yang warna hijau juga berubah serta akan menampilkan Transaction History di sebelah kanan Reservation Form

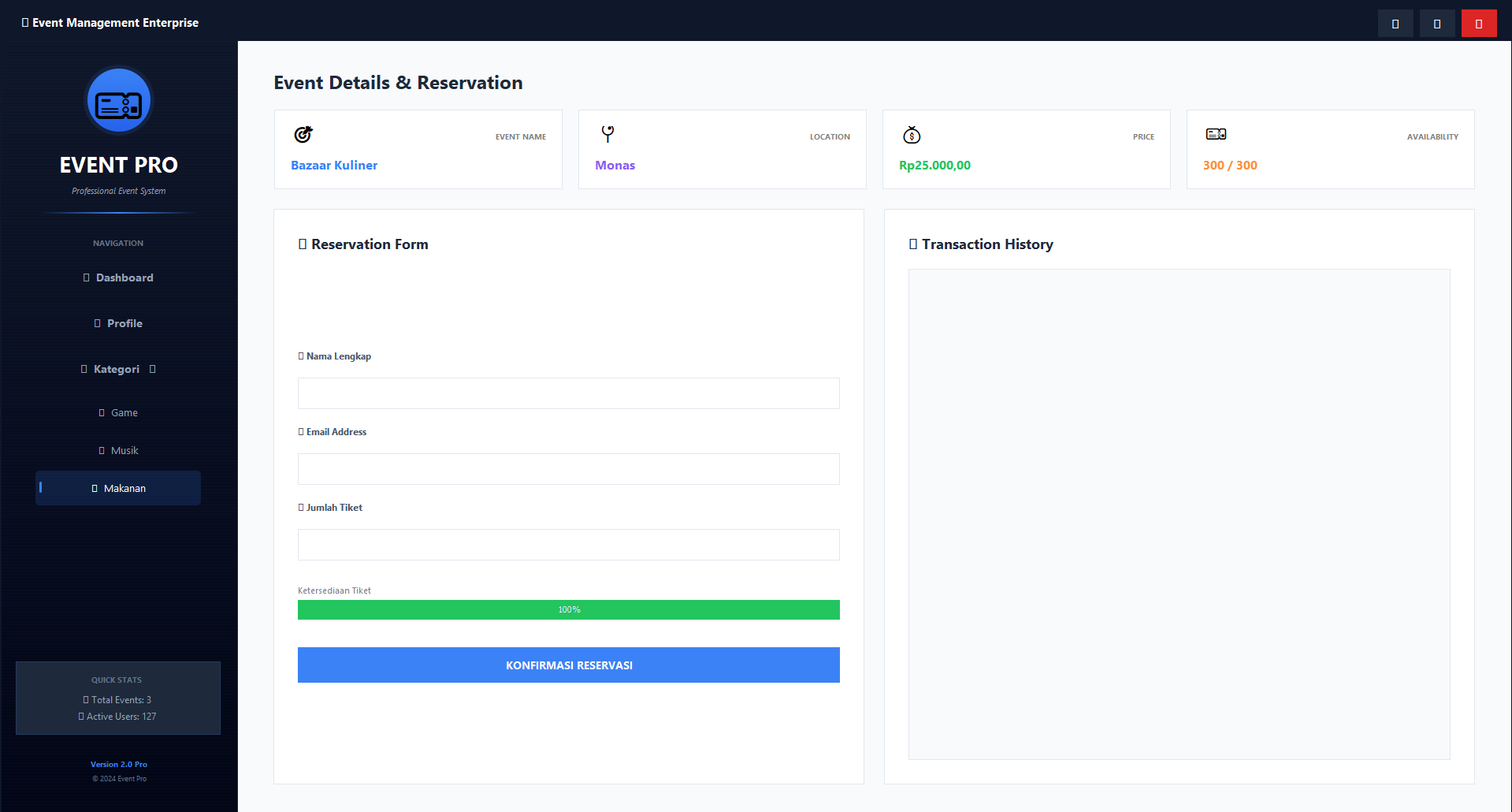
**4. Halaman Kategori ( Kategori Musik )**

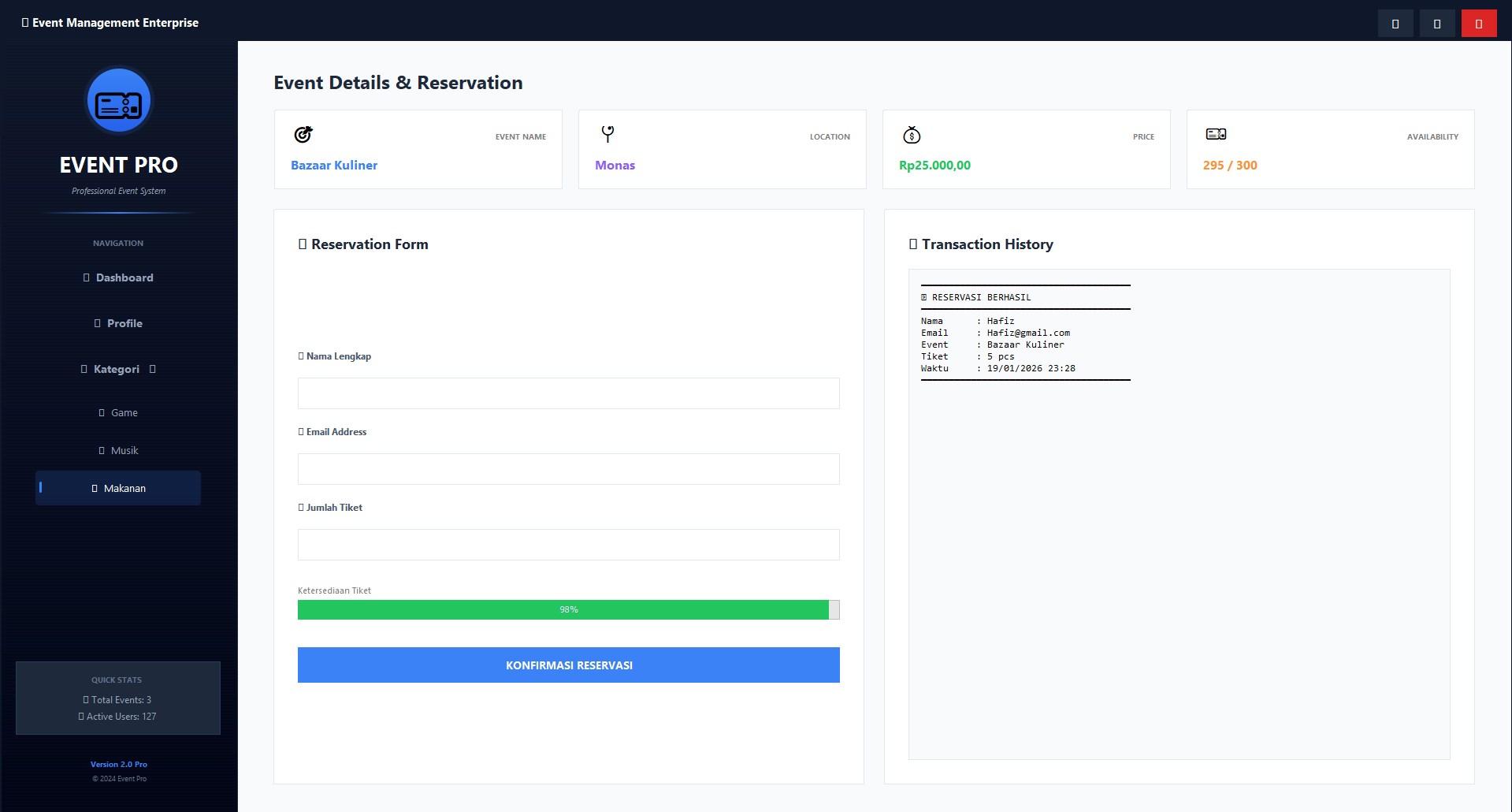
****

****

Gambar diatas itu Ketika sudah Menginput kan Nilai maka yang terjadi Pada Event Detail & Reservation Jumlah Tiketnya Berubah dan progress bar yang warna hijau juga berubah serta akan menampilkan Transaction History di sebelah kanan Reservation Form

**5. Halaman Kategori ( Kategori Makanan )**

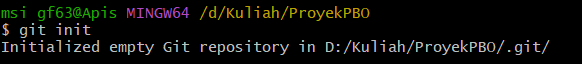
****

****

Gambar diatas itu Ketika sudah Menginput kan Nilai maka yang terjadi Pada Event Detail & Reservation Jumlah Tiketnya Berubah dan progress bar yang warna hijau juga berubah serta akan menampilkan Transaction History di sebelah kanan Reservation Form

**h.**

**1. Inisialisasi Repositori Lokal**

****

Langkah awal untuk memulai pelacakan perubahan pada folder proyek.

* Perintah: git init
* Penjelasan: Perintah ini membuat repositori Git baru yang tersembunyi (folder .git) di dalam direktori proyek Anda. Tanpa langkah ini, perintah Git lainnya tidak akan berfungsi.

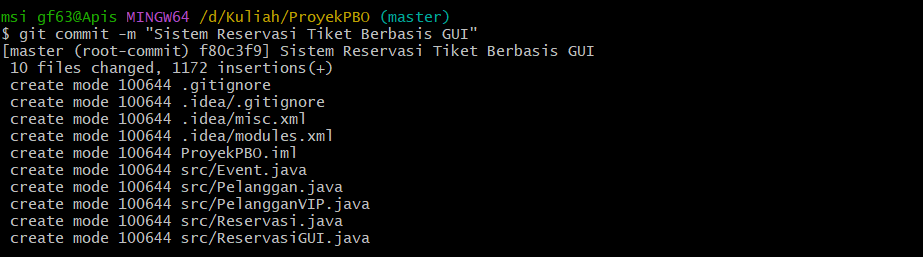
**2. Menyiapkan File (Staging)**

****

Langkahpertama adalah menandai file mana saja yang akan diunggah ke repositori.

* Perintah: git add .
* Penjelasan: Tanda titik (.) berarti Anda memasukkan semua perubahan dan file baru di folder proyek ke dalam area *staging* (antrean).
* Catatan: Muncul peringatan *warning: LF will be replaced by CRLF*, ini adalah hal normal di Windows yang mengatur cara Git membaca karakter ganti baris.

**3.** **Mengunci Perubahan (Commit)**

****

Setelah file siap, Anda perlu memberikan label atau catatan pada perubahan tersebut.

* Perintah: git commit -m "Sistem Reservasi Tiket Berbasis GUI"
* Penjelasan: Perintah ini menyimpan perubahan ke dalam database lokal Git dengan pesan tertentu.
* Hasil: Terlihat ada 10 file yang berubah, termasuk file inti seperti Event.java, Pelanggan.java, Reservasi.java, dan ReservasiGUI.java.

**4.** **Menghubungkan ke Repositori Online (Remote)**

****

Anda harus memberi tahu Git ke alamat GitHub mana file tersebut akan dikirim.

* Perintah: git remote add origin https://github.com/zulhanhafiz/pbo-Sistem-Reservasi-Tiket.git
* Penjelasan: Perintah ini mendaftarkan alamat URL GitHub Anda dengan nama alias origin.

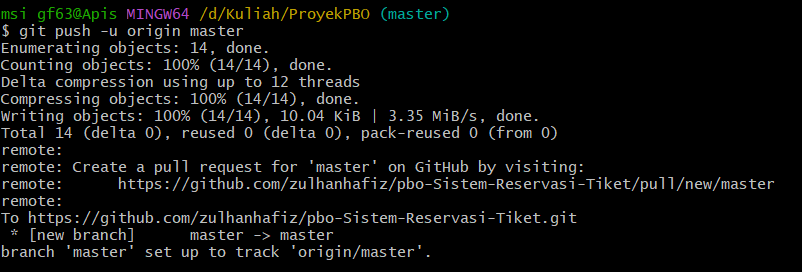
**5. Mengatur Autentikasi (Opsional)**

****

Jika repositori membutuhkan akses khusus, Anda bisa menyertakan *Personal Access Token* (PAT) agar tidak perlu mengisi password secara manual.

* Perintah: git remote set-url origin https://zulhanhafiz:ghp\_TOKEN\_ANDA@github.com/zulhanhafiz/pbo-Sistem-Reservasi-Tiket.git
* Penjelasan: Langkah ini memperbarui URL remote dengan menyertakan token keamanan agar proses pengiriman data berjalan lancar.

**6.** **Mengirim File ke GitHub (Push)**

****

**Langkah terakhir adalah mengirimkan semua kode dari komputer lokal ke server GitHub.**

* **Perintah: git push -u origin master**
* **Penjelasan: Perintah ini mengunggah seluruh data di branch master lokal ke repositori origin di GitHub.**
* **Hasil: Jika berhasil, akan muncul keterangan *branch 'master' set up to track 'origin/master'*, yang artinya file Anda sudah resmi tersimpan secara online di GitHub.**

**7. Mengubah Nama Branch Lokal**

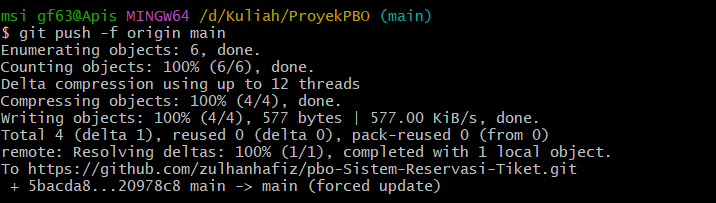
****

 **Perintah:** git branch -M main

 **Penjelasan:** Perintah ini memaksa (-M) Git untuk mengubah nama *branch* tempat Anda berada saat ini (yang sebelumnya bernama **master**) menjadi **main**.

 **Hasil:** Perhatikan perubahan pada teks di dalam kurung di baris berikutnya, dari (master) menjadi **(main)**. Sekarang, nama *branch* di komputer Anda sudah selaras dengan nama *branch* standar di GitHub.

**8.** Pengiriman Paksa ke GitHub (Force Push)

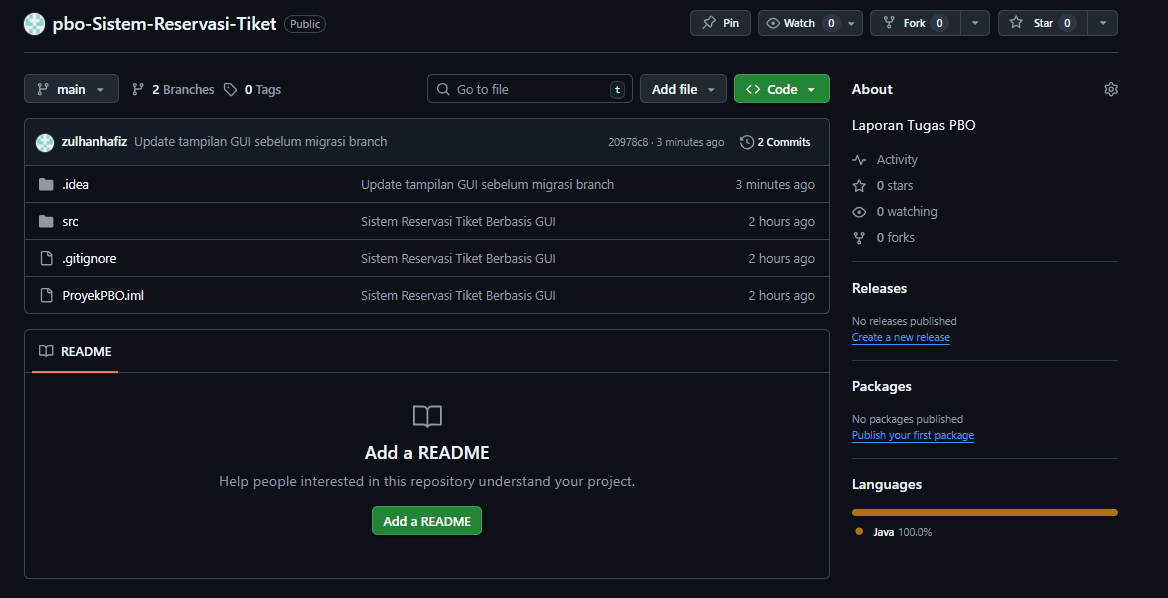
****

 **Perintah:** git push -f origin main

 **Penjelasan:** \* push: Mengirim data dari komputer lokal ke server.

* -f (**Force**): Ini adalah bagian paling penting. Anda memerintahkan Git untuk menimpa (*overwrite*) apapun yang ada di GitHub dengan apa yang ada di komputer Anda. Ini dilakukan karena riwayat *commit* di GitHub (yang berisi README.md) berbeda dengan riwayat di komputer Anda.
* origin main: Menentukan tujuan pengiriman ke repositori bernama origin pada *branch* main.

**9. Data Projek Telah terupload pada github**

****

**i. Analisis Pengerjaan Projek**

Berikut adalah pengembangan kalimat untuk laporan Anda agar terlihat lebih profesional, mendalam, dan teknis:

**Waktu Pelaksanaan**

- Proyek ini diselesaikan secara intensif dalam kurun waktu satu pekan menjelang pelaksanaan Ujian Akhir Semester (UAS). Proses pengerjaan meliputi beberapa tahapan mulai dari perancangan diagram *class*, implementasi logika bisnis berbasis Pemrograman Berorientasi Objek (PBO), hingga pengembangan antarmuka pengguna (GUI) yang memakan waktu paling signifikan guna memastikan aspek fungsionalitas dan estetika berjalan beriringan.

**- Ketercapaian Proyek**

Seluruh spesifikasi sistem dasar yang direncanakan telah berhasil diimplementasikan sepenuhnya tanpa kendala fungsional. Komponen inti seperti class Event untuk manajemen data acara, class Pelanggan (termasuk spesialisasi pada PelangganVIP) untuk pengelolaan data pengguna, serta class Reservasi sebagai jembatan logika transaksi, telah terintegrasi dengan baik. Sistem ini mampu menangani validasi stok tiket secara *real-time* dan mencatat riwayat transaksi secara akurat ke dalam komponen antarmuka.

**- Kendala yang Dihadapi**

Hambatan utama dalam pengembangan proyek ini terletak pada aspek tata letak (*layouting*) antarmuka (GUI). Penggunaan berbagai jenis *Layout Manager* di Java Swing—seperti GridBagLayout untuk presisi formulir dan BorderLayout untuk struktur utama—memerlukan ketelitian ekstra dalam pengaturan *insets*, *padding*, dan *alignment*. Menyeimbangkan elemen visual agar tetap rapi, proporsional, dan tetap responsif saat jendela aplikasi diubah ukurannya (*resize*) atau dalam mode layar penuh merupakan tantangan teknis terbesar selama proses pengodingan**.**

**- Tantangan Masa Depan**

Sebagai langkah pengembangan lebih lanjut, proyek ini memiliki potensi besar untuk ditingkatkan melalui integrasi dengan database relasional seperti MySQL. Saat ini, data masih tersimpan secara sementara dalam memori (*runtime*), sehingga integrasi database akan memungkinkan penyimpanan data secara permanen (*persistence*). Selain itu, fitur masa depan dapat mencakup sistem otentikasi login yang lebih aman, pembuatan laporan dalam format PDF, serta sinkronisasi data server agar aplikasi dapat digunakan oleh banyak admin secara bersamaan.