Rekayasa Lintas Platform (RLP)

Pertemuan 1 : Pengantar Rekayasa Lintas Platform

Alauddin Maulana Hirzan

Fakultas Teknologi Informasi dan Komunikasi
Universitas Semarang

Outline

Pengantar Rekayasa Lintas Platform

Lastly

Objektif Pembelajaran

Setelah menyelesaikan pertemuan ini, mahasiswa diharapkan mampu:

- Menjelaskan konsep dasar rekayasa lintas platform (cross-platform engineering).
- Membedakan pendekatan pengembangan Native, Hybrid, dan Cross-Platform.
- Menganalisis implikasi teknis dan trade-off dari masing-masing pendekatan arsitektur.
- Memahami karakteristik ekosistem multi-device dalam pengembangan aplikasi modern.

Persyaratan

- Pemahaman dasar tentang pemrograman berorientasi objek (OOP).
- Pengetahuan dasar mengenai pengembangan web (HTML, CSS, JavaScript) atau pengembangan mobile dasar.
- Konsep dasar mengenai client-server architecture.

Motivasi

Saat ini, pengguna berinteraksi dengan layanan digital melalui berbagai perangkat: smartphone (iOS & Android), tablet, laptop/desktop (Web), hingga perangkat smart home (IoT). Mengembangkan perangkat lunak secara terpisah untuk setiap perangkat (Native) membutuhkan biaya, waktu, dan sumber daya manusia yang sangat besar. Rekayasa lintas platform lahir dari kebutuhan industri untuk merilis produk secara cepat ke berbagai ekosistem sekaligus mempertahankan konsistensi antarmuka, logika bisnis, dan efisiensi pengembangan.

Kata Kunci

- Cross-Platform: Pendekatan pengembangan perangkat lunak di mana satu basis kode (codebase) dapat dijalankan di beberapa sistem operasi.
- Native: Pengembangan aplikasi menggunakan bahasa dan tools spesifik yang disediakan oleh pembuat OS (misal: Swift untuk iOS, Kotlin untuk Android).
- Hybrid: Aplikasi web (HTML, CSS, JS) yang dibungkus dalam container native (WebView) agar dapat diinstal seperti aplikasi mobile biasa.
- Trade-off: Kompromi atau pertukaran keuntungan dan kerugian saat memilih satu arsitektur arsitektur dibandingkan yang lain (misal: performa vs kecepatan pengembangan).
- Multi-device: Skenario di mana layanan dikonsumsi di berbagai jenis perangkat keras dengan ukuran layar dan kemampuan komputasi yang berbeda.

Konsep Dasar

Rekayasa lintas platform berfokus pada bagaimana kita merancang, mengimplementasikan, dan mendistribusikan solusi teknologi informasi yang dapat berjalan mulus di atas lingkungan sistem yang heterogen. Ini bukan sekadar memindahkan antarmuka web ke mobile, melainkan tentang merancang arsitektur terintegrasi, modularisasi codebase, dan memastikan pengalaman pengguna (UX) yang konsisten di mana pun aplikasi itu dibuka.

Definisi Lintas Platform

Rekayasa lintas platform adalah disiplin rekayasa perangkat lunak yang berupaya meminimalkan duplikasi kode dan usaha pengembangan saat menargetkan lebih dari satu platform (seperti iOS, Android, Web, dan Desktop). Pendekatan ini memanfaatkan framework atau compiler khusus yang menerjemahkan atau menjembatani kode umum ke komponen asli spesifik platform.

Karakteristik Multi-Device

Aplikasi modern harus adaptif. Karakteristik utama dari ekosistem multi-device meliputi:

- Responsivitas Layar: UI harus beradaptasi dari layar smartwatch berukuran 1 inci hingga monitor desktop 32 inci.
- Konteks Interaksi: Pengguna berinteraksi dengan sentuhan (touch) di mobile, mouse/keyboard di desktop, dan voice di perangkat IoT.
- Manajemen Daya dan Jaringan: Perangkat mobile seringkali memiliki bandwidth terbatas dan bergantung pada baterai, berbeda dengan desktop.

Trade-off Arsitektur

Setiap keputusan arsitektur memiliki kompromi:

- Native: Performa terbaik, akses API perangkat keras penuh, namun biaya dan waktu pengembangan paling tinggi (harus mempekerjakan tim Android dan iOS secara terpisah).
- Hybrid: Pengembangan paling cepat (berbasis web), namun performa rendering sering tertinggal dan pengalaman pengguna kadang terasa kaku (clunky).
- Cross-Platform (Modern): Berada di tengah-tengah. Performa mendekati native, 70-90% kode dapat dibagikan, namun ukuran aplikasi (bundle size) cenderung lebih besar dan terkadang butuh waktu menunggu pembaruan framework jika OS merilis fitur baru.

Cara Kerja

Step 1: Penulisan Kode Sentral

Pengembang menulis logika bisnis dan desain antarmuka menggunakan bahasa pemrograman perantara (seperti Dart, JavaScript/TypeScript, atau C*).

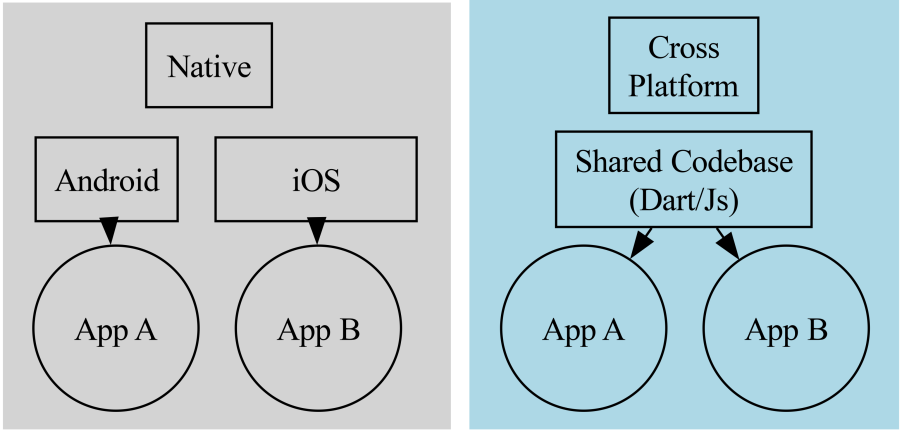
Step 2: Kompilasi / Bridging

Framework (misalnya Flutter atau React Native) memproses kode tersebut. Flutter akan mengompilasi kode menjadi instruksi mesin native (ARM/x86)

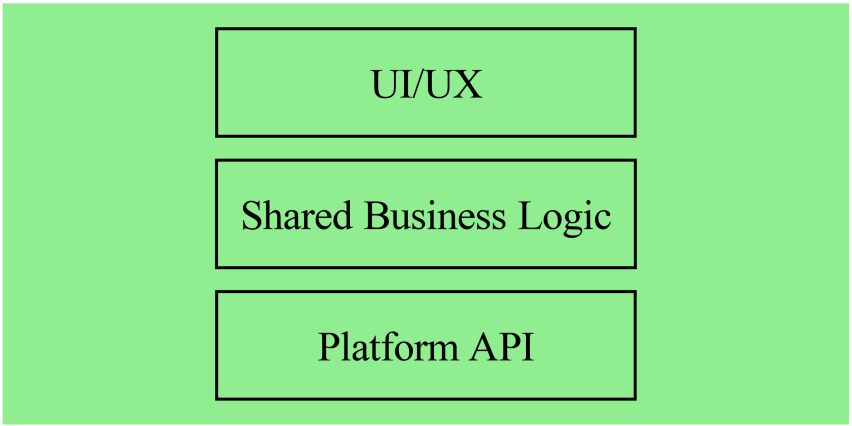
Step 3: Distribusi

Aplikasi yang dihasilkan dipaketkan menjadi file .apk*/*aab untuk Android, .ipa untuk iOS, dan file bundle untuk Web, siap didistribusikan ke masing-masing app store atau server.

Ilustrasi Cara Kerja



Ilustrasi Cara Kerja



Contoh

Misalkan sebuah startup ingin membuat aplikasi absensi karyawan.

- Jika menggunakan Native, mereka butuh 2 bulan untuk versi Android dan 2 bulan untuk iOS. Total biaya tinggi.
- Jika menggunakan Cross-platform (Flutter), mereka menulis logika "Ambil Lokasi GPS & Foto" sekali dalam bahasa Dart. Waktu pengembangan bisa ditekan menjadi 2.5 bulan untuk kedua platform sekaligus.

Detail Teknis

- Rendering Engine: Pendekatan modern seperti Flutter tidak bergantung pada UI bawaan OS (OEM widgets). Flutter menggunakan engine Skia (atau Impeller) yang menggambar setiap piksel sendiri dari kanvas kosong, memastikan desain terlihat persis 100% sama di semua OS.
- JavaScript Bridge: React Native menjalankan logika JS di thread terpisah dan mengirim pesan berformat JSON secara asinkron ke native thread untuk meminta OS menggambar elemen antarmuka.

Contoh Implementasi

Command untuk inisiasi proyek (Contoh Flutter):

```
bash
flutter create rekayasa_app
cd rekayasa_app
flutter run -d chrome  * Menjalankan di Web
flutter run -d android * Menjalankan di Emulator Android
```

Output:

Aplikasi yang sama dengan fungsi yang sama akan muncul di browser dan layar ponsel simulasi.

Studi Kasus

Industri: Airbnb

Airbnb awalnya mencoba pendekatan silang dengan React Native untuk menghemat waktu pengembangan. Namun, karena kompleksitas aplikasi mereka dan kebutuhan interaksi native yang sangat detail, mereka akhirnya memutuskan untuk kembali ke pengembangan Native setelah beberapa tahun.

Hal ini menjadi studi kasus penting mengenai trade-off bahwa meskipun menjanjikan, pendekatan lintas platform tidak selalu cocok untuk aplikasi dengan kompleksitas hardware/native yang sangat ekstrem.

Perbandingan

Fitur	Native	Hybrid (Cordova/Ionic)	Cross-Platform
Bahasa	Swift, Kotlin, Java	HTML, CSS, JS	Dart, JS/TS, C*
Akses Hardware	Langsung, Penuh	Terbatas (via plugin)	Tinggi
Performa UI	Sangat Cepat	Lambat (WebView)	Cepat
Code Sharing	Tidak Ada (~0%)	Sangat Tinggi (~95%)	Tinggi

Kelebihan

- Publikasi (Time-to-Market) lebih cepat: Fitur baru dapat dirilis secara bersamaan di iOS dan Android.
- Efisiensi Biaya: Hanya membutuhkan satu tim pengembang (stack) daripada dua atau tiga tim terpisah.
- Konsistensi Logika: Mengurangi bug karena logika bisnis hanya ditulis dan diuji di satu basis kode utama.

Batasan

- Performa *Overhead: Tidak peduli seberapa efisien, lapisan tambahan (compiler atau bridge) selalu menambah sedikit beban memori/CPU dibandingkan native murni.
- Ketergantungan Ekosistem Pihak Ketiga: Jika Apple/Google merilis fitur baru (misal: widget baru), Anda harus menunggu pembuat framework atau komunitas merilis plugin/update agar fitur itu bisa dipakai.

Common Misconceptions

- "Write once, run everywhere": Realitasnya seringkali adalah "Learn once, write anywhere". Anda tetap harus menyesuaikan beberapa detail UI/UX atau mengatur konfigurasi spesifik.
- Misalnya, perizinan kamera di ‘AndroidManifest.xml‘ untuk Android dan ‘Info.plist‘ untuk iOS.

Aplikasi Praktikal

- Aplikasi e-commerce, portal absensi, aplikasi pemesanan tiket, dashboard bisnis, dan aplikasi berita. Ini adalah skenario di mana konten data lebih dominan daripada animasi kompleks tingkat tinggi atau pemrosesan grafis 3D (seperti game AAA).

Lastly

Terima Kasih