**Judul Buku (Saran):**

**Mastering Android App Development with LAMW: From Zero to Hero with Lazarus FPC** *atau dalam Bahasa Indonesia:* **Mahir Membuat Aplikasi Android dengan LAMW dan Lazarus FPC: Dari Nol hingga Profesional**

**Outline Rinci Buku**

**Prakata**

* Ucapan selamat datang kepada pembaca.
* Siapa penulis dan latar belakangnya.
* Mengapa buku ini ditulis dan apa yang membuatnya unik.
* Ucapan terima kasih.

**Daftar Isi**

**BAGIAN 1: PENDAHULUAN DAN PERSIAPAN**

**Bab 1: Selamat Datang di Dunia Pemrograman Mobile dengan Pascal**

* **1.1. Apa itu Lazarus dan Free Pascal (FPC)?**
  + Sejarah singkat dan filosofi FPC dan Lazarus.
  + Kekuatan Lazarus: Cross-Platform, RAD (Rapid Application Development), Kompilasi Native.
* **1.2. Mengenal LAMW (Lazarus Android Module Wizard)**
  + Apa itu LAMW dan bagaimana cara kerjanya?
  + Peran LAMW sebagai jembatan antara Object Pascal dan Android Native API (NDK).
* **1.3. Mengapa Memilih Kombinasi Lazarus + LAMW?**
  + Keuntungan: Satu basis kode, performa tinggi, kemudahan pengembangan visual.
  + Perbandingan singkat dengan platform lain (misal: Java/Kotlin di Android Studio, React Native, Flutter).
* **1.4. Untuk Siapa Buku Ini?**
* **1.5. Peta Jalan: Apa yang Akan Anda Pelajari?**

**Bab 2: Instalasi dan Konfigurasi Lingkungan Pengembangan**

* **2.1. Kebutuhan Perangkat Keras dan Perangkat Lunak**
* **2.2. Mengunduh dan Menginstal Lazarus IDE**
  + Memilih versi yang stabil dan kompatibel.
* **2.3. Instalasi Java Development Kit (JDK)**
  + Menjelaskan mengapa JDK diperlukan untuk build Android.
* **2.4. Instalasi Android SDK dan NDK**
  + Menggunakan Android Studio SDK Manager atau Command-line tools.
  + Penjelasan perbedaan SDK dan NDK.
* **2.5. Mengunduh dan Menginstal Pustaka LAMW**
  + Instalasi melalui Online Package Manager atau manual dari GitHub.
* **2.6. Konfigurasi Path Penting di Lazarus**
  + Langkah-langkah detail mengatur path JDK, Android SDK, dan Android NDK di dalam IDE Lazarus.
* **2.7. Verifikasi Instalasi**
  + Cara memastikan semua komponen terhubung dengan benar sebelum membuat proyek pertama.

**BAGIAN 2: DASAR-DASAR PENGEMBANGAN DENGAN LAMW**

**Bab 3: Proyek Pertama Anda: "Halo Dunia!"**

* **3.1. Membuat Proyek LAMW Baru**
  + Mengenal wizard pembuatan proyek LAMW.
  + Struktur direktori dan file yang dihasilkan.
* **3.2. Tur Singkat Antarmuka Lazarus untuk LAMW**
  + Form Designer (Desainer Formulir)
  + Object Inspector (Inspektur Objek)
  + Source Editor (Editor Kode)
  + Component Palette (Palet Komponen)
* **3.3. Menambahkan Komponen Visual**
  + Menambahkan jTextView dan jButton.
  + Mengubah properti seperti Text, ID, Layout Params.
* **3.4. Menulis Kode Pascal Pertama Anda**
  + Membuat event handler OnClick untuk tombol.
  + Mengubah teks pada jTextView secara dinamis.
* **3.5. Kompilasi dan Menjalankan Aplikasi**
  + Proses build: Dari Pascal ke APK.
  + Menjalankan di Android Emulator (AVD).
  + Menjalankan di Perangkat Android Fisik (mengaktifkan mode developer dan USB debugging).

**Bab 4: Desain Antarmuka Pengguna (UI) Fundamental**

* **4.1. Konsep Layout di Android**
  + jLinearLayout: Pengaturan Vertikal dan Horizontal.
  + jRelativeLayout: Pengaturan relatif terhadap komponen lain.
  + Pentingnya LayoutParams (misal: MATCH\_PARENT, WRAP\_CONTENT).
* **4.2. Komponen Input Pengguna**
  + jEditText: Input teks satu baris dan multi-baris.
  + jCheckBox dan jRadioButton.
  + jToggleButton.
* **4.3. Komponen Tampilan (Display)**
  + jImageView: Menampilkan gambar dari resource dan file.
  + jProgressBar: Menampilkan progres.
* **4.4. Tombol dan Aksi**
  + jButton dan jImageButton.
* **4.5. Bekerja dengan Event**
  + Memahami event-event umum: OnClick, OnChanged, OnFocus, dll.

**Bab 5: Navigasi dan Manajemen Formulir (Activity)**

* **5.1. Konsep Activity di Android vs Form di Lazarus**
* **5.2. Membuat dan Menambah Form Baru**
* **5.3. Berpindah Antar Form**
  + Menggunakan Show dan ShowModal.
  + Mengirim data sederhana antar form menggunakan Intent.
* **5.4. Siklus Hidup Activity (Activity Lifecycle)**
  + Memahami OnCreate, OnResume, OnPause, OnDestroy.
  + Kapan dan mengapa menggunakannya (misal: menyimpan state).
* **5.5. Menutup Form dan Kembali ke Form Sebelumnya**

**BAGIAN 3: TOPIK MENENGAH**

**Bab 6: Bekerja dengan Data Lokal**

* **6.1. Menyimpan Preferensi Sederhana dengan jSharedPreferences**
  + Contoh kasus: Menyimpan pengaturan atau status login "Ingat Saya".
* **6.2. Membaca dan Menulis File**
  + Penyimpanan Internal: Aman dan privat.
  + Penyimpanan Eksternal: Memerlukan izin (permission).
  + Manajemen Izin (Permissions) di Android.
* **6.3. Pengenalan Database SQLite**
  + Kapan menggunakan SQLite?
* **6.4. Operasi Database (CRUD)**
  + Membuat Database dan Tabel.
  + Menambah Data (INSERT).
  + Membaca Data (SELECT).
  + Memperbarui Data (UPDATE).
  + Menghapus Data (DELETE).
* **6.5. Menampilkan Data dari SQLite ke jListView**

**Bab 7: Komponen UI Tingkat Lanjut**

* **7.1. Menampilkan Daftar dengan jListView**
  + Populasi data dari array statis.
  + Populasi data dari database.
  + Membuat layout item kustom.
* **7.2. Pilihan Drop-down dengan jSpinner**
* **7.3. Tampilan Web dengan jWebView**
  + Memuat URL dari internet.
  + Memuat konten HTML lokal.
* **7.4. Navigasi Tab dengan jTabLayout dan jViewPager**
* **7.5. Dialog dan Notifikasi**
  + Membuat dialog peringatan (ShowMessage, MessageDlg).
  + Membuat notifikasi di status bar.

**Bab 8: Mengakses Perangkat Keras dan Sensor**

* **8.1. Menggunakan Kamera dengan jCamera**
  + Mengambil foto.
  + Izin akses kamera.
* **8.2. Mendapatkan Lokasi GPS dengan jLocationSensor**
  + Mendapatkan koordinat Latitude dan Longitude.
  + Izin akses lokasi (Fine dan Coarse).
* **8.3. Sensor Gerak: jAccelerometerSensor**
  + Mendeteksi guncangan atau orientasi perangkat.
* **8.4. Fitur Lainnya**
  + Getaran (Vibrate).
  + Mengakses informasi perangkat (Device Info).

**BAGIAN 4: TOPIK MAHIR DAN STUDI KASUS**

**Bab 9: Komunikasi Jaringan dan API**

* **9.1. Pengenalan HTTP dan RESTful API**
* **9.2. Melakukan Permintaan Jaringan dengan jHttpClient**
  + Permintaan GET untuk mengambil data.
  + Permintaan POST untuk mengirim data.
* **9.3. Bekerja dengan JSON**
  + Mengapa JSON?
  + Parsing data JSON menggunakan pustaka fpjson.
  + Memetakan data JSON ke objek Pascal.
* **9.4. Menampilkan Data dari API ke dalam Aplikasi**
  + Contoh: Menampilkan daftar berita atau produk dari server.

**Bab 10: Teknik Pemrograman Lanjutan**

* **10.1. Multithreading: Menjaga UI Tetap Responsif**
  + Mengapa thread utama tidak boleh diblok.
  + Menjalankan tugas berat (jaringan, database) di thread terpisah.
  + Sinkronisasi kembali ke thread utama untuk update UI.
* **10.2. Layanan Latar Belakang (Background Services)**
  + Menjalankan tugas bahkan saat aplikasi tidak di depan.
  + Contoh: Sinkronisasi data periodik.
* **10.3. Menggambar Kustom dengan jCanvas**
  + Menggambar garis, bentuk, dan teks.
  + Membuat komponen visual kustom sederhana.
* **10.4. Interoperabilitas dengan Java (JNI)**
  + Dasar-dasar Java Native Interface.
  + Cara memanggil metode Java dari kode Pascal.
  + Cara menggunakan library .jar atau .aar pihak ketiga (tingkat lanjut).

**Bab 11: Studi Kasus: Membangun Aplikasi Lengkap**

* **11.1. Proyek 1: Aplikasi "Daftar Tugas" (Todo List)**
  + Fitur: Menambah, mengedit, menghapus, dan menandai tugas selesai.
  + Teknologi: UI, Multi-form, Database SQLite.
* **11.2. Proyek 2: Aplikasi "Katalog Produk Online"**
  + Fitur: Menampilkan daftar produk dari API, detail produk, pencarian.
  + Teknologi: HTTP Client, Parsing JSON, jListView dengan item kustom, jImageView untuk memuat gambar dari URL.

**Bab 12: Finalisasi dan Publikasi Aplikasi**

* **12.1. Mempersiapkan Aset Aplikasi**
  + Membuat ikon aplikasi dengan berbagai ukuran.
  + Menyiapkan screenshot untuk Play Store.
* **12.2. Proses Penandatanganan Aplikasi (Signing)**
  + Membuat Keystore.
  + Mengkonfigurasi Lazarus untuk build rilis (signed APK/AAB).
* **12.3. Mengunggah ke Google Play Store**
  + Mendaftar akun Google Play Developer.
  + Mengisi detail aplikasi (deskripsi, kebijakan privasi, rating konten).
  + Mengunggah file AAB (Android App Bundle).
  + Proses peninjauan oleh Google.
* **12.4. Memperbarui Aplikasi Anda**

**Lampiran**

* **A. Troubleshooting (Penyelesaian Masalah)**
  + Daftar error kompilasi yang sering terjadi dan solusinya.
  + Masalah umum saat runtime (misal: "Application Not Responding").
* **B. Referensi Cepat Komponen LAMW**
  + Tabel komponen, properti kunci, dan event utamanya.
* **C. Sumber Daya Tambahan**
  + Link ke forum Lazarus, wiki, repositori GitHub LAMW, dan komunitas relevan lainnya.

**Glosarium**

* Daftar istilah teknis dan penjelasannya.

**Indeks**

Semoga outline ini memberikan kerangka kerja yang solid dan terstruktur untuk buku Anda. Sukses dengan penulisannya!

Saya minta outline yang diperbarui

Tampilkan alur berpikir

Tentu saja. Berikut adalah draf outline yang diperbarui dan disempurnakan untuk tahun 2025, dengan mempertimbangkan praktik pengembangan Android modern, alur belajar yang lebih logis, dan penekanan pada topik-topik yang paling relevan saat ini.

**Judul Buku (Saran Edisi 2025):**

**Aplikasi Android Profesional dengan LAMW & Lazarus FPC: Edisi Terbaru 2025** *atau* **The LAMW Handbook: Panduan Lengkap Membangun Aplikasi Android Native dengan Pascal**

**Outline Buku (Edisi Diperbarui 2025)**

**Prakata Edisi 2025**

* Sambutan dan pengenalan.
* Apa yang baru di edisi ini? (Penekanan pada Android API terbaru, Scoped Storage, praktik modern).
* Mengapa LAMW masih menjadi pilihan yang relevan dan powerful di tahun 2025.

**Daftar Isi**

**BAGIAN 1: FONDASI DAN PERSIAPAN**

**Bab 1: Pengenalan Ekosistem Lazarus dan Android**

* 1.1. Lazarus & FPC: Kekuatan Kompilasi Native dan RAD.
* 1.2. LAMW: Jembatan Cerdas Antara Pascal dan Android NDK.
* 1.3. Arsitektur Aplikasi LAMW: Bagaimana Semuanya Bekerja di Balik Layar.
* 1.4. Keunggulan dan Keterbatasan LAMW di Lanskap Mobile Modern.
* 1.5. Peta Belajar: Dari Konsep Dasar hingga Aplikasi Siap Rilis.

**Bab 2: Setup Lingkungan Pengembangan Modern (Update 2025)**

* 2.1. Prasyarat Sistem.
* 2.2. Instalasi Lazarus IDE versi terkini.
* 2.3. Instalasi dan Manajemen Android SDK & NDK via Android Studio SDK Manager.
* 2.4. Instalasi Java Development Kit (JDK) yang Direkomendasikan.
* 2.5. Instalasi Pustaka LAMW dari Online Package Manager.
* 2.6. Konfigurasi Kritis: Menghubungkan Lazarus dengan Tools Android.
* 2.7. **Sub-bab Baru:** Troubleshooting Instalasi: Solusi untuk Masalah Umum (Path Not Found, Build Gagal, dll).
* 2.8. Menyiapkan Emulator (AVD) dan Perangkat Fisik untuk Debugging.

**BAGIAN 2: INTI PENGEMBANGAN APLIKASI**

**Bab 3: Proyek Pemanasan: Membangun Aplikasi Kalkulator Sederhana**

* 3.1. Bukan Sekadar "Halo Dunia": Membuat Proyek yang Fungsional.
* 3.2. Merancang Layout Kalkulator dengan jLinearLayout dan jGridLayout.
* 3.3. Menggunakan jTextView sebagai Layar dan jButton untuk Tombol.
* 3.4. Logika Dasar Kalkulator: Menangani Input Angka dan Operasi.
* 3.5. Kompilasi Pertama dan Merasakan Kemenangan Awal.
* 3.6. **Pelajaran Penting:** Apa yang Kita Pelajari dari Proyek Ini.

**Bab 4: Anatomi Aplikasi LAMW dan Desain Antarmuka (UI)**

* 4.1. Membedah Struktur Proyek: AppModule, MainForm, Resources.
* 4.2. Palet Komponen LAMW:
  + **Kontainer Layout:** jLinearLayout, jRelativeLayout, jScrollView.
  + **Komponen Input:** jEditText (dengan tipe input), jCheckBox, jSpinner.
  + **Komponen Aksi & Tampilan:** jButton, jImageView, jTextView.
* 4.3. **Sub-bab Baru:** Prinsip Desain Layout Modern:
  + Penggunaan dp (Density-independent Pixels) dan sp (Scale-independent Pixels).
  + Membuat Layout Responsif dengan Properti layout\_weight.
  + Mendukung Mode Terang & Gelap (Light/Dark Mode).
* 4.4. Bekerja dengan Event Listener (OnClick, OnTextChanged, dll).

**Bab 5: Alur Kerja Aplikasi dan Navigasi Antar Layar (Activity)**

* 5.1. Konsep Activity dan Intent di Dunia Android.
* 5.2. Mengelola Multiple Forms: Membuat, Menampilkan, dan Menutup Form.
* 5.3. Mengirim Data Antar Form (Activity) Menggunakan Intent.
  + Mengirim Tipe Data Primitif (String, Integer).
  + **Baru:** Mengirim Objek/Record Data yang Kompleks.
* 5.4. Siklus Hidup Activity (OnCreate, OnResume, OnPause, OnDestroy): Kapan dan Mengapa Penting.
* 5.5. Menggunakan Dialog (ShowMessage, MessageDlg) untuk Interaksi Singkat.

**BAGIAN 3: MENGELOLA DATA DAN JARINGAN**

**Bab 6: Manajemen Data Lokal Sesuai Aturan Modern**

* 6.1. Penyimpanan Preferensi dengan jSharedPreferences.
* 6.2. **Revisi Total:** Bekerja dengan File di Era **Scoped Storage**.
  + Penyimpanan Spesifik Aplikasi (Internal & Eksternal): Tidak perlu izin khusus.
  + Penyimpanan Bersama (Shared Storage) via Storage Access Framework (SAF): Membiarkan pengguna memilih file/folder.
* 6.3. **Bab Krusial:** Manajemen Izin Runtime (Runtime Permissions).
  + Memeriksa dan Meminta Izin (Kamera, Lokasi, Penyimpanan) saat aplikasi berjalan.
* 6.4. Database Lokal dengan SQLite:
  + Desain Skema Database (Tabel, Kolom, Tipe Data).
  + Operasi CRUD (Create, Read, Update, Delete) yang Efisien.
  + Menggunakan Transaksi untuk Integritas Data.

**Bab 7: Komunikasi Jaringan dan Mengonsumsi API**

* 7.1. Dasar-dasar Komunikasi HTTP (GET, POST) dan REST API.
* 7.2. Melakukan Request Jaringan Asinkron dengan jHttpClient.
* 7.3. Parsing Data JSON dengan Pustaka fpjson atau Pascal-JSON.
* 7.4. Menampilkan Data dari API ke Komponen UI (misal: jListView).
* 7.5. **Baru:** Memuat Gambar dari URL ke jImageView secara Efisien.
* 7.6. Penanganan Error Jaringan (Tidak Ada Koneksi, Server Error, dll).

**BAGIAN 4: FITUR LANJUTAN DAN PROFESIONAL**

**Bab 8: Komponen Canggih, Multimedia, dan Notifikasi**

* 8.1. Menguasai jListView dengan Adapter Kustom untuk Tampilan Kompleks.
* 8.2. Menggunakan jWebView untuk Mengintegrasikan Konten Web.
* 8.3. Mengakses Kamera dan Galeri dengan jCamera dan Intent.
* 8.4. Mengambil Data Lokasi GPS dengan jLocationSensor.
* 8.5. **Update:** Mengirim Notifikasi Push yang Kompatibel dengan Android Modern (menggunakan Notification Channels).

**Bab 9: Threading, Tugas Latar Belakang, dan Kustomisasi**

* 9.1. Menjaga UI Tetap Responsif dengan Threading.
* 9.2. **Update:** Strategi Tugas Latar Belakang Modern (Background Tasks).
  + Kapan Menggunakan Thread vs. Android Services.
  + Pengenalan konsep yang mirip dengan WorkManager Android untuk tugas yang terjadwal dan dapat diandalkan.
* 9.3. **Baru:** Teknik Debugging Tingkat Lanjut: Menggunakan Log.d dan Menganalisis Logcat.
* 9.4. **Baru:** Membuat Komponen Visual Kustom Anda Sendiri.

**Bab 10: Interoperabilitas dengan Ekosistem Java/Kotlin**

* 10.1. Pengantar Java Native Interface (JNI) dalam Konteks LAMW.
* 10.2. Memanggil Metode Java dari Kode Pascal.
* 10.3. Studi Kasus: Mengintegrasikan Library Android Pihak Ketiga (file .jar atau .aar), misalnya library grafik (chart).

**BAGIAN 5: PRODUKSI DAN PUBLIKASI**

**Bab 11: Studi Kasus Lengkap: Aplikasi "Catatan Berita"**

* 11.1. Perencanaan: Merancang Fitur dan Arsitektur Aplikasi.
* 11.2. Implementasi:
  + Mengambil berita dari REST API publik (misal: NewsAPI).
  + Menampilkan daftar berita dengan jListView kustom.
  + Halaman detail berita menggunakan jWebView.
  + Fitur "Bookmark" untuk menyimpan berita favorit ke database SQLite.
  + Menerapkan Runtime Permissions dan Scoped Storage.
* 11.3. Refactoring dan Pengujian Akhir.

**Bab 12: Merilis Aplikasi Anda ke Google Play Store**

* 12.1. Mempersiapkan Aset Rilis: Ikon, Screenshot, Deskripsi.
* 12.2. **Update Penting:** Membangun Rilis dengan Format **Android App Bundle (AAB)**.
* 12.3. Membuat Kunci Penandatanganan (Signing Keystore) dan Mengamankannya.
* 12.4. **Update Penting:** Panduan Mengisi **Formulir Keamanan Data** di Google Play Console.
* 12.5. Proses Upload, Pengujian Internal, dan Peluncuran Bertahap.
* 12.6. Strategi Pemeliharaan dan Merilis Update Aplikasi.
* **Judul Bab 13: Integrasi Peta Tingkat Lanjut dengan Google Maps**
* **A. Prompt untuk Pengantar Bab 13**
* **B. Prompt untuk Setiap Subbab**
* **13.1: Persiapan Awal: Kunci API dan Konfigurasi Proyek**
* **2. Prompt untuk Subbab 13.2: Menampilkan Peta Pertama Anda ("Hello, Map!")**
* **3. Prompt untuk Subbab 13.3: Mengontrol Kamera Peta**
* **4. Prompt untuk Subbab 13.4: Bekerja dengan Penanda (Markers)**
* **5. Prompt untuk Subbab 13.5: Menggambar di Atas Peta (Polylines dan Polygons)**
* **6. Prompt untuk Subbab 13.6: Menampilkan Lokasi Pengguna**
* **Judul Bab 14: Membangun Aplikasi Cerdas dengan Gemini AI API**
* **1. Prompt untuk Subbab 14.1: Persiapan dan Konfigurasi Gemini API**
* **2. Prompt untuk Subbab 14.2: Interaksi Pertama: Permintaan Teks-ke-Teks**
* **3. Prompt untuk Subbab 14.3: Kemampuan Multimodal: Mengirim Teks dan Gambar**
* **4. Prompt untuk Subbab 14.4: Membangun Chatbot Sederhana**
* **5. Prompt untuk Subbab 14.5: Etika dan Praktik Terbaik**
* **Judul Bab 15: Menjelajahi Dunia API Publik: Studi Kasus Lanjutan**
* **1. Prompt untuk Subbab 15.1: Studi Kasus 1: Aplikasi Cuaca dengan OpenWeatherMap API**
* **2. Prompt untuk Subbab 15.2: Studi Kasus 2: Konverter Mata Uang dengan ExchangeRate-API**
* **3. Prompt untuk Subbab 15.3: Studi Kasus 3: Peramban Film dengan TMDB API**
* **Judul Bab 16: Menguasai Multimedia: Audio dan Video di Aplikasi Anda**
* **1. Prompt untuk Subbab 16.1: Efek Suara Singkat dengan jSoundPool**
* **2. Prompt untuk Subbab 16.2: Pemutar Audio untuk Musik atau Podcast (jMediaPlayer)**
* **3. Prompt untuk Subbab 16.3: Video Sederhana dengan jVideoView**
* **4. Prompt untuk Subbab 16.4: Video Profesional dengan ExoPlayer (via JNI)**
* **5. Prompt untuk Subbab 16.5: Arsitektur Audio Latar Belakang (Background Audio)**
* **Judul Bab 17: Kontrol Kamera Tingkat Lanjut dengan CameraX**
* **1. Prompt untuk Subbab 17.1: Persiapan Proyek dan Dependensi CameraX**
* **2. Prompt untuk Subbab 17.2: Memahami Konsep Inti CameraX**
* **3. Prompt untuk Subbab 17.3: Implementasi Pratinjau Kamera (Camera Preview)**
* **4. Prompt untuk Subbab 17.4: Membangun UI Kustom dan Menangkap Gambar**
* **5. Prompt untuk Subbab 17.5: Fitur Tambahan: Beralih Kamera dan Kontrol Flash**
* **Judul Bab 18: Integrasi Hibrida: Menjembatani Pascal dan JavaScript dengan WebView**
* **1. Prompt untuk Subbab 18.1: Setup Lanjutan: WebViewClient dan WebChromeClient**
* **2. Prompt untuk Subbab 18.2: Memanggil Pascal dari JavaScript (JavaScript Interface)**
* **3. Prompt untuk Subbab 18.3: Memanggil JavaScript dari Pascal**
* **4. Prompt untuk Subbab 18.4: Mengontrol Navigasi dengan WebViewClient**
* **5. Prompt untuk Subbab 18.5: Meningkatkan Pengalaman UI dengan WebChromeClient**

**Penutup**

* Rangkuman perjalanan belajar.
* Ke mana selanjutnya? Menjaga skill tetap relevan.
* Motivasi dan ucapan terima kasih kepada pembaca.

**Lampiran**

* **A.** Referensi Cepat Properti Layout Android.
* **B.** Daftar Intent Android yang Umum Digunakan.
* **C.** Sumber Daya Online: Forum, Komunitas, dan Repositori Penting.

**Glosarium & Indeks**

**Bab 1: Pengenalan Ekosistem Lazarus dan Android**

Selamat datang di dunia pengembangan aplikasi mobile! Ini adalah awal dari sebuah perjalanan yang sangat menarik, di mana Anda akan belajar mengubah ide-ide kreatif menjadi aplikasi fungsional yang berpotensi berjalan di miliaran perangkat Android di seluruh dunia. Jika Anda pernah membayangkan aplikasi buatan Anda sendiri digunakan oleh teman, keluarga, atau bahkan publik secara luas, maka Anda telah berada di tempat yang tepat.

Untuk menaklukkan tantangan ini, kita tidak akan sendirian. Kita akan dibekali dengan tiga serangkai teknologi yang tangguh dan unik: kekuatan lingkungan pengembangan visual **Lazarus IDE**, kecepatan kompilasi native dari **Free Pascal (FPC)**, dan keajaiban dari pustaka **LAMW (Lazarus Android Module Wizard)** yang akan menjadi jembatan kita ke dunia Android. Kombinasi ini menawarkan sebuah pendekatan yang berbeda, menggabungkan kecepatan pengembangan visual dengan performa tinggi dari kode native.

Sebelum kita langsung terjun ke dalam baris-baris kode dan merancang antarmuka, bab pendahuluan ini memiliki satu tujuan penting: membangun fondasi pemahaman yang kokoh. Kita akan menjawab pertanyaan mendasar "Mengapa memilih jalur ini?" dan mengenal lebih dekat setiap komponen yang akan membawa ide-ide Anda menjadi aplikasi Android yang nyata. Mari kita mulai petualangan ini.

**1.1. Lazarus & FPC: Kekuatan Kompilasi Native dan RAD**

Mari kita mulai dengan mengenal dua pilar utama yang akan menjadi fondasi kita: Lazarus dan Free Pascal (FPC). Keduanya adalah satu kesatuan yang tak terpisahkan, bekerja sama untuk menciptakan lingkungan pengembangan yang luar biasa. Jika kita ibaratkan sebuah pesawat jet, maka **Free Pascal (FPC) adalah mesin jet-nya**—sebuah kompilator (compiler) Object Pascal yang sangat kuat, matang, dan andal. Sementara itu, **Lazarus adalah kokpit canggihnya**—sebuah Lingkungan Pengembangan Terpadu atau *Integrated Development Environment* (IDE) yang menyediakan semua alat bantu visual dan editor untuk "mengemudikan" FPC dengan mudah.

Kombinasi keduanya memberikan dua kekuatan utama yang menjadi alasan mengapa pendekatan ini begitu menarik.

**Kekuatan #1: Pengembangan Aplikasi Cepat (RAD)**

Lazarus menganut filosofi RAD, singkatan dari *Rapid Application Development*. Artinya, Anda dapat merancang dan membangun aplikasi dengan kecepatan yang sangat tinggi, terutama untuk bagian antarmuka pengguna (UI). Lazarus menyediakan serangkaian alat visual yang luar biasa untuk ini:

* **Form Designer:** Sebuah kanvas visual tempat Anda dapat mendesain tampilan setiap layar aplikasi dengan cara *drag-and-drop*.
* **Component Palette:** "Kotak perkakas" Anda yang berisi berbagai macam komponen siap pakai, mulai dari tombol, kotak teks, hingga komponen yang lebih kompleks.
* **Object Inspector:** Panel properti di mana Anda dapat dengan mudah mengubah setiap detail dari komponen yang Anda pilih—mulai dari warna, ukuran, teks, hingga perilaku (event) yang akan dijalankannya.

Pendekatan visual ini mempercepat proses desain antarmuka secara drastis, memungkinkan Anda untuk lebih fokus pada logika bisnis di balik aplikasi Anda.

**Kekuatan #2: Performa Kompilasi Native**

Saat Anda menekan tombol 'Compile' di Lazarus, FPC tidak menghasilkan kode perantara yang perlu diterjemahkan lagi saat aplikasi berjalan. Sebaliknya, FPC menerjemahkan kode Object Pascal Anda secara langsung menjadi **instruksi mesin**—bahasa level terendah yang dapat dieksekusi langsung oleh prosesor (CPU) perangkat Android. Inilah yang disebut dengan **kompilasi native**.

Hasilnya adalah aplikasi yang berjalan dengan kecepatan maksimal dan efisiensi tinggi, karena ia "berbicara" dalam bahasa yang paling dimengerti oleh perangkat keras. Ini berbeda dengan pendekatan berbasis *Virtual Machine* (VM) seperti pada Java/Kotlin di mana kode dijalankan di dalam sebuah lapisan abstraksi (ART), atau pendekatan interpretasi pada beberapa framework JavaScript yang menerjemahkan kode saat runtime. Performa native adalah salah satu keunggulan terbesar yang ditawarkan FPC.

Sebagai catatan tambahan yang menarik, keahlian yang Anda bangun di Lazarus tidak terbatas hanya pada Android. FPC dan Lazarus adalah juara dalam pengembangan lintas platform (*cross-platform*), artinya dengan basis kode yang sama, Anda berpotensi menargetkan Windows, macOS, dan Linux. Namun, dalam buku ini, kita akan memfokuskan semua kekuatan ini untuk satu tujuan: menaklukkan platform Android.

**1.2. LAMW: Jembatan Cerdas Antara Pascal dan Android NDK**

Kini kita sampai pada komponen ketiga dari trio teknologi kita, sang bintang utama dalam buku ini: LAMW. LAMW adalah singkatan dari **Lazarus Android Module Wizard**. Jika Lazarus adalah kokpit dan FPC adalah mesinnya, maka LAMW adalah **sistem navigasi dan penerjemah cerdas** yang memungkinkan pesawat kita mendarat dengan sempurna di landasan Android.

Peran utama LAMW adalah sebagai **jembatan penghubung**. Di satu sisi jembatan, kita memiliki dunia Object Pascal yang elegan dan terstruktur di dalam Lazarus. Di sisi lain, kita memiliki dunia Android yang kompleks dengan antarmuka penggunanya yang kaya dan API level rendahnya yang diakses melalui **Android Native Development Kit (NDK)**. Tanpa adanya jembatan, kedua dunia ini tidak akan bisa saling memahami. LAMW-lah yang memungkinkan kedua dunia ini berkomunikasi dengan lancar, menerjemahkan perintah-perintah dari Pascal menjadi sesuatu yang dimengerti oleh sistem operasi Android.

Bagaimana cara kerja 'sihir' ini? Secara konseptual, LAMW melakukan dua hal brilian:

1. **Menyediakan Komponen 'Pembungkus' (Wrapper):** LAMW menambahkan serangkaian komponen baru ke dalam *Component Palette* Lazarus. Anda akan menemukan komponen seperti jButton, jTextView, dan jEditText. Nama-nama ini sengaja dibuat mirip dengan komponen standar Android (Button, TextView, EditText) karena pada dasarnya, mereka adalah 'pembungkus' atau representasi Pascal dari komponen-komponen asli Android tersebut.
2. **Menghubungkan Kode Pascal ke UI Native:** Saat Anda meletakkan sebuah jButton di Form Designer dan menulis kode untuk event OnClick-nya, Anda sebenarnya sedang memberi instruksi kepada LAMW. Saat proses kompilasi, LAMW akan menghasilkan kode yang memberitahu Android untuk membuat sebuah **komponen Button asli (native) Android** di layar, dan memastikan bahwa ketika tombol itu diklik, kode Pascal yang telah Anda tulis akan dieksekusi.

Hasilnya adalah yang terbaik dari dua dunia: Anda bisa merancang dan menulis logika aplikasi dalam kenyamanan dan kekuatan Object Pascal, namun aplikasi final yang berjalan di perangkat pengguna menggunakan komponen antarmuka (UI) asli Android. Ini menjamin aplikasi Anda memiliki tampilan, nuansa, dan performa yang sepenuhnya native, bukan sebuah simulasi.

**1.3. Arsitektur Aplikasi LAMW: Bagaimana Semuanya Bekerja di Balik Layar**

Setelah mengetahui peran dari setiap teknologi, Anda mungkin penasaran, "Bagaimana sebenarnya kode Pascal yang saya tulis bisa berjalan di perangkat Android?" Pertanyaan ini sangat bagus, dan memahaminya akan memberi Anda gambaran yang jelas tentang kekuatan arsitektur LAMW.

Mari kita lihat prosesnya 'di balik layar', yang dapat kita bagi menjadi dua fase utama: fase kompilasi (apa yang terjadi di komputer Anda) dan fase eksekusi (apa yang terjadi di perangkat Android).

**Fase Kompilasi (Saat Anda Menekan Tombol 'Build')**

Ini adalah serangkaian proses yang mengubah kode sumber Anda menjadi sebuah paket aplikasi.

1. **Kode Pascal Anda adalah Awalnya.** Semua logika, desain form, dan *event handler* yang Anda tulis di dalam Lazarus IDE menggunakan bahasa Object Pascal. Inilah otak dari aplikasi Anda.
2. **FPC Mengambil Alih.** Kompilator Free Pascal (FPC) kemudian membaca seluruh kode Anda dan mengompilasinya menjadi sebuah **pustaka native** (*native shared library*). Ini adalah file dengan ekstensi .so yang berisi instruksi mesin level rendah, yang dirancang khusus untuk arsitektur prosesor ARM yang digunakan di sebagian besar perangkat Android.
3. **LAMW Mengemas Semuanya.** Di sinilah peran penting LAMW terlihat. Ia mengambil file .so hasil kompilasi tadi, lalu 'membungkusnya' bersama komponen-komponen penting lainnya ke dalam satu paket aplikasi (dengan format APK atau AAB):
   * **Kode 'Lem' Java:** Sejumlah kecil kode Java yang sudah disiapkan oleh LAMW. Kode ini berfungsi sebagai titik masuk aplikasi dan jembatan awal saat aplikasi pertama kali dijalankan.
   * **Sumber Daya (Resources):** Semua aset yang Anda gunakan, seperti gambar di drawable, ikon, dan definisi layout.
   * **Manifest Aplikasi:** File AndroidManifest.xml yang krusial, yang mendeskripsikan aplikasi Anda kepada sistem operasi Android.

**Fase Eksekusi (Saat Pengguna Membuka Aplikasi)**

Setelah aplikasi terinstal dan pengguna mengetuk ikonnya, inilah yang terjadi di perangkat.

1. **Aktivitas Java Dimulai.** Sistem Android akan memulai sebuah *Activity* (layar) yang berasal dari kode 'lem' Java yang telah disiapkan LAMW.
2. **Pustaka Pascal Dipanggil.** Tugas utama dari Activity Java ini adalah untuk memuat file pustaka native .so (yang berisi seluruh logika Pascal Anda) ke dalam memori perangkat.
3. **Komunikasi via JNI.** Sejak saat itu, setiap interaksi berjalan melalui sebuah 'jembatan' teknologi yang disebut **Java Native Interface (JNI)**. Saat pengguna menekan tombol, sistem Android akan memberitahu lapisan Java, yang kemudian akan meneruskan perintah tersebut melalui JNI ke kode Pascal Anda untuk diproses. Ketika kode Pascal Anda ingin memperbarui antarmuka, misalnya mengubah teks pada jTextView, perintah itu akan dikirim kembali melalui JNI ke lapisan Java untuk dieksekusi pada komponen UI asli Android.

Secara sederhana, alurnya dapat divisualisasikan seperti ini:

[Kode Pascal di Lazarus] -> [Kompiler FPC] -> [File Pustaka Native .so] -> [Dikemas oleh LAMW] -> [Aplikasi .apk/.aab] -> [Dijalankan di Android via JNI]

Dengan memahami arsitektur ini, Anda bisa melihat bahwa aplikasi Anda benar-benar berjalan secara *native*, memberikan performa terbaik, sambil tetap memberikan kemudahan pengembangan di lingkungan Lazarus yang Anda kenal.

**1.4. Keunggulan dan Keterbatasan LAMW di Lanskap Mobile Modern**

Setiap teknologi, sekuat apa pun, pasti memiliki keunggulan dan keterbatasan. Memahaminya sejak awal akan membantu Anda membuat keputusan yang tepat dan mengelola ekspektasi saat membangun aplikasi. Mari kita lihat secara jujur posisi LAMW dalam lanskap pengembangan mobile yang sangat kompetitif di tahun 2025 ini.

**Keunggulan LAMW**

* **Performa Native.** Ini adalah keunggulan utama. Karena kode Pascal Anda dikompilasi langsung menjadi instruksi mesin, performa untuk logika bisnis, pemrosesan data, dan kalkulasi berat sering kali lebih unggul dibandingkan beberapa pendekatan *cross-platform* lainnya.
* **Siklus Pengembangan Cepat (RAD).** Filosofi *Rapid Application Development* dari Lazarus memungkinkan Anda merancang dan membuat prototipe antarmuka pengguna dengan sangat cepat melalui metode *drag-and-drop*. Ini sangat berharga untuk aplikasi yang banyak menggunakan formulir atau untuk pengembangan proyek dengan tenggat waktu yang ketat.
* **Satu Bahasa untuk Logika Inti.** Anda dapat menulis logika inti aplikasi Anda dalam Object Pascal, sebuah bahasa yang matang, kuat, dan sangat terstruktur. Ini mengurangi kompleksitas kognitif karena tidak perlu sering beralih antara beberapa bahasa (seperti Java/Kotlin untuk logika dan XML untuk layout).
* **Ukuran Aplikasi yang Potensial Lebih Kecil.** Karena tidak membawa "beban" *virtual machine* atau *runtime* JavaScript yang besar, ukuran file APK atau AAB yang dihasilkan untuk aplikasi yang relatif sederhana bisa jadi lebih kecil. Ini berarti unduhan lebih cepat dan penggunaan ruang penyimpanan yang lebih sedikit bagi pengguna.
* **Ekosistem Pascal/Delphi yang Matang.** Anda bisa memanfaatkan ribuan algoritma, unit, dan pustaka non-visual yang telah teruji selama puluhan tahun di komunitas Delphi dan Free Pascal, yang sering kali dapat diadaptasi dengan mudah.

**Keterbatasan LAMW**

* **Adopsi API Android Terbaru.** Saat Google merilis fitur atau API Android baru, mungkin ada jeda waktu sebelum fitur tersebut "dibungkus" (*wrapped*) dan tersedia secara mudah di LAMW. Anda mungkin perlu menunggu update dari tim LAMW atau mengimplementasikannya sendiri melalui JNI.
* **Integrasi Library Pihak Ketiga.** Menggunakan *library* Android modern yang populer (misalnya, *library* canggih dari Google seperti Jetpack) tidak bisa dilakukan secara *out-of-the-box*. Ini memerlukan pemahaman tentang JNI (yang akan kita bahas di bab-bab selanjutnya) untuk menjembataninya.
* **Ukuran Komunitas.** Meskipun komunitasnya solid dan sangat membantu, jumlah developer LAMW tidak sebesar komunitas Flutter, React Native, atau Kotlin Native. Ini berarti pencarian solusi untuk masalah yang sangat spesifik di forum atau Stack Overflow mungkin memerlukan usaha lebih.
* **Mengikuti Tren Desain UI.** Walaupun LAMW menyediakan komponen UI *native*, untuk mengimplementasikan pedoman desain terbaru seperti Material You dengan semua nuansa animasi dan temanya mungkin memerlukan penyesuaian properti secara manual atau bahkan kustomisasi komponen lebih lanjut.

Singkatnya, LAMW bersinar untuk aplikasi yang mengutamakan performa komputasi, dikembangkan oleh tim atau individu yang familiar dengan Pascal/Delphi, atau untuk proyek yang membutuhkan pengembangan cepat. Namun, untuk aplikasi yang sangat bergantung pada fitur-fitur ekosistem Google terbaru, Anda harus siap untuk sesekali "menyelam" lebih dalam ke level teknis yang lebih rendah.

**1.5. Peta Belajar: Dari Konsep Dasar hingga Aplikasi Siap Rilis**

Anda telah memahami 'mengapa' dan 'apa' di balik teknologi yang akan kita gunakan. Sekarang, mari kita lihat 'ke mana' kita akan melangkah. Anggap saja bagian ini sebagai peta perjalanan Anda, sebuah gambaran besar dari petualangan yang terbentang di depan. Buku ini dirancang secara sistematis untuk membawa Anda dari seorang pemula menjadi seorang developer yang percaya diri dan kompeten.

Berikut adalah tahapan-tahapan yang akan kita lalui bersama:

1. **Bagian 1: Fondasi dan Persiapan (Bab 1-2).** Di sinilah kita berada sekarang. Kita membangun pemahaman konseptual dan, yang paling penting, menyiapkan 'bengkel kerja' atau lingkungan pengembangan kita dengan benar. Keberhasilan di bagian ini adalah kunci untuk kelancaran seluruh proses belajar.
2. **Bagian 2: Inti Pengembangan Aplikasi (Bab 3-5).** Setelah semua siap, kita akan langsung terjun ke dalam kode. Kita akan membangun aplikasi fungsional pertama kita, memahami anatomi sebuah proyek LAMW, merancang antarmuka pengguna (UI) dengan berbagai komponen visual, dan belajar cara membuat aplikasi dengan beberapa layar yang saling terhubung.
3. **Bagian 3: Mengelola Data dan Jaringan (Bab 6-7).** Aplikasi yang hebat butuh 'memori' dan koneksi. Di bagian ini, kita akan belajar cara menyimpan data secara permanen di perangkat menggunakan file dan database SQLite. Kita juga akan belajar cara menghubungkan aplikasi kita ke internet untuk mengambil dan menampilkan data dinamis dari sebuah API.
4. **Bagian 4: Fitur Lanjutan dan Profesional (Bab 8-10 dan bab-bab tambahan).** Di sini, kita akan naik level secara signifikan. Kita akan menambahkan fitur-fitur canggih seperti notifikasi, akses perangkat keras (kamera & GPS), menangani tugas berat di latar belakang agar aplikasi tetap responsif, hingga mempelajari 'ilmu sihir' interoperabilitas untuk menjembatani kode Pascal dengan *library* Java/Kotlin, termasuk integrasi dengan Google Maps dan AI.
5. **Bagian 5: Produksi dan Publikasi (Bab 11-12).** Inilah garis finis. Kita akan menyatukan semua yang telah dipelajari untuk membangun satu atau dua studi kasus aplikasi yang lengkap dari awal hingga akhir. Setelah itu, kita akan melalui proses profesional untuk mempersiapkan, menandatangani, dan mempublikasikan karya Anda ke Google Play Store agar dapat diunduh oleh seluruh dunia.

Setiap bagian dibangun di atas bagian sebelumnya, menciptakan kurva belajar yang solid dan progresif. Ikat sabuk pengaman Anda, karena perjalanan untuk menjadi developer aplikasi Android profesional dengan Lazarus dan LAMW dimulai sekarang!

**Bab 2: Setup Lingkungan Pengembangan Modern (Update 2025)**

Selamat datang di Bab 2. Bab ini bisa dibilang adalah fondasi paling krusial dalam seluruh perjalanan kita. Sebelum kita bisa membangun sebuah gedung pencakar langit, kita harus memastikan tanah dan struktur dasarnya benar-benar kokoh. Inilah yang akan kita lakukan sekarang: membangun "bengkel kerja" digital kita, tempat semua keajaiban pengembangan aplikasi akan terjadi.

Sejujurnya, bagi banyak calon developer, proses instalasi dan konfigurasi sering kali menjadi rintangan pertama yang paling menantang. Ada beberapa komponen berbeda yang harus dipasang dan dihubungkan dengan benar. Namun, jangan khawatir. Saya telah merancang bab ini dengan instruksi langkah-demi-langkah yang sangat detail. Jika Anda mengikutinya dengan cermat, teliti, dan **tidak melewatkan satu langkah pun**, saya jamin di akhir bab ini Anda akan memiliki lingkungan pengembangan yang berfungsi penuh dan siap tempur.

Anggap saja bab ini seperti sebuah daftar periksa pra-penerbangan. Kita akan menginstal dan mengonfigurasi semua 'peralatan' yang kita butuhkan, mulai dari Lazarus IDE itu sendiri, perkakas penting dari Android seperti SDK dan NDK, Java Development Kit (JDK) yang menjadi tulang punggungnya, hingga pustaka LAMW yang akan menjadi perekat semuanya. Siapkan secangkir kopi atau teh, luangkan waktu Anda, dan mari kita mulai proses penyiapan ini dengan saksama.

**2.1. Prasyarat Sistem**

Sebelum kita mulai mengunduh dan menginstal perangkat lunak, mari kita pastikan komputer Anda memenuhi spesifikasi yang dibutuhkan. Memenuhi spesifikasi yang direkomendasikan akan memberikan pengalaman pengembangan yang jauh lebih lancar dan menyenangkan, terutama saat bekerja dengan Android Emulator yang cukup memakan sumber daya.

Berikut adalah daftar periksa prasyarat sistem:

* **Sistem Operasi**
  + **Direkomendasikan:** Windows 11 (64-bit) atau Windows 10 (64-bit).
  + **Alternatif:** Distribusi Linux (64-bit) modern (seperti Ubuntu 22.04 LTS atau yang lebih baru) atau macOS versi terbaru.
  + **Catatan Penting:** Versi sistem operasi **32-bit tidak lagi didukung** untuk pengembangan Android modern.
* **RAM (Random Access Memory)**
  + **Minimum:** **8 GB**. Cukup untuk menjalankan IDE dan melakukan kompilasi, namun mungkin akan terasa lambat saat menjalankan Android Emulator.
  + **Direkomendasikan:** **16 GB atau lebih**. Sangat disarankan untuk pengalaman yang mulus, responsif, dan kemampuan menjalankan beberapa program sekaligus tanpa hambatan.
* **Ruang Penyimpanan (Disk Space)**
  + **Minimum:** **25 GB** ruang kosong pada drive instalasi Anda (biasanya C:). Ruang ini dibutuhkan untuk Lazarus, JDK, Android SDK, NDK, dan beberapa komponen dasar.
  + **Direkomendasikan:** **50 GB atau lebih** ruang kosong. Ini akan memberikan ruang yang cukup untuk beberapa versi Android SDK dan *system image* untuk emulator yang berbeda tanpa perlu khawatir kehabisan tempat.
* **Koneksi Internet**
  + Diperlukan koneksi internet yang **stabil dan cukup cepat**. Anda akan mengunduh beberapa Gigabyte data untuk semua perangkat lunak yang dibutuhkan, jadi koneksi yang andal akan sangat menghemat waktu Anda.

**2.2. Instalasi Lazarus IDE versi terkini**

Langkah pertama dalam membangun 'bengkel kerja' kita adalah menginstal 'kokpit'-nya, yaitu Lazarus IDE. Ini adalah lingkungan tempat kita akan menulis kode, merancang antarmuka, dan mengompilasi aplikasi kita.

**Langkah 1: Mengunduh Lazarus**

Buka browser web Anda dan kunjungi situs web resmi Lazarus IDE di: [**https://www.lazarus-ide.org/**](https://www.lazarus-ide.org/)

Di halaman unduhan, Anda akan melihat beberapa versi. Pastikan Anda mencari dan mengunduh versi **stabil (stable)** terbaru. Per Juni 2025, versi stabil yang umum digunakan adalah seri 3.x. Yang terpenting, pastikan Anda mengunduh installer untuk sistem operasi **Windows 64-bit**. Nama filenya akan terlihat seperti lazarus-3.x.x-fpc-3.x.x-win64.exe.

**Langkah 2: Proses Instalasi**

Proses instalasi Lazarus sangatlah mudah. Ikuti langkah-langkah berikut:

1. **Jalankan Installer:** Setelah unduhan selesai, temukan file installer yang baru saja Anda unduh dan klik dua kali untuk menjalankannya. Sistem Windows mungkin akan meminta konfirmasi keamanan; klik "Yes" atau "Run" untuk melanjutkan.
2. **Ikuti Wizard Instalasi:** Anda akan disambut oleh wizard instalasi standar. Cukup klik Next untuk melanjutkan. Untuk sebagian besar pengguna, **pilihan default yang diberikan oleh installer sudah sangat baik** dan tidak perlu diubah. Biarkan installer menempatkan file di lokasi defaultnya (biasanya di C:\lazarus).
3. **Tunggu hingga Selesai:** Proses instalasi akan menyalin semua file yang diperlukan. Ini mungkin memakan waktu beberapa saat. Setelah selesai, klik Finish untuk menutup wizard instalasi.

**Langkah 3: Verifikasi Instalasi**

Untuk memastikan Lazarus terinstal dengan benar, cari ikon Lazarus di Desktop atau Start Menu Anda dan jalankan programnya.

Saat pertama kali dibuka, Lazarus akan menampilkan jendela utama dengan editor kode, *Object Inspector* di sebelah kiri, *Source Editor* di tengah, dan sebuah *Form* aplikasi kosong. Jika Anda melihat tampilan ini, selamat! Lazarus IDE telah berhasil terpasang di komputer Anda. Anda bisa menutupnya untuk saat ini karena kita akan melanjutkan instalasi komponen lainnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan. Ini adalah bagian yang cukup panjang dan sangat penting.

Berikut adalah jawaban untuk prompt **Subbab 2.3: Instalasi dan Manajemen Android SDK & NDK via Android Studio SDK Manager**.

**2.3. Instalasi dan Manajemen Android SDK & NDK via Android Studio SDK Manager**

Setelah Lazarus terpasang, langkah selanjutnya adalah menginstal 'perkakas berat' untuk pengembangan Android: **SDK (Software Development Kit)** dan **NDK (Native Development Kit)**. SDK menyediakan platform dan alat build, sementara NDK memungkinkan kompilator FPC kita untuk menghasilkan kode native yang dimengerti oleh prosesor Android.

Mungkin terdengar aneh, tetapi cara termudah dan paling direkomendasikan oleh Google untuk mengelola SDK dan NDK adalah dengan menginstal **Android Studio**. Kita tidak akan menggunakannya untuk menulis kode Pascal kita, tetapi kita akan memanfaatkan fitur **SDK Manager** di dalamnya yang sangat hebat untuk mengunduh, meng-update, dan menjaga semua komponen Android kita tetap teratur.

**Langkah 1: Instalasi Android Studio**

1. Kunjungi situs web resmi Android Developer di: [**https://developer.android.com/studio**](https://developer.android.com/studio)
2. Unduh installer Android Studio versi stabil terbaru. Ukurannya cukup besar, jadi pastikan koneksi internet Anda stabil.
3. Jalankan installer tersebut. Sama seperti Lazarus, Anda bisa mengikuti wizard instalasi dengan pilihan default. Proses ini juga akan mengunduh komponen tambahan saat pertama kali dijalankan.

**Langkah 2: Membuka dan Menggunakan SDK Manager**

Setelah instalasi selesai, jalankan Android Studio. Anda akan disambut oleh layar selamat datang. Kita tidak perlu membuat proyek baru.

1. Dari layar selamat datang, cari menu More Actions (atau Configure pada beberapa versi lama), lalu pilih **SDK Manager**.
2. Jika Anda sudah terlanjur masuk ke dalam editor, Anda bisa mengaksesnya melalui menu Tools > SDK Manager.
3. SDK Manager adalah pusat kendali untuk semua paket perangkat lunak Android Anda.

**Langkah 3: Menginstal Platform SDK**

Di dalam jendela SDK Manager, Anda akan melihat beberapa tab. Pertama, klik tab **SDK Platforms**.

Tab ini berisi daftar versi sistem operasi Android. Untuk memulai, berikan centang pada versi Android yang stabil dan relatif baru. Sebagai contoh, centang **Android 14.0, "Upside Down Cake" (API Level 34)**. Ini akan mengunduh *library* dan *system image* untuk versi Android tersebut.

*(Catatan untuk penulis: Di sini adalah tempat yang tepat untuk menyertakan screenshot dari tab 'SDK Platforms' dengan API 34 yang dicentang).*

**Langkah 4: Menginstal Perkakas SDK (SDK Tools)**

Ini adalah langkah yang paling penting. Beralihlah ke tab **SDK Tools**. Di sini kita akan memilih perkakas spesifik yang dibutuhkan oleh LAMW. Pastikan Anda memberikan centang pada item-item berikut:

* **NDK (Side by side):** Ini adalah Native Development Kit yang akan digunakan oleh FPC. Opsi 'Side by side' memungkinkan beberapa versi NDK terinstal bersamaan tanpa konflik. Centang versi NDK yang direkomendasikan.
* **Android SDK Build-Tools:** Ini adalah kompilator dan perkakas yang digunakan Android untuk membangun aplikasi Anda. SDK Manager biasanya sudah memilih versi terbaru secara otomatis.
* **Android SDK Command-line Tools (latest):** Perkakas baris perintah yang esensial untuk berbagai proses build di belakang layar.
* **Android Emulator:** Beri centang jika Anda ingin menggunakan emulator resmi dari Android Studio untuk menjalankan dan menguji aplikasi Anda.

Setelah memberikan centang pada semua item di atas, klik Apply atau OK. SDK Manager akan menampilkan daftar paket yang akan diunduh. Setujui lisensinya dan lanjutkan. Proses ini mungkin akan memakan waktu cukup lama.

*(Catatan untuk penulis: Screenshot dari tab 'SDK Tools' dengan item-item di atas yang dicentang sangatlah krusial di sini).*

**Langkah 5: Mencatat Lokasi SDK (Langkah Kritis!)**

Sebelum Anda menutup SDK Manager, perhatikan bagian atas jendela tersebut. Ada sebuah kolom teks berlabel **Android SDK Location**.

**Salin (copy) seluruh path yang ada di kolom ini** (contohnya: C:\Users\NamaAnda\AppData\Local\Android\Sdk) dan **simpan di sebuah file Notepad** atau di mana pun yang mudah Anda akses. Path ini sangat vital dan akan kita butuhkan pada langkah konfigurasi selanjutnya untuk memberitahu Lazarus di mana semua perkakas Android ini berada. Jangan sampai lupa atau salah menyalin path ini.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 2.4: Instalasi Java Development Kit (JDK) yang Direkomendasikan**.

**2.4. Instalasi Java Development Kit (JDK) yang Direkomendasikan**

Komponen penting berikutnya adalah Java Development Kit atau JDK. Meskipun kita menulis kode dalam Pascal, banyak perkakas di dalam Android SDK yang sebenarnya ditulis dalam bahasa Java dan membutuhkan lingkungan Java untuk bisa berjalan. Proses *build* aplikasi Android sangat bergantung pada JDK. Oleh karena itu, instalasi JDK ini adalah sebuah keharusan.

**Memilih Versi dan Sumber JDK yang Tepat**

Dunia Java memiliki banyak versi dan distribusi. Agar kompatibel dengan perkakas Android modern dan menghindari masalah di kemudian hari, versi *Long-Term Support* (LTS) yang paling stabil dan direkomendasikan adalah **JDK 17**. Sangat disarankan untuk tidak menggunakan versi yang jauh lebih baru atau lebih lama kecuali Anda tahu pasti apa yang Anda lakukan.

Sumber terbaik untuk mengunduh OpenJDK (versi *open-source* dari JDK) yang bersih dan tepercaya adalah dari proyek **Eclipse Adoptium (Temurin)**.

**Langkah-langkah Instalasi**

1. **Kunjungi Situs Adoptium:** Buka browser Anda dan kunjungi [**https://adoptium.net/**](https://adoptium.net/).
2. **Unduh Installer:** Di halaman utama, pastikan Anda memilih versi **17 (LTS)** dan arsitektur **x64** untuk sistem operasi **Windows**. Penting untuk memilih format installer **.msi** agar kita mendapatkan wizard instalasi yang mudah dikonfigurasi.
3. **Jalankan Installer (Langkah Penting):**
   * Setelah unduhan selesai, jalankan file .msi tersebut.
   * Ikuti wizard hingga Anda sampai pada layar **'Custom Setup'**.
   * Pada layar ini, Anda akan melihat beberapa fitur. Klik ikon di sebelah kiri fitur **Set JAVA\_HOME variable** dan pilih opsi **'Will be installed on local hard drive'**.
   * Lakukan hal yang sama untuk fitur **Add to PATH** jika belum terpilih.

*(Catatan untuk penulis: Screenshot dari layar 'Custom Setup' installer Adoptium dengan pilihan JAVA\_HOME dan Add to PATH yang disorot sangat direkomendasikan di sini).*

Langkah ini **sangat penting** karena akan secara otomatis mengatur *environment variable* JAVA\_HOME dan Path untuk Anda. Melakukan ini akan mencegah banyak masalah konfigurasi di kemudian hari.

1. **Selesaikan Instalasi:** Klik Next dan Install untuk melanjutkan, lalu Finish setelah proses instalasi selesai.

**Verifikasi Instalasi**

Untuk memastikan JDK telah terinstal dan terkonfigurasi dengan benar, kita akan melakukan pengecekan melalui Command Prompt.

1. Buka Command Prompt (cari cmd di Start Menu).
2. Ketik perintah berikut dan tekan Enter:
3. java -version
4. Kemudian ketik perintah kedua dan tekan Enter:
5. javac -version

Jika kedua perintah tersebut berhasil dan menampilkan output yang mengandung openjdk version "17.x.x", berarti JDK Anda sudah siap. Jika Anda mendapatkan pesan error seperti '...is not recognized as an internal or external command...', ulangi langkah instalasi dan pastikan opsi 'Add to PATH' dan 'Set JAVA\_HOME' telah dipilih dengan benar.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke instalasi LAMW.

Berikut adalah jawaban untuk prompt **Subbab 2.5: Instalasi Pustaka LAMW dari Online Package Manager**.

**2.5. Instalasi Pustaka LAMW dari Online Package Manager**

Setelah semua perkakas pendukung (Lazarus, Android Tools, JDK) terpasang, kini saatnya kita menginstal 'lem perekat' yang akan menyatukan semuanya: pustaka LAMW itu sendiri. Untungnya, proses ini sangat mudah berkat fitur *Online Package Manager* (OPM) yang sudah terintegrasi di dalam Lazarus.

**Langkah-langkah Instalasi LAMW**

1. **Buka Lazarus IDE:** Jalankan Lazarus IDE yang telah kita instal sebelumnya.
2. **Buka Online Package Manager (OPM):** Dari menu bar di bagian atas, klik Package, lalu pilih Online Package Manager. Sebuah jendela baru akan terbuka, menampilkan daftar pustaka yang tersedia untuk diunduh dari repositori pusat.
3. **Cari Pustaka LAMW:** Di bagian atas jendela OPM, ada sebuah kotak pencarian. Ketik lamw di sana. Daftar pustaka akan tersaring secara otomatis dan menampilkan **'lamw' (Lazarus Android Module Wizard)**.
4. **Pilih dan Instal:** Klik pada lamw di daftar untuk memilihnya, lalu klik tombol Install yang ada di sebelah kanan. Sebuah jendela konfirmasi akan muncul, menanyakan apakah Anda ingin mengunduh dan menginstal paket tersebut beserta dependensinya. Klik Yes.

**Proses Rebuild Lazarus (Bagian Penting)**

Setelah Anda mengonfirmasi instalasi, Lazarus akan melakukan sesuatu yang mungkin terlihat aneh bagi pengguna baru: ia akan **membangun ulang (rebuild) dirinya sendiri**.

Ini adalah proses yang normal dan sangat penting. Lazarus sedang mengintegrasikan komponen-komponen baru dari LAMW ke dalam intinya, termasuk menambahkan item baru ke menu File > New... dan menambahkan semua komponen j-controls ke palet komponen. Proses ini mungkin memakan waktu beberapa menit. Anda akan melihat banyak pesan kompilasi di jendela *Messages*. **Biarkan saja proses ini berjalan tanpa gangguan hingga selesai.** Setelah selesai, Lazarus akan otomatis tertutup dan memulai ulang (restart).

**Verifikasi Instalasi**

Setelah Lazarus terbuka kembali, kita bisa dengan mudah memastikan bahwa LAMW telah terpasang dengan benar.

1. Klik menu File > New....
2. Di dalam jendela New Item yang muncul, Anda sekarang seharusnya melihat sebuah opsi proyek baru bernama **LAMW Android Module Project**.

Jika opsi ini ada, berarti instalasi LAMW telah sukses! Anda tidak perlu membuat proyek baru saat ini, cukup klik Cancel untuk menutup jendela New Item tersebut. Kita sudah sangat dekat untuk menyelesaikan semua persiapan.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.6.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke langkah konfigurasi. Ini adalah bagian yang menyatukan semuanya.

Berikut adalah jawaban untuk prompt **Subbab 2.6: Konfigurasi Kritis: Menghubungkan Lazarus dengan Tools Android**.

**2.6. Konfigurasi Kritis: Menghubungkan Lazarus dengan Tools Android**

Inilah langkah penyatuan, momen di mana kita menghubungkan semua komponen yang telah kita instal. Di langkah ini, kita akan memberitahu Lazarus di mana lokasi 'perkakas' JDK, Android SDK, dan NDK berada. Kesalahan kecil di sini akan menyebabkan build gagal di kemudian hari, jadi mari kita lakukan dengan sangat teliti.

**Langkah-langkah Konfigurasi Path**

1. **Buka Jendela Options:** Jalankan Lazarus IDE. Dari menu bar di bagian atas, klik Tools, lalu pilih Options.... Jendela Options akan muncul.
2. **Navigasi ke Pengaturan LAMW:** Di panel sebelah kiri pada jendela Options, gulir ke bawah hingga Anda menemukan ikon LAMW (biasanya bergambar robot Android hijau) dan klik padanya. Pengaturan spesifik untuk LAMW akan muncul di panel kanan.
3. **Mengisi Path (Bagian Paling Kritis):** Pastikan Anda berada di tab **Paths**. Anda akan melihat beberapa kolom yang perlu diisi. Inilah saatnya menggunakan informasi yang telah kita kumpulkan dari langkah-langkah sebelumnya.

*(Catatan untuk penulis: Screenshot dari jendela 'Options -> LAMW -> Paths' yang sudah terisi lengkap adalah gambar yang paling penting untuk disertakan di seluruh Bab 2 ini).*

* + **Path to JDK:** Klik tombol ... di sebelah kanan kolom ini. Sebuah dialog pemilihan folder akan muncul. Navigasikan ke folder tempat Anda menginstal JDK 17. Lokasi defaultnya biasanya di C:\Program Files\Eclipse Adoptium\jdk-17.x.x.x-hotspot\. Pilih folder utama JDK tersebut, lalu klik OK.
  + **Path to Android SDK:** Ini adalah path yang telah Anda **simpan di Notepad** pada Subbab 2.3. Salin dan tempelkan (paste) path tersebut di sini. Contohnya: C:\Users\NamaAnda\AppData\Local\Android\Sdk.
  + **Path to Android NDK:** Klik tombol .... Navigasikan ke dalam folder Android SDK Anda (path dari poin sebelumnya), lalu masuk ke dalam folder ndk. Di dalamnya, Anda akan menemukan satu atau lebih folder dengan nomor versi. Pilih folder versi NDK yang terinstal (misalnya, 25.2.9519653). Klik OK.

1. **Simpan Pengaturan:** Setelah ketiga path tersebut terisi dengan benar, periksa kembali sekali lagi. Jika sudah yakin, klik tombol **OK** di bagian bawah jendela Options untuk menyimpan semua perubahan.
2. **Restart Lazarus (Sangat Disarankan):** Untuk memastikan semua path yang baru Anda masukkan dibaca dan diterapkan dengan sempurna oleh IDE, tutup Lazarus sepenuhnya lalu jalankan kembali.

Setelah langkah ini, 'bengkel kerja' kita secara teori sudah terhubung sepenuhnya dan siap digunakan.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.7.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 2.7: Troubleshooting Instalasi: Solusi untuk Masalah Umum**.

**2.7. Troubleshooting Instalasi: Solusi untuk Masalah Umum**

Meskipun kita telah mengikuti semua langkah dengan hati-hati, terkadang masalah tetap bisa muncul. Perbedaan versi sistem operasi, konfigurasi keamanan, atau kesalahan kecil saat menyalin path bisa menyebabkan error. Jangan panik! Ini adalah bagian normal dari proses setup. Di bagian ini, kita akan membahas beberapa masalah yang paling sering terjadi dan cara mengatasinya.

**Masalah 1: Error JDK path not found atau javac: not found saat build.**

* **Penyebab:** Lazarus tidak dapat menemukan instalasi Java Development Kit Anda.
* **Solusi:**
  1. **Periksa Kembali Path di Lazarus:** Buka Tools > Options > LAMW > Paths. Pastikan Path to JDK sudah menunjuk ke direktori utama JDK 17 (misalnya, C:\Program Files\Eclipse Adoptium\jdk-17...), bukan sub-folder di dalamnya seperti bin.
  2. **Verifikasi Environment Variable:** Buka Command Prompt dan ketik echo %JAVA\_HOME%. Pastikan path yang muncul sudah benar. Jika tidak ada yang muncul, berarti proses instalasi JDK sebelumnya gagal mengatur variabel ini. Ulangi instalasi JDK dan pastikan Anda mencentang opsi 'Set JAVA\_HOME variable'.

**Masalah 2: Error Android SDK path not found.**

* **Penyebab:** Lazarus tidak tahu di mana lokasi perkakas Android Anda.
* **Solusi:**
  1. **Periksa Path SDK:** Buka kembali Tools > Options > LAMW > Paths. Bandingkan Path to Android SDK dengan path yang Anda salin dari Android Studio SDK Manager. Pastikan tidak ada salah ketik, spasi tambahan, atau karakter yang hilang.
  2. **Pastikan SDK Terinstal:** Buka Android Studio SDK Manager dan pastikan komponen-komponen dasar (seperti Platform-Tools dan Build-Tools) benar-benar sudah terinstal (statusnya 'Installed').

**Masalah 3: Error Could not find Build-Tools revision xx.x.x.**

* **Penyebab:** Proyek Anda membutuhkan versi Build-Tools tertentu yang belum terinstal di SDK Manager.
* **Solusi:**
  1. **Instal Versi yang Diminta:** Catat nomor versi yang diminta oleh pesan error.
  2. Buka Android Studio SDK Manager, masuk ke tab SDK Tools, dan centang opsi Show Package Details di pojok kanan bawah.
  3. Cari bagian Android SDK Build-Tools. Temukan dan centang versi persis yang diminta oleh pesan error (misalnya, 33.0.1), lalu klik Apply untuk menginstalnya.

**Masalah 4: Error NDK not found.**

* **Penyebab:** Mirip dengan masalah SDK, path ke NDK salah atau NDK belum terinstal.
* **Solusi:**
  1. **Verifikasi Instalasi NDK:** Di Android Studio SDK Manager (tab SDK Tools), pastikan NDK (Side by side) sudah tercentang dan terinstal.
  2. **Periksa Path NDK:** Di Tools > Options > LAMW > Paths, pastikan Path to Android NDK sudah menunjuk ke direktori NDK yang benar di dalam folder SDK (contoh: ...\Sdk\ndk\25.2.9519653).

**Masalah 5: Lazarus IDE Macet (Hang) atau Gagal saat Membangun Ulang (Rebuild).**

* **Penyebab:** Terjadi konflik atau masalah izin (permission) saat Lazarus mencoba menginstal paket LAMW.
* **Solusi:**
  1. **Jalankan sebagai Administrator:** Tutup Lazarus. Klik kanan pada ikon Lazarus di Desktop atau Start Menu, lalu pilih Run as administrator. Coba ulangi proses instalasi paket dari OPM.
  2. **Instalasi Bersih:** Jika masalah berlanjut, mungkin ada konflik. Uninstall Lazarus sepenuhnya, hapus folder C:\lazarus (setelah mencadangkan proyek Anda), lalu lakukan instalasi ulang Lazarus dan LAMW dari awal.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 2.8.

YAHOX

Tampilkan alur berpikir

Baik, kita selesaikan bab persiapan ini.

Berikut adalah jawaban untuk prompt **Subbab 2.8: Menyiapkan Emulator (AVD) dan Perangkat Fisik untuk Debugging**.

**2.8. Menyiapkan Emulator (AVD) dan Perangkat Fisik untuk Debugging**

Selamat, semua perangkat lunak dan konfigurasi di komputer Anda sudah selesai! Namun, sebuah aplikasi tentu perlu tempat untuk dijalankan dan diuji. Di bagian terakhir dari bab persiapan ini, kita akan menyiapkan 'panggung' tersebut. Anda memiliki dua pilihan utama: menggunakan perangkat virtual (Emulator) yang berjalan di komputer, atau menggunakan perangkat Android fisik (ponsel/tablet Anda sendiri).

**Opsi 1: Menyiapkan Emulator (Android Virtual Device - AVD)**

Emulator adalah simulasi perangkat Android yang sangat berguna untuk menguji aplikasi di berbagai ukuran layar dan versi Android tanpa perlu memiliki semua perangkat fisiknya. Kita akan menggunakan AVD Manager yang ada di dalam Android Studio.

1. **Buka Device Manager:** Jalankan **Android Studio**. Dari layar selamat datang, pilih More Actions > Virtual Device Manager. (Jika ada proyek yang terbuka, Anda bisa mengaksesnya dari menu Tools > Device Manager).
2. **Buat Perangkat Virtual Baru:** Di jendela Device Manager, klik tombol Create Device atau Create virtual device.
3. **Pilih Model Perangkat Keras:** Pilih kategori Phone dan pilih salah satu model perangkat yang populer, misalnya Pixel 7 atau Pixel 8. Ini akan menentukan ukuran layar dan resolusinya. Klik Next.
4. **Pilih System Image:** Anda akan diminta memilih versi Android yang akan dijalankan di emulator. Pilihlah versi yang sama dengan yang Anda instal di SDK Platforms, misalnya **API 34 (Android 14.0)**. Jika gambar sistem tersebut belum ada di komputer Anda, akan ada link Download di sebelahnya. Klik untuk mengunduh, dan setelah selesai, pastikan versi tersebut terpilih lalu klik Next.
5. **Konfigurasi Akhir:** Anda bisa memberi nama khusus untuk AVD Anda jika mau. Klik Finish untuk menyelesaikan pembuatan.
6. **Menjalankan Emulator:** Untuk menjalankan emulator, kembali ke daftar perangkat di Device Manager dan klik ikon 'Play' (▶) di sebelah nama emulator yang baru Anda buat. Tunggu beberapa saat hingga emulator selesai *booting* dan menampilkan layar utama Android.

**Opsi 2: Menyiapkan Perangkat Android Fisik (Sangat Direkomendasikan)**

Menguji aplikasi di perangkat fisik memberikan hasil yang paling akurat, terutama untuk performa dan fitur yang bergantung pada perangkat keras. Prosesnya sedikit lebih teknis tetapi sangat sepadan.

1. **Aktifkan Opsi Pengembang:** Di ponsel atau tablet Android Anda, buka Setelan > Tentang ponsel (*Settings > About phone*). Gulir ke bawah dan cari entri Nomor build (*Build number*). **Ketuk (tap) pada 'Nomor build' ini sebanyak 7 kali berturut-turut.** Anda akan melihat notifikasi hitungan mundur dan akhirnya pesan "Anda sekarang adalah seorang developer!".
2. **Aktifkan USB Debugging:** Kembali ke menu utama Setelan, cari menu baru bernama Opsi developer (*Developer options*) yang biasanya ada di dalam menu Sistem. Buka menu tersebut, cari, dan aktifkan saklar untuk **Debugging USB** (*USB debugging*).
3. **Hubungkan Perangkat ke Komputer:** Gunakan kabel USB yang berkualitas baik untuk menghubungkan perangkat Android Anda ke komputer.
4. **Berikan Otorisasi:** Saat pertama kali terhubung dengan mode debugging aktif, sebuah dialog akan muncul di layar ponsel Anda yang menanyakan "Izinkan debugging USB?". Centang opsi **'Selalu izinkan dari komputer ini'** (*Always allow from this computer*) lalu ketuk Izinkan (*Allow*). Ini adalah langkah keamanan penting agar komputer Anda bisa berkomunikasi dengan ponsel.
5. **(Jika Perlu) Instal Driver USB:** Untuk pengguna Windows, jika perangkat Anda tidak terdeteksi oleh komputer, Anda mungkin perlu menginstal driver USB khusus dari pabrikan ponsel Anda (misalnya, Samsung USB Driver, Xiaomi USB Driver, dll.). Cukup cari di Google dengan kata kunci "[merek ponsel Anda] USB driver" untuk menemukannya.

Dengan salah satu (atau kedua) opsi ini siap, 'bengkel kerja' dan 'panggung' Anda kini lengkap. Di bab selanjutnya, kita akhirnya akan mulai membangun aplikasi pertama kita!

Ini mengakhiri Bab 2. Selamat, semua persiapan yang rumit telah selesai! Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 3 (Pengantar Bab 3).

YAHOX

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 3 yang menarik.

Berikut adalah jawaban untuk prompt **Pengantar Bab 3**.

**Bab 3: Proyek Pemanasan: Membangun Aplikasi Kalkulator Sederhana**

Selamat! Anda telah berhasil melewati bab persiapan yang paling teknis. Semua 'perkakas' dan 'mesin' di bengkel kerja Anda sudah siap dan terhubung dengan baik. Kini saatnya untuk berhenti sejenak dari teori dan mulai melakukan hal yang paling menyenangkan: mengubah baris-baris kode menjadi aplikasi yang nyata.

Sebagai proyek pemanasan, kita tidak akan hanya membuat aplikasi "Halo Dunia!" yang sederhana. Kita akan langsung membuat sesuatu yang lebih nyata dan fungsional: sebuah **Aplikasi Kalkulator Sederhana**. Mengapa kalkulator? Karena ini adalah proyek pertama yang sempurna. Proyek ini akan langsung mengajak Anda untuk mempraktikkan tiga pilar utama pengembangan aplikasi:

* **Merancang Layout Antarmuka (UI):** Kita akan belajar menyusun tombol dan layar secara presisi.
* **Menangani Interaksi Pengguna:** Kita akan membuat tombol-tombol tersebut benar-benar berfungsi saat disentuh.
* **Menerapkan Logika Dasar:** Kita akan menulis kode Pascal untuk melakukan operasi-operasi sederhana.

Tujuan kita di bab ini sangat jelas. Di akhir bab nanti, Anda tidak hanya akan memahami prosesnya, tetapi Anda akan memiliki sebuah file .apk yang bisa Anda instal, jalankan, dan pamerkan di perangkat Android Anda sendiri. Anda akan merasakan kepuasan melihat aplikasi buatan tangan Anda sendiri bekerja.

Jadi, mari buka Lazarus IDE Anda, singsingkan lengan baju, dan kita mulai bangun sesuatu yang keren bersama-sama!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai membangun.

Berikut adalah jawaban untuk prompt **Subbab 3.1: Bukan Sekadar "Halo Dunia": Membuat Proyek yang Fungsional**.

**3.1. Bukan Sekadar "Halo Dunia": Membuat Proyek yang Fungsional**

Baik, Lazarus IDE sudah terbuka di hadapan kita. Langkah pertama untuk setiap aplikasi baru adalah membuat sebuah proyek. Proyek ini akan menjadi 'wadah' untuk semua file, kode, dan sumber daya yang akan kita gunakan untuk membangun kalkulator kita.

Mari ikuti langkah-langkah berikut untuk membuat proyek pertama Anda:

1. **Buka Dialog Proyek Baru.** Dari menu bar di bagian atas, klik File, kemudian pilih New.... Sebuah jendela New Item akan muncul, menampilkan berbagai jenis proyek yang bisa dibuat.
2. **Pilih Template Proyek LAMW.** Cari dan pilih **LAMW Android Module Project**. Opsi ini ada karena kita telah berhasil menginstal pustaka LAMW di bab sebelumnya. Klik OK.
3. **Isi Detail Proyek.** Anda sekarang akan melihat dialog **New Android Module Project**. Ini adalah langkah penting untuk memberikan identitas pada aplikasi kita. Mari kita isi kolom-kolomnya:
   * **Application Name:** Ketik nama aplikasi yang akan dilihat pengguna di layar ponsel mereka, misalnya: Kalkulator Sederhana.
   * **Package Name:** Ini adalah identitas unik aplikasi Anda di seluruh ekosistem Android. Aturannya adalah seperti nama domain terbalik untuk menghindari konflik dengan aplikasi lain. Sebagai contoh, gunakan format com.namaanda.kalkulatorsederhana. Gantilah namaanda dengan nama Anda atau nama perusahaan Anda. **Penting:** Nama paket ini tidak dapat diubah setelah aplikasi dipublikasikan ke Google Play Store, jadi pastikan Anda membuatnya unik.
   * **Opsi Lainnya:** Biarkan semua opsi lain pada nilai defaultnya untuk proyek pertama kita ini.
4. **Buat Proyek.** Klik OK. Lazarus akan bekerja sejenak untuk membuat semua file dan struktur direktori yang diperlukan untuk proyek baru Anda.

**Melihat Hasilnya: Proyek Kosong Anda**

Setelah proses selesai, Anda akan disambut oleh lingkungan kerja proyek baru Anda. Perhatikan beberapa hal di layar:

* **Form Designer:** Di tengah, Anda melihat sebuah area abu-abu yang merepresentasikan layar ponsel. Inilah kanvas kita, sebuah jForm kosong yang siap untuk kita desain.
* **Source Editor:** Di belakangnya, ada tab dengan unit uMainForm.pas, tempat kita akan menulis kode Pascal nanti.
* **Object Inspector:** Di sebelah kiri, panel ini menampilkan properti dari item yang sedang dipilih (saat ini adalah jForm1).

Proyek kosong kita kini telah siap. Di langkah selanjutnya, kita akan mulai 'menggambar' di atas kanvas ini untuk menciptakan antarmuka kalkulator kita.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke perancangan layout.

Berikut adalah jawaban untuk prompt **Subbab 3.2: Merancang Layout Kalkulator dengan jLinearLayout dan jGridLayout**.

**3.2. Merancang Layout Kalkulator dengan jLinearLayout dan jGridLayout**

Sekarang kita memiliki kanvas kosong. Sebelum meletakkan tombol-tombol dan layar, seorang desainer yang baik akan merencanakan tata letaknya terlebih dahulu. Untuk kalkulator kita, kita akan menggunakan kombinasi dua komponen layout yang sangat kuat: jLinearLayout untuk mengatur bagian-bagian utama, dan jGridLayout untuk menyusun tombol-tombol kita secara presisi.

Rencananya sederhana: kita akan memiliki satu kontainer utama vertikal. Di dalamnya, bagian atas akan diisi oleh layar display, dan bagian bawah akan diisi oleh sebuah grid (kotak-kotak) untuk tombol angka dan operasi.

**Langkah 1: Membuat Kontainer Utama (jLinearLayout)**

1. **Pilih Komponen:** Di **Component Palette** (biasanya berada di panel bagian atas IDE Lazarus), cari dan klik ikon jLinearLayout. Ikonnya biasanya terlihat seperti beberapa kotak yang tersusun berbaris.
2. **Letakkan di Form:** Arahkan kursor mouse Anda dan klik di mana saja di dalam area abu-abu jForm1 yang kosong. Sebuah jLinearLayout akan muncul.
3. **Atur Properti di Object Inspector:** Pastikan jLinearLayout1 yang baru saja Anda tambahkan sedang terpilih (ada kotak-kotak kecil di sekelilingnya). Sekarang, lihat ke panel **Object Inspector** di sebelah kiri. Cari dan ubah properti-properti berikut:
   * **ID:** Ubah namanya menjadi layoutUtama agar mudah dikenali di kode nanti.
   * **Orientation:** Ubah nilainya dari loHorizontal menjadi loVertical. Ini memberitahu layout untuk menyusun semua komponen di dalamnya dari atas ke bawah.
   * **layout\_width:** Ubah menjadi match\_parent. Artinya, lebar layout ini akan selalu mengikuti lebar layar perangkat.
   * **layout\_height:** Ubah juga menjadi match\_parent. Artinya, tinggi layout ini akan memenuhi seluruh layar perangkat.

**Langkah 2: Menambahkan Area untuk Layar dan Grid Tombol**

Sekarang kita akan mengisi layoutUtama kita dengan dua area utama.

1. **Tambahkan Area Layar:** Pilih komponen jTextView dari Component Palette, lalu klik di dalam layoutUtama. Sebuah jTextView akan muncul di bagian atas. Untuk saat ini, biarkan saja apa adanya, kita akan mengonfigurasinya di subbab berikutnya.
2. **Tambahkan Grid Tombol:** Selanjutnya, pilih komponen jGridLayout dari Component Palette dan klik lagi di dalam layoutUtama. Sebuah area grid akan muncul di bawah jTextView.
3. **Atur Properti jGridLayout:** Pilih jGridLayout1 yang baru Anda buat, dan di Object Inspector, atur properti berikut:
   * **ID:** Ubah menjadi gridTombol.
   * **layout\_width:** Atur menjadi match\_parent.
   * **layout\_height:** Atur menjadi wrap\_content. Ini artinya tinggi grid akan menyesuaikan dengan isi tombol-tombol di dalamnya nanti, tidak memenuhi sisa layar.
   * **ColumnCount:** Atur nilainya menjadi 4. Kalkulator kita akan memiliki 4 tombol per baris.
   * **RowCount:** Atur nilainya menjadi 5. Kita akan membutuhkan 5 baris untuk semua tombol kita.

**Memeriksa Struktur (Hierarki)**

Untuk melihat hasil dari pekerjaan kita, lihatlah panel **Object Tree View** (biasanya ada di atas Object Inspector). Anda akan melihat sebuah struktur hierarki yang rapi seperti ini, yang menunjukkan hubungan "induk-anak" antar komponen:

jForm1

└── layoutUtama (jLinearLayout)

├── jTextView1 (jTextView)

└── gridTombol (jGridLayout)

Struktur ini membuktikan bahwa kita telah berhasil merancang fondasi layout untuk kalkulator kita. jTextView1 dan gridTombol sekarang adalah 'anak' dari layoutUtama.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 3.3: Menggunakan jTextView sebagai Layar dan jButton untuk Tombol**.

**3.3. Menggunakan jTextView sebagai Layar dan jButton untuk Tombol**

Fondasi layout kita sudah kokoh. Sekarang adalah bagian yang menyenangkan: mengisi kerangka tersebut dengan komponen-komponen yang akan dilihat dan disentuh oleh pengguna, yaitu layar display dan tombol-tombol kalkulator.

**Langkah 1: Mengonfigurasi Layar Display (jTextView)**

Kita akan mengubah jTextView sederhana yang sudah kita letakkan menjadi sebuah layar display yang proper.

1. **Pilih Komponen:** Di *Form Designer*, klik pada komponen jTextView1 yang berada di bagian atas layoutUtama.
2. **Atur Properti di Object Inspector:** Di panel **Object Inspector** sebelah kiri, kita akan mengubah beberapa propertinya agar sesuai dengan fungsinya sebagai layar kalkulator:
   * **ID:** Ubah nilainya menjadi tvDisplay. Ini adalah nama variabel yang akan kita gunakan di kode Pascal untuk merujuk ke komponen layar ini.
   * **Text:** Ubah teks default-nya menjadi 0. Ini adalah nilai awal yang akan ditampilkan saat kalkulator pertama kali dibuka.
   * **TextSize:** Perbesar ukurannya agar mudah dibaca. Ketik 48 lalu tekan Enter.
   * **TextColor:** Agar terlihat lebih modern, ubah warnanya menjadi clWhite.
   * **Gravity:** Properti ini mengatur perataan teks. Dari daftar *dropdown*, pilih **right**. Ini akan membuat angka-angka kalkulator selalu rata ke kanan, seperti pada kalkulator umumnya.
   * **layout\_weight:** Ini properti yang penting untuk layout kita. Atur nilainya menjadi 1. Dengan nilai 1, tvDisplay akan meregang secara fleksibel untuk mengisi semua ruang vertikal yang tersisa di dalam layoutUtama.

**Langkah 2: Menambahkan Tombol-tombol ke Grid**

Sekarang kita akan mengisi gridTombol kita satu per satu. Prosesnya akan sedikit berulang, tetapi ini adalah latihan yang sangat baik untuk membiasakan diri dengan alur kerja di Lazarus: **Pilih komponen -> Letakkan di grid -> Atur properti ID dan Text**.

Mari kita mulai dari baris pertama:

1. **Tombol '7':** Pilih komponen jButton dari *Component Palette*, lalu klik pada sel pertama (kiri atas) di dalam gridTombol. Pilih tombol baru tersebut, dan di Object Inspector, atur:
   * **ID:** btn7
   * **Text:** 7
2. **Tombol '8':** Lakukan hal yang sama. Pilih jButton, klik di sel kedua dari kiri. Atur:
   * **ID:** btn8
   * **Text:** 8

Lanjutkan proses ini untuk semua tombol lainnya sesuai dengan tata letak kalkulator standar. Gunakan panduan lengkap di bawah ini untuk memastikan properti ID dan Text dari setiap tombol sudah benar. Memberi ID yang bermakna seperti ini akan sangat membantu kita saat menulis kode nanti.

* **Baris 1:**
  + btn7 (Text: 7), btn8 (Text: 8), btn9 (Text: 9), btnBagi (Text: /)
* **Baris 2:**
  + btn4 (Text: 4), btn5 (Text: 5), btn6 (Text: 6), btnKali (Text: \*)
* **Baris 3:**
  + btn1 (Text: 1), btn2 (Text: 2), btn3 (Text: 3), btnKurang (Text: -)
* **Baris 4:**
  + btnTitik (Text: .), btn0 (Text: 0), btnSamaDengan (Text: =), btnTambah (Text: +)
* **Baris 5:**
  + btnClear (Text: C), btnHapus (Text: DEL)

Untuk dua tombol terakhir (btnClear dan btnHapus), Anda bisa meletakkannya di dua sel pertama baris kelima, dan membiarkan dua sel sisanya kosong.

Setelah semua tombol diletakkan, antarmuka kalkulator kita secara visual sudah lengkap!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan untuk memberikan "nyawa" pada kalkulator kita.

Berikut adalah jawaban untuk prompt **Subbab 3.4: Logika Dasar Kalkulator: Menangani Input Angka dan Operasi**.

**3.4. Logika Dasar Kalkulator: Menangani Input Angka dan Operasi**

Antarmuka kalkulator kita sudah terlihat bagus, tetapi saat ini ia masih 'bisu dan tuli'. Tombol-tombolnya belum melakukan apa pun saat disentuh. Di bagian ini, kita akan memberikan 'nyawa' pada tombol-tombol tersebut dengan menulis logika Pascal di belakangnya menggunakan *event handler*.

**Logika Tombol Clear (C) - Cara Termudah**

Mari kita mulai dengan yang paling mudah: tombol 'C' (btnClear) yang berfungsi untuk mereset layar kembali menjadi '0'.

1. **Buat Event Handler:** Di *Form Designer*, **klik dua kali (double-click)** pada tombol **'C'**. Lazarus akan secara otomatis melakukan dua hal: membawa Anda ke *Source Editor* dan membuatkan sebuah prosedur kosong bernama TAndroidModule1.btnClearClick. Prosedur inilah yang akan dijalankan setiap kali tombol 'C' diklik.
2. **Tulis Kodenya:** Di dalam prosedur yang baru saja dibuat, ketikkan satu baris kode berikut:

Delphi

procedure TAndroidModule1.btnClearClick(Sender: TObject);

begin

tvDisplay.Text := '0';

end;

Kode ini sangat sederhana: ia mengakses komponen tvDisplay (layar kita) dan mengatur properti Text-nya kembali menjadi string '0'.

**Logika Tombol Angka - Cara Cerdas dengan Event Handler Bersama**

Sekarang, untuk tombol angka (0-9). Apakah kita harus membuat 10 prosedur terpisah untuk setiap tombol? Tentu tidak! Kita akan menggunakan teknik yang lebih efisien yang disebut **Shared Event Handler** (Penangan Event Bersama). Idenya adalah kita membuat *satu* prosedur, lalu menugaskan semua tombol angka untuk menjalankan prosedur yang sama tersebut saat diklik.

1. **Buat Prosedur Secara Manual:** Di *Source Editor*, di bagian private dari TAndroidModule1, deklarasikan prosedur baru kita. Kemudian tulis implementasinya di bawah.

Delphi

// Di bagian atas, di dalam 'type TAndroidModule1...'

private

{ private declarations }

procedure OnNumberClick(Sender: TObject); // <-- Tambahkan deklarasi ini

// ...

// Di bagian implementasi, di bawah 'end;' dari prosedur lain

procedure TAndroidModule1.OnNumberClick(Sender: TObject);

var

caption: string;

begin

// 1. Ambil teks dari tombol yang ditekan

caption := (Sender as jControl).GetText;

// 2. Logika untuk menampilkan angka

if (tvDisplay.Text = '0') and (caption <> '.') then

tvDisplay.Text := caption

else

tvDisplay.Text := tvDisplay.Text + caption;

end;

**Penjelasan Kode:**

* + caption := (Sender as jControl).GetText; — Ini adalah bagian kuncinya. Sender adalah parameter yang merujuk pada komponen *mana* yang memicu event ini (tombol mana yang diklik). Kita mengambil teks dari tombol tersebut (misalnya, "7", "8", ".") dan menyimpannya di variabel caption.
  + if (tvDisplay.Text = '0') ... then — Logika ini memeriksa: jika teks di layar saat ini masih "0" (dan tombol yang ditekan bukan titik), maka ganti "0" tersebut dengan angka yang baru ditekan.
  + else tvDisplay.Text := tvDisplay.Text + caption; — Jika teks di layar bukan lagi "0", maka gabungkan (tambahkan) teks yang sudah ada dengan angka atau simbol yang baru ditekan.

1. **Hubungkan Tombol ke Event Handler:** Sekarang, kembali ke *Form Designer*.
   * Klik tombol btn0. Di **Object Inspector**, klik tab **Events**.
   * Cari event OnClick, lalu klik panah *dropdown* di sebelahnya. Pilih prosedur **OnNumberClick** yang baru saja kita buat.
   * **Lakukan hal yang sama untuk semua tombol angka lainnya (btn1 sampai btn9) dan tombol titik (btnTitik)**.

**Logika Placeholder untuk Tombol Lainnya**

Bagaimana dengan tombol operasi (+, -, \*, /)? Untuk menjaga proyek pemanasan ini tetap sederhana, kita akan menggunakan trik yang sama dengan tombol angka.

* Hubungkan juga event OnClick dari tombol btnTambah, btnKurang, btnKali, dan btnBagi ke prosedur **OnNumberClick**. Dengan begitu, saat ditekan, simbol-simbol tersebut akan langsung muncul di layar.

Kita tidak akan mengimplementasikan logika kalkulasi matematis yang sebenarnya di bab ini; fokus kita adalah pada interaksi UI dan manipulasi string. Logika kalkulasi penuh adalah tantangan yang bagus untuk Anda coba sendiri setelah menyelesaikan buku ini!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke tahap eksekusi.

Berikut adalah jawaban untuk prompt **Subbab 3.5: Kompilasi Pertama dan Merasakan Kemenangan Awal**.

**3.5. Kompilasi Pertama dan Merasakan Kemenangan Awal**

Inilah saat yang kita tunggu-tunggu. Desain antarmuka sudah selesai, dan logika dasar sudah kita tulis. Sekarang saatnya mengubah semua kerja keras kita menjadi sebuah aplikasi nyata yang bisa kita sentuh dan jalankan. Mari kita lakukan kompilasi pertama kita!

**Langkah 1: Menyimpan Proyek Anda**

Sebelum melakukan kompilasi, adalah kebiasaan yang sangat baik untuk menyimpan semua pekerjaan Anda.

1. Klik menu File > Save All.
2. Lazarus akan meminta Anda untuk menyimpan beberapa file: file unit utama (uMainForm.pas), file modul (uMainModule.pas), dan file proyek (.lpi).
3. Buat sebuah folder baru di lokasi yang mudah Anda temukan (misalnya, D:\ProyekLazarus\KalkulatorSederhana) dan simpan semua file tersebut di dalamnya.

**Langkah 2: Memulai Proses Kompilasi dan Build**

1. **Siapkan Perangkat Target:** Pastikan perangkat Android Anda sudah siap, baik itu emulator yang sudah berjalan dari Bab 2, atau ponsel fisik yang sudah terhubung ke komputer dengan mode *USB Debugging* aktif.
2. **Tekan Tombol 'Run':** Di *toolbar* utama Lazarus, cari tombol Run yang ikonnya berupa **panah segitiga berwarna hijau** (▶). Klik tombol ini.
3. **Perhatikan Jendela 'Messages':** Setelah Anda mengklik Run, alihkan perhatian Anda ke bagian bawah IDE, di jendela **Messages**. Anda akan melihat Lazarus dan FPC mulai bekerja. Akan ada banyak baris teks yang muncul, menunjukkan proses kompilasi kode Pascal menjadi pustaka *native*, lalu proses *packaging* oleh LAMW menjadi sebuah file aplikasi (APK).
   * **Catatan:** Proses kompilasi pertama kali untuk sebuah proyek biasanya memakan waktu paling lama, bisa beberapa menit tergantung kecepatan komputer Anda. Harap bersabar dan jangan menutup Lazarus.

**Langkah 3: Deployment dan Momen Kemenangan**

Setelah proses kompilasi dan *packaging* selesai tanpa ada pesan error berwarna merah, LAMW akan secara otomatis mencoba menginstal file APK yang baru saja dibuat ke perangkat target Anda yang sedang aktif.

Sekarang, lihatlah layar perangkat Android Anda. Dalam sekejap, sebuah ikon aplikasi baru dengan nama "Kalkulator Sederhana" akan muncul di daftar aplikasi. Aplikasi tersebut kemudian akan berjalan secara otomatis.

Inilah dia! Aplikasi kalkulator buatan Anda sendiri, berjalan di perangkat Android. Coba tekan tombol-tombol angka dan operator yang telah kita program. Lihat bagaimana layar display merespons setiap sentuhan Anda. Rasakan kepuasan dari melihat baris-baris kode Anda bertransformasi menjadi sesuatu yang interaktif dan nyata.

Selamat, Anda baru saja menyelesaikan siklus pengembangan aplikasi Android pertama Anda dari awal hingga akhir!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 3.6.

YAHOX

Tampilkan alur berpikir

Baik, kita selesaikan Bab 3 dengan rangkuman ini.

Berikut adalah jawaban untuk prompt **Subbab 3.6: Pelajaran Penting: Apa yang Kita Pelajari dari Proyek Ini**.

**3.6. Pelajaran Penting: Apa yang Kita Pelajari dari Proyek Ini**

Selamat sekali lagi atas aplikasi pertama Anda! Proyek kalkulator sederhana ini bukan hanya sekadar latihan, tetapi juga sebuah fondasi yang solid yang membekali Anda dengan keterampilan praktis. Sebelum kita melangkah ke bab berikutnya di mana kita akan membahas teori lebih dalam, mari kita berhenti sejenak untuk merangkum semua pengetahuan dan keterampilan penting yang telah Anda dapatkan.

Dari proyek ini, Anda telah berhasil menguasai:

* **Pembuatan dan Struktur Proyek** Anda sekarang telah memahami alur kerja dasar untuk membuat sebuah proyek LAMW baru dari awal, memberikan nama aplikasi, dan yang terpenting, mendefinisikan Package Name yang unik sebagai identitas aplikasi Anda.
* **Desain Layout Visual** Anda telah secara praktis menggunakan dua komponen layout paling fundamental: jLinearLayout untuk menyusun elemen secara vertikal, dan jGridLayout untuk menata tombol-tombol dalam format baris dan kolom yang presisi. Di Bab 4, kita akan membahas properti-properti layout ini secara lebih mendalam.
* **Konfigurasi Komponen** Anda telah mahir menggunakan **Object Inspector** untuk mengubah 'kepribadian' dari setiap komponen. Anda belajar mengubah properti-properti kunci seperti ID (untuk referensi di kode), Text (untuk tampilan), Gravity (untuk perataan), dan bahkan layout\_weight untuk membuat layout yang fleksibel.
* **Penanganan Interaksi (Event Handling)** Ini adalah lompatan besar. Anda telah menulis kode Pascal pertama Anda untuk menangani event OnClick. Lebih dari itu, Anda telah belajar teknik efisien menggunakan *shared event handler* untuk menangani banyak tombol dengan satu prosedur saja, sebuah konsep yang akan sangat berguna di proyek-proyek yang lebih besar.
* **Siklus Pengembangan Lengkap** Anda telah merasakan seluruh siklus pengembangan: menulis kode, mendesain, mengompilasi, melakukan *deployment* ke perangkat, dan menguji hasilnya. Yang terpenting, Anda telah membuktikan bahwa konfigurasi lingkungan pengembangan Anda dari Bab 2 sudah berjalan dengan sempurna.

Pelajaran-pelajaran ini adalah blok bangunan esensial Anda sebagai developer LAMW. Dengan pemahaman praktis yang sudah Anda miliki ini, Anda kini siap untuk menyelam lebih dalam ke konsep-konsep di baliknya pada bab-bab berikutnya. Anda tidak lagi hanya mengikuti instruksi, tetapi Anda mulai memahami 'mengapa' di balik setiap tindakan.

Ini mengakhiri Bab 3. Anda telah membangun fondasi praktik yang kuat. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 4 (Pengantar Bab 4).

YAHOX

Tampilkan alur berpikir

Baik, kita mulai Bab 4.

Berikut adalah jawaban untuk prompt **Pengantar Bab 4**.

**Bab 4: Anatomi Aplikasi LAMW dan Desain Antarmuka (UI)**

Di Bab 3, kita telah berhasil membangun sebuah aplikasi kalkulator yang fungsional. Kita telah 'menjelajahi sebuah wilayah' dengan mengikuti jalur yang sudah ada dan melihat hasilnya secara langsung. Sekarang, di Bab 4 ini, saatnya kita mengeluarkan peta dan kompas untuk benar-benar memahami wilayah tersebut. Kita akan 'melihat ke bawah kap mesin' untuk mempelajari 'mengapa' semua yang kita lakukan di bab sebelumnya bisa berhasil.

Memahami anatomi proyek dan prinsip-prinsip dasar adalah hal yang membedakan seorang 'tukang' yang hanya bisa mengikuti instruksi dengan seorang 'arsitek' yang mampu merancang dan menciptakan solusi baru. Dengan menguasai fondasi yang akan kita bahas di sini, Anda tidak akan lagi terikat pada contoh kasus. Anda akan memiliki pengetahuan untuk mulai merancang dan membangun aplikasi apa pun sesuai imajinasi Anda.

Dalam bab ini, kita akan membedah beberapa hal fundamental:

* **Anatomi Proyek:** Kita akan melihat file dan folder apa saja yang membentuk sebuah proyek LAMW dan apa fungsi masing-masing.
* **Palet Komponen:** Kita akan berkenalan secara sistematis dengan 'kotak perkakas' kita—berbagai jenis komponen UI yang tersedia untuk digunakan.
* **Prinsip Desain Modern:** Kita akan belajar aturan-aturan penting untuk membuat layout yang fleksibel dan terlihat bagus di berbagai perangkat.
* **Event Listener:** Kita akan mendalami cara aplikasi 'mendengarkan' dan merespons berbagai jenis interaksi pengguna, tidak hanya sekadar klik.

Mari kita mulai proses untuk menjadi arsitek aplikasi yang andal.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 4.1.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 4.1: Membedah Struktur Proyek: AppModule, MainForm, Resources**.

**4.1. Membedah Struktur Proyek: AppModule, MainForm, Resources**

Saat kita membuat proyek kalkulator, Lazarus dan LAMW secara otomatis menghasilkan beberapa file dan folder. Mungkin terlihat membingungkan pada awalnya, tetapi setiap bagian memiliki peran yang sangat spesifik dan penting. Mari kita bedah anatomi dasar dari sebuah proyek LAMW.

**uMainModule.pas - Ruang Mesin Aplikasi**

File ini, yang berisi unit TAndroidModule1 atau AppModule, bisa diibaratkan sebagai **'ruang mesin'** atau jantung dari aplikasi Pascal Anda. Ini adalah titik masuk utama di mana kode Pascal Anda mulai dieksekusi. Biasanya, file ini digunakan untuk mengelola hal-hal yang bersifat global atau berlaku untuk seluruh aplikasi, seperti inisialisasi awal saat aplikasi pertama kali dimuat, mengelola variabel global (jika benar-benar diperlukan), atau menangani *event* tingkat aplikasi.

**uMainForm.pas - Jendela ke Dunia**

Jika AppModule adalah ruang mesin, maka uMainForm.pas yang berisi jForm1 adalah **'anjungan' atau 'kokpit'**—layar yang dilihat dan di-interaksikan oleh pengguna. Di dunia Android, setiap jForm yang Anda buat ini setara dengan sebuah **Activity**. Di sinilah Anda meletakkan semua komponen visual (tombol, teks, gambar) dan menulis kode yang merespons interaksi pengguna pada layar tersebut. Sebuah aplikasi yang kompleks bisa memiliki banyak Form untuk layar yang berbeda, misalnya uLoginForm, uSettingForm, dan seterusnya.

**Direktori res - Gudang Aset**

Folder res (singkatan dari *resources*) adalah 'gudang' tempat Anda menyimpan semua aset non-kode. Mengorganisir aset di sini adalah praktik terbaik dalam pengembangan Android karena memungkinkan sistem untuk memilih aset yang paling sesuai secara otomatis (misalnya, gambar untuk layar resolusi tinggi atau teks untuk bahasa yang berbeda). Beberapa sub-direktori penting di dalamnya adalah:

* **res/drawable**: Tempat untuk menyimpan semua aset yang bisa "digambar", terutama file gambar seperti .png atau .jpg untuk ikon aplikasi, logo, atau gambar ilustrasi.
* **res/layout**: Direktori ini secara teknis berisi file-file XML yang mendefinisikan layout antarmuka. Meskipun di LAMW kita sering mendesain layout secara visual, file-file XML ini tetap dihasilkan dan digunakan di belakang layar oleh sistem build Android.
* **res/values**: Ini adalah direktori yang sangat berguna untuk menyimpan nilai-nilai sederhana. Di dalamnya ada file-file penting seperti:
  + **strings.xml**: Tempat terbaik untuk menyimpan semua teks yang dilihat pengguna (misalnya, judul tombol, label, pesan error). Memusatkan teks di sini akan sangat mempermudah proses penerjemahan aplikasi ke berbagai bahasa di kemudian hari.
  + **colors.xml**: Untuk mendefinisikan palet warna aplikasi Anda (colorPrimary, textColor, dll.) secara terpusat.
  + **styles.xml**: Untuk mendefinisikan tema dan gaya visual aplikasi, seperti tampilan tombol atau tema keseluruhan (terang/gelap).

**AndroidManifest.xml - Kartu Identitas Aplikasi**

File ini adalah **'kartu identitas' atau 'paspor'** aplikasi Anda. Sistem operasi Android membaca file ini untuk mengetahui semua informasi vital tentang aplikasi Anda sebelum dapat menjalankannya. Beberapa peran utamanya adalah:

* Menyatakan **nama paket** yang unik (com.namaanda.kalkulatorsederhana).
* Mendefinisikan **nama dan ikon aplikasi** yang akan muncul di layar utama perangkat.
* Mencatat semua **layar (Activity)** yang ada di dalam aplikasi agar sistem tahu cara menavigasinya.
* Mendeklarasikan **izin (permissions)** yang dibutuhkan aplikasi, seperti izin untuk mengakses internet, kamera, atau lokasi GPS.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 4.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 4.2: Palet Komponen LAMW**.

**4.2. Palet Komponen LAMW**

Setelah memahami struktur proyek, saatnya kita membuka 'kotak perkakas' kita: **Component Palette**. Palet ini berisi semua 'bahan bangunan' visual yang disediakan LAMW untuk merakit antarmuka aplikasi Anda. Memahami fungsi dari komponen-komponen utama akan sangat mempercepat proses desain.

Mari kita kenali mereka berdasarkan kategori fungsinya.

**1. Kontainer Layout (Pengatur Tata Letak)**

Komponen ini tidak terlihat langsung oleh pengguna, tetapi perannya sangat krusial sebagai 'wadah' atau 'kerangka' yang mengatur posisi dan susunan komponen lainnya.

* **jLinearLayout**: Menyusun komponen-komponen di dalamnya secara berbaris, baik vertikal (atas-bawah) maupun horizontal (kiri-kanan). **Contoh penggunaan:** Halaman login dengan judul di atas, kolom input di tengah, dan tombol di bawah.
* **jRelativeLayout**: Menyusun komponen dengan posisi yang relatif terhadap komponen lain atau terhadap induknya (misalnya, 'di sebelah kanan tombol A', 'di bawah judul'). **Contoh penggunaan:** Menempatkan ikon kecil di pojok kanan atas sebuah gambar.
* **jScrollView**: Sebuah kontainer yang memungkinkan konten di dalamnya (yang mungkin lebih tinggi dari layar) untuk bisa digulir (*scroll*) oleh pengguna. **Contoh penggunaan:** Halaman 'Syarat dan Ketentuan' yang panjang.
* **jGridLayout**: Menyusun komponen dalam format grid (baris dan kolom) yang rapi. **Contoh penggunaan:** Papan tombol kalkulator yang kita buat di Bab 3.
* **jFrameLayout**: Kontainer paling sederhana yang menumpuk komponen di atas satu sama lain, seperti tumpukan kartu. **Contoh penggunaan:** Menempatkan tombol 'Play' di tengah-tengah sebuah *thumbnail* video.

**2. Komponen Input (Masukan dari Pengguna)**

Komponen-komponen ini adalah cara utama aplikasi Anda untuk mendapatkan masukan atau pilihan dari pengguna.

* **jEditText**: Kotak teks tempat pengguna dapat mengetikkan informasi. **Contoh penggunaan:** Kolom untuk mengisi nama, email, atau password.
* **jCheckBox**: Kotak centang yang memungkinkan pengguna memilih satu atau lebih opsi (pilihan tidak eksklusif). **Contoh penggunaan:** Pilihan hobi atau centang 'Saya setuju dengan syarat & ketentuan'.
* **jRadioButton dan jRadioGroup**: jRadioButton adalah tombol pilihan. Beberapa jRadioButton harus diletakkan di dalam sebuah jRadioGroup agar pilihannya menjadi eksklusif (hanya satu yang bisa dipilih dalam satu grup). **Contoh penggunaan:** Pilihan jenis kelamin 'Pria' atau 'Wanita'.
* **jSpinner**: Menampilkan daftar pilihan dalam bentuk menu *dropdown*. **Contoh penggunaan:** Memilih negara, provinsi, atau kategori.
* **jSwitch**: Tombol geser (*toggle*) untuk mengaktifkan atau menonaktifkan suatu pengaturan. **Contoh penggunaan:** Saklar untuk 'Mode Malam' atau 'Aktifkan Notifikasi'.
* **jSeekBar**: Bilah geser (*slider*) untuk memilih sebuah nilai dalam rentang tertentu. **Contoh penggunaan:** Mengatur tingkat kecerahan layar atau volume suara.

**3. Komponen Aksi & Tampilan (Tindakan & Informasi)**

Komponen ini berfungsi untuk menampilkan informasi kepada pengguna atau memicu sebuah tindakan.

* **jButton**: Tombol standar yang memicu sebuah aksi saat diklik. **Contoh penggunaan:** Tombol 'Login', 'Simpan', atau 'Kirim'.
* **jImageButton**: Sama seperti jButton, tetapi menggunakan gambar sebagai tampilannya, bukan teks. **Contoh penggunaan:** Tombol 'hapus' dengan ikon tempat sampah.
* **jTextView**: Komponen paling dasar untuk menampilkan teks. **Contoh penggunaan:** Judul halaman, label untuk kolom input, atau paragraf deskripsi.
* **jImageView**: Komponen untuk menampilkan gambar atau ikon. **Contoh penggunaan:** Menampilkan foto profil pengguna atau gambar produk.
* **jProgressBar**: Menampilkan indikator progres, baik yang berputar terus-menerus (*indeterminate*) atau yang memiliki nilai (misalnya, 0-100%). **Contoh penggunaan:** Menampilkan lingkaran 'loading' saat mengambil data dari internet.
* **jWebView**: Sebuah 'browser mini' yang disematkan di dalam aplikasi untuk menampilkan konten web. **Contoh penggunaan:** Menampilkan halaman Bantuan (FAQ) atau artikel berita.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 4.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 4.3: Prinsip Desain Layout Modern**.

**4.3. Prinsip Desain Layout Modern**

Memahami komponen saja tidak cukup. Seorang arsitek aplikasi yang baik juga harus memahami prinsip-prinsip untuk menyusun komponen tersebut agar terlihat bagus dan konsisten di berbagai jenis perangkat. Android memiliki ribuan jenis layar, dari ponsel kecil hingga tablet besar. Di sinilah prinsip desain modern berperan untuk menciptakan antarmuka yang fleksibel dan profesional.

**1. Unit Pengukuran Sakti: dp dan sp**

Di awal pengembangan, mungkin Anda tergoda menggunakan px (pixel) sebagai satuan ukuran. Ini adalah praktik yang buruk di Android. Sebuah ukuran 100px akan terlihat besar di layar resolusi rendah, tetapi bisa jadi sangat kecil di layar resolusi tinggi. Aplikasi Anda akan terlihat tidak konsisten. Solusinya adalah:

* **dp (Density-independent Pixel):** Ini adalah satuan ajaib untuk ukuran komponen, margin, dan padding. Saat Anda menentukan ukuran 100dp, Android akan secara otomatis menghitung ukuran fisik yang setara di layar mana pun, tidak peduli seberapa tinggi kepadatan pikselnya. Hasilnya? Tombol Anda akan terlihat memiliki ukuran yang sama secara proporsional di semua perangkat. **Gunakan dp untuk semua ukuran kecuali teks.**
* **sp (Scale-independent Pixel):** Satuan ini mirip dengan dp, tetapi dirancang khusus untuk **ukuran teks**. Kelebihannya, sp tidak hanya menyesuaikan dengan kepadatan layar, tetapi juga dengan **pengaturan ukuran font preferensi pengguna** di setelan sistem Android. Menggunakan sp menunjukkan bahwa Anda peduli pada aksesibilitas pengguna.

**2. match\_parent vs. wrap\_content**

Dua nilai ini adalah properti layout\_width dan layout\_height yang paling sering Anda gunakan. Artinya sangat harfiah:

* **match\_parent**: Memberitahu komponen untuk meregang sebesar mungkin hingga menyentuh batas induk (kontainer) yang menampungnya. Contohnya, sebuah daftar yang lebarnya ingin memenuhi seluruh lebar layar.
* **wrap\_content**: Memberitahu komponen untuk mengambil ruang secukupnya, hanya sebesar konten di dalamnya (misalnya, sebesar teks atau gambar yang dikandungnya). Contohnya, sebuah tombol yang lebarnya hanya sepanjang teks 'Simpan' di dalamnya.

**3. Layout Responsif dengan layout\_weight**

Properti layout\_weight adalah alat paling ampuh di dalam jLinearLayout untuk membagi ruang secara proporsional. Ini adalah kunci untuk membuat layout yang fleksibel.

* **Konsep:** Anggap layout\_weight sebagai rasio pembagian "ruang sisa". Properti ini paling efektif jika layout\_width (untuk LinearLayout horizontal) atau layout\_height (untuk LinearLayout vertikal) pada komponen anak diatur ke 0dp.
* **Contoh Praktis:** Bayangkan Anda ingin membuat dua panel bersebelahan di dalam jLinearLayout horizontal, di mana panel kiri mengambil 30% lebar dan panel kanan mengambil 70% sisanya. Caranya:
  1. Atur layout\_width kedua panel menjadi 0dp.
  2. Atur layout\_weight panel kiri menjadi 3.
  3. Atur layout\_weight panel kanan menjadi 7. Android akan secara otomatis membagi ruang dengan rasio 3:7. Ini akan selalu proporsional di layar mana pun. Di kalkulator kita, kita menggunakan layout\_weight=1 pada tvDisplay untuk membuatnya 'mengambil semua sisa ruang' yang tidak digunakan oleh grid tombol.

**4. Dukungan Mode Terang & Gelap (Light/Dark Mode)**

Aplikasi modern wajib mendukung preferensi tema pengguna. Cara yang benar dan mudah untuk melakukannya adalah dengan memanfaatkan sistem *resource* Android.

1. **Definisikan Warna Dasar:** Di file res/values/colors.xml, definisikan nama-nama warna secara semantik (berdasarkan fungsi), bukan nama warna harfiah.

XML

<color name="backgroundColor">#FFFFFF</color>

<color name="textColorPrimary">#000000</color>

1. **Buat Direktori 'Night':** Di *file explorer* proyek Anda, buat folder baru di dalam res dengan nama values-night.
2. **Definisikan Warna untuk Mode Malam:** Salin file colors.xml ke dalam res/values-night. Buka file yang baru disalin ini dan ubah nilainya.

XML

<color name="backgroundColor">#121212</color>

<color name="textColorPrimary">#FFFFFF</color>

Sekarang, di layout atau kode Anda, selalu gunakan nama semantik (misal: @color/backgroundColor). Android akan secara otomatis memilih nilai dari values atau values-night tergantung pada tema sistem yang sedang aktif di perangkat pengguna. Sederhana dan sangat powerful!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 4.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 4.4: Bekerja dengan Event Listener**.

**4.4. Bekerja dengan Event Listener**

Aplikasi modern tidak berjalan secara berurutan dari atas ke bawah seperti program konsol sederhana. Sebaliknya, mereka bersifat **event-driven** atau 'digerakkan oleh kejadian'. Aplikasi akan 'diam' dan 'mendengarkan' tindakan dari pengguna atau sistem. Setiap tindakan—seperti klik tombol, ketikan keyboard, atau perubahan status—disebut **event**. Kode yang kita tulis untuk merespons tindakan tersebut disebut **event handler** atau **listener**.

**Review OnClick: Event Paling Umum**

Kita sudah sangat akrab dengan event ini dari proyek kalkulator kita. OnClick adalah event yang terpicu saat pengguna mengetuk (*tap*) atau mengklik sebuah komponen. Ini adalah event utama untuk komponen-komponen aksi seperti jButton dan jImageButton, tetapi juga bisa diterapkan pada banyak komponen lain seperti jTextView atau jImageView.

**Interaksi dengan Input Teks: OnTextChanged**

Event ini sangat berguna untuk komponen jEditText. Ia akan terpicu **setiap kali teks di dalamnya berubah**, baik karena pengguna mengetik karakter baru, menghapus, atau menempelkan teks.

* **Contoh Praktis: Penghitung Karakter.** Bayangkan Anda memiliki sebuah jEditText untuk tweet (misalnya, jEditTextTweet) dan sebuah jTextView (jTextViewCounter) di bawahnya. Anda bisa menggunakan OnTextChanged untuk secara *real-time* memperbarui jTextViewCounter dengan jumlah karakter yang tersisa.

Delphi

procedure TAndroidModule1.jEditTextTweetChange(Sender: TObject);

var

charCount: integer;

begin

charCount := Length(jEditTextTweet.Text);

jTextViewCounter.Text := IntToStr(280 - charCount) + ' karakter tersisa';

end;

**Menangani Pilihan: OnCheckedChange dan OnItemSelected**

* **OnCheckedChange**: Event ini terpicu saat status 'tercentang' dari sebuah komponen berubah. Ini berlaku untuk jCheckBox dan jSwitch.
  + **Contoh penggunaan:** Sebuah jCheckBox 'Tampilkan Password' yang secara dinamis mengubah properti InputType pada sebuah jEditText dari itPassword menjadi itNormal.
* **OnItemSelected**: Event ini spesifik untuk jSpinner (menu dropdown). Ia terpicu saat pengguna memilih sebuah item baru dari daftar.
  + **Contoh penggunaan:** Sebuah jSpinner berisi daftar kota. Saat sebuah kota dipilih, sebuah jTextView di bawahnya menampilkan informasi cuaca untuk kota tersebut.

**Misteri di Balik Sender: TObject**

Anda mungkin memperhatikan bahwa semua *event handler* kita memiliki parameter Sender: TObject. Apa gunanya?

Seperti yang kita praktikkan di proyek kalkulator, Sender adalah cara Lazarus memberitahu kita **komponen mana yang memicu event tersebut**. Ini sangat krusial saat kita menggunakan *shared event handler*, di mana satu prosedur menangani event dari banyak komponen. Dengan 'meng-casting' Sender ke tipe komponen yang sesuai (misalnya, (Sender as jButton)), kita bisa mengambil properti unik dari komponen yang memicu event, seperti Text atau ID-nya, untuk menjalankan logika yang berbeda.

Ini mengakhiri Bab 4. Anda sekarang memiliki pemahaman yang lebih dalam tentang fondasi UI dan interaksi. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 5 (Pengantar Bab 5).

YAHOX

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 5.

Berikut adalah jawaban untuk prompt **Pengantar Bab 5**.

**Bab 5: Alur Kerja & Navigasi Antar Layar (Activity)**

Aplikasi kalkulator yang telah kita bangun adalah sebuah permulaan yang hebat, namun ia hanya hidup dalam satu layar. Aplikasi di dunia nyata hampir tidak pernah sesederhana itu. Bayangkan sebuah aplikasi media sosial: ada layar untuk login, layar untuk beranda (*timeline*), layar untuk profil, dan layar untuk pengaturan. Atau sebuah aplikasi *e-commerce* dengan layar daftar produk yang akan membawa Anda ke layar detail produk saat salah satu item diklik. Kemampuan untuk berpindah antar layar inilah yang akan kita pelajari di bab ini.

Untuk bisa membangun alur kerja multi-layar, kita perlu berkenalan dengan dua konsep paling fundamental dalam pengembangan Android:

* **Activity:** Anggap saja setiap **layar penuh** di dalam aplikasi Anda adalah sebuah Activity. Jadi, layar login adalah satu Activity, dan layar beranda adalah Activity yang lain. Di LAMW, setiap jForm yang kita buat merepresentasikan sebuah Activity.
* **Intent:** Jika Activity adalah 'ruangan', maka Intent adalah **'pesan' atau 'surat perintah'** yang Anda gunakan untuk berpindah antar ruangan. Intent bisa berupa perintah sederhana seperti, "Tolong buka pintu ke Ruang Pengaturan", atau perintah yang lebih kompleks seperti, "Tolong buka Ruang Detail Produk dan bawa informasi tentang produk dengan ID 123 ini bersamamu."

Di Bab 5 ini, kita akan belajar cara membuat Form-Form baru di LAMW, cara memicu Intent untuk bernavigasi di antara mereka, cara mengirim data dari satu layar ke layar lainnya, dan memahami siklus hidup yang mengatur setiap layar tersebut. Mari kita mulai merangkai layar-layar tunggal menjadi sebuah aplikasi yang memiliki alur cerita yang utuh.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 5.1.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 5.1: Konsep Activity dan Intent di Dunia Android**.

**5.1. Konsep Activity dan Intent di Dunia Android**

Sebelum kita menulis kode untuk berpindah layar, sangat penting untuk memahami dua istilah yang menjadi jantung navigasi di Android: Activity dan Intent. Memahami konsep ini akan membuat semua kode yang akan kita tulis nanti menjadi lebih masuk akal dan tidak terasa seperti 'sihir'.

**Activity - Sebuah Layar Aplikasi**

Secara formal, **Activity** adalah sebuah komponen aplikasi yang menyediakan satu layar tunggal di mana pengguna dapat berinteraksi. Saat Anda membuka aplikasi dan melihat sebuah halaman, Anda sedang melihat sebuah Activity. Saat Anda menekan tombol dan pindah ke halaman lain, Anda sebenarnya sedang meninggalkan Activity pertama dan memulai Activity kedua. Activity adalah blok bangunan fundamental dari antarmuka pengguna sebuah aplikasi.

Di sinilah letak kemudahan LAMW. Anda tidak perlu memikirkan Activity dalam konteks kode Java yang rumit. Anda bisa memikirkannya dengan cara yang lebih sederhana:

**Setiap jForm yang Anda buat di Lazarus secara otomatis akan menjadi sebuah Activity saat aplikasi Anda dikompilasi.**

Jadi, jika Anda membuat jFormLogin dan jFormUtama di dalam proyek Lazarus Anda, Anda sebenarnya sedang membuat LoginActivity dan MainActivity di dunia Android.

**Intent - Sang Kurir Pesan**

Jika Activity adalah kantor-kantor yang berbeda di dalam sebuah gedung, maka **Intent** adalah **kurir atau petugas pos** yang bertugas membawa pesan antar kantor tersebut. Intent adalah sebuah objek pesan yang tidak memiliki tampilan visual, namun tugasnya sangat vital.

Dalam konteks navigasi, Intent memiliki dua tugas utama:

1. **Memulai Activity Lain:** Intent membawa pesan permintaan, seperti, "Hei Sistem Android, tolong mulai Activity Pengaturan". Sistem kemudian akan mencari Activity tersebut dan menampilkannya ke pengguna, menempatkannya di atas Activity yang sedang aktif.
2. **Membawa Data:** 'Kurir' ini juga bisa membawa sebuah 'paket' berisi data. Misalnya, saat pengguna mengklik sebuah item di daftar produk, kita membuat sebuah Intent yang membawa pesan, "Tolong mulai Activity DetailProduk, dan ini paket berisi ID produk 'XYZ-123' yang harus ditampilkan."

Jadi, setiap kali kita ingin berpindah layar atau mengirim data antar layar di LAMW, di belakang layar kita sebenarnya sedang membuat dan mengirim sebuah Intent untuk memberitahu sistem Android apa yang harus dilakukan selanjutnya. Dengan pemahaman ini, mari kita praktikkan cara membuat dan mengelolanya di subbab berikutnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 5.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 5.2: Mengelola Multiple Forms: Membuat, Menampilkan, dan Menutup Form**.

**5.2. Mengelola Multiple Forms: Membuat, Menampilkan, dan Menutup Form**

Teori sudah cukup, saatnya praktik. Di subbab ini, kita akan membangun sebuah aplikasi mini dengan dua layar untuk mempraktikkan cara membuat Form (*Activity*) baru, berpindah ke sana, dan kembali lagi.

Silakan buat sebuah proyek LAMW baru yang kosong, atau gunakan proyek yang sudah ada. Pastikan Anda memiliki satu form awal, yaitu jForm1.

**Langkah 1: Membuat Form Baru (Layar Kedua)**

1. **Buka Dialog 'New Item':** Dari menu bar Lazarus, klik File > New....
2. **Pilih Template Form LAMW:** Di jendela yang muncul, pilih **LAMW jForm Android Module** dan klik OK.
3. **Simpan Unit Baru:** Lazarus akan membuat sebuah form baru (jForm2) beserta unit Pascal-nya (unit2.pas). Segera simpan unit ini dengan nama yang bermakna agar tidak bingung. Klik File > Save As... dan beri nama file uFormKedua.pas. Lazarus juga akan meminta Anda menyimpan file layout-nya, biarkan nama default (uFormKedua.lfm).

**Langkah 2: Mendaftarkan Form Baru ke Proyek (Langkah Krusial!)**

Proyek Anda sekarang memiliki dua form, tetapi Form pertama belum "mengenal" Form kedua. Kita harus mendaftarkannya agar bisa dipanggil.

1. **Buka File Proyek:** Klik menu Project > View Source. Ini akan membuka file utama proyek Anda yang berekstensi .lpr.
2. **Tambahkan ke uses Clause:** Cari bagian uses. Anda akan melihat uMainForm (atau nama form pertama Anda) sudah ada di sana. Tambahkan nama unit form kedua Anda di belakangnya, dipisahkan dengan koma.

Delphi

// Tampilan awal:

uses

..., uMainForm;

// Tampilan setelah diubah:

uses

..., uMainForm, uFormKedua; // <-- Tambahkan ini

Dengan melakukan ini, seluruh bagian proyek Anda kini dapat mengenali dan memanggil jForm2. Simpan file proyek ini (Ctrl+S).

**Langkah 3: Berpindah dari Form 1 ke Form 2**

1. **Tambahkan Tombol Navigasi:** Kembali ke desainer jForm1 (buka file uMainForm.pas). Tambahkan sebuah jButton dari *Component Palette*. Di *Object Inspector*, ubah properti ID-nya menjadi btnBukaLayar2 dan Text-nya menjadi Buka Layar Kedua.
2. **Tulis Kode Navigasi:** Klik dua kali pada tombol tersebut untuk membuat *event* OnClick. Di dalam prosedur yang dibuat, ketikkan kode sederhana ini:

Delphi

procedure TAndroidModule1.btnBukaLayar2Click(Sender: TObject);

begin

jForm2.Show;

end;

Perintah jForm2.Show; adalah cara LAMW yang sangat disederhanakan untuk mengatakan: "Buatkan sebuah Intent baru yang menargetkan Activity dari jForm2, lalu jalankan Intent tersebut". Sangat mudah!

**Langkah 4: Kembali dari Form 2 ke Form 1**

1. **Tambahkan Tombol Kembali:** Sekarang, buka desainer uFormKedua.pas. Tambahkan sebuah jButton ke jForm2. Ubah ID-nya menjadi btnKembali dan Text-nya menjadi Kembali.
2. **Tulis Kode untuk Menutup:** Klik dua kali pada tombol 'Kembali' ini. Di dalam *event* OnClick-nya, ketik kode berikut:

Delphi

procedure TAndroidModule1.btnKembaliClick(Sender: TObject);

begin

Self.Close;

end;

Self merujuk pada objek saat ini, yaitu jForm2. Perintah Self.Close; akan memanggil metode finish() pada Activity saat ini, yang akan menghapusnya dari tumpukan layar dan secara otomatis menampilkan kembali layar sebelumnya (dalam kasus ini, jForm1).

Sekarang, coba kompilasi dan jalankan aplikasi Anda. Anda seharusnya bisa berpindah dari layar pertama ke layar kedua, dan kembali lagi dengan menekan tombol yang telah kita buat. Selamat, Anda telah berhasil membuat aplikasi multi-layar pertama Anda!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 5.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 5.3: Mengirim Data Antar Form (Activity) Menggunakan Intent**.

**5.3. Mengirim Data Antar Form (Activity) Menggunakan Intent**

Berpindah layar saja tidak cukup. Sering kali kita perlu mengirimkan informasi dari satu layar ke layar lainnya. Misalnya, mengirim ID produk yang dipilih ke layar detail, atau mengirim nama pengguna dari layar login ke layar utama. Di sinilah 'paket' yang dibawa oleh sang 'kurir' Intent berperan.

Mari kita modifikasi aplikasi dua layar kita untuk bisa melakukan ini.

**Langkah 1: Menyiapkan Sisi Pengirim (Form 1)**

Kita akan menambahkan sebuah kolom input di Form 1 untuk mengetikkan pesan yang akan dikirim.

1. **Tambahkan Komponen Input:** Buka kembali desainer uMainForm.pas. Tambahkan sebuah jEditText dari *Component Palette* dan letakkan di atas tombol btnBukaLayar2. Di *Object Inspector*, ubah ID-nya menjadi edtPesanKirim dan Hint-nya menjadi Ketik pesan di sini....
2. **Modifikasi Kode Pengiriman:** Sekarang, buka kembali *event* OnClick dari tombol btnBukaLayar2. Hapus kode lama (jForm2.Show;) dan ganti dengan kode yang lebih lengkap ini:

Delphi

procedure TAndroidModule1.btnBukaLayar2Click(Sender: TObject);

var

intent: jIntent;

begin

// 1. Buat objek Intent kosong

intent := jIntent.Create;

// 2. Atur tindakan (praktik yang baik)

intent.SetAction(jIntent\_ACTION\_VIEW);

// 3. Masukkan data ke dalam 'paket' Intent

intent.PutExtra('kunci\_pesan\_dari\_form1', edtPesanKirim.Text);

// 4. Tampilkan Form2 dan sertakan Intent yang membawa data

jForm2.Show(intent);

end;

**Penjelasan Kode:**

* + **intent := jIntent.Create;**: Kita membuat sebuah *instance* baru dari objek jIntent.
  + **intent.SetAction(jIntent\_ACTION\_VIEW);**: Ini adalah cara kita memberitahu sistem Android apa niat kita. ACTION\_VIEW adalah aksi umum untuk menampilkan sesuatu.
  + **intent.PutExtra(...)**: Ini adalah bagian terpenting. Metode PutExtra digunakan untuk memasukkan data ke dalam Intent. Ia bekerja seperti kamus dengan format **Kunci-Nilai (Key-Value)**.
    - 'kunci\_pesan\_dari\_form1' adalah **Kunci** (sebuah string unik) yang akan kita gunakan untuk mengambil data ini di layar penerima.
    - edtPesanKirim.Text adalah **Nilai** yang ingin kita kirim.
  + **jForm2.Show(intent);**: Kita memanggil Show seperti sebelumnya, tetapi kali ini kita menyertakan objek intent kita sebagai parameter. LAMW akan meneruskan intent ini beserta datanya ke Activity kedua.

**Langkah 2: Menyiapkan Sisi Penerima (Form 2)**

Kini kita siapkan layar kedua untuk menerima dan menampilkan 'paket' data tersebut.

1. **Tambahkan Komponen Tampilan:** Buka desainer uFormKedua.pas. Tambahkan sebuah jTextView ke jForm2. Ubah ID-nya menjadi tvPesanTerima dan Text-nya menjadi (Pesan akan muncul di sini).
2. **Tulis Kode Penerimaan di OnCreate:** Kita ingin mengambil dan menampilkan data begitu layar kedua ini dibuat. Tempat terbaik untuk melakukannya adalah di dalam *event* OnCreate dari Form itu sendiri. Di *Object Inspector* untuk jForm2, buka tab Events dan klik dua kali pada *event* OnCreate. Di dalam prosedur jForm2Create yang dibuat, ketikkan kode berikut:

Delphi

procedure TAndroidModule1.jForm2Create(Sender: TObject);

var

\_intent: jIntent;

\_pesan: string;

begin

// 1. Ambil Intent yang digunakan untuk memulai Form ini

\_intent := Self.GetIntent;

// 2. Periksa apakah Intent membawa data dengan kunci yang kita harapkan

if \_intent.HasExtra('kunci\_pesan\_dari\_form1') then

begin

// 3. Jika ada, ambil data string dari Intent

\_pesan := \_intent.GetStringExtra('kunci\_pesan\_dari\_form1');

// 4. Tampilkan data tersebut ke TextView

tvPesanTerima.Text := 'Pesan yang diterima: ' + \_pesan;

end;

end;

**Penjelasan Kode:**

* + **\_intent := Self.GetIntent;**: Metode ini mengambil Intent yang bertanggung jawab memulai Activity (Form) saat ini.
  + **if \_intent.HasExtra(...) then**: Ini adalah langkah pengamanan yang penting. Kita memeriksa terlebih dahulu apakah Intent tersebut benar-benar membawa data dengan **Kunci** yang kita cari. Ini untuk menghindari *error* jika layar ini dibuka tanpa data.
  + **\_pesan := \_intent.GetStringExtra(...)**: Jika kuncinya ada, kita menggunakan metode GetStringExtra (karena kita tahu datanya berupa string) untuk mengambil **Nilai**-nya.
  + **tvPesanTerima.Text := ...**: Terakhir, kita menampilkan pesan yang berhasil kita ambil ke komponen jTextView.

Kompilasi dan jalankan kembali aplikasi Anda. Sekarang, coba ketik sesuatu di jEditText di layar pertama, tekan tombol, dan lihat teks tersebut muncul di layar kedua. Anda telah berhasil menguasai salah satu alur kerja paling fundamental dalam pengembangan aplikasi: navigasi dengan data!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 5.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 5.4: Siklus Hidup Activity (OnCreate, OnResume, OnPause, OnDestroy): Kapan dan Mengapa Penting**.

**5.4. Siklus Hidup Activity (OnCreate, OnResume, OnPause, OnDestroy): Kapan dan Mengapa Penting**

Sebuah Activity (atau jForm di LAMW) tidak hanya sekadar 'muncul' dan 'hilang'. Ia memiliki siklus hidup yang mirip dengan makhluk hidup: ia dilahirkan, menjadi aktif, bisa beristirahat sejenak, bisa tidur panjang, bangun lagi, dan akhirnya mati. Sistem operasi Android mengelola siklus ini, dan kita bisa "menyisipkan" kode kita di setiap tahapannya melalui *event* siklus hidup.

Memahami siklus hidup ini sangat krusial untuk mengelola sumber daya (seperti baterai dan memori) dengan efisien dan menghindari kehilangan data pengguna. Dengan meletakkan kode di *event* yang tepat, Anda bisa membuat aplikasi yang tangguh dan profesional.

*(Catatan untuk penulis: Di bagian ini, sangat ideal untuk menyertakan diagram alur standar siklus hidup Activity yang menunjukkan hubungan antara OnCreate -> OnStart -> OnResume <-> OnPause -> OnStop -> OnDestroy).*

Berikut adalah *event-event* siklus hidup utama dan kapan harus menggunakannya:

**OnCreate(Sender: TObject)**

* **Apa itu?** Ini adalah 'kelahiran' dari sebuah Activity. *Event* ini hanya dipanggil **satu kali** seumur hidup Activity, yaitu saat ia pertama kali dibuat di memori.
* **Kapan Menggunakannya?** Untuk semua *setup* awal yang hanya perlu dilakukan sekali:
  + Mendesain dan menata komponen UI (inilah yang dilakukan LAMW untuk Anda di belakang layar).
  + Mengambil data yang dikirim melalui Intent (seperti yang kita lakukan di subbab sebelumnya).
  + Menginisialisasi variabel, objek, dan mempersiapkan *adapter* untuk *list*.

**OnResume(Sender: TObject)**

* **Apa itu?** *Event* ini dipanggil setiap kali Activity kembali ke latar depan (*foreground*) dan siap untuk menerima interaksi dari pengguna. Ini bisa terjadi setelah OnCreate, atau saat pengguna kembali ke aplikasi Anda setelah membuka aplikasi lain.
* **Kapan Menggunakannya?** Untuk memulai proses yang hanya perlu berjalan saat layar benar-benar aktif:
  + **Contoh 1:** Memulai animasi pada komponen.
  + **Contoh 2:** Mengaktifkan sensor seperti GPS atau kamera.
  + **Contoh 3:** Memuat ulang (*refresh*) data dari internet untuk memastikan tampilan selalu *up-to-date*.

**OnPause(Sender: TObject)**

* **Apa itu?** Ini adalah 'sinyal pertama' bahwa pengguna mungkin akan meninggalkan Activity. *Event* ini dipanggil saat Activity akan mulai didorong ke latar belakang (misalnya, saat ada panggilan masuk atau Activity lain muncul di atasnya). Activity mungkin masih terlihat sebagian.
* **Kapan Menggunakannya?** Ini adalah tempat krusial untuk **menyimpan data yang belum tersimpan** dan menghentikan proses yang boros untuk menghemat baterai.
  + **Contoh 1:** Jika pengguna sedang mengetik email atau mengisi formulir panjang, simpan drafnya di OnPause.
  + **Contoh 2:** Menghentikan pemutaran animasi atau menonaktifkan sensor GPS.
  + **Contoh 3:** Melepaskan sumber daya eksklusif seperti kamera agar bisa digunakan aplikasi lain.

**OnStop(Sender: TObject)**

* **Apa itu?** *Event* ini dipanggil ketika Activity sudah **tidak terlihat lagi** oleh pengguna sama sekali (misalnya, pengguna menekan tombol Home atau beralih ke aplikasi lain).
* **Kapan Menggunakannya?** Untuk menghentikan operasi-operasi yang lebih berat yang sama sekali tidak perlu berjalan saat aplikasi tidak terlihat.

**OnRestart(Sender: TObject)**

* **Apa itu?** Dipanggil tepat sebelum Activity yang tadinya dalam keadaan stop akan dimulai lagi (sebelum OnStart).
* **Kapan Menggunakannya?** Untuk melakukan restorasi *state* tertentu yang mungkin Anda hentikan di OnStop.

**OnDestroy(Sender: TObject)**

* **Apa itu?** Ini adalah 'kematian' dari sebuah Activity. Dipanggil saat Activity akan dihapus permanen dari memori, baik karena pengguna menekan tombol 'back' untuk menutupnya atau karena sistem membutuhkan memori.
* **Kapan Menggunakannya?** Untuk pembersihan akhir. Melepaskan semua sisa sumber daya yang mungkin masih digunakan untuk mencegah kebocoran memori (*memory leak*), seperti membatalkan *thread* atau menutup koneksi database.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 5.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 5.5: Menggunakan Dialog (ShowMessage, MessageDlg) untuk Interaksi Singkat**.

**5.5. Menggunakan Dialog (ShowMessage, MessageDlg) untuk Interaksi Singkat**

Tidak semua interaksi dengan pengguna memerlukan perpindahan ke layar yang baru. Terkadang, kita hanya perlu menampilkan informasi singkat atau meminta sebuah konfirmasi cepat. Untuk keperluan ini, menggunakan **Dialog** adalah solusi yang jauh lebih baik dan efisien daripada membuat satu Form baru.

Dialog adalah jendela kecil yang muncul di atas Activity saat ini, 'memaksa' pengguna untuk berinteraksi dengannya sebelum bisa kembali ke layar utama. LAMW menyediakan dua fungsi bawaan yang sangat mudah digunakan untuk ini.

**ShowMessage - Untuk Informasi Cepat**

Fungsi ShowMessage adalah cara termudah untuk menampilkan pesan informasi kepada pengguna. Dialog yang muncul hanya akan memiliki satu tombol, yaitu 'OK', yang berfungsi untuk menutup dialog tersebut. Ini sangat cocok untuk memberitahu pengguna bahwa sebuah aksi telah berhasil diselesaikan.

* **Contoh Kode:**

Delphi

procedure TAndroidModule1.btnSimpanClick(Sender: TObject);

begin

// ... (logika untuk menyimpan data ke database atau file) ...

// Setelah selesai, tampilkan pesan konfirmasi

ShowMessage('Data berhasil disimpan!');

end;

**MessageDlg - Untuk Konfirmasi dan Pilihan**

MessageDlg adalah fungsi yang jauh lebih kuat. Ia memungkinkan kita untuk mengatur judul dialog, pesan, ikon, dan yang terpenting, jenis tombol yang akan ditampilkan (misalnya, Ya/Tidak, OK/Batal). Fungsi ini akan mengembalikan sebuah nilai yang merepresentasikan tombol mana yang ditekan oleh pengguna, sehingga kita bisa membuat logika percabangan berdasarkan pilihan tersebut.

* **Contoh Kode: Konfirmasi Keluar Aplikasi** Bayangkan kita ingin menampilkan dialog konfirmasi saat pengguna menekan tombol 'back' pada perangkat di layar utama aplikasi. Kita bisa menaruh kode ini di dalam *event* OnFormKeyUp dari jForm utama kita.
  1. Di *Object Inspector* untuk jForm1, buka tab **Events**.
  2. Klik dua kali pada event OnKeyUp.
  3. Isi prosedur yang dibuat dengan kode berikut:

Delphi

procedure TAndroidModule1.jForm1KeyUp(Sender: TObject; var KeyCode: integer;

var KeyShift: TShiftState);

begin

// Periksa apakah tombol yang ditekan adalah tombol 'Back'

if KeyCode = vkHardwareBack then

begin

// Mencegah aplikasi langsung tertutup

KeyCode := 0;

// Tampilkan dialog konfirmasi

if MessageDlg('Konfirmasi', 'Apakah Anda yakin ingin keluar dari aplikasi?',

mtConfirmation, [mbYes, mbNo], 0) = mrYes then

begin

// Jika pengguna menekan 'Yes', maka tutup aplikasi

Self.Close;

end;

// Jika pengguna menekan 'No', maka tidak terjadi apa-apa.

end;

end;

**Penjelasan Kode:**

* 1. **if KeyCode = vkHardwareBack then**: Kita memeriksa apakah tombol yang baru saja dilepaskan pengguna adalah tombol 'Back' fisik atau virtual.
  2. **KeyCode := 0;**: Ini adalah trik penting untuk 'mengkonsumsi' *event*. Dengan mengaturnya ke 0, kita mencegah perilaku default dari tombol 'back' (yaitu langsung menutup aplikasi), sehingga kita bisa menanganinya sendiri.
  3. **MessageDlg(...)**: Kita memanggil dialog dengan judul, pesan, tipe ikon konfirmasi (mtConfirmation), dan satu set tombol ([mbYes, mbNo]).
  4. **= mrYes then**: Kita memeriksa nilai yang dikembalikan oleh MessageDlg. Jika nilainya sama dengan mrYes (konstanta untuk tombol 'Yes'), maka kita eksekusi kode di dalam blok begin..end, yaitu menutup form aplikasi.

Ini mengakhiri Bab 5. Anda kini memiliki kemampuan untuk membangun aplikasi dengan alur kerja multi-layar yang utuh. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 6 (Pengantar Bab 6).

YAHOX

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 6.

Berikut adalah jawaban untuk prompt **Pengantar Bab 6**.

**Bab 6: Manajemen Data Lokal Sesuai Aturan Modern**

Mari kita ajukan sebuah pertanyaan penting: apa yang terjadi pada semua informasi di aplikasi Anda saat pengguna menutupnya? Tanpa penanganan khusus, jawabannya adalah: semuanya hilang. Aplikasi Anda akan menderita 'amnesia' setiap kali dimulai ulang. Tentu saja, ini bukanlah perilaku yang kita inginkan dari sebuah aplikasi yang andal.

Aplikasi yang profesional harus memiliki 'memori'. Ia harus bisa mengingat pengaturan pengguna, menyimpan dokumen yang sedang dikerjakan, atau mengelola daftar data yang kompleks. Kemampuan untuk menyimpan data secara permanen di perangkat ini disebut **persistensi data** (*data persistence*), dan ini adalah salah satu pilar fundamental dalam pengembangan aplikasi.

Di bab ini, kita akan mempelajari berbagai teknik untuk memberikan 'memori' pada aplikasi LAMW kita, tentunya dengan mengikuti aturan-aturan modern yang berlaku di platform Android. Kita akan menjelajahi tiga 'tingkatan' penyimpanan data, masing-masing dengan kegunaannya sendiri:

* **Penyimpanan Preferensi Sederhana (SharedPreferences):** Untuk menyimpan data kecil berformat kunci-nilai, seperti pengaturan 'Mode Malam', nama pengguna, atau skor tertinggi.
* **Penyimpanan Berbasis File:** Untuk menyimpan data yang lebih besar dan tidak terstruktur seperti catatan teks, gambar, atau dokumen. Kita akan mempelajari cara melakukannya dengan aman sesuai aturan **Scoped Storage**.
* **Database Relasional (SQLite):** 'Lemari arsip' canggih untuk menyimpan data yang terstruktur dan kompleks, seperti daftar kontak, inventaris produk, atau daftar tugas.

Seiring dengan kemampuan menyimpan data, datang pula tanggung jawab. Di dunia Android modern, kita tidak bisa begitu saja mengakses penyimpanan tanpa izin. Oleh karena itu, di bab ini kita juga akan membahas topik krusial tentang **Manajemen Izin Runtime**, yaitu cara meminta izin kepada pengguna secara sopan dan benar sebelum mengakses data atau fitur sensitif. Mari kita mulai bekali aplikasi kita dengan kemampuan untuk mengingat.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 6.1.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 6.1: Penyimpanan Preferensi dengan jSharedPreferences**.

**6.1. Penyimpanan Preferensi dengan jSharedPreferences**

Cara termudah dan paling umum untuk menyimpan data sederhana di Android adalah dengan menggunakan SharedPreferences. Anggap saja ini seperti sebuah 'buku catatan kecil' atau 'memo' di mana aplikasi Anda bisa menulis dan membaca potongan-potongan informasi kecil, seperti nama pengguna, status login, pilihan tema, atau skor tertinggi.

Data disimpan dalam format **kunci-nilai (key-value)**, mirip seperti kamus. Anda memberikan sebuah 'kunci' (berupa string) yang unik untuk setiap 'nilai' yang ingin Anda simpan.

**Menyimpan Data ke SharedPreferences**

Mari kita buat contoh di mana kita menyimpan nama pengguna dari sebuah jEditText (edtNama) dan status 'Mode Malam' dari sebuah jSwitch (swModeMalam). Proses penyimpanan ini bisa kita letakkan di dalam *event* OnClick sebuah tombol 'Simpan'.

Delphi

procedure TAndroidModule1.btnSimpanPreferensiClick(Sender: TObject);

var

prefs: jSharedPreferences;

editor: jSharedPreferences\_Editor;

begin

// 1. Dapatkan instance SharedPreferences dengan nama file tertentu

prefs := Self.GetSharedPreferences('pengaturan\_aplikasiku', 0);

// 2. Minta 'Editor' untuk mulai menulis atau mengubah data

editor := prefs.Edit;

// 3. Masukkan data dengan format Kunci-Nilai

editor.PutString('nama\_pengguna', edtNama.Text);

editor.PutBoolean('mode\_malam\_aktif', swModeMalam.IsChecked);

// 4. Simpan perubahan secara asinkron

editor.Apply;

ShowMessage('Preferensi berhasil disimpan!');

end;

**Penjelasan Kode:**

* **prefs := Self.GetSharedPreferences(...)**: Kita meminta sebuah file preferensi bernama 'pengaturan\_aplikasiku'. Jika file ini belum ada, Android akan membuatnya. Angka 0 adalah mode operasi default (pribadi hanya untuk aplikasi ini).
* **editor := prefs.Edit;**: Kita tidak bisa menulis langsung ke SharedPreferences. Kita harus meminta objek 'Editor' terlebih dahulu untuk memulai sesi pengeditan.
* **editor.PutString(...) dan editor.PutBoolean(...)**: Kita memasukkan data kita. Parameter pertama adalah **Kunci** (Anda bebas menentukannya), dan yang kedua adalah **Nilai** yang ingin disimpan.
* **editor.Apply;**: Ini adalah perintah untuk menyimpan semua perubahan yang telah kita masukkan ke editor. Metode Apply bekerja di latar belakang (*asynchronous*) dan tidak memblokir UI, sehingga ini adalah cara yang direkomendasikan. Alternatifnya adalah .Commit(), yang bekerja secara sinkron.

**Membaca Data dari SharedPreferences**

Setelah data tersimpan, kita tentu perlu membacanya kembali, misalnya saat aplikasi pertama kali dibuka. Tempat yang baik untuk melakukan ini adalah di *event* OnCreate dari Form.

Delphi

procedure TAndroidModule1.jForm1Create(Sender: TObject);

var

prefs: jSharedPreferences;

nama: string;

modeMalam: boolean;

begin

// 1. Dapatkan instance SharedPreferences dengan nama file yang sama

prefs := Self.GetSharedPreferences('pengaturan\_aplikasiku', 0);

// 2. Baca data menggunakan Kunci yang sama, sediakan nilai default

nama := prefs.GetString('nama\_pengguna', ''); // Default: string kosong

modeMalam := prefs.GetBoolean('mode\_malam\_aktif', False); // Default: False

// 3. Terapkan nilai yang dibaca ke komponen UI

edtNama.Text := nama;

swModeMalam.IsChecked := modeMalam;

end;

**Penjelasan Kode:**

* **prefs.GetString('nama\_pengguna', '');**: Kita mencoba membaca sebuah string dengan kunci 'nama\_pengguna'. Parameter kedua, '', adalah **nilai default**. Jika kunci 'nama\_pengguna' tidak ditemukan (misalnya saat aplikasi dijalankan pertama kali), maka fungsi ini akan mengembalikan *string* kosong. Ini mencegah aplikasi dari *error*.
* **prefs.GetBoolean('mode\_malam\_aktif', False);**: Logika yang sama berlaku di sini. Jika kunci 'mode\_malam\_aktif' tidak ditemukan, nilai False akan dikembalikan.
* **edtNama.Text := nama;**: Terakhir, kita menerapkan nilai yang telah kita baca ke komponen UI yang relevan agar pengguna bisa melihat pengaturan terakhir mereka saat membuka aplikasi.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 6.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke topik yang sangat penting di Android modern.

Berikut adalah jawaban untuk prompt **Subbab 6.2: Bekerja dengan File di Era Scoped Storage**.

**6.2. Bekerja dengan File di Era Scoped Storage**

Cara aplikasi berinteraksi dengan file di perangkat telah berubah secara drastis dalam beberapa tahun terakhir. Lupakan cara lama di mana aplikasi bisa dengan bebas membaca dan menulis di seluruh penyimpanan pengguna. Selamat datang di era **Scoped Storage**, sebuah kebijakan yang diberlakukan Google untuk meningkatkan privasi dan keamanan pengguna.

Tujuannya sederhana: mencegah aplikasi mengakses data yang bukan miliknya tanpa izin eksplisit dari pengguna, dan memastikan saat sebuah aplikasi di-uninstall, semua file 'sampah' yang dibuatnya ikut terhapus bersih.

Ada dua cara utama untuk bekerja dengan file di era ini:

**1. Penyimpanan Spesifik Aplikasi (Jalur Pribadi Anda)**

Ini adalah cara termudah dan paling direkomendasikan untuk menyimpan file yang hanya relevan untuk aplikasi Anda sendiri (misalnya, file konfigurasi, gambar *cache*, atau catatan pribadi yang dibuat di dalam aplikasi). Anggap ini sebagai 'kotak pasir' atau 'folder pribadi' milik aplikasi Anda yang tidak bisa diakses langsung oleh aplikasi lain.

**Kelebihan utamanya: Anda tidak memerlukan izin penyimpanan apa pun untuk menulis dan membaca di direktori ini.**

**Contoh Praktis: Menyimpan dan Membaca Catatan Teks**

Delphi

// Prosedur untuk menyimpan file

procedure TAndroidModule1.btnSimpanCatatanClick(Sender: TObject);

var

path: string;

fileContent: TStringList;

begin

// Dapatkan path lengkap ke file di direktori eksternal pribadi aplikasi

path := GetExternalFilesDir(nil) + '/catatanku.txt';

fileContent := TStringList.Create;

try

fileContent.Text := jEditTextCatatan.Text; // Ambil teks dari komponen input

fileContent.SaveToFile(path);

ShowMessage('Catatan disimpan di: ' + path);

finally

fileContent.Free;

end;

end;

// Prosedur untuk membaca file

procedure TAndroidModule1.btnBacaCatatanClick(Sender: TObject);

var

path: string;

fileContent: TStringList;

begin

path := GetExternalFilesDir(nil) + '/catatanku.txt';

if jFile.Exists(path) then

begin

fileContent := TStringList.Create;

try

fileContent.LoadFromFile(path);

jEditTextCatatan.Text := fileContent.Text; // Tampilkan isi file

finally

fileContent.Free;

end;

end

else

ShowMessage('File catatan tidak ditemukan!');

end;

**2. Penyimpanan Bersama (Shared Storage) - Berinteraksi dengan Dunia Luar**

Bagaimana jika Anda membangun editor teks atau galeri gambar yang perlu membuka dan menyimpan file di mana saja di perangkat pengguna, di luar 'kotak pasir' Anda? Untuk ini, kita menggunakan **Storage Access Framework (SAF)**.

SAF tidak memberikan akses bebas ke aplikasi Anda. Sebaliknya, ia **mendelegasikan kontrol kepada pengguna**. Aplikasi Anda akan meminta sistem Android untuk menampilkan dialog 'pemilih file' (*file picker*), dan pengguna-lah yang akan memilih file atau lokasi mana yang boleh diakses oleh aplikasi Anda.

* **Menyimpan File dengan SAF (ACTION\_CREATE\_DOCUMENT)** Gunakan ini saat Anda ingin pengguna memilih di mana file baru akan disimpan.

Delphi

// ... di dalam OnClick tombol 'Simpan Sebagai...'

const REQUEST\_CODE\_SAVE\_FILE = 101;

var intent: jIntent;

begin

intent := jIntent.Create;

intent.SetAction(jIntent\_ACTION\_CREATE\_DOCUMENT);

intent.AddCategory(jIntent\_CATEGORY\_OPENABLE);

intent.SetType('text/plain');

intent.PutExtra(jIntent\_EXTRA\_TITLE, 'catatan\_baru.txt');

StartActivityForResult(intent, REQUEST\_CODE\_SAVE\_FILE);

end;

Kode ini akan membuka *file manager* sistem. Setelah pengguna memilih lokasi dan menyimpan, hasilnya akan dikembalikan ke *event* OnActivityResult aplikasi Anda, yang berisi Uri (alamat unik) ke file baru tersebut yang bisa Anda gunakan untuk menulis data.

* **Membuka File dengan SAF (ACTION\_OPEN\_DOCUMENT)** Gunakan ini saat Anda ingin pengguna memilih file yang sudah ada untuk dibuka.

Delphi

// ... di dalam OnClick tombol 'Buka File...'

const REQUEST\_CODE\_OPEN\_FILE = 102;

var intent: jIntent;

begin

intent := jIntent.Create;

intent.SetAction(jIntent\_ACTION\_OPEN\_DOCUMENT);

intent.AddCategory(jIntent\_CATEGORY\_OPENABLE);

intent.SetType('text/plain');

StartActivityForResult(intent, REQUEST\_CODE\_OPEN\_FILE);

end;

Sama seperti sebelumnya, setelah pengguna memilih file, OnActivityResult akan dipicu, memberikan Anda Uri ke file yang dipilih untuk dibaca isinya.

**Catatan Penting Mengenai Izin Lama**

Anda mungkin melihat tutorial lama di internet yang menggunakan izin <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" /> untuk akses file. Untuk aplikasi yang menargetkan Android versi baru (API 29+), izin ini **tidak lagi memberikan akses tulis umum** ke seluruh penyimpanan. Mengandalkan izin ini adalah praktik yang sudah usang. **Selalu utamakan penggunaan Penyimpanan Spesifik Aplikasi atau Storage Access Framework (SAF).**

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 6.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 6.3: Bab Krusial: Manajemen Izin Runtime (Runtime Permissions)**.

**6.3. Bab Krusial: Manajemen Izin Runtime (Runtime Permissions)**

Di masa lalu, pengguna memberikan semua izin yang diminta aplikasi saat proses instalasi. Model ini memiliki kelemahan karena pengguna sering kali tidak sadar izin apa saja yang mereka berikan. Untuk meningkatkan privasi dan kontrol pengguna, Android modern memperkenalkan model **Izin Runtime** (*Runtime Permissions*).

Konsepnya sederhana namun sangat penting: **Minta izin hanya saat Anda akan menggunakan fiturnya.** Model ini berlaku untuk "izin berbahaya" (*dangerous permissions*)—izin yang terkait dengan data atau fungsionalitas sensitif seperti Kamera, Lokasi (GPS), Kontak, Mikrofon, dan lain-lain.

**Alur Kerja Tiga Langkah yang Wajib Diikuti**

Setiap kali Anda akan menggunakan fitur yang memerlukan izin berbahaya, Anda **wajib** mengikuti alur tiga langkah ini untuk memastikan aplikasi Anda bekerja dengan benar di semua versi Android modern.

1. **PERIKSA (Check):** Sebelum melakukan apa pun, periksa terlebih dahulu apakah pengguna **sudah** memberikan izin tersebut sebelumnya. Jika sudah, Anda bisa langsung melanjutkan eksekusi fitur.
2. **MINTA (Request):** Jika izin belum diberikan, inilah saatnya Anda meminta. Praktik terbaik adalah dengan menampilkan dialog penjelasan singkat terlebih dahulu (*rationale*) tentang mengapa aplikasi Anda membutuhkan izin ini, baru kemudian menampilkan dialog permintaan izin resmi dari sistem.
3. **TANGGAPI (Handle):** Setelah pengguna memilih 'Izinkan' atau 'Tolak' pada dialog sistem, aplikasi Anda akan menerima hasilnya. Anda harus menangani kedua kemungkinan tersebut dengan baik. Jika ditolak, jangan paksa pengguna atau membuat aplikasi *crash*; berikan pesan yang informatif bahwa fitur tersebut tidak dapat digunakan tanpa izin.

**Implementasi Praktis di LAMW**

Mari kita lihat contoh lengkap bagaimana alur ini diimplementasikan di LAMW untuk meminta izin CAMERA.

Delphi

const

REQUEST\_CODE\_CAMERA\_PERMISSION = 201; // Nomor unik untuk permintaan ini

// Prosedur ini bisa dipanggil dari OnClick sebuah tombol "Buka Kamera"

procedure TAndroidModule1.btnBukaKameraClick(Sender: TObject);

var

permissions: TJavaObjectArray<string>;

begin

// LANGKAH 1: PERIKSA IZIN

if not jPermissions.Check(Manifest\_permission\_CAMERA) then

begin

// LANGKAH 2: MINTA IZIN (jika belum ada)

// Di sini Anda bisa menampilkan dialog penjelasan (rationale) terlebih dahulu jika perlu

permissions := TJavaObjectArray<string>.Create(1);

permissions[0] := Manifest\_permission\_CAMERA;

jPermissions.Request(permissions, REQUEST\_CODE\_CAMERA\_PERMISSION);

permissions.Free;

end

else

begin

// Izin sudah ada, langsung jalankan logika kamera

JalankanLogikaKamera;

end;

end;

// LANGKAH 3: TANGGAPI HASIL PERMINTAAN

// Ini adalah event dari jForm yang harus di-assign di Object Inspector

procedure TAndroidModule1.OnRequestPermissionsResult(requestCode: integer;

permissions: TJavaObjectArray<string>; grantResults: TJavaObjectArray<integer>);

begin

if requestCode = REQUEST\_CODE\_CAMERA\_PERMISSION then

begin

if (grantResults.Length > 0) and (grantResults[0] = jPackageManager.PERMISSION\_GRANTED) then

begin

// Pengguna memberikan izin!

ShowMessage('Izin kamera diberikan. Anda sekarang bisa membuka kamera.');

JalankanLogikaKamera;

end

else

begin

// Pengguna menolak izin

ShowMessage('Aplikasi tidak bisa menggunakan kamera tanpa izin Anda.');

end;

end;

end;

// Prosedur imajiner tempat logika kamera Anda berada

procedure TAndroidModule1.JalankanLogikaKamera;

begin

// ... di sini letakkan kode untuk membuka kamera (misalnya dengan Intent)...

ShowMessage('Logika untuk membuka kamera dijalankan di sini!');

end;

**Penjelasan Kode:**

* REQUEST\_CODE\_CAMERA\_PERMISSION: Kita mendefinisikan sebuah nomor permintaan unik. Ini berguna di OnRequestPermissionsResult untuk mengetahui permintaan mana yang sedang direspons.
* jPermissions.Check(...): Fungsi *helper* dari LAMW untuk memeriksa apakah izin (Manifest\_permission\_CAMERA) sudah diberikan.
* jPermissions.Request(...): Jika izin belum ada, kita memanggil fungsi ini untuk menampilkan dialog permintaan resmi dari sistem.
* OnRequestPermissionsResult(...): Ini adalah *event* yang secara otomatis akan dipanggil oleh sistem setelah pengguna merespons dialog. Kita memeriksa requestCode untuk memastikan ini adalah respons untuk permintaan kamera kita, lalu memeriksa grantResults untuk melihat apakah hasilnya adalah PERMISSION\_GRANTED.
* JalankanLogikaKamera: Memisahkan logika utama ke dalam prosedur sendiri adalah praktik yang baik agar kode mudah dibaca.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 6.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 6.4: Database Lokal dengan SQLite**.

**6.4. Database Lokal dengan SQLite**

Ketika SharedPreferences tidak lagi cukup untuk data sederhana, dan file teks terlalu merepotkan untuk data yang terstruktur, saatnya kita menggunakan 'alat berat': **database**. Android secara bawaan menyertakan **SQLite**, sebuah sistem database relasional yang ringan, cepat, dan sangat andal, yang tersimpan dalam satu file di dalam direktori privat aplikasi Anda.

SQLite sempurna untuk menyimpan data yang memiliki struktur tabel, seperti daftar kontak, katalog produk, atau, seperti contoh kita kali ini, sebuah **Aplikasi Kontak Sederhana**. Kita akan mempelajari siklus operasi database yang paling fundamental, yang biasa disingkat **CRUD**: **C**reate (Membuat), **R**ead (Membaca), **U**pdate (Memperbarui), dan **D**elete (Menghapus).

**1. Setup Database dan Tabel**

Cara terbaik untuk mengelola database di Android adalah dengan membuat kelas *helper* turunan dari jSQLiteOpenHelper. Kelas ini akan menangani pembuatan database jika belum ada, atau proses *upgrade* jika skema database Anda berubah di versi aplikasi selanjutnya.

Delphi

// Disarankan ditaruh di unit terpisah (misal: uDBHelper.pas)

type

TDBHelper = class(jSQLiteOpenHelper)

private

// ...

public

constructor Create(AContext: jContext);

procedure onCreate(db: jSQLiteDatabase); override;

procedure onUpgrade(db: jSQLiteDatabase; oldVersion: integer; newVersion: integer); override;

end;

// ... implementasi ...

procedure TDBHelper.onCreate(db: jSQLiteDatabase);

const

SQL\_CREATE\_TABLE = 'CREATE TABLE kontak (id INTEGER PRIMARY KEY AUTOINCREMENT, '+

'nama TEXT NOT NULL, telepon TEXT);';

begin

db.ExecSQL(SQL\_CREATE\_TABLE);

end;

Di form utama Anda, Anda akan membuat *instance* dari *helper* ini untuk mendapatkan akses ke database:

Delphi

// Di dalam Form utama

var

dbHelper: TDBHelper;

db: jSQLiteDatabase;

//...

// di event OnCreate form

begin

dbHelper := TDBHelper.Create(Self);

db := dbHelper.GetWritableDatabase; // Minta akses untuk menulis & membaca

end;

**2. Operasi CRUD**

**CREATE - Menambah Kontak Baru** Kita menggunakan jContentValues untuk "mengemas" data yang akan dimasukkan ke dalam satu baris tabel.

Delphi

var

values: jContentValues;

begin

values := jContentValues.Create;

try

values.Put('nama', edtNama.Text); // Kolom 'nama', Nilai dari jEditText

values.Put('telepon', edtTelepon.Text); // Kolom 'telepon', Nilai dari jEditText

db.Insert('kontak', '', values); // Masukkan ke tabel 'kontak'

finally

values.Free;

end;

// ... (kode untuk me-refresh list) ...

end;

**READ - Membaca Semua Kontak** Hasil dari *query* database adalah sebuah jCursor. Anggap saja ini sebagai sebuah *pointer* yang menunjuk ke setiap baris hasil *query*, satu per satu.

Delphi

var

cursor: jCursor;

nama, telepon: string;

begin

// Query semua data dari tabel 'kontak', urutkan berdasarkan nama

cursor := db.Query('kontak', nil, '', nil, '', '', 'nama ASC');

try

while cursor.MoveToNext do

begin

nama := cursor.GetString(cursor.GetColumnIndex('nama'));

telepon := cursor.GetString(cursor.GetColumnIndex('telepon'));

// ... (tambahkan nama dan telepon ke sebuah jListView atau komponen lain) ...

end;

finally

cursor.Close; // Selalu tutup cursor setelah selesai untuk melepas sumber daya!

end;

end;

**UPDATE - Memperbarui Kontak** Untuk memperbarui, kita perlu jContentValues berisi data baru dan sebuah klausa WHERE untuk memberitahu database baris mana yang harus diubah (biasanya berdasarkan id unik).

Delphi

var

values: jContentValues;

args: TJavaObjectArray<string>;

begin

values := jContentValues.Create;

try

values.Put('telepon', '081234567890'); // Nomor telepon baru

// Siapkan argumen untuk klausa WHERE untuk keamanan

args := TJavaObjectArray<string>.Create(1);

args[0] := jObjectToString('Budi');

// Update kontak di mana nama = 'Budi'

db.Update('kontak', values, 'nama = ?', args);

args.Free;

finally

values.Free;

end;

end;

**DELETE - Menghapus Kontak** Menghapus data juga memerlukan klausa WHERE untuk menargetkan baris yang benar.

Delphi

var

args: TJavaObjectArray<string>;

begin

// Siapkan argumen untuk klausa WHERE

args := TJavaObjectArray<string>.Create(1);

args[0] := jObjectToString('Budi');

// Hapus kontak di mana nama = 'Budi'

db.Delete('kontak', 'nama = ?', args);

args.Free;

end;

Ini mengakhiri Bab 6. Anda kini memiliki persenjataan lengkap untuk mengelola berbagai jenis data lokal di aplikasi Anda. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 7 (Pengantar Bab 7).

YAHOX

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 7.

Berikut adalah jawaban untuk prompt **Pengantar Bab 7**.

**Bab 7: Komunikasi Jaringan dan Mengonsumsi API**

Sejauh ini, aplikasi yang kita bangun bagaikan sebuah pulau yang mandiri. Ia bisa menyimpan datanya sendiri dan berinteraksi dengan pengguna, tetapi masih terisolasi dari dunia luar yang luas. Di era digital saat ini, sebuah aplikasi yang tidak terhubung ke internet akan terasa sangat terbatas. Kekuatan sesungguhnya dari aplikasi modern terletak pada kemampuannya untuk berinteraksi dengan dunia: mengambil berita terbaru, menampilkan ramalan cuaca secara *real-time*, atau memuat *feed* media sosial yang terus berubah.

Di bab ini, kita akan membangun jembatan dari 'pulau' aplikasi kita ke 'benua' internet yang luas. Kita akan mengajarkan aplikasi kita cara 'berbicara' dengan server-server di seluruh dunia untuk meminta dan menerima informasi secara dinamis. Untuk melakukan ini, kita perlu memahami 'tiga serangkai suci' dalam komunikasi data modern:

* **HTTP (Hypertext Transfer Protocol):** Anggap ini sebagai **bahasa atau protokol** standar yang digunakan oleh semua komputer di internet untuk saling berbicara dan bertukar informasi.
* **API (Application Programming Interface):** Angap ini sebagai **'menu' atau 'pelayan'** dari sebuah server. Kita tidak bisa langsung mengambil data dari 'dapur' server; kita harus 'memesan' melalui API yang sudah disediakan secara resmi.
* **JSON (JavaScript Object Notation):** Anggap ini sebagai **format 'paket' data** yang paling umum digunakan untuk mengirim dan menerima informasi. Data yang kita minta dari API biasanya akan dibungkus dalam format JSON yang ringan dan mudah dibaca oleh manusia maupun mesin.

Dengan menguasai cara melakukan permintaan HTTP, berkomunikasi dengan API, dan 'membongkar' paket data JSON, Anda akan membuka potensi tak terbatas untuk aplikasi Anda. Aplikasi Anda tidak lagi hanya mengandalkan data yang ada di perangkat, tetapi bisa menjadi jendela menuju informasi dari seluruh dunia. Mari kita mulai membangun jembatan pertama kita.

Saya akan menunggu perintah 'yahok' dari Anda sebelum melanjutkan ke prompt Subbab 7.1.

PRESENT IT

Tampilkan alur berpikir

Baik, saya sajikan materi berikutnya.

Berikut adalah jawaban untuk prompt **Subbab 7.1: Dasar-dasar Komunikasi HTTP (GET, POST) dan REST API**.

**7.1. Dasar-dasar Komunikasi HTTP (GET, POST) dan REST API**

Sebelum kita mengirim permintaan jaringan pertama kita, kita perlu memahami 'tata krama' atau aturan main di dunia web. Ada beberapa konsep dasar yang menjadi fondasi dari hampir semua interaksi data di internet.

**HTTP - Bahasa Universal Web**

**HTTP (Hypertext Transfer Protocol)** adalah bahasa yang digunakan oleh perangkat Anda (klien) untuk 'berbicara' dengan server di internet. Setiap kali Anda membuka situs web atau aplikasi Anda meminta data, ia sedang mengirimkan sebuah **Permintaan HTTP (HTTP Request)**. Server kemudian akan membalas dengan sebuah **Respons HTTP (HTTP Response)** yang berisi data yang diminta atau status dari permintaan tersebut.

**Metode HTTP Umum: GET vs. POST**

Dalam sebuah permintaan HTTP, kita harus menyebutkan 'maksud' atau metode kita. Dua maksud yang paling umum adalah:

* **GET:** Digunakan untuk **meminta atau mengambil** data dari server. Ini seperti Anda bertanya kepada pustakawan, "Tolong berikan saya buku tentang sejarah Indonesia." Anda tidak mengubah apa pun di perpustakaan, Anda hanya mengambil informasi.
* **POST:** Digunakan untuk **mengirim atau membuat** data baru di server. Ini seperti Anda menyerahkan formulir pendaftaran anggota baru ke perpustakaan. Anda memberikan informasi baru untuk disimpan oleh sistem.

**API - Sang Pelayan Digital**

**API (Application Programming Interface)** adalah konsep yang paling penting untuk dipahami. Bayangkan Anda berada di sebuah restoran. Anda tidak bisa langsung masuk ke dapur dan memasak makanan Anda sendiri. Anda harus berinteraksi dengan **pelayan**.

Anda melihat **menu** (ini adalah dokumentasi API) untuk mengetahui apa saja hidangan yang tersedia. Anda kemudian memberitahu pelayan pesanan Anda (ini adalah saat Anda membuat *request* ke API), dan sang pelayan akan mengurus semuanya di dapur (server) lalu mengantarkan hidangan (data *response*) kepada Anda. **API adalah perantara resmi yang memungkinkan aplikasi kita berinteraksi dengan server secara terstruktur.**

**REST API - Aturan Main yang Populer**

Anda akan sering mendengar istilah **REST (Representational State Transfer)** atau **RESTful API**. Anggap saja REST ini adalah serangkaian 'aturan etiket' atau gaya arsitektur yang sangat populer untuk membangun API agar mudah dipahami dan digunakan. API yang mengikuti gaya REST (RESTful), misalnya, akan menggunakan metode GET untuk mengambil data pengguna dari *endpoint* /users dan metode POST ke /users untuk membuat pengguna baru. Ini membuat API menjadi prediktif dan logis.

**Endpoint - Alamat Tujuan**

Sebuah **Endpoint** adalah URL spesifik tempat kita mengirimkan permintaan kita. Setiap fungsi dalam sebuah API biasanya memiliki *endpoint*-nya sendiri. Contoh:

* https://api.cuaca.com/v1/jakarta adalah *endpoint* untuk mendapatkan data cuaca kota Jakarta.
* https://api.toko.com/v2/products/123 adalah *endpoint* untuk mendapatkan detail produk dengan ID 123.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 7.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 7.2: Melakukan Request Jaringan Asinkron dengan jHttpClient**.

**7.2. Melakukan Request Jaringan Asinkron dengan jHttpClient**

Sekarang kita akan menulis kode untuk meminta data. Namun, ada satu aturan emas dalam melakukan operasi jaringan di Android: **JANGAN PERNAH lakukan di Thread Utama (UI Thread)!**

Operasi jaringan bergantung pada kecepatan internet dan responsivitas server, yang bisa memakan waktu sepersekian detik hingga beberapa detik atau lebih. Jika Anda melakukannya di *thread* utama, seluruh antarmuka pengguna (UI) aplikasi Anda akan **macet total (freeze)** selama menunggu respons. Jika terlalu lama, sistem Android akan menampilkan dialog yang ditakuti semua developer: **'Application Not Responding' (ANR)**.

Solusinya adalah dengan melakukan permintaan secara **asinkron (*asynchronous*)**. Artinya, kita 'menyuruh' aplikasi untuk meminta data di latar belakang, dan biarkan aplikasi tetap berjalan tanpa menunggu. Ketika respons dari server datang, aplikasi akan 'diberitahu' melalui sebuah *event*.

**Pola Kerja Permintaan Asinkron**

Praktik terbaik untuk alur kerja ini adalah sebagai berikut:

1. **Mulai Aksi:** Pengguna menekan tombol 'Muat Data' atau membuka sebuah layar.
2. **Tampilkan Indikator Loading:** Segera tampilkan sebuah jProgressBar yang berputar untuk memberitahu pengguna bahwa aplikasi sedang bekerja. Nonaktifkan tombol untuk mencegah klik ganda.
3. **Eksekusi Permintaan Jaringan:** Jalankan permintaan HTTP di latar belakang.
4. **Tangani Hasil:** Saat respons diterima (baik sukses maupun gagal), sebuah *event handler* akan dipicu.
5. **Perbarui UI & Sembunyikan Indikator:** Di dalam *event handler* tersebut, perbarui komponen UI dengan data baru (atau tampilkan pesan error), lalu sembunyikan jProgressBar dan aktifkan kembali tombolnya.

**Contoh Praktis dengan jHttpClient**

LAMW menyediakan komponen jHttpClient untuk mempermudah proses ini. Mari kita lihat contoh cara menggunakannya untuk melakukan permintaan GET sederhana. Anggap kita punya sebuah jButton (btnGetData), jTextView (tvHasilJson), dan jProgressBar (jProgressBar1) di form kita.

1. Letakkan komponen jHttpClient dari *Component Palette* ke atas form Anda. Komponen ini tidak terlihat saat runtime. Beri nama jHttpClient1.
2. Tulis kode berikut:

Delphi

// Event OnClick untuk tombol

procedure TAndroidModule1.btnGetDataClick(Sender: TObject);

begin

// Langkah 2: Tampilkan loading, nonaktifkan tombol

jProgressBar1.Visible := True;

btnGetData.Enabled := False;

// Langkah 3: Atur URL target dan eksekusi secara asinkron

jHttpClient1.SetURL('https://api.publicapis.org/entries'); // Contoh API publik

jHttpClient1.Execute; // Ini tidak akan memblokir aplikasi

end;

// Event OnHttpRequestCompleted untuk komponen jHttpClient1

// Untuk membuatnya, pilih jHttpClient1, lalu di Object Inspector > Events, klik dua kali pada OnHttpRequestCompleted

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject;

responseCode: integer; responseText: string);

begin

// Langkah 5: Sembunyikan loading, aktifkan kembali tombol

jProgressBar1.Visible := False;

btnGetData.Enabled := True;

// Langkah 4: Tangani hasil

if responseCode = 200 then // Kode 200 artinya 'OK'/Sukses

begin

// Tampilkan teks JSON mentah ke TextView

tvHasilJson.Text := responseText;

end

else

begin

// Tampilkan pesan error jika gagal

tvHasilJson.Text := 'Gagal memuat data. Kode Error: ' + IntToStr(responseCode);

end;

end;

**Penjelasan Kode:**

* btnGetDataClick: Saat tombol diklik, kita hanya menampilkan *loading* dan memanggil jHttpClient1.Execute. Perhatikan, aplikasi tidak menunggu di sini; ia langsung melanjutkan dan tetap responsif.
* jHttpClient1HttpRequestCompleted: Ini adalah *event* yang secara otomatis akan dipanggil oleh jHttpClient1 **setelah** respons dari server diterima. Semua logika untuk menangani data diletakkan di sini.
* responseCode: integer: Parameter ini berisi kode status HTTP dari server (misalnya, 200 untuk sukses, 404 untuk tidak ditemukan).
* responseText: string: Parameter ini berisi data mentah yang dikirim oleh server, yang biasanya akan kita *parse* sebagai JSON di langkah selanjutnya.

Dengan pola ini, aplikasi Anda akan tetap terasa lancar bahkan saat sedang berkomunikasi dengan internet.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 7.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 7.3: Parsing Data JSON dengan Pustaka fpjson**.

**7.3. Parsing Data JSON dengan Pustaka fpjson**

Setelah berhasil menerima responseText dari server, kita mendapatkan sebuah *string* panjang. Bagi aplikasi kita, ini hanyalah sekumpulan teks acak. Agar bisa kita gunakan—misalnya, untuk menampilkan nama pengguna atau judul berita—kita perlu 'membongkar' atau mem-**parse** string ini menjadi struktur data yang bisa dimengerti. Format yang paling umum digunakan oleh hampir semua API modern untuk ini adalah **JSON**.

**Anatomi Singkat JSON**

**JSON (JavaScript Object Notation)** adalah format teks yang ringan untuk pertukaran data. Aturannya sangat sederhana dan mudah dibaca oleh manusia maupun mesin. Ada dua struktur utama yang perlu Anda kenali:

* **Objek { ... }**: Didefinisikan dengan kurung kurawal, berisi kumpulan pasangan **kunci-nilai** (*key-value*). Kunci selalu berupa *string* dalam tanda kutip. Contoh: {"nama": "Budi", "umur": 30, "aktif": true}.
* **Array [ ... ]**: Didefinisikan dengan kurung siku, berisi daftar nilai atau objek. Contoh: ["apel", "mangga", "jeruk"].

Struktur ini bisa digabungkan. Misalnya, sebuah objek bisa berisi *array*, dan sebuah *array* bisa berisi banyak objek, yang sangat umum kita temui pada respons API.

**Parsing JSON dengan fpjson di Pascal**

Free Pascal menyediakan pustaka bawaan yang sangat kuat untuk menangani JSON, yaitu **fpjson**. Untuk menggunakannya, pastikan Anda menambahkan fpjson dan jsonparser ke klausa uses di unit Anda.

**Contoh 1: Parsing Objek JSON Sederhana**

Mari kita ambil contoh *string* JSON yang merepresentasikan satu pengguna: '{"id": 7, "email": "michael.lawson@reqres.in", "first\_name": "Michael"}'

Berikut adalah cara mem-parse-nya di Pascal:

Delphi

uses

..., fpjson, jsonparser;

// ...

var

jsonString: string;

jsonObj: TJSONObject;

jsonValue: TJSONData;

nama: string;

id: integer;

begin

jsonString := '{"id": 7, "email": "michael.lawson@reqres.in", "first\_name": "Michael"}';

// 1. Parse string menjadi objek JSON generik

jsonValue := GetJSON(jsonString);

// 2. Ubah (cast) menjadi TJSONObject

jsonObj := TJSONObject(jsonValue);

try

// 3. Ambil nilai berdasarkan kuncinya

id := jsonObj.Get('id').AsInteger;

nama := jsonObj.Get('first\_name').AsString;

ShowMessage('ID: ' + IntToStr(id) + ', Nama: ' + nama);

finally

// 4. Jangan lupa bebaskan memori setelah selesai

jsonObj.Free;

end;

end;

**Contoh 2: Parsing Array Objek JSON**

Sering kali, API akan mengembalikan daftar data. Strukturnya mungkin terlihat seperti ini: '{ "page": 1, "data": [{"id": 1, "nama": "Buku Pascal"}, {"id": 2, "nama": "Buku LAMW"}] }'

Perhatikan bahwa daftar buku ada di dalam kunci "data" yang berisi sebuah *array*.

Delphi

uses

..., fpjson, jsonparser;

// ...

var

jsonString: string;

jsonObj: TJSONObject;

jsonArray: TJSONArray;

i: integer;

item: TJSONObject;

namaBuku: string;

begin

jsonString := '{ "data": [{"id": 1, "nama": "Buku Pascal"}, {"id": 2, "nama": "Buku LAMW"}] }';

// 1. Parse objek utamanya dulu

jsonObj := TJSONObject(GetJSON(jsonString));

try

// 2. Ambil array dari dalam objek berdasarkan kunci 'data'

jsonArray := TJSONArray(jsonObj.Get('data'));

// 3. Lakukan loop pada setiap item di dalam array

for i := 0 to jsonArray.Count - 1 do

begin

// Setiap item di dalam array adalah sebuah TJSONObject

item := TJSONObject(jsonArray.Items[i]);

namaBuku := item.Get('nama').AsString;

// Lakukan sesuatu dengan data, misal tampilkan di Logcat

Log.d('HASIL\_PARSE', namaBuku);

end;

finally

// 4. Cukup bebaskan objek utamanya, yang lain akan ikut terbebas

jsonObj.Free;

end;

end;

Dengan menguasai dua pola ini, Anda sudah siap untuk menangani sebagian besar format respons dari berbagai API.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 7.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 7.4: Menampilkan Data dari API ke Komponen UI (misal: jListView)**.

**7.4. Menampilkan Data dari API ke Komponen UI (misal: jListView)**

Sekarang kita sampai pada bagian paling memuaskan: menyatukan semua yang telah kita pelajari. Kita tahu cara meminta data (jHttpClient), dan kita tahu cara membongkar data (fpjson). Kini saatnya kita menggabungkan kedua keahlian tersebut untuk menampilkan data dinamis dari internet langsung ke antarmuka pengguna aplikasi kita.

Studi kasus kita adalah membuat sebuah **'Pembaca Berita Sederhana'**. Kita akan mengambil daftar judul artikel dari sebuah API publik gratis dan menampilkannya dalam sebuah jListView.

**1. Persiapan**

1. **Siapkan UI:** Di Form Anda, tambahkan komponen-komponen berikut:
   * Sebuah jButton dengan ID: btnMuatBerita dan Text: Muat Berita Terbaru.
   * Sebuah jProgressBar dengan ID: jProgressBar1, atur Visible-nya menjadi False.
   * Sebuah jListView dengan ID: lvBerita.
   * Pastikan Anda juga sudah menambahkan komponen jHttpClient ke form.
2. **Definisikan Struktur Data (Record):** Praktik yang baik adalah mendefinisikan sebuah record di Pascal untuk menampung data yang sudah di-parse. Ini membuat kode lebih rapi dan terstruktur. Di bagian type pada unit Anda, tambahkan:

Delphi

type

TArtikel = record

judul: string;

deskripsi: string;

end;

1. **Tambahkan uses Clause:** Jangan lupa tambahkan fpjson dan jsonparser ke klausa uses di bagian implementation unit Anda.

**2. Alur Lengkap dari Request hingga Tampil di UI**

Seluruh logika akan kita rangkai di dalam *event* OnClick tombol dan *event* OnHttpRequestCompleted dari jHttpClient.

**Kode untuk Tombol OnClick**

Saat tombol btnMuatBerita diklik, kita hanya perlu memulai prosesnya: menampilkan *loading* dan mengeksekusi permintaan.

Delphi

procedure TAndroidModule1.btnMuatBeritaClick(Sender: TObject);

begin

jProgressBar1.Visible := True;

btnMuatBerita.Enabled := False;

lvBerita.Clear; // Bersihkan list lama

// Kita akan menggunakan API publik gratis dari JSONPlaceholder

jHttpClient1.SetURL('https://jsonplaceholder.typicode.com/posts');

jHttpClient1.Execute;

end;

**Kode di OnHttpRequestCompleted (Inti Logika)**

Di sinilah semua keajaiban terjadi setelah data diterima dari server.

Delphi

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject;

responseCode: integer; responseText: string);

var

artikel: TArtikel;

jsonValue: TJSONData;

jsonArray: TJSONArray;

jsonItem: TJSONObject;

i: integer;

begin

jProgressBar1.Visible := False;

btnMuatBerita.Enabled := True;

if responseCode = 200 then

begin

try

// 1. Parse response string menjadi Array JSON

jsonValue := GetJSON(responseText);

jsonArray := TJSONArray(jsonValue);

// 2. Loop melalui setiap objek di dalam array JSON

for i := 0 to jsonArray.Count - 1 do

begin

// Ambil satu item JSON dari array

jsonItem := TJSONObject(jsonArray.Items[i]);

// 3. Buat record baru dan isi datanya dari JSON

artikel.judul := jsonItem.Get('title').AsString;

artikel.deskripsi := jsonItem.Get('body').AsString;

// 4. Langsung tambahkan judul ke ListView

lvBerita.Items.Add(artikel.judul);

end;

finally

// 5. Bebaskan memori objek JSON setelah selesai digunakan

jsonValue.Free;

end;

end

else

begin

ShowMessage('Gagal memuat berita. Kode Error: ' + IntToStr(responseCode));

end;

end;

Alur kerjanya sederhana: kita melakukan *request*. Jika sukses (kode 200), kita bersihkan *list* yang lama, lalu kita *parse* data JSON yang masuk. Kita lakukan *loop* pada setiap item berita, mengekstrak data yang kita butuhkan (dalam hal ini, hanya judulnya), lalu menambahkannya baris per baris ke dalam lvBerita. Jika gagal, kita tampilkan pesan error.

Sekarang, jalankan aplikasi Anda, tekan tombol "Muat Berita Terbaru", dan saksikan jListView Anda terisi dengan data langsung dari internet!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 7.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 7.5: Memuat Gambar dari URL ke jImageView secara Efisien**.

**7.5. Memuat Gambar dari URL ke jImageView secara Efisien**

Menampilkan teks dari internet itu relatif mudah dan cepat. Namun, bagaimana dengan gambar? Memuat gambar dari sebuah URL adalah operasi jaringan yang jauh lebih 'berat' dan memakan waktu. Melakukannya dengan cara yang salah dapat membuat aplikasi Anda terasa sangat lambat atau bahkan macet.

**Aturan Emas: Jangan Pernah Memuat Gambar di UI Thread!**

Sama seperti permintaan jaringan lainnya, proses mengunduh data gambar dari URL tidak boleh dilakukan secara langsung di *thread* utama (UI Thread). Jika Anda melakukannya, pengguna akan melihat aplikasi 'freeze' selama gambar diunduh, yang memberikan pengalaman pengguna yang sangat buruk dan berisiko memicu error ANR (*Application Not Responding*).

**Cara Termudah dengan Fitur Bawaan jImageView**

Kabar baiknya, tim LAMW memahami masalah ini. Komponen jImageView telah dilengkapi dengan metode pembantu (*helper*) yang sudah menangani proses pengunduhan asinkron ini untuk kita di belakang layar. Ini adalah cara termudah dan paling direkomendasikan untuk sebagian besar kasus.

* **Contoh Kode:**

Delphi

procedure TAndroidModule1.btnMuatGambarClick(Sender: TObject);

var

imageURL: string;

begin

imageURL := 'https://www.lazarus-ide.org/images/ide\_screenshot.png'; // URL gambar contoh

// Tampilkan ProgressBar sebelum memuat

jProgressBar1.Visible := True;

// Cukup panggil metode ini, LAMW akan menangani sisanya di latar belakang

// Parameter terakhir adalah untuk menampilkan ProgressBar selama proses loading

jImageView1.SetImageFromURL(imageURL, jProgressBar1);

end;

* **Penjelasan:** Saat Anda memanggil jImageView1.SetImageFromURL(imageURL, jProgressBar1), LAMW akan secara otomatis:
  1. Memulai sebuah tugas di latar belakang (*background thread*).
  2. Mengunduh data gambar dari URL tersebut.
  3. Mengubah data yang diunduh menjadi sebuah objek *Bitmap*.
  4. Kembali ke *thread* utama (UI Thread) dan menampilkan *Bitmap* tersebut di dalam jImageView1.
  5. Menyembunyikan jProgressBar1 setelah gambar selesai dimuat.

Selama proses ini, aplikasi Anda tetap sepenuhnya responsif.

**Memahami Pola Manual (Untuk Pengetahuan Lanjutan)**

Meskipun metode bawaan sudah sangat baik, sebagai developer, penting untuk memahami apa yang terjadi di baliknya. Jika suatu saat Anda perlu melakukan kustomisasi lebih, Anda perlu mengimplementasikan polanya secara manual.

Pola manualnya adalah sebagai berikut:

1. **Gunakan jHttpClient di Thread Terpisah:** Lakukan permintaan GET ke URL gambar di dalam sebuah TThread (akan dibahas di Bab 9). Respons yang berhasil tidak akan berupa responseText, melainkan responseStream (aliran data biner).
2. **Konversi Stream ke Bitmap:** Di dalam *thread* pekerja, setelah responseStream didapat, gunakan jBitmapFactory.DecodeStream(responseStream) untuk mengubahnya menjadi objek jBitmap.
3. **Update UI di Thread Utama:** Kirim objek jBitmap yang sudah jadi kembali ke *thread* utama (menggunakan TThread.Queue), lalu panggil jImageView1.SetImageBitmap(myBitmap) untuk menampilkannya.

Ini adalah proses yang lebih rumit, yang menunjukkan betapa bergunanya metode SetImageFromURL yang sudah disediakan oleh LAMW.

**Catatan tentang Caching dan Library Profesional**

Di aplikasi dunia nyata yang sangat bergantung pada gambar (seperti Instagram atau Pinterest), setiap kali memuat gambar dari URL akan memboroskan kuota data pengguna dan waktu. Solusinya adalah **caching**—menyimpan salinan gambar yang sudah diunduh di perangkat, sehingga saat gambar yang sama perlu ditampilkan lagi, aplikasi bisa mengambilnya dari penyimpanan lokal yang jauh lebih cepat.

Di ekosistem Android (Java/Kotlin), ada *library* pihak ketiga yang sangat canggih untuk ini, seperti **Glide** dan **Picasso**. *Library* ini secara otomatis menangani *caching* (baik di memori maupun di disk), optimasi ukuran gambar, dan banyak lagi. Untuk menggunakan *library* sekuat ini di LAMW, Anda perlu menggunakan teknik JNI yang akan kita pelajari di bab-bab selanjutnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 7.6.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 7.6: Penanganan Error Jaringan (Tidak Ada Koneksi, Server Error, dll)**.

**7.6. Penanganan Error Jaringan (Tidak Ada Koneksi, Server Error, dll)**

Sebuah aplikasi yang hebat tidak hanya bekerja dengan baik saat semua kondisi ideal. Ia juga harus tangguh dan bisa menangani masalah dengan elegan saat kondisi tidak ideal—misalnya, saat koneksi internet terputus atau server tujuan sedang bermasalah. Membuat aplikasi yang *crash* atau menampilkan pesan error yang membingungkan adalah cara tercepat untuk membuat pengguna meng-uninstall aplikasi Anda.

Di subbab ini, kita akan belajar beberapa teknik 'pemrograman defensif' untuk menangani *error* jaringan.

**1. Pertahanan Pertama: Periksa Koneksi Sebelum Meminta**

Langkah pertahanan pertama adalah yang paling sederhana: jangan melakukan permintaan jaringan jika perangkat tidak terhubung ke internet. LAMW menyediakan fungsi *helper* untuk memeriksa ini dengan mudah.

* **Contoh Kode:**

Delphi

procedure TAndroidModule1.btnMuatDataClick(Sender: TObject);

begin

// Periksa koneksi terlebih dahulu

if not jConnectivityManager.IsConnected then

begin

ShowMessage('Tidak ada koneksi internet. Silakan periksa pengaturan jaringan Anda.');

Exit; // Hentikan eksekusi prosedur, jangan lanjutkan

end;

// Jika ada koneksi, baru lanjutkan proses...

jProgressBar1.Visible := True;

jHttpClient1.SetURL('https://jsonplaceholder.typicode.com/posts');

jHttpClient1.Execute;

end;

Dengan menambahkan pengecekan jConnectivityManager.IsConnected di awal, kita bisa langsung memberikan umpan balik yang cepat kepada pengguna dan menghindari upaya permintaan jaringan yang sudah pasti akan gagal.

**2. Menangani Kode Status HTTP**

Terkadang, koneksi internet ada dan permintaan berhasil mencapai server, tetapi server merespons dengan sebuah *error*. Setiap respons HTTP memiliki sebuah **kode status**. Kita sudah terbiasa dengan kode 200 yang berarti 'OK/Sukses'. Kita harus siap menangani kode lainnya.

* **Kode 4xx (Client Error):** Kesalahan ada di pihak kita (aplikasi klien).
  + 404 Not Found: *Endpoint* atau sumber daya yang kita minta tidak ada di server.
  + 401 Unauthorized / 403 Forbidden: Kita tidak memiliki izin (atau Kunci API kita salah) untuk mengakses sumber daya ini.
* **Kode 5xx (Server Error):** Kesalahan ada di pihak server.
  + 500 Internal Server Error: Terjadi masalah umum di server.
  + 503 Service Unavailable: Server sedang *down* atau kelebihan beban.

Kita bisa memperluas logika di *event handler* OnHttpRequestCompleted kita untuk memberikan pesan yang lebih spesifik kepada pengguna.

Delphi

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject;

responseCode: integer; responseText: string);

begin

// ... (sembunyikan loading bar, aktifkan tombol, dll.) ...

case responseCode of

200: // Sukses

begin

// ... (proses parsing JSON seperti biasa) ...

end;

404:

ShowMessage('Error: Sumber daya yang Anda minta tidak ditemukan (404).');

401, 403:

ShowMessage('Error: Anda tidak memiliki izin untuk mengakses data ini.');

500..599: // Rentang untuk semua jenis server error

ShowMessage('Saat ini sedang terjadi masalah pada server. Silakan coba lagi nanti.');

else

// Untuk error lainnya yang tidak terduga

ShowMessage('Terjadi kesalahan jaringan yang tidak diketahui. Kode: ' + IntToStr(responseCode));

end;

end;

**3. Menggunakan try...except untuk Kesalahan Tak Terduga**

Bagaimana jika terjadi *error* sebelum respons diterima, misalnya, nama domain tidak bisa ditemukan (*UnknownHostException*) atau waktu tunggu habis (*Timeout*)? Atau bagaimana jika data JSON yang diterima dari server ternyata formatnya rusak? Untuk menangani ini, kita perlu membungkus kode yang berpotensi gagal di dalam blok try...except.

* **Contoh pada Parsing JSON:**

Delphi

// ... di dalam OnHttpRequestCompleted, setelah memeriksa responseCode = 200

try

// Kode parsing JSON yang mungkin gagal jika formatnya tidak valid

jsonValue := GetJSON(responseText);

// ...

except

on E: Exception do

ShowMessage('Gagal memproses data dari server. Error: ' + E.Message);

end;

Blok try...except akan 'menangkap' *exception* atau *crash* apa pun yang terjadi di dalamnya, mencegah aplikasi Anda tertutup secara paksa, dan memungkinkan Anda untuk menampilkan pesan *error* yang lebih terkontrol kepada pengguna.

Ini mengakhiri Bab 7. Anda sekarang memiliki kemampuan lengkap untuk membuat aplikasi yang terhubung dengan internet secara aman dan tangguh. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 8 (Pengantar Bab 8).

YAHOX

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 8.

Berikut adalah jawaban untuk prompt **Pengantar Bab 8**.

**Bab 8: Komponen Canggih, Multimedia, dan Notifikasi**

Selamat, Anda telah menguasai pilar-pilar fundamental pengembangan aplikasi: antarmuka, navigasi, data lokal, dan koneksi jaringan. Sekarang, saatnya kita naik level. Sebuah aplikasi yang hebat tidak hanya fungsional, tetapi juga memberikan pengalaman yang kaya, interaktif, dan terintegrasi dengan baik dengan kemampuan perangkat yang digunakannya.

Di bab ini, kita akan mempelajari cara menambahkan 'sihir' dan 'faktor wow' ke dalam aplikasi kita, mengubahnya dari sekadar alat menjadi sebuah pengalaman yang menyenangkan dan bermanfaat bagi pengguna. Kita tidak lagi hanya menampilkan data, tetapi menyajikannya dengan cara yang lebih menarik dan berinteraksi lebih dalam dengan ekosistem Android.

Anggap bab ini sebagai tur untuk mengenal fitur-fitur canggih yang akan membuat aplikasi Anda menonjol. Kita akan menjelajahi berbagai area, di antaranya:

* **Daftar yang Lebih Hidup:** Kita akan belajar mengubah jListView yang sederhana menjadi daftar yang indah dan informatif dengan *layout* kustom.
* **Integrasi Dunia Web:** Kita akan menyematkan konten web langsung di dalam aplikasi menggunakan jWebView.
* **Memanfaatkan Perangkat Keras:** Kita akan belajar cara mengakses perangkat keras esensial seperti **Kamera** untuk mengambil gambar dan sensor **GPS** untuk mengetahui lokasi perangkat.
* **Komunikasi Proaktif:** Kita akan mempelajari cara mengirim **Notifikasi Push** untuk memberitahu pengguna tentang informasi penting, bahkan saat aplikasi tidak sedang mereka buka.

Setiap subbab di sini akan membekali Anda dengan satu keahlian baru yang kuat. Dengan menggabungkan fitur-fitur ini, aplikasi yang Anda bangun akan terasa jauh lebih modern, profesional, dan terintegrasi penuh. Mari kita mulai tambahkan kekuatan-kekuatan baru ini ke dalam repertoar pengembangan Anda.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 8.1.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 8.1: Menguasai jListView dengan Adapter Kustom untuk Tampilan Kompleks**.

**8.1. Menguasai jListView dengan Adapter Kustom untuk Tampilan Kompleks**

Sejauh ini, jListView yang kita gunakan hanya mampu menampilkan satu baris teks sederhana. Ini cukup untuk daftar nama atau judul berita, tetapi bagaimana jika kita ingin menampilkan daftar yang lebih kaya? Misalnya, daftar kontak dengan foto di kiri, nama di atas, dan nomor telepon di bawahnya. Untuk ini, *adapter* bawaan tidak lagi cukup. Kita perlu membuat 'cetakan' dan 'pabrik' kita sendiri, yang disebut **Adapter Kustom (Custom Adapter)**.

Konsepnya adalah kita mendesain tampilan untuk satu baris, lalu kita membuat sebuah kelas 'pabrik' yang tahu cara mengambil data dan memasukkannya ke dalam cetakan tersebut, berulang-ulang untuk setiap item dalam daftar.

**Langkah 1: Mendesain 'Cetakan' Baris (Row Layout)**

Pertama, kita perlu mendesain tampilan untuk *satu baris* di dalam list kita.

1. Buat sebuah jForm baru di proyek Anda (File > New > LAMW jForm Android Module). Segera simpan dengan nama yang bermakna, misalnya uRowItemLayout.pas. Form ini tidak akan menjadi layar baru, melainkan hanya sebagai *template* visual.
2. Di desainer uRowItemLayout, atur jForm agar ukurannya kecil dan representatif untuk satu baris.
3. Tambahkan komponen-komponen yang Anda inginkan. Misalnya, sebuah jLinearLayout horizontal (ID: layoutBaris) yang berisi:
   * Sebuah jImageView di sebelah kiri (beri ID: imgItemFoto).
   * Sebuah jLinearLayout vertikal di sebelah kanannya, yang berisi dua jTextView (beri ID masing-masing: tvItemJudul dan tvItemSubJudul).

Desain ini akan menjadi 'cetakan' untuk setiap baris di jListView kita nanti.

**Langkah 2: Membangun 'Pabrik' (Kelas Adapter Kustom)**

Selanjutnya, kita akan membuat kelas Pascal yang berfungsi sebagai 'pabrik' atau adapter.

1. Buat sebuah unit Pascal baru (File > New Unit). Simpan dengan nama uMyCustomAdapter.pas.
2. Di dalam unit ini, deklarasikan sebuah kelas baru yang mewarisi dari jCustomAdapter.

Delphi

// Jangan lupa tambahkan unit layout baris kita ke uses clause

uses ..., uRowItemLayout;

type

// Definisikan juga record untuk menampung data kita

PMyDataItem = ^TMyDataItem;

TMyDataItem = record

Judul: string;

SubJudul: string;

UrlGambar: string;

end;

TMyCustomAdapter = class(jCustomAdapter)

private

FDataList: TList; // Untuk menyimpan daftar data kita

public

constructor Create(AContext: jContext; ADataList: TList);

destructor Destroy; override;

function GetCount: integer; override;

function GetItem(position: integer): jObject; override;

// Metode paling penting:

function GetView(position: integer; convertView: jView; parent: jViewGroup): jView; override;

end;

**Langkah 3: Mengimplementasikan Logika 'Pabrik' (GetView)**

Metode GetView adalah jantung dari adapter kita. Metode ini akan dipanggil oleh jListView untuk **setiap baris yang perlu ditampilkan di layar**. Tugas kita adalah mengembalikan sebuah tampilan (jView) yang sudah terisi data untuk posisi yang diminta.

Delphi

function TMyCustomAdapter.GetView(position: integer; convertView: jView; parent: jViewGroup): jView;

var

rowView: jView;

rowLayout: TAndroidModule1\_jForm2; // Tipe form layout baris kita

imgFoto: jImageView;

tvJudul, tvSubJudul: jTextView;

dataItem: PMyDataItem;

begin

// 1. Daur Ulang Tampilan (Penting untuk Performa)

if convertView = nil then

begin

// Jika belum ada tampilan yang bisa didaur ulang, buat 'cetakan' baru

rowLayout := TAndroidModule1\_jForm2.Create(Self);

rowLayout.layoutBaris.Parent := nil; // Lepaskan layout dari form template

rowView := rowLayout.layoutBaris;

end

else

rowView := convertView;

// 2. Ambil komponen UI dari dalam layout baris menggunakan ID-nya

imgFoto := jImageView(rowView.FindViewById(res.id.imgItemFoto));

tvJudul := jTextView(rowView.FindViewById(res.id.tvItemJudul));

tvSubJudul := jTextView(rowView.FindViewById(res.id.tvItemSubJudul));

// 3. Ambil data untuk posisi (baris) ini

dataItem := PMyDataItem(FDataList.Items[position]);

// 4. Isi komponen UI dengan data

tvJudul.Text := dataItem^.Judul;

tvSubJudul.Text := dataItem^.SubJudul;

if Assigned(imgFoto) then

imgFoto.SetImageFromURL(dataItem^.UrlGambar); // Contoh memuat gambar

// 5. Kembalikan tampilan baris yang sudah jadi

Result := rowView;

end;

**Catatan Performa (Pola ViewHolder):** Pemanggilan FindViewById berulang kali bisa memperlambat proses *scrolling*. Praktik terbaik di Android adalah menggunakan **Pola ViewHolder**, yaitu sebuah objek kecil untuk 'menyimpan' referensi ke komponen-komponen UI (imgFoto, tvJudul, dll.) saat baris pertama kali dibuat, sehingga tidak perlu dicari lagi saat di-daur ulang.

**Langkah 4: Menggunakan Adapter Kustom Anda**

Di form utama Anda, setelah Anda memiliki data dalam sebuah TList, Anda tinggal membuat *instance* dari adapter Anda dan menerapkannya ke jListView.

Delphi

var

myDataList: TList; // Anggap sudah diisi dengan PMyDataItem

myAdapter: TMyCustomAdapter;

begin

// ... (kode untuk mengisi myDataList dengan data) ...

// Buat instance adapter kita

myAdapter := TMyCustomAdapter.Create(Self, myDataList);

// Terapkan adapter ke ListView

lvBerita.Adapter := myAdapter;

end;

Dengan cara ini, Anda bisa membuat tampilan daftar yang sangat kompleks dan informatif, terbatas hanya oleh imajinasi Anda.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 8.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 8.2: Menggunakan jWebView untuk Mengintegrasikan Konten Web**.

**8.2. Menggunakan jWebView untuk Mengintegrasikan Konten Web**

jWebView adalah salah satu komponen paling serbaguna. Ia pada dasarnya adalah sebuah 'browser mini' yang bisa Anda sematkan di dalam *layout* aplikasi Anda. Ini membuka banyak sekali kemungkinan tanpa harus memaksa pengguna untuk membuka aplikasi browser eksternal seperti Chrome atau Firefox.

**Kapan Sebaiknya Menggunakan jWebView?**

Anda akan sering menggunakannya untuk:

* Menampilkan halaman statis seperti **'Bantuan', 'Tentang Kami', atau 'Kebijakan Privasi'** yang lebih mudah dibuat dan dipelihara dalam format HTML.
* Menampilkan **konten dinamis dari web**, seperti sebuah artikel berita atau postingan blog, tanpa perlu mem-parse dan menatanya ulang dengan komponen *native*.
* Mengintegrasikan bagian dari **aplikasi web yang sudah ada** ke dalam aplikasi *mobile* Anda.

**Contoh Penggunaan Praktis**

Berikut adalah beberapa cara umum untuk menggunakan jWebView.

**1. Memuat Halaman Web dari Internet**

Ini adalah penggunaan yang paling umum. Anda hanya perlu meletakkan komponen jWebView di form Anda dan memanggil metode LoadUrl dengan alamat web lengkap.

Delphi

procedure TAndroidModule1.btnBukaSitusClick(Sender: TObject);

begin

// Pastikan aplikasi Anda memiliki izin INTERNET di AndroidManifest.xml

jWebView1.LoadUrl('https://www.lazarus-ide.org/');

end;

**2. Memuat File HTML Lokal dari assets**

Terkadang, Anda ingin menyertakan halaman HTML langsung di dalam paket aplikasi Anda. Ini berguna untuk konten yang tidak memerlukan koneksi internet, seperti halaman bantuan.

* **Langkah Persiapan:** Pertama, di struktur proyek Anda (di *file explorer*), buat sebuah direktori baru di dalam res dengan nama assets. Letakkan file HTML Anda (misalnya, bantuan.html) di dalamnya.
* **Kode:** Panggil LoadUrl dengan format path khusus file:///android\_asset/...

Delphi

procedure TAndroidModule1.btnBukaBantuanLokalClick(Sender: TObject);

begin

jWebView1.LoadUrl('file:///android\_asset/bantuan.html');

end;

**3. Menampilkan String HTML Mentah**

Anda juga bisa membuat konten HTML secara dinamis di dalam kode Pascal Anda (misalnya, dari beberapa variabel) dan menampilkannya langsung.

Delphi

procedure TAndroidModule1.btnTampilHTMLDinamisClick(Sender: TObject);

var

htmlString: string;

begin

htmlString := '<html><body>' +

'<h1>Halo dari Kode Pascal!</h1>' +

'<p>Ini adalah paragraf yang dibuat secara <b>dinamis</b>.</p>' +

'</body></html>';

jWebView1.LoadData(htmlString, 'text/html', 'UTF-8');

end;

**4. Mengaktifkan JavaScript**

Demi keamanan, JavaScript dinonaktifkan secara *default* di jWebView. Jika halaman web yang Anda muat memerlukannya agar bisa berfungsi dengan baik (misalnya untuk interaktivitas, animasi, atau menampilkan grafik), Anda harus mengaktifkannya secara eksplisit.

* **Peringatan Keamanan:** Hanya aktifkan JavaScript jika Anda memuat konten dari sumber yang sepenuhnya Anda percayai.

Delphi

procedure TAndroidModule1.jForm1Create(Sender: TObject);

begin

// Praktik terbaik adalah mengaturnya saat form dibuat (OnCreate)

jWebView1.GetSettings.SetJavaScriptEnabled(True);

end;

Dengan menguasai beberapa metode ini, Anda dapat dengan mudah mengintegrasikan konten web yang kaya ke dalam aplikasi LAMW Anda.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 8.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 8.3: Mengakses Kamera dan Galeri dengan jCamera dan Intent**.

**8.3. Mengakses Kamera dan Galeri dengan Intent**

Banyak aplikasi membutuhkan kemampuan untuk menggunakan gambar, baik itu untuk foto profil, mengunggah bukti pembayaran, atau melaporkan kerusakan. Memberikan pengguna pilihan untuk mengambil foto baru atau memilih dari galeri mereka adalah fitur yang sangat fundamental.

Daripada membangun antarmuka kamera yang rumit dari nol, kita akan menggunakan cara yang lebih cerdas dan direkomendasikan oleh Android: **mendelegasikan tugas ini ke aplikasi lain melalui Intent**. Ini lebih aman, lebih mudah, dan menghormati aplikasi kamera atau galeri pilihan pengguna.

**Pola Kerja Umum: StartActivityForResult dan OnActivityResult**

Alur kerja untuk kedua skenario (kamera dan galeri) adalah sama:

1. **Buat Intent:** Kita membuat sebuah Intent yang mendeskripsikan apa yang kita inginkan (misalnya, 'ambil gambar' atau 'pilih konten').
2. **Jalankan Intent untuk Hasil:** Kita tidak memanggil StartActivity biasa, melainkan StartActivityForResult. Ini memberitahu Android, "Jalankan permintaan ini, dan setelah selesai, laporkan hasilnya kembali kepadaku."
3. **Tangani Hasil di OnActivityResult:** Setelah pengguna selesai (mengambil foto atau memilih gambar), sebuah *event* khusus bernama OnActivityResult akan dipicu di Form kita. Di sinilah kita akan menerima dan memproses hasilnya.

**Skenario 1: Mengambil Foto Baru dari Kamera**

**Persiapan:**

* **Izin:** Pastikan Anda sudah menambahkan <uses-permission android:name="android.permission.CAMERA" /> di AndroidManifest.xml dan sudah mengimplementasikan alur permintaan izin *runtime* untuk kamera seperti yang dibahas di Bab 6.

**Kode untuk Membuka Kamera:**

Delphi

const

REQUEST\_CODE\_CAMERA = 301; // Nomor unik untuk permintaan ini

procedure TAndroidModule1.btnBukaKameraClick(Sender: TObject);

var

intent: jIntent;

begin

// Pastikan Anda sudah memeriksa dan mendapatkan izin kamera di sini!

intent := jIntent.Create;

intent.SetAction(jMediaStore\_ACTION\_IMAGE\_CAPTURE);

if intent.ResolveActivity(GetPackageManager) <> nil then

StartActivityForResult(intent, REQUEST\_CODE\_CAMERA);

end;

**Kode untuk Menangani Hasil:** OnActivityResult adalah *event* dari jForm. Anda bisa membuatnya dari *Object Inspector* sama seperti OnCreate.

Delphi

procedure TAndroidModule1.OnActivityResult(requestCode: integer; resultCode: integer; data: jIntent);

var

imageBitmap: jBitmap;

begin

// Pastikan ini adalah respons dari permintaan kamera kita

if requestCode = REQUEST\_CODE\_CAMERA then

begin

// Pastikan pengguna menyelesaikan aksi (tidak menekan 'cancel')

if resultCode = jActivity.RESULT\_OK then

begin

if Assigned(data) and Assigned(data.GetExtras) then

begin

// Ambil gambar (biasanya berupa thumbnail) dari data 'extras'

imageBitmap := jBitmap(data.GetExtras.Get('data'));

if Assigned(imageBitmap) then

jImageView1.SetImageBitmap(imageBitmap); // Tampilkan di jImageView

end;

end;

end;

// ... (tambahkan blok 'else if' untuk permintaan lain di sini)

end;

**Skenario 2: Memilih Gambar dari Galeri**

Proses ini tidak memerlukan izin khusus (pada Android versi baru) karena kita menyerahkan pemilihan sepenuhnya kepada pengguna melalui aplikasi sistem.

**Kode untuk Membuka Galeri:**

Delphi

const

REQUEST\_CODE\_GALLERY = 302; // Nomor unik lainnya

procedure TAndroidModule1.btnBukaGaleriClick(Sender: TObject);

var

intent: jIntent;

begin

intent := jIntent.Create;

intent.SetAction(jIntent\_ACTION\_GET\_CONTENT);

intent.SetType('image/\*'); // Tentukan kita hanya ingin memilih file gambar

StartActivityForResult(intent, REQUEST\_CODE\_GALLERY);

end;

**Kode untuk Menangani Hasil:** Kita perlu menambahkan logika baru di OnActivityResult yang sudah ada untuk menangani respons dari galeri.

Delphi

procedure TAndroidModule1.OnActivityResult(requestCode: integer; resultCode: integer; data: jIntent);

// ... (isi dari contoh sebelumnya)

// ...

// Tambahkan pengecekan ini:

else if requestCode = REQUEST\_CODE\_GALLERY then

begin

if (resultCode = jActivity.RESULT\_OK) and Assigned(data) then

begin

// Hasil dari galeri adalah sebuah Uri (alamat ke file gambar)

// jImageView dapat langsung memuat gambar dari Uri

jImageView1.SetImageURI(data.GetData);

end;

end;

Perhatikan perbedaannya: kamera mengembalikan Bitmap kecil di dalam 'extras', sementara galeri mengembalikan Uri (alamat) ke gambar ukuran penuh, yang bisa langsung kita tampilkan menggunakan SetImageURI.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 8.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 8.4: Mengambil Data Lokasi GPS dengan jLocationSensor**.

**8.4. Mengambil Data Lokasi GPS dengan jLocationSensor**

Fitur berbasis lokasi adalah salah satu fitur paling kuat di aplikasi mobile, mulai dari aplikasi peta, ojek online, hingga media sosial yang bisa menandai lokasi sebuah postingan. Di LAMW, kita dapat dengan mudah mengakses data GPS dari perangkat menggunakan komponen jLocationSensor.

Namun, sebelum kita menyentuh komponen tersebut, ada dua hal yang **wajib** kita penuhi terlebih dahulu terkait privasi pengguna: **izin di Manifest dan izin Runtime**.

**Langkah 1: Persiapan Izin (Permissions)**

1. **Deklarasi di AndroidManifest.xml:** Pertama, buka file AndroidManifest.xml proyek Anda. Anda harus menambahkan setidaknya salah satu dari izin berikut, tergantung tingkat akurasi yang Anda butuhkan. Untuk GPS, kita membutuhkan akurasi tinggi.

XML

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

1. **Implementasi Izin Runtime:** Lokasi adalah 'izin berbahaya'. Oleh karena itu, Anda **wajib** mengimplementasikan alur permintaan izin *runtime* yang telah kita pelajari di Bab 6. Pastikan aplikasi Anda sudah memeriksa dan meminta izin Manifest\_permission\_ACCESS\_FINE\_LOCATION kepada pengguna sebelum mencoba mengaktifkan sensor.

**Langkah 2: Menggunakan jLocationSensor**

1. **Tambahkan Komponen:** Dari *Component Palette*, pilih jLocationSensor dan letakkan di form Anda. Komponen ini tidak akan terlihat saat aplikasi berjalan. Beri nama jLocationSensor1.
2. **Atur Properti:** Pilih jLocationSensor1. Di *Object Inspector*, perhatikan dua properti penting untuk mengontrol frekuensi *update* dan konsumsi baterai:
   * **MinTimeUpdate**: Interval waktu minimum (dalam milidetik) antara setiap *update* lokasi. Misalnya, 10000 berarti sensor hanya akan mencoba memberikan *update* paling cepat setiap 10 detik.
   * **MinDistanceUpdate**: Jarak minimum (dalam meter) yang harus ditempuh perangkat sebelum memberikan *update* lokasi. Misalnya, 10 berarti *update* baru akan dikirim jika perangkat bergeser setidaknya 10 meter dari lokasi *update* terakhir.

**Langkah 3: Menangani Update Lokasi**

Inti dari komponen ini adalah *event* OnLocationChanged. *Event* ini akan terpicu setiap kali sensor mendapatkan lokasi baru yang memenuhi kriteria MinTimeUpdate dan MinDistanceUpdate yang telah Anda atur.

Delphi

// Buat event OnLocationChanged dari Object Inspector untuk jLocationSensor1

procedure TAndroidModule1.jLocationSensor1LocationChanged(Sender: TObject;

ALatitude, ALongitude, AAltitude: double; ABearing, ASpeed: single);

begin

// Tampilkan koordinat yang diterima ke beberapa komponen jTextView

tvLatitude.Text := 'Latitude: ' + FloatToStr(ALatitude);

tvLongitude.Text := 'Longitude: ' + FloatToStr(ALongitude);

tvAltitude.Text := 'Altitude: ' + FloatToStr(AAltitude) + ' m';

tvSpeed.Text := 'Speed: ' + FloatToStr(ASpeed) + ' m/s';

end;

Parameter ALatitude dan ALongitude adalah data yang paling sering kita gunakan.

**Langkah 4: Mengelola Siklus Hidup Sensor (Praktik Terbaik)**

Sensor GPS sangat boros baterai. Kita tidak boleh membiarkannya menyala terus-menerus. Praktik terbaik adalah mengaktifkannya hanya saat layar aplikasi kita aktif, dan mematikannya saat aplikasi berada di latar belakang. Kita bisa memanfaatkan *event* siklus hidup Activity (jForm) yang telah kita pelajari di Bab 5.

Delphi

// Event OnResume dari jForm

procedure TAndroidModule1.jForm1Resume(Sender: TObject);

begin

// Mulai mendengarkan update lokasi saat aplikasi kembali aktif

// (setelah memastikan izin sudah ada)

if jPermissions.Check(Manifest\_permission\_ACCESS\_FINE\_LOCATION) then

jLocationSensor1.Start;

end;

// Event OnPause dari jForm

procedure TAndroidModule1.jForm1Pause(Sender: TObject);

begin

// Hentikan sensor saat aplikasi tidak lagi di latar depan untuk menghemat baterai

jLocationSensor1.Stop;

end;

Dengan arsitektur ini, aplikasi Anda akan menjadi jauh lebih efisien dan ramah baterai, yang merupakan salah satu ciri aplikasi yang profesional.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 8.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 8.5: Mengirim Notifikasi Push yang Kompatibel dengan Android Modern**.

**8.5. Mengirim Notifikasi Push yang Kompatibel dengan Android Modern**

Notifikasi adalah cara yang sangat efektif untuk berkomunikasi dengan pengguna, bahkan saat aplikasi Anda tidak sedang dibuka. Ini bisa digunakan untuk memberikan pengingat, memberitahu adanya pesan baru, atau menginformasikan selesainya sebuah proses di latar belakang.

Namun, sejak Android 8.0 (API 26), ada aturan baru yang wajib diikuti untuk menampilkan notifikasi: **Notification Channels** (Kanal Notifikasi).

**Konsep Wajib: Notification Channels**

Bayangkan aplikasi Anda memiliki berbagai jenis notifikasi: notifikasi untuk pesan baru, notifikasi untuk promosi, dan notifikasi untuk pengingat. Pengguna mungkin ingin menerima notifikasi pesan, tetapi tidak ingin diganggu oleh promosi. **Notification Channel** adalah cara kita mengelompokkan notifikasi-notifikasi ini.

Dengan membuat *channel* (misalnya, 'Channel Pesan' dan 'Channel Promosi'), kita memberikan pengguna kontrol untuk mengaktifkan atau menonaktifkan setiap jenis notifikasi secara terpisah melalui setelan sistem Android. **Membuat *channel* adalah langkah pertama yang wajib dilakukan sebelum Anda bisa mengirim notifikasi apa pun di Android modern.**

**Langkah-langkah Mengirim Notifikasi Lokal**

Prosesnya terdiri dari tiga langkah utama: membuat channel (hanya sekali), membangun notifikasi, dan menampilkannya.

**Langkah 1: Membuat Notification Channel (Lakukan Sekali Saja)**

Kode ini sebaiknya dijalankan saat aplikasi pertama kali dibuka, misalnya di OnCreate dari Form utama. Sistem akan mengabaikan panggilan ini jika *channel* dengan ID yang sama sudah pernah dibuat, jadi aman untuk memanggilnya setiap kali aplikasi dimulai.

Delphi

procedure TAndroidModule1.CreateNotificationChannel;

const

CHANNEL\_ID = 'channel\_pengingat\_01';

CHANNEL\_NAME = 'Pengingat Umum';

var

notificationManager: jNotificationManager;

channel: jNotificationChannel;

begin

// Notification Channel hanya diperlukan di Android 8.0 (API 26) ke atas

if GetSDKApiLevel < 26 then Exit;

notificationManager := jNotificationManager.Create(Self);

// Buat channel dengan ID unik, nama yang akan dilihat pengguna, dan tingkat kepentingan

channel := jNotificationChannel.Create(CHANNEL\_ID,

StrToJCharSequence(CHANNEL\_NAME),

jNotificationManager.IMPORTANCE\_DEFAULT);

channel.SetDescription('Channel untuk semua pengingat umum dari aplikasi.');

// Daftarkan channel ke sistem

notificationManager.CreateNotificationChannel(channel);

end;

**Langkah 2: Mendesain dan Membangun Notifikasi**

Setelah *channel* siap, kita bisa membuat notifikasi kapan pun kita mau, misalnya dari OnClick sebuah tombol.

Delphi

// Di dalam OnClick tombol 'Kirim Notifikasi'

const

CHANNEL\_ID = 'channel\_pengingat\_01'; // Harus sama dengan ID channel yang dibuat

NOTIFICATION\_ID = 101; // ID unik untuk notifikasi ini

var

notificationManager: jNotificationManager;

builder: jNotificationBuilder;

intent: jIntent;

pendingIntent: jPendingIntent;

begin

notificationManager := jNotificationManager.Create(Self);

// 2a. Buat Intent untuk membuka aplikasi saat notifikasi diklik

intent := GetPackageManager.GetLaunchIntentForPackage(GetPackageName);

intent.SetFlags(jIntent.FLAG\_ACTIVITY\_NEW\_TASK or jIntent.FLAG\_ACTIVITY\_CLEAR\_TASK);

pendingIntent := jPendingIntent.GetActivity(Self, 0, intent, jPendingIntent.FLAG\_IMMUTABLE);

// 2b. Gunakan NotificationBuilder untuk merancang notifikasi

builder := jNotificationBuilder.Create(Self, CHANNEL\_ID); // Wajib menyertakan Channel ID

builder.SetSmallIcon(res.drawable.ic\_notification); // Ganti dengan nama ikon Anda di folder drawable

builder.SetContentTitle(StrToJCharSequence('Tugas Baru'));

builder.SetContentText(StrToJCharSequence('Jangan lupa selesaikan laporan bulanan.'));

builder.SetPriority(jNotificationCompat.PRIORITY\_DEFAULT);

builder.SetContentIntent(pendingIntent); // Atur aksi saat notifikasi diklik

builder.SetAutoCancel(True); // Notifikasi akan hilang dari status bar setelah diklik

**Langkah 3: Menampilkan Notifikasi**

Langkah terakhir adalah memanggil notify untuk menampilkan notifikasi yang sudah kita bangun. Tambahkan baris ini di akhir kode dari langkah 2.

Delphi

// ... (melanjutkan kode dari langkah 2) ...

// 3. Tampilkan notifikasi ke pengguna

notificationManager.Notify(NOTIFICATION\_ID, builder.Build);

end;

**NOTIFICATION\_ID** adalah nomor unik untuk notifikasi spesifik ini. Jika Anda memanggil notify lagi dengan ID yang sama, notifikasi yang ada akan di-update. Jika Anda menggunakan ID yang berbeda, notifikasi baru akan muncul.

Ini mengakhiri Bab 8. Anda sekarang dapat menambahkan berbagai fitur canggih yang membuat aplikasi terasa lebih hidup dan profesional. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 9 (Pengantar Bab 9).

Yahok

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 9.

Berikut adalah jawaban untuk prompt **Pengantar Bab 9**.

**Bab 9: Threading, Tugas Latar Belakang, dan Kustomisasi**

Bayangkan antarmuka pengguna (UI) aplikasi Anda adalah seorang manajer di sebuah kantor yang sangat sibuk. Tugasnya adalah merespons setiap permintaan dari Anda (pengguna) dengan secepat mungkin: menampilkan menu, merespons klik tombol, dan menggulir daftar. Sekarang, bayangkan Anda memberikan tugas berat kepada sang manajer, seperti "Tolong unduh file 100MB ini" atau "Tolong proses ribuan data ini." Apa yang terjadi? Sang manajer akan sibuk mengerjakan tugas berat tersebut dan mengabaikan semua permintaan lain. Telepon (interaksi pengguna) tidak akan diangkat, dan seluruh kantor (aplikasi) akan 'macet' atau *freeze*. Inilah yang terjadi ketika kita membebani **Thread Utama (UI Thread)** dengan pekerjaan berat.

Seorang manajer yang cerdas akan mendelegasikan tugas-tugas berat kepada para 'pekerja' di belakang layar. Di dunia pemrograman, para pekerja ini adalah **Background Threads** dan **Services**. Mereka bekerja secara terpisah tanpa mengganggu sang manajer, dan hanya akan melapor kembali setelah tugasnya selesai. Di bab ini, kita akan mempelajari seni delegasi ini. Kita akan beralih dari sekadar membuat fitur menjadi membuat fitur yang **berperforma tinggi dan responsif**—sebuah ciri khas aplikasi profesional.

Perjalanan kita di bab ini akan mencakup:

* **Memahami UI Thread:** Mengapa ia begitu istimewa dan tidak boleh diganggu.
* **Bekerja dengan TThread:** Belajar mendelegasikan tugas berat ke latar belakang agar UI tetap lancar.
* **Strategi Tugas Jangka Panjang:** Mengenal *Services* untuk pekerjaan yang harus tetap berjalan bahkan saat aplikasi tidak di depan.
* **Debugging Tingkat Lanjut:** Menggunakan *Logcat* untuk 'mengintip' apa yang sedang dilakukan oleh para 'pekerja' di latar belakang.
* **Puncak Keahlian: Komponen Kustom:** Belajar membuat 'perkakas' kita sendiri dengan merancang komponen visual yang dapat digunakan kembali.

Bab ini bersifat sangat teknis, namun penguasaan konsep-konsep ini akan secara fundamental mengubah kualitas aplikasi yang Anda bangun. Mari kita mulai.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 9.1.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 9.1: Menjaga UI Tetap Responsif dengan Threading**.

**9.1. Menjaga UI Tetap Responsif dengan Threading**

Pusat dari semua interaksi di aplikasi Android adalah **Thread Utama**, yang lebih dikenal sebagai **UI Thread**. Ini adalah satu-satunya jalur eksekusi yang bertanggung jawab untuk menggambar semua yang Anda lihat di layar (tombol, teks, gambar) dan menangani semua input dari Anda (sentuhan, klik, gesekan). Karena tanggung jawabnya yang sangat penting ini, ada satu aturan emas yang tidak boleh dilanggar oleh developer profesional: **JANGAN PERNAH MEMBLOKIR UI THREAD**.

**Masalahnya: Aplikasi yang 'Freeze' (Membeku)**

Setiap operasi yang memakan waktu lebih dari sepersekian detik—seperti operasi jaringan, pemrosesan file yang besar, atau kalkulasi yang rumit—jika diletakkan di UI Thread, akan menyebabkan bencana.

Lihatlah kode yang terlihat tidak berbahaya ini, yang diletakkan di dalam OnClick sebuah tombol:

Delphi

procedure TAndroidModule1.btnProsesDataClick(Sender: TObject);

begin

// Simulasi tugas berat selama 5 detik

ShowMessage('Memulai proses berat...');

Sleep(5000); // <-- INI ADALAH BIANG MASALAH!

ShowMessage('Proses selesai!');

end;

Jika Anda menjalankan kode ini, setelah Anda menekan tombol, seluruh aplikasi Anda akan macet total selama 5 detik. Anda tidak bisa menekan tombol lain, tidak bisa *scroll*, tidak bisa melakukan apa pun. Kemungkinan besar, sistem Android akan menampilkan dialog yang ditakuti semua developer: **'Application Not Responding' (ANR)**. Ini terjadi karena Sleep(5000) memblokir UI Thread, sang 'manajer' yang seharusnya melayani Anda.

**Solusinya: Mendelegasikan Tugas ke TThread**

Solusi yang benar adalah mendelegasikan tugas berat ini ke 'pekerja' di latar belakang menggunakan kelas TThread bawaan dari Free Pascal. Dengan begitu, UI Thread bisa tetap bebas untuk melayani pengguna.

**Pola Implementasi TThread**

**1. Buat Kelas Thread Pekerja** Pertama, kita definisikan sebuah kelas baru yang mewarisi dari TThread. Kelas inilah yang akan menjadi cetak biru untuk 'pekerja' kita.

Delphi

type

TProsesBeratThread = class(TThread)

protected

procedure Execute; override;

public

// constructor Create sudah diwariskan

end;

**2. Letakkan Tugas Berat di Dalam Execute** Metode Execute adalah jantung dari *thread*. Semua kode yang berjalan lama dan berat harus diletakkan di sini. Kode di dalam Execute akan berjalan di *thread* terpisah dan tidak akan mengganggu UI.

**3. Komunikasi Kembali ke UI Thread (Sinkronisasi)** Ini adalah bagian yang sangat penting. **Thread pekerja di latar belakang TIDAK BOLEH menyentuh atau memanipulasi komponen UI secara langsung.** Jika mencoba, aplikasi Anda akan *crash*. Untuk mengirim hasil kembali ke UI, kita menggunakan TThread.Queue.

Delphi

procedure TProsesBeratThread.Execute;

begin

// Kode ini sekarang berjalan aman di latar belakang!

Sleep(5000);

// Setelah selesai, kirim sebuah prosedur 'tugas kecil' kembali ke UI Thread

// untuk menampilkan hasilnya.

TThread.Queue(nil,

procedure

begin

// Kode di dalam sini akan dieksekusi dengan aman di UI Thread

ShowMessage('Proses di latar belakang telah selesai!');

end);

end;

TThread.Queue adalah cara aman untuk mengirim sebuah 'pesan' atau prosedur dari *thread* pekerja untuk dieksekusi oleh UI Thread.

**4. Menjalankan Thread dari UI** Terakhir, kita ubah kode OnClick kita untuk membuat dan memulai 'pekerja' baru, bukan mengerjakan tugasnya sendiri.

Delphi

procedure TAndroidModule1.btnProsesDataClick(Sender: TObject);

begin

ShowMessage('Memulai proses di latar belakang...');

// Buat dan jalankan thread. Parameter 'False' berarti thread langsung berjalan.

TProsesBeratThread.Create(False);

// Perhatikan, setelah baris ini dieksekusi, UI Anda tetap responsif!

end;

Sekarang, saat tombol ditekan, aplikasi akan menampilkan pesan 'Memulai...', lalu UI akan tetap lancar. Lima detik kemudian, pesan 'Proses... selesai!' akan muncul tanpa membuat aplikasi macet.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 9.2.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 9.2: Strategi Tugas Latar Belakang Modern (Background Tasks)**.

**9.2. Strategi Tugas Latar Belakang Modern (Background Tasks)**

TThread yang kita pelajari sebelumnya sangat hebat untuk tugas-tugas yang berjalan selama pengguna masih berada di layar tersebut. Tapi, apa yang terjadi jika pengguna menekan tombol Home atau menutup aplikasi saat proses unduhan file besar sedang berjalan? Jawabannya: proses itu kemungkinan besar akan ikut dihentikan oleh sistem, karena TThread terikat pada siklus hidup aplikasi yang sedang terlihat.

Untuk tugas-tugas yang harus tetap berjalan hingga tuntas, bahkan jika pengguna sudah beralih ke aplikasi lain, Android menyediakan komponen yang lebih kuat yang disebut **Service**.

**Apa itu Android Service?**

**Service** adalah komponen aplikasi yang dapat melakukan operasi jangka panjang di latar belakang dan tidak menyediakan antarmuka pengguna. Ia berjalan secara independen dari Activity (layar) Anda. Anggap saja Service sebagai 'pekerja setia' yang terus bekerja di 'ruang mesin' bahkan saat sang 'manajer' (UI) sedang tidak ada di tempat.

Kapan kita harus menggunakan Service?

* **Mengunduh File Besar:** Memastikan unduhan tidak terputus meskipun pengguna membuka aplikasi lain.
* **Sinkronisasi Data:** Secara periodik mengambil data baru dari server di latar belakang.
* **Memutar Musik:** Ini adalah contoh paling klasik. Musik harus tetap berputar meskipun pengguna sedang membalas pesan WhatsApp atau mengunci layar perangkatnya.

**Implementasi jService di LAMW**

1. **Menambahkan Modul Service:** Di Lazarus, klik File > New... dan pilih **LAMW jService Module**. Simpan unit baru ini, misalnya dengan nama uMyBackgroundService.pas.
2. **Siklus Hidup Service:** Sama seperti Activity, Service juga memiliki siklus hidup. Dua *event* yang paling penting adalah:
   * **OnStartCommand**: Ini adalah jantung dari *service*. Logika utama Anda—misalnya, memulai unduhan atau *loop* sinkronisasi—diletakkan di sini. *Event* ini akan dipanggil setiap kali sebuah Activity mengirim perintah startService.
   * **OnDestroy**: Dipanggil saat *service* dihentikan. Ini adalah tempat untuk melakukan pembersihan (*cleanup*).
3. **Contoh Praktis: Service Pencatat Log** Mari kita buat *service* sederhana yang saat dijalankan akan menulis pesan ke Logcat untuk membuktikan ia berjalan di latar belakang.

*Kode di dalam uMyBackgroundService.pas*:

Delphi

procedure TAndroidModule2.jService1StartCommand(Sender: TObject; intent: jIntent;

flags, startId: integer): integer;

begin

Log.d('MyService', 'Service sedang berjalan di latar belakang...');

// Di aplikasi nyata, Anda akan memulai TThread untuk pekerjaan berat di sini.

// START\_STICKY berarti: jika service dimatikan sistem karena kekurangan memori,

// coba jalankan kembali saat memori sudah tersedia.

Result := jService.START\_STICKY;

end;

procedure TAndroidModule2.jService1Destroy(Sender: TObject);

begin

Log.d('MyService', 'Service telah dihentikan.');

end;

1. **Memulai Service dari Activity** Untuk memulai *service*, kita menggunakan Intent dari Activity kita (misalnya, dari OnClick sebuah tombol). Jangan lupa mendaftarkan unit uMyBackgroundService di *uses clause* file proyek (.lpr).

*Kode di dalam uMainForm.pas*:

Delphi

procedure TAndroidModule1.btnStartServiceClick(Sender: TObject);

var

intent: jIntent;

begin

intent := jIntent.Create;

// Beritahu Intent untuk menargetkan kelas service kita

intent.SetComponent(Self, 'uMyBackgroundService');

Self.StartService(intent);

intent.Free;

ShowMessage('Perintah untuk memulai service telah dikirim.');

end;

**Konteks Modern (2025): WorkManager**

Meskipun Service sangat kuat, untuk tugas-tugas modern yang bisa ditunda (misalnya, 'lakukan sinkronisasi data kapan pun ada koneksi Wi-Fi dan baterai sedang diisi'), Android sekarang sangat merekomendasikan penggunaan **WorkManager**.

WorkManager adalah pustaka yang lebih cerdas dan efisien dalam hal penggunaan baterai. Ia menjamin tugas Anda akan dijalankan dalam kondisi optimal, bahkan jika aplikasi ditutup atau perangkat di-*restart*. Saat ini, cara terbaik untuk menggunakan WorkManager dari LAMW adalah melalui **JNI (Java Native Interface)**, sebuah teknik yang akan kita pelajari di bab selanjutnya untuk mengakses *library* Java/Kotlin canggih.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 9.3.

Sumber

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 9.3: Teknik Debugging Tingkat Lanjut: Menggunakan Log.d dan Menganalisis Logcat**.

**9.3. Teknik Debugging Tingkat Lanjut: Menggunakan Log.d dan Menganalisis Logcat**

Ketika Anda bekerja dengan *threads* dan *services*, prosesnya berjalan 'tak terlihat' di latar belakang. Lalu, bagaimana kita tahu apa yang sedang terjadi? Apakah *thread* kita sudah mulai? Apakah *service* kita menerima perintah? Apakah sebuah variabel memiliki nilai yang benar di tengah proses? Menggunakan ShowMessage tidak lagi praktis dan efisien. Di sinilah kita membutuhkan 'mata-mata' atau 'jendela' ke dalam proses internal aplikasi kita. Alat untuk ini adalah **Logcat**.

**1. Mencetak Pesan ke Log**

Android menyediakan sebuah kelas Log untuk mengirim pesan dari kode kita ke sebuah sistem pencatatan log terpusat di perangkat. LAMW telah menyediakan *wrapper* yang mudah untuk ini.

**Tingkatan Log dan Penggunaannya:** Ada beberapa 'level' atau 'prioritas' log. Yang paling umum kita gunakan adalah:

* **Log.d (Debug):** Pesan untuk *debugging* selama masa pengembangan (misalnya, melacak nilai variabel).
* **Log.i (Info):** Pesan informatif tentang alur kerja normal aplikasi (misalnya, 'Koneksi berhasil dibuat').
* **Log.w (Warning):** Untuk menandai potensi masalah yang tidak menyebabkan *crash*.
* **Log.e (Error):** Untuk mencatat *error* atau *exception* yang terjadi.

**Konsep 'TAG':** Setiap pesan log sebaiknya disertai sebuah **TAG**, yaitu sebuah *string* singkat yang berfungsi sebagai label atau kategori. Ini akan sangat mempermudah kita saat memfilter ribuan pesan log. Praktik umumnya adalah mendefinisikan TAG sebagai konstanta di awal unit.

**Contoh Kode di Pascal:**

Delphi

const

TAG = 'MyAwesomeApp'; // Definisikan TAG untuk unit ini

procedure TAndroidModule1.btnLakukanSesuatuClick(Sender: TObject);

var

nilai: integer;

begin

nilai := 100;

Log.d(TAG, 'Tombol Lakukan Sesuatu diklik.');

Log.i(TAG, 'Memulai proses dengan nilai = ' + IntToStr(nilai));

if nilai > 50 then

Log.w(TAG, 'Peringatan: Nilai melebihi ambang batas wajar.');

end;

**2. Menganalisis Pesan dengan Logcat di Android Studio**

Mencetak log tidak ada gunanya jika kita tidak bisa membacanya. Alat terbaik untuk ini adalah panel **Logcat** yang ada di dalam Android Studio.

**Langkah-langkah Menggunakan Logcat:**

1. **Hubungkan Perangkat dan Buka Logcat:** Pastikan aplikasi Anda berjalan di perangkat (fisik atau emulator) yang terhubung ke komputer. Buka Android Studio, dan di bar bagian bawah, klik pada tab **Logcat**.
2. **Pilih Perangkat dan Aplikasi Anda:** Di bagian atas panel Logcat, pastikan Anda telah memilih perangkat dan proses aplikasi yang benar. Nama prosesnya adalah *package name* aplikasi Anda (misal: com.namaanda.myapp).
3. **Memfilter Pesan (Bagian Terpenting):** Anda akan melihat banjir pesan dari seluruh sistem Android. Untuk fokus pada pesan dari aplikasi kita, gunakan kotak filter di bagian atas:
   * **Filter berdasarkan TAG:** Cara paling efektif. Ketik tag:MyAwesomeApp di kotak filter. Sekarang, Logcat hanya akan menampilkan pesan yang kita kirim dengan TAG tersebut.
   * **Filter berdasarkan Level:** Di sebelah kanan kotak filter, ada menu *dropdown* untuk level (misalnya, 'Debug', 'Info', 'Warn'). Ini berguna untuk menyaring pesan berdasarkan tingkat kepentingannya.

*(Catatan untuk penulis: Screenshot panel Logcat di Android Studio dengan anotasi untuk pemilihan perangkat, kotak filter TAG, dan filter level akan sangat membantu pembaca).*

**3. Penerapan Praktis pada Contoh Sebelumnya**

Mari kita terapkan teknik ini pada *service* yang kita buat di subbab sebelumnya untuk melihat kapan ia dimulai dan dihentikan.

Modifikasi kode uMyBackgroundService.pas:

Delphi

// di uMyBackgroundService.pas

const

TAG = 'MyBackgroundService';

procedure TAndroidModule2.jService1StartCommand(...);

begin

Log.i(TAG, 'OnStartCommand dipanggil. Service dimulai.');

//...

end;

procedure TAndroidModule2.jService1Destroy(Sender: TObject);

begin

Log.w(TAG, 'OnDestroy dipanggil. Service akan dihentikan.');

end;

Sekarang, jalankan aplikasi Anda dan tekan tombol untuk memulai *service*. Buka Logcat di Android Studio, filter dengan tag:MyBackgroundService, dan Anda akan melihat dengan jelas kapan *service* Anda dimulai dan dihentikan. Ini adalah alat *debugging* yang tak ternilai untuk memahami alur kerja aplikasi Anda, terutama untuk proses di latar belakang.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 9.4.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 9.4: Membuat Komponen Visual Kustom Anda Sendiri**.

**9.4. Membuat Komponen Visual Kustom Anda Sendiri**

Sejauh ini, kita telah menggunakan komponen-komponen yang disediakan oleh LAMW. Namun, terkadang kita membutuhkan sebuah komponen yang memiliki perilaku atau tampilan yang sangat spesifik dan kita ingin menggunakannya berulang kali di banyak layar. Di sinilah letak puncak keahlian seorang developer: **membuat komponen kustom**.

Tujuannya adalah untuk **enkapsulasi** (menggabungkan beberapa komponen dan logikanya menjadi satu) dan **reusabilitas** (dapat digunakan kembali dengan mudah). Ini membuat kode utama kita lebih bersih dan proses desain di *Form Designer* menjadi jauh lebih cepat.

**Studi Kasus: Komponen jImageButtonWithCaption**

Bayangkan kita sering membutuhkan sebuah tombol yang memiliki ikon di atas dan sebuah label teks di bawahnya. Daripada harus menyusun jImageView dan jTextView di dalam jLinearLayout berulang kali, kita akan membuat satu komponen baru bernama TjImageButtonWithCaption yang bisa langsung kita *drag-and-drop*.

**Langkah 1: Membuat Unit dan Kelas Komponen**

1. Buat sebuah unit Pascal baru (File > New Unit). Simpan dengan nama uCustomComponents.pas.
2. Deklarasikan kelas baru kita. Karena kita ingin menggabungkan beberapa komponen secara vertikal, basis terbaik untuk komponen kita adalah jLinearLayout.

Delphi

unit uCustomComponents;

// ...

type

TjImageButtonWithCaption = class(jLinearLayout)

private

FImageView: jImageView;

FTextView: jTextView;

// Prosedur 'setter' dan 'getter' untuk properti kita

procedure SetCaption(AValue: string);

function GetCaption: string;

procedure SetImage(AValue: jBitmap);

public

constructor Create(AContext: jContext);

published

// Properti yang akan muncul di Object Inspector

property Caption: string read GetCaption write SetCaption;

property Image: jBitmap write SetImage;

end;

procedure Register; // Prosedur untuk mendaftarkan komponen ke IDE

**Langkah 2: Membangun UI di dalam Constructor**

Di dalam constructor, kita akan 'merakit' komponen-komponen internal kita (jImageView dan jTextView) secara terprogram.

Delphi

constructor TjImageButtonWithCaption.Create(AContext: jContext);

begin

inherited Create(AContext);

// Atur layout utama (diri sendiri) menjadi vertikal dan di tengah

Self.Orientation := loVertical;

Self.Gravity := gvCenter;

// Buat ImageView internal dan jadikan anak dari layout ini

FImageView := jImageView.Create(AContext);

FImageView.Parent := Self;

// Buat TextView internal dan jadikan anak dari layout ini

FTextView := jTextView.Create(AContext);

FTextView.Parent := Self;

FTextView.Gravity := gvCenter; // Teks juga di tengah

end;

**Langkah 3: Mengimplementasikan Properties**

Properti published adalah 'tombol-tombol' yang akan muncul di *Object Inspector*. Kita perlu menulis kode untuk prosedur *setter* dan *getter*-nya agar bisa mengubah komponen internal.

Delphi

procedure TjImageButtonWithCaption.SetCaption(AValue: string);

begin

FTextView.Text := AValue;

end;

function TjImageButtonWithCaption.GetCaption: string;

begin

Result := FTextView.Text;

end;

procedure TjImageButtonWithCaption.SetImage(AValue: jBitmap);

begin

FImageView.SetImageBitmap(AValue);

end;

**Langkah 4: Registrasi Komponen**

Agar Lazarus IDE mengenali komponen baru kita saat desain, kita harus mendaftarkannya di dalam prosedur Register.

Delphi

procedure Register;

begin

// Daftarkan komponen kita ke dalam tab palet bernama 'LAMW Custom'

RegisterComponents('LAMW Custom', [TjImageButtonWithCaption]);

end;

**Langkah 5: Instalasi sebagai Paket Desain**

Kode saja tidak cukup. Kita harus mengompilasi unit ini menjadi sebuah paket desain (.lpk) dan menginstalnya ke IDE.

1. Klik File > New... > Package. Sebuah jendela *Package Editor* akan muncul.
2. Simpan paket tersebut (File > Save). Beri nama, misalnya MyCustomLAMWComponents.lpk.
3. Di jendela *Package Editor*, klik Add... di sebelah kanan. Di dialog baru, klik tab Add Files dan pilih file uCustomComponents.pas kita. Klik Add unit(s).
4. Sekarang, klik tombol **Use > Install**. Lazarus akan menanyakan apakah Anda ingin membangun ulang IDE. Klik Yes.
5. Lazarus akan mengompilasi paket Anda dan me-restart dirinya sendiri.
6. Setelah Lazarus terbuka kembali, lihatlah di *Component Palette*. Anda akan menemukan sebuah tab baru bernama **LAMW Custom** dengan komponen TjImageButtonWithCaption Anda di dalamnya, siap untuk di-*drag-and-drop* ke form mana pun!

Ini mengakhiri Bab 9. Anda telah mempelajari konsep-konsep yang sangat kuat tentang performa, debugging, dan reusabilitas. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 10 (Pengantar Bab 10).

Yahok

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 10.

Berikut adalah jawaban untuk prompt **Pengantar Bab 10**.

**Bab 10: Interoperabilitas dengan Ekosistem Java/Kotlin**

Bayangkan Kerajaan Object Pascal yang telah kita bangun. Fondasinya kokoh, strukturnya elegan, dan performanya tinggi. Namun, sekuat apa pun kerajaan kita, ia tidak berdiri sendiri. Tepat di sebelahnya, ada sebuah Kekaisaran Java/Kotlin yang sangat luas dan megah, dipenuhi dengan jutaan 'harta karun' berupa *library* siap pakai untuk segala macam keperluan: dari grafik yang rumit, *machine learning*, hingga *toolkit* antarmuka resmi dari Google sendiri.

Bagaimana cara kita, sebagai penghuni Kerajaan Pascal, bisa memanfaatkan semua 'harta karun' tersebut tanpa harus meninggalkan kerajaan kita? Jawabannya adalah melalui 'jalur diplomasi' khusus yang memungkinkan kedua dunia ini berkomunikasi.

Jalur diplomasi ini dikenal dengan nama **JNI (Java Native Interface)**. JNI adalah sebuah jembatan teknologi yang memungkinkan kode *native* kita (hasil kompilasi Pascal) untuk berkomunikasi dua arah dengan kode yang berjalan di dunia Java/Kotlin. Mari kita jujur: topik ini adalah salah satu yang paling **mahir** dalam buku ini. Di dunia pengembangan Android standar, JNI sering dianggap rumit. Namun, kabar baiknya adalah LAMW telah melakukan banyak pekerjaan berat di belakang layar, menyediakan berbagai fungsi pembantu yang akan membuat proses ini jauh lebih mudah dipahami.

Di bab ini, kita akan menjadi seorang 'diplomat'. Kita akan belajar:

* **Memahami Jembatan JNI:** Mengenal konsep dasar di balik cara kerja jembatan ini dalam konteks LAMW.
* **Memanggil Metode Java:** Mempelajari resep langkah-demi-langkah untuk memanggil fungsi dari kode Java langsung dari dalam Pascal.
* **Studi Kasus Praktis:** Puncaknya, kita akan mempraktikkan semua ilmu diplomasi ini dengan mengintegrasikan sebuah *library* grafik pihak ketiga yang populer ke dalam aplikasi LAMW kita.

Menguasai bab ini akan memberikan Anda sebuah *superpower*: kemampuan untuk tidak lagi terbatas pada apa yang disediakan oleh LAMW, tetapi bisa memanfaatkan hampir semua *library* yang ada di seluruh ekosistem Android. Ini adalah kunci untuk membangun aplikasi yang benar-benar tanpa batas.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 10.1.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 10.1: Pengantar Java Native Interface (JNI) dalam Konteks LAMW**.

**10.1. Pengantar Java Native Interface (JNI) dalam Konteks LAMW**

Kita telah sepakat bahwa kita butuh 'jembatan diplomasi' untuk berkomunikasi dengan dunia Java/Kotlin. Nama resmi untuk jembatan tersebut adalah **JNI** atau **Java Native Interface**.

Secara definisi, JNI adalah sebuah kerangka kerja (*framework*) standar yang menjadi bagian dari platform Java. Ia menyediakan seperangkat aturan dan fungsi yang memungkinkan kode yang berjalan di dalam Java Virtual Machine (JVM)—dan di Android, ART (*Android Runtime*)—untuk berinteraksi dengan kode *native*. Kode *native* adalah kode yang telah dikompilasi menjadi instruksi mesin, seperti C, C++, dan tentu saja, hasil kompilasi Object Pascal dari FPC kita. JNI inilah yang menjadi fondasi dari seluruh arsitektur LAMW.

**LAMW sebagai Fasilitator (Penyederhana)**

Jika Anda mencari tutorial JNI di internet, Anda akan menemukan banyak contoh kode C/C++ yang terlihat rumit, panjang, dan rawan kesalahan hanya untuk memanggil satu fungsi sederhana. Di sinilah letak keunggulan dan kejeniusan LAMW. Ia telah melakukan sebagian besar pekerjaan 'kotor' untuk Anda:

* **Inisialisasi Otomatis:** Lingkungan JNI yang diperlukan untuk komunikasi sudah disiapkan secara otomatis oleh LAMW saat aplikasi Anda dimulai.
* **Unit Pembantu (*Helper Units*):** LAMW menyediakan unit-unit pustaka yang sangat penting, terutama Androidapi.JNI dan Androidapi.Helpers. Unit-unit ini berisi ratusan fungsi dan tipe data yang sudah 'diterjemahkan' ke dalam bahasa Pascal, sehingga kita tidak perlu berurusan langsung dengan sintaks JNI level rendah yang kompleks.
* **Tipe Data Jembatan:** Unit-unit tersebut menyediakan tipe data khusus seperti jstring, jint, jboolean, dan jobject yang berfungsi sebagai perantara antara tipe data Pascal (misal: string, integer) dan tipe data Java.

Singkatnya, LAMW mengubah panggilan JNI yang rumit menjadi pemanggilan fungsi Pascal yang terasa jauh lebih familiar dan aman.

**Konsep Kunci yang Perlu Diketahui**

Meskipun LAMW banyak membantu, ada beberapa konsep JNI yang tetap perlu kita ketahui agar bisa 'berbicara' dengan benar di jembatan ini.

1. **Referensi Objek (jobject):** Di dunia JNI, kita tidak pernah memegang objek Java secara langsung di dalam kode Pascal kita. Sebaliknya, kita memegang sebuah **'referensi' atau 'penunjuk'** ke objek tersebut. Di LAMW, referensi ini sering direpresentasikan dengan tipe jobject (untuk objek umum), atau tipe spesifik lainnya seperti jstring (untuk referensi ke objek String Java) atau jclass (untuk referensi ke sebuah Class Java).
2. **Signature Metode:** Bagaimana cara JNI mengetahui metode mana yang ingin kita panggil, terutama jika ada beberapa metode dengan nama yang sama tetapi parameter berbeda (*method overloading*)? Jawabannya adalah dengan **Signature**. Signature adalah sebuah *string* dengan format khusus yang secara unik mendeskripsikan parameter dan tipe kembalian sebuah metode.
   * Contoh: ()V artinya metode tanpa parameter (()) dan tidak mengembalikan apa-apa (V untuk void).
   * Contoh: (Ljava/lang/String;)I artinya metode yang menerima satu parameter bertipe String (Ljava/lang/String;) dan mengembalikan sebuah Integer (I).

Jangan khawatir jika ini terlihat rumit. Untungnya, saat kita memanggil metode nanti, sering kali kita bisa menggunakan nama metode dan parameternya secara langsung, dan fungsi *helper* dari LAMW akan membantu kita mencari *signature* yang tepat di belakang layar.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 10.2.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 10.2: Memanggil Metode Java dari Kode Pascal**.

**10.2. Memanggil Metode Java dari Kode Pascal**

Inilah inti dari interoperabilitas: memanggil fungsi atau mengakses data dari dunia Java langsung dari kode Pascal kita. Prosesnya selalu mengikuti sebuah pola atau 'resep' yang sama. Setelah Anda menguasai pola ini, Anda bisa menerapkannya untuk memanggil hampir semua metode Java.

Sebagai contoh pertama yang sangat sederhana, kita akan mencoba mengambil versi sistem operasi Android dari perangkat dengan mengakses *field* statis android.os.Build.VERSION.RELEASE.

**Pola Langkah-demi-Langkah Memanggil Kode Java**

Prosesnya dapat dipecah menjadi lima langkah utama:

1. **Dapatkan Referensi ke Class:** Pertama, kita harus memberitahu JNI, "Saya ingin berinteraksi dengan Class X". Kita perlu mendapatkan referensi ke *class* tersebut.
2. **Dapatkan ID dari Metode atau Field:** Setelah kita memiliki *class*-nya, kita perlu menunjuk ke *field* (variabel) atau *metode* (fungsi) spesifik yang kita inginkan. Setiap *field* dan *metode* memiliki ID unik di dalam JNI.
3. **Persiapkan Argumen:** Jika metode yang ingin kita panggil memerlukan parameter (misalnya, sebuah string atau integer), kita harus mengubah data dari tipe Pascal ke tipe JNI yang sesuai (misalnya, menggunakan StrToJString).
4. **Panggil Metode atau Akses Field:** Dengan referensi *class*, ID metode/field, dan argumen yang sudah siap, kita sekarang bisa mengeksekusi panggilannya.
5. **Proses Nilai yang Dikembalikan:** Jika panggilan tersebut mengembalikan sebuah nilai, nilai itu akan dalam format JNI. Kita perlu mengubahnya kembali ke tipe data Pascal agar bisa kita gunakan.

**Contoh Kode Lengkap**

Mari kita gabungkan semua langkah di atas ke dalam sebuah prosedur yang bisa dipanggil dari OnClick sebuah tombol untuk menampilkan versi Android.

Delphi

// Jangan lupa tambahkan Androidapi.JNI, Androidapi.Helpers ke uses clause di bagian implementation

uses

..., Androidapi.JNI, Androidapi.Helpers;

procedure TAndroidModule1.btnGetOSVersionClick(Sender: TObject);

var

\_classVERSION: JNIClass;

\_fieldRELEASE: JNIFieldID;

\_jStringRelease: JNIObject;

\_releaseVersion: string;

begin

try

// Langkah 1: Dapatkan referensi Class 'android.os.Build$VERSION'

// Catatan: '.' diganti '/' dan kelas inner dipisahkan dengan '$'

\_classVERSION := GetJClass('android/os/Build$VERSION');

if \_classVERSION = nil then raise Exception.Create('Class Build.VERSION tidak ditemukan');

// Langkah 2: Dapatkan ID dari field statis 'RELEASE' yang tipenya String

// 'Ljava/lang/String;' adalah signature untuk tipe data String di Java

\_fieldRELEASE := GetJStaticFieldID(\_classVERSION, 'RELEASE', 'Ljava/lang/String;');

if \_fieldRELEASE = nil then raise Exception.Create('Field RELEASE tidak ditemukan');

// Langkah 3: Persiapan Argumen (dilewati karena kita tidak memanggil metode dengan parameter)

// Langkah 4: Ambil nilai field statis tersebut. Karena hasilnya objek (String), kita gunakan GetJStaticObjectField

\_jStringRelease := GetJStaticObjectField(\_classVERSION, \_fieldRELEASE);

// Langkah 5: Ubah hasilnya dari JString ke string Pascal

\_releaseVersion := JStringToString(\_jStringRelease);

// Tampilkan hasilnya kepada pengguna

ShowMessage('Versi Android Perangkat Ini: ' + \_releaseVersion);

except

on E: Exception do

ShowMessage('Terjadi error JNI: ' + E.Message);

end;

end;

**Penjelasan Singkat:**

* Kita menggunakan GetJClass untuk mendapatkan referensi ke kelas Build.VERSION.
* Kita menggunakan GetJStaticFieldID untuk mendapatkan penunjuk ke *field* RELEASE.
* Kita menggunakan GetJStaticObjectField untuk mengambil nilai dari *field* tersebut, yang dikembalikan sebagai JNIObject (dalam hal ini, sebuah JString).
* Terakhir, kita menggunakan fungsi *helper* JStringToString untuk mengubahnya menjadi string Pascal yang bisa kita tampilkan.

Pola ini—**Get Class -> Get ID -> [Prepare Args] -> Call -> Process Result**—adalah resep yang akan Anda gunakan berulang kali. Dengan menguasai alur ini, Anda bisa mulai bereksperimen untuk memanggil metode-metode Java lainnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 10.3.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 10.3: Studi Kasus: Mengintegrasikan Library Android Pihak Ketiga (.jar atau .aar)**.

**10.3. Studi Kasus: Mengintegrasikan Library Android Pihak Ketiga**

Teori dan contoh sederhana sudah kita kuasai. Sekarang saatnya melihat kekuatan JNI yang sesungguhnya dengan melakukan studi kasus yang nyata dan sangat berguna: mengintegrasikan sebuah *library* grafik pihak ketiga yang populer, **MPAndroidChart**, ke dalam aplikasi LAMW kita.

Tujuannya adalah untuk menampilkan sebuah grafik garis (*line chart*) yang indah, yang dibuat sepenuhnya oleh *library* Java, tetapi dikontrol sepenuhnya dari kode Pascal kita. Ini akan membuktikan bahwa hampir tidak ada batasan untuk apa yang bisa kita capai dengan LAMW jika kita menguasai JNI.

**Konsep Utama: Arsitektur Wrapper**

Kita tidak akan memanggil JNI secara langsung di setiap bagian kode kita. Itu akan sangat merepotkan dan membuat kode sulit dibaca. Sebaliknya, kita akan menerapkan **arsitektur wrapper**. Kita akan membuat satu unit Pascal khusus (misalnya, uMPChartWrapper.pas) yang bertugas sebagai satu-satunya 'penerjemah'. Semua panggilan JNI yang rumit akan kita 'sembunyikan' di dalam unit ini. Dengan begitu, dari form utama kita, kita hanya perlu memanggil fungsi-fungsi Pascal yang bersih dan mudah dibaca.

**Langkah 1: Menambahkan Library MPAndroidChart ke Proyek**

1. **Unduh Library:** Kunjungi halaman rilis resmi dari MPAndroidChart di GitHub dan unduh file rilis terbarunya dalam format .jar.
2. **Tambahkan ke Proyek:** Di struktur direktori proyek LAMW Anda, temukan atau buat folder libs. Salin file .jar yang baru saja Anda unduh ke dalam folder ini.
3. **Pastikan Library Ter-load:** Pastikan pengaturan proyek LAMW Anda dikonfigurasi untuk menyertakan semua file .jar dari folder libs saat proses *build*. Biasanya ini sudah diatur secara default.

**Langkah 2: Membuat Unit dan Kelas-kelas Wrapper**

Buat sebuah unit Pascal baru, uMPChartWrapper.pas. Di dalamnya, kita akan mulai mendefinisikan kelas-kelas Pascal yang 'meniru' atau membungkus struktur kelas dari *library* MPAndroidChart.

Delphi

unit uMPChartWrapper;

//...

type

// Wrapper untuk class com.github.mikephil.charting.data.Entry

TjEntry = class(jObject)

public

constructor Create(x, y: single);

end;

// Wrapper untuk class com.github.mikephil.charting.data.LineDataSet

TjLineDataSet = class(jObject)

public

constructor Create(entries: jList; label: string);

procedure setColor(color: TColor);

end;

// Wrapper untuk class com.github.mikephil.charting.data.LineData

TjLineData = class(jObject) ... end;

// Wrapper untuk class com.github.mikephil.charting.charts.LineChart

TjLineChart = class(jView) // Mewarisi dari jView agar bisa ditampilkan

public

constructor Create(context: jContext);

procedure setData(data: TjLineData);

procedure invalidate;

end;

**Langkah 3: Mengimplementasikan Metode Wrapper (Contoh)**

Sekarang, kita implementasikan beberapa metode kunci. Setiap metode di dalam kelas Pascal kita akan berisi panggilan JNI untuk memanggil metode yang sesungguhnya di objek Java.

Delphi

// Contoh implementasi constructor untuk TjEntry

constructor TjEntry.Create(x, y: single);

var

\_class: JNIClass;

\_ctorID: JNIMethodID;

begin

\_class := GetJClass('com/github/mikephil/charting/data/Entry');

// Dapatkan constructor dengan parameter (float, float)

\_ctorID := GetJNIMethodID(\_class, '<init>', '(FF)V');

// Buat objek Java baru

FMe := GetJNIEnv.NewObject(\_class, \_ctorID, [jvalue(x), jvalue(y)]);

end;

// Contoh implementasi setColor untuk TjLineDataSet

procedure TjLineDataSet.setColor(color: TColor);

var

\_methodID: JNIMethodID;

begin

// Dapatkan metode 'setColor' dengan parameter (int)

\_methodID := GetJNIMethodID(GetJClass(FMe), 'setColor', '(I)V');

// Panggil metode tersebut di objek Java

GetJNIEnv.CallVoidMethod(FMe, \_methodID, [jvalue(color)]);

end;

Perhatikan polanya? Setiap fungsi Pascal yang kita buat menyembunyikan satu panggilan JNI yang spesifik. Kita lakukan ini untuk semua fungsi penting yang kita butuhkan.

**Langkah 4: Menggunakan Wrapper di Form Utama (Hasil Akhir)**

Setelah unit *wrapper* kita selesai (atau setidaknya cukup lengkap untuk satu contoh), menggunakannya di form utama menjadi sangat mudah dan bersih. Semua kerumitan JNI telah hilang.

* **UI Setup:** Anggap kita punya sebuah jFrameLayout bernama chartContainer di form kita sebagai tempat untuk menaruh grafik.
* **Kode di Form Utama:**

Delphi

// Jangan lupa tambahkan uMPChartWrapper ke uses clause

uses ..., uMPChartWrapper;

procedure TAndroidModule1.btnBuatChartClick(Sender: TObject);

var

lineChart: TjLineChart;

entries: jArrayList; // Gunakan jArrayList untuk kompatibilitas

dataSet: TjLineDataSet;

lineData: TjLineData;

i: integer;

begin

// 1. Buat objek Chart dan tambahkan ke layout

lineChart := TjLineChart.Create(Self);

chartContainer.AddView(lineChart);

// 2. Buat data dummy

entries := jArrayList.Create;

for i := 0 to 10 do

entries.Add(TjEntry.Create(i, Random \* 100)); // Gunakan constructor wrapper kita

// 3. Konfigurasi DataSet menggunakan wrapper kita

dataSet := TjLineDataSet.Create(entries, 'Data Penjualan');

dataSet.setColor(jColor.RED); // Panggil metode wrapper kita

// 4. Masukkan DataSet ke LineData

lineData := TjLineData.Create([dataSet]);

// 5. Tampilkan data di Chart

lineChart.setData(lineData);

lineChart.invalidate; // Perintahkan chart untuk menggambar ulang

end;

Lihatlah betapa bersihnya kode di atas! Kita berinteraksi seolah-olah sedang menggunakan komponen Pascal murni. Inilah kekuatan sesungguhnya dari arsitektur *wrapper* dan interoperabilitas JNI. Dengan teknik ini, hampir tidak ada *library* Android yang tidak bisa Anda taklukkan.

Ini mengakhiri Bab 10. Anda telah mempelajari salah satu keahlian paling fundamental yang membuka akses tak terbatas ke ekosistem Android. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 11 (Pengantar Bab 11).

Yahok

Tampilkan alur berpikir

Baik, kita sekarang memasuki Bab 11.

Berikut adalah jawaban untuk prompt **Pengantar Bab 11**.

**Bab 11: Studi Kasus: Membangun Aplikasi Lengkap**

Selamat datang di Bab 11. Jika bab-bab sebelumnya kita telah mengisi 'kotak perkakas' kita dengan berbagai macam keahlian—mulai dari palu (jButton), gergaji (jListView), hingga mesin bor canggih (JNI)—maka di bab ini, kita akan menggunakan semua perkakas tersebut untuk membangun sebuah 'rumah' yang utuh dari awal hingga akhir.

Ini adalah bab sintesis, di mana teori dan praktik parsial bertemu untuk menciptakan sebuah produk yang koheren dan fungsional. Ini adalah ujian akhir sekaligus pembuktian dari semua yang telah Anda pelajari. Kita tidak lagi hanya akan membuat satu fitur, tetapi merangkai banyak fitur menjadi sebuah alur aplikasi yang solid.

Untuk memastikan Anda menguasai dua domain utama pengembangan aplikasi, kita akan membangun dua studi kasus yang berbeda, masing-masing dengan tantangannya sendiri:

* **Proyek 1: Aplikasi "Daftar Tugas" (Todo List).** Ini adalah proyek klasik yang akan mengasah penguasaan Anda dalam membangun aplikasi *offline* yang andal. Fokus utamanya adalah pada **manajemen data lokal menggunakan database SQLite** dan menciptakan antarmuka yang interaktif untuk operasi **CRUD (Create, Read, Update, Delete)**.
* **Proyek 2: Aplikasi "Katalog Produk Online".** Proyek ini akan membawa kita ke dunia aplikasi *online* yang dinamis. Fokusnya adalah pada **komunikasi jaringan untuk mengambil data dari API**, **parsing data JSON**, dan menampilkannya dalam sebuah **daftar kustom yang kaya visual** (dengan gambar dan teks).

Setiap proyek akan memandu Anda dari tahap perencanaan, desain, implementasi, hingga sentuhan akhir. Tujuannya bukan hanya untuk memiliki dua aplikasi jadi di portofolio Anda. Tujuan utamanya adalah untuk memberikan Anda **pola pikir dan kepercayaan diri** seorang developer sejati. Setelah menyelesaikan bab ini, Anda akan melihat bahwa Anda tidak lagi hanya 'tahu caranya', tetapi Anda benar-benar 'bisa melakukannya'. Anda akan siap untuk mengambil ide Anda sendiri dan mulai mengubahnya menjadi kenyataan.

Mari kita mulai proyek pembangunan pertama kita.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 11.1.

Yahok

Tampilkan alur berpikir

Baik, kita mulai proyek pertama.

Berikut adalah jawaban untuk prompt **Subbab 11.1: Proyek 1: Aplikasi "Daftar Tugas" (Todo List)**.

**11.1. Proyek 1: Aplikasi "Daftar Tugas" (Todo List)**

Proyek pertama kita adalah membuat salah satu aplikasi paling fundamental dan berguna: aplikasi Daftar Tugas atau *Todo List*. Proyek ini akan menjadi ajang pembuktian pemahaman kita tentang desain antarmuka, manajemen data lokal dengan SQLite, dan penggunaan jListView dengan *adapter* kustom untuk menciptakan pengalaman pengguna yang baik.

**Tahap 1: Perencanaan dan Desain**

Sebelum menulis kode, kita rencanakan dulu apa yang akan kita bangun.

**1. Definisi Fitur:**

* Pengguna dapat mengetikkan tugas baru dan menyimpannya ke dalam database.
* Semua tugas yang tersimpan akan ditampilkan dalam sebuah daftar.
* Pengguna dapat mengetuk sebuah tugas untuk menandainya sebagai 'selesai' (atau sebaliknya).
* Tugas yang sudah selesai akan ditampilkan dengan efek coretan (*strikethrough*).
* Pengguna dapat menahan-klik sebuah tugas untuk menghapusnya secara permanen.

**2. Desain Database:** Kita akan menggunakan satu tabel SQLite bernama tasks dengan struktur sebagai berikut:

* id: INTEGER PRIMARY KEY AUTOINCREMENT (ID unik untuk setiap tugas)
* title: TEXT NOT NULL (Teks dari tugas itu sendiri)
* is\_completed: INTEGER DEFAULT 0 (Status tugas, 0 untuk belum selesai, 1 untuk sudah selesai)

**3. Desain Antarmuka (UI):** UI utama akan terdiri dari jLinearLayout vertikal yang berisi:

* Di bagian atas: sebuah jLinearLayout horizontal yang berisi jEditText untuk input tugas dan jButton 'Tambah'.
* Di bagian bawah: sebuah jListView yang akan memakan sisa layar untuk menampilkan daftar tugas.

**Tahap 2: Implementasi**

**A. Membuat Database Helper** Seperti yang kita pelajari di Bab 6, praktik terbaik adalah membuat unit Pascal baru (uDBHelper.pas) yang berisi kelas turunan dari jSQLiteOpenHelper. Di dalam metode onCreate-nya, kita eksekusi perintah SQL CREATE TABLE tasks ... sesuai dengan skema yang telah kita rancang.

**B. Membangun Logika Akses Data (CRUD)** Untuk menjaga kode tetap bersih, kita akan membuat satu unit lagi, uDataAccess.pas. Unit ini akan berisi semua fungsi untuk berinteraksi dengan database.

* function AddTask(const aTitle: string): boolean; -> Melakukan operasi INSERT.
* function GetAllTasks: TList; -> Melakukan SELECT \* FROM tasks, lalu mengisi TList dengan *record* data tugas.
* function UpdateTaskStatus(aTaskID: integer; aIsCompleted: boolean): boolean; -> Melakukan UPDATE.
* function DeleteTask(aTaskID: integer): boolean; -> Melakukan DELETE.

**C. Merangkai UI dan Logika** Di unit Form utama Anda, inilah cara kita menghubungkan semuanya:

1. **Menampilkan Data:** Di *event* OnCreate atau OnResume form, panggil uDataAccess.GetAllTasks. Hasil TList yang didapat kemudian dikirimkan ke *adapter* kustom untuk ditampilkan di jListView.
2. **Menambah Data:** Buat *event* OnClick untuk tombol 'Tambah'. Di dalamnya, ambil teks dari jEditText, panggil uDataAccess.AddTask, dan jika berhasil, panggil kembali prosedur untuk memuat ulang dan menampilkan semua tugas.
3. **Interaksi dengan List:**
   * **OnItemClick**: Saat sebuah item di jListView diklik, dapatkan id tugas dari data yang terikat pada posisi tersebut. Panggil uDataAccess.UpdateTaskStatus untuk mengubah status is\_completed-nya, lalu muat ulang daftar.
   * **OnItemLongClick**: Saat item ditahan-klik, tampilkan MessageDlg konfirmasi "Apakah Anda yakin ingin menghapus tugas ini?". Jika pengguna memilih "Ya", panggil uDataAccess.DeleteTask dengan id tugas tersebut, lalu muat ulang daftar.

**D. Memoles Tampilan dengan Adapter Kustom** Untuk menampilkan status selesai (centang dan coretan), kita akan menggunakan Adapter Kustom yang telah kita pelajari di Bab 8.

* **Desain Layout Baris:** Buat jForm baru (uTaskRowLayout.pas) sebagai *template* baris. Isinya sederhana: sebuah jCheckBox di kiri dan sebuah jTextView di kanan.
* **Logika di GetView Adapter:** Di dalam metode GetView pada *adapter* kustom Anda:
  + Isi jTextView dengan title tugas.
  + Atur status jCheckBox (.IsChecked) berdasarkan nilai is\_completed dari data.
  + **Trik Coretan:** Gunakan logika if untuk mengatur tampilan teks:

Delphi

if dataItem^.is\_completed = 1 then

myTextView.SetPaintFlags(myTextView.GetPaintFlags or jPaint.STRIKE\_THRU\_TEXT\_FLAG)

else

myTextView.SetPaintFlags(myTextView.GetPaintFlags and (not jPaint.STRIKE\_THRU\_TEXT\_FLAG));

**Tahap 3: Hasil Akhir**

Setelah semua langkah digabungkan, Anda kini memiliki sebuah aplikasi *Todo List* yang fungsional, andal, dan terasa profesional. Ia dapat menyimpan data bahkan setelah aplikasi ditutup dan dibuka kembali. Anda telah berhasil mengintegrasikan desain UI, manajemen data lokal yang robust, dan tampilan daftar yang dinamis menjadi satu produk yang utuh.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 11.2.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan ke proyek kedua.

Berikut adalah jawaban untuk prompt **Subbab 11.2: Proyek 2: Aplikasi "Katalog Produk Online"**.

**11.2. Proyek 2: Aplikasi "Katalog Produk Online"**

Setelah sukses dengan aplikasi *offline*, saatnya kita membawa keahlian kita ke dunia *online*. Di proyek kedua ini, kita akan membangun sebuah aplikasi **Katalog Produk Online**. Proyek ini akan menguji kemampuan kita untuk mengambil data dari internet (API), mem-parse format JSON yang kompleks, dan menampilkannya dalam sebuah daftar yang menarik secara visual, lengkap dengan gambar produk. Ini adalah sintesis dari Bab 5, 7, dan 8.

**Tahap 1: Perencanaan dan Desain**

**1. Definisi Fitur:**

* Saat aplikasi dibuka, ia akan secara otomatis mengambil daftar produk dari sebuah API publik.
* Setiap produk akan ditampilkan dalam daftar dengan gambar *thumbnail*, nama produk, dan harganya.
* Sebuah indikator *loading* akan muncul saat data sedang diambil.
* Saat pengguna mengetuk sebuah produk, aplikasi akan berpindah ke layar detail sederhana yang menampilkan nama produk tersebut.

**2. Sumber Data (API):** Kita akan menggunakan API publik gratis dari **dummyjson.com** yang tidak memerlukan kunci API. *Endpoint* yang akan kita gunakan adalah https://dummyjson.com/products. Sangat disarankan untuk membuka URL ini di browser Anda terlebih dahulu untuk melihat struktur JSON yang akan kita hadapi.

**3. Desain Antarmuka (UI):**

* **Layar Utama:** Sebuah jListView yang memenuhi layar dan sebuah jProgressBar di tengahnya (awalnya terlihat).
* **Layout Baris:** Kita akan membuat *layout* kustom untuk setiap baris di jListView, mirip dengan yang kita pelajari di Bab 8.1. *Layout* ini akan berisi jImageView (untuk *thumbnail* produk) dan dua jTextView (untuk nama dan harga produk).

**Tahap 2: Implementasi**

**A. Membuat Model Data dan Adapter Kustom**

1. **Buat Model Data:** Di unit utama Anda atau unit terpisah, definisikan sebuah record Pascal bernama TProduct untuk menampung data yang relevan:

Delphi

type

PProduct = ^TProduct;

TProduct = record

id: integer;

title: string;

price: double;

thumbnailUrl: string;

end;

1. **Buat Adapter Kustom:** Buat sebuah unit Adapter Kustom (uProductAdapter.pas) yang mewarisi dari jCustomAdapter dan menerima TList berisi pointer ke TProduct. Di dalam metode GetView-nya, Anda akan:
   * Mengambil data TProduct untuk posisi yang diminta.
   * Mengisi jTextView dengan product.title dan product.price.
   * Memanggil jImageView.SetImageFromURL(product.thumbnailUrl) untuk memuat gambar.

**B. Mengambil dan Mem-parse Data** Seluruh logika ini akan berada di Form utama Anda.

1. **Buat Prosedur FetchProducts:**

Delphi

procedure TAndroidModule1.FetchProducts;

begin

jProgressBar1.Visible := True;

jHttpClient1.SetURL('https://dummyjson.com/products');

jHttpClient1.Execute;

end;

1. **Panggil FetchProducts:** Panggil prosedur ini di dalam *event* OnCreate form agar data langsung dimuat saat aplikasi dibuka.
2. **Implementasikan OnHttpRequestCompleted:**

Delphi

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject; responseCode: integer; responseText: string);

var

productList: TList; // TList untuk menampung PProduct

// ... variabel JSON lainnya ...

begin

jProgressBar1.Visible := False;

if responseCode = 200 then

begin

productList := TList.Create;

try

// Parse JSON utama

jsonObj := TJSONObject(GetJSON(responseText));

// Ambil array dari kunci "products"

jsonArray := TJSONArray(jsonObj.Get('products'));

// Loop melalui array

for i := 0 to jsonArray.Count - 1 do

begin

jsonItem := TJSONObject(jsonArray.Items[i]);

// Buat record baru dan isi datanya

New(pProduct); // Alokasikan memori untuk pointer

pProduct^.id := jsonItem.Get('id').AsInteger;

pProduct^.title := jsonItem.Get('title').AsString;

pProduct^.price := jsonItem.Get('price').AsFloat;

pProduct^.thumbnailUrl := jsonItem.Get('thumbnail').AsString;

// Tambahkan pointer ke record ke dalam TList

productList.Add(pProduct);

end;

// Buat adapter dengan data yang sudah di-parse dan terapkan ke ListView

FProductAdapter := TProductAdapter.Create(Self, productList);

lvProducts.Adapter := FProductAdapter;

finally

jsonObj.Free;

// Catatan: TList sekarang bertanggung jawab atas memori pointer.

// Kita perlu membebaskannya saat TList di-destroy.

end;

end

else

ShowMessage('Gagal memuat data produk.');

end;

**C. Implementasi Navigasi ke Layar Detail**

1. Buat jForm baru bernama uDetailForm.pas dan daftarkan di file .lpr.
2. Di uDetailForm, tambahkan sebuah jTextView bernama tvDetailJudul.
3. Di Form utama, buat *event* OnItemClick untuk jListView.

Delphi

procedure TAndroidModule1.lvProductsItemClick(Sender: TObject; position: integer);

var

pProduct: PProduct;

intent: jIntent;

begin

// Dapatkan data produk yang diklik dari list

pProduct := PProduct(FProductAdapter.GetItem(position));

intent := jIntent.Create;

intent.SetAction(jIntent\_ACTION\_VIEW);

intent.PutExtra('product\_title', pProduct^.title);

jDetailForm.Show(intent);

end;

1. Di OnCreate pada uDetailForm, ambil data "product\_title" dari Intent dan tampilkan di tvDetailJudul.

**Tahap 3: Hasil Akhir**

Jalankan aplikasi Anda. Setelah beberapa saat, Anda akan melihat daftar produk yang kaya visual muncul di layar, diambil langsung dari server di internet. Saat Anda mengklik sebuah item, Anda akan dibawa ke layar detail. Anda telah berhasil membangun sebuah aplikasi online yang dinamis dan modern, menyatukan berbagai keahlian yang telah Anda pelajari!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 11.3.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan ke tahap akhir dari pembangunan proyek.

Berikut adalah jawaban untuk prompt **Subbab 11.3: Refactoring dan Pengujian Akhir**.

**11.3. Refactoring dan Pengujian Akhir**

Selamat! Anda telah berhasil membangun dua aplikasi yang lengkap dan fungsional dari awal hingga akhir. Namun, pekerjaan seorang developer profesional tidak berhenti saat aplikasi sudah 'berjalan'. Langkah selanjutnya, yang sering kali membedakan developer amatir dan profesional, adalah **merapikan kode (*refactoring*)** dan **mengujinya secara menyeluruh**.

Bab ini tidak akan berisi banyak kode baru, tetapi akan memberikan Anda pola pikir untuk meningkatkan kualitas, keterbacaan, dan keandalan kode yang telah Anda tulis.

**1. Refactoring: Merapikan 'Rumah' Kode Anda**

Secara sederhana, **refactoring** adalah proses menata ulang dan memperbaiki struktur internal dari kode yang sudah ada **tanpa mengubah perilakunya dari luar**. Tujuannya adalah membuat kode lebih bersih, lebih efisien, lebih mudah dipahami, dan lebih mudah dipelihara atau dikembangkan di masa depan.

**Contoh Praktis Refactoring:**

* **A. Pemisahan Tanggung Jawab (*Separation of Concerns*)**
  + **Masalah:** Pada proyek Todo List kita, semua logika UI (penanganan klik tombol) dan logika database (INSERT, SELECT, dll.) mungkin masih tercampur di dalam satu unit Form utama. Ini membuatnya sulit di-debug dan sulit untuk digunakan kembali di bagian lain.
  + **Solusi:** Pindahkan semua fungsi yang berhubungan dengan database (seperti AddTask, GetTasks, UpdateTaskStatus, DeleteTask) ke dalam satu unit Pascal baru yang terpisah, misalnya uDataAccess.pas. Dengan begitu, unit Form utama Anda hanya akan fokus pada urusan tampilan dan pemanggilan fungsi dari uDataAccess. Kode Anda menjadi lebih modular dan terorganisir.
* **B. Mengurangi Duplikasi Kode (Prinsip DRY - *Don't Repeat Yourself*)**
  + **Masalah:** Perhatikan di aplikasi Todo List, setiap kali kita melakukan aksi (menambah, mengubah, menghapus), kita selalu memanggil kode yang sama untuk memuat ulang dan menampilkan data ke jListView. Ini adalah duplikasi kode.
  + **Solusi:** Buat satu prosedur privat baru di form utama, misalnya procedure RefreshTaskList;. Letakkan semua logika untuk mengambil data dari database dan menerapkannya ke *adapter* di dalam prosedur ini. Sekarang, setiap kali Anda butuh memuat ulang daftar, Anda cukup memanggil RefreshTaskList; dalam satu baris saja. Ini lebih singkat, dan jika ada perubahan pada cara menampilkan data, Anda hanya perlu mengubahnya di satu tempat.

**2. Pengujian Akhir: Memastikan Kualitas**

Menguji aplikasi bukan sekadar memastikan aplikasi tidak *crash*. Ini adalah proses sistematis untuk menemukan 'lubang' dan 'kasus aneh' (*edge cases*) sebelum ditemukan oleh pengguna Anda.

**Checklist Pengujian yang Bisa Anda Lakukan:**

* **Pengujian Fungsional:**
  + Apakah semua tombol berfungsi sesuai harapan?
  + (Todo App) Apakah data benar-benar tersimpan setelah aplikasi ditutup dan dibuka lagi? Apakah status 'selesai' juga tersimpan?
  + (Katalog App) Apakah data dari API tampil dengan benar? Apakah gambar dimuat dengan baik?
* **Pengujian Antarmuka (UI/UX):**
  + Coba putar perangkat Anda ke mode lanskap. Apakah *layout*-nya masih terlihat bagus atau menjadi berantakan?
  + Coba jalankan di berbagai ukuran layar emulator (ponsel kecil, tablet). Apakah ada teks atau gambar yang terpotong?
  + Apakah alur navigasi dari daftar ke detail terasa intuitif dan lancar?
* **Pengujian Kasus Tepi (*Edge Cases*):**
  + (Todo App) Apa yang terjadi jika Anda mencoba menambah tugas dengan teks kosong?
  + (Todo App) Apa yang terjadi jika Anda menghapus semua tugas? Apakah aplikasi menampilkan daftar kosong dengan benar?
  + (Katalog App) Apa yang terjadi jika Anda membuka aplikasi tanpa koneksi internet? Apakah aplikasi menampilkan pesan *error* yang jelas atau malah *crash*?

**3. Sentuhan Akhir (Final Polish)**

Sebelum kita melangkah ke bab publikasi, lakukan pemeriksaan terakhir:

* **Komentar dan Penamaan:** Apakah ada bagian kode yang rumit? Tambahkan komentar untuk menjelaskannya kepada diri Anda di masa depan. Apakah semua variabel, fungsi, dan komponen memiliki nama yang konsisten dan deskriptif?
* **Pembersihan Kode:** Hapus semua variabel atau prosedur yang tidak lagi digunakan. Hapus juga semua kode Log.d yang tidak lagi relevan untuk versi rilis.

Dengan kode yang bersih, terorganisir, dan teruji, aplikasi Anda sekarang benar-benar siap untuk langkah selanjutnya: diperkenalkan kepada dunia.

Ini mengakhiri Bab 11. Anda tidak hanya bisa membangun aplikasi, tetapi juga tahu cara membuatnya berkualitas tinggi. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 12 (Pengantar Bab 12).

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan ke bab terakhir dari perjalanan inti kita.

Berikut adalah jawaban untuk prompt **Pengantar Bab 12**.

**Bab 12: Publikasi dan Pemeliharaan**

Selamat, Developer! Anda telah tiba di bab terakhir dari perjalanan inti kita. Anda telah merancang, mengodekan, menguji, dan merapikan aplikasi Anda hingga menjadi sebuah karya yang membanggakan. Anggaplah diri Anda seorang seniman yang baru saja menyelesaikan sebuah lukisan mahakarya. Lukisan itu sudah sempurna, tetapi saat ini ia masih terkurung di dalam studio Anda. Bab ini adalah tentang proses mempersiapkan mahakarya tersebut untuk dipamerkan di galeri terbesar dan paling ramai di dunia: **Google Play Store**.

Mempublikasikan aplikasi bukan sekadar menekan tombol 'upload'. Ini adalah sebuah proses profesional yang melibatkan persiapan aset, keamanan, dan kepatuhan terhadap aturan platform. Proses ini memastikan bahwa aplikasi yang sampai ke tangan pengguna adalah versi terbaik, teraman, dan paling optimal. Di bab ini, kita akan memandu Anda melalui setiap langkah penting dalam 'pameran' ini:

* Mempersiapkan 'bingkai dan brosur' (aset visual) untuk aplikasi Anda.
* Membuat 'sidik jari digital' yang aman (*Keystore*) untuk membuktikan keaslian karya Anda.
* Membangun paket rilis final (AAB) yang siap didistribusikan.
* Menavigasi 'galeri' Google Play Console untuk menyiapkan laman pameran Anda.
* Dan akhirnya, momen yang ditunggu-tunggu: merilis aplikasi Anda ke seluruh dunia.

Namun, seperti pameran yang sukses, pekerjaan tidak berhenti saat pintu galeri dibuka. Kita juga akan membahas secara singkat tentang apa yang terjadi setelah peluncuran: **pemeliharaan aplikasi**. Ini termasuk memantau performa, menanggapi ulasan pengguna, dan merilis pembaruan di masa depan. Ini adalah langkah terakhir yang akan mengubah Anda dari seorang pembuat aplikasi menjadi seorang penerbit aplikasi. Mari kita siapkan karya Anda untuk sorotan dunia.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.1.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 12.1: Mempersiapkan Aset Aplikasi**.

**12.1. Mempersiapkan Aset Aplikasi**

Sebelum kita menyentuh dasbor Google Play Console, langkah pertama adalah mempersiapkan 'materi promosi' untuk aplikasi kita. Aset-aset ini adalah hal pertama yang akan dilihat oleh calon pengguna di Play Store dan akan sangat memengaruhi keputusan mereka untuk mengunduh aplikasi Anda. Kesan pertama sangatlah penting, jadi mari kita siapkan dengan baik.

Berikut adalah daftar periksa aset-aset yang wajib Anda siapkan.

**1. Ikon Aplikasi (Resolusi Tinggi)**

Ini adalah wajah dari aplikasi Anda.

* **Spesifikasi Teknis:** Anda harus menyediakan ikon dengan resolusi **512 x 512 piksel**. Formatnya harus **32-bit PNG dengan lapisan transparansi (alpha)**.
* **Tips:** Desainlah ikon yang sederhana, mudah dikenali, dan tetap terlihat bagus saat ditampilkan dalam ukuran kecil di layar utama perangkat. Hindari teks berukuran kecil di dalam ikon. Untuk perangkat Android modern, pelajari juga tentang konsep **ikon adaptif (*adaptive icons*)** yang memungkinkan sistem menampilkan ikon Anda dalam berbagai bentuk (lingkaran, persegi, dll.) sesuai tema peluncur (*launcher*).

**2. Screenshot (Tangkapan Layar)**

Ini adalah etalase yang memamerkan aplikasi Anda.

* **Spesifikasi Teknis:** Anda wajib mengunggah **minimal 2 *screenshot*** (disarankan 4-8) untuk setiap jenis perangkat yang didukung (ponsel, tablet). Formatnya bisa **JPEG atau 24-bit PNG (tanpa alpha)**.
* **Tips:** Jangan hanya mengambil tangkapan layar biasa. Tampilkan fitur-fitur terbaik aplikasi Anda. Gunakan alat bantu editor gambar untuk menambahkan teks keterangan singkat di atas *screenshot* yang menjelaskan fungsi pada gambar tersebut. Susun *screenshot* Anda seperti sebuah cerita yang memandu pengguna melihat keunggulan aplikasi Anda.

**3. Grafik Fitur (Feature Graphic)**

Ini adalah 'spanduk' atau *banner* utama aplikasi Anda di Play Store.

* **Spesifikasi Teknis:** Sebuah gambar berukuran **1024 x 500 piksel** (JPEG atau 24-bit PNG).
* **Tujuan:** Grafik ini akan ditampilkan di bagian atas laman Play Store Anda, dan akan menjadi *thumbnail* jika Anda menyertakan video promosi. Buatlah gambar yang menarik secara visual, berani, dan mewakili *brand* aplikasi Anda.

**4. Judul & Deskripsi Aplikasi**

Ini adalah materi penjualan Anda dalam bentuk teks.

* **Judul:** Maksimal **30 karakter**. Buatlah nama yang unik, mudah diingat, dan jika memungkinkan, mengandung kata kunci utama yang relevan dengan fungsi aplikasi Anda.
* **Deskripsi Singkat:** Maksimal **80 karakter**. Ini adalah teks pertama yang dibaca pengguna di pratinjau daftar. Buatlah kalimat yang ringkas dan menarik yang merangkum fungsi utama aplikasi.
* **Deskripsi Lengkap:** Maksimal **4000 karakter**. Di sini Anda bisa menjelaskan semua fitur aplikasi secara detail. Gunakan poin-poin, *emoji*, dan format yang rapi agar mudah dibaca. Manfaatkan ruang ini untuk memasukkan berbagai **kata kunci (*keywords*)** yang mungkin dicari oleh pengguna (praktik ini disebut ASO - *App Store Optimization*).

**5. Kebijakan Privasi**

Ini adalah persyaratan hukum.

* Jika aplikasi Anda mengumpulkan data pengguna apa pun (termasuk data anonim dari *library* pihak ketiga seperti analitik) atau memerlukan izin sensitif, Anda **wajib** menyediakan URL yang mengarah ke halaman Kebijakan Privasi (*Privacy Policy*) Anda. Anda bisa menggunakan layanan gratis seperti 'Privacy Policy Generator' untuk membuatnya dan menempatkannya di layanan hosting gratis seperti GitHub Pages atau Blogger.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.2.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 12.2: Membangun Rilis: Android App Bundle (AAB)**.

**12.2. Membangun Rilis: Android App Bundle (AAB)**

Setelah semua aset visual dan teks siap, saatnya kita mempersiapkan 'paket' aplikasi yang akan diunggah ke Google Play. Di masa lalu, developer mengunggah file **APK (Android Package Kit)**. Namun, sejak beberapa tahun lalu, Google telah mewajibkan penggunaan format baru yang lebih canggih dan efisien: **Android App Bundle (AAB)**.

**Apa itu Android App Bundle (AAB)?**

Penting untuk dipahami: **AAB bukanlah file yang bisa Anda instal langsung di ponsel Anda.** AAB adalah **format publikasi**. Anggap saja AAB sebagai sebuah 'resep lengkap' dan 'semua bahan mentah' untuk aplikasi Anda, yang Anda serahkan kepada Google.

Saat Anda mengunggah file AAB ke Google Play, Play Store akan bertindak sebagai 'koki cerdas'. Ia akan menggunakan resep dan bahan tersebut untuk 'memasak' dan menyajikan APK yang paling optimal dan efisien untuk setiap pengguna yang mengunduh aplikasi Anda. Optimalisasi ini terjadi berdasarkan:

* **Arsitektur CPU:** Pengguna dengan ponsel ARM 64-bit hanya akan mengunduh kode yang relevan untuk arsitektur tersebut, bukan kode untuk arsitektur lain seperti x86.
* **Kepadatan Layar:** Pengguna dengan layar Full HD tidak perlu mengunduh aset gambar yang dirancang untuk layar 4K yang lebih besar.
* **Bahasa:** Pengguna hanya akan mengunduh sumber daya bahasa yang mereka gunakan di perangkatnya.

Hasilnya adalah **ukuran unduhan yang jauh lebih kecil** bagi pengguna, yang merupakan faktor penting untuk meningkatkan jumlah instalasi dan menghemat kuota data mereka.

**Cara Membangun File AAB di LAMW**

Untungnya, LAMW membuat proses pembuatan AAB ini sangat mudah. Kita hanya perlu mengubah beberapa pengaturan di opsi proyek.

1. **Buka Opsi Proyek:** Di menu bar Lazarus, klik Project > Options....
2. **Pilih Mode Release:** Di jendela Project Options yang muncul, lihat ke bagian atas. Ada menu *dropdown* untuk **Build Modes**. Selama ini kita bekerja dalam mode Debug, yang menyertakan informasi tambahan untuk *debugging*. Sekarang, pilih mode **Release**. Mode Release akan mengoptimalkan kode dan menghasilkan file yang lebih kecil, yang cocok untuk distribusi publik.
3. **Ubah Target Format:** Masih di jendela Project Options, navigasikan ke Application Settings > Target file format. Anda akan melihat sebuah menu *dropdown*.
4. Ubah pilihan dari Android Application (apk) menjadi **Android App Bundle (aab)**.
5. Klik OK untuk menyimpan pengaturan.

**Langkah Terakhir: Build!**

Sekarang, pastikan Anda masih berada dalam mode Release. Lakukan proses *build* dengan mengklik menu Run > Build (atau tekan Shift+F9). **Jangan klik Run (F9)**, karena kita hanya ingin membuat file rilisnya, bukan menjalankan sesi *debugging*.

Setelah proses kompilasi selesai, periksa folder *output* proyek Anda. Anda akan menemukan sebuah file baru dengan ekstensi .aab. Inilah file 'mentah' yang akan kita tandatangani (*sign*) di subbab berikutnya sebelum siap diunggah ke Google Play.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.3.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan. Ini adalah salah satu subbab paling kritis dalam keseluruhan proses.

Berikut adalah jawaban untuk prompt **Subbab 12.3: Membuat Kunci Penandatanganan (Signing Keystore) dan Mengamankannya**.

**12.3. Membuat Kunci Penandatanganan (Signing Keystore) dan Mengamankannya**

Kita sudah memiliki file .aab. Namun, sebelum mengunggahnya, Google Play mengharuskan paket tersebut ditandatangani secara digital. Tanda tangan ini adalah bukti kriptografis yang membuktikan dua hal: bahwa aplikasi tersebut benar-benar berasal dari Anda, dan bahwa aplikasi tersebut belum dimodifikasi sejak Anda membuatnya.

Tanda tangan digital ini disimpan dalam sebuah file yang disebut **Keystore**. Anggap saja *Keystore* ini sebagai **'stempel perusahaan' atau 'akta notaris digital'** Anda. Ia berisi kunci privat yang hanya Anda miliki. *Keystore* inilah yang akan menjadi identitas Anda sebagai developer untuk aplikasi ini selamanya.

**PERINGATAN SANGAT PENTING!**

File *Keystore* yang akan Anda buat adalah aset Anda yang paling berharga. **JANGAN PERNAH KEHILANGAN FILE INI BESERTA PASSWORD-NYA.** Buat beberapa cadangan (*backup*) dan simpan di tempat yang sangat aman (misalnya, di *cloud storage* pribadi, *flash drive* terpisah, dll.).

Jika Anda kehilangan *Keystore* ini, Anda **TIDAK AKAN PERNAH BISA** merilis *update* untuk aplikasi Anda lagi. Anda harus mempublikasikan aplikasi baru dengan nama paket yang berbeda dan kehilangan semua pengguna serta riwayat aplikasi lama Anda. **Tidak ada pengecualian.**

**Langkah 1: Membuat File Keystore dengan keytool**

Kita akan menggunakan sebuah alat baris perintah bernama keytool yang sudah disertakan bersama JDK yang telah kita instal di Bab 2.

1. **Buka Command Prompt:** Buka Command Prompt (atau PowerShell) di Windows.
2. **Navigasi ke Folder Proyek:** Gunakan perintah cd untuk pindah ke direktori tempat Anda ingin menyimpan file *keystore* (disarankan di dalam folder proyek aplikasi Anda agar mudah ditemukan). Contoh: cd D:\ProyekLazarus\KalkulatorSederhana.
3. **Jalankan Perintah keytool:** Salin dan tempelkan perintah berikut, lalu tekan Enter. Gantilah NAMA\_FILE\_KEYSTORE dan NAMA\_ALIAS\_KUNCI dengan nama yang Anda inginkan (tanpa spasi).

DOS

keytool -genkey -v -keystore NAMA\_FILE\_KEYSTORE.keystore -alias NAMA\_ALIAS\_KUNCI -keyalg RSA -keysize 2048 -validity 10000

1. **Jawab Pertanyaan:** keytool akan menanyakan serangkaian pertanyaan. Jawab dengan hati-hati:
   * **Enter keystore password:** Buat sebuah password yang kuat dan **catat baik-baik!** Anda akan mengetiknya dua kali.
   * **What is your first and last name?**: Nama Anda atau nama developer utama.
   * **What is the name of your organizational unit?**: Nama tim atau departemen (misal: 'Mobile Development').
   * **What is the name of your organization?**: Nama perusahaan atau organisasi Anda (misal: 'MyCompany').
   * **What is the name of your City or Locality?**: Nama kota Anda.
   * **What is the name of your State or Province?**: Nama provinsi Anda.
   * **What is the two-letter country code for this unit?**: Kode negara dua huruf (untuk Indonesia, gunakan ID).
   * **Is CN=... correct?**: Periksa datanya, lalu ketik yes dan tekan Enter.
   * **Enter key password for <NAMA\_ALIAS\_KUNCI>**: Anda akan diminta password lagi, khusus untuk *alias* ini. Untuk kesederhanaan, **tekan saja Enter** untuk menggunakan password yang sama dengan *keystore password*.

Setelah selesai, sebuah file baru (misalnya, kalkulator.keystore) akan dibuat di folder Anda.

**Langkah 2: Mengonfigurasi Penandatanganan di LAMW**

Sekarang kita perlu memberitahu Lazarus untuk menggunakan *keystore* ini setiap kali kita membuat build Release.

1. **Buka Opsi Proyek:** Kembali ke Lazarus, buka Project > Options....
2. **Masuk ke Tab Signing:** Pastikan Anda berada dalam *Build Mode* **Release**. Navigasikan ke Application Settings > Signing.
3. **Isi Detail Keystore:** Isi semua kolom dengan informasi yang baru saja Anda buat:
   * **Keystore file:** Klik ... dan arahkan ke file .keystore yang baru Anda buat.
   * **Keystore password:** Masukkan *password keystore* yang Anda buat tadi.
   * **Key alias:** Masukkan nama *alias* yang Anda definisikan (misal: kalkulator\_alias).
   * **Key password:** Masukkan *password* kunci Anda (jika Anda mengosongkannya tadi dengan menekan Enter, maka biarkan kolom ini kosong).
4. Klik OK untuk menyimpan pengaturan.

Selesai! Mulai sekarang, setiap kali Anda membuat *build* dalam mode Release (Run > Build), LAMW akan secara otomatis menggunakan *keystore* ini untuk menandatangani file .aab Anda. Paket aplikasi Anda kini aman, otentik, dan siap untuk langkah selanjutnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.4.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 12.4: Panduan Mengisi Formulir Keamanan Data di Google Play Console**.

**12.4. Panduan Mengisi Formulir Keamanan Data di Google Play Console**

Google sangat serius dalam hal privasi pengguna. Sebagai bagian dari transparansi, setiap aplikasi di Play Store wajib mengisi sebuah formulir bernama **Keamanan Data (Data Safety)**. Formulir ini memberitahu pengguna secara jelas dan sederhana, data apa saja yang Anda kumpulkan, mengapa Anda mengumpulkannya, dan dengan siapa Anda membagikannya.

Mengisi formulir ini dengan jujur dan akurat adalah sebuah **kewajiban**. Kesalahan dalam pengisian dapat menyebabkan aplikasi Anda ditolak atau bahkan dihapus dari Play Store.

**Cara Berpikir Sebelum Mengisi Formulir**

Sebelum membuka Play Console, duduk dan analisislah aplikasi Anda dengan menjawab pertanyaan-pertanyaan berikut:

* **Apakah aplikasi saya mengumpulkan data?** 'Mengumpulkan' berarti mengambil data dari perangkat. Ini termasuk data yang dimasukkan pengguna (nama, email), data otomatis (lokasi, ID perangkat), atau data lainnya.
* **Apakah aplikasi saya membagikan data?** 'Membagikan' berarti mengirim data pengguna ke pihak ketiga. Ini termasuk mengirim data ke server Anda sendiri, atau ke layanan pihak ketiga seperti Google Analytics, Firebase, atau layanan iklan.
* **Jenis data apa yang dikumpulkan/dibagikan?** Google menyediakan daftar yang sangat detail (Nama, Email, Lokasi, Info Keuangan, Aktivitas Aplikasi, dll.). Anda harus memetakannya satu per satu.
* **Untuk tujuan apa data digunakan?** (misalnya, untuk fungsionalitas aplikasi, analitik, personalisasi, iklan).
* **Apakah data dienkripsi saat transit?** Jika aplikasi Anda berkomunikasi dengan server melalui **HTTPS**, maka jawabannya adalah 'Ya'.
* **Apakah pengguna bisa meminta penghapusan data?**
* **Apakah saya punya Kebijakan Privasi (*Privacy Policy*)?** Untuk hampir semua kasus kecuali aplikasi yang paling sederhana, jawabannya harus 'Ya'.

**Studi Kasus Pengisian untuk Aplikasi Kita**

Mari kita gunakan dua aplikasi yang telah kita bangun di Bab 11 sebagai contoh.

**A. Aplikasi "Daftar Tugas" (Todo List - Offline)**

Aplikasi ini sangat sederhana dan semua datanya tersimpan secara lokal. Mari kita analisis:

* **Mengumpulkan data?** Ya, kita menyimpan teks 'tugas' yang dimasukkan oleh pengguna. Ini termasuk dalam kategori "Data yang Disediakan Pengguna".
* **Membagikan data?** Tidak. Semua data hanya tersimpan di database SQLite lokal di perangkat dan tidak pernah dikirim ke mana pun.

**Kesimpulan Pengisian:** Di formulir, Anda akan menyatakan bahwa aplikasi **'Mengumpulkan data'**. Saat ditanya jenis datanya, Anda bisa memilih kategori yang paling relevan (misalnya, "Konten Buatan Pengguna Lainnya"). Tujuannya adalah untuk **'Fungsionalitas Aplikasi'**. Anda akan menyatakan bahwa data **'Tidak dibagikan dengan pihak ketiga'**. Karena data tidak pernah dikirim, enkripsi saat transit tidak berlaku.

**B. Aplikasi "Katalog Produk Online"**

Aplikasi ini berkomunikasi dengan internet, tetapi hanya untuk *menerima* data.

* **Mengumpulkan data?** Tidak. Aplikasi ini hanya *menampilkan* data dari API. Ia tidak meminta atau menyimpan informasi apa pun *dari* pengguna.
* **Membagikan data?** Tidak. Aplikasi tidak mengirim data pengguna ke mana pun.

**Kesimpulan Pengisian:** Untuk aplikasi ini, Anda kemungkinan besar dapat menyatakan bahwa aplikasi **'Tidak mengumpulkan data apa pun'**. Ini adalah jawaban yang paling sederhana. Jika Anda menjawab ini, sisa dari formulir akan menjadi jauh lebih pendek.

*Catatan Penting:* Jika di masa depan Anda menambahkan fitur *pencarian* dan menyimpan riwayat pencarian pengguna di SharedPreferences, maka jawaban akan berubah menjadi 'Ya, mengumpulkan data (kategori: Aktivitas Aplikasi)'.

**Prinsip Utama: Jujur dan Transparan**

Prinsip terbaik saat mengisi formulir ini adalah: **Jika ragu, deklarasikan.** Lebih baik Anda mendeklarasikan pengumpulan data yang minimal daripada tidak mendeklarasikannya sama sekali dan dianggap melanggar kebijakan. Selalu baca kembali dokumentasi dari *library* pihak ketiga yang mungkin Anda gunakan, karena mereka bisa saja mengumpulkan data secara otomatis. Jujur dan transparan adalah kunci untuk membangun kepercayaan dengan pengguna dan Google Play.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.5.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 12.5: Proses Upload, Pengujian Internal, dan Peluncuran Bertahap**.

**12.5. Proses Upload, Pengujian Internal, dan Peluncuran Bertahap**

Semua persiapan telah selesai! File .aab yang sudah ditandatangani sudah ada, aset visual sudah siap, dan kita sudah memahami kebijakan data. Sekarang saatnya memasuki 'ruang pameran' yang sesungguhnya: **Google Play Console**. Di sinilah kita akan mengunggah, menguji, dan akhirnya merilis aplikasi kita.

*Catatan: Asumsinya, Anda sudah memiliki akun Google Play Developer, yang memerlukan biaya pendaftaran satu kali.*

**Langkah-langkah di Google Play Console**

1. **Buat Aplikasi Baru**
   * Login ke <https://play.google.com/console>.
   * Klik tombol Create app yang biasanya ada di pojok kanan atas.
   * Isi detail awal seperti Nama Aplikasi, bahasa default, jenis aplikasi (Aplikasi atau Game), dan apakah gratis atau berbayar. Setujui semua persyaratan dan klik Create app.
2. **Isi Detail Aplikasi (*Store Listing*)**
   * Anda akan diarahkan ke dasbor aplikasi. Di menu sebelah kiri, navigasikan ke Grow > Store presence > Main store listing.
   * Di sinilah Anda mengisi semua informasi yang telah kita siapkan di Subbab 12.1: **Judul, Deskripsi Singkat, Deskripsi Lengkap**.
   * Gulir ke bawah ke bagian *Graphics* dan unggah semua aset visual Anda: **Ikon Aplikasi, Grafik Fitur, dan Screenshot**.
   * Klik Save setelah selesai.
3. **Isi Kuesioner Rating Konten**
   * Setiap aplikasi harus memiliki rating konten. Di menu kiri, navigasikan ke Policy and programs > App content.
   * Anda akan melihat beberapa bagian yang perlu diisi. Mulailah dengan **Rating Konten (Content Rating)**.
   * Ikuti dan jawab kuesioner tersebut dengan jujur. Ini adalah proses yang dipandu dan akan menghasilkan rating seperti IARC 3+, 7+, dll., yang akan ditampilkan di laman aplikasi Anda.
4. **Unggah File AAB Anda ke Jalur Pengujian** Praktik terbaik seorang developer profesional adalah **jangan pernah langsung merilis ke Produksi**. Kita harus mengujinya terlebih dahulu di lingkungan Play Store yang sebenarnya.
   * Di menu kiri, navigasikan ke Release > Testing > Internal testing.
   * Klik Create new release.
   * Di halaman berikutnya, Anda akan melihat area untuk mengunggah file. **Seret (*drag*) file .aab Anda yang sudah ditandatangani** ke area ini. Google akan memprosesnya sejenak.
   * Tulis catatan rilis singkat (misalnya, 'Rilis internal pertama untuk pengujian'), lalu klik Save dan kemudian Review release.
   * Di halaman terakhir, Anda bisa menambahkan daftar email penguji internal Anda (misalnya, email Anda sendiri dan beberapa teman). Setelah itu, klik Start rollout to internal testing.

**Pahami Jalur Rilis (*Testing Tracks*)**

Google Play menyediakan beberapa 'jalur' untuk merilis aplikasi Anda, dari yang paling privat hingga publik:

* **Internal Testing:** Seperti yang baru kita lakukan. Untuk maksimal 100 penguji internal. *Update* akan tersedia bagi mereka dalam beberapa menit. Sangat cocok untuk *quality check* terakhir oleh tim Anda sendiri.
* **Closed Testing (Alpha):** Untuk grup penguji yang lebih besar yang Anda undang melalui email atau Google Groups. Cocok untuk mendapatkan masukan dari komunitas yang lebih luas tetapi masih terkontrol.
* **Open Testing (Beta):** Siapa pun bisa menemukan laman aplikasi Anda di Play Store dan memilih untuk bergabung dalam program beta. Berguna untuk pengujian skala besar sebelum rilis penuh.
* **Production:** Jalur rilis final yang akan tersedia untuk diunduh oleh semua pengguna di seluruh dunia.

**Rilis ke Produksi (dan Peluncuran Bertahap)**

Setelah Anda puas dengan hasil pengujian di jalur internal atau beta, Anda bisa 'mempromosikan' rilis tersebut ke produksi.

1. Navigasikan ke Release > Production.
2. Buat rilis baru. Anda bisa menggunakan kembali *bundle* yang sama dari jalur pengujian.
3. Saat merilis, Anda akan diberi opsi **Peluncuran Bertahap (*Staged Rollout*)**. Daripada merilis ke 100% pengguna secara serentak, Anda bisa memilih untuk merilis ke 1%, 5%, atau 10% pengguna terlebih dahulu. Ini adalah jaring pengaman yang sangat kuat. Anda bisa memantau laporan *crash* selama beberapa hari, dan jika semuanya aman, baru tingkatkan persentasenya hingga 100%.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 12.6.

Yahok

Tampilkan alur berpikir

Baik, kita selesaikan bab terakhir ini.

Berikut adalah jawaban untuk prompt **Subbab 12.6: Strategi Pemeliharaan dan Merilis Update Aplikasi**.

**12.6. Strategi Pemeliharaan dan Merilis Update Aplikasi**

Selamat! Aplikasi Anda kini telah mendunia. Anda telah berhasil melewati seluruh proses dari ide hingga publikasi. Namun, perjalanan seorang developer sejati tidak berhenti di sini. Merilis aplikasi adalah awal dari sebuah fase baru yang tidak kalah penting: **pemeliharaan dan pengembangan jangka panjang**.

**Pekerjaan Baru Saja Dimulai**

Anggap peluncuran aplikasi Anda seperti pembukaan sebuah restoran. Restoran itu sekarang terbuka untuk umum. Tugas Anda selanjutnya adalah memastikan 'dapur' tetap bersih (kode bebas bug), 'pelayanan' tetap cepat (aplikasi berperforma baik), dan Anda 'mendengarkan' masukan dari para 'pelanggan' (pengguna Anda) untuk terus berkembang.

**Memantau Kesehatan Aplikasi**

Google Play Console adalah dasbor kesehatan aplikasi Anda. Biasakan diri Anda untuk secara rutin memeriksa beberapa metrik kunci di menu sebelah kiri:

* **Vitals (di bawah menu *Quality*):** Ini adalah bagian terpenting. Pantau tingkat **ANR (*Application Not Responding*)** dan **Crashes**. Jika persentasenya tinggi, berarti ada masalah serius di aplikasi Anda yang harus segera ditemukan dan diperbaiki.
* **Ratings and reviews:** Baca ulasan dari pengguna. Mereka adalah sumber umpan balik gratis yang sangat berharga untuk mengetahui apa yang mereka sukai dan apa yang menjadi masalah atau *bug* yang mungkin Anda lewatkan.

**Menanggapi Ulasan Pengguna**

Berinteraksi dengan pengguna melalui ulasan akan membangun citra positif dan loyalitas.

* **Untuk Ulasan Positif:** Ucapkan terima kasih. Ini menunjukkan Anda menghargai dukungan mereka dan membuat mereka merasa didengar.
* **Untuk Laporan Bug atau Ulasan Negatif:** Jangan defensif. Ucapkan terima kasih atas masukannya. Jika itu adalah *bug*, akui dan beritahu mereka bahwa Anda akan memeriksanya untuk perbaikan di *update* selanjutnya. Jika itu adalah permintaan fitur, berterima kasihlah atas idenya. Respons yang profesional bisa mengubah pengguna yang kecewa menjadi pendukung setia.

**Proses Merilis Pembaruan (Update)**

Cepat atau lambat, Anda pasti akan merilis versi baru, baik untuk memperbaiki *bug* maupun menambah fitur. Prosesnya mirip dengan rilis pertama, tetapi ada satu langkah teknis yang **sangat krusial**.

1. **Perbaiki atau Tambah Fitur:** Lakukan perubahan yang diperlukan pada kode sumber Anda di Lazarus.
2. **(KRUSIAL) Naikkan versionCode:** Buka Project > Options.... Di bagian Application Settings, cari *field* **Version Code**. **Naikkan nilainya menjadi satu angka lebih tinggi** (misalnya, dari 1 menjadi 2, dari 2 menjadi 3, dan seterusnya). Google Play menggunakan versionCode ini untuk mengetahui bahwa ini adalah versi baru. **Jika Anda lupa menaikkannya, Google Play akan menolak unggahan Anda.**
3. **(Opsional) Ubah versionName:** Anda juga bisa mengubah **Version Name** menjadi sesuatu yang dilihat pengguna, misalnya dari '1.0' menjadi '1.0.1' (untuk perbaikan *bug* kecil) atau '1.1' (untuk penambahan fitur minor).
4. **Bangun AAB Baru:** Pastikan Anda masih dalam mode Release dan menggunakan **keystore yang sama persis** seperti rilis pertama. Lakukan *build* (Run > Build) untuk menghasilkan file .aab yang baru.
5. **Unggah dan Rilis:** Di Play Console, buat rilis baru di jalur yang Anda inginkan (misalnya, mulai dari *Internal Testing* lagi, atau langsung ke *Production*). Unggah file .aab yang baru, tulis catatan rilis "Apa yang baru" untuk pengguna, dan luncurkan.

Dengan memahami siklus pemeliharaan dan pembaruan ini, Anda telah siap untuk mengelola aplikasi Anda dalam jangka panjang. Siklus 'kembangkan -> rilis -> dengarkan -> perbaiki -> rilis lagi' adalah detak jantung dari seorang developer aplikasi yang sukses.

Ini mengakhiri Bab 12. Selamat! Anda telah menyelesaikan seluruh panduan inti dari ide hingga rilis dan pemeliharaan. Saya akan menunggu perintah "yahok" dari Anda untuk melanjutkan ke bagian **Penutup** buku.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan ke bagian terakhir dari narasi buku ini.

Berikut adalah jawaban untuk prompt **Penutup Buku**.

**Penutup**

Selamat, Developer! Anda telah berhasil mencapai akhir dari buku ini. Jika Anda telah mengikuti setiap bab, dari persiapan yang rumit di awal hingga merilis aplikasi ke Play Store, maka berikanlah tepuk tangan yang meriah untuk diri Anda sendiri. Anda telah menyelesaikan sebuah perjalanan yang tidak mudah, yang menuntut ketelitian, kesabaran, dan kemauan untuk belajar. Anda pantas merasa sangat bangga atas pencapaian ini.

Mari kita lihat sejenak ke belakang, betapa jauhnya Anda telah melangkah. Anda memulai dengan sebuah kanvas kosong. Kini, 'kotak perkakas' Anda telah terisi penuh. Anda tidak hanya tahu cara membuat tombol dan menampilkan teks, tetapi Anda juga telah menguasai cara merangkai alur aplikasi multi-layar, mengelola data secara lokal dengan database SQLite, berkomunikasi dengan API di seluruh dunia, memanfaatkan perangkat keras seperti kamera dan GPS, menangani tugas di latar belakang agar aplikasi tetap responsif, hingga mempelajari 'ilmu diplomasi' tingkat tinggi untuk menjembatani kode Pascal dengan ekosistem Java/Kotlin yang luas. Anda telah belajar cara membangun, bukan hanya sekadar mengode.

Dunia teknologi, terutama pengembangan aplikasi mobile, adalah sebuah sungai yang arusnya tidak pernah berhenti. Selalu ada versi Android baru, teknik baru, dan tantangan baru yang muncul. Buku ini telah memberikan Anda sebuah perahu yang kokoh dan kemampuan untuk bernavigasi, tetapi petualangan sesungguhnya baru saja dimulai. Teruslah merasa ingin tahu. Bergabunglah dengan forum-forum komunitas Lazarus, baca dokumentasi, coba integrasikan API baru yang menarik, dan jangan pernah takut untuk bereksperimen dan membuat kesalahan. Kesalahan adalah guru terbaik dalam perjalanan seorang developer.

Ide-ide cemerlang yang ada di kepala Anda kini tidak lagi hanya sekadar angan-angan. Anda memiliki pengetahuan dan kekuatan untuk mengubahnya menjadi aplikasi yang nyata, yang mungkin suatu saat akan memecahkan masalah, menghibur, atau membantu banyak orang.

Terima kasih telah mempercayakan saya untuk menjadi pemandu Anda dalam perjalanan ini. Dunia menantikan karya-karya hebat dari Anda. Sekarang, pergilah dan mulailah membangun.

Selamat berkarya!

Ini mengakhiri bagian Penutup. Saya akan menunggu perintah "yahok" dari Anda untuk melanjutkan ke bagian **Lampiran (Appendix)**, dimulai dengan Lampiran A.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan ke bagian Lampiran.

Berikut adalah jawaban untuk prompt **Lampiran A: Troubleshooting (Penyelesaian Masalah)**.

**Lampiran A: Troubleshooting (Penyelesaian Masalah)**

Setiap developer, tidak peduli seberapa berpengalaman, pasti akan bertemu dengan *bug* dan *error*. Bagian ini berfungsi sebagai panduan pertolongan pertama untuk masalah-masalah yang paling umum ditemui saat mengembangkan aplikasi dengan LAMW. Jangan panik, periksa daftarnya, dan temukan solusinya di sini.

**Error Kompilasi: Identifier not found "jTextView" (atau komponen lainnya)**

* **Masalah:** Saat mengompilasi, Lazarus menampilkan pesan *error* "Identifier not found" untuk sebuah komponen LAMW.
* **Penyebab Umum:** Unit yang berisi definisi komponen tersebut belum ditambahkan ke klausa uses di bagian implementation.
* **Solusi:**
  + Pastikan di bagian implementation dari unit Anda, klausa uses sudah mencakup unit yang diperlukan. Untuk sebagian besar komponen UI, unit utamanya adalah Android.Widget.
  + Contoh: uses ..., Android.Widget, Android.Base;

**Error Kompilasi: JDK/SDK/NDK path not found**

* **Masalah:** Proses *build* gagal total dengan pesan yang mengindikasikan salah satu *path* tidak ditemukan.
* **Penyebab Umum:** Konfigurasi *path* di dalam Lazarus IDE salah atau belum diatur.
* **Solusi:**
  + Buka kembali Tools > Options... > LAMW > Paths.
  + Periksa kembali setiap *path* (JDK, Android SDK, Android NDK) dan pastikan sudah menunjuk ke direktori yang benar sesuai yang telah kita siapkan di Bab 2. Pastikan tidak ada salah ketik atau spasi yang tidak perlu.

**Error Runtime: Aplikasi Crash dengan Access Violation atau Null Pointer Exception**

* **Masalah:** Aplikasi berjalan lalu tiba-tiba tertutup secara paksa. Logcat menunjukkan Null Pointer Exception.
* **Penyebab Umum:** Anda mencoba menggunakan sebuah objek atau komponen yang belum dibuat (*created*) atau nilainya nil.
* **Solusi:**
  + Sebelum menggunakan objek yang Anda buat secara manual, selalu periksa dengan if Assigned(MyObject) then ....
  + Untuk komponen visual, pastikan ID yang Anda panggil di kode sudah benar dan sama dengan properti ID yang Anda atur di *Object Inspector*.
  + Pastikan Anda sudah membuat *instance* dari objek tersebut, misalnya MyList := TList.Create; sebelum mencoba MyList.Add(...).

**Error Runtime: Application Not Responding (ANR)**

* **Masalah:** Aplikasi macet (*freeze*) dan tidak merespons sentuhan, lalu sistem menampilkan dialog ANR.
* **Penyebab Umum:** Anda melakukan operasi yang berjalan lama (lebih dari beberapa detik) di **UI Thread**. Contoh: operasi jaringan, pengolahan file besar, atau *looping* yang sangat berat.
* **Solusi:**
  + Pindahkan semua operasi yang berjalan lama ke *thread* latar belakang menggunakan TThread, seperti yang dibahas di Bab 9.
  + Gunakan TThread.Queue atau TThread.Synchronize untuk mengirim hasil kembali ke UI Thread dengan aman.

**Error Runtime: Aplikasi Crash karena Permission Denial**

* **Masalah:** Aplikasi *crash* saat mencoba mengakses kamera, lokasi, atau penyimpanan, dan Logcat menunjukkan SecurityException: Permission Denial.
* **Penyebab Umum:** Anda belum meminta izin *runtime* kepada pengguna atau lupa mendeklarasikannya di AndroidManifest.xml.
* **Solusi:**
  + Pastikan izin yang diperlukan (misal: <uses-permission android:name="android.permission.CAMERA" />) sudah ada di AndroidManifest.xml.
  + Implementasikan alur kerja tiga langkah untuk **Izin Runtime** (Periksa, Minta, Tanggapi) seperti yang dibahas di Bab 6 sebelum memanggil fungsi yang memerlukan izin tersebut.

**Masalah Lain: Aplikasi Berhasil di-Build, tapi Tidak Muncul di Perangkat**

* **Masalah:** Lazarus melaporkan "Build Successful", tetapi aplikasi tidak terinstal atau berjalan di perangkat.
* **Penyebab Umum:** Masalah koneksi ADB (*Android Debug Bridge*).
* **Solusi:**
  + Cabut dan pasang kembali kabel USB.
  + Di perangkat Anda, matikan lalu nyalakan kembali opsi Debugging USB di Opsi developer.
  + Pastikan Anda sudah memberikan otorisasi (Allow USB debugging?) di layar ponsel Anda saat pertama kali menghubungkan.
  + Restart ADB dengan mengetik adb kill-server lalu adb start-server di Command Prompt komputer Anda.

**Masalah Lain: Tampilan Layout Berantakan di Perangkat Berbeda**

* **Masalah:** Tampilan aplikasi Anda terlihat bagus di emulator, tetapi terpotong atau berantakan di perangkat fisik dengan ukuran layar berbeda.
* **Penyebab Umum:** Penggunaan satuan ukuran yang salah atau *layout* yang kaku.
* **Solusi:**
  + **Jangan pernah menggunakan px!** Selalu gunakan dp untuk ukuran komponen/margin dan sp untuk ukuran teks.
  + Manfaatkan properti match\_parent, wrap\_content, dan terutama layout\_weight di dalam jLinearLayout untuk membuat *layout* yang fleksibel dan responsif, seperti yang dibahas di Bab 4.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke Lampiran B.

Yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Lampiran B: Referensi Cepat Komponen LAMW**.

**Lampiran B: Referensi Cepat Komponen LAMW**

Lampiran ini berfungsi sebagai 'contekan' atau referensi cepat untuk komponen-komponen UI LAMW yang paling sering Anda gunakan. Gunakan tabel ini untuk mengingat kembali fungsi sebuah komponen atau properti kunci yang perlu diatur.

|  |  |  |
| --- | --- | --- |
| **Komponen** | **Deskripsi Singkat** | **Properti / Event Kunci** |
| **jLinearLayout** | Kontainer untuk menyusun komponen lain secara vertikal atau horizontal. | Orientation, Gravity, layout\_width, layout\_height, layout\_weight |
| **jRelativeLayout** | Kontainer yang menyusun komponen berdasarkan posisi relatif satu sama lain. | layout\_above, layout\_below, layout\_toLeftOf, layout\_alignParentBottom |
| **jScrollView** | Membuat konten di dalamnya (biasanya satu jLinearLayout) dapat digulir. | (Menampung satu komponen anak) |
| **jButton** | Tombol standar untuk memicu aksi. | Text, ID, Enabled, OnClick |
| **jTextView** | Menampilkan teks kepada pengguna. | Text, TextSize, TextColor, Gravity |
| **jEditText** | Kolom input bagi pengguna untuk mengetik teks. | Text, Hint, InputType, OnTextChanged, OnKeyUp |
| **jImageView** | Menampilkan gambar (*bitmap*) dari resource, file, atau URL. | Image, ScaleType, SetImageFromURL |
| **jListView** | Menampilkan daftar item yang dapat digulir secara vertikal. | Items, Adapter, OnItemClick, OnItemLongClick |
| **jCheckBox** | Kotak centang untuk pilihan ya/tidak (bisa lebih dari satu yang dipilih). | Text, IsChecked, OnCheckedChange |
| **jSwitch** | Tombol geser modern untuk status on/off. | Text, IsChecked, OnCheckedChange |
| **jRadioButton** | Tombol pilihan eksklusif (hanya satu yang bisa dipilih dalam satu jRadioGroup). | Text, IsChecked, OnCheckedChange |
| **jSpinner** | Menu *dropdown* untuk memilih satu item dari sebuah daftar. | Items, SelectedIndex, OnItemSelected |
| **jProgressBar** | Indikator visual untuk proses yang sedang berjalan (berputar atau bar). | Progress, Max, Visible, Style |
| **jWebView** | Menampilkan konten web (halaman internet atau file HTML lokal). | LoadUrl, LoadData, GetSettings.SetJavaScriptEnabled |
| **jMediaPlayer** | Komponen non-visual untuk memutar file audio atau video berdurasi panjang. | SetDataSource, Prepare, Start, Pause, Stop, SeekTo |
| **jLocationSensor** | Komponen non-visual untuk mendapatkan data lokasi dari GPS. | MinTimeUpdate, MinDistanceUpdate, Start, Stop, OnLocationChanged |

Ekspor ke Spreadsheet

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke Lampiran C.

Yahok

Tampilkan alur berpikir

Baik, ini adalah bagian terakhir dari lampiran.

Berikut adalah jawaban untuk prompt **Lampiran C: Sumber Daya Tambahan**.

**Lampiran C: Sumber Daya Tambahan**

Perjalanan belajar seorang developer tidak pernah berhenti. Buku ini adalah fondasi yang kuat, tetapi dunia pengembangan terus bergerak dan komunitas adalah sumber kekuatan yang luar biasa. Lampiran ini berisi kumpulan tautan dan sumber daya yang sangat direkomendasikan untuk melanjutkan eksplorasi Anda, mendapatkan bantuan saat mengalami kesulitan, dan mencari inspirasi untuk proyek selanjutnya.

**1. Situs Web Resmi dan Repositori**

Ini adalah sumber utama untuk dokumentasi, unduhan, dan berita terbaru.

* **Lazarus IDE:** <https://www.lazarus-ide.org/>

Tempat untuk mengunduh versi terbaru Lazarus dan membaca berita pengembangan.

* **Free Pascal Compiler (FPC):** <https://www.freepascal.org/>

Situs resmi untuk kompilator di balik Lazarus, lengkap dengan dokumentasi bahasa Object Pascal.

* **Lazarus & FPC Wiki:** <https://wiki.freepascal.org/>

Harta karun berupa tutorial, dokumentasi, dan contoh kode untuk berbagai macam topik. Ini adalah tempat pertama yang harus Anda kunjungi saat mencari informasi teknis.

* **LAMW on GitHub:** <https://github.com/jmpessoa/lazandroidmodulewizard>

Repositori kode sumber resmi untuk LAMW. Tempat untuk melihat perkembangan terbaru, melaporkan isu, dan bahkan berkontribusi jika Anda sudah mahir.

**2. Forum dan Komunitas**

Tempat terbaik untuk bertanya jika Anda mengalami kesulitan atau ingin berdiskusi dengan sesama developer.

* **Forum Resmi Lazarus-IDE:** <https://forum.lazarus.freepascal.org/>

Komunitas yang sangat aktif dan membantu. Pastikan untuk mencari di sub-forum 'Android Development' untuk pertanyaan spesifik terkait LAMW. Jangan ragu bertanya, tetapi pastikan Anda sudah mencari solusinya terlebih dahulu.

**3. API Publik Gratis untuk Latihan**

Cara terbaik untuk mengasah keahlian *networking* adalah dengan berlatih. Berikut beberapa API yang mudah digunakan.

* **JSONPlaceholder:** <https://jsonplaceholder.typicode.com/>

Menyediakan data JSON *dummy* (posts, comments, users) yang sempurna untuk pengujian *parsing* dan tampilan daftar tanpa perlu kunci API.

* **DummyJSON:** <https://dummyjson.com/>

Mirip dengan JSONPlaceholder tetapi dengan data yang lebih kaya dan realistis, termasuk gambar produk. Sangat bagus untuk mempraktikkan proyek seperti di Bab 11.

* **Public APIs Collection:** <https://github.com/public-apis/public-apis>

Sebuah daftar raksasa yang dikurasi oleh komunitas berisi ratusan API publik gratis untuk berbagai kategori, dari game, buku, hingga data pemerintahan.

**4. Sumber Daya Desain dan Aset**

Untuk membuat aplikasi Anda terlihat lebih profesional.

* **Google Fonts:** <https://fonts.google.com/>

Koleksi ratusan *font* gratis dan berkualitas tinggi yang bisa Anda sertakan di dalam aplikasi Anda.

* **Material Design Icons:** <https://fonts.google.com/icons>

Koleksi ikon resmi dari Google yang konsisten dengan gaya desain Android. Anda bisa mengunduhnya sebagai file PNG atau SVG.

* **Coolors:** <https://coolors.co/>

Sebuah generator palet warna yang sangat populer untuk membantu Anda menemukan kombinasi warna yang serasi untuk desain aplikasi Anda dengan cepat.

Ini adalah jawaban terakhir dari semua prompt yang telah kita susun. Seluruh outline buku, dari Bab 1 hingga Lampiran, kini telah lengkap. Semoga ini menjadi fondasi yang sangat kuat untuk buku Anda!

Buat pengantar penulis

Tampilkan alur berpikir

Tentu, dengan senang hati. Sebuah pengantar dari penulis adalah sentuhan personal yang dapat membangun hubungan dengan pembaca sejak halaman pertama.

Berikut adalah draf untuk "Pengantar Penulis", ditulis dengan gaya yang santai namun tetap formal, seolah-olah berasal dari penulis buku yang telah kita rancang bersama.

**Pengantar Penulis**

Salam hangat, para developer dan calon developer!

Jika Anda memegang buku ini, kemungkinan besar ada sesuatu yang menghubungkan kita: kecintaan atau setidaknya rasa penasaran terhadap kekuatan dan keanggunan bahasa Pascal serta kecepatan pengembangan visual yang ditawarkan oleh Lazarus IDE. Saya, seperti mungkin sebagian dari Anda, memulai perjalanan pemrograman saya di dunia Delphi dan Pascal, sebuah dunia di mana logika yang terstruktur bertemu dengan kemudahan *drag-and-drop* untuk menciptakan aplikasi desktop yang andal.

Namun, kita semua tahu ceritanya. Dalam satu dekade terakhir, dunia teknologi bergeser dengan cepat ke arah *mobile*. Tembok besar bernama Java, Kotlin, dan XML seolah berdiri kokoh, terkadang membuat para developer Pascal merasa seperti harus memulai lagi dari nol, meninggalkan 'rumah' yang nyaman untuk mempelajari ekosistem yang sama sekali baru dan berbeda. Perasaan inilah yang menjadi percikan api lahirnya buku ini.

Di tengah pencarian itulah saya menemukan LAMW (Lazarus Android Module Wizard). Awalnya saya skeptis, namun keraguan itu dengan cepat berubah menjadi kekaguman. Ini bukan sekadar *tool*; ini adalah sebuah jembatan yang tak terduga, sebuah portal yang memungkinkan kita membawa semua kekuatan, kecepatan, dan kejelasan logika Object Pascal langsung ke jantung pengembangan Android *native*. Momen ketika saya berhasil mengompilasi proyek LAMW pertama dan melihatnya berjalan mulus di perangkat Android adalah sebuah "momen aha!" yang ingin saya bagikan kepada Anda semua.

Buku ini lahir dari keinginan sederhana: untuk membagikan 'peta' perjalanan tersebut. Selama bertahun-tahun, saya mengumpulkan potongan-potongan informasi dari forum, wiki, dan eksperimen pribadi. Saya ingin menciptakan sumber daya terlengkap dalam bahasa Indonesia, sebuah panduan terstruktur yang membawa Anda dari nol—dari kebingungan saat melakukan setup—hingga menjadi seorang developer yang percaya diri untuk mempublikasikan aplikasi kompleks ke Google Play Store.

Anda akan melihat buku ini tidak hanya berisi resep kode. Saya percaya pemahaman yang mendalam datang dari fondasi yang kuat. Kita akan mulai dari memahami 'mengapa' di balik setiap konsep, sebelum menyelam ke 'bagaimana' cara implementasinya. Kita akan membahas praktik-praktik modern yang relevan hingga tahun 2025 ini, mulai dari menangani aturan penyimpanan Android terbaru, mengintegrasikan layanan Google Maps, hingga menyentuh dunia kecerdasan buatan dengan Gemini API.

Harapan terbesar saya adalah, setelah menyelesaikan bab terakhir, Anda tidak hanya memiliki serangkaian keahlian teknis, tetapi yang lebih penting, Anda memiliki **kepercayaan diri** untuk berkata, "Saya bisa membangun ini." Kepercayaan diri untuk melihat sebuah masalah, membukakan Lazarus, dan mulai merancang solusinya.

Tentu saja, karya ini tidak akan ada tanpa kerja keras para kontributor di komunitas *open source* global. Terima kasih yang tak terhingga kepada tim pengembang Free Pascal, Lazarus, dan terutama kepada José Marques Pessoa serta para kontributor lain yang telah mendedikasikan waktu dan keahlian mereka untuk menciptakan dan memelihara LAMW.

Kepada Anda, para pembaca, terima kasih atas kepercayaan Anda telah memilih buku ini sebagai pemandu. Mari kita mulai perjalanan yang luar biasa ini bersama-sama.

Selamat belajar, selamat berkarya, dan semoga sukses!

Hormat saya,

[Nama Penulis]

Juni 2025

mulai menjawab prompt bab 13 secara bertahap

Tampilkan alur berpikir

Tentu, dengan senang hati. Kita akan mulai proses untuk Bab 13.

Saya akan menjawab setiap prompt secara berurutan, satu per satu, dan menunggu instruksi "yahok" dari Anda sebelum melanjutkan.

Mari kita mulai dari awal bab.

**Jawaban untuk Prompt: Pengantar Bab 13**

**Judul Bab 13: Integrasi Peta Tingkat Lanjut dengan Google Maps**

Selamat datang di bab tambahan yang sangat menarik ini. Sejauh ini, aplikasi kita berinteraksi dengan data dan layanan, namun sebagian besar masih berada di dunia abstrak. Di bab ini, kita akan menghubungkan aplikasi kita dengan dunia fisik melalui salah satu fitur paling visual dan interaktif dalam pengembangan aplikasi modern: **Peta**. Bayangkan aplikasi pencari properti yang langsung menunjukkan lokasi rumah di peta, aplikasi travel yang memetakan rute perjalanan, atau aplikasi kuliner yang menampilkan semua restoran terdekat. Integrasi peta mengubah aplikasi dari sekadar penyaji informasi menjadi sebuah pemandu lokasi yang cerdas.

Sebelum kita melangkah lebih jauh, ada satu hal yang sangat penting untuk dipahami: **bab ini adalah materi tingkat mahir.** Kita akan mengintegrasikan **Google Maps Android API**, yang merupakan sebuah *library* Java/Kotlin. Oleh karena itu, keberhasilan Anda dalam mengikuti bab ini sangat bergantung pada pemahaman yang solid tentang konsep **interoperabilitas JNI** yang telah kita bahas tuntas di **Bab 10**. Anggap bab ini sebagai aplikasi nyata dan studi kasus lanjutan dari semua teori JNI tersebut.

Tujuan kita adalah membuktikan bahwa dengan LAMW dan JNI, tidak ada batasan untuk mengintegrasikan *library* pihak ketiga yang canggih sekalipun. Kita akan memandu Anda melalui seluruh proses, mulai dari persiapan yang rumit hingga menampilkan peta yang fungsional dan interaktif. Perjalanan kita akan mencakup:

* Mendapatkan Kunci API dari Google Cloud Platform.
* Menampilkan peta dasar di dalam aplikasi kita ('Hello, Map!').
* Mengontrol kamera untuk berpindah dan melakukan zoom ke lokasi tertentu.
* Menambahkan penanda (*markers*) untuk menandai titik-titik penting.
* Menggambar bentuk seperti garis rute dan area di atas peta.
* Dan puncaknya, menampilkan lokasi pengguna saat ini secara *real-time*.

Ini akan menjadi tantangan yang sangat memuaskan. Mari kita mulai petualangan kartografi digital kita.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai Bab 13.

Berikut adalah jawaban untuk prompt **Subbab 13.1: Persiapan Awal: Kunci API dan Konfigurasi Proyek**.

**13.1. Persiapan Awal: Kunci API dan Konfigurasi Proyek**

Sebelum kita bisa menampilkan sepotong pun peta di aplikasi kita, ada beberapa pekerjaan rumah administratif yang wajib kita selesaikan. Proses ini mungkin terasa sedikit panjang dan rumit, tetapi ini hanya perlu dilakukan sekali di awal proyek. Langkah-langkah ini bertujuan untuk mengautentikasi aplikasi kita ke layanan Google dan memastikan semua *library* yang dibutuhkan sudah terkonfigurasi. Ikuti setiap langkah dengan saksama.

**Langkah 1: Mendapatkan Kunci API dari Google Cloud Platform**

Peta Google bukanlah layanan gratis tanpa batas; ia memerlukan kunci API untuk melacak penggunaan dan mengautentikasi aplikasi Anda.

1. **Buat atau Pilih Proyek:** Buka browser Anda dan navigasikan ke **Google Cloud Console** di console.cloud.google.com. Login dengan akun Google Anda. Buat sebuah proyek baru (misalnya, 'Aplikasi Peta LAMW') atau pilih proyek yang sudah ada.
2. **Aktifkan Maps SDK for Android:** Di dalam dasbor proyek Anda, gunakan bilah pencarian di bagian atas untuk mencari **"Maps SDK for Android"**. Pilih hasil yang sesuai dari daftar, lalu klik tombol **Enable**. Anda mungkin akan diminta untuk mengaitkan akun penagihan (*billing account*) dengan proyek Anda. Jangan khawatir, Google menyediakan kuota penggunaan gratis yang sangat besar setiap bulannya yang cukup untuk tahap pengembangan dan aplikasi skala kecil.
3. **Buat Kunci API:** Setelah API diaktifkan, navigasikan ke menu APIs & Services > Credentials. Klik tombol + CREATE CREDENTIALS di bagian atas, lalu pilih **API key**. Sebuah kunci API baru akan dibuat dan ditampilkan dalam sebuah dialog.
4. **(SANGAT PENTING) Batasi Kunci API Anda:** Sebuah kunci API yang tidak dibatasi bisa disalahgunakan oleh orang lain dan berpotensi menimbulkan tagihan. Kita harus segera mengamankannya.
   * Klik tombol EDIT API KEY atau klik nama kunci API yang baru dibuat.
   * Di bawah **Application restrictions**, pilih opsi **Android apps**.
   * Klik **+ ADD AN ITEM**.
   * Anda akan diminta memasukkan **Nama Paket** (*Package Name*) aplikasi Anda (contoh: com.namaanda.aplikasipeta) dan **sidik jari sertifikat SHA-1**.
   * Untuk mendapatkan SHA-1, Anda bisa menggunakan perintah keytool pada file *keystore* yang kita buat di Bab 12. Buka Command Prompt, navigasikan ke folder *keystore* Anda, dan jalankan: keytool -list -v -keystore NAMA\_FILE\_KEYSTORE.keystore -alias NAMA\_ALIAS\_KUNCI
   * Salin nilai SHA-1 yang muncul dan tempelkan di kolom yang sesuai di Google Cloud Console.
   * Klik Save. Sekarang, kunci API Anda hanya bisa digunakan oleh aplikasi Anda yang sudah ditandatangani.
5. **Salin Kunci API Anda:** Kembali ke daftar kredensial dan salin kunci API Anda (berupa string panjang dan acak) ke Notepad. Kita akan menggunakannya sebentar lagi.

**Langkah 2: Menambahkan Dependensi Library Peta**

Aplikasi kita perlu menyertakan *library* Google Play Services untuk peta. Di LAMW, ini biasanya diatur dalam file konfigurasi *build* Gradle. Anda perlu memastikan bahwa dependensi berikut disertakan.

Groovy

// Di dalam file build.gradle (atau file konfigurasi setara di proyek LAMW Anda)

dependencies {

...

implementation 'com.google.android.gms:play-services-maps:18.2.0' // Versi bisa berubah

}

*Catatan: Pastikan untuk memeriksa dokumentasi LAMW terbaru untuk cara yang paling tepat dalam menambahkan dependensi Gradle eksternal.*

**Langkah 3: Menambahkan Kunci API ke AndroidManifest.xml**

Terakhir, kita harus menempatkan kunci API kita di dalam *manifest* aplikasi agar bisa dibaca oleh *library* peta saat aplikasi berjalan.

1. Buka file AndroidManifest.xml proyek Anda.
2. Di dalam tag <application>, tambahkan tag <meta-data> berikut:

XML

<application ...>

...

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="PASTE\_KUNCI\_API\_ANDA\_DI\_SINI" />

...

</application>

1. Ganti PASTE\_KUNCI\_API\_ANDA\_DI\_SINI dengan kunci API yang telah Anda salin dari Google Cloud Console.

Dengan semua persiapan ini selesai, proyek Anda sekarang secara teknis siap untuk menampilkan peta Google.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.2.

yahok

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 13.2: Menampilkan Peta Pertama Anda ("Hello, Map!")**.

**13.2. Menampilkan Peta Pertama Anda ("Hello, Map!")**

Setelah semua persiapan yang melelahkan di subbab sebelumnya selesai, saatnya kita melihat hasilnya. Di subbab ini, tujuan kita sederhana namun sangat memuaskan: menampilkan sebuah peta Google yang interaktif di dalam aplikasi LAMW kita. Ini adalah 'Hello, World!'-nya pengembangan peta.

**Langkah 1: Menyiapkan Kontainer Peta**

Peta Google ditampilkan oleh sebuah komponen UI dari *library* Java yang disebut **MapView**. Karena ini bukan komponen Pascal asli, kita tidak bisa langsung meletakkannya dari *Component Palette*. Sebaliknya, kita akan menyiapkan sebuah 'wadah' atau 'kontainer' kosong di *layout* kita, lalu kita akan memasukkan MapView ke dalamnya secara terprogram melalui JNI.

1. Di *Form Designer* Anda, tambahkan sebuah jFrameLayout.
2. Di *Object Inspector*, atur ID-nya menjadi mapContainer.
3. Atur layout\_width dan layout\_height-nya menjadi match\_parent agar kontainer ini memenuhi seluruh layar.

**Langkah 2: Inisialisasi Peta dan Siklus Hidup (Lifecycle)**

Seluruh logika ini akan kita tulis di unit Form kita. Ini akan menjadi contoh implementasi JNI yang cukup kompleks, jadi perhatikan setiap detailnya.

Pertama, di bagian private dari kelas Form (TAndroidModule1) Anda, kita butuh variabel untuk menyimpan referensi ke MapView dan GoogleMap kita:

Delphi

private

{ private declarations }

FMapView: jMapView;

FGoogleMap: jGoogleMap;

Selanjutnya, di *event* OnCreate form, kita akan membuat MapView dan menambahkannya ke kontainer.

Delphi

// di event OnCreate ...

// Buat MapView menggunakan wrapper JNI dari LAMW

FMapView := tjMapView.Create(Self);

// Tambahkan MapView ke dalam FrameLayout yang sudah kita siapkan

mapContainer.AddView(FMapView);

// PENTING: Teruskan event OnCreate ke MapView

FMapView.onCreate(nil);

**Meneruskan Event Siklus Hidup (Wajib!)** Objek MapView perlu tahu tentang siklus hidup Activity (Form) kita agar bisa mengelola sumber daya dengan benar (misalnya, menghentikan render saat aplikasi di-*pause*). Kita **wajib** meneruskan *event-event* siklus hidup ini. Buat *event-event* berikut untuk jForm1 dari *Object Inspector*:

Delphi

procedure TAndroidModule1.jForm1Resume(Sender: TObject);

begin

if Assigned(FMapView) then FMapView.onResume;

end;

procedure TAndroidModule1.jForm1Pause(Sender: TObject);

begin

if Assigned(FMapView) then FMapView.onPause;

end;

procedure TAndroidModule1.jForm1Destroy(Sender: TObject);

begin

if Assigned(FMapView) then FMapView.onDestroy;

end;

**Langkah 3: Mendapatkan Objek Peta dengan OnMapReadyCallback**

Menampilkan peta adalah proses asinkron. Kita tidak bisa langsung memanipulasi peta setelah onCreate. Kita harus 'meminta' objek peta dan 'menunggu' hingga ia siap. Proses ini dilakukan melalui sebuah *callback*.

1. **Implementasi Interface:** Form kita harus mengimplementasikan *interface* OnMapReadyCallback. Di deklarasi kelas Form Anda, tambahkan [Implements] dan deklarasikan metodenya.

Delphi

TAndroidModule1 = class(jForm)

...

private

FMapView: jMapView;

FGoogleMap: jGoogleMap;

public

[Implements(tiOnMapReadyCallback)] // <-- Tambahkan ini

procedure onMapReady(googleMap: jGoogleMap);

end;

1. **Minta Peta Secara Asinkron:** Kembali ke *event* OnCreate, setelah kode sebelumnya, tambahkan baris ini:

Delphi

// ... di dalam OnCreate, setelah FMapView.onCreate(nil);

FMapView.getMapAsync(Self); // 'Self' (form ini) adalah listener-nya

1. **Implementasi Metode onMapReady:** Sekarang, kita tulis kode untuk metode *callback* tersebut. Metode ini akan secara otomatis dipanggil oleh Google setelah peta berhasil dimuat dan siap digunakan.

Delphi

procedure TAndroidModule1.onMapReady(googleMap: jGoogleMap);

var

\_latLng: jLatLng;

begin

// Simpan referensi ke objek GoogleMap agar bisa kita gunakan nanti

FGoogleMap := googleMap;

// Peta sudah siap! Mari kita coba pindahkan kamera ke lokasi tertentu

// Contoh: Monumen Nasional, Jakarta

\_latLng := tjLatLng.Create(-6.175392, 106.827153);

FGoogleMap.moveCamera(tjCameraUpdateFactory.newLatLngZoom(\_latLng, 15));

ShowMessage('Peta berhasil dimuat!');

end;

Kompilasi dan jalankan aplikasi Anda. Jika semua konfigurasi Kunci API dan dependensi sudah benar, setelah beberapa saat Anda akan melihat peta interaktif yang terpusat di Monas. Selamat, Anda telah berhasil menampilkan peta Google pertama Anda di aplikasi LAMW!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 13.3: Mengontrol Kamera Peta**.

**13.3. Mengontrol Kamera Peta**

Setelah peta berhasil tampil, hal pertama yang pasti ingin kita lakukan adalah mengontrol 'kamera' atau sudut pandang peta. Kita ingin bisa memindahkan fokus ke lokasi manapun di dunia, mengatur tingkat zoom, dan melakukannya dengan gerakan yang mulus.

Semua perintah untuk mengubah posisi kamera dibuat menggunakan sebuah kelas pabrik bernama CameraUpdateFactory. Kode-kode berikut diasumsikan dijalankan di dalam atau dipanggil dari metode onMapReady, setelah kita yakin bahwa objek FGoogleMap kita sudah siap untuk digunakan.

**1. Pindah ke Lokasi Baru (newLatLng)**

Ini adalah perintah paling dasar: memindahkan pusat peta ke koordinat Latitude dan Longitude yang baru secara instan.

Delphi

procedure TAndroidModule1.FokusKeLokasi(ALatitude, ALongitude: double);

var

lokasiTujuan: jLatLng;

update: jCameraUpdate;

begin

if not Assigned(FGoogleMap) then Exit; // Pengaman jika peta belum siap

// 1. Tentukan koordinat tujuan

lokasiTujuan := tjLatLng.Create(ALatitude, ALongitude);

// 2. Buat perintah 'update kamera' untuk pindah ke lokasi tersebut

update := tjCameraUpdateFactory.newLatLng(lokasiTujuan);

// 3. Eksekusi perintah untuk memindahkan kamera

FGoogleMap.moveCamera(update);

end;

Anda bisa memanggil FokusKeLokasi(48.8583, 2.2944) untuk langsung melompat ke Menara Eiffel, Paris.

**2. Pindah Lokasi Sekaligus Zoom (newLatLngZoom)**

Sering kali, kita tidak hanya ingin pindah, tetapi juga langsung mengatur seberapa dekat kita melihat lokasi tersebut. Untuk ini, kita gunakan newLatLngZoom.

Tingkat zoom pada Google Maps biasanya berkisar dari 1 (melihat seluruh dunia) hingga sekitar 21 (melihat detail bangunan). Nilai antara **15-17** adalah tingkat yang umum untuk melihat detail jalanan di sebuah kota.

Delphi

procedure TAndroidModule1.FokusKeLokasiDenganZoom(ALatitude, ALongitude: double; zoomLevel: single);

var

lokasiTujuan: jLatLng;

update: jCameraUpdate;

begin

if not Assigned(FGoogleMap) then Exit;

// 1. Tentukan koordinat

lokasiTujuan := tjLatLng.Create(ALatitude, ALongitude);

// 2. Buat perintah 'update kamera' dengan informasi lokasi dan zoom

update := tjCameraUpdateFactory.newLatLngZoom(lokasiTujuan, zoomLevel);

// 3. Eksekusi perintah

FGoogleMap.moveCamera(update);

end;

Untuk melihat Masjid Raya Baiturrahman di Banda Aceh dari dekat, Anda bisa memanggil FokusKeLokasiDenganZoom(5.5539, 95.3174, 17).

**3. Gerakan Kamera yang Mulus (animateCamera)**

Perintah moveCamera akan memindahkan peta secara instan, yang terkadang terasa kasar dan meloncat. Untuk memberikan pengalaman pengguna yang lebih baik dan profesional, kita bisa menggunakan animateCamera. Perintah ini akan menggerakkan kamera dari posisi lama ke posisi baru dengan animasi yang mulus.

Penggunaannya sama persis, cukup ganti moveCamera dengan animateCamera.

Delphi

procedure TAndroidModule1.FokusKeLokasiAnimasi(ALatitude, ALongitude: double; zoomLevel: single);

var

lokasiTujuan: jLatLng;

update: jCameraUpdate;

begin

if not Assigned(FGoogleMap) then Exit;

lokasiTujuan := tjLatLng.Create(ALatitude, ALongitude);

update := tjCameraUpdateFactory.newLatLngZoom(lokasiTujuan, zoomLevel);

// Gunakan animateCamera untuk pergerakan yang mulus

FGoogleMap.animateCamera(update);

// Anda juga bisa mengatur durasi animasi dalam milidetik

// FGoogleMap.animateCamera(update, 2000, nil); // Animasi akan berjalan selama 2 detik

end;

Cukup dengan mengganti satu kata, pengalaman pengguna aplikasi Anda akan terasa jauh lebih premium.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 13.4: Bekerja dengan Penanda (Markers)**.

**13.4. Bekerja dengan Penanda (Markers)**

Peta tanpa penanda bagaikan buku tanpa nomor halaman. **Marker** (Penanda) adalah ikon atau 'pin' yang kita letakkan di koordinat tertentu untuk menandai lokasi penting, seperti restoran, kantor, tempat wisata, atau posisi teman. Di subbab ini, kita akan belajar cara menambah, mengkustomisasi, dan berinteraksi dengan *marker*.

**1. Menambahkan Marker Sederhana**

Cara kerja untuk menambahkan *marker* adalah dengan menggunakan pola 'builder'. Kita tidak membuat *marker* secara langsung, melainkan kita membuat sebuah objek **MarkerOptions** terlebih dahulu, mengonfigurasi semua propertinya (posisi, judul, dll.), lalu menyerahkan 'resep' ini ke peta untuk dibuatkan *marker*-nya.

Delphi

procedure TAndroidModule1.TambahMarkerSederhana(ALatitude, ALongitude: double; ATitle, ASnippet: string);

var

posisi: jLatLng;

markerOptions: jMarkerOptions;

begin

if not Assigned(FGoogleMap) then Exit; // Pengaman

// 1. Tentukan posisi geografis untuk marker

posisi := tjLatLng.Create(ALatitude, ALongitude);

// 2. Buat dan konfigurasi objek MarkerOptions

markerOptions := tjMarkerOptions.Create;

markerOptions.position(posisi);

markerOptions.title(StrToJCharSequence(ATitle)); // Judul tebal yang muncul saat diklik

markerOptions.snippet(StrToJCharSequence(ASnippet)); // Teks deskripsi di bawah judul

// 3. Tambahkan marker ke peta

FGoogleMap.addMarker(markerOptions);

end;

Saat Anda memanggil prosedur ini, misalnya TambahMarkerSederhana(5.5539, 95.3174, 'Masjid Raya Baiturrahman', 'Ikon Kota Banda Aceh'), sebuah pin merah standar akan muncul di lokasi tersebut. Ketika pin diklik, sebuah **Info Window** kecil akan muncul menampilkan judul dan *snippet*-nya.

**2. Interaksi dengan Marker (OnMarkerClickListener)**

Terkadang kita ingin melakukan aksi lebih dari sekadar menampilkan Info Window, misalnya membuka layar detail saat *marker* diklik. Untuk ini, kita gunakan OnMarkerClickListener.

1. **Implementasi Interface:** Pertama, *form* kita perlu mengimplementasikan *interface* OnMarkerClickListener.

Delphi

TAndroidModule1 = class(jForm)

...

public

[Implements(tiOnMapReadyCallback, tiOnMarkerClickListener)] // <-- Tambahkan tiOnMarkerClickListener

procedure onMapReady(googleMap: jGoogleMap);

function onMarkerClick(marker: jMarker): boolean; // <-- Deklarasikan metode callback ini

end;

1. **Daftarkan Listener:** Di dalam onMapReady, setelah FGoogleMap siap, kita daftarkan *form* ini sebagai *listener*-nya.

Delphi

// ...di dalam metode onMapReady...

FGoogleMap.setOnMarkerClickListener(Self);

1. **Implementasikan Metode onMarkerClick:** Tulis logika yang Anda inginkan saat *marker* diklik.

Delphi

function TAndroidModule1.onMarkerClick(marker: jMarker): boolean;

var

judul: string;

begin

judul := JCharSequenceToString(marker.getTitle);

ShowMessage('Anda mengklik marker: ' + judul);

// Return 'False' jika kita ingin Info Window standar tetap muncul setelah kode ini berjalan.

// Return 'True' jika kita sudah menangani event ini sepenuhnya dan tidak ingin Info Window muncul.

Result := False;

end;

**3. Kustomisasi Tampilan Marker**

Pin merah standar mungkin membosankan atau tidak sesuai dengan tema aplikasi Anda. Untungnya, kita bisa dengan mudah mengubahnya.

**A. Mengubah Warna Pin Bawaan** Kita bisa mengubah warna pin standar menggunakan konstanta HUE dari jBitmapDescriptorFactory. Modifikasi kode TambahMarkerSederhana Anda:

Delphi

// ... di dalam prosedur TambahMarkerSederhana ...

markerOptions.position(posisi);

markerOptions.title(StrToJCharSequence(ATitle));

// Tambahkan baris ini untuk mengubah warna

markerOptions.icon(tjBitmapDescriptorFactory.defaultMarker(jBitmapDescriptorFactory.HUE\_AZURE));

FGoogleMap.addMarker(markerOptions);

Ini akan mengubah pin menjadi berwarna biru. Tersedia berbagai warna lain seperti HUE\_GREEN, HUE\_VIOLET, HUE\_ORANGE, dll.

**B. Menggunakan Ikon Kustom dari drawable** Untuk kustomisasi penuh, kita bisa menggunakan gambar kita sendiri.

1. **Persiapan:** Letakkan file gambar .png Anda (misalnya, ikon\_toko.png) di dalam direktori res/drawable proyek Anda.
2. **Kode:** Ganti baris kustomisasi warna tadi dengan yang ini:

Delphi

// ... di dalam prosedur TambahMarkerSederhana ...

// Ganti res.drawable.ikon\_toko dengan nama resource Anda

markerOptions.icon(tjBitmapDescriptorFactory.fromResource(res.drawable.ikon\_toko));

// ...

Sekarang, *marker* Anda akan tampil sebagai ikon toko yang telah Anda desain, memberikan tampilan yang unik dan profesional pada peta Anda.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 13.5: Menggambar di Atas Peta (Polylines dan Polygons)**.

**13.5. Menggambar di Atas Peta (Polylines dan Polygons)**

Selain menandai titik-titik tunggal dengan *marker*, Google Maps API juga memungkinkan kita untuk menggambar bentuk-bentuk geometris langsung di atas peta. Ini sangat berguna untuk berbagai keperluan, seperti memvisualisasikan rute perjalanan, menandai sebuah area terlarang, atau menunjukkan batas sebuah wilayah. Peta Anda bisa menjadi sebuah kanvas informasi geografis yang dinamis.

Dua bentuk yang paling sering digunakan adalah **Polyline** (garis) dan **Polygon** (area tertutup).

**1. Menggambar Garis (Polylines)**

**Polyline** adalah serangkaian titik koordinat yang terhubung oleh segmen garis lurus. Ini sempurna untuk menggambarkan sebuah rute dari titik A ke B, jalur lari, atau batas sebuah jalan. Sama seperti *marker*, kita menggunakan pola 'builder' dengan membuat objek **PolylineOptions** terlebih dahulu.

* **Contoh Kode: Menggambar Rute Wisata Sederhana di Banda Aceh**

Delphi

procedure TAndroidModule1.GambarRuteWisata;

var

ruteOptions: jPolylineOptions;

titikA, titikB, titikC: jLatLng;

begin

if not Assigned(FGoogleMap) then Exit;

// 1. Definisikan titik-titik koordinat untuk rute kita

titikA := tjLatLng.Create(5.5539, 95.3174); // Masjid Raya Baiturrahman

titikB := tjLatLng.Create(5.5705, 95.3402); // Museum Tsunami Aceh

titikC := tjLatLng.Create(5.5583, 95.3378); // PLTD Apung

// 2. Buat dan konfigurasi PolylineOptions

ruteOptions := tjPolylineOptions.Create;

ruteOptions.add(titikA); // Tambahkan titik secara berurutan

ruteOptions.add(titikB);

ruteOptions.add(titikC);

ruteOptions.width(12); // Lebar garis dalam piksel

ruteOptions.color(jColor.BLUE); // Warna garis

// 3. Tambahkan polyline ke peta

FGoogleMap.addPolyline(ruteOptions);

end;

**2. Menggambar Area (Polygons)**

**Polygon** mirip dengan *polyline*, tetapi ia adalah sebuah bentuk tertutup dan bisa memiliki warna isian (*fill color*). Ini ideal untuk menandai sebuah zona, seperti area pengiriman gratis, batas sebuah taman, atau wilayah sebuah properti. Prosesnya hampir sama, tetapi menggunakan objek **PolygonOptions**.

* **Contoh Kode: Menandai Area Taman**

Delphi

procedure TAndroidModule1.GambarAreaTaman;

var

areaOptions: jPolygonOptions;

sudut1, sudut2, sudut3, sudut4: jLatLng;

begin

if not Assigned(FGoogleMap) then Exit;

// 1. Definisikan titik-titik sudut dari area (contoh imajiner)

sudut1 := tjLatLng.Create(5.5550, 95.3190);

sudut2 := tjLatLng.Create(5.5560, 95.3200);

sudut3 := tjLatLng.Create(5.5555, 95.3205);

sudut4 := tjLatLng.Create(5.5545, 95.3195);

// 2. Buat dan konfigurasi PolygonOptions

areaOptions := tjPolygonOptions.Create;

areaOptions.add(sudut1);

areaOptions.add(sudut2);

areaOptions.add(sudut3);

areaOptions.add(sudut4);

areaOptions.strokeWidth(7); // Lebar garis batas

areaOptions.strokeColor(jColor.RED); // Warna garis batas

areaOptions.fillColor($8000FF00); // Warna isian (Hijau dengan 50% transparansi)

// 3. Tambahkan polygon ke peta

FGoogleMap.addPolygon(areaOptions);

end;

**Catatan Warna:** Warna isian (fillColor) menggunakan format heksadesimal AARRGGBB (Alpha, Red, Green, Blue). Nilai 80 di awal merepresentasikan sekitar 50% transparansi, memungkinkan peta di bawahnya tetap terlihat.

Dengan *polylines* dan *polygons*, Anda dapat menyajikan data geografis dengan cara yang jauh lebih kaya dan informatif.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 13.6.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 13.6: Menampilkan Lokasi Pengguna**.

**13.6. Menampilkan Lokasi Pengguna**

Inilah salah satu fitur peta yang paling sering diminta dan paling berguna: menampilkan lokasi pengguna saat ini secara *real-time*. Untuk mengimplementasikan fitur ini, kita akan melakukan sebuah 'sintesis' atau penggabungan dari berbagai keahlian yang telah kita pelajari di bab-bab sebelumnya. Ini adalah bukti nyata bagaimana konsep-konsep yang berbeda dapat bekerja sama untuk menciptakan sebuah fitur yang canggih.

**Menyatukan Keahlian dari Bab-bab Sebelumnya**

Untuk menampilkan lokasi pengguna di peta, kita akan menggunakan:

* **Manajemen Izin Runtime (dari Bab 6):** Kita akan meminta izin ACCESS\_FINE\_LOCATION kepada pengguna sebelum melakukan apa pun.
* **Sensor Lokasi (jLocationSensor dari Bab 8):** Kita akan menggunakan komponen ini untuk mendapatkan update koordinat Latitude dan Longitude dari perangkat.
* **Kontrol Kamera Peta (dari Subbab 13.3):** Kita akan menggunakan CameraUpdateFactory untuk menggerakkan peta ke lokasi yang diberikan oleh sensor.

**Langkah 1: Mengaktifkan Layer 'My Location'**

Google Maps API menyediakan cara yang sangat mudah untuk menampilkan lokasi pengguna. Ia memiliki *layer* bawaan yang akan menampilkan titik biru sebagai penanda lokasi pengguna dan sebuah tombol 'target' untuk memusatkan peta ke lokasi tersebut secara otomatis.

**Prasyarat:** Pastikan Anda sudah memiliki kode untuk meminta dan mendapatkan izin ACCESS\_FINE\_LOCATION. Kode berikut hanya boleh dijalankan **setelah** izin diberikan oleh pengguna.

Delphi

// Prosedur ini bisa dipanggil dari onMapReady setelah FGoogleMap siap

procedure TAndroidModule1.AktifkanLayerLokasiSaya;

begin

if not Assigned(FGoogleMap) then Exit;

// Periksa kembali izinnya untuk keamanan, meskipun seharusnya sudah diperiksa sebelumnya

if jPermissions.Check(Manifest\_permission\_ACCESS\_FINE\_LOCATION) then

begin

// Menampilkan titik biru lokasi di peta

FGoogleMap.setMyLocationEnabled(True);

// Menampilkan tombol 'center' di pojok kanan atas peta

FGoogleMap.GetUiSettings.setMyLocationButtonEnabled(True);

end;

end;

Dengan memanggil prosedur ini, peta Anda kini "sadar lokasi". Pengguna bisa menekan tombol target untuk memusatkan peta ke lokasi mereka kapan pun.

**Langkah 2: Memusatkan Peta ke Lokasi Pengguna Saat Pertama Kali**

Terkadang, kita ingin peta langsung terpusat ke lokasi pengguna saat pertama kali dibuka, tanpa menunggu pengguna menekan tombol. Ini memberikan pengalaman awal yang lebih baik. Untuk ini, kita akan mengkombinasikannya dengan jLocationSensor.

**Arsitektur:** Kita akan mengambil **satu kali** *update* lokasi saat peta siap, memindahkan kamera ke sana, lalu mematikan sensor kita untuk menghemat baterai dan menyerahkan kontrol selanjutnya kepada tombol bawaan peta.

1. Pastikan Anda sudah meletakkan komponen jLocationSensor di form Anda.
2. Modifikasi onMapReady dan tambahkan prosedur *event handler* baru.

Delphi

// ... di dalam onMapReady ...

procedure TAndroidModule1.onMapReady(googleMap: jGoogleMap);

begin

FGoogleMap := googleMap;

// ... (kode setup lainnya seperti setOnMarkerClickListener, dll.) ...

// Cek izin, jika ada, langsung coba pusatkan peta

if jPermissions.Check(Manifest\_permission\_ACCESS\_FINE\_LOCATION) then

begin

AktifkanLayerLokasiSaya;

// Gunakan OnLocationChanged untuk mendapatkan lokasi pertama

jLocationSensor1.OnLocationChanged := @Self.OnFirstLocationUpdate;

jLocationSensor1.Start; // Aktifkan sensor untuk mencari lokasi

end;

end;

// Buat prosedur baru ini untuk menangani HANYA update lokasi pertama

procedure TAndroidModule1.OnFirstLocationUpdate(Sender: TObject;

ALatitude, ALongitude, AAltitude: double; ABearing, ASpeed: single);

var

lokasiSaatIni: jLatLng;

begin

// Setelah lokasi pertama didapat, langsung hentikan sensor untuk menghemat baterai

jLocationSensor1.Stop;

// Kosongkan event handler agar prosedur ini tidak dipanggil lagi untuk update selanjutnya

jLocationSensor1.OnLocationChanged := nil;

// Animasikan kamera ke lokasi pengguna dengan zoom yang sesuai

lokasiSaatIni := tjLatLng.Create(ALatitude, ALongitude);

FGoogleMap.animateCamera(tjCameraUpdateFactory.newLatLngZoom(lokasiSaatIni, 16));

end;

Pola ini sangat efisien. Kita hanya mengaktifkan sensor GPS sesaat untuk mendapatkan lokasi awal. Setelah lokasi didapat dan kamera dipindahkan, kita langsung mematikan sensornya dan menyerahkan navigasi selanjutnya kepada pengguna melalui tombol 'My Location' yang sudah disediakan oleh API.

Ini mengakhiri Bab 13. Selamat, Anda telah berhasil mengintegrasikan salah satu fitur paling canggih ke dalam aplikasi LAMW. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 14 (Pengantar Bab 14).

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Pengantar Bab 14**.

**Bab 14: Membangun Aplikasi Cerdas dengan Gemini AI API**

Selamat datang di garda terdepan inovasi aplikasi. Jika bab-bab sebelumnya kita telah belajar membuat aplikasi yang 'tahu' (data) dan 'melihat' (peta), maka di bab ini kita akan belajar membuat aplikasi yang bisa **'berpikir' dan 'berkreasi'**. Kecerdasan Buatan atau *Artificial Intelligence* (AI), khususnya AI Generatif, bukan lagi sekadar konsep masa depan. Ia telah menjadi sebuah *toolkit* praktis yang bisa diintegrasikan oleh developer untuk menciptakan fitur-fitur yang sebelumnya tidak terbayangkan.

Di bab ini, kita akan berkenalan dengan salah satu model AI paling canggih dari Google: **Gemini**. Gemini adalah model AI **multimodal**, artinya ia tidak hanya memahami teks, tetapi juga gambar, audio, dan video, memungkinkannya untuk melakukan penalaran yang kompleks. Tujuan kita adalah untuk 'menyuntikkan' kecerdasan Gemini ke dalam aplikasi LAMW kita. Kita akan belajar cara berkomunikasi dengan 'otak' Gemini melalui API-nya, memberikan perintah, dan menerima hasil kreasinya.

Bab ini bersifat sangat praktis dan teknis, serta sangat mengandalkan keahlian yang telah Anda bangun di **Bab 7 (Komunikasi Jaringan & API)**, karena semua interaksi dengan Gemini API dilakukan melalui permintaan HTTP dan data JSON. Kemampuan untuk menangani proses asinkron dari **Bab 9 (Threading)** juga akan sangat relevan.

Petualangan kita di dunia AI ini akan mencakup tiga proyek mini yang menarik:

* **Generator Teks Sederhana:** Mengirim sebuah *prompt* dan mendapatkan respons teks kreatif dari Gemini.
* **Penganalisis Gambar:** Mengirim sebuah gambar beserta pertanyaan, dan membiarkan Gemini 'melihat' dan mendeskripsikan apa yang ada di dalamnya.
* **Chatbot Dasar:** Membangun sebuah antarmuka percakapan sederhana di mana Gemini bisa mengingat konteks dari obrolan sebelumnya.

Membuka pintu ke dunia AI mungkin terasa mengintimidasi, tetapi Anda akan terkejut betapa mudahnya proses integrasi dasarnya. Bersiaplah untuk membuat aplikasi Anda menjadi jauh lebih pintar.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 14.1.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 14.1: Persiapan dan Konfigurasi Gemini API**.

**14.1. Persiapan dan Konfigurasi Gemini API**

Sama seperti layanan Google lainnya, untuk bisa berkomunikasi dengan Gemini, aplikasi kita memerlukan sebuah 'kunci' sebagai identitas dan otorisasi. Proses untuk mendapatkan kunci Gemini API untungnya jauh lebih sederhana dibandingkan Google Maps. Mari kita siapkan semuanya.

**Langkah 1: Mendapatkan Kunci API dari Google AI Studio**

1. **Kunjungi Google AI Studio:** Buka browser Anda dan navigasikan ke [**https://aistudio.google.com/**](https://aistudio.google.com/). Login dengan akun Google Anda.
2. **Buat Kunci API Baru:** Di dalam antarmuka AI Studio, cari tombol atau menu **Get API key**. Biasanya ada di panel sebelah kiri. Klik pada opsi tersebut.
3. **Hasilkan Kunci:** Anda akan dibawa ke halaman Google AI for Developers. Klik tombol **Create API key in new project**. Sebuah string panjang yang berisi huruf dan angka acak akan dibuat. Inilah Kunci API Anda.
4. **Salin dan Amankan:** **Segera salin (*copy*) Kunci API ini** dan simpan di tempat yang aman, misalnya di file Notepad untuk sementara selama proses pengembangan. Jangan pernah bagikan kunci ini kepada siapa pun.

**Langkah 2: Memahami Endpoint dan Header Permintaan**

Semua permintaan ke Gemini API akan dikirim ke sebuah URL dasar (*base URL*) yang sama, dengan beberapa variasi tergantung model yang digunakan. Untuk model teks standar gemini-pro yang akan kita gunakan, *endpoint*-nya adalah:

https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent

Selain itu, setiap permintaan yang kita kirim harus menyertakan dua **HTTP Header** yang wajib:

* **x-goog-api-key**: Di sinilah kita akan meletakkan nilai Kunci API yang baru saja kita dapatkan.
* **Content-Type**: Nilainya harus selalu application/json untuk memberitahu server bahwa data yang kita kirim dalam *body* permintaan adalah dalam format JSON.

**Peringatan Keamanan Kunci API (Sangat Penting!)**

**JANGAN PERNAH MENYIMPAN KUNCI API SECARA LANGSUNG DI KODE ANDA (*HARDCODE*) UNTUK APLIKASI PRODUKSI!**

Menulis Kunci API Anda langsung di dalam file .pas seperti const API\_KEY = 'AIzaSy...'; adalah praktik yang sangat tidak aman. Jika seseorang berhasil mendekompilasi aplikasi Anda, mereka bisa mencuri kunci tersebut dan menggunakannya atas nama Anda, yang berpotensi menimbulkan tagihan yang tidak diinginkan.

**Solusi yang Lebih Baik (untuk Produksi):**

* **Sisi Server:** Cara paling aman adalah aplikasi Anda berkomunikasi dengan server milik Anda sendiri, dan server inilah yang memiliki kunci dan berkomunikasi dengan API Gemini. Aplikasi Anda tidak pernah memegang kunci API secara langsung.
* **Konfigurasi Build:** Menyimpan kunci di file konfigurasi *build* (seperti gradle.properties) yang tidak ikut masuk ke dalam sistem kontrol versi (*source control*).

Untuk keperluan belajar di buku ini, kita mungkin akan menggunakannya sebagai konstanta agar contohnya sederhana. Namun, ingatlah untuk **tidak pernah melakukan ini untuk aplikasi yang akan dirilis ke publik**.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 14.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 14.2: Interaksi Pertama: Permintaan Teks-ke-Teks**.

**14.2. Interaksi Pertama: Permintaan Teks-ke-Teks**

Persiapan sudah selesai, saatnya beraksi! Interaksi paling dasar dengan Gemini adalah mengirimkan sebuah pertanyaan atau perintah (disebut *prompt*) dalam bentuk teks, dan menerima jawaban dalam bentuk teks. Kita akan membangun aplikasi sederhana untuk melakukan hal ini.

**Persiapan UI:** Siapkan UI Anda di *Form Designer*:

* Sebuah jEditText dengan properti MultiLine diatur ke True untuk tempat menulis *prompt* (beri ID: edtPrompt).
* Sebuah jButton dengan ID: btnKirimPrompt dan Text: Kirim.
* Sebuah jTextView untuk menampilkan hasil (ID: tvHasilGemini).
* Jangan lupa tambahkan jHttpClient dan jProgressBar ke form Anda.

**Langkah 1: Membuat Body Permintaan JSON**

Gemini API menerima permintaan dalam format JSON yang spesifik. Untuk permintaan teks sederhana, strukturnya terlihat seperti ini:

JSON

{

"contents": [

{

"parts": [

{

"text": "Tulis sebuah puisi singkat tentang kopi."

}

]

}

]

}

Kita perlu membuat *string* JSON ini secara dinamis berdasarkan input pengguna. Kita bisa membuat sebuah fungsi pembantu untuk ini.

Delphi

// Jangan lupa tambahkan fpjson, jsonparser ke uses clause

function CreateTextPromptJSON(const aPrompt: string): string;

var

jsonRequest, jsonContent, jsonPart: TJSONObject;

jsonArrayContents, jsonArrayParts: TJSONArray;

begin

jsonRequest := TJSONObject.Create;

jsonArrayContents := TJSONArray.Create;

jsonContent := TJSONObject.Create;

jsonArrayParts := TJSONArray.Create;

jsonPart := TJSONObject.Create;

try

jsonPart.Add('text', aPrompt);

jsonArrayParts.Add(jsonPart);

jsonContent.Add('parts', jsonArrayParts);

jsonArrayContents.Add(jsonContent);

jsonRequest.Add('contents', jsonArrayContents);

Result := jsonRequest.ToJSON;

finally

jsonRequest.Free; // Ini akan membebaskan semua objek anak

end;

end;

**Langkah 2: Mengirim Permintaan POST Asinkron**

Dengan fungsi pembuat JSON di tangan, sekarang kita bisa mengirim permintaan POST di dalam OnClick tombol 'Kirim'.

Delphi

const

GEMINI\_API\_KEY = 'MASUKKAN\_KUNCI\_API\_ANDA\_DI\_SINI';

GEMINI\_URL = 'https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent';

procedure TAndroidModule1.btnKirimPromptClick(Sender: TObject);

var

jsonBody: string;

begin

if edtPrompt.Text = '' then Exit;

jProgressBar1.Visible := True;

btnKirimPrompt.Enabled := False;

tvHasilGemini.Text := 'Sedang berpikir...';

// Buat body JSON dari input pengguna

jsonBody := CreateTextPromptJSON(edtPrompt.Text);

// Konfigurasi jHttpClient

jHttpClient1.SetURL(GEMINI\_URL);

// Tambahkan header yang wajib

jHttpClient1.RequestProperty.Clear; // Bersihkan header sebelumnya

jHttpClient1.RequestProperty.Add('x-goog-api-key', GEMINI\_API\_KEY);

jHttpClient1.RequestProperty.Add('Content-Type', 'application/json');

// Eksekusi sebagai permintaan POST dengan body JSON

jHttpClient1.ExecutePost(jsonBody);

end;

**Lahkah 3: Mem-parse Respons dan Menampilkan Hasil**

Respons dari Gemini juga dalam format JSON. Kita perlu mem-parse-nya di dalam OnHttpRequestCompleted untuk mendapatkan jawaban dari AI. Struktur responsnya kurang lebih seperti ini: {"candidates":[{"content":{"parts":[{"text":"...jawaban AI ada di sini..."}]}}]}.

Delphi

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject;

responseCode: integer; responseText: string);

var

jsonValue: TJSONData;

jsonObj, jsonContent, jsonPart: TJSONObject;

jsonCandidates, jsonParts: TJSONArray;

hasilTeks: string;

begin

jProgressBar1.Visible := False;

btnKirimPrompt.Enabled := True;

if responseCode = 200 then

begin

hasilTeks := '(Gagal mem-parse respons)';

try

jsonValue := GetJSON(responseText);

jsonObj := TJSONObject(jsonValue);

try

// Navigasi melalui struktur JSON untuk mendapatkan teks jawaban

jsonCandidates := TJSONArray(jsonObj.Get('candidates'));

if jsonCandidates.Count > 0 then

begin

jsonContent := TJSONObject(TJSONObject(jsonCandidates.Items[0]).Get('content'));

jsonParts := TJSONArray(jsonContent.Get('parts'));

if jsonParts.Count > 0 then

begin

jsonPart := TJSONObject(jsonParts.Items[0]);

hasilTeks := jsonPart.Get('text').AsString;

end;

end;

finally

jsonObj.Free;

end;

except

// Jika terjadi error saat parsing, biarkan hasilTeks berisi pesan error

end;

tvHasilGemini.Text := hasilTeks;

end

else

tvHasilGemini.Text := 'Error: ' + responseText;

end;

Sekarang jalankan aplikasi Anda, ketik sebuah pertanyaan seperti "Apa ibu kota provinsi Aceh?", tekan kirim, dan saksikan Gemini memberikan jawabannya langsung di aplikasi Anda!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 14.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 14.3: Kemampuan Multimodal: Mengirim Teks dan Gambar**.

**14.3. Kemampuan Multimodal: Mengirim Teks dan Gambar**

Kekuatan sejati Gemini terletak pada kemampuan **multimodal**-nya—kemampuan untuk memahami lebih dari sekadar teks. Di subbab ini, kita akan membuat aplikasi yang bisa 'melihat'. Kita akan memungkinkan pengguna untuk memilih sebuah gambar dari galeri mereka, mengajukan pertanyaan tentang gambar tersebut, dan mendapatkan jawaban cerdas dari Gemini.

Proyek ini akan menggabungkan keahlian kita dari **Bab 8** (memilih gambar dari galeri) dan **Bab 7** (*networking*/JSON).

**Persiapan UI dan Alur Kerja:**

* **UI:** Siapkan jImageView (imgPilihan) untuk menampilkan gambar, jButton (btnPilihGambar), jEditText (edtPromptGambar) untuk pertanyaan, jButton (btnKirimMulti), dan jTextView (tvHasilMulti) untuk jawaban.
* **Alur Kerja:** Pengguna menekan btnPilihGambar -> Galeri terbuka -> Pengguna memilih gambar -> Gambar ditampilkan di imgPilihan -> Pengguna mengetik pertanyaan -> Pengguna menekan btnKirimMulti -> Aplikasi mengirim gambar dan teks ke Gemini -> Hasil ditampilkan.

**Langkah 1: Memilih Gambar dan Mengubahnya ke Base64**

Saat kita mengirim data gambar melalui API web, kita tidak bisa mengirim file gambar secara langsung. Format yang paling umum digunakan adalah dengan meng-*encode* data biner dari gambar tersebut menjadi sebuah *string* teks panjang menggunakan pengkodean **Base64**.

1. **Memilih Gambar:** Gunakan pola Intent ACTION\_GET\_CONTENT dan OnActivityResult dari Bab 8.3 untuk membiarkan pengguna memilih gambar. Setelah gambar dipilih, kita akan mendapatkan sebuah Uri. Dari Uri ini, kita bisa membuat sebuah jBitmap.
2. **Bitmap ke String Base64:** Setelah mendapatkan objek jBitmap, kita perlu mengubahnya menjadi *string* Base64. Buatlah sebuah fungsi pembantu untuk ini.

Delphi

// Jangan lupa tambahkan Androidapi.Util, Androidapi.IO, Androidapi.Graphics ke uses clause

function BitmapToBase64(aBitmap: jBitmap): string;

var

outputStream: jByteArrayOutputStream;

byteArray: TJavaObjectArray<byte>;

encodedString: jString;

begin

Result := '';

if not Assigned(aBitmap) then Exit;

outputStream := tjByteArrayOutputStream.Create;

try

// 1. Kompres bitmap menjadi format JPEG ke dalam memory stream

aBitmap.compress(tjBitmap\_CompressFormat.JPEG, 90, outputStream); // 90 adalah kualitas kompresi

// 2. Dapatkan byte array dari stream

byteArray := outputStream.toByteArray;

// 3. Encode byte array menjadi string Base64

encodedString := tjBase64.EncodeToString(byteArray, tjBase64.DEFAULT);

Result := JStringToString(encodedString);

finally

outputStream.close;

outputStream.Free;

end;

end;

Di dalam OnActivityResult, setelah Anda berhasil memuat Uri menjadi Bitmap dan menampilkannya, panggil fungsi ini dan simpan hasilnya ke sebuah variabel global di *form* Anda, misalnya FBase64Image: string;.

**Langkah 2: Membuat Body JSON Multimodal**

Struktur JSON untuk permintaan multimodal sedikit lebih kompleks. Ia berisi beberapa "parts" di dalam *array* contents: satu *part* untuk teks *prompt*, dan satu *part* lagi untuk data gambar.

JSON

{

"contents": [{

"parts": [

{"text": "Jelaskan apa yang unik dari bangunan di gambar ini?"},

{"inline\_data": {

"mime\_type": "image/jpeg",

"data": "....string base64 yang sangat panjang ada di sini...."

}

}

]

}]

}

Anda bisa membuat fungsi baru CreateMultiModalPromptJSON yang menerima parameter *prompt* teks dan *string* Base64 untuk menghasilkan JSON dengan struktur di atas. Proses pembuatannya mirip dengan CreateTextPromptJSON, hanya saja Anda menambahkan dua objek ke dalam *array* parts.

**Langkah 3: Mengirim Permintaan dan Mem-parse Hasil**

Proses ini hampir sama persis dengan subbab sebelumnya.

1. **Mengirim Permintaan:** Di OnClick tombol btnKirimMulti, panggil fungsi CreateMultiModalPromptJSON dengan teks dari edtPromptGambar dan FBase64Image yang sudah Anda simpan. Lalu kirimkan *string* JSON hasilnya menggunakan jHttpClient.ExecutePost ke *endpoint* khusus untuk model visi, yaitu: https://generativelanguage.googleapis.com/v1beta/models/gemini-pro-vision:generateContent.
2. **Mem-parse Hasil:** Struktur respons dari Gemini untuk permintaan multimodal sama dengan respons teks-ke-teks. Jadi, Anda bisa menggunakan **kode *parsing* yang sama persis** dari subbab 14.2 di dalam OnHttpRequestCompleted untuk mengambil dan menampilkan jawabannya di tvHasilMulti.

Jalankan aplikasi Anda. Pilih sebuah foto—misalnya foto Jembatan Ampera atau Candi Borobudur—lalu ketik pertanyaan "Deskripsikan tempat ini". Anda akan takjub melihat bagaimana Gemini bisa 'melihat' dan memahami konten gambar yang Anda berikan. Kemampuan ini membuka pintu untuk aplikasi yang jauh lebih interaktif dan cerdas.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 14.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 14.4: Membangun Chatbot Sederhana**.

**14.4. Membangun Chatbot Sederhana**

Sejauh ini, setiap permintaan kita ke Gemini bersifat 'stateless' atau 'tanpa memori'. Gemini tidak mengingat pertanyaan kita sebelumnya. Untuk menciptakan pengalaman *chatbot* yang alami, di mana kita bisa bertanya "Kenapa begitu?" dan AI mengerti apa yang kita maksud, kita perlu membuatnya bisa 'mengingat' percakapan. Di subbab ini, kita akan belajar cara mengelola riwayat percakapan untuk membangun sebuah *chatbot* dasar.

**Konsep Kunci: Mengelola Riwayat Percakapan**

Kunci dari sebuah percakapan dengan AI adalah **konteks**. Saat kita mengirim pesan baru, kita tidak hanya mengirim pesan itu saja. Sebaliknya, kita mengirim **seluruh riwayat percakapan** dari awal hingga pesan terbaru. Dengan begitu, Gemini bisa memahami konteks dan memberikan jawaban yang relevan.

Setiap bagian dari percakapan harus diberi label peran (*role*): **user** untuk pesan dari kita, dan **model** untuk jawaban dari Gemini.

**Langkah 1: Menyiapkan Struktur Data dan UI**

1. **Struktur Data:** Kita perlu sebuah tempat untuk menyimpan riwayat percakapan secara terstruktur di aplikasi kita. Sebuah TList yang berisi *pointer* ke sebuah record adalah pilihan yang sangat baik. Definisikan *record* baru di bagian type:

Delphi

type

PChatMessage = ^TChatMessage;

TChatMessage = record

Role: string; // isinya 'user' atau 'model'

Text: string;

end;

Kemudian, di bagian private *form* Anda, deklarasikan sebuah variabel untuk menampung daftar ini: FChatHistory: TList;. Jangan lupa untuk membuatnya di *event* OnCreate (FChatHistory := TList.Create;).

1. **Desain UI:** UI untuk *chat* biasanya terdiri dari:
   * Sebuah jListView (lvChat) untuk menampilkan daftar percakapan.
   * Sebuah jLinearLayout horizontal di bagian bawah yang berisi jEditText (edtChatInput) dan jButton (btnKirimChat).

**Langkah 2: Membuat Body JSON untuk Chat**

Struktur JSON untuk *chat* adalah sebuah *array* contents, di mana setiap elemennya adalah sebuah objek yang memiliki role dan parts.

JSON

{

"contents": [

{ "role": "user", "parts": [{"text": "Halo, siapa namamu?"}] },

{ "role": "model", "parts": [{"text": "Saya adalah model bahasa besar..."}] },

{ "role": "user", "parts": [{"text": "Apa yang bisa kamu lakukan?"}] }

]

}

Kita akan membuat sebuah fungsi yang mengambil FChatHistory kita dan mengubahnya menjadi *string* JSON dengan format seperti di atas sebelum mengirimkannya ke API.

**Langkah 3: Alur Kerja Chat**

Berikut adalah alur logika yang terjadi saat pengguna menekan tombol 'Kirim':

1. **Ambil Pesan Pengguna:** Dapatkan teks dari edtChatInput.
2. **Tambahkan Pesan ke Riwayat:** Buat sebuah *record* TChatMessage baru dengan Role = 'user' dan Text = pesan pengguna. Tambahkan *pointer* ke *record* ini ke dalam FChatHistory. Kosongkan edtChatInput.
3. **Perbarui UI:** Tampilkan seluruh isi FChatHistory di jListView. Idealnya, gunakan *adapter* kustom untuk membuat tampilan gelembung *chat* yang berbeda untuk peran 'user' dan 'model'.
4. **Buat JSON dan Kirim Request:** Panggil fungsi pembuat JSON Anda untuk mengubah FChatHistory menjadi *string* JSON. Kirimkan *string* ini ke Gemini API menggunakan jHttpClient.ExecutePost.
5. **Tangani Respons:** Di OnHttpRequestCompleted, *parse* respons JSON dari Gemini untuk mendapatkan teks jawaban.
6. **Tambahkan Respons ke Riwayat:** Buat sebuah *record* TChatMessage baru dengan Role = 'model' dan Text = jawaban dari Gemini. Tambahkan *pointer*-nya ke FChatHistory.
7. **Perbarui UI Lagi:** Panggil kembali prosedur untuk menampilkan FChatHistory di jListView. Pengguna sekarang akan melihat jawaban dari Gemini muncul di bawah pertanyaan mereka.

**Contoh Kode Parsial (OnClick Tombol Kirim)**

Delphi

procedure TAndroidModule1.btnKirimChatClick(Sender: TObject);

var

userMessage: string;

pMsg: PChatMessage;

jsonBody: string;

begin

userMessage := edtChatInput.Text;

if userMessage = '' then Exit;

// Langkah 1 & 2

New(pMsg);

pMsg^.Role := 'user';

pMsg^.Text := userMessage;

FChatHistory.Add(pMsg);

edtChatInput.Text := '';

// Langkah 3

RefreshChatListView; // Prosedur kustom Anda untuk update ListView

// Langkah 4

jsonBody := CreateChatJSONFromHistory(FChatHistory); // Fungsi buatan Anda

// ... (kode untuk mengirim request dengan jHttpClient) ...

end;

Dengan mengikuti alur ini, Anda dapat membangun fungsionalitas *chatbot* yang cerdas dan mampu mengikuti alur percakapan dengan pengguna.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 14.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 14.5: Etika dan Praktik Terbaik**.

**14.5. Etika dan Praktik Terbaik**

Mengintegrasikan AI yang kuat seperti Gemini ke dalam aplikasi kita tidak hanya membawa kemampuan luar biasa, tetapi juga tanggung jawab. Sebagai developer, kita perlu memahami cara menggunakan alat ini secara efektif, aman, dan etis untuk menciptakan pengalaman yang positif bagi pengguna. Subbab ini membahas beberapa praktik terbaik yang harus Anda ingat.

**1. Dasar-dasar Rekayasa Prompt (Prompt Engineering)**

Kualitas jawaban dari Gemini sangat bergantung pada kualitas pertanyaan (*prompt*) yang Anda berikan. Prinsip 'sampah masuk, sampah keluar' juga berlaku di sini. Untuk mendapatkan hasil yang lebih baik, pertimbangkan tips berikut:

* **Berikan Konteks yang Jelas:** Jangan hanya bertanya, "Apa itu *polymorphism*?". Sebaliknya, berikan konteks: "Dalam konteks pemrograman berorientasi objek di Pascal, jelaskan apa itu *polymorphism* dengan contoh sederhana."
* **Tentukan Peran (Persona):** Awali *prompt* dengan "Berperanlah sebagai seorang guru yang ramah..." untuk mendapatkan gaya jawaban yang lebih edukatif, atau "Berperanlah sebagai seorang penulis teknis..." untuk jawaban yang lebih formal.
* **Minta Format Tertentu:** Anda bisa meminta jawaban dalam format spesifik, seperti "Jelaskan dalam tiga poin utama", "Buat sebuah tabel perbandingan", atau "Buat sebuah contoh kode".
* **Berikan Contoh (*Few-shot Prompting*):** Untuk tugas yang lebih kompleks, berikan satu atau dua contoh input dan output yang Anda inginkan, lalu berikan input Anda yang sebenarnya. Ini membantu AI memahami pola yang Anda cari.

**2. Memahami Pengaturan Keamanan (Safety Settings)**

Secara *default*, Gemini API memiliki filter keamanan yang sudah terpasang untuk memblokir konten yang tidak pantas seperti ujaran kebencian, pelecehan, atau konten berbahaya lainnya. Anda tidak perlu melakukan apa pun untuk mengaktifkannya. Penting untuk dipahami bahwa jika *prompt* Anda atau hasil dari AI memicu filter ini, API akan mengembalikan respons kosong atau sebuah blok. Aplikasi Anda harus bisa menangani kasus ini dengan baik, misalnya dengan menampilkan pesan, "Maaf, saya tidak bisa memproses permintaan itu."

**3. Transparansi dan Penggunaan yang Bertanggung Jawab**

* **Selalu Transparan:** Beri tahu pengguna Anda dengan jelas bahwa mereka sedang berinteraksi dengan sistem AI, bukan manusia. Ini membangun kepercayaan dan mengelola ekspektasi mereka.
* **Hindari untuk Keputusan Kritis:** Jangan gunakan AI generatif untuk membuat keputusan penting di bidang-bidang sensitif seperti diagnosis medis, nasihat hukum, atau keputusan keuangan tanpa adanya tinjauan dan verifikasi dari ahli manusia.
* **Verifikasi Fakta:** Ingatlah bahwa model AI terkadang bisa 'berhalusinasi' atau memberikan informasi yang terdengar meyakinkan tetapi sebenarnya salah. Jika aplikasi Anda menyajikan data faktual yang dihasilkan AI, selalu pertimbangkan untuk memiliki mekanisme verifikasi atau setidaknya memberikan peringatan kepada pengguna.

**4. Manajemen Biaya dan Kuota Penggunaan**

Ingatlah bahwa penggunaan API untuk model-model canggih seperti Gemini **biasanya tidak gratis** setelah melewati kuota penggunaan awal. Setiap permintaan yang Anda kirim ke API akan dihitung.

* **Pahami Batas Penggunaan (*Rate Limiting*):** Sebagian besar API memiliki batas jumlah permintaan yang bisa Anda kirim per menit. Aplikasi Anda harus bisa menangani *error* 429 Too Many Requests dengan cara, misalnya, menunggu sejenak sebelum mencoba lagi.
* **Pantau Penggunaan Anda:** Selalu periksa dasbor Google AI Studio atau Google Cloud Console Anda untuk memantau penggunaan dan tagihan agar tidak ada kejutan di akhir bulan.

Menggunakan AI adalah sebuah perjalanan yang menarik. Dengan menggabungkan kekuatan teknis dan kesadaran etis, Anda dapat membangun aplikasi yang tidak hanya cerdas, tetapi juga aman dan bermanfaat bagi semua orang.

Ini mengakhiri Bab 14. Anda kini memiliki kemampuan untuk mengintegrasikan kecerdasan buatan ke dalam aplikasi Anda. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 15 (Pengantar Bab 15).

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Pengantar Bab 15**.

**Bab 15: Menjelajahi Dunia API Publik: Studi Kasus Lanjutan**

Selamat datang di bab petualangan! Jika di bab-bab sebelumnya kita telah belajar cara membuat 'kapal' (aplikasi kita) dan cara 'membaca peta' (memahami cara kerja API), maka di bab ini kita akan benar-benar berlayar. Internet adalah sebuah samudra data yang tak terbatas, dan **API Publik** adalah pelabuhan-pelabuhan eksotis yang bisa kita kunjungi untuk mengambil 'barang dagangan' berupa informasi yang berharga.

Tujuan bab ini bukanlah untuk mempelajari satu teknologi baru yang besar, melainkan untuk mengasah keahlian Anda yang paling penting sebagai seorang developer: **kemampuan beradaptasi**. Setiap 'pelabuhan' API di dunia nyata memiliki aturan dan 'bahasa' (struktur data) yang sedikit berbeda. Ada yang meminta 'paspor' (kunci API) di depan gerbang, ada yang memintanya di setiap dermaga. Ada yang memberikan barang dalam kotak-kotak kecil yang rapi, ada yang memberikannya dalam kontainer besar yang isinya perlu kita pilah-pilah.

Keahlian teknis Anda untuk melakukan permintaan HTTP dan mem-parse JSON (dari Bab 7) tetap sama. Tantangan sebenarnya adalah **membaca dokumentasi API**, memahami struktur data yang unik, dan mengadaptasi kode Anda untuk menanganinya. Inilah yang akan kita latih di sini.

Untuk melatih kemampuan adaptasi ini, kita akan melakukan 'tur' dengan membangun tiga aplikasi mini yang sangat berbeda, masing-masing mengunjungi 'pelabuhan' API yang berbeda:

* **Aplikasi Cuaca:** Kita akan belajar mengambil data cuaca saat ini, menghadapi tantangan mem-parse JSON yang bersarang (*nested*) dan memetakan kode ikon ke gambar.
* **Konverter Mata Uang:** Sebuah aplikasi utilitas di mana kita akan mengambil data kurs yang relatif sederhana, namun fokus pada logika kalkulasi dan interaksi pengguna di sisi aplikasi.
* **Peramban Film:** Proyek yang lebih kompleks di mana kita akan belajar tentang otentikasi melalui *header*, menangani daftar data yang panjang (*pagination*), dan menghubungkan beberapa *endpoint* API untuk mendapatkan daftar dan detail film.

Setiap studi kasus akan memberikan Anda wawasan dan kepercayaan diri baru. Setelah menyelesaikan bab ini, Anda akan merasa siap untuk menjelajahi API publik apa pun yang Anda temukan di internet. Mari kita angkat sauh dan mulai pelayaran kita!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 15.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai studi kasus pertama.

Berikut adalah jawaban untuk prompt **Subbab 15.1: Studi Kasus 1: Aplikasi Cuaca dengan OpenWeatherMap API**.

**15.1. Studi Kasus 1: Aplikasi Cuaca dengan OpenWeatherMap API**

Studi kasus pertama kita adalah salah satu jenis aplikasi yang paling umum dan berguna: aplikasi cuaca. Kita akan menggunakan layanan dari **OpenWeatherMap** untuk mendapatkan data cuaca *real-time* untuk kota manapun di dunia. Proyek ini akan melatih kita dalam tiga hal: proses registrasi API, menangani otentikasi sederhana, dan yang terpenting, mem-parse struktur JSON yang realistis dan bersarang.

**Langkah 1: Persiapan - Mendapatkan Kunci API**

1. Buka browser Anda, kunjungi <https://openweathermap.org/> dan buat akun baru (pilih opsi *Sign Up*).
2. Setelah login, klik pada nama akun Anda di pojok kanan atas, lalu pilih **'My API keys'**.
3. Anda akan melihat sebuah kunci *default* sudah dibuatkan untuk Anda. **Salin (copy) 'Key'** tersebut dan simpan di tempat aman. Kunci ini kita sebut **APPID**.

**Langkah 2: Merancang UI dan Endpoint API**

* **UI:** Buat UI sederhana yang terdiri dari: jEditText (edtNamaKota), jButton (btnCariCuaca), dan beberapa jTextView untuk hasilnya (tvHasilKota, tvHasilSuhu, tvHasilDeskripsi) serta sebuah jImageView (imgCuaca).
* **Endpoint:** Berdasarkan dokumentasi OpenWeatherMap, *endpoint* untuk cuaca saat ini adalah: https://api.openweathermap.org/data/2.5/weather. Ia memerlukan beberapa parameter, terutama q (nama kota) dan appid (kunci API kita).

URL lengkapnya akan terlihat seperti ini: https://api.openweathermap.org/data/2.5/weather?q=Jakarta&units=metric&appid=KUNCI\_API\_ANDA

Perhatikan: units=metric adalah parameter tambahan yang sangat berguna agar suhu dikembalikan dalam satuan **Celcius**. Cara menyertakan kunci API di dalam URL seperti ini adalah contoh **otentikasi via query parameter**.

**Langkah 3: Implementasi Request dan Parsing**

Kita akan meletakkan logika permintaan di OnClick dan logika penanganan respons di OnHttpRequestCompleted.

**Kode OnClick Tombol:**

Delphi

const

// Simpan kunci API Anda sebagai konstanta

OWM\_API\_KEY = 'MASUKKAN\_KUNCI\_API\_ANDA\_DI\_SINI';

procedure TAndroidModule1.btnCariCuacaClick(Sender: TObject);

var

kota, url: string;

begin

kota := edtNamaKota.Text;

if kota = '' then

begin

ShowMessage('Silakan masukkan nama kota.');

Exit;

end;

// Tampilkan loading bar, dll...

url := 'https://api.openweathermap.org/data/2.5/weather?q=' + kota +

'&units=metric&lang=id&appid=' + OWM\_API\_KEY; // lang=id untuk deskripsi bahasa Indonesia

jHttpClient1.SetURL(url);

jHttpClient1.Execute;

end;

**Kode OnHttpRequestCompleted (Parsing JSON Bersarang):** Respons JSON dari OpenWeatherMap cukup kompleks. description dan icon ada di dalam *array* weather, dan temp ada di dalam objek main. Kita harus menavigasi struktur ini.

Delphi

procedure TAndroidModule1.jHttpClient1HttpRequestCompleted(Sender: TObject; responseCode: integer; responseText: string);

var

jsonObj, jsonMainObj, jsonWeatherObj: TJSONObject;

jsonWeatherArray: TJSONArray;

deskripsi, ikon, kota: string;

suhu: double;

begin

// ... (sembunyikan loading bar, dll) ...

if responseCode = 200 then

begin

try

jsonObj := TJSONObject(GetJSON(responseText));

try

// Ambil data dari level atas

kota := jsonObj.Get('name').AsString;

// Masuk ke objek 'main' untuk mengambil suhu

jsonMainObj := TJSONObject(jsonObj.Get('main'));

suhu := jsonMainObj.Get('temp').AsFloat;

// Masuk ke array 'weather' untuk mengambil deskripsi dan ikon

jsonWeatherArray := TJSONArray(jsonObj.Get('weather'));

if jsonWeatherArray.Count > 0 then

begin

jsonWeatherObj := TJSONObject(jsonWeatherArray.Items[0]); // Ambil item pertama dari array

deskripsi := jsonWeatherObj.Get('description').AsString;

ikon := jsonWeatherObj.Get('icon').AsString;

// Panggil prosedur untuk menampilkan ikon

TampilkanIkonCuaca(ikon);

end;

// Tampilkan semua data ke UI

tvHasilKota.Text := kota;

tvHasilSuhu.Text := FormatFloat('0.0', suhu) + ' °C';

tvHasilDeskripsi.Text := deskripsi;

finally

jsonObj.Free;

end;

except

on E: Exception do ShowMessage('Gagal memproses data: ' + E.Message);

end;

end

else

ShowMessage('Gagal mengambil data cuaca. Error: ' + IntToStr(responseCode));

end;

**Langkah 4: Pemetaan Kode Ikon ke Gambar**

API tidak memberikan URL gambar, hanya sebuah kode ikon seperti 01d (cerah siang), 01n (cerah malam), 10d (hujan siang), dst. Kita perlu memetakannya ke gambar yang sudah kita siapkan di res/drawable.

* **Persiapan:** Unduh satu set ikon cuaca dari internet (misalnya dari situs OpenWeatherMap sendiri) dan beri nama file sesuai kodenya (misal: icon\_01d.png, icon\_10n.png, dll.) lalu letakkan di folder drawable.
* **Kode Pemetaan:**

Delphi

procedure TAndroidModule1.TampilkanIkonCuaca(const kodeIkon: string);

var

resID: integer;

begin

// Dapatkan ID resource drawable berdasarkan namanya

resID := GetDrawableResourceId('icon\_' + kodeIkon);

if resID <> 0 then

imgCuaca.SetImageResource(resID)

else

imgCuaca.SetImageBitmap(nil); // Kosongkan jika ikon tidak ditemukan

end;

Dengan ini, aplikasi cuaca sederhana Anda telah selesai, mampu menangani otentikasi, *parsing* JSON kompleks, dan pemetaan data.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 15.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke studi kasus kedua.

Berikut adalah jawaban untuk prompt **Subbab 15.2: Studi Kasus 2: Konverter Mata Uang dengan ExchangeRate-API**.

**15.2. Studi Kasus 2: Konverter Mata Uang dengan ExchangeRate-API**

Studi kasus kedua kita adalah aplikasi utilitas yang sangat umum: Konverter Mata Uang. Berbeda dengan aplikasi cuaca yang fokus pada menampilkan data, proyek ini akan fokus pada **interaksi pengguna dan logika kalkulasi**. API di sini berperan sebagai penyedia data pendukung yang krusial, tetapi 'otak' aplikasi sebenarnya ada di sisi klien (aplikasi Pascal kita).

Kita akan menggunakan layanan dari **ExchangeRate-API.com**, yang menyediakan API yang sangat sederhana dan memiliki paket gratis yang memadai.

**Langkah 1: Persiapan - Mendapatkan Kunci API**

1. Kunjungi situs web <https://www.exchangerate-api.com/>.
2. Daftar untuk mendapatkan kunci API gratis dengan mengklik 'Get Free Key'.
3. Setelah mendaftar dan login, salin Kunci API Anda dari dasbor akun Anda.

**Langkah 2: Merancang UI dan Memahami Endpoint**

* **UI:** Buat UI yang fungsional, terdiri dari:
  + jEditText (edtJumlah) untuk memasukkan jumlah uang yang akan dikonversi, atur InputType-nya menjadi itNumberDecimal.
  + Dua jSpinner (spnDariMataUang dan spnKeMataUang) untuk memilih mata uang.
  + jButton (btnKonversi) dengan teks 'Konversi'.
  + jTextView (tvHasilKonversi) berukuran besar untuk menampilkan hasilnya.
* **Endpoint:** *Endpoint* untuk mendapatkan kurs terbaru relatif terhadap satu mata uang dasar (misalnya, USD) adalah: https://v6.exchangerate-api.com/v6/KUNCI\_API\_ANDA/latest/USD. Responsnya akan berupa objek JSON datar yang sangat mudah di-parse, contohnya:

JSON

{

"result": "success",

"base\_code": "USD",

"conversion\_rates": {

"USD": 1,

"EUR": 0.92,

"JPY": 157.5,

"IDR": 16250.7,

...

}

}

Perhatikan betapa datarnya objek conversion\_rates. Ini akan mudah di-parse.

**Langkah 3: Implementasi Logika Aplikasi**

**A. Mengisi Spinner saat Aplikasi Dimuat** Praktik yang baik adalah mengambil data kurs sekali saat aplikasi dimuat, lalu menyimpannya dalam sebuah variabel untuk digunakan dalam kalkulasi.

1. Di bagian private *form* Anda, deklarasikan: FRatesObject: TJSONObject;
2. Di OnCreate form, lakukan permintaan GET ke *endpoint* di atas.
3. Di OnHttpRequestCompleted, *parse* responsnya:

Delphi

// Di dalam OnHttpRequestCompleted, setelah cek responseCode = 200

// ...

jsonObj := TJSONObject(GetJSON(responseText));

// Simpan objek rates ke variabel global

FRatesObject := TJSONObject(jsonObj.Get('conversion\_rates'));

// Isi spinner dengan semua nama kunci (kode mata uang)

spnDariMataUang.Clear;

spnKeMataUang.Clear;

for i := 0 to FRatesObject.Count - 1 do

begin

spnDariMataUang.Items.Add(FRatesObject.Names[i]);

spnKeMataUang.Items.Add(FRatesObject.Names[i]);

end;

// Atur pilihan default

spnDariMataUang.SelectedIndex := spnDariMataUang.Items.IndexOf('USD');

spnKeMataUang.SelectedIndex := spnKeMataUang.Items.IndexOf('IDR');

// Jangan bebaskan FRatesObject karena akan kita gunakan nanti!

// Cukup bebaskan objek induknya

jsonObj.Remove('conversion\_rates'); // Lepaskan referensi agar tidak ikut ter-free

jsonObj.Free;

**B. Melakukan Konversi saat Tombol Ditekan** Inti logika terjadi di OnClick tombol btnKonversi. API memberikan kita semua kurs relatif terhadap satu mata uang dasar (USD). Jika kita ingin mengonversi dari EUR ke IDR, kita perlu menggunakan USD sebagai perantara.

**Rumusnya:** Jumlah Akhir = (Jumlah Awal / Kurs\_Awal\_thd\_USD) \* Kurs\_Tujuan\_thd\_USD

Delphi

procedure TAndroidModule1.btnKonversiClick(Sender: TObject);

var

jumlahAwal, nilaiTukarDari, nilaiTukarKe, hasil: double;

mataUangDari, mataUangKe: string;

begin

if not Assigned(FRatesObject) then

begin

ShowMessage('Data kurs belum selesai dimuat. Silakan coba sesaat lagi.');

Exit;

end;

// 1. Ambil input dari pengguna

jumlahAwal := StrToFloatDef(edtJumlah.Text, 0);

mataUangDari := spnDariMataUang.GetSelectedItem.ToString;

mataUangKe := spnKeMataUang.GetSelectedItem.ToString;

// 2. Ambil nilai tukar dari objek JSON yang sudah disimpan

nilaiTukarDari := FRatesObject.Get(mataUangDari).AsFloat;

nilaiTukarKe := FRatesObject.Get(mataUangKe).AsFloat;

// 3. Lakukan kalkulasi (dengan USD sebagai basis)

if nilaiTukarDari > 0 then

hasil := (jumlahAwal / nilaiTukarDari) \* nilaiTukarKe

else

hasil := 0;

// 4. Tampilkan hasil dengan format yang baik

tvHasilKonversi.Text := FormatFloat('#,##0.00', hasil) + ' ' + mataUangKe;

end;

**Langkah 4: Hasil Akhir**

Aplikasi konverter mata uang Anda kini telah selesai. Proyek ini mengajarkan kita bahwa terkadang API hanya berfungsi sebagai penyedia data mentah. Tugas kita sebagai developer adalah merancang antarmuka yang baik dan menerapkan logika bisnis atau kalkulasi di sisi klien untuk mengubah data mentah tersebut menjadi informasi yang berguna bagi pengguna.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 15.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan ke studi kasus terakhir di bab ini.

Berikut adalah jawaban untuk prompt **Subbab 15.3: Studi Kasus 3: Peramban Film dengan TMDB API**.

**15.3. Studi Kasus 3: Peramban Film dengan TMDB API**

Studi kasus terakhir kita adalah yang paling menantang dan paling mendekati aplikasi dunia nyata seperti Netflix atau IMDb. Kita akan membangun sebuah **Peramban Film Sederhana** menggunakan data dari **The Movie Database (TMDB)**. Proyek ini akan mengajarkan kita beberapa konsep yang sangat penting: otentikasi melalui *header*, menangani daftar data yang sangat panjang (*pagination*), dan menghubungkan beberapa *endpoint* API untuk menciptakan alur 'Daftar -> Detail'.

**Langkah 1: Persiapan - Mendapatkan Kunci API dari TMDB**

1. Kunjungi <https://www.themoviedb.org/>, buat akun, dan lakukan verifikasi email Anda.
2. Setelah login, klik ikon profil Anda, lalu masuk ke Settings > API.
3. Ajukan permohonan untuk kunci API (pilih tipe 'Developer'). Isi formulir yang dibutuhkan.
4. Setelah disetujui, Anda akan mendapatkan sebuah **'API Read Access Token (v4 auth)'**. Ini adalah *Bearer Token* Anda. Salin dan simpan baik-baik.

**Langkah 2: Otentikasi via Header dan Permintaan Awal**

* **Tantangan 1: Otentikasi Header.** Berbeda dengan API sebelumnya, TMDB menggunakan metode otentikasi yang lebih aman. Kunci API tidak diletakkan di URL, melainkan di dalam **HTTP Header** Authorization.
* **Endpoint & Gambar:**
  + *Endpoint* untuk film populer: https://api.themoviedb.org/3/movie/popular.
  + TMDB hanya memberikan *path* gambar poster (misal: /sM33v\_4SHI0iA5j4GzG24dJ4Rk.jpg). Kita harus menggabungkannya dengan URL dasar gambar: https://image.tmdb.org/t/p/w500.
* **Kode Permintaan:**

Delphi

const

TMDB\_TOKEN = 'PASTE\_BEARER\_TOKEN\_ANDA\_DI\_SINI';

procedure TAndroidModule1.FetchMovies(page: integer);

begin

// ... (tampilkan loading bar) ...

jHttpClient1.RequestProperty.Clear;

// Tambahkan header otentikasi

jHttpClient1.RequestProperty.Add('Authorization', 'Bearer ' + TMDB\_TOKEN);

jHttpClient1.RequestProperty.Add('Content-Type', 'application/json;charset=utf-8');

jHttpClient1.SetURL('https://api.themoviedb.org/3/movie/popular?page=' + IntToStr(page));

jHttpClient1.Execute;

end;

* **Tantangan 2: Pagination.** Perhatikan parameter ?page= di URL. Ini memungkinkan kita untuk mengambil halaman data yang berbeda (halaman 1, 2, 3, dst.). Untuk membuat fitur 'muat lebih banyak' atau *infinite scroll*, kita hanya perlu menambah nomor halaman ini dan melakukan *request* baru saat pengguna mencapai akhir daftar.

**Langkah 3: Menampilkan Daftar Film**

Di OnHttpRequestCompleted, *parse* JSON dari TMDB. Data film ada di dalam *array* pada kunci 'results'. Buat sebuah record TMovie dan *adapter* kustom TMovieAdapter. Di dalam metode GetView adapter Anda, pastikan Anda menggabungkan URL dasar gambar dengan *path* poster untuk memuatnya ke jImageView: fullUrl := 'https://image.tmdb.org/t/p/w500' + movie.posterPath;.

**Langkah 4: Menghubungkan ke Layar Detail (Multi-Endpoint)**

Ini adalah inti dari aplikasi yang kompleks: menghubungkan beberapa *endpoint*.

1. **Buat Form Detail:** Buat sebuah jForm baru, uDetailFilm.pas, yang berisi komponen-komponen untuk menampilkan judul, sinopsis (jTextView *multiline*), dan gambar poster. Daftarkan di file .lpr.
2. **Implementasi OnItemClick di Form Utama:**

Delphi

procedure TAndroidModule1.lvMoviesItemClick(Sender: TObject; position: integer);

var

pMovie: PMovie; // Pointer ke record TMovie Anda

intent: jIntent;

begin

// Dapatkan data film yang diklik dari adapter

pMovie := PMovie(FMovieAdapter.GetItem(position));

// Kirim HANYA ID film ke layar detail

intent := jIntent.Create;

intent.SetAction(jIntent\_ACTION\_VIEW);

intent.PutExtra('movie\_id', pMovie^.id); // Kirim ID-nya saja

jDetailFilm.Show(intent);

end;

1. **Ambil Data Detail di Layar Kedua:** Di *event* OnCreate pada uDetailFilm.pas, lakukan hal berikut:
   * Ambil movie\_id dari Intent yang diterima.
   * Gunakan jHttpClient lagi untuk melakukan **permintaan API baru** ke *endpoint* detail: https://api.themoviedb.org/3/movie/{movie\_id} (ganti {movie\_id} dengan ID yang Anda dapat). Jangan lupa sertakan *header* otentikasi yang sama.
   * *Parse* responsnya (yang sekarang berupa satu objek film, bukan *array*) untuk mendapatkan data detail seperti overview (sinopsis), release\_date, dll.
   * Tampilkan semua data detail tersebut di komponen-komponen UI di layar detail.

**Hasil Akhir**

Dengan menyelesaikan proyek ini, Anda telah berhasil membangun aplikasi yang meniru alur kerja aplikasi media modern. Anda mengambil daftar data, menampilkannya, dan saat pengguna berinteraksi, Anda mengambil data yang lebih spesifik dari *endpoint* lain. Keahlian ini sangat fundamental dan dapat diterapkan di berbagai jenis aplikasi yang kompleks.

Ini mengakhiri Bab 15. Anda sekarang memiliki pengalaman praktis dalam menghadapi berbagai jenis API. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 16 (Pengantar Bab 16).

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Pengantar Bab 16**.

**Bab 16: Menguasai Multimedia: Audio dan Video di Aplikasi Anda**

Selamat datang di Bab 16, di mana kita akan menambahkan dimensi baru ke dalam aplikasi kita: **suara dan gerakan**. Aplikasi yang hebat tidak hanya terlihat bagus, tetapi juga terdengar memuaskan dan bisa menyajikan konten visual yang dinamis. Elemen multimedia adalah kunci untuk menciptakan pengalaman yang benar-benar imersif dan menarik bagi pengguna. Bayangkan sebuah game dengan efek suara ledakan yang seru, aplikasi podcast yang bisa Anda dengarkan saat berolahraga di sekitar Lapangan Blang Padang, atau aplikasi resep masakan yang menyertakan video tutorial langkah-demi-langkah. Inilah kekuatan multimedia yang akan kita pelajari.

Tujuan dari bab ini adalah untuk memberikan Anda kemampuan untuk menjadi 'sutradara' dan '*sound engineer*' bagi aplikasi Anda sendiri. Kita akan belajar cara menyematkan, mengontrol, dan mengelola pemutaran audio dan video langsung dari dalam aplikasi LAMW.

Perjalanan kita di dunia multimedia ini akan mencakup:

* **Efek Suara Singkat:** Belajar cara memutar suara-suara pendek dengan cepat dan efisien, seperti suara klik tombol atau notifikasi.
* **Pemutar Audio Jangka Panjang:** Mengelola pemutaran musik atau podcast, lengkap dengan kontrol *play, pause*, dan *seek*.
* **Video Sederhana:** Cara termudah untuk menampilkan dan memutar video di dalam aplikasi.
* **Video Profesional dengan ExoPlayer:** Mengintegrasikan *library* video modern dari Google untuk kemampuan yang lebih canggih dan andal.
* **Audio di Latar Belakang:** Mempelajari arsitektur yang benar untuk membuat aplikasi pemutar musik yang audionya tetap berjalan meskipun aplikasi tidak lagi di layar depan.

Beberapa topik di bab ini, khususnya saat kita membahas *library* video profesional seperti ExoPlayer, akan kembali mengandalkan keahlian **interoperabilitas JNI** yang telah kita asah di **Bab 10**. Ini adalah contoh lain bagaimana JNI membuka pintu ke fungsionalitas yang sangat canggih. Bersiaplah untuk membuat aplikasi Anda tidak hanya bisa dilihat, tetapi juga bisa didengar dan ditonton. Mari kita mulai!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 16.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai Bab 16.

Berikut adalah jawaban untuk prompt **Subbab 16.1: Efek Suara Singkat dengan jSoundPool**.

**16.1. Efek Suara Singkat dengan jSoundPool**

Tidak semua audio perlu diperlakukan seperti musik atau podcast. Terkadang, kita hanya butuh suara pendek untuk memberikan umpan balik instan kepada pengguna—seperti suara 'klik' saat tombol ditekan, suara 'cring' saat sebuah item berhasil ditambahkan, atau suara ledakan singkat dalam sebuah game. Untuk keperluan ini, menggunakan jMediaPlayer terlalu 'berat' dan bisa mengalami sedikit jeda (*latency*).

Solusi yang tepat dan efisien untuk ini adalah **jSoundPool**. Anggap jSoundPool sebagai sebuah 'gudang' untuk efek-efek suara pendek. Ia akan memuat semua suara ke dalam memori terlebih dahulu sehingga bisa diputar secara instan dengan latensi yang sangat rendah saat dibutuhkan.

**Langkah 1: Menyiapkan File Suara**

Android paling baik bekerja dengan format audio terkompresi seperti .ogg atau .mp3 untuk efek suara.

1. Di *file explorer* proyek Anda, buka direktori res.
2. Buat sebuah direktori baru di dalamnya dengan nama raw. Ini adalah tempat standar untuk meletakkan file-file aset mentah.
3. Salin file-file suara Anda (misalnya, klik\_tombol.ogg dan notif\_sukses.ogg) ke dalam folder res/raw ini.

**Langkah 2: Inisialisasi dan Memuat Suara ke SoundPool**

Proses memuat suara ke memori sebaiknya dilakukan sekali saja saat aplikasi atau *form* dimulai, misalnya di *event* OnCreate.

1. Di bagian private dari kelas *form* (TAndroidModule1) Anda, deklarasikan variabel untuk SoundPool dan untuk menyimpan ID dari setiap suara:

Delphi

private

{ private declarations }

FSoundPool: jSoundPool;

FClickSoundID: integer;

FSuccessSoundID: integer;

1. Kemudian, di *event* OnCreate *form*, kita inisialisasi SoundPool dan muat suaranya:

Delphi

procedure TAndroidModule1.jForm1Create(Sender: TObject);

begin

// 1. Buat instance SoundPool. Parameter pertama adalah jumlah

// stream maksimum yang bisa diputar bersamaan.

FSoundPool := tjSoundPool.Create(4);

// 2. Muat setiap file suara dari resource 'raw' dan simpan ID-nya

FClickSoundID := FSoundPool.Load(res.raw.klik\_tombol, 1);

FSuccessSoundID := FSoundPool.Load(res.raw.notif\_sukses, 1);

end;

FSoundPool.Load akan membaca file dari res/raw dan mempersiapkannya di memori. Ia mengembalikan sebuah integer (Sound ID) yang akan kita gunakan sebagai referensi untuk memutar suara tersebut.

**Langkah 3: Memainkan Efek Suara**

Setelah suara dimuat, memainkannya menjadi sangat mudah. Kita cukup memanggil metode .Play dengan ID suara yang relevan.

Delphi

// Contoh di dalam OnClick sebuah tombol biasa

procedure TAndroidModule1.btnAksiBiasaClick(Sender: TObject);

begin

if Assigned(FSoundPool) then

FSoundPool.Play(FClickSoundID, 1.0, 1.0, 1, 0, 1.0);

// ... (lakukan aksi tombol seperti biasa) ...

end;

// Contoh di dalam OnClick tombol simpan yang sukses

procedure TAndroidModule1.btnSimpanSuksesClick(Sender: TObject);

begin

// ... (logika menyimpan data) ...

if Assigned(FSoundPool) then

FSoundPool.Play(FSuccessSoundID, 1.0, 1.0, 1, 0, 1.0);

ShowMessage('Data berhasil disimpan!');

end;

**Penjelasan Parameter .Play:**

* soundID: ID suara yang ingin diputar (misal: FClickSoundID).
* leftVolume, rightVolume: Volume untuk speaker kiri dan kanan (rentang 0.0 hingga 1.0).
* priority: Prioritas *stream* (biasanya 1).
* loop: Berapa kali suara diulang. 0 berarti putar sekali, -1 berarti putar terus-menerus.
* rate: Kecepatan pemutaran (1.0 untuk normal).

**Langkah 4: Melepaskan Sumber Daya**

Untuk praktik pemrograman yang baik, jangan lupa untuk melepaskan sumber daya SoundPool saat *form* dihancurkan untuk menghindari kebocoran memori.

Delphi

procedure TAndroidModule1.jForm1Destroy(Sender: TObject);

begin

if Assigned(FSoundPool) then

FSoundPool.Release;

end;

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 16.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 16.2: Pemutar Audio untuk Musik atau Podcast (jMediaPlayer)**.

**16.2. Pemutar Audio untuk Musik atau Podcast (jMediaPlayer)**

Jika jSoundPool adalah pemutar MP3 ringkas untuk klip pendek, maka **jMediaPlayer** adalah *stereo set* lengkap Anda. Komponen ini dirancang untuk menangani file media berdurasi panjang seperti lagu, rekaman suara, atau *podcast* yang di-*streaming* dari internet. Karena prosesnya lebih kompleks (bisa memuat dari file besar atau jaringan), jMediaPlayer memiliki siklus hidup atau 'status' yang harus kita kelola dengan hati-hati.

**Memahami Siklus Hidup MediaPlayer (State Machine)**

Anda tidak bisa begitu saja memanggil .Start(). MediaPlayer bekerja seperti sebuah mesin dengan berbagai status (*state*). Anda harus memindahkannya dari satu status ke status berikutnya secara berurutan. Jika Anda mencoba memanggil perintah yang salah di status yang salah (misalnya, memanggil Start saat *player* belum siap), aplikasi Anda akan *crash*.

*(Catatan untuk penulis: Di sini adalah tempat yang ideal untuk menyertakan diagram alur state machine MediaPlayer yang standar: Idle -> [setDataSource()] -> Initialized -> [prepareAsync()] -> Preparing -> [onPrepared] -> Prepared. Dari status Prepared, baru bisa bercabang ke start(), pause(), stop(), seekTo()).*

Secara umum, alur yang aman adalah: **Create** -> **SetDataSource** -> **PrepareAsync** -> (tunggu *event* OnPrepared) -> **Start**.

**Memainkan Audio dari Berbagai Sumber**

Pertama, siapkan UI dengan beberapa tombol (play, pause, stop) dan sebuah jSeekBar untuk progres. Lalu deklarasikan FMediaPlayer: jMediaPlayer; di bagian private *form* Anda.

**A. Dari Resource Lokal (res/raw)** Pastikan file musik (misal: lagu\_pembuka.mp3) ada di folder res/raw.

Delphi

procedure TAndroidModule1.btnSiapkanLokalClick(Sender: TObject);

begin

if Assigned(FMediaPlayer) then FMediaPlayer.Free; // Bebaskan player lama jika ada

FMediaPlayer := tjMediaPlayer.Create;

// Siapkan listener sebelum prepare

FMediaPlayer.OnPrepared := @Self.MediaPlayerPrepared;

FMediaPlayer.SetDataSource(res.raw.lagu\_pembuka);

FMediaPlayer.PrepareAsync; // Siapkan di latar belakang, akan memicu OnPrepared jika sudah siap

end;

**B. Dari File di Penyimpanan Perangkat**

Delphi

// Asumsikan kita sudah punya path lengkap ke file

// dan sudah mendapatkan izin baca penyimpanan (jika perlu)

FMediaPlayer := tjMediaPlayer.Create;

FMediaPlayer.OnPrepared := @Self.MediaPlayerPrepared;

FMediaPlayer.SetDataSource('/storage/emulated/0/Music/lagu\_favorit.mp3');

FMediaPlayer.PrepareAsync;

**C. Streaming dari URL Internet**

Delphi

// Pastikan sudah ada izin INTERNET di manifest

FMediaPlayer := tjMediaPlayer.Create;

FMediaPlayer.OnPrepared := @Self.MediaPlayerPrepared;

FMediaPlayer.SetDataSource('https://server.com/podcast.mp3');

FMediaPlayer.PrepareAsync;

**Mengontrol Pemutaran**

Setelah *event* OnPrepared terpicu, itu artinya media sudah siap diputar. Di dalam OnPrepared, kita bisa mengaktifkan tombol-tombol kontrol kita.

Delphi

procedure TAndroidModule1.MediaPlayerPrepared(Sender: TObject);

begin

// Media siap, aktifkan tombol play dan seekbar

btnPlay.Enabled := True;

jSeekBar1.Max := FMediaPlayer.GetDuration; // Atur panjang seekbar

end;

procedure TAndroidModule1.btnPlayClick(Sender: TObject);

begin

if Assigned(FMediaPlayer) then FMediaPlayer.Start;

jTimer1.Enabled := True; // Aktifkan timer untuk update progres

end;

procedure TAndroidModule1.btnPauseClick(Sender: TObject);

begin

if Assigned(FMediaPlayer) and FMediaPlayer.IsPlaying then

FMediaPlayer.Pause;

jTimer1.Enabled := False;

end;

**Menampilkan Progres dengan jTimer dan jSeekBar**

Untuk membuat pemutar media yang interaktif, kita perlu menampilkan progres pemutaran dan memungkinkan pengguna untuk menggeser (*seek*) ke bagian tertentu.

1. **Siapkan Timer:** Letakkan komponen jTimer di *form* Anda. Atur Interval-nya menjadi 500 atau 1000 (update setiap setengah atau satu detik).
2. **Update UI di Event OnTimer:**

Delphi

procedure TAndroidModule1.jTimer1Timer(Sender: TObject);

begin

if Assigned(FMediaPlayer) and FMediaPlayer.IsPlaying then

begin

// Update posisi 'thumb' di SeekBar sesuai posisi pemutaran saat ini

jSeekBar1.Progress := FMediaPlayer.GetCurrentPosition;

end;

end;

1. **Mengubah Posisi dengan seekTo:** Kita perlu menangani saat pengguna menggeser SeekBar. Gunakan *event* OnSeekBarChange.

Delphi

procedure TAndroidModule1.jSeekBar1SeekBarChange(Sender: TObject; newProgress: integer; fromUser: boolean);

begin

// Hanya ubah posisi pemutaran jika digeser oleh pengguna

if Assigned(FMediaPlayer) and fromUser then

FMediaPlayer.SeekTo(newProgress);

end;

Jangan lupa untuk melepaskan FMediaPlayer (FMediaPlayer.Release) di *event* OnDestroy *form* untuk menghindari kebocoran memori.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 16.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 16.3: Video Sederhana dengan jVideoView**.

**16.3. Video Sederhana dengan jVideoView**

Setelah memahami jMediaPlayer untuk audio, kita akan melihat cara termudah untuk menampilkan video di dalam aplikasi. Android menyediakan sebuah komponen UI khusus untuk ini, yaitu **VideoView**, yang dibungkus oleh LAMW sebagai **jVideoView**.

jVideoView adalah komponen 'paket hemat'. Ia sudah menggabungkan sebuah permukaan untuk menampilkan video (*SurfaceView*) dengan logika MediaPlayer di dalamnya. Ini adalah cara tercepat untuk membuat video bisa diputar di aplikasi Anda dengan sangat sedikit baris kode.

**Langkah 1: Menyiapkan UI dan Sumber Video**

1. **UI Setup:** Di *Form Designer*, letakkan sebuah komponen jVideoView dari *Component Palette*. Atur ID-nya menjadi jVideoView1 dan atur layout\_width serta layout\_height-nya menjadi match\_parent atau ukuran spesifik yang Anda inginkan.
2. **Sumber Video:** Sama seperti audio, video bisa berasal dari beberapa sumber. Cara termudah untuk contoh ini adalah dengan menempatkan file video pendek (misalnya, sample\_video.mp4) di dalam direktori res/raw proyek Anda.

**Langkah 2: Memainkan Video dengan Kontrol Bawaan**

Keindahan jVideoView adalah kesederhanaannya, terutama saat digabungkan dengan jMediaController untuk menyediakan kontrol UI standar (play, pause, seek bar) secara otomatis.

Mari kita buat sebuah tombol yang akan memulai pemutaran video.

Delphi

procedure TAndroidModule1.btnPutarVideoClick(Sender: TObject);

var

path: string;

mediaController: jMediaController;

begin

// 1. Buat path khusus untuk resource di folder 'raw'

// Ini adalah format Uri standar untuk merujuk ke resource di dalam paket aplikasi.

path := 'android.resource://' + GetPackageName + '/' + IntToStr(res.raw.sample\_video);

// 2. Atur sumber video untuk jVideoView

jVideoView1.SetVideoPath(path);

// 3. Buat dan tautkan MediaController untuk menampilkan kontrol UI

mediaController := tjMediaController.Create(Self);

jVideoView1.SetMediaController(mediaController);

mediaController.SetAnchorView(jVideoView1);

// 4. Atur fokus ke VideoView dan mulai pemutaran

jVideoView1.RequestFocus;

jVideoView1.Start;

end;

**Penjelasan:**

* **path := 'android.resource://...'**: Kita membuat sebuah Uri dalam bentuk *string* untuk memberitahu VideoView di mana file sample\_video.mp4 kita berada di dalam paket aplikasi.
* **jVideoView1.SetVideoPath(path)**: Kita memberitahu VideoView file mana yang harus diputar.
* **jVideoView1.SetMediaController(...)**: Inilah bagian kuncinya. Kita membuat sebuah jMediaController dan menautkannya ke jVideoView kita. Kontrol ini akan otomatis muncul saat pengguna menyentuh area video dan akan hilang setelah beberapa saat tidak ada interaksi.

Hanya dengan beberapa baris kode ini, Anda sudah memiliki pemutar video yang fungsional lengkap dengan kontrol dasar!

**Langkah 3: Keterbatasan VideoView**

Meskipun sangat mudah digunakan, VideoView memiliki beberapa keterbatasan yang perlu diketahui:

* **Dukungan Format Terbatas:** Ia hanya mendukung format-format video standar yang bisa ditangani oleh MediaPlayer bawaan Android.
* **Kustomisasi Rendah:** Sangat sulit untuk mengubah tampilan dan nuansa dari kontrol pemutaran bawaan (MediaController).
* **Kurang Andal untuk Streaming:** Untuk *streaming* dengan format modern seperti DASH atau HLS yang memerlukan adaptasi *bitrate*, VideoView sering kali tidak cukup andal.

Karena keterbatasan inilah, untuk aplikasi yang sangat berfokus pada video (seperti pemutar video kustom atau aplikasi sekelas YouTube), Google sangat merekomendasikan untuk menggunakan *library* yang lebih kuat dan fleksibel: **ExoPlayer**, yang akan kita bahas di subbab selanjutnya.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 16.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 16.4: Video Profesional dengan ExoPlayer (via JNI)**.

**16.4. Video Profesional dengan ExoPlayer (via JNI)**

Jika jVideoView adalah mobil sedan keluarga yang nyaman untuk perjalanan singkat, maka **ExoPlayer** adalah mobil sport modular yang bisa Anda rakit dan kustomisasi untuk performa maksimal di segala medan. ExoPlayer adalah *library* pemutar media *open-source* dari Google yang menjadi standar de facto untuk aplikasi Android yang serius menangani video.

**Mengapa Menggunakan ExoPlayer?**

Dibandingkan MediaPlayer bawaan yang digunakan oleh VideoView, ExoPlayer menawarkan banyak keunggulan:

* **Dukungan Format Luas:** Secara bawaan mendukung format *streaming* modern seperti DASH dan HLS, yang penting untuk *streaming* video dengan kualitas adaptif (berubah sesuai kecepatan internet).
* **Kustomisasi Penuh:** Hampir setiap bagian dari *player*—mulai dari cara data dimuat, di-*render*, hingga tampilan kontrol UI—dapat dikustomisasi.
* **Manajemen Playlist:** Sangat mudah untuk membuat *playlist*, beralih antar video, atau bahkan menyisipkan iklan.
* **Dikelola Aktif oleh Google:** ExoPlayer terus diperbarui untuk mendukung fitur-fitur Android terbaru dan perbaikan bug, tidak seperti MediaPlayer yang jarang di-update.

**Sebuah Proyek Interoperabilitas JNI**

Sama seperti Google Maps, ExoPlayer adalah sebuah *library* Java/Kotlin. Oleh karena itu, untuk menggunakannya di LAMW, kita akan kembali mempraktikkan keahlian **JNI (Java Native Interface)** dari **Bab 10**. Kita akan membuat *wrapper* Pascal sederhana untuk mengontrol objek-objek ExoPlayer.

**Langkah 1: Setup Dependensi ExoPlayer**

Langkah pertama adalah memberitahu sistem *build* Gradle untuk menyertakan *library* ExoPlayer. Anda perlu menambahkan dependensi yang relevan ke file build.gradle proyek Anda (atau file konfigurasi lain yang setara di proyek LAMW Anda).

Groovy

// Di dalam file build.gradle Anda

dependencies {

...

// Per Juni 2025, library media dipindah ke androidx.media3

implementation 'androidx.media3:media3-exoplayer:1.3.1'

implementation 'androidx.media3:media3-ui:1.3.1'

// Versi bisa berubah, selalu cek versi stabil terbaru dari dokumentasi Google

}

**Langkah 2: Komponen Inti ExoPlayer**

Ada tiga kelas utama dari ExoPlayer yang akan kita bungkus dan gunakan:

1. **PlayerView**: Komponen UI dari ExoPlayer yang akan menampilkan video dan kontrol pemutaran yang modern dan dapat dikustomisasi.
2. **ExoPlayer**: Objek 'mesin' pemutar itu sendiri, yang menangani semua logika pemutaran seperti *buffering*, *decoding*, dan *rendering*.
3. **MediaItem**: Objek yang merepresentasikan satu file media (bisa audio atau video) yang akan diputar.

**Langkah 3: Implementasi (via Wrapper JNI)**

Berikut adalah alur kerja yang disederhanakan untuk menampilkan video dari sebuah URL.

**A. Siapkan UI dan Wrapper**

* Di *Form Designer*, siapkan sebuah jFrameLayout sebagai kontainer (misal: playerContainer).
* Buat unit Pascal baru untuk *wrapper* JNI Anda (uExoPlayerWrapper.pas) dan definisikan kelas-kelas pembungkus seperti TjExoPlayer, TjPlayerView, dan TjMediaItem, mirip dengan yang kita lakukan di Bab 10.

**B. Inisialisasi Player di OnCreate** Di dalam *event* OnCreate form utama, kita akan membuat dan menyiapkan semuanya.

Delphi

// Asumsi kita punya variabel global FExoPlayer: TjExoPlayer dan FPlayerView: TjPlayerView

procedure TAndroidModule1.jForm1Create(Sender: TObject);

begin

// 1. Buat PlayerView dan tambahkan ke kontainer kita

FPlayerView := TjPlayerView.Create(Self); // Wrapper kita memanggil constructor PlayerView

playerContainer.AddView(FPlayerView);

// 2. Buat instance ExoPlayer menggunakan Builder-nya

FExoPlayer := TjExoPlayer.Create(Self); // Wrapper kita memanggil ExoPlayer.Builder(context).build()

// 3. Tautkan Player ke View-nya agar video bisa ditampilkan

FPlayerView.setPlayer(FExoPlayer);

end;

**C. Memainkan Video dari URL** Buat prosedur terpisah untuk memulai pemutaran.

Delphi

procedure TAndroidModule1.PutarVideoExo(const AUrl: string);

var

mediaItem: TjMediaItem;

begin

if (not Assigned(FExoPlayer)) or (AUrl = '') then Exit;

// 1. Buat MediaItem dari URL

mediaItem := TjMediaItem.fromUri(AUrl); // Wrapper kita memanggil MediaItem.fromUri(string)

// 2. Atur item yang akan diputar ke player

FExoPlayer.setMediaItem(mediaItem);

// 3. Siapkan player dan mulai putar

FExoPlayer.prepare;

FExoPlayer.play;

end;

Anda kemudian bisa memanggil PutarVideoExo('https://storage.googleapis.com/exoplayer-test-media-0/BigBuckBunny\_320x180.mp4') dari OnClick sebuah tombol untuk memulai video.

Meskipun memerlukan kerja JNI di awal untuk membuat *wrapper*, hasilnya adalah pemutar video yang jauh lebih kuat, fleksibel, dan sesuai dengan standar industri saat ini. Kemampuan untuk mengintegrasikan *library* seperti ExoPlayer menunjukkan bahwa aplikasi LAMW Anda benar-benar tidak memiliki batasan fungsionalitas.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 16.5.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 16.5: Arsitektur Audio Latar Belakang (Background Audio)**.

**16.5. Arsitektur Audio Latar Belakang (Background Audio)**

Kita telah belajar cara memutar audio. Tapi, coba perhatikan: jika Anda memulai pemutaran musik di aplikasi kita lalu menekan tombol Home, musiknya akan berhenti. Mengapa? Karena jMediaPlayer atau ExoPlayer yang kita buat hidup di dalam sebuah Activity, dan saat Activity di-*pause* atau di-*destroy*, semua yang ada di dalamnya akan ikut berhenti.

Lalu, bagaimana aplikasi seperti Spotify atau pemutar *podcast* bisa terus memutar audio bahkan saat kita membuka aplikasi lain atau mengunci layar? Jawabannya adalah dengan **memisahkan 'pemutar' dari 'antarmuka'** menggunakan arsitektur **Service**.

**Arsitektur yang Benar: UI sebagai Remote, Service sebagai Pemutar**

Untuk membuat pemutar audio latar belakang yang andal, kita harus mengubah cara berpikir kita. Bayangkan sebuah sistem stereo di rumah:

* **Activity (*Form*) adalah Remote Control Anda:** Tugasnya hanya menampilkan judul lagu, *cover art*, dan tombol-tombol (Play, Pause, Next). Ia tidak memutar musik itu sendiri. Ia hanya mengirim perintah.
* **Service adalah Stereo Set/Player-nya:** Ini adalah komponen yang benar-benar memegang *instance* ExoPlayer atau MediaPlayer. Ia yang bertanggung jawab penuh atas proses memuat, memutar, dan menghentikan audio. Ia hidup di latar belakang, terpisah dari UI.

**Komunikasi Antara Activity dan Service**

Bagaimana 'remote control' bisa memberi perintah ke 'stereo set'? Jawabannya adalah dengan **Intent**, yang telah kita pelajari di Bab 5.

**Alur Kerjanya:**

1. **Activity Mengirim Perintah:** Saat pengguna menekan tombol 'Play' di Activity, Activity tidak memanggil FExoPlayer.play() secara langsung. Sebaliknya, ia membuat sebuah Intent dengan sebuah 'aksi' khusus, misalnya ACTION\_PLAY, dan mengirimkannya ke Service menggunakan startService(intent).
2. **Service Menerima dan Bertindak:** Service akan menerima Intent ini di dalam *event* OnStartCommand. Di sana, ia akan memeriksa 'aksi' dari Intent tersebut. Jika aksinya adalah ACTION\_PLAY, maka Service-lah yang akan memanggil FExoPlayer.play() pada *instance* player yang dimilikinya.

**Contoh Kode (Parsial):**

Delphi

// Di Activity (UI)

procedure TAndroidModule1.btnPlayClick(Sender: TObject);

var intent: jIntent;

begin

intent := jIntent.Create;

intent.SetComponent(Self, 'uAudioService'); // Target adalah kelas Service kita

intent.SetAction('com.myapp.ACTION\_PLAY');

Self.StartService(intent);

intent.Free;

end;

// Di Service (uAudioService.pas)

procedure TAndroidModule2.jService1StartCommand(..., intent: jIntent; ...);

var action: string;

begin

if Assigned(intent) then

begin

action := JStringToString(intent.GetAction);

if action = 'com.myapp.ACTION\_PLAY' then

FExoPlayer.play // Asumsi FExoPlayer (atau FMediaPlayer) hidup di sini

else if action = 'com.myapp.ACTION\_PAUSE' then

FExoPlayer.pause;

// ... dan seterusnya untuk aksi lain seperti NEXT, PREVIOUS, STOP

end;

Result := jService.START\_STICKY;

end;

**Foreground Service: Agar Tidak 'Dibunuh' Sistem**

Agar *service* yang memutar audio di latar belakang tidak tiba-tiba dihentikan oleh sistem Android saat memori menipis, kita harus 'mempromosikannya' menjadi **Foreground Service**.

Syarat untuk menjadi *Foreground Service* adalah: **ia wajib menampilkan sebuah Notifikasi yang persisten (tidak bisa di-swipe oleh pengguna)**. Notifikasi inilah yang biasa Anda lihat di *notification drawer* saat memutar musik, yang biasanya berisi judul lagu dan tombol kontrol play/pause.

**Menyatukan Semua Keahlian:** Di sinilah semua pengetahuan kita dari bab-bab sebelumnya bertemu:

1. Di dalam Service, saat pemutaran dimulai, kita membuat sebuah **Notifikasi** (seperti di **Bab 8**) lengkap dengan **PendingIntents** untuk aksi play/pause.
2. Kemudian kita panggil startForeground(NOTIFICATION\_ID, myNotification). Panggilan ini akan menampilkan notifikasi persisten dan memberitahu sistem Android, "Proses ini sangat penting bagi pengguna, jangan hentikan."

Membangun pemutar audio latar belakang adalah tantangan arsitektur yang sangat baik. Ini memaksa kita untuk menggabungkan pengetahuan tentang Activity, Service (dari **Bab 9**), Intent (dari **Bab 5**), dan Notification (dari **Bab 8**) menjadi satu solusi yang elegan dan profesional.

Ini mengakhiri Bab 16. Anda sekarang memiliki kemampuan untuk memperkaya aplikasi Anda dengan berbagai bentuk multimedia. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 17 (Pengantar Bab 17).

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Pengantar Bab 17**.

**Bab 17: Kontrol Kamera Tingkat Lanjut dengan CameraX**

Selamat datang di salah satu bab paling canggih dalam buku ini. Di Bab 8, kita telah belajar cara 'meminta tolong'—yaitu menggunakan Intent untuk menyuruh aplikasi kamera bawaan mengambil gambar untuk kita. Cara itu cepat dan mudah, tetapi tidak fleksibel. Sekarang, kita akan naik ke level tertinggi: kita akan belajar menjadi **'fotografernya' sendiri** dengan membangun antarmuka kamera kustom langsung di dalam aplikasi kita.

Mengapa kita perlu melakukan ini? Bayangkan Anda ingin membuat aplikasi dengan filter *real-time* seperti Instagram, aplikasi pemindai (scanner) dokumen atau kode QR, atau sekadar ingin memberikan pengalaman mengambil foto yang mulus tanpa perlu keluar dari layar aplikasi Anda. Untuk semua ini, kita memerlukan kontrol penuh atas perangkat keras kamera.

Secara historis, mengontrol kamera langsung di Android sangatlah rumit. Untungnya, Google telah menyediakan *library* modern bernama **CameraX**. CameraX adalah sebuah lapisan abstraksi yang dirancang agar jauh lebih mudah digunakan, sadar-siklus-hidup (*lifecycle-aware*), dan konsisten di berbagai perangkat Android. Sama seperti ExoPlayer dan Google Maps, CameraX adalah *library* Java/Kotlin. Oleh karena itu, bab ini sekali lagi adalah sebuah **studi kasus JNI tingkat mahir**. Pemahaman yang mendalam tentang semua konsep yang telah kita pelajari di **Bab 10** adalah syarat mutlak untuk bisa mengikuti bab ini.

Di bab ini, kita akan membangun sebuah layar di aplikasi kita yang berfungsi sebagai *viewfinder* kamera. Perjalanan kita akan mencakup:

* Melakukan setup dependensi CameraX dan izin yang diperlukan.
* Memahami konsep inti CameraX seperti *Use Cases* dan *Lifecycle Binding*.
* Menampilkan pratinjau kamera (*camera preview*) secara langsung di dalam UI aplikasi kita.
* Membangun antarmuka pengguna kustom, seperti tombol *shutter* dan tombol untuk beralih kamera.
* Menangkap gambar secara terprogram dan menyimpannya ke perangkat.

Ini adalah tantangan yang akan menguji semua keahlian JNI Anda, tetapi imbalannya sangat besar: kemampuan untuk menciptakan pengalaman berbasis kamera yang benar-benar unik dan terintegrasi. Mari kita mulai.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 17.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai Bab 17.

Berikut adalah jawaban untuk prompt **Subbab 17.1: Persiapan Proyek dan Dependensi CameraX**.

**17.1. Persiapan Proyek dan Dependensi CameraX**

Sama seperti *library* Google lainnya, langkah pertama sebelum kita bisa menggunakan kekuatan CameraX adalah memastikan proyek kita dikonfigurasi dengan benar. Ini melibatkan dua hal utama: mendeklarasikan dan meminta izin yang diperlukan, serta menambahkan *library* CameraX sebagai dependensi ke dalam proyek kita.

**Langkah 1: Konfigurasi Izin (Permissions)**

**A. Deklarasi di AndroidManifest.xml** Pertama, pastikan file AndroidManifest.xml Anda memiliki deklarasi izin untuk menggunakan kamera. Tanpa ini, sistem tidak akan pernah memberikan akses ke perangkat keras kamera.

XML

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

<uses-feature android:name="android.hardware.camera.any" />

**B. Implementasi Izin Runtime (Wajib)** Seperti yang telah kita pelajari, Kamera termasuk dalam kategori 'izin berbahaya'. Oleh karena itu, Anda **wajib** mengimplementasikan alur permintaan izin *runtime* dari **Bab 6**. Buatlah sebuah prosedur, misalnya CheckAndRequestCameraPermission, yang akan memeriksa apakah izin Manifest\_permission\_CAMERA sudah diberikan. Jika belum, prosedur ini akan menampilkan dialog permintaan kepada pengguna. Semua logika untuk memulai kamera hanya boleh dijalankan **setelah** Anda memastikan izin telah diberikan oleh pengguna.

**Langkah 2: Menambahkan Dependensi CameraX ke Proyek**

CameraX adalah sekumpulan *library* modular yang tidak termasuk dalam SDK standar, jadi kita perlu menambahkannya secara manual ke dalam file konfigurasi *build* proyek kita. Untuk proyek Android modern, ini dilakukan di file build.gradle.

**Library yang Dibutuhkan:**

* **camera-core**: Pustaka inti yang berisi API dasar dan kelas-kelas utama CameraX.
* **camera-camera2**: Implementasi CameraX yang berjalan di atas API Camera2 (standar untuk perangkat modern).
* **camera-lifecycle**: Menyediakan integrasi dengan siklus hidup Android, yang sangat menyederhanakan manajemen sumber daya kamera (membuka dan menutup kamera secara otomatis).
* **camera-view**: Menyediakan komponen UI seperti PreviewView yang akan kita gunakan untuk menampilkan pratinjau kamera.

**Contoh di build.gradle:** Buka file build.gradle (level modul) proyek Anda dan tambahkan baris-baris berikut di dalam blok dependencies. Pastikan untuk memeriksa versi stabil terbaru dari situs Android Developer karena versi ini bisa berubah.

Groovy

// Per Juni 2025, versi stabil mungkin di sekitar 1.3.x atau 1.4.x

def camerax\_version = "1.3.1"

dependencies {

... // dependensi lain yang sudah ada

// CameraX

implementation "androidx.camera:camera-core:${camerax\_version}"

implementation "androidx.camera:camera-camera2:${camerax\_version}"

implementation "androidx.camera:camera-lifecycle:${camerax\_version}"

implementation "androidx.camera:camera-view:${camerax\_version}"

}

**Sinkronisasi Proyek:** Setelah menambahkan baris-baris ini, jangan lupa untuk melakukan 'Sync' atau 'Rebuild' proyek Anda. IDE akan mengunduh dan mengintegrasikan *library-library* tersebut, membuatnya siap untuk kita panggil melalui JNI.

Dengan izin yang sudah diatur dan *library* yang sudah ditambahkan, 'fondasi' proyek kita kini siap untuk mulai membangun fungsionalitas kamera kustom.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 17.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 17.2: Memahami Konsep Inti CameraX**.

**17.2. Memahami Konsep Inti CameraX**

Sebelum kita terjun ke dalam kode JNI yang detail, sangat penting untuk memahami filosofi dan konsep di balik CameraX. Berbeda dengan API kamera lama yang prosedural, CameraX dirancang dengan pendekatan berbasis **'Kasus Penggunaan' (Use Cases)** yang deklaratif dan sadar-siklus-hidup. Memahami konsep-konsep inti ini akan membuat proses implementasi nanti jauh lebih mudah.

**1. PreviewView - Jendela Kamera Anda**

Ini adalah komponen UI yang disediakan oleh *library* camera-view. Anggap saja ini sebagai 'kaca' atau 'jendela' khusus tempat pratinjau langsung dari kamera akan ditampilkan di dalam *layout* aplikasi Anda. Tugasnya sangat spesifik: merender *stream* video dari kamera dengan efisien. Kita akan menempatkan komponen ini di dalam *layout* kita, sama seperti komponen UI lainnya.

**2. Arsitektur Berbasis "Use Case"**

Inilah konsep paling fundamental dari CameraX. Alih-alih kita memberi perintah-perintah level rendah ke kamera ('set resolusi', 'atur fokus', 'mulai preview'), kita hanya mendeklarasikan **apa yang ingin kita lakukan** dengan kamera. CameraX akan mengurus detail implementasi yang rumit di belakang layar.

Tiga *use case* utama adalah:

* **Preview**: Kasus penggunaan ini memberitahu CameraX, "Tujuan utama saya adalah menampilkan pratinjau kamera ke pengguna." Kita akan menautkan *use case* ini ke PreviewView kita.
* **ImageCapture**: Kasus penggunaan ini memberitahu CameraX, "Saya butuh fungsionalitas untuk mengambil foto berkualitas tinggi." *Use case* ini menyediakan metode seperti takePicture() yang akan kita panggil.
* **ImageAnalysis**: Kasus penggunaan tingkat lanjut ini memberitahu CameraX, "Saya ingin menganalisis setiap *frame* dari video secara *real-time*." Ini tidak akan kita implementasikan, tetapi ini adalah dasar untuk fitur seperti pemindai kode QR, deteksi wajah, atau pemindai teks.

Kita bisa menggabungkan beberapa *use case* sekaligus, misalnya Preview dan ImageCapture, untuk menampilkan pratinjau dan sekaligus bisa mengambil foto.

**3. CameraSelector - Memilih Kamera yang Tepat**

Sebagian besar ponsel modern memiliki lebih dari satu kamera (belakang, depan, *wide-angle*, dll.). CameraSelector adalah objek sederhana yang kita gunakan untuk memberitahu CameraX kamera mana yang ingin kita gunakan. Pilihan paling umum adalah:

* **CameraSelector.DEFAULT\_BACK\_CAMERA**: Untuk memilih kamera belakang utama.
* **CameraSelector.DEFAULT\_FRONT\_CAMERA**: Untuk memilih kamera depan (selfie).

**4. Keajaiban "Lifecycle Binding"**

Ini adalah fitur yang membuat CameraX jauh lebih superior dan aman dibandingkan API lama. Dulu, developer sering lupa melepaskan (*release*) sumber daya kamera saat aplikasi di-*pause* atau di-*destroy*, yang menyebabkan *crash* atau boros baterai.

Dengan CameraX, kita tidak perlu mengelola ini secara manual. Kita cukup **'mengikat' (*bind*)** *use case-use case* yang kita inginkan ke siklus hidup dari Activity atau jForm kita.

* **Bagaimana cara kerjanya?** Saat kita memanggil metode bindToLifecycle, kita memberitahu CameraX, "Hai CameraX, tolong kelola *use case Preview* dan *ImageCapture* ini. Mulailah mereka saat jForm saya dalam keadaan aktif (*resumed*), dan hentikan serta lepaskan mereka secara otomatis saat jForm saya di-*pause* atau di-*destroy*."

Ini secara drastis mengurangi risiko kebocoran memori (*memory leak*) dan membuat kode kita jauh lebih bersih dan andal.

*(Catatan untuk penulis: Diagram sederhana yang menunjukkan jForm (sebagai LifecycleOwner) yang 'mengikat' Preview, ImageCapture, dan CameraSelector akan sangat membantu memvisualisasikan arsitektur ini).*

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 17.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 17.3: Implementasi Pratinjau Kamera (Camera Preview)**.

**17.3. Implementasi Pratinjau Kamera (Camera Preview)**

Setelah memahami konsepnya, kini saatnya kita masuk ke dalam kode. Di subbab ini, kita akan mengimplementasikan bagian paling fundamental: menampilkan pratinjau kamera secara langsung di layar aplikasi kita. Ini adalah langkah 'Hello, World!' untuk CameraX. Ingat, karena kita berinteraksi dengan *library* Java, semua kode ini akan melibatkan pemanggilan JNI melalui *wrapper* yang kita buat (seperti yang dipelajari di Bab 10).

**Langkah 1: Desain Layout UI**

Desain *layout* kita akan menggunakan jFrameLayout sebagai basis agar kita bisa menumpuk kontrol UI (seperti tombol *shutter*) di atas pratinjau kamera nantinya.

1. Di *Form Designer*, tambahkan sebuah jFrameLayout.
2. Atur ID-nya menjadi cameraContainer.
3. Atur layout\_width dan layout\_height-nya menjadi match\_parent agar memenuhi seluruh layar.

Kita tidak akan menambahkan PreviewView dari *Component Palette*. Sebaliknya, kita akan membuatnya secara terprogram dan menambahkannya ke dalam cameraContainer ini.

**Langkah 2: Inisialisasi dan Memulai Kamera**

Seluruh logika akan kita letakkan dalam sebuah prosedur, misalnya StartCamera, yang akan kita panggil dari OnCreate **setelah** kita memastikan izin kamera sudah diberikan.

Pertama, deklarasikan beberapa variabel di bagian private dari kelas *form* Anda untuk menyimpan referensi ke objek-objek CameraX:

Delphi

private

{ private declarations }

FCameraProvider: jProcessCameraProvider;

FImageCapture: jImageCapture;

FPreview: jPreview;

FCamera: jCamera;

Sekarang, buat prosedur StartCamera dan implementasikan logikanya.

Delphi

procedure TAndroidModule1.StartCamera;

var

cameraProviderFuture: jListenableFuture;

cameraSelector: jCameraSelector;

previewView: jPreviewView; // Asumsi ada wrapper TjPreviewView

begin

// Buat PreviewView dan tambahkan ke kontainer kita

previewView := TjPreviewView.Create(Self);

previewView.setLayoutParams(tjFrameLayout\_LayoutParams.Create(lparams.MATCH\_PARENT, lparams.MATCH\_PARENT));

cameraContainer.AddView(previewView);

// 1. Dapatkan instance CameraProvider secara asinkron

cameraProviderFuture := tjProcessCameraProvider.getInstance(Self);

// 2. Tambahkan listener untuk merespons saat CameraProvider siap

cameraProviderFuture.addListener(

// Prosedur anonim ini akan dijalankan saat provider siap

procedure

begin

try

// Dapatkan objek CameraProvider dari Future

FCameraProvider := jProcessCameraProvider(cameraProviderFuture.get);

// Buat use case Preview untuk menampilkan pratinjau

FPreview := tjPreview.Create;

// Buat use case ImageCapture untuk mengambil foto nanti

FImageCapture := tjImageCapture.Create;

// Pilih kamera belakang sebagai default

cameraSelector := tjCameraSelector.GetDEFAULT\_BACK\_CAMERA;

// Unbind semua use case sebelumnya untuk memastikan state bersih

FCameraProvider.unbindAll;

// 3. (KUNCI) Ikat use case ke siklus hidup form

FCamera := FCameraProvider.bindToLifecycle(Self, cameraSelector, [FPreview, FImageCapture]);

// 4. Hubungkan use case Preview ke PreviewView di UI

FPreview.setSurfaceProvider(previewView.getSurfaceProvider);

except

on E: Exception do Log.e('CameraX', 'Gagal mengikat use case: ' + E.Message);

end;

end,

GetMainExecutor); // Jalankan listener ini di thread utama

end;

**Penjelasan Detail Kode JNI**

* **tjProcessCameraProvider.getInstance(Self)**: Ini adalah cara kita meminta 'manajer kamera' utama. Prosesnya asinkron, artinya tidak langsung tersedia, sehingga ia mengembalikan sebuah objek ListenableFuture.
* **cameraProviderFuture.addListener(...)**: Karena prosesnya asinkron, kita memasang sebuah *listener* yang akan dieksekusi ketika CameraProvider sudah siap. Seluruh logika inti kita diletakkan di dalam *listener* ini.
* **FCameraProvider.bindToLifecycle(...)**: Inilah pemanggilan paling ajaib dan penting di CameraX. Kita memberitahu CameraProvider untuk:
  1. Menggunakan Self (yaitu jForm kita) sebagai pemilik siklus hidup.
  2. Menggunakan cameraSelector yang telah kita pilih (kamera belakang).
  3. Mengaktifkan dua *use case* yang kita butuhkan: FPreview dan FImageCapture. Metode ini mengembalikan objek jCamera yang bisa kita gunakan nanti untuk kontrol lebih lanjut (seperti *flash*).
* **FPreview.setSurfaceProvider(...)**: Langkah terakhir ini adalah 'menancapkan kabel' dari *output* Preview ke 'monitor' PreviewView kita, sehingga gambar pratinjau bisa muncul di layar.

Setelah Anda memanggil prosedur StartCamera ini (dari OnCreate atau setelah izin diberikan), aplikasi Anda akan menampilkan *feed* kamera langsung dari perangkat. Anda belum bisa mengambil gambar, tetapi pratinjau sudah aktif. Selamat, bagian tersulit dari integrasi CameraX telah berhasil Anda lewati!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 17.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 17.4: Membangun UI Kustom dan Menangkap Gambar**.

**17.4. Membangun UI Kustom dan Menangkap Gambar**

Pratinjau kamera sudah tampil, tetapi belum ada cara untuk mengambil gambar. Sekarang, kita akan melengkapi aplikasi kamera kita dengan menambahkan kontrol UI kustom—yaitu sebuah tombol *shutter*—dan menulis logika untuk menangkap gambar saat tombol tersebut ditekan.

**Langkah 1: Menambahkan Tombol Shutter ke Layout**

Karena kita menggunakan jFrameLayout sebagai kontainer utama, kita bisa dengan mudah menumpuk komponen lain di atas pratinjau kamera.

1. Di *Form Designer*, pilih jImageButton dari *Component Palette* dan letakkan di dalam cameraContainer (komponen jFrameLayout kita).
2. Di *Object Inspector*, atur propertinya:
   * **ID:** btnShutter
   * **Image:** Atur dengan sebuah ikon kamera atau lingkaran yang sudah Anda siapkan di res/drawable.
   * **layout\_gravity:** Atur menjadi **bottom|center\_horizontal** agar tombol selalu berada di tengah bawah layar.
   * **layout\_marginBottom:** Beri sedikit margin bawah agar tidak terlalu menempel di tepi, misalnya 32dp.

**Langkah 2: Menulis Logika Pengambilan Gambar**

Semua logika akan kita letakkan di dalam *event* OnClick dari btnShutter. Ingat, objek FImageCapture sudah kita buat dan ikat ke siklus hidup di subbab sebelumnya.

**Proses Asinkron:** Sama seperti proses lainnya yang melibatkan I/O (Input/Output), mengambil dan menyimpan gambar adalah proses yang butuh waktu dan tidak boleh memblokir UI. Metode takePicture bekerja secara asinkron dan akan memberikan hasilnya melalui sebuah *callback*.

**Kode OnClick Tombol Shutter:**

Delphi

procedure TAndroidModule1.btnShutterClick(Sender: TObject);

var

outputFileOptions: jImageCapture\_OutputFileOptions;

file: jFile;

imageSavedCallback: jImageCapture\_OnImageSavedCallback;

begin

// Pastikan use case ImageCapture sudah siap

if not Assigned(FImageCapture) then

begin

ShowMessage('Fitur kamera belum siap.');

Exit;

end;

// 1. Tentukan di mana file akan disimpan.

// Kita gunakan direktori pribadi aplikasi, seperti di Bab 6.

// Nama file dibuat unik berdasarkan waktu saat ini.

file := tjFile.Create(GetExternalFilesDir(nil), 'IMG\_' + DateTimeToTimeStampStr(Now) + '.jpg');

// 2. Buat konfigurasi output

outputFileOptions := tjImageCapture\_OutputFileOptions.Create(file);

// 3. Buat implementasi untuk callback yang akan menangani hasil

imageSavedCallback := tjImageCapture\_OnImageSavedCallback.Create;

imageSavedCallback.onImageSaved := procedure(outputFileResults: jImageCapture\_OutputFileResults)

begin

// Kode ini akan berjalan SETELAH gambar berhasil disimpan

// Kita berada di thread utama, jadi aman untuk update UI

ShowMessage('Foto berhasil disimpan di: ' + JStringToString(outputFileResults.getSavedUri.toString));

// Dari sini, Anda bisa membuka layar baru untuk menampilkan gambar dari Uri ini

end;

imageSavedCallback.onError := procedure(exception: jImageCaptureException)

begin

// Kode ini berjalan JIKA terjadi error saat mengambil atau menyimpan gambar

Log.e('CameraX', 'Gagal menyimpan foto: ' + JStringToString(exception.getMessage));

ShowMessage('Gagal mengambil foto.');

end;

// 4. Panggil takePicture!

FImageCapture.takePicture(outputFileOptions, GetMainExecutor, imageSavedCallback);

end;

**Penjelasan Detail Kode**

* **jImageCapture\_OutputFileOptions**: Objek ini memberitahu CameraX di mana dan bagaimana cara menyimpan file gambar yang akan diambil.
* **jImageCapture\_OnImageSavedCallback**: Ini adalah *listener* kita. Kita membuat sebuah *instance*-nya dan mengimplementasikan dua metode anonim:
  + **onImageSaved**: Dipanggil jika semua proses (pengambilan, pemrosesan, penyimpanan) berhasil. Ia memberikan kita sebuah OutputFileResults yang berisi Uri ke file yang baru saja disimpan.
  + **onError**: Dipanggil jika terjadi kegagalan di salah satu tahap.
* **FImageCapture.takePicture(...)**: Inilah pemanggilan utamanya. Kita memberikan tiga hal: konfigurasi *output* (outputFileOptions), sebuah *executor* untuk menjalankan *callback* (kita gunakan GetMainExecutor agar *callback* berjalan di UI Thread), dan objek *callback* itu sendiri.

Sekarang, jalankan aplikasi Anda. Arahkan kamera ke suatu objek, lalu tekan tombol *shutter* kustom Anda. Jika berhasil, Anda akan melihat sebuah pesan ShowMessage yang berisi lokasi file gambar yang baru saja disimpan. Anda telah berhasil menciptakan fungsionalitas kamera penuh dari awal hingga akhir!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 17.5.

YAHOX

Tampilkan alur berpikir

Baik, kita selesaikan Bab 17 ini.

Berikut adalah jawaban untuk prompt **Subbab 17.5: Fitur Tambahan: Beralih Kamera dan Kontrol Flash**.

**17.5. Fitur Tambahan: Beralih Kamera dan Kontrol Flash**

Aplikasi kamera kustom kita sudah bisa menampilkan pratinjau dan mengambil gambar. Sekarang, mari kita tambahkan beberapa fitur esensial yang akan membuatnya terasa seperti aplikasi kamera sungguhan: kemampuan untuk beralih antara kamera depan dan belakang, serta mengontrol *flash*.

**1. Beralih Kamera (Switch Camera)**

Fitur ini memungkinkan pengguna untuk mengambil foto *selfie*. Logikanya adalah dengan 'melepas ikatan' (*unbind*) semua *use case* yang sedang berjalan, lalu 'mengikatnya kembali' (*re-bind*) dengan CameraSelector yang berbeda.

1. **Tambahkan Tombol UI:** Di *layout* Anda, tambahkan sebuah jImageButton (btnSwitchCamera) dengan ikon 'balik kamera', letakkan di pojok atas layar.
2. **Simpan Status Kamera:** Kita perlu variabel untuk mengingat kamera mana yang sedang aktif. Di bagian private *form*, tambahkan: FCurrentCameraSelector: jCameraSelector;
3. **Modifikasi StartCamera:** Di prosedur StartCamera yang sudah kita buat, saat pertama kali memilih kamera, simpan pilihan tersebut ke variabel kita.

Delphi

// ... di dalam prosedur StartCamera ...

// Pilih kamera belakang sebagai default saat pertama kali

FCurrentCameraSelector := tjCameraSelector.GetDEFAULT\_BACK\_CAMERA;

// ... lalu gunakan FCurrentCameraSelector saat memanggil bindToLifecycle ...

FCamera := FCameraProvider.bindToLifecycle(Self, FCurrentCameraSelector, [FPreview, FImageCapture]);

1. **Implementasi OnClick Tombol Beralih:** Di *event* klik btnSwitchCamera, kita akan membalik pilihan dan memulai ulang sesi kamera.

Delphi

procedure TAndroidModule1.btnSwitchCameraClick(Sender: TObject);

begin

// Pastikan CameraProvider sudah siap

if not Assigned(FCameraProvider) then Exit;

// Balik pilihan kamera

if FCurrentCameraSelector = tjCameraSelector.GetDEFAULT\_BACK\_CAMERA then

FCurrentCameraSelector := tjCameraSelector.GetDEFAULT\_FRONT\_CAMERA

else

FCurrentCameraSelector := tjCameraSelector.GetDEFAULT\_BACK\_CAMERA;

// Mulai ulang kamera dengan selector yang baru.

// Pertama, lepas semua ikatan yang ada.

FCameraProvider.unbindAll;

// Kemudian, ikat kembali dengan selector yang baru.

FCamera := FCameraProvider.bindToLifecycle(Self, FCurrentCameraSelector, [FPreview, FImageCapture]);

end;

**2. Kontrol Flash**

CameraX memberikan kita kontrol mudah terhadap *flash* atau 'senter' (*torch*) pada perangkat.

**A. Menyalakan Senter (Torch)** Fitur ini berguna untuk menyalakan lampu kilat secara terus-menerus.

1. Tambahkan tombol jImageButton (btnToggleFlash) ke UI.
2. Kita bisa mendapatkan objek CameraControl dari objek Camera yang dikembalikan oleh bindToLifecycle.
3. Di *event* OnClick tombol *flash*:

Delphi

procedure TAndroidModule1.btnToggleFlashClick(Sender: TObject);

begin

// Pastikan objek Camera sudah ada

if not Assigned(FCamera) then Exit;

// Cek apakah perangkat memiliki unit flash

if FCamera.GetCameraInfo.HasFlashUnit then

begin

// Cek status senter saat ini, lalu balik statusnya

if FCamera.GetCameraInfo.GetTorchState.getValue = jTorchState.ON then

FCamera.GetCameraControl.enableTorch(False) // Matikan

else

FCamera.GetCameraControl.enableTorch(True); // Nyalakan

end;

end;

**B. Mengatur Mode Flash untuk Pengambilan Gambar** Jika Anda hanya ingin *flash* menyala saat foto diambil (bukan menyala terus-menerus), Anda harus mengaturnya di *use case* ImageCapture.

Delphi

// Atur ini saat inisialisasi atau saat pengguna memilih mode flash

// Mode bisa: FLASH\_MODE\_ON, FLASH\_MODE\_OFF, atau FLASH\_MODE\_AUTO

if Assigned(FImageCapture) then

FImageCapture.setFlashMode(jImageCapture.FLASH\_MODE\_AUTO);

**3. Sekilas tentang Fitur Lainnya: Tap-to-Focus**

Sebagai bocoran fitur yang lebih canggih, CameraX juga mendukung *tap-to-focus*. Konsepnya adalah:

1. Anda menangkap *event* sentuhan pada PreviewView.
2. Anda membuat sebuah **MeteringPoint** berdasarkan koordinat sentuhan tersebut.
3. Anda membuat sebuah **FocusMeteringAction** menggunakan *metering point* itu.
4. Anda memanggil FCamera.GetCameraControl.startFocusAndMetering(action). Kamera kemudian akan mencoba fokus ke titik yang Anda sentuh.

Implementasi detailnya memerlukan beberapa baris kode JNI lagi, tetapi ini menunjukkan betapa dalamnya tingkat kontrol yang bisa Anda dapatkan dengan CameraX untuk membangun aplikasi kamera yang benar-benar profesional.

Ini mengakhiri Bab 17. Anda kini memiliki kemampuan untuk membangun pengalaman kamera yang terintegrasi penuh di dalam aplikasi Anda. Saya akan menunggu perintah "yahok" dari Anda untuk memulai Bab 18 (Pengantar Bab 18).

YAHOX

Tampilkan alur berpikir

Baik, kita mulai bab terakhir dari seri pembahasan khusus ini.

Berikut adalah jawaban untuk prompt **Pengantar Bab 18**.

**Bab 18: Integrasi Hibrida: Menjembatani Pascal dan JavaScript dengan WebView**

Di Bab 8, kita telah belajar menggunakan jWebView sebagai sebuah 'jendela' pasif untuk menampilkan konten dari dunia web. Kita bisa menampilkan halaman bantuan, artikel, atau kebijakan privasi. Namun, interaksinya hanya satu arah: aplikasi kita menampilkan, dan pengguna melihat. Di bab terakhir dari seri pembahasan khusus ini, kita akan membuka sebuah level baru yang sangat kuat. Kita akan mengubah 'jendela' tersebut menjadi sebuah **'pintu dua arah'**. Kita akan membuat aplikasi Pascal *native* dan halaman web JavaScript kita bisa saling 'berbicara', berkolaborasi, dan bekerja sama secara mendalam.

Mengapa kita perlu melakukan ini? Kemampuan untuk menjembatani dunia *native* dan web membuka banyak strategi pengembangan yang canggih:

* **Memanfaatkan UI Web yang Sudah Ada:** Jika Anda sudah memiliki antarmuka pengguna yang kompleks dan interaktif yang dibangun dengan HTML/CSS/JS, Anda bisa menyematkannya di dalam aplikasi dan tetap mengontrolnya dari logika Pascal.
* **Konten yang Dapat Diperbarui dari Jarak Jauh:** Bayangkan sebuah halaman 'event promosi' di aplikasi Anda. Dengan membuatnya sebagai halaman web, Anda bisa mengubah konten promosi tersebut di server kapan saja tanpa perlu merilis *update* baru untuk aplikasi Anda di Play Store.
* **Memanggil Fungsionalitas Native:** Dari dalam halaman web, Anda bisa memicu fungsionalitas asli perangkat yang tidak bisa diakses oleh web biasa, seperti getaran (*vibrate*), menampilkan dialog *native*, atau menyimpan data ke database SQLite aplikasi.

Kunci dari semua keajaiban ini adalah sebuah mekanisme yang disebut **JavaScript Interface**. Ini adalah sebuah 'jembatan' yang kita bangun secara eksplisit untuk mengekspos fungsi-fungsi Pascal kita agar bisa 'dilihat' dan dipanggil oleh kode JavaScript di dalam WebView.

Perjalanan kita dalam membangun aplikasi hibrida ini akan mencakup:

* Mempelajari cara **memanggil fungsi Pascal dari JavaScript**.
* Mempelajari cara sebaliknya: **mengeksekusi JavaScript dari kode Pascal**.
* Mencegat dan **mengontrol navigasi link** di dalam WebView.
* Mengganti dialog standar JavaScript (seperti alert()) dengan dialog *native* Android untuk pengalaman yang lebih mulus.

Dengan menguasai teknik-teknik ini, Anda akan memiliki fleksibilitas untuk menggabungkan yang terbaik dari kedua dunia: performa dan akses perangkat keras dari kode *native* Pascal, dengan kecepatan dan kemudahan pembaruan dari teknologi web.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 18.1.

YAHOX

Tampilkan alur berpikir

Baik, kita mulai Bab 18.

Berikut adalah jawaban untuk prompt **Subbab 18.1: Setup Lanjutan: WebViewClient dan WebChromeClient**.

**18.1. Setup Lanjutan: WebViewClient dan WebChromeClient**

Untuk bisa 'menguping' dan 'mengintervensi' apa yang terjadi di dalam jWebView, kita tidak bisa bekerja dengan komponen jWebView itu sendiri. Sebaliknya, Android menyediakan dua 'asisten' atau 'klien' yang bisa kita tugaskan untuk mengawasinya. Dengan membuat versi kustom dari klien-klien ini, kita mendapatkan kontrol penuh atas hampir semua perilaku WebView.

**Review Singkat Pengaturan Dasar**

Seperti yang kita pelajari di Bab 8, dua pengaturan paling dasar yang hampir selalu kita lakukan adalah mengaktifkan JavaScript dan memuat sebuah halaman. Mari kita jadikan ini sebagai titik awal kita di dalam *event* OnCreate form.

Delphi

procedure TAndroidModule1.jForm1Create(Sender: TObject);

begin

// 1. Aktifkan JavaScript agar halaman web interaktif bisa berjalan

jWebView1.GetSettings.SetJavaScriptEnabled(True);

// 2. Muat halaman web lokal dari folder assets untuk contoh kita

// (Asumsikan Anda sudah membuat file ini)

jWebView1.LoadUrl('file:///android\_asset/halaman\_interaktif.html');

end;

**Klien #1: WebViewClient - Sang Pengawas Konten**

WebViewClient adalah asisten yang bertugas mengawasi semua hal yang berkaitan dengan **konten halaman web itu sendiri**. Tugas utamanya adalah:

* **Mencegat Navigasi URL:** Saat pengguna mengklik sebuah *link*, WebViewClient-lah yang pertama kali tahu dan bisa memutuskan apakah *link* tersebut akan dibuka di dalam WebView, di browser eksternal, atau dibatalkan sama sekali.
* **Menangani Error Halaman:** Ia bisa mendeteksi jika terjadi *error* saat memuat halaman (misalnya, halaman tidak ditemukan atau koneksi internet gagal).
* **Event Pemuatan Halaman:** Ia tahu kapan sebuah halaman mulai dimuat (onPageStarted) dan kapan selesai dimuat (onPageFinished), memberi kita kesempatan untuk menampilkan atau menyembunyikan indikator *loading*.

**Klien #2: WebChromeClient - Sang Pengawas 'Browser'**

WebChromeClient berbeda. Ia tidak peduli dengan konten halaman, tetapi dengan **elemen-elemen UI 'browser' di sekitarnya** (yang disebut "chrome"). Tugasnya meliputi:

* **Menangani Dialog JavaScript:** Ia mencegat panggilan alert(), confirm(), dan prompt() dari JavaScript, memberi kita kesempatan untuk menampilkannya sebagai dialog *native* Android yang lebih cantik.
* **Melaporkan Progres Pemuatan:** Ia bisa melaporkan progres pemuatan halaman (misalnya, 20%, 50%, 100%) yang bisa kita gunakan untuk memperbarui jProgressBar.
* **Menerima Ikon Halaman (Favicon) dan Judul Halaman.**
* Mengelola izin untuk fitur web canggih seperti Geolocation atau WebRTC.

**Langkah Implementasi Dasar**

Untuk menggunakan klien-klien ini, kita perlu membuat *instance*-nya dan menautkannya ke jWebView kita, biasanya di *event* OnCreate.

Delphi

// Di bagian private form, deklarasikan variabel untuk menampung klien

private

FWebViewClient: jWebViewClient;

FWebChromeClient: jWebChromeClient;

// Di dalam event OnCreate form Anda

procedure TAndroidModule1.jForm1Create(Sender: TObject);

begin

// ... (kode SetJavaScriptEnabled dan LoadUrl dari atas) ...

// Buat instance dan terapkan WebViewClient

FWebViewClient := tjWebViewClient.Create;

// Di subbab selanjutnya, kita akan meng-override metode di dalam FWebViewClient ini

jWebView1.SetWebViewClient(FWebViewClient);

// Buat instance dan terapkan WebChromeClient

FWebChromeClient := tjWebChromeClient.Create;

// Di subbab selanjutnya, kita juga akan meng-override metodenya

jWebView1.SetWebChromeClient(FWebChromeClient);

end;

Dengan menautkan klien-klien ini, jWebView kita sekarang tidak lagi berjalan 'sendirian'. Ia memiliki dua asisten yang siap menerima perintah kustom dari kita. Di subbab-subbab berikutnya, kita akan mulai 'mengajari' para asisten ini untuk melakukan tugas-tugas canggih yang menjembatani dunia Pascal dan JavaScript.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 18.2.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 18.2: Memanggil Pascal dari JavaScript (JavaScript Interface)**.

**18.2. Memanggil Pascal dari JavaScript (JavaScript Interface)**

Inilah inti dari aplikasi hibrida: kemampuan kode JavaScript yang berjalan di dalam WebView untuk 'menyeberang jembatan' dan memanggil prosedur atau fungsi di dalam kode Pascal kita. Mekanisme ini disebut **JavaScript Interface**. Dengan ini, kita bisa memberikan halaman web kita *superpower* untuk mengakses fungsionalitas *native*, seperti menampilkan notifikasi *toast*, menyimpan data ke SQLite, atau bahkan memicu getaran perangkat.

**Langkah 1: Membuat Kelas 'Jembatan' di Pascal**

Pertama, kita perlu membuat sebuah kelas di Pascal yang akan bertindak sebagai 'jembatan'. Hanya metode-metode di dalam kelas inilah yang bisa diakses oleh JavaScript.

Delphi

// Jangan lupa tambahkan unit-unit yang relevan

uses

..., Androidapi.JNI.Java.Interfaces, Androidapi.JNI.Os, Androidapi.Widget;

type

// Kelas ini akan menjadi jembatan kita

TWebAppInterface = class(jObject)

private

FContext: jContext;

public

constructor Create(AContext: jContext);

// Metode ini yang akan kita panggil dari JavaScript

// Anotasi [IsJavascriptInterface] adalah WAJIB untuk keamanan

procedure showToast(toastMessage: jString); [IsJavascriptInterface];

end;

// ----- Implementasi -----

constructor TWebAppInterface.Create(AContext: jContext);

begin

inherited Create;

FContext := AContext; // Simpan konteks untuk digunakan nanti

end;

procedure TWebAppInterface.showToast(toastMessage: jString);

var

msg: string;

begin

// Ubah tipe jString dari JavaScript ke string Pascal

msg := JStringToString(toastMessage);

// Gunakan konteks untuk menampilkan Toast native Android

jToast.MakeText(FContext, StrToJCharSequence(msg), jToast.LENGTH\_SHORT).show;

end;

**Penjelasan Kode Penting:**

* **TWebAppInterface = class(jObject)**: Kelas jembatan kita harus mewarisi dari jObject agar bisa berinteraksi dengan dunia Java.
* **[IsJavascriptInterface]**: Ini adalah **anotasi yang wajib** ada. Anotasi ini memberitahu sistem Android bahwa metode showToast ini aman dan secara eksplisit diizinkan untuk dipanggil dari JavaScript. Tanpa ini, panggilan tidak akan berhasil di versi Android modern karena alasan keamanan.
* **toastMessage: jString**: Perhatikan bahwa parameter yang diterima dari JavaScript akan berupa tipe data JNI, seperti jString. Kita perlu mengubahnya ke string Pascal menggunakan fungsi *helper* JStringToString.

**Langkah 2: Menambahkan Interface ke WebView**

Setelah kelas jembatan siap, kita perlu membuat *instance*-nya dan 'menyuntikkannya' ke dalam WebView dengan sebuah nama panggilan. Lakukan ini di *event* OnCreate *form* Anda.

Delphi

// di dalam OnCreate form...

var

jsInterface: TWebAppInterface;

begin

// ... (kode setup WebView dari subbab sebelumnya) ...

// Buat instance dari kelas jembatan kita

jsInterface := TWebAppInterface.Create(Self);

// Tambahkan interface ke WebView dengan nama 'AndroidBridge'

// Nama ini yang akan digunakan oleh JavaScript untuk memanggilnya

jWebView1.AddJavascriptInterface(jsInterface, 'AndroidBridge');

end;

Baris AddJavascriptInterface memberitahu WebView: "Sekarang, di dalam lingkungan JavaScript-mu, ada sebuah objek global baru bernama AndroidBridge. Objek ini memiliki semua metode yang ditandai dengan [IsJavascriptInterface] dari objek jsInterface."

**Langkah 3: Memanggil dari JavaScript**

Langkah terakhir adalah memanggil metode Pascal kita dari dalam file HTML yang dimuat oleh WebView. Di dalam file halaman\_interaktif.html (yang ada di folder assets), tambahkan sebuah tombol:

HTML

<!DOCTYPE html>

<html>

<body>

<h1>Komunikasi Web ke Native</h1>

<p>Klik tombol di bawah untuk memanggil kode Pascal dan menampilkan Toast native.</p>

<input type="button" value="Tampilkan Toast dari Pascal!"

onclick="panggilPascal()" />

<script>

function panggilPascal() {

// Panggil metode 'showToast' pada objek 'AndroidBridge'

AndroidBridge.showToast('Halo, ini pesan dari Dunia Web!');

}

</script>

</body>

</html>

**Penjelasan Kode:** Saat tombol HTML diklik, ia memanggil fungsi JavaScript panggilPascal(). Fungsi ini kemudian memanggil AndroidBridge.showToast(...). AndroidBridge adalah nama yang kita daftarkan di langkah 2, dan showToast adalah nama metode Pascal kita. *String* 'Halo, ini pesan dari Dunia Web!' akan dikirim sebagai parameter toastMessage.

Jalankan aplikasi Anda. Saat Anda menekan tombol di dalam halaman web, sebuah notifikasi *Toast* *native* Android akan muncul di bagian bawah layar. Anda telah berhasil menjembatani dua dunia yang berbeda!

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 18.3.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 18.3: Memanggil JavaScript dari Pascal**.

**18.3. Memanggil JavaScript dari Pascal**

Komunikasi dua arah berarti kita juga harus bisa memberi perintah dari kode Pascal ke JavaScript. Ini sangat berguna untuk memperbarui tampilan halaman web secara dinamis berdasarkan aksi di aplikasi *native* atau untuk mengambil data dari halaman web.

Metode modern dan yang paling direkomendasikan untuk melakukan ini adalah dengan menggunakan evaluateJavascript. Metode ini aman dan bekerja secara asinkron, sehingga tidak akan memblokir aplikasi Anda.

**1. Menjalankan Kode JavaScript Sederhana**

Penggunaan paling dasar adalah untuk mengeksekusi sebaris kode JavaScript untuk memanipulasi DOM (*Document Object Model*) dari halaman web yang sedang dimuat.

* **Contoh Praktis:** Mari kita buat sebuah tombol Pascal (jButton) yang saat diklik akan mengubah warna latar belakang halaman web yang sedang ditampilkan di jWebView.

Delphi

procedure TAndroidModule1.btnUbahWarnaWebClick(Sender: TObject);

var

jsCode: string;

begin

// Siapkan string kode JavaScript yang ingin dieksekusi

jsCode := 'document.body.style.backgroundColor = "lightblue";';

// Panggil evaluateJavascript. Parameter kedua (callback) kita isi nil

// karena kita tidak butuh nilai kembalian apa pun.

jWebView1.EvaluateJavascript(jsCode, nil);

end;

Saat tombol ini diklik, kode Pascal akan mengirimkan perintah JavaScript ke WebView. WebView akan menjalankan perintah tersebut, dan warna latar belakang halaman web akan langsung berubah menjadi biru muda. Perhatikan betapa mudahnya kode *native* kita memanipulasi tampilan web.

**2. Memanggil Fungsi JavaScript dan Menerima Hasilnya**

Bagaimana jika kita ingin mengambil sebuah nilai dari halaman web? Misalnya, mengambil isi dari sebuah elemen HTML atau hasil dari sebuah kalkulasi JavaScript. Metode evaluateJavascript juga bisa melakukan ini dengan menyediakan sebuah *callback*.

* **Langkah 1: Siapkan Fungsi di JavaScript** Pertama, di dalam file halaman\_interaktif.html, kita buat sebuah fungsi JavaScript yang mengembalikan sebuah nilai.

HTML

<p>Halo, <span id="namaUser">Budi</span>!</p>

<script>

function getInfoPengguna() {

var nama = document.getElementById('namaUser').innerText;

return "Nama pengguna di halaman web adalah: " + nama;

}

</script>

* **Langkah 2: Panggil dari Pascal dan Tangani Callback** Sekarang, kita buat tombol Pascal yang akan memanggil fungsi getInfoPengguna() dan menampilkan hasilnya menggunakan ShowMessage.

Delphi

// Jangan lupa tambahkan Androidapi.JNI.Webkit ke uses clause

uses ..., Androidapi.JNI.Webkit;

procedure TAndroidModule1.btnAmbilDataWebClick(Sender: TObject);

var

jsCode: string;

valueCallback: jValueCallback;

begin

jsCode := 'getInfoPengguna();'; // Nama fungsi JavaScript yang akan kita panggil

// Buat implementasi untuk ValueCallback untuk menerima hasilnya

valueCallback := tjValueCallback.Create;

valueCallback.onReceiveValue := procedure(value: JObject)

begin

// Kode ini akan berjalan SETELAH JavaScript selesai dan mengembalikan nilai.

// 'value' adalah hasil dari fungsi JS, dalam format jString.

ShowMessage('Hasil dari JavaScript: ' + JStringToString(jString(value)));

end;

// Jalankan JavaScript dan sediakan callback untuk menangani hasilnya

jWebView1.EvaluateJavascript(jsCode, valueCallback);

end;

**Penjelasan Kode:**

* **jValueCallback**: Ini adalah *interface callback* yang kita gunakan untuk menerima hasil. Kita membuat *instance*-nya dan mengimplementasikan metode onReceiveValue menggunakan prosedur anonim.
* **procedure(value: JObject)**: Prosedur ini akan dieksekusi secara asinkron saat JavaScript mengembalikan hasilnya. Parameter value berisi hasil tersebut (dalam contoh ini, sebuah jString yang perlu di-cast dan diubah ke *string* Pascal).

Dengan evaluateJavascript, jembatan komunikasi kita kini lengkap. Aplikasi Pascal kita tidak hanya bisa 'diperintah' oleh JavaScript, tetapi juga bisa 'memberi perintah' dan bahkan 'bertanya' kepada halaman web. Ini membuka potensi luar biasa untuk aplikasi hibrida yang dinamis dan terintegrasi.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 18.4.

YAHOX

Tampilkan alur berpikir

Baik, kita lanjutkan.

Berikut adalah jawaban untuk prompt **Subbab 18.4: Mengontrol Navigasi dengan WebViewClient**.

**18.4. Mengontrol Navigasi dengan WebViewClient**

Secara *default*, setiap kali pengguna mengklik sebuah *link* di dalam jWebView, halaman baru akan dimuat di dalam jWebView itu juga. Ini mungkin perilaku yang kita inginkan untuk *link* internal, tetapi sering kali tidak untuk *link* eksternal. Bayangkan sebuah artikel berita di aplikasi Anda memiliki *link* ke Twitter atau Instagram. Tentu akan lebih baik jika *link* tersebut dibuka di aplikasi Twitter/Instagram yang sesuai, bukan 'terjebak' di dalam WebView kita.

Di sinilah WebViewClient berperan. Dengan 'mengajari' asisten ini, kita bisa mencegat setiap permintaan navigasi dan memutuskan apa yang harus dilakukan.

**Metode Kunci: shouldOverrideUrlLoading**

Metode utama yang akan kita gunakan adalah shouldOverrideUrlLoading. Metode ini akan dipanggil oleh sistem **setiap kali** sebuah *link* akan dimuat di WebView, memberi kita kesempatan untuk mengambil alih.

Metode ini harus mengembalikan sebuah nilai boolean:

* **Return True**: Artinya, "Berhenti! Jangan lanjutkan memuat URL ini di WebView. Saya (kode Pascal) sudah menanganinya sendiri."
* **Return False**: Artinya, "Silakan lanjutkan. Biarkan WebView memuat URL ini seperti biasa."

**Langkah Implementasi**

Kita akan membuat kelas turunan dari jWebViewClient dan mengimplementasikan logika kita di sana.

**1. Buat Kelas WebViewClient Kustom** Di unit yang sama dengan form Anda (atau di unit terpisah), definisikan kelas baru:

Delphi

type

TMyWebViewClient = class(jWebViewClient)

public

function shouldOverrideUrlLoading(view: jWebView; request: jWebResourceRequest): boolean; override;

end;

**2. Implementasikan Logika Navigasi** Di dalam metode shouldOverrideUrlLoading, kita akan memeriksa URL yang diminta dan membuat keputusan.

Delphi

function TMyWebViewClient.shouldOverrideUrlLoading(view: jWebView; request: jWebResourceRequest): boolean;

var

url: string;

intent: jIntent;

begin

// Dapatkan URL yang akan dimuat dalam bentuk string Pascal

url := JStringToString(request.getUrl.toString);

// LOGIKA 1: Jika link adalah domain internal aplikasi kita, biarkan WebView yang buka.

if url.Contains('my-app-domain.com') then

begin

Result := False; // Beritahu sistem: "Lanjutkan pemuatan di WebView"

Exit;

end;

// LOGIKA 2: Jika link adalah nomor telepon, buka aplikasi dialer.

if url.StartsWith('tel:') then

begin

intent := tjIntent.Create(jIntent\_ACTION\_DIAL, tjUri.parse(url));

GetMainActivity.startActivity(intent);

Result := True; // Beritahu sistem: "Tugas sudah saya ambil alih"

Exit;

end;

// LOGIKA 3: Untuk semua link lainnya (misal: youtube.com, twitter, dll), buka di browser eksternal.

try

intent := tjIntent.Create(jIntent\_ACTION\_VIEW, tjUri.parse(url));

GetMainActivity.startActivity(intent);

except

// Tangani jika tidak ada browser terinstal (kasus yang sangat jarang)

on E: Exception do Log.e('WebViewNav', 'Gagal membuka link eksternal: ' + E.Message);

end;

Result := True; // Beritahu sistem: "Tugas sudah saya ambil alih"

end;

**3. Terapkan Klien Kustom ke WebView** Terakhir, jangan lupa untuk membuat *instance* dari klien kustom kita dan menerapkannya ke jWebView di OnCreate *form*, menggantikan tjWebViewClient.Create yang kita buat di subbab sebelumnya.

Delphi

// Di dalam event OnCreate form Anda...

// Ganti baris lama: FWebViewClient := tjWebViewClient.Create;

// Dengan yang baru:

FWebViewClient := TMyWebViewClient.Create;

jWebView1.SetWebViewClient(FWebViewClient);

Sekarang, muat sebuah halaman web di aplikasi Anda yang berisi berbagai macam *link*. *Link* yang mengarah ke domain internal Anda akan tetap berada di dalam aplikasi, sementara *link* ke situs eksternal akan secara otomatis membuka browser Chrome (atau browser *default* pengguna), memberikan pengalaman navigasi yang jauh lebih alami dan sesuai harapan pengguna.

Saya akan menunggu perintah "yahok" dari Anda sebelum melanjutkan ke prompt Subbab 18.5.

YAHOX

Tampilkan alur berpikir

Baik, ini adalah subbab terakhir dari pembahasan khusus kita.

Berikut adalah jawaban untuk prompt **Subbab 18.5: Meningkatkan Pengalaman UI dengan WebChromeClient**.

**18.5. Meningkatkan Pengalaman UI dengan WebChromeClient**

Kita telah berhasil mengontrol alur konten dengan WebViewClient. Sekarang, mari kita poles pengalaman pengguna (*User Experience*) dengan mengontrol elemen-elemen 'krom' atau UI dari *browser* itu sendiri. WebChromeClient memungkinkan kita untuk mencegat *event-event* UI dari JavaScript dan menggantinya dengan komponen *native* yang lebih indah dan konsisten dengan desain aplikasi kita.

**Studi Kasus: Mengganti Dialog alert() Bawaan**

* **Masalahnya:** Jika Anda menjalankan fungsi alert('Halo!'); dari JavaScript di dalam WebView, akan muncul sebuah dialog *popup* yang terlihat sangat standar, terkadang kuno, dan gayanya berbeda dari desain aplikasi Android modern. Ini bisa merusak kesan 'native' dari aplikasi hibrida kita.
* **Solusinya:** Kita akan mencegat panggilan alert() tersebut dan menampilkannya menggunakan ShowMessage dari LAMW yang terlihat jauh lebih menyatu dengan aplikasi.

**Langkah Implementasi**

1. **Buat Kelas WebChromeClient Kustom** Sama seperti sebelumnya, kita buat kelas turunan dari jWebChromeClient di dalam unit kita.

Delphi

type

TMyWebChromeClient = class(jWebChromeClient)

public

// Override metode onJsAlert

function onJsAlert(view: jWebView; url: jString; message: jString;

result: jJsResult): boolean; override;

end;

1. **Implementasikan Logika onJsAlert** Di dalam metode onJsAlert, kita akan mengambil pesan dari JavaScript, menampilkannya dengan ShowMessage, lalu memberitahu sistem bahwa kita sudah menanganinya.

Delphi

function TMyWebChromeClient.onJsAlert(view: jWebView; url: jString;

message: jString; result: jJsResult): boolean;

var

msg: string;

begin

// 1. Ambil pesan dari parameter dan ubah ke string Pascal

msg := JStringToString(message);

// 2. Tampilkan pesan menggunakan dialog native Android

ShowMessage(msg);

// 3. Konfirmasi ke JsResult bahwa dialog sudah ditutup (pengguna menekan OK)

result.confirm;

// 4. Return True untuk memberitahu WebView bahwa kita sudah menangani alert ini.

// WebView tidak perlu lagi menampilkan dialog alert bawaannya.

Result := True;

end;

1. **Terapkan Klien ke WebView** Terakhir, di OnCreate *form*, ganti pembuatan WebChromeClient standar dengan kelas kustom kita.

Delphi

// di dalam event OnCreate form Anda...

// Ganti baris lama: FWebChromeClient := tjWebChromeClient.Create;

// Dengan yang baru:

FWebChromeClient := TMyWebChromeClient.Create;

jWebView1.SetWebChromeClient(FWebChromeClient);

Sekarang, setiap kali kode JavaScript di dalam WebView memanggil alert(), aplikasi Anda akan menampilkan dialog ShowMessage yang rapi.

**Fitur Bermanfaat Lainnya dari WebChromeClient**

WebChromeClient bisa melakukan lebih banyak hal untuk memoles aplikasi Anda. Salah satu yang paling berguna adalah:

* **Menampilkan Progres Pemuatan Halaman:** Dengan meng-*override* metode onProgressChanged(view: jWebView; newProgress: integer), Anda bisa mendapatkan progres pemuatan halaman dari 0 hingga 100. Anda bisa menggunakan nilai newProgress ini untuk memperbarui properti Progress dari sebuah jProgressBar *native*. Ini memberikan umpan balik visual yang jelas kepada pengguna saat halaman yang berat sedang dimuat, jauh lebih baik daripada hanya menampilkan layar putih yang kosong.

**Kesimpulan Bab**

Dengan kemampuan untuk menjembatani Pascal dan JavaScript, mengontrol navigasi, dan memoles UI, Anda kini memiliki semua alat yang diperlukan untuk membangun aplikasi hibrida yang canggih. Anda bisa memilih bagian mana dari aplikasi Anda yang paling baik dibuat dengan teknologi *native* LAMW, dan bagian mana yang lebih efisien dikembangkan dengan teknologi web, lalu menggabungkan keduanya menjadi satu kesatuan yang mulus dan profesional.