BAB 1
PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi digital yang pesat dalam beberapa tahun terakhir telah mendorong meningkatnya kebutuhan akan aplikasi mobile yang efisien, scalable, dan mudah dikelola. Dengan semakin banyaknya pengguna smartphone dan tablet, serta meningkatnya permintaan untuk aplikasi yang dapat memenuhi berbagai kebutuhan. Pengembang aplikasi dihadapkan pada tantangan untuk menciptakan solusi yang tidak hanya menarik secara visual, tetapi juga memiliki performa yang optimal dan dapat diandalkan.

Salah satu tantangan utama yang dihadapi pengembang aplikasi saat ini adalah mengelola kompleksitas kode seiring bertambahnya fitur dan skala aplikasi. Kompleksitas kode tersebut sering kali menyebabkan kesulitan dalam pemeliharaan kode, penurunan performa, serta peningkatan risiko terjadinya bug. Hal ini menjadi semakin krusial dalam konteks pengembangan aplikasi menggunakan Flutter, sebuah framework open-source yang memungkinkan pengembangan lintas platform dengan satu basis kode. Framework tersebut menggunakan bahasa pemrograman Dart dan mendukung pengembangan aplikasi yang cepat dan efisien dengan fitur hot reload, arsitektur berlapis, dan kaya akan widget bawaannya[1].

Berdasarkan survei terbaru yang dilakukan oleh Asosiasi Penyelenggara Jasa Internet Indonesia (APJII), jumlah pengguna internet di Indonesia pada tahun 2024 telah mencapai 221.563.479 jiwa dari total populasi 278.696.200 jiwa, dengan tingkat penetrasi internet sebesar 79,5%[2]. Angka ini menunjukkan bahwa mayoritas masyarakat Indonesia kini telah terhubung dengan dunia digital, baik untuk keperluan komunikasi, edukasi, hiburan, maupun transaksi bisnis. Seiring dengan meningkatnya jumlah pengguna internet, kebutuhan terhadap aplikasi digital yang berkualitas pun semakin tinggi. Pengguna tidak hanya menuntut aplikasi yang fungsional, tetapi juga mengharapkan aplikasi yang cepat, responsif, mudah digunakan, serta mampu beradaptasi dengan perkembangan kebutuhan. Kondisi ini menjadi tantangan tersendiri bagi pengembang dalam menciptakan aplikasi yang tidak hanya berjalan dengan baik, tetapi juga mudah dikembangkan, dipelihara, dan ditingkatkan seiring waktu.

Sebagai bentuk respon terhadap kebutuhan tersebut, dikembangkanlah aplikasi UBP (Umroh Backpacker). Aplikasi ini dirancang untuk membantu pengguna dalam melaksanakan ibadah umroh dengan menyediakan berbagai fitur penting, seperti informasi jadwal sholat, Al-Quran digital, kumpulan doa-doa umroh, panduan umroh, tata cara pelaksanaan umroh, serta pelaksanaan umroh secara lengkap. Dengan mengintegrasikan berbagai layanan tersebut dalam satu platform, aplikasi UBP bertujuan untuk memberikan kemudahan bagi para jamaah umroh, khususnya yang menginginkan perjalanan ibadah yang lebih mandiri dan terorganisir.

Dalam mengembangkan aplikasi UBP, penting untuk menjaga struktur kode yang bersih, modular, dan mudah dipelihara. Salah satu pendekatan yang banyak digunakan dalam pengembangan aplikasi modern adalah Clean Architecture. Clean Architecture menjadi salah satu solusi untuk mengatasi kompleksitas pengembangan aplikasi. Pendekatan ini menekankan pemisahan tanggung jawab melalui lapisan-lapisan terpisah, yaitu presentasi, domain, dan data. Setiap lapisan memiliki tanggung jawab yang jelas, sehingga memudahkan pengembang untuk memodifikasi, menguji, dan mengelola aplikasi secara efisien[3]. Dengan struktur yang terorganisir tersebut, Clean Architecture tidak hanya meningkatkan keterbacaan kode, tetapi juga mendukung prinsip dependency inversion, yang memungkinkan kode menjadi lebih fleksibel dan mudah diuji. Hal tersebut sangat penting dalam pengembangan aplikasi yang terus berkembang dan memerlukan pembaruan secara berkala.

Studi sebelumnya menekankan bahwa pendekatan arsitektur yang terstruktur, seperti arsitektur bersih [4] atau arsitektur terpisah, membantu membedakan siapa yang bertanggung jawab atas apa yang dilakukan setiap bagian. Ini membuat proses pembuatan dan perawatan aplikasi lebih mudah. Selain itu, penelitian oleh Saputra et al. [5] yang mengatasi masalah informasi yang terfragmentasi dengan menggunakan pemisahan arsitektur antara backend (menggunakan Laravel) dan frontend (menggunakan React JS) juga terbukti menghasilkan platform yang fleksibel dan skalabel. Menurut studi tentang Clean Architecture [4], metode ini memungkinkan pemisahan yang jelas antara logika bisnis dan antarmuka pengguna. Ini berdampak pada kemudahan membaca kode dan pengujian. Selain itu, penelitian [4] menunjukkan bahwa penerapan prinsip SOLID dalam arsitektur bersih, seperti prinsip Single Responsibility (SRP) dan Dependency Inversion

Principle (DIP), dapat membantu mengurangi ketergantungan langsung antara modul dalam aplikasi.

Dalam Implementasinya, Clean Architecture memiliki keselarasan dengan penggunaan state management seperti Provider dan BLoC pada framework Flutter. State management dalam Flutter merupakan teknik untuk mengelola state aplikasi dan memisahkannya dari tampilan atau view. Teknik tersebut memungkinkan pengembang untuk menciptakan aplikasi yang lebih fleksibel, mudah diubah, sekaligus meningkatkan performa dan pengalaman pengguna. Dalam penelitian mengenai pendekatan state management, pendekatan tersebut memainkan peran yang cukup penting dalam memastikan perubahan data aplikasi dapat ditampilkan secara responsif pada UI tanpa menyebabkan alokasi sumber daya yang tidak perlu[6].

Pemilihan solusi state management dalam aplikasi Flutter berdampak signifikan pada keseluruhan arsitektur dan pengalaman pengembangan [6]. Seiring dengan meningkatnya kompleksitas aplikasi, state management yang tepat menjadi aspek krusial untuk menjaga kode tetap terorganisir dan memiliki performa yang optimal [7]. Baik Provider maupun BLoC menawarkan pendekatan berbeda untuk menangani state, masing-masing dengan kelebihannya sendiri tergantung pada persyaratan proyek [8]. Implementasi state management dengan prinsip Clean Architecture memungkinkan pengembang untuk menghasilkan aplikasi yang lebih robust dan maintainable [3]. Hal tersebut dapat dicapai melalui pemisahan concern yang efektif serta pengelolaan state aplikasi yang sistematis dan dapat diprediksi [9].

Penelitian terkait dengan perbandingkan performa state management Provider dan GetX pada aplikasi Flutter dengan mempertimbangkan parameter seperti penggunaan CPU, konsumsi memori, dan frame rate. Namun, penelitian tersebut belum membahas secara spesifik penerapan state management dalam konteks Clean Architecture[7]. Oleh karena itu, terdapat celah penelitian yang signifikan mengenai impementasi Clean Architecture, khususnya dari segi modifiability, testability, dan performance pada skenario pengembangan aplikasi kompleks. Hal ini menjadi penting mengingat bahwa pemilihan arsitektur dan metode state management yang tepat dapat berdampak besar pada keberhasilan proyek pengembangan aplikasi.

Berdasarkan fenomena tersebut, penelitian ini difokuskan pada analisis perbandingan implementasi Clean Architecture menggunakan state management Provider dan BLoC pada pengembangan aplikasi UBP. Penelitian ini bertujuan untuk mengevaluasi kelebihan dan kekurangan kedua pendekatan tersebut, sehingga dapat menentukan solusi paling efektif dalam membangun aplikasi yang efisien, terstruktur, dan mudah dikembangkan. Hasil penelitian ini diharapkan dapat menjadi acuan bagi pengembang aplikasi mobile, terutama dalam memilih pendekatan arsitektur dan metode state management yang paling sesuai dengan kebutuhan proyek. Dengan demikian, penelitian ini tidak hanya berkontribusi pada pengembangan ilmu pengetahuan di bidang teknologi informasi, tetapi juga memberikan manfaat praktis bagi industri pengembangan aplikasi menggunakan Flutter.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah penulis uraikan di atas, berikut masalah yang dapat dirumuskan diantaranya:

1. Bagaimana perbedaan struktur kode antara penggunaan state management Provider dan BLoC dalam penerapan Clean Architecture pada aplikasi UBP? (Modifiability)
2. Bagaimana perbedaan pengujian unit testing antara state management Provider dan BLoC dalam Clean Architecture pada aplikasi UBP? (Testability)
3. Seberapa besar perbedaan konsumsi memori dan waktu eksekusi antara Provider dan BLoC dalam Clean Architecture pada Flutter? (Performance)

1.3. Tujuan Penelitian

Berdasarkan rumusan masalah yang telah dirumuskan, adapun tujuan dari dilakukannya penelitian ini diantaranya:

1. Menganalisis dan membandingkan struktur kode dalam implementasi Clean Architecture menggunakan *state management* Provider dan BLoC pada pengembangan aplikasi UBP untuk mengidentifikasi perbedaan organisasi dan kompleksitas kode.
2. Mengevaluasi dan membandingkan pendekatan unit testing antara implementasi *state management* Provider dan BLoC dalam konteks Clean Architecture pada framework Flutter untuk menilai tingkat testability masing-masing pendekatan.

3. Mengukur dan menganalisis perbedaan performa *state management* antara Provider dan BLoC dalam implementasi Clean Architecture pada pengembangan aplikasi UBP menggunakan framework Flutter dari segi konsumsi memori aplikasi dan waktu render pada tiap pendekatan.
4. Memberikan rekomendasi terkait dengan pemilihan *state management* yang lebih sesuai berdasarkan hasil analisis untuk pengembangan aplikasi Flutter menggunakan Clean Architecture.

1.4. Manfaat Penelitian

Dengan berjalannya penelitian ini, diharapkan dapat memberikan beberapa manfaat sebagai berikut:

1. **Menunjukkan kelebihan dan kelemahan implementasi state management Provider dan BLoC.** Penelitian ini membandingkan secara langsung penggunaan Provider dan BLoC dalam Clean Architecture. Hasilnya membuktikan bahwa Provider lebih cepat digunakan untuk pengembangan di tahap awal. Di sisi lain, BLoC menawarkan struktur kode yang lebih rapi dan lebih mudah dikembangkan (skalabel) untuk jangka panjang.
2. **Menyajikan data perbandingan performa yang jelas.** Kajian ini memberikan data kinerja yang spesifik, yang menunjukkan bahwa BLoC berjalan lebih baik di HP dengan spek rendah (*low-end*). Temuannya meliputi penggunaan memori (RSS) yang lebih hemat, proses *Garbage Collection* (GC) yang lebih jarang, dan pengurangan lag (*frame jank*) secara signifikan saat aplikasi bekerja berat.
3. **Menghasilkan rekomendasi berbasis data untuk pengembang.** Hasil analisis ini memberikan panduan praktis bagi tim pengembang untuk memilih state management yang tepat, disesuaikan dengan skala proyek, kerumitan fitur, dan target HP pengguna.

1.5. Batasan Masalah

Adapun juga dibuat batasan masalah guna menjaga fokus jalannya penelitian ini:

1. *State management* yang dibandingkan dalam penelitian ini hanya mencakup Provider dan BLoC. Metode *state management* lain seperti Riverpod, GetX, Redux, dan lainnya tidak menjadi fokus dalam penelitian ini.

2. Platform pengembangan aplikasi dibatasi hanya pada platform mobile berbasis *Android* menggunakan framework Flutter. Penelitian ini tidak mencakup implementasi dan pengujian pada platform lain seperti iOS, web, ataupun desktop.
3. Pengujian aplikasi dilakukan pada perangkat fisik Android dengan spesifikasi terbatas. Penelitian tidak mencakup variasi pengujian pada berbagai jenis perangkat dengan perbedaan spesifikasi perangkat keras maupun sistem operasi.
4. Analisis performa aplikasi dibatasi pada dua aspek utama, yaitu konsumsi memori dan waktu render frame. Aspek lain seperti battery usage atau user experience secara subjektif tidak menjadi fokus pengujian.
5. Penelitian ini tidak membahas aspek keamanan aplikasi, integrasi dengan backend service secara mendalam, atau proses CI/CD. Fokus utama adalah pada perbandingan arsitektur dan state management dari sisi struktur kode dan performa.