Jurnal Elektronik Ilmu Komputer Udayana
Volume 12, No 4. May 2025

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Data Processing Using Change Data Capture for Real-Time Data Synchronization

I Made Ari Madya Santosa[a1], I Gusti Ngurah Anom Cahyadi Putra[a2], I Ketut Gede Suhartana[a3], I Gede Surya Rahayuda[a4]

[a]Program Studi Informatika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Udayana
Kuta Selatan, Badung, Bali, Indonesia
[1]arimadyasantosa@student.unud.ac.id
[2]anom.cp@unud.ac.id
[3]ikg.suhartana@unud.ac.id
[4]igedesuryarahayuda@unud.ac.id

Abstract

In an era increasingly reliant on data, seamless integration and synchronization between various systems are crucial for maintaining consistency, accuracy, and sustainability of information. Change Data Capture (CDC) is a highly effective method for detecting data changes in real-time, allowing systems to record each modification without reprocessing the entire dataset. By integrating CDC with an Event-Driven Architecture, systems can process only data changes, reducing server load and ensuring data consistency across interconnected systems. This study examines a data synchronization system built using CDC and Event-Driven Architecture on a dataset consisting of 1,000,000 records. The test results indicate strong performance, with the fastest average processing time recorded at 2.6622 milliseconds per data entry and a data loss rate of 0, ensuring high reliability. These findings demonstrate that the combination of CDC and Event-Driven Architecture offers an efficient, fast, and scalable solution for real-time data synchronization on a large scale, meeting modern system requirements.

Keywords: *Change Data Capture (CDC), Event-Driven Architecture, Data Synchronization, Real-Time Processing, Data Consistency.*

1. Pendahuluan

Di era yang berfokus pada data seperti sekarang, banyak organisasi bergantung pada *Sales force* sebagai pusat utama untuk mengoptimalkan proses bisnis, mengelola hubungan pelanggan, dan mendorong penjualan. Seiring bisnis berkembang dan semakin kompleksnya operasi mereka, kebutuhan untuk integrasi dan sinkronisasi data yang mulus di berbagai sistem menjadi sangat penting. Kemampuan untuk menjaga informasi yang konsisten, akurat, dan selalu terbaru di berbagai platform sangatlah penting untuk pengambilan keputusan, kepuasan pelanggan, dan kesuksesan bisnis secara keseluruhan [1]. Jika laporan yang bertentangan muncul, organisasi akan kesulitan membuat keputusan. Oleh karena itu, diperlukan suatu model pengolahan data terintegrasi yang dapat menangani ketidakkonsistenan data tersebut secara *real-time* [2].

Avatar Solution, perusahaan teknologi informasi di Denpasar, menghadapi masalah besar terkait sinkronisasi data antara data backup dan transaksi. Kesalahan sinkronisasi data dapat menyebabkan ketidaksesuaian antara kedua sistem serta masalah besar dengan keandalan dan akurasi data, yang sangat penting untuk operasi bisnis sehari-hari. Akibatnya, proses pemrosesan data dan penarikan laporan perusahaan menurun, yang menyebabkan pengambilan keputusan yang tidak optimal. Sehingga, dibutuhkan sebuah sistem yang mampu memproses data secara real-time untuk memastikan bahwa setiap perubahan data di sistem transaksional dapat disinkronisasi dengan tepat ke sistem backup. Solusi ini juga harus mampu secara otomatis menjaga konsistensi data di seluruh sistem.

Change Data Capture (CDC) adalah fitur penting dalam sistem manajemen basis data relasional yang memungkinkan identifikasi, pelacakan, dan ekstraksi modifikasi yang dilakukan pada data dalam basis data [3]. CDC melacak perubahan data secara *real-time*, dan jika menggunakan *binlog*, CDC dapat secara mandiri melacak perubahan yang dilakukan berdasarkan aplikasi sumber. Penggunaan CDC dengan *binlog MySQL* lebih disarankan daripada penggunaan *trigger MySQL*, karena penggunaan MySQL pemicu dapat memperlambat aplikasi [4].

Event-driven architecture memproses data saat data tiba secara real-time, daripada memprosesnya dalam batch. Pendekatan ini sangat berguna untuk aplikasi yang memerlukan latensi rendah dan *throughput* tinggi, karena memungkinkan pemrosesan data secara langsung, sehingga memberikan hasil dan respon yang cepat tanpa menunggu akumulasi data dalam jumlah besar [5].

Fokus penelitian ini adalah pengembangan sistem sinkronisasi data secara *real-time* yang memastikan konsistensi data dengan menggunakan metode *change data capture* dan *event-driven architecture*. Diharapkan, penelitian ini dapat memberikan solusi untuk dapat melakukan sinkronisasi data secara *real-time* sehingga konsistensi data antar sistem dapat terjaga dengan baik.

2. Metodologi Penelitian

2.1. Analisis Permasalahan

Peneliti melakukan observasi terhadap berbagai perusahaan teknologi informasi. Dari hasil pengamatan, muncul kebutuhan akan sebuah sistem yang mampu melakukan sinkronisasi data secara *real-time*, baik untuk data transaksional maupun *data backup*. Sistem ini diharapkan dapat menjaga konsistensi data dan mencegah kehilangan data. Selain itu, sistem juga harus dapat menerima dan memproses sejumlah besar data dalam waktu singkat.
Terdapat pula kebutuhan fungsionalitas lain untuk mempermudah dalam proses sinkronisasi seperti dapat dikontrol lewat sebuah tampilan antar muka dan kontrol terhadap jenis data yang akan di sinkronisasi berdasarkan jenis *data manipulation language* (DML).

2.2. Pengumpulan Data

Tahapan selanjutnya, penulis melakukan pengumpulan data. Dataset yang penulis gunakan didapatkan dari Kaggle. Penulis menggunakan data dari Kaggle, karena data yang dimiliki oleh perusahaan tidak diizinkan untuk disebarluaskan, karena data tersebut bersifat rahasia. Salah satu tabel pada dataset ini memiliki 1 juta baris data. Nantinya akan digenerasikan ulang menyesuaikan kebutuhan untuk merepresentasikan salah elemen dari Big Data, yaitu big volume.

2.3. Perancangan Sistem

Perancangan sistem yang digunakan dalam penelitian ini dibagi menjadi beberapa bagian yaitu perancangan alur umum sistem, perancangan *high-level change data capture* dan *event-driven architecture*, alur *change data capture* dan *event-driven architecture*, dan alur *synchronization handler service*.

2.3.1. Perancangan Alur Umum Sistem

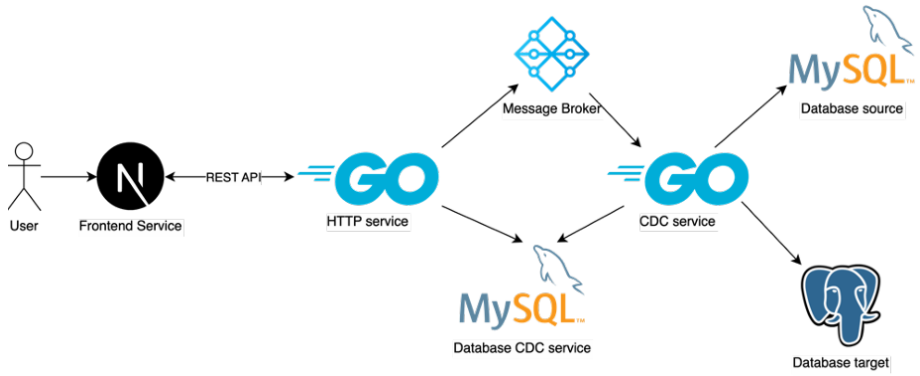
Sistem akan menerapkan metode *change data capture* dan *event-driven architecture*. Secara garis besar sistem akan melalui tahapan-tahapan pemrosesan 16 hingga data dapat tersinkronisasi. Pertama-tama user akan masuk ke dalam sistem, kemudian user mengirimkan data konfigurasi *database source*, *database* target dan *data manipulation language* (DML) ke HTTP Service. Kemudian HTTP Service akan menyimpan konfigurasi ke *database* CDC Service dan mengirimkan *message* ke NSQ sebagai *message broker* lalu NSQ akan langsung meneruskan *message* ke *consumer*. CDC Service akan menerima *message* dari NSQ dan menjalankan proses sinkronisasi. Data yang diperoleh CDC Service akan di cek apakah data akan disinkronisasi atau tidak sesuai dengan konfigurasi yang dikirimkan oleh *user*. Jika data akan disinkronisasi, maka data akan dikirimkan ke *database target*. Untuk melihat gambaran alur umum sistem dapat dilihat pada Gambar 1.

p-ISSN: 2301-5373
e-ISSN: 2654-5101

Gambar 1. Perancangan Alur Umum Sistem

2.3.2. Perancangan High-level dari Change Data Capture dan Event-Driven Architecture

Penelitian akan dibuat dengan menerapkan metode *change data capture* dan *event-driven architecture* agar dapat melakukan sinkronisasi data secara *real-time*. Perancangan *High-level change data capture* dan *event-driven architecture* dapat dilihat pada Gambar 2.

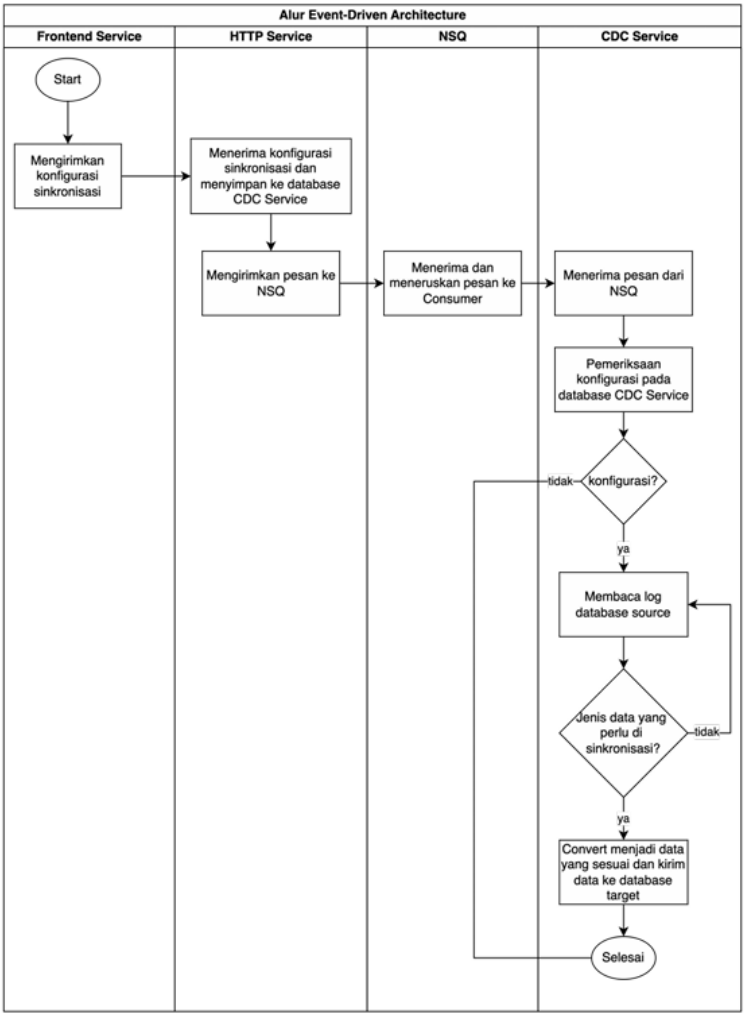


Gambar 2. Perancangan High-Level Change Data Capture dan Event-Driven Architecture

Terdapat *frontend service* menggunakan teknologi Next.JS yang dapat diakses oleh user untuk menginputkan konfigurasi dari *database source, database target*, dan juga konfirgurasi dari tipe *data manipulation language*. Kemudian terdapat juga HTTP *service* yang dibangun menggunakan Bahasa pemrograman Go berperan sebagai kontroler untuk menyimpan data konfigurasi sinkronisasi ke *database* CDC service. Terdapat juga CDC Service yang berperan sebagai tempat untuk dilakukannya proses sinkronisasi, dimana proses sinkronisasi ini akan membaca *log* dari *database source* yang kemudian diolah dan disinkronisasi ke *database target*. *Message broker* NSQ digunakan sebagai sebuah alat untuk melakukan *trigger* pada proses sinkronisasi.

2.3.3. Alur Event-Driven Architecture

Event-Driven Architecture pada penelitian ini mencakup beberapa elemen, yaitu Frontend Service, HTTP Service, NSQ sebagai *message broker*, database CDC Service, CDC Service dan *database target* yang menjadi destinasi data yang disinkronisasi. Untuk melihat alur dari *Event-Driven Architecture* dapat dilihat pada Gambar 3 dan dapat juga dilihat pada lampiran untuk melihat gambar dengan lebih jelas.



Gambar 3. Alur Event Driven Architecture

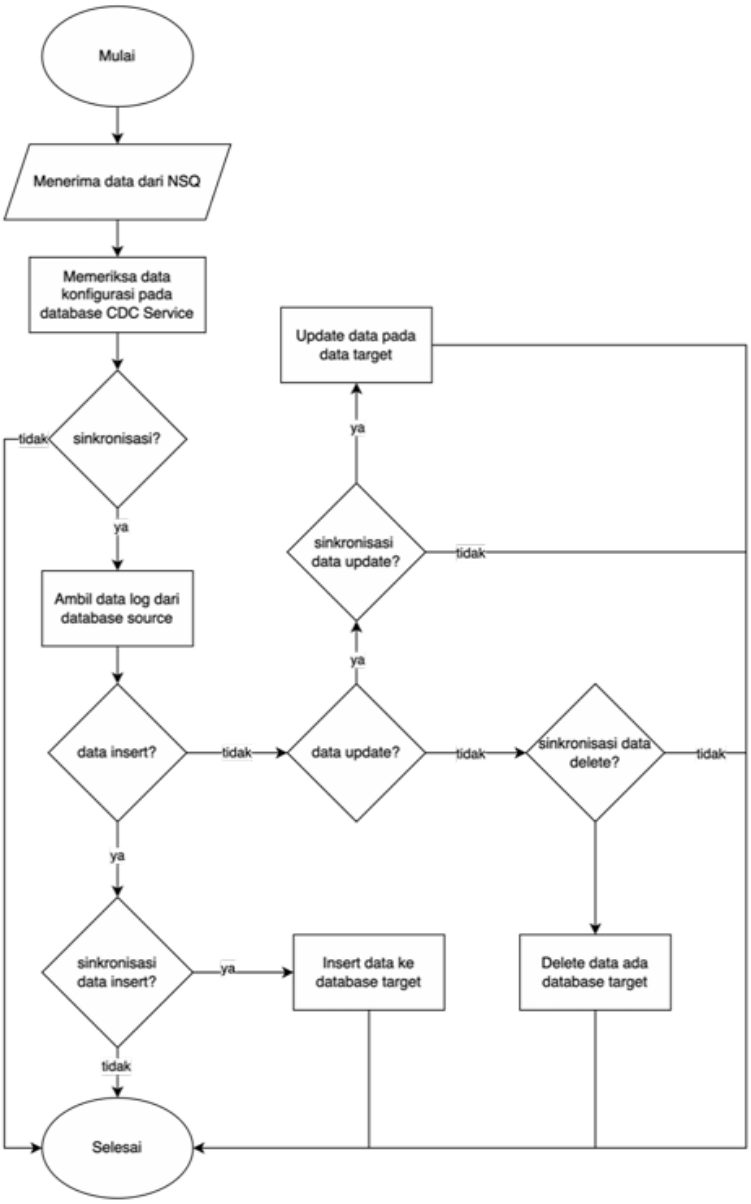
Sistem ini dimulai saat HTTP Service menerima *request* perubahan konfigurasi sinkronisasi dari Frontend Service, kemudian data perubahan tersebut akan disimpan di *database* CDC Service dan mengirimkan pesan ke NSQ. Saat NSQ menerima pesan maka pesan tersebut akan diteruskan ke *consumer*. *Consumer* dalam hal ini adalah CDC Service. CDC Service ketika menerima pesan dari NSQ akan melakukan pemeriksaan konfigurasi sinkronisasi pada database, jika konfigurasi sinkronisasi menunjukan perlu dilakukannya sinkronisasi, maka

p-ISSN: 2301-5373
e-ISSN: 2654-5101

proses sinkronisasi akan dilanjutkan pada pengamatan terhadap log dari *database source*. Ketika terdapat data baru yang masuk ke dalam log *database source*, data tersebut akan dibaca oleh CDC Service dan akan dilakukan pemeriksaan apakah tipe dari data tersebut perlu disinkronisasi sesuai dengan konfigurasi yang telah ditentukan. 19 Jika perlu, maka data tersebut akan diparsing terlebih dahulu menjadi data yang sesuai untuk dikirimkan ke *database target*.

2.3.4. Alur dari CDC Service

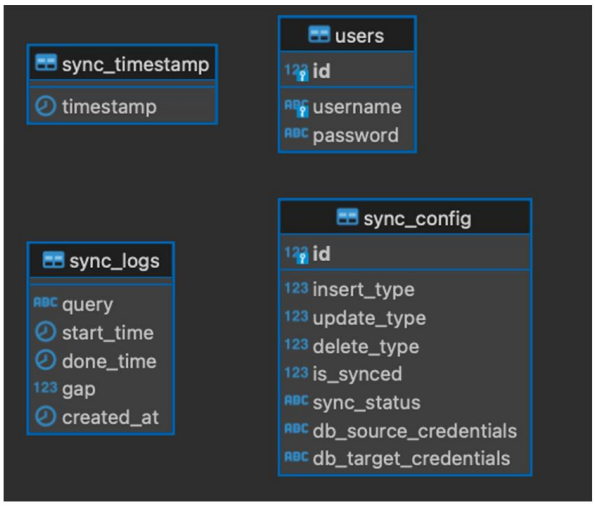
CDC Service akan memulai proses sinkronisasi dengan menghubungkan *service* ke NSQ agar CDC Service mendapatkan *trigger* ketika terdapat perubahan konfigurasi. Setelah menerima *trigger* dari NSQ, CDC Service akan melakukan pemeriksaan data konfigurasi sinkronisasi ke *database*. Jika data konfigurasi tersebut menunjukan bahwa sinkronisasi data harus dilakukan, maka CDC Service akan melakukan pengamatan data ke *log* dari *database source* dan akan melakukan sinkronisasi data saat data tersebut memang diinginkan untuk tersinkronisasi. Alur dapat dilihat pada Gambar 4.



Gambar 4. Alur Change Data Capture

2.3.5. Perancangan Skema Database CDC Service

Pada penelitian ini database CDC Service memiliki 4 tabel, terdapat tabel users yang digunakan untuk menyimpan data pengguna, kemudian terdapat tabel sync_config yang digunakan untuk menyimpan konfigurasi sinkronisasi dan terdapat pula tabel sync_timestamp yang digunakan untuk menyimpan timestamp terbaru dari data yang telah di sinkronisasi serta terdapat table sync_logs untuk menyimpan history proses sinronisasi. Untuk melihat skema dan atribut tabel dengan lebih jelas dapat dilihat pada Gambar 5.



Gambar 5. Rancangan Skema Database CDC Service

2.3.6. Perancangan Enkripsi Kredensial Database

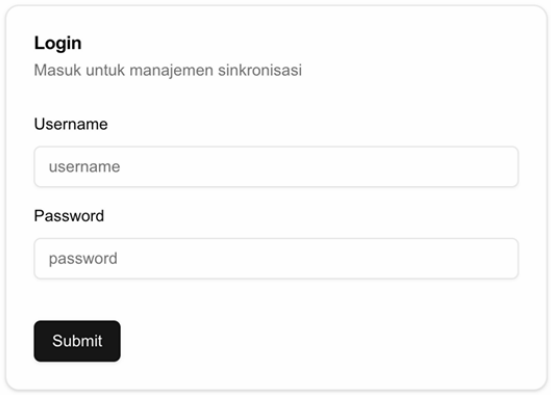
Seperti pada Gambar 5, tabel sync_config memiliki dua kolom bernama db_source_credentials dan db_target_credentials. Kedua kolom ini menyimpan string panjang hasil enkripsi yang berisi informasi kredensial lengkap untuk mengakses masing-masing database.

Proses enkripsi menggunakan algoritma HMAC SHA-256, yang menggabungkan hashing dan encoding. Hasil enkripsi berupa string panjang yang tetap dapat didekode, proses enkripsi dan validasi memerlukan secret key sebagai *signature*. Dengan secret key yang berbeda, meskipun data kredensial memiliki format dan isi yang sama, hasil enkripsinya tetap tidak valid. Hal ini memastikan bahwa jika ada perubahan pada nilai kolom tersebut yang tidak dilakukan oleh sistem, data tetap aman, dan proses sinkronisasi tidak akan terjadi.

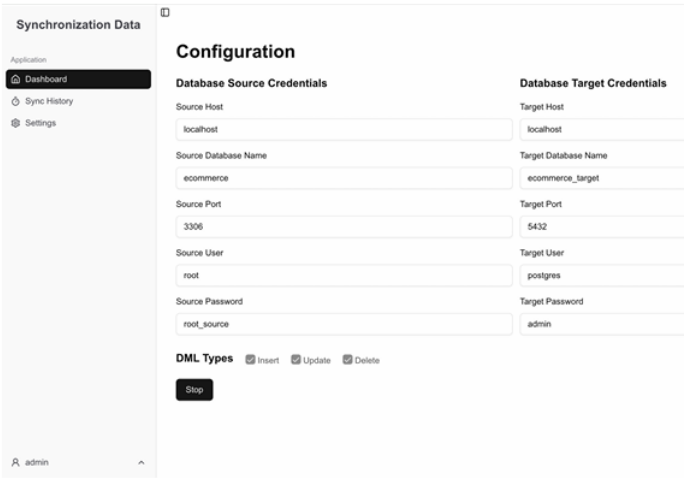
2.3.7. Perancangan Tampilan Antar Muka

Terdapat beberapa rancangan antar muka untuk mendukung dari penggunaan sistem yang dibuat. Terdapat halaman login pada Gambar 6 agar *user* dapat masuk kedalam sistem, kemudian terdapat halaman dashboard sinkronisasi pada Gambar 7 yang menjadi halaman untuk mengatur konfigurasi sinkronisasi, dan juga terdapat halaman pengaturan pada Gambar 8 untuk mengganti password user serta terdapat halaman history sinkronisasi yang dapat dilihat pada Gambar 9.

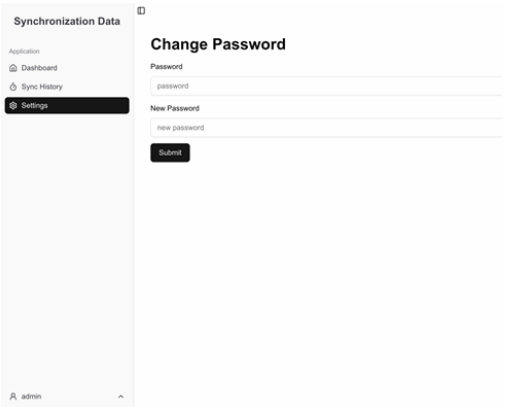
p-ISSN: 2301-5373
e-ISSN: 2654-5101



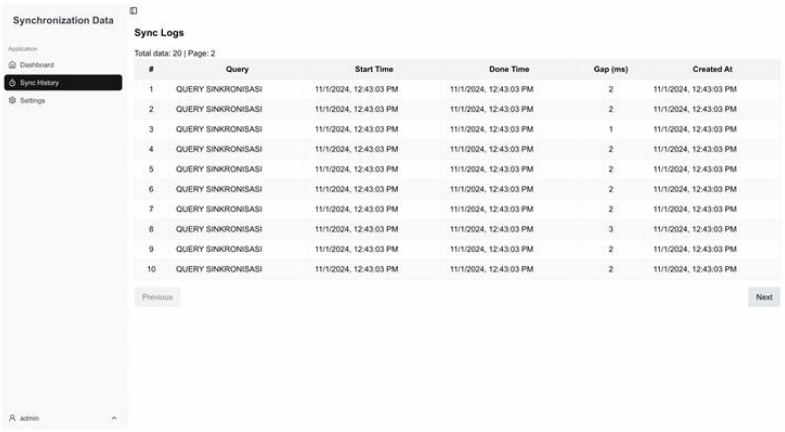
Gambar 6. Rancangan Halaman Login



Gambar 7. Rancangan Halaman Dashboard Sinkronisasi



Gambar 8. Rancangan Halaman Pengaturan



Gambar 9. Rancangan Halaman *Synchonization History*

2.3.8. Implementasi

Implementasi dilakukan dengan membangun sistem sesuai dengan desain yang telah dibuat. Proses pengerjaan atau implementasi akan menggunakan Visual Studio Code untuk melakukan proses pemrograman, DBeaver untuk mengintegrasikan *database* dan beberapa *service* pendukung seperti Docker dan NSQ. Selama proses implementasi akan menggunakan metodologi *Waterfall*, yang dimana tahap awal adalah perancangan dan juga analisis kebutuhan dimana penulis akan melakukan penelitian mengenai kebutuhan dan juga perancangan sistem. Kemudian penulis akan segera melakukan pengembangan dan terakhir melakukan pengujian pada tahap akhir pengerjaan dengan metodologi ini.

2.3.9. Pengujian dan Evaluasi Sistem

Pengujian sistem bertujuan untuk melihat apakah sistem yang telah dibuat sesuai dengan tujuan awal diproduksi dan layak untuk digunakan. Pengujian pada sistem ini menggunakan metode *BlackBox Testing* dan pengujian terhadap *system throughput* serta *loss analysis*. Tujuannya adalah untuk memastikan bahwa fungsi dan komponen dalam sistem aplikasi berjalan dengan benar, memperkirakan kecepatan proses terjadinya sinkronisasi data, dan menampilkan pesan kesalahan dengan benar jika terjadi kesalahan input data. Pengujian *BlackBox* menggunakan *Positive* dan *Negative Test Case* untuk mengevaluasi fungsionalitas sistem tanpa mengetahui detail implementasinya. *Positive Test Case* dirancang untuk

memastikan bahwa sistem bekerja dengan baik ketika diberikan data atau kondisi yang valid. *Negative Test Case* bertujuan untuk menguji bagaimana sistem merespons kondisi yang tidak valid. Dengan menggabungkan kedua jenis pengujian ini, pengembang dapat mengevaluasi keandalan sistem dalam menangani berbagai skenario, baik yang diantisipasi maupun tidak.
Untuk mengukur *system througput*, akan dilakukan perhitungan dengan menghitung rata-rata kecepatan sinkronisasi data. Perhitungan tersebut dapat dilihat pada persamaan (1).

$$T_{rata} = \frac{T_{total}}{N} \quad (1)$$

Keterangan:
T_{rata} : Waktu rata-rata yang dibutuhkan untuk sinkronisasi setiap data.
T_{total} : Total waktu yang diperlukan untuk sinkronisasi seluruh data.
N : Jumlah data yang disinkronisi

Perhitungan untuk mendapatkan *data loss* / *loss analysis* dapat dilihat pada persamaan (2)

$$Data\ Loss\ Rate = \frac{Jumlah\ Data\ Hilang}{Total\ Data} \times 100\% \quad (2)$$

3. Hasil dan Diskusi

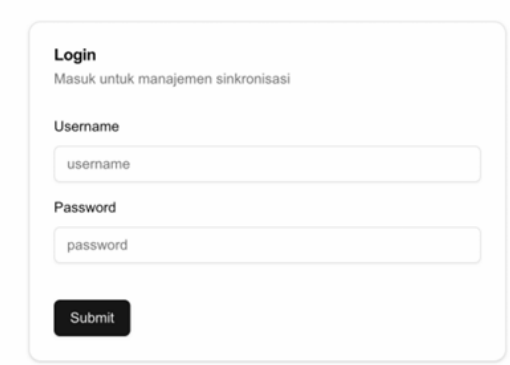
3.1. Hasil Pengujian Halaman Login

Pengujian yang dilakukan pada halaman login dapat dilihat pada Tabel 1.

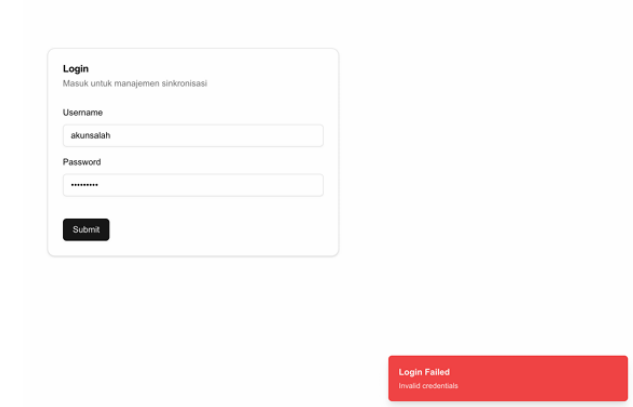
Tabel 1. Test Case Halaman Login

No	Skenario pengujian	Test Case	Harapan Hasil	Hasil	Keterangan
1	Halaman login menampilkan form input username dan password serta tombol submit.	Memeriksa tampilan awal halaman login dengan form input untuk username, password, dan tombol submit.	Halaman login menampilkan form input untuk username, password, dan tombol submit.	Halaman login menampilkan form input untuk username dan password, serta tombol submit sesuai dengan Gambar 12.	Sesuai
2	Menampilkan pesan ketika username dan password salah	Memasukkan username dan password yang salah, lalu menekan tombol submit untuk login	Pesan error muncul yang memberi tahu bahwa username atau password yang dimasukkan salah	Pesan error muncul dengan teks "Username atau password salah" sesuai dengan Gambar 13.	Sesuai

p-ISSN: 2301-5373
e-ISSN: 2654-5101



Gambar 10. Halaman Login Awal



Gambar 11. Halaman Login Gagal

Berdasarkan hasil yang dapat dilihat pada Tabel 1 yang dibuktikan pada Gambar 10 dan Gambar 11, hasil pengujian halaman login sudah sesuai dengan yang diharapkan.

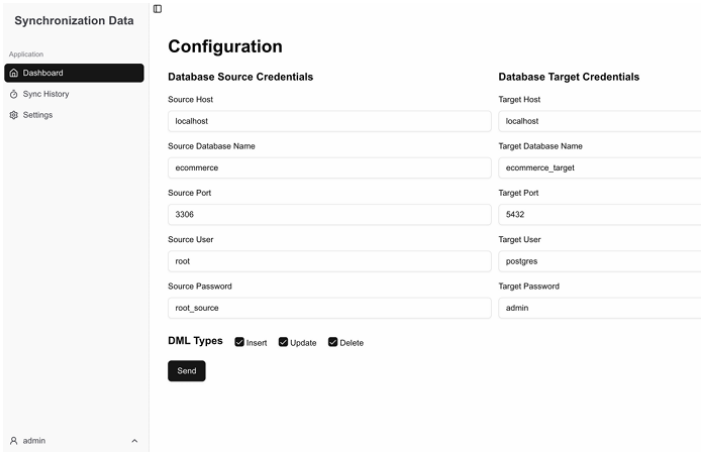
3.2. Hasil Pengujian Halaman *Dashboard* Sinkronisasi

Pengujian yang dilakukan pada halaman *dashboard* sinkronisasi dapat dilihat pada Tabel 2.

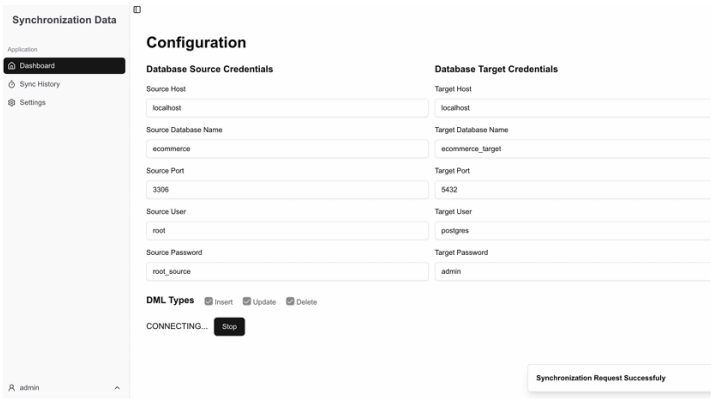
Tabel 2. Test Case Halaman *Dashboard* Sinkronisasi

No	Skenario pengujian	Test Case	Harapan Hasil	Hasil	Keterangan
1	Halaman dashboard menampilkan form input kredensial database dan jenis DML serta tombol “send” ketika proses sinkronisasi tidak dilakukan	Memeriksa tampilan awal dashboard sebelum sinkronisasi dilakukan.	Halaman menampilkan form input kredensial database, dropdown jenis DML, dan tombol "send".	Halaman menampilkan form input, dropdown jenis DML, dan tombol “send” sesuai dengan Gambar 14.	Sesuai
2	Halaman dashboard menampilkan form input kredensial database dan jenis DML serta tombol “stop” dan status “connecting” ketika tombol “send” diklik	Mengklik tombol "send" setelah mengisi form input kredensial database dan jenis DML.	Halaman menampilkan form input, tombol "stop", dan status "connecting"	Setelah tombol “send” diklik, halaman menampilkan tombol “stop” dan status “connecting” seperti pada Gambar 15.	Sesuai
3	Halaman dashboard menampilkan form input kredensial database dan jenis DML serta tombol “stop” ketika proses sinkronisasi berjalan.	Melihat status sinkronisasi pada dashboard saat sinkronisasi sedang berjalan.	Halaman menampilkan form input kredensial database, tombol "stop", dan tidak ada tombol "send" .	Proses sinkronisasi berjalan, dan halaman menampilkan tombol “stop” seperti pada Gambar 16.	Sesuai

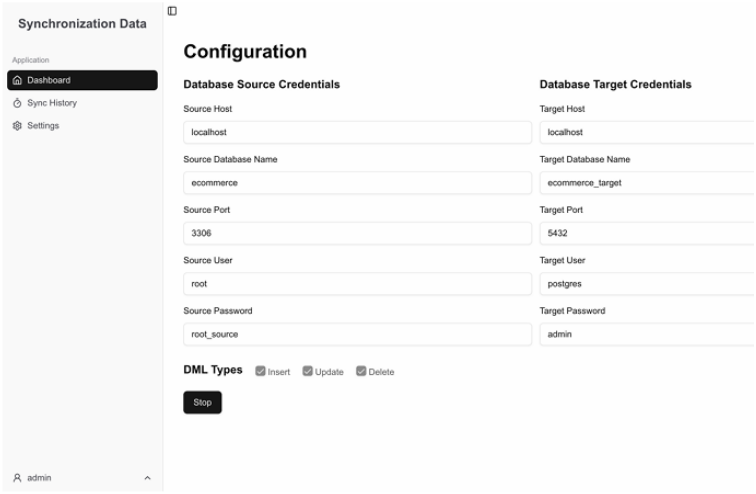
4	Setiap field input yang kosong pada halaman dashboard sinkronisasi akan memberikan pesan tidak boleh kosong	Mengosongkan salah satu atau semua field pada form dashboard sinkronisasi, lalu mencoba melanjutkan proses dengan menekan tombol "send".	Pesan error muncul di bawah field yang kosong dengan teks "Field ini tidak boleh kosong".	Pesan error ditampilkan di bawah field kosong, seperti pada Gambar 17.	Sesuai



Gambar 12. Halaman *Dashboard* Sinkronisasi Awal

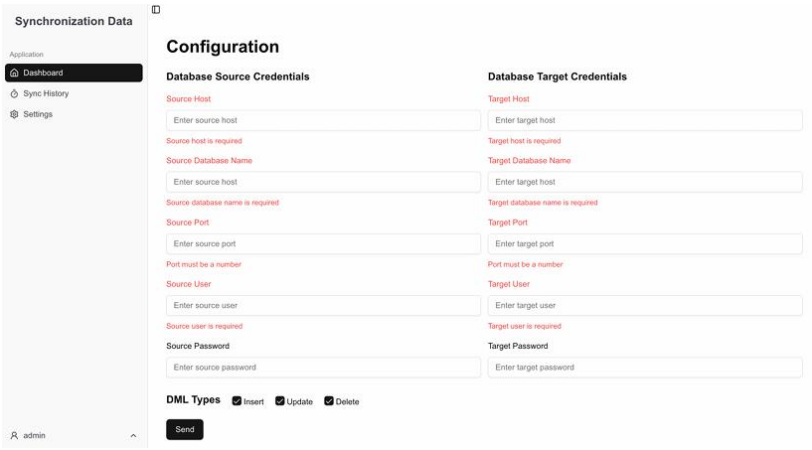


Gambar 13. Halaman *Dashboard* Sinkronisasi Proses Request



Gambar 14. Halaman *Dashboard* Sinkronisasi Berjalan

p-ISSN: 2301-5373
e-ISSN: 2654-5101



Gambar 15. Halaman *Dashboard* Sinkronisasi Kosong

Berdasarkan hasil yang dapat dilihat pada Tabel 2 yang dibuktikan pada Gambar 12, Gambar 13, Gambar 14, dan Gambar 15 hasil pengujian halaman *dashboard* sinkronisasi sudah sesuai dengan yang diharapkan.

3.3. Hasil Pengujian *System Throughput* dan *Loss Analysis*

Perhitungan *system throughput* dan loss analysis dilakukan dengan melakukan sinkronisasi 1 juta data pada setiap perangkat, terdapat 4 perangkat yang digunakan untuk melakukan pengujian ini dengan spesifikasi yang berbeda. Hasil data dari pengujian yang telah dilakukan dapat dilihat pada Tabel 3.

Tabel 3. Hasil Pengujian *System Throughput* dan *Loss Analysis*

Spesifikasi Perangkat	Total Kecepatan (ms)	Rata-rata Kecepatan (ms)	Waktu tercepat (ms)	Waktu Terlama (ms)	Data Hilang
CPU: Apple M1 RAM: 8GB	2.662.205	2,6622	1	447	0
CPU: Intel Core i7 1165G7 (4-core, 2,8GHz) RAM: 8GB	3.028.300	3,0283	1	509	0
CPU: Intel Core i5 1135G7 (4-core, 2,4GHz) RAM: 8GB	3.786.600	3,7866	1 .	573	0
CPU: Intel Core i3 1005G1 (2-core, 1,2GHz) RAM: 4GB	5.250.200	5,2502	1	699.	0

Berdasarkan hasil dari Tabel 3, dengan pengujian sinkronisasi 1 juta data secara beruntun, proses sinkronisasi memerlukan waktu yang bervariasi tergantung dengan spesifikasi perangkat yang digunakan, namun walaupun terdapat perbedaan waktu yang dibutuhkan untuk melakukan sinkronisasi, pengujian pada 4 perangkat tersebut tidak terdapat adanya data yang hilang.

4. Kesimpulan

Berdasarkan penelitian yang telah dilaksanakan dan juga hasil yang diperoleh dari penelitian, dapat ditarik beberapa kesimpulan sebagai berikut:

a. Change Data Capture (CDC) adalah sebuah metode yang dapat digunakan untuk mendeteksi perubahan data dalam sistem basis data. CDC memungkinkan sistem untuk memantau dan merekam setiap perubahan yang terjadi pada data, seperti operasi penambahan, penghapusan, atau pembaruan, secara real-time. Dengan memanfaatkan metode ini, perusahaan dapat memastikan bahwa data yang berada di berbagai sistem selalu up-to-date tanpa harus melakukan pemrosesan ulang seluruh dataset. Penggunaan CDC dan arsitektur Event-Driven secara bersamaan memberikan berbagai keuntungan signifikan. Selain memastikan bahwa data di seluruh sistem tetap konsisten, metode ini juga membantu mengurangi beban kerja server karena hanya perubahan data yang diproses, bukan seluruh dataset.
b. Sistem sinkronisasi data yang dirancang menggunakan metode Change Data Capture (CDC) dan Event-Driven Architecture terbukti dapat melakukan sinkronisasi dengan baik berdasarkan hasil pengujian. Dalam pengujian dengan 1.000.000 data yang dilakukan pada 4 perangkat dengan spesifikasi berbeda, sistem ini mencapai performa rata-rata waktu pemrosesan sebesar 2,6622 milidetik hingga 5,2502 milidetik per data tergantung dari spesifikasi perangkat yang digunakan. Selain itu, pengujian pada 4 perangkat dengan spesifikasi berbeda juga memperoleh hasil data yang hilang sebanyak 0 data.
c. Sistem sinkronisasi yang telah dibangun juga dapat digunakan dengan baik oleh pengguna, dibuktikan dengan hasil pengujian blackbox memperoleh hasil yang sesuai pada setiap test-case.

Referensi

[1] A. Tangudu, P. K. G. Pandian and S. Jain, "Developing Scalable APIs for Data Synchronization in Salesforce Environments," *Modern Dynamics: Mathematical Progressions,* vol. 1, no. 2, pp. 44-57, 2024.

[2] I. G. Adnyana and I. M. D. J. Sulastra, "Implementation of Data Backup and Synchronization Based on Identity Column Real Time Data Warehouse," *Lontar Komputer : Jurnal Ilmiah Teknologi Informasi,* vol. 11, no. 1, p. 9, 2020.

[3] K. K. Ganeeb, V. Jayaram, M. S. Krishnappa, P. K. Veerapaneni and S. R. Sankiti, "A Comprehensive Study of Custom Change Data Capture on a Large Scale RDBMS," *International Journal of Research and Analytical Reviews,* vol. 11, no. 3, p. 838, 2024.

[4] H. Setiawan, A. A. A. Sumitro and R. Gustriansyah, "The Change Data Capture and the Web Application Messaging Protocol on the Real Time Dashboard.," *International Journal of Engineering and Advanced Technology (IJEAT),* vol. 8, no. 4, 2019.

[5] A. Mantri, "Event Driven Data Architecture: Design and Implementation with Kinesis and Spark Streaming," *International Journal of Science and Research (IJSR),* vol. 13, no. 7, pp. 653-655, 2024.