BAB 2
TINJAUAN PUSTAKA

2.1 Tinjauan Tempat Penelitian

Tinjauan tempat penelitian bertujuan untuk mengetahui keadaan yang ada di tempat penelitian di antaranya adalah sejarah berdirinya, visi dan misi, dan struktur organisasi dari pihak yang nantinya terlibat di dalam sistem.

2.1.1 Sejarah Singkat PT Bukalapak

Bukalapak didirikan pada tanggal 10 Januari 2010 oleh Achmad Zaky, Nugroho Herucahyono dan Fajrin Rasyid di sebuah rumah kos semasa berkuliah di Institut Teknologi Bandung. Momentum awal bagi kemajuan Bukalapak adalah ketika tren pengguna sepeda lipat melonjak pada tahun 2010. Pada saat itu, terdapat banyak komunitas yang menjual berbagai sepeda dan aksesorisnya dengan harga terjangkau sehingga meramaikan dan meningkatkan pertumbuhan pengguna di Bukalapak secara signifikan.

2.1.2 Visi, Misi dan Tujuan

Bukalapak merupakan salah satu online *marketplace* terkemuka di Indonesia. Seperti halnya situs layanan jual-beli menyediakan sarana jual-beli dari konsumen ke konsumen. Siapa pun dapat membuka toko online di Bukalapak dan melayani pembeli dari seluruh Indonesia untuk transaksi satuan maupun banyak. Bukalapak telah membuat aplikasi jual beli online yang menghubungkan jutaan pembeli dan pelapak di seluruh Indonesia. Bukalapak tidak menjual atau menyediakan barang/produk, melainkan hanya sebagai perantara.

1. Visi: Menjadi online marketplace nomor 1 di Indonesia.
2. Misi: Memberdayakan UKM yang ada di seluruh penjuru Indonesia.

2.2 Landasan Teori

Subbab landasan teori berisikan teori-teori pendukung yang digunakan pada proses analisis dan implementasi pada penelitian ini untuk pengembangan perangkat lunak di Metode Penerapan *Clean Architecture* dengan *Kotlin Multiplatform Mobile* pada Aplikasi Bukalapak. Adapun teori-teori yang digunakan sebagai berikut.

2.2.1 *Reengineering*

Reengineering adalah proses menganalisis sistem yang ada ke dalam bentuk baru untuk meningkatkan kualitas beroperasi, kemampuan sistem, fungsional, kinerja, dan kemampuan untuk berkembang dengan biaya yang murah. Dalam tahap siklus, *reengineering* mencangkup jangkauan luas, mulai dari *existing implementation*, *recapturing* atau *recreation* dari desain, dan mengartikan *requirements* secara nyata dalam implementasi oleh subyek sistem.

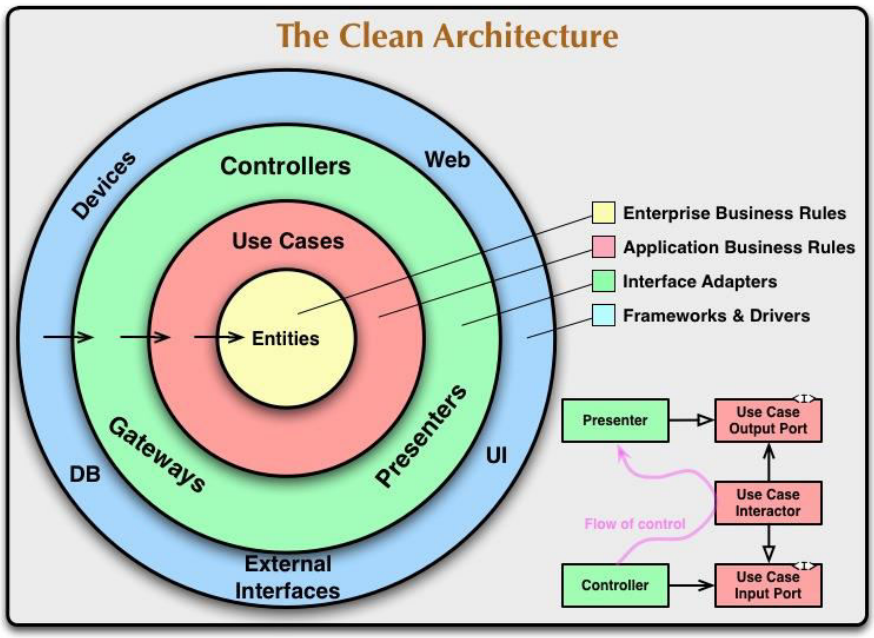
Reengineering dapat digunakan untuk mengekstraksi informasi-informasi yang ada dalam perangkat lunak. Informasi yang diekstraksi dapat digunakan untuk membangun kembali perangkat lunak ataupun membuat dokumentasi yang lebih *terupdate* dan akurat terhadap perangkat lunak. Saat melakukan *reengineering* ada beberapa faktor yang menjadi penyebab sistem untuk dilakukan *reengineering*. Faktor-faktor yang menyebabkan *reengineering* yaitu sebagai berikut.

1. Hilangnya atau tidak lengkapnya desain/spesifikasi.
2. Kadaluwarsa, tidak sesuai atau hilangnya dokumentasi.
3. Meningkatnya kompleksitas program.
4. Kurang terstrukturnya *source code*.
5. Kebutuhan untuk menerjemahkan program ke dalam bahasa program yang berbeda.

Dalam melakukan reengineering pada perangkat lunak memiliki enam tahapan atau proses utama yang perlu dilakukan. Proses *reengineering* meliputi *Inventory Analysis, Document Restrcuturing, Reverse Engineering, Code Restructuring, Data Restructuring* dan *Forward Engineering*[3].

2.2.2 Clean Architecture

Proses *reengineering* dalam penelitian ini menggunakan *Clean Architecture*, dimana hal ini bertujuan agar membagi kode program menjadi beberapa *layer* sehingga kode tidak keterkaitan dengan *platform*, Komponen UI, dan *services* ataupun *database.* Hal ini bertujuan agar dapat mengganti UI tergantung pada platform ataupun mengganti *database* yang ada tanpa perlu mengganti atau menyinggung bisnis kode utama kita (*business code*)[1].



Gambar 2-1 Diagram implementasi Clean Architecture

Selain itu kode yang kita buat menjadi lebih mudah untuk ditest, karena kode kita tidak bersinggungan dengan fungsi – fungsi *native* dari spesifik platform. Berikut penjelasan dari setiap layernya.

1. ***Data Layer***

   Pada *layer* ini kita akan mengimplementasi kode dengan kebutuhan pada segala hal yang berhubungan dengan data, sebagai contoh baik itu pengambilan data dari database, pemanggilan *API* dari suatu *webservice* ataupun dari suatu

library yang menyediakan suatu data. *Data Layer* akan mengimplement kontrak yang akan didefinisikan pada *Domain Layer* sehingga *Domain Layer* tidak akan tahu database apa yang dipakai atau dari manakah suatu data itu berasal. Selain itu pada *layer* ini juga terjadi proses *mapping* agar data yang dikirim bisa sesuai dengan yang dibutuhkan oleh *Domain Layer* atau *datasource* lainnya (database, webservice, dan lain - lain). Karena pada setiap pembuatan fitur aplikasi *datasource* yang dipakai pada antar platform (Android dan iOS) itu sama, maka implementasi *Data Layer* ini bisa kita lakukan dengan menggunakan KMM.

2. *Domain Layer*

Domain Layer atau ada juga yang menyebutnya *Business Layer* ini merupakan inti kode dari suatu fitur dimana *business logic* banyak diimplementasikan di *Layer* ini. *Layer* ini akan memproses data yang masuk baik itu dari *Presentation Layer* atau *Data Layer*. *Layer* ini dapat memvalidasi dan juga mengolah data sesuai dengan kebutuhan dari bisnis, sebagai contoh kita akan membuat suatu fitur login, pada *layer* ini kita akan mendefinisikan kebutuhan apa saja yang akan diperlukan untuk memenuhi kriteria login itu sendiri, misalkan untuk memenuhi syarat login, kita memerlukan email dengan format yang valid selain itu juga kita memerlukan *password* dengan minimal karakter lebih dari sama dengan enam karakter, semua kriteria itu kita implementasi pada kode pada bagian *Layer* ini, tujuan dari pemisahan kode ini adalah, agar bisnis kode kita tidak ada *dependency* pada platform atau dari suati *datasource*.

Domain Layer hanya mengetahui data apa yang diterima atau dikirim balik pada setiap layernya. Sehingga *layer* ini berisi kode – kode yang dapat berjalan tanpa harus tau platform apa yang menjalankannya atau database apa yang dipakai. *Domain layer* ini akan memberikan kontrak pada masing – masing *layer*, pada *Data Layer* akan diberikan kontrak *interface* dengan nama

Repository sedangkan untuk *Presentation Layer* akan diberikan kontrak *Presenter.*

Implementasi kode bisnis pun kemungkinan besar sudah pasti sama, sebagai contoh kita tidak mungkin akan memiliki kriteria validasi login yang berbeda antara Android dan iOS, format email yang dimasukan sama – sama harus valid dan juga untuk password pasti memiliki kriteria yang sama. Akan tetapi pada *layer* ini sering ditemukannya implementasi yang berbeda karena kode pada *layer* ini sering dikerjakan oleh orang yang berbeda, hal ini disebabkan karena pada perusahaan biasanya suatu developer hanya akan memegang satu platform saja. Karena implementasi kode bisnis pada antar platform itu sama, maka implementasi *Domain Layer* ini bisa kita lakukan dengan menggunakan KMM.

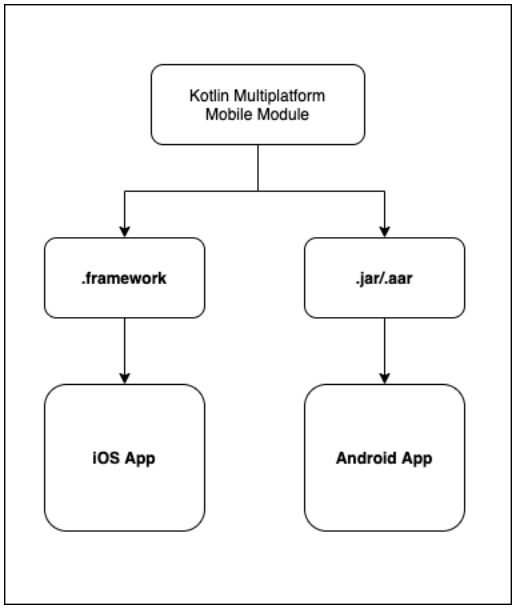
3. ***Presentation Layer***

Presentation Layer merupakan layer yang sangat berhubungan dengan *native* code, karena untuk implementasi Metode Penerapan Clean Architecture dengan Kotlin Multiplatform Mobile pada Aplikasi Mobile ini khususnya untuk UI akan menggunakan komponen / *view* dari masing – masing platform. Untuk *layer* ini seharusnya tidak akan banyak *logic* yang diimplementasi, karena fungsi utama kode yang ada di *layer* ini hanya untuk mengirim atau menerima data dari *Domain Layer*, *layer* ini hanya bertanggung jawab penuh atas tampilan / *output* yang diberikan pada user.

2.2.3 *Kotlin Multiplatform Mobile* (KMM)

Untuk menyelesaikan masalah perbedaan implementasi antar *platform* dan untuk memenuhi proses *Forward Engineering*, maka penelitian ini mencoba untuk mengimplementasi *reengineering* dengan *Kotlin Multiplatform Mobile*. *Kotlin Multiplatform Mobile* (KMM) adalah SDK untuk pengembangan mobile *cross platform* yang disediakan oleh JetBrains. Karena KMM ini dapat mengkompilasi

kode menjadi *binary*, sehingga output dari kompilasi dapat digunakan pada *platform* manapun dan hal ini dirancang untuk membuat pengalaman *end to end* dalam membangun aplikasi lintas platform *mobile* senyaman dan seefisien mungkin. KMM menawarkan gabungan manfaat dari membuat aplikasi lintas platform (*hybrid*) dan *native*. Dengan SDK ini kita dapat membuat kode yang dapat diadopsi pada beberapa platform (*shared code),* khususnya pada Android dan iOS.



Gambar 2-2 Diagram implementasi Clean Architecture menggunakan KMM

Kode yang ditulis pada modul *Kotlin Multiplatform Mobile* (KMM) dapat dikompilasi menjadi *.framework* atau *.jar/.aar* tergantung pada platform yang akan memakainya. Sehingga kita bisa membuat satu kode KMM dan dapat mengimplementasi hasil kompilasinya pada masing – masing platform (Android & iOS). Hal ini dapat mengurangi waktu pembuatan kode, selain itu KMM dapat mengurangi kesalahan (*Human Error*) dalam implementasi pembuatan kode antar platform.

2.2.4 Java Class Library (Library)

Java Class Library (Java API) adalah suatu kumpulan dari banyaknya kelas / *class* yang sudah terdefinisi *(Java predefined classes)*. Setiap kelas – kelas yang telah dibuat oleh *Software Engineer* diletakan pada paket / *package*, hal ini bertujuan untuk mengelompokan kelas – kelas yang mempunyai fungsi yang mirip (*related class*). Dengan begitu, kelas – kelas yang sudah berbentuk paket ini dapat digunakan kembali oleh *Software Engineer* lainnya dalam pembuatan program.

2.2.5 Software Development Kit (SDK)

Software Development kit adalah kumpulan alat untuk pengembangan atau pembuatann perangkat lunak dalam satu paket yang dapat diinstal. SDK memfasilitasi pembuatan aplikasi dengan memiliki kompiler, debugger, dan mungkin kerangka perangkat lunak. SDK biasanya dibuat khusus untuk spesifik platform perangkat keras dan kombinasi sistem operasi.

2.2.6 Android

Android adalah sebuah sistem operasi (OS) yang berbasis *Linux*. Sistem operasi untuk khusus dirancang dan dipasang pada *smartphone* dan *tablet*. Selain itu sistem operasi dapat menyesuaikan dengan spesifikasi dari setiap *device smartphone* yang ada, mulai dari *low-end* hingga spesifikasi yang *high-end*. Sistem ini pertama kali dikembangkan oleh perusahaan yang bernama Android Inc. Sistem operasi Android ini sekarang menduduki posisi pertama sebagai sistem operasi seluler terpopuler.

2.2.7 iOS

iOS atau dahulu dikenal sebagai iPhone OS adalah sistem operasi yang dibuat dan dikembangkan oleh perusahaan Apple Inc. khusus untuk iPhone, iPod Touch dan

iPad (versi dahulu sebelum iPadOS). Sistem operasi iOS ini adalah sistem operasi seluler terpopuler kedua di dunia setelah Android.

2.2.8 Gradle

Gradle adalah alat otomatisasi build untuk pengembangan perangkat lunak multi-bahasa. Gradle mengontrol proses pengembangan dalam tugas kompilasi dan pengemasan untuk pengujian, penerapan, dan penerbitan. Bahasa yang didukung termasuk *Java* (*Kotlin, Groovy, Scala*), C / C ++, JavaScript.

2.2.9 Metode Use Case Point (UCP)

Metode Use Case Point pertama kali dipopulerkan oleh Gustav Karner pada tahun 1993 yang merupakan turunan dari metode Function Point Analysis (FPA) yang bertujuan untuk menyediakan metode estimasi sederhana dengan berorientasi pada objek proyek perangkat lunak [4].

1. **Unadjusted Use Case Point (UUCP)**

   Langkah pertama adalah mengidentifikasi dan mengklasifikasikan jenis actor dari Use Case Diagram. Jenis aktor diklasifikasikan menjadi 3 (tiga) kategori yaitu; simple, average dan complex. Simple actor merupakan merupakan aktor yang berinteraksi melalui API seperti Command Prompt. Average actor adalah aktor yang berinteraksi melalui protokol seperti TCP/IP, FTP, HTTP atau disebut aktor yang melakukan penyimpanan data (file, RDBMS). Sedangkan Complex Actor merupakan aktor yang berinteraksi melalui GUI atau halaman web [5].

Table 2-1 Tipe dan Bobot aktor pada UAW

Tipe	Bobot
Simple	1
Average	2
Complex	3

Unadjusted Actor Weights (UAW) diperoleh dari berapa banyak aktor dari masing-masing tipe aktor kemudian dikali dengan total bobot faktor masing-masing.

Table 2-2 Tipe dan Bobot UseCase

Tipe	Bobot
Simple	5
Average	10
Complex	15

Langkah kedua, setiap Use Case diklasifikasikan ke dalam salah satu dari tiga jenis yaitu Simple, Average dan Complex. Simple Use Case mengandung maksimum 3 transaksi. Average Use Case berisi antara 4 sampai 7 transaksi dan Complex Use Case mengandung lebih dari 7 transaksi. Unadjusted Use Case Weights (UUCW) dihitung dengan mengalikan jumlah bobot masing-masing Use Case lalu dijumlahkan. Unadjusted Use Case Point (UUCP) diperoleh dari penjumlahan Unadjusted Actor Weights (UAW) dengan Uadjusted Use Case Weights (UUCW) seperti pada Persamaan 1.

$$\mathbf{UUCP = UAW + UUCW} \qquad (1)$$

2. *Technical Complexity Factor* (TCF)

Technical Complexity Factors (TCF) berisi 13 faktor. Setiap nilai faktor merupakan pengaruh terhadap produktifitas perangkat lunak yang memiliki peringkat antara 0 sampai 5 (0 berarti tidak ada pengaruh dan 5 berarti pengaruh yang kuat). TCF dihitung dengan mengalikan nilai setiap bobot faktor

(T1-T13) kemudian menambahkan semua angka-angka untuk mendapatkan jumlah yang disebut TFactor lalu menerapkan Persamaan 2.

$$\mathbf{TCF = 0.6 + (0.01 * TFactor)} \qquad (2)$$

Table 2-3 ***Technical Factors***

Technical Complexity Factor	**Bobot**
Distributed System Required	2
Response Time	2
End User Efficiency	1
Complex Internal Processing Required	1
Reusable Code	1
Easy to Install	0,5
Easy to Use	0,5
Portable	2
Easy to Change	1
Concurrent	1
Security Features	1
Access for Third Parties	1
Special Training Required	1

3. *Environment Complexity Factor* (ECF)

Environment Factor (EF) dihitung dengan mengalikan nilai faktor (F1-F8) dengan bobot nilai masing-masing kemudian dijumlahkan untuk mendapatkan jumlah yang disebut Efactor.

$$\mathbf{ECF = 1.4 + (-0.03 * EFactor)} \qquad (3)$$

Table 2-4 *Environment Factors*

Environment Complexity Factor	**Bobot**
Familiarity With The Project	1,5
Application Experience	0,5
OO Programming Experience	1,0
Lead Analyst Capability	0,5
Motivation	1,0
Stable Requirements	2,0
Part Time Staff	-1,0
Difficult Programming Language	-1,0

4. *Estimation Effort*

Pada langkah terakhir didapatkan hasil Use Case Point dengan mengalikan Unadjusted Use Case Point dengan Technical Complexity Factor.

UCP = UUCP * TCF * ECF (4)