KOMPARASI FLUTTER DAN REACT NATIVE DALAM PENGEMBANGAN PERANGKAT BERGERAK

Disusun Oleh:

N a m a : Zakki Abdurrahman Haris

NIM : 15523164

PROGRAM STUDI TEKNIK INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2021

HALAMAN PENGESAHAN DOSEN PEMBIMBING

KOMPARASI FLUTTER DAN REACT NATIVE DALAM PENGEMBAGAN PERANGKAT BERGERAK

TUGAS AKHIR

Disusun Oleh:

N a m a : Zakki Abdurahman Haris

NIM : 15523164

Yogyakarta, 10 Maret 2021

Pembimbing,

(Andhik Budi Cahyono, S.T., M.T.)

HALAMAN PENGESAHAN DOSEN PENGUJI

KOMPARASI FLUTTER DAN REACT NATIVE DALAM PENGEMBANGAN PERANGKAT BERGERAK

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Teknik Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 10 Maret 2021

Tim Penguji

Andhik Budi Cahyono, S.T., M.T.

Anggota 1

Hanson Prihantoro Putro, S.T., M.T.

Anggota 2

Beni Suranto, ST., M.Soft.Eng.

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Zakki Abdurrahman Haris

NIM : 15523164

Tugas akhir dengan judul:

KOMPARASI FLUTTER DAN REACT NATIVE DALAM PENGEMBANGAN PERANGKAT BERGERAK

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 10 Maret 2021

(Zakki Abdurrahman Haris)

HALAMAN PERSEMBAHAN

Dengan mengucapkan syukur Alhaumdullilahirabbil'alamin, tugas akhir ini dapat diselesaikan. Saya persembahkan tugas akhir ini kepada orang-orang yang saya cintai:

Kedua orang tua tercinta
Bapak Slamet Abadi dan Ibu Sri Zubiarti

Yang selalu mendoakan saya tiada henti, membimbing saya dari kecil, dan memberikan nasehat dan semangat. Semoga dengan selesainya tugas akhir ini, dapat membuat kedua orang tua bangga dan menjadi wujud bakti kepada orang tua.

Kakak serta Adik
Annisa Ardi & Ananda Ayu

Yang selama ini selalu menemani dan membantu memberikan semangat tiada henti dan mendorong saya sehingga tugas akhir ini dapat selesai. Semoga dengan selesainya tugas akhir ini, dapat memberikan semangat kepada kakak.

Keluarga dekat
Bapak Risdiyono dan istri.

Yang selama ini membantu saya dari awal masa pendidikan sehingga sampai saat ini. Semoga dengan selesainya tugas akhir ini, dapat membuat bangga atas perjuanggan selama ini.

HALAMAN MOTO

"Allah SWT tidak membebani seseorang melainkan sesuai dengan kesanggupannya"

(QS. Al-Baqarah: 286)

"Ilmu itu dimiliki sebelum berkata dan beramal"

(Muqaddimah Shahih Al-Bukhari)

"Sebaik-baik manusia adalah yang paling bermanfaat bagi manusia"

(HR. Ahmad, ath-Thabrani, ad-Daruqutni)

KATA PENGANTAR

Assalamu'alaikum warahmatulllahi wabarakatuh

Alhamdulillahirabilalamin, puji syukur kita panjatkan kepada Allah Subhanahu wa ta'ala yang telah memberikan rahmat, taufiq serta hidayahNya kepada kita semua. Sholawat serta salam kita haturkan kepada nabi kita Nabi Muhammad ﷺ yang telah mengantarkan kita dari zaman jahiliyah ke zaman yang terang benderang seperti sekarang dan penuh dengan ilmu pengetahuan sehingga penulis dapat menyelesaikan laporan Tugas Akhir yang berjudul "Komparasi Flutter dan React Native dalam Pengembangan Perangkat Bergerak".

Laporan tugas akhir ini disusun sebagai salah satu syarat yang harus dipenuhi untuk memperoleh gelar sarjana Strata 1 di Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia. Ketika pengerjakan penulisan dan pengerjaan tugas akhir, penulis menyadari bahwa semuanya tidak terlepas dari berbagai pihak yang turut andil dalam membantu dan mendukung penulis. Maka dari itu pada kesempatan yang ada ini penulis ingin menyampaikan rasa terima kasih sebesar-besarnya kepada:

1. Allah SWT, yang telah memberikan kemudahan dalam segala pengerjaan laporan Tugas Akhir ini.
2. Kedua orang tua Bapak Slamet Abadi dan Ibu Sri Zubiarti, kakak serta adik Annisa Ardi dan Ananda Ayu serta Keluarga besar yang selalu mendukung dan mendoakan penulis dalam mengerjakan laporan Tugas Akhir.
3. Bapak Prof. Fathul Wahid, S.T., M.Sc., Ph.D, selaku Rektor Universitas Islam Indonesia.
4. Bapak Prof. Dr. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Hendrik, S.T., M.Eng., selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
6. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Prodi Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
7. Bapak Andhik Budi Cahyono, S.T., M.T., selaku pembimbing tugas akhir yang telah meluangkan waktu, membagikan ilmu, serta membimbing penulis dalam menyelesaikan tugas akhir.
8. Bapak dan ibu dosen jurusan Teknik Informatika yang telah memberikan ilmu yang bermanfaat kepada penulis pada masa pendidikan.
9. Teman-teman Asrama The Cave yang telah memberikan doa dalam menyelesaikan Tugas Akhir ini.

10. Semua pihak yang telah membantu yang tidak dapat penulis tuliskan namanya satu-persatu.

Penulis Menyadari bahwa laporan yang dibuat oleh penulis ini masih belum sempurna, dikarenakan keterbatasan, kemampuan serta pengalaman dalam prosesnya. Maka dari itu penulis mengharapkan kritik dan saran yang membangun demi kesempurnaan laporan Tugas Akhir ini. Akhir kata, penulis berharap semoga laporan ini dapat memberikan manfaat bagi semua pihak.

Yogyakarta, 10 Maret 2021

(Zakki Abdurrahman Haris)

SARI

Perangkat bergerak merupakan piranti yang banyak digunakan untuk berbagai macam kebutuhan. Jumlah pemilik perangkat bergerak terus tumbuh, demikian juga teknologi yang berada pada perangkat tersebut juga berkembang. Dukungan komunitas, industri, dan kakas pengembangan aplikasi bergerak juga semakin banyak. Salah satu yang terus berkembang adalah bahasa pemrograman dan *framework* pengembangan perangkat bergerak seperti Flutter dan React Native. Tidak sedikit pengembang yang kesulitan untuk memilih *framework*. Oleh karena itu, diperlukan pengujian dengan melakukan komparasi untuk mengetahui *framework* yang sesuai dengan yang diharapkan pengembang aplikasi.

Komparasi dilakukan dengan menggunakan beberapa kriteria yang disebut domain seperti *customizability, modifiability, testability,* dan *performance* yang kemudian akan diimplementasikan dan diuji pada aplikasi yang dikembangkan yaitu aplikasi *lost and found*. Dari implementasi dan pengujian ini kemudian diidentifikasi perbedaan kedua *framework* tersebut berdasarkan subdomain.

Hasil yang diidentifikasi dalam pengujian dengan beberapa domain menunjukkan bahwa React Native baik untuk *customizability* (*setup* dan *configuration*), *modifiability* (level *coupling* data), dan *performance* (mode debug). Sedangkan Flutter baik untuk modifiability (jumlah fungsi, level *cohesion procedural*), *performance* (mode *profile*), dan *testability*.

Dengan demikian Flutter baik digunakan ketika pengembang ingin meminimalkan fungsi dalam komponen dan kemudahan dalam pengujian. Sedangkan React Native baik digunakan ketika pengembang ingin lebih banyak varian konfigurasi dan kinerja pada saat pengembangan serta kemudahan dalam mengelola data.

Kata kunci: komparasi, flutter, react native.

GLOSARIUM

API	Penerjemah komunikasi antara klien dengan server
DOM	Antar muka pemprograman yang bersifat independen
FPS	Jumlah bingkai gambar yang ditunjukkan dalam satu detik.
Library	Kumpulan fungsi atau prosedur atau modul
Memori	Media penyimpanan data sementara pada komputer
NDK	Rangkaian alat yang memungkinkan menimplementasikan bagian aplikasi dalam kode native
OEM	Perusahaan yang memporduksi suatu produk
State	Data dinamis yang dapat digunakan untuk menentukan bagaimana suatu komponen digambar
User Interface	Tampilan grafis yang berhubungan langsung dengan pengguna
Virtual Machine	Perangkat lunak dari sebuah mesin komputer yang dapat menjalankan program salam seperti layaknya sebuah komputer.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Mobile phone adalah piranti yang saat ini tidak bisa dipisahkan dari kehidupan manusia untuk bermacam kebutuhan. Hampir semua jenis aplikasi disediakan dalam perangkat bergerak karena hampir semua orang memilikinya. Jumlah pemilik perangkat bergerak terus tumbuh (Pusparisa, 2020), demikian juga teknologi yang berada dibalik perangkat tersebut yang juga semakin berkembang. Dukungan komunitas, industri, dan kakas pengembangan aplikasi bergerak juga semakin banyak. Salah satu yang terus berkembang adalah bahasa pemrograman dan *framework* pengembangan perangkat bergerak.

Saat ini sistem operasi utama yang disematkan dalam perangat bergerak ada dua yaitu Android OS (Google) dan IOS (Apple). Sistem operasi yang lain seperti MIUI (Xiaomi), Color OS (OPPO-VIVO), dan beberapa OS yang lain adalah pengembangan dari Android OS. Beberapa aplikasi dikembangkan untuk kedua sistem operasi tersebut, tersedia di Android OS dan juga IOS. Karena itu munculah bahasa dan *framework* pengembangan yang menghasilkan aplikasi yang bisa berjalan di keduanya. *Framework* yang demikian dikenal dengan istilah *cross platform framework* (Waranashiwar & Ukey, 2018). Salah satu *cross platform framework* yang saat ini sangat diminati oleh pengembang adalah Flutter.

Flutter adalah *framework* open-source yang dikembangkan oleh google. Digunakan untuk mengembangkan *mobile phone* yang dapat berjalan pada perangkat android dan IOS dengan *single codebase*. Dibuat menggunakan bahasa pemprograman C, C++, dart, dan Skia Graphics Engine (Flutter dev, 2021). Flutter pertama kali rilis pada mei 2017 dan sudah berada pada versi stabil yaitu v1.0.0 pada 5 desember 2018 dan sekarang sudah mencapai versi v1.12.13.

Cross platform framework yang lain adalah React Native. React native adalah framework open-source yang dikembangkan oleh facebook (Occhino, 2015). Sama seperti Flutter, kerangka kerja ini dapat digunakan untuk mengembangkan aplikasi perangkat bergerak yang dapat berjalan pada sistem operasi android dan iOS. Dibuat dengan menggunakan bahasa javascript, java, C++, Objective-C, dan Objective-C++ (Shergin et al., 2021). React Native pertama kali rilis pada 26 maret 2015 dan sudah berada pada versi stabil yaitu 0.61.5 (pada 23 November 2019) dan termasuk versi terbarunnya.

Dari kedua *framework* tersebut mana yang lebih baik dan harus dipilih oleh pengembang? Penelitian ini bertujuan untuk menjawab pertanyaan tersebut. Tidak sedikit pengembang yang kesulitan untuk memilih *framework* perangkat bergerak karena setiap *framework* memiliki kelebihan dan kekurangan. Oleh karena itu penelitian ini ingin meneliti dengan mengkomparasikan kedua *framework* berdasar kriteria pengukuran yang sudah ditentukan sebelumnya. Dengan adanya komparasi ini diharapkan dapat membantu pengembang untuk memilih framework pengembangan yang sesuai dengan kebutuhan proyek pengembangan aplikasi bergeraknya.

Dalam melakukan komparasi diperlukan kriteria-kriteria yang umumnya digunakan dalam membandingkan dua atau lebih *framework* pemrograman. Kriteria ini selanjutnya disebut dengan domain. Domain merupakan serangkaian batasan konsep yang ditentukan untuk menjadi dasar penilaian sesuatu, domain digunakan agar mengurangi kompleksitas dalam komparasi *framework* dengan memisahkan menjadi area yang lebih kecil agar mudah untuk dikelola. Domain yang dipilih dalam penelitian ini adalah sebagai berikut:

1. *Customisability*, kemudahan pengubahan perangkat lunak oleh penggguna dengan memodifikasi parameter desain (Jiao & Tseng, 2004).
2. *Performance*, sejauh mana sistem atau perangkat lunak dalam mencapai tujuan dengan waktu yang ditentukan (Oshana, 2013).
3. *Modifiability*, kemudahan dalam perubahan dan penambahan sistem (Saliu et al., 2009).
4. *Testability*, karakteristik yang menunjukkan suatu komponen dalam menentukan status secara akurat (Garousi et al., 2019).

Penelitian yang sejenis dilakukan oleh (Gerdessen, 2007) berjudul "*Framework comparison method comparing two frameworks based on technical domains, focussing on customisability and modifiability*". Penelitian tersebut membandingkan antara dua *framework java* yang berfokus pada kemudahan pengembangan dengan menggunakan domain *customizability* dan *modifiability.* Perbedaan penelitian ini antara penelitian sebelumnya terletak pada objek penelitian yaitu Flutter dan React Native, dengan penambahan beberapa domain.

1.2 Rumusan Masalah

Bagaimana komparasi *framework* Flutter dan React Native dalam pengembangan perangkat bergerak berdasarkan domain komparasi yang sudah ditentukan yaitu *customisability*, *performance*, *modifiability*, dan *testability*?

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini yaitu:

a. Komparasi berfokus pada domain customisability, performance, modifiability, dan testability.
b. Versi *framework* React Native yang digunakan yaitu versi 0.61 sedangkan untuk Flutter yaitu versi 1.9.1. Versi ini digunakan karena versi yang stabil dan terbaru pada bulan November 2019.
c. Penelitian hanya berfokus pada sistem operasi android.
d. Studi kasus dalam penelitian ini adalah Sistem Informasi Pengelolaan Barang Temuan dan Barang Hilang.

1.4 Tujuan Penelitian

Tujuan yang ingin dicapai dalam penelitian ini adalah melakukan komparasi terhadap *framework* Flutter dengan *framework* React Native dalam pengembangan perangkat bergerak sesuai dengan domain yang sudah ditentukan yaitu *customisability*, *performance*, *modifiability*, dan *testability*.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah agar pengembang perangkat bergerak mudah dalam menentukan *framework* untuk pengembangan perangkat bergerak sesuai dengan apa yang diharapkan.

1.6 Metodologi Penelitian

a. Studi literatur.
b. Pemilihan objek yang akan dikomparasi.
c. Pemilihan parameter komparasi.
d. Penentuan sekenario komparasi.
e. Proses komparasi.
f. Analisis komparasi.
g. Kesimpulan komparasi.

1.7 Sistematika Penulisan

Sistematikan penulisan yang digunakan pada penelitian ini adalah sebagai berikut:

Bab I Pendahuluan

Bab ini memuat tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelititan, manfaat penelitian, metodologi penelitian, sistematika penulisan.

Bab II Landasan Teori

Bab ini menjelaskan landasan teori yang digunakan dalam penelitian. Dalam bab ini terdapat beberapa penjelasan tentang *mobile application framework* dan domain.

Bab III Metodologi Penelitian

Bab ini memuat tentang analisis dan langkah-langkah penyelesaian masalah yang terdapat pada penelitian.

Bab IV Hasil dan Pembahasan

Bab ini berisikan tentang hasil dan pembahasan dari pengujian framework dengan studi kasus dan domain yang sudah ditentukan.

Bab V Kesimpulan dan Saran

Bab ini berisikan tentang kesimpulan dan saran mengenai komparasi agar dapat digunakan sebagai bahan penelitian ataupun dikembangkan dari kekurangan dan kesalahan pada proses maupun metode komparasi.

BAB II
LANDASAN TEORI

2.1 *Mobile Aplication Framework*

Banyak kerangka kerja dan SDK (Software development kits) yang tersedia untuk membangun sebuah perangkat bergerak yang kompatibel dengan dua sistem operasi, dalam kasus ini IOS dan android. *Mobile Aplication Framework* tersebut digunakan untuk memudahkan pengembang dalam membuat perangkat bergerak dengan *single codebase*. *Single codebase* memungkinkan pengembang untuk membuat aplikasi yang berbeda platform hanya dengan satu bahasa pemprograman. Dengan *single codebase* pengembang dapat menghasilkan perangkat lunak bergerak yang dapat berjalan di berbagai sistem operasi terutama IOS dan android. Kerangka kerja yang demikian dikenal juga dengan istilah *cross platform framework.*

Cross platform framework merupakan native atau kombinasi antara native aplikasi dan web aplikasi yang dapat membuat lebih dari satu *platform*, dikembangkan menggunakan *library* yang bersifat terbuka dan memiliki akses ke dalam beberapa bagian fungsi perangkat seperti kamera, GPS, map, dst (Caicedo, 2015). Untuk pembuatan aplikasi dalam kasus ini menggunakan *framework* Flutter dan React Native karena kedua *framework* tersebut merupakan cross platform *framework* yang disukai dan diinginkan (Stackoverflow, 2019).

Menurut (Liu & Yu, 2011), dapat dikatakan android *framework* jika meliputi:

a. Terdapat banyak komponen yang dapat dikembangkan dan digunakan untuk membangun antarmuka seperti text, tombol, dan lainnya.
b. Dapat mengakses data dari aplikasi lain maupun data sendiri.
c. *Resource manager* menyediakan sumber daya seperti grafik, tata letak berkas.
d. *Notification manager* menampilkan status peringatan khusus.
e. *Activity manager* mengelola siklus aplikasi dan menyediakan navigasi.

2.2 Flutter

Flutter adalah *toolkit* antar muka untuk membuat aplikasi yang dikompilasi secara native untuk seluler, website, desktop dari *single codebase* (Flutter web, 2020). Flutter dirilis petama kali pada tahun 2017 oleh google, dibangun menggunakan bahasa C, C++, Dart, dan Skia (a 2D *rendering* engine).

Saat pengembangan aplikasi, Flutter didukung dengan adanya fitur *hot-reload* yang dianggap salah satu fitur utama yang diunggulkan untuk meningkatkan pengembangan. Fitur *hot-reload* memudahkan pengembang dalam membuat *user interface*, menambahkan fitur aplikasi, dan meperbaiki *bug*. *Hot-reload* berkerja dengan memasukkan kode yang sudah diperbarui ke dalam *dart virtual machine* yang sedang berjalan. Setelah *virtual machine* memperbarui *class* dan fungsi, secara otomatis kerangka kerja Flutter akan membangun ulang *widget* sehingga pengembang dapat secara langsung melihat efek perubahan (Flutter web, 2020).

2.2.1 Dart

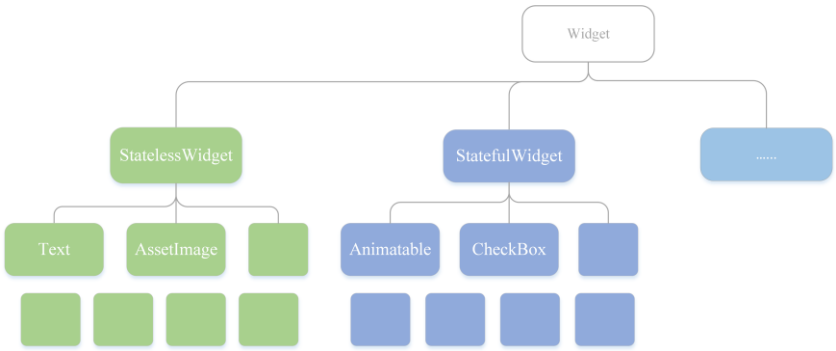
Flutter menggunakan bahasa pemporgraman Dart sebagai bahasa utamanya. Dart adalah bahasa pemprograman yang dikembangkan dan dikelola oleh google pada tahun 2011. Google menggunakan dart untuk membuat aplikasi berkualitas tinggi untuk iOS, android, dan web dengan fitur yang ditujukan untuk pengembangan sisi klien (Sharma & Gupta, 2020).

Pada awalnya dart dikembangkan sebagai pengganti dan penerus javascript. Dart mengimplementasi sebagian karakteristik dalam javascript (ES7) seperti "*async*" dan "*await*". Namun, untuk menarik pengembang yang tidak terbiasa dengan JavaScript, Dart memiliki sintaksis yang lebih mirip dengan java (Wu, 2018).

2.2.2 Widget

Widget merupakan komponen penting dalam kerangka kerja Flutter untuk membangun tampilan. Flutter memiliki model objek yang konsisten dan terpadu, sehingga setiap widget memiliki deklarasi dalam pembangunan tampilan yang tidak dapat diubah. Flutter memiliki struktur widget yang dapat disesuaikan dan diperluas, serta yang paling utama Flutter tidak menggunakan komponen atau widget OEM tetapi menyediakan widget sendiri (Fayzullaev et al., 2018).

Ada dua jenis widget utama yang ada dalam Flutter yaitu *stateless widget* dan *stateful widget*. *Statefull widget* memiliki *state* yang dapat berubah. *State* adalah informasi yang dapat dibaca secara bersama ketika sebuah widget dibuat dan dapat berubah selama pemakaian widget (Flutter Api, 2020). Sedangkan *stateless widget* tidak membutuhkan *state*, keduanya berada dalam level hierarki yang sama seperti terlihat pada Gambar 2.1.

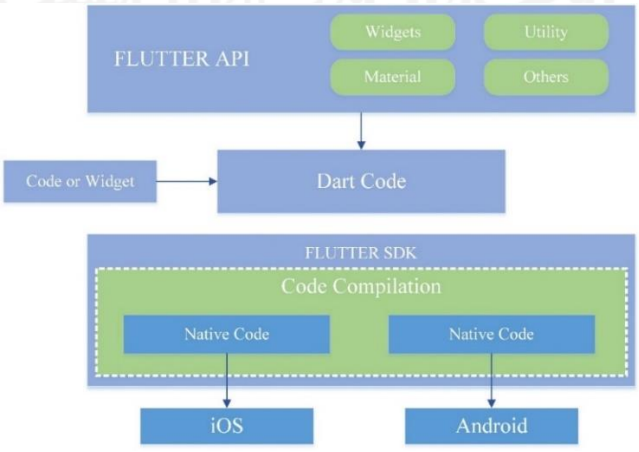


Gambar 2.1 Hierarki Flutter Class Widget.

Sumber: (Fayzullaev et al., 2018)

2.2.3 Arsitektur Flutter

Flutter menjalankan sebagian besar kerangka kerja dan kode aplikasi di dalam Dart *virtual machine* yang ringan. Kerangka kerja Flutter dibangun menggunakan Dart sedangakan rendering diimplementasikan dalam C++. Kode sumber dikompilasikan ke kode *native* menggunakan fitur kompilasi AoT (*Ahead of Time*) Dart seperti pada Gambar 2.2. Dalam android kode program C/C++ dikompilasi oleh AoT ke dalam kode *native* dengan NDK Android, sedangakan iOS akan dikompilasi dengan LLVM (*Low Level Virtual Machine*), dan juga semua kode program Dart (Fayzullaev et al., 2018).



Gambar 2.2 Arsitektur Flutter

2.3 React Native

React Native adalah kerangka kerja yang digunakan untuk memebuat aplikasi seluler *cross platform* yang pertama kali dikenalkan pada tahun 2015. React Native dirilis pertama kali pada awal tahun 2015 oleh facebook yang hanya mendukung pengembangan aplikasi iOS, tetapi pada bulan September 2015 kerangka kerja diperluas untuk mendukung pengembangan aplikasi android (Witte & Weitershausen, 2015).

React Native dibangun dengan prinsip dan konsep seperti ReactJS yang merupakan kerangka kerja *open source* pada 2013. Secara struktur kode sangat mirip, yang membedakan antara React Native dengan ReactJs adalah bahwa ReactJS beroperasi pada *Document Object Model* (DOM) dalam web browser, sedangkan React Native beroperasi pada aplikasi seluler (Hansson & Vidhall, 2016).

Sama halnya dengan Flutter yang memiliki fitur *hot-reload*, untuk mempercepat dalam pengembangan aplikasi, React Native memiliki fitur *fast refresh*. *Fast refresh* merupakan fitur yang memungkinkan pengembang mendapatkan upan balik secara langsung dalam perubahan *component*.

2.3.1 JSX

Dalam pengembangan, React Native menggunakan bahasa pemporgraman Javascript sebagai bahasa utamanya. Untuk mempermudah pembangunan aplikasi React Native, facebook mengembangkan ekstensi sintaksis JavaScript yang disebut JSX (JavaScript Syntax Transformer) yang bertujuan untuk memepermudah dalam membaca dan menulis kode program. JSX merupakan ekstensi untuk mengatur tampila. JSX akan dikompilasi menjadi objek Javascript normal ketika sedang dikompilasi (Wu, 2018). Penulisan sintaksis JSX dengan Javascrip regular tidak jauh berbeda, JSX terlihat mirip dengan XML, perbedaan penulisan sintaksis dalam React bisa dilihat pada Gambar 2.3.

```
var React = require('react');
var ReactDOM = require('react-dom');

ReactDOM.render(
  <h1> Hello World!<h1>;
  document.getById('example')
);
```

JSX

```
var React = require('react');
var ReactDOM = require('react-dom');

ReactDOM.render(
  React.DOM.h1 (null, "Hello World!"),
  document.getById('example')
);
```

JS

Gambar 2.3 Menulis Hello World dalam React dengan JSX dan JavaScript regular.

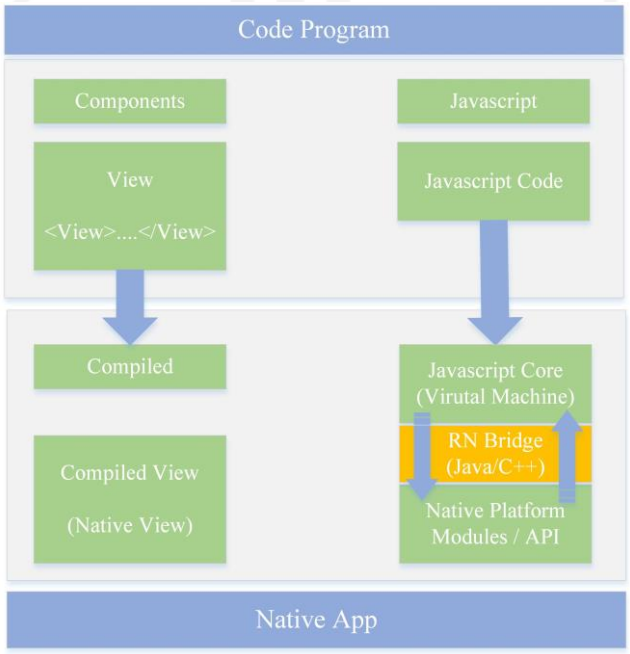
Sumber: (Hansson & Vidhall, 2016)

2.3.2 Component

Component merupakan bagian penting dalam pengbangunan tampilan pada React Native. Secara konsep, *component* seperti *function* Javascript digunakan untuk memebuat tampilan yang terbagi dalam beberapa bagian serta dapat digunakan kembali (ReactJS, 2020). *Component* memiliki dua model data yaitu *prop* dan *state*. *Prop* atau *property* biasanya digunakan untuk pertukaran data antara *component parent* dengan *component child*, sedangkan state hanya bisa dilakukan dalam suatu *component* dan data tidak bisa diakses dari *component* lain. State digunakan untuk mengelola data yang berubah seiring waktu yang berasal dari interaksi pengguna (*React Fundamentals*, 2020).

2.3.3 Arsitektur React Native

Arsitektur utama framework React Native terdiri dari javascript core (*virtual machine*), React Native *bridge*, dan modul native seperti terlihat pada Gambar 2.4. Kode program javascript berjalan pada *virtual machine* bersama *library* yang digunakan dalam pengembangan. Untuk menghubungkan kode program pada *virtual machine* dengan modul *native* dan *library* pihak ketiga, React Native menggunakan bridge (Kuitunen, 2019). React Native Bridge merupakan konsep paling penting dalam arsitektur React Native karena bridge digunakan untuk berkomunikasi dua arah dengan teknologi yang berbeda.



Gambar 2.4 Arsitektur React Native.

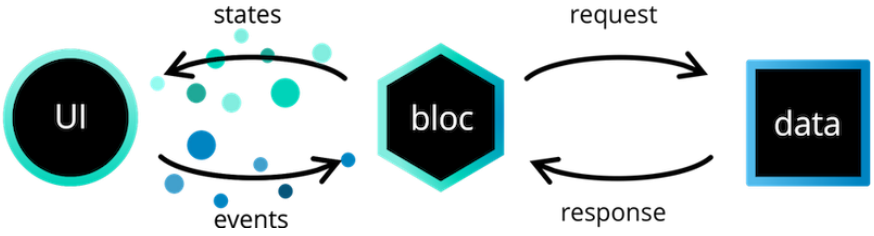
2.4 State Management

State management atau pola desain berfungsi untuk memisahkan bisnis logika dengan tampilan antarmuka. Pemilihan dan penggunaan *state management* berpengaruh terhadap proses pengembangan tertentu karena hampir semua bagian dari aplikasi bergantung pada management yang dipilih (Faust & Klein, 2020). Penggunaan *state management* berpengaruh terhadap domain penelitian yang digunakan terutama *performance*, salah satunya pertukaran *state* antara component. Ada banyak *state management* yang dapat digunakan dalam Flutter maupun React Native, tetapi pada penelitian ini menggunakan BloC sebagai *state management* Flutter dan Redux sebagai *state management* React Native karena kedua state management tersebut direkomendasikan oleh setiap pengembang kedua *framework* tersebut, selain itu juga BloC dan Redux sangat populer sehingga banyak digunakan oleh pengembang aplikasi.

2.4.1 Flutter BloC

The Business Logic Component atau Flutter BloC dibuat oleh Google dan di kenalkan dalam konferensi Google I/O pada tahun 2018, BloC merupakan *library* yang membantu untuk memisahkan antara *business logic* dan *presentation* seperti terlihat pada Gambar 2.5. Dalam BloC memungkinkan kita untuk memisahkan aplikasim menjadi 3 lapisan utama yaitu:

a. *Presentation* (UI), menampilkan antar muka sesuai dengan *state* BloC dengan me-*render* dirinya sendiri berdasarkan satu atau dua BloC *state*, dan juga menangani masukan dari pengguna dengan *events* kedalam BloC.
b. *Bussiness Logic*, semua perubahan state terjadi pada lapisan ini, *Bussiness Logic* akan menanggapi masukan dari lapisan *presentation* dengan *state* baru, dengan mengambil data pada satu atau beberapa *repository* untuk membangun *state* dalam aplikasi.
c. *Data* (*Repository* dan *Data Provider*), mengambil satu atau lebih sumber data seperti *database*, *network request*.



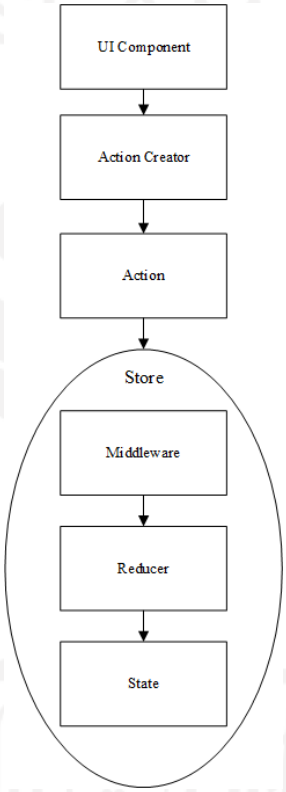
Gambar 2.5 Arsitektur data BloC.

Sumber: (*BloC Architecture*, 2020)

2.4.2 React Native Redux

Redux merupakan *library* javascript terbuka dengan menerapkan ide dasar dari pola flux yang digunakan untuk memanajemen *state*. *Redux* memiliki aliran data seperti terlihat pada Gambar 2.6. Aliran data tersebut terbagi dalam 3 bagian utama (Abramov, 2020).

a. *Action*, merupakan proses yang memuat informasi dengan mengirim data dari aplikasi menuju ke *store.*
b. *Reducer*, betugas mengolah data dengan merubah *state* menjadi *response*.
c. *Store*, bertugas untuk menggabungkan antara action dengan reducer.



Gambar 2.6 Arsitektur data Redux.

Sumber: (Kuparinen, 2019)

2.5 Domain

Menurut Gerdessen (2007), domain adalah bidang pengetahuan untuk masalah atau tugas tertentu dengan serangkaian konsep dan istilah yang terbatas. Tujuan digunakannya domain antara lain yaitu untuk mengurangi kompleksitas masalah dengan memisahkan menjadi area yang lebih kecil agar mudah untuk dikelola. Terdapat banyak domain yang dapat digunakan untuk mengkomparasikan *framewrok* seperti pada Tabel 2.1.

Tabel 2.1 Domain Komparasi

Sumber (Gerdessen, 2007)

No.	Domain
1	Scalability
2	Persistency
3	Security
4	Availability
5	Customisability
6	Interoperability
7	Skeleton implementation
8	Performance
9	Separation of concern
10	Modifiability
11	Testability

Dari beberapa domain pada Tabel 2.1 dipilih beberapa domain yang memungkinkan untuk diimplementasikan dengan melakukan analisis pada kerangka kerja berdasarkan domain yang paling berdapak pada pengembang. Sehingga didapat domain serta parameter yang digunakan dalam membandingkan kedua *framework* sebsgai berikut:

a. *Customisability*, kemudahan pengubahan perangkat lunak oleh penggguna dengan memodifikasi parameter desain (Jiao & Tseng, 2004). Pemilihan *customizability* untuk melakukan komparasi *framewrok* karena berdampak terhadap kemudahan pengembang pada saat memulai pembangunan aplikasi dan penambahan fitur aplikasi. Untuk mengukur *customizability* dengan melakukan pengukuran tingkat kustomisasi pada fitur yang ada pada masing-masing *framewrok*.

b. *Modifiability*, kemudahan dalam perubahan dan penambahan sistem (Saliu et al., 2009). Pemilihan *modifiability* untuk komparasi karena berdampak terhadap pengembang ketika melakukan perubahan dan penambahan fitur pada aplikasi. Untuk mengukur *modifiability*

dengan mengukur jumlah fungsi dan dependensi, pengukuran jumlah fungsi dan dependensi mempengaruhi proses lamanya pengembang dalam melakukan perubahan pada aplikasi karena banyak komponene yang harus diubah.

c. *Performance*, sejauh mana sistem atau perangkat lunak dalam mencapai tujuan dengan waktu yang ditentukan (Oshana, 2013). Pemilihan *performance* berdampak terhadap pengembang pada saat melakukan pembuatan aplikasi karena memudahkan pengembang dalam merepresentasikan *performance* aplikasi yang sedang dikembangkan terhadap *preformance* setelah aplikasi rilis. Pengukuran dapat dilakukan dengan mengukur parameter memori, CPU, dan *frame per second* karena sudah tersedia fitur pada *framewrok.*

d. *Testability*, karakteristik yang menunjukkan suatu komponen dalam menentukan status secara akurat (Garousi et al., 2019). Pemilihan *testability* berdampak terhadap pengembang ketika melakukan pengujian aplikasi agar sesuai yang diinginkan. Pengujian yang dapat dilakukan pada kedua kerangka kerja dapat dilakukan dengan mengimplementaskan *unit testing* dan *integration testing*.

2.6 Studi Literatur

Berikut ini merupakan referensi mengenai penelitian komparasi perangkat lunak yang menjadi inspirasi dalam penelitian ini:

Pada penelitian yang dilakukan oleh Fauzi Sholichin, dkk pada tahun 2019 dengan judul "*Review of IOS Architectural Pattern for Testability, Modifiability, and Performance Quality*", memebahas mengenai perbandingan empat pola arsitektur iOS yaitu MVC, MVP, MVVM, VIPER dengan tiga kriteria utama yaitu *testability*, *modifiability*, dan *performance*. Studi kasus yang digunakan dalam perbandingan yaitu aplikasi kontak telepon dengan tiga fitur yang diuji yaitu dafta kontak, pencarian kontak, detail kontak. Dari hasil pengujian *testability* dengan membandingkan banyak kode program dan waktu eksekusi dalam menjalankan aplikasi, MVVM lebih unggul dibandingkan arsitektur lainnya. Pengujian *modifiability* dilakukan dengan menganalisis dependensi suatu komponen dan menghitung jumlah fungsi, dengan hasil VIPER lebih unggul dalam pengujian dependensi komponen dan MVVM lebih unggul dengan sedikit jumlah fungsi. Pengujian *performance* dilakukan dengan mengukur konsumsi *Memory* dan *Central Processing Unit* (CPU), dengan hasil MVVM unggul dalam mengkonsumsi *memory*, dan VIPER unggul dalam mengkonsumsi CPU(Sholichin et al., 2019).

Pada penelitian yang dilakukan oleh Anton Gerdessen pada tahun 2007 dengan judul "*Framework comparison method comparing two frameworks based on technical domains, focussing on customisability and modifiability*". Penelitian ini membandingkan antara dua *framework* java yaitu Spring dan Blueprints yang berfokus pada kemudahan pengembangan dengan menggunakan domain *customizability* dan *modifiability*. Domain dibandingkan dengan membagi ke dalam 5 langkah. Langkah satu sampai empat akan mengukur apa yang dapat disesuaikan, sedangkan langkah kelima akan mengukur kemudahan dalam melakukan perubahan. Hasil penelitian domain *customizability* menunjukkan bahwa Spring menawarkan lebih baik dibandingkan Blueprint, sedangkan dalam penelitian *modifiability* Blueprint menawarkan sedikit lebih baik dibandingkan Spring (Gerdessen, 2007).

Pada penelitian yang dilakukan oleh Jakub Jagiello pada tahun 2019 dengan judul *"Performance Comparison between React Native and Flutter.* Penelitian ini membandingkan kinerja antara kerangka kerja React Native dan Flutter dengan mengukur *dropped frames, render time,* dan *miliseconds* pada *components* atau *widget.* Pengujian dilakukan terhadap dua aplikasi yang berbeda sebanyak 10 kali dengan durasi masing-masing 30 detik. Dari hasil pengukuran tidak ada perbedaan dalam pengukuran *millisecond* dan *render time.* Perbedaan hanya terdapat pada pengukuran *dropped frame,* dengan hasil React Native lebih unggul dibandingkan dengan Flutter (Jagie, 2019).

Pada penelitian yang dilakukan oleh Wenhao Wu pada tahun 2018 dengan judul "*React Native vs Flutter, Cross-platform Mobile Application Framewroks*". Penelitian ini membandingkan proses pengembangan dan kinerja antara kerangka kerja React Native dan Flutter, studi kasus yang digunakan dalam perbandingan menggunakan *Movie App.* Proses pengembangan berfokus pada *routing* dan *view,* sedangkan untuk mebangdingkan kinerja menggunakan *Scroll list* dan diukur menggunakan *FPS tracking.* Dari proses pengukuran menghasilkan FPS rata-rata lebih dari 60. Namun FPS pada *thread user interface* sepenuhnya tidak mengungkapkan kinerja React Native karena sebagian besar aplikasi React Native berjalan di *thread* Javascript (Wu, 2018).

Pada penelitian yang dilakukan oleh Faraz Quazi dan Nishant Sinha pada tahun 2018 dengan judul "*Android-Platform Based Detemination Fastest Cross-Platform Framewrok*". Penelitian ini membandingkan tiga kerangka kerja listas platform yaitu Ionic, React Native, dan PhoneGap dengan membandingkan aplikasi berdasarkan waktu eksekusi dengan satuan *miliseconds* yang diambil oleh kerangka kerja untuk melakukan operasi pengambilan status dan level baterai, perangkat, kontak, basis data, jaringan, dan perangkat keras pada android.

Dari hasil perbandingan dengan mengukur waktu eksekusi program, React Native dua kali lipat jauh lebih unggul dibandingkan dengan Ionic dan PhoneGap (Quazi & Sinha, 2018).

Pada penelitian yang dilakukan oleh Wei Sun, dkk pada tahun 2008 dengan judul *"Software as a Service: Configuration and Customization perspectives"*. Penelitian ini menjelaskan mengenai model konfigurasi dan kustomisasi SaaS (Software as a Service). Tujuan penelitian adalah membantu vendor SaaS untuk merencanakan dan mengevaluasi kemampuan dalam konfigurasi dan kustomisasi layanan. Strategi penelitian dilakukan dengan membagi menjadi 4 model pendekatan. Setelah menentukan model kemudian menentukan nilai kompetensi dengan membagi menjadi 6 bagian perspektif yaitu *Data Structure & processing, Organization Structure*, *User Interface*, *Workflow, Business Rule, Reporting*. Penilaian kompetensi dilakukan dengan mengukur 6 bagian perspektif dengan 5 tingkat kompetensi dari rendah ke tinggi yaitu *Entry, Aware, Capable, Mature,* dan *World Class*. Dengan perbedaan pendekatan memungkinkan tingkat kompetensi berbeda. Penelitian menghasilkan konfigurasi parameter pada tingkat *Aware* dengan parameter yang telah ditentukan(Sun et al., 2008).

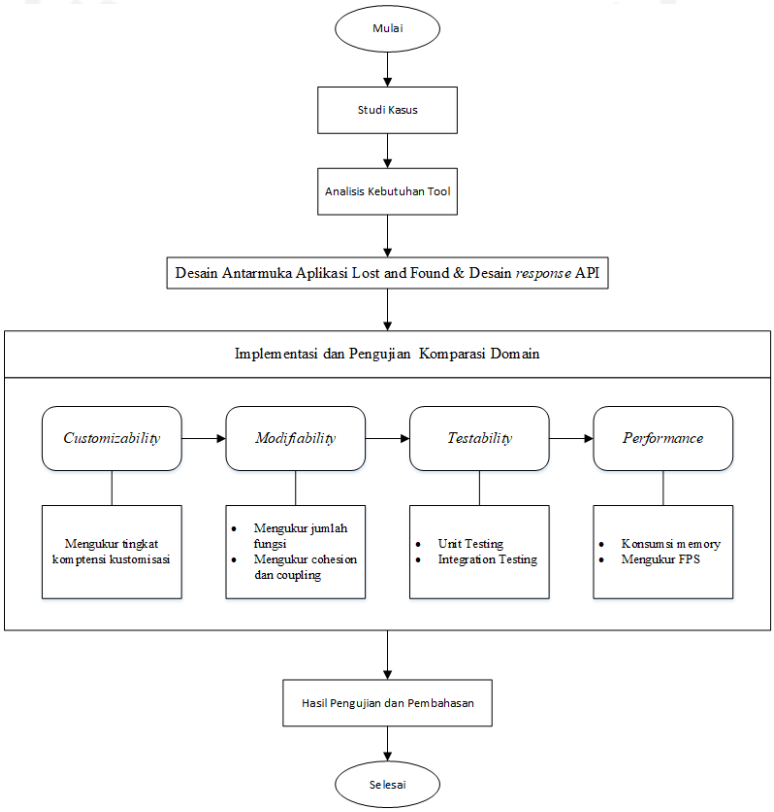
Pada penelitian yang dilakukan oleh Matias Salohonka pada tahun 2020 dengan judul "*Automated Testing of React Native Applications*". Penelitian ini menjelaskan mengenai kualitas perangkat lunak dengan melakukan otomatisasi pengujian dengan menggunakan kerangka kerja React Native. Tujuan penelitian adalah untuk mengidentifikasi dan mendeskripsikan berbagai jenis pengujian untuk aplikasi sesluler tertentu dan cara melakukan pengujian, sehingga dapat digunakan untuk memfasilitasi perusahaan dalam mengimplementasikan pengujian aplikasi. Pengujian yang dilakukan seperti *unit testing, view testing, state management testing, intergration testing,* dan *system testing.* Hasil penggunaan *testing* sangat relevan terhadap perusahaan yang sedang mengembangkan perangkat bergerak menggunakan React Native (Salohonka, 2020).

Pada penelitian ini menggunakan parameter pembanding yang digunakan pada penelitian (Sholichin et al., 2019) dan (Gerdessen, 2007) karena parameter sangat cocok dengan *framework* yang diuji. Parameter pembanding terdiri dari *customizability, modifiability, performace, dan testability*. Pengujian parameter *customizability* menggunakan jenis pengujian pada penelitian (Sun et al., 2008) karena memudahkan peneliti dalam menentukan tingkat perubahan pada perangkat lunak. Pengujian parameter *modifiability* menggunakan jenis pengujian pada penelitian (Sholichin et al., 2019) karena memiliki konsep arsitektur data yang tidak jauh berbeda. Pengujian parameter *performance* menggunakan jenis pengujian pada penelitian (Jagie, 2019) dan (Wu, 2018) karena memudahkan pengujian dengan menggunakan

fitur bawaan *framework*. Pengujian parameter *testability* menggunakan jenis pengujian pada penelitian (Salohonka, 2020) karena sebagian menggunakan pedoman pengujian dari dokumentasi *framework* yang sedang diuji.

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan dijelaskan langkah-langkah yang dilakukan dalam penelitian yang kemudian dijadikan acuan dalam meyelesaikan masalah dalam penelitian, menganalisis hasil penelitian, dan menulis laporan dari hasil penelitian. Alur metode penelitian dalam membandingkan *framework* dengan empat domain ditunjukkan pada Gambar 3.1. Penelitian ini memiliki beberapa langkah dimulai dari menganalisis alat, pembuatan aplikasi, pengujian aplikasi, dan pembahasan.



Gambar 3.1 Alur Metode Penelitian Komparasi

3.1 Studi Kasus

Studi kasus yang digunakan dalam penelitian ini adalah tata kelola barang temuan dan barang hilang (*Lost and Found App*). Studi kasus ini digunakan untuk membuktikan kualitas perangkat lunak menggunakan kedua *framework*. Penggunaan studi kasus ini difokuskan untuk pengujian pada *performance* dan implemenatsi *testability*. Dalam pengujian *performance* dibutuhkan fitur yang dapat menekan atau memberatkan memori dan CPU seperti pengguliran

daftar data, animasi, dan perpindahan data. Dalam implementasi *testability* studi kasus dibutuhkan dalam melakukan pengujian *unit testing* dan *integration testing* seperti pemanggilan API dan terdapat beberapa halaman yang terintegrasi. Studi kasus yang lainnya dapat digunakan jika terdapat beberapa hal tersebut misalkan seperti aplikasi kontak telepon.

3.1.1 Lost and Found App

Lost and Found App adalah aplikasi untuk mencari barang hilang dan barang temuan, dibuatnya aplikasi ini bertujuan untuk membantu dan mempermudah dalam menyebarkan atau mencari informasi dari barang sesuai kaidah Islam (Busra, 2019). Pembuatan aplikasi ditulis menggunakan Flutter dan React Native secara bersamaan. Terdapat empat halaman utama pada aplikasi yaitu:

a. Halaman utama *Lost and Found App*

   Halaman ini digunakan sebagai halaman awal aplikasi. Dalam halaman ini menampilkan menu navigasi menuju halaman daftar barang hilang dan temuan.

b. Halaman daftar barang hilang dan temuan

   Halaman ini menampilkan daftar barang hilang dan barang temuan. Data diambil dari server melalui API (*Aplication Programing Interface*) secara *asynchronous. Asynchronous* merupakan operasi yang dilakukan tanpa menunggu operasi sebelumnya selesai.

c. Detail barang

   Halaman ini menyajikan informasi detail dari setiap barang hilang maupun barang temuan. Untuk menampilkan data barang dengan melemparkan data dari halaman daftar barang sesuai item yang ditentukan menuju ke halaman detail tanpa mengambil data dari server.

d. Pencarian barang

   Halaman ini menyajikan daftar barang dari hasil pencarian tertentu.

3.1.2 Klasifikasi Studi Kasus

Agar dapat lebih mudah dalam memahami studi kasus, penulis mengklasifikasikan penelitian seperti terlihat pada Tabel 3.1. Tidak semua fitur digunakan karena beberapa fitur sudah mewakili dalam studi kasus. Studi kasus dikembangkan menggunakan *framework* Flutter dan React Native dengan sumber data didapatkan dari database lokal dengan menggunakan API (*Aplication programing Interface*) yang bisa dikonsumsi dengan format JSON.

Tabel 3.1 Klasifikasi Studi Kasus

Kriteria	**Lost and Found App**
Fitur yang diuji	- Daftar barang hilang dan barang temuan - Mencari barang hilang dan barang temuan - Detail barang hilang dan barang temuan
Sumber data	API (*Aplication programing Interface*)
Kerangka kerja	Flutter (Dart), React Native (Javascript), Codeigniter (PHP)
Satate management	Flutter BloC (flutter_bloc), React Redux
Kualitas yang akan diukur	*Customizability*, *Modifiability*, *Perfromance*, dan *Testability*

3.1.3 Analisis Kebutuhan Tool

Tahap ini dilakukan untuk menganalisis kebutuhan dari aplikasi yang akan dibuat. Kebutuhan yang digunakan dalam penelitian adalah kebutuhan nonfungsional. Kebutuhan nonfungsinal adalah kebutuhan yang menitikberatkan pada properti perilaku yang dimiliki oleh sistem.

Kebutuhan nonfungsional terdiri dari perangkat keras dan perangkat lunak yang digunakan dalam membangun dan menguji aplikasi *Lost and Found App.* Kebutuhan ini berpengaruh terhadap pengujian yang dilakukan terutama pada pengujian *performance,* semakin bagus alat yang digunakan dalam pengujian maka hasil akan lebih akurat karena tidak terlalu mempengaruhi dalam masa pengujian. Perbedaan perangkat yang digunakan memungkinkan adanya perbedaan hasil pengujian. Kebutuhan nonfungsionalnya adalah sebagai berikut:

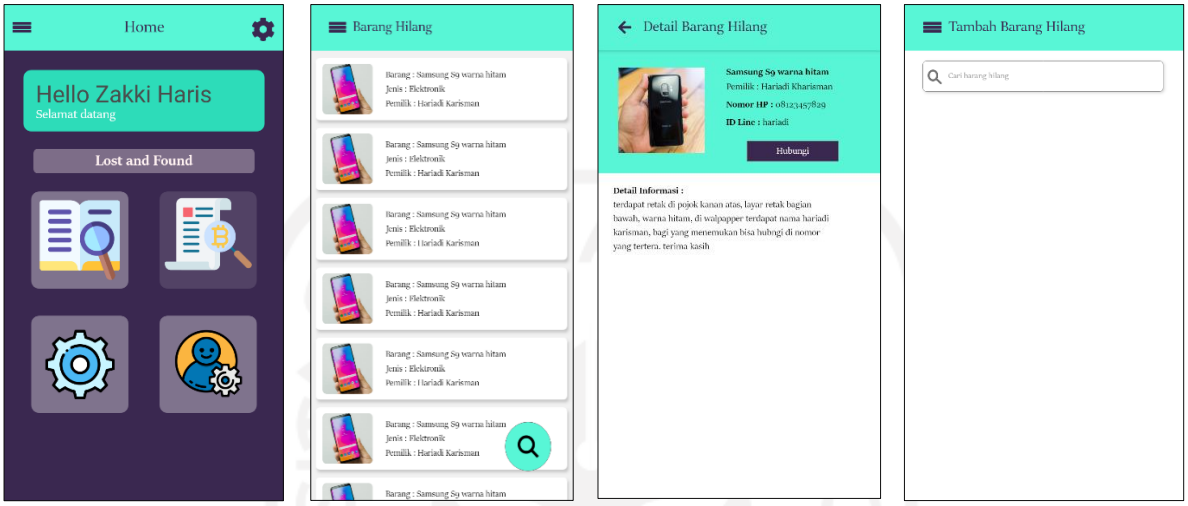
a. Perangkat laptop
    - *Notebook* ASUS X550ZE
    - Prosesor AMD A10 7400P
    - RAM DDR3 8GB
    - SSD 120GB
    - Sistem Operasi Windows 10
    - Text editor dan compiler Visual Studio Code
b. Perangkat gawai
    - OPPO A37
    - Versi Android 5.1.1 (Lolipoop)
    - Prosesor Quad-core 1.2 GHz
    - RAM 2GB

3.1.4 Desain Halaman Aplikasi dan Bentuk Respons API

Untuk memudahkan penelitian diperlukan sebuah rancangan awal dalam membuat aplikasi. Terdapat desain halaman dan bentuk response API yang nantinya akan digunakan sebagai acuan dalam pembuatan aplikasi pada masing-masing *framework* React Native maupun Flutter. Dengan demikian dapat dipastikan aplikasi yang dibuat tidak jauh berbeda satu dengan lainnya.

Terdapat empat halaman pada desain rancangan apikasi yaitu *home*, daftar barang, detail barang, dan cari barang seperti terlihat pada Gambar 3.2. Setiap desain halaman sudah dijelaskan pada poin 3.1.1 mengenai *Lost and Found App.* Sedangkan untuk respons API berbentuk JSON (*JavaScript Object Notation*) yang mewakili satu buah barang seperti pada Gambar 3.3.

Respons dibuat simple agar memudahkan dalam pemanggilan data pada setiap *framework*. Respons akan dibuat sebanyak 50 buah barang agar memberatkan aplikasi sehingga memudahkan dalam pengukuran pada domain *performance*. API akan dijalankan pada *server* lokal yang bertujuan untuk meminimalkan perbedaan kecepatan respons pada saat pengukuran.



Gambar 3.2 Desain Aplikasi


```
{
        "id": "133",
        "id_user": "59",
        "nama": "Zakki Haris",
        "peran": "Mahasiswa",
        "nama_barang": "Laptop",
        "jenis_barang": "Elektronik",
        "gambar": "gambar",
        "detail_informasi": "Laptop merek ASUS warna hitam abu-abu dengan
         spesifikasi AMD A10 Quad core Radeon dual-graphics , didalam
         casingnya terdapat stiker logo UII dan logo android",
        "tempat_kehilangan": "Fotokopi Azzahra Kalurang KM 7",
        "waktu_kehilangan": "2020-01-16",
        "nomor_hp": "089674276651",
        "id_line": "jack",
        "status": "0",
        "waktusekarang": "2020-01-23 21:39:58"
}
```


Gambar 3.3 Bentuk Respons JSON API

3.2 Customizability

Customizability merupakan kemudahan pengubahan perangkat lunak oleh penggguna dengan memodifikasi parameter desain (Jiao & Tseng, 2004). Menurut Sun, dkk (2008), terdapat 5 tingkat model kompetensi dalam menentukan tingkat kustomisasi seperti terlihat pada Tabel 3.2.

Tabel 3.2 Model Tingkat Kompetensi Kustomisasi

Kompetensi	**Deskripsi**	**Tingkat variansi**
Entry	Tingkat standar tanpa dukungan kustomisasi	Tidak ada
Aware	Perubahan relative standar yang sudah ditentukan sebelumnya	Rendah
Capable	Perubahan relative standar yang ditentukan oleh pengguna	Sedang
Mature	Perubahan dasar dengan lingkungan yang dapat di program untuk memungkinkan kustomisasi pilihan pengguna	Tinggi
World Class	Perubahan yang didukung oleh model pemprograman dan alat untuk memungkinkan kostumasi yang sangat kuat	Sangat tinggi

Model ini dapat digunakan untuk menilai prespektif kerangka kerja dalam pembuatan perangkat lunak dalam aspek kustomisasi, semakin tinggi tingkat kompetensi prespektif klien maka semakin banyak potensi kerangka kerja dalam penyesuaian pengembangan. Untuk menentukan tingkat dalam kustomisasi terdapat 2 perspektif yaitu perspektif dari pengembang dan pengguna *framewrok*, dalam kasus ini kami mencoba menganalisis dari perspektif pada pengguna karena tujuan penelitian berfokus pada pengembangan aplikasi. Terdapat 4 prespektif yang akan diukur yaitu: *components*, *file structure*, dan *configuration.* Hasil dari perbandingan akan dibuat dalam bentuk tabel yang memuat prespektif dan tingkat kompetensi. Contoh tabel yang digunakan untuk memuat tingkat kompetensi dapat dilihat pada Tabel 3.3.

Tabel 3.3 Contoh Tabel Model Tingkat Kompetensi Kustomisasi

Prespektif	**Tingkat Kompetensi**				
	Entry	**Aware**	**Capable**	**Mature**	**World Class**
Components					
File Structure			● ●		
Setup and Configuration					

● React Native

● Flutter

Pada contoh Tabel 3.3 menejelaskan contoh hasil bahwa prespektif dari data structure memiliki tingkat kompetensi *capable*, maka perubahan yang berada dalam *framework* relative satandar yang dilakukan oleh pengguna. Tingkatan kompetensi *capable* dikategorikan dalam tingkat kustomisasi sedang.

3.3 Modifiability

Modifiability dapat diukur menggunakan metrik evaluasi dengan menentukan jumlah fungsi, cohesion, dan coupling (Zhao et al., 2011).

3.3.1 Mengukur Jumlah Fungsi.

Jumlah fungsi sangat mempengaruhi proses modifikasi suatu sistem yang dibangun. Semakin banyak fungsi komponen dalam sistem, semakin kompleks pula dalam modifkasi (Bachmann et al., 2007). Penghitungan jumlah fungsi komponen dilakukan menggunakan pendekatan *Functional Size Measurement* (FSM) dengan mengevaluasi fungsi menggunakan file, antar muka, dan interaksi antara aplikasi. FSM dapat dipetakan menjadi 5 faktor fungsional seperti terlihat pada Tabel 3.4(Zhao et al., 2011).

Tabel 3.4 Lima Faktor Fungsional

Faktor Fungsional	Deskripsi
Internal logic file	Menyimpan data yang digunakan dalam suatu komponen
Internal interface file	Berisi data external yang diterima dari lingkuangan operasional
External input	Input yang menyimpan data exernal
External output	Mengembalikan data ke lingkungnan operasional
External inquires	Mengnakap akses data

Untuk mengevaluasi faktor fungsional, FSM memiliki tiga tingkat kompleksitas yaitu tingkat rendah, sedang dan tinggi. Tingkat kompleksitas dapat dikatakan rendah jika memiliki kurang dari 50 titik fungsi dan setiap tingkat kompleksitas memiliki bobot faktor fungsional masing-masing (Sholichin et al., 2019). Adapun bobot faktor fungsional yang sesuai dengan lima faktor fungsional dapat dilihat pada Tabel 3.5(Zhao et al., 2011).

Jumlah fungsi dihitung dengan mengalikan bobot faktor fungsional dengan jumlah faktor fungsional yang ada. Misalkan terdapat komponen yang memiliki 2 fungsi *external input*, maka 2 fungsi dikali dengan bobot faktor fungsional *external input* yaitu 3, maka jumlah poin fungsinya adalah 6 poin. Hasil akhir dihitung dengan menjumlahkan setiap poin yang berada pada semua faktor fungsional.

Tabel 3.5 Bobot Faktor Fungsional dalam FSM

Faktor Fungsional	Rendah	Sedang	Tinggi
Internal logic file	7	10	15
Internal interface file	5	7	10
External input	3	4	6
External output	4	5	7
External inquires	3	4	6

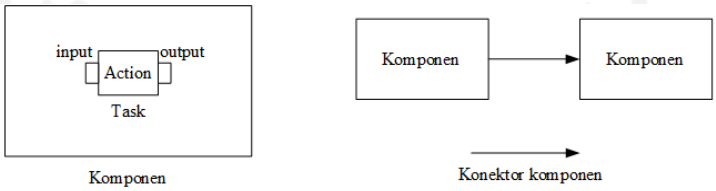
3.3.2 Mengukur Cohesion dan Coupling

Cohesion digunakan untuk mengukur kekuatan dependensi dalam suatu komponen arsitektur, sedangkan *coupling* menilai kekuatan saling ketergantungan antara komponen arsitektur, suatu arsitektur dapat dikatakan baik ketika *cohesion* memiliki nilai tinggi dan *coupling* rendah (Zhao et al., 2011).

Langkah utama dalam melakukan pengukuran yaitu menentukan tingkat cohesion dan coupling dengan cara menganalisis karakteristik komponen pada arsitektur atau pola desain yang digunakan. *Coupling* dan *cohesion* memiliki karakteristik data dan tugas dalam suatu komponen untuk menentukan tingkat *coupling* dan *cohesion* yang akan diukur. Beberapa karakteristik dapat dilihat pada Tabel 3.6 untuk *cohesion* dan Tabel 3.7 untuk *coupling* (Zhao et al., 2011).

Setelah menentukan tingkat *cohesion* dan *coupling* yang sesuai dengan pola desain yang dipakai, kemudian menentukan komponen dari pola desain yang digunakan dengan melakukan analisis pada alur kerja *state management* seperti pada Gambar 3.4. Setiap komponene memiliki beberapa *task* atau tugas seperti *human taks*, *automatic task*, dan *data-acces task* (Zhao et al., 2011).

a. *Human task*, berinteraksi dengan pengguna untuk mengambil masukan dari pengguna dan menampilkan hasil dari permintaan pengguna.
b. *Automatic task*, melakukan komputasi dalam komponen atau perangkat lunak.
c. *Data-acces task*, tugas yang memanipulasi *database* seperti *insert*, *update*, *query*, dan *delete*.



Gambar 3.4 *Components*

Dari beberapa komponen dan *task* yang didapat kemudain digunakan untuk mengukur jumlah dependensi pada masing-masing tingkat dengan persamaan (3.1) untuk *cohesion* dan persamaan (3.2) untuk *coupling*. Secara garis besar pengukuran ini lebih merujuk pada *state management* yang digunakan pada masing-masing *framework* yaitu BloC dan Redux.

Tabel 3.6 Tingkat *Cohesion* dan Karakteristiknya

Kategori	**Tingkat *cohesion***	**Karakteristik**
Tingkat tinggi	*Functional*	Komponen yang melakukan satu tugas.
Tingkat menengah	*Squential*	Data berurutan dikirim ke seluruh komponen.
	Communicational	Komponen berbagi data masukan atau keluaran yang sama.
	Procedural	Tugas dalam sebuah komponen dihubungkan dengan konektor kontrol
Tingkat rendah	*Temporal*	Tugas suatu komponen dihubungkan oleh hubungan temporal.
	Logical	Tugas dikelompokkan untuk menjalankan fungsi yang sama.

Tabel 3.7 Tingkat *Coupling* dan Karakteristiknya

Kategori	**Tingkat *coupling***	**Karakteristik**
Tingkat tinggi	*Content*	Komponen menggunakan data atau control yang dikelola oleh komponen lain.
	Common	Komponen berbagi data global.
	External	Komponen terikat dengan entitas luar seperti peragkat atau data dari luar.
Tingkat menengah	*Control*	Kontrol mengalir diseluruh komponen.
Tingkat rendah	*Data structure*	Data terstruktur ditransfer antar komponen.
	Data	Data primitif diteruskan diantara komponen.
	Message	Komponen berkomunikasi memulai antarmuka standar.

Cohesion

Kekuatan *cohesion* suatu komponen dievaluasi dengan menganalisis jumlah dependensi dalam komponen seperti pada persamaan (3.1). Kekuatan *cohesion* akan meningkat dengan bertambahanya jumlah ketergantungan dalam komponen (Zhao et al., 2011).

$$S_{Cohesion} = \frac{\sum_{i=1}^{m} A_i}{m}, A_i = \frac{\mu_i}{n_i^2} \quad (3.1)$$

Ai adalah kekuatan ketergantungan dalam komponen, µi adalah jumlah dependensi dalam komponene ke-I, ni adalah jumlah tugas dalam komponene dan m adalah jumlah komponene dalam arsitektur.

Coupling

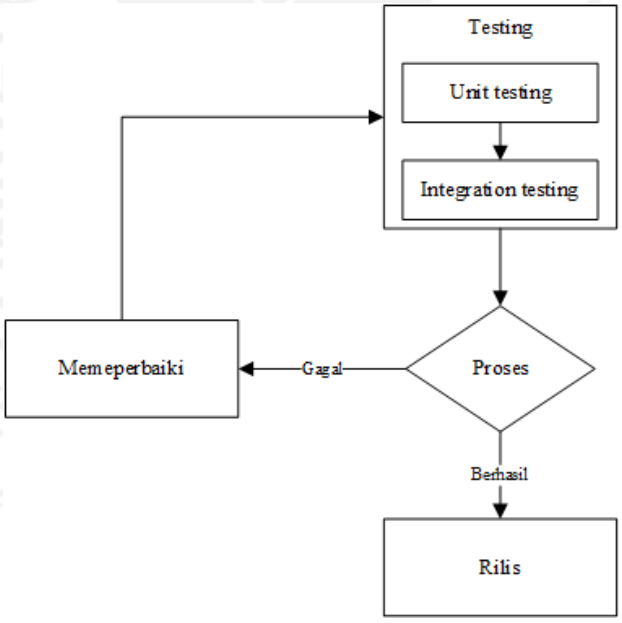
Kekuatan *coupling* diukur berdasarkan rata-rata kekuatan antara komponen, untuk mengukur nilai *coupling* antara dua komponen dengan memeriksa jumlah dependensi antara dua komponen menggunakan persamaan (3.2).

$$S_{Coupling} = \frac{\sum_{i,j=1}^{m} E_{i,j}}{m(m-1)/2}, E_{i,j} = \begin{cases} 0 & i = j \\ \varepsilon_{i,j} & i \neq j \\ 2n_i n_j & \end{cases} \quad (3.2)$$

Ei,j adalah saling ketergantungan antara komponen ke-i dan komponene ke-j. εi,j adalah jumlah total dependensi antara dua komponen. ni dan nj adalah jumlah tugas dalam dua komponen dan m adalah jumlah total komponen dalam arsitektur.

3.4 Testability

Semaikin banyak fitur yang ada dalam sebuah aplikasi maka semakin sulit untuk mengujinya. Flutter maupun React Native memiliki cara pengujian masing-masing seperti unit testing, integration testing. *Testability* sangat erat kaitanya dengan pengujian *Modifiability* salah satunya dalam masalah ketergatungan suatu *components*. Alur untuk melakukan testing dimulai dari *unit testing* kemudian *integration testing*, jika dalam *testing* terdapat kegagalan (*failed*) pada program maka akan diperbaiki dan dilakukan *testing* kembali, jika berhasil (*passed*) maka proses *testing* selelsai (Romeo, 2003). Alur *testing* seperti terlihat pada Gambar 3.5. *Unit testing* menguji sebagian kecil dari kode program seperti *function, method,* atau *class*. Sedangkan *integration testing* menguji aplikasi lengkap dari berbagai fungsi.



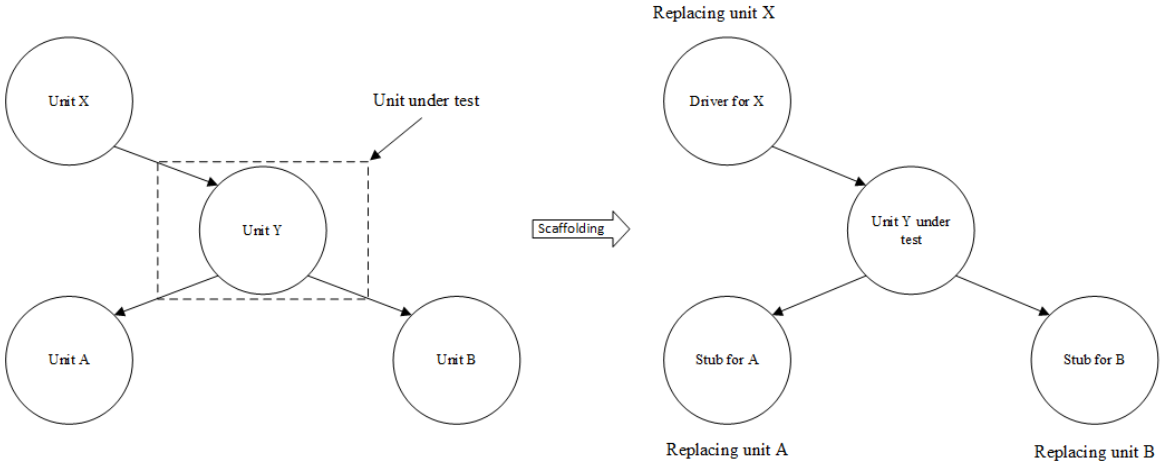
Gambar 3.5 Alur Testing

3.4.1 Unit Testing

Unit testing merupakan pengujian paling umum dilakukan dalam suatu organisasi. *Unit testing* menguji bagian kecil dari suatu kode seperti *function*, *method*, dan *class.* Tujuan dari pengujian ini adalah untuk memastikan kebenaran suatu unit logika dalam berbagai kondisi.

Unit testing umumnya tidak membaca atau menulis dalam penyimpanan, menampilkan ke layar atau menerima tanggapan oleh pengguna (*Testing Flutter App*, 2020).

Dalam melakukan unit testing, sebuah unit tidak sepenuhnya independen, setiap unit memiliki ketergantungan terhadap unit lainnya, sehingga mengakibatkan kesulitan dalam pengujian. Unit dapat terdiri dari satu unit memanggil beberapa unit lain atau hanya tediri dari satu unit. Ketergantungan suatu unit dapat diselesaikan dengan memalsukan ketergantungan dengan menuliskan *stub* dan *driver* yang menggantikan ketergantungan unit seperti terlihat pada Gambar 3.6. *Stub* akan menggantikan unit yang dipanggil oleh komponen yang di tes dengan antar muka unit, sehingga dapat melakukan manipulasi data dan kemudian mengembalikan unit yang sedang diuji. *Driver* kebanyakan dari berasal dari program utama yang menerima tes dengan memasukkan data kedalam komponen yang sedang dilakukan tes, kemudian mencetak hasilnya (Romeo, 2003).



Gambar 3.6 Unit testing dengan *stub* dan *driver*
Sumber: (Salohonka, 2020)

Alat yang digunakan untuk melakukan unit testing pada React Native menggunakan *libtary* jest. Jest merupakan library resmi yang digunakan oleh Facebook untuk melakukan *testing* pada react maupun React Native. Sedangkan untuk Flutter menggunakan *library* test atau flutter_test yang terdapat pada dokumentasi resmi Flutter. Untuk mengimplementasikan *stub* membutuhkan library mockito.

3.4.2 Integration Testing

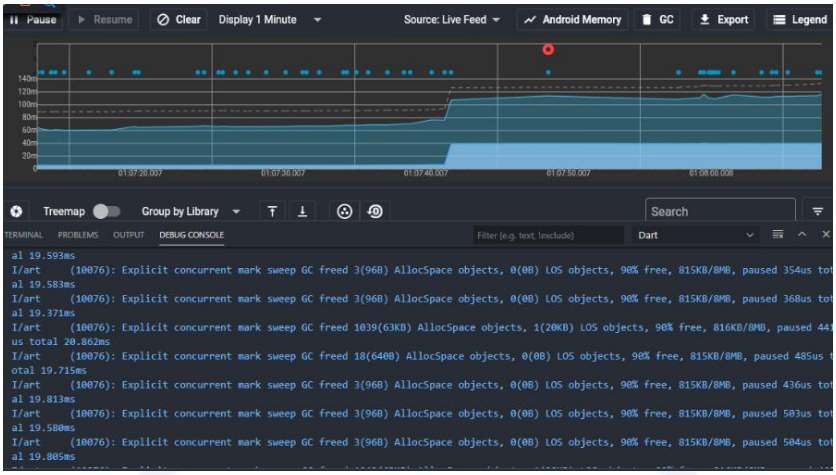
Dalam pengujian unit individu akan digabungkan dan diuji bersama, pengujian ini bertujuan untuk memastikan bahwa semua widget dan layanan yang diuji berfungsi seperti yang diharapkan (*Testing Flutter App*, 2020). Integration testing menindaklanjuti pengujian widget yang sudah dilakukan dalam unit testing. Integration testing dilakukan dengan secara bersamaan pada suatu program menggunakan library pengujian pada masin-masing *framework* seperti React Native menguakan *library* jest, sedangkan Flutter menggunakan *library* flutter_driver.

3.5 Performance

Keterbatasan *performance* perangkat seluler sangat berpengaruh dalam pengembangan, *performance* merupakan faktor utama untuk memilih metode pengembangan perangkat lunak karena kinerja aplikasi seluler sangat penting untuk pengalaman dan kepuasan pengguna. Dalam penelitian ini menggunakan *framework cross-platform* yang memiliki sistem arsitektur yang berbeda dalam penerjemahan kode program kedalam *native* yang sangat mempengaruhi *performance.* Pengukuran *performance* menggunakan beberapa mode seperti mode *debug* pada React Native dan Flutter serta mode *profile* pada Flutter (Jagie, 2019). Mode *profile* memungkinkan untuk menjalankan aplikasi hampir sama seperti mode rilis dalam masa pengujian. Ada beberapa aspek yang digunakan dalam mengukur performance yaitu:

3.5.1 Konsumsi Memori

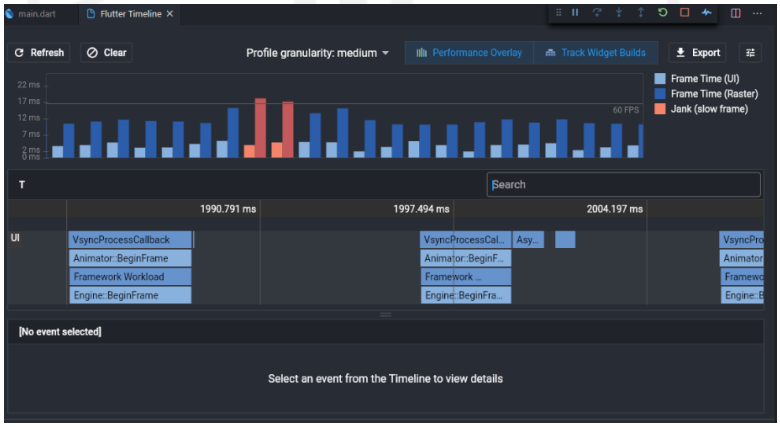
Mengukur konsumsi memori bedasarkan rata-rata alokasi memori dalam milidetik (Jagie, 2019). Untuk menghitung memori yang dikonsumsi menggunakan alat yaitu *DevTools* bawaan *framework* seperti terlilhat pada Gambar 3.7. Selain menggunakan *DevTools* dalam pengukuran juga menggunakan fitur *profiler* pada android studio agar tidak ada keraguan dalam hasil karena perbedaan *DevTools* yang digunakan. Pengukuran dilakukan dengan menggulir daftar pada aplikasi dengan meminta data dari API dan melempar data dari suatu halaman ke halaman yang berbeda. Semakin sedikit aplikasi dalam mengkonsumsi memori maka semakin baik kinerjanya.



Gambar 3.7 *DevTools* Pengukuran Konsumsi Memori pada Flutter.

3.5.2 Frame per Second

FPS (*Frame per second*) merupakan angka untuk menunjukkan berapa banyak *frame* yang di-*render* selama satu detik (Wu, 2018). Untuk mengetahui FPS menggunakan alat yaitu *DevTools* bawaan *framework* seperti terlilhat pada Gambar 3.8. Pengukuran dilakukan dengan menggulir daftar pada aplikasi dengan meminta data dari API dan melempar data dari suatu halaman ke halaman yang berbeda. Semakin banyak jumlah FPS yang dihasilkan maka gerakan akan halus dan lancer.



Gambar 3.8 *DevTools* Pengukuran FPS pada Flutt

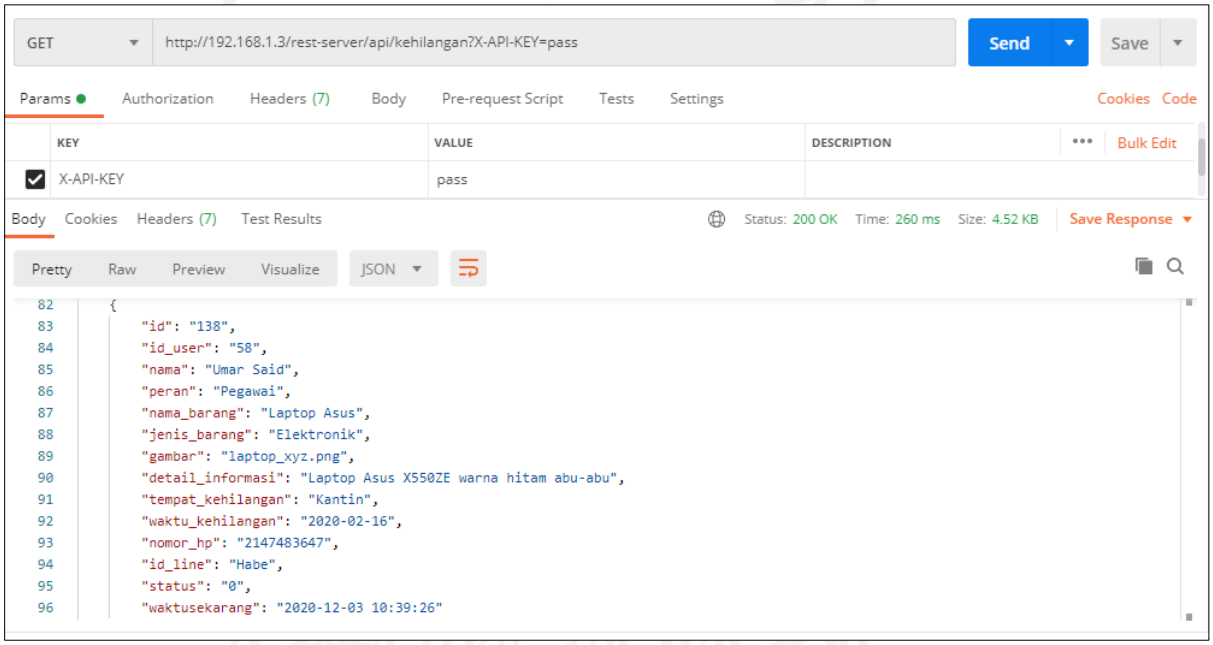
BAB IV
HASIL DAN PEMBAHASAN

4.1 Tahap Implementasi

Hasil implementasi akan dijelaskan berdasarkan domain perbandingan yaitu customizability, modifiability, performance, dan testability.

4.1.1 API (Application Programming Interface)

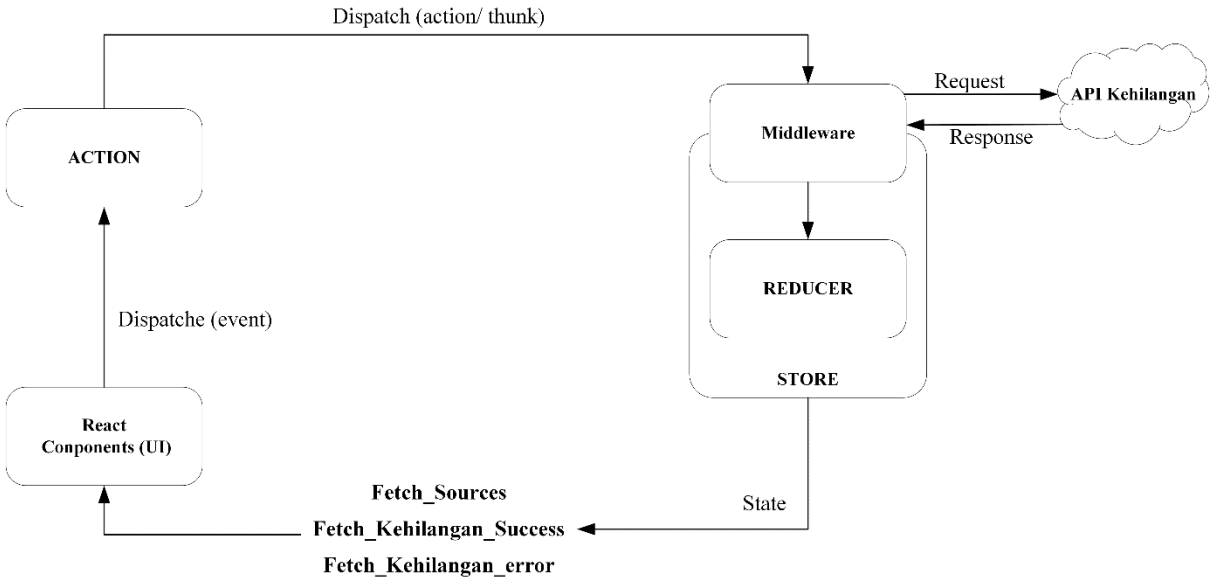
API dalam studi kasus ini dibuat menggunakan bahasa pemprograman PHP dengan framework codeigniter. Terdapat 50 data kehilangan berbentuk JSON yang digunakan untuk menguji aplikasi, dan setiap datanya dapat dilihat pada Gambar 4.1.



Gambar 4.1 *Request* API

4.1.2 React Native Redux

Redux memiliki tiga komponen utama yaitu *action*, *reducer*, *store*. Implementasi Redux pada React Native memiliki peran untuk melakukan perubahan *state* yang dibutuhkan oleh komponen. Alur kerja Redux dalam merubah *state* pada aplikasi *Lost and Found* dapat dilihat pada Gambar 4.2.



Gambar 4.2 Alur Kerja Redux

Pada Gambar 4.2 alur kerja Redux dimulai dari bagian *react components* dengan mengirimkan *event* menuju pada bagian *action.* Bagian *action* terdiri dari beberapa *action creator* bertipe objek seperti terlihat pada Gambar 4.3.

```
17  export const fetchKehilangan = () => ({
18      type: FETCH_KEHILANGAN,
19  });
20
21  export const fetchSources = () => ({
22      type: FETCH_SOURCES,
23  });
24
25  export const fetchKehilanganSuccess = (payload) => ({
26      type: FETCH_KEHILANGAN_SUCCESS,
27      payload
28  });
29
30  export const fetchKehilanganError = error => ({
31      type: FETCH_KEHILANGAN_ERROR,
32      error
33  });
```

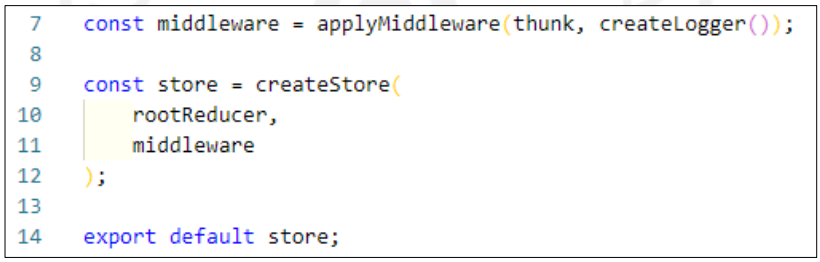
Gambar 4.3 Action React Native Redux

Dari bagian *action* yang berbentuk objek kemudian dikirim ke bagian *store. Store* merupakan *global state* yang bertugas untuk menggabungkan antara *action* dengan *reducer.* Bagian dalam *store* terdapat *middleware* dan *reducer* seperti terlihat pada Gambar 4.2 dan Gambar 4.5. *Middleware* digunakan untuk merubah hasil dari *request API* sebelum masuk

menjadi *response*. Hasil perubahan yang dilakukan oleh *middleware* berupa data *asynchronous (async)* yang nantinya menjadi *response* untuk dikirimkan kebagian *reducer* melalui *store.* Proses pemanggilan dan perubahan menjadi *asynchronous* dapat dilihat pada Gambar 4.4. *Library* yang digunakan untuk membuat *middleware* yaitu *redux-thunk.*

```
export const callApiKehilangan = async (URL) => {
    try {
        const res = await fetch(URL);
        if (res.status >= 400) {
            throw new Error("Bad request from server");
        }else if(res.status >= 204){
            throw new Error("id not found or data empty");
        }
        const response = await res.json();
        return response;
    } catch (err) {
        return { data: err.message };
    }
}

export function fetchSources() {
    return dispatch => {
        dispatch({ type: 'FETCH_SOURCES' });
        callApiKehilangan('http://192.168.1.3/api/kehilangan?X-API-KEY=pass')
            .then(response => {
                dispatch({ type: 'FETCH_KEHILANGAN_SUCCESS', payload: response });
            })
            .catch(err => {
                dispatch({ type: 'FETCH_KEHILANGAN_ERROR', payload: err });
            })

    };
}
```

Gambar 4.4 Request & Response Redux Thunk



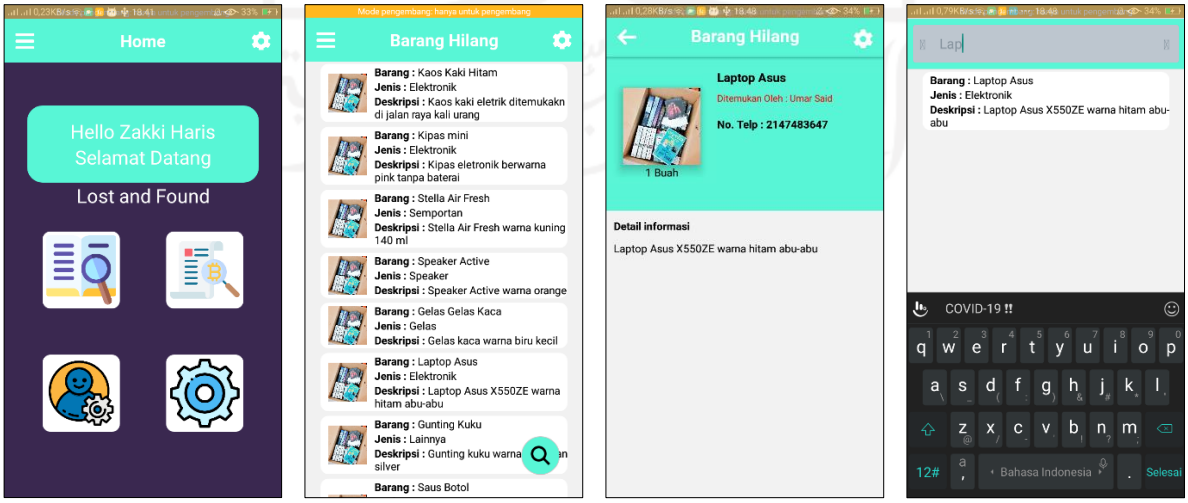
```
const middleware = applyMiddleware(thunk, createLogger());

const store = createStore(
    rootReducer,
    middleware
);

export default store;
```

Gambar 4.5 Store React Native Redux

Setelah melakukan perubahan data melalui *middleware*, kemudian mengarah ke dalam *reducer*. Reducer digunakan untuk merubah *state* berdasaran tipe *action* yang dikirimkan dari *middleware* pada Gambar 4.4. Terdapat tiga tipe *action* yang dikirim dari *middleware* menuju *reducer* yaitu "FETCH_SOURCES", "FETCH_KEHILANGAN_SUCCESS", dan "FETCH_KEHILANGAN_ERROR". *Reducer* akan mengembalikan *state* sesuai tipe *action* seperti pada Gambar 4.6 dan menuju bagian *react components* untuk ditampilkan dalam layar. Hasil implementasi redux pada React Native dapat dilihat pada Gambar 4.7.

```
4   const initialState = {
5       loading: false,
6       data: [],
7       error: null
8   };
9
10  const sourcesKehilangan = (state = initialState, action) =>
11      switch (action.type) {
12          case 'FETCH_SOURCES':
13              return {
14                  ...state,
15                  loading: true,
16                  error: null
17              };
18          case 'FETCH_KEHILANGAN_SUCCESS':
19              return {
20                  ...state,
21                  loading: false,
22                  error: null,
23                  data: action.payload,
24              };
25          case 'FETCH_KEHILANGAN_ERROR':
26              return {
27                  ...state,
28                  loading: false,
29                  error: action.error,
30                  data: []
31              };
32          default:
33              return state;
34      }
35  };
36
37  export default sourcesKehilangan;
```

Gambar 4.6 Reducer React Native Redux

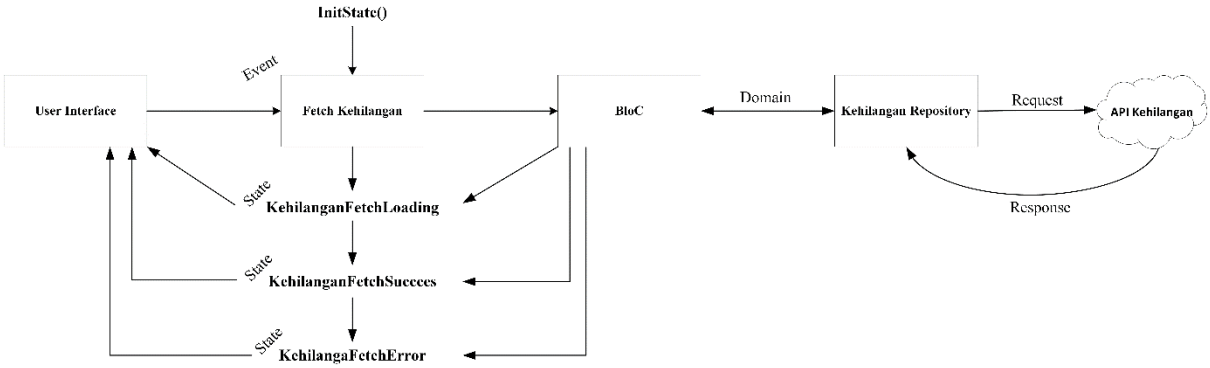


Home Daftar Barang Detail Barang Cari Barang

Gambar 4.7 Hasil Aplikasi React Native

4.1.3 Flutter BloC

Terdapat tiga bagian penting dalam BloC yaitu *Persentation* (*User Interface*), *Bussiness Logic (BloC), dan Data (Repository).* Ada beberapa cara untuk mengimplementasikan BloC pada Flutter, tetapi dalam kasus ini menggunakan *library* flutter_bloc. Alur kerja BloC dalam mengelola state pada Flutter dapat dilihat pada Gambar 4.8.



Gambar 4.8 Alur Kerja BloC

Alur kerja BloC pada Gambar 4.8 dimulai dari *User Interface (UI). User Interface* akan mengirimkan *action* atau *event* menuju BloC berupa objek seperti pada Gambar 4.9.

```
3   abstract class KehilanganEvent extends Equatable {
4     const KehilanganEvent();
5
6     @override
7     List<Object> get props => null;
8   }
9
10  class KehilanganFetching extends KehilanganEvent {}
```

Gambar 4.9 Event Flutter BloC

Setelah menerima *event* dari *user interface*, pada Gambar 4.10 BloC akan melakukan *request* kepada *repository* melalui perantara domain pada Gambar 4.11. Kemudian *repository* akan melakukan *response* berupa data yang diambil dari API dalam bentuk List *asynchronous* pada Gambar 4.12. Setelah menerima *response* BloC akan mengirim state berupa data sesuai dengan *event*. Terdapat tiga *event* yang akan dikirim oleh *state* yang berasal dari BloC yaitu "KehilanganFetchLoading", "KehilanganFetchSuccess", dan "KehilanganFetchError". Setelah itu *state* akan meneruskan data sesuai *event* menuju *user interface* untuk ditampilkan

dilayar seperti pada Gambar 4.13. Hasil implementasi BloC pada Flutter dapat dilihat pada Gambar 4.14.

```
11  class KehilanganBloc extends Bloc<KehilanganEvent, KehilanganState> {
12    KehilanganDomain kehilanganDomain;
13
14    KehilanganBloc({@required this.kehilanganDomain})
15        : assert(kehilanganDomain != null),
16          super(null);
17
18    KehilanganState get initialState => KehilanganFetchLoading();
19
20    @override
21    Stream<KehilanganState> mapEventToState(KehilanganEvent event) async* {
22      if (event is KehilanganFetching) {
23        yield KehilanganFetchLoading();
24        try {
25          List<Kehilangan> kehilangan = await kehilanganDomain.getKehilangan();
26          yield KehilanganFetchSuccess(kehilangan: kehilangan);
27        } catch (e) {
28          yield KehilanganFetchError(error: e.toString());
29        }
30      }
31    }
32  }
```

Gambar 4.10 BloC Kehilangan

```
4   class KehilanganDomain {
5     final KehilanganRepository kehilanganRepository;
6
7     KehilanganDomain(this.kehilanganRepository);
8
9     Future<List<Kehilangan>> getKehilangan() {
10      return kehilanganRepository.getKehilangan();
11    }
12  }
```

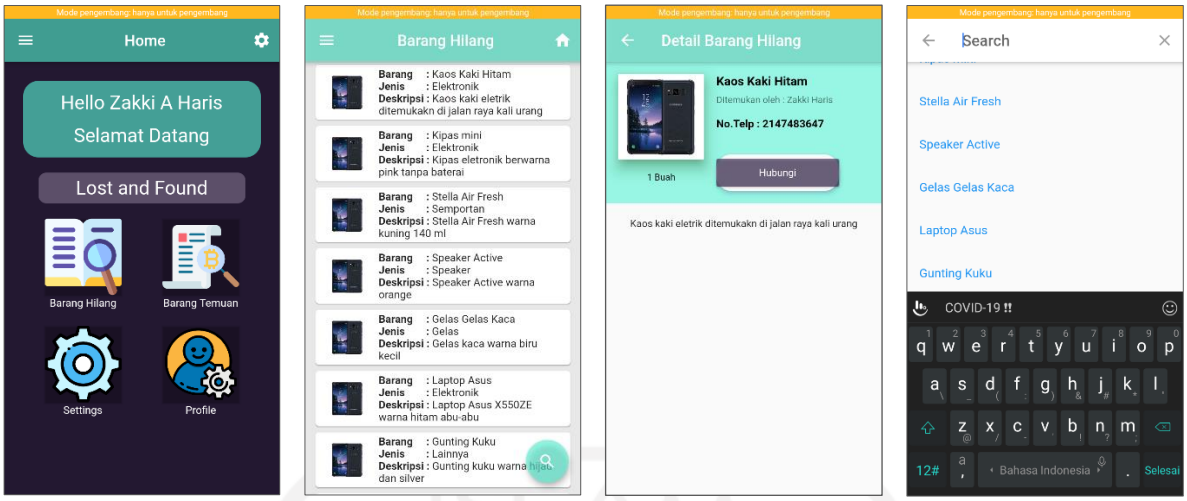
Gambar 4.11 Domain BloC

```dart
class KehilanganRepository {
  final baseUrl = "http://192.168.1.3/api";

  Future<List<Kehilangan>> getKehilangan() async {
    final response = await http.get(
      "$baseUrl/kehilangan",
      headers: {
        "HttpHeaders.contentTypeHeader": "application/json",
        "X-API-KEY": "pass"
      },
    );

    if (response.statusCode == 200) {
      var data = json.decode(response.body);
      return List<Kehilangan>.from(
          data.map((item) => Kehilangan.fromJson(item))); // List.from
    } else {
      throw Exception('Not connected');
    }
  }
}
```

Gambar 4.12 Repository (Request & Response)

```dart
abstract class KehilanganState extends Equatable {
  const KehilanganState();
}

class KehilanganFetchLoading extends KehilanganState {
  @override
  List<Object> get props => [];
}

class KehilanganFetchSuccess extends KehilanganState {
  final List<Kehilangan> kehilangan;

  const KehilanganFetchSuccess({this.kehilangan});

  @override
  List<Object> get props => [kehilangan];
}

class KehilanganFetchError extends KehilanganState {
  final String error;

  const KehilanganFetchError({this.error});

  @override
  List<Object> get props => [error];

  @override
  String toString() => '{error: $error}';
}
```

Gambar 4.13 State BloC



Home Daftar Barang Detail Barang Cari Barang

Gambar 4.14 Hasil Aplikasi Flutter

4.2 Hasil Perbandingan

4.2.1 Customizability

Tabel 4.1 merupakan hasil dari analisis tingkat kompetensi kustomisasi yang dilakukan pada Flutter dan React Native.

Tabel 4.1 Tingkat Kompetensi Kustomisasi pada *Framework*.

Prespektif	**Tingkat Kompetensi**				
	Entry	**Aware**	**Capable**	**Mature**	**World Class**
Components				● ●	
File Structure			● ●		
Setup and Configuration		●	●		

● React Native

● Flutter

A. Components

React Native dan Flutter memiliki memiliki banyak *components* yang mudah dibuat, diatur, dan dipelihara. Dalam React Native, *Components* disebut dengan React Native Components sedangkan dalam Flutter disebut Flutter Widget, dan seterusnya widget akan disebut dengan nama *component*. *Components* dapat dilakukan kustomisasi sesuai dengan pengembang inginkan untuk menghindari penggunaan *components* yang berlebihan atau

berulangan seperti *components Header* pada Gambar 4.15. Dalam gambar tersebut dapat dilihat bahwa terdapat components yang hampir sama. Pengulangan *comoponent* yang sama akan menambahkan baris kode serta menyulitkan pengembang dalam melakukan pengembangan ataupun perbaikan. Untuk mengurangi pengulangan dengan menerapkan kustomisasi *components* itu sendiri.

```
<View style={styles.header}>
    <TouchableOpacity
    onPress={()  => navigation.navigate('Home')}>
    <View style={{ paddingLeft: 15 }}>
        <Image
            style={{ width: 24, height: 24 }}
            source={images.bars} />
    </View>
    </TouchableOpacity>
    <View>
        <Text style={styles.textCenter}>Barang Hilang</Text>
    </View>
    <TouchableOpacity
    onPress={()  => navigation.navigate()}>
        <View style={{ paddingRight: 15 }}>
            <Image
                style={{ width: 24, height: 24 }}
                source={images.cog} />
        </View>
    </TouchableOpacity>
</View>
```

Gambar 4.15 Contoh pengulangan *components* pada *Header*

- **React Native Components**

Dilihat dari contoh pada Gambar 4.15 dapat dilakukan kutomisasi dengan membuat components *header* menggunakan *library* react-native-elements dan membagi menjadi beberapa komponen, sehingga menjadi seperti pada Gambar 4.16. Setelah dibagi sesuai component yang diinginkan, kemudian membuat kustom *components* untuk mereferensikan *component* agar dapat dimanfaatkan dalam layar yang berbeda dengan menambahkan *component* yang memungkinkan adanya perubahan *state* seperti contoh pada Gambar 4.17. Dari gambar components tersebut dapat digunakan dimanapun dengan merubah *text* dan *routes pages.* Kode lengkap dapat ditemukan pada lampiran A.

```
<View>
    <Header
        leftComponent={
            <TouchableOpacity
            onPress={this.props.leftOnPress}>
                <View style={{ paddingLeft: 15 }}>
                    <Image
                        style={{ width: 22, height: 22 }}
                        source={images.bars} />
                </View>
            </TouchableOpacity>
        }
        centerComponent={
            <View>
                <Text style={styles.textCenter}>{this.props.centerText}</Text>
            </View>
        }
        rightComponent={
            <TouchableOpacity
            onPress={this.props.rightOnPress}>
                <View style={{ paddingRight: 15 }}>
                    <Image
                        style={{ width: 24, height: 24 }}
                        source={images.cog} />
                </View>
            </TouchableOpacity>
        }
    />
</View>
```

Gambar 4.16 Membagi menjadi beberapa *component*

```
<HeaderComponents
    leftOnPress={() => navigation.navigate('Home')}
    centerText='Barang Hilang'
    rightOnPress={() => navigation.navigate('Setting') }
/>
```

Gambar 4.17 Kustom *Components* React Native

- **Flutter Widget**

Pada pembahsana sebelumnya kustomisasi React Native dilakukan dengan beberapa *components* sekaligus. Dalam implementasi Flutter akan dilakukan pada sebuah *components*, yaitu *component text* seperti terlihat pada Gambar 4.18. Kustomisasi *component text* pada gambar tersebut memiliki berbagai *properties* yang diinginkan seperti *color, size, bold, padding*. Untuk penggunaan *component text* dengan mengirimkan properties yang diingikan, jika properties tidak dikirimkan maka components akan memiliki properties yang sudah ditentukan sebelumnya pada saat kustomisasi components. Cara memanggil kustomisasi components dapat dilihat pada Gambar 4.19.

```
text(String data,
        {Color color = Colors.black,
        num size = 14,
        EdgeInsetsGeometry padding = EdgeInsets.zero,
        bool isBold = false}) =>
    Padding(
      padding: padding,
      child: Text(
        data,
        style: TextStyle(
            color: color,
            fontSize: size.toDouble(),
            fontWeight: isBold ? FontWeight.bold : FontWeight.normal),
      ), // Text
    ); // Padding
```

Gambar 4.18 Kustom *components* Flutter

```
text(kehilangan.namaBarang,
    size: 16, isBold: true, padding: EdgeInsets.only(top: 16.0)),
text(
  'Ditemukan oleh : ' + kehilangan.nama,
  color: Colors.black54,
  size: 12,
  padding: EdgeInsets.only(top: 8.0, bottom: 16.0),
),
```

Gambar 4.19 Penggunaan kustomisasi *components*

- **Kesimpulan Components**

Dari implementasi yang berbeda pada kustomisasi *components* dapat disimpulkan bahwa tingkat kustomisasi *components* pada kedua *framework* berada pada tingkat kompetensi yang sama yaitu “Mature” dengan tingkat variansi tinggi, karena memiliki perubahan dasar yang dapat diprogram sesuai dengan kostumisasi pengguna. Walaupun dalam melakukan implementasi berbeda, setiap framework dapat melakukan implementasi sama dengan yang lainnya lakukan.

B. File Structure

Mengatur kode program dalam banyak *file* agar terstruktur sangatlah rumit. Bagus tidaknya struktur sangat berpengaruh terhadap jalannya pengembangan aplikasi terutama ketika pengembangan dilakukan dengan adanya tim maupun ketika penambahan dan perbaikan fitur pada aplikasi. Banyak kode baru yang masuk sehingga memungkinkan adanya duplikasi kode program dengan mengulangi pembuatan *components* dan logika bisnis. Kemungkinan terburuknya sulit ditemukanya duplikasi karena struktur folder tidak ramah dengan adanya tim.

Tidak ada solusi yang cocok untuk menyusun structur *file* pada aplikasi. Sebagian besar aplikasi React Native maupun Flutter memiliki direktori yang menjelaskan *components, screen, state management,* dan *utilities.* Ada beberapa rekomendasi untuk membuat struktur *file* yaitu dengan mengelompokkan *file* bedasarkan tipe seperti terlihat pada Gambar 4.20 pada bagian direktori redux dan berdasarkan fitur seperti terlihat pada Gambar 4.21 pada bagian direktori bloc.

Gambar 4.20 React Native Struktur File

Gambar 4.21 Flutter Struktur File

- **Kesimpulan File Structure**

Kebebasan dalam pembutan struktur direktori sangatalah bagus bagi pengembang. Tetapi biarkan aplikasi yang memilih strukturnya, karena setiap aplikasi memiliki kompleksitas yang berbeda-beda. Ada aplikasi yang membutuhkan banyak *components* sederhana yang bersatu pada sebuah file dan ada juga aplikasi yang membutuhkan ratusan *components* dalam direktori dengan sedikit variansi.

Maka dari itu dapat simpulkan bahwa tingkat kustomisasi pada direktori kedua framework berada pada tingkat kompetensi "Capable" dengan tingkat variansi sedang, karena memiliki perubahan standar yang ditentukan oleh pengguna tanpa bisa diprogram.

C. Setup and Configuration

Mebuat projek baru dengan konfigurasi Flutter lebih mudah dibandingakn dengan React Native. Flutter memiliki konfigurasi yang sederhana dan mudah untuk dipahami, dalam konfigurasi Flutter memiliki fitur yang bernama *flutter doctor* yang digunakan untuk memeriksa alat yang perlu diperbaiki dan dikonfigursi.

React Native lebih sulit dibandingkan dengan Flutter dalam melaukan konfigurasi karena memiliki berbagai macam konfigurasi. React Native menawarkan dua cara untuk melakukan *setup* awal yaitu dengan menggunaka React Native CLI (*command line interface*) dan Expo CLI. Hal tersebut memungkinkan pengguna dapat memilih setup dan konfigurasi yang dinginkan.

- **Kesimpulan Setup and Configuration**

Kebebasan dalam melakukan konfigurasi pada *framework* sangatlah menguntungkan bagi pengguna karena banyak varian konfigurasi yang dapat diatur di dalamnya, disamping itu selain keuntungkan dalam melakukan konfigurasi juga memiliki kelemahan karena banyak yang harus dikonfigurasi.

Secara tingkat kostumasi React Native lebih unggul dibandingkan dengan Flutter karena memiliki berbagai pilihan *setup* dan *configuration* sehiggan dapat dikatakan React Native berada pada tingkat kompetensi "Capable" dengan tingkat variansi sedang karena memiliki perubahan relative standar yang ditentukan oleh pengguna yang terletak pada pemilihan *setup* awal dan konfigurasi didalamnya.

Flutter berada pada tingkat kompetensi "Aware" dengan tingkat variansi rendah karena perubahan relative standar yang sudah ditentukan sebelumnya. Perlu diingat bahwa

customizability adalah kemudahan dalam melakukan perubahan perangkat lunak. Secara tingkat kompetensi React Native memang lebih unggul dibanding dengan Flutter, tetapi dalam hal kemudahan Flutter lebih unggul dibandingkan dengan React Native karena tingkat kompetensi rendah sehingga tidak banyak setup dan konfigurasi yang dilakukan.

4.2.2 Modifiability

A. Mengukur Jumlah Fungsi

Pengukuran dilakukan dengan mencari titik-titik fungsi bedasarkan pada masing-masing pola desain dan faktor fungsional. Misalkan kode program *state* BloC yang terlihat pada Gambar 4.13. Dalam kode program tersebut terdapat 4 fungsi *state* yang digunakan untuk menangkap data dari BloC dan meneruskan ke *User Interface* seperti dijelaskan pada Gambar 4.8. Maka *state* pada BloC yang berjumlah 4 tersebut dikategorikan dalam faktor fungsional "*External inquires*" karena menangkap akses data. Hal tersebut dilakukan pada beberapa bagian pada pola desain BloC dan Redux sehingga menghasilkan beberapa fungsi seperti terlihat pada Tabel 4.2.

Tabel 4.2 Fungsi dalam Flutter BloC dan React Native Redux

Faktor Fungsional	**Fungsi Flutter BloC**	**Fungsi React Native Redux**
Internal logic file	1 (Request repository)	1 (Request repository)
Internal interface file	2 (BloC in-out)	2 (Store in-out)
External input	2 (Event)	1 (Event)
External output	1 (Response repository)	1 (Response repository)
External inquires	3 (State) 3 (UI) 1 (Domain)	4 (Action) 2 (Midleware) 3 (Reducer) 2 (UI)
Jumlah Titik	**13**	**16**

Pada Tabel 4.2 merupakan hasil dari pemilihan fungsi yang berada pada Redux dan BloC berdasarkan kode program dalam *state management* pada Gambar 4.2 dan Gambar 4.8. Setiap fungsi yang berada pada Redux dan BloC memiliki faktor fungsional masing-masing, seperti:

1. *Internal logic file* Redux dan Flutter BloC terdapat pada bagian *components input-output Store* dan dan *input-output* BloC seperti pada Gambar 4.5 untuk Redux dan Gambar 4.10 untuk Flutter karena *components Store* dan BloC bertugas untuk

menyimpan atau mengelola data yang akan digunakan dalam suatu *components*, data berasal dari *event* dan *repository*.

2. *Internal interface file* Redux dan Flutter BloC terdapat pada bagian *components request repository* yang berjumlah satu fungsi seperti pada Gambar 4.4 untuk Redux dan Gambar 4.12 untuk Flutter BloC, *request repository* dikategorikan dalam *internal interface file* karena berisikan data dari luar yang diterima dari lingkuang operasional seperti API.
3. *External input* terdapat pada bagian *components event* yang berjumlah dua fungsi untuk Flutter BloC pada Gambar 4.9 dan satu fungsi untuk Redux pada Gambar 4.2. *Event* dikategorikan *external input* karena merupakan input yang menyimpan atau menerima data dari luar, misalkan tindakan yang dilakukan pengguna terhadap aplikasi berupa data *event* seperti menekan tombol.
4. *External output* terdapat pada bagian *components response repository* yang berjumlah masing-masing satu fungsi seperti pada Gambar 4.4 untuk Redux dan Gambar 4.12 untuk Flutter BloC. *External output* berkebalikan dengan *internal interface file* yaitu mengemebalikan data ke lingkuangan operasional (API).
5. *External inquires* terdapat pada bagian *components state, domain,* dan *user interface* pada Flutter sedangkan Redux terdapat pada *action middleware reducer* dan *user interface.* Dari beberapa *components* tersebut dikategorikan *external inquires* karena bertugas menangkap akses data. Misalkan pada Gambar 4.8, *state* pada Flutter menangkap data yang berasal dari BloC kemudian diteruskan kebagian *user interface.* Fungsi *state* yang dikirimkan berjumlah tiga yang berasal dari kode program pada Gambar 4.13. Gambar tersebut terdapat tiga fungsi *state* yaitu “KehilanganFetchLoading”, “KehilanganFetchSuccess”, dan “KehilanganFetchError”.

Setelah mengetahui titik-titik fungsi beserta faktor fungsional yang berada dalam BloC yang berjumlah 13 dan Redux berjumlah 16 kemudian menentukan bobot faktor fungsional dengan menghitung jumlah titik fungsi. Bobot fungsional dikatakan rendah jika jumlah titik fungsi tidak melebihi 50 atribut atau titik. Sehingga dapat di simpulkan bahwa bobot faktor fungsional berada pada tingkat rendah seperti terlihat pada Tabel 4.3.

Tabel 4.3 Bobot Faktor Fungsional Tingkat Rendah

Faktor Fungsional	**Tingkat Rendah**
Internal logic file	7
Internal interface file	5
External input	3
External output	4
External inquires	3

Dengan ditentukannya bobot faktor fungsional yang memiliki tingkat rendah, maka dari itu dapat dihitung jumlah total fungsi dengan mengalikan bobot faktor fungsional dengan beberapa titik fungsi sehingga hasilnya seperti terlihat pada Tabel 4.4 untuk React Native Redux dan Tabel 4.5 untuk Flutter BloC.

Tabel 4.4 Jumlah Fungsi React Native Redux

Faktor Fungsional	**Bobot**	**Fungsi React Native Redux**	**Hasil**
Internal logic file	7	2 (Store in-out)	7 x 2= 14
Internal interface file	5	1 (Request repository)	5 x 1 = 5
External input	3	1 (Event)	3 x 1 = 3
External output	4	1 (Response repository)	4 x 1 = 4
External inquires	3	4 (Action) 2 (Midleware) 3 (Reducer) 2 (UI)	3 x 11 = 33
TOTAL			**59**

Tabel 4.5 Jumlah Fungsi Flutter BloC

Faktor Fungsional	Bobot	Fungsi Flutter BloC	Hasil
Internal logic file	7	2 (BloC in-out)	7 x 2 = 14
Internal interface file	5	1 (Request repository)	5 x 1 = 5
External input	3	2 (Event)	3 x 2 = 6
External output	4	1 (response repository)	4 x1 = 4
External inquires	3	3 (State) 3 (UI) 1 (Domain) 1(Model)	3 x 8 = 24
TOTAL			**53**

Dari hasil kedua penghitungan tersebut dapat disimpulkan bahwa Flutter BloC lebih unggul dibandingkan dengan React Native Redux karena Flutter BloC memiliki lebih sedikit total fungsi. Semakin banyak fungsi maka semakin komplek pula untuk dimodifikasi dan begitupun sebaliknya.

B. Mengukur Cohesion dan Coupling

Pengukuran cohesion dan coupling dilakukan dengan menentukan komponen dan *task* (tugas) pada Redux serta BloC. Ada beberapa komponen dan *task* yang ada pada pola desain yang digunakan seperti terlihat pada Tabel 4.6. Komponen dan *task* pada gambar tersebut didapat dari menganalisis pola desain pada Gambar 4.2 dan Gambar 4.8.

Tabel 4.6 Komponen dan *Task* (tugas)

Komponen React Native Redux	Komponen Flutter BloC
Action = 4 Task Reducer = 3 Task Store = 1 Task UI = 2 Task	Event = 1 Task BloC = 3 Task State = 3 Task UI = 2 Task

Pada Tabel 4.6 merupakan hasil dari pemilihan komponen yang berada pada Redux dan BloC dalam *state management* pada Gambar 4.2 dan Gambar 4.8. Berdasarkan pada Gambar 4.2 React Native Redux memiliki 4 komponen utama yaitu *Action, Reducer, Store,* dan *User Interface*.

a. *Action*, komponen *action* memiliki 4 *data-acces task* seperti pada Gambar 4.3 yaitu "fetchKehilangan", "fetchSources", "fetchKehilanganSuccess", dan "fetchKehilanganError".
b. *Reducer*, komponen *reducer* memiliki 3 *data-acces task* seperti pada Gambar 4.6 yaitu bertugas untuk mengembalikan kondisi sumber data, kondisi sukses dengan data, dan kondisi eror.
c. *Store*, komponen *store* memiliki 1 *data-acces task* seperti pada Gambar 4.5 yang bertugas untuk menghubungkan antara *action* dengan *reducer*.
d. *User Interface*, komponen *user Interface* memiliki 2 *human task* yang bertugas untuk mengirim dan menangkap tanggapan kepada pengguna.

Sedangakan berdasarkan pada Gambar 4.8, Flutter BloC memiliki 4 komponen utama yaitu *Event, BloC, State,* dan *User Interface*.

a. *Event*, komponen *event* memiliki 1 *data-acces task* seperti pada Gambar 4.9 yaitu "KehilanganEvent" yang bertugas untuk mengirimkan *action* berupa objek menuju BloC.
b. *BloC*, komponen *bloC* memiliki 3 *data-acces task* seperti pada Gambar 4.10 yaitu bertugas untuk mengembalikan kondisi sumber data, kondisi sukses dengan data, dan kondisi eror.
c. *State*, komponen *state* memiliki 3 *data-acces task* seperti pada Gambar 4.13 yaitu "KehilanganFetchLoading", "KehilanganFetchSuccess", dan "KehilanganFetchError".
d. *User Interface*, komponen *user Interface* memiliki 2 *human task* yang bertugas untuk mengirim dan menangkap tanggapan kepada pengguna.

- **Cohesion**

Sebelum melakkan penghitunga perlu menentukan tingkat *cohesion* pada pola desain untuk mengetahui tingkat modifikasi. Tingkat *cohesion* ditentukan mengunakan Tabel 4.6 sehingga dapat disimpulkan bahwa kategori pada pola desain adalah *procedural* dengan tingkat *cohesion* menengah karena pola desain yang digunakan memiliki tugas dalam sebuah komponen dihubungkan dengan konektor control. Setelah mengetahui tingkat cohesion kemudian menghitung komponen dan *task* berdasaran persaman (3.1) sehingga menghasilkan hitunga seperti pada Tabel 4.7.

Tabel 4.7 Menghitung *Cohesion*

Menghitung *Cohesion* React Native Redux	**Menghitung *Cohesion* Flutter BloC**
$S_{Cohesion} = \frac{\sum_{i=1}^{m} A_i}{m}, A_i = \frac{\mu_i}{n_i^2}$	$S_{Cohesion} = \frac{\sum_{i=1}^{m} A_i}{m}, A_i = \frac{\mu_i}{n_i^2}$
$A_{Action} = \frac{4}{4^2} = \frac{1}{4}$ $A_{Reducer} = \frac{3}{3^2} = \frac{1}{3}$ $A_{Store} = \frac{1}{1^2} = \frac{1}{1}$ $A_{UI} = \frac{2}{2^2} = \frac{1}{2}$	$A_{Event} = \frac{1}{1^2} = \frac{1}{1}$ $A_{State} = \frac{3}{3^2} = \frac{1}{3}$ $A_{BloC} = \frac{3}{3^2} = \frac{1}{3}$ $A_{UI} = \frac{2}{2^2} = \frac{1}{2}$
$S_{Cohesion} = \frac{\frac{1}{4} + \frac{1}{3} + \frac{1}{1} + \frac{1}{2}}{m}$	$S_{Cohesion} = \frac{\frac{1}{1} + \frac{1}{3} + \frac{1}{3} + \frac{1}{2}}{m}$
$S_{Cohesion} = \frac{\frac{3+4+12+6}{12}}{4} = \frac{\frac{25}{12}}{4} = \frac{2{,}08}{4} = \mathbf{0,52}$	$S_{Cohesion} = \frac{\frac{6+2+2+3}{6}}{4} = \frac{\frac{13}{6}}{4} = \frac{2{,}16}{4} = \mathbf{0,54}$

- **Coupling**

Berdasarkan tingkat coupling pada Tabel 3.7, pola desain dikategorikan rendah dan berada pada tingkat *coupling* “data” karena memiliki data primitif yang diteruskan diantara komponen berbentuk *array* (himpunan) atau objek. Berbedan dengan cara menghitung *cohesion*, coupling membutuhkan skenario aplikasi untuk mengetahui alur antar komponen. Skenario yang digunakan adalah menampilkan daftar barang hilang.

Skenario untuk menampilkan daftar barang hilang:

1. Melakukan klik ikon barang hilang pada *user interface*
2. Mengirim *event/action* untuk meminta data barang hilang ke *store*/*bloc*
3. Menampilkan *loading state*
4. *Store*/*bloc* akan meminta data ke repository API
5. Setelah melakukan permintaan akan dikirimkan ke *user interface* melalui *reducer/state*.
6. *Reducer/state* akan mengrimkan pesan sukses ke *user interface* jika medapatkan data, dan pesan kesalahan ketika tidak mendapatkan data.
7. Menutup *loading state* dan menampilkan data kedalam *user interface*

Dari skenario tersebut dapat diketahui mana saja components yang berhubungan antara satu dengan yang lain secara urut sehingga dapat digunaakn untuk menghitung *coupling* atau ketergantungan antar komponen. Penghitungan dilakukan dengan persamaan (3.2) sehingga menghasilkan hitungan seperti pada Tabel 4.8.

Tabel 4.8 Menghitung *Coupling*

Menghitung *Coupling* React Native Redux	**Menghitung *Coupling* Flutter BloC**
$S_{Coupling} = \frac{\sum_{i,j=1}^{m} E_{i,j}}{\frac{m(m-1)}{2}}, E_{i,j} = \begin{cases} 0 & i = j \\ \varepsilon_{i,j} & i \neq j \\ 2n_i n_j \end{cases}$ $E_{i,j} = \frac{\varepsilon_{i,j}}{2n_i n_j}$	
Coupling (Data) $E_{UI,Action} = \frac{6}{2\ x\ 2\ x\ 4} = \frac{6}{16} = 0{,}375$ $E_{Action,Store} = \frac{5}{2\ x\ 4\ x\ 1} = \frac{5}{8} = 0{,}625$ $E_{Store,Reducer} = \frac{4}{2\ x\ 1\ x\ 3} = \frac{4}{6} = 0{,}666$ $E_{Reducer,UI} = \frac{5}{2\ x\ 3x\ 2} = \frac{5}{12} = 0{,}416$ $S_{Coupling} = \frac{0,375 + 0,625 + 0{,}666 + 0{,}416}{\frac{4(4-1)}{2}}$ $S_{Coupling} = \frac{2{,}082}{\frac{12}{2}} = \frac{0{,}1735}{2} = \mathbf{0,0867}$	***Coupling (Data)*** $E_{UI,Event} = \frac{3}{2\ x\ 2\ x\ 1} = \frac{3}{4} = 0{,}75$ $E_{Event,BloC} = \frac{4}{2\ x\ 1\ x\ 3} = \frac{4}{6} = 0{,}666$ $E_{BloC,State} = \frac{6}{2\ x\ 3\ x\ 3} = \frac{6}{18} = 0{,}333$ $E_{State,UI} = \frac{5}{2\ x\ 3\ x\ 2} = \frac{5}{12} = 0{,}416$ $S_{Coupling} = \frac{0,75 + 0{,}666 + 0{,}333 + 0,416}{\frac{4(4-1)}{2}}$ $S_{Coupling} = \frac{2{,}165}{\frac{12}{2}} = \frac{0{,}1804}{2} = \mathbf{0,0902}$

C. Kesimpulan Modifiability

Dari perhitungan *modifiability* React Native Redux dan Flutter BloC menghasilkan pengukuran seperti pada Tabel 4.9. Dari pengukuran jumlah fungsi, Flutter BloC lebih unggul karena titik fungsi yang berada dalam BloC tidak begitu banyak dibandingkan dengan Redux, semakin banyak fungsi yang berada dalam pola desain maka akan semakin sulit untuk melakukan modifikasi. Aliran pada Redux lebih kompleks dibandingkan dengan BloC Karena lebih banyak fungsi yang digunakan didalamnya.

Dalam pengukuran *Cohesion*, BloC lebih unggul sedangkan dari pengukuran *Coupling* Redux lebih unggul. Dari segi komponen keduanya memiliki jumlah yang sama yaitu sebanyak 4 komponen, sedangkan dari segi *task* (tugas) BloC lebih sedikit dibanding dengan Redux. Semakin tinggi *cohesion* maka semakin bagus pola desain, sedangkan sebaliknya semakin rendah *coupling* maka semakin bagus pola desain yang digunakan.

Tabel 4.9 Hasil Pengukuran Jumlah Fungsi, *Cohesion (Prosedural)* dan *Couplig (Data)*

Framewrok (State Management)	**Jumlah Fungsi**	**Cohesion (Procedural)**	**Coupling (Data)**
React Native Redux	57	0,52	**0,0867**
Flutter BloC	**51**	**0, 54**	0,0902

Pengukuran *modifiability* berdasarkan jumlah fungsi, *cohesion*, dan *coupling* masih kurang. Jumlah baris kode program juga mempengaruhi dalam *modifiability* karena setiap fungsi memiliki banyak baris kode yang berbeda-beda.Yang perlu diperhatikan dalam pengukuran jumlah baris kode program adalah cara mengimplementasikan kode program tersebut dengan menuliskan kode program sesuai dengan dokumentasi yang sudah disediakan pada masing-masing pola desain.

4.2.3 Testabilty

Bagian ini berisi dari hasil implementasi yang besisi daftar kode program dan langkah-langkah dalam pengujian *unit testing* dan *integration testing*.

A. Unit Testing

Dalam melakukan *unit testing* perlu diketahui bahwa terdapat beberapa penerapan pengujian seperti menggunakan *scaffolding* dalam bentu *stub* dan tanpa *scaffolding*. Pada kasus ini akan dilakukan dengan menggunakan *scaffolding* dalam bentuk *stub* yang sudah dijelaskan pada pembahasan sebelumnya bagian unit testing.

Pengujian dilakukan pada bagian pemanggilan API yang ada pada repository, ada beberapa pengujiannya misalnya seperti pemanggilan data ketika sukses dan tidak terhubung dengan server (*bad request*).

- **Unit Test Peamanggilan API Asynchronous Sukses**

Pengujian unit pemanggilan API pada Flutter dibantu dengan *library* mockito, implementasi kode program dalam melakukan unit testing dapat dilihat pada Gambar 4.22. Dalam gambar pada baris 29 sampai 51 terdapat proses *mocking* (*stubbing*) atau pembuatan objek serupa tanpa berhubungan langsung dengan objek aslinya. Objek yang dibuat berupa *array* seperti pada baris 32 sampai 49 akan dibandingkan dengan objek yang berada pada projek aslinya dengan memanggil fungsi pemanggilan API seperti pada baris 52 dan 53. Pada baris 53 berharap hasilnya seperti *mocking* atau objek yang sudah dibuat.

```
28      test('returns result call api', () async {
29        final response = MockResponse();
30        when(response.statusCode).thenReturn(200);
31        when(response.body).thenReturn('''
32          [
33            {
34              "id": "133",
35              "id_user": "59",
36              "nama": "Zakki Haris",
37              "peran": "Mahasiswa",
38              "nama_barang": "Kaos Kaki Hitam ",
39              "jenis_barang": "Elektronik",
40              "gambar": "gambar_kaos.png",
41              "detail_informasi": "Kaos kaki eletrik ditemukakn di jalan raya kali urang",
42              "tempat_kehilangan": "Kaliurang KM 14",
43              "waktu_kehilangan": "2020-01-16",
44              "nomor_hp": "2147483647",
45              "id_line": "jack",
46              "status": "0",
47              "waktusekarang": "2020-12-13 09:35:56"
48            }
49          ]
50        ''');
51        when(httpClient.get(any)).thenAnswer((_) async => response);
52        final actual = await kehilanganRepository.getKehilangan();
53        expect(actual, isA<List<Kehilangan>>());
54      });
```

Gambar 4.22 Unit testing pemanggilan API dengan *stub* pada Flutter

Implementasi *unit testing* pada React Native dilakukan dengan *mockito* yang sudah termasuk dengan library *jest.* Pada Gambar 4.23 menjelaskan implementasi React Native dalam melakukan *unit testing* pada pemanggilan API. Proses *mocking* dilakukan dengan fungsi tiruan *jest* seperti pada baris 24 dan 25, yang dapat melakukan tanggapan dan masukan yang diberikan pada kode baris 31, 32, dan 33. Pengujian dilakukan dengan memanggil fungsi API kehilangan seperti pada baris 27 dan mengembalikan dalam 3 bentuk harapan. Pada baris 31 berharap mendapatkan tanggapan dalam pemanggilan, baris 32 berharap tidak ada kesalahan dalam

pemanggilan, baris 33 berharap data yang dipanggil sama dengan apa yang diharapkan seperti baris 34 sampai 49.

```
22  test('returns result call api', () => {
23    fetch.mockResponseOnce(JSON.stringify([{ id: 113 }]));
24    const onResponse = jest.fn();
25    const onError = jest.fn();
26
27    return callApiKehilangan(url + '?X-API-KEY=pass')
28      .then(onResponse)
29      .catch(onError)
30      .finally(() => {
31        expect(onResponse).toHaveBeenCalled();
32        expect(onError).not.toHaveBeenCalled();
33        expect(onResponse.mock.calls[0][0][0]).toEqual(
34          {
35            "id": "133",
36            "id_user": "59",
37            "nama": "Zakki Haris",
38            "peran": "Mahasiswa",
39            "nama_barang": "Kaos Kaki Hitam ",
40            "jenis_barang": "Elektronik",
41            "gambar": "gambar_kaos.png",
42            "detail_informasi": "Kaos kaki eletrik ditemukakn di jalan raya kali urang",
43            "tempat_kehilangan": "Kaliurang KM 14",
44            "waktu_kehilangan": "2020-01-16",
45            "nomor_hp": "2147483647",
46            "id_line": "jack",
47            "status": "0",
48            "waktusekarang": "2020-12-13 09:35:56"
49          }
50        );
51      });
52  });
```

Gambar 4.23 Unit testing pemanggilan API dengan *stub* pada React Native

- **Unit Test *Bad Request***

Pengujian unit dalam melakukan penanganan ketika mendapatkan *bad request* (400) dalam Flutter dapat dilihat pada Gambar 4.24. Flutter menggunakan *library Mockito* untuk membuat objek yang serupa seperti pada baris 72. *Mockito* akan membuat beberapa kondisi objek dimana ketika saat menerima kode status 400 seperti pada baris 73 dan ketika *http client* menerima tanggapan seperti pada baris 74. Baris kode 75 sampai 77 memanggil fungsi kegagalan dalam *request* pada projek. Kemudian dibandingan antara objek yang diterima dengan objek yang dibuat dengan *mockito*. Jika sama maka test yang dilakukan berhasil.

```
71        test('throws an error if bad request', () async {
72          final response = MockResponse();
73          when(response.statusCode).thenReturn(400);
74          when(httpClient.get(any)).thenAnswer((_) async => response);
75          expect(
76            () async => await kehilanganRepository.getKehilangan(),
77            throwsA(isA<KehilanganRequestFailure>()),
78          );
79        });
80      });
81    });
```

Gambar 4.24 Unti testing *bad request* dengan *stub* pada Flutter

Implementasi pada React Native dilakukan dengan mengunakan *library mockito* drprti pada Gambar 4.25 dengan membuat *stub* seperti pada baris 68. Kemudian API yang memiliki *bad reques*t akan dipanggil dan dikembalikan dengan dua kondisi. Kondisi pertama API sudah dipanggila dan mengembalikan objek seperti pada baris 73. Kondisi yang kedua, objek yang diterima sama dengan objek yang dibuat seperti pada baris 74, jika kondisi keduanya terpenuhi maka *testing* berhasil.

```
67  test('throws an error if bad request', () => {
68    fetch.mockResponseOnce(JSON.stringify([{}]));
69    const onResponse = jest.fn();
70    return callApiKehilangan(url + '?X-API-KEY=pas')
71      .then(onResponse)
72      .finally(() => {
73        expect(onResponse).toHaveBeenCalled();
74        expect(onResponse.mock.calls[0]).toEqual([{"data": "Bad request from server"}])
75      });
76  });
```

Gambar 4.25 Unit testing *bad request* dengan *stub* pada React Native

B. Integration Testing

Integration testing merupakan kelanjutan dari *unit testing*. Pengujianya dapat dilakukan dengan berbagai cara, tetapi pengujian sederhana dilakukan dalam pengujian dengan melakukan pengecekan *widget* pada aplikasi yang kita buat agar sesuai dengan apa yang diharapkan. Seperti pengujian yang dilakukan pada React Native yang terlihat pada Gambar 4.26. Pada gambar tersebut menguji *widget header* dan *widget* yang berisikan data sebanyak 50 yang berasal dari API. Pengujian dilakukan dengan melakukan render pada fungsi kehilangan dan mencari berdasarkan teks yang berada dalam halaman kehilangan seperti pada baris 34, 35 dan 36. Kemudain dibandingkan agar sesuai harapan dengan *header widget* memiliki teks berupa “Barang Hilang” dan *item* sebanyak 50 data.

```
27      test('Test components header dan 50 item data', async () => {
28          const component = (
29              <NavigationContainer>
30                  <Kehilangan />
31              </NavigationContainer>
32          );
33
34          const { findByText, findAllByText} = render(component);
35          const header = await findByText('Barang Hilang');
36          const items = await findAllByText(50);
37
38          expect(header).toBeTruthy();
39          expect(items.length).toBe(50);
40      });
```

Gambar 4.26 *Integration testing widget* pada React Native

Implementasi pengujian pada Flutter serupa dengan yang dilakukan oleh React Native yaitu dengan melakukan pengecekan pada widget seperti terlihat pada Gambar 4.27. Pada Gambar tersebut pengecekan dilakukan dari halaman “HomePage” dengan melakukan pencarian berdasarkan *key* yang sudah dibuat pada halaman yang diuji seperti pada baris ke 15 dan 17. Dari *key* tersebut akan digunakan untuk berpindah halaman dari “HomePage” menuju halaman “Barang Hilang” seperti pada baris ke 19. Setelah sampai ke halaman “Barang Hilang” dilakukan pengecekan *widget Appbar* dan *text* yang dimana *text* tersebut harus bersisikan *text* “Barang Hilang” seperti pada baris 22 sampai 26.

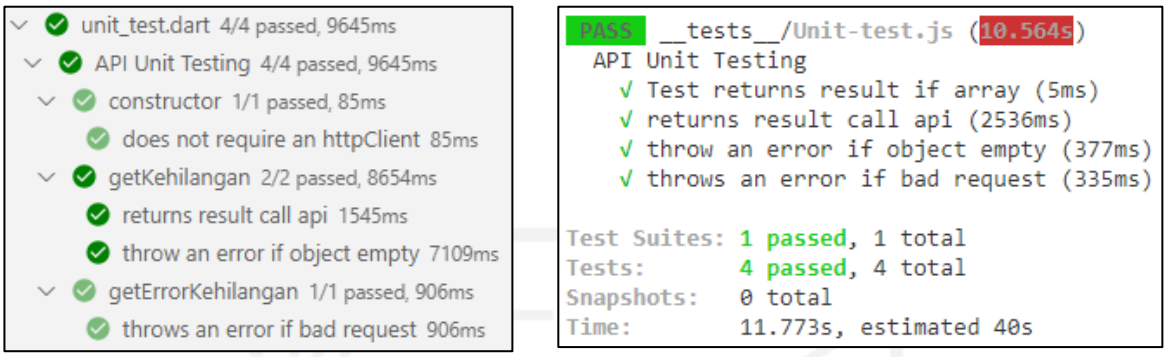
```
14    testWidgets('Test widget Appbar tittle, ', (WidgetTester tester) async {
15      app.HomePage();
16      await binding.traceAction(() async {
17        final Finder kehilangan = find.byKey(Key('tapKehilangan'));
18        await tester.pumpAndSettle(const Duration(seconds: 2));
19        await tester.tap(kehilangan);
20        await tester.pumpAndSettle(const Duration(seconds: 2));
21        expect(
22            find.byWidgetPredicate((widget) =>
23                widget is AppBar &&
24                widget.title is Text &&
25                (widget.title as Text).data.startsWith("Barang Hilang")),
26            findsOneWidget);
27      });
28    });
```

Gambar 4.27 *Integration testing widget* pada Flutter

C. Kesimpulan Testability

Setelah melakukan implementasi *unit dan integration test* pada React Native dan Flutter terdapat beberapa kesimpulan bahwa secara garis besar dalam melakukan kode testing kedua *framework* tidak jauh berbeda serta hasilnya juga tidak berbeda seperti terelihat pada Gambar 4.28. Tetapi dalam React Native perlu upaya lebih dalam memulai *testing* karena React Native tidak didukung secara resmi oleh pengembang dalam melakukan *Integration testing*, sehingga dibutuhkan pihak ketiga untuk melakukan *testing* seperti Jest, Appium, dan Detox.

Berbeda dengan Flutter yang menyediakan berbagai fitur pengujian untuk menguji aplikasi serta memiliki dokumentasi yang lengkap dan bagus. Maka dari itu bisa dibilang Flutter lebih unggul dibandingkan dengan React Native.



Flutter Testing React Native Testing

Gambar 4.28 Hasil Unit Testing

4.2.4 Performance

Pengujian perfomance dilakukan dengan mengukur konsumsi memori yang dibutuhkan oleh aplikasi dan frame per detik pada aplikasi. Pengujian dilakukan dengan 3 mode yaitu React Native *debug*, Flutter *debug*, Flutter *profile*. Pengujian menggunakan scenario penggunaan aplikasi agar dapat mengetahui rata-rata pengunaan memori dan letak untuk mengukur *frame per second.*

Skenario penggunaan aplikasi untuk pengujian:

1. Klik tombol barang hilang menuju halaman daftar barang hilang
2. Menggulirkan daftar barang hilang
3. Buka detail barang hilang
4. Kembali ke daftar barang
5. Klik tombol pencarian pada pojok kanan bawah
6. Buka *keyboard* pencarian
7. Proses pencarian daftar barang hilang
8. Buka detail pencarian barang hilang
9. Kembali ke menu awal

A. Konsumsi Memori

Pengukuran konsumsi memori yang pertama menggunakan *DevTools* yang dimiliki pada masing-masing *framework*. Pengukuran sesuai dengan scenario yang sudah dibuat sehingga menghasilkan pengujian memori seperti terlihat pada Tabel 4.10. Dari tabel tersebut perbedaan hasil konsumsi memori dari Flutter debug dengan React Native debug sangat jauh berbeda. Perbedaan tersebut disebabkan karena menggunakan alat pengujian yang berbeda maka hasilnya tidak dapat dijadika acuan perbandingan yang tepat. Maka dari itu pengujian dilanjutkan dengan menggunakan *profiler* pada Android Sudio. Hasil pengujian memori selebihnya dapat dilihat pada Lampiran B.

Tabel 4.10 Hasil Pengujian Memori dengan *DevTolls*

Framewrok	Rata-rata Memori (Mb)
React Native Debug	17,5
Flutter Debug	75,7
Flutter Profile	24,6

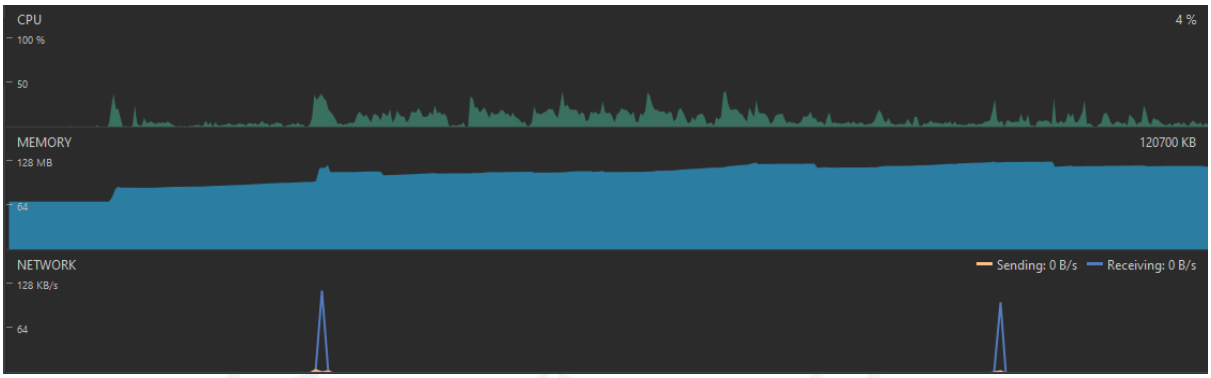
Pengukuran memori dengan menggunakan *profiler* android studio memiliki hasil yang tidak berat sebelah jika dibandingkan dengan mengunakan *DevTolls* pada masing –masing *framework* karena dihitung dengan menggunakan akat yang sama. Hasil dari pengukuran memori menggunakan profiler dapat dilihat pada Tabel 4.11.

Hasil dari pengukuran React Native *debug* dengan Flutter *debug* hampir sama dalam rata-rata mengkonsumsi memori, React Native *debug* sedikit lebih unggul dibanding Flutter *debug*. Jika dilihat pada Gambar 4.29 React Native *debug* cenderung kurang stabil dalam mengkonsumsi memori tetapi sedikit dan konsisten dalam mengkonsumsi CPU. Sedangkan pada Gambar 4.30 Flutter *debug* lebih stabil dalam mengkonsumsi memori tetapi membutuhkan lebih banyak konsumsi CPU dan tidak konsisten.

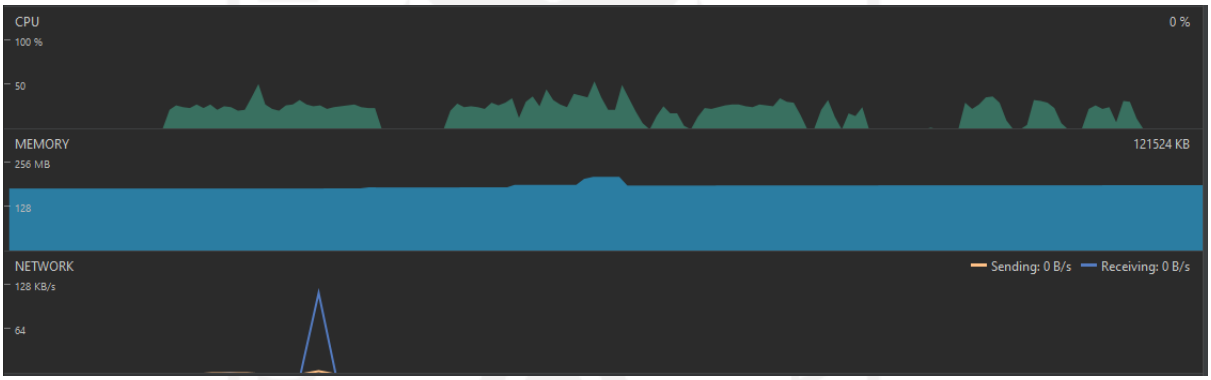
Berbeda dengan mode *debug,* Flutter juga memiliki mode *profile* yang dimana mode ini memungkinkan menjalankan aplikasi dengan kecepatan yang hampir sama seperti aplikasi dalam versi rilis. Jika dilihat pada Tabel Tabel 4.11 Flutter mode *profile* mengkonsumsi hampir setengah dari pengukuran pada pada mode *debug*. Dibandingkan dengan mode *debug,* hasil dari Flutter *profile* lebih stabil dalam mengkonsumsi memori dan CPU seperti terlihat pada Gambar 4.31.

Tabel 4.11 Hasil Pengujian Memori dengan Android Studio

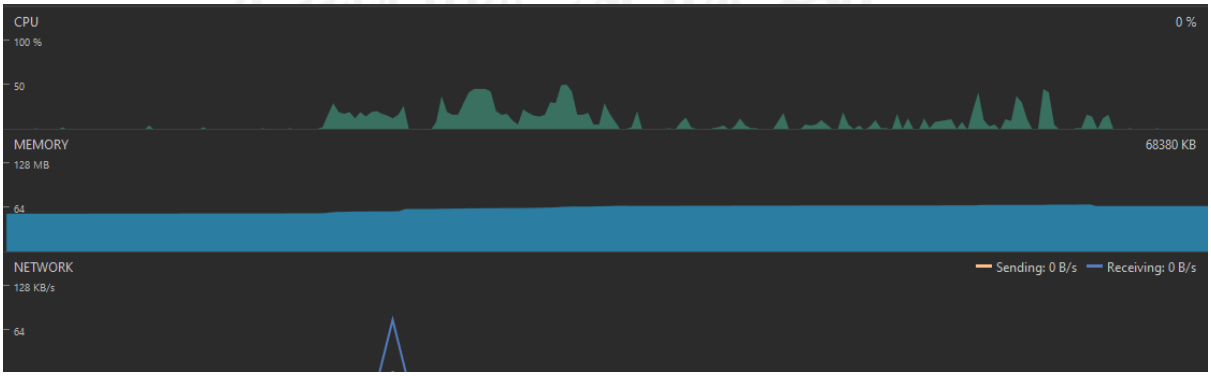
Framework	Rata-rata Memori (Mb)
React Native Debug	120
Flutter Debug	121
Flutter Profile	**68**



Gambar 4.29 Hasil *profiling* React Native mode *debug*



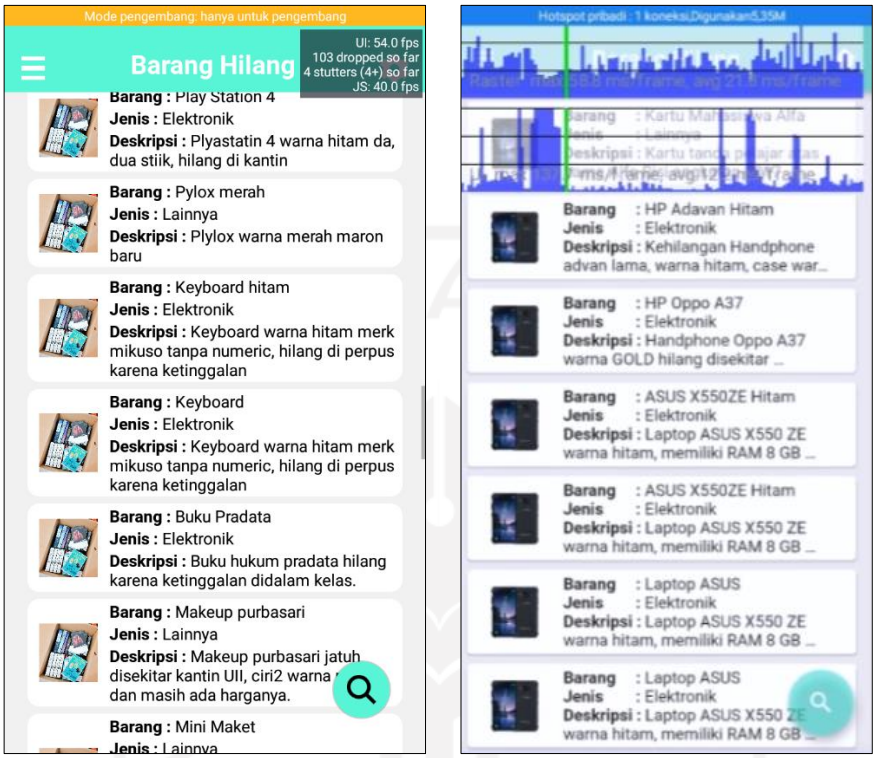
Gambar 4.30 Hasil *profiling* Flutter mode *debug*



Gambar 4.31 Hasil *profiling* Flutter mode *profile*

B. Frame per Second

Pengukuran *frame per second* menggunakan skenario penggunaan aplikasi yang sudah dibuat. Masing-masing skenario akan diukur menggunakan fitur yang ada dalam masing *framework* seperti terlihat pada Gambar 4.32.



Pengukuran FPS pada React Native Pengukuran FPS pada Flutter

Gambar 4.32 Fitur proses pengukuran FPS

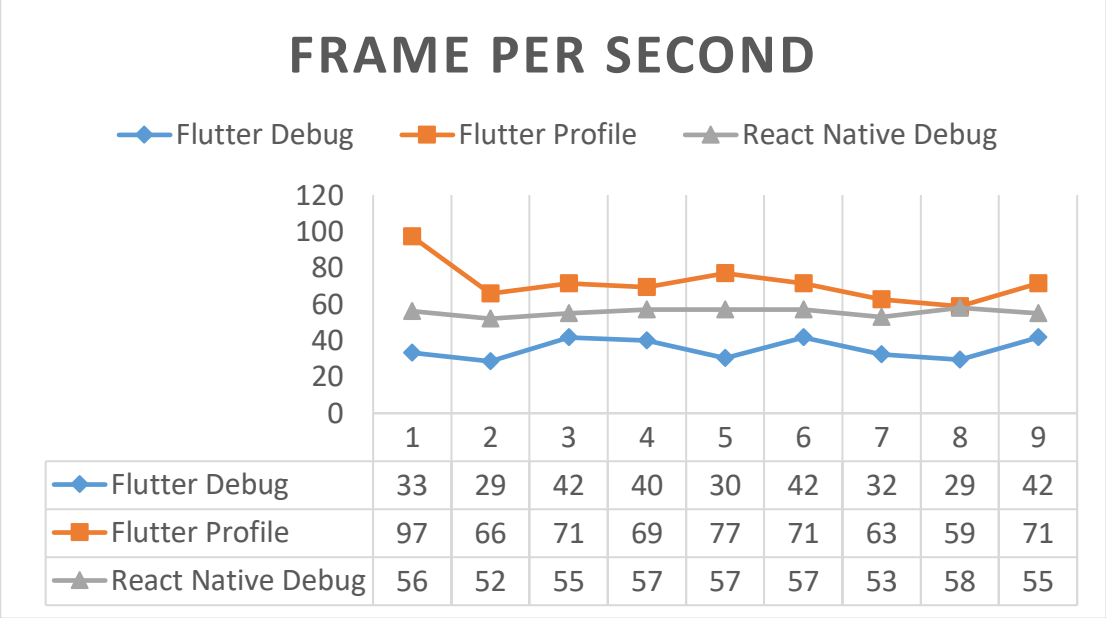
Pada Gambar 4.32 Flutter tidak menggunakan *fps* dalam pengukurannya, tetapi menggunakan *millisecond* per *frame* (*ms/frame*). Untuk mengubah dari *ms* menjadi *fps* yaitu dengan membagi 1000 dengan (ms/frame) seperti contoh yang terlihat pada persamaan (4.1). Hasil dari pengukuran yang dilakukan pada *user interface thread* dapat dilihat pada Tabel 4.12.

$$17\ ms = \frac{1000}{17} = 59\ fps \tag{4.1}$$

Tabel 4.12 Hasil Pengukuran FPS pada *User Interface*

Skenario	React Native Debug	Flutter Debug	Flutter Profile
1	56	63	123
2	52	33	97
3	55	29	66
4	57	42	71
5	57	40	69
6	57	30	77
7	53	42	71
8	58	32	63
9	55	29	59
Rata-rata	**56**	**38**	**77**

Pada Tabel 4.12 menyajikan jumlah rata-rata *frame per second* dalam pengujian yang berjalan pada skenario. Dari hasil tersubut Flutter *profile* lebih halus dan lancer, sedangkan dalam mode *debug* React Native lebih lancer dibandingkan dengan Flutter. Untuk lebih jelasnya dapat dilihat pada Gambar 4.33, dari gambar tersebut menampilkan grafik rata-rata pada setiap skenario yang diukur. Aplikasi sengaja dibuat dengan komputasi yang berat sehingga FPS yang dihasilkan rendah. Pada pengukuran *frame per second* saat ini kurang efektif karena menggunakan alat yang berbeda, dengan perbedaan alat memungkinkan hasil tidak terlalu akurat, sehingga membutuhkan alat yang sama untuk mengukur *frame pre second*.



Gambar 4.33 Grafik hasil pengukuran FPS *User Interface Thread*

C. Kesimpulan Performance

Dari pengukuran konsumsi memori dan *frame per seconds* dapat disimpulkan bahwa konsumsi momori tidak jauh berbeda antara keduanya. React Native lebih berfokus pada kestabilan pada CPU sedangkan Flutter pada memori. Sedangkan dalam pengukuran *frame per second* pada *user interface thread* React Native debug lebih unggul dibandingkan dengan Flutter.

Perlu diketahui bahwa UI *thread* pada Flutter dapat dibilang sama dengan yang ada pada React Native, tetapi jangan diperlakukan sama karena kedua *thread* melakukan pekerjaan yang berbeda. *Thread* pada Flutter melakukan eksekusi pada kerangka kerja Flutter dan kode aplikasi yang dibuat di Dart yang dikerjakan pada *virtual machine* Dart. Ketika Flutter membuat lapisan tampilan untuk ditampilkan, maka UI *thread* akan membuat pohon tampilan untuk dikirimkan ke GPU yang berjalan pada CPU dalam perangkat lunak untuk me-*render* melalui *raster thread*.

Sedangkan React Native membuat pohon tampilan dari UI *thread* akan dikirim ke JavaScript *thread* dan diproses oleh memori untuk ditampilkan. Maka dari itu penggunaan memori pada Flutter lebih stabil dibandingkan dengan React Native, sebaliknya penggunaan CPU pada React Native lebih stabil dibandingkan dengan Flutter. Selain itu pengunaan perangkat keras juga berpengaruh terhadap hasil pengukuran *performance*, pada saat pengukuran sebaiknya menggunakan perangkat yang lebih stabil dalam CPU maupun memori.

4.3 Kesimpulan Komparasi

Keseluruhan hasil dari proses pengukuran dan pengujian dengan domain *customizability*, *modifiability*, dan *performance* dapat dilihat pada Tabel 4.13 serta domain *testability* yang sudah dilakukan pada pembahasan sebelumnya.

Tabel 4.13 Hasil komparasi framework Flutter dan React Native

Domain	React Native	Flutter
Customizability		
Components	Mature	Mature
File Structure	Capable	Capable
Setup and Configuration	**Capable**	Aware
Modifiability		
Jumlah Fungsi	57	**51**
Cohesion Procedural	0,52	**0,54**
Coupling Data	**0,0867**	0,0902
Performance		
Memori Debug Mode (Mb)	**120**	121
Memori Profile Mode (Mb)	-	**68**
FPS Debug Mode	**56**	33
FPS Profile Mode	-	**77**

Hasil pengukuran pada *customizability* menunjukkan bahwa pengukuran secara keseluruhan React Native sedikit lebih unggul karena Flutter menawarkan kustomisasi standar pada *setup* dan *configuration*. React Native didukung berbagai *library* atau *tools* yang digunakan dalam melakukan *setup* dan *configuration*. Walaupun Flutter memiliki kustomisasi standar pada *setup* dan *configuration* itu sudah sangat cukup untuk membangun sebuah aplikasi berbasis android. React Native baik digunakan ketika membutuhkan berbagai varian konfigurasi dalam pengembangan.

Hasil selanjutnya adalah pada arsitektur yang memiliki kualitas *modifiability* terbaik, yang dimana dapat dikatakan baik jika *cohesion* tinggi, *coupling* rendah dan jumlah fungsi rendah. Dalam hal ini tingkat *cohesion* yang digunakan adalah *procedural* sedangkan tingkat *coupling* adalah data. Hasil penelitian menunjukkan bahwa Flutter BloC baik dalam tingkat *cohesion procedural* dan jumlah fungsi, selain itu React Native Redux baik pada tingkat *coupling* data. Maka dari itu, Flutter BloC baik digunakan ketika ingin meminimalkan fungsi dalam *components* karena dapat mengurangi tugas antar komponen, sedangkan React Native Redux baik dalam melakukan pengelolaan data antar *components* karena memiliki komponen yang banyak daripada Flutter BloC.

Hasil pengukuran yang baik dalam *performance* berdasarkan konsumsi memori yang lebih sedikit dan *frame per second* yang tinggi. Hasil pengukuran memori mode *debug* pada React Native dan Flutter hanya memiliki selisih sangat sedikit, yaitu dalam pengukuran ini sebesar satu *megabyte* yang diunggulkan oleh React Native. Begitu pula dengan pengukuran *frame per second* menggunakan DevTools diunggulkan oleh React Native. Flutter diunggulkan karena adanya mode *profile* yang berguna untuk mengetahui *performance* seperti pada saat aplikasi rilis. Perlu diketahui bahwa pengukuran ini memiliki studi kasus sendiri yang sederhana, berbeda jika memiliki studi kasus kompleks atau skala besar yang memungkinkan perbedaan *performance* akan terlihat lebih jelas. Salah satunya dalam penelitian ini React Native lebih ketergantungan terhadap *library*, semakin banyak *library* yang digunakan dalam suatu aplikasi memungkinkan adanya sedikit penurunan *performance* terutama pada *library* yang tidak stabil karena menambah beban pada *bridge* dalam melakukan komunikasi dua arah dengan teknologi yang berbeda. Flutter baik digunakan ketika pengembang ini menigkatkan *performance* dengan adanya minimalisir penggunaan *library* pada aplikasi, sedangkan React Native bagus digunakan ketika menginginkan *performance* yang baik dengan banyaknya pengelolaan *library*.

Implementasi *testability* dari masing-masing kerangka kerja tidak jauh berbeda, tetapi Flutter lebih unggul dibandingkan React Native dalam segi fasilitas *testing*. Flutter menawarkan beragam pengujian seperti *unit testing, integration testing,* dan lainnya dengan dokumentasi yang lengkap dan mudah. Sedangkan untuk React Native hanya menawarkan *unit testing* dalam dokumentasi sehingga harus menggunakan pihak ketiga untuk melakukan *integration testing* seperti Jest, Appium, dan Detox. Maka dari itu, Flutter bagus digunakan ketika pengembang menginginkan kemudahan dalam melakukan pengujian aplikasi dengan dokumentasi bawaan kerangka kerja, sedangkan React Native bagus digunakan ketika pengembang ini menggunakan berbagai variasi alat pengujian aplikasi dengan bantuan *library* seperti Appium, Detox, dan Jest. .

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan data yang diperoleh dari implementasi dan pengujian dapat diperoleh kesimpulan sebagai berikut:

1. Flutter lebih unggul pada domain *modifiability* dan *testabilty*. Hal tersebut dikarenakan Flutter memiliki lebih sedikit jumlah fungsi dan ketergantungan antara komponen pada pola desain serta menawarkan beberapa macam fitur pengujian bawaan. Dengan demikian Flutter bagus digunakan untuk melakukan perubahan atau penambahan fitur pada aplikasi dan pengujian aplikasi.
2. React Native lebih unggul pada domain *customizability* dan *performace*. Hal tersebut disebabkan karena React Native menawarkan berbagai konfigurasi kustomisasi dan memiliki kinerja pada *virtual machine* yang bagus dalam komunikasi dua arah dengan aplikasi skala kecil. Dengan demikian React Native bagus digunkan untuk melakukan perancangan awal fitur pada aplikasi pada dan bagus dalam kemudahan memaksimalkan kinerja aplikasi.

5.2 Saran

Pada saat mengerjakan penelitian ini, proses-proses yang dilakukan masih jauh dari kata sempurna. Oleh karena itu, saran sangat dibutuhkan guna untuk penelitian yang akan datang sebagai berikut:

1. Pengujian *customizability* dapat dilakukan dengan mengunakan metode lainnya yang lebih akurat dalam mengukur.
2. Pengujian *modifability* tidak hanya menghitung jumlah fungsi tetapi juga dapat dilakukan dengan menghitung banyak baris kode program.
3. Pengujian *performance* pada FPS (*frame per second*) dapat dilakukan dengan menggunakan alat yang sama.
4. Pengujian *performance* dapat dilakukan dengan menggunakan hardware yang memiliki spesifikasi yang lebih tinggi dan stabil agar tidak berdampak pada saat pengujian.
5. Implementasi *testability* dapat dilakukan dengan berbagai *library* yang lain terutama pada React Native.

DAFTAR PUSTAKA

Abramov, D. (2020). *Redux data flow*. https://redux.js.org/basics/basic-tutorial

BloC Architecture. (2020). https://bloclibrary.dev/#/architecture

Caicedo, C. (2015). *Cross-Platform Mobile Application Development Abstract : October*.

Faust, S., & Klein, V. (2020). Using Google ´ s Flutter Framework for the Development of a Large-Scale Reference Application. *Tribal*.

Fayzullaev, J., Supervisor, X., & Mynttinen, T. (2018). *Native-like Cross-Platform Mobile Development : Multi-OS Engine & Kotlin Native vs Flutter*. https://www.theseus.fi/bitstream/handle/10024/148975/thesis_Jakhongir_Fayzullaev.pdf?sequence=1

Flutter Api. (2020). *StatefulWidget class*. https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

Flutter dev. (2021). *Flutter architectural overview*. Flutter.Dev. https://flutter.dev/docs/resources/architectural-overview

Flutter web. (2020). *Flutter*. https://flutter.dev/

Garousi, V., Felderer, M., & Kılıçaslan, F. N. (2019). A survey on software testability. *Information and Software Technology*, *108*, 35–64. https://doi.org/10.1016/j.infsof.2018.12.003

Gerdessen, A. (2007). *Framework comparison method Comparing two frameworks based on technical domains, focussing on customisability and modifiability*. *August*. https://homepages.cwi.nl/~paulk/thesesMasterSoftwareEngineering/2007/AntonGerdessen.pdf

Hansson, N., & Vidhall, T. (2016). *Effects on performance and usability for cross-platform application development using React Native*. 92.

Jagie, J. (2019). *Performance comparison between react native and flutter*.

Jiao, J., & Tseng, M. M. (2004). Customizability analysis in design for mass customization. *CAD Computer Aided Design*, *36*(8), 745–757. https://doi.org/10.1016/j.cad.2003.09.012

Kuitunen, M. (2019). *Cross-Platform Mobile Application Development with React Native*. *February*, 1–26. https://trepo.tuni.fi/bitstream/handle/123456789/27139/Kuitunen.pdf?sequence=4&isAllowed=y#page=16&zoom=100,0,98

Kuparinen, E. (2019). *Managing application state and control flow using Redux and Redux-Saga in a web application*. https://lutpub.lut.fi/handle/10024/160107

Liu, J., & Yu, J. (2011). Research on development of android applications. *Proceedings - 2011 4th International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2011*, 69–72. https://doi.org/10.1109/ICINIS.2011.40

Occhino, T. (2015). *React Native: Bringing modern web techniques to mobile*. Engineering.Fb.Com. https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/

Oshana, R. (2013). Software Performance Engineering for Embedded Systems. In *Software Engineering for Embedded Systems* (First Edit). Elsevier Inc. https://doi.org/10.1016/B978-0-12-415917-4.00010-4

Pusparisa, Y. (2020). *Pengguna Smartphone diperkirakan Mencapai 89% Populasi pada 2025*. Databoks. https://databoks.katadata.co.id/datapublish/2020/09/15/pengguna-smartphone-diperkirakan-mencapai-89-populasi-pada-2025

Quazi, F. U. R., & Sinha, N. (2018). Android-Platform Based Determination of Fastest Cross-Platform Framework. *International Journal of Computer Science and Mobile Computing*, *7*(9), 1–12. http://files/186/Quazi and Sinha - 2018 - Android-Platform Based Determination of Fastest Cr.pdf

React Fundamentals. (2020). https://reactnative.dev/docs/intro-react

ReactJS. (2020). *Components and Props*. https://reactjs.org/docs/components-and-props.html

Romeo. (2003). Testing dan Implementasi Sistem. *Testing Dan Implementasi Sistem*, 52.

Saliu, M. O., Ruhe, G., Lindvall, M., & Ackermann, C. (2009). Chapter 7 Evaluating the Modifiability of Software Architectural Designs. *Advances in Computers*, *77*(09), 243–297. https://doi.org/10.1016/S0065-2458(09)01207-8

Salohonka, M. (2020). *Automated Testing of React Native Applications*. 93. http://files/186/Quazi and Sinha - 2018 - Android-Platform Based Determination of Fastest Cr.pdf

Sharma, Y., & Gupta, S. (2020). *A Study of Flutter and React Native for Mobile App Development*. *27*(27), 691–698.

Shergin, V., Vacca, D., Baets, P. De, & Nara, R. (2021). *Github React Native*. https://github.com/facebook/react-native

Sholichin, F., Adham, M., Halim, S. A., & Firdaus Bin Harun, M. (2019). Review of Ios Architectural Pattern for Testability, Modifiability, and Performance Quality. *Journal of*

Theoretical and Applied Information Technology, *15*(15), 4021–4035. www.jatit.org

Stackoverflow. (2019). *Developer Survey Results*. https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-other-frameworks-libraries-and-tools

Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. (2008). Software as a service: Configuration and customization perspectives. *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008, PART2*, 18–24. https://doi.org/10.1109/SERVICES-2.2008.29

Waranashiwar, J., & Ukey, M. (2018). *Ionic Framework with Angular for Hybrid App Development*. *5*, 1–2.

Witte, D., & Weitershausen, P. (2015). *React Native for Android: How we built the first cross-platform React Native app*. https://engineering.fb.com/developer-tools/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/

Wu, W. (2018). *React Native vs Flutter, cross-platform mobile application frameworks*. *March*, 1–7. https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1

Zhao, X., Khomh, F., & Zou, Y. (2011). Improving the modifiability of the architecture of business applications. *Proceedings - International Conference on Quality Software*, 176–185. https://doi.org/10.1109/QSIC.2011.36

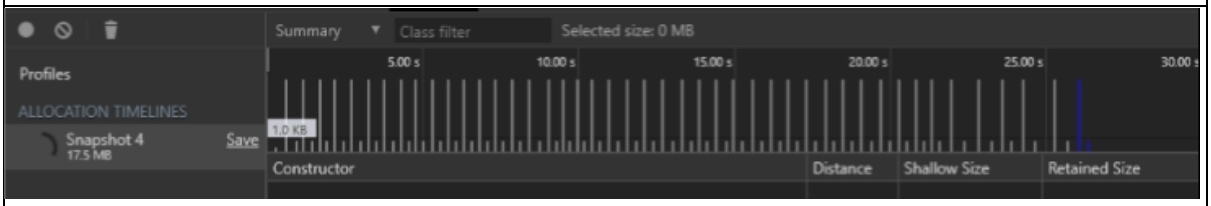
LAMPIRAN

Lampiran A. Kode Customizabiltiy Components React Native

```
import React, { Component } from 'react';

import { View, Text, Image, TouchableOpacity, StyleSheet } from 'react-native';
import images from '../res/images'
import {Header} from 'react-native-elements';
import PropTypes from 'prop-types';

class HeaderComponents extends Component{
    render(){
        return (
            <View>
                <Header
                    leftComponent={
                        <TouchableOpacity
                        onPress={this.props.leftOnPress}>
                            <View style={{ paddingLeft: 15 }}>
                                <Image
                                    style={{ width: 22, height: 22 }}
                                    source={images.bars} />
                            </View>
                        </TouchableOpacity>
                    }
                    centerComponent={
                        <View>
                            <Text style={styles.textCenter}>{this.props.centerText}</Text>
                        </View>
                    }
                    rightComponent={
                        <TouchableOpacity
                        onPress={this.props.rightOnPress}>
                            <View style={{ paddingRight: 15 }}>
                                <Image
                                    style={{ width: 24, height: 24 }}
                                    source={images.cog} />
                            </View>
                        </TouchableOpacity>
                    }
                />
            </View>
        )
    }
}

HeaderComponents.propTypes = {
    centerText: PropTypes.string,
    leftOnPress: PropTypes.func,
    rightOnPress: PropTypes.func,
};

export default HeaderComponents;

const styles = StyleSheet.create({
    textCenter:{
        color: 'white',
        fontSize: 22,
        fontWeight: 'bold'
    },
})

```

Lampiran B. Hasil Pengujian Konsumsi Memori Menggunakan *DevTolls*

Konsumsi memori pada React Native *debug*



Konsumsi memori pada Flutter *debug*

Konsumsi memori pada Flutter Profiler

UNIVERSITAS ISLAM INDONESIA