IMPLEMENTASI CLOUD FIRESTORE TRANSACTION BERBASIS OPTIMISTIC CONCURRENCY CONTROL UNTUK MENCEGAH RACE CONDITION PADA APLIKASI PAYMENT POINT ONLINE BANK

Tugas Akhir

Diajukan oleh :

Fauzul Azmy
220705028

Mahasiswa Fakultas Sains dan Teknologi
Program Studi Teknologi Informasi

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI AR-RANIRY
BANDA ACEH
2026 M / 1448 H

LEMBAR PERSETUJUAN

IMPLEMENTASI CLOUD FIRESTORE TRANSACTION BERBASIS OPTIMISTIC CONCURRENCY CONTROL UNTUK MENCEGAH RACE CONDITION PADA APLIKASI PAYMENT POINT ONLINE BANK

TUGAS AKHIR

Diajukan Kepada Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN) Ar-Raniry Banda Aceh
Sebagai Salah Satu Beban Studi Memperoleh Gelar Sarjana (S1)
Dalam Ilmu Teknologi Informasi

Oleh:
Fauzul Azmy
220705028

Mahasiswa Fakultas Sains dan Teknologi
Program Studi Teknologi Informasi

Disetujui Untuk Diseminarkan Oleh:

Pembimbing I,	Pembimbing II,
Mulkan Fadhli, S.T., M.T	**Aulia Syarif Aziz, S.Kom., M.Si**
NIP. 198811282020121006	**NIP. 199305212022031001**

Mengetahui,
Ketua Program Studi Teknologi Informasi

Malahayati, M.T.
NIP. 198301272015032003

LEMBAR PENGESAHAN

IMPLEMENTASI CLOUD FIRESTORE TRANSACTION BERBASIS OPTIMISTIC CONCURRENCY CONTROL UNTUK MENCEGAH RACE CONDITION PADA APLIKASI PAYMENT POINT ONLINE BANK

TUGAS AKHIR

Telah Diuji oleh Panitia Ujian Munaqasyah Tugas Akhir
Fakultas Sains dan Teknologi UIN Ar-Raniry Banda Aceh dan Dinyatakan Lulus
Serta Diterima Sebagai Salah Satu Beban Studi Program Sarjana (S1)
Dalam Program Studi Teknologi Informasi

Pada Hari/Tanggal: Kamis, 27 Agustus 2026 M
14 Rabiul Awal 1448 H
Di Darussalam, Banda Aceh

Panitia Ujian Munaqasyah Tugas Akhir:

Ketua,	Sekretaris,
Mulkan Fadhli, S.T., M.T.	**Aulia Syarif Aziz, S.Kom., M.Si.**
NIP. 198811282020121006	**NIP. 198607042014031001**
Penguji I,	Penguji II,
Ghufran Ibnu Yasa, M.T.	**Dr. Hendri Ahmadian, M.I.M.**
NIP. 198409262014031005	**NIP. 198301042014031002**

Mengetahui:
Dekan Fakultas Sains dan Teknologi
UIN Ar-Raniry Banda Aceh

Prof. Dr. Ir. Muhammad Dirhamsyah, M.T., IPU
NIP. 196210021988111001

LEMBAR PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama	:	Fauzul Azmy
NIM	:	220705028
Program Studi	:	Teknologi Informasi
Judul	:	IMPLEMENTASI CLOUD FIRESTORE TRANSACTION BERBASIS OPTIMISTIC CONCURRENCY CONTROL UNTUK MENCEGAH RACE CONDITION PADA APLIKASI PAYMENT POINT ONLINE BANK

Dengan ini menyatakan bahwa dalam penulisan tugas akhir ini, saya:

1. Tidak menggunakan ide orang lain tanpa mampu mengembangkan dan mempertanggungjawabkan;
2. Tidak melakukan plagiasi terhadap naskah karya orang lain;
3. Tidak menggunakan karya orang lain tanpa menyebutkan sumber asli atautanpa izin pemilik karya;
4. Tidak memanipulasi dan memalsukan data;
5. Mengerjakan sendiri karya ini dan mampu bertanggungjawab atas karya ini.

Bila dikemudian hari ada tuntutan dari pihak lain atas karya saya, dan telah melalui pembuktian yang dapat dipertanggungjawabkan dan ternyata memang ditemukan bukti bahwa saya telah melanggar pernyataan ini, maka saya siap dikenai sanksi berdasarkan aturan yang berlaku di Fakultas Sains dan Teknologi UIN Ar-Raniry Banda Aceh.

Demikian pernyataan ini saya buat dengan sesungguhnya dan tanpa paksaan dari pihak manapun.

Banda Aceh, 24 Agustus 2026

Yang Menyatakan,

Fauzul Azmy

ABSTRAK

Perkembangan layanan pembayaran digital menuntut aplikasi pembayaran untuk mampu memproses transaksi secara bersamaan dengan tetap menjaga konsistensi dan integritas data. Salah satu permasalahan yang dapat terjadi pada transaksi konkuren adalah *race condition*, yang dapat menyebabkan *lost update*, transaksi ganda, serta ketidaksesuaian saldo pengguna. Sistem dikembangkan dalam bentuk aplikasi mobile yang terintegrasi dengan backend dan layanan pembayaran pihak ketiga. Pengujian fungsional dilakukan menggunakan metode Black Box Testing, sedangkan pengujian konkurensi dilakukan menggunakan dua *real device* yang mengirimkan permintaan secara bersamaan terhadap dokumen saldo pengguna yang sama. Pengujian dilakukan dalam dua kondisi, yaitu tanpa sinkronisasi dan dengan sinkronisasi menggunakan Cloud Firestore Transaction. Pada kondisi tanpa sinkronisasi ditemukan konflik transaksi berupa transaksi ganda, pemotongan saldo yang melebihi kondisi seharusnya, serta kesalahan dalam proses *rollback*. Sementara itu, pada kondisi dengan sinkronisasi, seluruh skenario pengujian berhasil mempertahankan saldo akhir sesuai dengan nilai yang diharapkan serta mencegah *lost update* dan perubahan saldo yang tidak sah. Berdasarkan delapan skenario pengujian, empat skenario tanpa sinkronisasi dinyatakan gagal, sedangkan empat skenario dengan sinkronisasi dinyatakan berhasil berdasarkan kesesuaian antara saldo akhir aktual dan saldo akhir yang diharapkan. Pengujian *idempotency* juga menunjukkan bahwa pengiriman ulang permintaan dengan identitas transaksi yang sama tidak menyebabkan transaksi diproses kembali, sehingga dapat mencegah terjadinya transaksi duplikat. Dengan demikian, penerapan Cloud Firestore Transaction yang menggunakan OCC dan MVCC efektif dalam mencegah *race condition* serta menjaga konsistensi dan integritas data transaksi pada aplikasi PPOB berbasis mobile.

Kata kunci: Cloud Firestore Transaction, Optimistic Concurrency Control, Multi-Version Concurrency Control, *Race Condition*, *Concurrent Programming*, Payment Point Online Bank.

ABSTRACT

The development of digital payment services requires payment applications to process concurrent transactions while maintaining data consistency and integrity. One of the problems that may occur in concurrent transactions is a race condition, which can lead to lost updates, duplicate transactions, and inconsistencies in user balances. The system was developed as a mobile application integrated with a backend and third-party payment services. Functional testing was conducted using the Black Box Testing method, while concurrency testing was performed using two real devices that simultaneously sent requests to the same user balance document. The testing was conducted under two conditions: without synchronization and with synchronization using Cloud Firestore Transaction. Under the unsynchronized condition, transaction conflicts were identified, including duplicate transactions, excessive balance deductions, and errors during the rollback process. In contrast, under the synchronized condition, all test scenarios successfully maintained the final balance according to the expected value and prevented lost updates and unauthorized balance changes. Based on eight test scenarios, four scenarios without synchronization were declared unsuccessful, while four scenarios with synchronization were declared successful based on the consistency between the actual and expected final balances. Idempotency testing also demonstrated that resending a request with the same transaction identity did not cause the transaction to be processed again, thereby preventing duplicate transactions. Therefore, the implementation of Cloud Firestore Transaction using Optimistic Concurrency Control (OCC) and Multi-Version Concurrency Control (MVCC) was effective in preventing race conditions and maintaining the consistency and integrity of transaction data in the mobile-based Payment Point Online Bank (PPOB) application.

Keywords: Cloud Firestore Transaction, Optimistic Concurrency Control, Multi-Version Concurrency Control, *Race Condition*, *Concurrent Programming*, Payment Point Online Bank.

KATA PENGANTAR

Puji syukur ke hadirat Allah Subhanahu wa Ta'ala atas limpahan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi ini yang berjudul "Implementasi Cloud Firestore Transaction Berbasis Optimistic Concurrency Control Untuk Mencegah Race Condition Pada Aplikasi Payment Point Online Bank".

Skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana pada Program Studi Teknologi Informasi. Penelitian ini berfokus pada analisis dan implementasi mekanisme sinkronisasi transaksi untuk mencegah terjadinya *race condition* pada aplikasi pembayaran digital berbasis mobile. Penelitian dilakukan dengan memanfaatkan React Native sebagai framework pengembangan aplikasi, Firebase sebagai layanan backend, serta mekanisme transaksi berbasis *Optimistic Concurrency Control (OCC)* untuk menjaga konsistensi data ketika beberapa transaksi dijalankan secara bersamaan. Hasil penelitian ini diharapkan dapat memberikan solusi dalam meningkatkan keandalan dan konsistensi sistem transaksi digital, khususnya pada aplikasi yang memiliki tingkat akses data secara konkuren (*concurrent access*) yang tinggi.

1. Ibu Malahayati, M.T., selaku Ketua Program Studi Teknologi Informasi Fakultas Sains dan Teknologi Universitas Islam Negeri Ar-Raniry Banda Aceh, telah memberikan arahan kepada penulis selama menjalani perkuliahan.
2. Bapak Mulkan Fadhli, S.T., M.T, selaku Pembimbing I, yang telah meluangkan waktu untuk memberikan bimbingan, arahan, saran, dan masukan selama pelaksanaan penelitian dan penyusunan skripsi ini.
3. Bapak Aulia Syarif Aziz, S.Kom., M.Si., selaku Pembimbing II, yang telah memberikan bimbingan, arahan, serta masukan yang konstruktif kepada penulis selama proses penelitian hingga penyusunan dan penyelesaian skripsi ini.
4. Bapak Nazaruddin Ahmad, M.T., selaku Pembimbing Akademik, yang telah memberikan arahan, nasihat, dan dukungan kepada penulis selama menjalani proses perkuliahan hingga penyelesaian studi.

5. Ibu Cut Ida Rahmadiana, S.Si., selaku staf Program Studi Teknologi Informasi, yang telah memberikan bantuan, pelayanan, dan dukungan dalam berbagai keperluan akademik selama proses perkuliahan hingga penyusunan tugas akhir ini.
6. Seluruh dosen dan staf Program Studi Teknologi Informasi Fakultas Sains dan Teknologi UIN Ar-Raniry Banda Aceh yang telah memberikan ilmu, pengalaman, dan pelayanan akademik kepada penulis selama masa perkuliahan.
7. Kedua orang tua tercinta, almarhum Bapak Manshur dan Ibu Juliar, serta seluruh keluarga yang senantiasa memberikan doa, kasih sayang, dukungan, pengorbanan, dan motivasi kepada penulis.
8. Sahabat, teman dan rekan-rekan mahasiswa Program Studi Teknologi Informasi, khususnya angkatan 2022, yang telah memberikan dukungan, bantuan, dan semangat selama proses perkuliahan dan penyusunan skripsi.
9. Seluruh pihak yang tidak dapat disebutkan satu per satu yang telah membantu penulis, baik secara langsung maupun tidak langsung, dalam menyelesaikan penelitian dan skripsi ini.

Penulis menyadari bahwa skripsi ini masih memiliki banyak kekurangan, baik dari sisi penulisan maupun isi. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan untuk perbaikan di masa yang akan datang. Akhir kata, penulis memohon kepada Allah Swt. agar segala bantuan, dukungan, dan kebaikan yang telah diberikan oleh seluruh pihak mendapatkan balasan yang berlipat ganda. Semoga skripsi ini dapat memberikan manfaat bagi pembaca dan pihak-pihak yang membutuhkan

Banda Aceh, 2026

Fauzul Azmy

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi informasi dan internet telah mendorong transformasi berbagai sektor kehidupan, termasuk pada sistem pembayaran digital. Internet sebagai jaringan komputer global memungkinkan pertukaran informasi dan komunikasi secara cepat sehingga berbagai layanan yang sebelumnya dilakukan secara konvensional kini dapat diakses secara daring (*online*). Perkembangan tersebut mendorong hadirnya berbagai perangkat lunak (*software*) yang mampu memberikan layanan secara *real-time*, termasuk pada bidang keuangan. Melalui pemanfaatan teknologi tersebut, proses transaksi menjadi lebih cepat, efisien, serta dapat dilakukan kapan saja dan di mana saja tanpa dibatasi oleh lokasi maupun waktu.

Salah satu implementasi teknologi pembayaran digital di Indonesia adalah Payment Point Online Bank (PPOB). PPOB merupakan sistem pembayaran yang bekerja sama dengan perbankan untuk melayani berbagai jenis transaksi secara *online* dan *real-time*, seperti pembayaran tagihan listrik, air, BPJS, telekomunikasi, pembelian pulsa, token listrik, serta berbagai layanan digital lainnya. Kehadiran PPOB memberikan kemudahan bagi masyarakat karena berbagai transaksi dapat dilakukan melalui satu platform sehingga transaksi dapat diproses secara lebih akurat dan aman. (Subagio & Muttaqin, 2022)

Meningkatnya penggunaan layanan pembayaran digital menyebabkan aplikasi PPOB harus mampu menangani banyak transaksi yang berlangsung secara bersamaan (*concurrent transactions*). Kondisi tersebut berkaitan dengan konsep concurrent programming, yaitu kemampuan sistem untuk mengeksekusi beberapa proses secara bersamaan agar waktu respons tetap terjaga dan sumber daya dapat dimanfaatkan secara optimal. Penerapan pemrograman konkuren menjadi kebutuhan penting pada sistem pembayaran karena banyak pengguna dapat mengakses dan melakukan transaksi terhadap sumber daya yang sama dalam waktu yang hampir bersamaan.

Di balik peningkatan kinerja tersebut, pemrograman konkuren juga menghadirkan tantangan berupa race condition (Hermawan & Suharso, 2025). Race condition terjadi ketika dua atau lebih proses atau transaksi mengakses serta memodifikasi data yang sama secara bersamaan tanpa mekanisme sinkronisasi yang memadai. Akibatnya, hasil transaksi bergantung pada urutan eksekusi yang tidak dapat diprediksi sehingga dapat menyebabkan *lost update*, inkonsistensi data, bahkan kegagalan transaksi. Pada sistem pembayaran digital, kondisi ini berpotensi mengakibatkan ketidaksesuaian saldo, status transaksi, maupun riwayat transaksi sehingga dapat menimbulkan kerugian finansial dan menurunkan kepercayaan pengguna terhadap sistem.

Permasalahan tersebut berkaitan erat dengan integritas data, yaitu kondisi ketika data tetap akurat, konsisten, dan sesuai dengan keadaan sebenarnya selama proses transaksi berlangsung. Oleh karena itu, sistem pembayaran memerlukan mekanisme yang mampu menjaga konsistensi data meskipun beberapa transaksi dijalankan secara bersamaan. Salah satu pendekatan yang banyak diterapkan pada sistem Basis Data modern adalah Optimistic Concurrency Control (OCC). Berbeda dengan pendekatan berbasis locking, OCC memungkinkan beberapa transaksi berjalan secara bersamaan, kemudian melakukan validasi ketika transaksi akan dikomit (*commit*). Apabila tidak terjadi konflik, transaksi akan berhasil disimpan. Sebaliknya, apabila ditemukan perubahan data oleh transaksi lain, transaksi akan dibatalkan atau dijalankan kembali (*retry*). Pada Cloud Firestore, mekanisme tersebut didukung oleh Multi-Version Concurrency Control (MVCC) yang digunakan untuk mengelola versi data selama proses transaksi berlangsung. Dengan demikian, OCC mampu mempertahankan tingkat konkurensi yang tinggi sekaligus menjaga konsistensi data transaksi.

Berdasarkan permasalahan tersebut, penelitian ini berfokus pada implementasi mekanisme pencegahan *race condition* dengan memanfaatkan transaksi Cloud Firestore yang menerapkan Optimistic Concurrency Control (OCC) dan didukung oleh Multi-Version Concurrency Control (MVCC). Penelitian dilakukan untuk menganalisis kemampuan mekanisme transaksi tersebut dalam menjaga konsistensi dan integritas data ketika beberapa transaksi dijalankan secara

bersamaan sehingga konflik akses terhadap data dapat diminimalkan.

Hasil penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan aplikasi pembayaran digital yang mampu mempertahankan integritas dan konsistensi data pada kondisi transaksi yang bersifat konkuren. Selain itu, penelitian ini diharapkan menjadi referensi bagi pengembang perangkat lunak dalam menerapkan mekanisme sinkronisasi transaksi untuk meminimalkan *race condition*, sehingga sistem pembayaran digital memiliki tingkat keandalan, akurasi, dan konsistensi yang lebih baik.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan di atas, rumusan masalah dalam penelitian ini adalah:

1. Bagaimana implementasi Transaksi Cloud Firestore yang menerapkan Optimistic Concurrency Control (OCC) untuk mencegah terjadinya race condition pada aplikasi Payment Point Online Bank (PPOB) berbasis mobile?
2. Bagaimana pengujian dan evaluasi keberhasilan implementasi sinkronisasi Transaksi Cloud Firestore dalam mencegah race condition pada aplikasi Payment Point Online Bank (PPOB) berbasis mobile?

1.3. Tujuan Penelitian

Berdasarkan rumusan masalah yang telah ditetapkan, maka tujuan yang ingin dicapai dalam penelitian ini adalah:

1. Mengimplementasikan Transaksi Cloud Firestore yang menerapkan Optimistic Concurrency Control (OCC) untuk mencegah terjadinya race condition pada aplikasi Payment Point Online Bank (PPOB) berbasis mobile.
2. Menguji dan mengevaluasi keberhasilan implementasi sinkronisasi Transaksi Cloud Firestore dalam mencegah race condition pada aplikasi Payment Point Online Bank (PPOB) berbasis mobile.

1.4. Manfaat Penelitian

Dengan adanya penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Bagi Penulis

   Penelitian ini dapat menambah wawasan dan pemahaman penulis mengenai implementasi mekanisme transaksi pada Cloud Firestore yang menerapkan *Optimistic Concurrency Control* (OCC) untuk mencegah *race condition* serta menjaga konsistensi dan integritas data pada aplikasi PPOB berbasis mobile.

2. Bagi Pengguna

   Aplikasi yang dikembangkan diharapkan mampu memberikan layanan transaksi digital yang lebih aman, andal, dan efisien. Selain itu, penerapan mekanisme sinkronisasi pada transaksi diharapkan dapat meminimalkan risiko terjadinya *race condition*, menjaga konsistensi data, serta meningkatkan kepercayaan pengguna terhadap sistem.

3. Bagi Pengembang Sistem

   Hasil penelitian ini diharapkan dapat menjadi referensi bagi pengembang aplikasi transaksi keuangan digital dalam memanfaatkan mekanisme transaksi Cloud Firestore yang menerapkan *Optimistic Concurrency Control* (OCC) untuk mencegah *race condition* serta menjaga konsistensi dan integritas data pada sistem.

4. Bagi Institusi / Akademik

   Penelitian ini diharapkan dapat menjadi referensi bagi mahasiswa dan peneliti dalam pengembangan penelitian di bidang sistem pembayaran digital, *concurrent programming*, *Optimistic Concurrency Control* (OCC), serta penerapan mekanisme transaksi pada Basis Data NoSQL berbasis dokumen.

1.5. Batasan Masalah

Agar penelitian ini lebih terarah dan tidak menyimpang dari tujuan yang telah ditetapkan, maka diperlukan batasan masalah sebagai berikut:

1. Penelitian ini dibatasi pada penerapan dan pengujian *race condition* yang berfokus pada permasalahan *lost update* dalam proses pemotongan saldo pengguna secara konkuren.
2. Untuk mempersingkat waktu penelitian, penelitian difokuskan pada mekanisme sinkronisasi transaksi pada proses debit saldo pengguna. Penelitian tidak membahas mekanisme sinkronisasi pada pengisian saldo aplikasi (*top up/deposit*) .
3. Jenis layanan yang disediakan dalam aplikasi dibatasi pada beberapa transaksi umum, seperti pembelian saldo e-wallet dan transfer ke bank.
4. Sistem pembayaran dalam aplikasi memanfaatkan layanan pihak ketiga sebagai penyedia produk dan metode pembayaran digital.
5. Penelitian ini hanya membahas alur proses transaksi dari sisi pengguna hingga sistem backend tanpa mengkaji secara mendalam sistem internal dari pihak ketiga.

BAB II
TINJAUAN PUSTAKA

2.1. Penelitian Terdahulu

Penelitian terdahulu merupakan salah satu landasan yang digunakan untuk mengetahui perkembangan penelitian terkait topik yang dikaji serta mengidentifikasi perbedaan antara penelitian yang telah dilakukan dengan penelitian yang sedang dilakukan. Melalui kajian terhadap penelitian terdahulu, peneliti dapat memahami berbagai metode yang digunakan dalam mengatasi permasalahan *race condition*, mekanisme sinkronisasi, serta penerapan *concurrency control* pada berbagai sistem. Selain itu, penelitian terdahulu juga menjadi acuan dalam menentukan pendekatan yang sesuai untuk mencapai tujuan penelitian.

Tabel 2. 1 Penelitian Terdahulu

Judul & Peneliti	Objek Penelitian	Metode Sinkronisasi	Layer Implementasi
Implementasi Optimistic Concurrency Control pada Sistem Aplikasi E-Commerce Berdasarkan Arsitektur Microservices Menggunakan Kubernetes Ammar (Ammar Dwi Anwari et al., 2021)	Sistem e-commerce berbasis microservices	Optimistic Concurrency Control (OCC) dengan row versioning	Database (DBMS)
Optimalisasi Sinkronisasi Proses: Analisis Efektivitas Semaphore, Mutex, dan Monitor (Fadhila et al., 2025)	Sinkronisasi proses pada sistem operasi	Semaphore, Mutex, dan Monitor	Operating System (OS)
Studi Mengenai Web Race Condition Sebagai Acuan Pembuatan Modul Praktikum (Hana Lolita BLT et al., 2021)	Web race condition	Table-level locking dan Row-level locking	Database (DBMS)

Web-Based Batik Sales System Uses Locking Mechanism to Prevent Race Condition (Wirayuda & Usman, 2025)	Sistem penjualan batik berbasis web	Pessimistic Locking	Database (DBMS)
Implementasi Cloud Firestore Transaction Berbasis Optimistic Concurrency Control Untuk Mencegah Race Condition Pada Aplikasi Payment Point Online Bank	Aplikasi PPOB berbasis mobile	Cloud Firestore Transaction (OCC+MVCC)	Database (DBMS)

Berdasarkan penelitian terdahulu tersebut, dapat disimpulkan bahwa berbagai mekanisme sinkronisasi, seperti Semaphore, Mutex, Monitor, maupun mekanisme *locking*, terbukti mampu mengurangi atau mencegah terjadinya *race condition* pada sistem yang melayani akses secara bersamaan. Namun, sebagian besar penelitian masih berfokus pada sinkronisasi di tingkat sistem operasi atau menggunakan pendekatan *locking* pada sistem Basis Data. Oleh karena itu, penelitian ini berfokus pada implementasi Optimistic Concurrency Control (OCC) melalui mekanisme transaksi sebagai pendekatan sinkronisasi modern untuk mencegah *race condition* serta menjaga konsistensi dan integritas data pada proses transaksi yang berjalan secara konkuren. Penelitian ini tidak hanya mengimplementasikan mekanisme sinkronisasi transaksi, tetapi juga mengevaluasi bagaimana mekanisme tersebut mampu menjaga konsistensi dan integritas data ketika beberapa transaksi mengakses sumber daya yang sama secara bersamaan. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi terhadap pengembangan sistem pembayaran digital yang lebih andal, aman, dan mampu menangani transaksi konkuren tanpa menimbulkan konflik data maupun inkonsistensi transaksi.

2.2. Sistem Pembayaran

Sistem pembayaran merupakan suatu sistem yang terdiri atas seperangkat aturan, lembaga, instrumen, serta mekanisme yang digunakan untuk memfasilitasi proses perpindahan dana dari satu pihak kepada pihak lain dalam rangka memenuhi kewajiban yang timbul dari suatu aktivitas ekonomi. Keberadaan sistem pembayaran tidak dapat dipisahkan dari perkembangan konsep uang sebagai alat tukar (*medium of exchange*) yang berfungsi untuk mempermudah transaksi barang, jasa, maupun aktivitas keuangan. Secara umum, proses dalam sistem pembayaran melibatkan tiga tahapan utama, yaitu otorisasi (authorization) sebagai proses verifikasi dan persetujuan transaksi, kliring (clearing) sebagai proses pertukaran serta pencocokan data transaksi antar pihak yang terlibat, dan penyelesaian akhir (settlement) sebagai tahap pemindahan dana secara final sehingga hak dan kewajiban setiap pihak dinyatakan telah terpenuhi. Berbeda dengan masa lalu Sistem pembayaran mengalami perkembangan yang sangat panjang seiring dengan kemajuan peradaban manusia.

Gambar 2. 1 Evolusi Sistem Pembayaran. Sumber : (Bank Indonesia, 2024)

Pada masa prasejarah, aktivitas ekonomi dilakukan menggunakan sistem barter, yaitu pertukaran barang atau jasa secara langsung tanpa menggunakan alat pembayaran. Sistem ini memiliki berbagai keterbatasan, seperti kesulitan

menemukan pihak yang memiliki kebutuhan yang saling sesuai (*double coincidence of wants*). Untuk mengatasi permasalahan tersebut, masyarakat mulai menggunakan uang komoditas, yaitu barang-barang yang memiliki nilai ekonomi dan diterima secara umum sebagai alat tukar, seperti ternak, garam, maupun hasil pertanian. Di beberapa wilayah, seperti Kepulauan Pasifik, kerang juga digunakan sebagai alat pembayaran karena memiliki nilai dan diterima oleh pengguna. Selanjutnya, perkembangan teknologi metalurgi melahirkan uang logam yang terbuat dari emas, perak, maupun tembaga sehingga transaksi menjadi lebih praktis, memiliki nilai yang lebih stabil, dan mudah dibawa.

Seiring berkembangnya sistem ekonomi dan pemerintahan, muncul konsep legal tender atau alat pembayaran yang sah yang diterbitkan dan dijamin oleh otoritas pemerintah. Setelah itu, beberapa peradaban, seperti Tiongkok, mulai memperkenalkan uang kulit dan kemudian uang kertas sebagai pengganti uang logam untuk mempermudah transaksi dalam jumlah besar. Perkembangan tersebut terus berlanjut hingga lahirnya uang kertas yang diterbitkan oleh bank sentral, yang menjadi dasar sistem moneter modern dan digunakan secara luas hingga saat ini sebagai alat pembayaran resmi di berbagai negara.

Memasuki era modern, perkembangan teknologi informasi membawa perubahan besar pada sistem pembayaran. Transaksi tidak lagi bergantung pada uang fisik, tetapi mulai memanfaatkan sistem elektronik melalui transfer dana elektronik, kartu kredit, Anjungan Tunai Mandiri (ATM), serta Electronic Funds Transfer at Point of Sale (EFTPOS). Selanjutnya, perkembangan jaringan internet melahirkan internet banking dan mobile banking yang memungkinkan transaksi dilakukan secara daring melalui komputer maupun telepon pintar. Inovasi terus berlanjut dengan hadirnya contactless card, uang elektronik (electronic money), hingga aset kripto seperti Bitcoin yang memanfaatkan teknologi blockchain sebagai media pertukaran digital. Perkembangan tersebut menunjukkan bahwa sistem pembayaran terus berevolusi menuju mekanisme yang semakin cepat, efisien, aman, dan mudah diakses oleh pengguna.

Berdasarkan mekanisme penyelesaiannya, sistem pembayaran secara umum dapat diklasifikasikan menjadi dua jenis, yaitu sistem pembayaran tunai dan sistem pembayaran non-tunai. Klasifikasi ini menjadi dasar dalam memahami

perkembangan teknologi pembayaran, karena sistem pembayaran non-tunai kemudian berkembang menjadi berbagai layanan pembayaran digital yang banyak digunakan saat ini.

2.2.1. Pembayaran Tunai

Pembayaran tunai merupakan metode pembayaran yang menggunakan uang kartal, yaitu uang logam dan uang kertas yang diterbitkan oleh bank sentral sebagai alat pembayaran yang sah. Pada sistem ini, proses pembayaran dilakukan secara langsung melalui penyerahan uang fisik dari pihak pembayar kepada pihak penerima tanpa melibatkan perantara maupun sistem elektronik. Pembayaran tunai merupakan bentuk sistem pembayaran yang paling awal digunakan dalam perkembangan aktivitas ekonomi dan hingga saat ini masih banyak dimanfaatkan dalam transaksi sehari-hari, terutama untuk transaksi bernilai kecil maupun di wilayah yang belum memiliki infrastruktur pembayaran digital yang memadai. Keunggulan pembayaran tunai terletak pada kemudahan penggunaannya, tidak memerlukan jaringan komunikasi atau perangkat elektronik, serta memungkinkan transaksi diselesaikan secara langsung. Namun, sistem pembayaran tunai juga memiliki beberapa keterbatasan, seperti risiko kehilangan atau pencurian uang, kesulitan pada pengembalian uang, serta kurang efisien untuk transaksi dalam jumlah besar maupun transaksi jarak jauh.

2.2.2. Pembayaran Non-Tunai

Pembayaran non-tunai merupakan metode pembayaran yang dilakukan tanpa menggunakan uang fisik sebagai alat transaksi. Pada sistem ini, perpindahan dana dilakukan melalui instrumen pembayaran yang diterbitkan oleh bank maupun penyedia jasa pembayaran dengan memanfaatkan infrastruktur perbankan dan teknologi informasi. Dibandingkan pembayaran tunai, pembayaran non-tunai menawarkan berbagai keunggulan, seperti proses transaksi yang lebih cepat, aman, efisien, serta mampu mendukung pencatatan transaksi secara otomatis. Perkembangan teknologi digital juga mendorong munculnya berbagai

layanan pembayaran yang dapat diakses melalui jaringan internet maupun perangkat bergerak (mobile), sehingga transaksi dapat dilakukan kapan saja dan di mana saja. Saat ini, sistem pembayaran non-tunai telah menjadi bagian penting dalam aktivitas ekonomi modern karena mampu meningkatkan efisiensi transaksi sekaligus mendukung perkembangan ekonomi digital. Sistem pembayaran non-tunai memiliki berbagai jenis instrumen yang digunakan sesuai dengan kebutuhan pengguna, di antaranya sebagai berikut.

A. Kartu Debit

Kartu debit merupakan alat pembayaran non-tunai yang diterbitkan oleh bank dan terhubung langsung dengan rekening tabungan nasabah. Setiap transaksi yang dilakukan menggunakan kartu debit akan secara otomatis mengurangi saldo rekening sesuai nominal transaksi. Selain digunakan untuk pembayaran di merchant melalui mesin Electronic Data Capture (EDC), kartu debit juga dapat digunakan untuk penarikan tunai melalui Anjungan Tunai Mandiri (ATM) maupun transaksi pembayaran secara daring.

B. Kartu Kredit

Kartu kredit merupakan instrumen pembayaran non-tunai yang menggunakan mekanisme pinjaman dari bank penerbit kepada pemegang kartu. Pengguna dapat melakukan pembayaran terlebih dahulu, kemudian melunasi tagihan sesuai periode yang telah ditentukan. Selain memberikan kemudahan dalam bertransaksi, kartu kredit juga sering menawarkan berbagai fasilitas seperti cicilan, cashback, reward point, maupun promo belanja. Namun, penggunaan kartu kredit harus dilakukan secara bijaksana karena adanya bunga dan biaya apabila pembayaran tagihan tidak dilakukan tepat waktu.

C. Transfer Bank

Transfer bank merupakan mekanisme pemindahan dana dari satu rekening ke rekening lainnya melalui jaringan perbankan. Proses transfer dapat dilakukan antar rekening dalam bank yang sama maupun antar bank yang berbeda. Saat ini transfer dana dapat

dilakukan melalui ATM, mobile banking, internet banking, maupun layanan transfer cepat yang disediakan oleh sistem pembayaran nasional. Metode ini banyak digunakan karena memiliki tingkat keamanan yang tinggi serta mampu memproses transaksi secara cepat.

D. Internet Banking

Internet banking merupakan layanan perbankan berbasis web yang memungkinkan nasabah melakukan berbagai transaksi melalui jaringan internet menggunakan komputer maupun perangkat bergerak. Layanan ini menyediakan berbagai fitur, seperti transfer dana, pembayaran tagihan, pembelian produk digital, hingga pengelolaan rekening tanpa harus mengunjungi kantor cabang bank.

E. Mobile Banking

Mobile banking merupakan layanan perbankan yang disediakan dalam bentuk aplikasi pada perangkat smartphone. Melalui mobile banking, nasabah dapat melakukan berbagai transaksi seperti transfer dana, pembayaran tagihan, pembelian pulsa, pembayaran menggunakan QRIS, hingga pengecekan saldo secara real-time. Kemudahan akses melalui perangkat mobile menjadikan layanan ini salah satu metode pembayaran non-tunai yang paling banyak digunakan saat ini.

F. Uang Elektronik (E-Money/E-Wallet)

Uang elektronik merupakan alat pembayaran yang nilai dananya disimpan secara elektronik pada media tertentu, baik berbentuk kartu (chip based) maupun aplikasi (server based). Pengguna terlebih dahulu melakukan pengisian saldo (*top-up*) sebelum dapat melakukan transaksi. Perkembangan teknologi juga melahirkan dompet digital (*electronic wallet* atau *e-wallet*) seperti DANA, GoPay, OVO, dan ShopeePay yang memungkinkan pembayaran dilakukan secara cepat melalui aplikasi smartphone.

G. QRIS (Quick Response Code Indonesian Standard)

QRIS merupakan standar nasional pembayaran menggunakan kode QR yang ditetapkan oleh Bank Indonesia. Melalui QRIS,

pengguna cukup melakukan pemindaian (scan) terhadap satu kode QR menggunakan aplikasi pembayaran yang mendukung QRIS, seperti mobile banking maupun dompet digital. Standarisasi ini memungkinkan interoperabilitas antar penyedia jasa pembayaran sehingga satu kode QR dapat digunakan oleh berbagai aplikasi pembayaran.

H. Payment Point Online Bank

Payment Point Online Bank (PPOB) adalah sebuah sistem layanan pembayaran yang memungkinkan pengguna untuk melakukan berbagai jenis pembayaran seperti tagihan listrik, air, telepon, BPJS, dan lain-lain melalui jaringan perbankan secara online dan terpusat. Dalam PPOB, semua transaksi diproses secara digital dan terhubung langsung ke sistem bank atau penyedia layanan (provider), sehingga transaksi lebih cepat, akurat, dan dapat dilakukan kapan saja. PPOB mengintegrasikan sistem keuangan dengan teknologi informasi agar transaksi pembayaran dapat diakses dengan mudah oleh pengguna, termasuk melalui agen atau aplikasi berbasis mobile. Alur kerja PPOB secara umum melibatkan beberapa komponen utama:

2.3. Software

Aplikasi (software) adalah sekumpulan perintah atau program komputer yang dirancang untuk menjalankan fungsi tertentu pada perangkat pengguna. Dalam konteks mobile, aplikasi merujuk pada perangkat lunak yang dikembangkan untuk berjalan pada perangkat seluler seperti smartphone dan tablet. Aplikasi mobile dirancang agar dapat digunakan secara langsung oleh pengguna untuk menyelesaikan berbagai tugas, mulai dari komunikasi, hiburan, pendidikan, hingga transaksi keuangan. Aplikasi mobile memiliki kelebihan dalam hal portabilitas, kemudahan akses, dan kemampuan untuk memanfaatkan fitur perangkat seperti kamera, GPS, dan notifikasi.

Dalam konteks sistem pembayaran dan layanan keuangan, aplikasi mobile berperan sebagai media utama untuk memfasilitasi transaksi digital. Aplikasi

semacam ini biasa dikenal sebagai e-wallet, mobile banking, atau aplikasi PPOB (Payment Point Online Bank). Melalui aplikasi, pengguna dapat melakukan berbagai transaksi seperti pembayaran tagihan listrik, pembelian pulsa, transfer dana, hingga pembayaran via QR code. Dengan dukungan teknologi seperti payment gateway, API, dan sistem keamanan yang kuat, aplikasi mobile mampu menyediakan layanan keuangan yang cepat, aman, dan mudah diakses kapan saja. Peran aplikasi dalam layanan keuangan juga sangat penting dalam mendorong inklusi keuangan, terutama bagi pengguna yang belum terjangkau layanan perbankan konvensional.

2.4. Application Programming Interface

Application Programming Interface (API) adalah sekumpulan aturan, protokol, dan alat yang memungkinkan satu sistem atau aplikasi berkomunikasi dan bertukar data dengan sistem lain. Dalam konteks teknologi, API berfungsi sebagai jembatan penghubung antara aplikasi frontend (seperti aplikasi mobile) dengan sistem backend, database, atau layanan pihak ketiga. Contohnya, saat pengguna melakukan pembayaran di aplikasi PPOB, API digunakan untuk mengirim permintaan pembayaran ke server, lalu menerima respons dan menampilkan status transaksi kepada pengguna melalui callback menuju ke webhook. Terdapat beberapa jenis API yang umum digunakan dalam pengembangan aplikasi:

A. RESTful API (Representational State Transfer)

   Merupakan API berbasis HTTP yang menggunakan metode seperti GET, POST, PUT, DELETE. REST adalah jenis API yang paling banyak digunakan karena sederhana dan mudah diimplementasikan. Contoh: mengambil data pengguna, mengirim data transaksi.

B. WebSocket

   Merupakan protokol komunikasi dua arah yang berjalan secara real-time dan selalu terbuka, cocok untuk aplikasi yang membutuhkan komunikasi langsung tanpa jeda, seperti chat, notifikasi instan, atau dashboard transaksi langsung.

C. GraphQL

   API yang dikembangkan oleh Facebook, memungkinkan klien meminta

data secara fleksibel dan efisien. Cocok digunakan ketika kebutuhan data sangat dinamis atau kompleks. Dalam aplikasi PPOB, RESTful API paling umum digunakan karena cocok untuk komunikasi antara aplikasi mobile dengan server pembayaran dan database.

Application Programming Interface (API) bekerja melalui mekanisme pertukaran data antara dua sistem yang saling terhubung. Dalam proses tersebut terdapat beberapa komponen penting, yaitu request, response, callback, dan webhook. Keempat komponen ini berperan dalam memastikan informasi transaksi dapat dikirim, diproses, dan diterima secara akurat oleh masing-masing sistem yang terlibat.

A. Request

Request merupakan permintaan yang dikirimkan oleh suatu aplikasi kepada server atau layanan lain untuk melakukan suatu proses tertentu. Pada aplikasi PPOB, request biasanya dikirim ketika pengguna melakukan transaksi seperti pembelian pulsa, pembayaran tagihan, atau transfer dana. Data yang dikirim dalam request dapat berupa identitas pengguna, nomor tujuan, nominal transaksi, serta informasi pendukung lainnya yang diperlukan untuk memproses transaksi.

B. Response

Response merupakan balasan yang diberikan oleh server setelah menerima dan memproses request. Informasi yang dikirimkan dalam response umumnya berisi status keberhasilan atau kegagalan transaksi, kode respons, pesan sistem, serta data tambahan yang diperlukan aplikasi. Melalui response, aplikasi dapat mengetahui apakah permintaan yang dikirim berhasil diproses atau memerlukan tindakan lanjutan.

C. Callback

Callback merupakan mekanisme pengiriman informasi dari server kepada sistem pengirim setelah suatu proses selesai dilakukan. Pada transaksi keuangan digital, callback digunakan untuk mengirimkan hasil akhir transaksi yang sebelumnya diproses secara asynchronous. Sebagai contoh, ketika pengguna melakukan pembayaran, server penyedia layanan dapat mengirimkan callback kepada server aplikasi untuk menginformasikan

bahwa transaksi telah berhasil, gagal, atau masih dalam proses.

D. Webhook

Webhook merupakan sebuah endpoint atau URL yang disediakan oleh suatu sistem untuk menerima callback dari sistem lain. Ketika terjadi perubahan status transaksi, penyedia layanan akan mengirimkan data ke URL webhook yang telah didaftarkan sebelumnya. Dengan adanya webhook, aplikasi dapat menerima pembaruan status transaksi secara otomatis tanpa harus melakukan pengecekan berulang (polling) ke server penyedia layanan.

Dalam implementasi aplikasi PPOB, alur komunikasi API umumnya dimulai dari pengiriman request oleh aplikasi kepada server. Setelah request diterima, server akan memproses transaksi dan mengembalikan response awal kepada aplikasi. Selanjutnya, apabila proses transaksi memerlukan waktu tertentu, penyedia layanan dapat mengirimkan callback ke webhook yang dimiliki sistem untuk memberikan informasi status akhir transaksi. Mekanisme ini memungkinkan aplikasi memperoleh informasi transaksi secara real-time, meningkatkan efisiensi komunikasi data, serta mengurangi jumlah permintaan yang tidak diperlukan ke server.

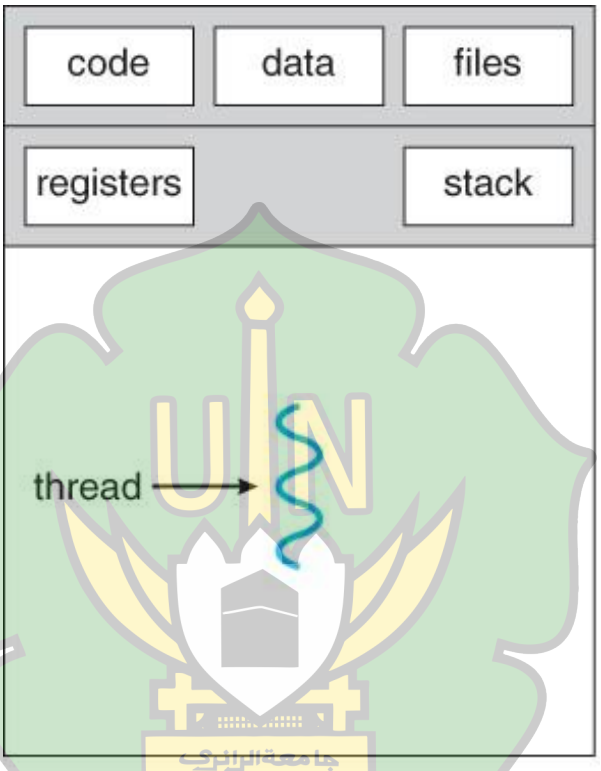
2.5. Thread

Thread merupakan unit eksekusi terkecil yang dapat dijadwalkan oleh sistem operasi. Dalam satu *process* dapat terdapat satu atau lebih *thread* yang berjalan secara konkuren. Setiap *thread* memiliki *program counter*, *register*, dan *stack* sendiri, namun berbagi kode program, data, dan sumber daya lainnya dengan *thread* lain dalam *process* yang sama. Pendekatan ini memungkinkan peningkatan efisiensi dan responsivitas aplikasi, tetapi juga berpotensi menimbulkan masalah sinkronisasi seperti *race condition* ketika beberapa *thread* mengakses sumber daya yang sama secara bersamaan.

2.5.1. Singlethreading

Singlethreading merupakan model eksekusi program yang hanya menggunakan satu thread untuk menjalankan seluruh instruksi secara berurutan (*sequential execution*). Pada model ini, setiap proses atau tugas akan dieksekusi satu

per satu sesuai urutan program, sehingga suatu tugas harus selesai terlebih dahulu sebelum tugas berikutnya dapat dijalankan. Namun, model ini memiliki keterbatasan dalam memanfaatkan sumber daya prosesor secara optimal, terutama pada sistem yang harus menangani banyak pekerjaan secara bersamaan, sehingga dapat menyebabkan penurunan responsivitas dan kinerja aplikasi.



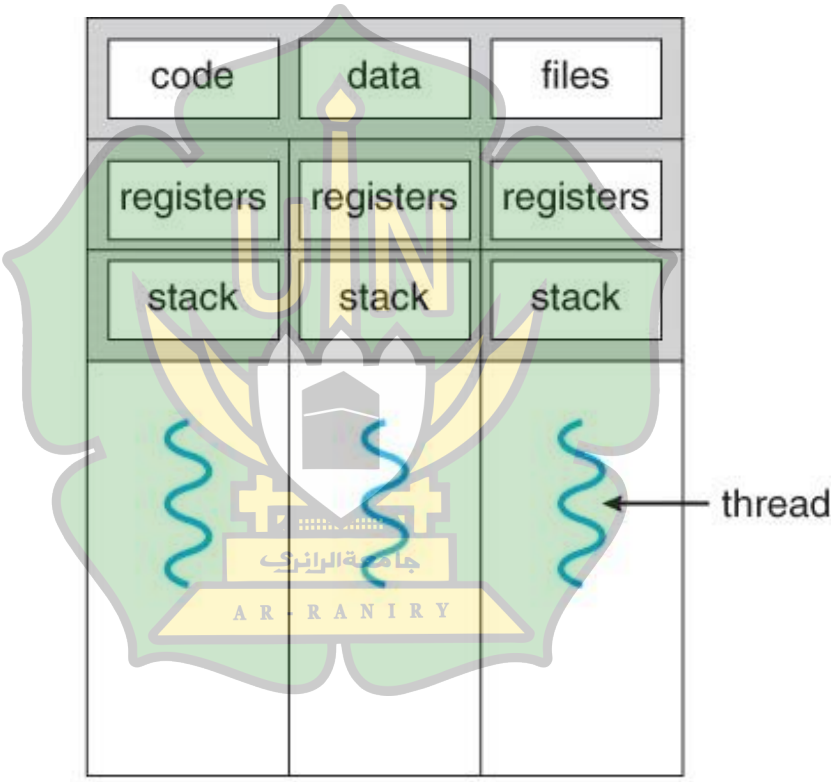
Gambar 2. 2 Singlethread

komponen Code, Data, dan Files yang merupakan sumber daya milik *process*. Bagian Code berisi instruksi program yang akan dieksekusi, Data berisi variabel dan data yang digunakan aplikasi, sedangkan Files merepresentasikan berkas atau sumber daya eksternal yang sedang diakses oleh program. Di bawahnya terdapat Registers dan Stack yang dimiliki oleh *thread*. *Registers* digunakan untuk menyimpan informasi sementara yang dibutuhkan CPU saat menjalankan instruksi, sedangkan *Stack* digunakan untuk menyimpan parameter fungsi, variabel lokal, serta alamat pengembalian (*return address*) selama proses eksekusi program. Garis bergelombang yang ditunjukkan pada gambar merepresentasikan satu alur eksekusi

(*thread of execution*) yang menjalankan seluruh instruksi program secara berurutan. Karena hanya terdapat satu *thread*, maka seluruh pekerjaan dalam aplikasi harus dieksekusi secara bergantian dalam satu jalur eksekusi.

2.5.2. Multithreading

Multithread adalah teknik dalam pemrograman dan arsitektur komputer yang membagi sebuah proses atau program menjadi beberapa unit eksekusi yang lebih kecil (disebut thread). Unit-unit ini dapat diproses secara bersamaan atau paralel oleh CPU, sehingga tugas yang kompleks bisa diselesaikan dengan jauh lebih cepat dan efisien.



Gambar 2. 3 Multi Thread

seluruh thread berbagi sumber daya yang sama berupa Code, Data, dan Files. Artinya setiap thread dapat mengakses kode program, variabel, serta berkas yang sama dalam satu process. Namun, masing-masing thread memiliki Registers dan Stack sendiri sehingga setiap thread dapat menjalankan instruksi secara independen tanpa mengganggu konteks eksekusi thread lainnya.

2.6. Concurrent Programming

Concurrent Programming atau pemrograman konkuren merupakan paradigma pemrograman yang memungkinkan beberapa proses, tugas (*task*), atau alur eksekusi (*thread*) berjalan secara bersamaan dalam suatu sistem komputer. Tujuan utama dari pemrograman konkuren adalah meningkatkan efisiensi penggunaan sumber daya sistem, meningkatkan responsivitas aplikasi, serta memungkinkan sistem menangani banyak permintaan (*request*) dalam waktu yang bersamaan. pemrograman konkuren digunakan untuk mengelola beberapa aktivitas yang berjalan secara simultan sehingga aplikasi mampu memberikan kinerja yang lebih baik dibandingkan pemrograman sekuensial yang menjalankan proses satu per satu.

Dalam sistem informasi modern, khususnya aplikasi berbasis internet dan transaksi digital, konsep *concurrent programming* menjadi sangat penting karena sistem harus mampu melayani banyak pengguna secara bersamaan. Pada aplikasi Payment Point Online Bank (PPOB), misalnya, pengguna dapat melakukan berbagai transaksi seperti pembelian pulsa, pembayaran tagihan, maupun transfer dana pada waktu yang hampir bersamaan. Setiap transaksi tersebut akan menghasilkan permintaan ke sistem yang harus diproses secara cepat dan akurat tanpa mengganggu transaksi lainnya.

Meskipun memberikan keuntungan dalam hal efisiensi dan performa, pemrograman konkuren juga menimbulkan berbagai tantangan. Salah satu permasalahan yang sering muncul adalah ketika beberapa proses mengakses dan memodifikasi sumber daya yang sama (*shared resource*) secara bersamaan. Kondisi ini dapat menyebabkan terjadinya inkonsistensi data apabila tidak terdapat mekanisme pengendalian yang memadai. Sebagai contoh, dua transaksi yang mengakses saldo pengguna pada waktu yang bersamaan berpotensi menghasilkan nilai saldo yang tidak sesuai apabila proses pembaruan data dilakukan tanpa sinkronisasi.

Untuk mengatasi permasalahan tersebut, sistem umumnya menerapkan berbagai mekanisme pengendalian konkurensi (*concurrency control*), seperti penguncian data (*locking*), transaksi atomik (*atomic transaction*), sinkronisasi proses (*synchronization*), serta validasi status transaksi. Mekanisme tersebut

bertujuan untuk memastikan bahwa setiap proses yang berjalan secara bersamaan tetap dapat mengakses dan memodifikasi data secara aman tanpa menyebabkan kerusakan atau ketidaksesuaian data.

Dalam konteks aplikasi PPOB berbasis mobile, pemrograman konkuren memiliki hubungan yang erat dengan integritas data dan keamanan transaksi. Banyaknya transaksi yang terjadi secara bersamaan dapat meningkatkan risiko terjadinya *race condition*, yaitu kondisi ketika hasil akhir suatu proses bergantung pada urutan eksekusi yang tidak dapat diprediksi. Oleh karena itu, pemahaman mengenai konsep *concurrent programming* menjadi dasar penting dalam merancang sistem yang mampu menangani transaksi keuangan digital secara aman, andal, dan konsisten.

2.7. Integritas Data

Integritas data (*data integrity*) merupakan kondisi di mana data tetap akurat, konsisten, lengkap, dan dapat dipercaya selama proses penyimpanan, pemrosesan, maupun pertukaran data dalam suatu sistem informasi. Integritas data menjadi salah satu aspek penting dalam pengelolaan sistem informasi karena berperan dalam memastikan bahwa informasi yang digunakan oleh sistem maupun pengguna sesuai dengan kondisi sebenarnya dan tidak mengalami perubahan yang tidak sah.

integritas data adalah mekanisme yang digunakan untuk menjaga keakuratan dan konsistensi data selama siklus hidup data. Integritas data memastikan bahwa data yang tersimpan dalam Basis Data tetap valid, tidak mengalami kerusakan, dan hanya dapat dimodifikasi melalui prosedur yang telah ditentukan. Dalam sistem transaksi keuangan digital, integritas data memiliki peran yang sangat penting karena setiap kesalahan atau ketidaksesuaian data dapat menimbulkan kerugian finansial serta menurunkan kepercayaan pengguna terhadap sistem.

Pada aplikasi Payment Point Online Bank (PPOB), integritas data berkaitan dengan keakuratan informasi saldo pengguna, status transaksi, riwayat transaksi, serta data keuangan lainnya. Setiap transaksi yang dilakukan harus menghasilkan perubahan data yang sesuai dengan kondisi sebenarnya. Sebagai contoh, ketika pengguna melakukan transfer dana atau pembelian produk digital, sistem harus

memastikan bahwa saldo berkurang sesuai nominal transaksi dan status transaksi tercatat dengan benar. Apabila terjadi ketidaksesuaian antara data yang tersimpan dengan kondisi transaksi yang sebenarnya, maka integritas data dapat dikatakan terganggu.

Integritas data umumnya mencakup beberapa aspek utama, yaitu akurasi (*accuracy*), konsistensi (*consistency*), kelengkapan (*completeness*), dan validitas (*validity*). Akurasi menunjukkan bahwa data yang tersimpan sesuai dengan fakta atau kondisi sebenarnya. Konsistensi memastikan bahwa data memiliki nilai yang sama pada seluruh bagian sistem yang saling berhubungan. Kelengkapan menunjukkan bahwa seluruh informasi yang diperlukan tersedia tanpa adanya data yang hilang. Sementara itu, validitas memastikan bahwa data yang tersimpan telah memenuhi aturan dan ketentuan yang berlaku dalam sistem.

Dalam sistem transaksi digital, berbagai faktor dapat mengancam integritas data, seperti kesalahan pemrosesan, kegagalan jaringan, manipulasi data, transaksi duplikat, maupun kondisi balapan (*race condition*). Race condition terjadi ketika dua atau lebih proses mengakses dan memodifikasi data yang sama secara bersamaan tanpa mekanisme pengendalian yang memadai. Kondisi tersebut dapat menyebabkan ketidaksesuaian data, seperti saldo yang tidak sesuai, transaksi tercatat lebih dari satu kali, atau perubahan data yang saling menimpa (*lost update*).

Untuk menjaga integritas data, sistem perlu menerapkan berbagai mekanisme pengendalian, seperti validasi transaksi, pengendalian akses data (*access control*), manajemen transaksi (*transaction management*), serta mekanisme sinkronisasi data. Selain itu, penerapan prinsip atomisitas (*atomicity*) pada transaksi juga diperlukan untuk memastikan bahwa seluruh proses transaksi berhasil dijalankan secara lengkap atau dibatalkan sepenuhnya apabila terjadi kegagalan. Dengan demikian, data yang tersimpan akan tetap konsisten dan sesuai dengan kondisi yang sebenarnya.

2.8. Race Condition

Race condition merupakan kondisi yang terjadi ketika dua atau lebih proses, *thread*, atau permintaan (*request*) mengakses dan memodifikasi sumber daya (*shared resource*) yang sama secara bersamaan tanpa mekanisme sinkronisasi yang

memadai. Akibatnya, hasil akhir yang diperoleh sistem bergantung pada urutan eksekusi proses yang tidak dapat diprediksi (*non-deterministic*). Kondisi ini dapat menyebabkan inkonsistensi data, kehilangan pembaruan data (*lost update*), duplikasi transaksi, maupun kesalahan logika yang berpotensi merugikan pengguna dan penyedia layanan sistem.

Race condition adalah salah satu permasalahan utama pada sistem paralel dan terdistribusi. Race condition terjadi ketika program dieksekusi, lokasi memori yang sama diakses 2 kali pada waktu yang sama. Race condition dapat terjadi pada sistem web karena adanya delay pada pemrosesan asynchronous (tidak terjadi bersamaan) dan waktu transfer pada jaringan, sementara perilaku sistem dapat berubah-ubah tergantung pada urutan eksekusi dari proses yang bekerja bersamaan.

Dalam klasifikasi keamanan perangkat lunak yang dikeluarkan oleh MITRE melalui Common Weakness Enumeration (CWE), *race condition* dikategorikan sebagai CWE-362 (*Concurrent Execution using Shared Resource with Improper Synchronization*). Kerentanan ini terjadi ketika beberapa proses mengakses sumber daya yang sama secara bersamaan tanpa sinkronisasi yang memadai sehingga dapat mengakibatkan kerusakan data, pelanggaran integritas data, maupun gangguan terhadap ketersediaan sistem. (Hana Lolita BLT et al., 2021)

Pada aplikasi transaksi keuangan digital, *race condition* dapat terjadi ketika beberapa transaksi mencoba mengakses dan memperbarui data yang sama pada waktu yang hampir bersamaan. Sebagai contoh, apabila dua transaksi melakukan pengurangan saldo terhadap akun yang sama secara bersamaan, masing-masing transaksi dapat membaca nilai saldo yang sama sebelum salah satu transaksi selesai memperbarui data. Kondisi tersebut dapat menyebabkan saldo akhir yang tersimpan tidak sesuai dengan kondisi sebenarnya. Permasalahan ini dikenal sebagai *lost update*, yaitu ketika hasil pembaruan dari suatu proses tertimpa oleh pembaruan dari proses lain.

Selain *lost update*, *race condition* juga dapat menimbulkan masalah berupa transaksi ganda (*duplicate transaction*), pemrosesan data yang tidak konsisten, serta ketidaksesuaian status transaksi. Pada sistem PPOB yang menangani transaksi pembayaran, transfer dana, dan pembelian produk digital secara real-time, risiko terjadinya *race condition* menjadi semakin tinggi karena sistem harus memproses

banyak permintaan pengguna secara bersamaan dalam waktu yang sangat singkat.

Salah satu bentuk *race condition* yang sering ditemukan pada sistem transaksi adalah *Time-of-Check to Time-of-Use (TOCTOU)*. Kondisi ini terjadi ketika sistem melakukan pemeriksaan terhadap suatu data terlebih dahulu, namun data tersebut berubah sebelum digunakan dalam proses berikutnya. Sebagai contoh, sistem memeriksa bahwa saldo pengguna mencukupi untuk melakukan transaksi, tetapi sebelum transaksi selesai diproses, terdapat transaksi lain yang telah mengurangi saldo tersebut. Akibatnya, keputusan yang diambil sistem menjadi tidak sesuai dengan kondisi data terkini. (Hana Lolita BLT et al., 2021)

Untuk mengurangi risiko *race condition*, sistem perlu menerapkan mekanisme pengendalian konkurensi (*concurrency control*) seperti sinkronisasi proses (*synchronization*), transaksi atomik (*atomic transaction*), penguncian data (*locking*), validasi status transaksi, serta penerapan *idempotency* pada proses transaksi. Mekanisme tersebut bertujuan untuk memastikan bahwa setiap transaksi diproses secara terkontrol sehingga integritas data tetap terjaga meskipun terdapat banyak transaksi yang berjalan secara bersamaan.

2.9. Critical Section

Bagaimana menghindari race conditions? Kunci untuk mencegah masalah ini dan di situasi yang lain yang melibatkan shared memori, shared berkas, and shared sumber daya yang lain adalah menemukan beberapa jalan untuk mencegah lebih dari satu proses untuk melakukan proses writing dan reading kepada shared data pada saat yang sama. Dengan kata lain kita memutuhkan mutual exclusion, sebuah jalan yang menjamin jika sebuah proses sedang menggunakan shared berkas, proses lain dikeluarkan dari pekerjaan yang sama. Kesulitan yang terjadi karena proses 2 mulai menggunakan variabel bersama sebelum proses 1 menyelesaikan tugasnya. (Hartono et al., 2018)

Masalah menghindari *race condition* dapat diformulasikan secara lebih abstrak. Sebagian besar waktu, suatu proses menjalankan operasi internal yang tidak berinteraksi dengan proses lain sehingga tidak berpotensi menimbulkan konflik. Namun, ketika proses tersebut perlu mengakses sumber daya bersama, seperti memori, berkas, atau data yang digunakan oleh proses lain, maka terdapat

kemungkinan terjadinya konflik akses. Bagian dari program yang mengakses sumber daya bersama tersebut disebut sebagai Critical Section atau Critical Region. Agar akses terhadap sumber daya bersama berlangsung secara aman, setiap proses harus melewati beberapa tahapan, yaitu Entry Section, Critical Section, Exit Section, dan Remainder Section. (Dubey et al., 2019)

A. Entry Section

Entry Section merupakan bagian program yang bertugas mengatur mekanisme sebelum suatu proses atau *thread* memasuki *critical section*. Pada tahap ini, proses akan melakukan permintaan akses terhadap sumber daya bersama serta memeriksa apakah sumber daya tersebut sedang digunakan oleh proses lain. Apabila sumber daya masih digunakan, proses harus menunggu hingga memperoleh hak akses. Tujuan utama *entry section* adalah memastikan bahwa hanya satu proses yang dapat memasuki *critical section* pada suatu waktu sehingga prinsip *mutual exclusion* dapat terpenuhi. Berbagai algoritma sinkronisasi, seperti Dekker, Peterson, Bakery, semaphore, maupun mutex, bekerja pada tahap ini untuk menentukan proses mana yang berhak memasuki *critical section*.

B. Critical Section

Critical Section merupakan bagian program yang secara langsung mengakses atau memodifikasi sumber daya bersama (*shared resource*). Pada tahap inilah operasi yang bersifat kritis dilakukan, seperti membaca, memperbarui, atau menulis data yang digunakan oleh beberapa proses secara bersamaan. Karena perubahan yang dilakukan pada tahap ini dapat memengaruhi proses lain, hanya satu proses atau *thread* yang diperbolehkan berada di dalam *critical section* pada satu waktu. Apabila lebih dari satu proses memasuki *critical section* secara bersamaan tanpa mekanisme sinkronisasi yang memadai, maka dapat terjadi *race condition*, *lost update*, maupun inkonsistensi data.

C. Exit Section

Exit Section merupakan bagian program yang dijalankan setelah proses selesai melakukan operasi pada *critical section*. Pada tahap ini, proses akan melepaskan hak akses terhadap sumber daya bersama sehingga

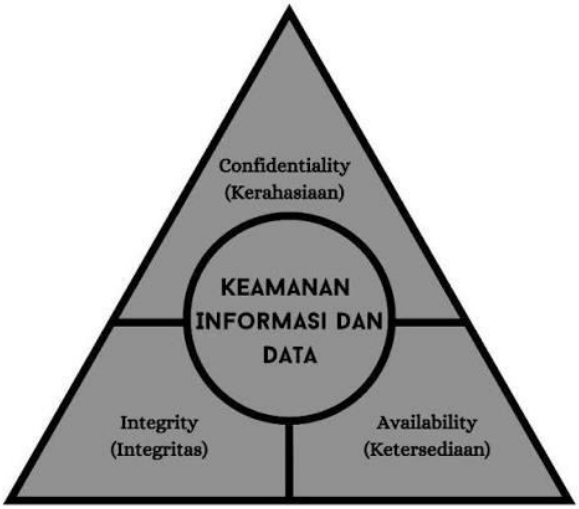
proses lain yang sedang menunggu dapat memasuki *critical section*. Selain melepaskan mekanisme sinkronisasi, seperti *lock*, *mutex*, atau *semaphore*, tahap ini juga berfungsi untuk memastikan bahwa seluruh perubahan pada sumber daya bersama telah selesai dilakukan sebelum akses diberikan kepada proses berikutnya.

D. Remainder Section

Remainder Section merupakan bagian program yang berisi aktivitas di luar *critical section*. Pada tahap ini, proses menjalankan operasi yang tidak melibatkan sumber daya bersama sehingga tidak berpotensi menimbulkan konflik dengan proses lain. Selama berada pada *remainder section*, proses dapat melakukan berbagai perhitungan atau tugas lain tanpa memerlukan mekanisme sinkronisasi. Setelah seluruh pekerjaan pada *remainder section* selesai dan proses kembali membutuhkan akses terhadap sumber daya bersama, proses akan kembali memasuki *entry section* untuk mengajukan permintaan akses.

2.10. Cia Triad

CIA Triad merupakan model dasar dalam keamanan informasi (*information security*) yang digunakan sebagai pedoman untuk menjaga keamanan data dan sistem informasi. Model ini terdiri atas tiga prinsip utama, yaitu Confidentiality (kerahasiaan), Integrity (integritas), dan Availability (ketersediaan). Ketiga prinsip tersebut saling melengkapi dalam memastikan bahwa informasi hanya dapat diakses oleh pihak yang berwenang, tetap akurat dan tidak mengalami perubahan yang tidak sah, serta selalu tersedia ketika dibutuhkan. CIA Triad menjadi salah satu konsep fundamental yang banyak digunakan dalam perancangan, pengelolaan, dan evaluasi sistem informasi agar mampu memberikan jaminan keamanan terhadap data maupun layanan yang disediakan.



Gambar 2. 4 CIA Triad. Sumber : (Harahap et al., 2023)

A. Confidentiality

Confidentiality atau kerahasiaan merupakan prinsip yang menjamin bahwa informasi hanya dapat diakses oleh pengguna atau pihak yang memiliki hak akses. Tujuan utama dari prinsip ini adalah mencegah terjadinya pengungkapan informasi kepada pihak yang tidak berwenang. Untuk menjaga kerahasiaan data, berbagai mekanisme keamanan dapat diterapkan, seperti autentikasi pengguna, otorisasi hak akses, enkripsi data, serta penggunaan kata sandi yang kuat. Dengan adanya confidentiality, informasi yang bersifat sensitif dapat terlindungi dari pencurian maupun penyalahgunaan.

B. Integrity

Integrity atau integritas merupakan prinsip yang menjamin bahwa data tetap akurat, utuh, konsisten, dan tidak mengalami perubahan tanpa izin selama proses penyimpanan, pengiriman, maupun pemrosesan. Integritas memastikan bahwa informasi yang diterima oleh pengguna sama dengan informasi yang dikirimkan atau disimpan sebelumnya. Untuk menjaga integritas data, sistem umumnya menerapkan mekanisme validasi data, checksum, hash function, digital signature, serta pengendalian transaksi pada Basis Data. Dengan demikian, setiap perubahan data dapat dipastikan berasal dari proses yang sah dan tidak disebabkan oleh manipulasi maupun

kesalahan sistem.

C. Availability

Availability atau ketersediaan merupakan prinsip yang menjamin bahwa sistem, layanan, maupun data selalu dapat diakses oleh pengguna yang berwenang ketika diperlukan. Ketersediaan menjadi aspek penting karena meskipun data telah terjaga kerahasiaan dan integritasnya, sistem tetap dianggap tidak memenuhi kebutuhan apabila layanan tidak dapat digunakan. Untuk menjaga availability, organisasi umumnya menerapkan mekanisme seperti pencadangan data (*backup*), replikasi server, sistem pemulihan bencana (*disaster recovery*), redundansi perangkat keras, serta pemantauan infrastruktur secara berkelanjutan sehingga layanan tetap berjalan meskipun terjadi gangguan.

2.11. Kriteria Pencegahan Race Condition

Race condition merupakan permasalahan yang timbul ketika dua atau lebih proses atau *thread* mengakses sumber daya bersama (*shared resource*) secara bersamaan tanpa adanya mekanisme sinkronisasi yang memadai. Kondisi tersebut dapat menyebabkan inkonsistensi data karena hasil akhir eksekusi bergantung pada urutan (*timing*) setiap proses yang dijalankan. Oleh karena itu, diperlukan mekanisme sinkronisasi untuk mengatur akses terhadap sumber daya bersama sehingga setiap proses dapat berjalan secara aman, konsisten, dan terhindar dari konflik akses.

Secara umum, mekanisme pencegahan race condition dapat diterapkan pada dua tingkat yang berbeda, yaitu sinkronisasi pada level sistem operasi (Operating System Level) dan sinkronisasi pada level sistem manajemen Basis Data (Database Management System Level). Sinkronisasi pada level sistem operasi berfokus pada pengaturan akses terhadap *shared resource* oleh proses atau *thread* melalui konsep *critical section*, sehingga hanya satu proses yang dapat mengakses sumber daya tertentu pada satu waktu. Sementara itu, sinkronisasi pada level Basis Data berfokus pada pengelolaan transaksi (*transaction management*) agar beberapa transaksi yang berjalan secara bersamaan tetap menjaga konsistensi dan integritas data. Kedua pendekatan tersebut memiliki tujuan yang sama, yaitu mencegah terjadinya race

condition, namun diterapkan pada lapisan sistem yang berbeda sesuai dengan karakteristik permasalahan yang dihadapi.

2.11.1. Sinkronisasi Level OS (Low Level)

Sinkronisasi pada level sistem operasi (*low-level synchronization*) merupakan mekanisme yang digunakan untuk mengatur akses beberapa proses atau thread terhadap sumber daya bersama (*shared resource*) agar dapat berjalan secara terkoordinasi. Mekanisme ini bekerja pada tingkat proses atau thread yang dikelola oleh sistem operasi, sehingga bertujuan mencegah terjadinya konflik akses ketika beberapa proses mencoba membaca atau mengubah data yang sama secara bersamaan. Tanpa adanya sinkronisasi, eksekusi proses yang berlangsung secara konkuren dapat menyebabkan race condition, yaitu kondisi ketika hasil akhir bergantung pada urutan eksekusi proses yang tidak dapat diprediksi. Oleh karena itu, algoritma sinkronisasi pada level sistem operasi dirancang untuk mengatur kapan suatu proses diperbolehkan memasuki *critical section* dan kapan proses lain harus menunggu hingga sumber daya bersama kembali tersedia.

Agar mekanisme sinkronisasi dapat bekerja dengan benar, terdapat tiga kriteria utama yang harus dipenuhi, yaitu mutual exclusion, progress, dan bounded waiting. Ketiga kriteria tersebut menjadi dasar dalam perancangan berbagai algoritma sinkronisasi klasik, seperti Peterson Algorithm, Dekker Algorithm, Bakery Algorithm, Semaphore, dan Monitor. Dengan terpenuhinya ketiga kriteria tersebut, sistem dapat menjamin bahwa akses terhadap *critical section* berlangsung secara aman, adil, dan efisien sehingga risiko terjadinya race condition dapat diminimalkan.

A. Mutual Exclusion

Kondisi ini menjamin kebebasan dan keamanan data dengan memastikan bahwa hanya ada satu proses yang boleh berada di dalam *critical section* pada satu waktu. Jika suatu proses sedang mengeksekusi kode atau mengakses memori bersama di bagian kritis tersebut, semua proses lain yang ingin masuk harus tertahan dan menunggu di area *entry section* sampai proses pertama benar-benar selesai dan keluar. Hal ini mutlak diperlukan agar data tetap konsisten dan tidak terjadi tumpang tindih eksekusi (*race condition*).

B. Progress

Kondisi ini memastikan bahwa sistem tidak akan mengalami kebuntuan (*deadlock*) ketika *critical section* sedang kosong. Jika tidak ada proses yang sedang berjalan di dalam *critical section* dan ada beberapa proses lain yang mengantre untuk masuk, maka keputusan proses mana yang boleh masuk tidak boleh dihalangi oleh proses lain yang sedang berada di luar area sinkronisasi. Dengan kata lain, proses yang tidak sedang membutuhkan atau tidak sedang bersaing menggunakan *critical section* tidak boleh memblokir jalannya proses lain yang ingin maju memanfaatkan sumber daya tersebut.

C. Bounded Waiting

Kondisi ini memberikan jaminan keadilan bagi setiap proses agar terhindar dari masalah kelaparan sumber daya (*starvation*). Harus ada batasan atau prediksi waktu tunggu yang jelas mengenai berapa kali proses lain diizinkan masuk ke dalam *critical section* setelah suatu proses mengajukan permintaan masuk. Batasan ini memastikan bahwa setiap proses yang mengantre pada akhirnya akan mendapatkan gilirannya untuk mengeksekusi kode mereka, sehingga tidak ada satu proses pun yang harus menunggu tanpa kepastian dalam waktu yang tak terbatas.

2.11.2. Sinkronisasi Level DBMS (High Level)

Berbeda dengan sinkronisasi pada level sistem operasi yang mengatur akses proses atau thread terhadap *shared resource*, sinkronisasi pada level *Database Management System* (DBMS) berfokus pada pengelolaan transaksi yang mengakses data secara bersamaan (*concurrent transactions*). Pada lingkungan Basis Data, beberapa transaksi dapat berjalan secara paralel untuk meningkatkan kinerja sistem. Namun, kondisi tersebut juga berpotensi menimbulkan konflik akses data, seperti *race condition*, *lost update*, *dirty read*, maupun inkonsistensi data apabila tidak dikelola dengan mekanisme sinkronisasi yang tepat.

Untuk menjaga agar setiap transaksi tetap menghasilkan data yang akurat, konsisten, dan dapat diandalkan, DBMS menerapkan seperangkat aturan yang dikenal sebagai ACID (Atomicity, Consistency, Isolation, dan

Durability) (Diogo et al., 2019). Keempat prinsip tersebut menjadi dasar dalam pengelolaan transaksi pada sistem Basis Data modern. Atomicity menjamin bahwa transaksi dijalankan secara utuh, Consistency memastikan data tetap berada pada kondisi yang valid, Isolation mencegah transaksi yang berjalan bersamaan saling memengaruhi, sedangkan Durability menjamin bahwa hasil transaksi yang telah berhasil disimpan tidak akan hilang meskipun terjadi kegagalan sistem. Dengan menerapkan prinsip ACID, sistem Basis Data mampu mempertahankan integritas dan konsistensi data meskipun banyak transaksi diproses secara bersamaan. (Firmawati & Avorizano, 2026)

A. Atomicity

Atomicity adalah prinsip "all or nothing". Sebuah transaksi diperlakukan sebagai satu kesatuan yang tidak dapat dipisahkan. Jika seluruh langkah dalam transaksi berhasil dijalankan, maka perubahan akan disimpan (*commit*). Sebaliknya, jika satu langkah saja gagal, maka seluruh perubahan dibatalkan (*rollback*) sehingga keadaan data kembali seperti sebelum transaksi dimulai. Dengan demikian, tidak akan ada kondisi di mana hanya sebagian dari transaksi yang berhasil dieksekusi. Atomicity memastikan bahwa transaksi selalu berada pada dua kemungkinan, yaitu berhasil sepenuhnya atau gagal sepenuhnya.

B. Consistency

Consistency adalah kemampuan sistem untuk menjaga data tetap berada pada kondisi yang valid sebelum dan sesudah transaksi berlangsung. Setiap transaksi harus mematuhi seluruh aturan, batasan, dan integritas data yang telah ditetapkan oleh sistem. Apabila suatu transaksi menyebabkan pelanggaran terhadap aturan tersebut, maka transaksi harus ditolak. Dengan kata lain, consistency memastikan bahwa transaksi hanya dapat mengubah data dari satu keadaan yang valid ke keadaan valid lainnya tanpa merusak struktur maupun aturan yang berlaku pada Basis Data.

C. Isolation

Isolation adalah sifat yang menjamin bahwa transaksi yang

berjalan secara bersamaan (*concurrent transactions*) tidak saling mengganggu. Selama transaksi masih berlangsung, perubahan sementara yang terjadi di dalam transaksi tersebut tidak boleh terlihat oleh transaksi lain. Dari sudut pandang pengguna, setiap transaksi seolah-olah berjalan sendirian meskipun sebenarnya sistem sedang mengeksekusi banyak transaksi secara paralel. Tujuan isolation adalah mencegah terjadinya konflik data dan memastikan hasil akhir yang diperoleh sama seperti apabila transaksi dijalankan satu per satu secara berurutan.

D. Durability

Durability adalah jaminan bahwa data yang telah berhasil dikomit akan tersimpan secara permanen. Setelah sistem menyatakan bahwa transaksi berhasil diselesaikan, hasil transaksi tersebut tidak boleh hilang meskipun terjadi gangguan seperti kegagalan perangkat keras, kerusakan sistem operasi, atau pemadaman listrik. Untuk memenuhi sifat durability, sistem Basis Data biasanya menyimpan perubahan ke media penyimpanan permanen dan menyediakan mekanisme pemulihan (*recovery*) apabila terjadi kegagalan sistem. Dengan demikian, pengguna dapat mempercayai bahwa data yang telah berhasil disimpan akan tetap tersedia di masa mendatang.

2.12. Algoritma Pencegahan Race Condition

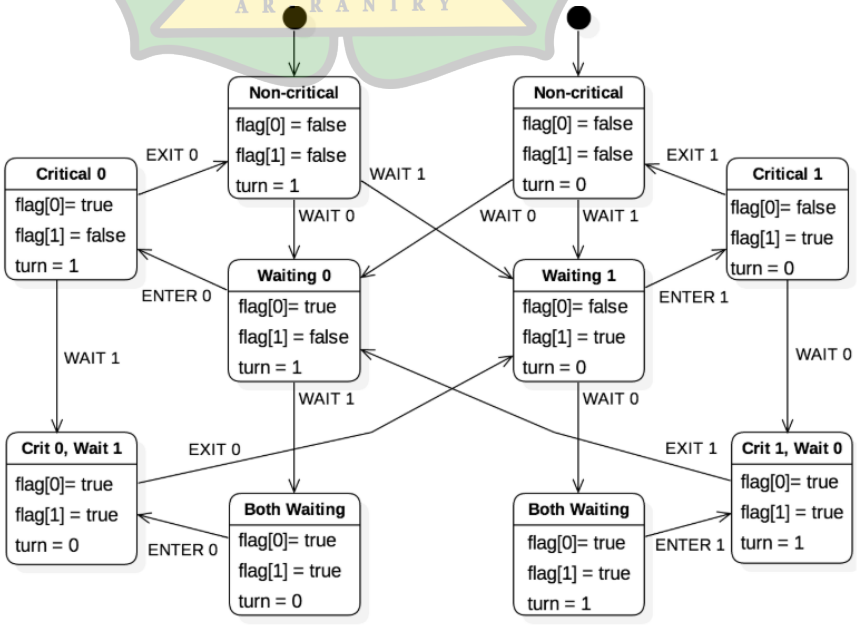
Setelah konsep *Critical Section Problem* dan kriteria sinkronisasi dipahami, diperlukan suatu mekanisme yang mampu mengimplementasikan prinsip-prinsip tersebut ke dalam bentuk algoritma. Algoritma pencegahan *race condition* merupakan sekumpulan metode yang digunakan untuk mengatur akses beberapa proses atau transaksi terhadap sumber daya bersama (*shared resource*) sehingga konflik akses data dapat dihindari. Seiring berkembangnya teknologi komputasi, mekanisme sinkronisasi mengalami perkembangan dari pendekatan berbasis perangkat lunak (*software-based synchronization*) yang bekerja pada tingkat proses (*process/thread*) hingga mekanisme transaksi yang dikelola secara otomatis oleh sistem manajemen Basis Data (*Database Management System*). Berdasarkan lapisan

implementasinya, algoritma pencegahan *race condition* pada penelitian ini dibedakan menjadi algoritma sinkronisasi klasik (*low-level synchronization*) dan algoritma sinkronisasi modern (*high-level synchronization*).

Pada algoritma sinkronisasi klasik, mekanisme sinkronisasi dirancang untuk memenuhi tiga kriteria utama penyelesaian *Critical Section Problem*, yaitu Mutual Exclusion, Progress, dan Bounded Waiting, sehingga hanya satu proses yang dapat mengakses *critical section* pada satu waktu serta mencegah terjadinya konflik akses. Sementara itu, pada algoritma sinkronisasi modern yang diterapkan pada sistem manajemen Basis Data, mekanisme sinkronisasi dikembangkan berdasarkan prinsip ACID (Atomicity, Consistency, Isolation, dan Durability) untuk menjamin bahwa transaksi yang berjalan secara bersamaan tetap menjaga konsistensi, integritas, dan keandalan data. Dengan demikian, baik algoritma sinkronisasi klasik maupun modern memiliki tujuan yang sama, yaitu mencegah terjadinya *race condition*, namun menggunakan pendekatan yang berbeda sesuai dengan lapisan sistem tempat mekanisme sinkronisasi diterapkan.

2.12.1. Algoritma Sinkronisasi Klasik

Salah satu algoritma sinkronisasi klasik Adalah Peterson Algorithm. Algoritma ini bekerja dengan memanfaatkan dua buah variabel bersama (*shared variables*), yaitu flag dan turn. Variabel flag[i] digunakan untuk menunjukkan bahwa proses ke-*i* ingin memasuki *critical section*, sedangkan



variabel turn digunakan sebagai penentu giliran apabila kedua proses mengajukan permintaan akses secara bersamaan. Algoritma ini hanya dirancang untuk dua proses, yaitu Process 0 dan Process 1, serta bertujuan memenuhi tiga kriteria penyelesaian *Critical Section Problem*, yaitu *Mutual Exclusion*, *Progress*, dan *Bounded Waiting*.

Berdasarkan Gambar di atas, proses diawali pada keadaan Non-critical, yaitu kondisi ketika kedua proses masih menjalankan pekerjaannya masing-masing dan belum membutuhkan akses ke *critical section*. Pada keadaan ini nilai flag[0] dan flag[1] bernilai false, yang menandakan bahwa tidak ada proses yang sedang meminta akses ke sumber daya bersama. Selama berada pada keadaan ini, kedua proses dapat berjalan secara independen tanpa saling memengaruhi.

Ketika Process 0 membutuhkan akses ke *critical section*, proses tersebut berpindah dari keadaan Non-critical menuju Waiting 0 melalui transisi WAIT 0. Pada tahap ini Process 0 mengubah nilai flag[0] = true sebagai tanda bahwa proses tersebut ingin memasuki *critical section*, kemudian mengatur turn = 1 sebagai bentuk pemberian prioritas kepada Process 1 apabila kedua proses mengajukan permintaan secara bersamaan. Apabila Process 1 tidak sedang meminta akses (flag[1] = false), maka Process 0 langsung berpindah ke keadaan Critical 0 melalui transisi ENTER 0 dan mulai mengeksekusi bagian program yang mengakses *shared resource*.

Proses yang sama juga berlaku untuk Process 1. Ketika Process 1 membutuhkan akses ke *critical section*, proses akan berpindah dari Non-critical menuju Waiting 1 melalui WAIT 1, kemudian mengubah flag[1] = true dan turn = 0. Jika Process 0 tidak sedang meminta akses, maka Process 1 akan langsung memasuki keadaan Critical 1 melalui ENTER 1.

Kondisi yang paling penting terjadi ketika kedua proses mengajukan permintaan secara bersamaan. Pada keadaan ini diagram menunjukkan perpindahan menuju Both Waiting, yaitu kondisi ketika flag[0] = true dan flag[1] = true. Karena kedua proses sama-sama ingin memasuki *critical section*, maka variabel turn berfungsi sebagai penentu giliran (*tie breaker*).

Proses yang tidak memperoleh giliran akan tetap berada pada kondisi Waiting (*busy waiting*), sedangkan proses yang memperoleh giliran diperbolehkan memasuki *critical section*. Dengan mekanisme ini, hanya satu proses yang dapat berada di dalam *critical section* pada suatu waktu sehingga *race condition* dapat dihindari.

Sebagai contoh, apabila Process 0 berhasil memasuki Critical 0 terlebih dahulu, sementara Process 1 masih menunggu, maka sistem berada pada keadaan Crit 0, Wait 1. Pada kondisi ini Process 0 masih menjalankan operasi pada *shared resource*, sedangkan Process 1 tetap berada dalam perulangan (*busy waiting*) hingga Process 0 selesai. Setelah pekerjaan selesai, Process 0 melakukan transisi EXIT 0 dengan mengubah flag[0] = false sebagai tanda bahwa proses tersebut telah meninggalkan *critical section*. Perubahan nilai flag[0] menyebabkan kondisi perulangan pada Process 1 menjadi tidak terpenuhi sehingga Process 1 dapat berpindah dari keadaan Waiting 1 menuju Critical 1 melalui transisi ENTER 1. Dengan demikian, akses terhadap *critical section* berpindah secara bergantian tanpa pernah memperbolehkan kedua proses berada di dalamnya secara bersamaan.

Setelah Process 1 menyelesaikan pekerjaannya, proses tersebut melakukan EXIT 1 dengan mengubah flag[1] = false, sehingga sistem kembali ke keadaan Non-critical. Siklus ini akan terus berulang setiap kali salah satu proses atau kedua proses kembali membutuhkan akses ke *critical section*. Seluruh mekanisme tersebut memastikan bahwa Peterson Algorithm mampu memenuhi prinsip Mutual Exclusion, Progress, dan Bounded Waiting, sehingga konflik akses terhadap sumber daya bersama dapat dicegah.

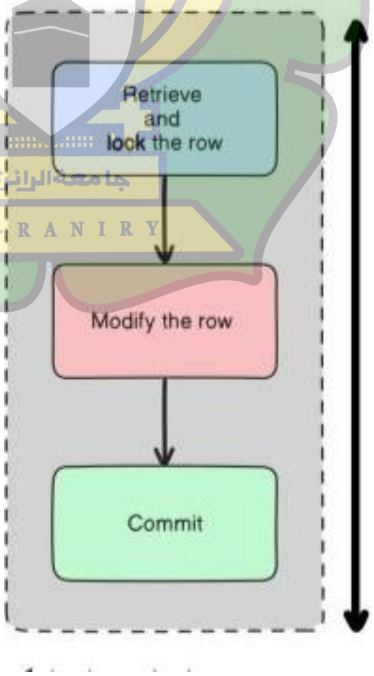
2.12.2. Algoritma Sinkronisasi Modern

Perkembangan sistem informasi dan Basis Data menyebabkan mekanisme sinkronisasi tidak lagi hanya diterapkan pada tingkat proses (*process*) atau *thread* sebagaimana pada algoritma sinkronisasi klasik. Pada sistem modern, terutama sistem yang menangani transaksi secara bersamaan

(*concurrent transactions*), sinkronisasi dilakukan pada tingkat transaksi (*transaction level*) dan dikelola langsung oleh *Database Management System* (DBMS). Pendekatan ini memungkinkan beberapa transaksi dijalankan secara paralel tanpa mengorbankan konsistensi maupun integritas data.

Secara umum, algoritma sinkronisasi modern dibedakan menjadi dua pendekatan utama, yaitu Pessimistic Concurrency Control (PCC) dan Optimistic Concurrency Control (OCC). Pessimistic Concurrency Control mencegah konflik dengan melakukan penguncian (*locking*) terhadap data sebelum transaksi dijalankan, sedangkan Optimistic Concurrency Control mengizinkan transaksi berjalan tanpa penguncian terlebih dahulu dan melakukan validasi konflik ketika transaksi akan disimpan (*commit*). Kedua pendekatan tersebut memiliki karakteristik, kelebihan, dan kekurangan yang berbeda sehingga penerapannya disesuaikan dengan kebutuhan serta karakteristik sistem yang dikembangkan. (Adjie Eriyadi et al., 2025)

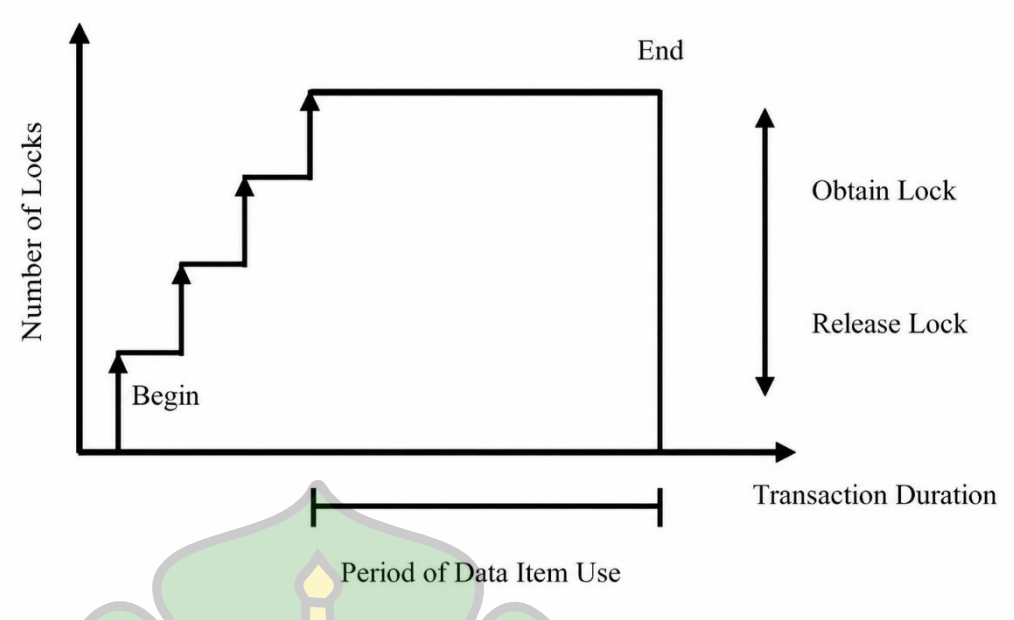
A. Pessimistic Concurrency Control



Gambar 2. 6 Pessimistic Concurrency Control

Pessimistic Concurrency Control (PCC) merupakan

pendekatan sinkronisasi yang berasumsi bahwa konflik antar transaksi memiliki kemungkinan besar untuk terjadi. Oleh karena itu, setiap transaksi harus memperoleh hak akses (*lock*) terhadap data sebelum melakukan operasi baca maupun tulis. Selama data masih berada dalam kondisi terkunci, transaksi lain yang ingin mengakses data yang sama harus menunggu hingga transaksi pertama selesai. Pendekatan ini bertujuan untuk mencegah konflik sejak awal sehingga dua transaksi tidak dapat memodifikasi data yang sama secara bersamaan.Keunggulan PCC adalah mampu menjaga konsistensi data dengan tingkat keamanan yang tinggi karena konflik dicegah sebelum terjadi. Namun, mekanisme ini menyebabkan transaksi lain harus menunggu apabila data sedang digunakan oleh transaksi lain. Akibatnya, pada sistem dengan jumlah transaksi yang tinggi dapat terjadi peningkatan waktu tunggu (*waiting time*), penurunan *throughput*, bahkan kondisi *deadlock*, yaitu keadaan ketika dua atau lebih transaksi saling menunggu pelepasan *lock* sehingga tidak ada transaksi yang dapat dilanjutkan. Salah satu protokol *locking* yang paling banyak digunakan dalam implementasi *Pessimistic Concurrency Control (PCC)* adalah *Strict Two-Phase Locking (Strict 2PL)*, yang mengatur proses perolehan dan pelepasan *lock* untuk menjaga konsistensi data selama transaksi berlangsung.



Gambar 2. 7 Strict Two Phase Locking

Strict Two-Phase Locking (Strict 2PL) merupakan pengembangan dari protokol *Two-Phase Locking (2PL)* yang banyak digunakan dalam implementasi Pessimistic Concurrency Control (PCC). Seperti pada 2PL, setiap transaksi memiliki dua fase utama, yaitu Growing Phase dan Shrinking Phase. Pada Growing Phase, transaksi hanya diperbolehkan memperoleh (*acquire*) *lock* terhadap data yang diperlukan dan tidak diperbolehkan melepaskan *lock* yang telah dimiliki. Setelah seluruh *lock* yang dibutuhkan berhasil diperoleh, transaksi memasuki Shrinking Phase, yaitu fase ketika transaksi mulai melepaskan *lock*. Perbedaannya terletak pada aturan pelepasan *lock*, di mana pada Strict 2PL seluruh exclusive lock yang digunakan untuk operasi penulisan (*write*) harus tetap dipertahankan hingga transaksi selesai melakukan commit atau rollback. Dengan demikian, tidak ada transaksi lain yang dapat membaca maupun memodifikasi data yang masih berada dalam proses transaksi.

Mekanisme tersebut mampu mencegah terjadinya konflik akses data, seperti *lost update*, *dirty read*, dan *race condition*, karena

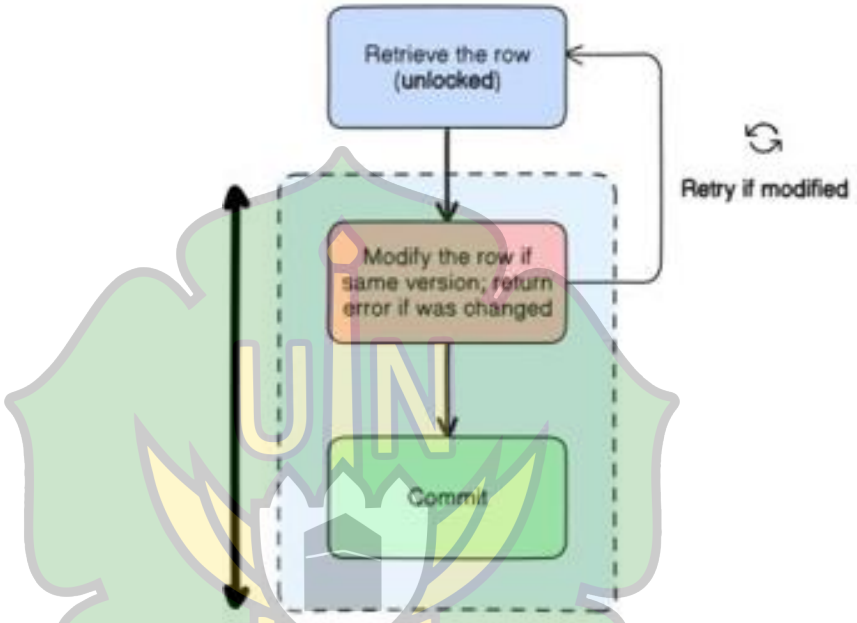
transaksi lain harus menunggu hingga transaksi yang sedang berjalan benar-benar selesai sebelum memperoleh akses terhadap data yang sama. Selain menghasilkan jadwal eksekusi (*schedule*) yang bersifat *serializable*, Strict 2PL juga menjamin *recoverability*, sehingga apabila suatu transaksi gagal dan melakukan *rollback*, transaksi lain tidak akan pernah membaca data yang belum dikomit (*uncommitted data*). Oleh karena itu, protokol Strict 2PL banyak diterapkan pada sistem Basis Data yang mengutamakan konsistensi dan integritas data. Namun demikian, penggunaan mekanisme ini dapat meningkatkan waktu tunggu (*waiting time*) serta berpotensi menyebabkan *deadlock* apabila dua atau lebih transaksi saling menunggu pelepasan *lock*.

B. Optimistic Concurrency Control

Optimistic Concurrency Control (OCC) merupakan pendekatan sinkronisasi yang berasumsi bahwa konflik antar transaksi relatif jarang terjadi. Oleh karena itu, transaksi diperbolehkan membaca dan memodifikasi data tanpa melakukan *locking* selama proses eksekusi berlangsung. Konflik baru akan diperiksa ketika transaksi akan disimpan (*commit*). Apabila tidak ditemukan perubahan data oleh transaksi lain selama proses berlangsung, maka transaksi akan berhasil disimpan. Sebaliknya, apabila ditemukan konflik, transaksi akan dibatalkan (*rollback*) atau dijalankan kembali

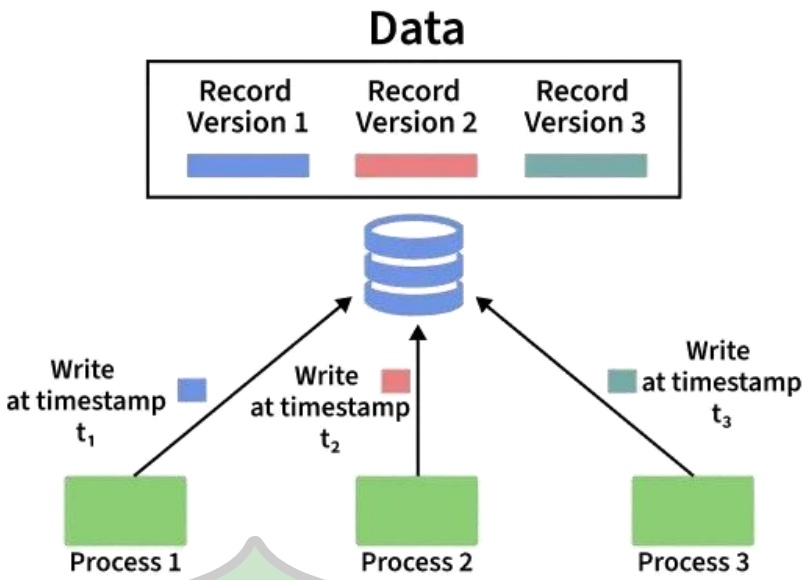
(*retry*) menggunakan data terbaru.

Pendekatan OCC memiliki keunggulan berupa tingkat konkurensi yang lebih tinggi karena transaksi tidak perlu menunggu pelepasan *lock*. Hal ini menjadikan OCC lebih efisien pada sistem yang memiliki jumlah operasi baca (*read*) lebih banyak dibandingkan



Gambar 2. 8 Optimistic Concurrency Control

operasi tulis (*write*) serta memiliki tingkat konflik transaksi yang rendah. Namun, apabila konflik sering terjadi, mekanisme *retry* yang berulang dapat meningkatkan waktu eksekusi dan menurunkan performa sistem. Salah satu mekanisme yang paling umum digunakan dalam implementasi *Optimistic Concurrency Control (OCC)* adalah *Multi-Version Concurrency Control (MVCC)*, yang mengelola beberapa versi data untuk memungkinkan transaksi berjalan secara bersamaan tanpa bergantung pada mekanisme *locking*.



Gambar 2. 9 Multi-Version Concurrency Control

Multi-Version Concurrency Control (MVCC) merupakan salah satu mekanisme yang umum digunakan untuk mendukung implementasi *Optimistic Concurrency Control*. Berbeda dengan pendekatan *locking*, MVCC menyimpan lebih dari satu versi data sehingga transaksi dapat membaca versi data yang konsisten tanpa terganggu oleh transaksi lain yang sedang melakukan perubahan. Ketika suatu transaksi memulai eksekusinya, transaksi tersebut akan memperoleh *snapshot* dari data pada waktu tertentu. Selama transaksi berlangsung, proses pembacaan tetap menggunakan *snapshot* tersebut meskipun terdapat transaksi lain yang telah melakukan perubahan terhadap data yang sama. (Kanungo & Morena, 2024)

Sebelum transaksi dikomit, sistem akan melakukan proses validasi untuk memastikan bahwa data yang digunakan selama transaksi masih sesuai dengan kondisi data terbaru. Apabila tidak ditemukan konflik, perubahan akan disimpan sebagai versi baru dari data. Sebaliknya, apabila ditemukan bahwa data telah diubah oleh transaksi lain, maka transaksi dapat dibatalkan atau diulang menggunakan versi data terbaru. Dengan memanfaatkan beberapa versi data, MVCC mampu meningkatkan tingkat konkurensi dan

mengurangi kebutuhan penggunaan *locking*, sehingga transaksi baca dan tulis dapat berlangsung secara lebih efisien dibandingkan mekanisme *locking* konvensional.

2.13. Cloud Firestore Transaction

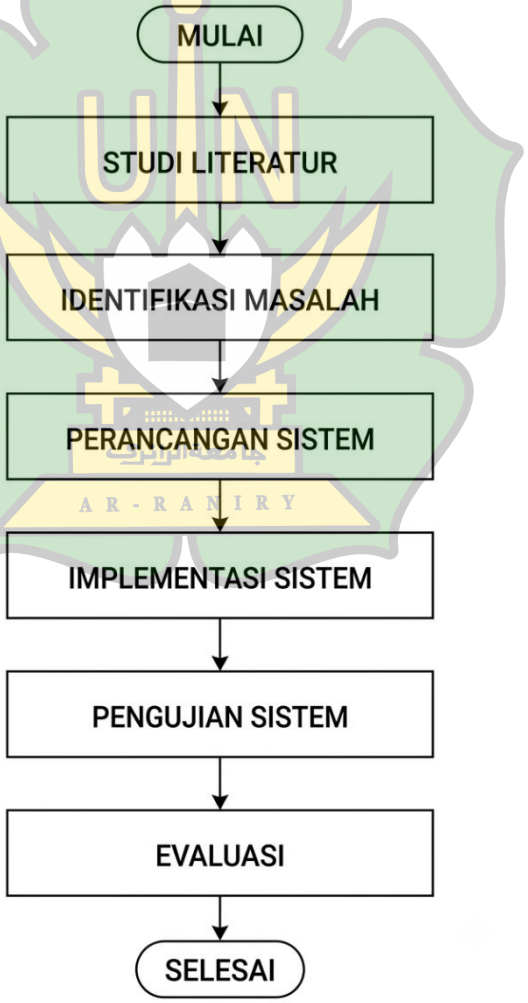
Cloud Firestore Transaction merupakan mekanisme transaksi yang disediakan oleh Cloud Firestore untuk melakukan pembacaan dan pembaruan data secara atomik. Mekanisme ini digunakan ketika beberapa operasi terhadap data harus dilakukan secara konsisten sehingga perubahan data tidak terjadi secara sebagian apabila proses transaksi mengalami kegagalan. Dalam konteks aplikasi yang memiliki akses data secara bersamaan, penggunaan transaction dapat membantu memastikan bahwa data yang dibaca selama proses transaksi masih sesuai dengan kondisi data pada saat transaksi akan disimpan.

Dengan demikian, Cloud Firestore Transaction memiliki peran penting dalam menjaga integritas data pada aplikasi yang menangani transaksi secara konkuren. Penggunaan mekanisme transaksi memungkinkan proses pembacaan dan pembaruan saldo dilakukan secara terkontrol sehingga transaksi yang mengalami konflik tidak menyebabkan pembaruan data yang tidak sah. Dalam aplikasi PPOB, mekanisme tersebut menjadi penting karena kesalahan pada proses pembaruan saldo dapat menyebabkan ketidaksesuaian antara saldo pengguna dengan riwayat transaksi. Oleh karena itu, Cloud Firestore Transaction dalam penelitian ini digunakan sebagai mekanisme untuk menjaga konsistensi dan integritas data saldo ketika terjadi akses transaksi secara bersamaan.

BAB III
METODE PENELITIAN

3.1. Tahapan Penelitian

Pelaksanaan penelitian menggunakan pendekatan kuantitatif serta dilakukan melalui tahapan yang disusun secara sistematis sesuai dengan alur penelitian pada Gambar 3.1. Tahapan tersebut dimulai dari studi literatur sebagai dasar penyusunan penelitian, kemudian dilanjutkan dengan proses pengembangan sistem hingga tahap evaluasi untuk mengukur kinerja sistem yang dihasilkan.



Gambar 3. 1 Diagram Alur Penelitian

3.1.1. Studi Literatur

Studi literatur merupakan tahapan yang dilakukan untuk memperoleh landasan teori dan informasi ilmiah yang mendukung pelaksanaan penelitian. Pada tahap ini dilakukan pengumpulan serta kajian terhadap berbagai sumber referensi, seperti buku, jurnal ilmiah, prosiding, standar, dan dokumentasi resmi yang berkaitan dengan sistem pembayaran digital, Payment Point Online Bank, concurrent programming, race condition, integritas data, concurrency control, Optimistic Concurrency Control, Multi-Version Concurrency Control, serta mekanisme transaksi pada sistem Basis Data. Hasil studi literatur digunakan sebagai dasar dalam merancang mekanisme sinkronisasi transaksi, menentukan metode yang digunakan, serta menjadi acuan dalam proses implementasi dan evaluasi untuk mencegah terjadinya race condition pada aplikasi PPOB berbasis mobile.

3.1.2. Identifikasi Masalah

Berdasarkan kajian literatur, teridentifikasi tiga permasalahan utama yang menjadi fokus penyelesaian dalam penelitian ini, yaitu:

1. Potensi Race Condition pada Transaksi Konkuren, proses transaksi pada aplikasi PPOB memungkinkan beberapa pengguna melakukan akses dan perubahan terhadap data yang sama secara bersamaan. Tanpa mekanisme sinkronisasi yang memadai, kondisi tersebut dapat menyebabkan race condition sehingga hasil transaksi bergantung pada urutan eksekusi yang tidak dapat diprediksi.
2. Risiko Inkonsistensi dan Kehilangan Pembaruan Data (Lost Update), ketika dua atau lebih transaksi melakukan operasi baca dan tulis terhadap data yang sama secara bersamaan, terdapat kemungkinan perubahan yang dilakukan oleh salah satu transaksi menimpa perubahan transaksi lainnya. Kondisi ini dapat menyebabkan saldo, status transaksi, maupun riwayat transaksi tidak sesuai dengan kondisi sebenarnya sehingga mengurangi integritas data.
3. Kebutuhan Mekanisme Sinkronisasi Transaksi yang Efisien, penggunaan mekanisme sinkronisasi berbasis locking dapat menjaga konsistensi data, namun berpotensi meningkatkan waktu tunggu

(waiting time) serta menurunkan tingkat konkurensi sistem. Oleh karena itu, diperlukan mekanisme sinkronisasi yang mampu menjaga konsistensi dan integritas data tanpa mengurangi kemampuan sistem dalam memproses transaksi secara bersamaan.

Berdasarkan permasalahan tersebut, sistem yang dikembangkan dalam penelitian ini dirancang untuk memenuhi kebutuhan sebagai berikut:

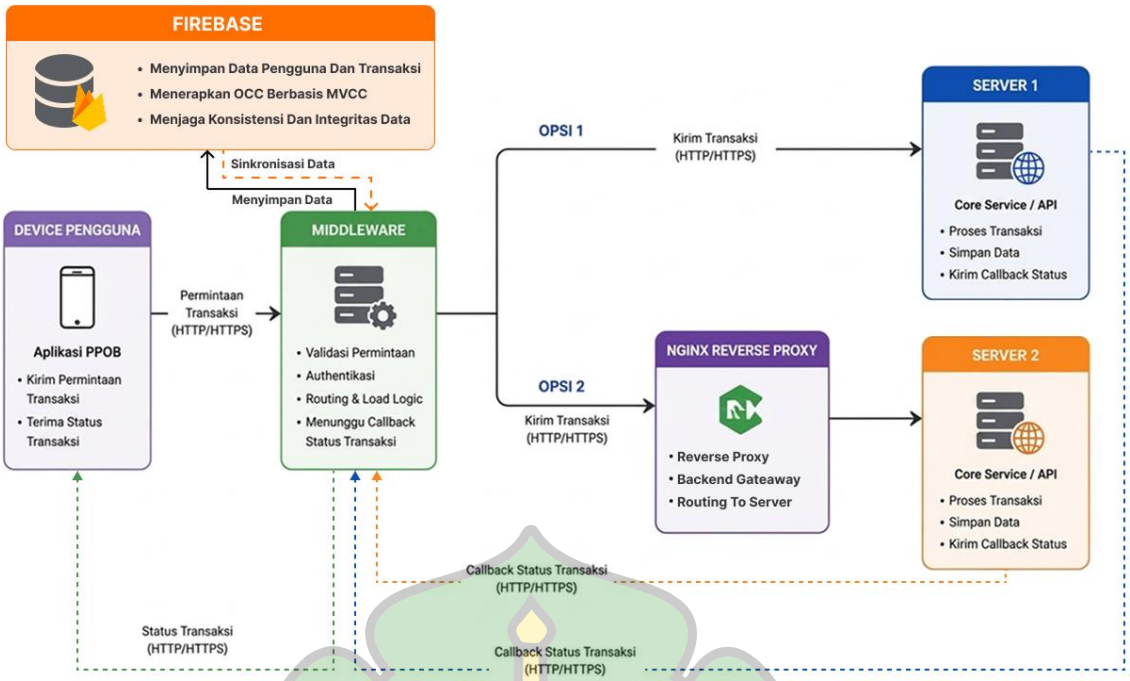
1. Sistem harus mampu mengelola transaksi yang berjalan secara bersamaan menggunakan mekanisme sinkronisasi berbasis Optimistic Concurrency Control (OCC) sehingga konflik transaksi dapat dideteksi dan ditangani sebelum perubahan data disimpan.
2. Sistem harus mampu menjaga konsistensi dan integritas data transaksi, mendukung pemrosesan transaksi secara konkuren, serta meminimalkan terjadinya race condition tanpa mengurangi keandalan dan performa sistem.

3.1.3. Perancangan Sistem

Perancangan sistem merupakan tahapan yang dilakukan untuk mendefinisikan struktur, komponen, serta mekanisme yang akan digunakan dalam membangun sistem sesuai dengan kebutuhan penelitian. Pada tahap ini dirancang arsitektur sistem, alur proses transaksi, mekanisme sinkronisasi, struktur Basis Data, serta komponen pendukung lainnya yang berperan dalam mencegah terjadinya race condition. Hasil dari tahap perancangan ini menjadi acuan pada proses implementasi sehingga setiap komponen sistem dapat bekerja secara terintegrasi dan sesuai dengan tujuan penelitian.

3.1.3.1. Arsitektur Topologi Sistem

Arsitektur sistem pada penelitian ini dirancang untuk menggambarkan struktur serta alur komunikasi antar komponen yang terlibat dalam proses transaksi pada aplikasi Payment Point Online Bank. Dengan adanya arsitektur sistem ini, hubungan antar komponen serta alur pertukaran data selama proses transaksi dapat dipahami secara lebih jelas. Arsitektur sistem yang digunakan pada penelitian ini ditunjukkan pada Gambar 3.2.



Gambar 3. 2 *Topologi Arsitektur Sistem*

Gambar 3.2 menunjukkan proses transaksi diawali ketika pengguna melakukan permintaan melalui aplikasi yang terpasang pada perangkat mobile. Setiap permintaan transaksi, seperti pembelian produk atau proses pencairan dana, terlebih dahulu dikirimkan ke middleware. Middleware berfungsi sebagai pusat pengendali (central controller) yang menerima seluruh permintaan dari aplikasi, melakukan validasi data, autentikasi, serta menentukan layanan yang akan digunakan berdasarkan jenis transaksi yang dipilih oleh pengguna.

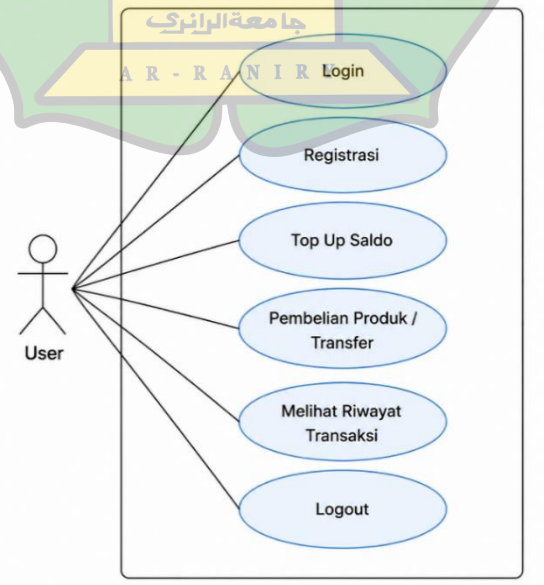
Untuk Server 1, middleware dapat mengirimkan permintaan transaksi secara langsung karena layanan yang disediakan tidak memiliki persyaratan khusus terkait asal permintaan (source request). Sementara itu, Server 2 menerapkan kebijakan keamanan berupa IP Whitelisting, yaitu hanya menerima permintaan yang berasal dari alamat IP tertentu yang telah didaftarkan sebelumnya. Kondisi tersebut menyebabkan alamat IP middleware tidak dapat diakses secara langsung oleh Server 2. Oleh karena itu, digunakan web server yang berfungsi sebagai reverse proxy. Reverse proxy menerima seluruh permintaan dari middleware menggunakan alamat IP publik yang telah didaftarkan pada Server 2, kemudian meneruskan permintaan tersebut ke Server 2 tanpa mengubah alur komunikasi aplikasi. Dengan mekanisme ini, persyaratan keamanan yang ditetapkan oleh

Server 2 tetap terpenuhi sekaligus memungkinkan middleware tetap menjadi pusat pengendali seluruh transaksi.

Mekanisme sinkronisasi transaksi pada penelitian ini diimplementasikan pada layer middleware melalui pemanggilan fungsi transaksi Cloud Firestore. Middleware bertugas menjalankan logika bisnis, melakukan validasi data, serta menginisiasi proses transaksi ke Basis Data. Adapun proses pencegahan *race condition* tidak dilakukan oleh middleware secara langsung, melainkan oleh mekanisme transaksi Cloud Firestore yang menerapkan *Optimistic Concurrency Control* (OCC) dengan dukungan *Multi-Version Concurrency Control* (MVCC). Dengan demikian, middleware berperan sebagai penginisiasi transaksi, sedangkan deteksi konflik, proses *retry*, dan *commit* transaksi sepenuhnya ditangani oleh Cloud Firestore.

3.1.3.2. Use Case Diagram

Use Case Diagram menunjukkan interaksi antara aktor dengan sistem serta fungsi-fungsi utama yang dapat dijalankan selama proses penggunaan aplikasi. Dengan diagram ini, ruang lingkup dan layanan yang disediakan oleh sistem dapat dipahami secara lebih jelas sebelum memasuki pembahasan mengenai alur proses pada setiap fungsi. Use Case Diagram pada penelitian ini ditunjukkan pada Gambar 3.3.



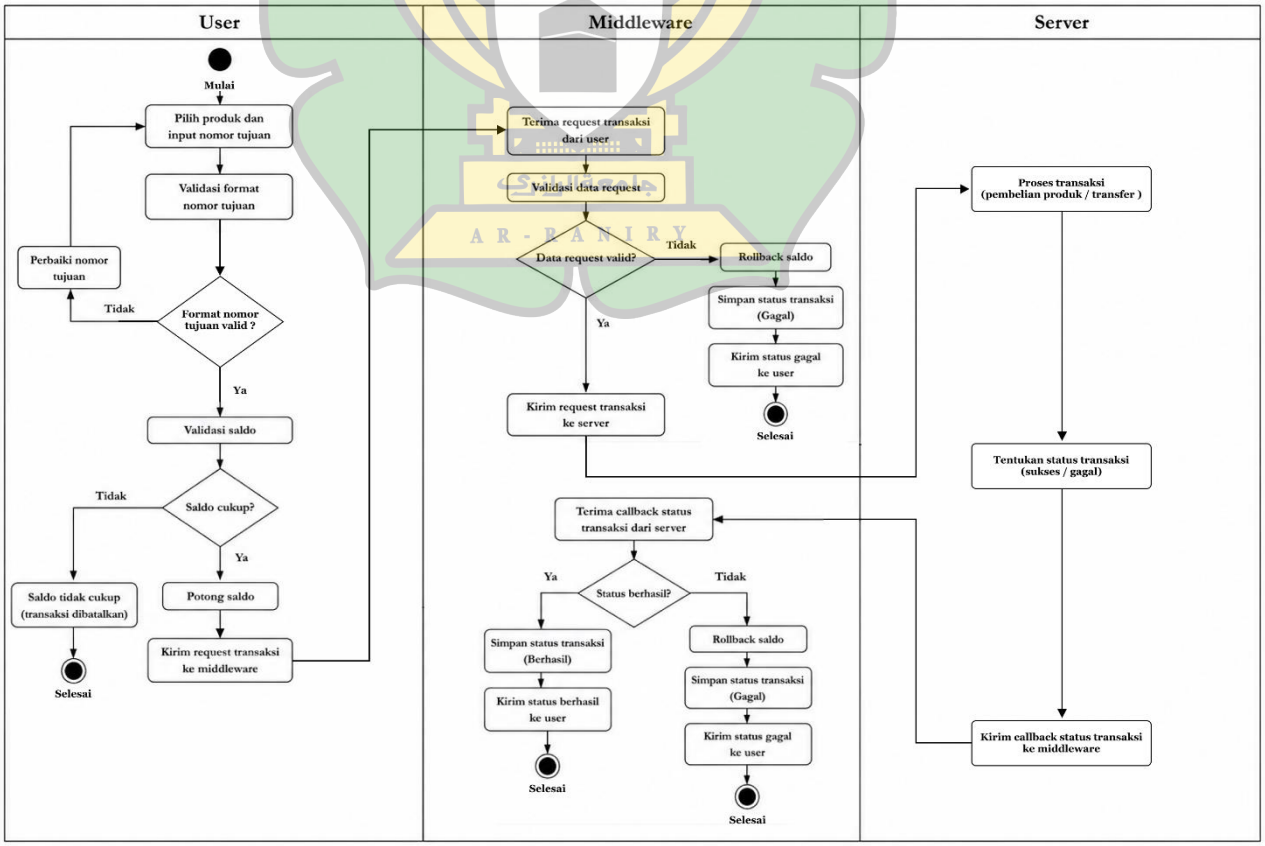
Gambar 3. 3 Diagram Use Case

Berdasarkan Use Case Diagram pada Gambar 3.3, terdapat satu aktor yang berinteraksi secara langsung dengan sistem, yaitu:

1. **User.** User merupakan pengguna aplikasi yang memiliki hak akses untuk memanfaatkan seluruh layanan yang tersedia setelah berhasil melakukan proses autentikasi. Sebelum menggunakan sistem, user dapat melakukan registrasi untuk membuat akun baru, kemudian login menggunakan akun yang telah terdaftar. Setelah berhasil masuk ke dalam sistem, user dapat melakukan top up saldo sebagai sumber dana transaksi, melakukan pembelian produk atau transfer yang mencakup berbagai layanan Payment Point Online Bank maupun proses disbursement, serta melihat riwayat transaksi untuk memantau seluruh aktivitas transaksi yang telah dilakukan. Setelah seluruh aktivitas selesai, user dapat melakukan logout untuk mengakhiri sesi penggunaan aplikasi.

3.1.3.3. Activity Diagram

Activity Diagram pada penelitian ini digunakan untuk menggambarkan alur aktivitas proses transaksi pembelian produk pada aplikasi PPOB.



Gambar 3. 4 Diagram Activity Proses Transaksi

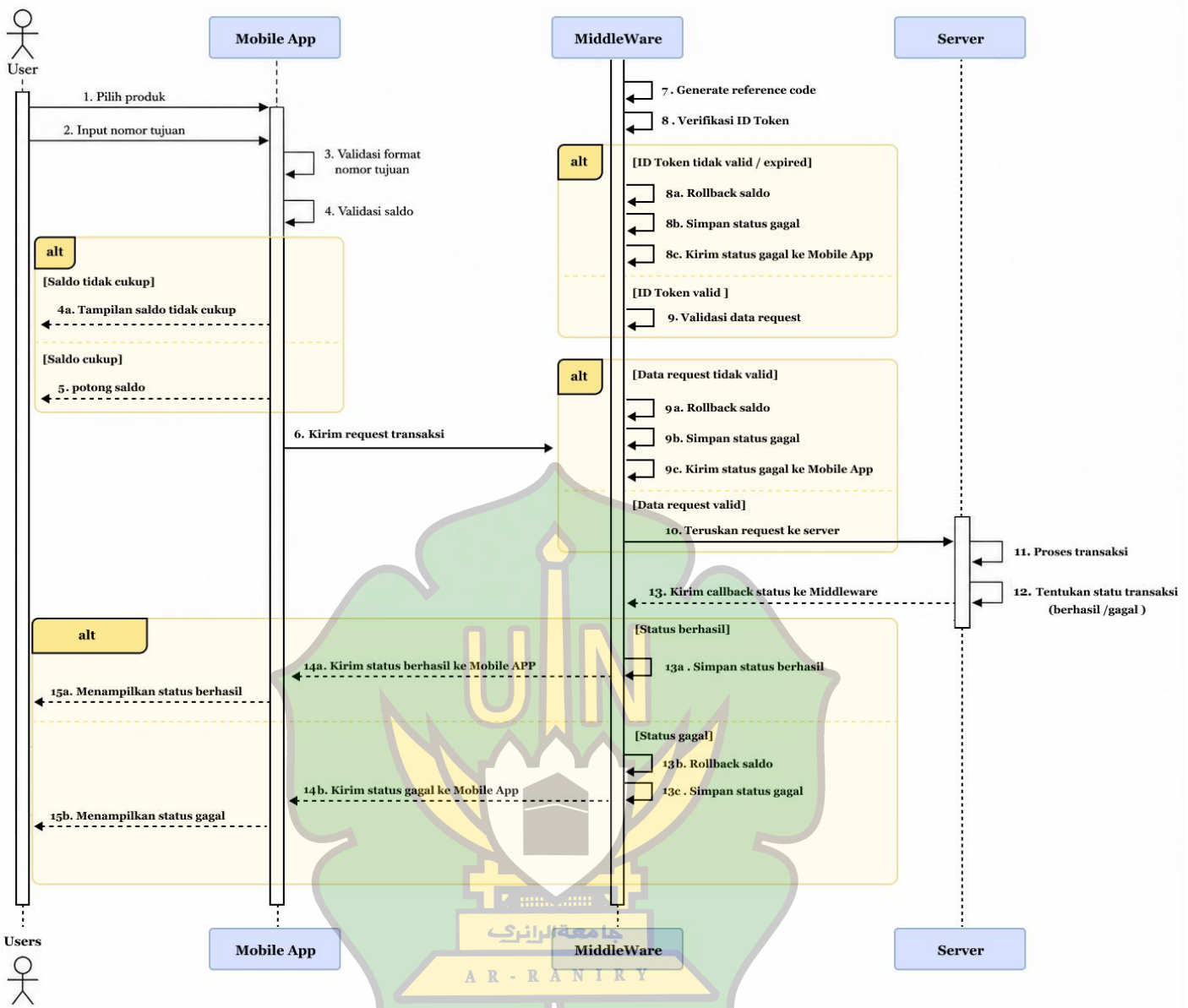
Berdasarkan Activity Diagram pada Gambar 3.4, proses transaksi diawali ketika pengguna memilih produk dan memasukkan nomor tujuan yang akan digunakan dalam transaksi. Selanjutnya, aplikasi melakukan validasi terhadap format nomor tujuan untuk memastikan data yang dimasukkan telah sesuai dengan ketentuan produk yang dipilih. Apabila format nomor tujuan tidak valid, pengguna diminta untuk memperbaiki data tersebut sebelum dapat melanjutkan proses transaksi. Setelah nomor tujuan dinyatakan valid, sistem melakukan validasi saldo. Jika saldo tidak mencukupi, transaksi dibatalkan dan proses berakhir. Sebaliknya, apabila saldo mencukupi, saldo akan dipotong sementara kemudian permintaan transaksi dikirimkan ke middleware.

Middleware menerima permintaan transaksi dari pengguna dan melakukan validasi terhadap data request untuk memastikan seluruh parameter yang dikirim telah lengkap dan sesuai dengan kebutuhan layanan. Apabila validasi data request gagal, middleware melakukan proses rollback saldo, menyimpan status transaksi sebagai gagal, kemudian mengirimkan informasi kegagalan kepada pengguna sehingga proses transaksi dihentikan. Jika data request valid, middleware meneruskan permintaan transaksi ke server untuk diproses.

Pada sisi server, sistem memproses permintaan transaksi sesuai jenis layanan yang dipilih, seperti pembelian produk digital maupun transfer dana. Setelah proses selesai, server menentukan status transaksi, yaitu berhasil atau gagal, kemudian mengirimkan callback status transaksi kepada middleware. Berdasarkan callback tersebut, middleware memperbarui status transaksi. Jika transaksi berhasil, status transaksi disimpan sebagai berhasil dan hasilnya dikirimkan kepada pengguna. Sebaliknya, apabila transaksi gagal, middleware melakukan rollback saldo, menyimpan status transaksi sebagai gagal, kemudian mengirimkan informasi kegagalan kepada pengguna. Dengan alur tersebut, proses transaksi dapat berlangsung secara terstruktur serta menjaga konsistensi data dan saldo pengguna selama proses transaksi berlangsung.

3.1.3.4. Sequence Diagram

Sequence Diagram pada Gambar 3.5 menggambarkan urutan interaksi antar komponen sistem selama proses transaksi, mulai dari pengguna melakukan permintaan hingga sistem mengembalikan status akhir transaksi.

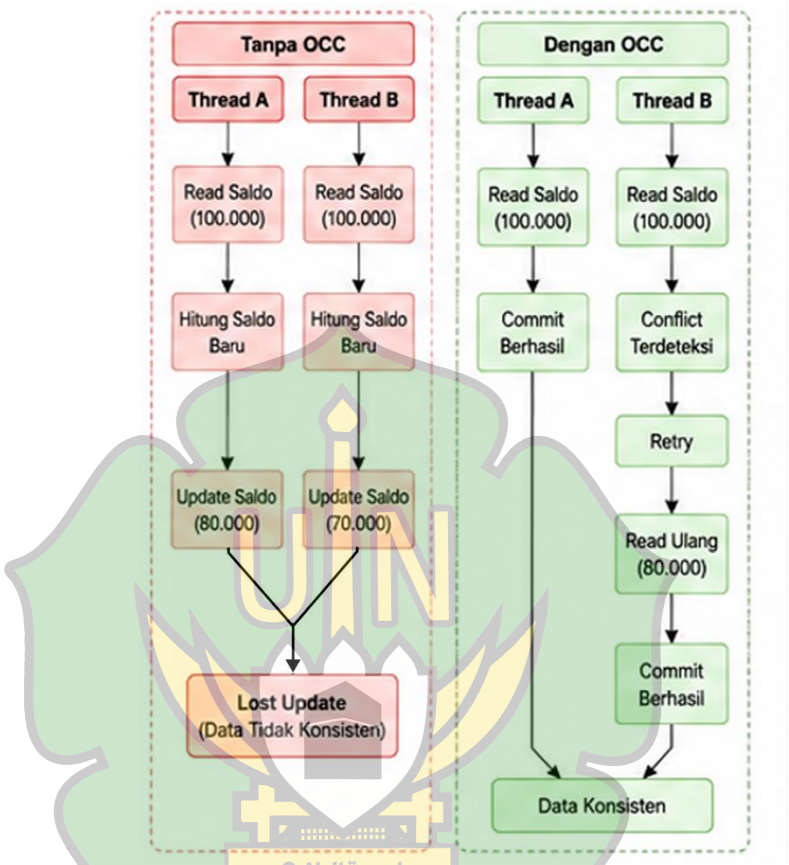


Gambar 3. 5 Diagram Sequence Alur Transaksi

Berdasarkan Sequence Diagram pada Gambar 3.5, proses transaksi dirancang untuk menggambarkan urutan komunikasi antara pengguna, aplikasi mobile, middleware, dan server dalam memproses transaksi. Diagram ini menunjukkan bahwa setiap permintaan transaksi melalui serangkaian validasi, seperti validasi saldo, verifikasi ID Token, dan validasi data request sebelum diteruskan ke server. Selain itu, mekanisme callback, rollback saldo, dan pembaruan status transaksi diterapkan untuk memastikan setiap transaksi menghasilkan status akhir yang konsisten serta menjaga integritas data.

3.1.3.5. Ilustrasi Pencegahan Race Condition

Untuk memberikan gambaran mengenai mekanisme penyelesaian race condition, Gambar 3.6 menyajikan perbandingan proses transaksi tanpa dan dengan menggunakan Optimistic Concurrency Control (OCC).



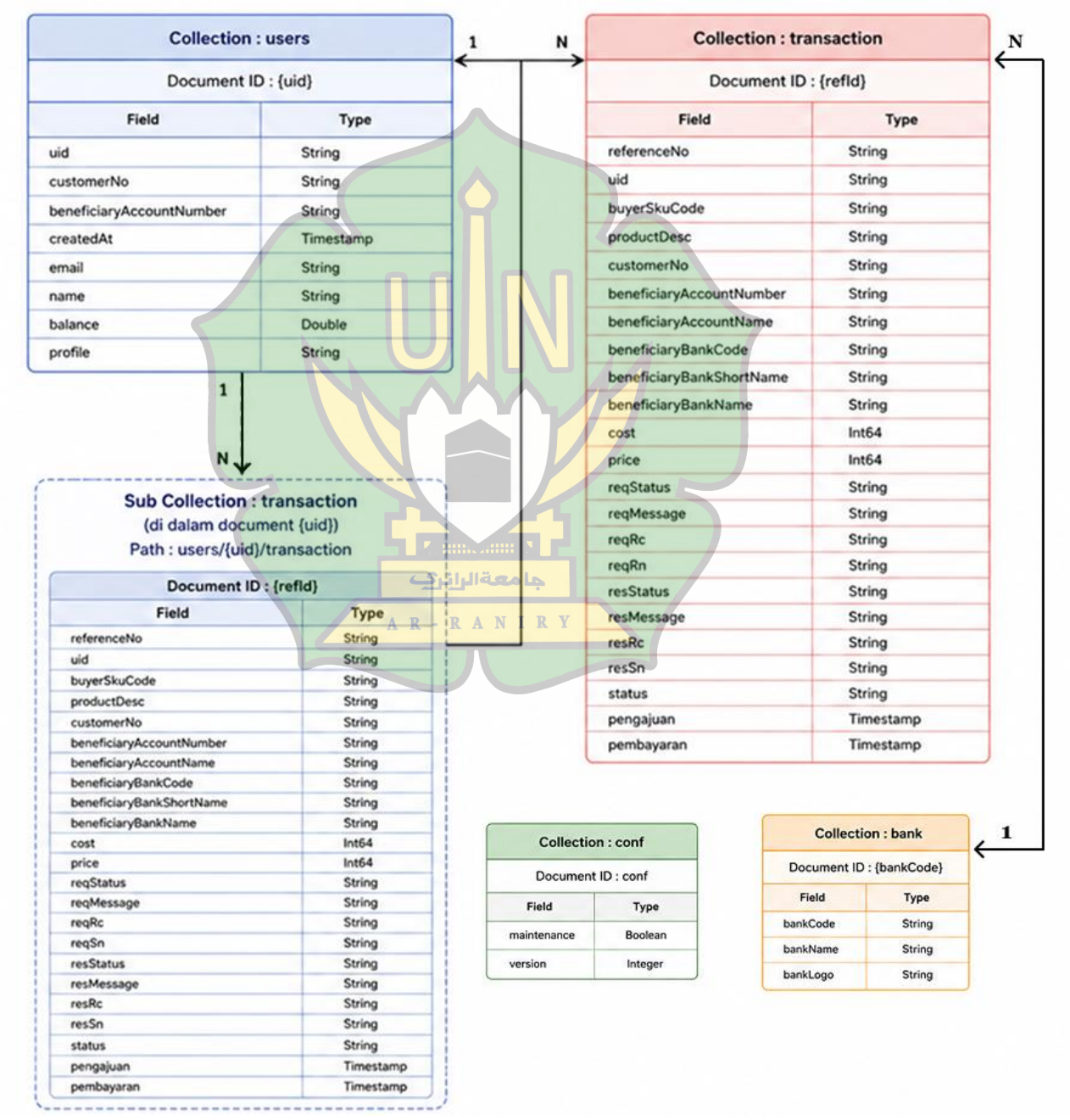
Gambar 3. 6 Ilustrasi Pencegahan Race Condition

Berdasarkan Gambar 3.6, tanpa penerapan Optimistic Concurrency Control (OCC), dua transaksi dapat membaca data saldo yang sama secara bersamaan, kemudian masing-masing menghitung dan memperbarui saldo berdasarkan nilai awal tersebut. Kondisi ini menyebabkan terjadinya lost update, yaitu salah satu perubahan data menimpa perubahan lainnya sehingga saldo akhir menjadi tidak konsisten. Sebaliknya, dengan penerapan OCC, kedua transaksi tetap dapat membaca data secara bersamaan, namun sistem akan melakukan validasi sebelum proses commit. Transaksi pertama berhasil disimpan, sedangkan transaksi berikutnya yang mendeteksi adanya perubahan data akan mengalami konflik (conflict detected), kemudian melakukan retry dengan membaca data terbaru

sebelum melakukan commit. Melalui mekanisme tersebut, OCC mampu mencegah terjadinya race condition serta menjaga konsistensi dan integritas data selama proses transaksi konkuren berlangsung.

3.1.3.6. Perancangan Basis Data

Gambar 3.7 menunjukkan model struktur collection dan document yang digunakan untuk mendukung proses autentikasi, pengelolaan data pengguna, transaksi, konfigurasi aplikasi, serta penyimpanan data bank.



Gambar 3. 7 Perancangan Basis Data

A. Collection Users

Collection users digunakan untuk menyimpan informasi setiap pengguna yang terdaftar pada aplikasi. Data yang disimpan meliputi identitas pengguna, saldo, nomor pelanggan, serta informasi rekening tujuan yang digunakan dalam proses transaksi. Collection ini menjadi sumber utama dalam proses autentikasi, validasi saldo, dan identifikasi pengguna selama sistem berjalan.

Tabel 3. 1 Collection Users

Field	Tipe Data	Keterangan
uid	String (PK)	ID pengguna
customerNo	String	Nomor pelanggan
beneficiaryAccountNumber	String	Nomor rekening
createdAt	Timestamp	Waktu dibuat
email	String	Email pengguna
name	String	Nama pengguna
balance	Double	Saldo pengguna
profile	String	Foto profil

B. Collection Transaction

Collection transaction digunakan untuk menyimpan seluruh data transaksi yang diproses oleh sistem, baik transaksi pembelian produk digital maupun transfer dana. Collection ini menyimpan informasi permintaan transaksi, respons dari server, status transaksi, serta waktu pengajuan dan penyelesaian transaksi. Data tersebut digunakan sebagai pencatatan transaksi sekaligus sebagai sumber informasi dalam proses monitoring dan pelacakan transaksi.

Tabel 3. 2 Collection Transaction

Field	Tipe Data	Keterangan
referenceNo	String (PK)	Nomor referensi
uid	String	ID pengguna
buyerSkuCode	String	Kode produk
productDesc	String	Nama produk

customerNo	String	Nomor pelanggan
beneficiaryAccountNumber	String	Nomor rekening tujuan
beneficiaryAccountName	String	Nama penerima
beneficiaryBankCode	String	Kode bank
beneficiaryBankShortName	String	Singkatan bank
beneficiaryBankName	String	Nama bank
cost	Int64	Harga modal
price	Int64	Harga jual
reqStatus	String	Status request
reqMessage	String	Pesan request
reqRc	String	Kode respons request
reqSn	String	Serial number request
resStatus	String	Status respons
resMessage	String	Pesan respons
resRc	String	Kode respons
resSn	String	Serial number respons
status	String	Status transaksi
pengajuan	Timestamp	Waktu pengajuan
pembayaran	Timestamp	Waktu pembayaran

C. Collection Bank

Collection bank digunakan untuk menyimpan daftar bank yang tersedia sebagai tujuan transfer dana. Informasi yang disimpan meliputi kode bank, nama bank, dan logo bank yang digunakan aplikasi untuk menampilkan daftar bank kepada pengguna ketika melakukan transaksi transfer.

Tabel 3. 3 Collection Bank

Field	Tipe Data	Keterangan
bankCode	String (PK)	Kode bank
bankName	String	Nama bank
bankLogo	String	Logo bank

D. Collection Conf

Collection conf digunakan untuk menyimpan konfigurasi global aplikasi yang dapat diakses oleh seluruh pengguna. Pada penelitian ini, collection tersebut menyimpan status maintenance untuk

mengatur ketersediaan layanan aplikasi serta informasi version yang digunakan untuk melakukan pengecekan versi aplikasi sehingga pengguna dapat menggunakan versi yang sesuai dengan ketentuan sistem.

Tabel 3. 4 Collection Conf

Field	Tipe Data	Keterangan
maintenance	Boolean	Status maintenance
version	Integer	Versi aplikasi

3.1.4. Implementasi Sistem

Implementasi sistem merupakan tahapan penerapan hasil perancangan ke dalam bentuk aplikasi yang dapat menjalankan seluruh proses transaksi sesuai kebutuhan penelitian. Pada tahap ini, dilakukan implementasi Basis Data, middleware, server, aplikasi mobile, serta mekanisme *Optimistic Concurrency Control* (OCC) berbasis *Multi-Version Concurrency Control* (MVCC) agar proses transaksi dapat berjalan secara terintegrasi dan menjaga konsistensi data.

A. Implementasi Basis Data

Implementasi Basis Data dilakukan menggunakan Cloud Firestore yang menerapkan model Basis Data NoSQL berbasis dokumen (*document-oriented database*). Struktur data disusun dalam bentuk *collection*, *document*, dan *subcollection* untuk menyimpan data pengguna, transaksi, konfigurasi aplikasi, serta informasi bank secara terorganisir.

B. Implementasi Middleware

Middleware diimplementasikan sebagai perantara antara aplikasi mobile dan server transaksi. Komponen ini bertugas menerima permintaan transaksi, menghasilkan *reference code*, memverifikasi *ID Token*, memvalidasi data *request*, serta meneruskan permintaan ke server. Apabila proses validasi gagal, middleware akan melakukan *rollback* saldo, memperbarui status transaksi, dan mengirimkan informasi kegagalan kepada pengguna.

C. Implementasi Server

Server diimplementasikan untuk memproses transaksi sesuai layanan yang dipilih pengguna. Setelah transaksi selesai diproses, server mengirimkan *callback* kepada middleware yang berisi status berhasil atau gagal sebagai dasar pembaruan status transaksi.

D. Implementasi Aplikasi Mobile

Aplikasi mobile diimplementasikan sebagai antarmuka pengguna untuk melakukan transaksi, mulai dari pemilihan produk, validasi data, pengecekan saldo, hingga menampilkan hasil transaksi berdasarkan respons yang diterima dari middleware.

E. Implementasi Mekanisme OCC

Mekanisme OCC diimplementasikan menggunakan transaksi Cloud Firestore yang memanfaatkan MVCC untuk mendeteksi perubahan data selama proses transaksi. Mekanisme ini memastikan setiap transaksi hanya dapat melakukan *commit* apabila data tidak mengalami perubahan, sehingga *race condition* dapat dicegah dan integritas data tetap terjaga.

3.1.5. Pengujian Sistem

Pengujian sistem merupakan tahapan untuk memastikan bahwa sistem yang telah diimplementasikan dapat berfungsi sesuai dengan kebutuhan fungsional serta mampu menerapkan mekanisme pencegahan *race condition* menggunakan *Optimistic Concurrency Control* (OCC). Pengujian dilakukan terhadap setiap komponen sistem, mulai dari proses transaksi, validasi data, hingga mekanisme sinkronisasi transaksi ketika terjadi akses secara bersamaan. Selain itu, pengujian juga bertujuan untuk memastikan bahwa integritas data tetap terjaga selama sistem memproses transaksi secara konkuren.

3.1.5.1. Pengujian Fungsional

Pengujian fungsional menggunakan Black Box Testing untuk memastikan setiap fungsi sistem berjalan sesuai dengan kebutuhan yang telah

dirancang.

A. Proses transaksi, Validasi saldo dan data request, Verifikasi ID Token
B. Pembaruan status transaksi, Mekanisme rollback ketika transaksi gagal
C. Memastikan setiap fungsi memberikan output sesuai skenario pengujian
D. Memastikan sistem dapat menangani kondisi berhasil maupun gagal dengan benar. Hasil pengujian digunakan untuk memastikan fungsi utama sistem berjalan sesuai spesifikasi.

.

3.1.5.2. Pengujian Konkurensi

Untuk membuktikan efektivitas mekanisme Optimistic Concurrency Control (OCC) berbasis Multi-Version Concurrency Control (MVCC) pada Cloud Firestore, dirancang sebuah skenario eksperimen pengujian konkurensi secara terukur.

A. Skenario Pengujian

Eksperimen konkurensi mensimulasikan kondisi persaingan data (*data contention*) di mana 2 *real device* mengirimkan permintaan (*request*) pemotongan saldo secara simultan (paralel) terhadap satu dokumen saldo pengguna yang sama (*single document account*) pada Cloud Firestore. Pengujian ini terbagi ke dalam 2 kelompok kondisi utama:

- **Kondisi Tanpa Sinkronisasi:** Pengujian pemotongan saldo dan *rollback* saldo yang dieksekusi secara bersamaan oleh 2 perangkat tanpa memanfaatkan fitur *Firestore Transaction*.
- **Kondisi Dengan Sinkronisasi:** Pengujian pemotongan saldo dan *rollback* saldo yang dieksekusi secara bersamaan oleh 2 perangkat dengan menerapkan fitur *Cloud Firestore Transaction*.

A. Kriteria Keberhasilan

Mekanisme pencegahan *race condition* dinyatakan BERHASIL dan

EFEKTIF apabila memenuhi kriteria matematis/kuantitatif berikut:

- Zero Lost Update (0 Lost Update): Tidak boleh ada pembaruan data yang hilang atau tertimpa secara tidak sah.
- Konsistensi Saldo Akhir secara Mutlak:

$$\overbrace{S_{akhir} = S_{awal} - \sum_{i=1}^{T_{valid}} D_i + \sum_{j=1}^{T_{rollback}} P_j}^{S_{Hasil}}$$

Keterangan Simbol:

S_{akhir}	=	Saldo akhir akun pada dokumen Cloud Firestore setelah seluruh pengujian selesai dieksekusi.
S_{awal}	=	Saldo awal akun pada dokumen Cloud Firestore sebelum pengujian dimulai.
S_{Hasil}	=	Nilai saldo akhir ekspektasi hasil dari kalkulasi matematis.
T_{valid}	=	Total jumlah transaksi pemotongan saldo yang berhasil/valid diproses oleh sistem ($T_{valid} \leq N$).
$T_{rollback}$	=	Total jumlah transaksi gagal yang berhasil di-*rollback* oleh sistem.
D_i	=	Nominal debit/pemotongan saldo pada transaksi sukses ke-i.
P_j	=	Nominal debit/penambahan saldo pada transaksi sukses ke-i.
i	=	Indeks iterasi transaksi yang berstatus sukses ($i = 1,2, \dots, T_{valid}$).

B. Integritas Konsistensi Data

Seluruh saldo yang terpotong berbanding lurus dengan jumlah riwayat transaksi yang berstatus *SUCCESS* pada *collection transaction*.

3.1.5.3. Pengujian Idempotency

Pengujian idempotency dilakukan dengan mengirimkan kembali request terhadap transaksi yang sebelumnya telah berhasil diproses untuk mengetahui apakah sistem dapat mencegah pemrosesan transaksi yang sama secara berulang. Pengujian dilakukan dengan menggunakan identitas transaksi ref_id yang sama dengan request sebelumnya. Kondisi data sebelum dan sesudah request ulang dibandingkan untuk memastikan bahwa transaksi tidak diproses atau saldo tidak dikurangi kembali. Aspek yang diverifikasi:

A. Pencegahan Pemrosesan Transaksi Ganda

   Memastikan request yang dikirim kembali dengan key yang sama tidak menyebabkan transaksi diproses untuk kedua kalinya. Sistem diharapkan mengenali bahwa request tersebut merupakan pengulangan dari transaksi yang telah diproses sebelumnya.

B. Keakuratan Saldo

   Memastikan saldo pengguna setelah request ulang tetap sama dengan saldo setelah transaksi pertama dan tidak mengalami pengurangan kembali akibat pengiriman request yang sama.

3.1.6. Evaluasi

Evaluasi merupakan tahapan akhir dalam metode penelitian yang bertujuan untuk menilai apakah sistem yang telah diimplementasikan mampu memenuhi tujuan penelitian. Evaluasi dilakukan berdasarkan hasil pengujian fungsional, pengujian mekanisme *Optimistic Concurrency Control* (OCC), dan pengujian integritas data untuk mengetahui efektivitas mekanisme yang diterapkan dalam mencegah *race condition* pada proses transaksi yang berjalan secara konkuren.

Pada tahap ini, hasil pengujian dianalisis dengan membandingkan kondisi sistem sebelum dan sesudah penerapan mekanisme OCC berbasis *Multi-Version Concurrency Control* (MVCC). Aspek yang dievaluasi meliputi keberhasilan sistem dalam mendeteksi konflik transaksi, menjaga konsistensi saldo pengguna, mencegah terjadinya *lost update*, serta memastikan bahwa status transaksi yang dihasilkan sesuai dengan kondisi sebenarnya. Hasil evaluasi tersebut selanjutnya

digunakan sebagai dasar dalam menarik kesimpulan mengenai efektivitas mekanisme OCC dalam menjaga integritas data pada aplikasi *Payment Point Online Bank* (PPOB).

3.2. Waktu Penelitian

Penelitian ini dilaksanakan selama November 2025 hingga juli 2026. Kegiatan penelitian diawali dengan identifikasi masalah dan studi literatur, kemudian dilanjutkan dengan perancangan sistem, implementasi, pengujian, serta evaluasi sistem. Tahapan penelitian tersebut disusun secara sistematis untuk memastikan tujuan penelitian dapat tercapai sesuai dengan ruang lingkup dan metodologi yang telah ditetapkan.

3.3. Alat dan Bahan

Alat dan bahan penelitian merupakan komponen yang digunakan untuk mendukung proses perancangan, implementasi, dan pengujian sistem. Alat penelitian terdiri atas perangkat keras (*hardware*) dan perangkat lunak (*software*), sedangkan bahan penelitian berupa referensi ilmiah, dokumentasi resmi, serta data yang digunakan sebagai dasar dalam pengembangan sistem.

3.3.1. Perangkat Keras

Perangkat keras merupakan komponen fisik yang digunakan untuk mendukung proses pengembangan, implementasi, dan pengujian aplikasi. Spesifikasi perangkat keras yang digunakan dalam penelitian ini disajikan pada Tabel 3.5.

Tabel 3. 5 Perangkat Keras Yang Digunakan

No.	Perangkat	Spesifikasi
1	Laptop	Lenovo ThinkPad, AMD Ryzen 7, RAM 8 GB, SSD 512 GB
2	Smartphone Android	Samsung Galaxy A54, RAM 8 GB, penyimpanan internal 128 GB, Android 14

3	Smartphone Android	Realme 5 Pro, RAM 8 GB, penyimpanan internal 128 GB, Android 11

3.3.2. Perangkat Lunak

Perangkat lunak merupakan aplikasi dan *tools* yang digunakan untuk mendukung proses pengembangan, implementasi, serta pengujian sistem. Perangkat lunak yang digunakan dalam penelitian ini disajikan pada Tabel 3.6.

Tabel 3. 6 Perangkat Lunak Yang Digunakan

No.	Perangkat Lunak	Versi
1	Windows	11 Pro 64-bit
2	Visual Studio Code	1.102.3
3	Android Studio	Meerkat 2024.3.1
4	React Native	0.79.2
5	Node.js	22.14.0
6	Java Development Kit (JDK)	17
7	Android SDK	API Level 36
8	Gradle	8.14.3
9	Cloud Firestore	22.2.1
11	Nginx	1.29.0
12	Postman	11.57.4
13	Git	2.50.1

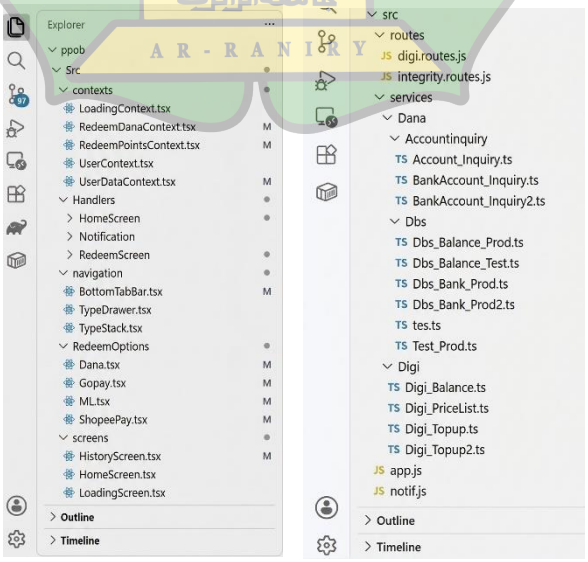
BAB IV
HASIL DAN PEMBAHASAN

4.1. Hasil Implementasi Sistem

Tahap implementasi sistem merupakan realisasi dari rancangan arsitektur, Basis Data, dan antarmuka yang telah disusun pada bab sebelumnya. Sistem yang dikembangkan adalah aplikasi Payment Point Online Bank (PPOB) berbasis mobile yang memanfaatkan Cloud Firestore sebagai Basis Data NoSQL utama. Sistem ini dirancang khusus untuk menangani transaksi pemotongan saldo secara simultan (concurrent transactions) tanpa mengorbankan integritas data.

4.1.1. Implementasi Arsitektur Sistem

Arsitektur sistem yang diimplementasikan terdiri atas dua bagian utama, yaitu client-side sebagai antarmuka aplikasi mobile dan server-side sebagai backend yang menangani proses bisnis serta komunikasi dengan Basis Data dan layanan pihak ketiga. Kedua bagian tersebut saling berkomunikasi melalui API untuk mendukung proses transaksi dan pengelolaan data aplikasi. Implementasi arsitektur ini dirancang secara modular agar setiap komponen memiliki fungsi dan tanggung jawab yang jelas.



Gambar 4. 1 Struktur Aplikasi

Berdasarkan gambar 4.1, pada sisi *client-side* (ppob/Src), aplikasi mobile disusun secara modular untuk mengelola data dan tampilan secara efisien. Lapisan ini mencakup pengelola status global (contexts/) seperti UserDataContext.tsx dan RedeemDanaContext.tsx yang menangani data profil pengguna serta *state* proses transaksi. Logika interaksi diakomodasi oleh pengelola *event* (Handlers/), yang terhubung dengan sistem penjelajahan halaman (navigation/) dan modul layanan digital (RedeemOptions/) seperti Dana.tsx, Gopay.tsx, dan ShopeePay.tsx. Seluruh elemen ini bermuara pada lapisan tampilan utama (screens/), termasuk HomeScreen.tsx untuk dashboard utama dan HistoryScreen.tsx untuk pemantauan riwayat transaksi.

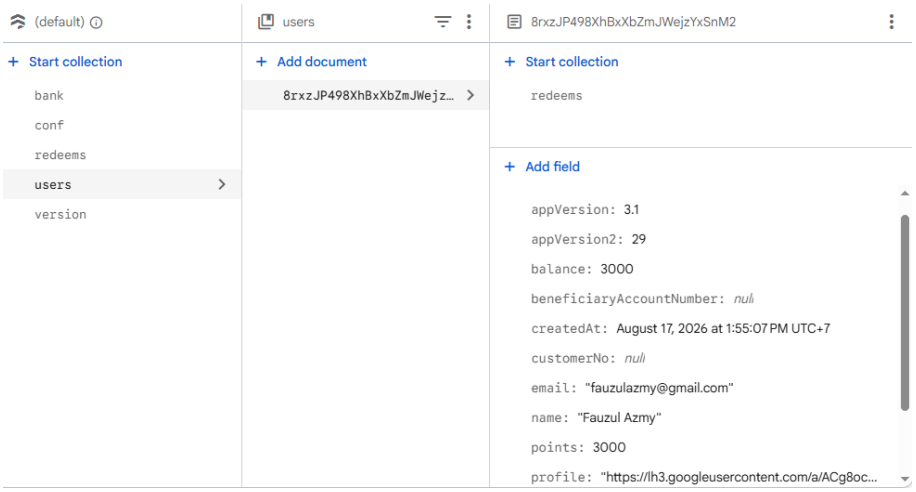
pada sisi *server-side* (src/), *backend middleware* dibangun menggunakan arsitektur berbasis layanan (*service-oriented architecture*) yang bertugas memproses permintaan API dan menghubungkan sistem dengan Cloud Firestore maupun penyedia layanan pihak ketiga. Berkas app.js bertindak sebagai *entry point* utama untuk menginisiasi server Express, yang kemudian mengarahkan permintaan HTTP melalui lapisan routing (routes/) seperti digi.routes.js dan integrity.routes.js. Pemrosesan logika bisnis diisolasi di dalam lapisan layanan (services/), yang terbagi menjadi modul pemrosesan transaksi e-wallet/bank (services/Dana/) serta modul transaksi produk digital (services/Digi/).

4.1.2. Implementasi Basis data

Implementasi Basis Data pada Cloud Firestore dirancang menggunakan struktur NoSQL Document-Oriented yang fleksibel dan berkinerja tinggi. Data diorganisasikan ke dalam beberapa collection utama serta subcollection untuk mendukung kebutuhan transaksi PPOB, validasi versi aplikasi, hingga integrasi data perbankan secara terpusat.

A. Collection Users

Collection users berfungsi sebagai tempat penyimpanan utama data profil pengguna dan status akun secara individual.

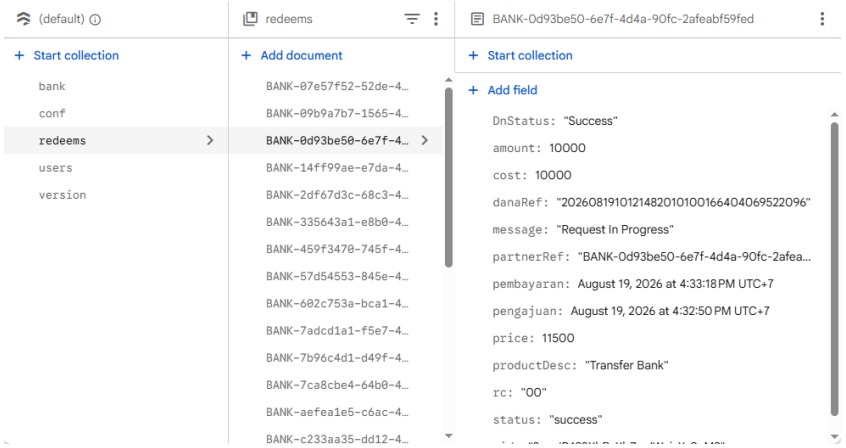


Gambar 4. 2 Collection Users

Gambar 4.2 Menunjukkan Setiap dokumen di dalam *collection* ini diidentifikasi menggunakan User ID (*UID*) unik dari proses autentikasi. Di dalamnya terdapat berbagai atribut (*field*) penting seperti name (nama pengguna, misalnya "Fauzul Azmy"). email, NomorRedeem (nomor telepon/akun penukaran), balance (saldo aktif pengguna), points (jumlah poin yang dimiliki), createdAt (waktu pembuatan akun), serta appVersion dan appVersion2 yang mencatat versi aplikasi.

B. Collection Redeems

Redeems pada tingkat *root* (*top-level collection*) digunakan untuk mengelola serta mengagregasi seluruh riwayat transaksi penukaran saldo dari seluruh pengguna ke dalam satu wadah global.

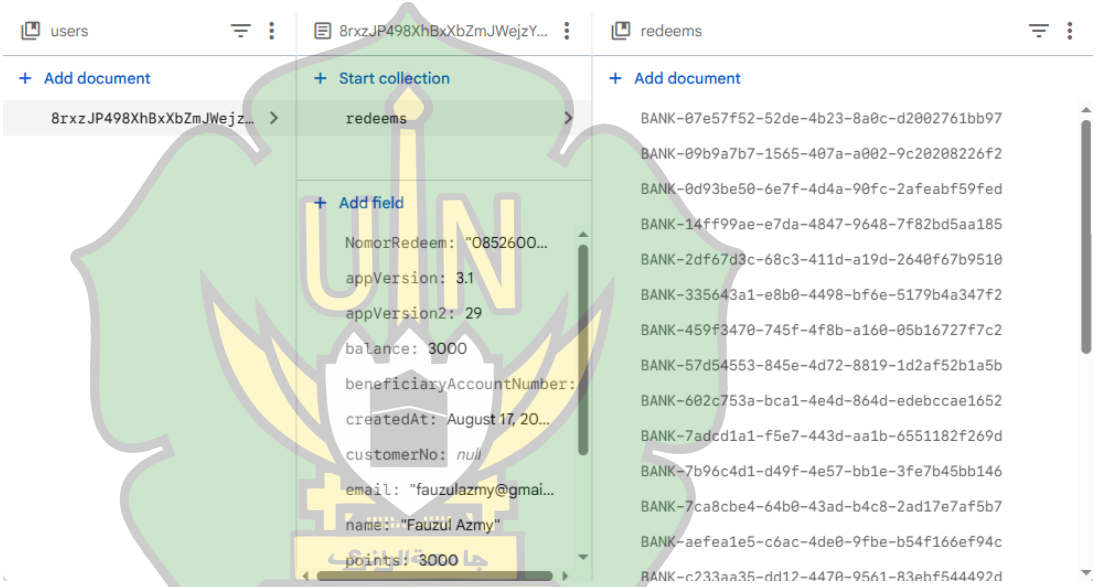


Gambar 4. 3 Collection Redeems

Seperti pada Gambar 4.3, Struktur ini mempermudah proses pemantauan transaksi secara terpusat oleh server *middleware*, audit integritas data, serta kompilasi laporan transaksi PPOB tanpa harus melakukan pemindaian (*scan*) ke seluruh dokumen pengguna secara individual.

C. SubCollection Redeems

Selain *collection* tingkat atas, sistem mengimplementasikan *subcollection* redeems yang terletak di dalam setiap dokumen pengguna pada *collection* users (users/{uid}/redeems).

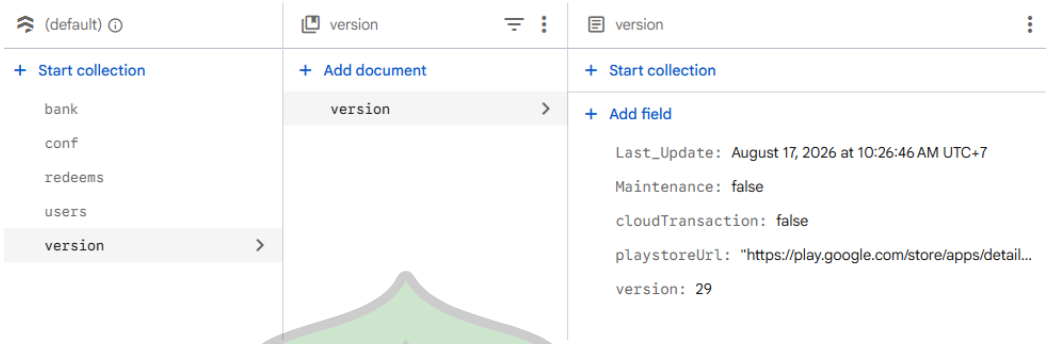


Gambar 4. 4 SubCollection Redeems

Berdasarkan Gambar 4.4, *Subcollection* ini menyimpan dokumen transaksi yang spesifik hanya milik pengguna tersebut, di mana ID dokumen diformat secara terstruktur (seperti BANK-07e57f52-... atau BANK-602c753a-...). Penataan hirarkis ini memastikan aplikasi mobile dapat memuat halaman riwayat transaksi pengguna (*HistoryScreen*) secara cepat dan hemat konsumsi *read operations* Cloud Firestore karena query terisolasi pada cakupan *document path* pengguna yang sedang aktif.

D. Collection Version

Collection version dibuat untuk menyimpan konfigurasi sistem, validasi kompatibilitas aplikasi, dan kontrol operasional secara *real-time*.

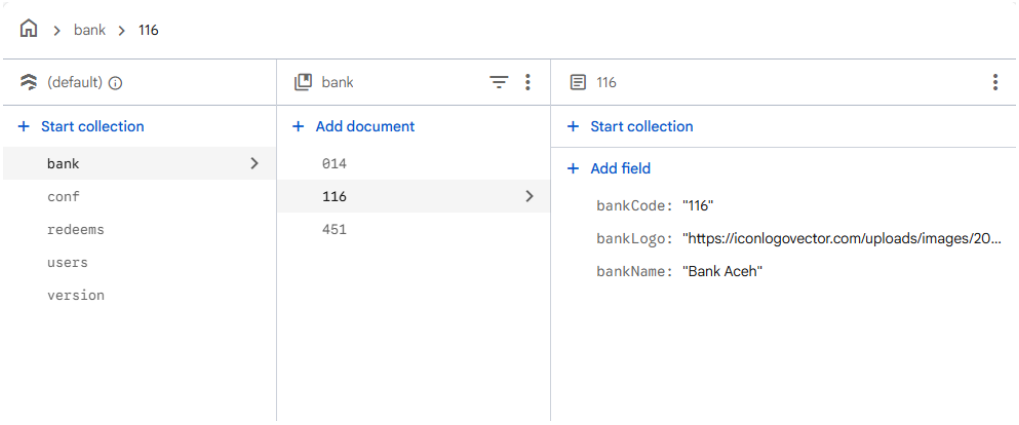


Gambar 4. 5 Collection Version

Pada Gambar 4.5, Dokumen di dalamnya memuat atribut version (versi aplikasi terbaru, seperti versi 29), Last_Update (stempel waktu pembaruan terakhir), playstoreUrl (tautan pembaruan aplikasi), serta dua atribut berjenis boolean yaitu Maintenance dan cloudTransaction. Atribut Maintenance berfungsi untuk menonaktifkan sementara akses transaksi aplikasi saat pemeliharaan sistem, sedangkan cloudTransaction mengontrol izin pemrosesan transaksi berbasis cloud secara terpusat.

E. Collection Bank

Collection bank menyimpan data referensi lembaga perbankan yang didukung oleh sistem untuk keperluan transfer saldo atau penukaran. Setiap dokumen mewakili satu kode bank tertentu (seperti dokumen 014, 116, atau 451). Gambar 4.6 di bawah menunjukkan dokumen yang tersimpan berupa informasi atribut bankCode (kode unik bank, misalnya "116"), bankName (nama bank, seperti "Bank Aceh"), serta bankLogo yang memuat URL tautan gambar logo bank untuk ditampilkan pada antarmuka aplikasi mobile.



Gambar 4. 6 Collection Bank

4.1.3. Implementasi Pencegahan Race Condition

Pencegahan *race condition* pada pemotongan saldo pengguna diimplementasikan pada lapisan *middleware* dengan memanfaatkan variabel kontrol cloudTransaction. Logika program dirancang ke dalam dua mekanisme eksekusi yang berbeda, yaitu eksekusi berbasis transaksi atomik (*Transactional*) sebagai mekanisme utama produksi, serta eksekusi tanpa transaksi (*Non-Transactional*) yang difungsikan khusus sebagai lingkungan pembanding untuk pengujian *race condition* dan *lost update*.

A. Cloud Firestore Transactional

Mekanisme ini diaktifkan ketika nilai cloudTransaction bernilai *true* dengan mengeksekusi metode db.runTransaction(). Dalam blok transaksi ini, Cloud Firestore membaca snapshot dokumen pengguna (transaction.get(userRef)) dan mencatat versi dokumennya secara otomatis berbasis *Optimistic Concurrency Control* (OCC).

```
if (cloudTransaction) {
        await db.runTransaction(async (transaction)
   => {
              const      userSnap      =      await
   transaction.get(userRef);
         if (!userSnap.exists) {
           throw new Error("USER_NOT_FOUND");
```

```
      const userData = userSnap.data();
                        const    currentBalance    =
    Number(userData?.balance ?? 0);
      const transferAmount = Number(cost);
           if (currentBalance < transferAmount) {
            throw new Error("INSUFFICIENT_BALANCE");
           }
        const   newBalance   =   currentBalance   -
      transferAmount;
           transaction.update(userRef, {
             balance: newBalance,
           });
         });
       }
```

Seperti pada Kode di atas, setelah memeriksa keberadaan dokumen dan memvalidasi bahwa saldo saat ini (currentBalance) mencukupi nominal pemotongan (transferAmount), transaksi menyiapkan saldo baru (newBalance) dan memperbarui dokumen menggunakan transaction.update(). Jika selama proses ini terjadi perubahan data saldo oleh transaksi paralel lain, Cloud Firestore secara otomatis membatalkan transaksi (*rollback*) dan melakukan percobaan ulang (*retry*) menggunakan snapshot data versi terbaru, sehingga integritas saldo tetap terjaga dan tidak terjadi *lost update*.

B. Cloud Firestore Non-Transactional

Mekanisme ini dieksekusi ketika nilai cloudTransaction bernilai *false* dan sengaja dibuat tanpa menggunakan pemrosesan atomik untuk mensimulasikan terjadinya *race condition* saat pengujian beban.

```
else {
 const userSnap = await userRef.get();
     if (!userSnap.exists) {
        throw new Error("USER_NOT_FOUND"); }
```

```
const userData = userSnap.data();
const currentBalance = Number(userData?.balance ??
    0);
const transferAmount = Number(cost);
        if (currentBalance < transferAmount) {
          throw new Error("INSUFFICIENT_BALANCE");
        }
        await sleep(3000);
const newBalance = currentBalance - transferAmount;
        // UPDATE BIASA, TANPA TRANSACTION
        await userRef.update({
          balance: newBalance,
        });
      }
```

Kode di atas menunjukkan pembacaan dokumen dilakukan menggunakan operasi *get* biasa (userRef.get()) di luar blok transaksi, dilanjutkan dengan pemeriksaan saldo dan jeda buatan (sleep(3000)) selama 3 detik untuk memperlebar rentang waktu (*time window*) datangnya permintaan paralel lain. Setelah jeda, sistem memperbarui saldo menggunakan userRef.update() secara langsung. Tanpa adanya validasi versi data dari OCC, dua atau lebih permintaan paralel yang membaca nilai currentBalance yang sama akan saling menimpa data saldo akhir (*lost update*), yang membuktikan pentingnya penggunaan transaksi atomik pada sistem PPOB.

4.1.4. Implementasi API Transaksi

Implementasi API transaksi dilakukan untuk menyediakan layanan komunikasi antara aplikasi dengan server backend dalam menjalankan proses transaksi. API yang diimplementasikan mencakup proses pemeriksaan rekening tujuan, permintaan transfer bank, top up saldo melalui e-wallet, serta penerimaan notifikasi hasil transaksi melalui webhook callback.

Pada tahap pengujian, backend dijalankan pada lingkungan pengembangan lokal. Untuk mensimulasikan kondisi endpoint yang dapat diakses dari jaringan publik sebagaimana pada lingkungan production, digunakan mekanisme **tunneling menggunakan ngrok**. Ngrok berfungsi meneruskan request dari alamat publik menuju server backend yang berjalan pada komputer lokal. Dengan demikian, layanan eksternal dapat mengakses endpoint backend melalui protokol HTTP/HTTPS tanpa backend harus terlebih dahulu ditempatkan pada server production.

Penggunaan tunneling ini terutama diperlukan pada pengujian API yang melibatkan layanan eksternal dan mekanisme webhook callback. Layanan eksternal membutuhkan alamat endpoint yang dapat diakses melalui internet untuk mengirimkan hasil transaksi kepada backend. Dengan menggunakan ngrok, endpoint lokal dapat dipublikasikan sementara sehingga alur komunikasi antara layanan eksternal dan backend dapat disimulasikan menyerupai kondisi production.

A. API Transaksi Account Inquiry Transfer Bank

Endpoint account inquiry digunakan untuk melakukan pemeriksaan terhadap rekening bank tujuan sebelum transaksi transfer dilakukan. Proses ini digunakan untuk memperoleh informasi rekening tujuan dan memastikan bahwa rekening yang dimasukkan oleh pengguna dapat diproses oleh layanan transfer. Endpoint yang digunakan dalam pengujian adalah:

```
https://<ngrok-domain>/bank/account-inquiry
```

Ketika aplikasi mengirimkan permintaan account inquiry, backend menerima data rekening tujuan dan meneruskan permintaan tersebut kepada layanan pembayaran. Hasil pemeriksaan kemudian dikembalikan oleh backend kepada aplikasi sehingga pengguna dapat mengetahui informasi rekening tujuan sebelum melanjutkan transaksi.

B. API Transaksi Request Transfer Bank

Endpoint request transfer bank digunakan untuk melakukan permintaan transaksi transfer ke rekening bank tujuan. Endpoint ini merupakan bagian utama dalam proses transaksi transfer bank karena digunakan untuk

mengirimkan instruksi transaksi kepada layanan pembayaran. Endpoint yang digunakan dalam pengujian adalah:

```
https://<ngrok-domain>/bank/topup
```

Backend menerima data transaksi dari aplikasi, melakukan validasi terhadap data yang diperlukan, kemudian meneruskan permintaan transfer kepada layanan pembayaran. Setelah permintaan diproses, backend menerima respons awal dari layanan tersebut dan menyimpan informasi transaksi yang diperlukan untuk proses selanjutnya. Proses transfer tidak hanya bergantung pada respons awal tersebut karena status akhir transaksi dapat diterima melalui mekanisme callback. Oleh karena itu, endpoint request transfer dan webhook callback merupakan bagian yang saling berkaitan dalam implementasi transaksi.

C. API Transaksi Topup E-Wallet

Endpoint top up e-wallet digunakan untuk melakukan transaksi pengisian saldo ke akun e-wallet pengguna. Dalam implementasi ini, salah satu layanan e-wallet yang digunakan adalah DANA. Endpoint yang digunakan dalam pengujian adalah:

```
https://<ngrok-domain>/dana/topup
```

Aplikasi mengirimkan permintaan top up kepada backend. Backend kemudian melakukan validasi terhadap data transaksi dan meneruskan permintaan kepada layanan DANA. Informasi transaksi kemudian dicatat oleh backend untuk digunakan dalam proses pemantauan status transaksi.

D. Webhook Callback Transaksi

Webhook callback digunakan sebagai endpoint penerima notifikasi dari layanan pembayaran ketika terjadi perubahan atau penyelesaian status transaksi. Endpoint ini memiliki peranan penting karena hasil akhir transaksi dapat dikirimkan oleh pihak penyedia layanan pembayaran secara asynchronous kepada backend. Endpoint callback yang digunakan dalam pengujian adalah:

```
https://<ngrok-domain>/cb/v1.0/disbursement/notify
```

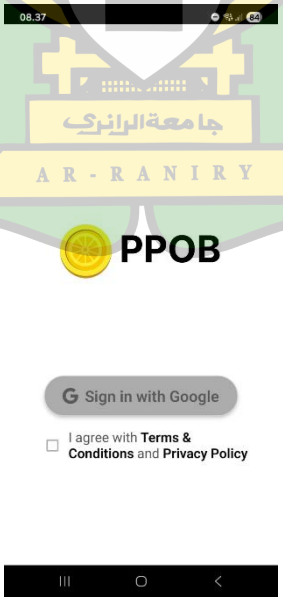
Endpoint tersebut dipublikasikan melalui ngrok agar dapat diakses oleh layanan pembayaran dari jaringan internet. Ketika layanan pembayaran mengirimkan callback, request akan diteruskan oleh ngrok menuju server backend yang berjalan pada lingkungan pengembangan. Backend kemudian menerima callback, melakukan validasi terhadap informasi transaksi, dan memperbarui status transaksi pada Basis Data. Mekanisme ini memungkinkan status transaksi pada sistem diperbarui berdasarkan informasi dari layanan pembayaran.

4.1.5. Implementasi Antarmuka Pengguna

Implementasi antarmuka pengguna (*User Interface*) dirancang menggunakan framework React Native untuk memberikan pengalaman pengguna yang responsif, intuitif, dan konsisten di berbagai perangkat mobile. Antarmuka ini menghubungkan pengguna secara langsung dengan fungsi-fungsi utama aplikasi PPOB, mulai dari autentikasi akun, navigasi layanan digital, hingga eksekusi transaksi keuangan secara *real-time*.

A. Implementasi Login Screen

Halaman *Login Screen* yang diilustrasikan pada Gambar 4.7 mengintegrasikan metode autentikasi ringkas dan aman berbasis *Google Sign-In*.

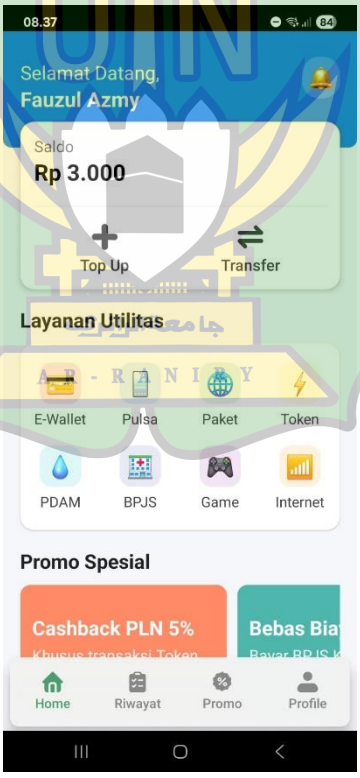


Gambar 4. 7 Antarmuka Login Screen

Antarmuka ini menyajikan logo utama aplikasi ("PPOB"), tombol sekali klik *Sign in with Google*, serta kotak persetujuan (*checkbox*) untuk syarat dan ketentuan layanan (*Terms & Conditions and Privacy Policy*). Rancangan ini bertujuan untuk mempercepat proses pendaftaran maupun masuknya pengguna ke dalam sistem tanpa harus mengisi formulir manual yang rumit.

B. Implementasi Home Screen

Visualisasi Gambar 4.8 memperlihatkan antarmuka *Home Screen* yang difungsikan sebagai dasbor utama aplikasi. Pada halaman ini, tertera informasi penting meliputi nama pengguna ("Fauzul Azmy"), ikon notifikasi, dan kartu status saldo PPOB aktif (Rp 3.000). Pengguna juga disediakan dua tombol aksi cepat yaitu *Top Up* dan *Transfer*, grid menu **Layanan Utilitas** (E-Wallet, Pulsa, Paket, Token, PDAM, BPJS, Game, Internet), spanduk **Promo Spesial**, serta *Bottom Navigation Bar* untuk kemudahan akses ke seluruh fitur aplikasi.

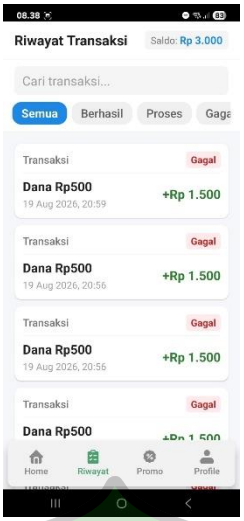


Gambar 4. 8 Antarmuka Home Screen

C. Implementasi History Screen

Struktur antarmuka *History Screen* disajikan pada Gambar 4.9 untuk

memfasilitasi pemantauan riwayat transaksi pengguna secara terorganisir.



Gambar 4. 9 Antarmuka History Screen

Halaman ini dilengkapi dengan fitur pencarian, bilah filter status (Semua, Berhasil, Proses, Gagal), serta deretan kartu riwayat transaksi yang memuat informasi nama produk (seperti "Dana Rp500"), nominal transaksi (+Rp 1.500), stempel waktu ("19 Aug 2026, 20:59"), dan indikator label berwarna merah yang menandai status transaksi gagal.

D. Implementasi Promo Screen

Tampilan halaman *Promo Screen* ("Promo & Voucher Spesial") ditunjukkan oleh Gambar 4.10, menyajikan daftar penawaran diskon serta *cashback* interaktif.

Gambar 4. 10 Antarmuka Promo Screen

Antarmuka ini dibekali filter kategori (Semua, PLN, Pulsa, E-Wallet) dan kartu promo yang menginformasikan besaran potongan (seperti "5% Max Rp 10.000"), detail deskripsi, batas waktu berlaku, kode voucher unik (PLNHEMAT / PULSAMURAH), serta tombol *Pakai* untuk menerapkan promo secara otomatis saat transaksi.

E. Implementasi Menu Deposit Aplikasi

Berdasarkan Gambar 4.11, Menu Deposit diimplementasikan dalam bentuk jendela pop-up (*modal dialog*) "Top Up Saldo" yang muncul ketika opsi *Top Up* pada dasbor dipilih.

Gambar 4. 11 Antarmuka Menu Deposit

Antarmuka dialog ini memuat kolom input nominal manual (Rp 0), tombol pilihan nominal instan (mulai dari Rp 5.000, Rp 10.000, Rp 20.000, hingga Rp 50.000), dan tombol eksekusi *Top Up* untuk memicu alur penambahan saldo akun.

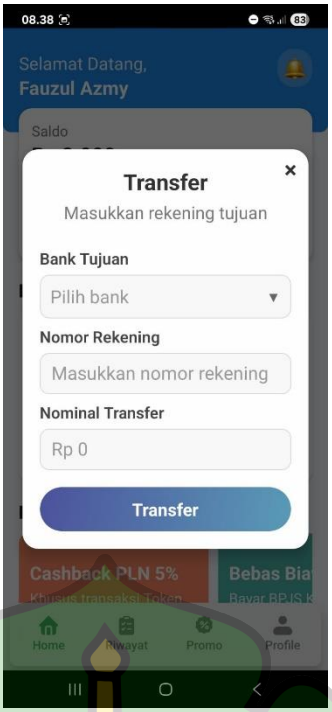
F. Implementasi Menu Topup E-Wallet

Pada Gambar 4.12, tertera rancangan antarmuka *Topup E-Wallet* yang dikhususkan untuk pengisian ulang saldo dompet digital seperti DANA, GoPay, ShopeePay, dan Mobile Legends. Halaman ini menginformasikan sisa saldo aktif (Rp 3.000), bilah pilihan *provider*, serta katalog produk yang mencantumkan nama denom ("DANA Rp 500"), rasio poin (3000/1500), dan rincian harga final yang harus dibayar pengguna (Rp 1.500).

Gambar 4. 12 Antarmuka Menu Topup E-Wallet

G. Implementasi Menu Transfer Bank

Bentuk interaktif dari Menu Transfer Bank ditunjukkan pada Gambar 4.7 berupa jendela pop-up (*modal dialog*) "Transfer" untuk kebutuhan pengiriman uang antarbank. Formulir pada antarmuka ini terdiri dari menu *dropdown* pemilihan bank tujuan ("Pilih bank"), kolom input nomor rekening target ("Masukkan nomor rekening"), kolom masukan nominal dana (Rp 0), serta tombol perintah *Transfer* yang akan meneruskan instruksi ke server *backend*.



Gambar 4. 13 Antarmuka Menu Transfer Bank

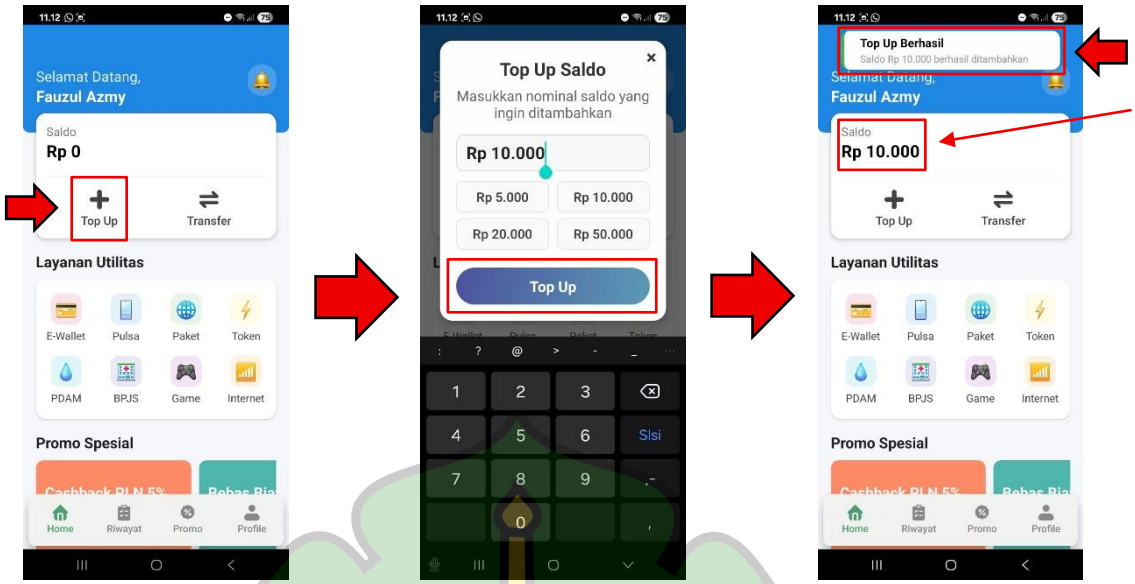
4.2. Hasil Pengujian Fungsional

Pengujian fungsional dilakukan untuk memastikan setiap fungsi utama pada aplikasi dapat berjalan sesuai dengan kebutuhan dan rancangan sistem yang telah ditentukan. Pengujian dilakukan dengan menjalankan setiap proses transaksi mulai dari pengguna melakukan permintaan transaksi, sistem memproses permintaan melalui API, hingga sistem memperbarui status dan data transaksi pada Basis Data. Pengujian fungsional meliputi transaksi deposit aplikasi, transfer bank, dan top up e-wallet.

4.2.1. Pengujian Transaksi Deposit Aplikasi

Pengujian transaksi deposit aplikasi dilakukan untuk memastikan proses penambahan saldo pengguna dapat berjalan sesuai dengan alur yang telah dirancang. Berdasarkan Gambar 4.14 di bawah, Pengujian mencakup proses pengguna melakukan permintaan deposit, sistem memproses transaksi, serta memastikan saldo pengguna diperbarui setelah transaksi berhasil. Hasil pengujian

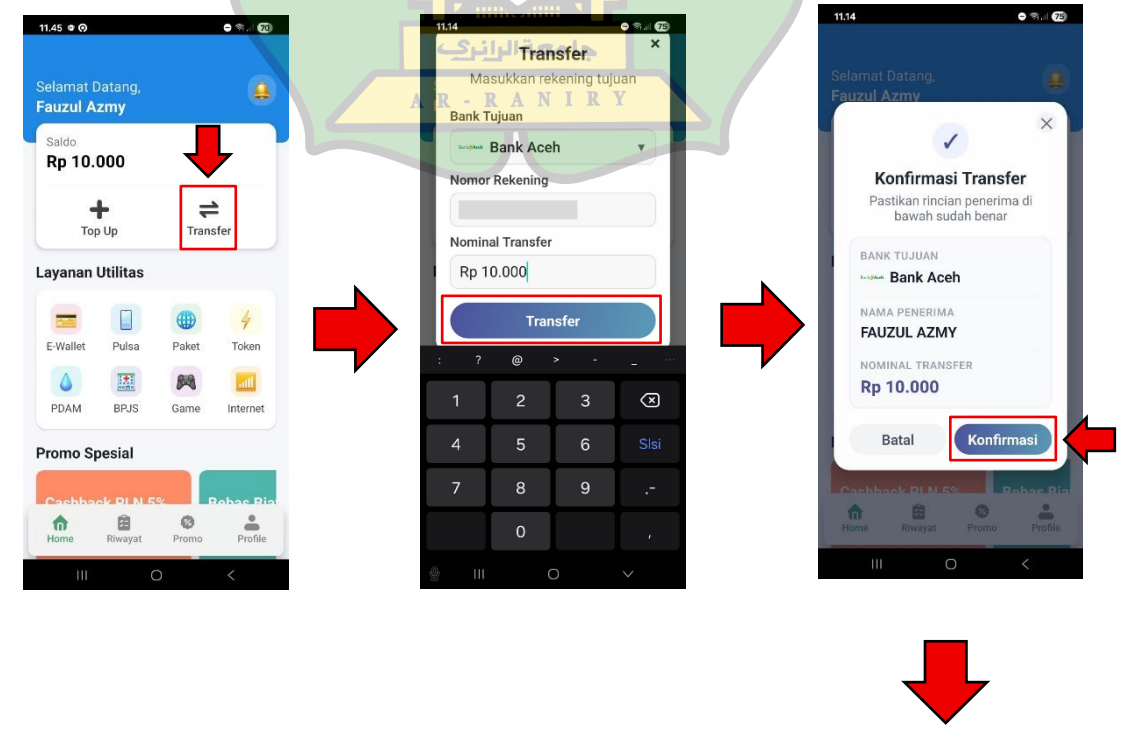
digunakan untuk memastikan data transaksi dan saldo pengguna tercatat dengan benar pada sistem.

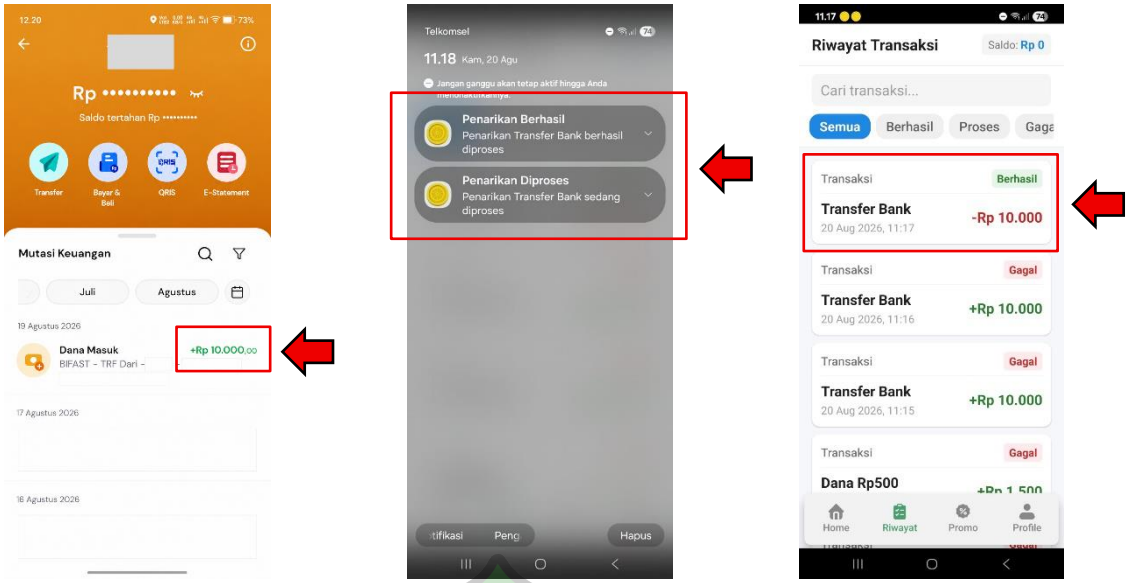


Gambar 4. 14 Pengujian Deposit Aplikasi

4.2.2. Pengujian Transaksi Transfer Bank

Pengujian transaksi transfer bank dilakukan untuk memastikan sistem dapat menjalankan proses transfer dana ke rekening bank tujuan.



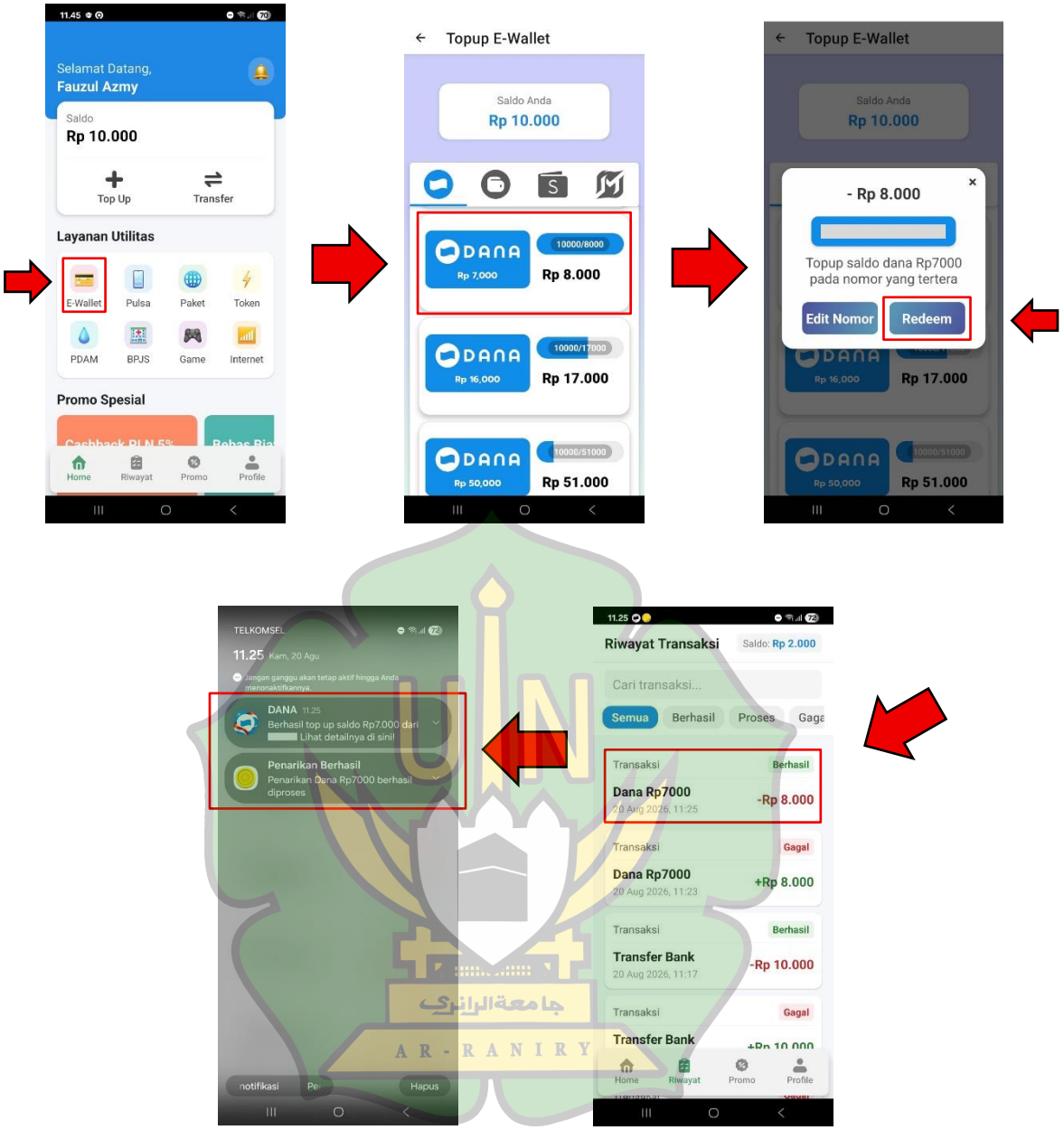


Gambar 4. 15 Tahapan Transfer Bank

Gambar 4.15 menunjukkan proses pengujian transaksi transfer bank yang meliputi pemeriksaan rekening tujuan melalui **account inquiry**, pengiriman permintaan transfer, pemrosesan transaksi oleh layanan pembayaran, hingga penerimaan status transaksi melalui **webhook callback**. Pengujian dilakukan untuk memastikan bahwa setiap tahapan transaksi dapat berjalan sesuai dengan alur yang telah dirancang, mulai dari validasi rekening tujuan hingga transaksi memperoleh status akhir. Selain itu, pengujian juga memastikan bahwa respons dari layanan pembayaran dapat diterima dan diproses oleh backend dengan baik, sehingga status transaksi pada sistem dapat diperbarui sesuai dengan kondisi transaksi yang sebenarnya.

4.2.3. Pengujian Transaksi Topup E-Wallet

Pengujian transaksi top up e-wallet dilakukan untuk memastikan sistem dapat memproses pengisian saldo ke akun e-wallet tujuan. Pengujian mencakup pengiriman permintaan top up dari aplikasi, pemrosesan transaksi oleh backend, komunikasi dengan layanan e-wallet, serta pembaruan status transaksi berdasarkan hasil pemrosesan. Hasil pengujian digunakan untuk memastikan transaksi top up dapat dilakukan sesuai alur yang telah dirancang dan informasi transaksi tersimpan dengan benar pada sistem.



Gambar 4. 16 Tahapan Topup E-Wallet

4.3. Hasil Pengujian Konkurensi

Pengujian dilakukan dengan membandingkan dua kondisi, yaitu sistem tanpa menggunakan Cloud Firestore Transaction dan sistem dengan menggunakan Cloud Firestore Transaction. Pengujian dilakukan melalui lingkungan **real device**. Pengujian real device digunakan untuk mensimulasikan kondisi penggunaan aplikasi secara langsung oleh pengguna,. Setiap kondisi pengujian terdiri atas skenario transaksi berhasil dan skenario transaksi gagal yang membutuhkan proses

penambahan atau rollback saldo.

4.3.1. Tanpa Sinkronisasi

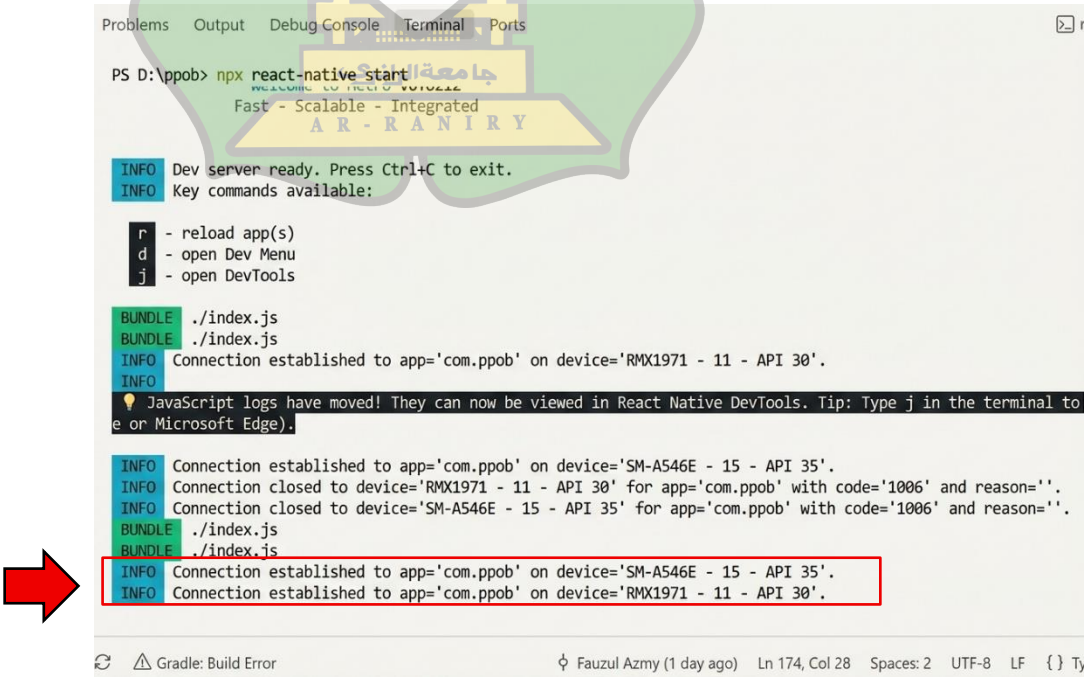
Pengujian konkurensi dengan mengaktifkan mekanisme *Cloud Firestore Transaction* dilakukan pada dua perangkat fisik terpisah secara simultan untuk membuktikan pencegahan *race condition* dan *double spending*.

A. Pengujian Pengurangan Saldo (Skenario Transaksi Berhasil)

Pengujian konkurensi pada perangkat fisik (*real device*) dilakukan untuk mengidentifikasi potensi *race condition* (kondisi balapan) ketika dua permintaan transaksi dikirimkan secara bersamaan tanpa mekanisme *locking* atau transaksi atomik (*Cloud Firestore Transaction*). Pengujian ini mencakup dua skenario layanan utama, yaitu layanan Transfer Bank dan Topup E-Wallet.

1) Inisialisasi Koneksi Multi-Device (Frontend)

Berdasarkan log Metro Bundler pada Gambar 4.17, pengujian diawali dengan menghubungkan dua perangkat fisik berbeda secara bersamaan ke server aplikasi, yaitu perangkat RMX1971 (Android API 30) dan SM-A546E (Android API 35).

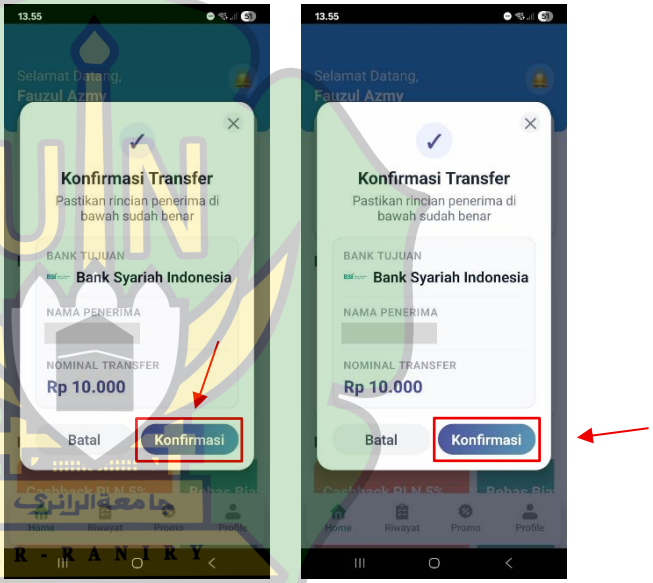


Gambar 4. 17 Log Metro Bundler

Hal ini membuktikan bahwa dua sesi pengguna aktif dapat melakukan pengiriman permintaan (*request*) secara paralel dari dua fisik perangkat terpisah.

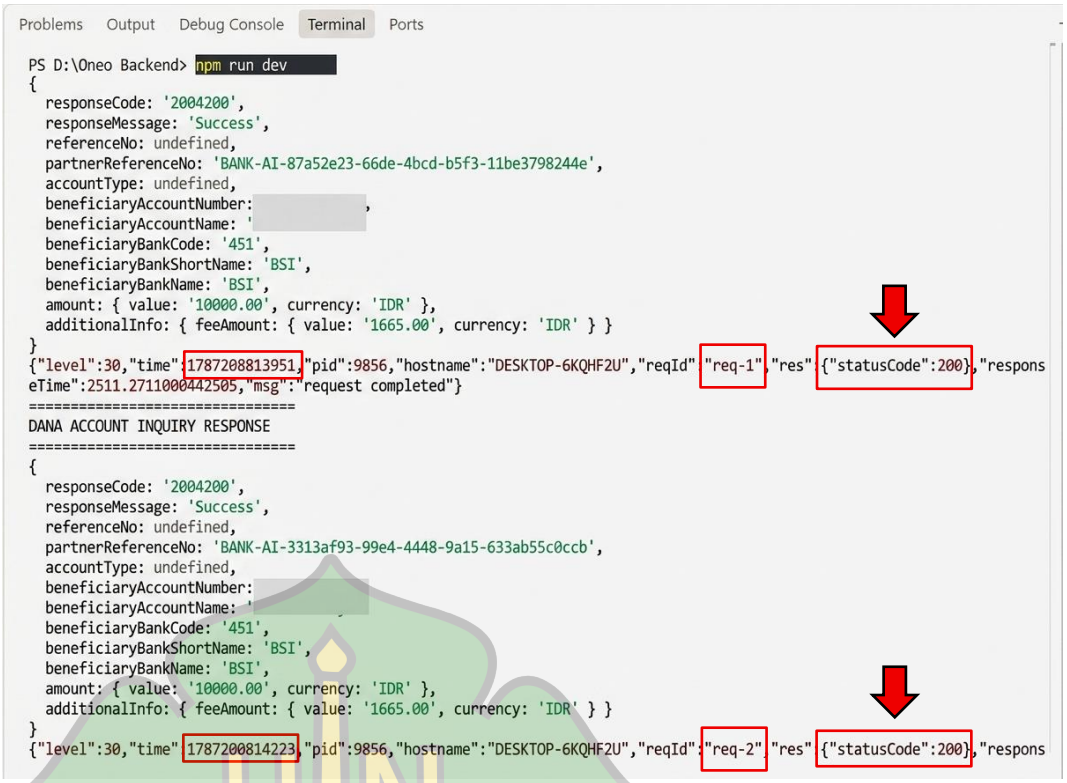
2) Skenario Transaksi Transfer Bank

- Transaksi diawali dengan saldo pengguna sebesar Rp 10.000 seperti pada gambar 4.15.
- Pada gambar 4.18, dua device pengguna melakukan konfirmasi transfer bank sebesar Rp 10.000 ke rekening Bank Syariah Indonesia (BSI) pada waktu bersamaan.



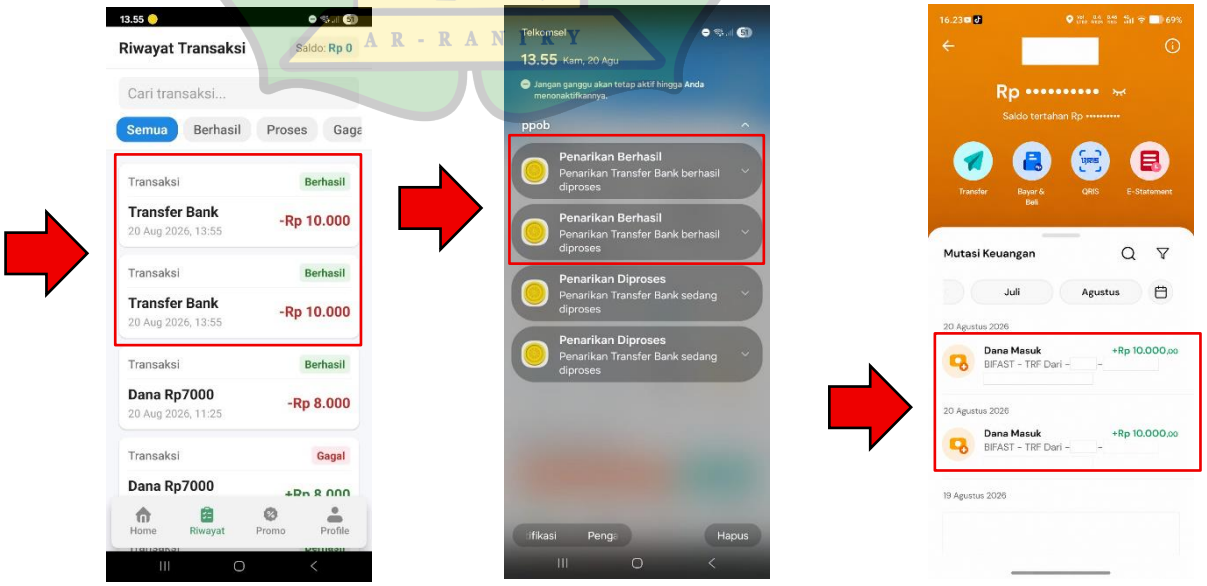
Gambar 4. 18 Dua Device Konfirmasi Transfer Bank

- Berdasarkan Gambar 4.19 Log terminal backend mencatat bahwa dua permintaan transaksi *inquiry* (req-1 dan req-2) masuk dan diproses hampir bersamaan pada timestamp yang sama (1787208813951 dan 1787208814223). Kedua *request* ini mengembalikan respon responseCode: 2004200 (Success).



Gambar 4. 19 Log Terminal Backend Transfer Bank

- Pada gambar 4.20 di bawah, sistem mengirimkan ganda notifikasi "Penarikan Berhasil" dan "Penarikan Diproses" pada waktu bersamaan (13:55).



Gambar 4. 20 Notifikasi Transaksi Transfer Bank

Riwayat transaksi mencatat dua kali pemotongan saldo berturut-turut masing-masing -Rp 10.000 pada tanggal 20 Aug 2026 jam 13:55, sehingga total nilai transaksi yang lolos mencapai Rp 20.000 meskipun saldo pengguna pada awalnya hanya mencukupi untuk satu kali transaksi.

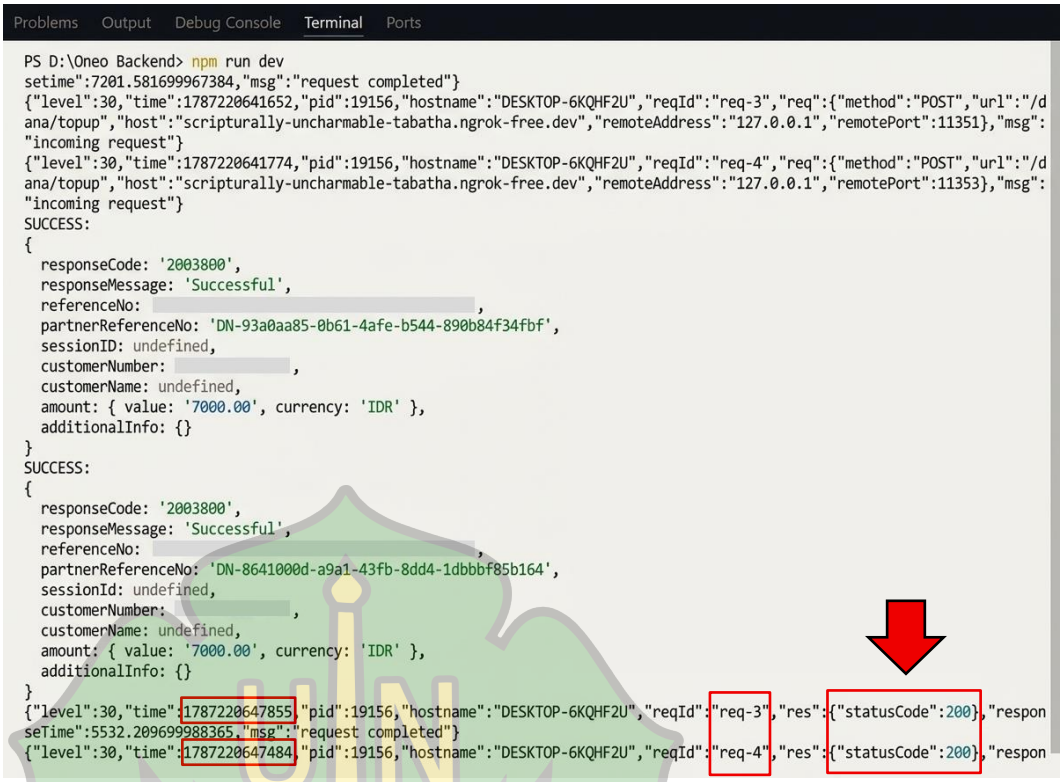
3) Skenario Transaksi Topup E-Wallet

- Transaksi diawali dengan saldo Rp 10.000 seperti pada gambar 4.16.
- Pada gambar 4.21, dua device pengguna mengeksekusi *redeem* Topup DANA sebesar Rp 7.000 dengan potongan saldo Rp 8.000 pada waktu bersamaan.

Gambar 4. 21 Dua Device Top up Saldo E-Wallet

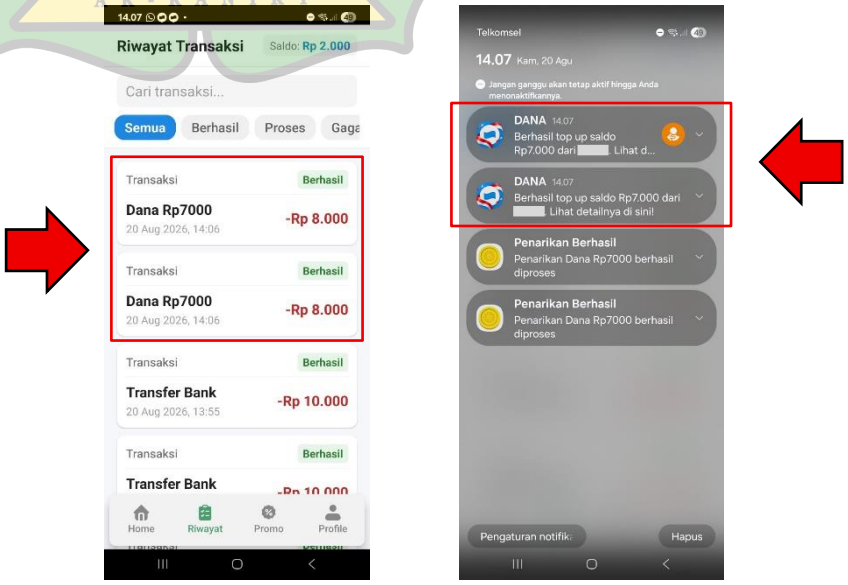
- Log backend pada Gambar 4.22 memperlihatkan dua permintaan *topup* DANA (/dana/topup) yang masuk secara konkuren (req-3 dan req-4) dan berhasil diproses oleh sistem dengan status SUCCESS:. Kedua transaksi mengembalikan responseCode: '2003800' dan responseMessage: 'Successful' untuk dua nilai partnerReferenceNo yang berbeda, yaitu DN-93a0... dan DN-8641... Kedua pemrosesan tersebut berhasil

diselesaikan dengan *status code* 200 (statusCode: 200).



Gambar 4. 22 Log Terminal Backend Top up Saldo E-Wallet

- Seperti terlihat pada Gambar 4.23, pengguna menerima dua kali notifikasi masuk dari aplikasi DANA "Berhasil top up saldo Rp7.000 pada jam 14:07.



Gambar 4. 23 Notifikasi Transaksi Top up Saldo E-Wallet

Pada halaman riwayat aplikasi, kedua transaksi tercatat dengan status **Berhasil** diproses pada waktu bersamaan (14:06) dengan pemotongan ganda -Rp 8.000 dan -Rp 8.000.

B. Pengujian Rollback Saldo (Skenario Transaksi Gagal)

Skenario pengujian ini dirancang untuk mensimulasikan kegagalan transaksi pada sisi server penyedia layanan dengan sengaja memanipulasi format data (*field*) pada *payload* yang dikirimkan. Pengujian ini bertujuan untuk menguji keandalan mekanisme penambahan kembali saldo (*rollback*) menggunakan fungsi *increment* ketika transaksi dibatalkan atau mengalami *error*.

1) Simulasi Error Payload dan Respon Server Backend

Berdasarkan log terminal backend pada Gambar 4.24, ketika dua permintaan diproses secara bersamaan (req-5 dan req-6), server penyedia layanan mengembalikan respon kegagalan dengan kode responseCode: '4004302' dan pesan responseMessage: 'Invalid Mandatory Field beneficiaryBankCode'.

Gambar 4. 24 Log Terminal Backend Invalid Field Transfer Bank

backend aplikasi tetap menyelesaikan siklus eksekusi dengan statusCode: 200 untuk memicu logika *rollback* penambahan kembali saldo yang sebelumnya telah terpotong.

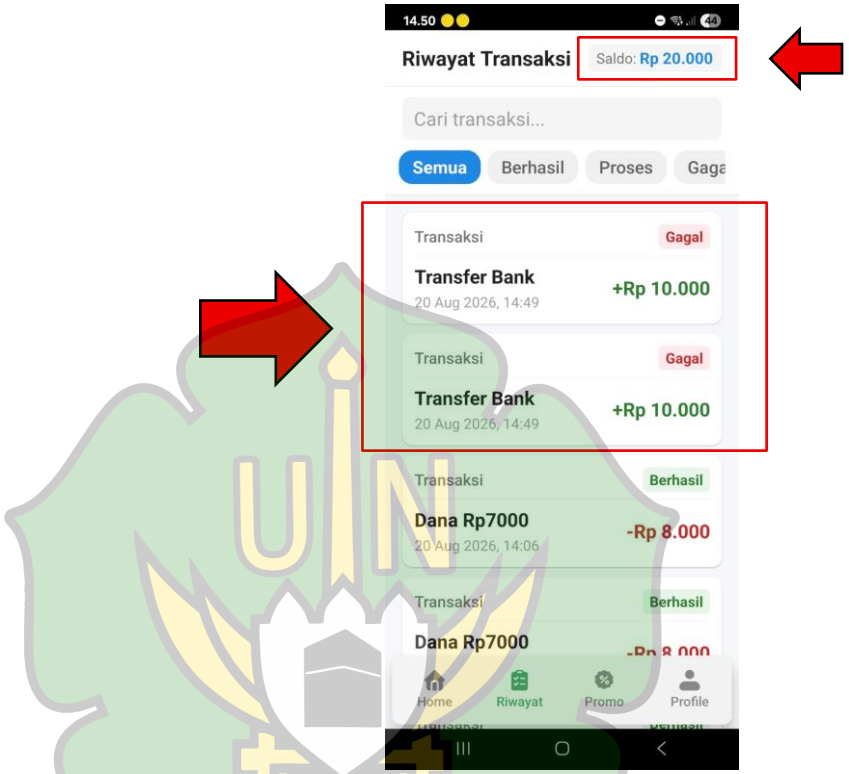
2) Skenario Rollback pada Layanan Transfer Bank

- Dua device Pengguna mengonfirmasi transaksi Transfer Bank BSI nominal Rp 10.000 pada waktu bersamaan seperti pada Gambar 4.18. Akibat kesalahan format *payload*, sistem membatalkan transaksi dan mengirimkan notifikasi ganda "Penarikan Gagal: Penarikan Transfer Bank gagal, coba lagi nanti" seperti pada Gambar 4.25 di bawah.

Gambar 4. 25 Notifikasi Transaksi Gagal Transfer Bank

- Pada awal transaksi, saldo pengguna bernilai Rp 10.000. Saat dua permintaan dikirimkan secara bersamaan tanpa *Cloud Firestore Transaction*, saldo awal dipotong oleh kedua transaksi. Namun, ketika respon *error* diterima, sistem mengeksekusi *rollback* menggunakan fungsi *increment*

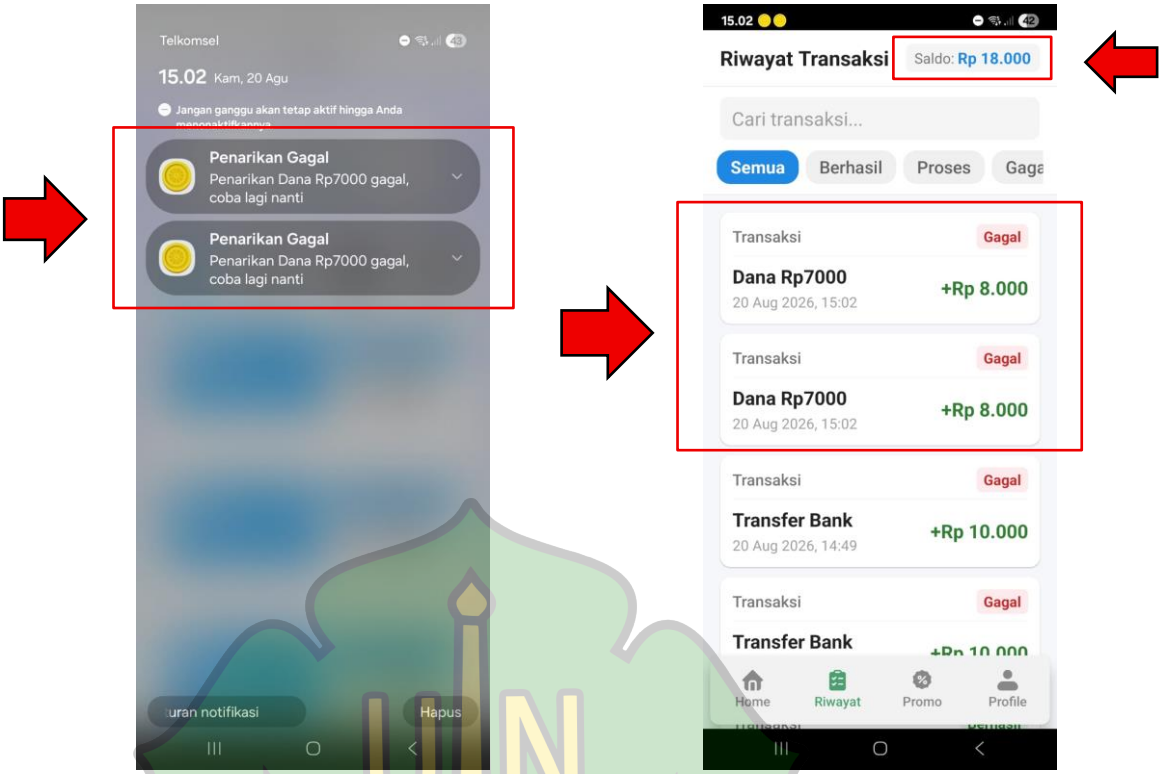
(+10.000) untuk masing-masing transaksi. Berdasarkan Gambar 4.26 karena kedua permintaan memicu *increment* secara terpisah, saldo akhir pengguna melonjak menjadi **Rp 20.000**. Terdapat dua catatan riwayat transaksi gagal dengan pemulihan saldo masing-masing +Rp 10.000.



Gambar 4. 26 Notifikasi Transaksi Gagal Transfer Bank

3) Skenario Rollback pada Layanan Topup E-Wallet

- Dua device pengguna melakukan skenario penarikan Topup DANA sebesar Rp7.000 dengan potongan saldo sebesar Rp8.000 pada waktu yang bersamaan, seperti ditunjukkan pada Gambar 4.21. Pada kondisi tersebut, terjadi kegagalan sistem yang menyebabkan kedua permintaan transaksi tidak dapat diselesaikan dengan baik. Kegagalan tersebut kemudian memicu munculnya notifikasi ganda "Penarikan Gagal: Penarikan Dana Rp7.000 gagal, coba lagi nanti" pada kedua perangkat, sebagaimana ditunjukkan pada Gambar 4.27.



Gambar 4. 27 Notifikasi Dan History Transaksi Gagal Transfer Bank

- Sebelum pengujian ini dilakukan, saldo awal pengguna berada di angka Rp 10.000. Berdasarkan Gambar 4.27, Ketika transaksi gagal, mekanisme *rollback* menjalankan fungsi *increment* sebesar +Rp 8.000 sebanyak dua kali sesuai jumlah permintaan konkurensi yang masuk. Hal ini menyebabkan saldo pengguna yang seharusnya kembali ke Rp 10.000 justru bertambah secara tidak valid menjadi Rp 18.000, ditunjukkan dengan dua riwayat transaksi gagal bertanda +Rp 8.000. Increment sebesar +Rp 8000 dikarenakan harga dari produk Top Up E-Wallet Adalah sebesar Rp 8000

4.3.2. Menggunakan Sinkronisasi

Pengujian konkurensi dengan mengaktifkan mekanisme *Cloud Firestore Transaction* dilakukan pada dua perangkat fisik terpisah secara simultan untuk membuktikan pencegahan *race condition* dan *double spending*.

A. Pengujian Pengurangan Saldo (Skenario Transaksi Berhasil)

Pengujian konkurensi dengan mengaktifkan mekanisme *Cloud Firestore Transaction* dilakukan pada dua perangkat fisik terpisah secara simultan untuk membuktikan pencegahan *race condition* dan *double spending*. Dengan implementasi transaksi atomik, sistem menjamin pembacaan dan pembaruan saldo dilakukan dalam satu kesatuan operasi yang saling mengunci (*ACID compliance*).

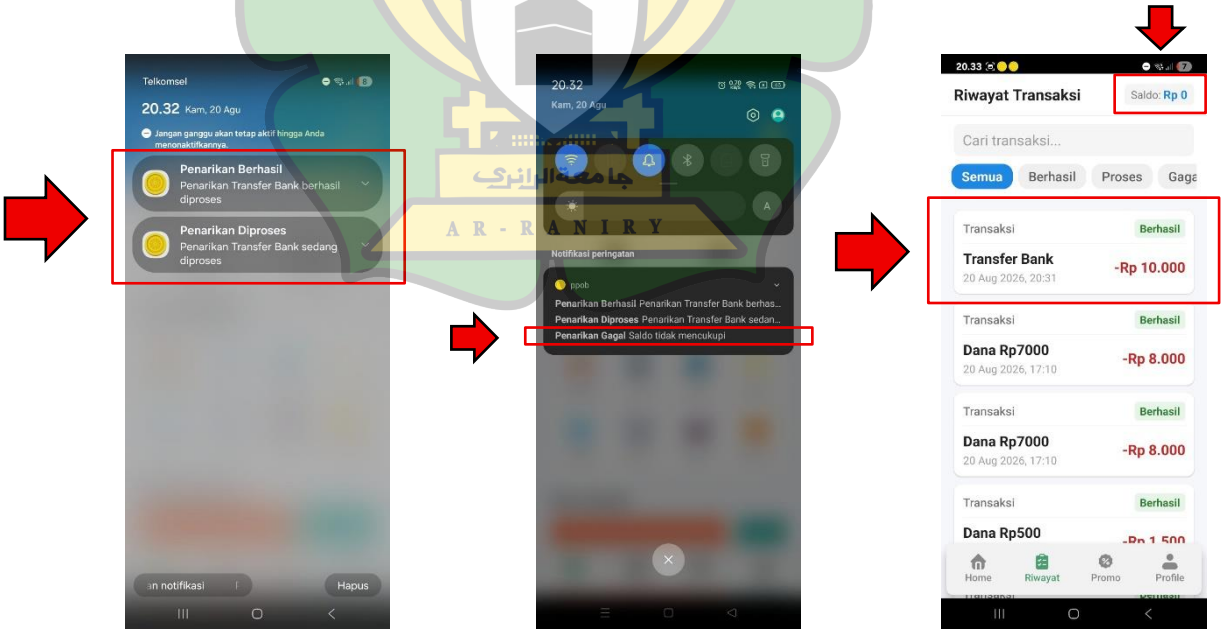
1) Skenario Transaksi Transfer Bank

- Berdasarkan Gambar 4.18, Pada pengujian Transfer Bank nominal Rp 10.000 dengan saldo awal Rp 10.000, dua perangkat mengirimkan permintaan secara bersamaan (req-3 dan req-4).

Gambar 4. 28 Log Terminal Backend Transfer Bank (Occ)

Log terminal backend menunjukkan bahwa transaksi pertama (req-3) berhasil diproses oleh *Cloud Firestore Transaction* dengan status statusCode: 200 (responseCode: '2024300'). Namun, permintaan kedua (req-4) yang diproses secara atomik setelahnya secara otomatis ditolak oleh sistem dengan *error log*: INSUFFICIENT_BALANCE dan mengembalikan *status code* 400 (statusCode: 400) sesuai dengan Gambar 4.28 di atas.

- Gambar 4.29 di bawah ini menunjukkan bahwa Perangkat pertama menerima notifikasi "Penarikan Berhasil: Penarikan Transfer Bank berhasil diproses", sedangkan perangkat kedua yang pengajuannya terblokir menerima notifikasi "Penarikan Gagal: Saldo tidak mencukupi". Pada riwayat transaksi, pemotongan saldo bernilai -Rp 10.000 hanya terjadi satu kali sehingga sisa saldo pengguna secara tepat menunjukkan angka **Rp 0**.



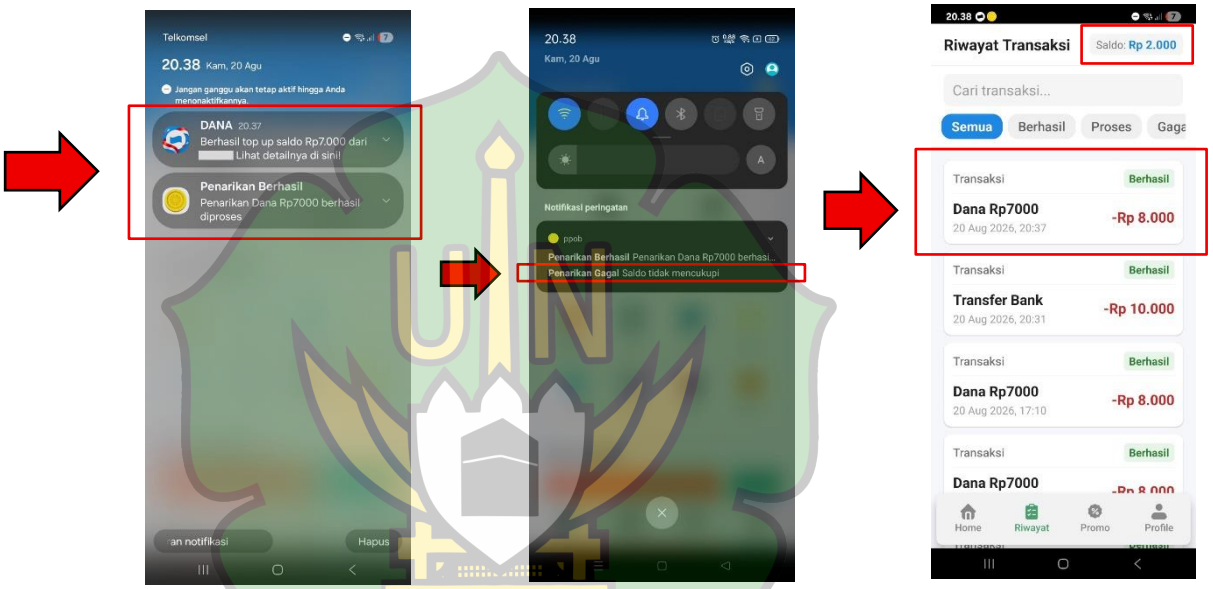
Gambar 4. 29 Notifikasi Dan History Transaksi Transfer Bank (Occ)

2) Skenario Transaksi Topup E-Wallet

- Pengujian dilanjutkan pada layanan Topup DANA Rp 7.000 dengan potongan saldo Rp 8.000 menggunakan saldo awal Rp 10.000. Log terminal backend pada Gambar 4.30.png mencatat dua *request* eksekusi paralel (req-a dan req-b). Transaksi req-b dieksekusi terlebih dahulu oleh *transaction lock* dan terdeteksi gagal dengan pesan INSUFFICIENT_BALANCE (statusCode: 400) karena perbedaan waktu milidetik atau urutan antrean transaksi. Sementara itu, transaksi req-a berhasil memenangi kunci transaksi atomik dan tereksekusi dengan responseCode: '2003800' (statusCode: 200).

Gambar 4. 30 Log Terminal Backend Top up Saldo E-Wallet (Occ)

- Berdasarkan Gambar 4.31 Perangkat yang berhasil memproses transaksi menerima notifikasi ganda DANA dan aplikasi "Penarikan Berhasil: Penarikan Dana Rp7000 berhasil diproses", sedangkan perangkat yang tertolak menerima notifikasi "Penarikan Gagal: Saldo tidak mencukupi". Halaman riwayat transaksi mencatat satu kali transaksi berhasil sebesar -Rp 8.000 dengan saldo akhir yang tersisa secara akurat bernilai **Rp 2.000**.



Gambar 4. 31 History Transaksi Gagal Topup Saldo E-Wallet (Occ)

B. Pengujian Rollback Saldo (Skenario Transaksi Gagal)

Pengujian *rollback* saldo menggunakan *Cloud Firestore Transaction* dilakukan pada dua perangkat fisik (*real device*) yang terhubung secara bersamaan. Skenario ini sengaja dirancang dengan memalsukan/menyalahkan *payload* yang dikirimkan ke server penyedia layanan (*provider*) untuk memicu kondisi *error* respons. Pengujian ini bertujuan membuktikan bahwa mekanisme transaksi atomik mampu mengembalikan saldo pengguna (*rollback*) secara tepat dan presisi tanpa kelebihan atau kekurangan nominal, sekaligus mencegah *race condition* dan *double spending* saat terjadi kegagalan

sistem secara simultan.

1) Skenario Transaksi Transfer Bank Gagal

- Eksekusi dan Respon Backend: Dua permintaan transfer bank bernilai Rp 10.000 dikirim bersamaan oleh dua perangkat fisik pada saldo awal Rp 10.000 (req-3 dan req-4) pada Gambar 4.32.

Gambar 4. 32 Log Terminal Backend Invalid Field Transfer Bank (Occ)

Log backend memperlihatkan respons penolakan dari *provider* akibat kesengajaan salah *payload* (responseCode: '4004302', responseMessage: 'Invalid Mandatory Field beneficiaryBankCode'). Transaksi pertama (req-3) yang gagal memicu fungsi pengembalian saldo (*rollback*), sehingga saldo dikembalikan utuh ke posisi Rp 10.000. Karena saldo dikembalikan tepat ke nilai awal, transaksi kedua (req-4) yang diproses secara atomik setelahnya membaca bahwa saldo hanya mencukupi untuk satu kali

eksekusi dan terhalang oleh *transaction lock*, sehingga dikategorikan *INSUFFICIENT_BALANCE* (statusCode: 400).

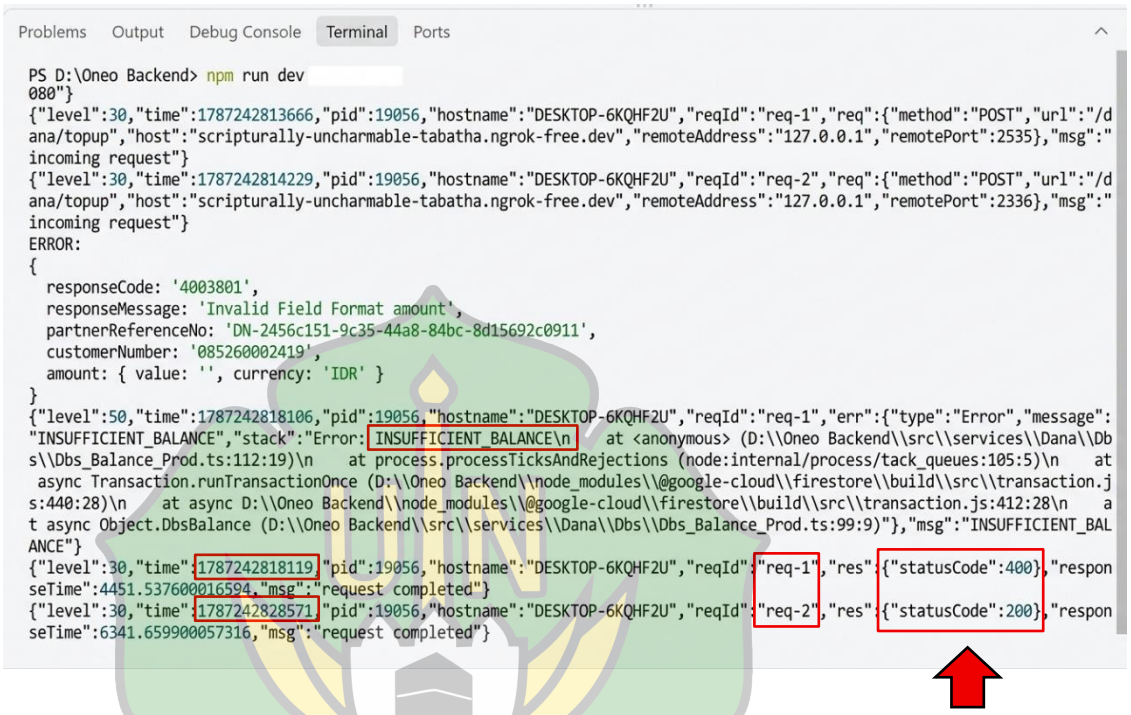
- Berdasarkan Gambar 4.32 Perangkat pertama menerima notifikasi "Penarikan Gagal: Penarikan Transfer Bank gagal, coba lagi nanti", sementara perangkat kedua menerima notifikasi saldo tidak mencukupi. Pada riwayat aplikasi, transaksi yang gagal secara otomatis mencatat pengembalian dana bertanda hijau (+Rp 10.000), dan nilai Saldo: Rp 10.000 tetap utuh sesuai saldo awal.

Gambar 4. 33 History Transaksi Gagal Transfer Bank (Occ)

2) Skenario Transaksi Topup E-Wallet Gagal

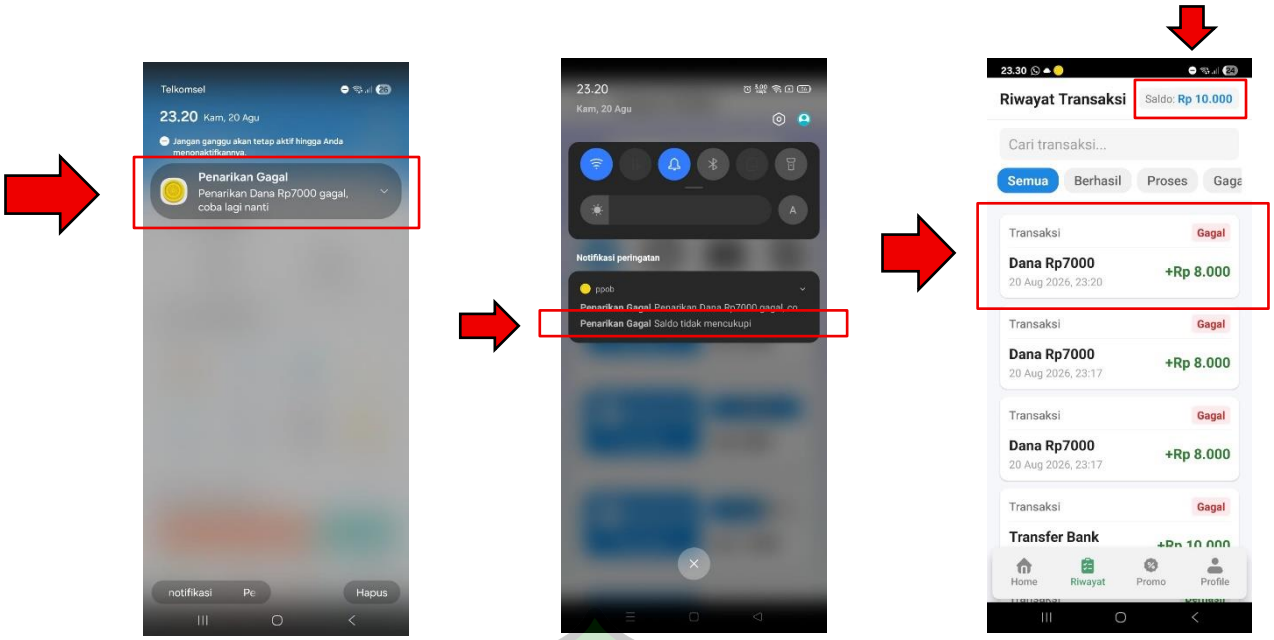
- Eksekusi dan Respon Backend: Pada pengujian Topup DANA Rp 7.000 (pemotongan Rp 8.000) dengan saldo awal Rp 10.000, dua perangkat mengirimkan *request* paralel (req-1 dan req-2) seperti pada Gambar 4.34. Server merespons *error* skenario kesalahan *payload* (responseCode: '4003801', responseMessage: 'Invalid Field Format amount'). Eksekusi

Cloud Firestore Transaction secara atomik menangani *rollback* saldo secara otomatis dan menolak pemrosesan ganda dengan log *INSUFFICIENT_BALANCE* (statusCode: 400).



Gambar 4. 34 Terminal Backend Invalid Field Top Up Saldo E-Wallet (Occ)

- Gambar 4.35 menunjukkan masing-masing perangkat menampilkan notifikasi peringatan "Penarikan Gagal: Penarikan Dana Rp7000 gagal, coba lagi nanti" serta notifikasi saldo tidak mencukupi. Pada riwayat transaksi, transaksi dicatat sebagai Gagal dengan pengembalian saldo bertanda hijau (+Rp 8.000). Pengembalian ini memastikan Saldo: Rp 10.000 tidak berkurang maupun bertambah dari posisi semula. Increment sebesar +Rp 8000 dikarenakan harga dari produk Top Up E-Wallet Adalah sebesar Rp 8000.



Gambar 4. 35 History Transaksi Gagal Top Up Saldo E-wallet (Occ)

4.4. Hasil Pengujian Idempotency

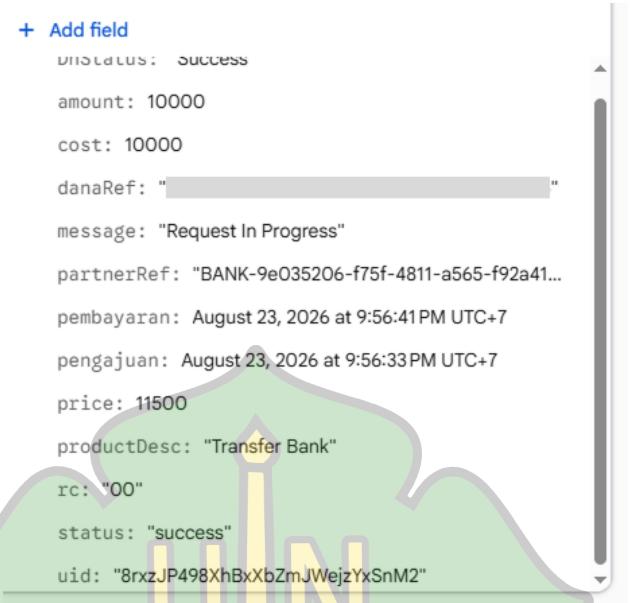
Pengujian diawali dengan mengirim request dengan ref id yang sudah berhasil di proses oleh transaksi sebelumnya. Eksekusi awal di pukul 21:56:33 WIB, sistem mengirimkan request transaksi pertama dengan ID referensi BANK-9e035206-f75f-4811-a565-f92a417001af seperti pada Gambar 4.36 dibawah.

```
const request: TransferToBankRequest = {
  partnerReferenceNo: "BANK-9e035206-f75f-4811-a565-f92a417001af",
  customerNumber: "...",
  beneficiaryAccountNumber: beneficiaryAccountNumber,
  beneficiaryBankCode: beneficiaryBankCode,
```

Gambar 4. 36 Request Transaksi

Server DANA menerima request tersebut sebagai transaksi baru yang valid, kemudian meresponnya dengan *response code* 00 (*Success / Request In Progress*) serta menerbitkan nomor referensi resmi DANA. Data ini berhasil disimpan di database Firestore dengan status "success".

Pengujian *idempotency* terjadi pada request kedua di pukul 22:25:56 WIB, di mana sengaja memicu ulang request menggunakan nilai partnerReferenceNo yang sama persis.

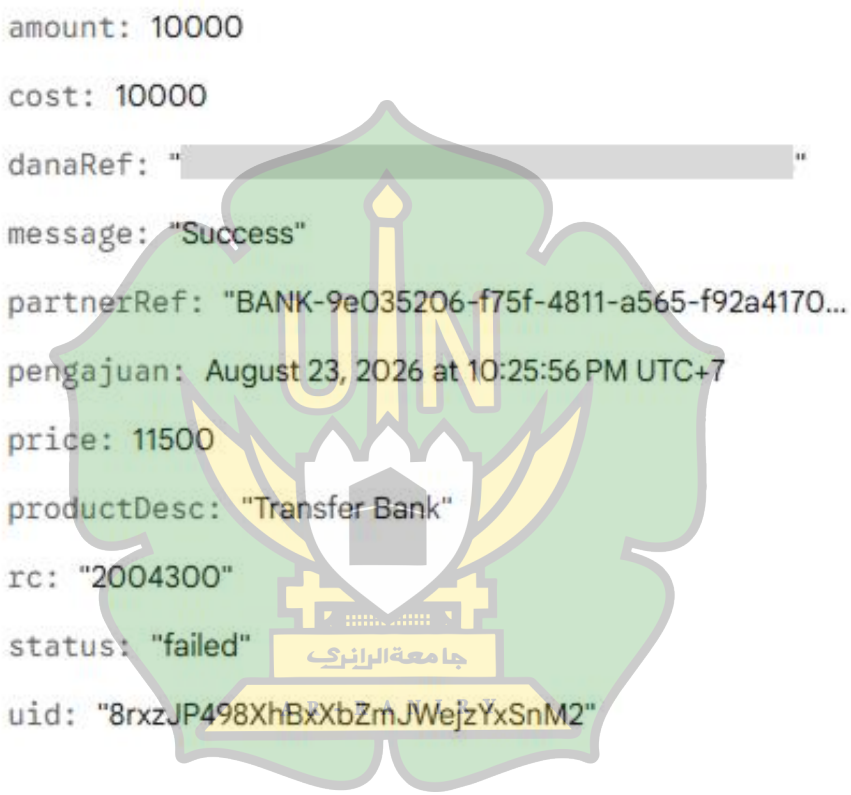


Gambar 4. 37 Sukses Transfer Bank

Berdasarkan Gambar 4.37, API DANA secara akurat mengenali bahwa transaksi dengan ID referensi tersebut sudah pernah diproses sebelumnya. Alih-alih memotong saldo atau mencairkan dana untuk kedua kalinya, DANA menolak request tersebut dengan mengembalikan *response code* 2004300 *(Berhasil)*, namun transaksi tidak di proses melainkan mengembalikan response history transaksi yang sudah di proses sebelumnya, kemudian menandai status eksekusi kedua sebagai "failed" dan memicu logika *rollback* saldo pada backend seperti pada Gambar 4.38 dibawah

Gambar 4. 38 Log Tranfer Bank

Hasil pengujian ini membuktikan bahwa mekanisme perlindungan Idempotency sudah berjalan dengan sangat baik dan berhasil mencegah kerugian finansial akibat transaksi duplikat. Namun, skenario ini juga memperlihatkan pentingnya mengimplementasikan hal ini. Dengan menerapka konsep ini request duplikat serupa di masa mendatang akan langsung dihentikan seperti pada Gambar 4.39 memperlihatkan status failed dikarenakan ref id sudah pernah di proses.



Gambar 4. 39 Gagal Transfer Bank

4.5. Temuan Penelitian

4.5.1. Tabel Hasil Pengujian Konkurensi

Tabel 4. 1 Tabel Hasil Pengujian Konkurensi

Kondisi	Layanan	No.	Skenario	Saldo Awal	Pengurangan Saldo	Penambahan Saldo	Saldo Akhir	Hasil Pengujian
Tanpa Sinkronisasi	Transfer Bank	1	Pengurangan saldo (transaksi berhasil)	Rp10.000	Rp20.000	Rp0	Rp0	**Gagal**
	Topup Dana	2	Pengurangan saldo (transaksi berhasil)	Rp10.000	Rp16.000	Rp0	Rp2.000	**Gagal**
	Transfer Bank	3	Rollback saldo (transaksi gagal)	Rp10.000	Rp20.000	Rp20.000	Rp20.000	**Gagal**
	Topup Dana	4	Rollback saldo (transaksi gagal)	Rp10.000	Rp16.000	Rp16.000	Rp18.000	**Gagal**
Dengan Sinkronisasi	Transfer Bank	5	Pengurangan saldo (transaksi berhasil)	Rp10.000	Rp10.000	Rp0	Rp0	**Berhasil**
	Topup Dana	6	Pengurangan saldo (transaksi berhasil)	Rp10.000	Rp8.000	Rp0	Rp2.000	**Berhasil**
	Transfer Bank	7	Rollback saldo (transaksi gagal)	Rp10.000	Rp10.000	Rp10.000	Rp10.000	**Berhasil**
	Topup Dana	8	Rollback saldo (transaksi gagal)	Rp10.000	Rp8.000	Rp8.000	Rp10.000	**Berhasil**

Tabel 4.1 menunjukkan perbandingan performa integritas data antara sistem Tanpa Sinkronisasi dan sistem Dengan Sinkronisasi (Cloud Firestore Transaction) saat menerima akses secara paralel oleh dua *real device*. Setiap skenario menguji layanan *Transfer Bank* dan *Topup Dana* dengan mengamati perubahan nilai pada kolom Saldo Awal, Pengurangan Saldo, Penambahan Saldo, serta Saldo Akhir. Status pada kolom Hasil Pengujian menentukan apakah sistem berhasil mencegah kesalahan pembaruan data (*race condition/lost update*) dan mampu menjaga pengembalian dana (*rollback*) secara tepat. Pengujian integritas saldo ini dilakukan secara paralel menggunakan 2 *real device* untuk mensimulasikan kondisi konkurensi data (*race condition*). Evaluasi keberhasilan sistem diukur menggunakan rumus matematika integritas saldo berikut:

$$S_{akhir} = S_{awal} - \sum_{i=1}^{T_{valid}} D_i + \sum_{j=1}^{T_{rollback}} P_j \qquad S_{hasil}$$

Di mana S_{awal} adalah saldo awal, D_i adalah nominal pengurangan saldo (debit) untuk transaksi valid (T_{valid}), dan P_j adalah nominal penambahan/pengembalian saldo (*rollback*) apabila transaksi dibatalkan ($T_{rollback}$). Sistem dinyatakan Berhasil jika nilai saldo akhir yang tercatat pada tabel hasil pengujian (S_{akhir}) persis sama dengan hasil kalkulasi rumus ekspektasi (S_{hasil}). Sebaliknya, sistem dinyatakan Gagal apabila terjadi penyimpangan data, baik karena pemotongan saldo melebihi saldo awal yang ada maupun kegagalan pengembalian dana (*rollback*) secara penuh (*Atomic Transaction failure*).

A. Kelompok Kondisi Tanpa Sinkronisasi

1) **Skenario 1 (Transfer Bank - Pengurangan Saldo / Transaksi Berhasil)**

   Dua *device* mengirim secara bersamaan untuk pemotongan saldo

nominal $D = \text{Rp}20.000$ dari saldo awal $S_{awal} = \text{Rp}10.000$. Karena saldo awal tidak mencukupi untuk memproses transaksi 20.000, maka tidak ada transaksi yang sah ($T_{valid} = 0$).

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp0
- Kesimpulan: GAGAL. Terjadi anomali saldo di mana saldo akhir menjadi Rp0 padahal transaksi gagal memotong saldo yang sah ($S_{akhir} \neq S_{hasil}$).

2) Skenario 2 (Topup Dana - Pengurangan Saldo / Transaksi Berhasil)

Dua *device* mengirim secara paralel dengan nominal $D = \text{Rp}16.000$ dari saldo awal $S_{awal} = \text{Rp}10.000$. Transaksi ini seharusnya ditolak ($T_{valid} = 0$) karena melampaui saldo awal.

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp2.000
- Kesimpulan: GAGAL. Terjadi *Race Condition / Dirty Read* yang menyebabkan saldo akhir tercatat secara salah menjadi Rp2.000 ($S_{akhir} \neq S_{hasil}$).

3) Skenario 3 (Transfer Bank - Rollback Saldo / Transaksi Gagal)

Dua *device* memicu skenario transaksi gagal sebesar Rp20.000 yang harus memicu *rollback* penambahan saldo $P = \text{Rp}20.000$.

- Kalkulasi Rumus Ekspektasi:

  Karena transaksi dibatalkan di tengah alur, saldo harus kembali utuh seperti semula ($T_{valid} = 0, P_1 = \text{Rp}0$ karena dana belum sempat terpotong secara sah).

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp20.000
- Kesimpulan: GAGAL. Mekanisme *rollback* tanpa sinkronisasi mengalami kesalahan pembacaan data sehingga saldo bertambah ganda menjadi Rp20.000 ($S_{akhir} \neq S_{hasil}$).

4) **Skenario 4 (Topup Dana - Rollback Saldo / Transaksi Gagal)**

Dua *device* memicu transaksi gagal sebesar Rp16.000 secara simultan dari saldo awal Rp10.000.

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp18.000
- Kesimpulan: GAGAL. Sistem mengalami *lost update* saat proses *rollback* sehingga saldo akhir membengkak menjadi Rp18.000 ($S_{akhir} \neq S_{hasil}$).

B. Kelompok Kondisi Dengan Sinkronisasi (Cloud Firestore Transaction)

1) **Skenario 5 (Transfer Bank - Pengurangan Saldo / Transaksi Berhasil)**

Dua *device* mengirim transaksi pemotongan saldo $D = \text{Rp}10.000$ secara paralel dari saldo awal $S_{awal} = \text{Rp}10.000$. Berkat sinkronisasi transaksi, hanya ada tepat 1 transaksi yang sah ($T_{valid} = 1$), sedangkan 1 transaksi lainnya ditolak karena saldo habis.

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - \sum_{i=1}^{1} (\text{Rp}10.000) + 0 = \text{Rp}0$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp0
- Kesimpulan: BERHASIL. Nilai saldo akhir pada tabel sesuai sempurna dengan kalkulasi matematis ($S_{akhir} = S_{hasil}$).

2) **Skenario 6 (Topup Dana - Pengurangan Saldo / Transaksi Berhasil)**

Dua *device* mengirim secara serentak untuk pemotongan $D = \text{Rp}8.000$ dari saldo awal $S_{awal} = \text{Rp}10.000$. Berkat mekanisme penguncian (*locking/OCC*), hanya 1 transaksi yang berhasil dikomit ($T_{valid} = 1$).

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - \sum_{i=1}^{1} (\text{Rp}8.000) + 0 = \text{Rp}2.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp2.000
- Kesimpulan: BERHASIL. Pemotongan saldo terisolasi dengan aman sehingga menyisakan saldo Rp2.000 ($S_{akhir} = S_{hasil}$).

3) **Skenario 7 (Transfer Bank - Rollback Saldo / Transaksi Gagal)**

Dua *device* memicu transaksi bernominal Rp10.000 yang mengalami kegagalan sistem di pertengahan proses.

- Kalkulasi Rumus Ekspektasi:

  Berkat sifat *Atomicity* Cloud Firestore, transaksi yang gagal sepenuhnya dibatalkan (*rollbacked*) tanpa mengubah keadaan saldo ($T_{valid} = 0$).

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp10.000
- Kesimpulan: BERHASIL. Saldo utuh Rp10.000 dan tidak terpengaruh oleh kegagalan transaksi ($S_{akhir} = S_{hasil}$).

4) **Skenario 8 (Topup Dana - Rollback Saldo / Transaksi Gagal)**

Dua *device* memicu transaksi gagal bernominal Rp8.000 secara paralel dari saldo awal Rp10.000.

- Kalkulasi Rumus Ekspektasi:

$$S_{hasil} = \text{Rp}10.000 - 0 + 0 = \text{Rp}10.000$$

- Saldo Akhir pada Tabel (S_{akhir}): Rp10.000
- Kesimpulan: BERHASIL. Mekanisme *rollback* dengan sinkronisasi berhasil mempertahankan konsistensi saldo akhir ($S_{akhir} = S_{hasil}$).

4.5.2. Tabel Hasil Pengujian Idempotency

Tabel 4. 2 Tabel Hasil Pengujian Idempotency

No.	Skenario Pengujian	Request	Saldo Awal	Hasil Pemrosesan	Saldo Akhir	Status
1	Transaksi pertama	Request awal	Rp20.000	Transaksi berhasil diproses	Rp10.000	Berhasil
2	Request ulang transaksi yang sama	Retry	Rp10.000	Transaksi ditolak	Rp10.000	Gagal

Berdasarkan tabel 4.2 hasil pengujian, pengujian idempotency dilakukan dalam dua tahap, yaitu transaksi pertama dan request ulang terhadap transaksi yang sama. Pada transaksi pertama, pengguna memiliki saldo awal sebesar Rp20.000 dan melakukan transaksi sebesar Rp10.000. Transaksi berhasil diproses sehingga saldo pengguna berkurang menjadi Rp10.000. Selanjutnya, dilakukan request ulang terhadap transaksi yang sama menggunakan identitas idempotency yang sama. Sistem mengenali bahwa request tersebut telah diproses sebelumnya sehingga request tidak diproses kembali. Transaksi kedua ditolak dan saldo pengguna tetap Rp10.000.

Hasil tersebut menunjukkan bahwa mekanisme idempotency berhasil mencegah terjadinya pemrosesan transaksi secara berulang. Meskipun request dikirimkan kembali, saldo pengguna tidak mengalami pengurangan kedua kalinya dan transaksi tidak diproses ulang. Dengan demikian, integritas saldo tetap terjaga dan tidak terjadi transaksi ganda akibat request yang sama dikirimkan kembali.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilakukan pada penelitian mengenai *Implementasi Mekanisme Pencegahan Race Condition pada Concurrent Programming Menggunakan Optimistic Concurrency Control Berbasis Multi Version Concurrency Control untuk Menjaga Integritas Data Aplikasi Payment Point Online Bank*, maka dapat ditarik beberapa kesimpulan sebagai berikut:

1. Mekanisme Cloud Firestore Transaction yang menerapkan Optimistic Concurrency Control (OCC) berbasis Multi-Version Concurrency Control (MVCC) berhasil diimplementasikan pada aplikasi Payment Point Online Bank (PPOB) berbasis mobile. Mekanisme tersebut diterapkan pada proses transaksi yang berkaitan dengan pemotongan dan pengembalian saldo pengguna. Implementasi transaksi memungkinkan proses pembacaan dan pembaruan saldo dilakukan secara terkontrol sehingga transaksi yang mengalami konflik dapat ditolak atau diproses kembali sesuai mekanisme transaksi yang diterapkan. Mekanisme OCC pada penelitian ini memanfaatkan perubahan versi data untuk mendeteksi adanya konflik sebelum transaksi melakukan *commit*.
2. Penerapan Cloud Firestore Transaction terbukti mampu mencegah terjadinya race condition dan lost update pada transaksi yang dijalankan secara konkuren. Berdasarkan pengujian menggunakan dua *real device* yang mengirimkan permintaan secara paralel terhadap dokumen saldo pengguna yang sama, kondisi tanpa sinkronisasi menyebabkan terjadinya konflik pembaruan data. Pada pengujian tersebut ditemukan transaksi ganda, pemotongan saldo yang melebihi saldo awal, serta kesalahan pada proses *rollback*.
3. Pada kondisi tanpa sinkronisasi, seluruh skenario pengujian tidak memenuhi kriteria integritas saldo. Pada skenario pengurangan saldo,

dua transaksi dapat diproses secara bersamaan meskipun saldo hanya mencukupi untuk satu transaksi. Selain itu, pada skenario transaksi gagal, proses *rollback* dapat dilakukan lebih dari satu kali sehingga saldo akhir menjadi lebih besar daripada saldo yang seharusnya. Hal tersebut menunjukkan adanya *race condition*, *dirty read*, dan *lost update* pada proses transaksi konkuren.

4. Pada kondisi dengan sinkronisasi menggunakan Cloud Firestore Transaction, seluruh skenario pengujian berhasil memenuhi kriteria integritas saldo. Pada transaksi transfer bank dengan saldo awal Rp10.000 dan dua permintaan pemotongan sebesar Rp10.000 secara bersamaan, hanya satu transaksi yang berhasil diproses, sedangkan transaksi lainnya ditolak karena saldo telah habis. Saldo akhir tercatat sebesar Rp0 sesuai dengan hasil perhitungan yang diharapkan.
5. Mekanisme sinkronisasi juga berhasil menjaga konsistensi saldo pada transaksi yang membutuhkan proses rollback. Pada transaksi yang mengalami kegagalan, Cloud Firestore Transaction mampu menjaga agar pengembalian saldo tidak dilakukan secara ganda. Hasil pengujian menunjukkan saldo akhir tetap sesuai dengan saldo yang seharusnya, yaitu Rp10.000, sehingga tidak terjadi penambahan saldo yang tidak sah.
6. Berdasarkan delapan skenario pengujian, penerapan Cloud Firestore Transaction menunjukkan hasil yang lebih baik dibandingkan sistem tanpa sinkronisasi. Empat skenario pada kondisi tanpa sinkronisasi dinyatakan gagal, sedangkan empat skenario pada kondisi dengan sinkronisasi dinyatakan berhasil. Penilaian keberhasilan dilakukan dengan membandingkan saldo akhir aktual dengan saldo akhir yang diperoleh berdasarkan rumus ekspektasi integritas saldo. Sistem dinyatakan berhasil apabila nilai saldo akhir aktual sama dengan nilai saldo akhir yang diharapkan. Idempotency juga berhasil dicegah sesuai dengan pengujiannya
7. Dengan demikian, tujuan penelitian untuk mengimplementasikan mekanisme transaksi Cloud Firestore berbasis OCC dalam mencegah race condition serta menjaga konsistensi dan integritas data pada

transaksi yang berjalan secara bersamaan telah tercapai. Hasil pengujian menunjukkan bahwa penggunaan mekanisme transaksi memberikan perlindungan terhadap konflik akses pada data saldo sehingga transaksi dapat diproses secara lebih konsisten dan mencegah terjadinya pemrosesan transaksi yang melebihi kondisi saldo sebenarnya. Hal ini sesuai dengan tujuan penelitian yang berfokus pada pencegahan race condition dan penjagaan integritas data pada proses transaksi konkuren.

5.2. Saran

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa saran yang dapat digunakan sebagai bahan pengembangan sistem maupun penelitian selanjutnya, yaitu:

1. Pengujian konkurensi dapat dikembangkan dengan jumlah perangkat dan request yang lebih banyak. Penelitian ini menggunakan dua *real device* untuk mensimulasikan kondisi transaksi secara konkuren. Penelitian selanjutnya dapat menggunakan jumlah pengguna atau request yang lebih besar untuk mengetahui kemampuan mekanisme OCC dalam menghadapi tingkat *data contention* yang lebih tinggi.
2. Penelitian selanjutnya dapat melakukan perbandingan dengan metode concurrency control lainnya. Mekanisme OCC yang digunakan dalam penelitian ini dapat dibandingkan dengan pendekatan lain seperti *Pessimistic Concurrency Control*, *locking*, atau metode sinkronisasi lainnya untuk mengetahui perbedaan dari sisi konsistensi data, tingkat konflik transaksi, jumlah *retry*, waktu pemrosesan, serta penggunaan sumber daya.
3. Pengujian dapat diperluas pada berbagai jenis transaksi dan kondisi kegagalan. Penelitian ini berfokus pada mekanisme sinkronisasi terutama pada proses debit saldo, sedangkan mekanisme pengisian saldo (*top up/deposit*) tidak menjadi fokus utama penelitian. Oleh karena itu, penelitian selanjutnya dapat menguji lebih banyak jenis transaksi serta berbagai kondisi kegagalan jaringan, kegagalan layanan pihak ketiga, dan transaksi yang mengalami *timeout*. Batasan penelitian terkait fokus

pada debit saldo dan layanan tertentu juga telah ditetapkan pada penelitian ini.

4. Pengembangan selanjutnya dapat menambahkan pengukuran performa secara kuantitatif. Selain mengukur keberhasilan integritas saldo, penelitian berikutnya dapat mengukur waktu respons transaksi, jumlah transaksi yang mengalami konflik, jumlah *retry*, tingkat keberhasilan transaksi, serta penggunaan sumber daya. Dengan demikian, efektivitas OCC tidak hanya dinilai dari aspek konsistensi data, tetapi juga dari aspek performa sistem.

DAFTAR PUSTAKA

Adjie Eriyadi, R., Aulia Dini, R., Novianti, W., Putra Kirana, S., & Kamil, I. (2025). *Kohesi: Jurnal Multidisiplin Saintek PENTINGNYA SERIALIZABILITY DALAM MANAJEMEN TRANSAKSI: TINJAUAN MEKANISME KONTROL KONKURENSI UNTUK MENJAMIN KONSISTENSI DATA*. *10*(7). https://doi.org/10.8734/Kohesi.v1i2.365

Ammar Dwi Anwari, Rizky Januar Akbar, & Royyana Muslim Ijtihadie. (2021). Implementasi Optimistic Concurrency Control pada Sistem Aplikasi E-Commerce Berdasarkan Arsitektur MicroServices Menggunakan Kubernetes. *Jurnal Teknik ITS*, *10*, A182–A188.

Bank Indonesia. (2024). *Sistem Pembayaran*. Bank Indonesia. https://www.bi.go.id/id/fungsi-utama/sistem-pembayaran/default.aspx

Diogo, M., Cabral, B., & Bernardino, J. (2019). Consistency models of NoSQL databases. In *Future Internet* (Vol. 11, Number 2). MDPI AG. https://doi.org/10.3390/fi11020043

Dubey, M. K., Lowe, D., & Galhotra, B. (2019). A Study on Race Condition and Dynamic Data Race Detection Techniques. *International Journal of Computer Sciences and Engineering*, *7*(6), 41–46. https://doi.org/10.26438/ijcse/v7i6.4146

Fadhila, A., Nugraha, A., Mazaya, C., Ramadhan, M., & Kusnendar, J. (2025). Optimalisasi Sinkronisasi Proses: Analisis Efektivitas Semaphore, Mutex, dan Monitor. *Jurnal Komputer, Informasi Dan Teknologi*, *5*(1), 15. https://doi.org/10.53697/jkomitek.v5i1.2595

Firmawati, E., & Avorizano, A. (2026). *ANALISIS MEKANISME KONTROL KONKURENSI PADA SISTEM RESERVASI FASILITAS OLAHRAGA BERBASIS WEB: STUDI KASUS KLUB KELAPA GADING JAKARTA*.

Hana Lolita BLT, T., Kabetta, H., Kriptografi, R., & Siber dan Sandi Negara, P. (2021). *Studi Web Race Condition Sebagai Acuan Pembuatan Modul Praktikum*.

Harahap, A. H., Difa Andani, C., Christie, A., Nurhaliza, D., & Fauzi, A. (2023). Pentingnya Peranan CIA Triad Dalam Keamanan Informasi dan Data Untuk

Pemangku Kepentingan atau Stakholder. *Jurnal Manajemen Pendidikan Dan Ilmu Sosial (JMPD)*, *1*, 73–83.

Hartono, Dahlan Abdullah, Fadlisyah, & Cut Ita Erliana. (2018). *Sistem Operasi*. SEFA BUMI PERSADA.

Hermawan, M. L., & Suharso, W. (2025). *IMPLEMENTATION OF CONCURRENCY CONTROL TO PREVENT RACE CONDITION IN A WEB-BASED BILLIARD TABLE RESERVATION SYSTEM IMPLEMENTASI CONCURRENCY CONTROL UNTUK MENCEGAH RACE CONDITION PADA SISTEM PEMESANAN MEJA BILLIARD BERBASIS WEBSITE*. *10*(3), 1971–1981.

Kanungo, S., & Morena, R. D. (2024). Concurrency versus consistency in NoSQL databases. *Journal of Autonomous Intelligence*, *7*(3), 1–16. https://doi.org/10.32629/jai.v7i3.936

P. A. Kirkpatrick, & M. P. Amos. (2023). Critical Sections and Peterson's Solution. In *OpenCSF* (7.2). OpenCSF Project. https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/CritSect.html

Subagio, A. W., & Muttaqin, F. (2022). TEKNO Jurnal Teknologi Elektro dan Kejuruan Arif Widiasan Subagio, Faisal Muttaqin | Penerapan Clean Architecture pada Pengembangan Sistem Payment Point Online… Penerapan Clean Architecture pada Pengembangan Sistem Payment Point Online Bank. In *Jurusan Teknik Elektro* (Vol. 32). http://journal2.um.ac.id/index.php/tekno

Wirayuda, H., & Usman, A. (2025). Web-Based Batik Sales System Uses Locking Mechanism to Prevent Race Condition. *Journal of Technology and Computer (JOTECHCOM)*, *2*(3), 165–170.