Perbandingan Jetpack Compose dan Flutter dalam Pengembangan dan Penggunaan Resource pada Aplikasi Link D Law

Rasid[1], I Made Suartana[2]

[1,2] Program Studi S1 Teknik Informatika, Universitas Negeri Surabaya
[1]rasid.20085@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

***Abstrak*— Penelitian ini bertujuan untuk membandingkan kinerja framework Flutter dan Jetpack Compose dalam pengembangan aplikasi Link D Law, dengan fokus pada kecepatan pengembangan, kemudahan pembelajaran, efisiensi dan kemudahan perawatan kode, serta kinerja aplikasi terkait penggunaan memori, CPU, dan framerate. Hasil pengujian menunjukkan bahwa Jetpack Compose unggul dalam efisiensi pengembangan dan penggunaan memori. Pengembangan dengan Jetpack Compose rata-rata 12,23% lebih cepat dibandingkan dengan Flutter, terutama karena fleksibilitas dalam penyesuaian UI dan struktur kode yang lebih terorganisir. Meskipun demikian, Flutter lebih mudah dipelajari berkat dokumentasi yang lebih lengkap dan dukungan komunitas yang kuat. Dalam aspek penggunaan memori, Jetpack Compose menunjukkan efisiensi yang lebih tinggi dengan rata-rata penghematan sebesar 43,36% dibandingkan Flutter. Penggunaan CPU bervariasi tergantung pada fitur yang diuji, di mana Jetpack Compose lebih efisien pada fitur yang kompleks, sementara Flutter lebih unggul pada fitur yang lebih sederhana. Dalam aspek framerate, kedua framework menunjukkan hasil yang sebanding, meskipun Flutter sedikit lebih baik pada fitur dengan interaksi UI sederhana, sedangkan Jetpack Compose unggul pada fitur dengan tampilan yang lebih kompleks.**

Kata Kunci— **Flutter, Jetpack Compose, Perbandingan Framework, Pengembangan Aplikasi Mobile, Android.**

I. PENDAHULUAN

Dalam era digital saat ini, pengembangan aplikasi mobile telah menjadi salah satu bidang yang paling dinamis dan inovatif dalam teknologi informasi, didorong oleh kebutuhan untuk memenuhi ekspektasi pengguna yang terus berkembang untuk aplikasi yang efisien, responsif, dan mudah digunakan. Perangkat mobile telah menjadi kebutuhan utama dalam rutinitas harian miliaran orang di seluruh penjuru dunia[1]. Dengan skala penggunaan yang demikian luas, peningkatan efisiensi yang minimal pada aplikasi yang digunakan secara rutin dapat mengakumulasikan penghematan waktu yang signifikan bagi keseluruhan pengguna. Pembaruan kecil pada aplikasi dapat mengalihkan pengalaman pengguna dari kesan yang kurang responsif dan penuh gangguan menjadi interaksi yang lancar dan memuaskan. Dalam upaya meningkatkan performa aplikasi, antarmuka pengguna menjadi area penting untuk diperhatikan. Lebih lanjut, dalam konteks antarmuka pengguna, pemilihan framework dan toolkit berperan penting dalam menunjang kinerja aplikasi[2].

Proses pengembangan aplikasi mobile menjadi sangat penting dengan meningkatnya penggunaan perangkat mobile[3]. Sistem operasi Android dan IOS, yang merupakan sistem operasi mobile yang paling banyak dipilih, memiliki alat pengembangan dan bahasa pemrograman yang berbeda. Dalam ekosistem pengembangan aplikasi PlayStore, Kotlin sebagai bahasa pemrograman android native menempati peringkat 1 dengan persentase 71%, diikuti oleh Flutter menempati peringkat 2 dengan persentase 17% dan React Native menempati peringkat 3 dengan persentase 16% yang memiliki kemampuan pengembangan lintas platform. Jetpack Compose, meskipun baru telah menempati peringkat 6 dengan persentase 9%[4].

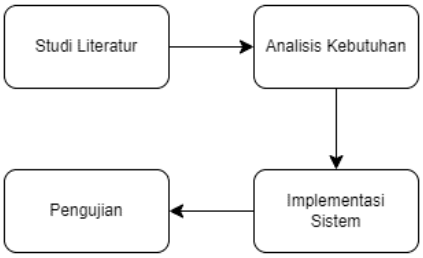
Flutter, dirilis oleh Google pada tahun 2018, adalah sebuah open-source user interface (UI) software development kit (SDK) yang memungkinkan pengembang untuk membangun aplikasi mobile cross-platform yang indah dan berkinerja tinggi dari satu basis kode[5]. Dengan menggunakan bahasa pemrograman Dart, Flutter menawarkan pendekatan yang unik dalam pengembangan aplikasi dengan fitur hot reload yang memungkinkan pengembang untuk melihat perubahan langsung dalam aplikasi tanpa perlu memulai ulang seluruh aplikasi[6]. Ini membantu mempercepat proses pengembangan dan memungkinkan pengembang untuk bereksperimen dengan UI dan fitur aplikasi secara real-time.

Sedangkan Jetpack Compose, diperkenalkan pada tahun 2019 dan mulai digunakan pada tahun 2021. Merupakan bagian dari keluarga Jetpack , sebuah kumpulan library yang dirancang untuk mempermudah pengembangan aplikasi Android. Sebagai toolkit modern untuk membangun UI native android, Jetpack Compose mengusung pendekatan deklaratif dalam mendefinisikan UI, memungkinkan kode yang lebih bersih dan lebih intuitif dibandingkan dengan pendekatan imperatif tradisional yang menggunakan XML[7]. Dengan Jetpack Compose, pengembang dapat membangun antarmuka pengguna dengan kode Kotlin yang lebih ringkas, meningkatkan produktivitas pengembangan dan memungkinkan pembuatan UI yang lebih konsisten dan mudah dipelihara[8].

Perbandingan antara Flutter dan Jetpack Compose menjadi penting karena setiap proyek pengembangan aplikasi memiliki serangkaian kebutuhan dan kriteria yang harus dipenuhi. Untuk aplikasi seperti Link D Law, yang menuntut keandalan tinggi dan pengalaman pengguna yang lancar, pemilihan framework bukan hanya tentang preferensi teknologi, melainkan tentang memaksimalkan efisiensi operasional dan meminimalkan biaya pemeliharaan. Dengan membandingkan kedua teknologi ini, penelitian ini bertujuan untuk mengidentifikasi framework mana yang menawarkan performa terbaik dalam penggunaan resource seperti memory, CPU, dan storage, serta mengevaluasi efisiensi mereka dalam proses pengembangan—dari kecepatan pembangunan fitur, efesiensi kode hingga kemudahan pemeliharaan dan skalabilitas aplikasi.

II. METODE PENELITIAN

Penelitian ini menggunakan metode yang terdiri dari beberapa tahapan utama seperti yang terlihat pada gambar 1. Dimulai dengan studi literatur yang berfungsi sebagai dasar teori, diikuti dengan analisis kebutuhan untuk mengidentifikasi elemen-elemen yang diperlukan dalam sistem. Setelah itu, implementasi sistem dilakukan berdasarkan hasil analisis kebutuhan. Tahap terakhir adalah pengujian untuk mengevaluasi kinerja sistem secara menyeluruh.



Gbr. 1 Diagram Alur Tahapan Penelitian

Dari gambar di atas dapat diketahui tahapan penelitian yang akan diterapkan pada penelitian ini, yaitu:

A. *Studi Literatur*

Pada tahapan studi literatur, peneliti ini akan mengumpulkan berbagai literatur yang relevan untuk mendukung dan menjadi referensi. Literatur yang akan digunakan berasal dari beragam sumber, termasuk buku, artikel, serta jurnal baik dari dalam negeri maupun luar negeri. Adapun topik yang dijadikan acuan meliputi Native Android, Flutter, Studi Komparatif, Jetpack Compose, dan Cross-platform.

B. *Analisis Kebutuhan*

Analisis kebutuhan dilakukan untuk mendukung proses pengembangan aplikasi dan proses perbandingan dalam penelitian ini. Komponen yang telah dianalisa dan digunakan dalam penelitian ini meliputi perangkat keras (*hardware)* dan perangkat lunak (*software*). Komponen tersebut adalah sebagai berikut:

1. *Hardware*
   a. Processor AMD Ryzen 5
   b. Random Access Memory 16 GB
   c. Kapasitas SSD 512GB dan 1 TB

2. *Software*
   a. Sistem Operasi Windows 11
   b. Android Studio
   c. Visual Studio Code
   d. Emulator Android
   e. Postman
   f. Figma
   g. Bahasa Pemrograman Kotlin
   h. Bahasa Pemrograman Dart

C. *Implementasi Sistem*

Pada tahap ini implementasi pengembangan aplikasi Link D Law akan menggunakan framework Jetpack Compose dan Flutter. Berikut adalah langkah-langkah proses implementasi jika menggunakann masing-masing framework:

1. *Jetpack Compose*
   a. Pemilihan Library: Identifikasi library Jetpack Compose yang dibutuhkan untuk memaksimalkan proses pengembangan aplikasi.
   b. Pengembangan UI: Implementasikan antarmuka pengguna menggunakan komponen UI Jetpack Compose berupa fungsi-fungsi seperti Button, Text, Image, Column, Row, Lazylist, dan lain-lain.
   c. Pengelolaan State: Menerapkan State dan ViewModel untuk mengelola state aplikasi dan mengikatnya ke UI. Gunakan fungsi-fungsi seperti *remember* dan *mutableStateOf* untuk mengelola state.
   d. Navigasi: Impelementasi navigasi antar halaman layer menggunakan NavHost dan NavController untuk menentukan serta menanggapi perubahan rute navigasi.

2. *Flutter*
   a. Pemilihan Library: Identifikasi library Flutter yang dibutuhkan untuk memaksimalkan proses pengembangan aplikasi.
   b. Pengembangan UI: Implementasi antarmuka pengguna menggunakan widget-widget Flutter seperti TextField, Button, ListView, dan lain-lain. Struktur antarmuka dibuat dengan menempatkan widget-widget dalam hirarki yang tepat.

c. Pengelolaan State: Menggunakan *setState*, *Provider* atau paket-paket manajemen state lainnya untuk mengelola state aplikasi.
d. Navigasi: Implementasi navigasi antar halaman layer menggunakan widget-widget seperti Navigator dan MaterialPageRoute. Penentuan dan navigasi rute antar layer disesuaikan dengan kebutuhan aplikasi.

D. *Pengujian*

Penelitian ini bertujuan untuk membandingkan Flutter sebagai framework Cross-platform dan Jetpack Compose sebagai framework Native Android. Masing-masing aplikasi yang telah dikembangkan menggunakan kedua framework akan dilakukan 2 pengujian, yaitu:

1. *Pengujian Pengembangan (Development Testing)*
   Pengujian ini akan berjalan bersamaan dengan proses pengembangan aplikasi. Beberapa aspek yang akan diuji pada pengujian ini adalah:
   a. Kecepatan Pengembangan: Mengukur waktu yang diperlukan untuk mengembangkan fitur tertentu dalam aplikasi menggunakan kedua framework.
   b. Keunggulan Pembelajaran: Menilai seberapa mudah bagi pengembang untuk mempelajari dan menggunakan kedua framework.
   c. Keunggulan dan Efesiensi Kode: Menilai kemudahan dalam membaca dan mengevaluasi efesiensi kode yang diketik dari kedua framework.
   d. Keunggulan Perawatan Kode: Mengevaluasi seberapa mudah melakukan pembaruan atau perubahan pada kode aplikasi.

2. *Pengujian Kinerja (Performance Testing)*
   Pengujian ini akan menggunakan App Profiler dari Android Studio untuk mengukur kinerja dari fitur aplikasi yang dikembangkan menggunakan framework Flutter dan Jetpack Compose. Beberapa aspek yang akan diuji pada pengujian ini adalah:
   a. Penggunaan Memori: Mengukur total penggunaan memori yang digunakan aplikasi dari kedua framework.
   b. Penggunaan CPU: Mengamati efesiensi penggunaan CPU pada perangkat.
   c. Framerate aplikasi: Memastikan aplikasi berjalan lancar tanpa jeda atau lag, terutama saat melakukan transisi atau animasi.

III. HASIL DAN PEMBAHASAN

Bab ini membahas mengenai implementasi sistem dari rancangan yang telah dibuat serta proses pengujian yang dilakukan sesuai dengan perencanaan sebelumnya.

A. *Analisis Kebutuhan*

Tahap ini menjelaskan bagaimana analisis kebutuhan dilakukan dan bagaimana hal ini mempengaruhi proses pengembangan serta hasil penelitian.

1. *Kaitan Kebutuhan dengan Framework*

   Analisis kebutuhan ini membantu dalam menentukan framework yang paling sesuai untuk pengembangan aplikasi Link D Law. Flutter dan Jetpack Compose masing-masing memiliki kelebihan dan kekurangan dalam memenuhi kebutuhan tersebut.

   a. Flutter

      Menawarkan pengembangan lintas platform dengan satu basis kode. Cocok untuk pengembangan cepat dan iterasi, namun mungkin memerlukan optimasi tambahan untuk kinerja pada perangkat Android.

   b. Jetpack Compose

      Menawarkan kinerja optimal dan integrasi yang kuat dengan ekosistem Android. Cocok untuk aplikasi yang membutuhkan kinerja tinggi dan stabilitas.

B. *Implementasi Sistem*

Tahap ini menjelaskan secara rinci langkah-langkah yang diambil untuk mengimplementasikan sistem menggunakan framwork Flutter dan Jetpack Compose. Implementasi mencakup pemilihan library, pengembangan UI, pengelolaan state, dan navigasi.

1. *Implementasi dengan Jetpack Compose*

   a. Pemilihan Library

      Pada tahap awal implementasi, dilakukan pemililhan library yang akan digunakan untuk memaksimalkan proses pengembangan aplikasi. Beberapa library yang dipilih adalah Compose Destination, ViewModel, LiveData, Retrofit, Room, Koin, Datastore, Paging3, dan Bouquet.

   b. Pengembangan UI

      Dapat dilihat pada gambar 2 dan 3 yang menjelaskan implementai halaman Login menggunakan Jetpack Compose.

```kotlin
MainActivity
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      WindowCompat.setDecorFitsSystemWindows(window, false)
      setContent {
          DlawTheme {
              Surface(
                  modifier = Modifier.fillMaxSize(),
                  color = MaterialTheme.colorScheme.background
              ) {
                  LoginScreen()
              }
          }
      }
  }
}
```

Gbr. 2 Memanggil LoginScreen di MainActivity

```kotlin
LoginScreen
@Composable
fun LoginScreen() {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center,
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Image(painter = painterResource(id = R.drawable.logo), contentDescription = null)
        Text(text = "Masuk", fontSize = 24.sp, fontWeight = FontWeight.Bold)
        TextField(value = "", onValueChange = {}, placeholder = { Text(text = "Email") })
        TextField(
            value = "",
            onValueChange = {},
            placeholder = { Text(text = "Password") },
            visualTransformation = PasswordVisualTransformation()
        )
        Button(onClick = {}) {
            Text(text = "Masuk")
        }
    }
}
```

Gbr. 3 Tampilan Kode Layout LoginScreen Dengan Jetpack Compose

(1) SetContent

Jetpack Compose menggunakan setContent di dalam activity untuk mendefinisikan layout.

(2) Composable Funstions

- Semua elemen di Compose adalah sebuah fungsi yang diberi anotasi @Composable.
- Composable functions digunakan untuk membuat elemen UI. Misalnya, Text, Button, Column, dan lain-lain.

(3) Hirarki Composable

- Jetpack Compose juga menggunakan struktur hierarki yang berbasis tree. Namun, tidak seperti widget, ini menggunakan composable functions.
- Sebagai contoh, Column adalah composable function yang dapat memiliki children composables untuk mengatur layout secara vertikal.

(4) Deklarasi Composable

Composable dideklarasikan sebagai fungsi. Misalnya, TextField, Image, Text, Button, dan lain-lain.

(5) Styling dan Theming

Styling dilakukan melalui parameter fungsi composable itu sendiri. Misalnya, TextStyle untuk teks, Modifier untuk layout dan dekorasi, dan lain-lain.

c. Pengelolaan State

Jetpack Compose menggunakan pendekatan deklaratif dengan prinsip Recomposition. Recomposition adalah proses di mana Jetpack Compose memperbarui UI ketika state yang digunakan oleh Composable berubah. Proses pengelolaan state dan pembaruan UI di Jetpack Compose:

(1) Deklaratif UI

Pendefinsian UI dalam bentuk fungsi dengan anotasi @Composable untuk menggambarkan bagaimana UI harus terlihat berdasarkan state saat ini.

(2) State-Driven

UI di Compose dikendalikan oleh state. Ketika state berubah, Compose akan secara otomatis memicu recomposition dari fungsi @Composable yang menggunakan state tersebut.

(3) Fine-gained Recomposition

Recomposition di Compose sangat efisien karena hanya composable yang state-nya berubah yang akan di-recompose. Ini berarti hanya bagian dari UI tree yang terpengaruh perubahan state yang akan di-render ulang, bukan seluruh UI tree.

(4) Skipping Recomposition

Compose secara otomatis menentukan bagian mana dari UI tree yang perlu di-recompose. Jika bagian tertentu tidak terpengaruh oleh perubahan state, Compose akan melewatinya dan tidak melakukan re-render.

Seperti contoh pada gambar 4, variabel name diubah statenya dari “Compose” menjadi “Jetpack” sehingga fungsi yang akan direkomposisi hanya Greeting dan tidak semua layout UI.

Contoh Pengelolaan State Kotlin

```kotlin
@Composable
fun Greeting(name: String) {
    Text(text = "Hello, $name!")
}

@Composable
fun MyScreen() {
    var name by remember { mutableStateOf("Compose") }

    Column {
        Greeting(name)
        Button(onClick = { name = "Jetpack" }) {
            Text("Change Name")
        }
    }
}
```

Gbr. 4 Contoh Manajemen State di Jetpack Compose

d. Navigasi

Navigasi antar layer dilakukan dengan Compose Destination. Berikut implementasi navigasi dapat dilihat pada gambar 5, 6, dam 7.

MainAppScreen Kotlin

```kotlin
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@Composable
fun MainAppScreen(navController: NavHostController) {

    Scaffold(
        bottomBar = { BottomBar(navController = navController) }
    ) { paddingValue ->
        DestinationsNavHost(
            modifier = Modifier.padding(paddingValue),
            navController = navController,
            navGraph = NavGraphs.root,
        )
    }
}
```

Gbr. 5 Menggunakan DestinationNavHost Di MainAppScreen

Gambar 5 menjelaskan pemanggilan DestinationNavHost di dalam MainAppScreen dengan menyimpan nilai dari navController.

MainActivity Kotlin

```kotlin
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        WindowCompat.setDecorFitsSystemWindows(window, false)
        setContent {
            val navController = rememberNavController()
            DlawTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    MainAppScreen(navController = navController)
                }
            }
        }
    }
}
```

Gbr. 6 Memanggil MainAppScreen Di Dalam MainActivity

Gambar 6 menjelaskan pemanggilan MainAppScreen yang sudah dibuat seperti yang ada pada gambar 5.

HomeScreen Kotlin

```kotlin
@Destination<BerandaGraph>(start = true)
@Composable
fun HomeScreen(destinationsNavigator: DestinationsNavigator) {
  HomeContent(
      toChatAI = {
          destinationsNavigator.navigate(ChatbotScreenDestination(""))
      },
      cardAction = {
          destinationsNavigator.navigate(LawScreenDestination)
      },
      toSetting = {
          destinationsNavigator.navigate(SettingScreenDestination)
      }
  )
}
```

Gbr. 7 Navigasi Dilakukan Menggunakan DestinationsNavigator

Gambar 7 menjelaskan penggunaan navigasi dengan menggunakan destinationNavigator yang diberi nilai berupa tujuan screen dalam bentuk namaScreen + Destination seperti yang tertera di gambar yaitu LawScreenDestination.

2. *Implementasi dengan Flutter*

a. Pemilihan Library

Pemilihan library untuk Flutter dilakukan untuk memaksimalkan proses pengembangan.. Beberapa library yang dipilih adalah provider, json_annotation, flutter_svg, dio, retrofit, get_it, build_value, shared_preferences, sqflite, path, dan flutter_pdfview.

b. Pengembangan UI

Dapat dilihat pada gambar 8 dan 9 yang menjelaskan implementasi halaman Login menggunakan Flutter.

main.dart Kotlin

```kotlin
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => const SplashScreen(),
        '/login': (context) => const LoginScreen(),
        '/home': (context) => const MainScreen(),
      },
    );
  }
}
```

Gbr. 8 Memanggil MyApp Di Dalam main.dart dan Menambahkan Routes LoginScreen

LoginScreen.dart Kotlin

```
class LoginScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Image.asset('assets/logo.png'),
            Text('Masuk', style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold)),
            TextField(decoration: InputDecoration(hintText: 'Email')),
            TextField(decoration: InputDecoration(hintText: 'Password'), obscureText: true),
            ElevatedButton(onPressed: () {}, child: Text('Masuk')),
          ],
        ),
      ),
    );
  }
}
```

Gbr. 9 Tampilan Kode Layout LoginScreen Dengan Flutter

(1) MaterialApp

Flutter menggunakan MaterialApp sebagai entry point untuk aplikasi yang menyediakan beberapa fitur seperti tema, navigasi, dan lain-lain.

(2) Widget

- Flutter menggunakan konsep widget untuk segala hal dalam UI. Setiap elemen UI adalah sebuah widget.
- StatelessWidget dan StatefulWidget digunakan untuk membuat UI yang tidak berubah dan yang bisa berubah (stateful).

(3) Hierarki Widget

- Flutter menggunakan struktur hierarki widget yang berbasis tree. Setiap widget memiliki parent dan child.
- Misalnya, Column adalah widget yang dapat memiliki beberapa children untuk mengatur layout secara vertikal.

(4) Deklarasi Widget

Widget dideklarasikan dalam method build. Misalnya, TextField, Image, Text, ElevatedButton, dan lain-lain.

(5) Styling dan Theming

Styling dilakukan melalui properti widget itu sendiri. Misalnya, TextStyle untuk teks, BoxDecoration untuk dekorasi, dain lain-lain.

c. Pengelolaan State

Flutter menggunakan pendekatan deklaratif untuk membangun UI, tetapi dengan cara yang berbeda dalam mengelola state dan pembaruan UI. Proses pengelolaan state dan pembaruan UI di Flutter:

(1) Deklaratif UI

Pendefinisian UI dilakukan di dalam method build dari Widget.

(2) State Management

Flutter menggunakan StatefulWidget untuk mengelola state. Ketika state berubah, Flutter memanggil ulang build method dari StatefulWidget.

(3) Full Widget Rebuild

Ketika state berubah, build method dari StatefulWidget akan dipanggil ulang dan seluruh sub-tree dari widget tersebut akan di-rebuild. Ini berarti bahwa semua child widget dalam sub-tree akan di-rebuild, bukan hanya bagian yang terpengaruh oleh perubahan state.

(4) Element Tree

Flutter menggunakan Element tree untuk mengelola widget dan memperbarui UI secara efisien. Element tree membantu Flutter menentukan bagian mana dari UI tree yang perlu diperbarui, meskipun secara umum seluruh sub-tree dari StatefulWidget akan di-rebuild.

Seperti contoh pada gambar 10, variabel name diubah nilainya dari “Flutter” menjadi “Flutter & Dart” dengan memanggil method setState(). Karena ada perubahan, Flutter memanggil ulang method build untuk memperbarui tampilan.

MyScreen Kotlin

```
class MyScreen extends StatefulWidget {
  @override
  _MyScreenState createState() => _MyScreenState();
}
class _MyScreenState extends State<MyScreen> {
  String name = "Flutter";
  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        Text("Hello, $name!"),
        ElevatedButton(
          onPressed: () {
            setState(() {
              name = "Flutter & Dart";
            });
          },
          child: Text("Change Name"),
        ),
      ],
    );
  }
}
```

Gbr 10 Contoh Manajemen State Di Flutter

d. Navigasi

Navigasi antar layer dilakukan dengan menggunakan navigator dan MaterialPageRoute.

main.dart Kotlin

```
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => const SplashScreen(),
        '/login': (context) => const LoginScreen(),
        '/home': (context) => const MainScreen(),
      },
    );
  }
}
```

Gbr. 11 Menggunakan MaterialApp di main.dart

Gambar 11 menjelaskan penggunaan MaterialApp yang di dalamnya sudah disertakan initialRoute dan routes untuk navigasi.

navigasi Kotlin

```
if (viewModel.loginResponse?.ok == true) {
  Navigator.pushReplacementNamed(
      context, '/home');
}
ATAU
return lawCardItem(lawType.description, () {
  Navigator.push(
     context,
     MaterialPageRoute(builder: (context) =>
LawListScreen(code: lawType.code ?? '',))
   );
)};
```

Gbr. 12 Contoh Penggunaan Navigasi

Gambar 12 menjelaskan contoh penggunaan navigasi jika menggunakan routes yang terdaftar pada MaterialApp gambar 10 dan navigasi dengan langsung memanggil Screen yang diinginkan.

C. *Pengujian*

Tahap ini menjelaskan metode dan hasil pengujian yang dilakukan untuk membandingkan kinerja framework Flutter dan Jetpack Compose dalam mengembangkan aplikasi Link D Law. Pengujian ini bertujuan untuk mengevaluasi kecepatan pengembangan, kemudahan pembelajaran, efesiensi dan kemudahan perawatan kode, serta kinerja aplikasi dalam hal penggunaan memori, CPU, dan framerate.

1. *Pengujian Pengembangan*

Pengujian pengembangan dilakukan untuk mengevaluasi aspek-aspek seperti kecepatan pengembangan, keunggulan pembelajaran, keunggulan dan efisiensi kode, serta keunggulan perawatan kode. Berikut adalah hasil dari pengujian tersebut:

a. Kecepatan Pengembangan

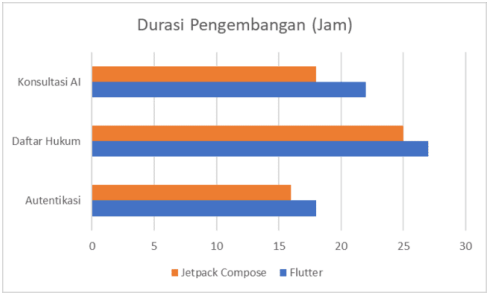
Pengujian ini mengukur waktu yang diperlukan untuk mengembangkan fitur-fitur tertentu dalam aplikasi menggunakan kedua framework.

TABEL I
HASIL PENGUJIAN KECEPATAN PENGEMBANGAN

Fitur Aplikasi	Durasi Pengembangan dengan Flutter (jam)	Durasi Pengembangan dengan Jetpack Compose (jam)
Autentikasi	18	16
Daftar Hukum	27	25
Konsultasi AI	22	18

Analisis:

Dari table 1 dan grafik pada gambar 13, dapat dilihat bahwa Jetpack Compose lebih unggul dibandingkan Flutter dalam pengembangan fitur-fitur aplikasi Link D Law. Meskipun kedua framework menggunakan pendekatan deklaratif, peneliti menemukan bahwa Jetpack Compose unggul karena kemampuannya dalam penyesuaian UI yang lebih baik, menjaga hierarki kode tetap mudah dibaca meski semakin kompleks, dan meminimalkan kebutuhan akan boilerplate code. Penyesuaian UI yang lebih efisien memungkinkan pengembang untuk lebih cepat menyesuaikan desain dan fitur, sementara struktur kode yang lebih rapi memudahkan dalam pemeliharaan dan pengembangan berkelanjutan. Faktor-faktor ini memberikan kontribusi signifikan terhadap kecepatan dan efisiensi pengembangan menggunakan Jetpack Compose.



Gbr. 13 Grafik Durasi Pengembangan

Gambar 13 adalah bentuk grafik dari hasil pengujian durasi pengembangan seperti yang dijelaskan dalam tabel 1.

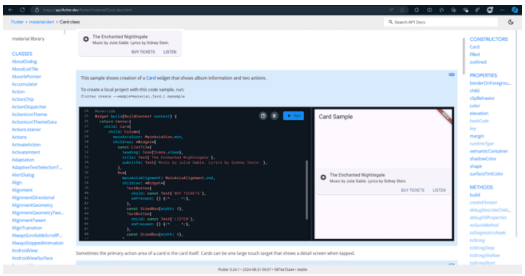
b. Keunggulan Pembelajaran

Pengujian ini menilai seberapa mudah bagi pengembang untuk mempelajari dan menggunakan kedua framework.

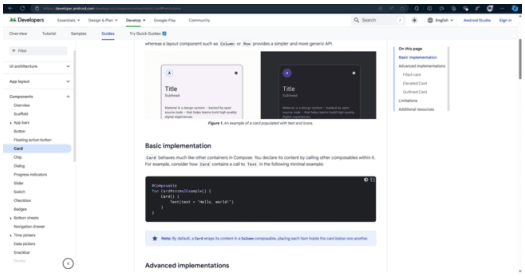
Hasil Pengujian:

Pengujian ini menilai seberapa mudah bagi pengembang untuk mempelajari dan menggunakan kedua framework. Pengujian ini salah satunya adalah menilai website resmi dokumentasi dari kedua framework seperti yang dapat dilihat pada gambar 14 dan 15. Hasil pengujian menunjukkan bahwa Flutter lebih unggul dalam hal pembelajaran, terutama karena dokumentasinya yang sangat lengkap, dukungan online compiler yang memungkinkan pengembang untuk langsung mencoba kode komponen, serta komunitas yang besar dan aktif. Dokumentasi yang lengkap dan tools tambahan seperti online compiler memudahkan pengembang baru dalam memahami dan mengimplementasikan fitur dengan cepat, sementara komunitas yang besar menyediakan banyak sumber daya tambahan dan dukungan cepat.

Di sisi lain, meskipun dokumentasi Jetpack Compose cukup lengkap, peneliti melaporkan bahwa diperlukan lebih banyak eksplorasi secara mandiri. Hal ini mungkin memerlukan lebih banyak waktu untuk mempelajari fitur-fitur yang kurang terdokumentasi atau untuk memecahkan masalah yang belum banyak dibahas dalam komunitas. Oleh karena itu, dalam konteks pembelajaran, Flutter menawarkan pengalaman yang lebih ramah bagi pengembang, terutama bagi mereka yang baru memulai atau membutuhkan dukungan komunitas yang lebih kuat.



Gbr. 14 Dokumentasi Flutter



Gbr. 15 Dokumentasi Jetpack Compose

Gambar 14 dan 15 adalah contoh dokumentasi dari kedua framework yang digunakan dalam pengujian keunggulan pembelajaran.

c. Keunggulan dan Efisiensi Kode

Pengujian ini menilai kemudahan dalam membaca dan menulis kode serta efisiensi kode yang dihasilkan.

Hasil Pengujian:

Pengujian ini menilai kemudahan dalam membaca dan menulis kode serta efisiensi kode yang dihasilkan. Hasil pengujian menunjukkan bahwa kode UI yang ditulis dengan Jetpack Compose lebih mudah dibaca meskipun memerlukan lebih banyak kode. Jetpack Compose memungkinkan penulisan kode yang modular dan reusable, yang sangat membantu dalam pengelolaan state dan mengurangi duplikasi kode. Struktur yang lebih jelas dan pendekatan deklaratifnya mempermudah pengembang dalam memahami dan memelihara kode, terutama dalam proyek besar atau kompleks.

Sebaliknya, meskipun Flutter menghasilkan kode yang lebih pendek, struktur kode yang dihasilkan sering kali lebih rumit dan kurang terbaca karena banyaknya widget nesting. Meskipun Flutter juga mendukung penulisan kode yang modular dan reusable, kompleksitas dalam hierarki widget dapat menjadi tantangan dalam pengembangan dan pemeliharaan aplikasi. Oleh karena itu, Jetpack Compose lebih unggul dalam hal keterbacaan dan efisiensi pengelolaan state, sementara Flutter mungkin lebih cocok untuk proyek yang memerlukan hasil cepat dengan kompleksitas UI yang lebih rendah.

d. Keunggulan Perawatan Kode

Pengujian ini mengevaluasi seberapa mudah melakukan pembaruan atau perubahan pada kode aplikasi.

Hasil Pengujian:

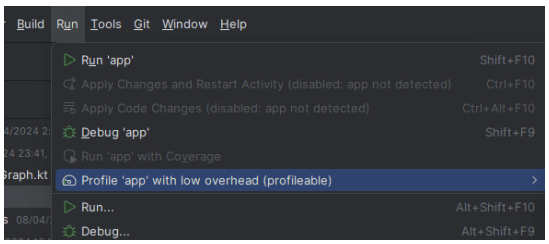
Pengujian ini mengevaluasi seberapa mudah melakukan pembaruan atau perubahan pada kode aplikasi. Hasil pengujian menunjukkan bahwa Jetpack Compose memiliki keunggulan dalam kemudahan perawatan kode berkat pendekatan deklaratifnya yang intuitif dan penggunaan Kotlin dengan sintaks yang bersih. Pendekatan ini memungkinkan pengembang untuk dengan mudah memperbarui dan memodifikasi bagian-bagian aplikasi tanpa risiko mempengaruhi keseluruhan kode secara negatif. Modularitas yang ditawarkan oleh Jetpack Compose juga mempermudah refactoring dan pembaruan fitur, yang sangat penting dalam perawatan kode jangka panjang.

Di sisi lain, meskipun Flutter juga menggunakan pendekatan deklaratif, tantangan utama dalam perawatan kode terletak pada struktur widget yang kompleks. Struktur ini dapat membuat refactoring lebih sulit dan berisiko, terutama dalam proyek yang besar dan kompleks. Namun, Flutter tetap menawarkan kelebihan dalam pengembangan cepat, terutama dengan fitur seperti hot reload yang mendukung iterasi cepat dan debugging.

Berdasarkan analisis ini, Jetpack Compose lebih unggul dalam hal perawatan kode, terutama untuk proyek-proyek yang membutuhkan stabilitas dan fleksibilitas jangka panjang. Sedangkan Flutter mungkin lebih cocok untuk proyek yang memerlukan hasil cepat dengan iterasi berkelanjutan meski menghadapi tantangan dalam struktur kode.

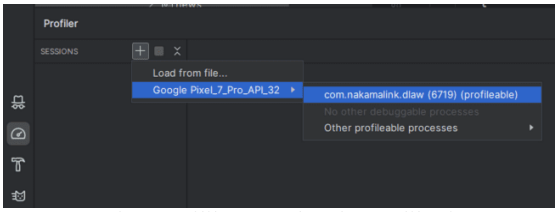
2. *Pengujian Kinerja*

Pengujian kinerja dilakukan untuk mengevaluasi penggunaan memori, penggunaan CPU, dan framerate aplikasi yang dikembangkan menggunakan kedua framework. Pengujian CPU dan Memory menggunakan App Profiler dari Android Studio.



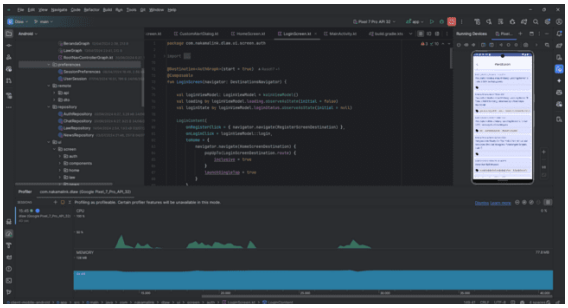
Gbr. 16 Run Aplikasi Dengan Mode Profile

Gambar 16 menjelaskan sebelum melakukan pengujian aplikasi harus di-run menggunakan mode profile. Mode profile lebih mencerminkan kondisi sebenarnya dari aplikasi saat dijalankan oleh pengguna akhir. Menguji dalam mode profile membantu memastikan bahwa aplikasi akan berperforma baik dalam kondisi yang sebenarnya dihadapi oleh pengguna.



Gbr. 17 Pilih Perangkat dan Aplikasi

Kemudian buka tools App Profiler, pilih perangkat dan aplikasi yang sedang berjalan seperti yang terlihat di gambar 17.



Gbr. 18 Statistik Aplikasi dari App Profiler

Setelah memilih perangkat dan aplikasi. Pengujian dapat dilakukan dengan mengamati statistik yang ditunjukkan App Profiler. Dapat dilihat pada gambar 18 yang menampilkan contoh statistik dari aplikasi yang berjalan.

a. Penggunaan Memori

Pengujian ini mengukur total penggunaan memori yang digunakan aplikasi dari kedua framework.

TABEL II

HASIL PENGUJIAN PENGGUNAAN MEMORI

Fitur Aplikasi	Penggunaan Memori	Penggunaan Memori dengan

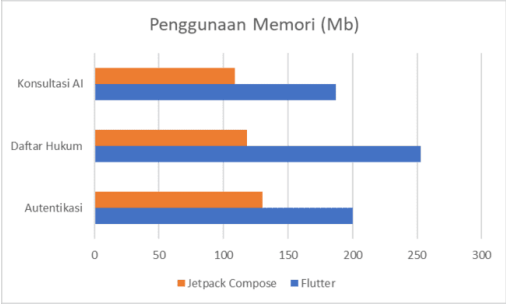
	dengan Flutter (mb)	Jetpack Compose (mb)
Autentikasi	200	130
Daftar Hukum	253	118
Konsultasi AI	187	109

Analisis:

Dari tabel 2 di atas, dapat dilihat bahwa Jetpack Compose menggunakan memori lebih sedikit dibandingkan Flutter untuk semua fitur yang diuji, menunjukkan efisiensi yang lebih tinggi dalam pengelolaan resource. Perbedaan signifikan dalam penggunaan memori, terutama pada fitur 'Daftar Hukum' dan 'Konsultasi AI', menunjukkan bahwa Jetpack Compose mampu mengelola data dan state dengan lebih efisien.

Efisiensi ini mungkin disebabkan oleh arsitektur Jetpack Compose yang lebih ringan dan optimal dalam penggunaan komponen UI. Implikasi dari efisiensi ini adalah kinerja aplikasi yang lebih baik, terutama pada perangkat dengan resource terbatas, di mana penggunaan memori yang lebih rendah dapat mengurangi lag dan meningkatkan pengalaman pengguna.

Berdasarkan hasil pengujian ini, Jetpack Compose lebih cocok digunakan dalam pengembangan aplikasi yang membutuhkan efisiensi memori tinggi dan performa yang stabil, terutama untuk aplikasi yang berjalan pada perangkat dengan spesifikasi rendah atau aplikasi yang memproses banyak data secara real-time.



Gbr. 19 Grafik Penggunaan Memori

Gambar 19 adalah bentuk grafik dari hasil pengujian penggunaan memori seperti yang dijelaskan dalam tabel 2.

b. Penggunaan CPU

Pengujian ini mengamati efisiensi penggunaan CPU pada perangkat.

TABEL III

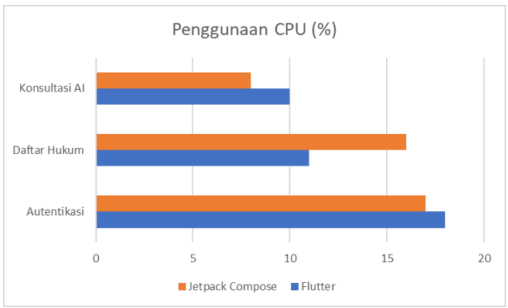
HASIL PENGUJIAN PENGGUNAAN CPU

Fitur Aplikasi	Penggunaan CPU dengan Flutter (%)	Penggunaan CPU dengan Jetpack Compose (%)
Autentikasi	18	17
Daftar Hukum	11	16
Konsultasi AI	10	8

Analisis:

Dari tabel 3 di atas, dapat dilihat bahwa penggunaan CPU antara Flutter dan Jetpack Compose bervariasi tergantung pada fitur yang diuji. Pada fitur 'Autentikasi,' penggunaan CPU hampir sama, menunjukkan bahwa kedua framework memiliki efisiensi yang sebanding untuk tugas-tugas sederhana. Namun, untuk fitur 'Daftar Hukum,' Jetpack Compose menggunakan lebih banyak CPU dibandingkan Flutter, yang mungkin disebabkan oleh kebutuhan komputasi tambahan dalam rendering atau pengelolaan data dalam fitur ini. Sebaliknya, untuk fitur 'Konsultasi AI,' Jetpack Compose lebih efisien dalam penggunaan CPU.

Variasi ini menunjukkan bahwa efisiensi penggunaan CPU di antara kedua framework tidak hanya bergantung pada framework itu sendiri tetapi juga pada jenis tugas atau fitur yang dikembangkan. Oleh karena itu, dalam memilih framework, penting untuk mempertimbangkan jenis fitur yang akan dikembangkan dan bagaimana masing-masing framework mengelola penggunaan CPU. Jetpack Compose mungkin lebih cocok untuk fitur-fitur yang membutuhkan efisiensi CPU dalam pengolahan data kompleks, sedangkan Flutter mungkin lebih unggul dalam fitur yang lebih ringan dalam penggunaan CPU.

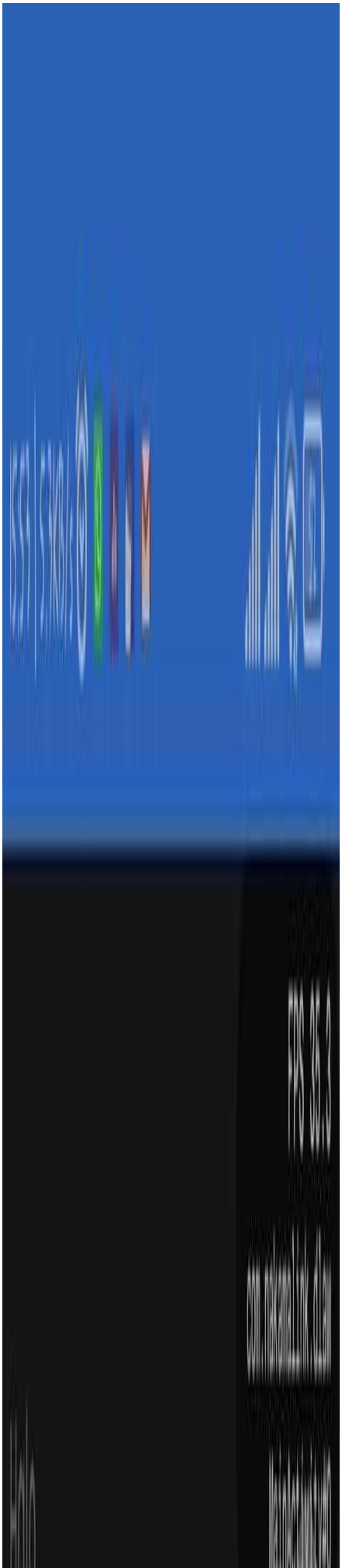


Gbr. 20 Grafik Penggunaan CPU

Gambar 20 adalah bentuk grafik dari hasil pengujian penggunaan CPU seperti yang dijelaskan dalam tabel 3.

c. Framerate Aplikasi

Pengujian ini memastikan aplikasi berjalan lancar tanpa jeda atau lag, terutama saat melakukan transisi. Pengujian framerate menggunakan aplikasi pihak ketiga bernama TakoStats. Gambar 21 adalah contoh penggunaan TakoStats saat pengujian.



Gbr. 21 Pengujian Framerate Dengan TakoStats

TABEL IV

HASIL PENGUJIAN FRAMERATE APLIKASI

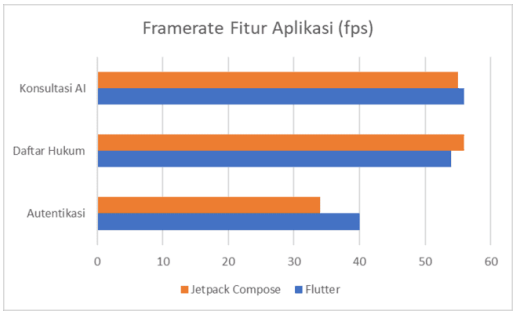
Fitur Aplikasi	Framerate dengan Flutter (fps)	Framerate dengan Jetpack Compose (fps)
Autentikasi	40	34
Daftar Hukum	54	56
Konsultasi AI	56	55

Analisis:

Dilihat dari tabel 4 di atas, performa framerate untuk fitur aplikasi yang diuji menunjukkan hasil yang bervariasi antara Flutter dan Jetpack Compose. Pada fitur 'Autentikasi,' Flutter menunjukkan framerate yang lebih tinggi, yang menunjukkan bahwa framework ini mungkin lebih efisien dalam rendering elemen UI sederhana dan menangani proses login dengan responsivitas yang baik.

Sebaliknya, pada fitur 'Daftar Hukum,' Jetpack Compose memiliki framerate yang sedikit lebih tinggi, yang mungkin disebabkan oleh kemampuannya dalam mengelola tampilan yang lebih kompleks dengan banyak elemen data atau daftar. Sementara itu, untuk fitur 'Konsultasi AI,' kedua framework menunjukkan performa yang hampir sebanding, mengindikasikan bahwa keduanya mampu menangani proses rendering yang melibatkan interaksi dengan algoritma AI atau data yang dinamis dengan baik.

Hasil ini mengindikasikan bahwa pemilihan framework harus mempertimbangkan jenis fitur yang akan dikembangkan. Flutter mungkin lebih cocok untuk fitur-fitur yang membutuhkan rendering cepat dan responsif, sementara Jetpack Compose mungkin lebih unggul dalam mengelola tampilan yang lebih kompleks dengan banyak elemen data.



Gbr. 22 Grafik Framerate Fitur Aplikasi

Gambar 22 adalah bentuk grafik dari hasil pengujian framerate aplikasi seperti yang dijelaskan dalam tabel 4.

IV. KESIMPULAN

Pada perbandingan proses pengembangan, Jetpack Compose menunjukkan keunggulan yang signifikan dalam proses pengembangan aplikasi Link D Law dibandingkan dengan Flutter. Dalam hal durasi pengembangan, Jetpack Compose memungkinkan pengembangan fitur 12.23% lebih cepat secara rata-rata dibandingkan Flutter, dengan keunggulan terbesar tercatat pada fitur Konsultasi AI, di mana Jetpack Compose mampu mengurangi waktu pengembangan hingga 18.18%. Meskipun Flutter lebih unggul dalam aspek pembelajaran berkat dokumentasi yang lengkap dan dukungan komunitas yang kuat, Jetpack Compose menawarkan struktur kode yang lebih mudah dipahami dan dikelola, serta memberikan efisiensi yang lebih tinggi dalam hal pengelolaan dan perawatan kode. Oleh karena itu, Jetpack Compose lebih unggul dalam kecepatan dan efisiensi pengembangan, terutama dalam proyek-proyek yang memerlukan stabilitas dan fleksibilitas jangka panjang.

Pada perbandingan performa fitur aplikasi Link D Law, Jetpack Compose juga menunjukkan efisiensi yang lebih tinggi dalam penggunaan memori dengan rata-rata penghematan memori sebesar 43.36% dibandingkan Flutter, terutama pada fitur 'Daftar Hukum' yang mencatat penghematan tertinggi sebesar 53.36%. Dalam penggunaan CPU, hasil pengujian bervariasi tergantung pada fitur yang diuji. Jetpack Compose lebih unggul dalam penggunaan CPU pada fitur 'Konsultasi AI' dengan efisiensi 20% lebih baik, sementara Flutter lebih efisien pada fitur 'Daftar Hukum' dengan penghematan 45.45% penggunaan CPU. Dalam aspek framerate, Flutter menunjukkan keunggulan dalam rendering UI sederhana dengan framerate 17.65% lebih tinggi pada fitur 'Autentikasi,' sedangkan Jetpack Compose lebih unggul dalam mengelola tampilan kompleks seperti 'Daftar Hukum' dengan framerate 3.7% lebih tinggi. Kesimpulannya, Jetpack Compose lebih cocok untuk aplikasi yang memerlukan efisiensi memori tinggi dan pengolahan data kompleks, sedangkan Flutter unggul dalam fitur yang membutuhkan rendering cepat dan responsif.

REFERENSI

[1] Diantoni, C., Komarudin, O., & Rizal, A. (2024). ARSITEKTUR MVVM DAN FRAMEWORK JETPACK COMPOSE PADA PENGEMBANGAN APLIKASI ANDROID. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *8*(3), 3216–3224. https://doi.org/10.36040/jati.v8i3.9638

[2] Noori, Z., & Eriksson, C. (2023). *UI Performance Comparison of Jetpack Compose and XML in Native Android Applications*.

[3] Hussain, H., Khan, K., Farooqui, F., Ali Arain, Q., & Farah Siddiqui, I. (2021). Comparative Study of Android Native and Flutter App Development. *Memory*, *47*, 36–37.

[4] Appfigures. (2024, March 31). *Top Development SDKs Installed in iOS & Android Apps*. https://appfigures.com/top-sdks/development/all (Accessed on: 30/03/2024)

[5] Flutter Developers. (2024, March 31). *Flutter*. https://flutter.dev/ (Accessed on: 31/03/2024)

[6] Yunus, C. D. P., & Ulum, M. B. (2023). Pengembangan Aplikasi Penjadwalan Konten Instagram Otomatis bagi Pelaku UMKM dengan Flutter Framework. *JURNAL ILMIAH INFORMATIKA*, *11*(02), 196–205. https://doi.org/10.33884/jif.v11i02.8038

[7] Marchenko, S. (2023). JETPACK COMPOSE: NEW APPROACHES TO ANDROID UI DEVELOPMENT. *Publishing House "Baltija Publishing."*

[8] Android Developers. (2024, March 31). *Jetpack Compose*. https://developer.android.com/jetpack/compose (Accessed on: 31/03/2024)