

TUGAS AKHIR

RANCANG BANGUN ALAT PENGUBAH KOTLIN DATA
CLASS KE PROTOCOL BUFFERS MESSAGE



DISUSUN OLEH:

FATKHI NUR AKHSAN

NIM. 20106050026

PROGRAM STUDI INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
YOGYAKARTA

2024

INTISARI

Perkembangan teknologi, terutama dalam perangkat komputasi mobile dan smartphone, telah mendorong penggunaan aplikasi mobile secara masif. Dua sistem operasi yang mendominasi pasar adalah Android dan iOS. Dalam pengembangan aplikasi Android, Google telah mengadopsi Kotlin sebagai bahasa resmi sejak 2017. Salah satu fitur Kotlin adalah Data Class yang memudahkan penyimpanan data. Namun, penggunaan Jetpack DataStore untuk penyimpanan data di Android memerlukan definisi skema dengan Protocol Buffers, yang memiliki sintaks berbeda dengan Kotlin Data Class. Hal ini menimbulkan tantangan bagi developer dalam mengonversi struktur data Kotlin ke Protocol Buffers. Tugas akhir ini bertujuan untuk merancang dan membangun alat yang dapat mengubah Kotlin Data Class menjadi Protocol Buffers Message secara otomatis. Alat ini diimplementasikan sebagai plugin untuk IntelliJ IDEA dan Android Studio, yang diharapkan dapat menyederhanakan proses konversi, meminimalisir kesalahan, dan meningkatkan produktivitas developer.

Kata Kunci: Protocol Buffers, Protobuf, Kotlin, Data Class, Konversi, Plugin, Pengembangan Perangkat Lunak.

ABSTRACT

The rapid advancement of technology, particularly in mobile computing and smartphones, has led to massive adoption of mobile applications. Android and iOS are the two dominant operating systems. In Android application development, Google adopted Kotlin as the official language in 2017. One of Kotlin's features is Data Class, which facilitates data storage. However, using Jetpack DataStore for data storage in Android requires schema definitions with Protocol Buffers, which have different syntax from Kotlin Data Class. This presents a challenge for developers in converting Kotlin data structures to Protocol Buffers. This thesis aims to design and develop an automated tool to convert Kotlin Data Class to Protocol Buffers Message. The tool is implemented as a plugin for IntelliJ IDEA and Android Studio, which is expected to simplify the conversion process, minimize errors, and enhance developer productivity.

Kata Kunci: Protocol Buffers, Protobuf, Kotlin, Data Class, Conversion, Plugin, Software Development.

KATA PENGANTAR

DAFTAR ISI

INTISARI.....	i
ABSTRACT.....	ii
DAFTAR ISI	iv
DAFTAR GAMBAR	vi
DAFTAR TABEL.....	vii
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	5
1.3 Tujuan.....	5
1.4 Manfaat.....	5
BAB II KAJIAN PUSTAKA	7
2.1 Kotlin.....	7
2.2 Kotlin Data Class	8
2.3 Protocol Buffers	8
2.4 Protocol Buffers Message	9
2.5 Plugin	10
BAB III METODE PENGEMBANGAN SISTEM	11
3.1 Alat dan Bahan	11
3.2 Metode Pengembangan	11
BAB IV PERANCANGAN DAN IMPLEMENTASI SISTEM.....	14
4.1 Perencanaan (<i>Planning</i>)	14
4.1.1 Analisis Kebutuhan Fungsional.....	15
4.1.2 Analisis Kebutuhan Non-Fungsional.....	16
4.2 Perancangan (<i>Design</i>)	16
4.2.1 Activity Diagram	17
4.2.2 Wireframe	18
4.3 Implementasi (<i>Coding</i>).....	20
4.3.1 Iterasi Pengembangan Pertama	20
4.3.2 Iterasi Pengembangan Kedua	24

4.3.3 Iterasi Pengembangan Ketiga	25
4.4 Pengujian (<i>Testing</i>).....	26
4.4.1 Unit Testing	27
4.4.2 Black Box Testing.....	29
4.4.3 Panduan Pengguna.....	32
4.4.4 Simulasi Hasil.....	34
BAB V KESIMPULAN DAN SARAN.....	37
5.1 Kesimpulan.....	37
5.2 Saran.....	37

DAFTAR GAMBAR

Gambar 1 Tahapan Metode Extreme Programming.....	12
Gambar 2 Activity Diagram Plugin	17
Gambar 3 Desain Wireframe Plugin	19
Gambar 4 Editor Popup Menu	21
Gambar 5 Protocol Buffers Keywords	22
Gambar 6 Scalar Type	23
Gambar 7 Graphical User Interface Plugin Form	24
Gambar 8 Error Dialog.....	25
Gambar 9 Success Notification	25
Gambar 10 Hasil Pengujian PluginExceptionTest	27
Gambar 11 Hasil Pengujian GenerateProtobufActionControllerTest	28
Gambar 12 Hasil Pengujian GenerateActionListener	28
Gambar 13 Hasil Pengujian KotlinDataTypeParser	29
Gambar 14 Instalasi Plugin	32
Gambar 15 File ThemeModel.kt	34
Gambar 16 File theme_prefs.proto	35
Gambar 17 Generated java files from proto file	35
Gambar 18 Aplikasi android “ProtoTheme” dengan tema dalam mode terang.....	36
Gambar 19 Aplikasi android “ProtoTheme” dengan tema dalam mode gelap	36

DAFTAR TABEL

Tabel 1 Pengujian Black Box.....	30
----------------------------------	----

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi dalam beberapa dekade terakhir sangatlah pesat, salah satu teknologi yang berkembang pesat adalah perangkat komputasi mobile dan smartphone dalam 3 dekade terakhir. Saat ini perangkat genggam sudah memiliki daya komputasi yang sama besarnya dengan ruangan yang penuh dengan komputer pada tahun 1970-an. Perkembangan daya komputasi tersebut tidak terlepas dari peningkatan tajam jumlah transistor yang dapat ditampung dalam sebuah *chip* akan berlipat ganda setiap dua tahun, sesuai dengan hukum Moore [1].

Meskipun memiliki kecepatan proses dan konektivitas yang masih lebih lambat dibandingkan dengan perangkat dekstop, penggunaan perangkat mobile dan aplikasi mobile lebih dipilih oleh pengguna dibandingkan dengan perangkat dekstop terutama untuk menyelesaikan pekerjaan yang mudah [2]. Salah satunya yaitu aplikasi *mobile-commerce* yang marak digunakan ketika masa pandemi COVID-19, penggunaan aplikasi *mobile-commerce* juga diprediksi masih berlanjut saat periode endemi [2]. Senada dengan hal itu, laporan dari Cisco menunjukkan bahwa hampir 300 miliar aplikasi seluler akan diunduh pada tahun 2023, dengan aplikasi berjenis media sosial, game, dan bisnis akan menjadi unduhan yang paling populer [3].

Diantara penggunaan masif perangkat mobile terdapat dua sistem operasi yang sudah mendominasi pasar dengan cakupan 90% dari market, yaitu Android (Google, Mountain View, CA) and iOS (Apple) sejak tahun 2008 [1]. Untuk mendukung produktifitas developer, Google mengumumkan dukungannya pada Google I/O 2017 untuk menjadikan bahasa pemrograman Kotlin sebagai bahasa resmi untuk pengembangan aplikasi android [4]. Setelah dua tahun berselang, tepatnya saat Google I/O 2019, pengembangan aplikasi mobile android akan menjadi *kotlin-first* setelah Google mengumumkannya, hal ini berarti Kotlin akan menjadi pilihan pertama Google dalam mengembangkan alat dan konten pengembangan aplikasi android [5].

Bahasa pemrograman Kotlin sendiri merupakan bahasa pemrograman open source berjenis statis yang mendukung pemrograman berorientasi objek dan fungsional [6]. Kotlin pada mulanya dirancang oleh JetBrains, kemudian terus dikembangkan oleh para kontributor dan dikelola oleh Kotlin Foundation yang didirikan oleh JetBrains dan Google [7]. Meskipun tergolong bahasa yang modern, Kotlin juga dapat dikatakan bahasa yang sudah matang. Bahasa ini ringkas, aman, dapat dioperasikan dengan Java dan bahasa lain, dan menyediakan banyak cara untuk menggunakan kembali kode di antara berbagai platform untuk pemrograman yang produktif [8]. Oleh karena itu, beberapa API pada Android seperti AndroidKTX yang dikhususkan untuk Kotlin tetap dapat kompatibel dengan Bahasa Java yang sudah lebih dulu menjadi bahasa resmi untuk pengembangan aplikasi android. Dengan interoperabilitas yang baik dengan Java tersebutlah, Kotlin berhasil menghasilkan pengalaman pengembangan aplikasi android yang lebih nyaman [6].

Salah satu library yang memiliki AndroidKTX sebagai ekstensi adalah library Jetpack DataStore. Jetpack DataStore adalah salah satu library yang menyediakan solusi penyimpanan data lokal yang terdapat pada platform android, khususnya untuk kumpulan data yang kecil dan sederhana dan tidak memerlukan pembaruan parsial atau integritas referensial. Jetpack DataStore dapat menyimpan data berupa pasangan *key-value* atau objek yang ditulis menggunakan protocol buffers. Oleh karena itu, Jetpack DataStore dibagi menjadi dua jenis, yaitu Preferences DataStore, yang menyimpan dan mengakses data menggunakan kunci dan Proto DataStore, yang menyimpan data berupa instansiasi objek yang sebelumnya telah didefinisikan menggunakan schema yang ditulis menggunakan protocol buffers. Jetpack DataStore menggunakan Kotlin Coroutines dan Flow untuk menyimpan data secara asinkron, konsisten, dan transaksional [9].

Protocol Buffers atau yang juga dapat disingkat dengan protobuf, merupakan mekanisme yang dapat diekstensi untuk membuat serialisasi data terstruktur yang *language-neutral* dan *platform-neutral*. Protocol Buffers sendiri unggul dalam hal ukuran dan kecepatan dibandingkan dengan format data lain seperti JSON dan XML, dan dapat menghasilkan binding bahasa asli. Protocol Buffers adalah

kombinasi bahasa definisi (dibuat dalam file `.proto`), kode yang dibuat oleh kompiler proto untuk berinteraksi dengan data, pustaka runtime yang *language-specific*, format serialisasi untuk data yang ditulis ke file (atau dikirim melalui koneksi jaringan), dan data yang terserialisasi [10].

Salah satu cara untuk menampung data pada bahasa pemrograman kotlin adalah dengan menggunakan Data Class yang ditandai dengan keyword “data”. Untuk setiap Data Class, kompiler akan secara otomatis menghasilkan fungsi tambahan yang memungkinkan untuk mencetak instance ke output yang dapat dibaca, membandingkan instance, menyalin instance, dan banyak lagi. Hal ini mengurangi kode *boilerplate* yang dihasilkan dibandingkan dengan mekanisme penampungan data yang terdapat pada bahasa pemrograman lain, seperti POJO pada bahasa Java [11]. Sama halnya dengan Data Class, Protocol Buffers juga erat berkaitan dengan data-data terutama untuk mendefinisikan protokol komunikasi (bersama dengan gRPC) dan untuk penyimpanan data. Struktur data pada Protocol Buffers didefinisikan di file-file dengan ekstensi `.proto` yang dibuat oleh engineer. Pada file tersebut service-service dan message-message didefinisikan. Message inilah yang dijadikan schema untuk memperoleh kode dan runtime library yang lebih *language-specific* sehingga dapat digunakan pada berbagai bahasa dan *framework* pemrograman, khususnya dalam konteks ini yaitu untuk menyimpan data menggunakan Jetpack Proto DataStore [10].

Meskipun keduanya, yaitu Kotlin Data Class dan Protocol Buffers memiliki beberapa persamaan. Namun, keduanya memiliki perbedaan sintaks, hal ini menjadi permasalahan dan tantangan tersendiri bagi developer android yang ingin menggunakan Jetpack Proto DataStore sebagai solusi penyimpanan lokal pada aplikasi yang dikembangkan. Semakin kompleks struktur data yang dipakai tentunya akan menambahkan tingkat kesulitan yang dialami developer dalam merepresentasikannya kedalam bentuk skema yang dibuat dari Protocol Buffers Message. Bahkan, ketika struktur data tersebut tidaklah kompleks pengembang yang belum pernah memakai Jetpack Proto DataStore tetap perlu untuk mempelajari Protocol Buffers. Proses pembuatan skema inilah yang seringkali memakan waktu dan rentan terhadap kesalahan, terutama ketika berhadapan dengan

struktur data yang rumit dikarenakan prosesnya yang masih manual [12]. Permasalahan tersebutlah yang menciptakan kebutuhan akan sebuah alat yang dapat mengotomatisasi proses perubahan struktur data yang berasal dari Kotlin Data Class menjadi struktur data berupa skema yang didefinisikan melalui Protocol Buffers Message dengan cepat dan efisien. Kebutuhan tersebut sangat penting bagi pengembang yang perlu bekerja dengan data yang sudah ada dalam bentuk struktur data Kotlin Data Class kemudian ingin memperolehnya dalam bentuk Protocol Buffers Message dengan cepat dan efisien.

Untuk mengatasi permasalahan dan memenuhi kebutuhan dari pengembang tersebut, diperlukan sebuah alat yang dapat mengubah struktur atau model data dari Kotlin Data Class ke dalam bentuk Protocol Buffers Message secara otomatis. Alat tersebut diharapkan dapat mengatasi permasalahan dan memenuhi kebutuhan sekaligus menghadirkan beberapa manfaat bagi para pengembang, terutama pengembang aplikasi android, seperti: menyederhanakan dan mempercepat proses pembuatan Protocol Buffers Message sebagai skema data yang sebelumnya sudah pernah dibuat dalam bentuk Kotlin Data Class, meminimalisir kesalahan yang mungkin terjadi selama proses perubahan dan pembuatan skema, memudahkan pengembang dalam memelihara kode program [13]. Penggunaan alat konversi otomatis juga dapat meningkatkan konsistensi dalam struktur kode sehingga praktik-praktik kode yang baik dapat dijalankan dengan lebih mudah, memfasilitasi pembaruan yang lebih mudah ketika skema data berubah, dan memungkinkan pengembang untuk lebih berfokus pada logika bisnis inti daripada tugas-tugas transformasi data yang repetitif [14].

Terdapat dua plugin yang menjadi inspirasi dalam membuat perancangan dan pembangunan alat yang dapat mengubah Kotlin Data Class menjadi Protocol Buffers Message secara otomatis dalam bentuk plugin yang dapat dipasang dan berjalan pada IntelliJ IDEA dan Android Studio. Pertama, RoboPOJOGenerator, yaitu plugin yang dapat menghasilkan file Java, Java Records, dan Kotlin POJO dari JSON: GSON, FastJSON, AutoValue (GSON), Logan Square, Jackson, Lombok, Jakarta JSON Binding, dan *empty annotations template* [27]. Kedua, pojo to proto, yaitu plugin yang dapat menghasilkan google protobuf message dari Java

POJO *class* [28]. Namun, keduanya tidak melakukan konversi dari Kotlin Data Class menjadi Protocol Buffers Message. Juga, belum ditemukan plugin untuk IDE IntelliJ IDEA maupun Android Studio dengan fungsionalitas yang sama sejauh pencarian penulis.

Hal-hal tersebutlah yang memotivasi penulis untuk merancang dan membangun alat yang dapat mengubah Kotlin Data Class menjadi Protocol Buffers Message secara otomatis dalam bentuk plugin yang dapat dipasang dan berjalan pada IntelliJ IDEA dan Android Studio, dimana IntelliJ IDEA sendiri merupakan Integrated Development Environment (IDE) terdepan dalam pengembangan yang menggunakan bahasa Kotlin [15], kemudian Android Studio sebagai IDE Resmi untuk mengembangkan aplikasi Android [16].

1.2 Rumusan Masalah

Dari latar belakang yang telah diuraikan, masalah dapat dirumuskan sebagai berikut: “Bagaimana cara merancang dan membangun alat yang dapat mengubah struktur data pada Kotlin Data Class menjadi skema yang didefinisikan melalui Protocol Buffers Message?”

1.3 Tujuan

Dari rumusan masalah yang telah disebutkan, tujuan yang ingin dicapai adalah untuk merancang dan membangun alat yang dapat mengubah struktur data pada Kotlin Data Class menjadi skema yang didefinisikan melalui Protocol Buffers Message berupa plugin yang dapat dipasang dan dijalankan pada IntelliJ IDEA dan Android Studio.

1.4 Manfaat

Hasil dari rancang bangun ini diharapkan dapat memberikan beberapa manfaat sebagai berikut:

- Membantu pengembang aplikasi android yang menggunakan bahasa pemrograman kotlin dalam membangun aplikasi android yang menggunakan *library* Jetpack Proto DataStore menjadi lebih efisien, cepat, dan mudah dipelihara.
- Meningkatkan produktivitas *developer* dengan mempercepat proses pembuatan skema pada Protocol Buffers melalui proses perubahan yang diperoleh dari struktur data pada Kotlin Data Class, menghemat waktu *developer* sehingga dapat berfokus pada logika bisnis inti, dan meningkatkan efisiensi workflow.
- Berkontribusi pada pengembangan bahasa pemrograman kotlin, aplikasi android, dan ekosistem keduanya.
- Menyediakan studi kasus, membuka peluang penelitian optimasi konversi Protocol Buffers Message dan pengembangan *tools*.

BAB II KAJIAN PUSTAKA

2.1 Kotlin

Kotlin adalah bahasa pemrograman *cross-platform* tingkat tinggi yang *open source*, *statically typed*, dan juga didesain sebagai *general-purpose programming language* dengan inferensi tipe. Kotlin dapat digunakan dengan berbagai macam paradigma pemrograman seperti pemrograman berorientasi objek, fungsional, dan *imperative*. Kotlin termasuk *compiled programming language* sehingga memerlukan proses kompilasi sebelum dapat dijalankan. Kotlin dapat dikompilasikan menjadi *bytecode* Java sehingga dapat berjalan pada *Java Virtual Machine* (JVM). Dikarenakan hal tersebut, interoperabilitas Kotlin terhadap Java sangatlah tinggi, sehingga kode java dapat dengan mudah digunakan dan dijalankan bersamaan dengan kode kotlin di dalam satu proyek yang sama [17].

Selain JVM, kotlin juga menargetkan beberapa platform lain seperti Web ketika kode kotlin dikompilasikan menjadi WebAssembly (Wasm) dan juga Android dengan tingkat adopsi oleh professional Android developers mencapai lebih dari 50%. Selain itu, kotlin juga dapat dikompilasikan menjadi native binaries sehingga dapat dijalankan tanpa menggunakan *virtual machine* sehingga kode kotlin dapat dijalankan untuk platform di mana mesin virtual tidak diinginkan atau dimungkinkan, seperti pada iOS dan *embedded devices*. Terakhir, kode Kotlin juga dapat ditransplasikan menjadi kode JavaScript [17].

Kotlin secara resmi mengumumkan rilis 1.0 pertamanya pada bulan Februari 2016. Setahun kemudian, Google mengumumkan Kotlin sebagai bahasa resmi untuk pengembangan aplikasi Android tepatnya pada Google I/O 2017 [4]. Setelah dua tahun berselang, tepatnya saat Google I/O 2019, Google kemudian mengumumkan bahwa pengembangan aplikasi mobile android akan menjadi *kotlin-first*, hal ini berarti Kotlin akan menjadi pilihan pertama Google dalam mengembangkan alat dan konten pengembangan aplikasi android [5]. Hingga sekarang telah banyak API dan Library pada Android yang menggunakan kotlin

beserta fitur-fiturnya, seperti pada Android KTX [6]. Lebih dari 95% dari seribu aplikasi Android teratas telah menggunakan Kotlin [18].

2.2 Kotlin Data Class

Kotlin menyediakan sebuah class khusus untuk mempertahankan data, class khusus ini bernama data class. Data Class sendiri ditandai dengan adanya keyword “data” sebelum keyword “class”. Berbeda dengan kelas biasa, Data Class pada Kotlin tidak dapat dibuat *abstract*, *open*, *sealed*, maupun *inner*. Perbedaan lainnya adalah property pada Kotlin Data Class dideklarasikan lewat constructor. Untuk setiap property, Kotlin Data Class akan menambahkan beberapa member functions, seperti fungsi copy, equals, componentN, dan toString. Hal tersebut memungkinkan Data Class untuk mengurangi potensi kode boilerplate dituliskan [11]. Berikut adalah contoh sederhana penulisan Kotlin Data Class bernama “Example”:

```
data class Example (
    val id: Int,
    var content: String = "",
    val mutable: Boolean = false
)
```

2.3 Protocol Buffers

Protocol Buffers atau yang juga dapat disingkat dengan protobuf, merupakan mekanisme yang dapat diekstensi untuk membuat serialisasi data terstruktur yang *language-neutral* dan *platform-neutral*. Protobuf sendiri dikembangkan oleh Google untuk kebutuhan internal mereka sejak tahun 2001, kemudian Google mempublikasikan protobuf melalui versi keduanya yaitu proto2. Pada versi tersebut kode-kode dalam Protocol Buffers telah dibersihkan dan tidak memiliki ketergantungan apa pun pada library Google yang belum open-source. Hingga saat ini Protocol Buffers mempunyai satu edisi yaitu edisi 2023 dan dua versi yaitu Proto2 dan Proto3.

Protobuf sering digunakan untuk mendefinisikan protokol komunikasi (bersama dengan gRPC) dan untuk penyimpanan data. Protobuf sendiri telah digunakan secara luas di segala jenis layanan dan proyek-proyek milik Google. Protocol Buffers sendiri terdiri dari kombinasi bahasa definisi (dibuat dalam file .proto), kode yang dibuat oleh kompiler proto untuk berinteraksi dengan data, pustaka runtime yang *language-specific*, format serialisasi untuk data yang ditulis ke file (atau dikirim melalui koneksi jaringan), dan data yang terserialisasi. File proto yang digunakan sebagai definisi memiliki ekstensi file “.proto” dan dinamakan menggunakan format *lower_snake_case*, sehingga contoh penamaan file tersebut adalah seperti berikut: `example_protobuf.proto` [10].

2.4 Protocol Buffers Message

Struktur data pada Protocol Buffers didefinisikan di file-file dengan ekstensi .proto yang dibuat oleh engineer. Pada file tersebut service-service dan message-message didefinisikan. Message inilah yang dijadikan schema untuk memperoleh kode dan runtime library yang lebih *language-specific* sehingga dapat digunakan pada berbagai bahasa dan *framework* pemrograman. Sebuah Protocol Buffer Message adalah serangkaian pasangan key-value. Versi biner dari message akan menggunakan field's number tersebut sebagai key, nama dan tipe yang terdeklarasikan pada setiap field hanya dapat ditentukan pada saat decoding dengan merujuk pada message type's definition (yaitu file .proto). ketika sebuah message di encode, setiap pasangan key-value diubah menjadi sebuah catatan yang terdiri dari field number, wire type, dan payload [10].

Nama message pada protobuf ditulis menggunakan format PascalCase atau UpperCamelCase. Di dalam message dapat terdapat field-field dan atau message-message lainnya. Field pada protobuf ditulis menggunakan format *lower_snake_case*, setiap field memiliki field type dan juga field number, dengan field number pada setiap field didalam message yang sama haruslah bersifat unik. Field type dapat berupa scalar type seperti integer dan string, juga dapat berupa enumeration atau juga dapat menjadikan Message lainnya sebagai field type [10].

Berikut adalah contoh sederhana penulisan sebuah message bernama “Example” yang terdapat pada .proto file :

```
message Example {  
    int32 id = 1;  
    optional string content = 2;  
    bool mutable = 3;  
}
```

2.5 Plugin

Plugin adalah sebuah program komputer yang dibuat untuk menambahkan suatu fungsionalitas tertentu kedalam program atau *software* utama. Istilah-istilah lain yang sering dipakai sebagai padanan kata plugin adalah *add-on* dan *extension*. Keuntungan dari menggunakan plugin adalah ketidakharusan untuk memodifikasi program utama ketika ingin menambahkan fitur ataupun fungsionalitas baru yang biasanya bersifat opsional. Contoh *plugin* yang saat ini telah banyak digunakan adalah *plugin* Google Translate yang dipublish melalui Chrome Web Store dan dapat digunakan pada *web browser* berbasis *chromium*. Tujuan tugas akhir ini adalah untuk merancang dan membangun plugin yang dapat dipasang dan dijalankan pada *IDE* IntelliJ IDEA dan Android Studio [19].

BAB III

METODE PENGEMBANGAN SISTEM

3.1 Alat dan Bahan

Dalam rancang bangun “Alat Pengubah Kotlin Data Class ke Protocol Buffers Message”, berbagai alat dan bahan digunakan untuk memastikan perancangan dan pembangunan dilakukan dengan optimal. Alat dan bahan tersebut dibagi menjadi dua jenis yaitu, perangkat keras (*hardware*) dan perangkat lunak (*software*) dengan rincian sebagai berikut:

1. Perangkat Keras (*hardware*)

- a. Laptop Lenovo IdeaPad 3 14IIL05

Processor Intel(R) Core(TM) i5-1035G1 CPU @1.00GHz
Installed RAM 4 + 4 (8,00) GB DDR4
System type 64-bit operating system, x64-based processor
Storage SSD 512 GB
Discrete GPU NVIDIA GeForce MX330 2GB GDDR5
Screen Resolution 1920 x 1080@60Hz

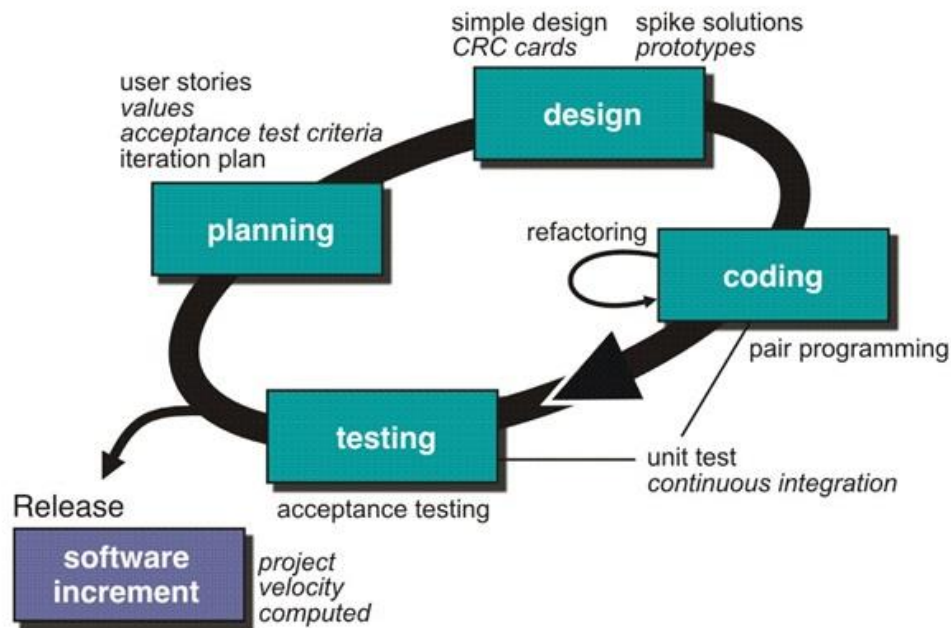
2. Spesifikasi Perangkat Lunak

- a. Sistem Operasi Windows 10 Home Single Language
 - b. IntelliJ IDEA Community Edition 2024
 - c. Android Studio Jellyfish 2023
 - d. Whimsical
 - e. PlantUML

3.2 Metode Pengembangan

Rancang bangun alat pengubah Kotlin Data Class ke Protocol Buffers Message menggunakan **Metode Agile Extreme Programming (XP)** sebagai metode pengembangan. Metode ini membawa praktik pemrograman tradisional ke tingkat ekstrem, dimana langkah-langkah pendukung akan diminimalkan yang berarti kebutuhan untuk membuat dokumentasi dan *project requirements* tidak terlalu diperlukan. Metode ini cocok untuk diterapkan pada pengembangan yang memiliki tim-tim kecil dan sangat berorientasi pada tim. Keuntungan yang

didapatkan saat menggunakan metode ini adalah dapat memudahkan pekerjaan tim pengembang untuk menghasilkan perangkat lunak berkualitas tinggi [20]. Metode ini dapat dibagi menjadi empat tahapan utama yang dapat dilihat pada gambar berikut:



Gambar 1 Tahapan Metode Extreme Programming

Seperti yang dapat dilihat pada gambar 1, terdapat empat tahapan utama yang dapat dilakukan pada metode extreme programming (XP) yaitu:

1. *Planning* (Perencanaan)

Tahapan ini merupakan tahapan paling awal dari proses pengembangan sistem, yang didalamnya dilakukan sejumlah kegiatan perencanaan yaitu identifikasi masalah, analisis kebutuhan, hingga penentuan jadwal pelaksanaan pengembangan sistem.

2. *Design* (Perancangan)

Tahapan selanjutnya adalah perancangan atau desain, pada tahapan ini dilakukan kegiatan perancangan dan pemodelan, seperti pemodelan sistem, pemodelan arsitektur, hingga perancangan antarmuka pengguna jika ada.

3. *Coding* (Implementasi)

Tahapan ketiga ini merupakan tahapan yang berfokus pada implementasi dari desain atau model yang sudah dibuat ke dalam bentuk *code* sehingga desain atau model yang dibuat dapat terealisasi dan digunakan oleh *user*.

4. *Testing* (Pengujian)

Tahapan terakhir sebelum *software* dirilis adalah tahapan testing atau pengujian. Tahapan ini dapat dilakukan setelah tahapan implementasi selesai. Pengujian dilakukan untuk mengetahui apakah *software* sudah sesuai dengan kebutuhan yang telah didapatkan pada tahapan *planning*, juga untuk mengetahui kesalahan dan *bugs* apa saja yang dapat terjadi pada saat aplikasi sedang berjalan agar nantinya kesalahan tersebut dapat diperbaiki sebelum *software* digunakan oleh pengguna [21].

BAB IV

PERANCANGAN DAN IMPLEMENTASI SISTEM

4.1 Perencanaan (*Planning*)

Alat pengubah Kotlin Data Class ke Protocol Buffers Message dirancang agar dapat digunakan oleh pengembang aplikasi android maupun pengembang yang menggunakan kotlin sebagai bahasa pengembangan sistem yang juga akan memerlukan protocol buffers sebagai salah satu solusi serialisasi struktur data. Kedua pengembang tersebut dapat menggunakan Android Studio atau IntelliJ IDE sebagai IDE untuk membantu pengembangan aplikasinya, sehingga rencananya alat ini akan dibuat dalam bentuk *plugin* yang dapat dipasang dan dijalankan pada kedua IDE tersebut agar dapat dengan mudah digunakan oleh pengembang tersebut.

Alat ini dapat digunakan dengan cara memilih file kotlin yang berisikan Data Class yang ingin dikonversi menjadi Protocol Buffers Message, kemudian pengembang akan mengisi informasi-informasi dan konfigurasi-konfigurasi yang diperlukan untuk melakukan konversi, terakhir pengembang harus menekan tombol *Convert* untuk menjalankan proses konversi. Apabila terjadi *error* maka plugin akan memunculkan pesan error dan membatalkan proses konversi, sebaliknya jika tidak terjadi *error* maka proses konversi akan berlangsung dan file Protocol Buffers yang berisi *Messages* akan dibuat. Pengembangan sistem ini berjalan dengan menggunakan metode agile extreme programming sehingga berfokus pada tahap *coding*, pengembangan sistem dapat dibagi menjadi tiga iterasi dengan kegiatan utama pada tiap iterasi adalah sebagai berikut:

1. Merancang dan membangun *setup* dasar untuk mengembangkan *plugin* yang meliputi fitur dasar pada sistem sehingga menjadi *plugin* yang dapat dipasang dan dijalankan pada Android Studio dan IntelliJ IDEA, menerapkan dependency injection menggunakan *library* Koin, kemudian juga melakukan pendataan *keyword* dan *style guide* yang terdapat pada Protocol Buffers kemudian menerapkannya pada sistem.
2. Merancang dan membangun antarmuka pengguna untuk *plugin*, menerapkan model-model yang dibutuhkan untuk proses bisnis utama dan

konfigurasi pengguna, juga merancang dan membangun beberapa fitur seperti fitur untuk mengambil informasi terkait lingkungan project yang sedang dibuka dan penampil pesan terkait hasil dari proses yang telah dijalankan plugin.

3. Merancang dan membangun fitur *parsing* dari Kotlin Data Class menjadi Protocol Buffer Message kemudian menulis dan menyimpan hasilnya ke dalam sebuah *file* baru dengan ekstensi “.proto”.

4.1.1 Analisis Kebutuhan Fungsional

Kebutuhan fungsional dari sistem yang dirancang dan dibangun pada tugas akhir ini adalah kebutuhan-kebutuhan yang diperlukan untuk memenuhi kebutuhan pengguna, antara lain:

1. Memilih file Kotlin

Fitur ini memungkinkan pengguna untuk memilih *file* Kotlin yang berisikan Kotlin Data Class yang ingin diubah menjadi *file* yang berisi Protocol Buffers Message.

2. Mengisi informasi dan memilih konfigurasi

Fitur ini memungkinkan pengguna untuk mengisi informasi-informasi dan memilih konfigurasi-konfigurasi yang sesuai dengan preferensinya terkait hasil yang ingin didapatkan sehingga *plugin* dapat menggunakannya untuk melakukan konversi, seperti fitur yang memungkinkan pengguna untuk menentukan nama file yang digunakan untuk menyimpan hasil konversi.

3. Memperoleh hasil

Fitur ini memungkinkan *plugin* memberikan hasil terkait proses konversi yang dapat diakses oleh pengguna, baik itu ketika proses konversi berhasil yaitu ketika *file* yang berisi Protocol Buffers Message berhasil dibuat, maupun ketika proses konversi gagal sehingga memberikan hasil berupa pesan dari kegagalan tersebut.

4.1.2 Analisis Kebutuhan Non-Fungsional

Kebutuhan nonfungsional dari sistem yang dirancang dan dibangun pada tugas akhir ini adalah kebutuhan yang mencakup perilaku dari sistem, antara lain:

1. Sistem melakukan konversi dengan akurat

Plugin diharuskan dapat melakukan konversi Kotlin Data Class menjadi Protocol Buffers Message dengan cakupan minimal sesuai dengan tipe data skalar secara akurat.

2. Sistem dilengkapi dengan panduan pengguna

Plugin diharuskan memiliki panduan pengguna yang dapat diakses oleh pengguna yang menjelaskan tentang panduan pemasangan dan penggunaan *plugin*.

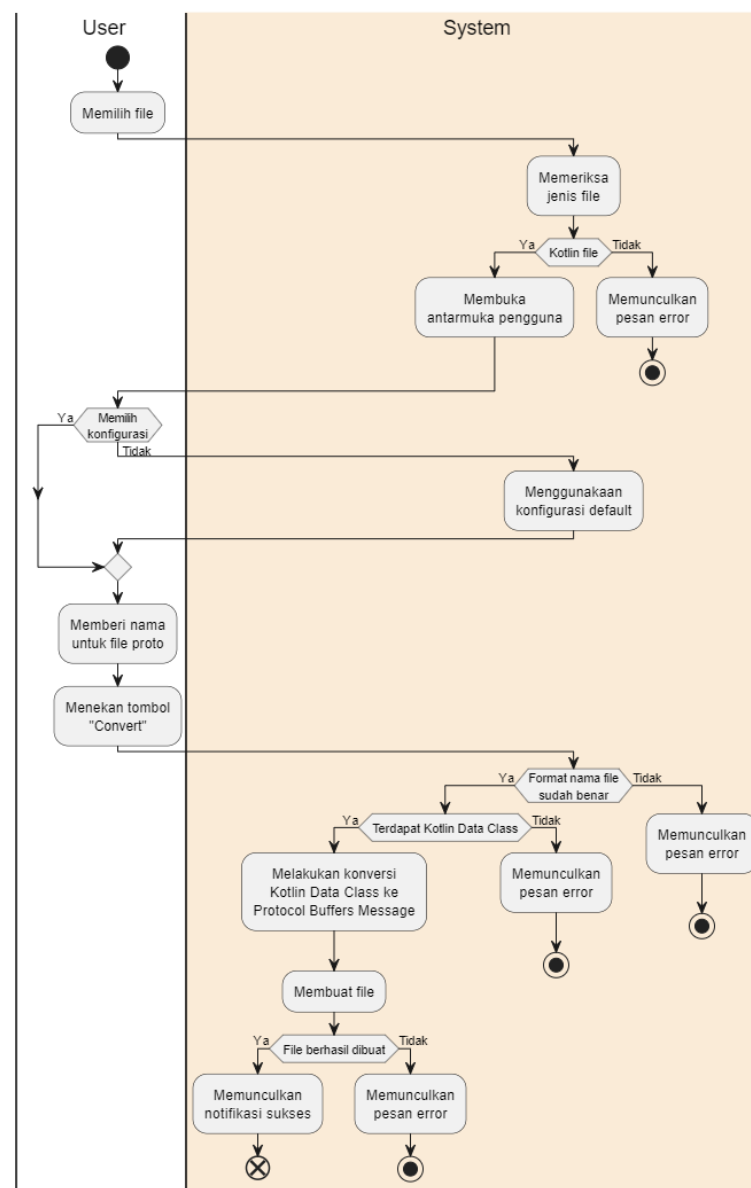
4.2 Perancangan (*Design*)

Perancangan sistem dibuat agar dapat memudahkan pengembangan sistem yang sesuai dengan kebutuhan-kebutuhan yang telah didapatkan pada tahapan perencanaan. Salah satu rancangan yang dapat dibuat adalah dengan menggunakan model. Model memainkan peran serupa dalam pengembangan perangkat lunak seperti halnya *blueprints* dan rancangan lainnya (peta lokasi, ketinggian, model fisik) dalam pembangunan gedung pencakar langit. Penggunaan model membantu meningkatkan penjaminan kelengkapan fungsionalitas bisnis dan kebutuhan *end-user* terpenuhi sebelum implementasi dalam kode membuat perubahan menjadi sulit dan mahal untuk dilakukan [22].

Pada tahapan perancangan, Unified Modelling Languages (UML) akan digunakan untuk memodelkan sistem, dengan Activity Diagram sebagai diagram yang dipakai untuk memodelkan perilaku sistem. UML dipilih dikarenakan kecocokannya untuk sistem yang dibangun menggunakan bahasa dan lingkungan yang berorientasi objek [22]. Selain itu, wireframe juga akan digunakan sebagai kerangka antarmuka sistem sehingga dapat membantu untuk tetap fokus pada fungsionalitas dasar dan tetap pada arah pengembangan yang benar [23].

4.2.1 Activity Diagram

Activity diagram merupakan salah satu diagram yang dikategorikan sebagai behavioral diagram, yang digunakan untuk memodelkan transisi dari satu aktivitas ke aktivitas lainnya pada sebuah sistem sehingga dapat menunjukkan bagaimana aktivitas pada sistem saling berkoordinasi [24]. Pada gambar 2 menunjukkan aktivitas-aktivitas yang dapat terjadi dari awal hingga akhir ketika *plugin* dijalankan beserta pelaku aktivitasnya.



Gambar 2 Activity Diagram Plugin

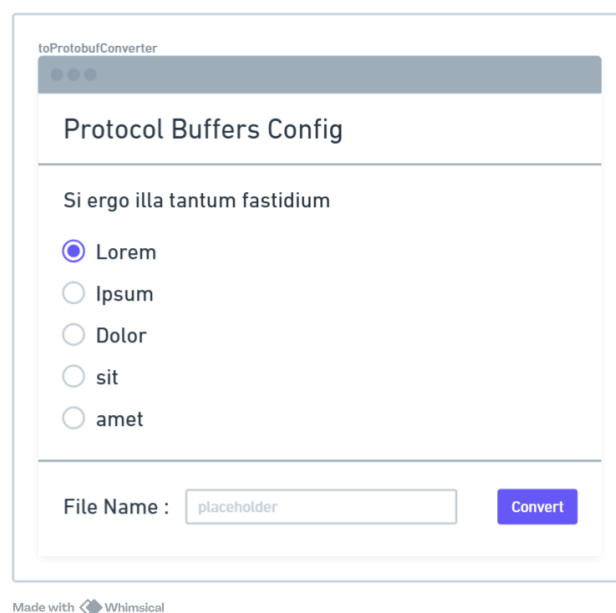
Gambar 2 adalah Activity Diagram pada plugin yang dibuat menggunakan plantUML yang menjelaskan bahwa aktivitas pada *plugin* dapat dimulai ketika pengguna memilih file yang ingin dikonversi. Kemudian sistem akan memeriksa jenis file yang dipilih sebelumnya, apakah file tersebut adalah file kotlin atau bukan, jika file tersebut adalah file kotlin maka akan membuka antarmuka pengguna dari plugin, sebaliknya jika file tersebut bukan file kotlin maka sistem pada plugin akan memunculkan pesan error. Pada antarmuka pengguna yang ditampilkan plugin, pengguna dapat memilih konfigurasi yang tersedia. Jika pengguna tidak memilih konfigurasi yang tersedia maka konfigurasi yang digunakan akan otomatis dipilih oleh sistem menggunakan konfigurasi bawaan. Selain itu, pengguna juga dapat memberi nama file hasil dari konversi. Setelah selesai mengisi konfigurasi dan informasi yang diperlukan oleh sistem, pengguna dapat menekan tombol “Convert” untuk memulai proses konversi.

Proses konversi dimulai dengan aktivitas pengecekan oleh sistem tentang format nama file yang sebelumnya telah diisi oleh *user*, apabila format nama file salah maka sistem akan memunculkan pesan error. Sistem akan melanjutkan dengan pengecekan apakah pada file kotlin tersebut terdapat data class didalamnya apabila format nama file sudah benar. Jika tidak terdapat data class sama sekali maka sistem akan memunculkan pesan error, sebaliknya proses konversi Kotlin Data Class menjadi Protocol Buffers Message akan dilakukan apabila terdapat data class pada file kotlin tersebut. Setelah melakukan proses konversi, sistem selanjutnya akan membuat file dengan ekstensi “.proto” yang berisikan hasil dari proses konversi. Apabila sistem gagal untuk membuat file tersebut, maka akan menampilkan pesan kegagalan. Activity diagram pada plugin berakhir setelah menampilkan notifikasi sukses tentang keberhasilan membuat file yang merupakan aktivitas terakhir yang dapat dilakukan dengan *plugin*.

4.2.2 Wireframe

Wireframe adalah *blueprints* dasar yang membantu tim menyelaraskan *requirements* dan menjaga pembahasan desain pengalaman pengguna agar tetap

fokus dan konstruktif. Sebagian besar desain wireframe terbagi ke dalam tiga tingkat *fidelity*, yaitu *low-fidelity*, *mid-fidelity*, dan *high-fidelity* [23]. Desain wireframe yang dipakai dalam perancangan plugin ini adalah wireframe dengan tingkat *low-fidelity* yang mengandalkan teks lorem ipsum dengan beberapa komponen sudah mempunyai konten sehingga setingkat dengan *mid-fidelity*. Berikut adalah desain wireframe yang digunakan sebagai kerangka antarmuka sistem:



Gambar 3 Desain Wireframe Plugin

Gambar 3 adalah wireframe yang dibuat menggunakan fitur wireframe pada platform Whimsical. Pada gambar 3 tersebut dapat dilihat desain wireframe plugin yang nantinya akan digunakan sebagai kerangka dasar untuk tampilan antarmuka pengguna pada plugin. Pada desain tersebut terdapat label-label seperti tulisan “Protocol Buffers Config”, “File Name :”, dan “Si ergo illa tantum fastidium” sebagai padanan dari lorem ipsum. Label tersebut digunakan sebagai informasi tambahan untuk komponen yang dilabeli. Selain itu terdapat grup yang berisi lima radio button yang nantinya akan digunakan ketika pengguna ingin memilih konfigurasi yang tersedia. Pada panel bawah terdapat *text box* untuk pengguna dapat mengisi nama file hasil konversi, juga terdapat *button* bertuliskan “Convert” untuk memulai proses konversi.

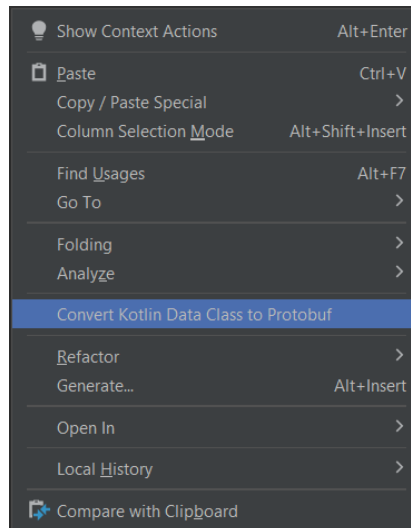
4.3 Implementasi (*Coding*)

Tahap implementasi adalah tahapan dimana desain atau model yang sudah dirancang pada tahap sebelumnya diimplementasikan ke dalam bentuk *code* sehingga dapat direalisasikan dan kemudian digunakan oleh *user*. Oleh karena itu, fokus utama pada tahapan ini adalah untuk membangun *plugin* yang dapat dijalankan pada IntelliJ IDEA dan Android Studio sesuai dengan hasil pada tahapan perancangan agar dapat memenuhi kebutuhan yang sudah direncanakan. Pada tahapan ini digunakan IntelliJ IDEA sebagai IDE untuk membangun plugin dan perangkat keras berupa laptop digunakan untuk membangun dan menjalankan plugin dengan detail mengenai keduanya telah dijelaskan pada bagian alat dan bahan di BAB III. Tahapan implementasi ini terbagi menjadi tiga tahapan iterasi pengembangan yang meliputi perancangan dan pembangunan sesuai dengan apa yang telah disebutkan pada bagian perencanaan di BAB IV sebelumnya, dengan rinciannya akan dijelaskan sebagai berikut:

4.3.1 Iterasi Pengembangan Pertama

Langkah pertama yang akan dilakukan pada iterasi pertama adalah merancang dan membangun setup dasar untuk mengembangkan plugin yang meliputi fitur dasar pada sistem sehingga menjadi plugin yang dapat dipasang dan dijalankan pada Android Studio dan IntelliJ IDEA. Setup dasar ini meliputi mendaftarkan dependency yang akan digunakan untuk mengembangkan *plugin*, yaitu library koin yang digunakan sebagai dependency injection dan library mockk yang digunakan untuk membantu pengujian *plugin*. Selain itu, dilakukan juga setup pada plugin.xml untuk mendaftarkan informasi-informasi terkait *plugin* seperti id, nama yaitu toProtobufConverter, *plugin dependencies*, *services*, dan *action* untuk membuka *plugin*. Action ini diambil dari class GenerateProtobufAction pada file GenerateProtobufAction.kt pada package action. Action tersebut akan ditampilkan dengan tulisan “Convert Kotlin Data Class to Protobuf” dan berada pada grup

EditorPopupMenu3, sehingga aksi untuk membuka plugin tersebut adalah melalui editor menu ketika sedang membuka sebuah file, seperti pada gambar 4.



Gambar 4 Editor Popup Menu

Selanjutnya akan diterapkan dependency injection menggunakan koin pada package bernama “di” yang didalamnya terdapat file AppModule.kt yang isinya akan bertambah seiring bertambahnya *class* yang ingin diinjeksi dan juga file PluginApplication.kt untuk menginisiasikan koin dan menginstansiasikan controller. Class GenerateProtobufActionController pada file GenerateProtobufActionController.kt di package controller digunakan sebagai *blueprint* untuk object yang diinstansiasikan pada file PluginApplication.kt. Controller ini sangatlah penting untuk menjalankan berbagai aktivitas ketika action untuk membuka plugin diterima, seperti memanggil fungsi untuk menampilkan pesan error dan antarmuka pengguna dan juga memperoleh informasi terkait lingkungan project, yang seluruhnya akan dikembangkan pada iterasi kedua. Selain itu terdapat listener yang akan menjalankan aktivitas-aktivitas terkait konversi ketika tombol “Convert” ditekan. Listener ini terdapat pada package listeners yang berisikan file GuiFormEventListener.kt dan GenerateActionListener.kt, dimana terdapat interface GuiFormEventListener sebagai kontrak ketika diimplementasi dan class GenerateActionListener untuk menghasilkan listener.

Setelah selesai melakukan setup dasar, selanjutnya akan dilakukan pendataan keyword dan style guide yang terdapat pada Protocol Buffers sehingga dapat

menerapkannya pada sistem. Keyword pada protocol buffers ditemukan pada dokumentasi tentang spesifikasi Protocol Buffers, dengan jumlah sebanyak empat puluh (40) kata kunci yang dapat dilihat pada gambar 5. Keyword tersebut kemudian dimasukkan kedalam file Keyword.kt yang terdapat pada package converter.template sebagai variabel konstanta pada object Keyword. Selain itu terdapat

```
syntax = "syntax" .    oneof = "oneof" .    int32 = "int32" .
edition = "edition" .  map = "map" .    int64 = "int64" .
import = "import" .    extensions = "extensions" .  uint32 = "uint32" .
weak = "weak" .    reserved = "reserved" .  uint64 = "uint64" .
public = "public" .    rpc = "rpc" .    sint32 = "sint32" .
package = "package" .  stream = "stream" .  sint64 = "sint64" .
option = "option" .    returns = "returns" .  fixed32 = "fixed32" .
inf = "inf" .    to = "to" .    fixed64 = "fixed64" .
nan = "nan" .    max = "max" .    sfixed32 = "sfixed32" .
message = "message" .  repeated = "repeated" .  sfixed64 = "sfixed64" .
enum = "enum" .    optional = "optional" .  bool = "bool" .
service = "service" .  required = "required" .  float = "float" .
extend = "extend" .    string = "string" .  double = "double" .
group = "group" .    bytes = "bytes" .
```

Gambar 5 Protocol Buffers Keywords

Pendataan mengenai style guide yang dapat ditemukan pada dokumentasi resmi Protocol Buffers dilakukan untuk mengetahui mengenai panduan tentang gaya dan struktur penulisan serta pembuatan file Protocol Buffers. Disana dapat diketahui bahwa nama file berekstensi “.proto” sebaiknya ditulis dengan menggunakan format lower case snake, sehingga dapat diambil contoh nama yang baik adalah sebagai berikut: “lower_snake_case.proto”. Dari panduan untuk nama file tersebut dapat diimplementasikan aktivitas pada plugin untuk memvalidasi nama file. Aktivitas tersebut diimplementasikan pada fungsi validateFileName pada MessageConversionHelper pada file MessageConversionHelper.kt yang terdapat pada package converter.utils. Fungsi tersebut mencocokkan nama file dengan pattern regex “^[a-z0-9_]*\$” sehingga apabila nama diisi dengan karakter diluar huruf kecil dan angka maka sistem akan mengeluarkan error WrongFileNameException().

Pada style guide juga dapat diketahui struktur file protocol buffers berurutan sebagai berikut: License Header (jika ada), File overview, Syntax, Package, Imports (secara berurutan), File options, dan yang lainnya seperti messages. Struktur file tersebut diimplementasikan melalui interface ProtobufTemplateHelper yang terdapat pada file kotlin dengan nama yang sama di package converter.template

sebagai bantuan untuk menyusun file protocol buffers. Kemudian pada setiap field yang ada pada message dapat mempunyai field labels dan field options. Selain itu field juga diharuskan memiliki tipe, nama, dan juga field number yang unik diantara semua field yang terdapat pada message tersebut. Semua ini diimplementasikan dalam bentuk variabel konstanta pada package converter.template yang masing-masing dapat dilihat pada file VersionTemplate.kt untuk syntax, PackageTemplate.kt untuk package, ImportTemplate untuk imports, FileOptionTemplate.kt untuk file options, dan FieldOptionTemplate.kt untuk field options.

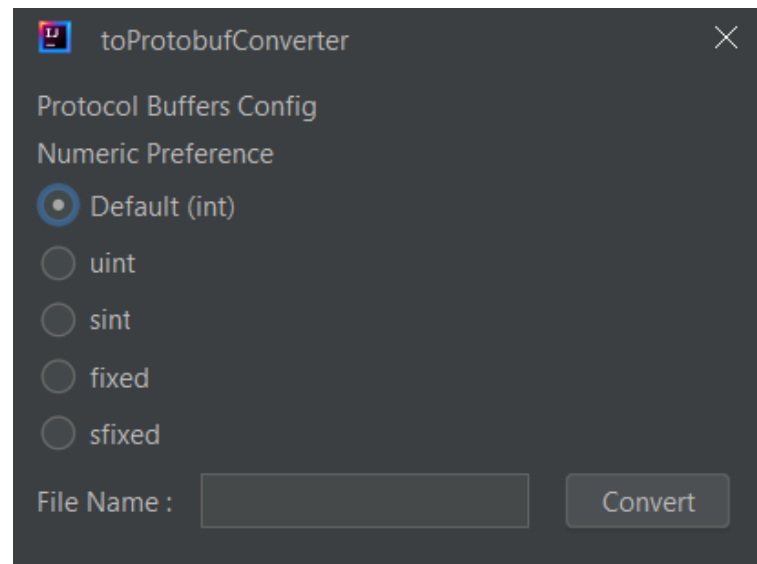
Lalu terdapat sebuah keharusan untuk mempunyai indentasi sebanyak dua *spaces* dan mengakhiri dengan semicolon untuk setiap field, hal ini diterapkan melalui variabel konstanta pada file CommonTemplate.kt yang ada pada package yang sama dengan template sebelumnya. Dapat diketahui juga dari keyword yang ada, terdapat scalar types yang dapat digunakan sebagai tipe data pada field seperti pada gambar 6. Pada scalar types terdapat beberapa macam tipe data integer sehingga hal ini dapat menjadi konfigurasi yang dapat dijadikan fitur pada plugin. Selain scalar types terdapat juga well-known types and common types yaitu tipe field tambahan yang dibuat oleh Google untuk Google APIs.

Scalar type names	Description
<code>int32</code> , <code>sint32</code> , <code>sfixed32</code>	32-bit signed integers, in the range $[-2^{31}, 2^{31})$
<code>int64</code> , <code>sint64</code> , <code>sfixed64</code>	64-bit signed integers, in the range $[-2^{63}, 2^{63})$
<code>uint32</code> , <code>fixed32</code>	32-bit unsigned integers, in the range $[0, 2^{32})$
<code>uint64</code> , <code>fixed64</code>	64-bit unsigned integers, in the range $[0, 2^{64})$
<code>float</code>	32-bit (single precision) IEEE 754 floating point value
<code>double</code>	64-bit (double precision) IEEE 754 floating point value
<code>bool</code>	A boolean value (true or false)
<code>string</code>	A sequence of UTF-8-encoded characters, of length zero or greater.
<code>bytes</code>	A sequence of bytes, of length zero or greater.

Gambar 6 Scalar Type

4.3.2 Iterasi Pengembangan Kedua

Langkah pertama pada iterasi pengembangan kedua adalah untuk merancang dan membangun antarmuka pengguna untuk plugin. Antarmuka pengguna ini dirancang dan dibangun menggunakan Java Swing UI dengan membuat GUI Form bernama KotlinConverterForm yang terletak pada package presentation.form. Fitur untuk konfigurasi pada plugin yang direncanakan di iterasi pengembangan pertama adalah berupa pemilihan jenis tipe data integer yang diinginkan oleh *user*. Sehingga tampilan antarmuka pengguna pada plugin menjadi seperti pada gambar 7.



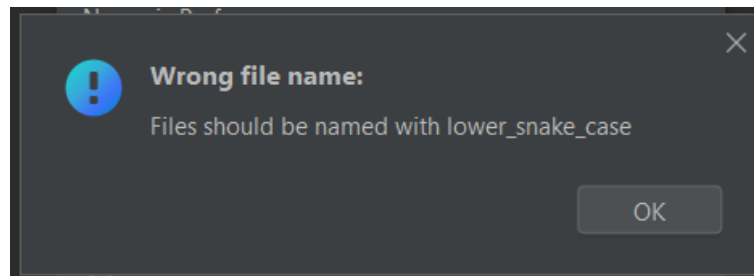
Gambar 7 Graphical User Interface Plugin Form

Setelah membuat tampilan untuk antarmuka pengguna, selanjutnya akan diterapkan model-model yang dibutuhkan untuk proses bisnis utama dan konfigurasi pengguna. Model-model ini terdapat pada package core.models yang berisikan dua file yaitu ConversionModel.kt dan ControlsModel.kt. Pada ConversionModel.kt terdapat data class ConversionModel sebagai model untuk proses konversi dan data class ProjectModel sebagai model untuk informasi terkait project yang sedang dibuka.

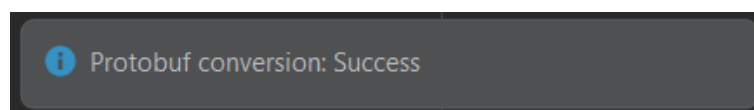
Terakhir untuk melengkapi antarmuka pengguna dan model, presentation layer akan dibuat sehingga antarmuka pengguna dan model dapat berinteraksi. Presentation layer ini terdapat pada package presentation dan berisikan 4 file, yaitu ViewModelMapper.kt yang digunakan untuk mapping nilai yang didapat dari

element pada view kedalam `ConversionModel`, lalu ada `PropertiesFactory.kt` untuk membuat `ControlsModel` beserta nilainya, dan ada `ViewStateManager.kt` yang mengatur states yang terdapat pada view di `ConverterForm`, dan yang terakhir adalah `ConverterViewFactory.kt` untuk memilih dan membuat view yang akan ditampilkan sistem beserta mengikatnya dengan states.

Untuk melakukan operasi CRUD pada states, dibuat sebuah service yang bernama `ViewStateService`, service ini diletakkan pada file `ViewStateService.kt` yang terdapat di package `services`. Terdapat satu package yang bernama `delegates`, didalamnya terdapat dua file yaitu `EnvironmentDelegate.kt` yang digunakan untuk mengambil informasi terkait lingkungan project yang sedang dibuka kemudian memasukkan nilainya kedalam `ProjectModel` kemudian menjadikannya nilai kembalian dan `MessageDelegate.kt` yang digunakan untuk menampilkan dialog yang berisi pesan error ketika plugin gagal dalam menjalankan sebuah aktivitas seperti pada Gambar 8 atau menampilkan notifikasi sukses ketika aktivitas konversi berhasil seperti pada Gambar 9.



Gambar 8 Error Dialog



Gambar 9 Success Notification

4.3.3 Iterasi Pengembangan Ketiga

Iterasi ketiga yang merupakan iterasi pengembangan terakhir akan berfokus pada perancangan dan pembangunan fitur parsing dari Kotlin Data Class menjadi Protocol Buffer Message kemudian menulis dan menyimpan hasilnya ke dalam sebuah file baru dengan ekstensi “.proto”. Pertama akan dibuat sebuah processor

pada package `converter.processor` untuk mengimplementasikan interface `ProtobufTemplateHelper`. Oleh karena itu, file `KotlinToProtobufProcessor.kt` dibuat sehingga dapat merealisasikan template yang dijadikan bantuan untuk membuat file protocol buffer dari kotlin data class menjadi terealisasi. Pada file tersebut terdapat pengecekan terkait isi dari file apakah terdapat kotlin data class atau tidak, sehingga dapat mewakili aktivitas yang sama yang telah dirancang sebelumnya pada `ActivityDiagram`.

Processor tersebut juga memanggil parser untuk melakukan parsing sehingga field pada Kotlin Data Class dapat diuraikan dan diambil tipe datanya kemudian diubah tipe datanya sehingga menjadi relevan untuk digunakan pada field di Protocol Buffers Message. Parser tersebut dibuat pada package `converter.parser` tepatnya pada file `KotlinDataTypeParser.kt`. `KotlinDataTypeParser` mendukung parsing tipe data scalar pada protocol buffers dan beberapa collection yang dimiliki kotlin seperti List, Set, dan Map, hingga tipe data Arrays. Namun, tipe data yang khusus dari kotlin tersebut hanya dapat dilakukan parsing ketika hanya mempunyai satu dimensi. Setelah iterasi pengembangan berakhir, plugin siap untuk diuji coba oleh developer yang menggunakan kotlin dan protocol buffers untuk membantu pengembangan sistem mereka.

4.4 Pengujian (*Testing*)

Tahapan pengujian atau testing dilakukan dengan harapan untuk dapat menemukan kesalahan yang dapat terjadi pada sistem perangkat lunak yang diuji diuji dan bagaimana sistem mengatasinya. Selain itu, pengujian juga dilakukan untuk mengetahui kesesuaian perangkat lunak yang dibuat dengan kebutuhan yang telah ditetapkan sebelumnya. Pengujian pada sistem perangkat lunak dilakukan menggunakan dua metode testing, yaitu metode unit testing yang menguji bagian terkecil dari perangkat lunak yang dapat diverifikasi [25], dan metode black box, yang berfokus pada fungsi perangkat lunak tanpa mempertimbangkan struktur dan implementasinya [26]. Selain pengujian pada sistem perangkat lunak, dilakukan juga pengujian dari output yang dihasilkan sistem perangkat lunak yang berupa file

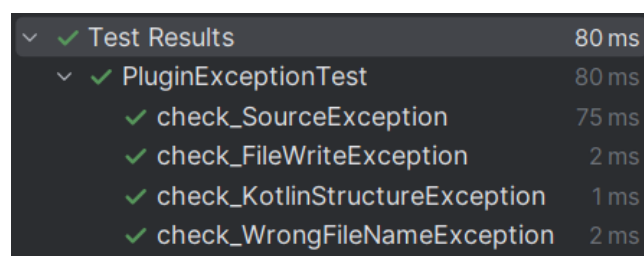
protocol buffers. File tersebut akan disimulasikan untuk dipakai menggunakan proto datastore untuk menyimpan data sederhana dalam pengembangan aplikasi android. Dengan rangkaian pengujian yang dilakukan pada sistem perangkat lunak dapat diketahui kesesuaian antara perangkat lunak dengan tujuan perancangan dan pembangunan serta kebutuhan yang ada.

4.4.1 Unit Testing

Unit testing dilakukan dengan menguji beberapa fungsi, *method*, ataupun *class* dalam sistem perangkat lunak. Beberapa *method* atau *class* yang akan diuji tersebut kemudian dikelompokkan sesuai dengan kelas yang menampungnya sehingga menghasilkan beberapa kelompok pengujian, diantaranya adalah:

1. PluginExceptionTest

PluginExceptionTest menguji beberapa class yang digunakan sebagai blueprint untuk exception-exception yang dapat terjadi pada sistem. Pengujian yang dilakukan terkait dengan kesesuaian nilai dari header dan message masing-masing exception. Beberapa Exception yang diuji yaitu `FileNotFoundException()` yang didapatkan ketika proses penulisan file gagal, `SourceException()` yang didapatkan ketika file yang akan dikonversi bukan merupakan file yang didukung oleh plugin dalam hal ini file kotlin, `KotlinStructureException()` yang didapatkan ketika file kotlin yang akan dikonversi tidak mempunyai kotlin data class didalamnya, dan terakhir `WrongFileNameException()` yang didapatkan ketika nama file yang dimasukkan oleh pengguna kosong atau tidak sesuai dengan format `lower_snake_case`. Hasil pengujian ini dapat dilihat pada gambar 10, dimana sistem pada plugin berhasil lulus dalam pengujian.

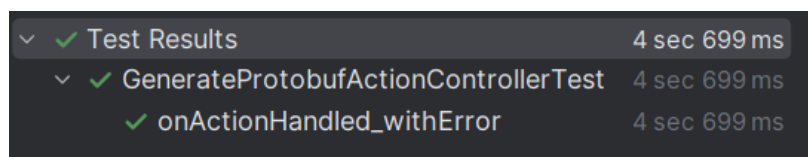


✓ Test Results	80 ms
✓ PluginExceptionTest	80 ms
✓ check_SourceException	75 ms
✓ check_FileWriteException	2 ms
✓ check_KotlinStructureException	1 ms
✓ check_WrongFileNameException	2 ms

Gambar 10 Hasil Pengujian PluginExceptionTest

2. GenerateProtobufActionControllerTest

GenerateProtobufActionControllerTest melakukan pengujian pada *method* `onActionHandled()`, yaitu fungsi pada `GenerateProtobufActionController` yang menangkap aksi pada plugin kemudian meresponnya dengan menjalankan beberapa proses melalui `method` `proceed()` pada *class* yang sama. Pengujian yang dilakukan adalah untuk memastikan ketika terjadi kegagalan pada fungsi tersebut, sistem akan dapat menanganinya dan menampilkan pesan kegagalan kepada user melalui fungsi `onPluginExceptionHandled()` pada *interface* `MessageDelegate`. Hasil pengujian ini dapat dilihat pada gambar 11, dimana sistem pada plugin berhasil lulus dalam pengujian.

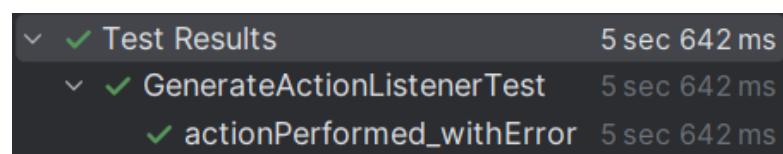


✓ Test Results	4 sec 699 ms
✓ GenerateProtobufActionControllerTest	4 sec 699 ms
✓ onActionHandled_withError	4 sec 699 ms

Gambar 11 Hasil Pengujian `GenerateProtobufActionControllerTest`

3. GenerateActionListenerTest

GenerateActionListenerTest melakukan pengujian pada *method* `actionPerformed()`, yaitu fungsi pada *class* `GenerateActionListenerTest` yang digunakan untuk menjalankan beberapa aktivitas seperti melakukan validasi nama file dari masukan pengguna ketika terjadi *action event* pada plugin. Pengujian yang dilakukan adalah untuk memastikan ketika terjadi kegagalan pada fungsi tersebut, sistem akan dapat menanganinya dan menampilkan pesan kegagalan kepada user melalui fungsi `onPluginExceptionHandled()` pada *interface* `MessageDelegate`. Hasil pengujian ini dapat dilihat pada gambar 12, dimana sistem pada plugin berhasil lulus dalam pengujian ketika terjadi kegagalan saat proses validasi nama file hasil dari masukan pengguna.

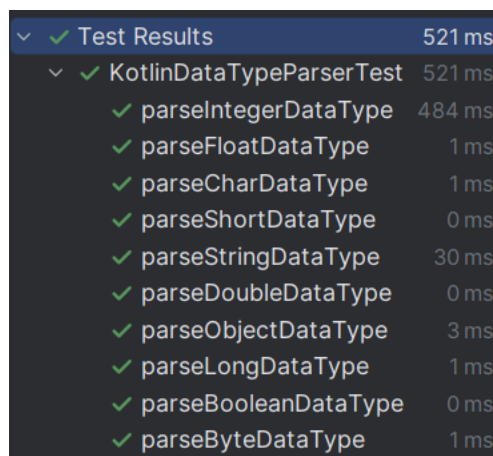


✓ Test Results	5 sec 642 ms
✓ GenerateActionListenerTest	5 sec 642 ms
✓ actionPerformed_withError	5 sec 642 ms

Gambar 12 Hasil Pengujian `GenerateActionListener`

4. KotlinDataTypeParserTest

KotlinDataTypeParserTest melakukan pengujian terhadap *method* `parseDataType()`, yaitu fungsi pada *class* `KotlinDataTypeParser` yang digunakan untuk melakukan parsing terhadap field-field di kotlin data class menjadi field-field di protocol buffers message. Pengujian yang dilakukan adalah untuk memastikan fungsi dapat menangani field dengan yang bertipe data Boolean, Short, Integer, Long, Float, Double, Char, String, Byte, dan Object yang didapat dari instansiasi sebuah *class*. Hasil pengujian ini dapat dilihat pada gambar 13, dimana sistem pada plugin berhasil lulus dalam pengujian.



✓ Test Results	521 ms
✓ KotlinDataTypeParserTest	521 ms
✓ parseIntegerDataType	484 ms
✓ parseFloatDataType	1 ms
✓ parseCharDataType	1 ms
✓ parseShortDataType	0 ms
✓ parseStringDataType	30 ms
✓ parseDoubleDataType	0 ms
✓ parseObjectDataType	3 ms
✓ parseLongDataType	1 ms
✓ parseBooleanDataType	0 ms
✓ parseByteDataType	1 ms

Gambar 13 Hasil Pengujian KotlinDataTypeParser

4.4.2 Black Box Testing

Pengujian black box dilakukan untuk menguji fungsionalitas sistem perangkat lunak. Pengujian dilakukan dengan memberikan beberapa skenario pengujian dari pengguna kepada sistem. Skenario pengujian dikelompokkan menjadi dua, yaitu happy path dimana fitur pada sistem berhasil melakukan skenario pengujian tanpa mendapatkan kegagalan dan unhappy path dimana fitur pada sistem melakukan skenario pengujian dan mendapatkan kegagalan. Skenario pengujian pada sistem beserta dengan hasilnya dapat dilihat pada tabel berikut:

Tabel 1 Pengujian Black Box

Skenario happy path dan hasil uji			
Skenario uji	Yang diharapkan	Yang dihasilkan	Kesimpulan
Memilih file kotlin terlebih dahulu sebelum memanggil plugin	<i>Plugin</i> dapat dijalankan	<i>Plugin</i> berjalan	Berhasil
Melakukan konversi Kotlin Data Class dan menghasilkan <i>output</i> yang sesuai	Dapat menghasilkan file Protocol Buffers dengan Message yang sesuai	Menghasilkan file Protocol Buffers dengan Message yang sesuai	Berhasil
Melakukan konversi Kotlin Data Class dan menghasilkan <i>output</i> sesuai dengan konfigurasi yang dipilih	Dapat menghasilkan file Protocol Buffers dengan Message yang sesuai dengan konfigurasi yang dipilih	Menghasilkan file Protocol Buffers dengan Message yang sesuai dengan konfigurasi yang dipilih	Berhasil
Kasus dan hasil Uji Salah			
Skenario uji	Yang diharapkan	Yang dihasilkan	Kesimpulan
Memilih file selain kotlin untuk memanggil <i>plugin</i>	Dapat menampilkan pesan bahasa belum tersedia	Menampilkan pesan bahasa belum tersedia	Berhasil

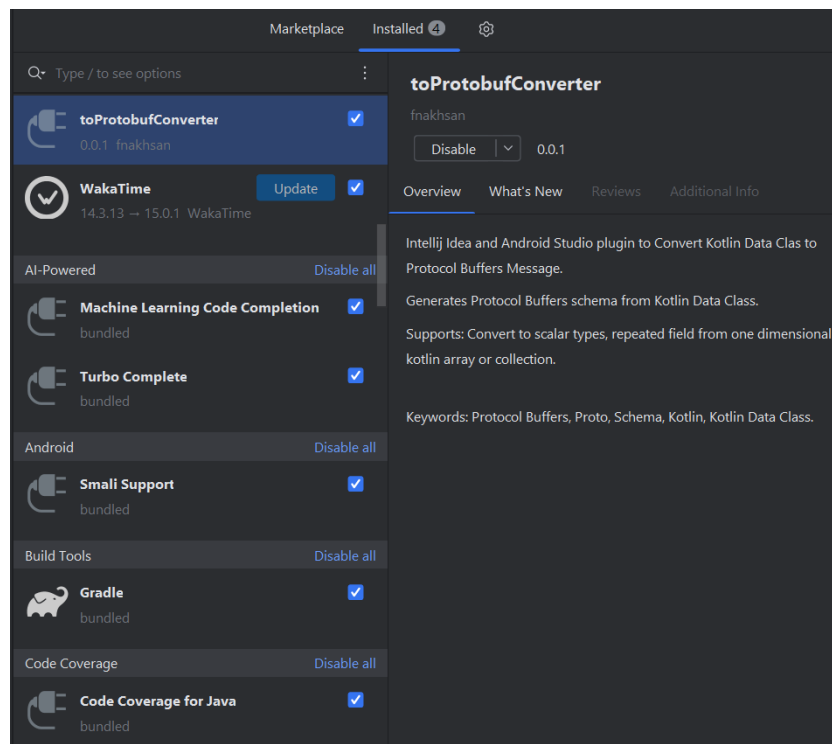
Mengosongkan nama file atau memberikan nama file dengan format yang salah	Dapat menampilkan pesan nama file tidak boleh kosong dan harus dengan format <i>lower_snake_case</i>	Menampilkan pesan nama file tidak boleh kosong dan harus dengan format <i>lower_snake_case</i>	Berhasil
Memilih file kotlin yang tidak mempunyai Kotlin Data Class	Dapat menampilkan pesan kesalahan struktur	Menampilkan pesan kesalahan struktur	Berhasil
Melakukan konversi dari Kotlin Data Class yang mempunyai field dengan tipe data yang memiliki dimensi lebih dari satu	Dapat menghasilkan file Protocol Buffers dengan <i>output</i> yang sesuai	Tidak dapat menghasilkan file Protocol Buffers dengan <i>output</i> yang sesuai	Tidak berhasil
Melakukan konversi yang dapat menghasilkan <i>output field</i> yang menggunakan <i>well-known</i> atau <i>common types</i>	Dapat menghasilkan <i>output field</i> yang menggunakan <i>well-known</i> atau <i>common types</i>	Tidak dapat menghasilkan <i>output field</i> yang menggunakan <i>well-known</i> atau <i>common types</i>	Tidak berhasil

4.4.3 Panduan Pengguna

Plugin ini dapat dipasang dan dijalankan pada dua IDE yaitu IntelliJ IDEA dan Android Studio. Untuk dapat dijalankan pada kedua IDE tersebut, dilakukanlah proses build dari kode sumber dari *plugin* menjadi *file* bertipe *zip*. File tersebut kemudian dipasang terlebih dahulu pada IDE sebelum *plugin* dapat dijalankan. Berikut merupakan langkah-langkah detail terkait pemasangan *plugin*:

1. Pastikan Anda memiliki IntelliJ IDEA atau Android Studio versi terbaru yang terinstal di komputer Anda.
2. *Download* file *zip plugin* dari link berikut [toProtobufConverter](#).
3. Buka menu *plugin* pada IDE.
4. Klik ikon berbentuk roda gigi pada kanan atas jendela menu *plugins* dan pilih Install Plugin from Disk.
5. Pilih file *zip plugin* yang telah Anda *download* pada langkah kedua.
6. Klik OK untuk memulai proses pemasangan *plugin*.
7. *Restart* IDE setelah proses pemasangan selesai.

Berikut merupakan tampilan setelah *plugin* berhasil terpasang pada menu Plugins:



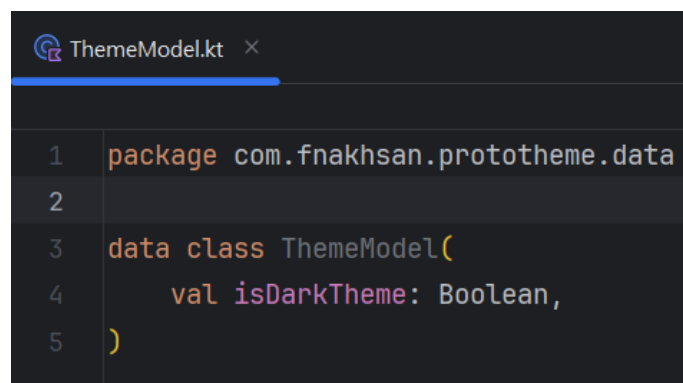
Gambar 14 Instalasi Plugin

Setelah pemasangan plugin selesai, plugin akan siap untuk digunakan untuk memudahkan pengembangan sistem yang memerlukan Kotlin dan Protocol Buffers. Berikut merupakan langkah-langkah detail terkait tata cara penggunaan *plugin*:

1. Buka IntelliJ IDEA atau Android Studio dan buat *project* baru atau buka *project* yang sudah ada.
2. Klik kanan pada *editor view* yang menampilkan file kotlin yang berisi data class yang ingin dikonversi dan pilih "Convert Kotlin Data Class to Protobuf".
3. Sebuah jendela form akan muncul.
4. Pilih konfigurasi untuk tipe *numeric/integer* yang diinginkan
5. Masukkan nama file *output*
6. Klik tombol "Convert" untuk menghasilkan file Protocol Buffers.
7. *File output* akan dihasilkan dan disimpan pada package yang sama dengan *file* sumber.
8. Anda dapat menggunakan file skema Protocol Buffers tersebut dalam *project* Anda.

4.4.4 Simulasi Hasil

Simulasi hasil dari konversi dilakukan untuk mengetahui apakah output dapat digunakan dan dimanfaatkan dalam pengembangan sistem yang menggunakan protocol buffers. Simulasi ini dilakukan pada pengembangan aplikasi android sederhana yang menggunakan proto datastore untuk menyimpan data sederhana. Aplikasi android sederhana yang dikembangkan memiliki fitur untuk mengganti dan menyimpan konfigurasi tema. Konversi dilakukan dari file Kotlin yang berisi Data Class yang dapat dilihat pada gambar 14, dengan hasil konversi dari file tersebut berupa file Protocol Buffers yang diberi nama “theme_prefs” yang dapat dilihat pada gambar 15. Proyek pada pengembangan aplikasi android tersebut kemudian dilakukan proses build sehingga file protocol buffer hasil dari konversi dapat dikompilasi menjadi beberapa file java yang dapat dilihat pada gambar 16. File java yang dihasilkan oleh compiler protocol buffers tersebut kemudian digunakan dengan proto datastore sehingga dapat menyimpan data berupa konfigurasi tema. Simulasi aplikasi android sederhana ini dapat dilihat pada gambar 17 dan 18 dimana aplikasi dapat berganti tema dari tema terang ke gelap dan sebaliknya.

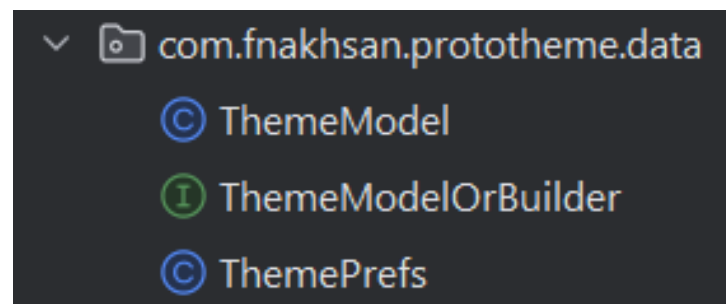


```
ThemeModel.kt x
1 package com.fnakhsan.prototheme.data
2
3 data class ThemeModel(
4     val isDarkTheme: Boolean,
5 )
```

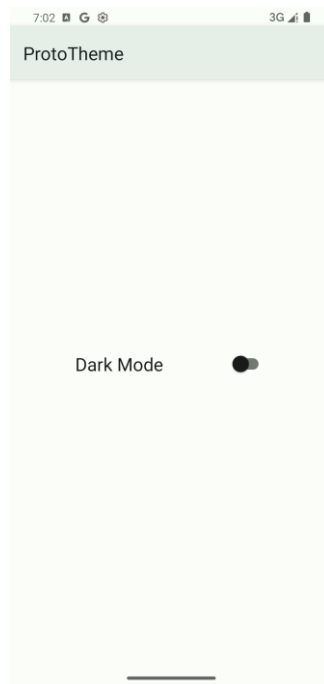
Gambar 15 File ThemeModel.kt

```
1 syntax = "proto3";
2
3 package com.fnakhsan.prototheme.data;
4
5 option java_package = "com.fnakhsan.prototheme.data";
6 option java_multiple_files = true;
7 option java_outer_classname = "ThemePrefs";
8
9 message ThemeModel {
10     bool is_dark_theme = 1;
11 }
```

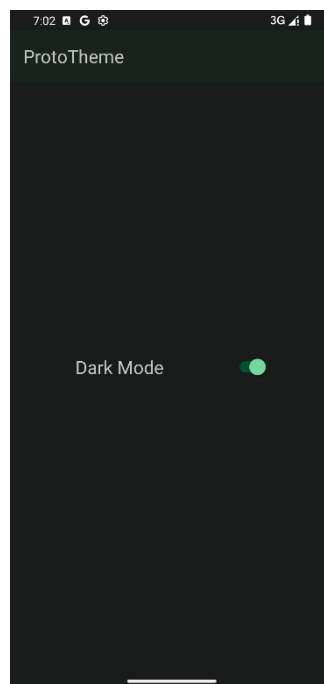
Gambar 16 File theme_prefs.proto



Gambar 17 Generated java files from proto file



Gambar 18 Aplikasi android “ProtoTheme” dengan tema dalam mode terang



Gambar 19 Aplikasi android “ProtoTheme” dengan tema dalam mode gelap

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Alat pengubah struktur data pada Kotlin Data Class menjadi skema yang didefinisikan melalui Protocol Buffers Message yang dirancang dan dibangun mampu untuk melakukan konversi dengan cakupan minimal berupa seluruh tipe data skalar secara akurat. Selain itu, alat yang dirancang dan dibangun menyediakan fungsionalitas untuk pengguna dapat menentukan nama file *output* dan konfigurasi tipe data *integer/numeric* sesuai yang diinginkan.

Alat ini dapat dimanfaatkan oleh developer sehingga dapat menghemat waktu, meminimalisir kesalahan, meningkatkan konsistensi, dan memfokuskan diri pada logika bisnis aplikasi daripada tugas-tugas transformasi data yang repetitif. Hal-hal tersebut berpotensi untuk meningkatkan kualitas dan stabilitas serta memudahkan pemeliharaan sistem yang menggunakan Kotlin dan Protocol Buffers. Kemudahan dalam menggunakan *plugin* juga berpotensi mendorong adopsi teknologi yang menggunakan Protocol Buffers seperti library Jetpack Proto DataStore yang digunakan pada pengembangan aplikasi android. Dengan demikian diharapkan perancangan dan pembangunan *plugin* ini dapat memberikan kontribusi positif bagi komunitas developer Kotlin dan Protocol Buffers beserta ekosistemnya sehingga dapat meningkatkan kualitas pengembangan sistem yang memakainya.

5.2 Saran

Beberapa saran yang dapat diterapkan untuk pengembangan plugin di masa depan antara lain:

1. Meningkatkan antarmuka pengguna pada plugin sehingga dapat lebih intuitif dan ramah pengguna.
2. Menambahkan dukungan konversi field bertipe *well-known* atau *common*.
3. Menambahkan dukungan untuk konversi dari Kotlin Array atau Collections dengan lebih dari satu dimensi.
4. Dukungan konversi dari bahasa pemrograman selain Kotlin.

DAFTAR PUSTAKA

- [1] B. S. Rothman, R. K. Gupta, and M. D. McEvoy, “Mobile Technology in the Perioperative Arena: Rapid Evolution and Future Disruption,” *Anesth. Analg.*, vol. 124, no. 3, pp. 807–818, 2017, doi: 10.1213/ANE.0000000000001858.
- [2] M. Ashoer, M. H. Syahnur, J. S. Tjan, A. Junaid, A. Pramukti, and A. Halim, “The Future of Mobile Commerce Application in a Post Pandemic Period; An Integrative Model of UTAUT2,” *E3S Web Conf.*, vol. 359, pp. 1–8, 2022, doi: 10.1051/e3sconf/202235905005.
- [3] “Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper,” *Cisco*, Jan. 2022. <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html> (accessed Jul. 04, 2024).
- [4] Google for Developers. *Google I/O Keynote (Google I/O '17)*. (May 17, 2017). Accessed: Jul. 04, 2024. [Online video]. Available: <https://www.youtube.com/live/Y2VF8tmLFHw?si=zgCrETDalg1xflXH&t=5230>.
- [5] “Android’s Kotlin-first approach,” *Android Developers*, 2019. <https://developer.android.com/kotlin/first> (accessed Jul. 04, 2024).
- [6] “Ringkasan Kotlin,” *Android Developers*, 2023. <https://developer.android.com/kotlin/overview?hl=id> (accessed Jul. 04, 2024).
- [7] “Kotlin Foundation – official site,” *Kotlin Foundation – official site*, 2023. <https://kotlinfoundation.org/> (accessed Jul. 04, 2024).
- [8] “Kotlin Programming Language,” *Kotlin*, 2024. <https://kotlinlang.org/> (accessed Jul. 04, 2024).
- [9] “App Architecture: Data Layer - DataStore - Android Developers,” *Android Developers*, 2024.

- <https://developer.android.com/topic/libraries/architecture/datastore>
(accessed Jul. 04, 2024).
- [10] Protocol Buffers, “Protocol Buffers,” *Protobuf.dev*, 2023. <https://protobuf.dev/> (accessed Jul. 04, 2024).
- [11] Kotlin, “Data classes,” *Kotlin Documentation*, 2024. <https://kotlinlang.org/docs/data-classes.html> (accessed Jul. 04, 2024).
- [12] M. E. Joorabchi, A. Mesbah, and P. Kruchten, “Real Challenges in Mobile App Development,” in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 15–24, doi: 10.1109/ESEM.2013.9.
- [13] M. Voelter, *Generic Tools, Specific Languages*. Delft University of Technology, 2014.
- [14] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, vol. 1. 2012.
- [15] JetBrains, “IntelliJ IDEA,” *JetBrains*, 2024. <https://www.jetbrains.com/idea/> (accessed Jul. 04, 2024).
- [16] “Download Android Studio & App Tools - Android Developers,” *Android Developers*, 2024. <https://developer.android.com/studio> (accessed Jul. 04, 2024).
- [17] “Kotlin Help,” *Kotlin Help*, 2016. <https://kotlinlang.org/docs/faq.html> (accessed Jul. 06, 2024).
- [18] “Kotlin Developer Stories,” *Android Developers*, 2024. <https://developer.android.com/kotlin/stories> (accessed Jul. 06, 2024).
- [19] H. Kolengsusu, “Rancang Bangun Plugin untuk Sistem Informasi Akademik dengan Ajax dan Web Services,” *BIMAFIKA J. MIPA, Kependidikan dan Terap.*, vol. 4, no. 1, pp. 425–434, 2012, [Online]. Available: <https://unidar.e-journal.id/bima/article/view/190>.
- [20] A. Shrivastava, I. Jaggi, N. Katoch, D. Gupta, and S. Gupta, “A Systematic Review on Extreme Programming,” *J. Phys. Conf. Ser.*, vol. 1969, no. 1, 2021, doi: 10.1088/1742-6596/1969/1/012046.

- [21] A. Restu Mukti, C. Mukmin, E. Randa Kasih, D. Palembang Jalan Jenderal Ahmad Yani No, S. I. Ulu, and S. Selatan, “Perancangan Smart Home Menggunakan Konsep Internet of Things (IOT) Berbasis Microcontroller,” *J. JUPITER*, vol. 14, no. 2, pp. 516–522, 2022.
- [22] “What is UML | Unified Modeling Language,” *Uml.org*, 2024. <https://www.uml.org/what-is-uml.htm> (accessed Jul. 16, 2024).
- [23] “What is Wireframing? The Complete Guide [Free Checklist] Figma,” *Figma*, 2022. <https://www.figma.com/resource-library/what-is-wireframing/> (accessed Jul. 16, 2024).
- [24] freeCodeCamp.org. *UML Diagrams Full Course (Unified Modeling Language)*. (April 22, 2021). Accessed: Jul. 16, 2024. [Online video]. Available: <https://www.youtube.com/watch?v=WnMQ8HlmeXc&t=3869s>
- [25] A. Nordeen, *Learn Software Testing in 24 Hours: Definitive Guide to Learn Software Testing for Beginners*. Guru99, 2020.
- [26] A. Verma, A. Khatana, and S. Chaudhary, “A Comparative Study of Black Box Testing and White Box Testing,” *Int. J. Comput. Sci. Eng.*, vol. 5, no. 12, pp. 301–304, 2017, doi: 10.26438/ijcse/v5i12.301304.
- [27] “RoboPOJOGenerator - IntelliJ IDEs Plugin | Marketplace,” *JetBrains Marketplace*, 2024. <https://plugins.jetbrains.com/plugin/8634-robopojogenerator> (accessed Jul. 22, 2024).
- [28] “pojo to proto - IntelliJ IDEs Plugin | Marketplace,” *JetBrains Marketplace*, 2024. <https://plugins.jetbrains.com/plugin/14691-pojo-to-proto> (accessed Jul. 22, 2024).