PENERAPAN ARSITEKTUR ENTERPRISE POLA FINANSIAL PADA APLIKASI BERBASIS *MICROSERVICES*

Disusun Oleh:

N a m a : Muhammad Hanif Faturohman

NIM : 1753082

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2020

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENERAPAN ARSITEKTUR ENTERPRISE POLA FINANSIAL PADA APLIKASI BERBASIS *MICROSERVICES*

TUGAS AKHIR

Disusun Oleh:

N a m a : Muhammad Hanif Faturohman

NIM : 1753082

Yogyakarta, 1 September 2021

Pembimbing,

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

HALAMAN PENGESAHAN DOSEN PENGUJI

PENERAPAN ARSITEKTUR ENTERPRISE POLA FINANSIAL PADA APLIKASI BERBASIS *MICROSERVICES*

TUGAS AKHIR

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

Yogyakarta, 4 Agustus 2021

Tim Penguji

Dr. Raden Teduh Dirgahayu, S.T., M.Sc. _____

Anggota 1

Hendrik, S.T., M.Eng. _____

Anggota 2

Andhika Giri Persada, S.Kom., M.Eng. _____

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

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

iv

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Muhammad Hanif Faturohman

NIM : 1753082

Tugas akhir dengan judul:

PENERAPAN ARSITEKTUR ENTERPRISE POLA FINANSIAL PADA APLIKASI BERBASIS *MICROSERVICES*

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

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

Cilegon, 1 September 2021

(Muhammad Hanif Faturohman)

HALAMAN PERSEMBAHAN

Al-hamdu lillahi rabbil 'alamin, segala puji dan syukur atas nikmat umur, iman, dan rezeki yang berlimpah yang telah Allah SWT berikan. Sholawat serta salam selalu terucap kepada Rasulullah Muhammad. *Alhamdulillah* dengan izin Allah SWT penulis dapat menjalani seluruh proses perkuliahan serta menyelesaikan tugas akhir berupa penelitian ini. Semoga apa yang menjadi tujuan dari penelitian ini dapat tersampaikan serta memberikan manfaat tidak hanya bagi saya selaku penulis tetapi juga untuk yang membaca. Penelitian ini saya persembahkan khususnya kepada keluarga saya, Ayah, Ibu, serta Adik yang selalu memberikan dukungan yang tak terhitung dalam perjalanan kuliah saya. Selain itu, penelitian ini juga saya persembahkan kepada kedua dosen pembimbing saya, bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., dan bapak Hanson Prihantoro Putro, S.T., M.T. serta seluruh dosen informatika yang telah memberikan ilmu, waktu, dan juga sabar dalam membimbing saya. Tidak lupa saya persembahkan juga kepada seluruh rekan saya yang sudah memberikan semangat dan bantuan kepada saya, serta seluruh pihak yang terlibat dalam pembuatan laporan penelitian ini.

HALAMAN MOTO

*"Masih banyak hal yang tidak bisa kulakukan, jadi aku harus **bekerja lebih keras** lagi."*

(Midoriya Izuku)

*"Jika aku tidak **bekerja lebih keras** dari orang lain, aku tak bisa apa-apa."*

(Midoriya Izuku)

***Bekerja lebih keras** dan lampaui terus batasanmu, plus ultra!!!*

(All Might)

KATA PENGANTAR

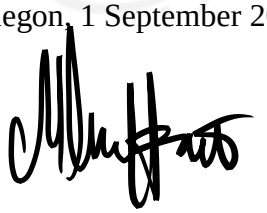
Assalamualaikum Wr. Wb.

Al-hamdu lillahi rabbil 'alamin, segala puji dan syukur atas nikmat umur, iman, dan rezeki yang berlimpah yang telah Allah SWT berikan. Shalawat serta salam penulis ucapkan kepada Nabi Muhammad *Sallallahu Alaihi Wasallam*. Puji syukur yang sebesar-besarnya selalu dihaturkan kepada Allah terutama atas berkah dan izinNya, proses kuliah hingga penulisan laporan tugas akhir yang berupa penelitian berjudul "Penerapan Arsitektur Enterprise Pola Finansial pada Teknologi *Microservices"* dapat diselesaikan dengan baik. Tidak lupa juga saya berterima kasih kepada pihak yang mendukung serta membantu. Untuk itu, saya menyampaikan rasa terima kasih saya kepada:

1. Ayah, Ibu, dan Adik saya, Terima kasih kepada kedua orang tua yang selalu memberikan semangat serta doa untuk menyelesaikan tugas akhir.
2. Dosen pembimbing dan seluruh dosen Informatika, Terima kasih kepada dosen pembimbing saya, bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., dan bapak Hanson Prihantoro Putro, S.T., M.T. serta seluruh dosen informatika yang telah memberikan ilmu, waktu, dan juga sabar dalam membimbing saya.
3. Seluruh rekan saya, Terima kasih kepada teman-teman yang juga memberikan semangat dan bantuan kepada saya.
4. Seluruh pihak yang terlibat, Terima kasih juga kepada seluruh pihak yang terlibat dalam pembuatan laporan tugas akhir saya atas dukungannya.

Laporan ini telah dibuat dengan usaha terbaik dari penulis, tetapi masih jauh dari kata sempurna. Sehingga diperlukan saran dan kritik yang membangun dari pembaca untuk penyempurnaan laporan ini. Akhir kata, penulis berharap dari disusunnya laporan ini dapat memberikan manfaat bagi semua pihak.

Cilegon, 1 September 2021



(Muhammad Hanif Faturohman)

SARI

Arsitektur enterprise merupakan cetak biru dari enterprise yang mewakili seluruh visi, misi, serta fungsionalitasnya. Sedangkan, *microservices* merupakan arsitektur teknologi yang sedang populer di berbagai enterprise dalam lima tahun belakangan. Sejauh ini, literatur mengenai penerapan arsitektur enterprise pada aplikasi berbasis *microservices* masih sulit ditemukan. Masalah ini tentu menyulitkan beberapa pihak enterprise yang ingin menerapkan arsitektur enterprisenya menjadi aplikasi berbasis *microservices*. Oleh karena itu melalui makalah ini, penulis bertujuan untuk menambah literatur terkait hal tersebut dengan menyediakan deskripsi mengenai bagaimana proses penerapan arsitektur enterprise menjadi sebuah aplikasi berbasis *microservices* jika menggunakan metodologi yang diusulkan. Studi kasus yang digunakan pada penelitian ini adalah arsitektur enterprise pola finansial yang sudah tersedia sebelumnya. Metodologi untuk menerapkan arsitektur enterprise pola finansial tersebut secara berurutan yakni: dekomposisi arsitektur enterprise pola finansial, analisis kebutuhan API layanan, analisis kebutuhan basis data, penerapan basis data, penerapan API layanan, pengujian API layanan, penerapan antarmuka, pengujian antarmuka. Hasil penerapan dari arsitektur enterprise pola finansial tersebut adalah aplikasi finansial yang mampu mengelola kebutuhan finansial. Dapat ditarik kesimpulan bahwa penelitian ini memiliki delapan tahap pada metodologi, serta menghasilkan aplikasi finansial berbasis *microservices* yang dapat mengelola kebutuhan finansial. Diharapkan penelitian ini dapat memberikan referensi terkait penerapan arsitektur enterprise pada aplikasi berbasis *microservices*.

Kata kunci: arsitektur enterprise, arsitektur enterprise pola finansial, penerapan arsitektur enterprise ,*microservices*, metodologi penerapan *microservices*, penerapan *microservices*,

GLOSARIUM

API Layanan	Sebuah program tanpa *Graphic User Interface* (GUI) yang bisa diakses pengguna untuk memodifikasi program lainnya, misal: API layanan lain, basis data, dan sebagainya.
Collection	Merupakan sebutan untuk entitas pada basis data MongoDB

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
Pendahuluan

1.1 Latar Belakang

Arsitektur Enterprise (AE) merupakan deskripsi mengenai enterprise dari segi bisnis, sistem Teknologi Informasi (TI), beserta korelasinya (Tamm et al., 2011). Beberapa hal yang dideskripsikan dalam AE adalah : tujuan bisnis, strategi bisnis, prinsip bisnis, proses bisnis, kebutuhan data, kebutuhan teknologi dan infrastuktur, dan lain lain (Perroud & Inversini, 2013). Deksripsi ini dapat berbentuk sebagai dokumen, diagram, atau artefak lainnya (Tamm et al., 2011). Pada enterprise, AE digunakan untuk menyelaraskan aktivitas TI dengan tujuan bisnis dari enterprise. Penggunaan AE ini diharapkan dapat meminimalisir kesalahan ketika mengambil keputusan dalam pengembangan sistem TI (Perroud & Inversini, 2013).

Menariknya, berbagai enterprise kerap kali mendapat masalah yang sama jika situasi dan kondisinya terpenuhi. Kejadian itu lantas memicu munculnya sebuah konsep yang bernama pola AE. Secara definisi, pola AE ini merupakan sebuah generalisasi dari AE untuk menyelesaikan masalah yang kerap kali terjadi di berbagai enterprise. Sejauh ini tidak ada aturan khusus untuk mendokumentasikan pola AE, tetapi biasanya pola AE mendokumentasikan masalah serta solusinya dalam format yang sama dengan dokumentasi dari AE. Selaras dengan AE, penggunaan pola AE juga diharapkan dapat meminimalisir kesalahan ketika mengambil keputusan (Perroud & Inversini, 2013).

Salah satu pola AE yang tersedia pada buku (Perroud & Inversini, 2013) adalah pola finansial. Secara definisi, finansial merupakan cabang ilmu yang mempelari mengenai sebab-akibat sebuah aset terhadap individu atau organisasi dalam jangka waktu pendek maupun panjang. Finansial ini merupakan kegiatan manajemen yang menjadi fondasi utama dalam setiap enterprise. Dalam bidang finansial juga terdapat beberapa masalah yang dapat diselesaikan dengan solusi tertentu, misal: sering terlambat membayar tagihan dapat diselesaikan dengan menggunakan sistem TI yang dapat menyortir informasi tagihan. Hal semacam itulah yang kemudian dirumuskan menjadi sebuah AE pola finansial.

Pada saat ini sedang terjadi banyak revolusi arsitektur teknologi di berbagai enterprise dari arsitektur monolitik ke arsitektur *microservices*. Di tahun 2020, sudah terdapat 78% organisasi yang menggunakan *microservices*, dan 61% di antaranya baru menggunakan *microservices* sejak satu hingga lima tahun belakangan ini (Loukides & Swoyer, 2020). Keterbatasan

arsitektur monolitik pada skalabilitas dan ketergantungan kuat antar modulnya membuat beberapa enterprise kesulitan untuk mengimbangi kompleksitas bisnisnya yang makin meningkat dari tahun ke tahun dan beralih ke *microservices* (Indrasiri & Siriwardena, 2018)

Arsitektur *microservices* ini merupakan arsitektur yang menekankan kepada pembagian aplikasi menjadi layanan-layanan yang berukuran kecil, bersifat independen, serta dapat saling berkomunikasi (Lewis & Fowler, 2014). Kecil di sini berarti bahwa tiap layanan hanya mewakili proses bisnis tertentu, sedangkan independen di sini berarti bahwa tiap layanan dapat berjalan otomatis tanpa terikat dengan layanan lain (Lewis & Fowler, 2014; Newman, 2015). Konsep *microservice* sendiri dipopulerkan oleh Martin Fowler lewat beberapa publikasinya, serta Netflix lewat dokumentasi proyeknya (Dragoni et al., 2018; Soldani et al., 2018).

Namun masalahnya, sejauh ini literatur yang menyediakan deskripsi penerapan AE atau pola AE pada aplikasi yang berbasis *microservices* masih sulit ditemukan. Oleh karena itu, disusunlah penelitian ini dengan tujuan untuk menambah literatur terkait hal tersebut. Penelitian ini menggunakan AE pola finansial yang tersedia pada buku (Perroud & Inversini, 2013) sebagai studi kasus untuk diterapkan pada aplikasi berbasis *microservices* menggunakan metodologi pengembangan yang diusulkan. Diharapkan penelitian ini dapat memberikan referensi terkait penerapan AE atau pola AE pada aplikasi berbasis *microservices* bagi seluruh pihak yang terkait.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam penelitian ini adalah bagaimana cara menerapkan AE pola finansial yang tersedia pada buku (Perroud & Inversini, 2013) jika menggunakan metodologi pengembangan yang diusulkan.

1.3 Batasan Masalah

Berikut merupakan hal yang bukan merupakan fokus dari penelitian ini:

a. Urgensi penggunaan AE pola finansial,
b. Komparasi pola finansial dengan pola lain,
c. Komparasi keunggulan *microservices* dengan monolitik atau arsitektur teknologi lainnya pada penerapannya,
d. Komparasi performa metodologi yang digunakan pada penelitian ini dengan metodologi lainnya.
e. Komparasi performa MERN *stack* dengan *stack* yang lain,

f. Kesesuaian fitur pada hasil aplikasi dengan kasus tertentu pada realita,
g. Proses desain UI/UX pada antarmuka web.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk melihat deskripsi proses penerapan dari AE pola finansial yang tersedia pada buku (Perroud & Inversini, 2013) pada aplikasi berbasis *microservices* menggunakan metodologi yang diusulkan.

1.5 Manfaat Penelitian

1.5.1 Manfaat Bagi Enterprise

Mereka dapat mengetahui langkah-langkah penerapan pola AE menjadi aplikasi berbasis *microservices.*

1.5.2 Manfaat Bagi *Stakeholder* pada Domain Finansial

Mereka dapat mengetahui seperti apa aplikasi yang digunakan untuk mengelola kegiatan finansial pada tingkat enterprise.

1.5.3 Manfaat Bagi Pengembang Aplikasi

Mereka dapat meminimalisir kesalahan dalam mengambil keputusan ketika menerapkan AE menjadi aplikasi berbasis *microservices.*

1.6 Metodologi Penelitian

Berikut merupakan metodologi penelitian yang diusulkan dan digunakan pada penelitian ini:

a. Dekomposisi AE pola finansial

Pada tahap ini dilakukan proses mendekomposisi AE pola finansial menjadi bagian-bagian yang lebih kecil. Hal ini dilakukan agar AE pola finansial sesuai dengan prinsip *microservices*, yakni: kecil, dan independen. Pada penelitian ini strategi dekomposisi yang digunakan adalah strategi dekomposisi menggunakan *Domain Driven Design* (DDD). Meski sebenarnya DDD bukan strategi yang dibuat khusus untuk penerapan *microservices,* namun strategi ini merupakan salah satu pilihan terbaik ketika mau mendekomposisi sebuah aplikasi atau rancangan aplikasi (Richards, 2018)

b. Analisis kebutuhan pada API layanan

Pada tahap ini dilakukan penggalian informasi terkait kebutuhan API layanan. Tahapan ini dilakukan agar seluruh API layanan dapat berjalan lancar ketika diterapkan. Beberapa informasi yang digali disini antara lain: fungsi pada tiap API layanan, aliran kolaborasi tiap fungsi, format komunikasi yang digunakan, dan sebagainya.

c. Analisis kebutuhan pada basis data

Pada tahap ini dilakukan penggalian informasi terkait data yang dibutuhkan oleh tiap API layanan. Tahapan ini dilakukan untuk meminimalisir revisi pada basis data, atau baris kode aplikasi akibat perubahan data ketika aplikasi sudah berjalan (*live*).

d. Penerapan basis data

Pada tahap ini dilakukan pembangunan basis data sesuai hasil analisis kebutuhan yang sudah dilakukan pada tahap sebelumnya.

e. Penerapan API layanan

Pada tahap ini dilakukan pembangunan API layanan sesuai hasil analisis kebutuhan yang sudah dilakukan pada tahap sebelumnya.

f. Pengujian API layanan

Pada tahap ini dilakukan pengujian pada API layanan yang sudah dikembangkan pada tahap sebelumnya. Tahap ini dilakukan untuk mengukur kualitas API layanan dengan cara mencari kemungkinan *error* pada tiap API layanan.

g. Penerapan antarmuka web

Pada tahap ini dilakukan pembangunan antarmuka agar API layanan semakin mudah untuk diakses oleh pengguna. Menurut pengalaman penulis sendiri, butuh ilmu apriori yang berkaitan agar dapat menjalankan aplikasi yang hanya berbentuk API layanan tanpa antarmuka. Selain itu, antarmuka ini juga merupakan sesuatu yang diusulkan dalam menerapkan AE pola finansial oleh buku (Perroud & Inversini, 2013). Pada penerapannya antarmuka ini diterapkan pada *platform web* dan dikembangkan secara terpisah dari API layanan.

h. Pengujian antarmuka

Pada tahap ini dilakukan pengujian pada antarmuka yang sudah dikembangkan pada tahap sebelumnya. Tahap ini dilakukan untuk mengukur kualitas antarmuka dengan cara mencari kemungkinan *error* pada antarmuka.

1.7 Sistematika Penulisan

Penulisan penelitian ini dibuat secara terstruktur untuk mengetahui apa saja pembahasan yang ada dalam setiap bab. Penelitian ini dibagi menjadi lima bab pembahasan. Berikut merupakan sistematika penulisan pada penelitian ini.

a. BAB I PENDAHULUAN

Bab ini berisi tentang pembahasan masalah umum yang menginisiasi topik ini untuk diangkat menjadi objek penelitian. Hal tersebut meliputi: latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

b. BAB II LANDASAN TEORI

Bab ini berisi teori dasar yang diambil dari berbagai sumber dalam rangka mendukung penelitian ini.

c. BAB III METODOLOGI PENELITIAN

Bab ini berisi tentang uraian terkait langkah-langkah yang dilakukan dalam menerapkan pola AE finansial pada aplikasi berbasis *microservices*.

d. BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi tentang hasil dari seluruh proses metodologi penelitian yang dilakukan beserta pembahasan dan dokumentasi atas hasil tersebut.

e. BAB V KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan dari keseluruhan tahap penelitian, serta saran untuk penelitian berikutnya berdasarkan hasil penelitian saat ini.

BAB II
Landasan Teori

2.1 Arsitektur Enterprise

2.1.1 Definisi

Pada umumnya AE dibuat oleh seorang arsitek enterprise. Dalam membuat AE, arsitek akan mulai dengan memodelkan perspektif dari seluruh pihak yang terkait. Selanjutnya untuk mengelola kompleksitasnya, arsitek akan mengkategorikan perspektif tersebut berdasarkan ranah arsitekturnya lalu menyusunnya ke dalam skema lapisan sesuai kerangka kerja AE yang digunakan.

Penggunaan AE yang baik dapat mendatangkan beberapa keuntungan. Berikut merupakan keuntungan dari penggunaan AE pada sebuah enterprise (Tamm et al., 2011):

a. Meningkatkan ketangkasan enterprise dalam menghadapi perubahan;
b. Menurunkan resiko kesalahan dalam mengambil keputusan;
c. Meningkatkan komunikasi dan kolaborasi antar *stakeholder* pada enterprise;
d. Menurunkan biaya dari sistem TI;
e. Menyelaraskan antara bisnis dan sistem TI;
f. Meningkatkan efektifitas proses bisnis;
g. Meningkatkan efektifitas sistem TI;
h. Menghemat sumber daya enterprise;
i. Meningkatkan integrasi dalam enterprise;
j. Standardisasi enterprise;
k. Meningkatkan stabilitas enterprise;

2.1.2 Kerangka Kerja Arsitektur Enterprise

Kerangka kerja AE menyediakan semacam panduan mengenai apa saja yang harus dilakukan ketika mengembangkan AE. Saat ini sudah banyak kerangka kerja AE yang dapat digunakan oleh para arsitek AE, misal: Zachman, DoDAF, NAF, FEAF, TOGAF. Pada penelitian ini, kerangka kerja yang digunakan adalah kerangka kerja TOGAF.

TOGAF dikembangkan oleh sebuah grup terbuka pada 1995. Panduan yang disediakan oleh TOGAF lebih banyak berupa prinsip. Hal tersebut cukup unik karena panduan yang disediakan oleh beberapa kerangka kerja yang lain lebih banyak berupa metode pembuatan.

Salah satu prinsip utama TOGAF dalam merancang AE adalah prinsip susunan blok. Secara singkat, prinsip ini merupakan prinsip yang memungkinkan enterprise menyusun AE seperti menyusun blok yang dapat dilepas-pasang sesuai kebutuhan. (Schekkerman, 2004). Berikut empat susunan blok utama pada TOGAF yang biasanya disebut sebagai lapisan (Perroud & Inversini, 2013):

a. Lapisan Bisnis

Lapisan ini mendeskripsikan segala hal terkait bisnis yang dijalankan oleh organisasi seperti: strategi penjualan produk / jasa, struktur bisnis organisasi, dan sebagainya. Lapisan ini sangat penting sehingga lapisan lain bergantung pada lapisan ini.

b. Lapisan Data

Lapisan ini mendeksripsikan data yang digunakan atau dikumpulkan oleh proses bisnis. Lapisan ini mendeskripsikan bentuk data, relasi data, serta format data.

c. Lapisan Aplikasi

Lapisan ini bertanggung jawab atas segala aspek yang berkaitan dengan aplikasi untuk mendukung proses bisnis.

d. Lapisan Teknologi

Lapisan ini bertanggung jawab terhadap infrastruktur teknologi yang digunakan oleh aplikasi dalam rangka mendukung proses bisnis.

2.2 Pola Arsitektur Enterprise

Pola merupakan cara untuk mendeskripsikan sebuah solusi terhadap masalah yang terjadi berulang-ulang. Dahulu kala, minimnya literatur mengenai pola AE membuat banyak enterprise terpaksa melakukan eksperimen dalam menyelesaikan sebuah masalah. Hal tersebut lantas memicu beberapa organisasi untuk mengumpulkan dan mendokumentasikan pengalaman berharga serta praktik terbaik yang dilakukan oleh berbagai enterprise dalam eksperimennya, yang kemudian diformulasikan menjadi sebuah pola (Perroud & Inversini, 2013).

Salah satu tujuan terpenting dari pola AE ini adalah *reusability*. *Reusability* ini merupakan konsep untuk menggunakan ulang suatu hal. Penggunaan konsep ini menghasilkan beberapa keuntungan, yakni: mengurangi jumlah pekerjaan, menghindari kesalahan yang sama, mempercepat waktu pengerjaan. Agar konsep *reusability* ini dapat diterapkan, sebuah pola AE harus diformulasikan sebagai dokumen yang mudah dipahami, dan terstandardisasi (Perroud & Inversini, 2013).

2.3 Arsitektur Enterprise Pola Finansial

2.3.1 Diagram Arsitektur Enterprise Pola Finansial

Berikut merupakan beberapa masalah finansial yang dialami banyak organisasi (Perroud & Inversini, 2013) :

a. Tagihan dibayarkan terlalu lama, hal ini membuat organisasi melewatkan potongan pembayaran jangka cepat atau bahkan terkena denda karena melebihi batas pembayaran.
b. Pengiriman tagihan kepada alamat yang salah.
c. Proses pengelolaan informasi finansial secara manual terlalu melelahkan.
d. Secara keseluruhan biaya manajemen finansial terlalu besar.
e. Banyak kekeliruan pada laporan finansial.
f. Data finansial yang hilang atau rusak.

Berdasarkan permasalahan di atas, berikut merupakan diagram AE pola finansial berdasarkan 3 perspektif yang didokumentasikan menggunakan kerangka kerja TOGAF (Perroud & Inversini, 2013).

A. Perspektif Holistik

Diagram perspektif holistik pada Gambar 2.1 menjelaskan garis besar dari tiga lapisan berdasarkan kerangka kerja TOGAF. Tiap lapisannya dipisahkan menggunakan garis putus-putus. Berikut detail penjelasan tiap lapisannya:

a. Lapisan bisnis

Lapisan ini memuat aktor beserta proses bisnisnya. Para aktor yang berperan adalah *financial staff* dan *business user*. *Financial staff* merupakan pengguna dari departemen finansial, sedangkan *business user* merupakan pengguna dari departemen selain departemen finansial. Perbedaan dari kedua aktor tersebut ada pada proses bisnis yang dapat mereka akses.

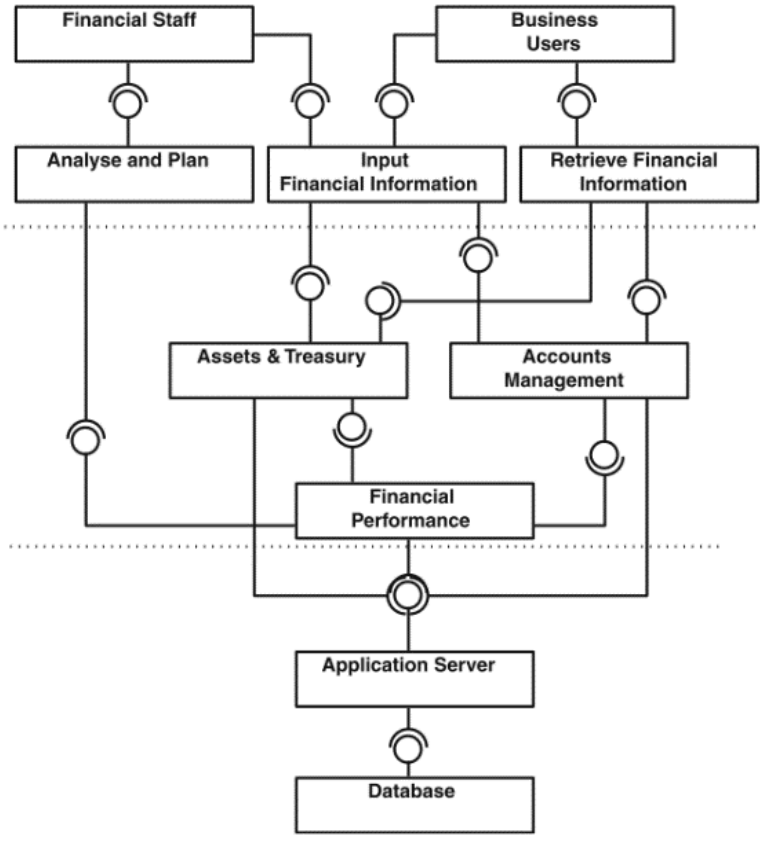
Kemudian, terdapat 3 proses bisnis, yakni : *analyse and plan*, *input financial information*, dan *retrieve financial information*. *Analyse and plan* digunakan untuk menganalisis dan merencanakan keuangan milik enterprise. *Input financial* digunakan untuk mencatat seluruh data mengenai kejadian yang melibatkan uang milik enterprise, seperti: membeli stok, menjual barang, dan sebagainya. *Retrieve financial information* digunakan untuk melihat seluruh kejadian finansial pada enterprise dalam bentuk grafik atau tabel.

b. Lapisan aplikasi

Lapisan ini memuat lapisan aplikasi yang sudah digabungkan. Terdapat 3 kelompok aplikasi, yakni: kelompok *asset & treasury*, kelompok *accounts management*, dan kelompok *financial performance*. *Asset & treasury* mengelola seluruh data perputaran aset pada enterprise. *Accounts management* mengelola seluruh akun rekening pada enterprise. *Financial performance* mengelola data laporan finansial.

c. Lapisan teknologi

Teknologi yang dideskripsikan dalam lapisan ini masih tergolong abstrak. Terdapat 2 teknologi yang dideskripsikan, yakni: *application server* dan *database*. *Application server* merupakan server yang menjalankan aplikasi dan dapat diakses oleh penggunanya secara *online. Database* merupakan tempat untuk menyimpan seluruh data milik *application server.*



Gambar 2.1 Perspektif holistik pola AE finansial (Perroud & Inversini, 2013)

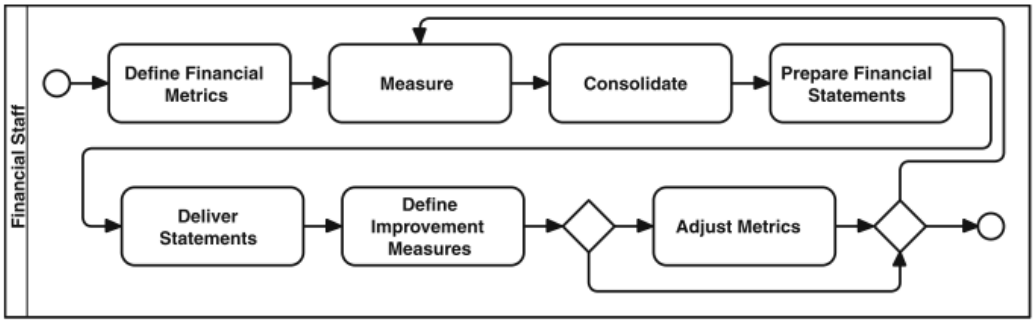
B. Perspektif Proses Bisnis

Perspektif ini mendeksripsikan proses bisnis yang terdapat pada perspektif holistik dengan lebih detail. Berikut penjelasan mengenai masing-masing proses bisnis pada pola AE finansial.

a. Proses bisnis *analyse and plan*

Proses bisnis ini hanya dapat diakses oleh aktor *financial staff*. Seperti yang terlihat pada Gambar 2.2, terdapat tujuh aktivitas dalam proses bisnis ini. Aktivitas *define financial metrics* memetakan ukuran atau metrik keuangan pada tiap akun rekening. Ativitas *measure* mengkalkulasi metrik yang sudah dibuat sebelumnya. Aktivitas *consolidate* menggabungkan seluruh metrik di tiap akun rekening. Aktivitas *prepare financial statements* mendokumentasikan metrik menjadi dokumen finansial.

Aktivitas *deliver statements* menyampaikan dokumen finansial kepada *stakeholder* enterprise. Aktivitas *define improvement measure* mendefinisikan peningkatan keuangan enterprise berdasarkan data finansial sebelumnya. Aktivitas *adjust metrics* mengatur ulang seluruh metrik yang dibuat pada aktivitas sebelumnya.

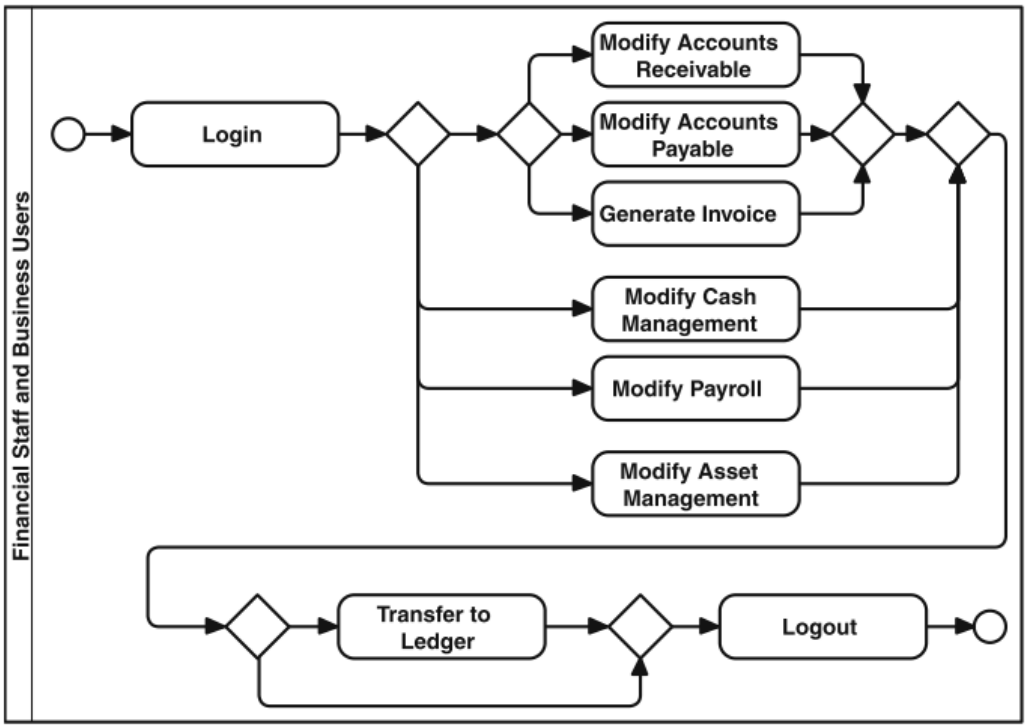


Gambar 2.2 Proses bisnis *analyse and plan* (Perroud & Inversini, 2013)

b. Proses bisnis *input financial information*

Proses bisnis ini dapat diakses oleh aktor *financial staff* dan *business user*. Seperti yang terlihat pada Gambar 2.3, Terdapat sembilan aktivitas dalam proses bisnis ini. Aktivitas *login* memverifikasi keabsahan pengguna. Enam aktivitas berikutnya merupakan aktivitas untuk memodifikasi data keuangan enterprise. Aktivitas *modify accounts receivable* untuk memodifikasi akun rekening *account receivable*. Aktivitas *modify account payable* untuk memodifikasi akun rekening *account payable*.

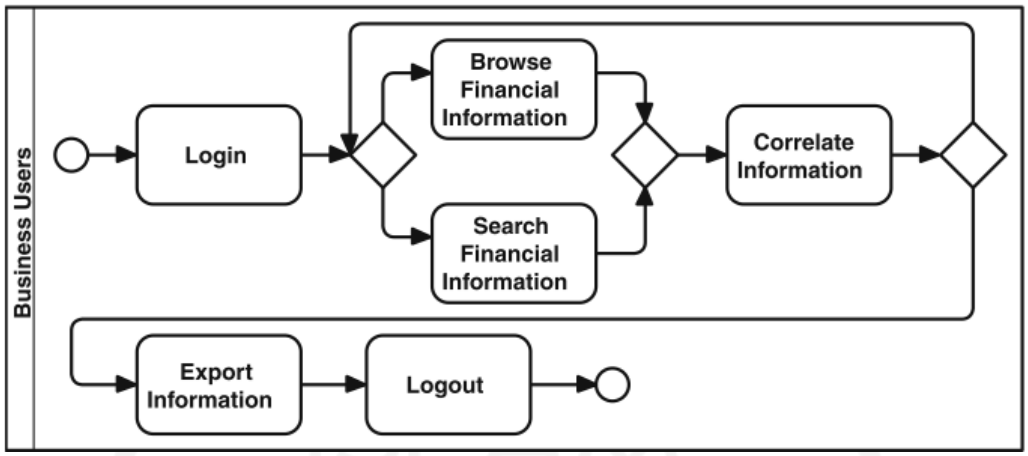
Aktivitas *generate invoice* untuk membuat nota tagihan. Aktivitas *modify cash management* untuk memodifikasi data arus kas. Aktivitas *modify payroll* untuk memodifikasi data penggajian. Aktivitas *modify asset management* untuk memodifikasi data aset. Aktivitas *transfer to ledger* menyatukan seluruh data yang dimodifikasi pada aktivitas sebelumnya pada pusat data *ledger*. Aktivitas *logout* mengakhiri sesi penggunaan aplikasi.



Gambar 2.3 Proses bisnis *input financial information* (Perroud & Inversini, 2013)

c. Proses bisnis *retrieve financial information*

Proses bisnis ini hanya dapat diakses oleh aktor *business user*. Seperti yang terlihat pada Gambar 2.4, Terdapat enam aktivitas dalam proses bisnis ini. Aktivitas *login* memverifikasi keabsahan pengguna. Aktivitas *browse financial information* untuk menjelajah informasi finansial. Aktivitas *search financial information* untuk mencari informasi finansial yang spesifik. Aktivitas *correlate information* mengkorelasikan data finansial dari aktivitas sebelumnya. Aktivitas *export information* mengekspor informasi finansial ke dalam format lain, misal: pdf, epub, csv, dan sebagainya. Aktivitas *logout* mengakhiri sesi penggunaan aplikasi.



Gambar 2.4 Proses bisnis *retrieve financial information* (Perroud & Inversini, 2013)

C. Perspektif Data dan Aplikasi

Perspektif ini mendeksripsikan aplikasi dan data yang terdapat pada perspektif holistic dengan lebih detail. Seperti yang terlihat pada Gambar 2.5, terdapat tiga kelompok aplikasi yang sama seperti pada Gambar 2.1. Berikut penjelasan masing-masing kelompok aplikasi.

d. Kelompok *assets & treasury*

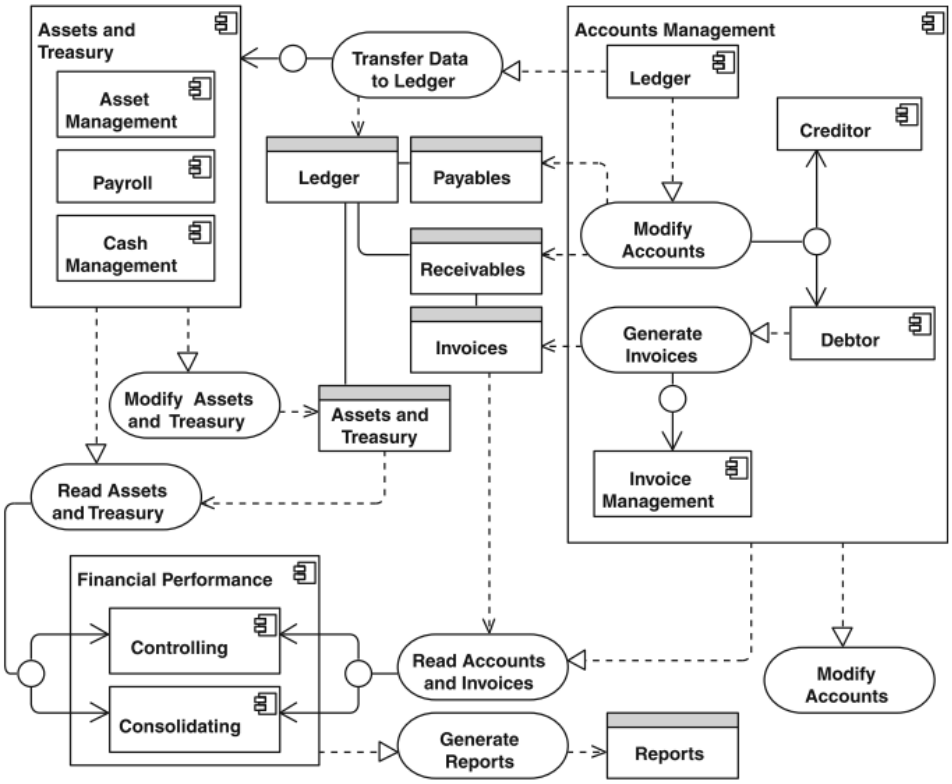
Dalam kelompok ini terdapat tiga aplikasi, yakni: *asset management*, *payroll*, *cash management*. Kelompok ini menyediakan dua layanan, yakni: *modify assets and treasury* dan *read assets and treasury*, serta menggunakan satu layanan, yakni *transfer to ledger*. Layanan yang disediakan aplikasi berinteraksi dengan satu basis data, yakni: *assets and treasury*.

e. Kelompok *accounts management*

Dalam kelompok ini terdapat empat aplikasi, yakni: *ledger*, *creditor*, *debtor*, *invoice management*. Kelompok ini menyediakan empat layanan, yakni: *transfer to ledger, generate invoices, read accounts and invoices*, dan *modify accounts*, serta menggunakan dua layanan, yakni: *modify accounts, dan generate invoices*. Layanan yang disediakan aplikasi berinteraksi dengan empat basis data, yakni: *ledger, payables, receivables, invoices.*

f. Kelompok *financial performance*

Dalam kelompok ini terdapat terdapat dua aplikasi, yakni: *controlling*, dan *consolidating.* Kelompok ini menyediakan satu layanan, yakni: *generate reports*, serta menggunakan dua layanan, yakni: *read assets and treasury*, dan *read accounts and invoices.* Layanan yang disediakan aplikasi berinteraksi dengan satu basis data, yakni: *reports.*



Gambar 2.5 Perspektif data dan aplikasi (Perroud & Inversini, 2013)

2.4 Microservices

Istilah *micro* pada *microservices* sering dikaitkan dengan ukuran yang sebenarnya pada layanan. Namun pada kenyataannya, ukuran yang sebenarnya ini bukanlah indikator yang bagus dalam menilai kualitas dari layanan. Jika ditelaah, konsep *micro* pada *microservices* ini lebih mengacu kepada sebuah layanan yang dapat dikembangkan oleh tim **sekecil** mungkin dan waktu kolaborasi dengan tim lain **sekecil** mungkin serta tanggung jawab **sekecil** mungkin (Richards, 2018).

Selain kecil, karakteristik lain dari *microservices* adalah independen. Independensi antar layanan pada *microservices* ini membuat tiap layanan dapat diubah tanpa mempengaruhi layanan lainnya. Independensi ini juga berarti bahwa tiap layanan tidak dapat mengakses basis data lain secara langsung. Tiap layanan tersebut hanya dapat mengakses basis data melalui layanan yang mengelola basis data tersebut. Selain itu, independensi ini merupakan salah satu kunci penting untuk meningkatkan waktu pengembangan dari sebuah aplikasi, termasuk waktu untuk pemeliharaan dan pengujiannya (Richards, 2018).

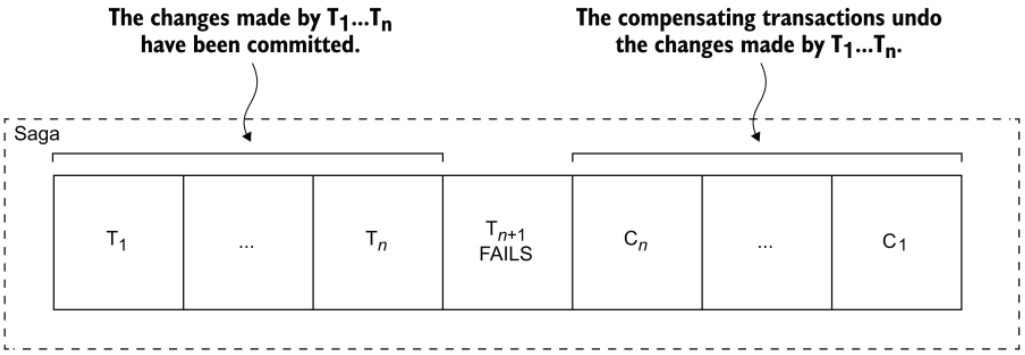
Mengembangkan aplikasi sebagai layanan-layanan kecil yang independen tersebut memberikan beberapa keuntungan dan kerugian. Keuntungan tersebut yakni: fleksibilitas

pemilihan teknologi, resiliensi lebih terhadap *bug*, kemudahan dalam skalabilitas, kemudahan dalam mengembangkan masing-masing layanan. Sedangkan kerugiannya yakni: kerumitan dalam mengelola seluruh layanan, sulit menjaga konsistensi terhadap data, serta risiko kegagalan komunikasi yang cukup besar (Newman, 2015; Richardson, n.d.).

2.4.1 Pola Saga pada *Microservices*

Pada *microservices* sering terjadi kasus dimana satu layanan mengirimkan data yang sama pada dua basis data yang berbeda (transaksi terdistribusi). Kemudian, jika salah satu transaksi itu gagal, maka akan terjadi masalah dimana data menjadi tidak konsisten. Pola saga ini merupakan pola yang dirancang untuk mengatasi masalah transaksi terdistribusi tersebut. Pola saga ini menyusun transaksi yang terdistribusi menjadi transaksi lokal yang berurutan. Jika ada kegagalan transaksi pada urutan tersebut maka transaksi akan di-*rollback* sesuai urutannya secara hitung mundur (Richards, 2018).

Pada Gambar 2.6, T_1 hingga T_n merupakan transaksi lokal berurutan yang berhasil. Namun karena T_{n+1} gagal, maka T_n hingga T_1 harus di-*rollback* secara berurutan menggunakan C_n hingga C_1 .



Gambar 2.6 Ilustrasi algoritma pola saga (Richards, 2018).

2.5 MERN *Stack*

MERN *stack* merupakan istilah dari susunan empat kerangka kerja yang digunakan untuk membangun aplikasi berbasis web (Samikshya, 2020; Widodo, n.d.). Secara berurutan MERN tersusun atas: sMongoDB, ExpressJS, NodeJS, dan ReactJS.

a. MongoDB

MongoDB merupakan basis data terbuka yang berorientasi dokumen. MongoDB memiliki keunggulan pada fleksibilitas struktur data (Aggarwal & Verma, 2018).

b. ExpressJS

ExpressJS merupakan kerangka kerja terbuka berbasis Javascript yang dibangun di atas platform NodeJS. Dalam MERN, ExpressJS digunakan untuk mengembangkan aplikasi bagian *server*. ExpressJS ini memiliki keunggulan pada proses pengembangannya yang sangat mudah dan cepat (Aggarwal & Verma, 2018).

c. NodeJS

NodeJS yang merupakan *platform* terbuka yang digunakan untuk menjalankan kode Javascript di luar *browser*. Hal tersebut membuat Javascript dapat digunakan untuk menjalankan sebuah *server*. NodeJS ini memiliki keunggulan pada eksekusi kodenya yang dapat dijalankan secara *asynchronous* (Samikshya, 2020).

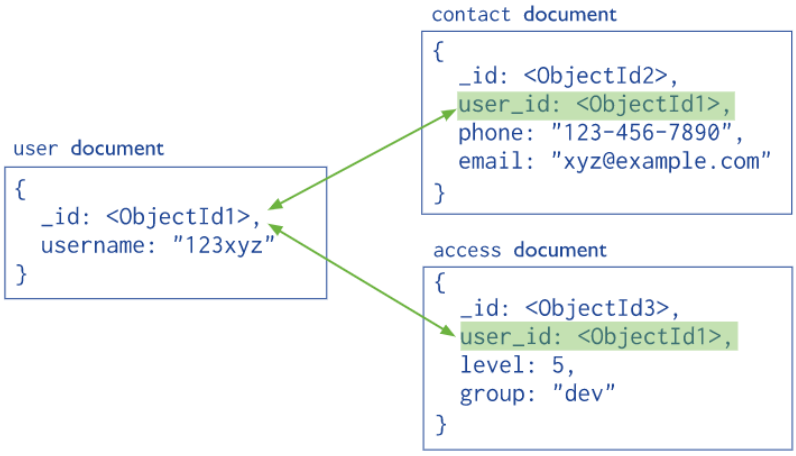
d. ReactJS

ReactJS yang merupakan kerangka kerja terbuka khusus antarmuka yang berbasis bahasa Javascript. ReactJS memiliki keunggulan pada efisiensi kode dalam proses pengembangan aplikasi (Samikshya, 2020).

2.6 Model Skema Basis Data MongoDB

MongoDB ini memiliki dua cara dalam merelasikan antar entitas atau atribut pada entitas. Cara pertama adalah dengan format *sub embedded*. Format *Sub embedded* ini menyimpan data yang berelasi di dalam basis data yang sama, yang contohnya dapat dilihat pada Gambar 2.7. Keuntungan dari format *sub embedded* ini adalah performa *query* yang lebih cepat. Cara kedua adalah dengan format *reference*. Format *reference* ini mirip dengan relasi pada basis data relasional, dimana entitas pertama menyimpan salah satu data pada entitas kedua sebagai referensi untuk dirujuk nantinya, yang contohnya dapat dilihat pada Gambar 2.8. Keuntungan dari format *reference* ini adalah minimnya duplikasi antar data pada basis data. Pada praktiknya, kedua format tersebut dipakai sesuai dengan kebutuhan yang muncul (*Data Model Design — MongoDB Manual*, n.d.).

Gambar 2.7 Contoh data relasi format *sub-embedded* (*Data Model Design — MongoDB Manual*, n.d.)

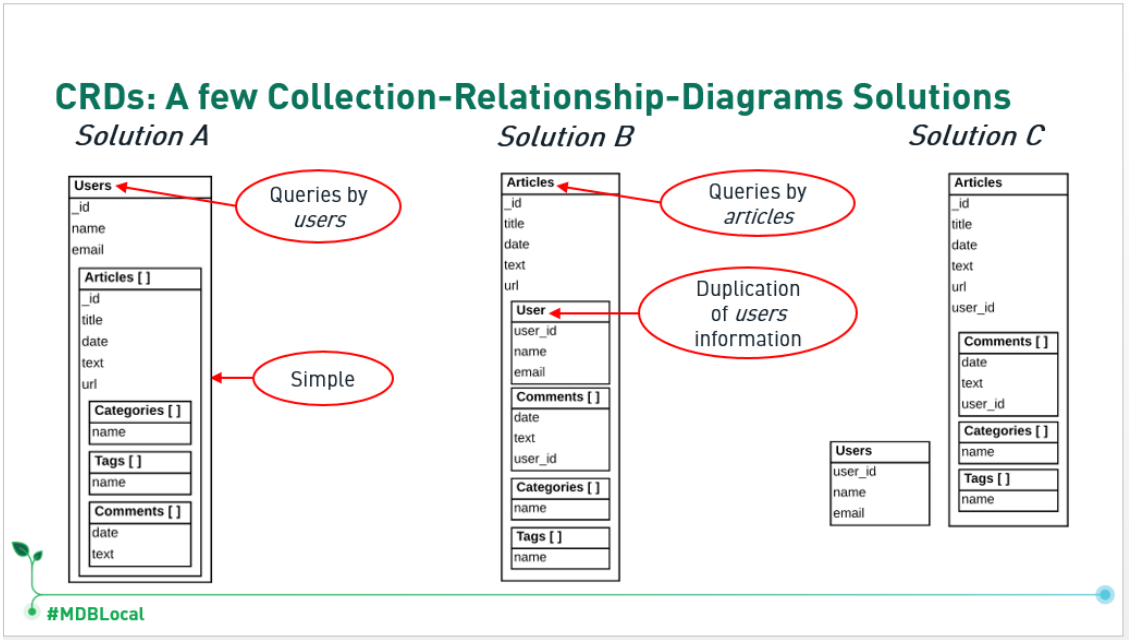


Gambar 2.8 Contoh data relasi format *reference* (*Data Model Design — MongoDB Manual*, n.d.)

Karena MongoDB tersebut memiliki format data yang berbeda dengan data SQL, maka komunitas MongoDB mengusulkan skema model yang lebih cocok untuk diterapkan ke MongoDB (*MongoDB Application Modernization Guide | MongoDB*, n.d.). Gambar 2.9 di bawah merupakan contoh dari skema model yang diusulkan tersebut. Contoh tersebut memodelkan sebuah studi kasus dalam merancang kebutuhan data untuk *blog* pribadi. Yang mana, terdapat tiga kemungkinan solusi dalam bentuk skema yang dapat diterapkan.

Pada Gambar 2.9, solusi A merupakan skema yang menggunakan format *sub embedded*. Dimana entitas *Users* memiliki entitas *article* yang disisipkan secara *sub embedded*. Kemudian, entitas *article* tersebut memiliki tiga entitas yang disisipkan secara *sub* embedded, yakni: *categories*, *tags*, *comments*. Sedangkan, solusi C merupakan skema yang menggunakan format

sub embedded beserta format *reference*. Dimana entitas article memiliki tiga entitas yang disisipkan secara *sub embedded*, yakni: *comments*, *categories*, *tags*. Kemudian, entitas *categories* tersebut memiliki satu entitas yang disisipkan secara *reference* menggunakan atribut *name*.



Gambar 2.9 Contoh diagram relasi antar *collection* untuk skema basis data NoSQL (*MongoDB Application Modernization Guide | MongoDB*, n.d.)

2.7 Peneltian Sebelumnya

Pada penelitian ini dilakukan kajian mengenai empat penelitian serupa pada topik AE, dan lima topik serupa pada topik *microservices.* Pada kajian pustaka ini ditemukan bahwa empat penelitian serupa terkait AE hanya membahas mengenai deskripsi perancangan AE atau pola AE, dan belum ditemukan penelitian mengenai deskripsi implementasi AE atau pola AE. Meski begitu, empat penelitian serupa terkait AE yang dikaji memberikan pengetahuan tersendiri bagi penulis mengenai metodologi perancangan AE serta referensi berbagai pola AE.

Sedangkan, untuk lima penelitian terkait *microservices* belum ditemukan deskripsi implementasi yang didasari oleh pola AE. Namun, lima penelitian serupa terkait *microservices* memberikan pengetahuan tersendiri bagi penulis mengenai proses pemecahan layanan, metodologi pengembangan, serta penggunaan teknologinya.

Berikut detail uraian tiap kajian pada masing-masing pustaka yang dikaji:

a. Pustaka (Ilin et al., 2017)

Pustaka ini membahas contoh perancangan pola AE dengan kasus AE pola teknik pertambangan. Dalam pustaka ini diuraikan seluruh deskripsi terkait teknik pertambangan pada tingkat enterprise, kemudian seluruh deskripsi tersebut dipetakan ke dalam kerangka kerja TOGAF dan menjadi sebuah AE pola teknik pertambangan. Pada pustaka ini, keluaran yang dihasilkan adalah artefak AE pola pertambangan.

b. Pustaka (Riku & Setyohadi, 2017)

Pustaka ini membahas contoh perancangan AE pada perusahaan manufaktur minyak, PT. Bestonindo Central Lestari. Dalam pustaka ini diuraikan langkah-langkah dalam merancang AE menggunakan deksripsi yang sudah tersedia. Pada pustaka ini, keluaran yang dihasilkan adalah artefak AE yang bisa diimplementasikan menjadi sistem TI.

c. Pustaka (Sidiq & Bhakti, 2020)

Pustaka ini membahas contoh perancangan AE pada perusahaan manufaktur saus, PT. Tin Tin. Dalam pustaka ini diuraikan langkah-langkah dalam merancang AE dengan kerangka kerja AE Zachman berdasarkan deksripsi yang sudah tersedia. Disini juga diusulkan untuk menggunakan arsitektur monolitik pada implementasinya. Keluaran yang terdapat pada pustaka ini adalah artefak AE yang bisa diimplementasikan menjadi sistem TI.

d. Pustaka (Perroud & Inversini, 2013)

Pustaka ini membahas banyak tiga belas kasus perancangan pola AE. Pustaka ini berfokus pada proses perancangan pola AE berdasarkan deskripsi yang sudah tersedia. Dalam perancangannya, pustaka ini menggunakan kerangka kerja TOGAF. Keluaran yang dihasilkan pada pustaka ini adalah artefak pola AE.

e. Pustaka (Bucchiarone et al., 2018)

Pustaka ini membahas mengenai migrasi proyek aplikasi berbasis monolitik ke *microservices* pada kasus domain perbankan. Pustaka ini berfokus pada deskripsi proses pemecahan aplikasi menjadi sebuah layanan, serta penggunaan teknologinya.

f. Pustaka (Fan & Ma, 2017)

Pustaka ini membahas mengenai migrasi proyek aplikasi berbasis monolitik ke *microservices* pada kasus forum belajar. Pustaka ini berfokus pada deskripsi metodologi dalam migrasi aplikasi berbasis monolitik ke *microservices.*

g. Pustaka (Hasselbring & Steinacker, 2017)

Pustaka ini membahas mengenai implementasi *microservices* pada kasus *e-commerce*, Otto de. Pustaka ini berfokus pada pembuktian keuntungan serta kerugian *microservices*.

h. Pustaka (Suryotrisongko et al., 2017)

Pustaka ini membahas mengenai implementasi *microservices* sebagai solusi dari masalah pada sistem *e-government*. Pustaka ini berfokus pada deskripsi desain arsitektur *microservices* berdasarkan analisi kebutuhan yang sudah dilakukan sebelumnya.

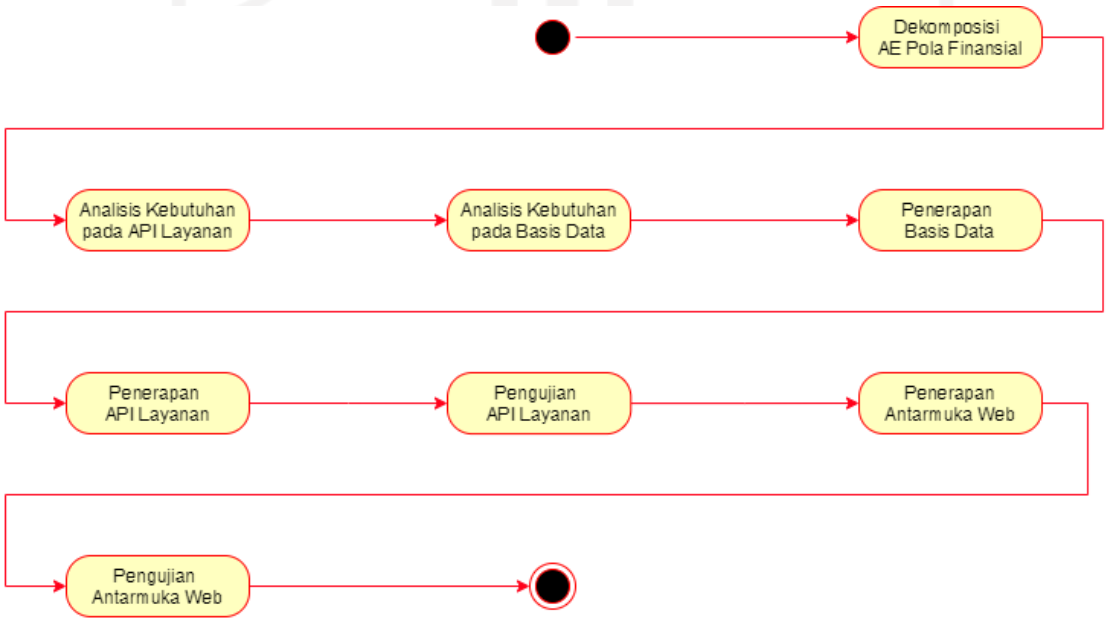
i. Pustaka (Wu et al., 2019)

Pustaka ini membahas mengenai implementasi pada *business to business e-commerce.* Pustaka ini berfokus pada deskripsi proses implementasinya yang berdasarkan kebutuhan yang sudah tersedia sebelumnya.

BAB III
Metodologi Penelitian

3.1 Diagram Metodologi Penelitian

Seperti yang terlihat pada Gambar 3.1, terdapat 8 aktivitas utama yang dilakukan pada penelitian ini. Penelitian ini dimulai dengan mendekomposisi AE pola finansial dengan strategi yang diusulkan dari buku (Richards, 2018), yakni DDD (*Domain Driven Design*). Selanjutnya, dilakukan analisis kebutuhan pada API layanan. Pada tahap ini, terdapat beberapa kebutuhan yang harus diidentifikasi agar seluruh layanan dapat berjalan lancar ketika diterapkan. Kemudian, dilakukan analisis kebutuhan pada basis data. Pada tahap ini, kebutuhan data pada tiap basis data dianalisis lalu dimodelkan dalam sebuah skema rancangan. Selanjutnya, dilakukan penerapan basis data lalu API layanan. Penerapan tersebut dilakukan berdasarkan analisis kebutuhan yang sudah dilakukan sebelumnya. Setelah itu, hasil penerapan API layanan diuji pada tahap pengujian untuk mengukur diukur kualitasnya. Kemudian, dilakukan penerapan antarmuka web serta pengujian pada antarmuka tersebut. Antarmuka web ini diterapkan dalam rangka mempermudah pengaksesan data yang diolah oleh API layanan.

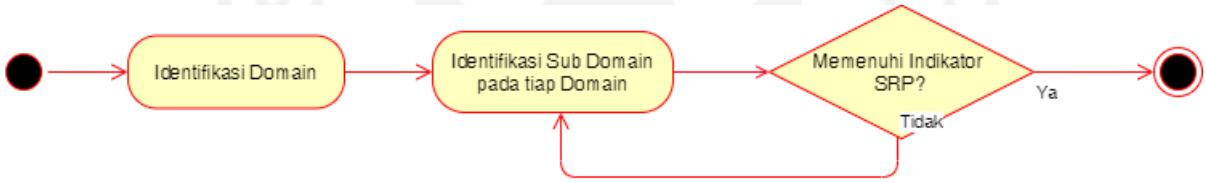


Gambar 3.1 *Activity Diagram* metodologi penelitian

3.2 Dekomposisi Arsitektur Enterprise Pola Finansial

Dari beberapa strategi dekomposisi yang tersedia, penelitian ini menggunakan strategi DDD. DDD ini pada dasarnya merupakan strategi yang diusulkan oleh Eric Evans di tahun 2003 untuk membangun aplikasi kompleks dengan pemrograman berbasis objek. Meski demikian, DDD juga dapat diterapkan ketika membangun aplikasi berbasis *microservices*, khususnya di tahap dekomposisi (Richards, 2018).

Seperti yang terlihat pada Gambar 3.2, aktivitas mendekomposisi pola AE finansial dimulai dengan mengidentifikasi domain yang terdapat dalam AE pola finansial. Domain ini merupakan semacam bidang keahlian yang terdapat pada keseluruhan AE pola finansial. Kemudian, dilakukan identifikasi sub domain pada tiap domain yang teridentifikasi. Sub domain ini merupakan sub bidang keahlian yang terdapat pada suatu domain. Dan selanjutnya, dilakukan proses pengujian dari hasil dekomposisi menggunakan indikator Single Responsibility Principle (SRP).



Gambar 3.2 *Activity Diagram* metodologi dekomposisi

Dalam mendefinisikan domain dan sub domain diperlukan pemahaman yang kuat mengenai proses bisnis, dan struktur organisasi dari enterprise itu sendiri. Biasanya domain atau sub domain ini berkaitan dengan satu permasalahan tertentu (Richards, 2018). Sebagai contoh, pada sebuah enterprise terdapat domain manufaktur, dan kantin. Kemudian, pada domain kantin terdapat sub domain order, dan dapur. Pada contoh tersebut, domain manufaktur berkaitan dengan masalah pengolahan bahan baku dan domain kantin berkaitan dengan masalah makanan karyawan. Selanjutnya, sub domain order pada kantin berkaitan dengan masalah pengelolaan pesanan makanan karyawan dalam lingkup kantin dan sub domain dapur pada kantin berkaitan dengan masalah pengolahan bahan baku menjadi makanan dalam lingkup kantin.

Dalam mendefinisikan domain dan sub domain permasalahan yang sering muncul adalah pengistilahan dari suatu konsep. Suatu konsep yang sama terkadang memiliki istilah yang berbeda pada antar domain. Dan suatu konsep berbeda juga dapat memiliki istilah yang sama

antar domain. Yang mana, hal tersebut akan berakibat pada kebingungan dan kesalahan komunikasi (Richards, 2018).

SRP pada dasarnya merupakan strategi yang diusulkan oleh Robert C. Martin pada tahun 1995 untuk merancang baris kode pada aplikasi dengan prinsip o*bject oriented programming.* Jika diterapkan pada dekomposisi *microservice,* SRP mengharuskan setiap layanan untuk memiliki hanya satu tanggung jawab. Jika layanan memiliki lebih dari 1 tanggung jawab maka kemungkinan layanan tersebut masih terlalu besar dan harus dipecah lagi Hal tersebut selaras dengan prinsip *microservices,* yakni: kecil dan independen.

Dalam tahap dekomposisi ini, keluaran yang dihasilkan adalah model sub domain dari AE pola finansial. Yang mana, model tersebut nantinya digunakan sebagai panduan untuk mengidentifikasi API layanan di tahap selanjutnya.

3.3 Analisis Kebutuhan pada API layanan

Informasi terkait kebutuhan pada API layanan pada *microservices* ini merupakan informasi yang diusulkan pada buku (Richards, 2018). Beberapa informasi yang perlu digali pada tahap ini, yakni:

a. Identifikasi API layanan,

Analisis kebutuhan ini dimulai dengan mengidentifikasi API layanan yang mau diterapkan berdasarkan model sub domain yang diperoleh dari tahap sebelumnya. Identifikasi ini dilakukan karena bisa jadi ada kendala yang membuat beberapa sub domain pada AE pola finansial tidak dapat diterapkan. Kendala tersebut bisa jadi berupa biaya, waktu, atau semacamnya.

b. Identifikasi fungsi pada API layanan

Kemudian, dilakukan proses identifikasi fungsi pada tiap API layanan yang mau diterapkan. Proses ini dilakukan untuk memberikan gambaran mengenai apa saja yang dapat dilakukan oleh setiap API layanan. Pada AE pola finansial, proses ini menggunakan diagram proses bisnis sebagai referensi dalam mengidentifikasi fungsi tersebut.

c. Identifikasi proses kolaborasi antar API layanan

Selanjutnya, dilakukan identifikasi proses kolaborasi antar API layanan dalam rangka menjalankan proses bisnisnya. Pada tahapan sebelumnya, proses bisnis didekomposisi menjadi beberapa sub domain yang tiap aktivitasnya diwakili oleh fungsi pada API layanan. Oleh karena itu, fungsi-fungsi tersebut perlu dirangkai kembali agar proses bisnis tersebut dapat

berjalan. Yang mana, fungsi tersebut dirangkai menggunakan proses kolaborasi antar API layanan.

d. Identifikasi format komunikasi pada API layanan

Setelah itu, dilakukan proses identifikasi terhadap format komunikasi yang digunakan pada API layanan. Komunikasi ini merupakan sebuah cara bagi API layanan agar dapat berkolaborasi dalam rangka menjalankan proses bisnisnya.

e. Identifikasi kebutuhan pola saga pada API layanan

Dalam *microservices* terdapat masalah konsistensi data yang kerap kali terjadi dalam proses transaksi data yang terdistribusi antara API layanan dan banyak basis data. Untuk menyiasati masalah itu, maka diterapkan pola saga pada API layanan yang memiliki masalah dengan konsistensi data tersebut. Adapun detail dari pola saga tersebut sudah dijelaskan pada bab 2.4.1

3.4 Analisis Kebutuhan Basis Data

Setelah selesai menganalisis kebutuhan API layanan, selanjutnya dilakukan analisis kebutuhan basis data. Informasi yang digali pada tahap ini yakni: identifikasi kebutuhan basis data pada tiap API layanan, identifikasi entitas, atributnya, serta relasi antar entitas dan atribut jika ada. Kemudian, hasil identifikasi tersebut disusun kedalam skema rancangan basis data.

Pada kasus AE pola finansial ini, Basis data diidentifikasi berdasarkan kebutuhan dari API layanan. Biasanya tiap layanan memiliki satu basis data tersendiri yang berdiri secara independen, namun ada juga kasus dimana satu layanan memiliki dua basis data, atau dua layanan memiliki satu basis data yang sama, atau semacanya.

Pada penelitian ini, sistem manajemen basis data yang digunakan adalah MongoDB. MongoDB ini merupakan sistem manajemen basis data yang berbasis NoSQL. Pada basis data NoSQL ini penulis belum menemukan aturan resmi dalam memodelkan skema datanya. Namun, sebuah komunitas mengusulkan sebuah diagram pemodelan yang dapat digunakan pada tahap analisis kebutuhan. Diagram ini disebut sebagai diagram relasi antar *collection,* yang mana contohnya sudah dipaparkan pada 2.6 (*Data Model Design — MongoDB Manual*, n.d.; *MongoDB Application Modernization Guide | MongoDB*, n.d.).

3.5 Penerapan Basis Data

Setelah proses analisis kebutuhan basis data selesai, selanjutnya dilakukan proses penerapan basis data tersebut berdasarkan hasil analisis kebutuhannya. Pada penelitian ini,

basis data akan diterapkan pada MongoDB Atlas. MongoDB Atlas sendiri merupakan layanan basis data milik MongoDB yang berbasis *cloud*. Keluaran yang dihasilkan dari tahap ini adalah basis data yang siap digunakan oleh API layanan.

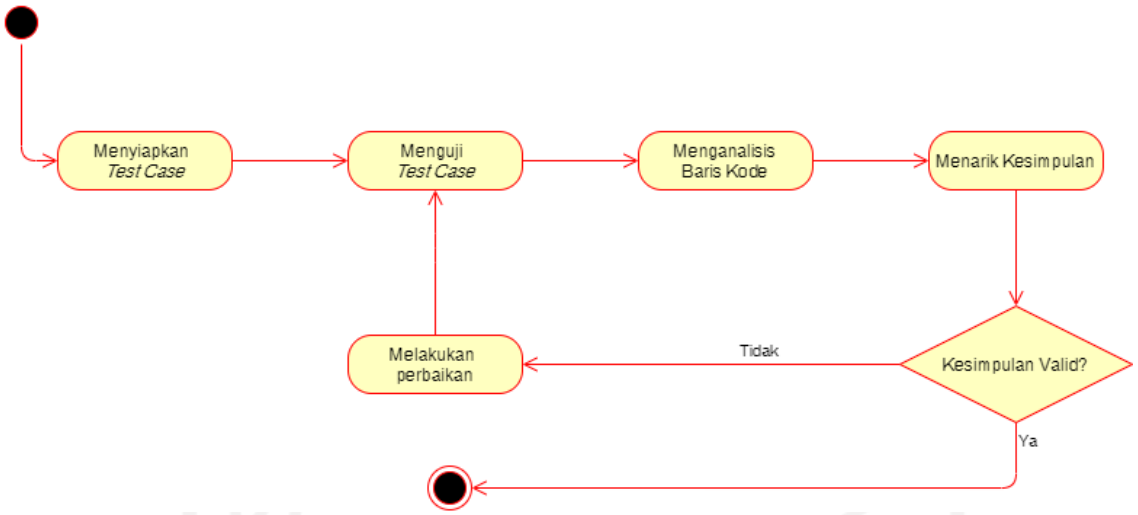
3.6 Penerapan API Layanan

Setelah proses penerapan basis data selesai dilakukan, selanjutnya dilakukan proses penerapan API layanan berdasarkan analisis kebutuhannya. Pada penelitian ini, API layanan akan diterapkan menggunakan teknologi ExpressJS, dan NodeJS. ExpressJS ini merupakan sebuah paket *library* yang berjalan di atas *platform* NodeJS. ExpressJS ini memuat fitur untuk mengembangkan API layanan dengan cepat (*GitHub - Expressjs/Express: Fast, Unopinionated, Minimalist Web Framework for Node.*, n.d.). Keluaran yang dihasilkan pada tahap ini adalah API layanan yang siap diuji kualitasnya.

3.7 Pengujian API Layanan

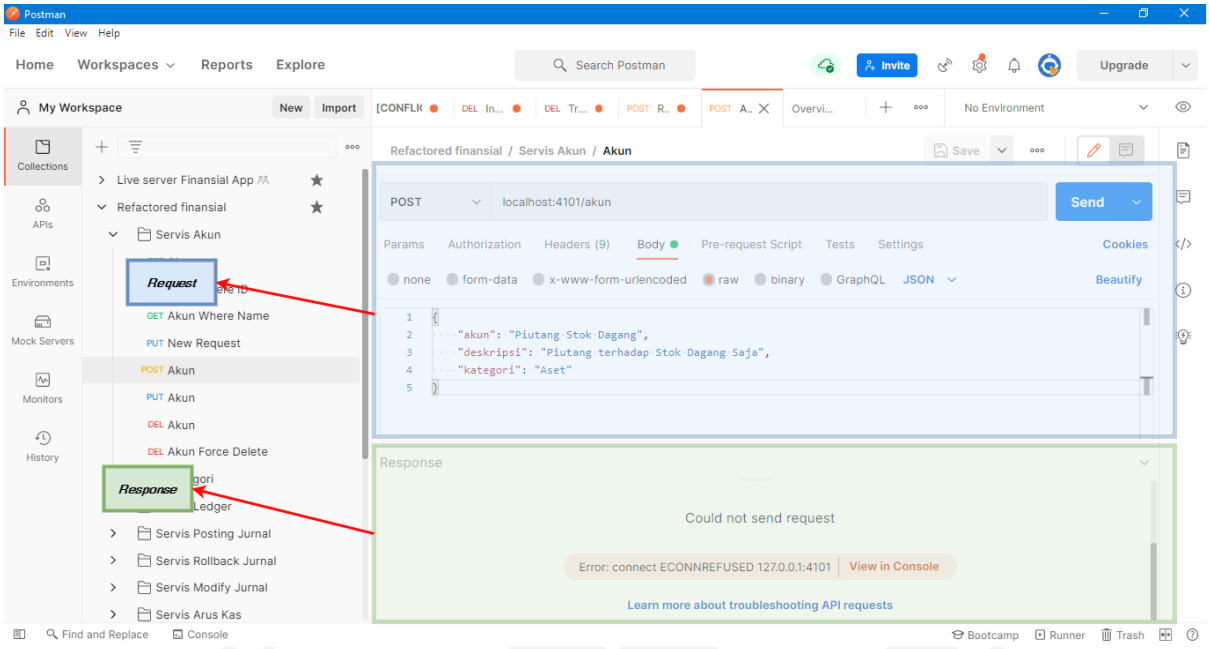
Setelah API layanan berhasil diterapkan, selanjutnya dilakukan proses pengujian API layanan. Pengujian ini dilakukan pada dua lingkup, yakni: lingkup *unit*, dan lingkup *integration.* Lingkup *unit* merupakan pengujian yang menguji baris kode dalam lingkup kecil, misal: *class*, fungsi, dan sebagainya. Sedangkan, lingkup *integration* merupakan pengujian yang menguji baris kode pada dua atau lebih *unit* yang saling terintegrasi (Nidhra, 2012). Pada kasus AE pola finansial, lingkup pengujian API layanan akan menyesuaikan kebutuhan dari API layanan itu sendiri. Contohnya, jika API layanan memiliki fungsi yang terintegrasi, maka pengujian lingkup *integration* dilakukan, dan begitu juga sebaliknya

Selain itu, proses pengujian ini dilakukan secara *white box*. Secara definisi, *white box* ini merupakan sebuah metode pengujian yang melakukan pengujian sembari menganalisis baris kodenya (Nidhra, 2012). Seperti yang terlihat pada Gambar 3.3, tahap yang dilakukan secara berurutan pada pengujian ini adalah menyusun *test case*, menguji *test case*, menganalisis baris kode penyusunnya, lalu menarik kesimpulan. Jika kesimpulan valid, maka pengujian selesai. Namun jika tidak valid, maka API layanan akan diperbaiki lalu diuji ulang.



Gambar 3.3 *Activity diagram* metodologi pengujian

Dalam menguji *test case*-nya digunakan aplikasi yang bernama Postman, yang contohnya dapat dilihat pada Gambar 3.4. Postman ini merupakan aplikasi yang memiliki berbagai fitur untuk mempermudah pengembangan API layanan, termasuk proses pengujian (*GitHub - Postmanlabs/Postman-Docs: Documentation for Postman, a Collaboration Platform for API Development. Available for Mac, Windows and Linux.*, n.d.). Pengujian lewat Postman ini dilakukan dengan cara mengirimkan sebuah *request* yang berisi parameter sesuai spesifikasi pada API layanannya. Kemudian setelah mengirimkan request, akan muncul sebuah keluaran berupa *response*-nya. Seperti yang terlihat pada Gambar 3.4, penulis coba mengirimkan *request* dengan *method* POST, dan parameter berupa data *raw* yang berformat JSON ke *endpoint* "localhost:4101/akun" yang kemudian mendapat *response "error connection refused".*

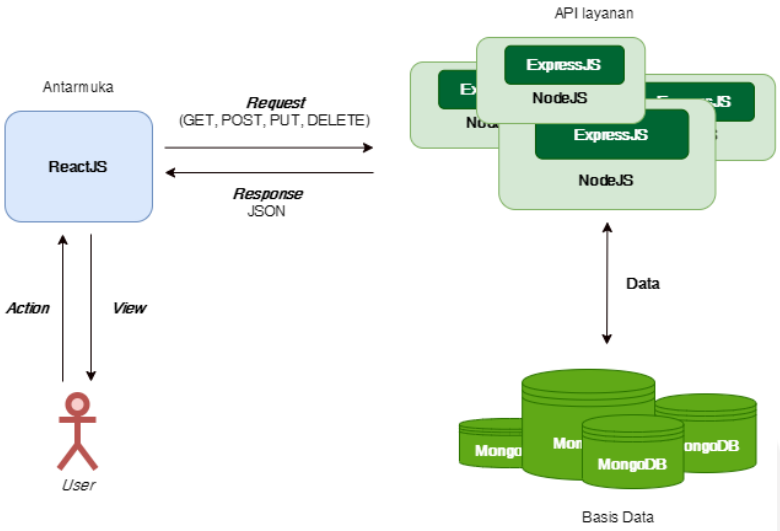


Gambar 3.4 Contoh penggunaan Postman

3.8 Penerapan Antarmuka Web

Setelah seluruh API layanan selesai diuji, selanjutnya dilakukan proses penerapan antarmuka web. Pada penelitian ini, antarmuka akan diterapkan menggunakan teknologi ReactJS. ReactJS ini merupakan paket *library* yang menyediakan kerangka kerja dalam mengembangkan antarmuka (*GitHub - Facebook/React: A Declarative, Efficient, and Flexible JavaScript Library for Building User Interfaces.*, n.d.).

Seperti yang terlihat pada Gambar 3.5, antarmuka web ini merupakan aplikasi yang digunakan langsung oleh pengguna. Berbeda dengan API layanan dan basis data, antarmuka ini dikembangkan menggunakan arsitektur monolitik. Yang mana, hanya terdapat satu aplikasi besar pada antarmuka. Selain itu, karena antarmuka ini dikembangkan secara terpisah dari API layanan, maka diperlukan paket tambahan yang dapat membantu antarmuka untuk berkomunikasi dengan banyak API layanan, yakni: paket Axios.



Gambar 3.5 Arsitektur MERN *stack*

3.9 Pengujian Performa Antarmuka Web

Setelah antarmuka web berhasil diterapkan, selanjutnya dilakukan proses pengujian antarmuka. Karena antarmuka tersebut dibangun dalam rangka mendukung pengaksesan API layanan, maka antarmuka tidak memiliki fungsi yang berdiri sendiri. Atau dengan kata lain, seluruh fungsi yang dibuat pada antarmuka tersebut merupakan fungsi yang terintegrasi dengan API layanan. Oleh karena itu, antarmuka tidak diuji pada lingkup *unit*. Antarmuka ini diuji pada lingkup *integration* saja.

Sama seperti API layanan, antarmuka ini diuji secara *white box*. Yang mana, tahapan pengujiannya sama seperti yang tertera pada Gambar 3.3. Namun, pada antarmuka ini *test case* langsung diuji dengan melakukan interaksi langsung dengan aplikasi yang dijalankan pada web browser.

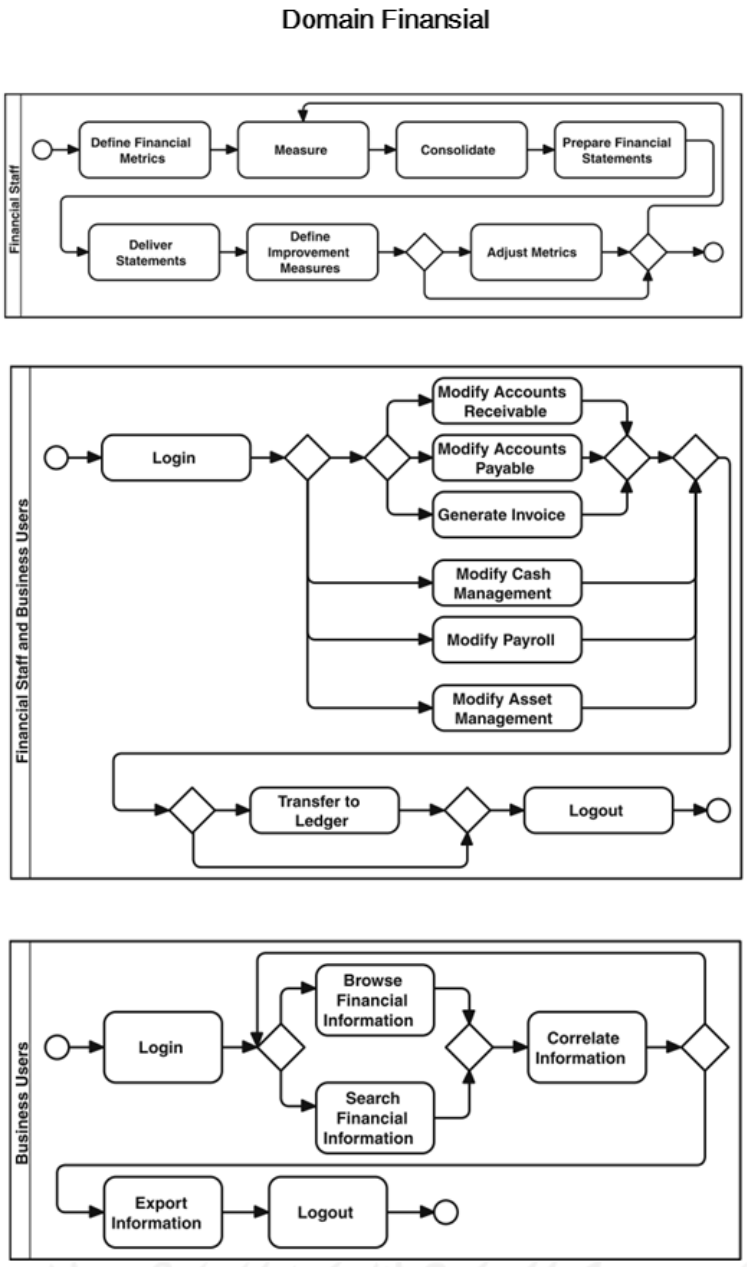
BAB IV
Hasil dan Pembahasan

4.1 Dekomposisi Arsitektur Enterprise Pola Finansial

Proses pertama yang dilakukan untuk menerapkan AE pola finansial pada *microservices* adalah dekomposisi. Seperti yang dijelaskan pada bab 3.2, tahap ini memiliki 3 tahap yang dilakukan, yakni: identifikasi domain, identifikasi sub domain, dan pengujian hasil identifikasi sub domain.

4.1.1 Identifikasi Domain

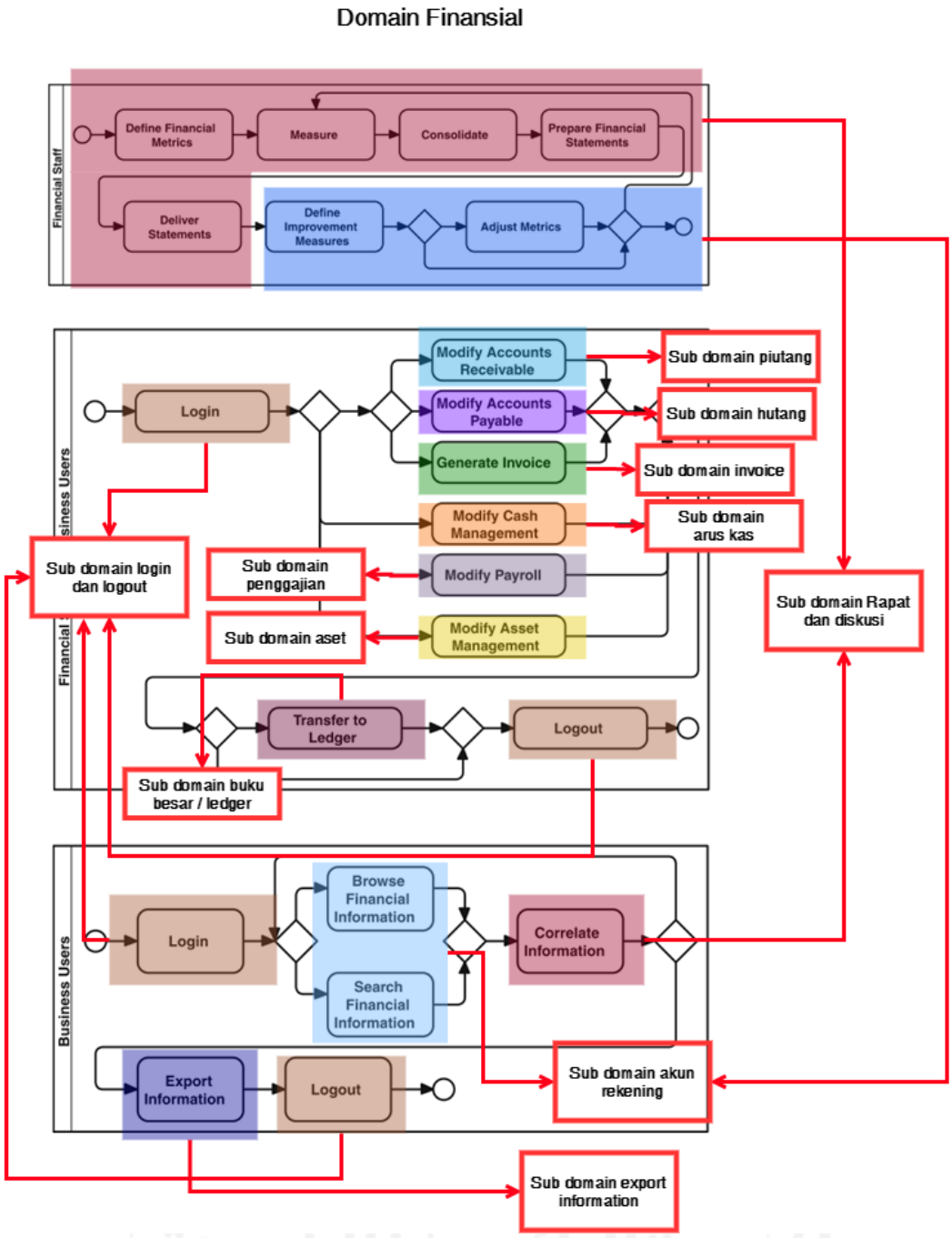
Tahapan pertama dalam mendekomposisi AE pola finansial adalah mengidentifikasi domain. Seperti yang terlihat pada Gambar 4.1, Pada AE pola finansial terdapat tiga proses bisnis, yakni: *analyse and plan, input financial information, retrieve financial information.* Adapun masing-masing aktivitas dari tiap proses bisnis tersebut sudah dijelaskan secara detail pada bab 2.3.1. Jika diamati, ketiga proses bisnis tersebut semuanya berkaitan dengan pengelolaan finansial. Maka dari itu, hanya terdapat satu domain pada kasus AE pola finansial ini, yakni: domain finansial.



Gambar 4.1 Proses bisnis pada AE pola finansial (Perroud & Inversini, 2013)

4.1.2 Identifikasi Sub Domain

Setelah berhasil mengidentifikasi domain, selanjutnya dilakukan proses identifikasi sub domain. Sub domain ini merupakan semacam bidang keahlian yang terdapat pada tiap domain. Dalam mengidentifikasi sub domain ini, diperlukan pemahaman yang kuat terkait proses bisnis. Cara mengidentifikasi sub domain ini adalah dengan menganalisis proses bisnis berdasarkan struktur organisasi, dan sub bidang keahliannya (Richards, 2018). Seperti yang terlihat pada Gambar 4.2, dalam kasus AE pola finansial terdapat sebelas sub domain yang dapat diidentifikasi.



Gambar 4.2 Hasil dekomposisi sub domain dari domain finansial

Pada Gambar 4.2, aktivitas: *modify account receivable*, *modify account payable*, *modify cash management*, *modify payroll*, *modify asset management*, dan *generate invoice* merupakan aktivitas yang sama-sama menginput data transaksi terkait akun rekening. Namun, ketujuh aktivitas tersebut memiliki kebutuhan data yang unik serta memerlukan bidang keahlian yang berbeda dalam mengolah datanya. Maka dari itu, masing-masing dari tujuh aktivitas tersebut dijadikan sub domain tersendiri.

pada Gambar 4.2, aktivitas *login,* dan *logout* merupakan aktivitas pada bidang verifikasi pengguna. Maka dari itu, dua aktivitas *login* dan dua aktivitas *logout* ini dikelompokkan menjadi sub domain *login* dan *logout*. Pada Gambar 4.2, aktivitas *export information* merupakan aktivitas pada bidang pengeksporan informasi. Maka dari itu, aktivitas ini dikelompokkan menjadi sub domain *export information*. Pada Gambar 4.2, aktivitas: *define improvement measure*, *adjust metrics*, *browse financial information*, dan *search financial information* merupakan aktivitas pada bidang pengelolaan akun rekening. Maka dari itu, keempat aktivitas ini dikelompokkan menjadi sub domain akun rekening.

Pada Gambar 4.2, Aktivitas *transfer to ledger* merupakan aktivitas pada bidang pengelolaan *ledger* / buku besar. Maka dari itu, aktivitas ini dikelompokkan menjadi sub domain *ledger*. Pada Gambar 4.2, Aktivitas: *define financial metrics*, *measure*, *consolidate*, *prepare financial statement*, *deliver statement*, dan *correlate information* merupakan aktivitas pada bidang rapat dan diskusi. Maka dari itu, aktivitas tersebut dikelompokkan menjadi sub domain rapat dan diskusi.

4.1.3 Pengujian Hasil Identifikasi Sub Domain

Selanjutnya, dilakukan pengujian terhadap sub domain yang telah diidentifikasi. Seperti yang sudah dijelaskan pada bab 3.2, Pengujian ini dilakukan menggunakan indikator SRP. Hasil pengujian terhadap sub domain pada AE pola finansial ini dapat dilihat pada Tabel 4.1.

Tabel 4.1 Hasil prngujian SRP

No.	Sub Domain	Tanggung Jawab	Kesimpulan
1	*Login* dan *logout*	Verifikasi pengguna	Valid
2	Penggajian	Mengelola data penggajian	Valid
3	Aset	Mengelola data aset	Valid
4	*Ledger*	Mencatat seluruh perubahan pada akun rekening	Valid
5	Akun rekening	Mengelola data yang terkait akun rekening	Valid
6	*Export information*	Ekspor informasi yang tersedia pada aplikasi ke dalam format lain, misal: pdf	Valid
7	Piutang	Mengelola data piutang	Valid
8	Hutang	Mengelola data hutang	Valid
9	*Invoice*	Mengelola data invoice / penagihan	Valid

No.	Sub Domain	Tanggung Jawab	Kesimpulan
10	Arus kas	Mengelola data transaksi kas	Valid
11	Rapat dan diskusi	Mengelola berjalannya rapat dan diskusi	Valid

4.2 Analisis Kebutuhan pada API Layanan

Setelah berhasil mendekomposisi, selanjutnya dilakukan analisis kebutuhan pada API layanan yang diterapkan. Seperti yang dijelaskan pada bab 3.3, tahap ini memiliki lima proses yang dilakukan, yakni: identifikasi API layanan, identifikasi fungsi pada API layanan, identifikasi proses kolaborasi antar API layanan. Identifikasi format komunikasi, dan analisis kebutuhan pola saga pada API layanan.

4.2.1 Identifikasi API Layanan

Proses pertama yang dilakukan dalam menganalisis kebutuhan pada API layanan adalah mengidentifikasi terlebih dahulu API layanan mana yang akan diterapkan. Seperti yang terlihat pada Tabel 4.2, AE pola finansial memiliki sebelas sub domain, namun dari sebelas sub domain tersebut hanya delapan sub domain saja yang diterapkan sebagai API layanan. Meski tidak diterapkan, aktivitas pada sub domain tersebut tetap dilakukan, namun dilakukan di luar aplikasi bukan di dalam aplikasi.

Sub domain *login* dan *logout* tidak diterapkan, namun sebagai gantinya diasumsikan bahwa pengguna diverifikasi secara manual sebelum menggunakan komputer yang terdapat program finansial ini. Sub domain *export information* tidak diterapkan, namun sebagai gantinya pengguna dapat menggunakan fitur bawaan *browser* untuk mengekspor sebagai pdf. Sub domain rapat dan diskusi tidak diterapkan, namun sebagai gantinya pengguna harus melakukan rapat secara manual atau menggunakan aplikasi pihak ketiga seperti Zoom / Google Meet.

Meski begitu, sub domain yang belum diterapkan ini dapat diterapkan di kemudian hari tanpa banyak mengganggu layanan yang sudah berjalan. Hal ini dapat dilakukan karena keunggulan *microservices* yang tinggi pada sisi skalabilitasnya (Newman, 2015; Richards, 2018).

Tabel 4.2 Hasil identifikasi layanan

No.	Sub domain	Status
1	Login dan logout	**Tidak** diterapkan sebagai API layanan dalam aplikasi
2	Penggajian	Diterapkan sebagai API layanan dalam aplikasi

No.	Sub domain	Status
3	Aset	Diterapkan sebagai API layanan dalam aplikasi
4	*Ledger*	Diterapkan sebagai API layanan dalam aplikasi
5	Akun rekening	Diterapkan sebagai API layanan dalam aplikasi
6	*Export information*	**Tidak** diterapkan sebagai API layanan dalam aplikasi
7	Piutang	Diterapkan sebagai API layanan dalam aplikasi
8	Hutang	Diterapkan sebagai API layanan dalam aplikasi
9	*Invoice*	Diterapkan sebagai API layanan dalam aplikasi
10	Arus kas	Diterapkan sebagai API layanan dalam aplikasi
11	Rapat dan diskusi	**Tidak** diterapkan sebagai API layanan dalam aplikasi

4.2.2 Identifikasi Fungsi pada API Layanan

Proses kedua yang dilakukan dalam menganalisis kebutuhan API layanan adalah mengidentifikasi fungsi yang akan dimuat pada masing-masing API layanan tersebut. Hasil identifikasi fungsi ini dapat dilihat pada Tabel 4.3.

Tabel 4.3 Hasil identifikasi fungsi pada tiap API layanan

No.	layanan	Identifikasi fungsi	Kegunaan Fungsi
1	Penggajian	CreatePenggajian ()	Menambah transaksi penggajian baru
		ReadPenggajian ()	Melihat seluruh transaksi penggajian
		DeletePenggajian ()	Menghapus transaksi penggajian
		ReadKaryawan ()	Melihat seluruh karyawan
2	Aset	CreateAset ()	Menambah transaksi pembelian aset baru
		ReadAset ()	Melihat seluruh data transaksi pembelian aset
		DeleteAset ()	Menghapus transaksi pembelian aset
3	*Ledger*	GetLedger ()	Melihat seluruh data *ledger*
		PostLedger ()	Menambah data *ledger* baru
		EditLedger ()	Mengedit data *ledger*
4	Akun rekening	GetAkun ()	Melihat seluruh data akun
		GetAkunById ()	Melihat data akun dengan id tertentu

No.	layanan	Identifikasi fungsi	Kegunaan Fungsi
		GetAkunByName ()	Melihat data akun dengan nama tertentu
		PostNewAkun ()	Menambah data akun baru
		EditAkun ()	Mengubah data akun
		DeleteAkun()	Menghapus data akun
		ChangeTargetNominal ()	Mengubah target nominal bulanan tiap akun rekening
		GetKategoriAkun ()	Melihat seluruh data kategori akun
		GetLaporanLabaRugi ()	Mengoperasikan nominal akun untuk mencari laba rugi penjualan
5	Piutang	CreatePiutang ()	Menambah piutang baru
		ReadPiutang ()	Melihat data seluruh piutang
		DeletePiutang ()	Menghapus data piutang
		LunaskanCicilanPiutang ()	Melunaskan cicilan piutang
6	Hutang	CreateHutang ()	Menambah hutang baru
		ReadHutang ()	Melihat data seluruh hutang
		DeleteHutang ()	Menghapus data hutang
		LunaskanCicilanHutang ()	Melunaskan cicilan hutang
7	*Invoice*	Create*Invoice* ()	Menambah data *invoice* baru
		Read*Invoice* ()	Melihat seluruh data *invoice*
		Delete*Invoice* ()	Menghapus data *invoice*
8	Arus kas	CreateArusKas ()	Menambah data transaksi kas baru
		ReadArusKas ()	Melihat seluruh data transaksi kas
		DeleteArusKas ()	Menghapus data transaksi kas

4.2.3 Identifikasi Proses Kolaborasi Antar API Layanan

Proses ketiga yang dilakukan dalam menganalisis kebutuhan pada API layanan adalah mengidentifikasi kolaborasi antar fungsi pada seluruh API layanan. Kolaborasi ini disusun berdasarkan aliran proses bisnis yang sebenarnya. Dengan kata lain, proses ini merupakan

proses untuk menyatukan kembali proses bisnis yang sudah terpecah. Hasil indetifikasi dari proses kolaborasi ini dapat dilihat pada Tabel 4.4

Tabel 4.4 Hasil identifikasi kolaborator per fungsi

No.	layanan	Identifikasi fungsi	Kolaborator
1	Penggajian	CreatePenggajian ()	PostLedger ()
			EditAkun ()
		ReadPenggajian ()	-
		DeletePenggajian ()	EditLedger ()
			PostLedger ()
			EditAkun ()
		ReadKaryawan ()	-
2	Aset	CreateAset ()	PostLedger ()
			EditAkun ()
		ReadAset ()	-
		DeleteAset ()	EditLedger ()
			PostLedger ()
			EditAkun ()
3	*Ledger*	GetLedger ()	-
		PostLedger ()	-
		EditLedger ()	-
4	Akun rekening	GetAkun ()	-
		GetAkunById ()	-
		GetAkunByName ()	-
		PostNewAkun ()	-
		EditAkun ()	-
		DeleteAkun()	-
		ChangeTargetNominal ()	-
		GetKategoriAkun ()	-
		GetLaporanLabaRugi ()	-

No.	layanan	Identifikasi fungsi	Kolaborator
5	Piutang	CreatePiutang ()	PostLedger ()
			EditAkun ()
		ReadPiutang ()	-
		DeletePiutang ()	EditLedger ()
			PostLedger ()
			EditAkun ()
		LunaskanCicilanPiutang ()	PostLedger ()
			EditAkun ()
6	Hutang	CreateHutang ()	PostLedger ()
			EditAkun ()
		ReadHutang ()	-
		DeleteHutang ()	EditLedger ()
			PostLedger ()
			EditAkun ()
		LunaskanCicilanHutang ()	PostLedger ()
			EditAkun ()
7	*Invoice*	Create*Invoice* ()	-
		Read*Invoice* ()	-
		Delete*Invoice* ()	-
8	Arus kas	CreateArusKas ()	PostLedger ()
			EditAkun ()
		ReadArusKas ()	-
		DeleteArusKas ()	EditLedger ()
			PostLedger ()
			EditAkun ()

Seperti yang terlihat pada Tabel 4.1, fungsi: CreatePenggajian (), CreateAset (), CreatePiutang (), CreateHutang(), CreateArusKas () berkolaborasi dengan dua fungsi yang sama, yakni PostLedger () pada layanan *ledger*, dan fungsi EditAkun () pada layanan akun.

Kedua fungsi tersebut merupakan fungsi yang digunakan untuk mengirimkan data nominal perubahan akun ketika terjadi transaksi baru kepada *ledger* dan akun.

Sedangkan, untuk fungsi: DeletePenggajian (), DeleteAset (), DeletePiutang (), DeleteHutang(), DeleteArusKas () berkolaborasi dengan tiga fungsi yang sama, yakni: EditLedger (), dan PostLedger (), dan EditAkun (). Ketiga fungsi tersebut merupakan deretan fungsi yang dipanggil untuk mengirimkan data nominal perubahan ketika terjadi penghapusan suatu transaksi kepada: layanan *ledger, dan* layanan akun. Proses dalam menghapus data disini cukup rumit karena data pada layanan *ledger* tidak dapat dihapus untuk alasan keamanan datanya. Jadi yang dilakukan ketika menghapus data adalah dengan menambah data yang bersifat *inverse* dari data yang ingin dihapus agar nominalnya menjadi nol.

Namun penulis merasa bahwa jika API layanan diterapkan seperti pada Tabel 4.4, akan terdapat banyak duplikasi baris kode yang harus ditulis. Karena pada penerapannya hanya berbeda pada data nominal yang dikirim untuk dikalkulasi secara internal oleh masing-masing layanan *ledger* dan akun, maka dari itu penulis mencoba memindahkan baris kode yang memuat seluruh proses kalkulasi dalam dua layanan yang baru, yakni: layanan posting ledger, dan layanan rollback ledger. Dengan begitu tiap layanan yang ingin mengirim data ke ledger hanya tinggal mengirimkan nominal yang mau dikalkulasi oleh *posting ledger* atau *rollback ledger*, jadi tidak perlu mengkalkulasi sendiri dan mengirimkan hasilnya ke masing-masing layanan akun dan *ledger*. Hasil proses kolaborasi setelah ditambahkan dua layanan baru ini dapat dilihat pada Tabel 4.5.

Tabel 4.5 Hasil proses kolaborasi setelah ditambahkan dua layanan baru

No.	Layanan	Identifikasi fungsi	Kolaborator
1	*Posting Ledger*	PostJurnal ()	PostLedger ()
			EditAkun ()
2	*Rollback Ledger*	RollbackJurnal ()	EditLedger ()
			PostJurnal ()
3	Penggajian	CreatePenggajian ()	**PostJurnal ()**
		ReadPenggajian ()	-
		DeletePenggajian ()	**RollbackJurnal ()**
		ReadKaryawan ()	-

No.	Layanan	Identifikasi fungsi	Kolaborator
4	Aset	CreateAset ()	**RollbackJurnal ()**
		ReadAset ()	-
		DeleteAset ()	**RollbackJurnal ()**
5	*Ledger*	GetLedger ()	-
		PostLedger ()	-
		EditLedger ()	-
6	Akun rekening	GetAkun ()	-
		GetAkunById ()	-
		GetAkunByName ()	-
		PostNewAkun ()	-
		EditAkun ()	-
		DeleteAkun()	-
		ChangeTargetNominal ()	-
		GetKategoriAkun ()	-
		GetLaporanLabaRugi ()	-
7	Piutang	CreatePiutang ()	**PostJurnal ()**
		ReadPiutang ()	-
		DeletePiutang ()	**RollbackJurnal ()**
		LunaskanCicilanPiutang ()	**PostJurnal ()**
8	Hutang	CreateHutang ()	**PostJurnal ()**
		ReadHutang ()	-
		DeleteHutang ()	**RollbackJurnal ()**
		LunaskanCicilanHutang ()	**PostJurnal ()**
9	*Invoice*	Create*Invoice* ()	-
		Read*Invoice* ()	-
		Delete*Invoice* ()	-
10	Arus kas	CreateArusKas ()	**PostJurnal ()**
		ReadArusKas ()	-

No.	Layanan	Identifikasi fungsi	Kolaborator
		DeleteArusKas ()	**RollbackJurnal ()**

4.2.4 Identifikasi Format Komunikasi Antar API Layanan

Terdapat dua hal yang harus diidentifikasi pada format komunikasi ini, yakni: tipe komunikasi, dan format pesan. Pada penelitian ini, tipe komunikasi yang digunakan pada penerapannya adalah tipe *syncronous*. Sedangkan, format pesan yang digunakan adalah berbasis text dengan format JSON.

Syncronous ini merupakan tipe interaksi secara *request* dan *response* berurutan. Yang mana, tiap ada *request* masuk ke API layanan, maka pasti ada *response* dari API tersebut setelah *request* tersebut selesai diproses. Sedangkan, JSON merupakan struktur data berbentuk text yang masing-masing datanya disusun secara berpasang-pasangan (*key* dan *value*) (Richards, 2018).

4.2.5 Identifikasi Kebutuhan Pola Saga Pada API layanan

Pada AE pola finansial ini terdapat kasus dimana satu layanan secara tidak langsung melakukan transaksi pada dua basis data. Kasus tersebut terjadi pada layanan *Posting Ledger* dan layanan *Rollback Ledger*. Posting ledger melakukan transaksi ke layanan akun dan ledger secara bersamaan, begitu pula dengan layanan *rollback ledger*. Karena pada dasarnya basis data MongoDB tidak menyediakan *error exception* ketika terjadi kegagalan dalam transaksi yang melibatkan dua basis data dalam sekali jalan, maka diterapkan pola saga. Pola saga ini nantinya diterapkan pada level aplikasi di layanan *posting ledger* dan *rollback ledger*.

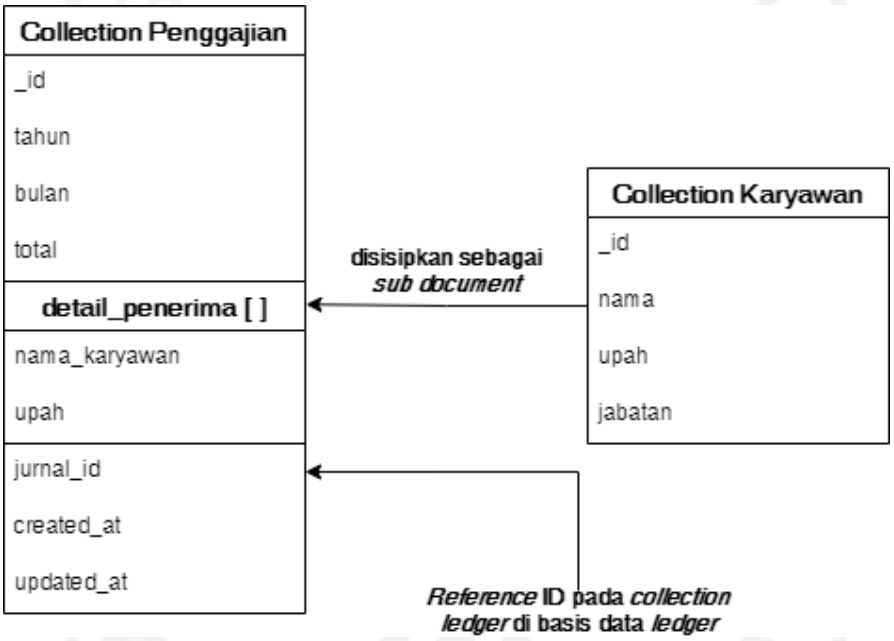
4.3 Analisis Kebutuhan Pada Basis Data

Setelah berhasil menganalisis kebutuhan pada API layanan, selanjutnya dilakukan analisis kebutuhan pada basis data. Analisis ini dilakukan untuk mengidentifikasi data yang dibutuhkan oleh tiap layanannya. Pada kasus AE pola finansial ini, terdapat delapan basis data yang diterapkan. Dua layanan yang tidak memiliki basis data adalah layanan *posting ledger* dan *rollback ledger*.

4.3.1 Skema Basis Data Penggajian

Seperti yang terlihat pada Gambar 4.3, terdapat dua *collection* pada basis data penggajian, yakni: *collection* penggajian dan *collection* karyawan. Pada *collection* penggajian terdapat

delapan atribut data, yakni: _id, tahun, bulan, total, detail_penerima, jurnal_id, *created_at*, dan *updated_at*. Atribut _id di sini digunakan sebagai *primary key* pada *collection* ini. Atribut detail_penerima merupakan data *array* yang berisi daftar karyawan yang menerima gaji. Atribut jurnal_id merupakan *foreign key* dari basis data layanan *ledger*. Atribut jurnal_id ini digunakan untuk merujuk data yang harus di-*rollback* pada layanan *ledger* ketika ada data pada layanan penggajian yang dihapus. Pada collection karyawan, terdapat empat atribut data, yakni: _id, nama, upah, jabatan. Nantinya collection karyawan ini diisi oleh data *dummy*.



Gambar 4.3 Diagram relasi antar *collection* pada basis data penggajian

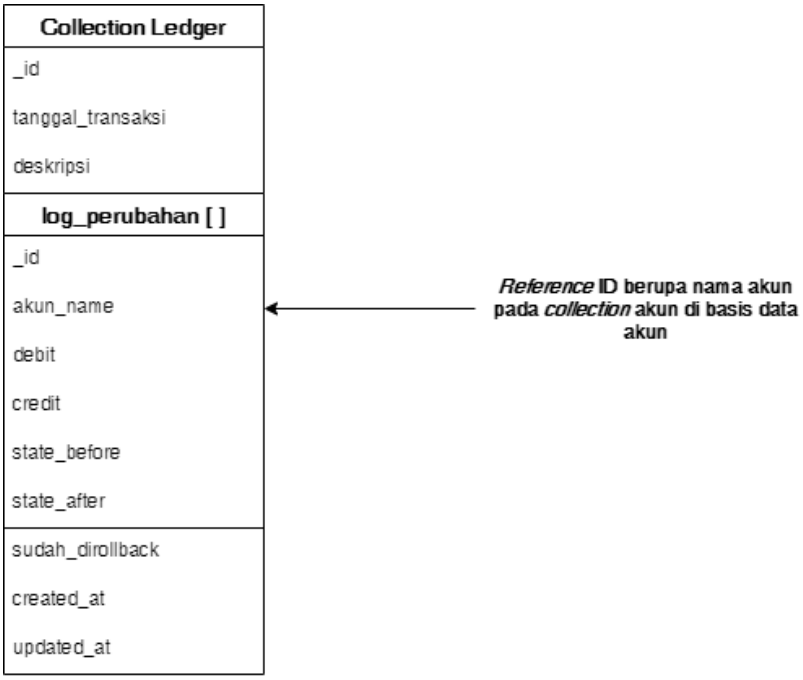
4.3.2 Skema Basis Data Aset

Seperti yang terlihat pada Gambar 4.4, terdapat satu *collection* pada basis data aset, yakni: *collection* aset. Pada *collection* aset ini terdapat sembilan atribut data, yakni: _id, tanggal_pembelian, nama_stok, harga_satuan, jumlah_stok, total_harga, jurnal_id, *created_at*, *updated_at*. Atribut _id di sini digunakan sebagai *primary key* pada *collection*. Atribut jurnal_id merupakan *foreign key* dari basis data layanan *ledger*. Atribut jurnal_id ini digunakan untuk merujuk data yang harus di-*rollback* pada layanan *ledger* ketika ada data pada layanan aset yang ingin dihapus.

Gambar 4.4 Diagram relasi antar *collection* pada basis data aset

4.3.3 Skema Basis Data *Ledger*

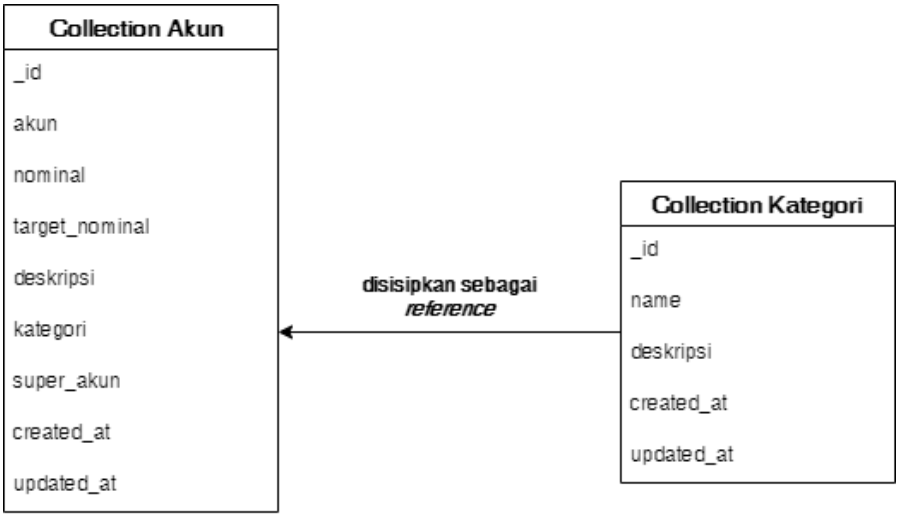
Seperti yang terlihat pada Gambar 4.5, terdapat satu collection pada basis data ledger, yakni: *collection ledger*. Pada *collection* ledger ini terdapat tujuh atribut data, yakni: _id, tanggal_transaksi, deskripsi, *log*_perubahan, sudah_dirollback, *created_at*, *updated_at*. Atribut _id di sini digunakan sebagai *primary key* pada *collection*. Atribut *log*_perubahan merupakan *sub document* yang disisipkan dalam bentuk data array. Atribut log_perubahan ini memiliki lima sub atribut, yakni: _id, akun_name, debit, credit, state_before, state_after.



Gambar 4.5 Diagram relasi antar *collection* pada basis data *ledger*

4.3.4 Skema Basis Data Akun

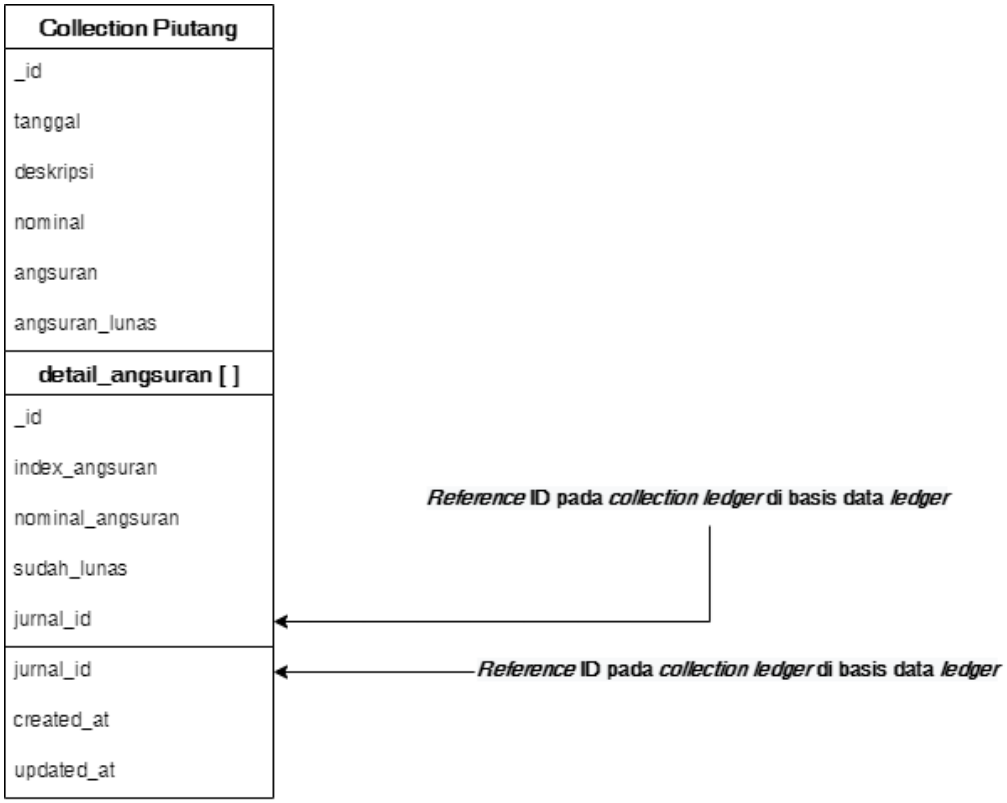
Seperti yang terlihat pada Gambar 4.6, terdapat dua collection pada basis data akun, yakni: *collection* akun, dan *collection* kategori. Pada *collection* akun terdapat sembilan atribut data, yakni: _id, akun, nominal, target nominal, deskripsi, kategori, super_akun, created_at, updated_at. Atribut _id disini digunakan sebagai *primary key* pada *collection* akun. Atribut kategori merupakan semacam *foreign key* yang menghubungkan antara akun dan kategori. Pada collection kategori terdapat lima atribut, yakni: _id, name, deskripsi, created_at, updated_at. Atribut _id di sini merupakan *primary key* pada *collection* kategori.



Gambar 4.6 Diagram relasi antar *collection* pada basis data akun

4.3.5 Skema Basis Data Piutang

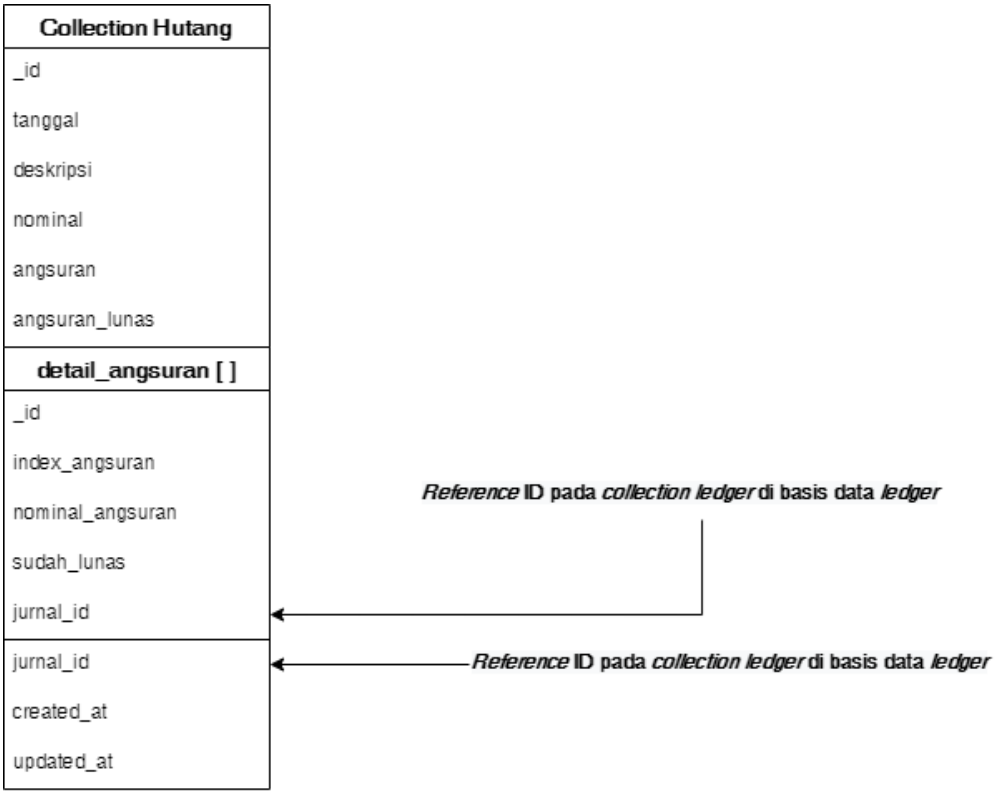
Seperti yang terlihat pada Gambar 4.7, terdapat satu *collection* pada basis data piutang, yakni: *collection* piutang. Pada *collection* piutang ini terdapat sepuluh atribut data, yakni: _id, tanggal, deskripsi, nominal, angsuran, angsuran_lunas, detail_angsuran, jurnal_id ,*created_at*, *updated_at*. Atribut _id disini digunakan sebagai *primary key* pada *collection* piutang. Atribut jurnal_id merupakan *foreign key* dari basis data layanan *ledger*. Atribut jurnal_id ini digunakan untuk merujuk data yang harus di-*rollback* pada layanan *ledger* ketika ada data pada layanan piutang yang ingin dihapus. Pada atribut detail_angsuran terdapat lima sub atribut, yakni: _id, index_angsuran, nominal_angsuran, sudah_lunas, jurnal_id.



Gambar 4.7 Diagram relasi antar *collection* pada basis data piutang

4.3.6 Skema Basis Data Hutang

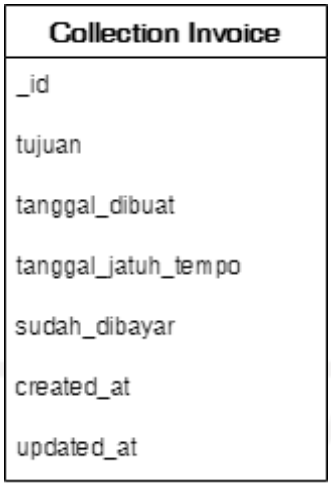
Seperti yang terlihat pada Gambar 4.8, terdapat satu *collection* pada basis data hutang, yakni: *collection* hutang. Pada *collection* hutang ini terdapat sepuluh atribut data, yakni: _id, tanggal, deskripsi, nominal, angsuran, angsuran_lunas, detail_angsuran, jurnal_id ,*created_at*, *updated_at*. Atribut _id di sini digunakan sebagai *primary key* pada *collection* hutang. Atribut jurnal_id merupakan *foreign key* dari basis data layanan ledger. Atribut jurnal_id ini digunakan untuk merujuk data yang harus di-*rollback* pada layanan *ledger* ketika ada data pada layanan hutang yang ingin dihapus. Pada atribut detail_angsuran terdapat lima sub atribut, yakni: _id, index_angsuran, nominal_angsuran, sudah_lunas, jurnal_id



Gambar 4.8 Diagram relasi antar *collection* pada basis data hutang

4.3.7 Skema Basis Data *Invoice*

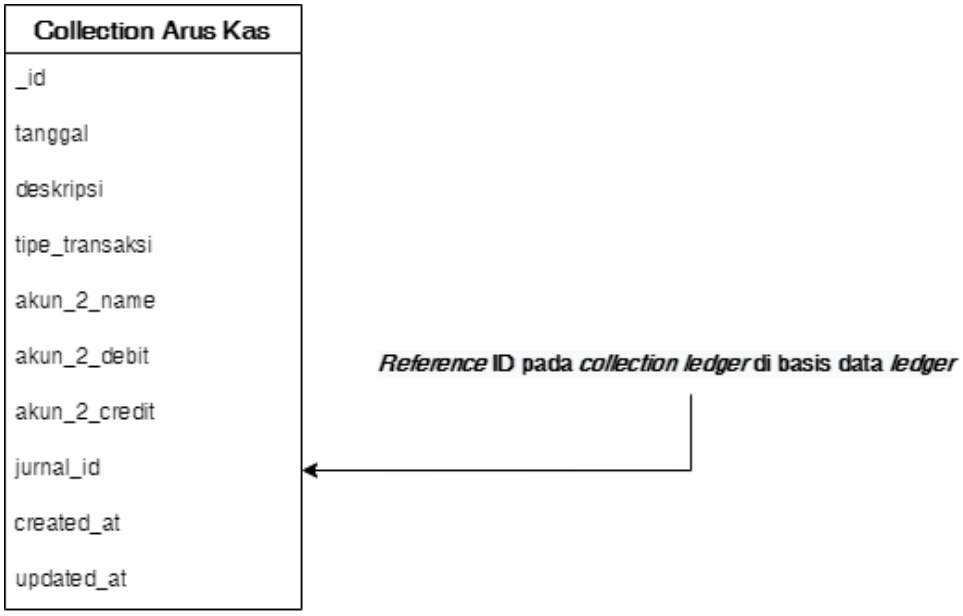
Seperti yang terlihat pada Gambar 4.9, terdapat satu *collection* pada basis data *invoice*, yakni: *collection invoice*. Pada *collection invoice* ini terdapat tujuh atribut data, yakni: _id, tujuan, tanggal_dibuat, tanggal_jatuh_tempo, sudah_dibayar, *created_at*, *updated_at*. Atribut _id disini digunakan sebagai *primary key* pada *collection invoice*.



Gambar 4.9 Diagram relasi antar *collection* pada basis data *invoice*

4.3.8 Skema Basis Data Arus Kas

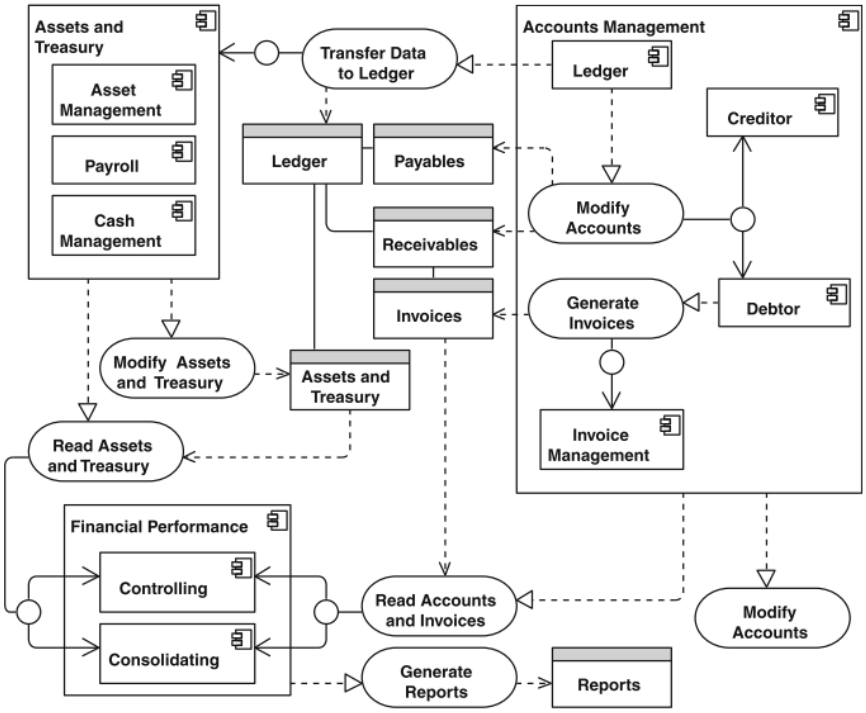
Seperti yang terlihat pada Gambar 4.10, terdapat satu *collection* pada basis data arus kas, yakni: *collection* arus kas. Pada *collection* arus kas ini terdapat sepuluh atribut data, yakni: _id, tanggal, deskripsi, tipe_transaksi, akun_2_name, akun_2_debit, akun_2_credit, jurnal_id ,*created_at*, *updated_at*. Atribut _id di sini digunakan sebagai *primary key* pada *collection* arus kas. Atribut jurnal_id merupakan *foreign key* dari basis data layanan *ledger*. Atribut jurnal_id ini digunakan untuk merujuk data yang harus di-*rollback* pada layanan *ledger* ketika ada data pada layanan arus kas yang ingin dihapus.



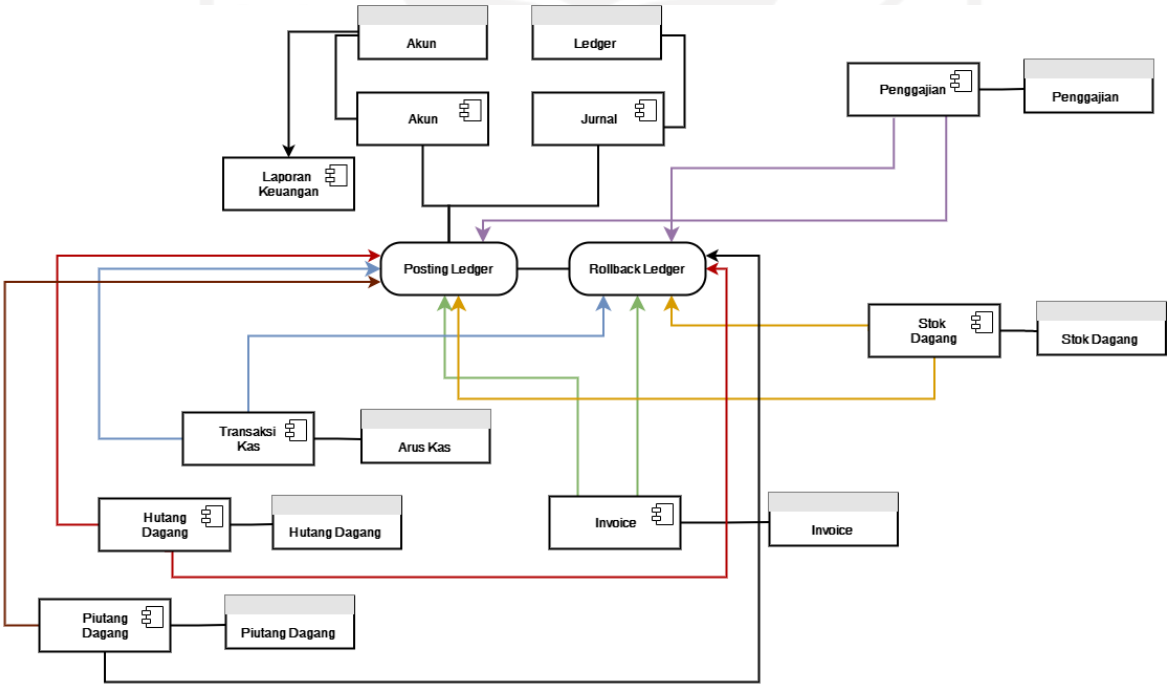
Gambar 4.10 Diagram relasi antar *collection* pada basis data arus kas

4.4 Komparasi Diagram Perspektif Aplikasi Data

Sebelum masuk ke bab penerapan, penulis melakukan komparasi antara diagram komponen AE pola finansial pada buku (Perroud & Inversini, 2013), dan setelah didekomposisi dan dianalisis ulang kebutuhannya. Seperti yang terlihat pada Gambar 4.11 dan Gambar 4.12, perbedaan yang signifikan adalah independensi layanan, dan kepemilikan basis data. Layanan pada Gambar 4.12 sudah berdiri secara independen dan tidak lagi dikelompokan menjadi kelompok aplikasi besar lagi seperti pada Gambar 4.11. Selain itu, layanan pada Gambar 4.12 juga sudah memiliki satu basis data tersendiri kecuali laporan keuangan yang masih menggunakan basis data akun. Pada kasus laporan keuangan ini karena pada laporan keuangan ini mengolah informasi pada basis data akun untuk dijadikan informasi.



Gambar 4.11 Diagram komponen AE pola fiansial pada buku (Perroud & Inversini, 2013)



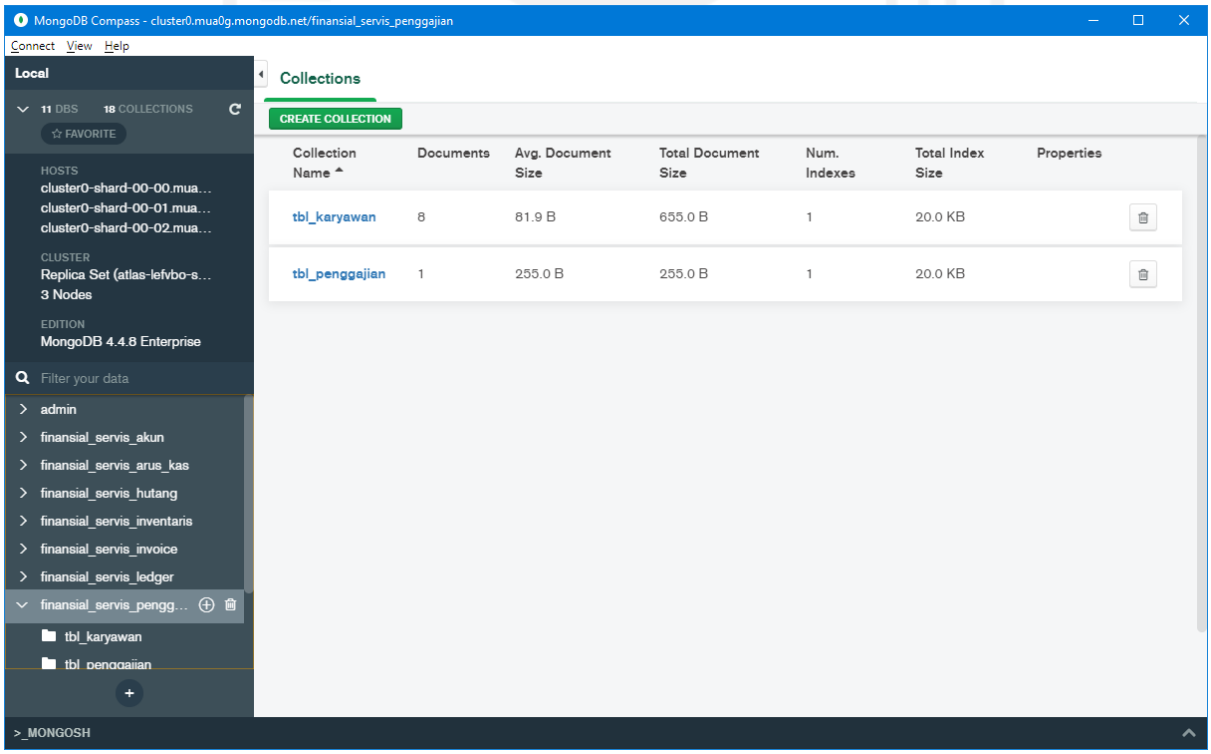
Gambar 4.12 Diagram komponen AE pola fiansial pada aplikasi berbasis *microservices*

4.5 Penerapan Basis Data

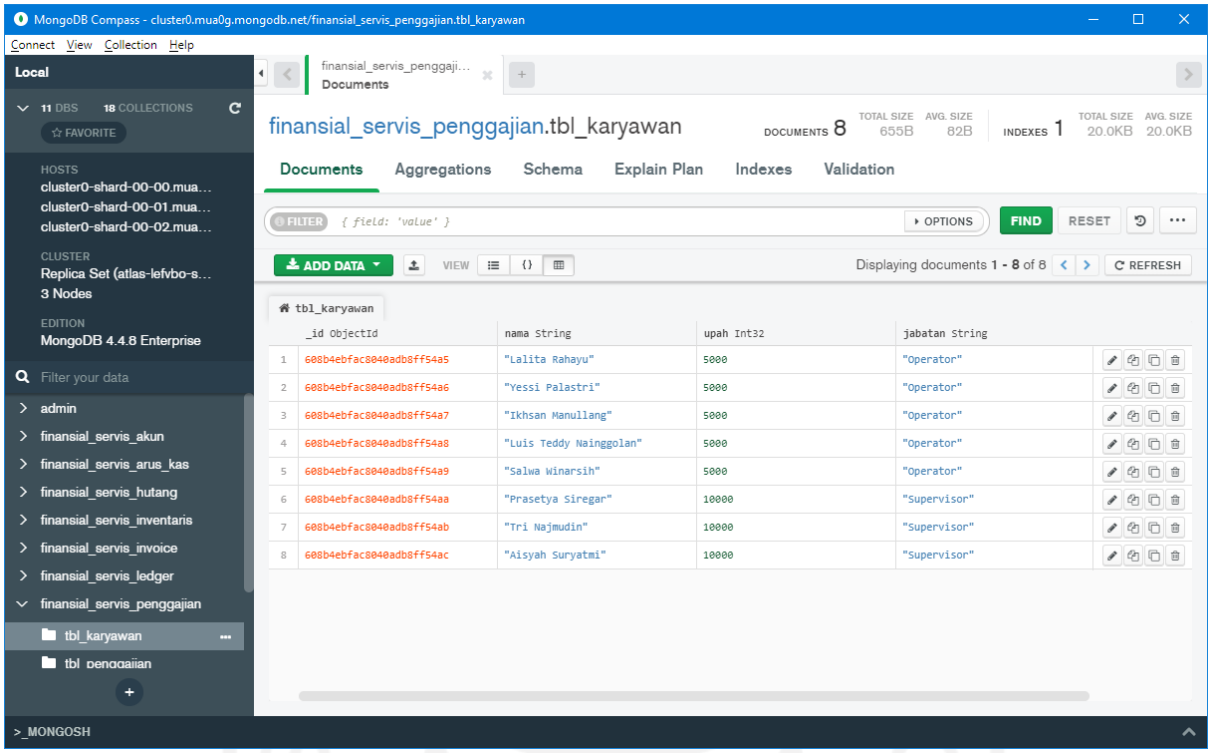
Setelah berhasil menganalisis kebutuhan pada basis basis data, selanjutnya dilakukan proses penerapan pada basis data. Pada tahapan ini, aktivitas yang dilakukan hanya pembuatan *collection* di tiap basis data, dan menambahkan data *dummy* pada *collection* yang membutuhkan. Aplikasi yang digunakan untuk melakukan hal tersebut adalah MongoDB *Compass*. MongoDB *Compass* ini merupakan aplikasi untuk mengelola basis data pada MongoDB dengan *Graphical User interface* (GUI). Untuk pembatasan tipe data pada tiap atributnya nanti dilakukan pada level aplikasi.

4.5.1 Penerapan Basis Data Penggajian

Hasil penerapan basis data penggajian yang sudah dilakukan dapat dilihat pada Gambar 4.13, yang mana terdapat dua *collection*, yakni: tbl_karyawan, dan tbl_penggajian. Untuk penamaannya sendiri ditambahkan "tbl" sebelum nama *collection*-nya. Hal tersebut merupakan preferensi pribadi penulis dalam memberi nama, dan tidak berpengaruh ke performa apapun. Selain itu, Hasil penerapan data *dummy* pada *collection* penggajian dapat dilihat pada Gambar 4.14.



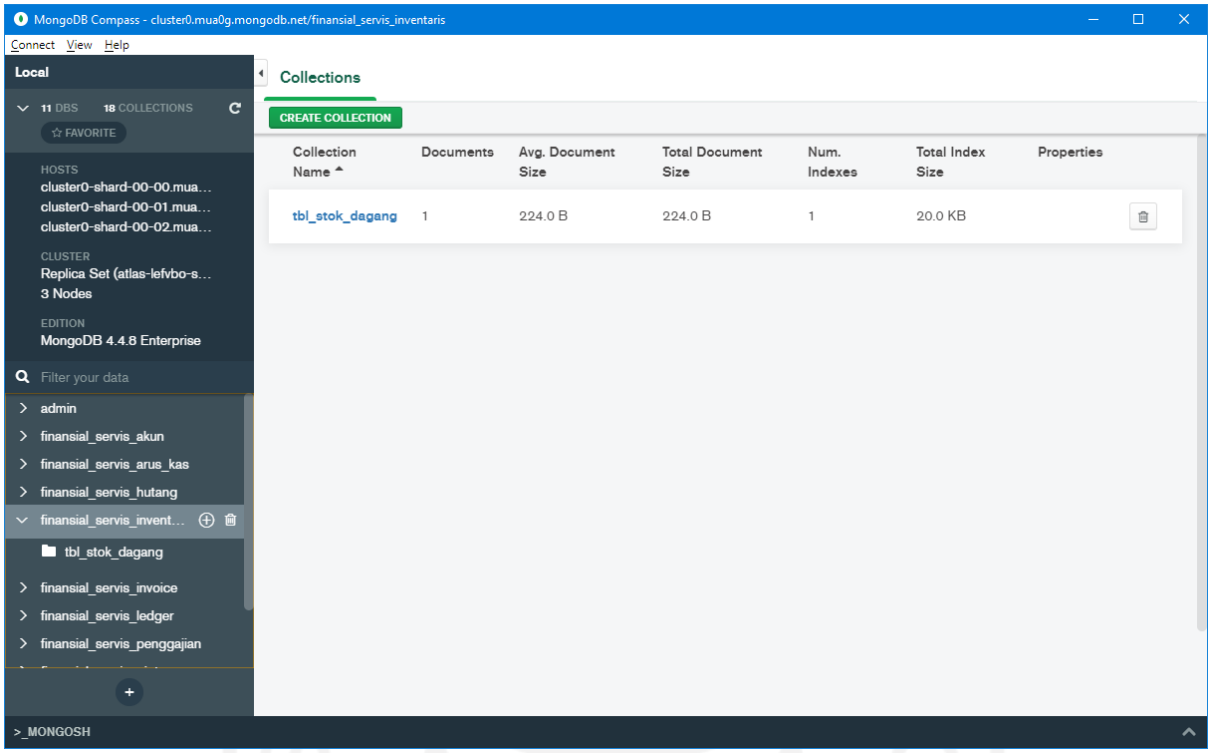
Gambar 4.13 Hasil penerapan basis data penggajian



Gambar 4.14 Hasil penerapan data dummy pada *collection* karyawan

4.5.2 Penerapan Basis Data Aset

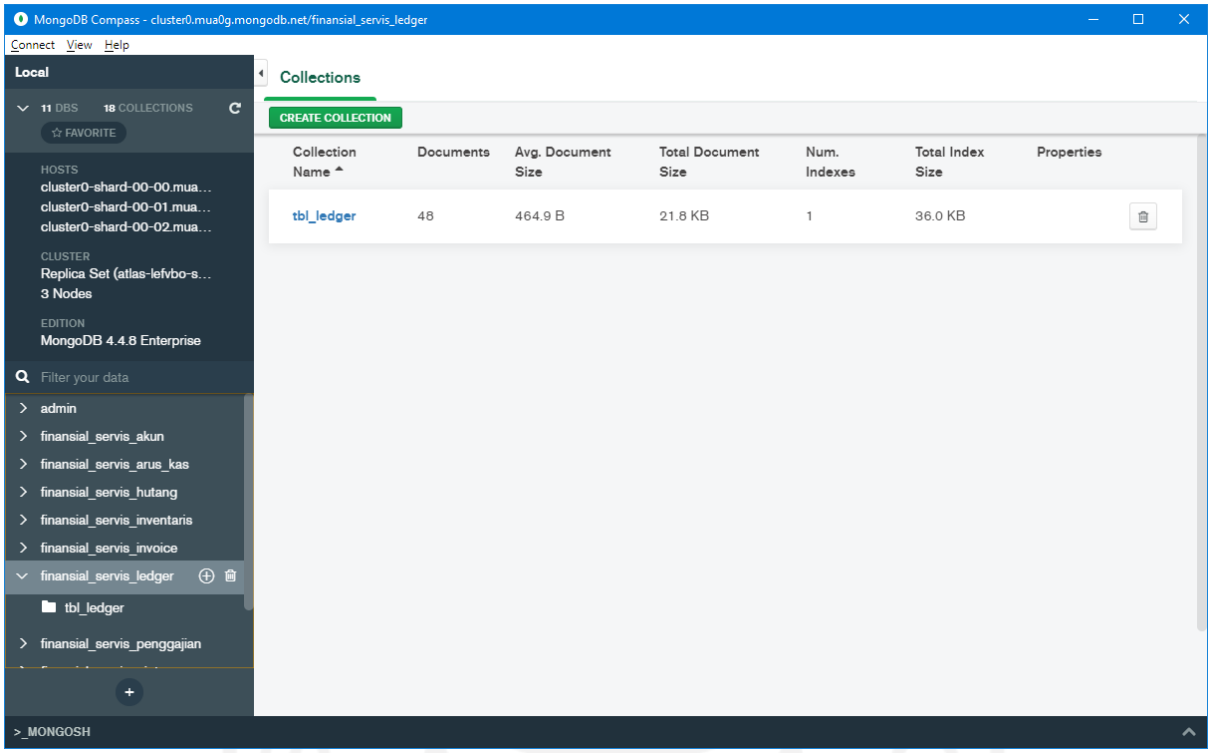
Hasil penerapan basis data penggajian yang sudah dilakukan dapat dilihat pada Gambar 4.15, yang mana terdapat satu *collection*, yakni: tbl_stok_dagang.



Gambar 4.15 Hasil penerapan basis data aset

4.5.3 Penerapan Basis Data *Ledger*

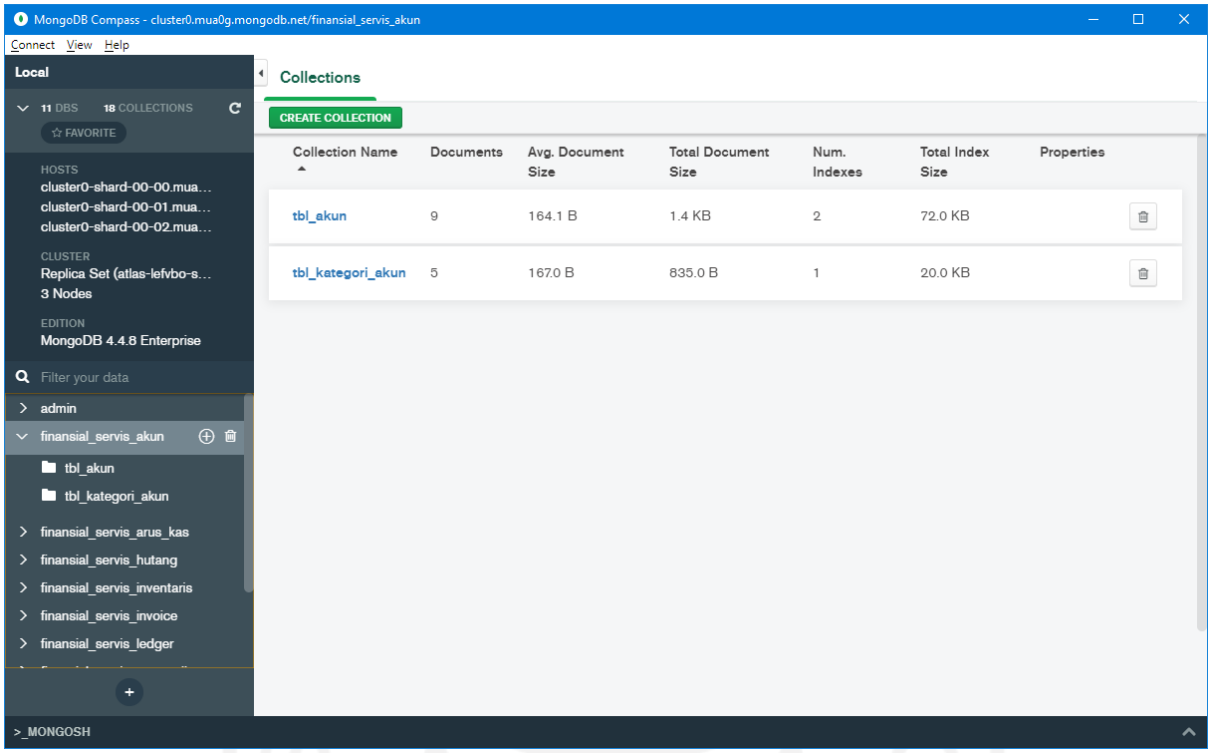
Hasil penerapan basis data penggajian yang sudah dilakukan dapat dilihat pada Gambar 4.16, yang mana terdapat satu *collection*, yakni: tbl_ledger.



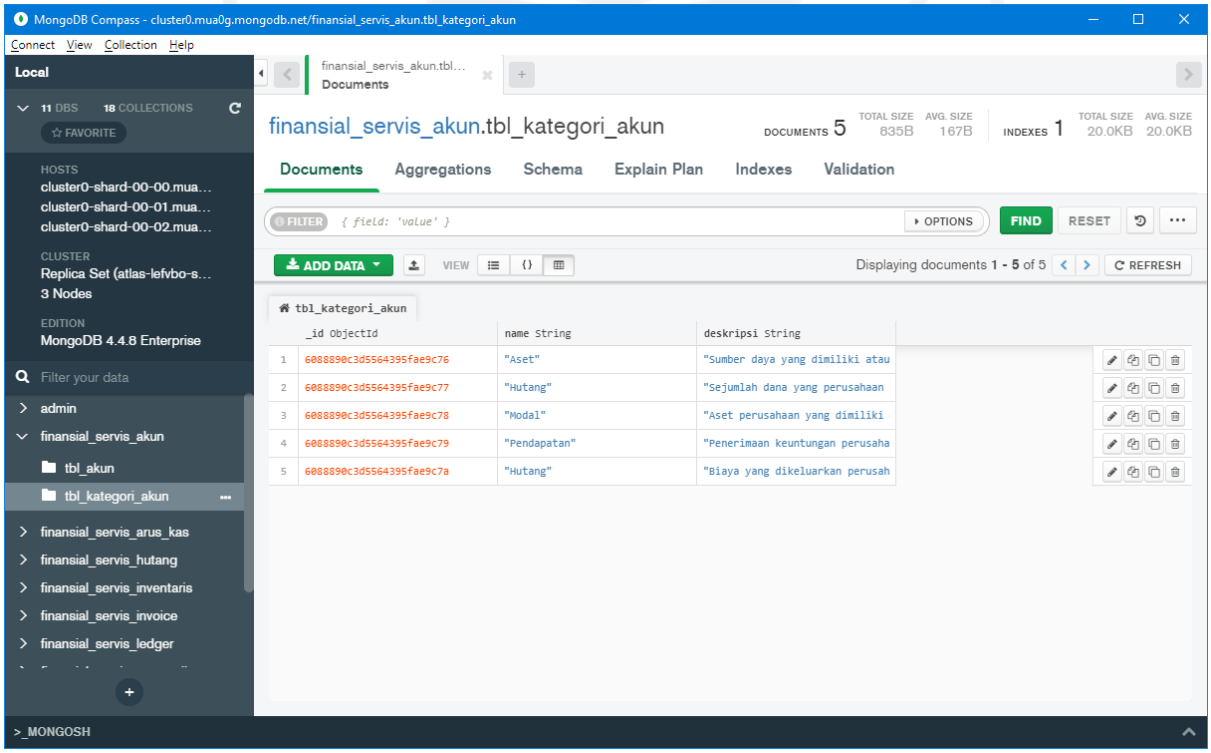
Gambar 4.16 Hasil penerapan basis data *ledger*

4.5.4 Penerapan Basis Data Akun

Hasil penerapan basis data penggajian yang sudah dilakukan dapat dilihat pada Gambar 4.17, yang mana terdapat dua *collection*, yakni: tbl_akun, dan tbl_kategori_akun. Hasil penerapan dari penambahan data kategori dilihat pada Gambar 4.18. Kelima kategori yang ditambahkan ini merupakan kategori yang cukup krusial pada kegiatan akunting.



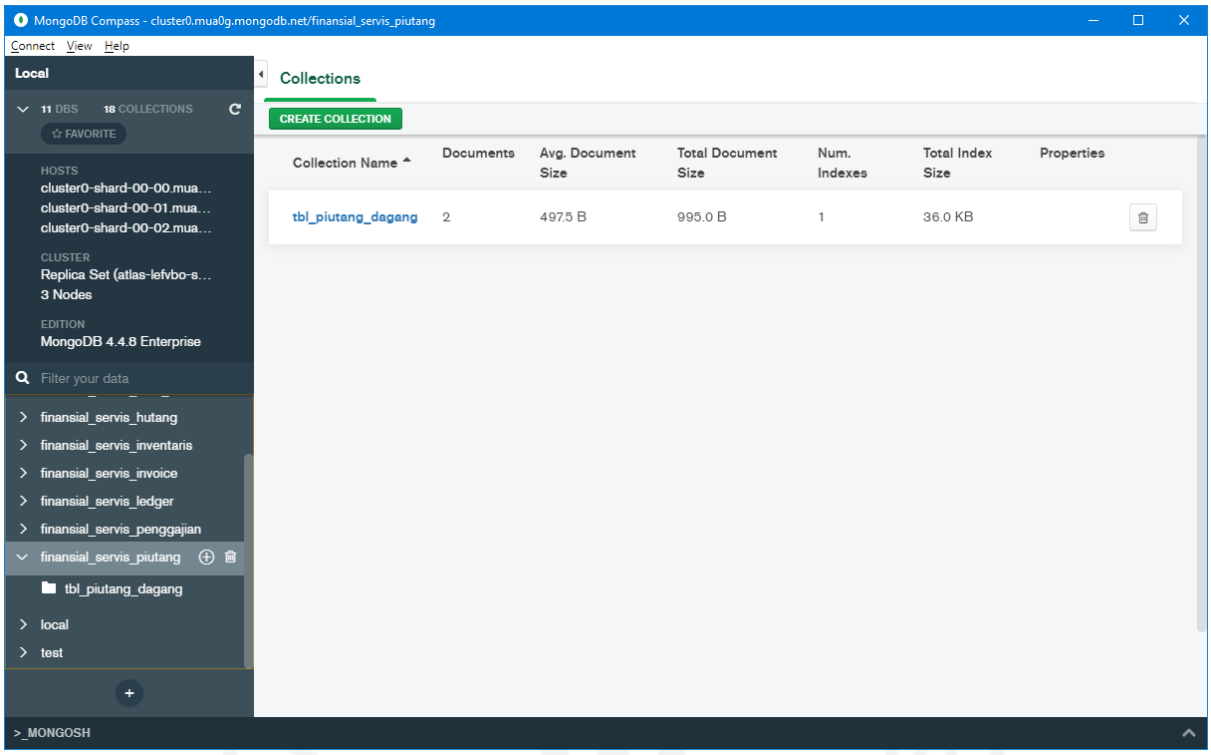
Gambar 4.17 Hasil penerapan basis data akun



Gambar 4.18 Hasil penerapan dari penambahan data pada *collection* kategori

4.5.5 Penerapan Basis Data Piutang

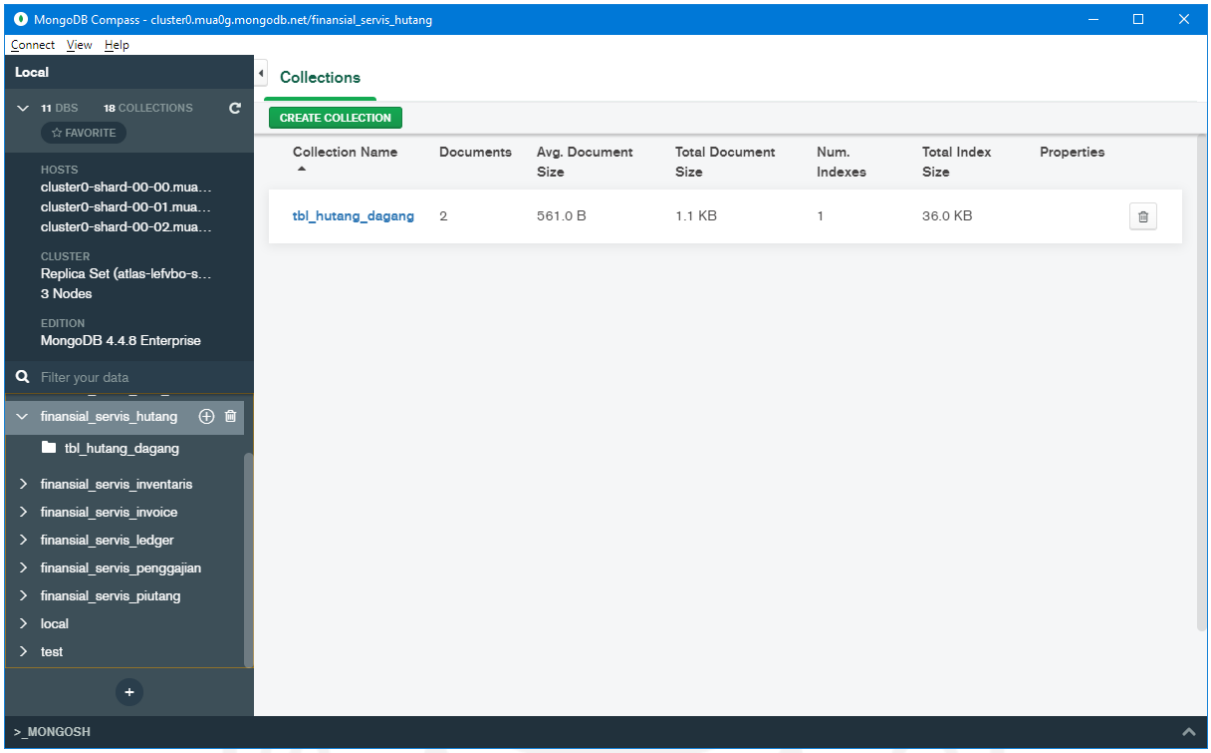
Hasil penerapan basis data piutang yang sudah dilakukan dapat dilihat pada Gambar 4.19, yang mana terdapat satu *collection*, yakni: tbl_piutang_dagang.



Gambar 4.19 Hasil penerapan basis data piutang

4.5.6 Penerapan Basis Data Hutang

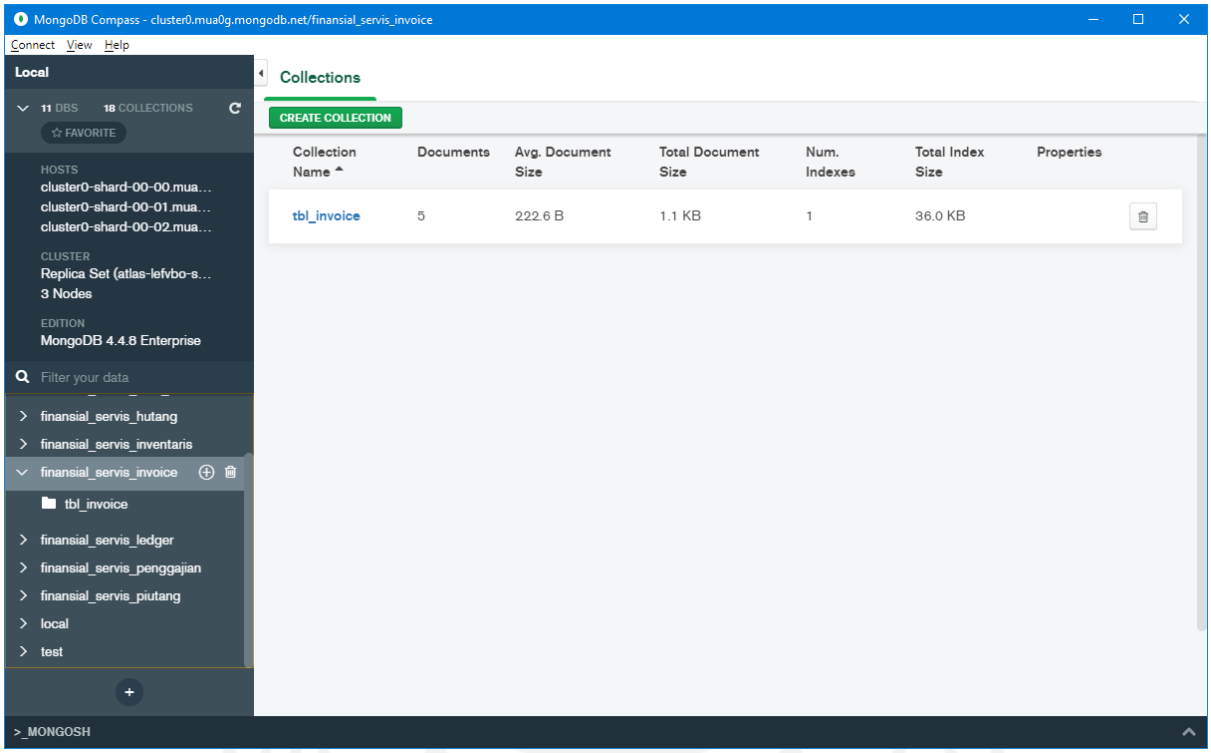
Hasil penerapan basis data hutang yang sudah dilakukan dapat dilihat pada Gambar 4.20, yang mana terdapat satu *collection*, yakni: tbl_hutang_dagang.



Gambar 4.20 Hasil penerapan basis data hutang

4.5.7 Penerapan Basis Data *Invoice*

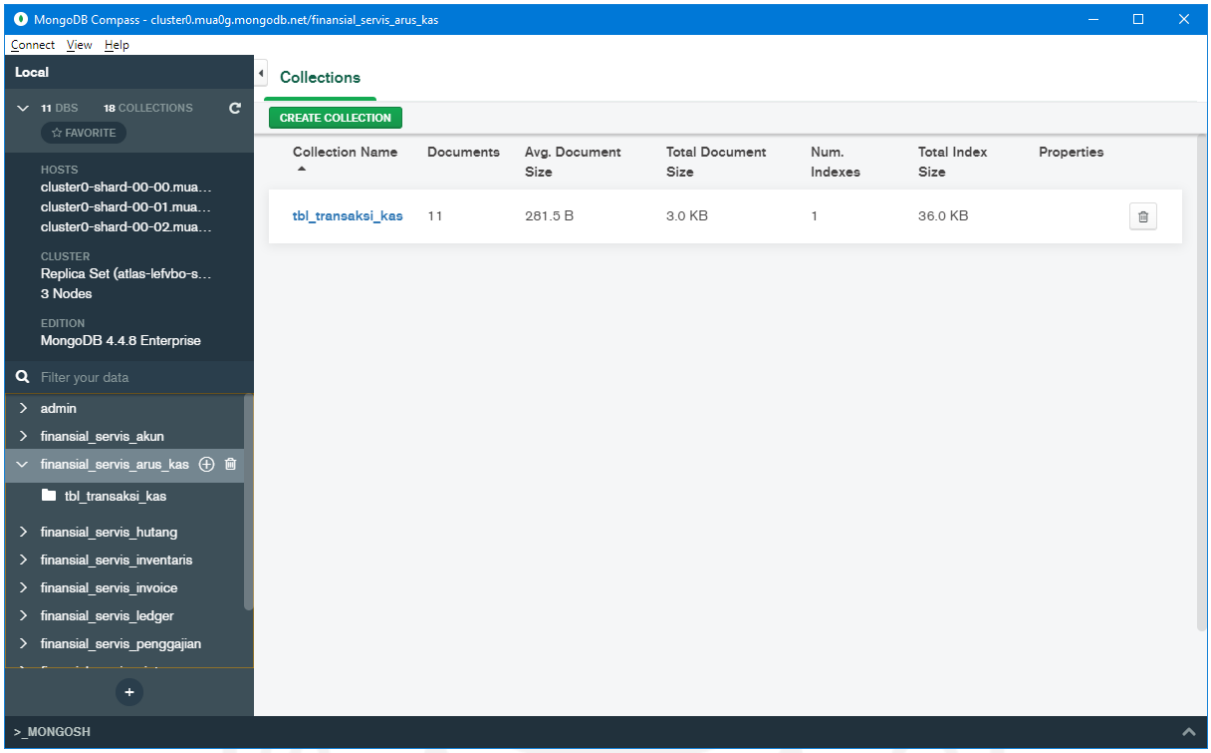
Hasil penerapan basis data *Invoice* yang sudah dilakukan dapat dilihat pada Gambar 4.21, yang mana terdapat satu *collection*, yakni: tbl_invoice.



Gambar 4.21 Hasil penerapan basis data invoice

4.5.8 Penerapan Basis Data Arus Kas

Hasil penerapan basis data arus kas yang sudah dilakukan dapat dilihat pada Gambar 4.22, yang mana terdapat satu *collection*, yakni: tbl_arus_kas.



Gambar 4.22 Hasil penerapan basis data arus kas

4.6 Penerapan API Layanan

Setelah berhasil menerapkan basis data, selanjutnya dilakukan proses penerapan API layanan. Sesuai dengan hasil analisis kebutuhannya, terdapat sepuluh API layanan yang diterapkan.

4.6.1 Paket *Library* yang dipasang

Pada penerapannya, terdapat beberapa paket library yang dipasang **pada tiap** API layanan agar memudahkan proses penerapannya. Berikut daftar paket tersebut beserta penjelasannya:

a. ExpressJS

Seperti yang sudah disebutkan sebelumnya, ExpressJS ini merupakan paket tambahan yang dipasang di atas NodeJS.

a. Axios

Paket ini memuat fitur untuk melakukan request dan menerima response secara HTTP pada NodeJS. Pada penelitian ini, paket ini cukup krusial karena digunakan pada tiap layanan untuk berkomunikasi (*GitHub - Axios/Axios: Promise Based HTTP Client for the Browser and Node.Js*, n.d.).

b. Body-parser

Paket ini memuat memuat fitur untuk mengubah tipe data pada pesan yang digunakan untuk berkomunikasi menjadi tipe data yang mudah untuk diolah, misal: JSON ke *string* (*GitHub - Expressjs/Body-Parser: Node.Js Body Parsing Middleware*, n.d.).

c. Cors

Paket ini memuat fitur untuk melimitasi *client* yang dapat mengakses endpoint pada tiap layanan. Paket ini sangat berguna agar endpoint layanan tidak dapat diakses oleh sembarang client (*GitHub - Expressjs/Cors: Node.Js CORS Middleware*, n.d.).

d. Dotenv

Paket ini memuat fitur untuk menyimpan beberapa data penting dalam bentuk variabel pada file khusus, misal: *port* API, *key* basis data, URL API lain. Jadinya pada aplikasi nanti cukup menuliskan variabelnya saja tidak perlu menulis ulang *key* / URL yang cukup panjang berulang kali. Selain itu, hal ini juga dapat meningkatkan keamanan karena *key* tidak dapat dilihat pada kode program (*GitHub - Motdotla/Dotenv: Loads Environment Variables from .Env for Nodejs Projects.*, n.d.).

e. Mongoose

Paket ini memuat banyak fitur untuk mengelola transaksi dengan basis data, seperti penyederhanaan baris kode untuk melakukan koneksi dengan mongoDB dan *query* data. Selain itu, mongoose ini juga dapat melimitasi tipe data yang akan masuk ke basis data (*Mongoose v5.13.7: Schemas*, n.d.).

f. Nodemon

Paket ini memuat fitur untuk melakukan *restart* otomatis pada aplikasi ketika terjadi perubahan baris kode. Dengan paket ini, pengembang tidak perlu lagi mematikan dan menghidupkan ulang *server* ketika mengedit baris kode (*GitHub - Remy/Nodemon: Monitor for Any Changes in Your Node.Js Application and Automatically Restart the Server - Perfect for Development*, n.d.)

4.6.2 Hasil *Endpoint* yang Dapat Dipakai

Hasil *endpoint* beserta *method* tiap fungsi pada API layanan yang dapat diakses dapat dilihat pada Tabel 4.6.

Tabel 4.6 Hasil endpoint pada penerapan API layanan

No.	Layanan	fungsi	Method	Endpoint yang dapat diakses
1	Posting Ledger	PostJurnal ()	POST	https://posting-ledger.herokuapp.com/posting-jurnal
2	Rollback Ledger	RollbackJurnal ()	POST	https://rollback-ledger.herokuapp.com/rollback-jurnal/:id
3	Penggajian	CreatePenggajian ()	POST	https://penggajian-skripsi2021.herokuapp.com/penggajian
		ReadPenggajian ()	GET	https://penggajian-skripsi2021.herokuapp.com/penggajian
		DeletePenggajian ()	DELETE	https://penggajian-skripsi2021.herokuapp.com/penggajian/:id
		ReadKaryawan ()	GET	https://penggajian-skripsi2021.herokuapp.com/karyawan
4	Aset	CreateAset ()	POST	https://stok-dagang-skripsi2021.herokuapp.com/stok-dagang
		ReadAset ()	GET	https://stok-dagang-skripsi2021.herokuapp.com/stok-dagang
		DeleteAset ()	DELETE	https://stok-dagang-skripsi2021.herokuapp.com/stok-dagang/:id
5	*Ledger*	GetLedger ()	GET	https://gentle-earth-65812.herokuapp.com/ledger
		GetLedgerById ()	GET	https://gentle-earth-65812.herokuapp.com/ledger/:id
		PostLedger ()	POST	https://gentle-earth-65812.herokuapp.com/ledger
		EditLedger ()	PUT	https://gentle-earth-65812.herokuapp.com/ledger

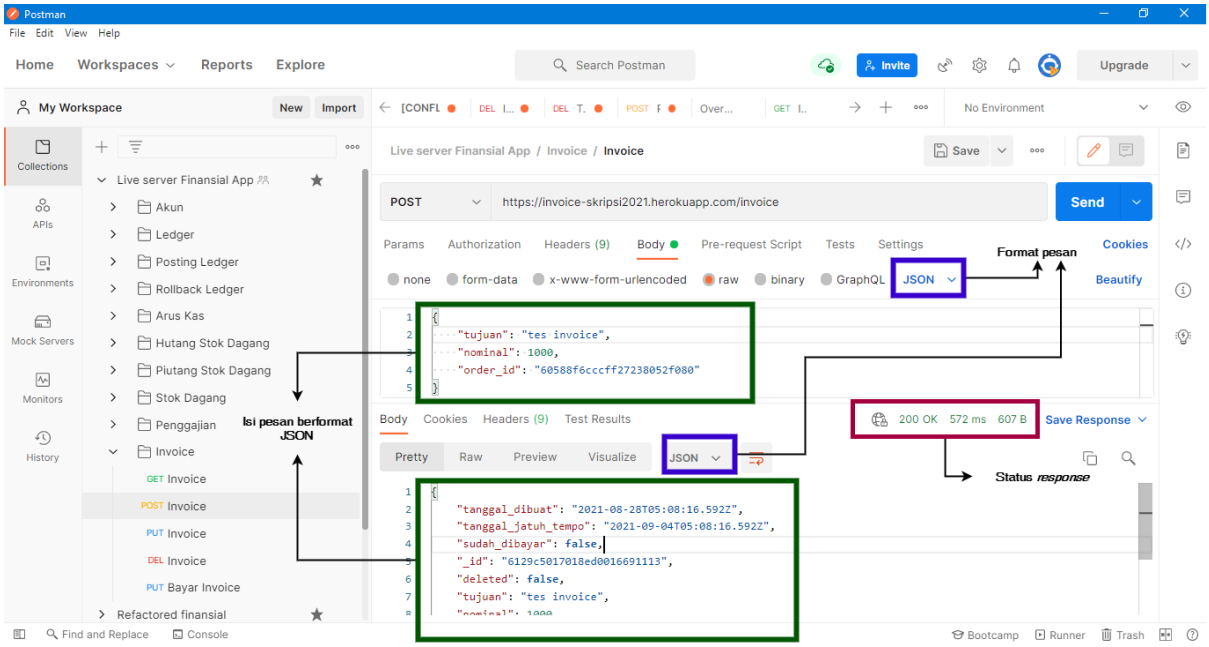
No.	Layanan	fungsi	Method	Endpoint yang dapat diakses
		DeleteLedger ()	DELETE	https://gentle-earth-65812.herokuapp.com/ledger/:id
6	Akun rekening	GetAkun ()	GET	https://lit-anchorage-31799.herokuapp.com/akun
		GetAkunById ()	GET	https://lit-anchorage-31799.herokuapp.com/akun/:id
		GetAkunByName ()	GET	https://lit-anchorage-31799.herokuapp.com/akun/name/:name
		PostNewAkun ()	POST	https://lit-anchorage-31799.herokuapp.com/akun
		EditAkun ()	PUT	https://lit-anchorage-31799.herokuapp.com/akun
		DeleteAkun()	DELETE	https://lit-anchorage-31799.herokuapp.com/akun/force-delete/:id
		ChangeTargetNominal ()	PUT	https://lit-anchorage-31799.herokuapp.com/akun
		GetKategoriAkun ()	GET	https://lit-anchorage-31799.herokuapp.com/kategori
		GetLaporanLabaRugi ()	GET	https://lit-anchorage-31799.herokuapp.com/akun/:id
7	Piutang	CreatePiutang ()	POST	https://piutang-dagang-skripsi2021.herokuapp.com/piutang-dagang
		ReadPiutang ()	GET	https://piutang-dagang-skripsi2021.herokuapp.com/piutang-dagang
		DeletePiutang ()	DELETE	https://piutang-dagang-skripsi2021.herokuapp.com/piutang-dagang/delete/:id
		LunaskanCicilanPiutang ()	PUT	https://piutang-dagang-skripsi2021.herokuapp.com/piutang-dagang/lunaskan-cicilan/:id

No.	Layanan	fungsi	Method	Endpoint yang dapat diakses
8	Hutang	CreateHutang ()	POST	https://hutang-stok-dagang-skripsi2021.herokuapp.com/hutang-dagang
		ReadHutang ()	GET	https://hutang-stok-dagang-skripsi2021.herokuapp.com/hutang-dagang
		DeleteHutang ()	DELETE	https://hutang-stok-dagang-skripsi2021.herokuapp.com/hutang-dagang/delete/:id
		LunaskanCicilanHutang ()	PUT	https://hutang-stok-dagang-skripsi2021.herokuapp.com/hutang-dagang/bayar-cicilan/:id
9	*Invoice*	Create*Invoice* ()	POST	https://invoice-skripsi2021.herokuapp.com/invoice
		Read*Invoice* ()	GET	https://invoice-skripsi2021.herokuapp.com/invoice
		Delete*Invoice* ()	DELETE	https://invoice-skripsi2021.herokuapp.com/invoice/delete/:Id
10	Arus kas	CreateArusKas ()	POST	https://arus-kas-skripsi2021.herokuapp.com/transaksi-kas
		ReadArusKas ()	GET	https://arus-kas-skripsi2021.herokuapp.com/transaksi-kas
		DeleteArusKas ()	DELETE	https://arus-kas-skripsi2021.herokuapp.com/transaksi-kas/:id

4.6.3 Hasil Penerapan Format Komunikasi

Gambar 4.23 Contoh pesan komunikasi pada API layanan di bawah ini merupakan contoh pesan komunikasi pada layanan *invoice*. Tipe komunikasi *syncronous* dapat dilihat dari adanya *request* dan *response* secara berurutan. Yang mana pada kotak merah di Gambar 4.23 terlihat bahwa *response* tersebut dikirimkan 572*ms* setelah pesan request dikirim. Untuk contoh format

JSON sendiri dapat dilihat pada kotak hijau di Gambar 4.23. Yang mana, tulisan berwarna merah merupakan *key* dan tulisan yang berwarna biru merupakan *value*.



Gambar 4.23 Contoh pesan komunikasi pada API layanan

4.6.4 Hasil Penerapan Pola Saga

Gambar 4.24 di bawah ini merupakan baris kode pola saga pada layanan *posting ledger*. Namun, untuk baris kode: kalkulasi nominal transaksi akun, transaksi basis data, dan *rollbacknya* tidak disisipkan karena terlalu panjang. Pada Gambar 4.24 tersebut terdapat empat baris *try-catch*. Baris kode dalam *try* merupakan baris kode yang dijalankan, sedangkan baris kode dalam *catch* merupakan baris kode yang dijalankan **hanya jika** baris kode *try* mengalami *error*.

```
const Axios = require('axios');
require('dotenv/config');

exports.postJurnal = async (req, res, next) => {

try {
    // Kalkulasi nominal transaksi akun

}catch (error){
    // Return error

}

try {
```

```
    // T1 = transaksi menambah data dengan basis data ledger melalui layanan
ledger

}catch (error){
    // Rollback T1
    // Return error
}

try {
    // T2 = transaksi edit data akun 1 dengan basis data akun melalui layanan
akun

}catch (error){
    // Rollback T2
    // Rollback T1
    // Return error

}

try {
    // T3 = transaksi menambah edit data akun 2 dengan basis data akun melalui
layanan akun


}catch (error){
    // Rollback T3
    // Rollback T2
    // Rollback T1
    // Return error

}
```

Gambar 4.24 Baris kode pola saga pada layanan *posting ledger*

4.7 Pengujian API Layanan

Setelah API layanan berhasil diterapkan, selanjutnya dilakukan proses pengujian API layanan. Seperti yang sudah dijelaskan pada bab 3.7, pengujian pada AE pola finansial ini dilakukan dengan cara mengakses satu persatu *endpoint* pada API layanan lalu menjalankan skenario *test case*-nya menggunakan Postman.

4.7.1 Pengujian Layanan *Posting Ledger*

Hasil akhir pengujian terhadap layanan penggajian dapat dilihat pada Tabel 4.7.

Tabel 4.7 Hasil pengujian pada layanan *posting ledger*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
D. Fungsi CreatePenggajian ()				
1	*Unit*	Mengkalkulasi nominal transaksi berdasarkan parameter yang dikirim ke endpoint PostingLedger ()	Hasil kalkulasi tersebut sesuai yang diprediksikan	Valid
2	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan *ledger*	Data pada: basis data akun berubah, basis data ledger bertambah	Valid
3	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan akun	Data pada: basis data akun berubah, basis data ledger bertambah	Valid
4	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan *ledger*, **namun** layanan *ledger* dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah	Valid
5	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan akun, **namun** layanan akun dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah	Valid

4.7.2 Pengujian Layanan *Rollback Ledger*

Hasil akhir pengujian terhadap layanan penggajian dapat dilihat pada Tabel 4.7.

Tabel 4.8 Hasil pengujian pada layanan *rollback ledger*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
E. Fungsi CreatePenggajian ()				
1	*Unit*	Mengkalkulasi nominal *inverse* dari transaksi yang ingin di-rollback berdasarkan parameter yang dikirim ke endpoint PostingLedger ()	Hasil kalkulasi *inverse* tersebut sesuai yang diprediksikan	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
2	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan *posting ledger*	Data pada: basis data akun berubah, basis data *ledger* bertambah	Valid
3	*Integration*	Mengirimkan hasil kalkulasi tersebut pada layanan *posting ledger*, **namun** layanan *posting ledger* dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah	Valid

4.7.3 Pengujian Layanan Penggajian

Hasil akhir pengujian terhadap layanan penggajian dapat dilihat pada Tabel 4.9.

Tabel 4.9 Hasil pengujian pada layanan penggajian

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreatePenggajian ()				
1	*Unit*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang sesuai	Data penggajian pada basis data penggajian bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data penggajian	Valid
3	*Integration*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger bertambah, dan basis data penggajian bertambah	Valid
4	*Integration*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang **tidak** sesuai, **dan** layanan *posting* ledger yang menyala	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data penggajian tidak bertambah	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
5	*Integration*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data penggajian tidak bertambah	Valid
6	*Integration*	Mengakses *endpoint* CreatePenggajian () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala, **namun** saldo akun rekening tidak cukup untuk melakukan penggajian	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah dan basis data penggajian tidak bertambah	Valid
B. Fungsi ReadPenggajian ()				
7	*Unit*	Mengakses *endpoint* ReadPenggajian ()	Seluruh data penggajian dapat ditampilkan	Valid
C. Fungsi DeletePenggajian ()				
8	*Unit*	Mengakses *endpoint* DeletePenggajian () dengan parameter data yang sesuai	Data penggajian pada basis data penggajian terhapus	Valid
9	*Unit*	Mengakses *endpoint* DeletePenggajian () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data penggajian tidak terhapus	Valid
10	*Integration*	Mengakses *endpoint* DeletePenggajian () dengan parameter data yang sesuai, **dan** layanan *rollback* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger ter-*rollback,* dan basis data penggajian terhapus	Valid
11	*Integration*	Mengakses *endpoint* DeletePenggajian () dengan parameter data yang **tidak**	*Error*, dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		sesuai, **dan** layanan *rollback* ledger yang menyala	*rollback,* dan basis data penggajian tidak terhapus	
12	*Integration*	Mengakses *endpoint* DeletePenggajian () dengan parameter data yang sesuai, namun layanan *rollback* ledger dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-*rollback* dan basis data penggajian tidak terhapus	Valid
D. Fungsi ReadKaryawan()				
13	*Unit*	Mengakses *endpoint* ReadKaryawan ()	Seluruh data karyawan dapat ditampilkan	Valid

4.7.4 Pengujian Layanan Aset

Hasil akhir pengujian terhadap layanan aset dapat dilihat pada Tabel 4.10.

Tabel 4.10 Hasil pengujian layanan aset

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreateAset ()				
1	*Unit*	Mengakses *endpoint* CreateAset () dengan parameter data yang sesuai	Data aset pada basis data aset bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreateAset() dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data aset	Valid
3	*Integration*	Mengakses *endpoint* CreateAset () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger bertambah, dan basis data aset bertambah	Valid
4	*Integration*	Mengakses *endpoint* CreateAset () dengan parameter data yang **tidak**	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		sesuai, **dan** layanan *posting* ledger yang menyala	bertambah, dan basis data aset tidak bertambah	
5	*Integration*	Mengakses *endpoint* CreateAset () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	*Error,* dan data pada basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data aset tidak bertambah	Valid
6	*Integration*	Mengakses *endpoint* CreateAset () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala, **namun** saldo akun rekening tidak cukup untuk melakukan pembelian aset baru	*Error,* dan data pada basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data aset tidak bertambah	Valid
B. Fungsi ReadAset ()				
7	*Unit*	Mengakses *endpoint* ReadAset ()	Seluruh data aset dapat ditampilkan	Valid
C. Fungsi DeleteAset ()				
8	*Unit*	Mengakses *endpoint* DeleteAset () dengan parameter data yang sesuai	Data aset pada basis data aset terhapus	Valid
9	*Unit*	Mengakses *endpoint* DeleteAset () dengan parameter data yang **tidak** sesuai	*Error,* dan data aset pada basis data aset tidak terhapus	Valid
10	*Integration*	Mengakses *endpoint* DeleteAset () dengan parameter data yang sesuai, **dan** layanan *rollback* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger ter-*rollback,* dan basis data aset terhapus	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
11	*Integration*	Mengakses *endpoint* DeleteAset () dengan parameter data yang **tidak** sesuai, **dan** layanan *rollback* ledger yang menyala	*Error*, dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-*rollback,* dan basis data aset tidak terhapus	Valid
12	*Integration*	Mengakses *endpoint* DeleteAset () dengan parameter data yang sesuai, **namun** layanan *rollback* ledger dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-*rollback* dan basis data aset tidak terhapus	Valid

4.7.5 Pengujian Layanan *Ledger*

Hasil akhir pengujian terhadap layanan *ledger* dapat dilihat pada Tabel 4.11.

Tabel 4.11 Hasil pengujian pada layanan *ledger*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi GetLedger ()				
1	*Unit*	Mengakses *endpoint* GetLedger ()	Seluruh data ledger dapat ditampilkan	Valid
B. Fungsi GetLedgerById ()				
2	*Unit*	Mengakses *endpoint* GetLedgerById () dengan parameter data yang sesuai	Seluruh data ledger yang sesuai dengan parameter dapat ditampilkan	Valid
C. Fungsi PostLedger ()				
3	Unit	Mengakses *endpoint* PostLedger () dengan parameter data yang sesuai	Data ledger pada basis data ledger bertambah	Valid
4	Unit	Mengakses *endpoint* PostLedger () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data ledger tidak bertambah	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
D. Fungsi EditLedger ()				
5	*Unit*	Mengakses *endpoint* EditLedger () dengan parameter data yang sesuai	Data pada basis data ledger dapat berubah sesuai parameter yang dimasukan	Valid
6	*Unit*	Mengakses *endpoint* EditLedger () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data ledger tidak berubah	Valid
E. Fungsi DeleteLedger ()				
7	*Unit*	Mengakses *endpoint* DeleteLedger () dengan parameter data yang sesuai	Data pada basis data *ledger* terhapus	Valid
8	*Unit*	Mengakses *endpoint* DeleteLedger () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data ledger tidak terhapus	Valid

4.7.6 Pengujian Layanan Akun

Hasil akhir pengujian terhadap layanan akun dapat dilihat pada Tabel 4.12.

Tabel 4.12 Hasil pengujian layanan akun

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi GetAkun ()				
1	*Unit*	Mengakses *endpoint* GetAkun ()	Seluruh data akun dapat ditampilkan	Valid
B. Fungsi GetAkunById ()				
2	*Unit*	Mengakses *endpoint* GetAkunById () dengan parameter data yang sesuai	Seluruh data akun yang sesuai dengan parameter dapat ditampilkan	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
C. Fungsi GetAkunByName ()				
3	*Unit*	Mengakses *endpoint* GetAkunByName () dengan parameter data yang sesuai	Seluruh data akun yang sesuai dengan parameter dapat ditampilkan	Valid
D. Fungsi PostNewAkun ()				
4	*Unit*	Mengakses *endpoint* PostNewAkun () dengan parameter data yang sesuai	Data akun pada basis data akun bertambah	Valid
5	*Unit*	Mengakses *endpoint* PostNewAkun () dengan parameter data yang **tidak** sesuai	*Error,* dan data akun pada basis data akun tidak bertambah	Valid
E. Fungsi EditAkun ()				
6	*Unit*	Mengakses *endpoint* EditAkun () dengan parameter data yang sesuai	Data akun pada basis data akun dapat berubah sesuai parameter yang dimasukan	Valid
7	*Unit*	Mengakses *endpoint* EditAkun () dengan parameter data yang **tidak** sesuai	*Error,* dan data akun pada basis data akun tidak berubah	Valid
F. Fungsi DeleteAkun ()				
8	*Unit*	Mengakses *endpoint* DeleteAkun () dengan parameter data yang sesuai	Data akun pada basis data akun terhapus	Valid
9	*Unit*	Mengakses *endpoint* DeleteAkun () dengan parameter data yang **tidak** sesuai	*Error,* dan data akun pada basis data akun tidak terhapus	Valid
G. Fungsi ChangeTargetNominal ()				
10	*Unit*	Mengakses *endpoint* ChangeTargetNominal () dengan parameter data yang sesuai	Data nominal akun pada basis data akun dapat berubah sesuai parameter yang dimasukan	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
11	*Unit*	Mengakses *endpoint* ChangeTargetNominal () dengan parameter data yang **tidak** sesuai	*Error,* dan data nominal akun pada basis data akun tidak berubah	Valid
H. Fungsi GetKategoriAkun ()				
12	*Unit*	Mengakses *endpoint* GetKategoriAkun ()	Seluruh data kategori dapat ditampilkan	Valid
I. Fungsi GetLaporanLabaRugi ()				
13	*Unit*	Mengakses *endpoint* GetLaporanLabaRugi ()	Seluruh data laporan laba rugi dapat ditampilkan	Valid

4.7.7 Pengujian Layanan Piutang

Hasil akhir pengujian terhadap layanan piutang dapat dilihat pada Tabel 4.13.

Tabel 4.13 Hasil pengujian pada layanan piutang

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreatePiutang ()				
1	*Unit*	Mengakses *endpoint* CreatePiutang () dengan parameter data yang sesuai	Data piutang pada basis data piutang bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreatePiutang () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data piutang	Valid
3	*Integration*	Mengakses *endpoint* CreatePiutang () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger bertambah, dan basis data piutang bertambah	Valid
4	*Integration*	Mengakses *endpoint* CreatePiutang () dengan parameter data yang **tidak**	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		sesuai, **dan** layanan *posting* ledger yang menyala	bertambah, dan basis data piutang tidak bertambah	
5	*Integration*	Mengakses *endpoint* CreatePiutang () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	*Error,* dan data pada basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data piutang tidak bertambah	Valid
B. Fungsi ReadPiutang ()				
6	*Unit*	Mengakses *endpoint* ReadPiutang () dengan parameter data yang sesuai	Seluruh data piutang yang sesuai dengan parameter dapat ditampilkan	Valid
C. Fungsi DeletePiutang ()				
7	*Unit*	Mengakses *endpoint* DeletePiutang () dengan parameter data yang sesuai	Data piutang pada basis data aset terhapus	Valid
8	*Unit*	Mengakses *endpoint* DeletePiutang () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data piutang tidak terhapus	Valid
9	*Integration*	Mengakses *endpoint* DeletePiutang () dengan parameter data yang sesuai, **dan** layanan *rollback* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger ter-*rollback,* dan basis data piutang terhapus	Valid
10	*Integration*	Mengakses *endpoint* DeletePiutang () dengan parameter data yang **tidak** sesuai, **dan** layanan *rollback* ledger yang menyala	*Error*, dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-*rollback* dan basis data piutang tidak terhapus	Valid
11	*Integration*	Mengakses *endpoint* DeletePiutang () dengan parameter data yang sesuai,	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		namun layanan *rollback* ledger dimatikan	*rollback* dan basis data piutang tidak terhapus	
D. Fungsi LunaskanCicilanPiutang ()				
12	*Unit*	Mengakses *endpoint* LunaskanCicilanPiutang () dengan parameter data yang sesuai	Data cicilan piutang yang sesuai dengan parameter statusnya menjadi lunas	Valid
13	*Unit*	Mengakses *endpoint* LunaskanCicilanPiutang () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data cicilan piutang yang berubah statusnya	Valid
14	*Integration*	Mengakses *endpoint* Edit LunaskanCicilanPiutang Akun () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger bertambah, dan cicilan piutang yang sesuai dengan parameter pada basis data piutang statusnya menjadi lunas	Valid
15	*Integration*	Mengakses *endpoint* LunaskanCicilanPiutang () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	Data pada: basis data akun tidak berubah, basis data ledger tidak bertambah, dan tidak ada data cicilan piutang pada basis data piutang yang berubah statusnya	Valid

4.7.8 Pengujian Layanan Hutang

Hasil akhir pengujian terhadap layanan hutang dapat dilihat pada Tabel 4.14.

Tabel 4.14 Hasil pengujian layanan hutang

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreateHutang ()				
1	*Unit*	Mengakses *endpoint* CreateHutang () dengan parameter data yang sesuai	Data piutang pada basis data hutang bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreateHutang () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data hutang	Valid
3	*Integration*	Mengakses *endpoint* CreateHutang () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data *ledger* bertambah, dan basis data hutang bertambah	Valid
4	*Integration*	Mengakses *endpoint* CreateHutang () dengan parameter data yang **tidak** sesuai, **dan** layanan *posting* ledger yang menyala	*Error,* dan data pada: basis data akun tidak berubah, basis data *ledger* tidak bertambah, dan basis data hutang tidak bertambah	Valid
5	*Integration*	Mengakses *endpoint* CreateHutang () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	*Error,* dan data pada basis data akun tidak berubah, basis data *ledger* tidak bertambah, dan basis data hutang tidak bertambah	Valid
B. Fungsi ReadHutang ()				
6	*Unit*	Mengakses endpoint ReadHutang () dengan parameter data yang sesuai	Seluruh data hutang yang sesuai dengan parameter dapat ditampilkan	Valid
C. Fungsi DeleteHutang ()				
7	*Unit*	Mengakses *endpoint* DeleteHutang () dengan parameter data yang sesuai	Data hutang pada basis data aset terhapus	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
8	*Unit*	Mengakses *endpoint* DeleteHutang () dengan parameter data yang **tidak** sesuai	*Error,* dan data pada basis data hutang tidak terhapus	Valid
9	*Integration*	Mengakses *endpoint* DeleteHutang () dengan parameter data yang sesuai, **dan** layanan *rollback* ledger yang menyala	Data pada: basis data akun berubah, basis data *ledger* ter-*rollback,* dan basis data hutang terhapus	Valid
10	*Integration*	Mengakses *endpoint* DeleteHutang () dengan parameter data yang **tidak** sesuai, **dan** layanan *rollback* ledger yang menyala	*Error*, dan data pada: basis data akun tidak berubah, basis data *ledger* tidak ter-*rollback* dan basis data hutang tidak terhapus	Valid
11	*Integration*	Mengakses *endpoint* DeleteHutang () dengan parameter data yang sesuai, **namun** layanan *rollback* ledger dimatikan	*Error,* dan data pada: basis data akun tidak berubah, basis data *ledger* tidak ter-*rollback* dan basis data hutang tidak terhapus	Valid
D. Fungsi LunaskanCicilanHutang ()				
12	*Unit*	Mengakses *endpoint* LunaskanCicilanHutang () dengan parameter data yang sesuai	Data cicilan hutang yang sesuai dengan parameter statusnya menjadi lunas	Valid
13	*Unit*	Mengakses *endpoint* LunaskanCicilanHutang () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data cicilan hutang yang berubah statusnya	Valid
14	*Integration*	Mengakses *endpoint* LunaskanCicilanHutang () dengan parameter data yang	Data pada: basis data akun berubah, basis data ledger bertambah, dan cicilan hutang yang sesuai dengan	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		sesuai, **dan** layanan *posting* ledger yang menyala	parameter pada basis data hutang statusnya menjadi lunas	
15	*Integration*	Mengakses *endpoint* LunaskanCicilanHutang () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	Data pada: basis data akun tidak berubah, basis data ledger tidak bertambah, dan tidak ada data cicilan hutang pada basis data hutang yang berubah statusnya	Valid

4.7.9 Pengujian Layanan *Invoice*

Hasil akhir pengujian terhadap layanan *invoice* dapat dilihat pada Tabel 4.15.

Tabel 4.15 Hasil pengujian layanan *invoice*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
E. Fungsi CreateInvoice ()				
1	*Unit*	Mengakses *endpoint* CreateInvoice () dengan parameter data yang sesuai	Data *invoice* pada basis data *invoice* bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreateInvoice () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data *invoice*	Valid
F. Fungsi ReadInvoice ()				
3	*Unit*	Mengakses endpoint ReadInvoice () dengan parameter data yang sesuai	Seluruh data *invoice* yang sesuai dengan parameter dapat ditampilkan	Valid
G. Fungsi DeleteInvoice ()				
4	*Unit*	Mengakses *endpoint* DeleteInvoice () dengan parameter data yang sesuai	Data *invoice* pada basis data *invoice* terhapus	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
5	*Unit*	Mengakses *endpoint* DeleteInvoice () dengan parameter data yang **tidak** sesuai	*Error,* dan data *invoice* pada basis data *invoice* tidak terhapus	Valid

4.7.10 Pengujian Layanan Arus Kas

Hasil akhir pengujian terhadap layanan *invoice* dapat dilihat pada Tabel 4.16.

Tabel 4.16 Hasil pengujian layanan arus kas

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreateArusKas ()				
1	*Unit*	Mengakses *endpoint* CreateArusKas () dengan parameter data yang sesuai	Data kas pada basis data aset bertambah	Valid
2	*Unit*	Mengakses *endpoint* CreateArusKas () dengan parameter data yang **tidak** sesuai	*Error,* dan tidak ada data yang masuk ke basis data kas	Valid
3	*Integration*	Mengakses *endpoint* CreateArusKas () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger bertambah, dan basis data kas bertambah	Valid
4	*Integration*	Mengakses *endpoint* CreateArusKas () dengan parameter data yang **tidak** sesuai, **dan** layanan *posting* ledger yang menyala	*Error,* dan data pada: basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data kas tidak bertambah	Valid
5	*Integration*	Mengakses *endpoint* CreateAset () dengan parameter data yang sesuai, **namun** layanan *posting* ledger dimatikan	*Error,* dan data pada basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data kas tidak bertambah	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
6	*Integration*	Mengakses *endpoint* CreateArusKas () dengan parameter data yang sesuai, **dan** layanan *posting* ledger yang menyala, **namun** saldo akun rekening tidak cukup untuk melakukan pembelian aset baru	*Error,* dan data pada basis data akun tidak berubah, basis data ledger tidak bertambah, dan basis data kas tidak bertambah	Valid
B. Fungsi ReadArusKas ()				
7	*Unit*	Mengakses endpoint ReadArusKas () dengan parameter data yang sesuai	Seluruh data kas yang sesuai dengan parameter dapat ditampilkan	Valid
C. Fungsi DeleteArusKas ()				
8	*Unit*	Mengakses *endpoint* DeleteArusKas () dengan parameter data yang sesuai	Data kas pada basis data kas terhapus	Valid
9	*Unit*	Mengakses *endpoint* DeleteArusKas () dengan parameter data yang **tidak** sesuai	*Error,* dan data kas pada basis data kas tidak terhapus	Valid
10	*Integration*	Mengakses *endpoint* DeleteArusKas () dengan parameter data yang sesuai, **dan** layanan *rollback* ledger yang menyala	Data pada: basis data akun berubah, basis data ledger ter-*rollback,* dan basis data kas terhapus	Valid
11	*Integration*	Mengakses *endpoint* DeleteArusKas () dengan parameter data yang **tidak** sesuai, **dan** layanan *rollback* ledger yang menyala	*Error*, dan data pada: basis data akun tidak berubah, basis data ledger tidak ter-*rollback,* dan basis data kas tidak terhapus	Valid
12	*Integration*	Mengakses *endpoint* DeleteArusKas () dengan	*Error,* dan data pada: basis data akun tidak berubah,	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		parameter data yang sesuai, **namun** layanan *rollback* ledger dimatikan	basis data ledger tidak ter-*rollback* dan basis data kas tidak terhapus	

4.8 Penerapan Antarmuka

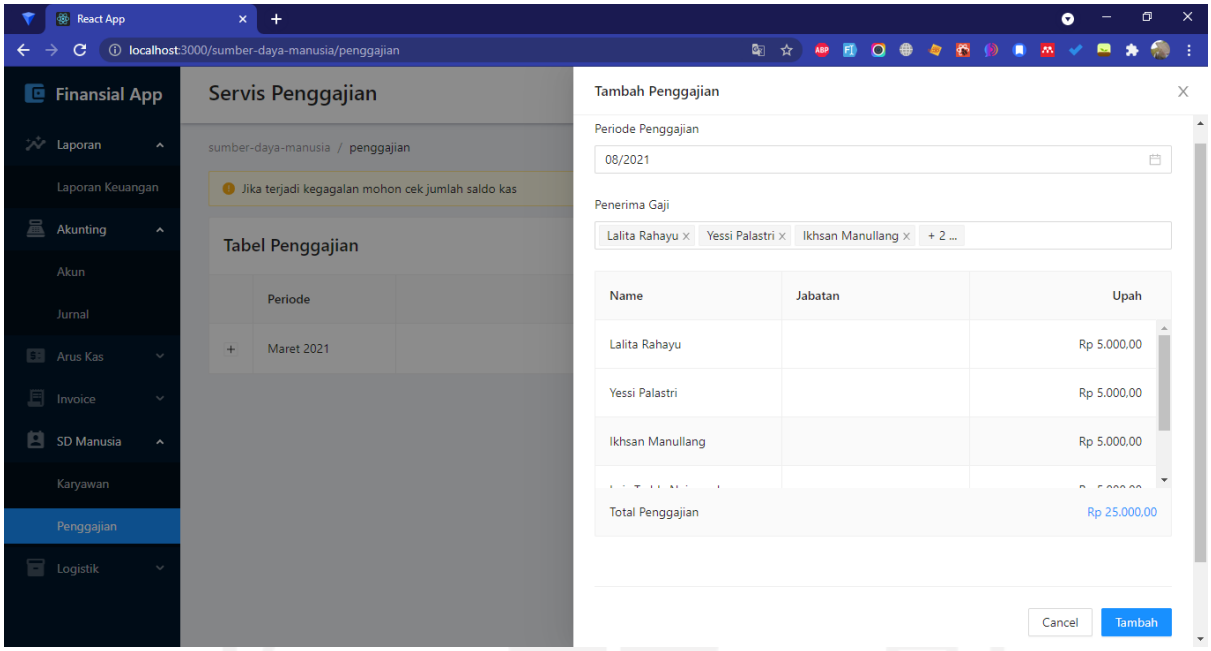
Setelah seluruh API layanan lolos dari tahap pengujian, selanjutnya dilakukan proses penerapan antarmuka. Seperti yang sudah dijelaskan pada bab 3.8, antarmuka diterapkan pada arsitektur monolitik dan menggunakan ReactJS.

4.8.1 Penerapan Halaman Penggajian

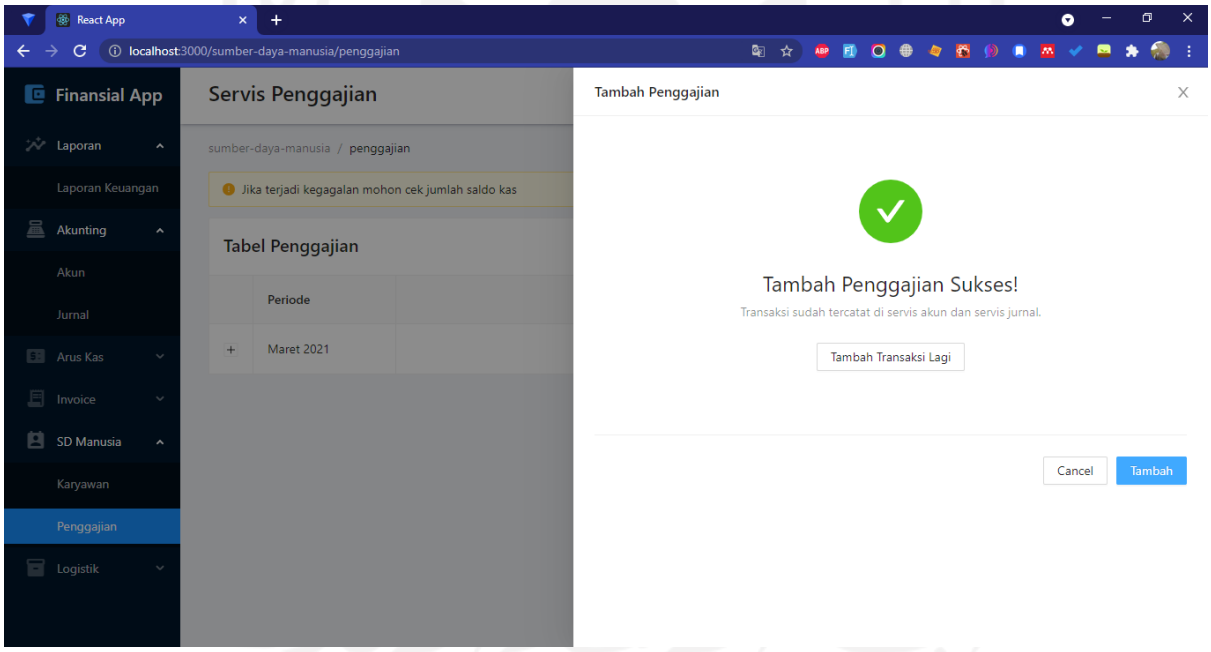
Pada halaman penggajian ini terdapat empat fungsi yang diterapkan pada antarmuka, yakni: CreatePenggajian (), ReadPenggajian (), DeletePenggajian (), ReadKaryawan ().

A. Fungsi CreatePenggajian ()

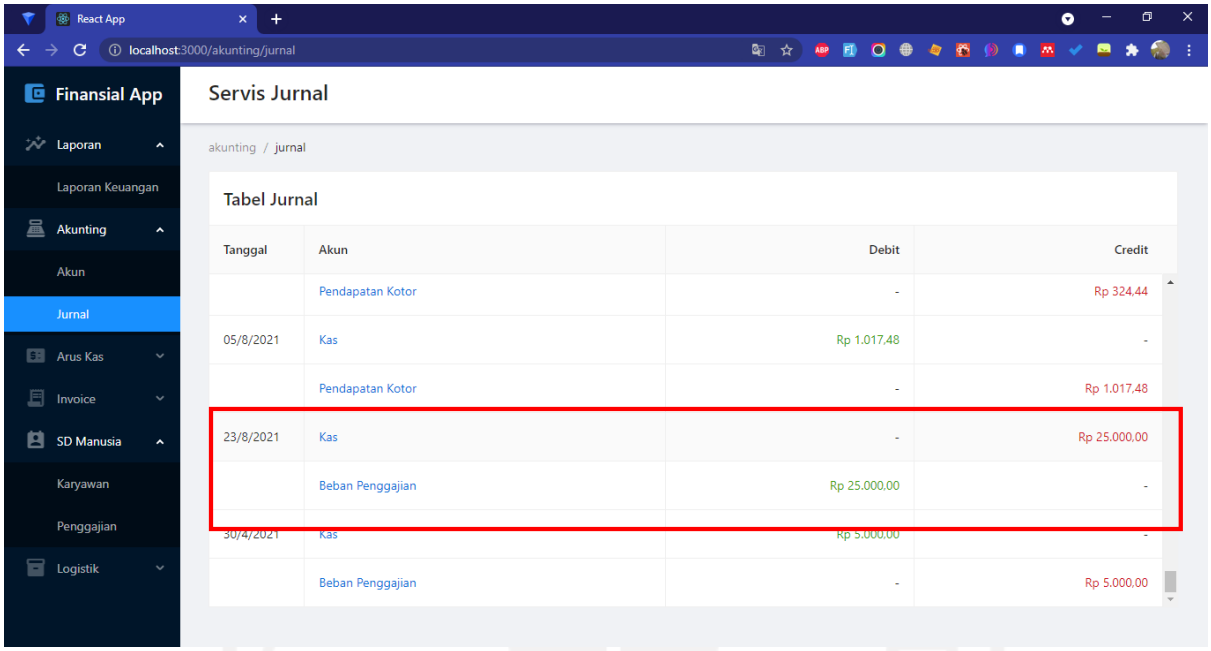
Gambar 4.25 merupakan tampilan untuk menambah penggajian. Kemudian, Gambar 4.26 merupakan tampilan *feedback* ketika sistem berhasil menambah data penggajian sesuai *form* yang dikirim. Selanjutnya, Gambar 4.27 merupakan tampilan nominal penggajian yang ditransfer ke layanan *ledger*.



Gambar 4.25 Tampilan form CreatePenggajian ()



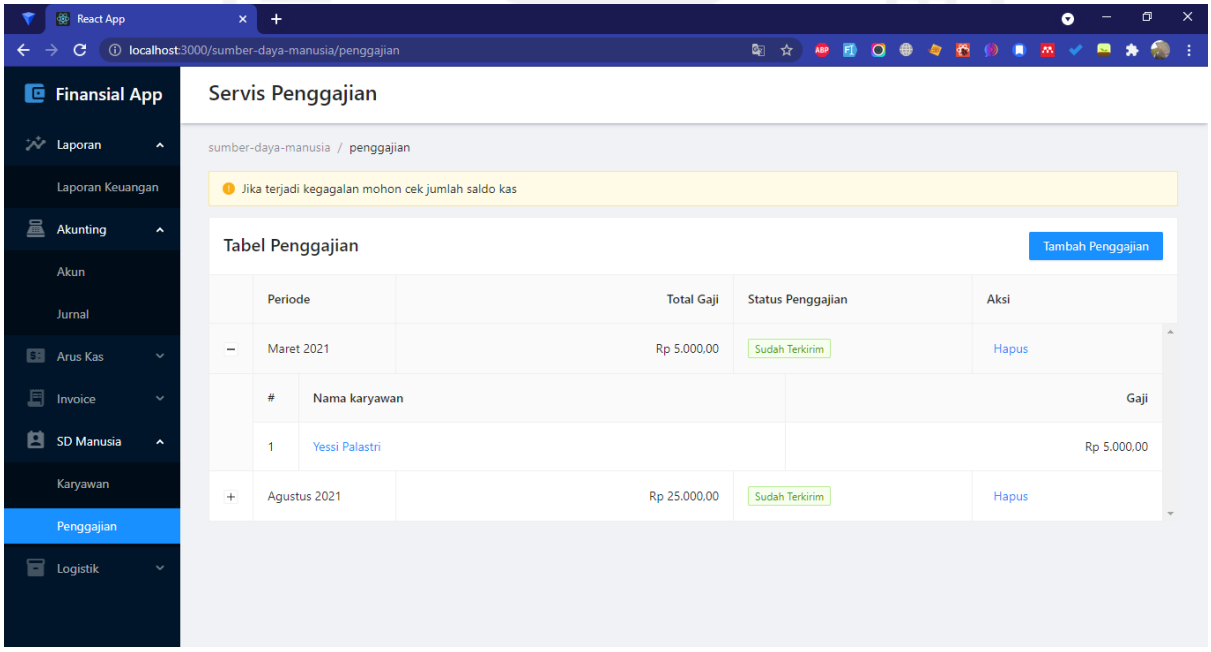
Gambar 4.26 Tampilan *feedback* ketika sistem berhasil menambah penggajian



Gambar 4.27 Tampilan transaksi penggajian yang ditransfer ke layanan *ledger*

B. Fungsi Read Penggajian ()

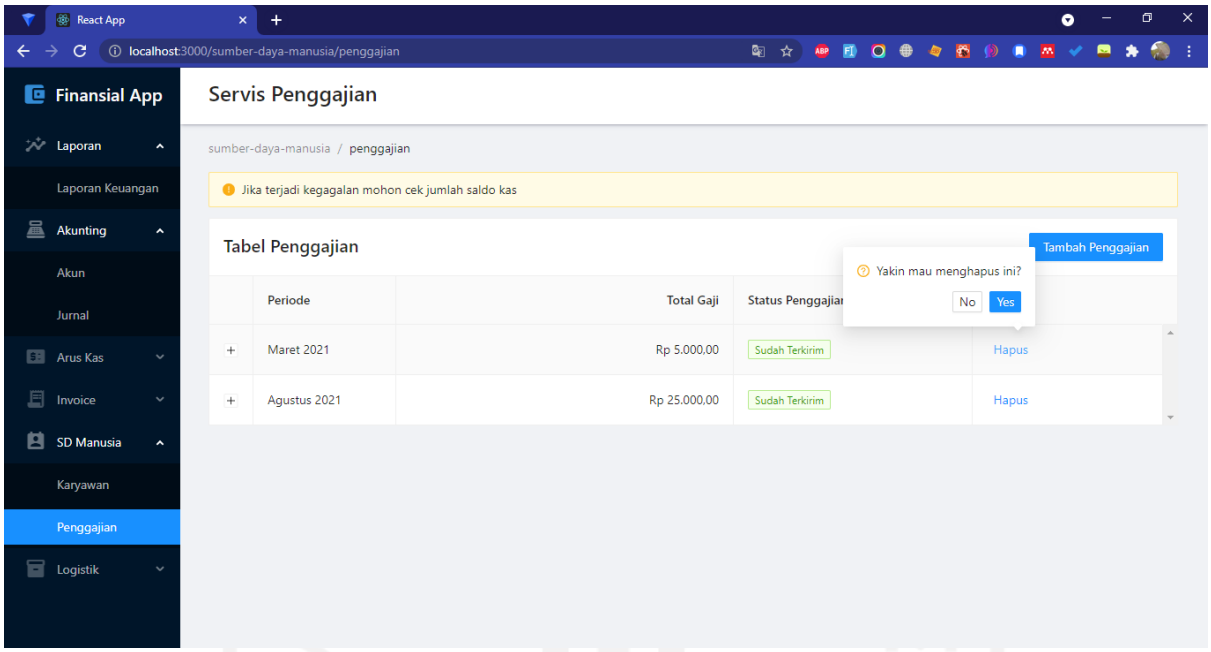
Gambar 4.28 merupakan tampilan untuk melihat seluruh data penggajian.



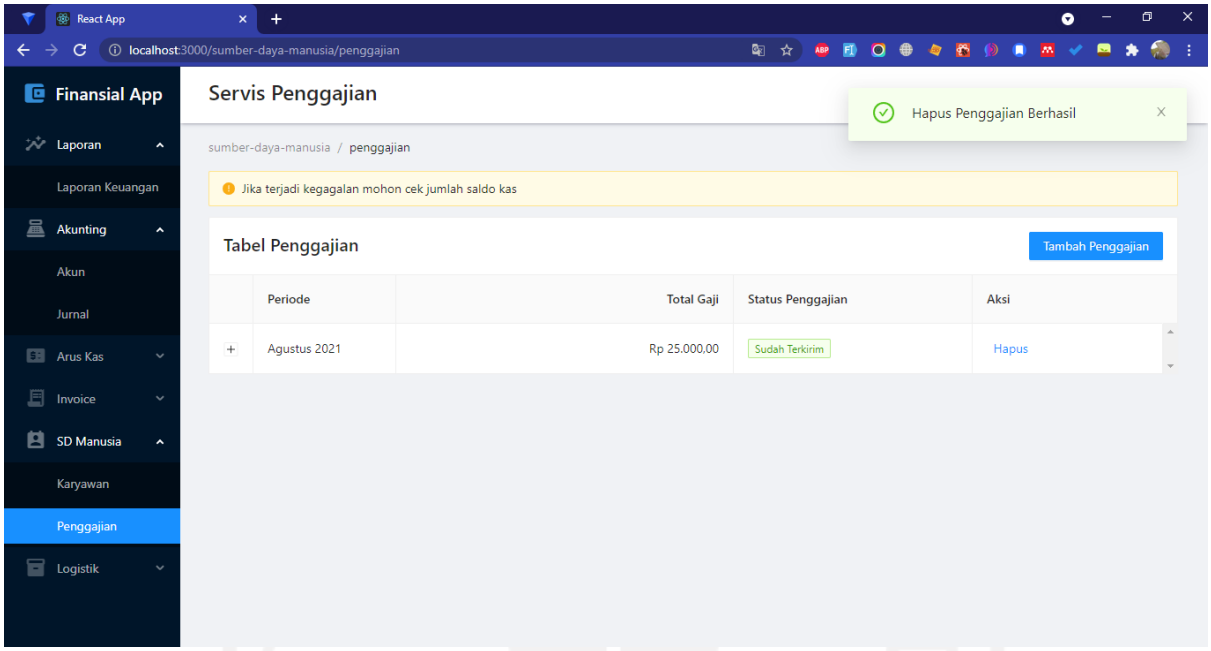
Gambar 4.28 Tampilan tabel data penggajian

C. Fungsi DeletePenggajian ()

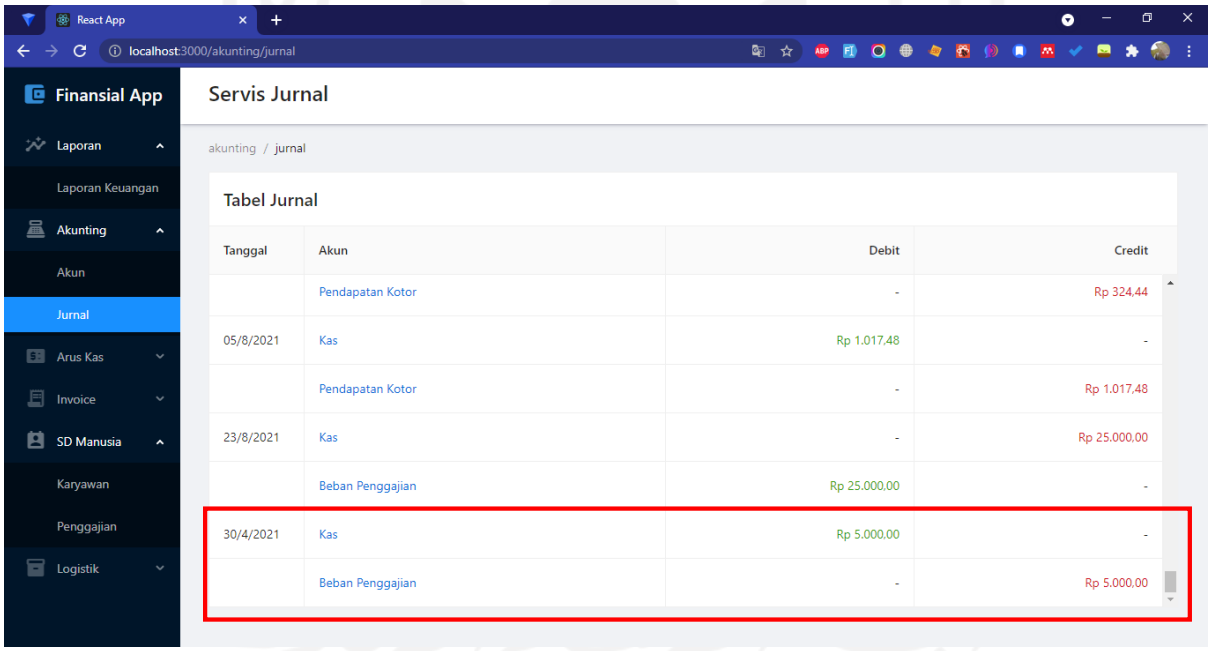
Gambar 4.29 merupakan tampilan konfirmasi ketika ingin menghapus data penggajian. Gambar 4.30 merupakan tampilan ketika muncul notifikasi terkait sistem yang berhasil menghapus data penggajian. Gambar 4.31 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada penggajian yang dihapus.



Gambar 4.29 Tampilan konfirmasi hapus data penggajian



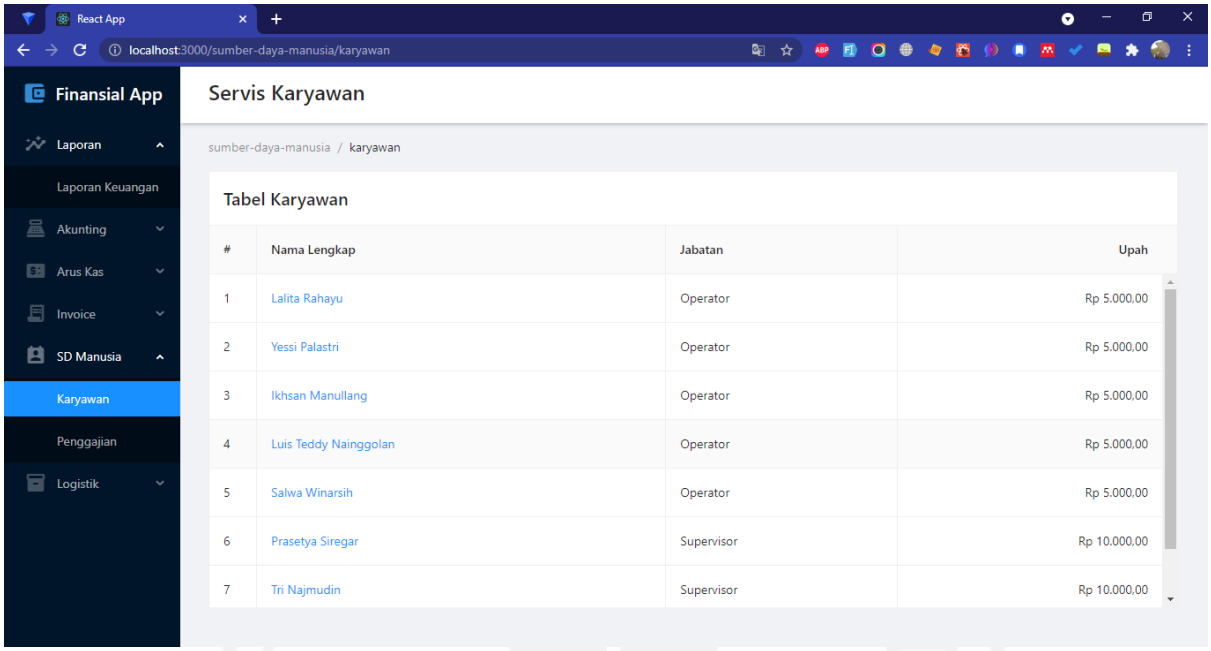
Gambar 4.30 Tampilan *feedback* ketika sistem berhasil menghapus penggajian



Gambar 4.31 Tampilan transaksi penggajian yang dihapus ditransfer ke layanan *ledger*

D. Fungsi ReadKaryawan ()

Gambar 4.32 merupakan tampilan untuk melihat seluruh data karyawan.



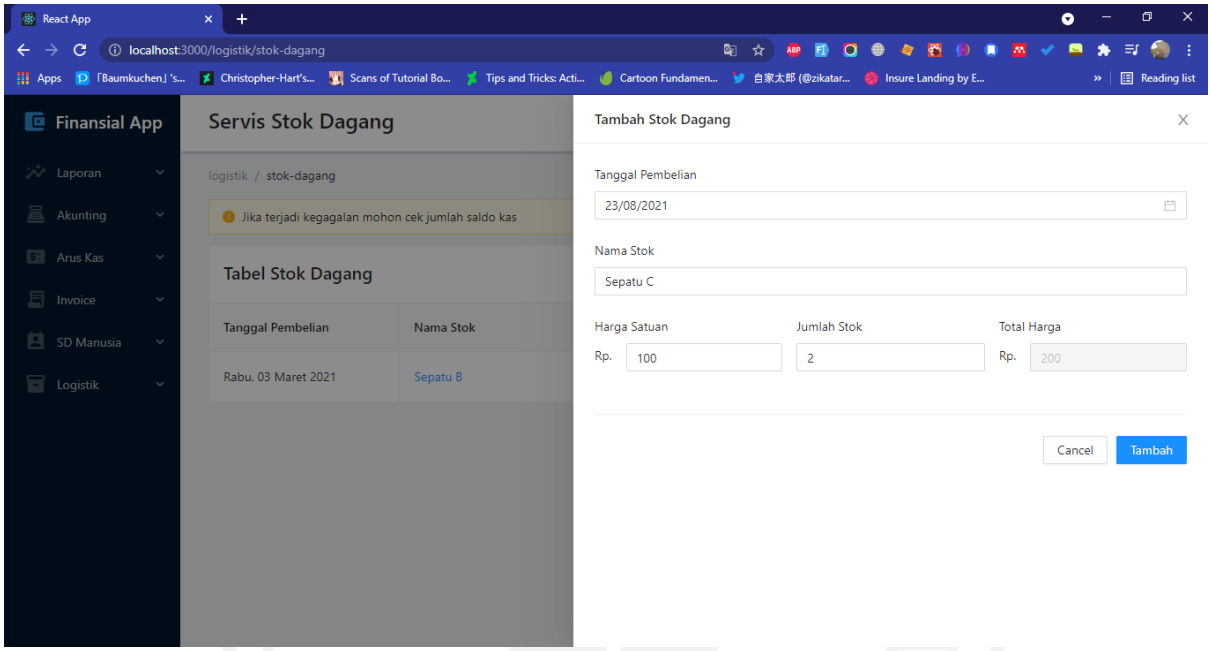
Gambar 4.32 Tampilan tabel data karyawan

4.8.2 Penerapan Halaman Aset

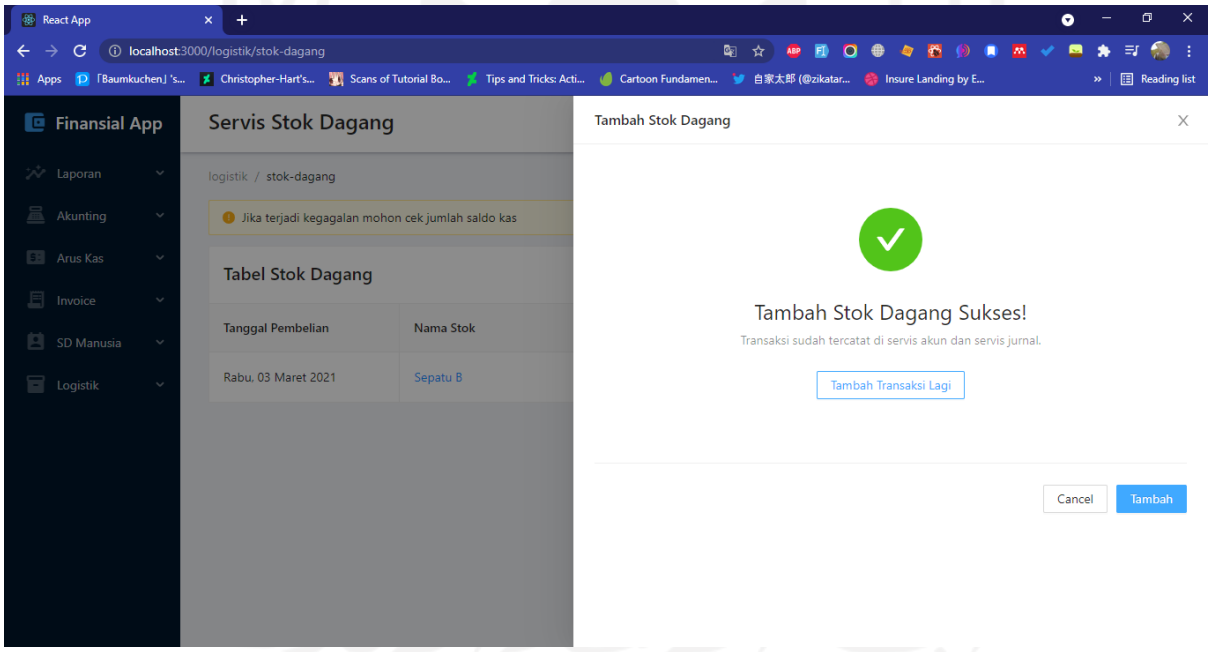
Pada halaman aset ini terdapat tiga fungsi yang diterapkan pada antarmuka, yakni: CreateAset (), ReadAset (), DeleteAset ().

A. Fungsi CreateAset ()

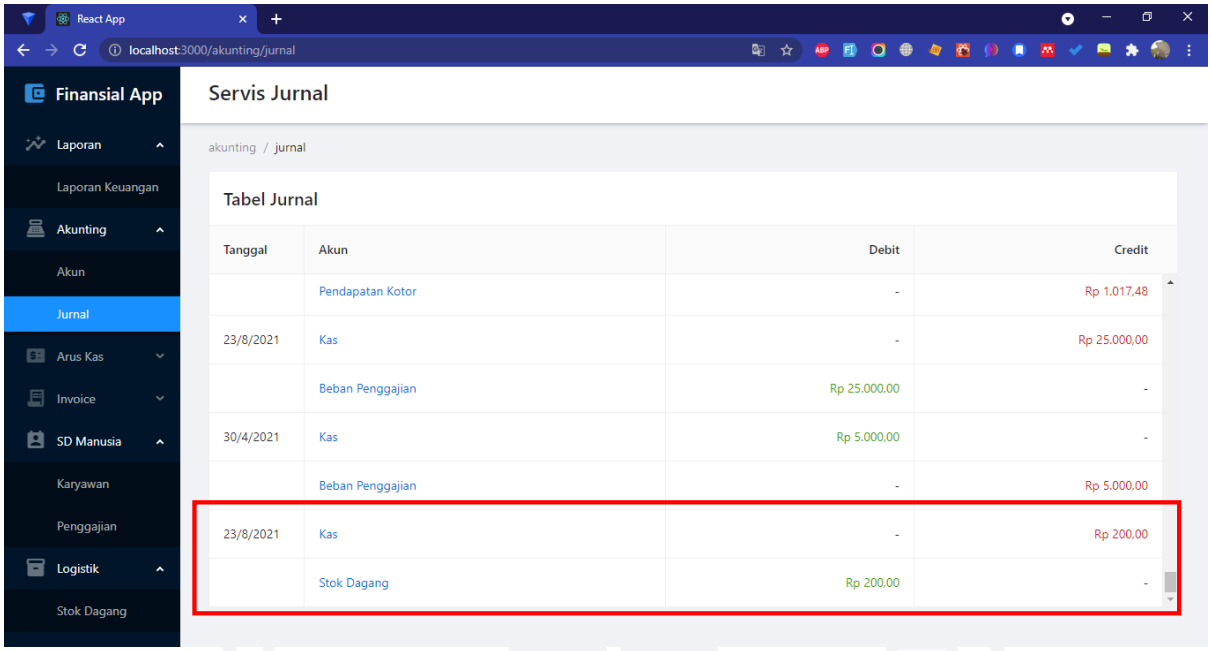
Gambar 4.33 merupakan tampilan untuk menambah aset. Kemudian, Gambar 4.34 merupakan tampilan *feedback* ketika sistem berhasil menambah aset sesuai *form* yang dikirim. Selanjutnya, Gambar 4.35 merupakan tampilan nominal aset yang ditransfer ke layanan *ledger*.



Gambar 4.33 Tampilan form CreateAset ()



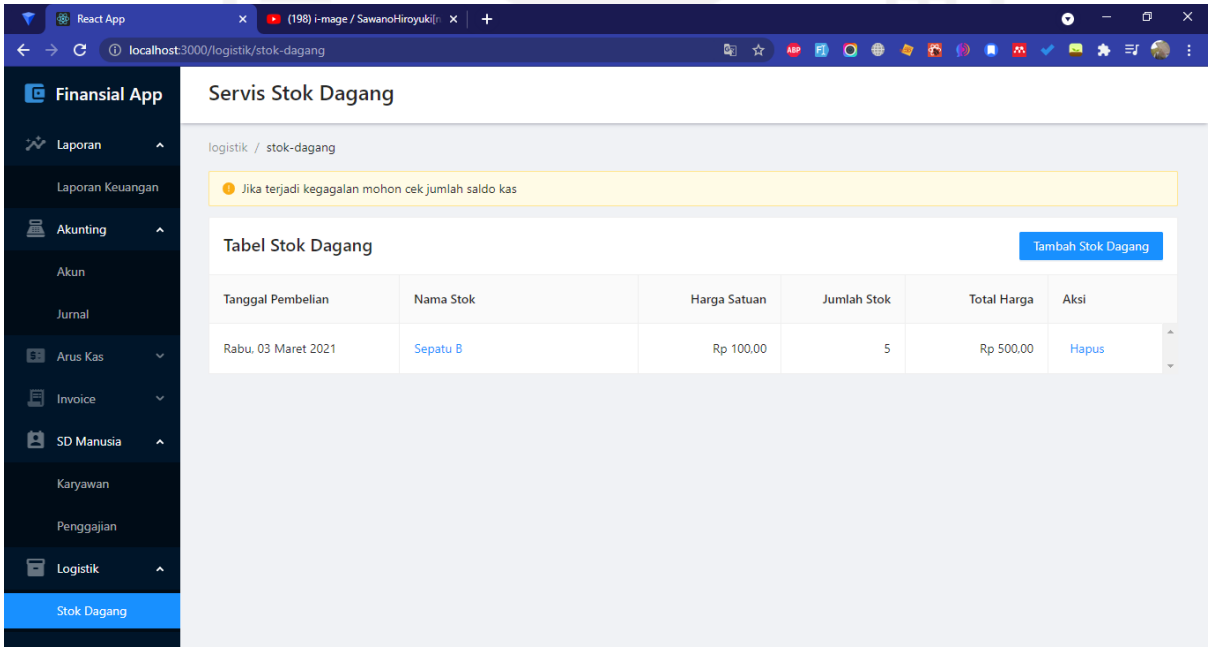
Gambar 4.34 Tampilan *feedback* ketika sistem berhasil menambah aset



Gambar 4.35 Tampilan transaksi aset yang ditransfer ke layanan *ledger*

B. Fungsi ReadAset ()

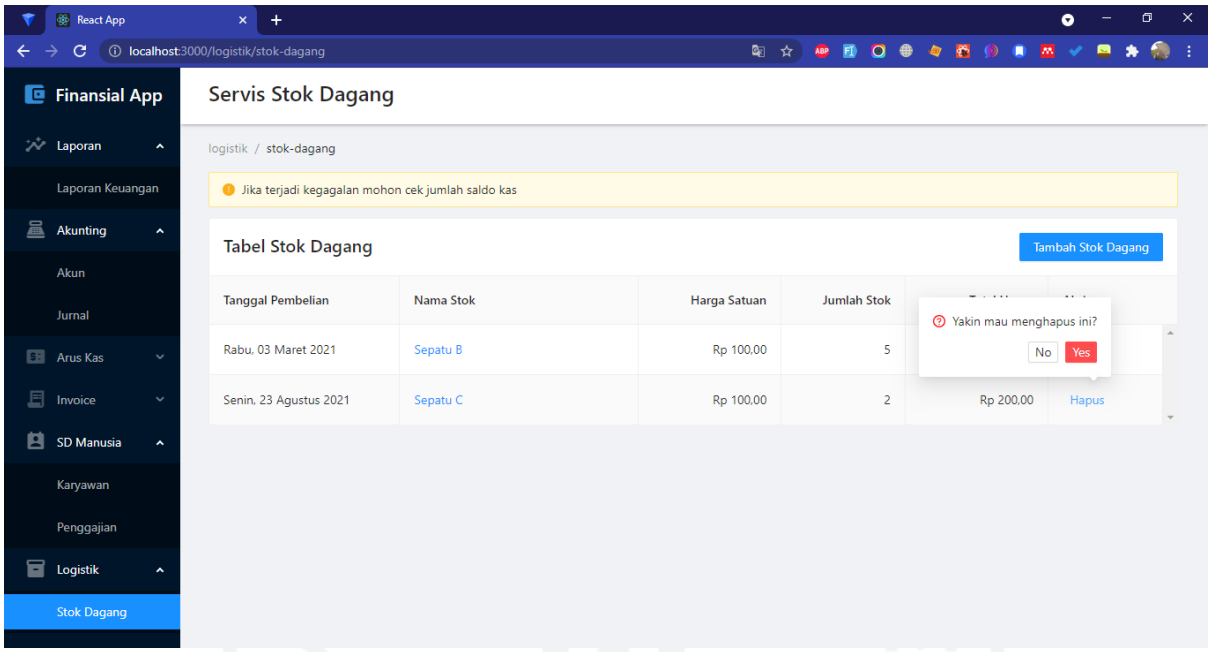
Gambar 4.36 merupakan tampilan untuk melihat seluruh data aset.



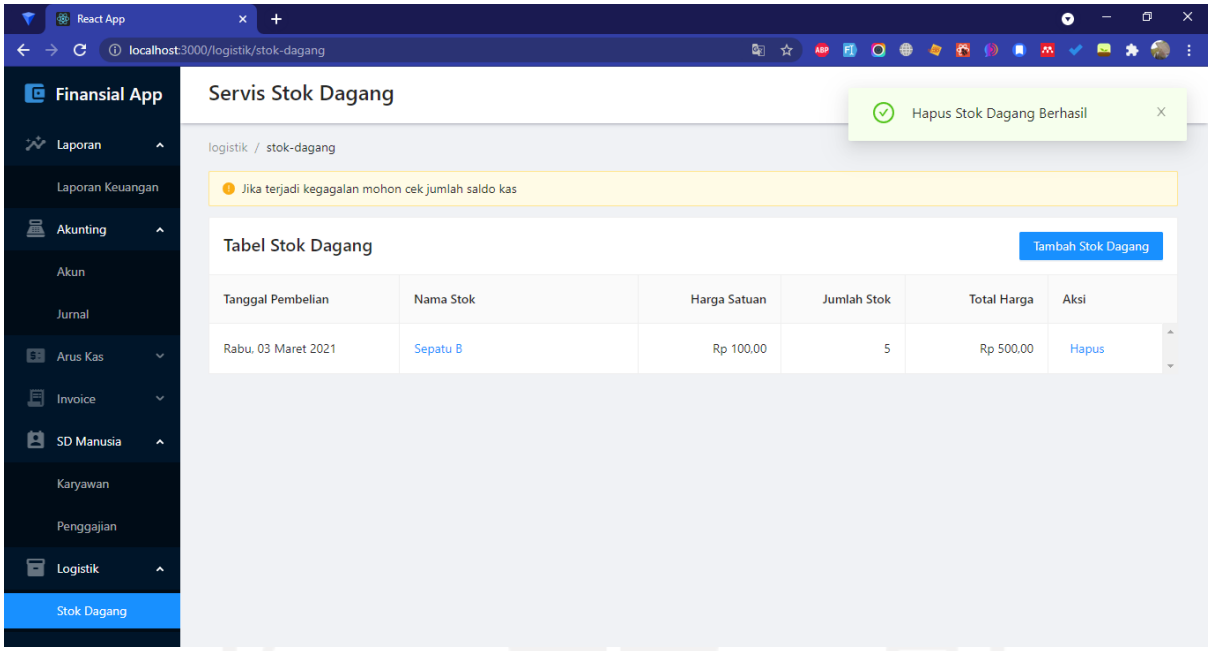
Gambar 4.36 Tampilan tabel data aset

C. Fungsi DeleteAset ()

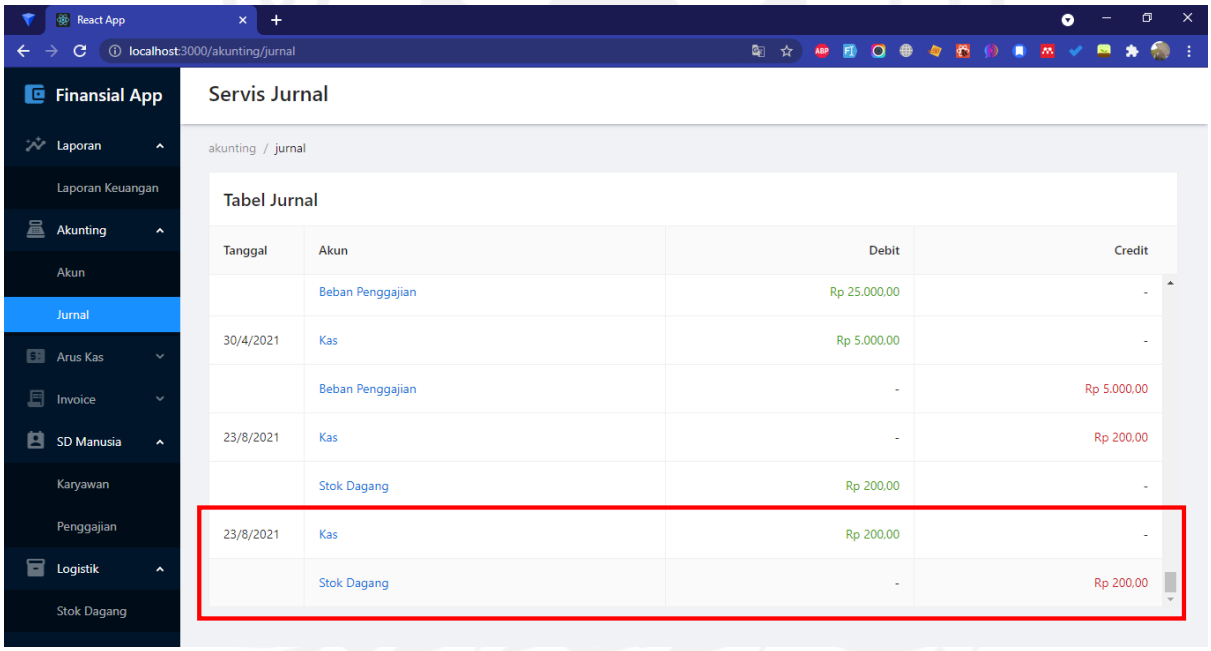
Gambar 4.37 merupakan tampilan konfirmasi ketika ingin menghapus data aset. Gambar 4.38 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil menghapus data aset. Gambar 4.39 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada aset yang dihapus.



Gambar 4.37 Tampilan konfirmasi hapus data aset



Gambar 4.38 Tampilan *feedback* ketika sistem berhasil menghapus aset



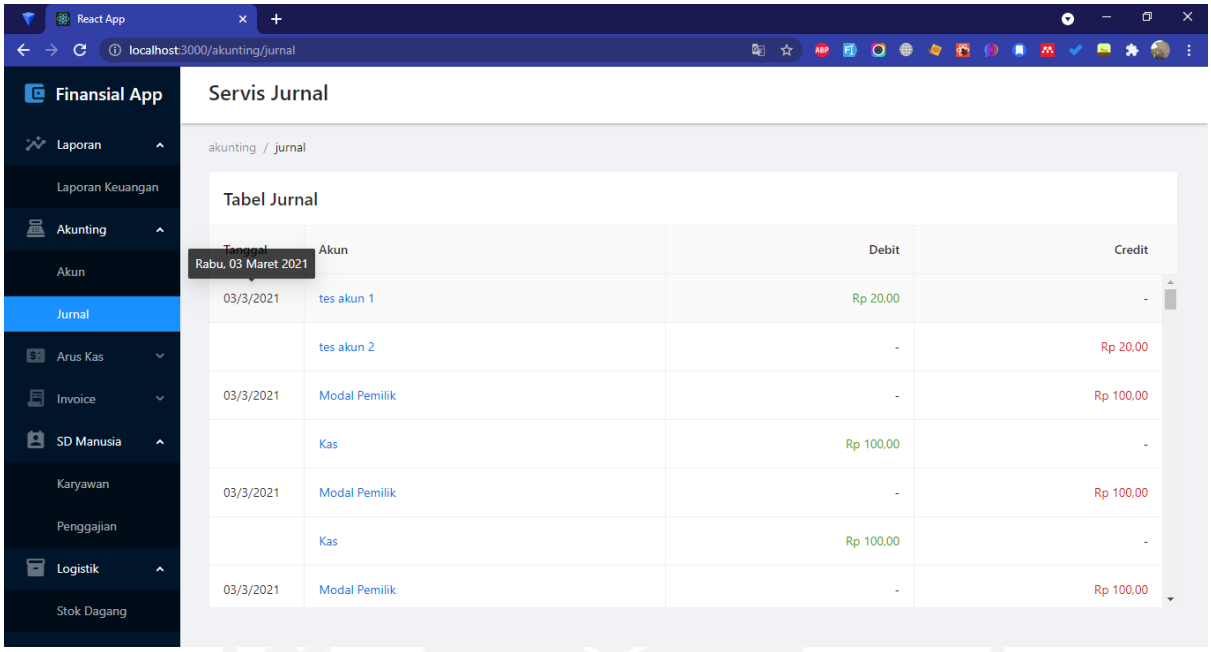
Gambar 4.39 Tampilan transaksi aset yang dihapus ditransfer ke layanan *ledger*

4.8.3 Penerapan Halaman *Ledger*

Pada halaman ledger ini terdapat satu fungsi yang diterapkan pada antarmuka, yakni: ReadLedger ().

A. Fungsi ReadLedger ()

Gambar 4.40 merupakan tampilan untuk melihat seluruh data aset.



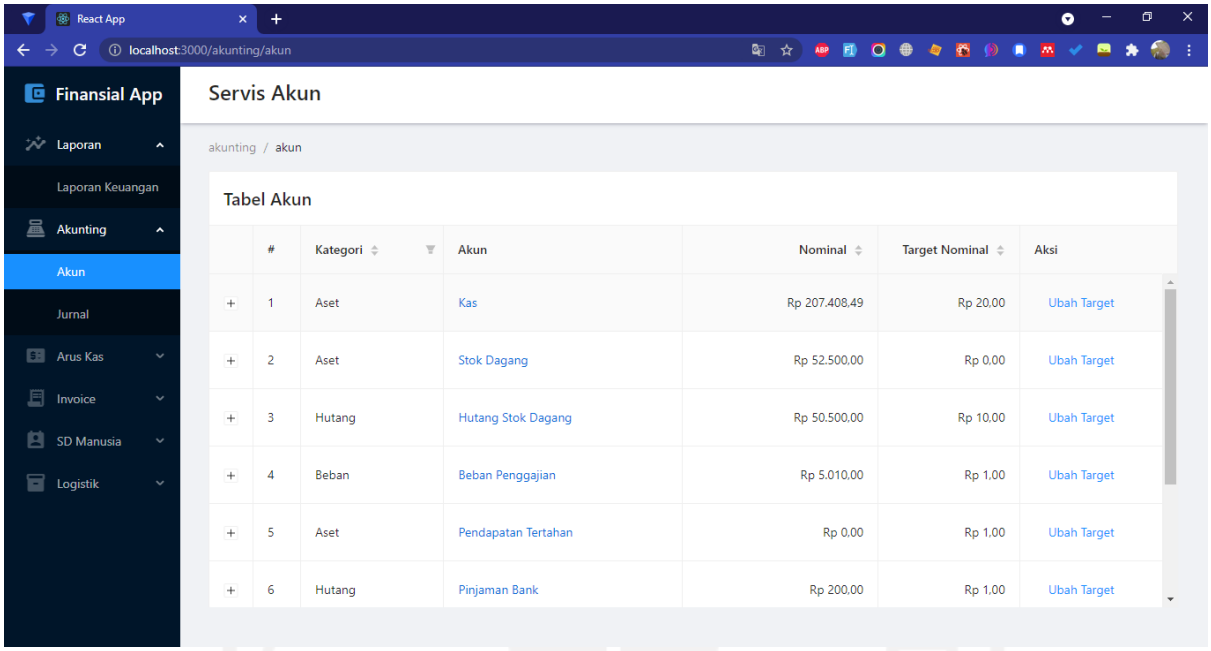
Gambar 4.40 Tampilan tabel data *ledger*

4.8.4 Penerapan Halaman Akun

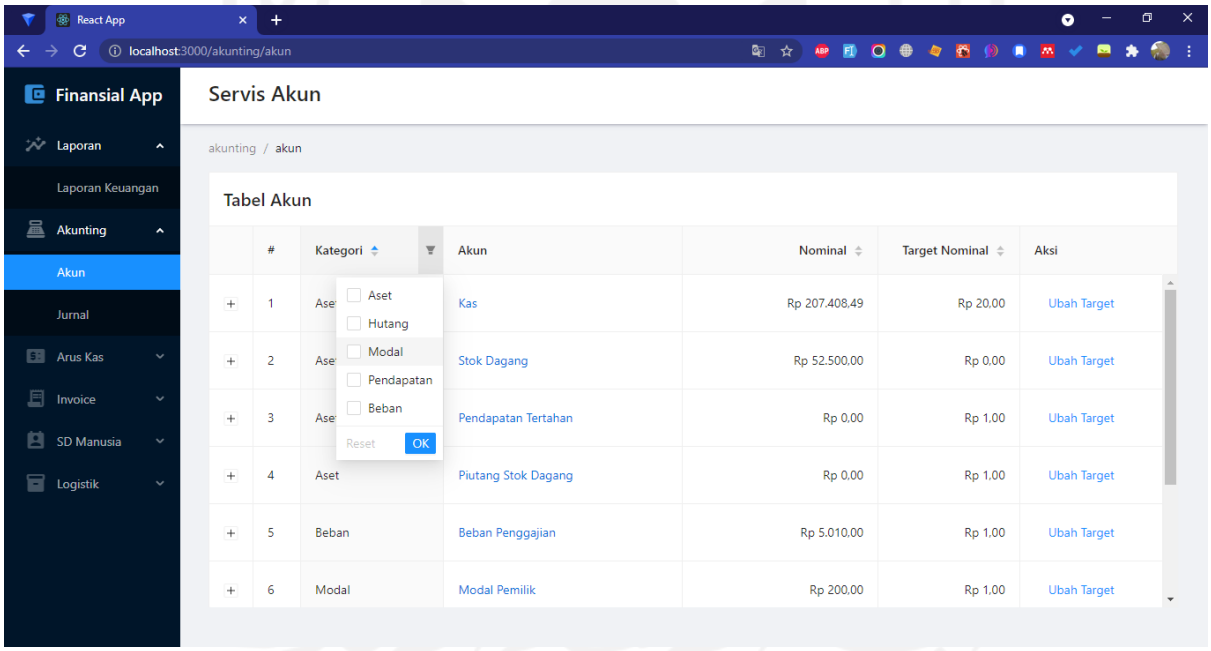
Pada halaman ledger ini terdapat tiga fungsi yang diterapkan pada antarmuka, yakni: ReadAkun (), ChangeTargetNominal (), GetLaporanKeuangan().

A. Fungsi ReadAkun ()

Gambar 4.41 merupakan tampilan untuk melihat seluruh data aset. Gambar 4.42 merupakan tampilan opsi sortir akun.



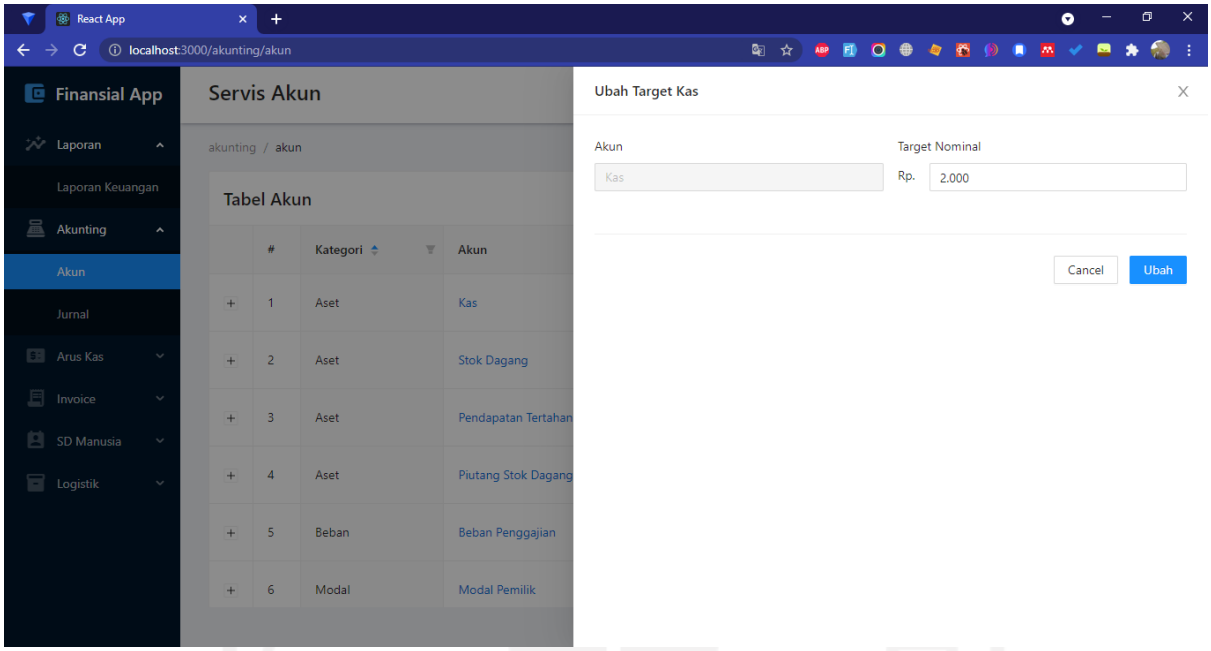
Gambar 4.41 Tampilan tabel data akun



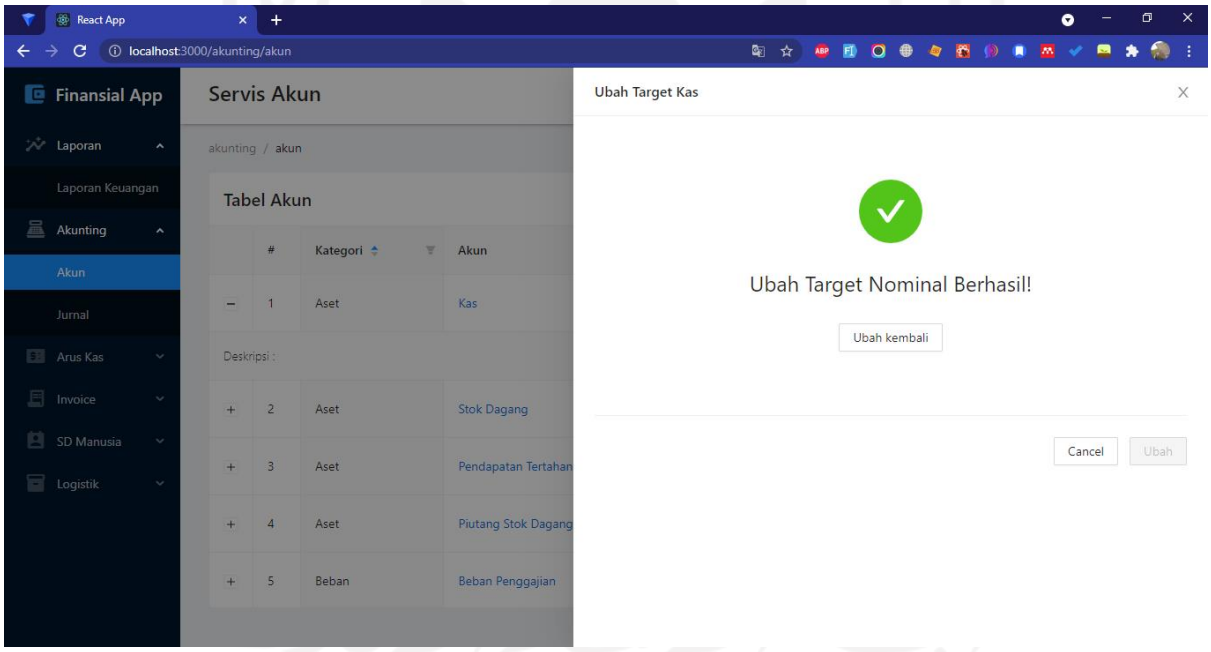
Gambar 4.42 Tampilan opsi *filter* pada tabel data akun

B. Fungsi ChangeTargetNominal ()

Gambar 4.43 merupakan tampilan ketika merubah target bulanan dari akun kas. Gambar 4.44 merupakan tampilan ketika akun kas sudah berubah.



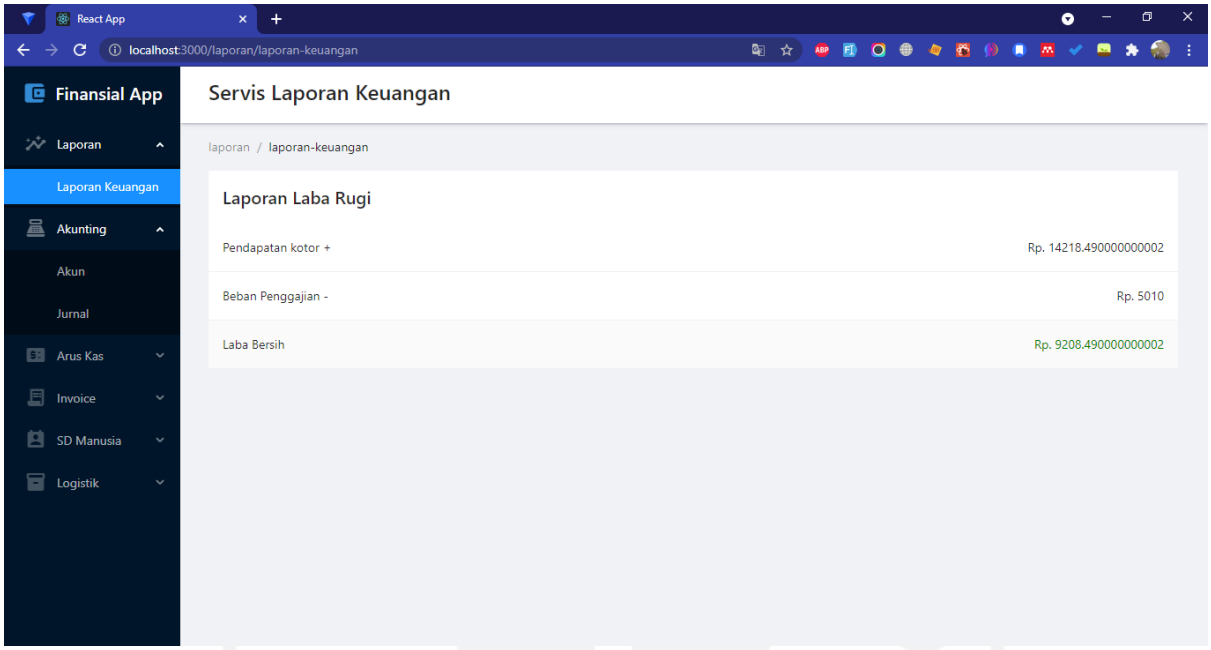
Gambar 4.43 Tampilan opsi *filter* pada tabel data akun



Gambar 4.44 Tampilan *feedback* ketika sistem berhasil mengganti target nominal

C. Fungsi GetLaporanKeuangan()

Gambar 4.45 merupakan tampilan data laporan laba rugi.



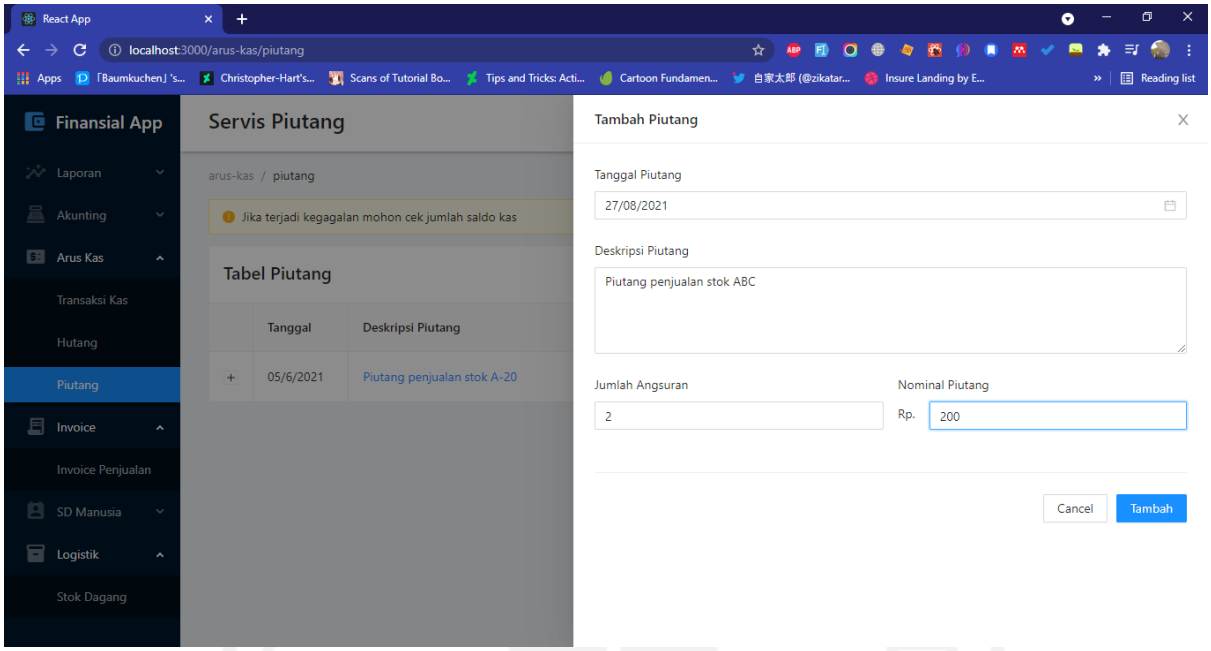
Gambar 4.45 Tampilan data laporan laba rugi

4.8.5 Penerapan Halaman Piutang

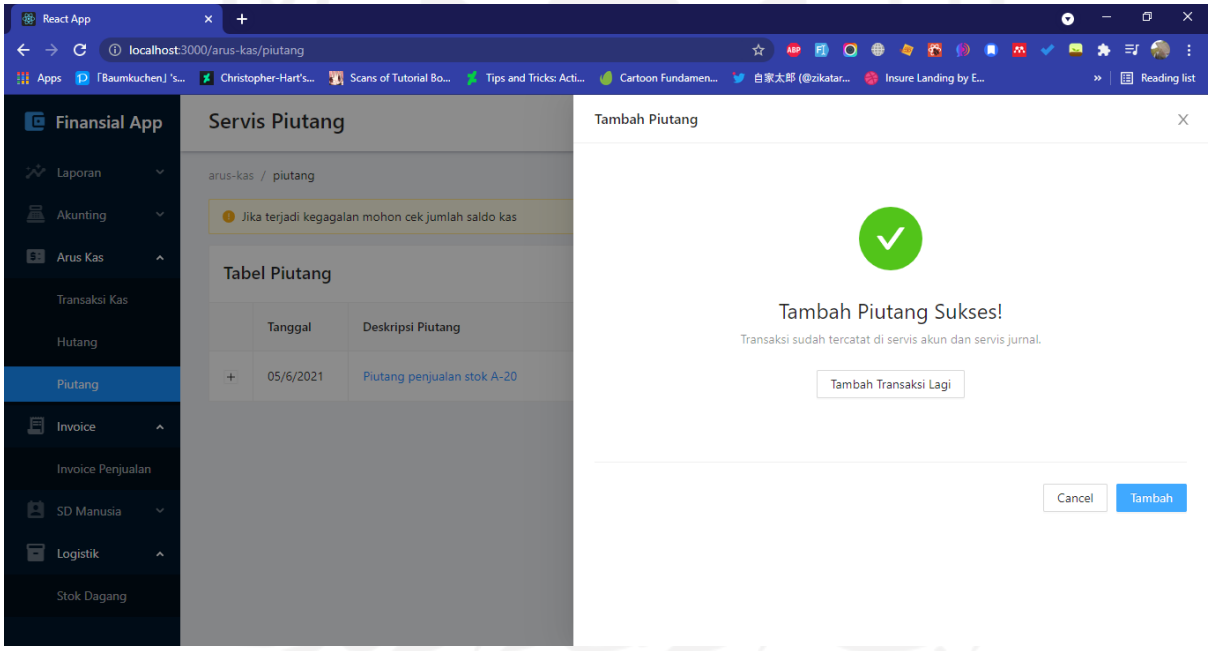
Pada halaman ledger ini terdapat empat fungsi yang diterapkan pada antarmuka, yakni: CreatePiutang (), ReadPiutang (), DeletePiutang (), LunaskanCicilanPiutang ().

A. Fungsi CreatePiutang ()

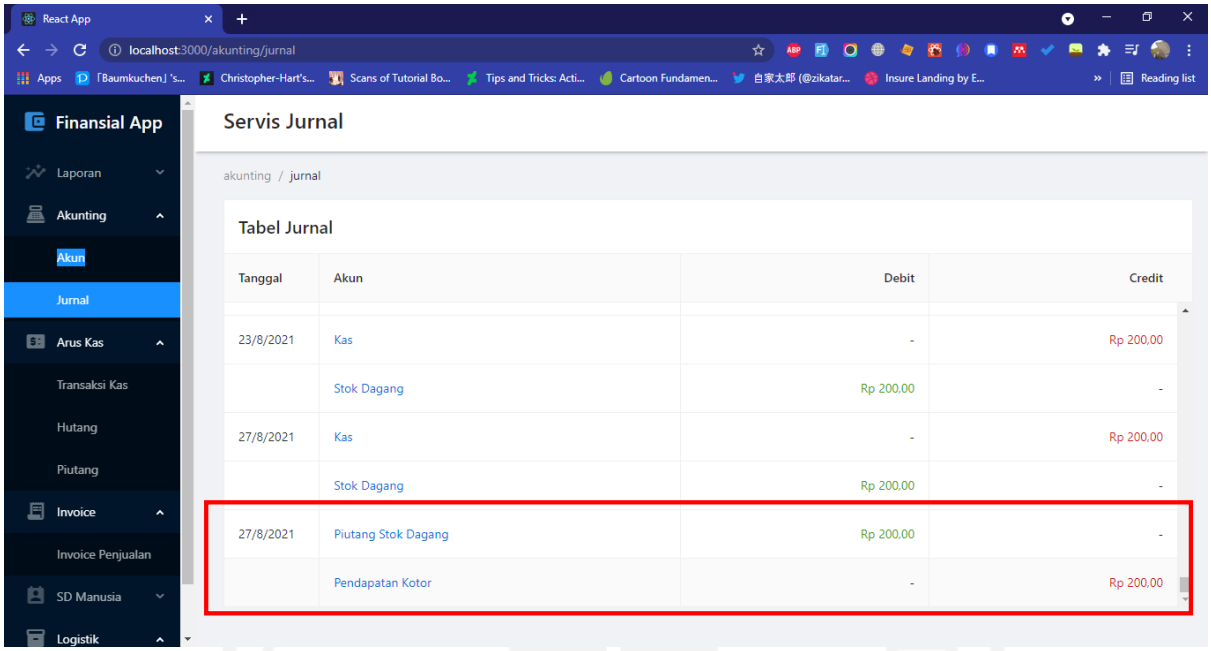
Gambar 4.46 merupakan tampilan untuk menambah piutang. Kemudian, Gambar 4.47 merupakan tampilan *feedback* ketika sistem berhasil menambah piutang sesuai *form* yang dikirim. Selanjutnya, Gambar 4.48 merupakan tampilan nominal piutang yang ditransfer ke layanan *ledger*.



Gambar 4.46 Tampilan form CreatePiutang ()



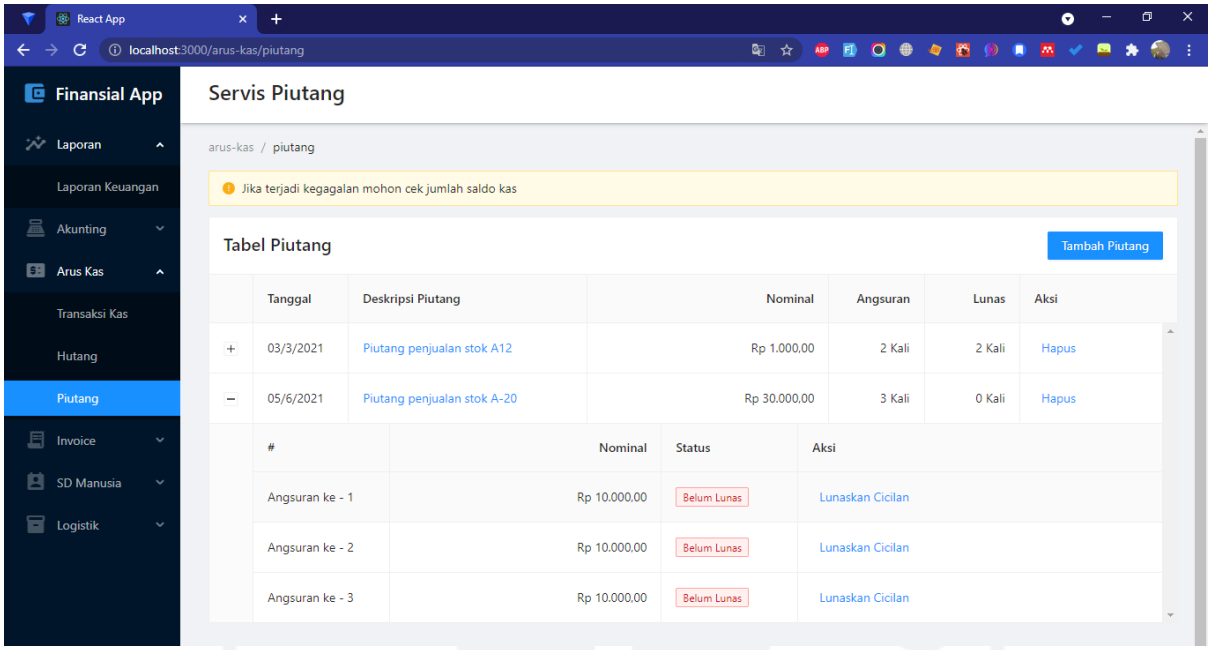
Gambar 4.47 Tampilan *feedback* ketika sistem berhasil menambah piutang



Gambar 4.48 Tampilan transaksi piutang yang ditransfer ke layanan *ledger*

B. Fungsi ReadPiutang ()

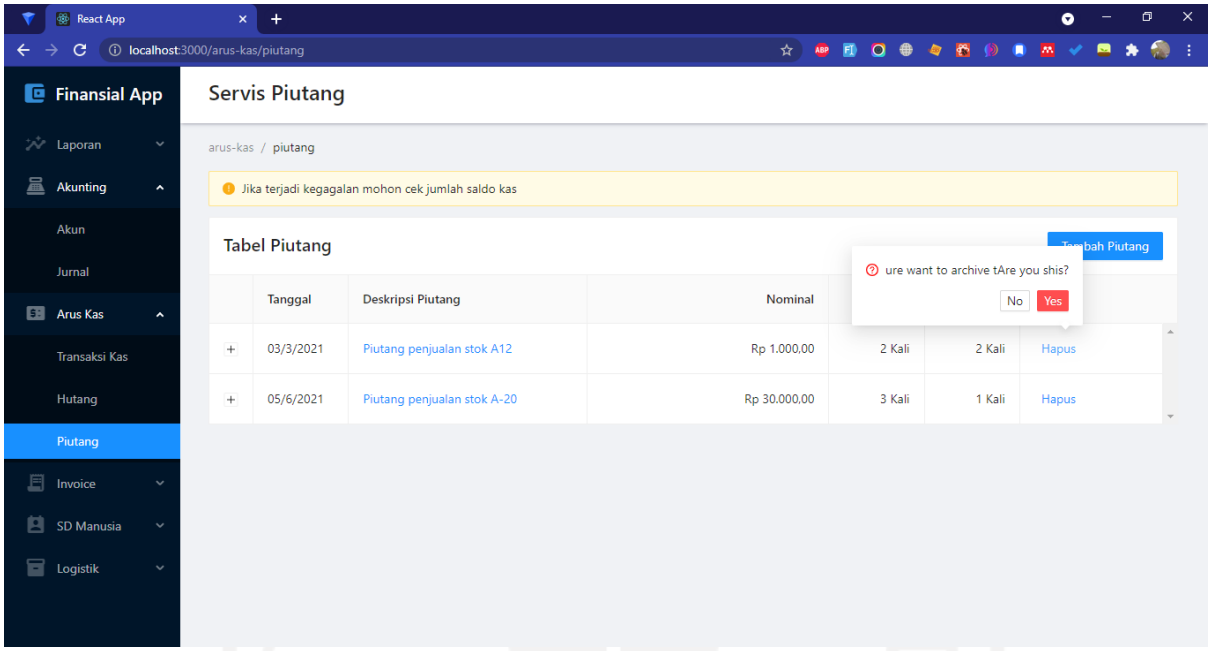
Gambar 4.36 merupakan tampilan untuk melihat seluruh data aset.



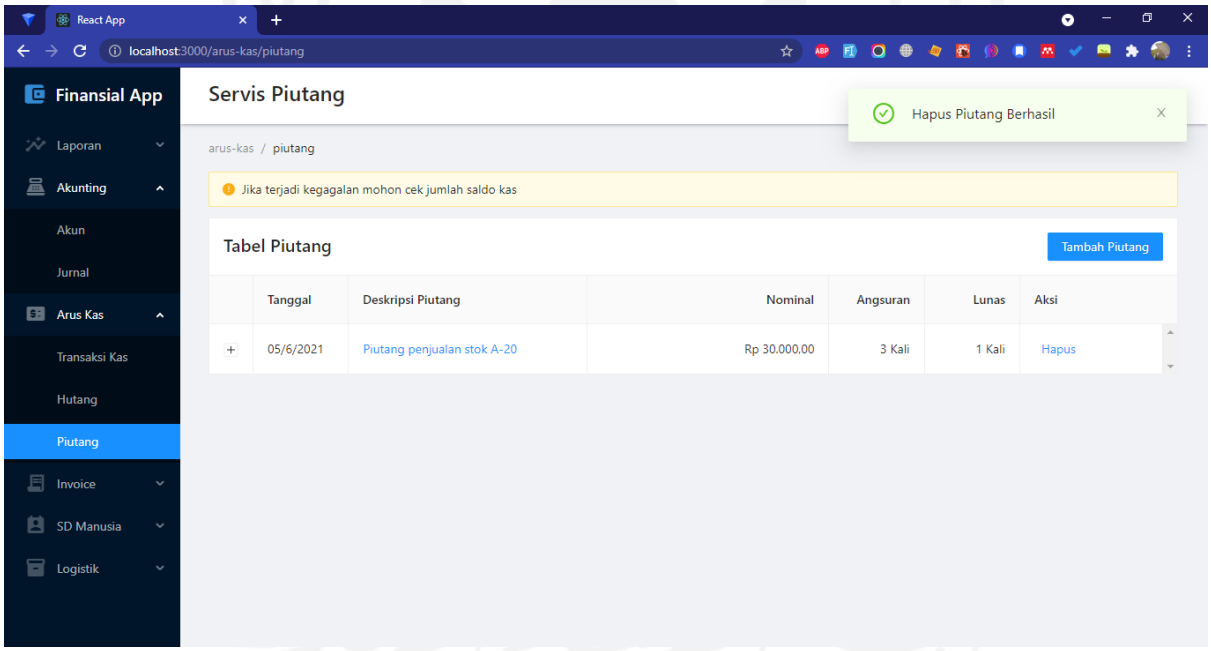
Gambar 4.49 Tampilan tabel data piutang

C. Fungsi DeletePiutang ()

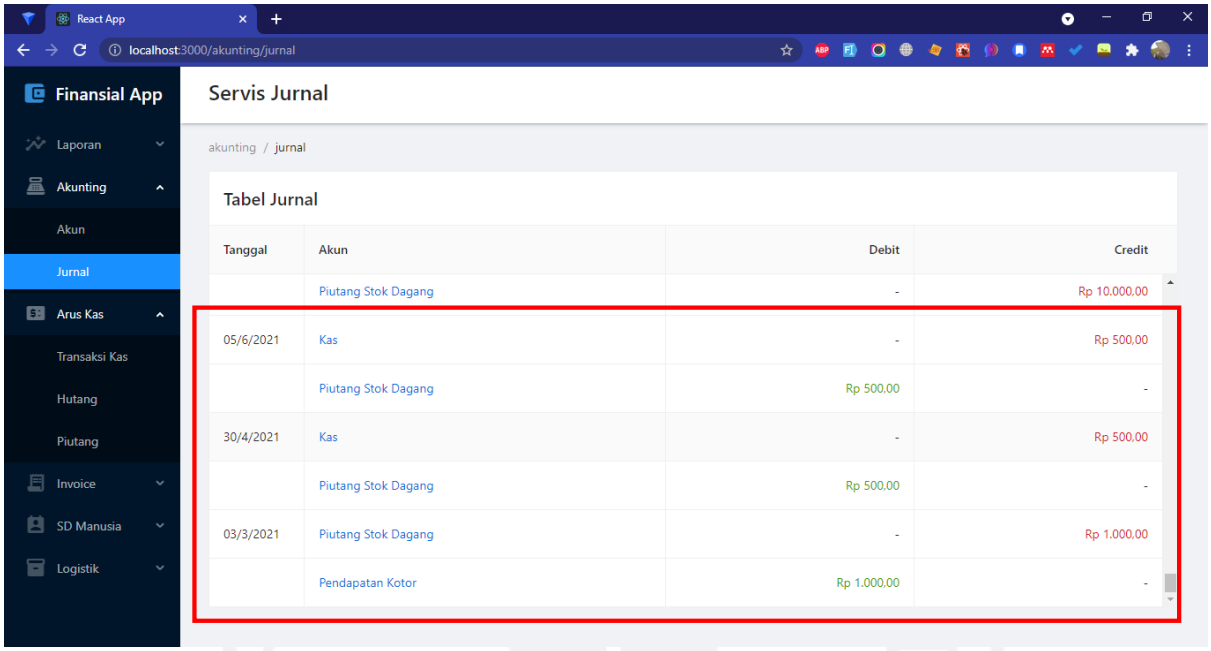
Gambar 4.50 merupakan tampilan konfirmasi ketika ingin menghapus data piutang. Gambar 4.51 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil menghapus data piutang. Gambar 4.52 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada piutang yang dihapus.



Gambar 4.50 Tampilan konfirmasi hapus data piutang



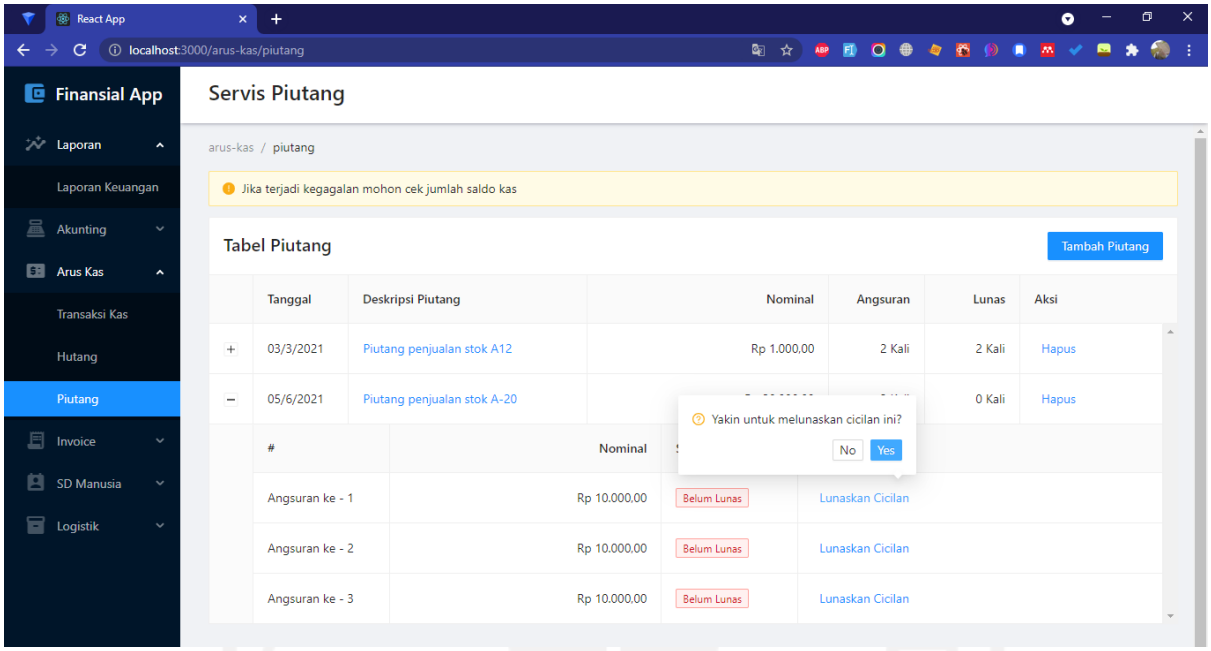
Gambar 4.51 Tampilan *feedback* ketika sistem berhasil menghapus piutang



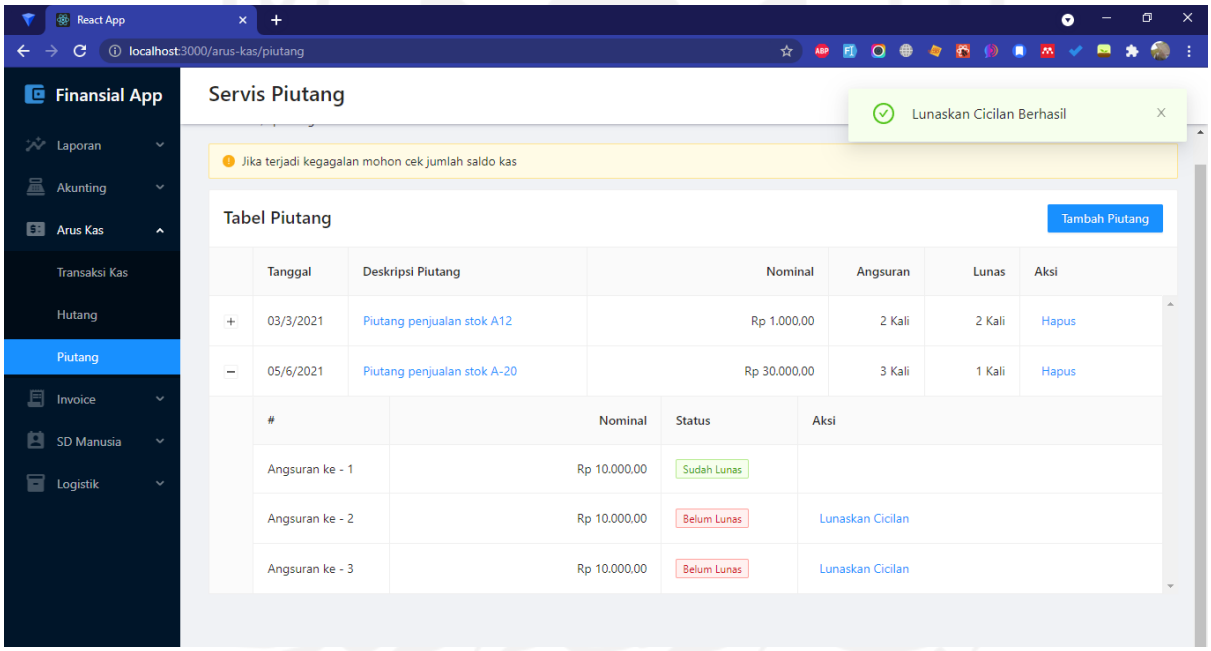
Gambar 4.52 Tampilan transaksi piutang yang dihapus ditransfer ke layanan *ledger*

D. Fungsi LunaskanCicilanPiutang ()

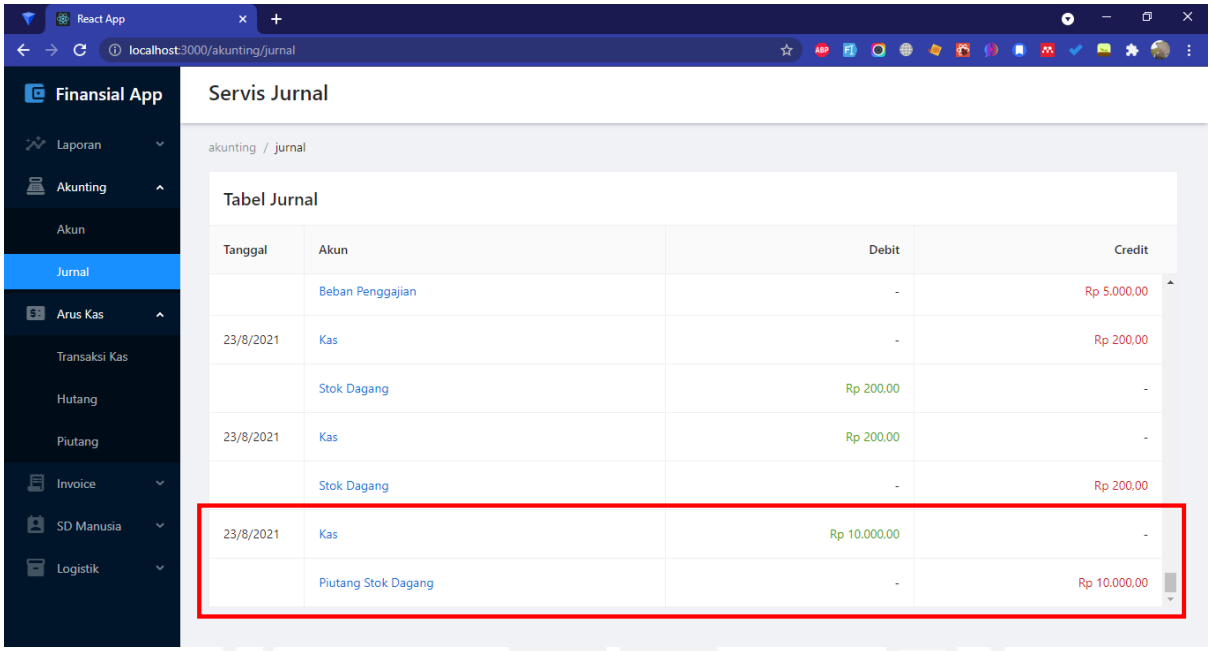
Gambar 4.53 merupakan tampilan konfirmasi ketika ingin melunaskan cicilan piutang. Gambar 4.54 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil melunaskan data cicilan piutang. Gambar 4.55 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada piutang yang dilunaskan.



Gambar 4.53 Tampilan konfirmasi lunaskan data piutang



Gambar 4.54 Tampilan *feedback* ketika sistem berhasil melunaskan piutang



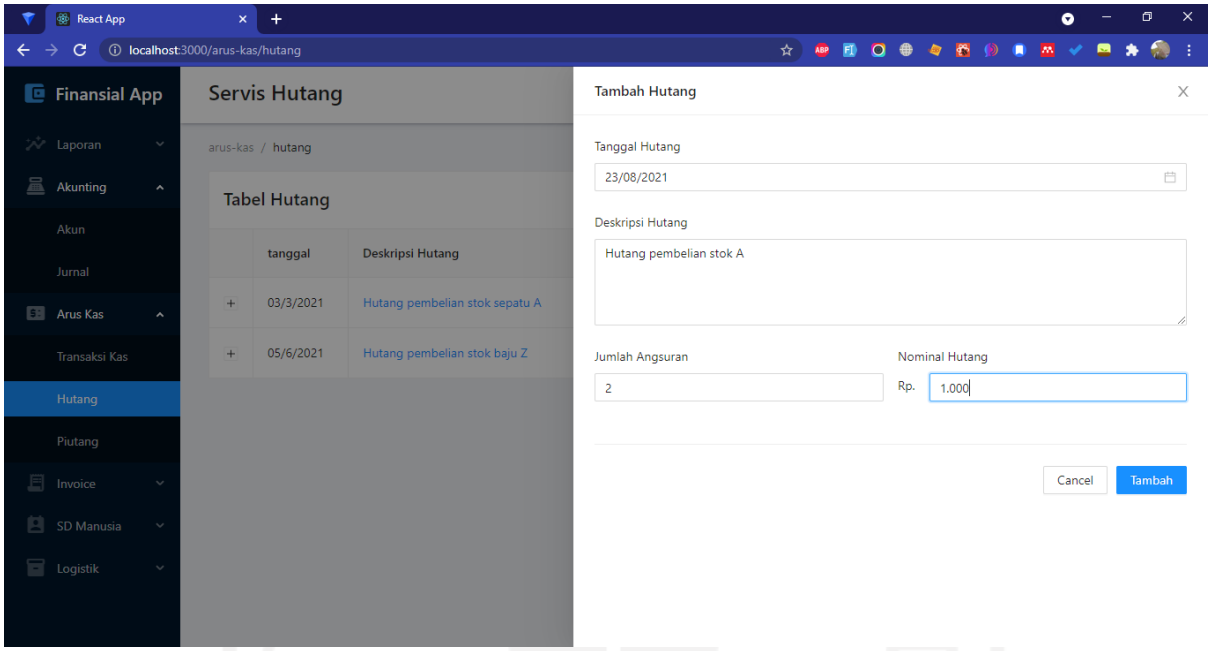
Gambar 4.55 Tampilan transaksi piutang yang dilunaskan ditransfer ke layanan *ledger*

4.8.6 Penerapan Halaman Hutang

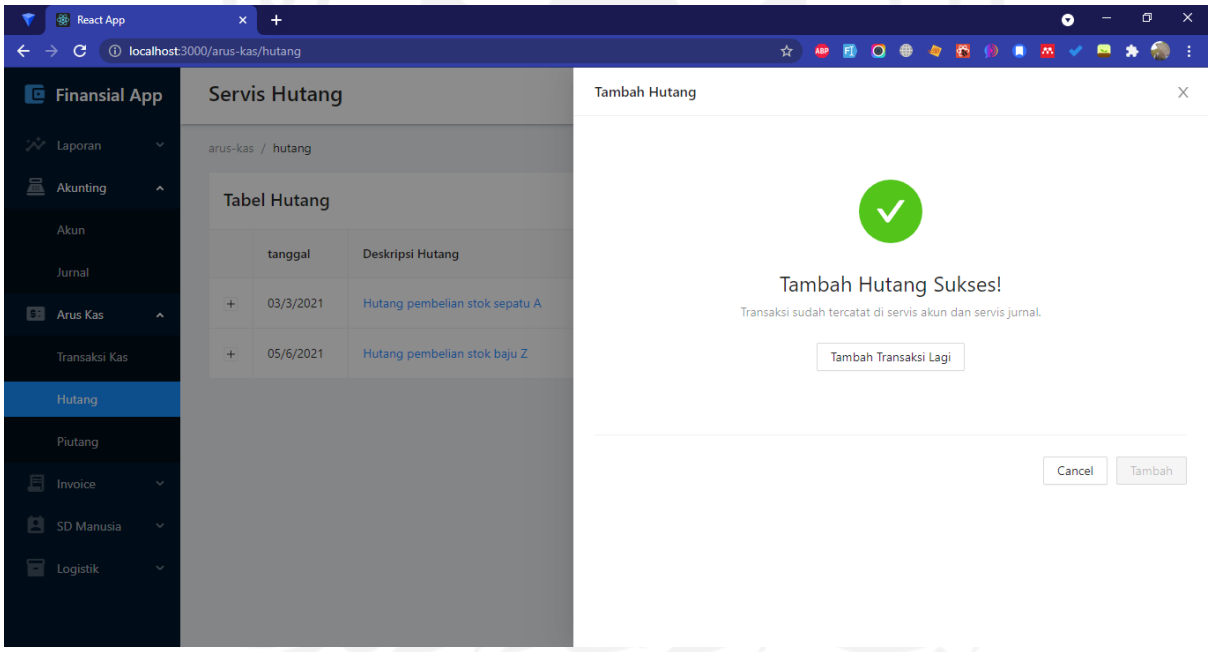
Pada halaman ledger ini terdapat empat fungsi yang diterapkan pada antarmuka, yakni: CreateHutang (), ReadHutang (), DeleteHutang (), LunaskanCicilanHutang ().

A. Fungsi CreateHutang ()

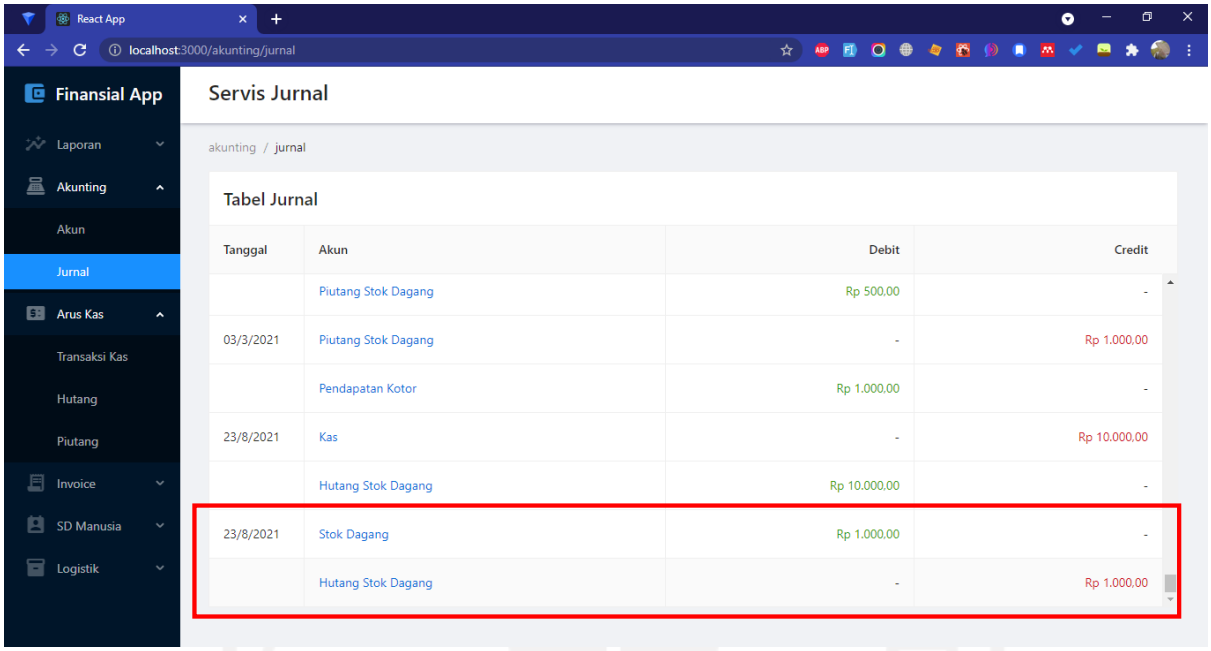
Gambar 4.56 merupakan tampilan untuk menambah hutang. Kemudian, Gambar 4.57 merupakan tampilan *feedback* ketika sistem berhasil menambah hutang sesuai *form* yang dikirim. Selanjutnya, Gambar 4.58 merupakan tampilan nominal hutang yang ditransfer ke layanan *ledger*.



Gambar 4.56 Tampilan *form* CreateHutang ()



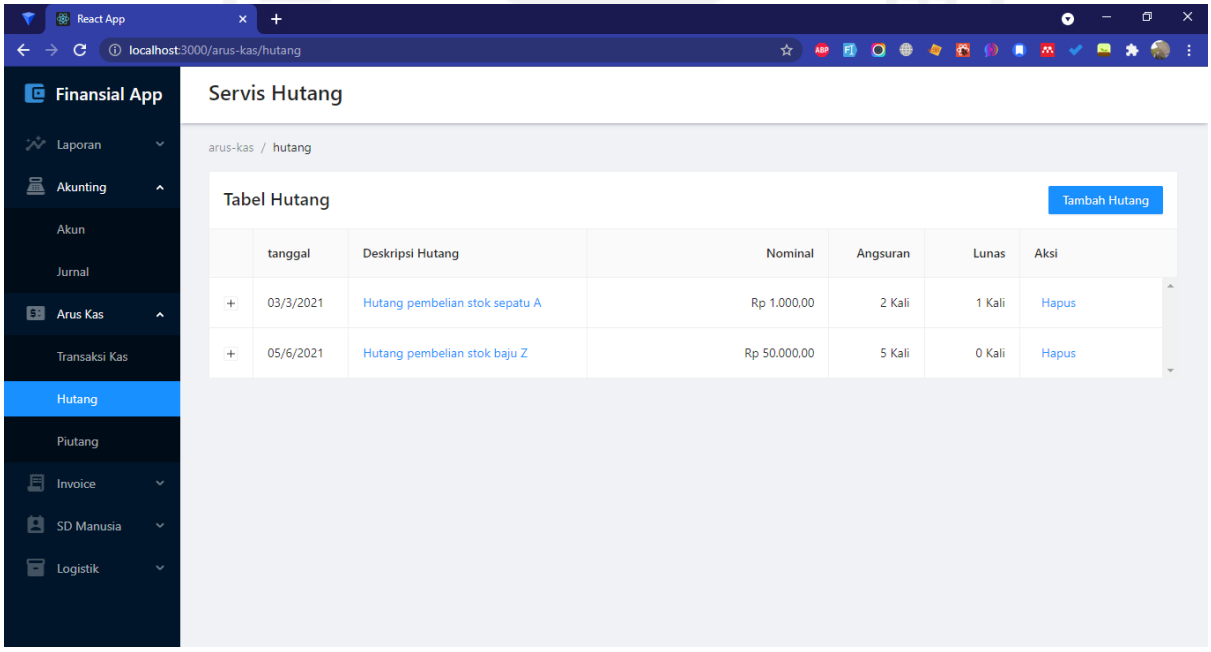
Gambar 4.57 Tampilan *feedback* ketika sistem berhasil menambah piutang



Gambar 4.58 Tampilan transaksi piutang yang ditransfer ke layanan *ledger*

B. Fungsi ReadHutang ()

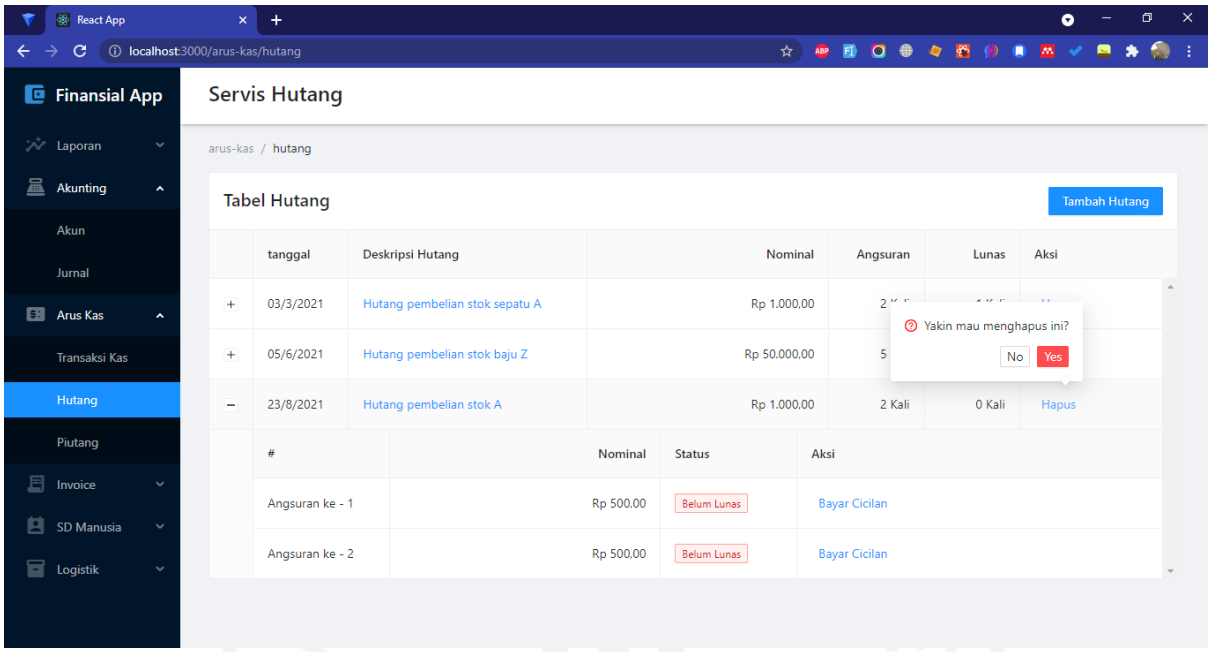
Gambar 4.59 merupakan tampilan untuk melihat seluruh data aset.



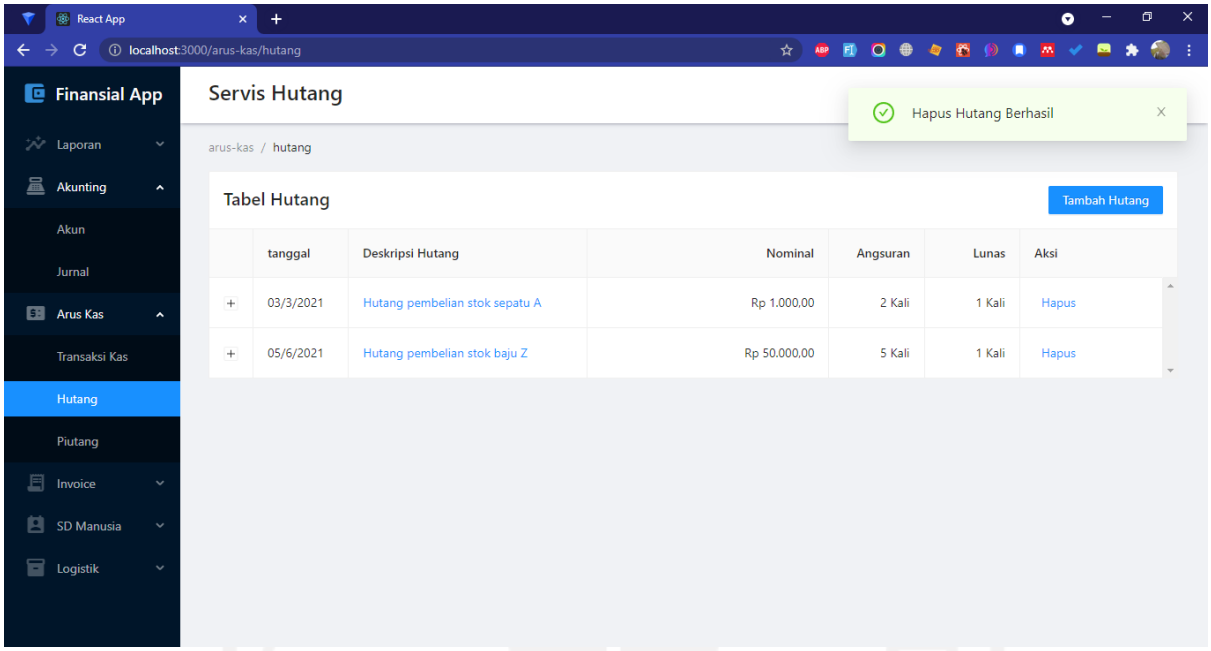
Gambar 4.59 Tampilan tabel data hutang

C. Fungsi DeleteHutang ()

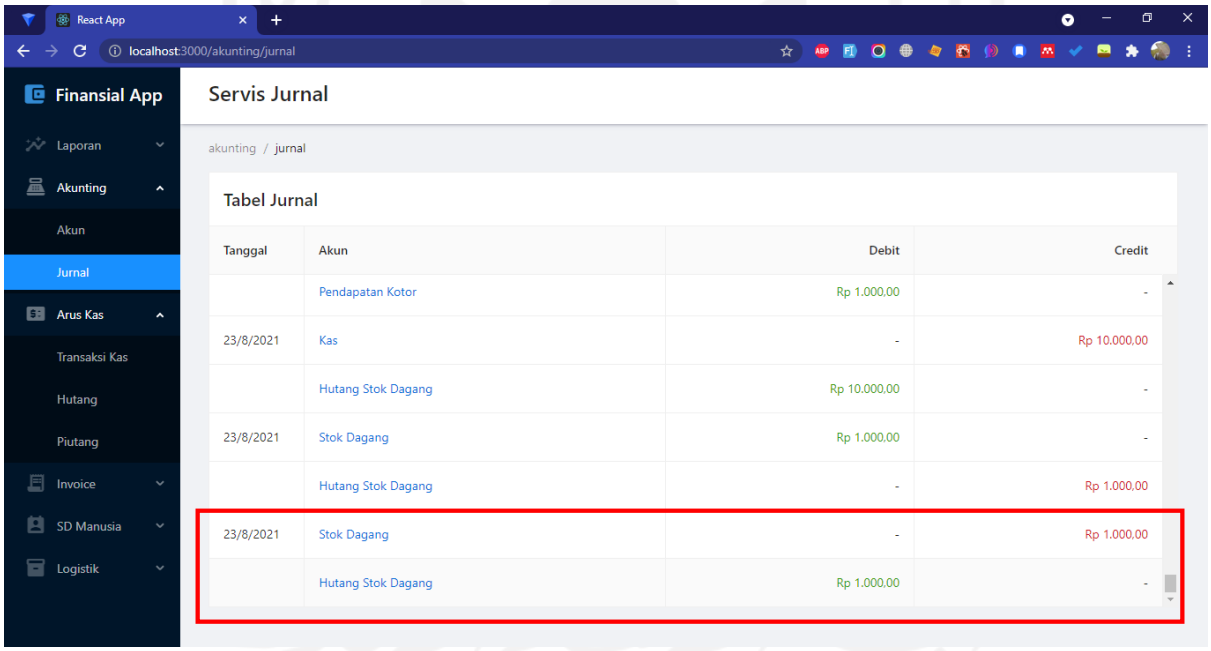
Gambar 4.60 merupakan tampilan konfirmasi ketika ingin menghapus data hutang. Gambar 4.61 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil menghapus data hutang. Gambar 4.62 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada hutang yang dihapus.



Gambar 4.60 Tampilan konfirmasi hapus data hutang



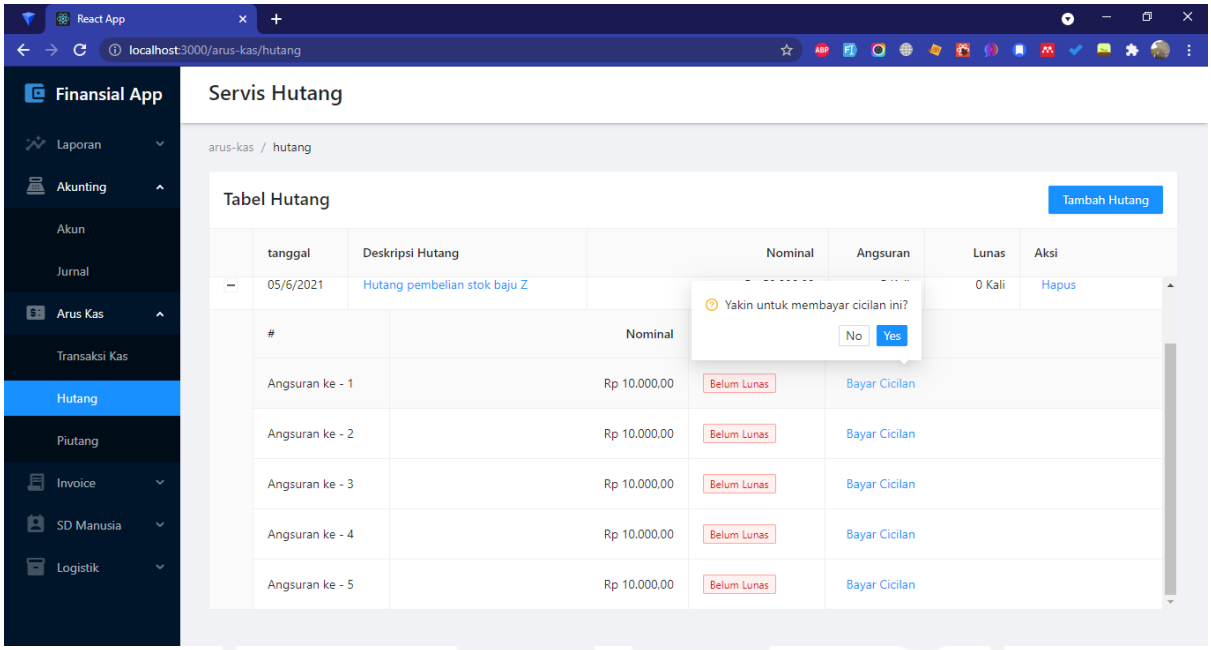
Gambar 4.61 Tampilan *feedback* ketika sistem berhasil menghapus hutang



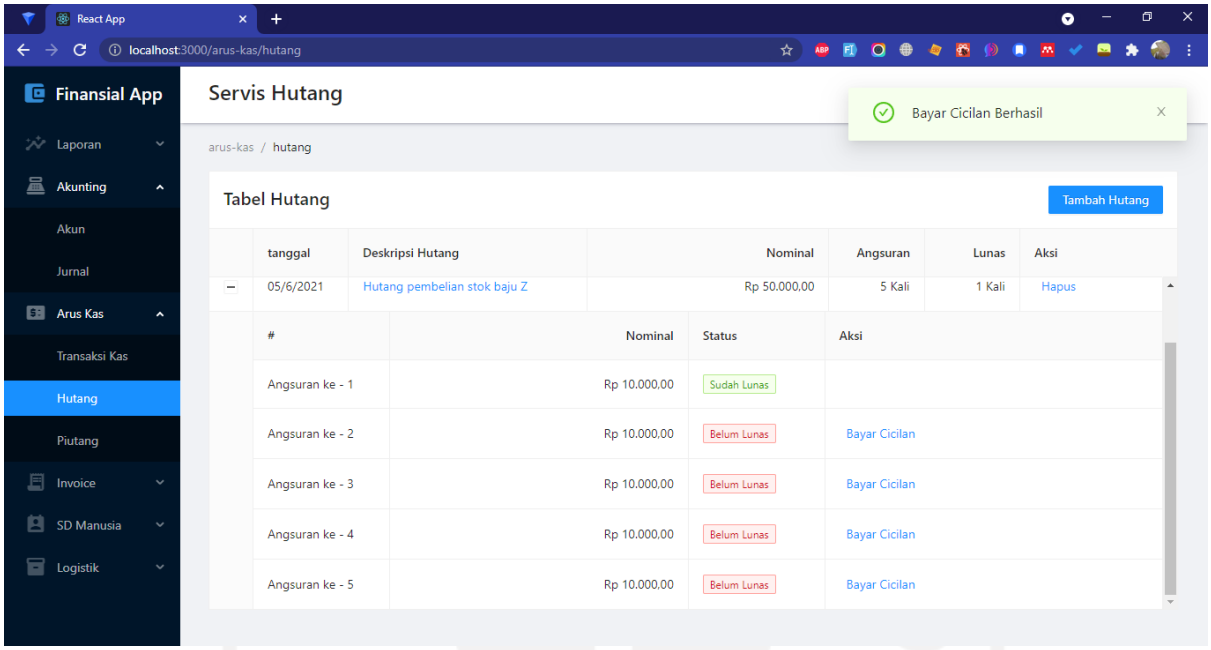
Gambar 4.62 Tampilan transaksi hutang yang dihapus ditransfer ke layanan *ledger*

D. Fungsi LunaskanCicilanHutang ()

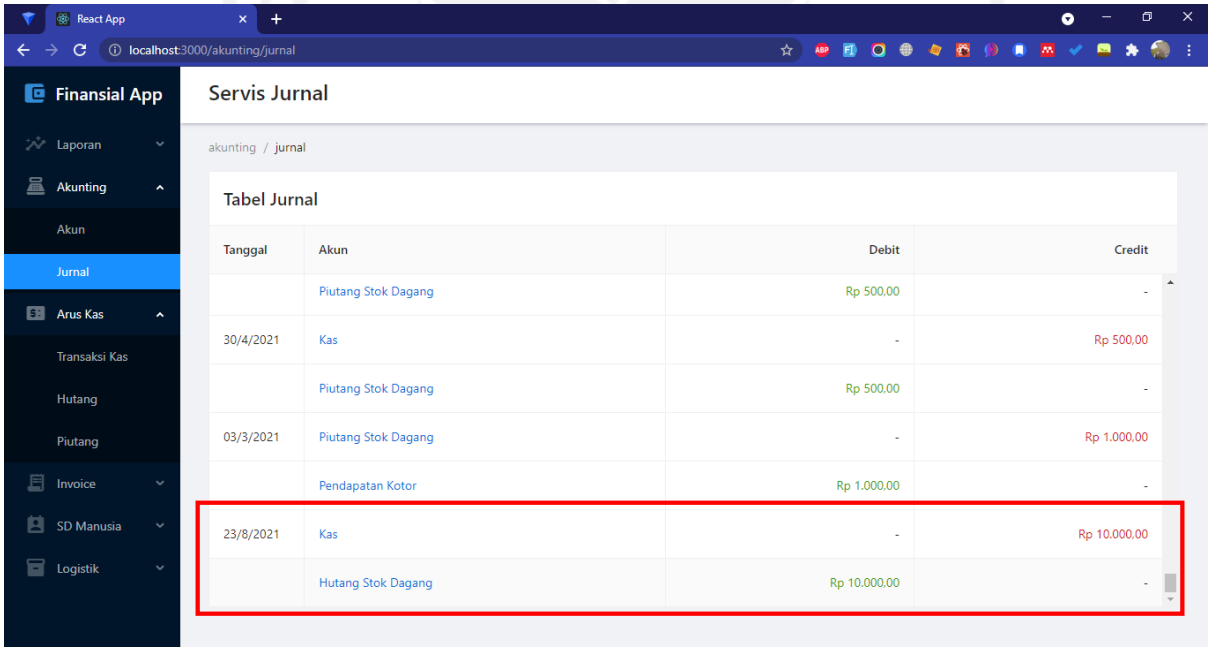
Gambar 4.63 merupakan tampilan konfirmasi ketika ingin melunaskan cicilan hutang. Gambar 4.65 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil melunaskan data cicilan hutang. **Error! Reference source not found.** merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada hutang yang dilunaskan.



Gambar 4.63 Tampilan konfirmasi lunaskan data hutang



Gambar 4.64 Tampilan *feedback* ketika sistem berhasil melunaskan hutang



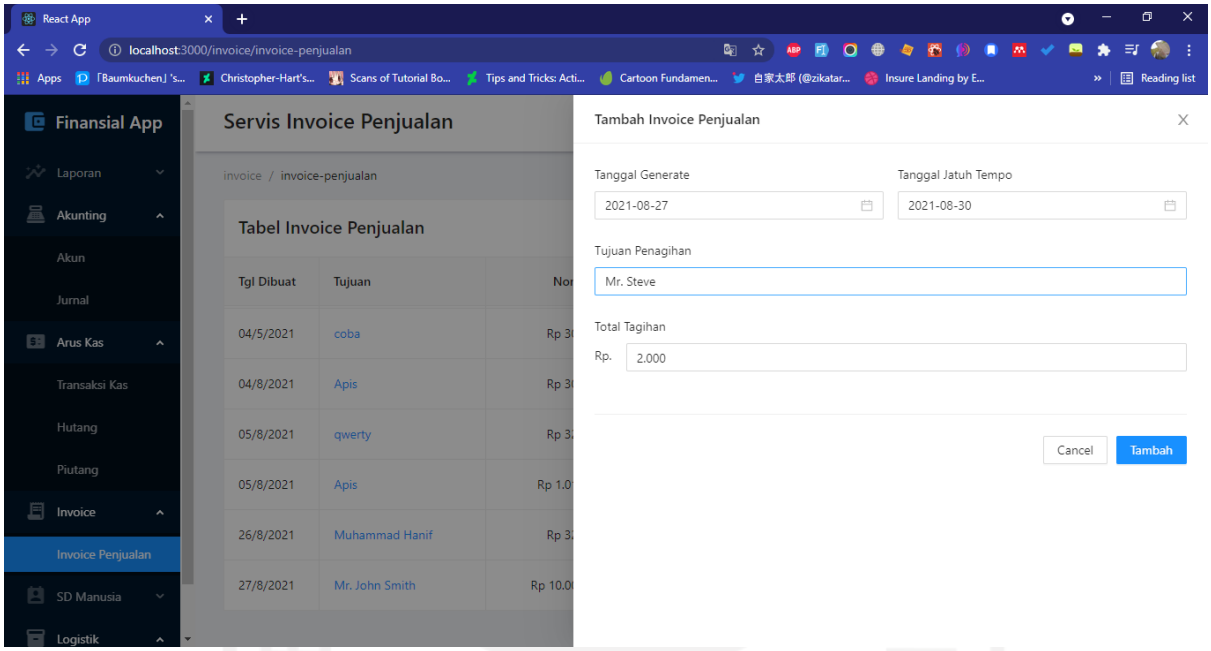
Gambar 4.65 Tampilan transaksi hutang yang dilunaskan ditransfer ke layanan *ledger*

4.8.7 Penerapan Halaman *Invoice*

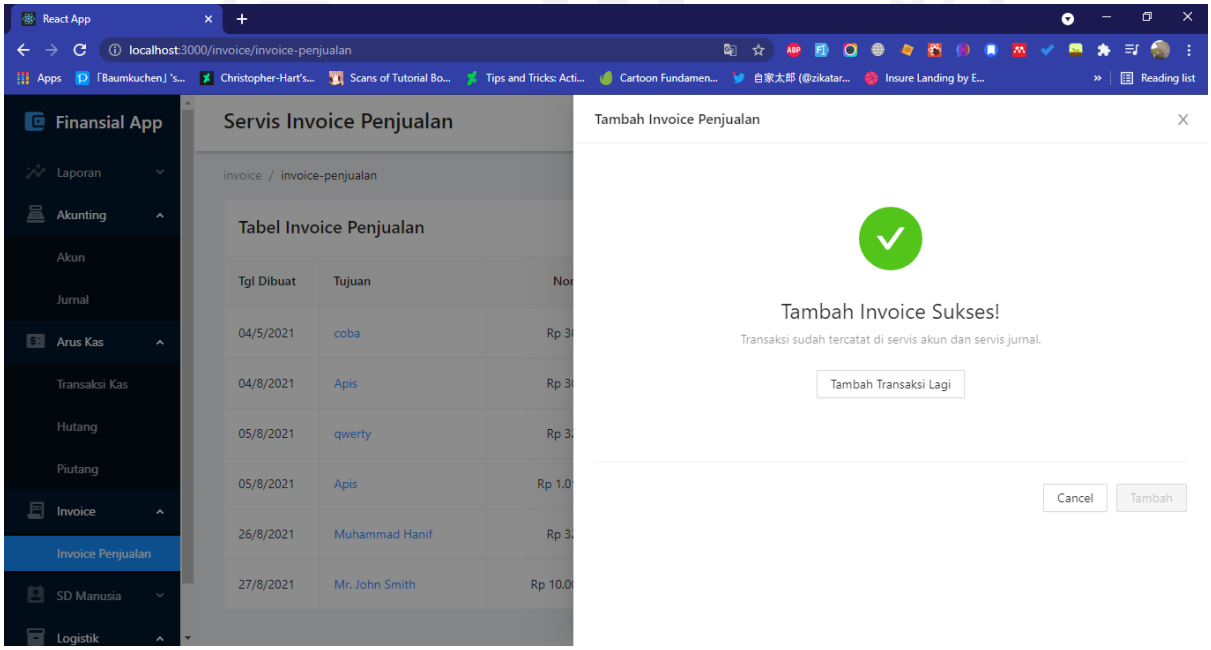
Pada halaman aset ini terdapat tiga fungsi yang diterapkan pada antarmuka, yakni: CreateInvoice (), ReadInvoice (), DeleteInvoice ().

A. Fungsi CreateInvoice ()

Gambar 4.66 merupakan tampilan untuk menambah invoice. Kemudian, Gambar 4.67 merupakan tampilan *feedback* ketika sistem berhasil menambah invoice sesuai *form* yang dikirim.



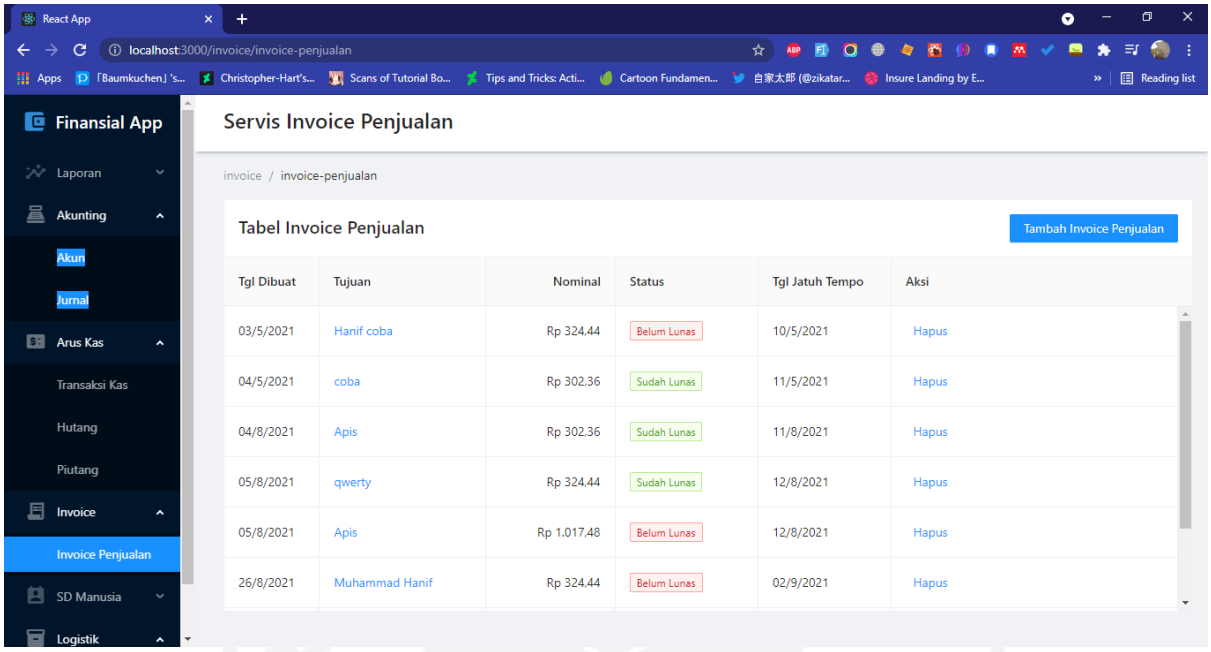
Gambar 4.66 Tampilan *form* CreateInvoice ()



Gambar 4.67 Tampilan *feedback* ketika sistem berhasil menambah invoice

B. Fungsi ReadInvoice ()

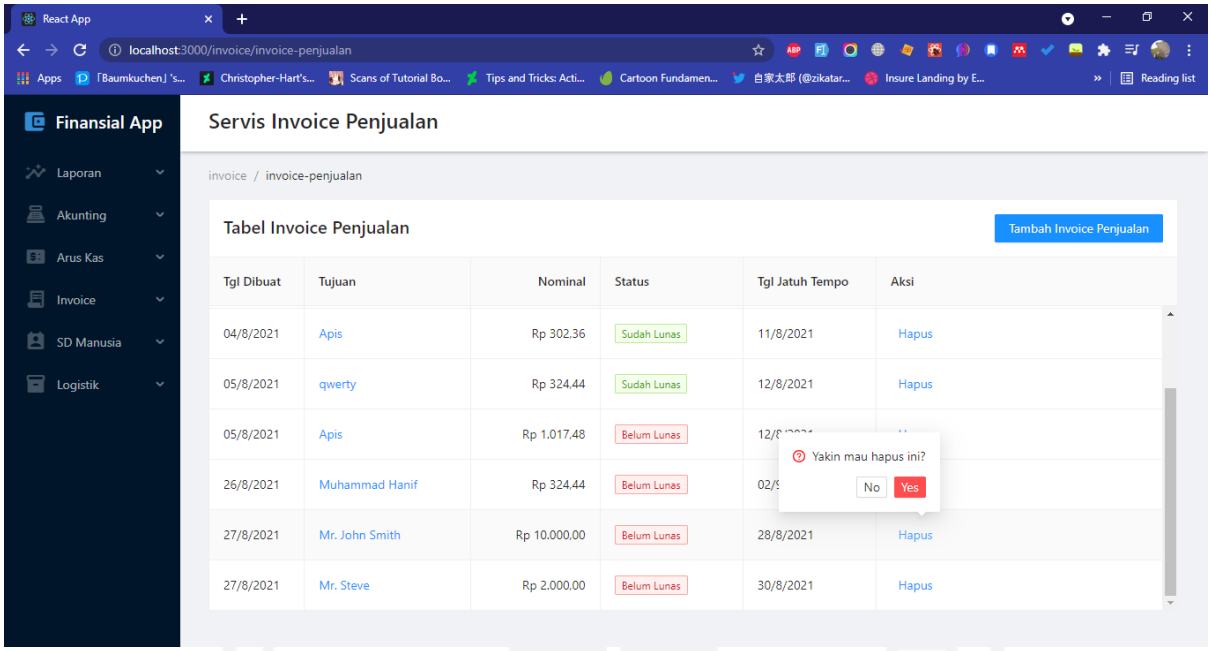
Gambar 4.68 merupakan tampilan untuk melihat seluruh data invoice.



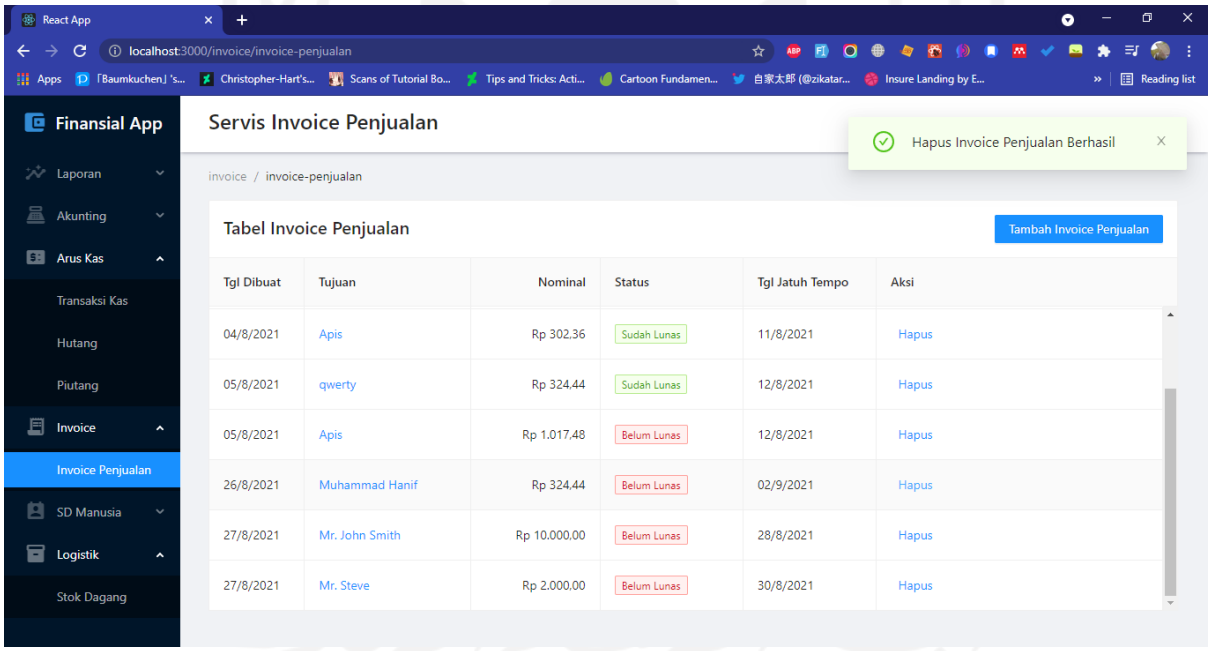
Gambar 4.68 Tampilan tabel data invoice

C. Fungsi DeleteInvoice ()

Gambar 4.69 merupakan tampilan konfirmasi ketika ingin menghapus data aset. Gambar 4.70 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil menghapus data *invoice*.



Gambar 4.69 Tampilan konfirmasi hapus data invoice



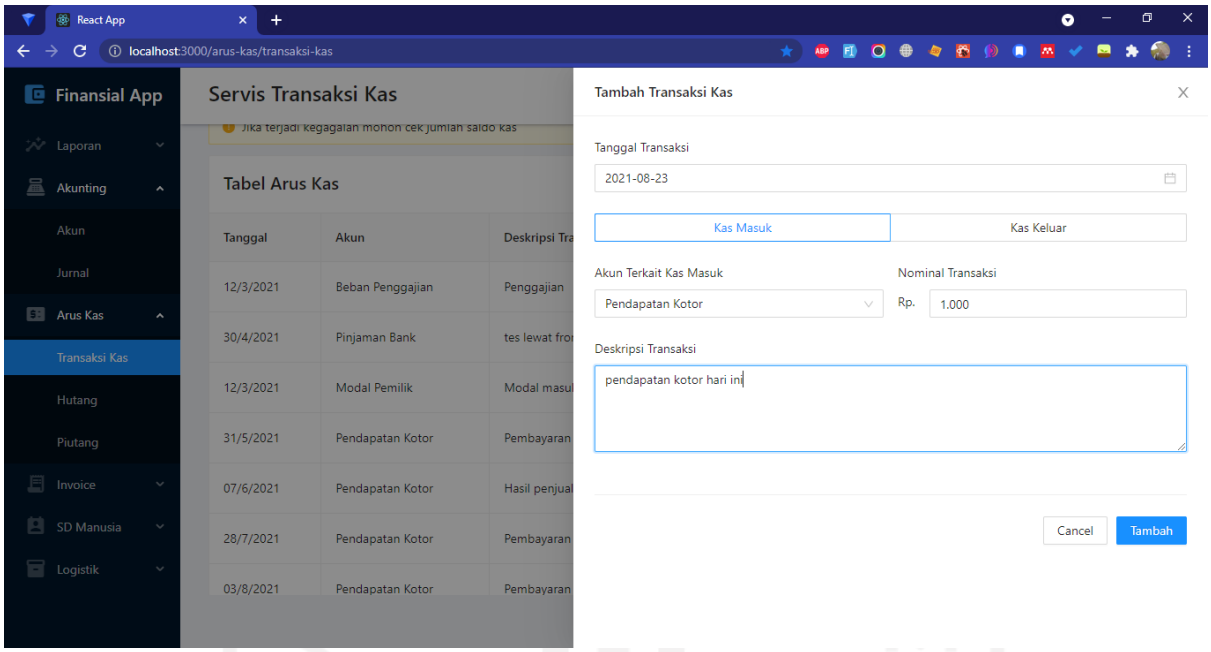
Gambar 4.70 Tampilan *feedback* ketika sistem berhasil menghapus invoice

4.8.8 Penerapan Halaman Arus Kas

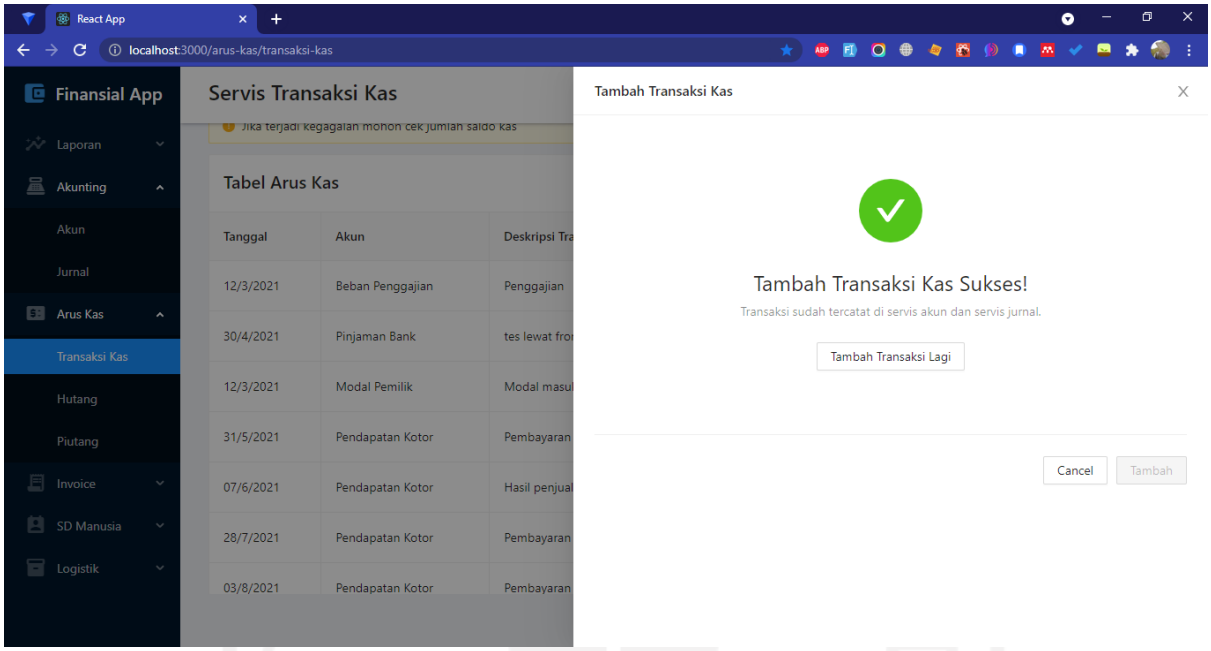
Pada halaman aset ini terdapat tiga fungsi yang diterapkan pada antarmuka, yakni: CreateArusKas (), ReadArusKas (), DeleteArusKas ().

A. Fungsi CreateArusKas ()

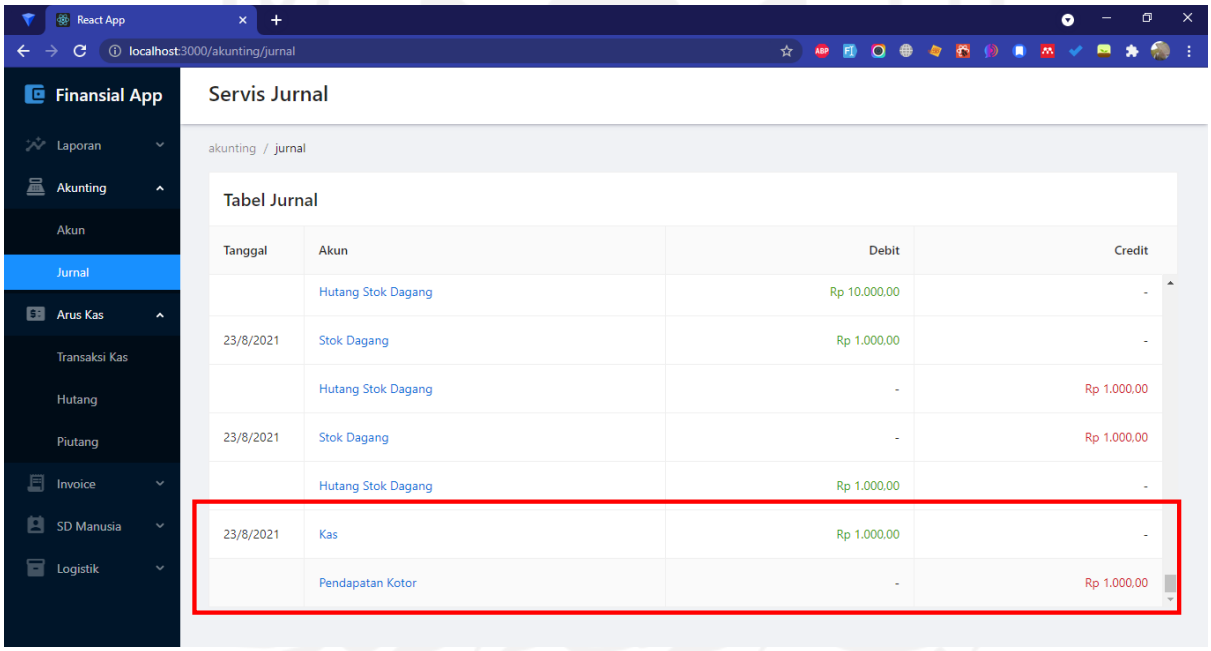
Gambar 4.71 merupakan tampilan untuk menambah data kas. Kemudian, Gambar 4.72 merupakan tampilan *feedback* ketika sistem berhasil menambah data kas sesuai *form* yang dikirim. Selanjutnya, Gambar 4.73 merupakan tampilan nominal data kas yang ditransfer ke layanan *ledger*.



Gambar 4.71 Tampilan *form* CreateArusKas ()



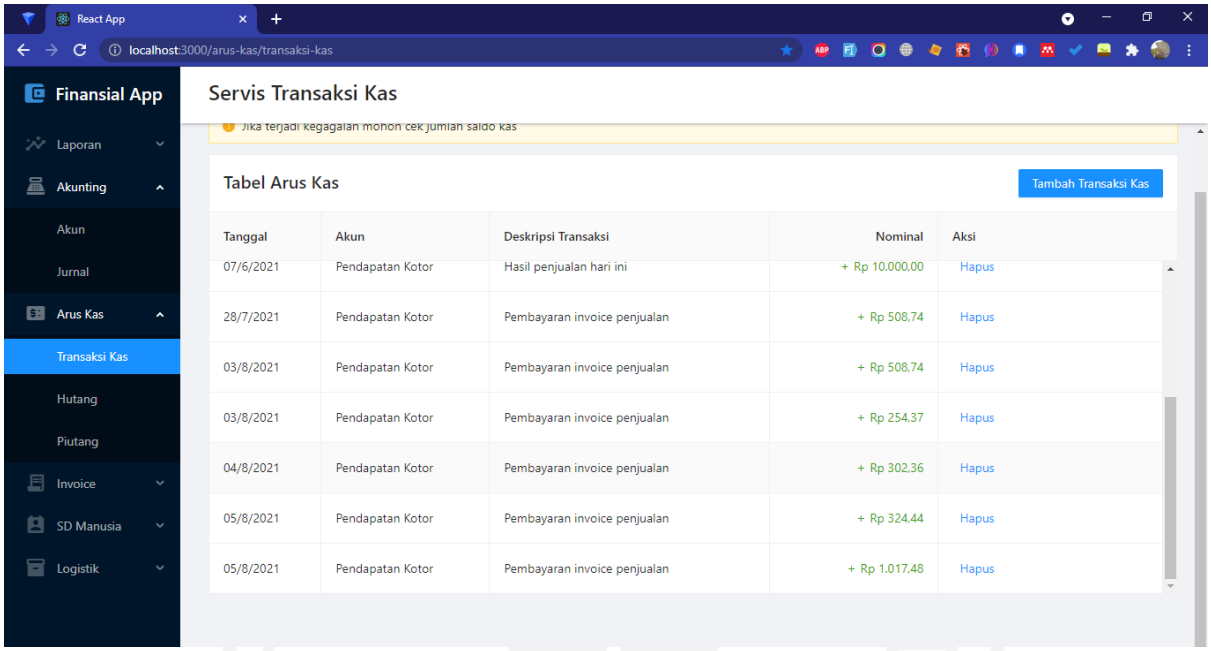
Gambar 4.72 Tampilan *feedback* ketika sistem berhasil menambah data kas



Gambar 4.73 Tampilan transaksi data kas yang ditransfer ke layanan *ledger*

B. Fungsi ReadArusKas ()

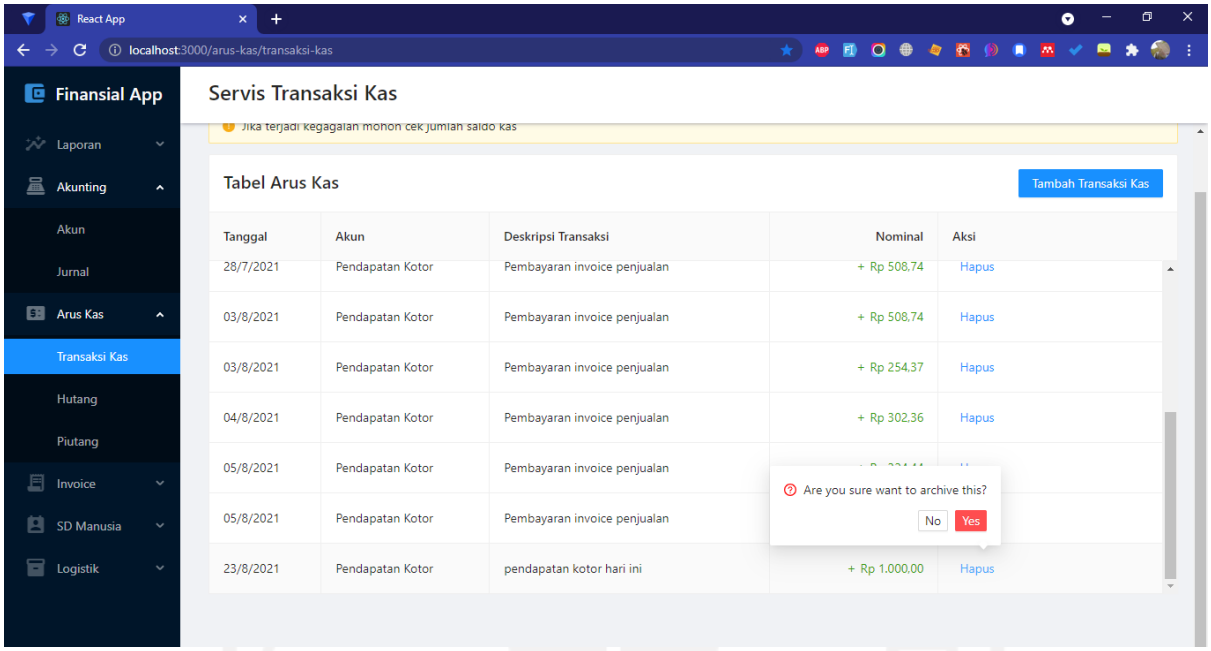
Gambar 4.74 merupakan tampilan untuk melihat seluruh data kas.



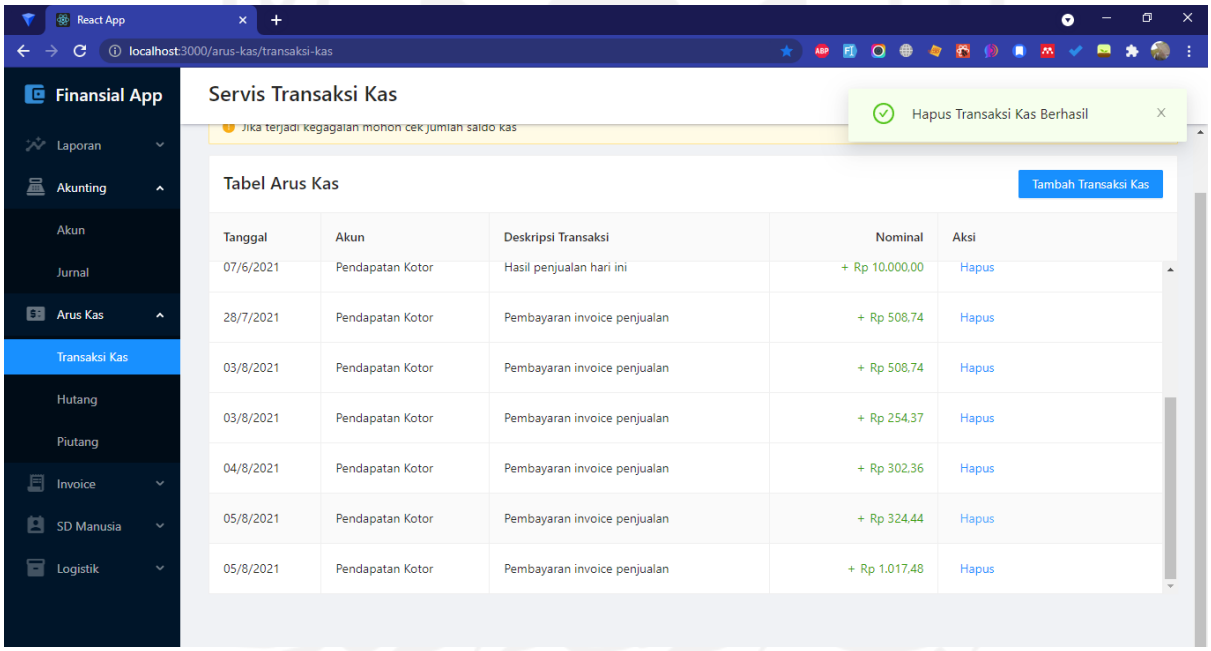
Gambar 4.74 Tampilan tabel data kas

C. Fungsi DeleteArusKas ()

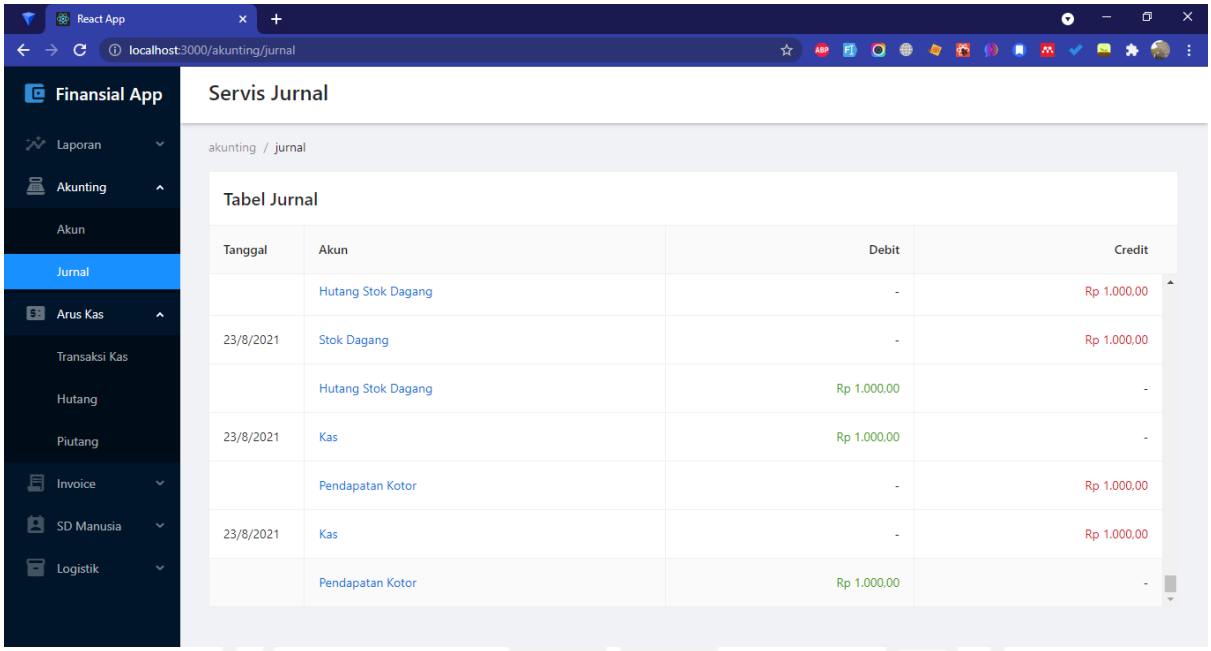
Gambar 4.75 merupakan tampilan konfirmasi ketika ingin menghapus data arus kas. Gambar 4.76 merupakan tampilan halaman ketika muncul notifikasi terkait sistem yang berhasil menghapus data arus kas. Gambar 4.77 merupakan tampilan *rollback* yang terjadi pada layanan *ledger* ketika ada arus kas yang dihapus.



Gambar 4.75 Tampilan konfirmasi hapus data arus kas



Gambar 4.76 Tampilan *feedback* ketika sistem berhasil menghapus arus kas



Gambar 4.77 Tampilan transaksi arus kas yang dihapus ditransfer ke layanan *ledger*

4.9 Pengujian Antarmuka

Setelah antarmuka berhasil diterapkan, selanjutnya dilakukan proses pengujian antarmuka. Seperti yang sudah dijelaskan pada bab 3.9, Pengujian pada antarmuka AE pola finansial dilakukan dengan melakukan interaksi langsung pada aplikasi yang dijalankan di *web browser* lalu menjalankan skenario *test case*-nya. Subbab berikut memaparkan hasil akhir dari pengujian yang dilakukan setelah melakukan perbaikan berulang kali sebelumnya.

4.9.1 Pengujian Halaman Penggajian

Hasil akhir pengujian terhadap halaman penggajian dapat dilihat pada Tabel 4.17.

Tabel 4.17 Hasil akhir pengujian halaman penggajian

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreatePenggajian ()				
1	*Integration*	Klik *button* tambah penggajian, lalu mengisi *form* tambah penggajian, lalu klik tambah	Data penggajian bertambah, data akun berubah, dan data *ledger* bertambah sesuai nominal penggajian,	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
B. Fungsi ReadPenggajian ()				
2	*Integration*	Klik *side menu* penggajian	Muncul seluruh data penggajian pada tabel	Valid
C. Fungsi DeletePenggajian ()				
3	*Integration*	Klik button hapus penggajian, lalu klik ''ya'' pada popup konfirmasi	Data penggajian terhapus, data akun berubah, dan data *ledger* bertambah sesuai nominal penggajian	Valid
D. Fungsi ReadKaryawan ()				
4	*Integration*	Klik *side menu* karyawan	Muncul seluruh data karyawan pada tabel	Valid

4.9.2 Pengujian Halaman Aset

Hasil akhir pengujian terhadap halaman aset dapat dilihat pada Tabel 4.18.

Tabel 4.18 Hasil akhir pengujian halaman aset

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
E. Fungsi CreateAset ()				
1	*Integration*	Klik *button* tambah aset, lalu mengisi *form* tambah penggajian, lalu klik tambah	Data aset bertambah, data akun berubah, dan data *ledger* bertambah sesuai nominal penggajian,	Valid
F. Fungsi ReadAset ()				
2	*Integration*	Klik *side menu* aset	Muncul seluruh data aset pada tabel	Valid
G. Fungsi DeleteAset ()				
3	*Integration*	Klik button hapus aset, lalu klik ''ya'' pada *popup* konfirmasi	Data penggajian terhapus, data akun berubah, dan data *ledger*	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
			bertambah sesuai nominal aset	

4.9.3 Pengujian Halaman *Ledger*

Hasil akhir pengujian terhadap halaman *ledger* dapat dilihat pada Tabel 4.19.

Tabel 4.19 Hasil akhir pengujian halaman *ledger*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
H. Fungsi ReadLedger ()				
1	*Integration*	Klik *side menu ledger*	Muncul seluruh data *ledger* pada tabel	Valid

4.9.4 Pengujian Halaman Akun

Hasil akhir pengujian terhadap halaman akun dapat dilihat pada Tabel 4.20.

Tabel 4.20 Hasil akhir pengujian halaman akun

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi ChangeTargetNominal ()				
1	*Integration*	Klik *button* ubah, lalu mengisi *form* ubah target nominal, lalu klik ubah	Data target nominal berubah	Valid
B. Fungsi ReadAkun ()				
2	*Integration*	Klik *side menu* akun	Muncul seluruh data aset pada tabel	Valid
C. Fungsi GetlaporanLabaRugi ()				
3	*Integration*	Klik *side menu* laporan keuangan	Muncul seluruh data laba rugi pada tabel	Valid

4.9.5 Pengujian Halaman Piutang

Hasil akhir pengujian terhadap halaman piutang dapat dilihat pada Tabel 4.21.

Tabel 4.21 Hasil akhir pengujian halaman piutang

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreatePiutang ()				
1	*Integration*	Klik *button* tambah piutang, lalu mengisi *form* tambah piutang, lalu klik tambah	Data piutang bertambah, data akun berubah, dan data *ledger* bertambah sesuai nominal piutang,	Valid
B. Fungsi ReadPiutang ()				
2	*Integration*	Klik *side menu* piutang	Muncul seluruh data piutang pada tabel	Valid
C. Fungsi DeletePiutang ()				
3	*Integration*	Klik *button* hapus piutang, lalu klik ''ya'' pada *popup* konfirmasi	Data piutang terhapus, data akun berubah, dan data *ledger* bertambah sesuai nominal aset	Valid
D. Fungsi LunaskanCicilanPiutang ()				
4	*Integration*	Klik *button* hapus piutang, lalu klik ''ya'' pada *popup* konfirmasi	Status cicilan piutang menjadi lunas, data akun berubah, dan data *ledger* bertambah sesuai nominal cicilan	Valid

4.9.6 Pengujian Halaman Hutang

Hasil akhir pengujian terhadap halaman hutang dapat dilihat pada Tabel 4.22.

Tabel 4.22 Hasil akhir pengujian halaman hutang

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
E. Fungsi CreateHutang ()				
1	*Integration*	Klik *button* tambah hutang, lalu mengisi *form* tambah hutang, lalu klik tambah	Data hutang bertambah, data akun berubah, dan data *ledger* bertambah sesuai nominal hutang,	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
F. Fungsi ReadHutang ()				
2	*Integration*	Klik *side menu* hutang	Muncul seluruh data hutang pada tabel	Valid
G. Fungsi DeleteHutang ()				
3	*Integration*	Klik *button* hapus hutang, lalu klik ''ya'' pada *popup* konfirmasi	Data hutang terhapus, data akun berubah, dan data *ledger* bertambah sesuai nominal aset	Valid
H. Fungsi LunaskanCicilanHutang ()				
4	*Integration*	Klik *button* lunaskan cicilan, lalu klik ''ya'' pada *popup* konfirmasi	Status cicilan hutang menjadi lunas, data akun berubah, dan data *ledger* bertambah sesuai nominal cicilan	Valid

4.9.7 Pengujian Halaman *Invoice*

Hasil akhir pengujian terhadap halaman *invoice* dapat dilihat pada Tabel 4.23.

Tabel 4.23 Hasil akhir pengujian halaman *invoice*

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreateInvoice ()				
1	*Integration*	Klik *button* tambah *invoice*, lalu mengisi *form* tambah *invoice*, lalu klik tambah	Data *invoice* bertambah	Valid
B. Fungsi ReadInvoice ()				
2	*Integration*	Klik *side menu invoice*	Muncul seluruh data *invoice* pada tabel	Valid
C. Fungsi DeleteInvoice ()				
3	*Integration*	Klik *button* hapus *invoice*, lalu klik ''ya''	Data *invoice* terhapus	Valid

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
		pada *popup* konfirmasi		

4.9.8 Pengujian Halaman Arus Kas

Hasil akhir pengujian terhadap halaman arus kas dapat dilihat pada Tabel 4.24.

Tabel 4.24 Hasil akhir pengujian halaman arus kas

No	Lingkup	Test Case	Hasil yang diharapkan	Hasil pengujian
A. Fungsi CreateArusKas ()				
1	*Integration*	Klik *button* tambah arus kas, lalu mengisi *form* tambah arus kas, lalu klik tambah	Data arus kas bertambah, data akun berubah, dan data *ledger* bertambah sesuai nominal arus kas,	Valid
B. Fungsi ReadArusKas ()				
2	*Integration*	Klik *side menu* arus kas	Muncul seluruh data arus kas pada tabel	Valid
C. Fungsi DeleteArusKas ()				
3	*Integration*	Klik button hapus arus kas, lalu klik ''ya'' pada *popup* konfirmasi	Data arus kas terhapus, data akun berubah, dan data *ledger* bertambah sesuai nominal aset	Valid

BAB V
Kesimpulan dan Saran

5.1 Kesimpulan

Penelitian ini bertujuan untuk melihat deskripsi penerapan AE pola finansial pada aplikasi berbasis *microservices* menggunakan metodologi yang diusulkan. Berikut metodologi yang diusulkan tersebut secara berurutan :

a. Dekomposisi AE pola finansial menggunakan DDD (*Domain Driven Design*),
b. Analisis kebutuhan API layanan,
c. Analisis kebutuhan basis data menggunakan skema usulan dari komunitas MongoDB,
d. Penerapan basis data menggunakan MongoDB Atlas,
e. Penerapan API layanan menggunakan ExpressJS dan NodeJS,
f. Pengujian API layanan secara *white box*,
g. Penerapan antarmuka menggunakan ReactJS,
h. Pengujian antarmuka secara *white box*.

Berdasarkan apa yang sudah dilakukan pada penelitian ini, metodologi yang diusulkan tersebut terbukti dapat menerapkan AE pola finansial sebagai studi kasus pada aplikasi berbasis *microservices.* Hal tersebut dapat dilihat dari hasil aplikasi yang mampu mengelola penggajian, aset, piutang, hutang, *invoice*, arus kas serta memiliki layanan akun dan *ledger* untuk melihat keadaan finansial secara menyeluruh. Adapun *microservices*-nya berupa sepuluh API layanan yang berhasil diterapkan, yakni: layanan *posting ledger*, *rollback ledger*, penggajian, aset, *ledger*, akun, piutang, hutang, *invoice*, arus kas.

5.2 Saran

Penelitian ini hanya menyajikan deskripsi terkait hasil penerapan AE pola finansial menjadi aplikasi berbasis *microservices* dengan metodologi yang diusulkan. Penelitian ini belum bisa mengukur performa atas metodologi yang diusulkan tersebut. Selain itu terkait AE pola finansial, penelitian ini juga belum mampu untuk mengukur efektifitas pola terhadap kasus nyata yang terjadi pada domain finansial. Oleh karena itu untuk menyempurnakan penelitian ini, perlu dilakukan penelitian lebih lanjut mengenai pengukuran kuantitatif dari metodologi penerapan aplikasi berbasis *microservices* seperti waktu pengerjaan, kurva belajar, biaya pengembangan dan semacamnya agar performa antar metodologi dapat diukur dan

dibandingkan. Kemudian, perlu dilakukan penelitian lebih lanjut juga terkait pengukuran efektifitas AE pola finansial jika diterapkan pada kasus yang nyata.

DAFTAR PUSTAKA

Aggarwal, S., & Verma, J. (2018). Comparative analysis of MEAN stack and MERN stack. *International Journal of Recent Research Aspects*, *5*(1), 127–132.

Boyd, K., Epstein, L., Holtzman, M. P., Kass-Shraibman, F., S., M. L. V., Tage, S. J. A. T., & Welytok, C. T. J. G. (2014). *Accounting All-In-One For Dummies*.

Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2018). From Monolithic to Microservices: An Experience Report from the Banking Domain. *IEEE Software*, *35*(3), 50–55. https://doi.org/10.1109/MS.2018.2141026

Data Model Design — MongoDB Manual. (n.d.). Retrieved August 12, 2021, from https://docs.mongodb.com/manual/core/data-model-design/

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., & Safina, L. (2018). Microservices: How to make your application scale. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10742 LNCS*, 95–104. https://doi.org/10.1007/978-3-319-74313-4_8

Fan, C. Y., & Ma, S. P. (2017). Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. *Proceedings - 2017 IEEE 6th International Conference on AI and Mobile Services, AIMS 2017*, 109–112. https://doi.org/10.1109/AIMS.2017.23

GitHub - axios/axios: Promise based HTTP client for the browser and node.js. (n.d.). Retrieved August 22, 2021, from https://github.com/axios/axios

GitHub - expressjs/body-parser: Node.js body parsing middleware. (n.d.). Retrieved August 22, 2021, from https://github.com/expressjs/body-parser

GitHub - expressjs/cors: Node.js CORS middleware. (n.d.). Retrieved August 22, 2021, from https://github.com/expressjs/cors

GitHub - expressjs/express: Fast, unopinionated, minimalist web framework for node. (n.d.). Retrieved August 22, 2021, from https://github.com/expressjs/express

GitHub - facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces. (n.d.). Retrieved August 25, 2021, from https://github.com/facebook/react

GitHub - motdotla/dotenv: Loads environment variables from .env for nodejs projects. (n.d.). Retrieved August 22, 2021, from https://github.com/motdotla/dotenv

GitHub - postmanlabs/postman-docs: Documentation for Postman, a collaboration platform

for API development. Available for Mac, Windows and Linux. (n.d.). Retrieved August 25, 2021, from https://github.com/postmanlabs/postman-docs

GitHub - remy/nodemon: Monitor for any changes in your node.js application and automatically restart the server - perfect for development. (n.d.). Retrieved August 22, 2021, from https://github.com/remy/nodemon

Hasselbring, W., & Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 243–246. https://doi.org/10.1109/ICSAW.2017.11

Ilin, I., Levina, A., & Iliashenko, O. (2017). Enterprise architecture approach to mining companies engineering. *MATEC Web of Conferences*, *106*. https://doi.org/10.1051/matecconf/201710608066

Indrasiri, K., & Siriwardena, P. (2018). *Microservices for the Enterprise: Designing, Developing, and Deploying*.

Lewis, J., & Fowler, M. (2014). *Microservices*. https://martinfowler.com/articles/microservices.html

Loukides, M., & Swoyer, S. (2020, July 15). *Microservices Adoption in 2020 – O'Reilly*. https://www.oreilly.com/radar/microservices-adoption-in-2020/

MongoDB Application Modernization Guide | MongoDB. (n.d.). Retrieved August 24, 2021, from https://www.mongodb.com/collateral/mongodb-application-modernization-guide

Mongoose v5.13.7: Schemas. (n.d.). Retrieved August 15, 2021, from https://mongoosejs.com/docs/guide.html

Newman, S. (2015). Building Microservices. In *O'Reilly*. https://www.google.hr/books?hl=en&lr=&id=jjl4BgAAQBAJ&pgis=1%5Cnhttp://oreilly.com/catalog/errata.csp?isbn=9781491950357

Nidhra, S. (2012). Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, *2*(2), 29–50. https://doi.org/10.5121/ijesa.2012.2204

Perroud, T., & Inversini, R. (2013). *Enterprise Architecture Patterns*.

Richards, C. (2018). *Chris Richards* (Vol. 2018, Issue March).

Richardson, C. (n.d.). *Developing event-driven microservices with event sourcing and CQRS svcc 2015.key*.

Riku, M. O., & Setyohadi, D. B. (2017). Strategic plan with enterprise architecture planning

for applying information system at PT. Bestonindo Central Lestari. *2017 5th International Conference on Cyber and IT Service Management, CITSM 2017*. https://doi.org/10.1109/CITSM.2017.8089274

Samikshya, A. (2020). *MERN STACK WITH MODERN WEB PRACTICES-Developers Connecting Application*.

Schekkerman, J. (2004). How to survive in the jungle of Enterprise Architecture Frameworks. In *Framework* (p. 113). http://www.amazon.com/Survive-Jungle-Enterprise-Architecture-Frameworks/dp/141201607X

Sidiq, P., & Bhakti, D. (2020). *Architecture Planning Information Manufacture Systems Using Enterprise Architecture Planning At PT. Tin Tin*. *January 2020*. https://doi.org/10.4108/eai.11-7-2019.2298090

Singal, A. (2020). *Finance for Non-Finance Executives-Business Expert Press*.

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. J. (2018). The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software*, *146*, 215–232. https://doi.org/10.1016/j.jss.2018.09.082

Suryotrisongko, H., Jayanto, D. P., & Tjahyanto, A. (2017). Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot. *Procedia Computer Science*, *124*, 736–743. https://doi.org/10.1016/j.procs.2017.12.212

Tamm, T., Seddon, P. B., Shanks, G., & Reynolds, P. (2011). How does enterprise architecture add value to organisations? *Communications of the Association for Information Systems*, *28*(1), 141–168. https://doi.org/10.17705/1cais.02810

Widodo, M. P. (n.d.). *Pengembangan Aplikasi Pelaporan Progress-Plan- Problem untuk Manajemen Tugas dan Penentuan OKR di Krafthaus Indonesia*.

Wu, M., Ding, X., & Hou, R. (2019). Design and implementation of B2B E-commerce platform based on microservices architecture. *ACM International Conference Proceeding Series*, 30–34. https://doi.org/10.1145/3339363.3339369

LAMPIRAN