SKRIPSI

IMPLEMENTASI FITUR LOGGING PADA SDK PAYMENT ANDROID MENGGUNAKAN METODE SCRUM

Disusun Oleh:

N a m a : Ardian Dwi Rifai

NIM : 17523093

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2022

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI FITUR LOGGING PADA SDK PAYMENT ANDROID MENGGUNAKAN METODE SCRUM

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a	: Ardian Dwi Rifai
NIM	: 17523093

Yogyakarta, 30 Desember 2021

Pembimbing,

(Erika Ramadhani, S.T., M.Eng.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI FITUR LOGGING PADA SDK PAYMENT ANDROID MENGGUNAKAN METODE SCRUM

TUGAS AKHIR JALUR MAGANG

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

Yogyakarta, 6 Januari 2022

Tim Penguji

Ketua Penguji
Galang Prihadi Mahardhika, S.Kom., M.Kom. _____

Anggota 1
Ahmad Luthfi, S.Kom., M.Kom. _____

Anggota 2
Moh. Idris, S.Kom., M.Kom. _____

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	:	Ardian Dwi Rifai
NIM	:	17523093

Tugas akhir dengan judul:

IMPLEMENTASI FITUR LOGGING PADA SDK PAYMENT ANDROID MENGGUNAKAN METODE SCRUM

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian 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 risiko dan konsekuensi apapun.

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

Yogyakarta, 30 Desember 2021

(Ardian Dwi Rifai)

HALAMAN PERSEMBAHAN

Skripsi ini saya persembahkan kepada kedua orang tua saya yang telah mendukung, membina, membiayai, dan menyemangati saya dari kecil hingga mampu menyelesaikan tanggung jawab ini. Tak lupa skripsi ini merupakan persembahan bagi diri saya sendiri karena telah mampu bertahan dan berjuang melampaui ekspektasi diri sendiri.

HALAMAN MOTO

"*To Infinity and Beyond"*

– Buzz Lightyear

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Puji syukur atas kehadirat Allah SWT yang telah memberikan nikmatnya sehingga penulis dapat menyusun laporan tugas akhir dengan judul "Implementasi Fitur Logging pada Sdk Payment Android Menggunakan Metode Scrum".

Laporan tugas akhir ini disusun dengan tujuan untuk memenuhi syarat utama dalam menyelesaikan studi jenjang strata 1 jurusan Informatika Universitas Islam Indonesia. Penulis menyadari bahwa dalam penyusunan laporan ini masih memiliki kekurangan dan dengan dukungan dari berbagai pihak penulis dapat menyelesaikannya dengan lancar. Untuk itu, penulis tidak lupa menyampaikan terima kasih kepada:

1. Allah SWT yang telah melimpahkan nikmat, rahmat, serta hidayatnya yang telah memberikan kesempatan dan kemampuan dalam menyelesaikan tugas akhir ini.
2. Kedua orang tua saya yang telah banyak memberikan dukungan secara material dan moral.
3. Ibu Erika Ramadhani selaku dosen pembimbing tugas akhir yang telah memberikan bimbingan dan arahan dalam penyelesaian tugas akhir ini.
4. Bapak dan Ibu dosen program studi Informatika yang telah memberikan ilmu dan arahan selama masa perkuliahan.
5. Saudara Farid dan Afriandi selaku pembimbing lapangan selama program magang di PT Aino Indonesia.
6. Saudari Abigail selaku HC di PT Aino Indonesia yang telah membantu kelancaran proses kegiatan magang.
7. Rekan-rekan kerja di PT Aino Indonesia yang selalu membantu dan memberi wawasan yang melebihi harapan saya.
8. Dina, Andri, Khozim, dan teman-teman jalur magang lainnya yang selalu memberi semangat dan motivasi satu sama lain selama menjalani program magang
9. Saudara Chandra sebagai sahabat yang telah membantu dengan memberi saran dalam penyusunan laporan ini.
10. Saudara Yudha yang memberi semangat dan motivasi selama penyusunan laporan ini.

Semoga semua dukungan, bantuan, arahan, dan bimbingan yang telah diberikan dapat menjadi amal yang baik dan dibalas kebaikannya oleh Allah SWT. Semoga program magang ini dapat menjadi pengalaman dan pengajaran bagi penulis dalam menekuni dunia pekerjaan yang

sesungguhnya. Selama proses penyusunan ini penulis menyadari bahwa laporan ini masih belum sempurna. Sehingga penulis mengharapkan kritik dan saran yang membangun dari pembaca. Akhir kata, semoga laporan ini dapat memberikan manfaat untuk penulis dan para pembaca lainnya.

Wassalamu'alaikum Wr. Wb.

Yogyakarta, 30 Desember 2021

(Ardian Dwi Rifai)

SARI

Uang elektronik atau *e-money* merupakan inovasi teknologi yang memungkinkan pemrosesan transaksi pembayaran menjadi lebih mudah dan praktis. PT Aino Indonesia sebagai perusahaan *payment gateway* melayani proses pembayaran menggunakan teknologi ini dari berbagai segmen bisnis. Untuk memenuhi setiap kebutuhan segmen bisnis, Aino perlu mengembangkan berbagai macam aplikasi. Aplikasi-aplikasi ini memerlukan sistem pembayaran yang sama agar lebih cepat untuk diintegrasikan dan di-*maintenance*. Tim *payment* di Aino telah mengembangkan sebuah modul atau SDK untuk memberikan standar sistem pembayaran pada aplikasi berbasis Android. Dua fitur baru akan ditambahkan pada SDK ini yaitu fitur logging dan pengunggahan *file* log. Fitur *logging* akan mencatat data yang dihasilkan setiap proses transaksi ke dalam *file* log. Fitur selanjutnya adalah pengunggahan *file* log ke server untuk membantu pengambilan data transaksi yang diunggah dari perangkat. Kedua fitur akan membantu proses pelacakan *issue* dan masalah yang mungkin saja timbul saat proses transaksi berlangsung. Pengembangan kedua fitur pada SDK ini menggunakan metode *Scrum* untuk menghasilkan fitur yang berjalan dengan baik.

Kata kunci: logging, SDK, Scrum, *library*, *Logback*.

GLOSARIUM

Logging	Proses perekaman pesan, *events,* proses*,* atau lainnya pada sebuah *file* atau media lainnya.
Scrum	Kerangka kerja dalam metodologi pengembangan perangkat lunak yang prosesnya dilakukan dalam beberapa siklus iterasi yang dilakukan secara berkelanjutan.
SDK	Sekumpulan alat dan fungsionalitas yang disusun dalam suatu modul dan dapat digunakan untuk mengembangkan perangkat lunak.
Sprint	Iterasi dalam kerangka kerja Scrum dengan periode waktu tertentu untuk para anggota sebuah tim bekerja demi mencapai tujuan dari *sprint* tersebut.
Logback	*Library* pada bahasa pemrograman Java untuk melakukan proses *logging* pada beberapa media.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar belakang

Perkembangan teknologi informasi yang semakin pesat dapat memudahkan masyarakat dalam menjalankan kegiatannya sehari-hari. Ekonomi merupakan salah satu bidang yang turut dipengaruhi oleh pesatnya perkembangan teknologi. Hal ini dapat kita lihat pada inovasi sistem pembayaran yang tidak lagi menggunakan uang tunai atau yang biasanya disebut dengan sistem pembayaran nontunai.

Salah satu teknologi pada pembayaran nontunai adalah uang elektronik atau *e-money*. Uang elektronik adalah nilai uang atau produk prabayar yang nilai atau dananya disimpan pada perangkat elektronik yang dimiliki oleh seseorang (Bank for International Settlements, 1996). Salah satu contoh perangkat yang sering digunakan masyarakat adalah kartu *smartcards*. Nilai uang atau besar dana disimpan pada *microchips* yang sudah dibenamkan pada *smartcards* tersebut. Dengan menggunakan kartu ini, kegiatan transaksi bisa dilakukan dengan lebih cepat dan praktis karena hanya perlu menempelkan kartu pada alat pembaca atau *reader* yang sudah disediakan.

PT Aino Indonesia adalah perusahaan yang bergerak di bidang jasa pembayaran elektronik terintegrasi dan nontunai. PT Aino Indonesia merupakan anak perusahaan dari PT Gama Multi Usaha Mandiri yang merupakan *holding company* dari unit-unit bisnis milik Universitas Gadjah Mada. Beberapa segmen bisnis yang ditangani antara lain transportasi umum, jalan tol, parkir, pariwisata, dan ritel. Aino memiliki tujuan untuk menjadi perusahaan penyedia solusi sistem pembayaran terintegrasi. Dengan tujuan ini, Aino ingin mewujudkan visi *lesh cash city* sehingga penggunaan uang *cash* dapat dikurangi dan memaksimalkan penggunaan pembayaran elektronik terintegrasi.

Aino telah mengembangkan sejumlah sistem pembayaran tanpa kontak dengan mengintegrasikan produk-produk *e-money* yang dikeluarkan oleh beberapa bank dalam negeri. Salah satu sistem yang dikembangkan adalah menggunakan perangkat *mobile point of sales* atau mPOS yang berbasis sistem operasi Android. Perangkat ini dapat memproses transaksi berbasis *chip* dengan menggunakan *smartcards* dan transaksi berbasis server. Perangkat ini kemudian digunakan dengan aplikasi khusus yang dikembangkan sendiri oleh Aino. Perangkat ini biasanya digunakan pada segmen retail seperti *merchant* makanan dan minuman, *vending machine*, dan UMKM. Perangkat mPOS ini memberi beberapa keuntungan seperti perangkat yang mudah

dibawa, aplikasi yang dapat disesuaikan dengan kebutuhan bisnis, dan dapat menerima banyak sumber pembayaran. Dengan sistem ini proses layanan dan transaksi dapat dilakukan dengan mudah dan cepat sehingga dapat meningkatkan pengalaman pengguna dan dapat menghemat biaya yang sebelumnya diperlukan untuk alat kasir biasa(Lin, Ha, & Lin, 2015).

Dalam memenuhi kebutuhan segmen bisnis yang beragam, Aino mengembangkan aplikasi yang berbeda. Perbedaan dapat berupa desain antarmuka, fitur-fitur yang dikembangkan, cara penggunaan, dan lain-lain. Walaupun berbeda, fitur pembayaran tetap harus mengikuti proses yang sudah ditentukan oleh tim *payment* Aino. Hal tersebut menyebabkan sering munculnya pertanyaan-pertanyaan yang sama mengenai integrasi sistem pembayaran pada aplikasi. Pertanyaan-pertanyaan yang sama ini dapat memperlambat proses pengembangan aplikasi dan kinerja para *developer.* Sehingga diperlukan sebuah modul yang khusus untuk menangani integrasi sistem pembayaran pada aplikasi.

Modul yang dikembangkan untuk keperluan integrasi pembayaran berupa SDK atau *software development kit* untuk perangkat dengan sistem operasi Android. SDK ini memungkinkan para pengembang untuk mengimplementasikan sistem pembayaran ke dalam aplikasinya dengan mudah. Ada beberapa fitur yang disediakan, antara lain pembayaran *chip based,* pembayaran *server based,* dan cetak bukti transaksi. Lalu akan dikembangkan sebuah fitur baru yaitu fitur *logging*. Fitur ini akan memungkinkan aplikasi untuk merekam dan menyimpan data yang dihasilkan pada saat proses pembayaran berlangsung.

Penambahan fitur *logging* pada SDK merupakan sebuah solusi dari permasalahan yang muncul pada saat pengoperasian aplikasi di lapangan. Beberapa aplikasi yang sudah pernah dikembangkan tidak memiliki fitur ini. Akibatnya Aino perlu mengirimkan personel ke lapangan untuk menangani permasalahan yang muncul. Hal ini tentu memakan waktu dan biaya yang tidak sedikit. Dengan fitur *logging* ini setiap transaksi dapat dicatat pada *file log* dan dikelompokkan berdasarkan periode waktu tertentu. Selain itu, *file log* dapat diunggah ke server milik Aino sehingga proses *troubleshooting* dapat dilakukan tanpa harus mengirimkan personel ke lapangan. SDK ini dapat mempermudah proses integrasi fitur karena adanya standar yang sama pada berbagai aplikasi. Selain itu, proses *maintenance* akan lebih mudah dilakukan dan menghemat waktu serta biaya.

1.2 Ruang Lingkup Magang

Pelaksanaan kegiatan magang di PT Aino Indonesia berlangsung selama 6 bulan dimulai dari bulan Oktober 2020 hingga April 2021. Posisi yang diambil selama kegiatan magang adalah

mobile app developer khususnya sebagai Android *developer*. Aktivitas yang dilakukan sebagai berikut:

a. Mengembangkan fitur *logging* dengan fungsionalitas sebagai berikut:
    1. Mengembangkan fitur mencatat data transaksi ke *file* dengan format .log
    2. Mengembangkan fitur penamaan *file* sesuai dengan hari
    3. Mengembangkan fitur kompresi beberapa *file* log berdasarkan periode yang ditentukan
b. Mengembangkan fitur *upload file* log ke FTP *server* milik Aino

1.3 Tujuan

Mengembangkan fitur *logging* pada SDK *payment* pada perangkat dengan sistem operasi Android menggunakan metode Scrum.

1.4 Manfaat

Manfaat fitur *logging* yang diimplementasikan pada SDK *payment* adalah sebagai berikut:

a. Menuliskan informasi transaksi yang dilakukan dalam *file* log sehingga informasi lebih tertata dan mudah dibaca
b. Memudahkan pengembang untuk mengintegrasikan fitur yang dibutuhkan ke dalam aplikasinya.
c. Mempermudah perusahaan dalam pelacakan terhadap masalah yang muncul dengan melihat *file* log proses transaksi

1.5 Sistematika Penulisan

Pembahasan topik yang dibahas akan diuraikan dengan sistematika penulisan sebagai berikut:

a. Bab I adalah Pendahuluan, yang membahas tentang latar belakang pengembangan SDK pembayaran Aino, ruang lingkup magang, tujuan pengembangan SDK, manfaat pengembangan SDK, dan sistematika penulisan laporan.
b. Bab II adalah Dasar Teori, yang mengulas tentang teori-teori yang berkaitan dengan pengembangan SDK pada laporan ini.
c. Bab III adalah Pelaksanaan Magang, yang mengurai tentang pelaksanaan magang yang dilalui selama kegiatan magang di PT Aino Indonesia. Topik yang dibahas adalah

metode pengembangan yang digunakan, manajemen proyek di PT Aino Indonesia, dan proses pengembangan fitur *logging* SDK *payment.*

d. Bab IV adalah Refleksi Pelaksanaan Magang, yang menguraikan mengenai refleksi selama mengikuti pengembangan fitur *logging* SDK Payment.
e. Bab V adalah Kesimpulan dan Saran yang berisi tentang kesimpulan dari pengembangan SDK *payment* dan saran untuk pengembangan selanjutnya.

BAB II
DASAR TEORI

2.1 *Logging*

Menurut Cambridge Dictionary, sebuah log adalah catatan tertulis lengkap dari sebuah perjalanan, periode waktu, atau peristiwa. *Logging* dalam sebuah aplikasi adalah suatu cara yang sistematis dan terkontrol untuk merepresentasikan kondisi sebuah aplikasi sehingga dapat dibaca oleh manusia (Gupta, 2003). Sebuah log dapat merekam berbagai *events* yang terjadi pada saat aplikasi dijalankan. Rekaman-rekaman log ini dapat membantu pengembang aplikasi untuk melakukan analisis dan mengidentifikasi permasalahan yang muncul. Informasi-informasi yang direkam dapat memberikan petunjuk pada saat proses pelacakan bagian aplikasi yang menimbulkan perilaku yang tidak diinginkan (Davidekova & Gregusml, 2016). Dengan demikian proses *logging* cukup penting untuk diterapkan pada aplikasi yang dikembangkan.

Mengingat pentingnya proses *logging* maka proses harus dirancang dengan baik dan terstruktur. Proses *logging* yang dirancang dengan baik harus menghasilkan informasi yang relevan dengan *events* dan tidak menimbulkan permasalahan baru. Proses *logging* yang dilakukan terlalu sedikit tentunya tidak dapat memberi petunjuk yang berarti pada pengembang. Proses *logging* yang berlebihan juga tidak dapat memberikan manfaat yang baik. Proses yang dilakukan secara berlebihan akan mengonsumsi sumber daya yang tidak sedikit sehingga dapat mengganggu kinerja utama aplikasi. Data yang dihasilkan juga tidak maksimal karena adanya data yang berulang dan tidak penting. Untuk menghindari hal-hal tersebut, para pengembang perlu memperhatikan di mana *logging* dilakukan dan apa saja yang perlu di-*log* pada sistem yang dikembangkan (Fu dkk., 2014).

2.2 Logback

Logback adalah sebuah *logging framework* untuk aplikasi berbasis Java dan merupakan penerus dari *library* log4j yang cukup populer (Gülcü, Pennec, & Harris, t.t.). Logback memiliki beberapa keunggulan yang lebih banyak apabila dibandingkan dengan pendahulunya. Meskipun begitu, kedua *framework* tersebut dikembangkan oleh pengembang yang sama sehingga keduanya memiliki konsep dan cara penggunaan yang hampir sama(Gülcü dkk., t.t.).

Arsitektur Logback terdiri dari beberapa lapisan yang memiliki fungsi yang berbeda. Terdapat tiga komponen utama yang memungkinkan pengembang aplikasi untuk melakukan proses *logging* dan mengontrol bagaimana proses dilakukan. Ketiga komponen tersebut yaitu:

a. Logger

Objek *logger* memiliki tanggung jawab untuk melakukan proses *logging*. Objek ini menyediakan beberapa metode untuk melakukan proses *logging* informasi dengan cara yang berbeda. Setiap *logger* terikat oleh LoggerContext yang bertanggung jawab untuk membuat objek tersebut sekaligus menyusunnya dalam bentuk hierarki. Terdapat aturan penamaan untuk menentukan hierarki setiap objek. Sebuah *logger* merupakan *parent* dari *logger* lainnya apabila namanya diikuti dengan sebuah titik dan merupakan awalan dari nama turunannya. Sebagai contoh, "com.foo" merupakan *parent* dari *logger* bernama "com.foo.Bar". Setiap hierarki memiliki sebuah *root logger* yang berada pada tingkatan paling atas dan merupakan bagian awal dari seluruh hierarki.

Setiap *logger* diberikan sebuah level sehingga hanya bisa melakukan proses log pada level yang sama atau di atasnya. Terdapat lima level yang dapat diberikan pada sebuah *logger*, yaitu TRACE, DEBUG, INFO, WARN, dan ERROR. Jika tidak diberikan level, sebuah *logger* akan diwarisi oleh objek pada tingkatan di atasnya. Untuk memastikan bahwa semua *logger* dapat mewarisi level, *logger* pada tingkatan *root* selalu memiliki level. Secara *default* level yang diberikan pada *root logger* adalah DEBUG.

b. Appender

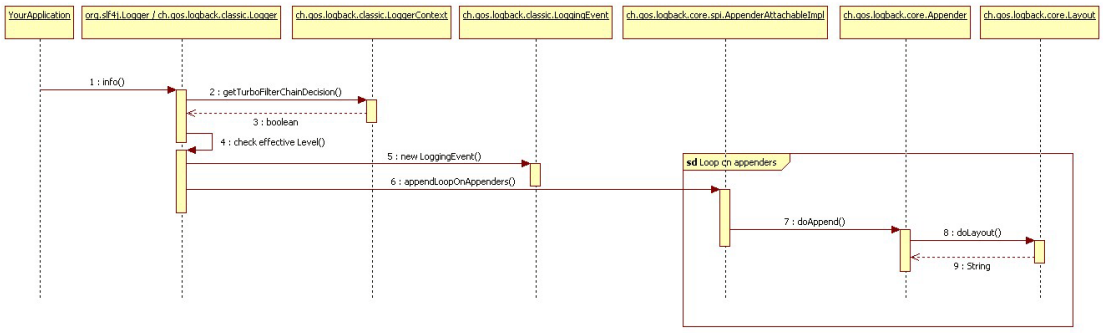
Objek *appender* memiliki tanggung jawab untuk mengarahkan informasi log ke suatu tujuan. Setiap *appender* memiliki satu destinasi log, seperti *console*, *file*, atau *socket*. Sebuah objek *logger* dapat memiliki satu atau lebih objek *appender*. Setiap proses *logging* yang dilakukan oleh objek *logger* akan diteruskan ke *appenders* objek tersebut sekaligus ke *appenders* milik *logger* tingkatan yang lebih tinggi pada hierarki.

c. Layout

Objek ini berfungsi untuk menentukan bagaimana format informasi *logging* dicetak dengan *style* yang sudah ditentukan. *Layout* akan memformat *event logging* menjadi objek *string.* Objek *appender* menggunakan *layout* sebelum mengirimkan informasi *logging.*

Logback mengimplementasi *facade* SLF4J. SLF4J merupakan jembatan antar berbagai macam *framework logging*. Logback mengekspos metode-metode *logging* dengan melalui *facade* ini. Dengan mengimplementasikan *facade* ini, proses penggantian *framework* bisa dilakukan dengan mudah.

Proses *logging* sebuah informasi melibatkan tiga komponen utama beserta komponen tambahan lainnya. Gambar 2.1 memberikan ilustrasi mengenai bagaimana proses *logging* dilakukan pada Logback.



Gambar 2.1 Diagram UML alur proses *logging*

Sumber: logback.qos.ch (2020) diakses pada 29 Agustus 2021

Dalam proses *logging* terdapat beberapa langkah. Langkah yang dilakukan Logback saat metode *printing* dipanggil adalah sebagai berikut:

1. Aplikasi memanggil salah satu metode *printing* dan mengirimkan pesan pada objek dan mengirimkan informasi *logging.*
2. Selanjutnya adalah proses *filtering*. Jika TurboFilter diaktifkan, komponen tersebut akan melakukan *filter* terhadap *event* berdasarkan Marker, Level, informasi pesan, dan Throwable pada *request logging*. Apabila balasan filter berupa FilterReply.DENY, *request* akan ditolak. Balasan FilterReply.Neutral akan diteruskan ke tahap pembandingan Level dan balasan FilterReply.ACCEPT akan diteruskan ke tahap pembuatan objek LoggingEvent.
3. Tahap selanjutnya Logback akan melakukan perbandingan terhadap *level request* dengan *level* efektif *logger*. Jika *request* tidak sesuai, maka Logback akan membatalkan *request*. Jika sesuai, Logback akan meneruskan ke tahap selanjutnya.
4. Setelah pengecekan level efektif, Logback akan membuat objek LoggingEvent yang mengandung parameter-parameter dari *logging request.*

5. Setelah objek LoggingEvent berhasil dibuat, Logback akan memanggil metode *doAppend*() sehingga informasi log dapat diproses oleh *appender*.
6. Tahap berikutnya adalah pendelegasian pemformatan *logging event* oleh objek Layout. Objek Layout akan memformat *logging event* menjadi sebuah objek *string*.
7. Setelah proses *formatting* selesai dilakukan, hasilnya kemudian dikembalikan ke *appender* yang akan mencetak informasi log ke tujuan yang telah ditentukan.

2.3 FTPClient

FTPClient adalah sebuah *library* dalam Bahasa pemrograman Java yang dikembangkan oleh The Apache Software Foundation. FTPClient mengenkapsulasi semua fungsionalitas yang diperlukan untuk menyimpan dan mendapatkan *file* dari *server* FTP ("FTPClient (Apache Commons Net 3.8.0 API)," t.t.). Karena berbasis SocketClient, *library* ini sebelum digunakan harus terkoneksi terlebih dahulu dengan *server* yang dituju dan menutup koneksi tersebut apabila proses yang dilakukan telah selesai. FTPClient mempermudah seluruh proses komunikasi dengan server FTP dengan menyediakan *interface* yang mudah digunakan dan menangani seluruh interaksi detail tingkat rendah antara perangkat dengan server .

2.4 Scrum

Scrum adalah sebuah kerangka kerja atau *framework* dalam pengembangan perangkat lunak yang bersifat *iterative* dan *incremental* (Sutherland, 2007). Scrum merupakan kerangka kerja yang memungkinkan sebuah tim untuk membangun kemampuan menyelesaikan tantangan yang kompleks dan adaptif dalam pengembangan dan penyampaian sebuah produk yang berkualitas dengan cara meningkatkan kolaborasi, kreativitas, dan produktivitas (Bhavsar, Shah, & Gopalan, 2020). Scrum merupakan salah satu jenis metode Agile yang populer dan telah digunakan oleh banyak perusahaan. Scrum pertama kali diciptakan oleh Jeff Sutherland pada tahun 1993 dan kerangka kerjanya disusun oleh Ken Schwaber pada tahun 1995 (Sutherland, 2007).

Scrum dilaksanakan dengan landasan proses empiris. Proses empiris menekankan bahwa pengetahuan berasal dari pengalaman dan penentuan keputusan dilakukan berdasarkan observasi dari pengalaman tersebut. Untuk mengimplementasikan proses empiris, Scrum menjunjung tiga pilar, yaitu (Schwaber & Sutherland, 2020):

a. *Transparency*

   Transparency adalah memberikan keterbukaan atau transparansi atas proses dan perkembangan kepada pihak-pihak yang terkait.

b. *Inspection*

Seluruh perkembangan dan hasil pekerjaan harus melalui proses inspeksi atau pemeriksaan untuk mengurangi kemungkinan permasalahan baru.

c. *Adaptation*

Apabila ada aspek dari proses yang menyimpang dari batasan, adaptasi harus dilakukan secepatnya sehingga tidak menyimpang lebih jauh.

Dalam Scrum, proses pengembangan perangkat lunak menggunakan konsep *iterative*. Konsep ini menstrukturkan sebuah proyek ke dalam siklus-siklus singkat yang diberi nama *Sprint*. Sebuah *Sprint* biasanya berlangsung selama dua sampai empat minggu. Setiap anggota tim berkomitmen dalam menyelesaikan suatu pekerjaan yang sudah dipilih dan ditentukan pada awal *Sprint* dimulai. Pada saat *Sprint* berlangsung, tidak ada perubahan pekerjaan yang dapat mempengaruhi tujuan *Sprint*. Pekerjaan dinyatakan selesai apabila telah memenuhi kebutuhan produk. Dalam hal perangkat lunak, suatu pekerjaan dapat dinyatakan selesai apabila kode sudah terintegrasi, sudah diuji sepenuhnya, dan berpotensi dapat di-*deploy* (Sutherland, 2007). Apabila sebuah *Sprint* telah selesai, akan dilanjutkan dengan *Sprint* baru. Siklus ini akan dilakukan secara berulang hingga tujuan Scrum sudah terpenuhi.

2.4.1 Peran-peran dalam Scrum Team

Setiap orang yang dalam lingkup sebuah Scrum membentuk sebuah tim yang diberi nama dengan Scrum Team. Untuk mengimplementasikan Scrum, lingkungan kerja harus memiliki lima Scrum Values yaitu *courage*, *focus*, *commitment*, *respect*, dan *openness*. Setiap anggota dari Scrum Team perlu untuk mempelajari dan mengimplementasikan seluruh nilai-nilai tersebut untuk membentuk kepercayaan antar peran dan mendukung kesuksesan berlangsungnya Scrum (Gonçalves, 2018). Dalam Scrum Team terdapat tiga peran sebagai berikut:

a. *Product Owner*

Product Owner bertanggung jawab untuk memaksimalkan nilai produk dengan mengidentifikasi fitur produk, membuat daftar prioritas fitur yang akan dikembangkan, dan menentukan prioritas teratas untuk dikerjakan pada *Sprint* berikutnya (Sutherland, 2007). Untuk mencapai keberhasilan dalam Scrum, semua pihak harus menghormati keputusan yang dibuat oleh *Product Owner*. Hal ini dapat dilihat pada *Product Backlog* yang sudah disusun sebelumnya. *Product Owner* memiliki tanggung jawab untuk mengelola *backlog* secara efektif sebagai berikut (Schwaber & Sutherland, 2020):

- Mengembangkan dan mengomunikasikan tujuan produk secara eksplisit.

- Membuat dan mengomunikasikan item-item *backlog* secara jelas.
- Mengurutkan item-item *backlog.*
- Memastikan bahwa *backlog* transparan, terlihat, dan dapat dipahami.

b. Development Team

Development Team mengembangkan produk yang dibutuhkan oleh *customer.* Dalam sebuah tim terdiri dari beberapa orang yang memiliki keahlian yang beragam seperti *programmer, quality assurance,* dan UI *designer*. Semua anggota dalam tim dapat disebut sebagai *developer* tanpa melihat pekerjaan yang dilakukan (Gonçalves, 2018). Dalam Scrum tidak ada peran manajer atau manajer proyek. Sehingga *Development Team* mengatur dirinya sendiri dalam memutuskan apa yang harus dilakukan dan cara melakukannya dengan penuh tanggung jawab (Sutherland, 2007).

c. Scrum Master

Scrum Master adalah seorang pemimpin dalam *Scrum Team*. *Scrum Master* bertanggung jawab untuk memastikan bahwa setiap anggota tim telah memahami dan mengikuti langkah-langkah yang telah dirumuskan pada Scrum. Hal ini dapat dilakukan dengan cara membantu setiap anggota untuk memahami teori dan praktik Scrum. Selain itu, *Scrum master* dapat membantu memimpin organisasi untuk menangani perubahan yang sulit untuk mencapai kesuksesan dalam pengembangan produk menggunakan metodologi Agile (Sutherland, 2007).

2.4.2 Scrum Artifacts

Scrum Artifacts merepresentasikan pekerjaan atau nilai yang dilakukan oleh suatu tim. *Artifacts* memberikan transparansi tentang pekerjaan. Terdapat tiga *Scrum Artifacts* sebagai berikut:

a. Product Backlog

Product Backlog adalah daftar kebutuhan produk yang harus dipenuhi. Daftar ini menjadi sumber utama dari pekerjaan yang dikerjakan oleh tim yang sudah diurutkan berdasarkan prioritas. *Product Backlog* akan selalu diperbarui dan disempurnakan oleh *Product Owner* karena adanya perubahan kebutuhan, inovasi baru, kendala teknis, dan lain-lain (Sutherland, 2007).

b. *Sprint Backlog*

Sprint Backlog adalah daftar pekerjaan yang harus dilakukan pada sebuah *sprint*. Daftar ini direncanakan sekaligus dikerjakan oleh Developer Team. *Sprint* Backlog menggambarkan secara jelas dan *real-time* mengenai apa yang akan dicapai selama *Sprint* (Schwaber & Sutherland, 2020).

c. Increment

Increment adalah versi dari produk yang merepresentasikan kemajuan dari proses pengembangan produk tersebut (Gonçalves, 2018). Setiap *increment* yang telah selesai akan ditambah *increment* berikutnya sehingga produk memenuhi kebutuhan yang telah dirancang. Sebuah pekerjaan dapat dikatakan sebagai *Increment* apabila telah memenuhi kriteria *Definition of Done*. *Definition of Done* adalah keadaan suatu *Increment* ketika telah memenuhi kualitas yang diperlukan sebuah produk (Schwaber & Sutherland, 2020). Hal ini akan memberikan kepastian dan kejelasan tentang bagaimana sebuah pekerjaan dapat diselesaikan sebagai bagian dari sebuah *Increment*.

2.4.3 *Scrum Events*

Terdapat beberapa aktivitas yang dilakukan dalam siklus *Sprint*. Tujuan dari aktivitas-aktivitas ini adalah menjaga keterbukaan antar anggota *Scrum Team*. *Scrum Events* juga dapat menjadi kesempatan untuk memeriksa produk dan mengadaptasi produk apabila terdapat perubahan. Terdapat empat *events* yang dilakukan dalam satu siklus *sprint*, yaitu:

a. *Sprint Planning*

Sprint diawali dengan sebuah pertemuan yang disebut dengan *Sprint Planning*. Pertemuan ini dipimpin oleh *Product Owner* dengan agenda merencanakan serta mendiskusikan pekerjaan dari *Product Backlog* dan bagaimana cara mencapai tujuan dari *Sprint* tersebut (Schwaber & Sutherland, 2020). Aktivitas ini secara kolaboratif melibatkan semua anggota dari *Scrum Team* dan pihak luar apabila diperlukan. *Sprint Planning* biasanya berlangsung dalam satu hari selama kurang dari delapan jam untuk *Sprint* yang berdurasi empat minggu.

b. *Daily Scrum*

Setiap hari selama *Sprint* berlangsung diawali dengan *Daily* Scrum. *Daily* Scrum biasanya berlangsung selama lima belas menit dan diadakan pada waktu yang sama

setiap harinya. Pada saat *Daily Scrum* berlangsung, satu per satu anggota dari tim melaporkan tiga hal, yaitu apa yang sudah diselesaikan sejak pertemuan terakhir, apa yang akan dikerjakan sampai pertemuan berikutnya, dan hambatan atau rintangan yang menghalangi pekerjaan (Sutherland, 2007). *Daily* Scrum meningkatkan komunikasi antar anggota dan menjaga semua pekerjaan tetap teratur sehingga tujuan *sprint* dapat tercapai.

c. *Sprint Review*

Setelah sebuah *Sprint* berakhir, diadakan *Sprint Review*. *Sprint Review* adalah pertemuan antara *Product Owner* dengan *Development Team* serta *stakeholders* apabila diperlukan. Pada pertemuan ini *Product Owner* dan *Development Team* melakukan peninjauan mengenai apa saja yang telah dicapai dan apa saja perubahan yang terjadi. Setelah didapatkan umpan balik dari pencapaian sebelumnya, perlu diadakan penyesuaian pada *Product Backlog* untuk memenuhi perubahan baru. Pada pertemuan ini harus ada diskusi mendalam antara *Product Owner* dan *Development Team* dan tidak hanya presentasi pencapaian produk saja (Sutherland, 2007).

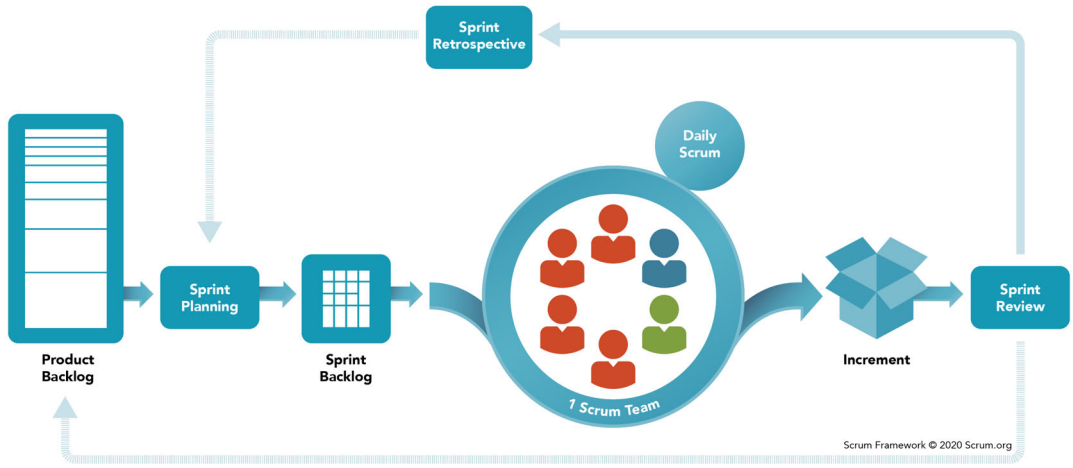
d. *Sprint Retrospective*

Sprint Retrospective diadakan setelah *Sprint Review* selesai dilakukan. Tujuan diadakannya pertemuan ini adalah meningkatkan efektivitas serta kualitas *Sprint* (Schwaber & Sutherland, 2020). Seluruh anggota tim dapat memeriksa dan mengamati *Sprint* yang telah selesai dilakukan pada aspek interaksi, individu, proses, dan lain-lain. Selain itu, Scrum Team dapat merencanakan perbaikan-perbaikan yang dapat dilakukan untuk meningkatkan efektivitas pelaksanaan *Sprint* selanjutnya (Gonçalves, 2018).

BAB III
PELAKSANAAN MAGANG

3.1 Metode Pengembangan Proyek

Proses pengembangan perangkat lunak membutuhkan suatu metode untuk mengelola berbagai aspek yang dibutuhkan pada saat proses pengembangan berlangsung. Penggunaan metode dapat membantu perancangan, analisis, dan mempercepat proses pengembangan sehingga menghasilkan sistem yang berkualitas serta sesuai dengan kebutuhan bisnis (Bolung & Tampangela, 2017). Pengembangan proyek ini mengimplementasikan metode pengembangan perangkat lunak Agile dengan kerangka kerja Scrum. Scrum memiliki beberapa tahapan seperti pada Gambar 3.1.



Gambar 3.1 Tahapan metode Scrum
Sumber: Scrum.org (2020) diakses pada 29 Agustus 2021

Pengembangan SDK Payment dilakukan dengan dengan tahapan sebagai berikut:

a. *Sprint Planning*

   Sprint Planning berlangsung selama kurang dari delapan jam karena Aino melaksanakan sebuah *Sprint* dengan durasi dua minggu. *Sprint Planning* dilaksanakan pada awal *Sprint* dan diikuti oleh *Product Owner*, seluruh tim *development*, dan *Scrum Master*. *Sprint Planning* ini membahas mengenai beberapa hal, seperti tujuan *sprint*,

keadaan dari tim *development*, menguraikan *backlog*, dan umpan balik atau peningkatan dari *sprint* sebelumnya.

b. *Daily Scrum*

Daily Scrum diadakan setiap hari pada saat *sprint* berlangsung pada pukul 09.30 pagi. Pertemuan ini hanya dilakukan selama lima belas menit. *Daily Scrum* membahas mengenai perkembangan proyek yang dilakukan dan hambatan yang dihadapi. Seluruh aktivitas yang dilakukan pada *sprint* dilaporkan dengan menggunakan aplikasi Jira. Aplikasi ini memudahkan pemantauan pengembangan proyek yang dikerjakan dan pelacakan mengenai *issue*, *bug*, atau masalah yang muncul. *Daily Scrum* ini merupakan bentuk dari prinsip transparansi metode Scrum. Dengan ini seluruh anggota tim harus terbuka mengenai ide, pendapat dan permasalahan yang ditemui selama *sprint* berlangsung.

c. *Sprint Review*

Sprint Review dilaksanakan pada akhir *Sprint* selama empat jam. Pertemuan ini membahas mengenai tugas-tugas yang sudah dikerjakan disertai dengan demonstrasi dari fitur yang dikembangkan. Demonstrasi ini selanjutnya menjadi bahan pertimbangan untuk penyesuaian *backlog* yang akan dikerjakan pada iterasi *sprint* selanjutnya. Untuk itu transparansi dari setiap anggota tim Scrum sangat penting untuk memastikan bahwa proyek yang dikembangkan tetap pada jalur yang tepat.

d. *Sprint* Retrospective

Sprint Restrospective dilakukan setelah *Sprint Review* selesai dilaksanakan. Aktivitas ini meninjau pelaksanaan *Sprint* yang baru saja dilakukan. Beberapa hal yang ditinjau antara lain interaksi antar anggota tim. Hasil dari aktivitas ini adalah peningkatan dan umpan balik mengenai kinerja tim yang dapat diterapkan pada *sprint* selanjutnya.

3.2 Manajemen Proyek

Pelaksanaan program magang di PT Aino Indonesia dilakukan selama kurang lebih enam bulan terhitung dari 1 Oktober 2020 hingga 1 April 2021. Selama mengikuti program magang, penulis ikut serta dalam segala kegiatan yang dilakukan bersama dengan karyawan lain. Penulis ditempatkan sebagai Android *developer* selama kegiatan magang berlangsung. Tugas utama sebagai Android *developer* yaitu mengikuti proses pengembangan proyek, membantu melakukan

dokumentasi dari aplikasi dan sistem, dan melakukan proses final *quality control* terhadap aplikasi. Ada beberapa divisi atau segmen di PT Aino Indonesia yang berkaitan dengan proses *development* yaitu AFC, *Retail*, dan *Payment*. Penulis pada saat awal magang ditempatkan pada segmen AFC yang memiliki peran dalam pengembangan pembayaran segmen bisnis transportasi, parkir dan lainnya. Namun penulis ditempatkan pada segmen *Payment* selama dua *sprint* karena segmen tersebut memerlukan Android *developer* tambahan untuk mengejar *deadline* pengembangan fitur pada SDK *Payment*.

Dalam melakukan proses pengembangan suatu aplikasi atau perangkat lunak, PT Aino Indonesia memiliki prosedur yang merupakan sebuah siklus dalam pengembangan perangkat lunak. Gambar 3.2 menjabarkan proses dan alur dalam siklus pengembangan perangkat lunak. Prosedur diawali dari *planning* atau perencanan yang merupakan agenda dengan periode tertentu. Proses selanjutnya adalah *development* yang merupakan proses pengembangan perangkat lunak oleh *software engineer*, pengujian oleh *quality assurance*, dan proses lainnya. Selanjutnya adalah tahap pra-release yaitu proses pengujian SIT dan UAT. Pada tahap *pra-release* ini dokumentasi dibuat seperti *user manual*, *technical documentation*, dan lain-lain. Tahap selanjutnya adalah *release*. Pada tahap ini dilakukan *handover* produk ke tim operasional sehingga proses-proses selanjutnya dilakukan oleh tim operasional. Tim operasional akan melakukan *deployment* berdasarkan dokumentasi yang sudah dibuat sebelumnya oleh tim developer. Selain itu tim operasional akan melakukan PTR atau *production test run* langsung di lapangan. Tahap terakhir adalah *launching* yaitu tahap peluncuran aplikasi, fitur, atau *enhancement* yang langsung dapat digunakan oleh pengguna. Apabila akan dikembangkan fitur baru atau pengembangan lainnya pada suatu perangkat lunak, proses pengembangan akan kembali ke tahap *planning* dan meneruskan tahap lainnya seperti yang dilakukan sebelumnya.

Gambar 3.2 Prosedur Bisnis Pengembangan Perangkat Lunak di PT Aino Indonesia

Pada saat kegiatan magang, penulis ikut serta pada siklus pengembangan terutama pada tahap *development*. Pengembangan perangkat lunak menggunakan metode Agile. Pada saat pengembangan fitur SDK di segmen *Payment*, kerangka kerja yang digunakan adalah Scrum. Pengembangan fitur SDK ini berlangsung selama dua *sprint* atau empat minggu. Anggota tim Scrum dalam segmen *payment* terdiri dari *Product Owner*, *full-stack developer*, *quality assurance*, *technical lead*, dan *mobile developer*. Untuk pengembangan fitur logging ini, divisi payment menambah dua *mobile developer* untuk mempercepat pengembangan SDK *Payment*. Kedua *developer* tersebut adalah penulis dengan satu senior Android *developer* yang berasal dari segmen AFC.

Ada beberapa tahap yang dilakukan saat aktivitas pengembangan fitur *logging* pada SDK Payment. Tahap-tahap tersebut adalah sebagai berikut:

3.2.1 Definisi Proyek

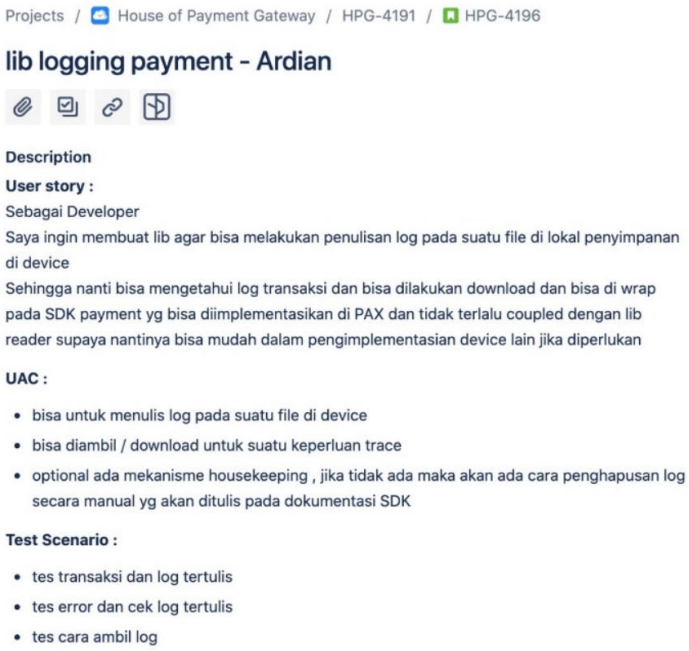
SDK *Payment* adalah sebuah *library* atau modul yang dikembangkan PT Aino Indonesia. SDK ini dikembangkan oleh segmen *payment* yang berfokus pada pengembangan proses transaksi

pembayaran. Modul ini memiliki beberapa fungsionalitas utama, seperti memproses transaksi berbasis *chip* untuk *smartcard* yang diluncurkan oleh beberapa bank nasional, mencetak hasil transaksi pembayaran, dan proses *logging* transaksi. Fitur *logging* adalah fitur baru yang akan ditambahkan ke dalam modul ini. Fitur ini dapat digunakan untuk menuliskan data atau informasi dari suatu *event* atau proses yang dijalankan pada aplikasi.

3.2.2 Fase *Inception*

Sebelum memulai proses-proses pada kerangka kerja Scrum, terdapat satu fase yang perlu dilakukan yaitu fase *inception.* Tujuan dari fase ini adalah mengkomunikasikan tujuan, visi, dan konteks dari proyek sehingga dapat menjadi landasan untuk pembuatan keputusan dan eksekusi dari proyek yang dikembangkan (Rasmusson, 2010). Beberapa aktivitas yang dilakukan pada fase ini antara lain membuat produk *backlog*, *story mapping*, menentukan *definition of done*, dan menentukan komposisi tim.

Fase *inception* ini dipimpin oleh *Product Owner* dengan melibatkan anggota tim lainnya. Pengembangan ini melibatkan tiga developer Android dan satu *quality assurance*. Penulis sebagai developer Android memiliki tugas untuk mengembangkan fitur *logging* dengan proses pencatatan log ke dalam *file* dan *upload file* log. Dua developer lainnya memiliki tugas mengembangkan transaksi *chipbase* untuk dua bank dan fitur *print* transaksi. Dalam aktivitas *story mapping*, metode *user story* diterapkan untuk memahami lebih baik mengenai kebutuhan dan spesifikasi dari fitur SDK yang akan dikembangkan. *User story* merupakan sebuah metode untuk mendeskripsikan kebutuhan sebuah produk dari perspektif *client* atau pengguna produk tersebut (Cohn, 2004). *User story* dituliskan dengan pola sederhana seperti "*Sebagai (peran) saya ingin (tujuan), sehingga (tujuan)*". Pada Gambar 3.3 adalah *user story* yang dihasilkan pada saat sesi diskusi. *Definition of done* dari fitur ini adalah ketika fitur sudah memenuhi kebutuhan sesuai dengan *user story*. Kebutuhan tersebut adalah mengembangkan fitur logging yang dapat digunakan untuk menulis log transaksi. Kemudian *file* log harus bisa diambil dari perangkat yang digunakan. Fitur juga diintegrasikan dengan SDK sehingga bisa digunakan bersama fitur lainnya.



Gambar 3.3 *User story* fitur logging SDK Payment

3.2.3 Perencanaan Proyek

Setelah fase *inception* telah berhasil merancang tujuan dan *backlog* dari proyek, langkah selanjutnya adalah perencanaan *sprint*. Proses pengembangan fitur *logging* ini direncanakan untuk dikembangkan selama dua *sprints* dengan durasi masing-masing *sprint* adalah dua minggu. Pembahasan rencana *sprint* ini dilakukan pada saat aktivitas *Sprint Planning*. Perencanaan *sprint* dilakukan seperti pada Tabel 3.1. Perencanaan juga membahas mengenai teknologi yang akan digunakan seperti pada Tabel 3.2.

Perencanaan selanjutnya adalah mengenai struktur kode program yang dikembangkan. struktur kode program dikembangkan dengan prinsip *Clean Architecture*. Prinsip ini memiliki tujuan utama *separation of concerns* dengan membagi komponen aplikasi menjadi beberapa layer individu yang modular (Robert C. Martin, 2018). Dengan prinsip ini SDK menjadi mudah di-*maintain* dan lebih mudah untuk menambah atau mengganti komponen tanpa mempengaruhi komponen lainnya.

Tabel 3.1 Perencanaan *sprint*

No	Aktivitas	Jumlah *Sprint*
1	Mengembangkan proses *logging* dan *upload file* log	1 *sprint*
2	Integrasi dengan *master project* dan menyusun dokumentasi	1 *sprint*

Tabel 3.2 Teknologi yang digunakan dalam fitur logging SDK Payment

No	Teknologi	Alasan
1	Kotlin	Kotlin merupakan bahasa pemrograman *open source* dan sudah menjadi *first class language* dalam pengembangan aplikasi Android. Kotlin memiliki keuntungan sebagai bahasa yang *concise* dan memiliki fitur *null safety* yang dapat mengurangi kesalahan akibat NullPointerException. Kotlin juga dapat digunakan bersamaan dengan kode program berbasis Java.
2	Library Logback	Memiliki beragam pilihan konfigurasi untuk proses *logging* dan menggunakan *file* xml.
3	Library FTPClient	Mudah digunakan dan mudah di-*maintain*.

3.2.4 Pelaksanaan *Sprint* 1 Pengembangan Fitur *Logging* dan *Upload* Log

Sesuai dengan perencanaan yang sudah dilakukan, agenda pada *Sprint* pertama adalah mengembangkan fitur *logging* dan *upload file* log. Secara pribadi, penulis membagi waktu dua minggu dengan fokus pekerjaan tertentu. Minggu pertama fokus untuk mengembangkan *logging* dan minggu kedua fokus untuk pengembangan *upload file* log. Beberapa aktivitas yang dilakukan selama *sprint* pertama adalah sebagai berikut:

1. Riset mengenai penggunaan library Logback

   Informasi terkait penggunaan *library* logback didapatkan melalui dokumentasi resmi pada halaman web Logback tersebut. Selain itu, informasi juga diperoleh dari contoh implementasi *library* pada proyek orang lain di internet. Fitur *logging* sendiri sebenarnya sudah ada pada Android yaitu menggunakan *library* android util.log. Tetapi *library* ini masih memiliki proses yang terbatas. Catatan log hanya dapat dicetak pada *console* dan bukan pada *file.* Logback digunakan untuk memenuhi kebutuhan. Fitur-fitur yang ada seperti menuliskan pesan ke dalam *file* dengan konfigurasi tertentu, kompresi *file*, dan format pencatatan pesan log. Penggunaan *library* ini dapat mempercepat proses pengembangan fitur dan mempermudah integrasi dengan fitur lainnya.

2. Set up implementasi fitur *logging*

   Untuk menggunakan library logback, perlu menambahkan beberapa implementasi pada *file* build.gradle modul. Gambar 3.4 menunjukkan *dependencies* yang digunakan untuk fitur *logging*. Logback yang digunakan adalah versi yang dikembangkan oleh tony19 untuk platform Android. Karena Logback menggunakan *façade* SLF4J, *library* slf4j perlu diimplementasikan. Proses *logging* ini tidak akan secara langsung terlihat oleh pengguna. Untuk itu proses *logging* ini perlu dijalankan secara asinkron. Untuk mengakomodasi proses asinkron, SDK menggunakan Coroutine yang merupakan fitur dari Kotlin. Kelas implementasi fitur *logging* ini membutuhkan dua objek yaitu *logger* dan *loggerContext*.

Kedua objek tersebut berasal dari *library* Logback. Penggunaannya seperti pada Gambar 3.5.

```
dependencies {
     //Logback
    implementation 'org.slf4j:slf4j-api:1.7.25'
    implementation 'com.github.tony19:logback-android:2.0.0'

    //Coroutine
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.4.1'
    …
}
```

Gambar 3.4 *Dependencies* untuk Logback dan Coroutine

```
class FileLogger : CoroutineScope {
    private val job = Job()
    override val coroutineContext: CoroutineContext
        get() = Dispatchers.IO + job

    private lateinit var logger: ch.qos.logback.classic.Logger
    private val loggerContext = LoggerFactory.getILoggerFactory() as
    LoggerContext

    …
}
```

Gambar 3.5 Objek yang digunakan dalam kelas implementasi *logging*

3. Implementasi konfigurasi Logback secara *programatically*

   Pengembangan fitur dilakukan pada *file* proyek yang terpisah dengan *file* proyek utama. Hal ini dilakukan untuk memastikan fitur yang dikembangkan dapat berjalan. Selain itu, proyek utama merupakan sebuah modul yang memiliki arsitektur *clean architecture* yang sudah dibuat sebelumnya dengan fitur lainnya. Dengan menggabungkan fitur baru ke dalam proyek utama akan membutuhkan waktu dan usaha yang tidak sedikit. Integrasi dilakukan apabila program sudah bisa dipastikan berjalan. Proses integrasi dapat dilakukan dengan cepat dan mudah karena sifat *clean architecture* yang modular.

Pada fitur ini terdapat dua *method* yang dapat digunakan oleh aplikasi. Pertama adalah *method kickstart* yang memiliki tanggung jawab dalam memulai proses konfigurasi *logger*. Kedua adalah *method write* yang akan melakukan proses pencatatan log. Perancangan kedua *method* ini mengikuti *method* yang sudah dibuat pada *interface* fitur *logging* pada proyek utama.

Langkah pertama dalam pengembangan fitur *logging* adalah melakukan konfigurasi terhadap *library* Logback. Proses konfigurasi dapat dilakukan dengan dua cara yaitu melalui *file* xml atau dilakukan secara langsung pada sumber kode program. Konfigurasi dilakukan secara langsung pada sumber kode program guna mendapatkan konfigurasi yang dapat diubah-ubah. Dalam penyusunan konfigurasi ini beberapa komponen perlu diperhatikan, seperti *logger context*, *appender*, *rolling policy*, *pattern* layout, dan objek *logger*.

Appender yang digunakan adalah RollingFileAppender yang memiliki kemampuan *rollover*. Kemampuan ini memungkinkan *appender* untuk menuliskan pesan log ke dalam sebuah *file* dan mengganti target *file* apabila suatu kondisi yang ditentukan sudah terpenuhi. Kondisi ini ditentukan dari komponen *rolling policy*. *Rolling policy* yang digunakan adalah TimeBasedRollingPolicy yang melakukan proses *rollover* dengan ketentuan periode waktu tertentu seperti harian, mingguan, atau bulanan. Penentuan periode *rollover* ditentukan dengan penamaan pada *file* hasil proses *rollover*. Pada Gambar 3.6 penamaan *file* menggunakan format "%d{yyyy-MM-dd}" yang menentukan proses *rollover* dilakukan setiap hari dengan format nama *file* seperti hari proses *logging* dilakukan. Gambar 3.7 merupakan *pattern layout* yang akan menentukan pola pesan log ditulis pada *file*.

```
/** Create daily rolling policy log to file with given path **/
private fun createRollingPolicy(directory: String): RollingPolicy {
val rollingPolicy = TimeBasedRollingPolicy<ILoggingEvent>().apply {
      fileNamePattern = "$directory/%d{yyyy-MM, aux}/log.%d{yyyy-MM-dd}.txt"
            maxHistory = 31
            context = loggerContext
    }
    rollingPolicy.setParent(rollingFileAppender)
    return rollingPolicy
}
```

Gambar 3.6 Kode program konfigurasi *rolling policy*

```
/** Create pattern for the logging **/
private fun createEncoder(): PatternLayoutEncoder =
PatternLayoutEncoder().apply {
      pattern = "%logger{35} - %msg%n"
      context = loggerContext
}
```

Gambar 3.7 Kode program konfigurasi *pattern layout*

Setelah kedua komponen tersebut selesai dibuat, *appender* siap untuk dikonfigurasi. Konfigurasi *appender* berada pada *method kickstart* seperti pada Gambar 3.8 . *Logger* selanjtunya dikonfigurasi dengan mengaitkan komponen *appender*, level debug, dan kemampuan *additive*. Dengan ini, *logger* sudah dikonfigurasi dan dimulai sehingga proses *logging* dapat dijalankan.

```
/** Start the logger with given path **/
    fun kickStart(filePath: String) {
        loggerContext.stop()
        val directory = createAbsoluteFilePath(filePath)
        val mRollingPolicy = createRollingPolicy(directory)
        val mEncoder = createEncoder()

        mRollingPolicy.start()
        mEncoder.start()

        /** Preparing and starting the Appender **/
        rollingFileAppender.apply {
            isAppend = true
            context = loggerContext
            file = "$directory/log.txt"
            rollingPolicy = mRollingPolicy
            encoder = mEncoder
            name = "RollingFileAppender"
        }
        rollingFileAppender.start()

        /** Starting the Logger **/
        logger = LoggerFactory.getLogger("FileLogger") as
                 ch.qos.logback.classic.Logger
        with(logger) {
            addAppender(rollingFileAppender)
            level = Level.DEBUG
            isAdditive = false
        }

        StatusPrinter.print(loggerContext)
    }
```

Gambar 3.8 Kode konfigurasi *appender* dan *logger* pada *method kickstart*

4. Implementasi *method write*

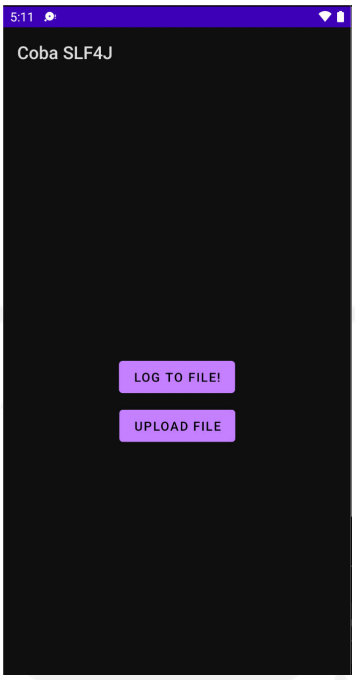
   Method kedua yang dikembangkan adalah *method write*. Gambar 3.9 menunjukkan implementasi kode program pada *method write. Method* ini akan melakukan proses pencatatan log menggunakan objek *logger* yang sudah di deklarasikan. *Method logger* yang digunakan adalah debug. Terdapat tiga parameter yang nantinya akan dicatat oleh *logger*. Parameter pertama adalah *tag* menandakan bagian proses aplikasi yang memanggil proses *logging* tersebut. Kedua adalah *identifier* yang memberi identitas pada log yang dibuat. Parameter terakhir adalah *message* berisikan informasi yang akan dicatat pada log.

```
fun write(tag: String, identifier: String, message: String) {
   this.launch {
      logger.debug("{} {} {}", tag, identifier, message)
   }
}
```

Gambar 3.9 Kode program *method write*

5. Uji coba fitur *logging* menggunakan aplikasi simulator

   Selanjutnya adalah pembuatan antarmuka aplikasi yang akan digunakan untuk menguji fitur *logging*. Gambar 3.10 menunjukkan antarmuka yang sederhana dengan hanya menampilkan tombol untuk menjalankan proses *logging* dan satu tombol lainnya untuk menjalan proses mengunggah *file* log. Apabila tombol untuk menjalankan proses *logging* ditekan, proses akan berjalan dengan menuliskan pesan pada *file* log.



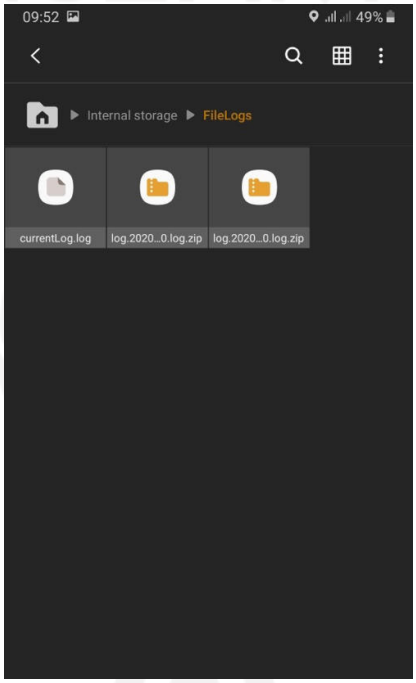
Gambar 3.10 Antarmuka aplikasi

Penggunaan proses *logging* dilakukan pada MainActivity aplikasi simulator seperti pada Gambar 3.11. Penggunaan fitur *logging* dapat dilakukan dengan mendeklarasikan objek FileLogger yang diberi nama *logger*. Kemudian pada *method* onCreate objek *logger* tersebut diinisiasi dengan objek FileLogger. Untuk melakukan proses logging maka perlu memanggil *method kickstart* pada objek *logger*. Proses *logging* akan berjalan Ketika tombol log to *file* ditekan. Untuk itu maka dibuat listener apabila tombol tersebut ditekan. Pada *listener* tersebut, objek *logger* akan memanggil *method write* seperti pada Gambar 3.11.

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
    private var storagePermission = false
    private lateinit var logger: FileLogger
    …

   override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        checkStoragePermission()

        if (storagePermission) {
            logger = FileLogger()
            logger.kickStart()

            btn_main.setOnClickListener(this)
            btn_upload.setOnClickListener(this)
        } else {
            finish()
        }
    }

    override fun onClick(view: View?) {
        when (view?.id) {
            R.id.btn_main -> {
               logger.write(this::class.java.simpleName, "identifier",
               "Holo")
            }
            …
        }
    }
}
```

Gambar 3.11 Penggunaan *logging* pada MainActivity aplikasi simulator

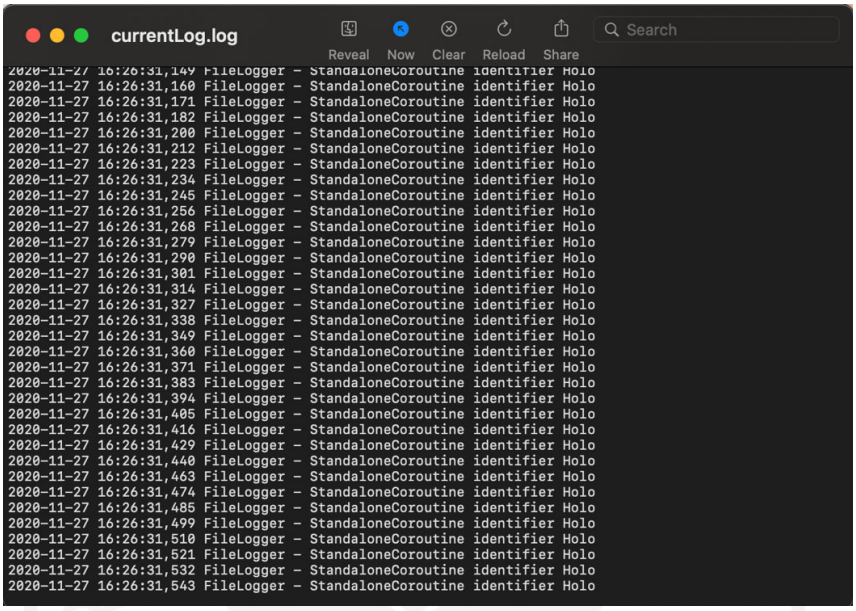
Untuk mengetahui proses *logging* berhasil atau tidak, perlu dilihat pada direktori perangkat seperti pada Gambar 3.12. Pada direktori tersebut terdapat *file* dengan nama log dan berisi pesan log. Dengan ini fitur *logging* sudah dapat berjalan. Untuk menguji apakah *rollover* berhasil, tanggal pada perangkat ditambah satu hari. Selanjutnya aplikasi dijalankan kembali dan tombol ditekan. Pada direktori, *file* pada hari sebelumnya berubah nama menjadi tanggal pada saat proses *logging* dilakukan dan terdapat *file* log baru untuk tempat proses *logging* pada hari yang baru.



Gambar 3.12 Tampilan file hasil proses logging pada direktori perangkat

Apabila *file* log tersebut dibuka maka akan terlihat seperti pada Gambar 3.13. Dalam *file* tersebut terdapat beberapa baris informasi log yang di-*generate* pada saat proses pengujian *logging* menggunakan *coroutine*. Dalam sekali proses log, *logger* akan mencatatkan beberapa informasi. Yang pertama adalah waktu pada saat proses logging tersebut dieksekusi. Selanjutnya adalah nama dari *logger* yang digunakan. Pada Gambar 3.13

dapat dilihat bahwa proses *logging* dilakukan oleh *logger* yang diberi nama *FileLogger*. Selanjutnya adalah informasi yang diberikan pada saat pemanggilan *method write*.



Gambar 3.13 Informasi pada file log hasil proses logging

6. Riset penggunaan FTPClient

Fitur selanjutnya yang dikembangkan adalah *upload* atau mengunggah *file* log ke *server*. Sebelumnya, ada beberapa pilihan *server* yang bisa digunakan, seperti Google Drive, Microsoft Sharepoint, dan FTP *server*. FTP *server* milik Aino dipilih karena infrastrukturnya sudah tersedia. Selain itu, ada beberapa keuntungan dengan menggunakan *server* FTP milik Aino sendiri, yaitu Fitur FTP ini membutuhkan sebuah *library* yang yang dapat membantu proses pengunggahan *file*.

Library FTPClient dari Apache Commons dipilih untuk membantu pengembangan fitur ini. *Library* ini dapat menangani segala kebutuhan untuk menyimpan dan mendapatkan *file* dari server FTP. Untuk mengetahui penggunaan FTPClient, penulis mempelajarinya dari dokumentasi yang ada pada *website* FTPClient, Library FTPClient ini mampu untuk melakukan proses upload *file* ke FTP *server* sekaligus mampu untuk membuat suatu direktori dan mencari suatu *file* pada server tersebut. Fitur-fitur ini cocok untuk digunakan dalam pengembangan SDK Payment yang membutuhkan kemampuan untuk mengunggah *file* dan mengecek apakah *file* telah diunggah pada server.

7. Implementasi FTPClient

Dependency yang dibutuhkan untuk FTPClient seperti pada Gambar 3.14. Pemanfaatan *library* FTPClient dilakukan dengan membuat suatu kelas bernama FTPClientHelper. Kelas ini nantinya dibuat menjadi sebuah objek yang bisa digunakan untuk melakukan proses pengunggahan *file*. Tanggung jawab dari kelas ini adalah membuka koneksi untuk komunikasi antara *server* dan perangkat, menutup koneksi, membuat atau mengubah direktori pada server, dan mengunggah *file* ke server. *Method* yang berada dalam kelas ini tidak dipanggil secara langsung oleh aplikasi, tetapi digunakan pada kelas implementasi dari kelas *interface* pada proyek utama.

```
dependencies{
    …
    //FTP
    implementation 'commons-net:commons-net:3.7.2'
}
```

Gambar 3.14 *Dependency* untuk FTPClient

Kelas implementasi *fitur upload* memiliki tiga *method* utama yang dapat dipanggil. Seluruh *method* pada kelas ini menggunakan objek dari FTPClientHelper yang sudah dibuat sebelumnya. *Method* yang pertama kali harus dipanggil adalah *init*. *Method* ini digunakan untuk menyiapkan informasi mengenai direktori penyimpanan, *username*, dan *password* dari *server* FTP. *Method* ini membuka koneksi antara *client* pada aplikasi dan server FTP dengan memanggil *method open* dari objek FTPClientHelper. *Method close* digunakan untuk menutup koneksi antara aplikasi dengan server. *Method* ini dipanggil apabila semua proses *upload file* sudah selesai dilakukan. Gambar 3.15 menunjukkan kode program pada *method init* dan *close*.

```
override fun init(uploadDirectory: String, username: String, password:
String) {
  this.uploadDirectory = "transaction_log/$uploadDirectory"
  this.username = username
  this.password = password
  client.open(username, password)
}

override fun close() {
  client.close()
}
```

Gambar 3.15 Kode program pada *method init* dan *close*

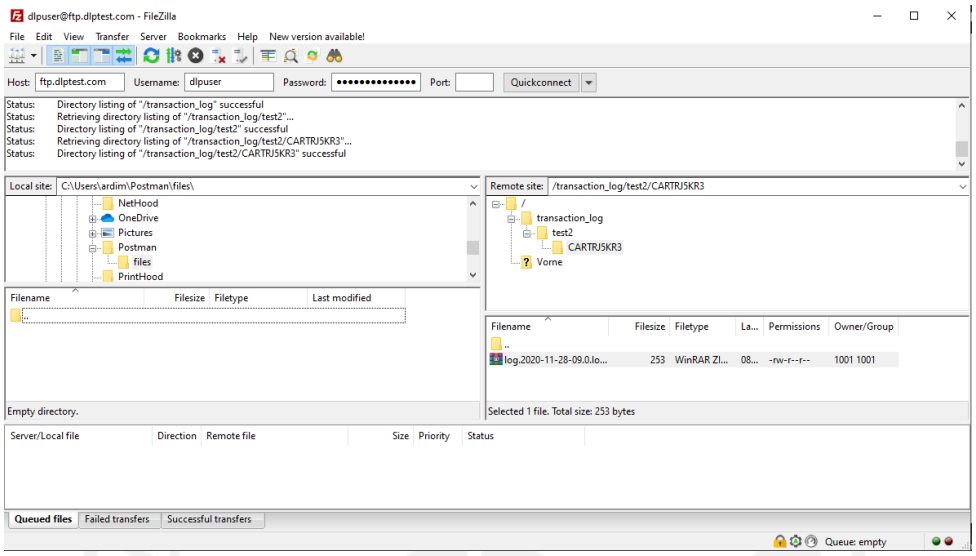
Method selanjutnya adalah *uploadLog* yang digunakan untuk mengunggah *file*. Proses pertama yang dilakukan adalah mencari *file* terbaru yang dapat diunggah. Apabila terdapat *file* yang dapat diunggah, proses pengunggahan *file* dijalankan dengan memanggil *method upload* pada objek dari FTPClientHelper. Proses pengunggahan *file* ini menggunakan fitur Coroutine yang disediakan oleh Kotlin. Fitur ini memungkinkan suatu proses untuk dijalankan secara asinkron. Dengan ini, proses pengunggahan *file* dapat dijalankan tanpa mengganggu dan memblokir proses lain pada aplikasi. Gambar 3.16 menunjukkan kode program pada *method uploadLog.*

```
override fun uploadLog(filePath: String) {
   val logFile = getLatestFile(filePath)
   if (logFile != null) {
      launch {
         val uploaded = client.uploadFile(logFile,
         uploadDirectory)
         if (!uploaded) {
            Log.e("Upload Process", "Upload Failed")
         }
      }
   }
}
```

Gambar 3.16 Kode program *method uploadLog*

8. Percobaan fungsi *upload file log*

   Untuk mengetahui apakah fitur *upload* ini berjalan, perlu diuji dengan mengunggah *file* log ke *server* FTP. Namun pada saat proses pengembangan berlangsung *server* FTP belum disiapkan. Untuk itu pengujian menggunakan server lain yang tersedia. *Server* FTP milik DLPTest merupakan *server* publik dan terbuka untuk semua orang. *Server* yang disediakan tidak berbayar dan *file* yang disimpan akan dihapus secara berkala. Pengujian dilakukan dengan menekan tombol *upload* pada antarmuka aplikasi. Program akan mengunggah *file* log yang sudah dibuat sebelumnya pada saat proses *logging*. Gambar 3.17 menunjukkan hasil proses *upload* pada server DLPTest menggunakan aplikasi FileZilla.



Gambar 3.17 Hasil pengunggahan *file* pada server FTP DLPTest

3.2.5 Pelaksanaan *Sprint* 2 Integrasi Fitur dan Penyusunan Dokumentasi

Perencanaan aktivitas yang dilakukan pada *Sprint* kedua adalah integrasi fitur yang penulis kembangkan ke SDK dan juga penyusunan dokumentasi mengenai penggunaan SDK pada aplikasi. Namun pada saat percobaan fitur *logging*, muncul suatu masalah sehingga perlu diperbaiki. Aktivitas yang dilakukan pada *sprint* kedua adalah sebagai berikut:

1. Konfigurasi ulang Logback menggunakan xml

   Permasalahan yang ditemukan adalah proses *rollover file* hanya bisa dilakukan selama aplikasi masih berjalan. Apabila aplikasi dimatikan dan kemudian dijalankan kembali, proses *rollover* akan diatur ulang dan *file* yang sebelumnya sudah tercatat akan terhapus. Kondisi ini tidak ideal bagi aplikasi karena tidak ada jaminan perangkat yang digunakan dapat berjalan terus menerus. Oleh karena itu perlu dicari sumber masalahnya dan diperbaiki sehingga proses *logging* dapat berjalan sesuai dengan kebutuhan fitur.

   Penyebab masalah pada proses *logging* ini terletak pada proses konfigurasi *logger*. Proses konfigurasi yang dilakukan langsung pada sumber kode memiliki kelemahan tersendiri. Proses *logging* akan selalu dikonfigurasi ulang pada saat aplikasi dibuka. Proses konfigurasi ini mengatur ulang proses *rollover* sebelumnya sehingga aplikasi akan melakukan proses *rollover* baru dan menghapus *file* log yang sudah ada. Untuk itu diperlukan sebuah solusi untuk mempertahankan konfigurasi proses *rollover*.

Cara konfigurasi lain yang dapat dilakukan adalah menggunakan *file* xml. Konfigurasi menggunakan *file* xml hanya dilakukan sekali saat aplikasi dijalankan pertama kali. Walaupun aplikasi ditutup atau dimatikan, informasi mengenai *rollover* dan data *file* yang sudah dibuat sebelumnya tetap dipertahankan.

Konfigurasi menggunakan *file* xml lebih mudah dilakukan dan cukup sederhana. *File* xml diberi nama logback.xml dan ditempatkan di dalam direktori assets. Logback akan mencari *file* konfigurasi pada direktori tersebut secara otomatis. Komponen yang digunakan pada konfigurasi kurang lebih sama dengan konfigurasi sebelumnya. Terdapat beberapa komponen yang diubah , yaitu pola penamaan *file* dan *rolling policy*. Gambar 3.18 menunjukkan konfigurasi *logger* pada *file* xml.

```
<configuration>
    <property name="DATA_DIR" value="/storage/emulated/0/FileLogs" />
        <appender name="FILE_APPENDER-${id}"
         class="ch.qos.logback.core.rolling.RollingFileAppender">
            <file>${DATA_DIR}/currentLog.log</file>
            <rollingPolicy
      class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
                <fileNamePattern>${DATA_DIR}/log.%d{yyyy-MM-dd}-
                 %d{HH,aux}.%i.log.zip</fileNamePattern>
                <maxHistory>14</maxHistory>
                <maxFileSize>3MB</maxFileSize>
                <totalSizeCap>1GB</totalSizeCap>
            </rollingPolicy>

            <encoder>
                <pattern>%d %logger{35} - %msg%n</pattern>
            </encoder>
        </appender>
    <root level="DEBUG">
        <appender-ref ref="SIFT" />
    </root>
</configuration>
```

Gambar 3.18 Konfigurasi logger pada file xml

Pada konfigurasi ini *rolling policy* yang digunakan berbeda dengan yang sebelumnya. SizeAndTimeBasedRollingPolicy digunakan untuk membantu memenuhi kebutuhan dari fitur *logging*. Kondisi *rollover* yang ditentukan oleh *policy* ini bergantung dari ukuran *file* dan periode waktu. Elemen maxFileSize menentukan ukuran terbesar dari sebuah *file*. Apabila sebuah *file* telah melampaui batas ukuran tersebut, *file* akan di-*rollover* dan *file* log baru dibuat. *File* yang dihasilkan oleh proses *rollover* dalam kurun waktu satu hari bisa saja lebih dari satu. Untuk membedakannya, ditambahkan elemen "%i" pada fileNamePattern yang merupakah sebuah angka penanda urutan pembuatan *file* pada hari tersebut.

Elemen maxHistory pada konfigurasi tersebut menjadi penentu lama sebuah *file* dapat disimpan pada perangkat. Apabila telah melewati maxHistory, *file* akan dihapus oleh Logback secara otomatis. Elemen selanjutnya adalah totalSizeCap. Elemen ini akan menentukan ukuran besar dari semua *file*. Elemen totalSizeCap menentukan jumlah seluruh *file* log pada perangkat. *File* terlama akan dihapus apabila ukuran seluruh *file* telah melampaui batas.

2. Penambahan kompresi *file*

Terdapat satu kebutuhan yang belum dipenuhi oleh fitur *logging* pada *sprint* sebelumnya. Pada proses *logging* sebelumnya, *file* yang dihasilkan oleh proses *rollover* masih dihasilkan dalam format .log dan belum terkompresi. Kebutuhan fitur menyatakan bahwa *file* yang akan diunggah ke server perlu dikompresi terlebih dahulu. Penambahan proses kompresi pada Logback cukup mudah yaitu dengan menambahkan pola ".zip" pada *fileNamePattern*. Dengan penambahan ini, Logback akan secara otomatis melakukan kompresi *file* pada proses *rollover*.

Kode program pada *method kickStart* menjadi lebih sederhana karena pemindahan konfigurasi ini. Kini *method* tersebut digunakan untuk menyiapkan objek *logger* yang nantinya akan melakukan proses *logging*. Gambar 3.19 menunjukkan *method kickStart* pada kode program. Aplikasi kemudian dijalankan kembali untuk memastikan proses *logging* dapat berjalan dengan benar dan sesuai kebutuhan.

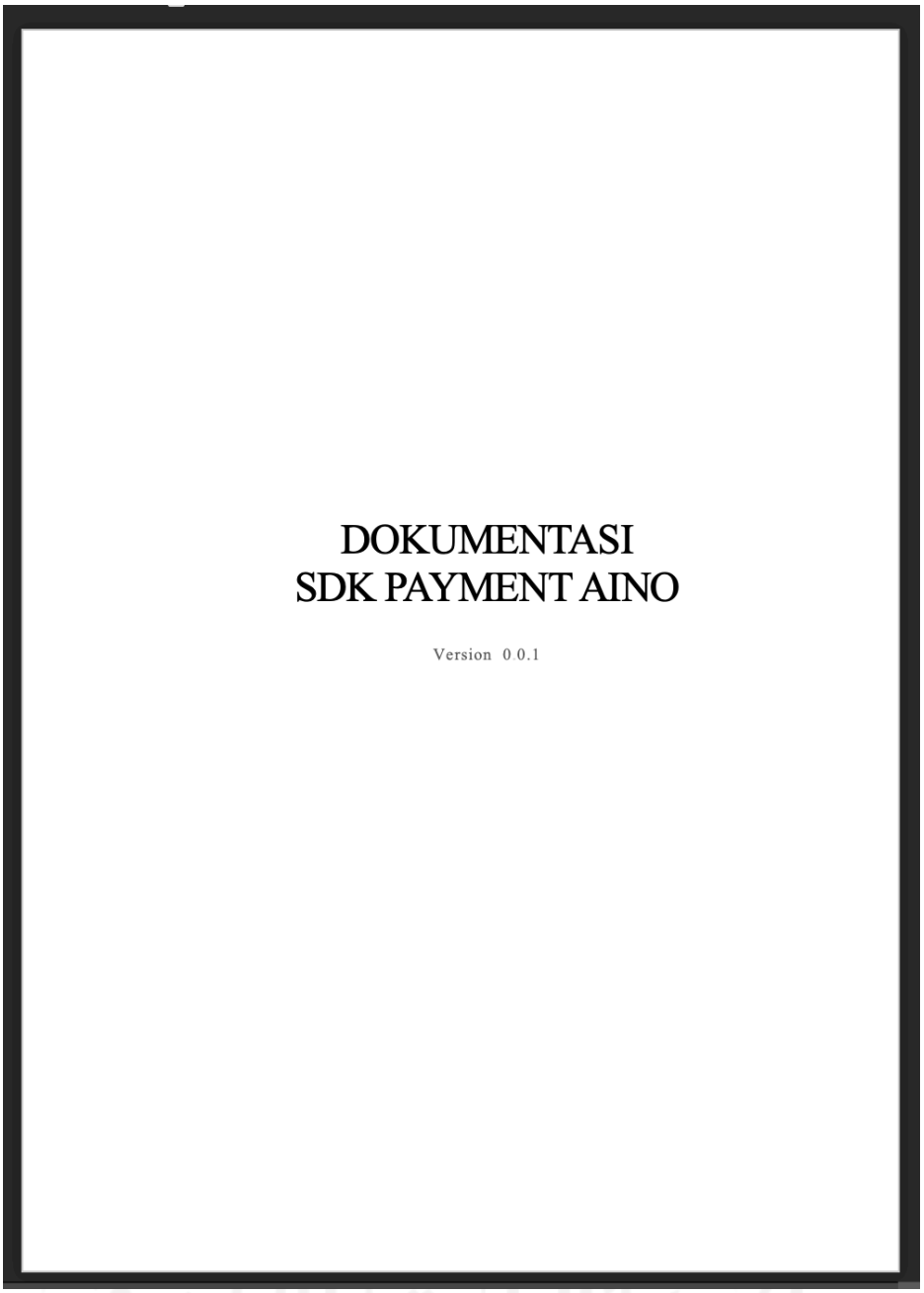
```
fun kickStart() {
    /** Starting the Logger **/
    logger = LoggerFactory.getLogger("TestLogger") as
    ch.qos.logback.classic.Logger
    StatusPrinter.print(loggerContext)
}
```

Gambar 3.19 Kode program *method kickStart*

Setelah semua fitur yang dikembangkan sudah dipastikan berjalan dengan benar dan sesuai dengan kebutuhan, langkah selanjutnya adalah integrasi dengan proyek utama. Kode program ditempatkan pada direktori yang sudah dirancang sebelumnya. Aplikasi yang menggunakan SDK ini dapat mengakses fitur dengan memanggil *method* yang ada pada *interface* fitur tersebut. Apabila suatu saat ada penggantian library yang digunakan, hanya perlu mengganti kelas implementasi dari *interface* tersebut.

3. Penyusunan dokumentasi SDK

Aktivitas yang dilakukan selanjutnya adalah pembuatan dokumentasi teknis. Dokumentasi ini dapat menjadi petunjuk bagi pengembang dalam mengimplementasikan SDK ini ke dalam aplikasi yang dikembangkan. Pengembang dapat mengetahui *method* apa saja yang bisa digunakan beserta cara menggunakan dan parameter apa saja yang dibutuhkan. Dokumentasi ini berisi dari berbagai fitur yang ada pada SDK. Informasi mengenai fitur lain seperti pencetakan nota transaksi dan transaksi *chipbased* diperoleh dari *programmer* lain yang ikut serta dalam pengembangan SDK ini. Gambar 3.20 adalah *cover* dari dokumentasi SDK *payment* baru yang disusun penulis.



Gambar 3.20 *Cover* dokumentasi SDK *payment* baru

3.2.6 Pemantauan dan Pengendalian Proyek

Pemantauan dan pengendalian dilakukan untuk memastikan bahwa pengembangan proyek tetap berada pada jalurnya. Hasil dari proses ini nantinya dapat digunakan sebagai evaluasi dan umpan balik bagi pengembangan berikutnya. Proses pemantauan dan pengendalian proyek ini dibantu dengan beberapa alat, yaitu:

a. Jira

Jira adalah layanan yang dapat digunakan dalam pengelolaan proyek, *bug tracing*, dan *issue tracing*. Aplikasi ini menjadi media utama dalam pengelolaan segala kegiatan yang dilakukan dalam metode Scrum terutama dalam pelaksanaan *Sprint*.

b. GitLab

Seluruh kode program pada pengembangan proyek sudah diintegrasikan dengan sistem pengontrol versi atau VCS. VCS yang digunakan adalah Git menggunakan GitLab. Gitlab adalah layanan penyimpanan atau *repository* Git berbasis web. Penggunaan alat ini memungkinkan kolaborasi antar *developer* dan peninjauan kode program.

c. Microsoft Teams

Microsoft Teams adalah aplikasi video *meeting* atau video *conference* yang dikembangkan oleh Microsoft. Aplikasi memudahkan komunikasi antar pihak dalam sebuah tim terutama yang bekerja dengan pola *work from home*. Seluruh pertemuan yang dilakukan dalam siklus *sprint* dilakukan melalui aplikasi ini.

3.2.7 Penutupan Proyek

Proyek pengembangan fitur dinyatakan selesai ketika sudah memenuhi *definition of done* yang sudah ditentukan sebelumnya. Penulis telah mengembangkan dua proses pada fitur *logging*. Pertama adalah proses *logging* yang dapat mencatat informasi *log* ke dalam *file* dengan konfigurasi nama sesuai dengan tanggal proses, *compress file log* dan konfigurasi *file* hanya akan disimpan di dalam perangkat selama 31 hari. Proses kedua adalah *upload file* hasil proses *logging* ke *server* FTP. Setelah fitur diintegrasikan ke dalam SDK, developer utama yang ada pada segmen *Payment* akan mengambil alih dan menyempurnakan fitur tersebut. Selain itu, penulis juga menyerahkan dokumentasi yang sudah dibuat pada saat *sprint* kedua. Setelah semua tanggung jawab selesai dilaksanakan, penulis dan satu developer lainnya kembali ke segmen AFC dan menyelesaikan pekerjaan pada segmen tersebut.

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Teknis

Dalam pengembangan fitur baru SDK *payment* ini penulis ikut serta dalam sebagian proses pengembangan. Selama penulis ikut serta dalam proses pengembangan tersebut, ada beberapa hal teknis yang dapat diambil dan menjadi refleksi pelaksanaan magang sehingga dapat menjadi pengalaman dan umpan balik untuk proses pengembangan selanjutnya. Hal-hal teknis yang penulis dapatkan adalah sebagai berikut:

4.1.1 Implementasi Scrum

Scrum adalah kerangka kerja yang dapat meningkatkan produktivitas tim pada proses pengembangan perangkat lunak. Scrum mampu meningkatkan kecepatan dan mempersingkat waktu saat pengembangan aplikasi berlangsung. Pada penelitian yang dilakukan oleh Harjono dan Muhammad Hamka, Scrum dapat meningkatkan kecepatan waktu dalam pengembangan perangkat lunak. Hal ini ditunjukkan oleh garis *actual task remaining* yang selalu berada di bawah garis *ideal task remaining* dan rata-rata *focus factor* sebesar 0.6 (Hamka, 2017). Kecepatan waktu yang diperoleh karena Scrum memfokuskan pekerjaan pada suatu fitur atau kebutuhan pokok yang harus diselesaikan terlebih dahulu (Kurniawan & Sani, 2019).

Scrum tepat digunakan pada perangkat lunak yang mudah mengalami perubahan kebutuhan atau fitur. Perubahan-perubahan yang muncul dapat didiskusikan pada saat sesi pertemuan *Sprint* Review. Perubahan tersebut kemudian dibuat menjadi pekerjaan baru yang akan ditempatkan pada Backlog sehingga dapat dikerjakan pada iterasi *Sprint* selanjutnya (Lesmana & Ramdhani, t.t.). Iterasi *Sprint* yang dilakukan pada Scrum dapat membuat pekerjaan yang dilakukan menjadi tidak kaku. *Sprint* memungkinkan setiap anggota pada tim Scrum untuk membahas pekerjaan yang perlu dilakukan, dimodifikasi, diperbaiki, dan ditambahkan berdasarkan evaluasi dan umpan balik dari *Sprint* yang telah dilakukan (Prasetio dkk., 2014). Oleh karena itu, Scrum sangat tepat diterapkan pada pengembangan aplikasi perangkat *smartphone* karena perkembangan teknologi *smartphone* yang semakin cepat dengan kebutuhan yang selalu berubah (Prasetio dkk., 2014).

Scrum memungkinkan sebuah proyek untuk dikembangkan dengan cepat oleh tim yang memiliki anggota sedikit. Proses pengembangan proyek dapat dilakukan dengan cepat karena sistem didesain sederhana dan sesuai dengan kebutuhan dari pengguna sistem tersebut

(Firmansyah, 2017). Jumlah anggota yang sedikit dapat dimaksimalkan apabila seluruh anggota tim memiliki kemampuan dan pemahaman yang baik mengenai implementasi Scrum dalam proses pengembangan. Setiap anggota perlu memahami tugas dan tanggung jawab dari peran dalam sebuah Tim Scrum sehingga tidak terjadi keterlambatan pengerjaan proyek yang sudah direncanakan (Firmansyah, 2017).

Seluruh proses yang dilakukan pada pengembangan SDK menggunakan metode Scrum. Metode Scrum ini digunakan untuk mendapatkan hasil produk yang maksimal . Scrum memastikan bahwa proses pengembangan yang dilakukan tetap pada jalur yang benar dan mudah beradaptasi terhadap perubahan yang terjadi. Pelaksanaan Scrum di PT Aino Indonesia mengikuti petunjuk yang ada pada *Scrum Guide.* Namun terdapat beberapa perbedaan pelaksanaan metode Scrum pada pengembangan proyek SDK ini. Tabel 4.1 menunjukkan perbandingan pelaksanaan *Scrum Events*

Tabel 4.1 Perbandingan pelaksanaan *Scrum Events* di *Scrum Guide* dan PT Aino Indonesia

Scrum Events	**Scrum Guide**	**PT Aino Indonesia**
Sprint Planning	Pertemuan awal untuk memulai iterasi *Sprint* dengan memaparkan tugas-tugas atau *backlog* yang akan dikerjakan dalam *sprint* tersebut.	*Product Owner* dan *system analyst* menguraikan tugas-tugas yang ada di *Backlog* dan menetapkan tugas yang akan dikerjakan pada *Sprint* yang akan berlangsung kepada seluruh anggota tim.
Daily Scrum	Pertemuan harian untuk memeriksa kemajuan pekerjaan untuk mencapai tujuan dari *Sprint* dan adaptasi *backlog* terhadap perubahan yang muncul. Pertemuan ini berlangsung selama kurang lebih lima belas menit.	Pada aktivitas ini, setiap anggota tim memaparkan tentang pekerjaan yang dilakukan pada hari sebelumnya dan yang akan dikerjakan pada hari itu. Anggota tim juga memaparkan mengenai kendala atau *stopper* yang dihadapi saat melakukan pekerjaan. *Daily Scrum* berlangsung selama lima belas hingga tiga puluh menit. Hal ini dikarenakan pembahasan kendala dan koordinasi yang tidak sebentar.
Sprint Review	Tujuan dari *Sprint Review* adalah memeriksa hasil *Sprint* dan adaptasi yang akan dilakukan pada iterasi selanjutnya. Setiap anggota tim akan mendiskusikan serta mempresentasikan hasil pekerjaan dan kemajuan menuju tujuan kepada para *stakeholders* dan anggota tim lainnya.	Pada *Sprint Review*, setiap anggota tim memaparkan pekerjaan yang dilakukan selama *sprint* berlangsung dan pekerjaan apa saja yang perlu di-*carry over* pada *sprint* selanjutnya. Developer akan mendemokan kemajuan aplikasi dan akan diberi tanggapan oleh *Quality Assurance* dan anggota tim lainnya.

Sprint Retrospective	*Sprint Retrospective* memiliki tujuan untuk meningkatkan kualitas dan efektivitas dari pelaksanaan *Sprint*. Tim Scrum memeriksa pelaksanaan *sprint* sebelumnya dan mengidentifikasi perubahan yang paling membantu efektivitas pelaksanaan *sprint*. *Sprint Restrospective* merupakan kesimpulan dari satu iterasi *Sprint*.	*Sprint Retrospective* dilakukan setelah *Sprint* Review dilaksanakan. Pada pertemuan ini *Product Owner* mengevaluasi kinerja tim selama pelaksanaan *Sprint*. Kinerja tim dilihat dari jumlah pekerjaan yang telah selesai dan yang harus dikerjakan pada *sprint* selanjutnya. Setiap anggota tim akan memaparkan keluhan atau kendala yang dihadapi. Dari tinjauan kinerja dan kendala yang dihadapi, tim akan mendiskusikan perubahan dan solusi sehingga *sprint* selanjutnya dapat berjalan lebih baik.

Proses pengembangan fitur *logging* pada SDK ini dilakukan selama dua *sprint* atau empat minggu. Implementasi Scrum secara garis besar telah mengikuti panduan dari Scrum Guide. Terdapat beberapa perbedaan yang dilakukan terkait kondisi tertentu dan kesepakatan bersama antar anggota tim. Perbedaan tersebut seperti pelaksanaan *Daily* Scrum yang terkadang melebihi lima belas menit karena pembahasan tidak hanya mengenai *update* dari pekerjaan yang dilakukan.

Scrum membantu tim merencanakan pekerjaan dengan baik dan matang. Pertemuan *Sprint* Planning membahas setiap pekerjaan anggota tim secara detail. Setiap anggota tim menentukan lingkup dan ukuran pekerjaan sehingga dapat memperkirakan durasi pengerjaan dan sumber daya apa saja yang diperlukan selama *sprint* berlangsung. *Daily* Scrum memastikan bahwa pekerjaan tetap pada jalur yang tepat dan perubahan dapat ditangani dengan cepat. Pertemuan-pertemuan yang pada *Sprint* dapat menjadi wadah bagi setiap anggota tim untuk menyampaikan ide, tanggapan, atau kendala yang dialami.

4.1.2 Pengembangan Fitur *logging* SDK *Payment*

Penggunaan *library* Logback dan FTPClient sangat membantu pada saat mengembangkan fitur *logging* ini. Kedua *library* ini telah menyediakan proses dasar seperti pencatatan log ke *file*, penamaan *file*, dan pengunggahan *file* ke server. Dengan ini, penulis hanya perlu menggunakan proses-proses tersebut dengan memanggil *method* yang sudah disediakan. Karena proses dasar dari fitur ini sudah ada, proses pengembangan dapat difokuskan kepada kebutuhan fitur yang harus dipenuhi. Fitur logging ini merupakan fitur baru yang dikembangkan pada SDK payment. Sehingga penulis perlu mengembangkannya dari awal. Untuk itu penulis perlu mengetahui dan mempelajari penggunaan Logback. Penulis mendapatkan beberapa informasi dari dokumentasi yang terdapat pada website resmi Logback.

Tugas lain yang diberikan kepada penulis pada saat pengembangan SDK ini adalah penyusunan dokumentasi. Selama masa perkuliahan, penulis tidak mendapatkan pembelajaran atau informasi mengenai penyusunan dokumentasi dengan baik dan benar. Sehingga pada saat penyusunan dokumentasi ini, penulis mengikuti contoh dokumentasi yang sudah ada sebelumnya dan mengikuti beberapa format dokumentasi di internet. Dokumentasi yang dibuat meliputi berbagai fitur pada SDK *payment* termasuk fitur yang tidak dikembangkan oleh penulis. Untuk itu, penulis mencari informasi yang akan dituliskan dengan bertanya kepada pengembang lain yang ikut serta mengerjakan fitur baru pada SDK. SDK ini nantinya akan digunakan oleh tim lain atau pihak lain dalam aplikasinya. Dokumentasi ini dapat menjadi petunjuk saat proses implementasi SDK pada aplikasi tersebut. Selain itu dokumentasi ini dapat menjadi petunjuk atau alat *transfer knowledge* untuk pengembangan SDK selanjutnya.

4.2 Non Teknis

Program magang dapat menjadi sarana pembelajaran dan pengenalan pada dunia kerja yang sebenarnya. Manfaat yang didapatkan dari program ini dapat mempersiapkan seseorang untuk terjun ke dunia kerja. Manfaat yang didapatkan dari pelaksanaan program magang di PT Aino Indonesia antara lain:

a. Kemampuan Komunikasi

   Proses pengembangan proyek melibatkan beberapa orang dengan latar belakang dan keahlian yang beragam. Keberagaman ini membutuhkan kemampuan komunikasi yang baik terutama dengan pihak yang memiliki bidang berbeda. Penulis dapat memahami bagaimana perbedaan cara berkomunikasi terhadap rekan satu tim dan karyawan yang memiliki jabatan lebih tinggi. Dengan adanya berbagai pertemuan pada sebuah *sprint* melatih cara berkomunikasi penulis saat menyampaikan *progress*, kendala, dan pendapat sehingga dapat dipahami orang lain dengan tepat.

b. Pengalaman Pekerjaan dan *Working Culture*

   Manfaat utama yang didapatkan dari program magang adalah pengalaman langsung dalam pekerjaan di bidang pengembangan aplikasi. Dengan bekerja secara langsung, penulis mengetahui bagaimana proses pengembangan dilakukan dan hal-hal yang perlu diperhatikan pada saat pengembangan. Selain itu, berada langsung di lingkungan perusahaan peserta magang dapat merasakan *working culture* atau budaya pekerjaan

yang ada. Budaya yang dialami ini seperti interaksi antar pegawai, diskusi penyampaian ide, rutinitas yang diadakan di perusahaan baik yang berurusan dengan pekerjaan atau lainnya.

c. Implementasi Metodologi Pengembangan

Proyek di Aino dikerjakan menggunakan metodologi Agile. *Framework* Agile yang digunakan yaitu Scrum pada segmen *payment* dan Kanban pada segmen AFC. Keterlibatan pada kedua kerangka kerja tersebut memberikan pengalaman langsung yang sebelumnya hanya dipahami secara teorinya saat perkuliahan. Pada perkuliahan, penulis hanya mengerti teorinya saja dan tidak mengerti bagaimana cara implementasi, pembagian tugas-tugas setiap orang, dan manfaat dari Scrum ini. Beberapa pemahaman yang ditemukan selama mengikuti proses *framework* seperti pemahaman tahap-tahap pengembangan aplikasi, komunikasi antar anggota tim dan *stakeholder*, dan tanggung jawab yang dimiliki setiap pihak.

d. Ilmu pemrograman

Beberapa pengetahuan pemrograman yang dipelajari selama perkuliahan terkadang kurang *update* atau kurang sesuai dengan apa yang diminta perusahaan. Perusahaan terkadang meminta menggunakan teknologi yang baru sedangkan pada saat perkuliahan hanya dikenalkan dengan teknologi lama. Maka dari itu, penulis mempelajari beberapa teknologi secara otodidak. Selain itu, proses magang juga memberi beberapa ilmu baru yang didapatkan dari senior di perusahaan dalam bentuk umpan balik dan arahan mengenai bagaimana menulis kode yang baik. Beberapa pengetahuan baru yang dipelajari seperti *framework* aplikasi *mobile* baru, arsitektur perangkat lunak, dan berbagai macam penggunaan *design pattern* pada aplikasi. Penulis juga dikenalkan dengan *clean architecture* yang mengatur berbagai *source code* sedemikian rupa sehingga lebih rapi, mudah untuk di-*maintenance*, dan mudah dipahami oleh developer lainnya.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada periode pengembangan ini, penulis telah mengembangkan fitur *logging* dengan dua proses yaitu pencatatan informasi ke *file* dan mengunggah *file* log yang sudah dibuat ke server FTP. Fitur ini di-*wrap* ke dalam SDK *payment* bersamaan dengan fitur lainnya sehingga dapat diimplementasikan pada aplikasi berbasis Android. Proses pencatatan log pada SDK ini dapat mencatat segala informasi yang dihasilkan pada saat suatu proses dijalankan. Data dan informasi dari proses transaksi tersebut kemudian direkam ke dalam sebuah *file* yang disimpan pada media penyimpanan perangkat. Proses *upload* akan mengunggah *file* log yang pada perangkat ke FTP server yang sudah disediakan. Kedua proses ini dapat mempermudah pelacakan *issue* atau masalah yang timbul pada saat proses transaksi berjalan. Pengembang hanya perlu melihat data transaksi yang sudah direkam pada *file* log untuk mencari sumber masalah tersebut. Dengan adanya fitur *upload*, pihak Aino tidak perlu mengirimkan personel ke lapangan untuk mengambil data dari perangkat. SDK ini menjadi sebuah standar dari proses yang diterapkan pada berbagai aplikasi. Sehingga proses pengembangan aplikasi dapat dilakukan dengan lebih mudah, cepat, dan tidak membuang banyak sumber daya.

Penggunaan Scrum sebagai metode pengembangan membantu developer dalam pengerjaan suatu *task* yang telah direncanakan. Scrum memfokuskan pengembang aplikasi untuk mengerjakan suatu tugas sehingga proses pengembangan tetap terarah dan tidak keluar dari skema yang telah ditentukan. Perubahan yang muncul secara tiba-tiba dapat diimplementasikan dengan aman karena prinsip transparansi, adaptasi, dan inspeksi yang diterapkan pada seluruh proses pengembangan.

5.2 Saran

Proses *rollover* yang dilakukan pada fitur yang dikembangkan hanya dikonfigurasi untuk diakukan per hari. Dan proses kompresi dilakukan per hari pula. Akan lebih baik apabila pengguna SDK ini bisa mengatur seberapa rutin proses *rollover* dilakukan dan kompresi *file* dapat menggabungkan *file* dari periode waktu tertentu seperti mingguan, dua minggu atau per bulan.

File log yang dihasilkan dari fitur logging dapat didunggah ke server menggunakan protokol FTP. Protokol FTP sendiri memiliki beberapa kekurangan dari segi keamanan dan privasi.

Sehingga diperlukan metode lain yang dapat digunakan untuk mengunggah atau menyimpan *file* log dengan fitur keamanan dan privasi yang lebih baik. Protokol SFTP dapat digunakan apabila ingin mengunggah *file* log ke server.

DAFTAR PUSTAKA

Bank for International Settlements. (1996). Implications For Central Banks Of The Development Of Electronic Money. *Bis*, (October), 16.

Bhavsar, K., Shah, V., & Gopalan, S. (2020). Scrum: An Agile Process Reengineering in Software Engineering. *International Journal of Innovative Technology and Exploring Engineering*, *9*(3), 840–848. https://doi.org/10.35940/ijitee.c8545.019320

Bolung, M., & Tampangela, H. R. K. (2017). Analisa Penggunaan Metodologi Pengembangan Perangkat Lunak. *Jurnal ELTIKOM*, *1*(1), 1–10. https://doi.org/10.31961/eltikom.v1i1.1

Cohn, M. (2004). User Role Modeling. Dalam *User Stories Applied for Agile Software Development*.

Davidekova, M., & Gregusml, M. (2016). Software application logging: Aspects to consider by implementing knowledge management. *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, (August), 102–107. https://doi.org/10.1109/OBD.2016.22

Firmansyah. (2017). Pengembangan Enterprise Resource Planning (ERP) dengan Scrum. *Jurnal Sistem Informasi STMIK Antar Bangsa*, *6*(2), 167. https://doi.org/10.51998/jsi.v6i2.174

FTPClient (Apache Commons Net 3.8.0 API). (t.t.). Diambil 20 Januari 2022, dari https://commons.apache.org/proper/commons-net/apidocs/org/apache/commons/net/ftp/FTPClient.html

Fu, Q., Zhu, J., Hu, W., Lou, J. G., Ding, R., Lin, Q., … Xie, T. (2014). Where do developers log? An empirical study on logging practices in industry. *36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings*, 24–33. https://doi.org/10.1145/2591062.2591175

Gonçalves, L. (2018). Scrum: The methodology to become more agile. *Controlling & Management Review*, (4), 40–42.

Gülcü, C., Pennec, S., & Harris, C. (t.t.). Chapter 1: Introduction. Diambil 19 Januari 2022, dari https://logback.qos.ch/manual/introduction.html

Gupta, S. (2003). Introduction to Application Logging. *Logging in Java with the JDK 1.4 Logging API and Apache log4j*, 1–9. https://doi.org/10.1007/978-1-4302-0765-8_1

Hamka, M. (2017). *IMPLEMENTASI FRAMEWORK SCRUM DALAM PENGEMBANGAN SISTEM INFORMASI JABATAN FUNGSIONAL AKADEMIK*. 7.

Kurniawan, I., & Sani, R. R. (2019). Pemodelan SCRUM dalam Pengembangan Sistem Informasi Kesehatan pada Klinik Ar-Rokhim Sragen Kabupaten Sragen. *JOINS (Journal of Information System)*, *4*(1), 76–86. https://doi.org/10.33633/joins.v4i1.2530

Lesmana, K., & Ramdhani, Y. (t.t.). *PERANCANGAN HELPDESK TICKETING DAN PROJECT MANAGEMENT SYSTEM MENGGUNAKAN METODE SCRUM (STUDI KASUS: PT IHSAN SOLUSI INFORMATIKA)*. 8.

Lin, Y., Ha, N.-H., & Lin, K.-S. (2015). The Role of mPOS System in Process Change and Strategy Change: A Situated Change Perspective. *Technologies*, *3*(4), 198–218. https://doi.org/10.3390/technologies3040198

Prasetio, Y. L., Hanafiah, N., Yosanny, A., Yolanda, C., Musbar, F. P., & Septianto, D. (2014). Pengembangan Aplikasi Penjadwalan Wisata Harian pada Smartphone dengan Pendekatan Scrum. *ComTech: Computer, Mathematics and Engineering Applications*, *5*(2), 534. https://doi.org/10.21512/comtech.v5i2.2175

Rasmusson, J. (2010). *The Agile Samurai: How Agile Masters Deliver Great Software* (1st ed.). Pragmatic Bookshelf.

Robert C. Martin. (2018). *Clean Architecture: A Craftman's Guide to SoftwareStructure and Design*. London, England: Prentice Hall.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide*. Diambil dari https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Sutherland, J. (2007). *The Scrum Papers: Nuts , Bolts , and Origins of an Agile Process*.

LAMPIRAN

1. *Task* untuk mengembangkan proses *upload file*

2. Sertifikat magang

AINO
PAYMENT SOLUTION

SERTIFIKAT MAGANG

No: 11404/CHC/SKE/III/2021

Manager Human Capital PT.Aino Indonesia, Alief Rizka Husniawan, menerangkan bahwa:

Nama	: Ardian Dwi Rifai
Tempat, tanggal lahir	: Sleman, 14 Oktober 1998
Jurusan	: Informatika
Nomor Induk Mahasiswa	: 17523093
Universitas	: Universitas Islam Indonesia

Telah melaksanakan Program Magang di:

Departemen	: *Product Department*
Bagian	: *Mobile Developer Internship*
Dari Tanggal	: 01 Oktober 2020
Sampai Tanggal	: 01 April 2021
Dengan predikat	: Sangat Memuaskan

Demikian agar dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 01 April 2021

Manager Human Capital

AINO PAYMENT SOLUTION PT AINO INDONESIA

(Alief Rizka Husniawan)

PT AINO INDONESIA

Head Office
Jl. Prof Dr. Sardjito
No. 25 Yogyakarta 55223
P. +62 274 518682. +62 274 554466

Jakarta Office
UGM Samator Pendidikan Building.
Tower B. 10th Floor Jl. Dr. Saharjo
No. 83 Manggarai, Tebet, Jakarta 12850
P. +62 21 29069515

ainosi.co.id