IDENTIFIKASI *SYNTATICAL FAULT* DAN *SEMANTICAL FAULT* DALAM *EVENT LOG* MENGGUNAKAN *TRACE CLUSTERING*

TUGAS AKHIR

Disusun oleh:

Yoshua Fernandes Sirait

3311501050

Disusun untuk memenuhi salah satu syarat kelulusan Program Diploma III

PROGRAM STUDI TEKNIK INFORMATIKA JURUSAN TEKNIK INFORMATIKA

POLITEKNIK NEGERI BATAM

BATAM

2018

HALAMAN PENGESAHAN

IDENTIFIKASI *SYNTATICAL FAULT* DAN *SEMANTICAL FAULT* DALAM *EVENT LOG* MENGGUNAKAN *TRACE CLUSTERING*

Disusun oleh:
Yoshua Fernandes Sirait
3311501050

Telah dikonsultasikan dengan dosen pembimbing
sebagai persyaratan untuk melaksanakan sidang Tugas Akhir II

Batam, 16 Agustus 2018

Disetujui oleh:

Pembimbing

Metta Santiputri, S.T., M.Sc., Ph.D.

NIK 197707202012122004

HALAMAN PERNYATAAN

Dengan ini, saya:

NIM : 3311501050

Nama : Yoshua Fernandes Sirait

adalah mahasiswa Program Studi Teknik Informatika Politeknik Negeri Batam menyatakan bahwa Tugas Akhir dengan judul:

IDENTIFIKASI *SYNTATICAL FAULT* DAN *SEMANTICAL FAULT* DALAM *EVENT LOG* MENGGUNAKAN *TRACE CLUSTERING*

disusun dengan:

1. Tidak melakukan plagiat terhadap naskah karya orang lain
2. Tidak melakukan pemalsuan data
3. Tidak menggunakan karya orang lain tanpa menyebut sumber asli atau tanpa ijin pemilik

Jika kemudian terbukti terjadi pelanggaran terhadap pernyataan di atas, maka saya bersedia menerima sanksi apapun termasuk pencabutan gelar akademik.

Lembar pernyataan ini juga memberikan hak kepada Politeknik Negeri Batam untuk mempergunakan, mendistribusikan ataupun memproduksi ulang seluruh hasil Tugas Akhir ini.

Batam, 16 Agustus 2018

Yoshua Fernandes Sirait
3311501050

ABSTRAK

IDENTIFIKASI *SYNTATICAL FAULT* DAN *SEMANTICAL FAULT* DALAM *EVENT LOG* MENGGUNAKAN *TRACE CLUSTERING*

Sebuah perusahaan atau organisasi sendiri biasanya terdapat beberapa proses bisnis, proses bisnis tersebut akan menghasilkan rekaman data dengan jumlah yang besar. Salah satu bentuk rekamannya adalah yang disebut sebagai *event log. Event log* ini dianalisis dengan teknik *process mining* dengan metode *Trace Clustering*. Tujuan dari *process mining* adalah mendapatkan informasi yang berguna dari *event log*, dan menggunakannya untuk menentukan *fault* atau kesalahan yang pernah terjadi.

Fault yang akan diidentifikasi yaitu kesalahan fungsional atau *functional fault*. *Functional fault* sendiri dibedakan menjadi dua jenis yaitu *syntactical fault* (yaitu *fault* yang terjadi karena urutan proses yang tidak sesuai yang telah ditetapkan dalam desain atau rancangan) dan *semantical fault* (yaitu *fault* yang terjadi karena *output* atau hasil tidak sesuai dengan desain atau rancangan). Sistem ini diharapkan dapat bermanfaat sebagai media informasi dan bagi perusahaan atau organisasi, sehingga mampu mengenal kesalahan-kesalahan yang sering terjadi pada proses bisnis.

Kata kunci: *Event Log, Semantical Fault,Syntatical Fault, Trace Clustering, process mining.*

ABSTRACT

SYNTATICAL FAULT AND SEMANTICAL FAULT IDENTIFICATION IN EVENT LOGS USING TRACE CLUSTERING

A company or an organization usually do several business processes. These business processes will produce large amount of data, usually called an event log. This event log is analyzed using process mining technique with Trace Clustering Method. The purpose of process mining is to obtain useful information from the event log and use it to determine the fault or error that has occurred.

Fault to be identified are functional errors or functional faults. Functional fault itself is divided into two types: the syntactical fault (i.e., the fault that occurred due to inappropriate process sequence that has been specified in the design) and semantical fault (i.e., the fault that occurred because the output or the result did not match the design). The system is expected to be useful as an information source for companies and organizations to recognize the errors that often occur in business processes.

Keywords: Event Log, Semantical Fault, Syntatical Fault, Trace Clustering, Process Mining

KATA PENGANTAR

Puji Syukur atas rahmat Tuhan Yang Maha Esa, karena atas rahmat dan hidayatnya penulis dapat menyelesaikan Tugas Akhir ini dengan judul ”Identifikasi *Syntatical Fault* dan *Semantical Fault* Dalam Event Log Menggunakan *Trace Clustering*”. Penulisan Tugas Akhir ini ditujukan untuk memenuhi salah satu persyaratan kelulusan program Diploma III pada jurusan Teknik Informatika di Politeknik Negeri Batam.

Selama proses pengerjaan tugas akhir ini penulis telah banyak menerima bantuan, petunjuk dan saran dari berbagai pihak. Pada kesempatan ini penulis ingin menyampaikan ucapan terima kasih yang sebesar-besarnya kepada semua pihak yang terlibat, antara lain:

1. Kedua orang tua yang selalu memberikan dorongan semangat dan doa-doa yang tulus.
2. Ibu Metta Santiputri, S.T., M.Sc, Selaku pembimbing dalam mengerjakan penulisan Laporan Tugas Akhir.
3. Seluruh rekan - rekan mahasiswa khususnya Teknik Informatika.
4. Seluruh staff pengajar Politeknik Negeri Batam yang telah memberikan materi perkuliahan sehingga menunjang pengerjaan Laporan Tugas Akhir.
5. Serta seluruh pihak yang telah memberikan bantuan yang berguna bagi kelancaran penyusunan Tugas Akhir ini.

Penulis menyadari sepenuhnya bahwa Laporan Tugas Akhir ini masih banyak kekurangan dalam penyusunannya, sehingga penulis sangat mengharapkan kritik dan saran yang bersifat membangun demi kesempurnaan penyusunan Laporan Tugas Akhir ini.

Batam, 16 Agustus 2018

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Inteligensi Bisnis (*Business Intelligence* atau IB) adalah sekumpulan teknik dan alat untuk mentransformasi dari data mentah menjadi informasi yang berguna dan bermakna untuk tujuan analisis bisnis. Teknologi IB dapat menangani data yang tak terstruktur dalam jumlah yang sangat besar untuk membantu mengidentifikasi, mengembangkan, serta membuka kesempatan strategi bisnis yang baru. Tujuan dari IB yaitu untuk memudahkan interpretasi dari jumlah data yang besar tersebut.Data yang digunakan dalam IB sendiri merupakan data historis yang merupakan rekaman aktivitas dalam sebuah enterprise atau organisasi.

Dalam suatu perusahaan atau organisasi sendiri biasanya terdapat beberapa proses bisnis, dimana pada pelaksanaan proses bisnis tersebut akan menghasilkan rekaman data dengan jumlah besar. Salah satu bentuk rekamannya adalah yang disebut sebagai *event log* yang merupakan rekaman eksekusi seluruh aktivitas di dalam sebuah *enterprise*. *Event log* tersebut merekam urutan atau rangkaian aktivitas yang terjadi.

Dari rekaman aktivitas yang terdapat dalam *event log* tersebut, dapat digali berbagai informasi yang berguna. Salah satu informasi yang bisa didapatkan adalah mengenai *fault* atau kesalahan yang pernah terjadi. Dalam Tugas Akhir ini, *fault* yang akan diidentifikasi yaitu kesalahan fungsional atau *functional fault*. *Functional fault* sendiri dibedakan menjadi dua jenis yaitu *syntactical fault* (yaitu fault yang terjadi karena urutan proses yang tidak sesuai yang telah ditetapkan dalam desain atau rancangan) dan *semantical fault* (yaitu fault yang terjadi karena *output* atau hasil tidak sesuai dengan desain atau rancangan).

Untuk mengidentifikasi kesalahan atau fault yang terjadi, maka *trace* atau urutan aktivitas yang direkam di dalam *event log* dikelompokkan (*trace clustering*). Sehingga untuk setiap *cluster* nantinya dapat ditentukan *fault* atau kesalahan yang terjadi serta penyebabnya.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut dapat dirumuskan masalah untuk membangun aplikasi tersebut adalah bagaimana cara mengindentifikasi kesalahan data dalam jumlah yang banyak?

1.3 Batasan Masalah

Dalam analisis sistem perancangan sistem informasi manajemen tugas proyek ada pembatasan masalah agar objek penelitian dapat terarah sesuai dengan apa yang dimaksud. Batasan masalah untuk aplikasi ini yaitu:

1. Mengidentifikasi dua jenis *fault* yaitu *Syntatical Fault* dan *Semantical Fault*
2. Sumber data berupa *Event Log*
3. Data dalam format CSV

1.4 Tujuan

Adapun tujuan yang ingin dicapai adalah untuk merancang dan membangun aplikasi yang memiliki fungsionalitas di antaranya:

a. Mendeteksi *Funcional Fault* berupa *Syntatical Fault* dan *Semantical Fault* dalam *Event Log*
b. Mengidentifikasi kesalahan yang berdasarkan rekaman dalam *event log*
c. Menyimpan data yang *fault* denga format CSV

1.5 Manfaat

Penelitian ini membantu menganalisis proses atau tahapan yang benar dan yang salah dari setiap alur kegiatan yang sudah dilakukan berdasarkan rekaman data dalam jumlah besar. Diharapkan identifikasi dan analisis terhadap kesalahan yang ditemukan tersebut dapat memperkecil kemungkinan terjadinya kesalahan yang sama pada masa yang akan datang.

1.6 Sistematika Penulisan

Sistematika penulisan tugas akhir ini disusun dengan urutan sebagai berikut:

BAB I : Pendahuluan. Berisi tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian dan sistematika penulisan untuk memberikan gambaran isi laporan tugas akhir.

BAB II : Landasan Teori, Berisi tentang ulasan penelitia-penelitian yang pernah dikerjakan sebelumnya sebagai referensi dari tugas akhir dan teori-teori yang mendukung pembuatan tugas akhir.

BAB III : Analisis dan perancangan . Berisi tentang analisis, perancangan sistem dan perancangan antarmuka tugas akhir ini

BAB IV : Implementasi dan Pengujian. Berisi tentang hasil implementasi, dan pengujian pada tugas akhir.

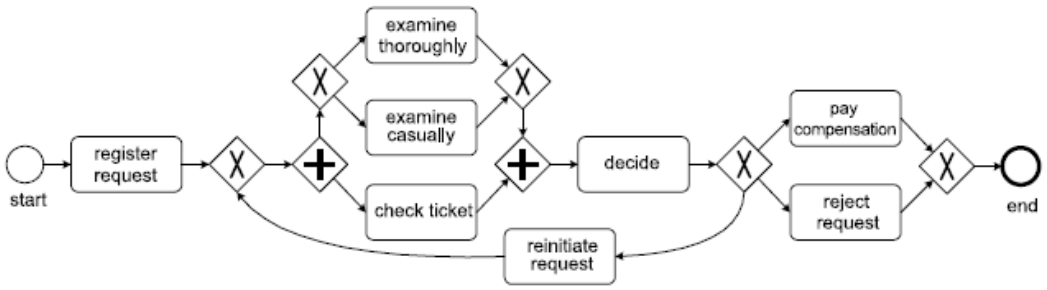
BAB V : Kesimpulan dan Saran. Berisi tentang kesimpulan dari tugas akhi beserta saran yang sifatnya membangun untuk pembuatan dan pengembangan aplikasi

BAB II
LANDASAN TEORI

2.1 Event Log

Tugas Akhir ini menggunakan sumber data berupa *event.* Keterangan mengenai *event log* ini di ambil dari buku van der Aalst (2016) dan jurnal (Ferreira, Diogo, et al. (2007): 360-374.), (Song, Minseok, Christian W. Günther, and Wil MP Van der Aalst (2008).

Gambar 1 memperlihatkan model proses yang mendeskripsikan penanganan permintaan kompensasi dalam sebuah perusahaan penerbangan. Pelanggan dapat mengajukan permintaan kompensasi dengan berbagai alasan, misalnya terjadi *delay* atau pembatalan penerbangan. Selain itu, proses ini dimulai dengan mendaftarkan permintaan. Aktivitas ini dinamakan dengan *register request.* Setelah permintaan didaftarkan, maka berikutnya adalah melakukan cek administrative untuk memeriksa apakah pelanggan tersebut berhak mengajukan permintaan kompensasi. Aktivitas ini dinamakan dengan *check ticket*. Salah satu dari dua aktivitas, yaitu (1) jika permintaan dinilai mencurigakan atau kompleks, maka dilakukan pemeriksaan secara mendalam (*examine thoroughly*), atau (2) jika permintaan dianggap tidak mencurigakan atau cukup jelas, maka dilakukan pemeriksaan secukupnya (*examine casually*). Hasilnya kemudian diputuskan dalam aktivitas *decide*. Ada tiga kemungkinan keputusan, yaitu permintaan kompensasi dipenuhi (*pay compensation*), permintaan ditolak (*reject request*), atau diperlukan pemrosesan lebih lanjut (*reinitiate request*). Jika diperlukan pemrosesan lebih lanjut, maka proses kembali ke setelah *register request*, atau dengan kata lain, tidak diperlukan pendaftaran lagi. Proses akan berakhir ketika kompensasi dibayarkan atau ditolak.



Gambar 1. Model proses penanganan permintaan kompensasi

Tabel 1 memperlihatkan cuplikan log yang merekam proses penanganan permintaan kompensasi tersebut. Setiap baris mewakili satu *event*. Beberapa *event* dikelompokkan menjadi sebuah *case*. *Case* 1 memiliki lima *event*. *Event* pertama dalam *case* 1 adalah pelaksanaan aktivitas mendaftarkan permintaan (*register request*) oleh Pete pada 30 Desember 2010. Tabel 1 juga menunjukkan id khusus untuk *event* ini: 35654423. Id ini digunakan agar dapat mengidentifikasi sebuah *event*, misalnya untuk membedakan *event* tersebut dengan *event* 35654483 yang juga merupakan pelaksanaan aktivitas *register request* (*event* pertama dari *case* kedua).Selain itu dalam tabel ini juga menyajikan tanggal dan waktu (*timestamp*) untuk setiap *event*. Ada kemungkinan dalam beberapa *event log*, informasi ini tidak diberikan secara detail dan hanya tanggal atau urutan *event* yang ada. Namun mungkin juga dalam kasus lain, terdapat informasi waktu yang lebih detil seperti kapan sebuah aktivitas dimulai, kapan selesai, serta kapan hasilnya ditampilkan pada *user*. Kolom waktu atau *timestamp* menunjukkan waktu selesainya sebuah aktivitas. Dalam *event log* ini, aktivitas dianggap atomik dan Tabel 1 tersebut tidak merekam durasi aktivitas. Dalam tabel tersebut setiap *event* berhubungan dengan sebuah *resource*. Dalam beberapa log, mungkin saja informasi ini tidak tersedia. Sedangkan dalam log yang lain, terdapat informasi yang mendetail mengenai *resource*, misalnya role atau jabatan dari *resource* tersebut atau mengenai autorisasi yang terjadi antar *resource*. Tabel 1 juga menyajikan informasi mengenai *cost* atau biaya untuk setiap event. Informasi ini merupakan contoh atribut data. Ada banyak atribut data yang lain, seperti *outcome* atau hasil dari setiap *event*se, serta dapat

disertakan jumlah kompensasi yang diminta. Atribut ini dapat merupakan atribut dari seluruh *case* atau disimpan sebagai atribut dari event *register request*.

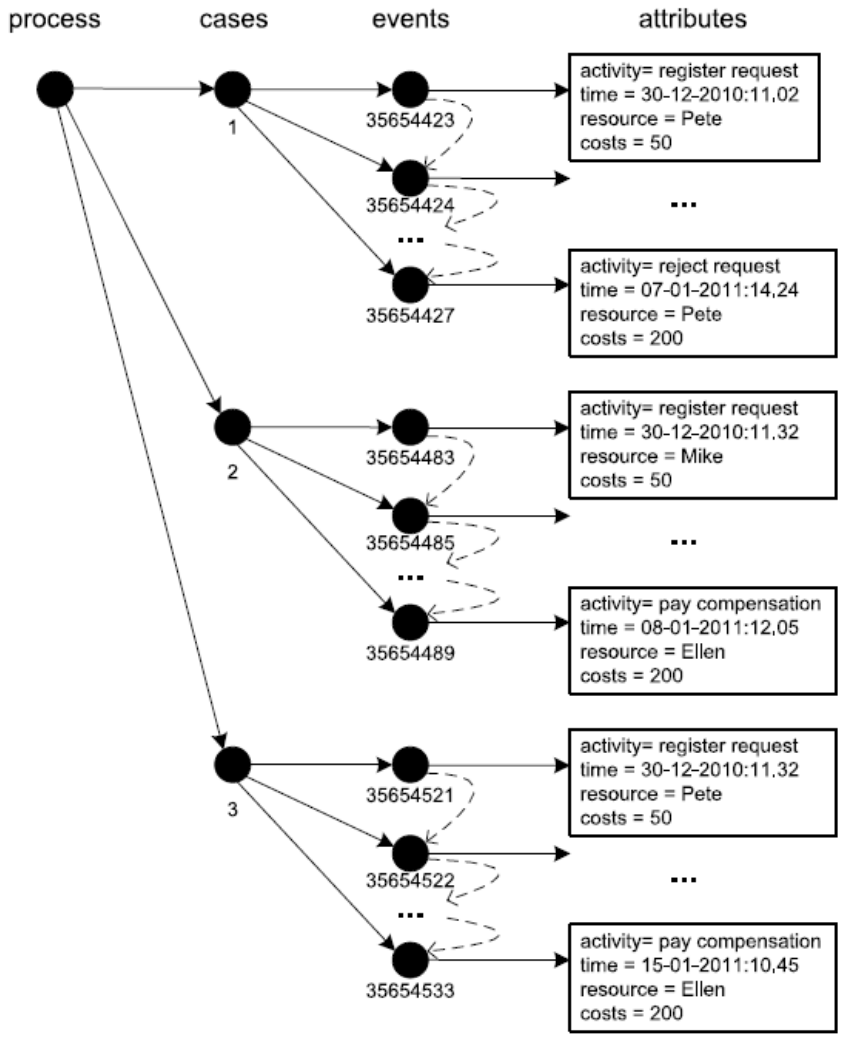
Tabel 1. Cuplikan sebuah *event log*

Case id	Event id	Properties			
		Timestamp	Activity	Resource	Cost
1	35654423	30-12-2012:11.02	Register request	Pete	50
	35654424	31-12-2010:10.06	Examine thoroughly	Sue	400
	35654425	05-01-2011:15.12	Check ticket	Mike	100
	35654426	06-01-2011:11.18	Decide	Sara	200
	35654427	07-01-2011:14.24	Reject request	Pete	200
2	35654483	30-12-2010:11.32	Register request	Mike	50
	35654485	30-12-2010:12.12	Check ticket	Mike	100
	35654487	30-12-2010:14.16	Examine casually	Pete	400
	35654488	05-01-2011:11.22	Decide	Sara	200
	35654489	08-01-2011:12.05	Pay compensation	Ellen	200
3	35654521	30-12-2010:14.32	Register request	Pete	50
	35654522	30-12-2010:15.06	Examine casually	Mike	400
	35654524	30-12-2010:16.34	Check ticket	Ellen	100
	35654525	06-01-2011:09.18	Decide	Sara	200
	35654526	06-01-2011:12.18	Reinitiate request	Sara	200
	35654527	06-01-2011:13.06	Examine thoroughly	Sean	400
	35654530	08-01-2011:11.43	Check ticket	Pete	100
	35654531	09-01-2011:09.55	Decide	Sara	200
	35654533	15-01-2011:10.45	Pay compensation	Ellen	200
4	35654641	06-01-2011:15.02	Register request	Pete	50
	35654643	07-01-2011:12.06	Check ticket	Mike	100
	35654644	08-01-2011:14.43	Examine thoroughly	Sean	400
	35654645	09-01-2011:12.02	Decide	Sara	200
	35654647	12-01-2011:15.44	Reject request	Ellen	200

Case id	Event id	Properties			
		Timestamp	Activity	Resource	Cost
5	35654711	06-01-2011:09.02	Register request	Ellen	50
	35654712	07-01-2011:10.16	Examine casually	Mike	400
	35654714	08-01-2011:11.22	Check ticket	Pete	100
	35654715	10-01-2011:13.28	Decide	Sara	200
	35654716	11-01-2011:16.18	Reinitiate request	Sara	200
	35654718	14-01-2011:14.33	Check ticket	Ellen	100
	35654719	16-01-2011:15.50	Examine casually	Mike	400
	35654720	19-01-2011:11.18	Decide	Sara	200
	35654721	20-01-2011:12.48	Reinitiate request	Sara	200
	35654722	21-01-2011:09.06	Examine casually	Sue	400
	35654724	21-01-2011:11.34	Check ticket	Sara	100
	35654725	23-01-2011:13.12	Decide	Sara	200
	35654726	24-01-2011:14.56	Reject request	Mike	200
6	35654871	06-01-2011:15.02	Register request	Mike	50
	35654873	06-01-2011:16.06	Examine casually	Ellen	400
	35654874	07-01-2011:16.22	Check ticket	Mike	100
	35654875	07-01-2011:16.52	Decide	Sara	200
	35654877	16-01-2011:11.47	Pay compensation	Mike	200

Tabel 1 memperlihatkan informasi yang biasanya ada dalam sebuah *event log*. Informasi bagian mana yang digunakan tergantung pada teknik dan pertanyaan yang akan dijawab. Gambar 2 memperlihatkan struktur dari sebuah *event log*, yaitu:

- Sebuah *proses* terdiri dari banyak *case*.
- Sebuah *case* terdiri dari banyak *event*, dimana setiap *event* terhubung dengan satu *case*.
- *Event* di dalam *case* terurut.
- *Event* dapat memiliki atribut, contohnya aktivitas, waktu, biaya, dan *resource*.



Gambar 2. Struktur sebuah *event log*.

Informasi yang mewakili aturan minimum tersebut adalah "*case id*" dan "*activity*" seperti terlihat dalam Tabel 1. Dengan menggunakan kedua informasi tersebut, maka kita mendapatkan bentuk yang lebih singkat seperti diperlihatkan dalam Tabel 2. Dalam tabel tersebut, setiap *case* diperlihatkan sebagai urutan aktivitas yang disebut sebagai *trace*. Untuk mempermudah, setiap nama aktivitas telah diubah menjadi label yang terdiri dari satu huruf, misalnya *a* melambangkan aktivitas *register request*.

Tabel 2. Representasi yang lebih singkat dari log di Tabel 1

Case id	Trace
1	<*a, b, d, e, h*>
2	<*a, d, c, e, g*>
3	<*a, c, d, e, f, b, d, e, g*>
4	<*a, d, b, e, h*>
5	<*a, c, d, e, f, d, c, e, f, c, d, e, h*>
6	<*a, c, d, e, g*>

a = *register request*, b = *examine thoroughly*, c = *examine casually*, d = *check ticket*, e = *decide*, f = *reinitiate request*, g = *pay compensation*, dan h = *reject request*

Table 1 juga mengandung informasi mengenai resource, waktu (*timestamp*) dan biaya (*cost*). Informasi tersebut dapat digunakan untuk mengungkapkan hal-hal lain terkait dengan proses. Sebagai contoh, kita dapat memperoleh *social network* atau jaringan sosial berdasarkan pola interaksi antar individual yang terlibat. Jaringan sosial ini bisa berdasarkan metrik “pendelegasian tugas”, misalnya semakin sering individu *x* melakukan sebuah aktivitas diikuti dengan sebuah aktivitas lain yang dilakukan oleh individu *y*, maka hubungan antara *x* dengan *y* juga makin kuat. (van der Aalst, Wil. "Data Science in Action." *Process Mining*. Springer Berlin Heidelberg, 2016. 3-23.)

2.2 Penelitian Terkait

Log konsisten dengan proses yang disebutkan di atas. Setiap baris mengacu pada satu kasus dan direpresentasikan sebagai urutan kejadian. Perhatikan bahwa kita menggunakan istilah "*case id*" untuk merujuk pada baris tertentu dan istilah "*trace*" ke urutan kejadian dalam sebuah kasus. Peristiwa ditunjukkan oleh identifikasi kasus (dilambangkan dengan baris, misalnya, 1), identifikasi aktivitas (elemen pertama, misal A), dan originator (elemen kedua, misalnya John).

Berdasarkan log ini, algoritma α secara otomatis menyusun model jaring Petri yang digambarkan pada gambar di bawah ini.

Case ID	log events
1	(A,John),(B,Mike),(D,Sue),(E,Pete),(F,Mike),(G,Jane),(I,Sue)
2	(A,John),(B,Fred),(C,John),(D,Clare),(E,Robert),(G,Mona),(I,Clare)
3	(A,John),(B,Pete),(D,Sue),(E,Mike),(F,Pete),(G,Jane),(I,Sue)
4	(A,John),(C,John),(B,Fred),(D,Clare),(H,Clare),(I,Clare)
5	(A,John),(C,John),(B,Robert),(D,Clare),(E,Fred),(G,Robert),(I,Clare)
6	(A,John),(B,Mike),(D,Sue),(H,Sue),(I,Sue)

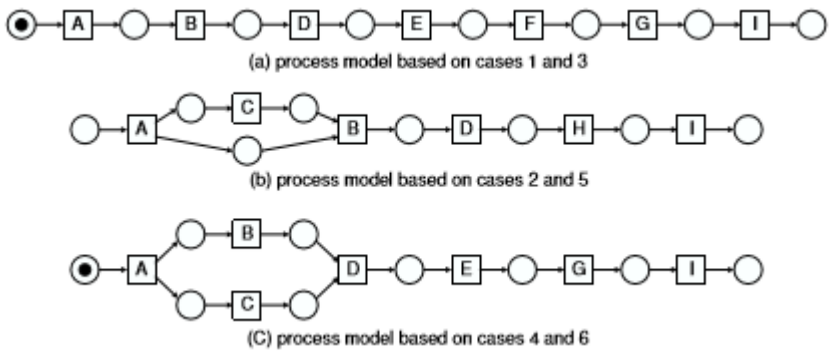
Gambar 3. Contoh proses log

(A: Receive a item and repair request, B: Check the item, C: Check the warranty, D: Notify the customer, E: Repair the item, F: Test the repaired product, G: Issue payment, H: Send the cancellation letter, I: Return the item).

Jika kita membagi log menjadi beberapa subkelompok dan menyusun model proses, kita bisa mendapatkan model yang lebih akurat. Misalnya, berdasarkan tugas, log bisa dibagi menjadi tiga kelompok.

- Kelompok pertama terdiri dari kasus-kasus di mana sistem navigasi perlu diperbaiki (yaitu, kasus 1 dan 3), yaitu kasus-kasus di mana tugas "Periksa garansi" hilang namun dengan tugas "Uji produk yang diperbaiki".
- Kelompok kedua sesuai dengan proses memperbaiki ponsel (yaitu, kasus 2 dan 5). Kasus-kasus ini tidak memiliki tugas "Uji produk yang diperbaiki", namun mintalah tugas "Periksa garansi".
- Kelompok ketiga sesuai dengan kasus di mana perbaikan dibatalkan, yaitu kasus 4 dan kasus 6 milik kelompok ini.

Gambar 4 menunjukkan model proses dari masing-masing kelompok yang dibangun menggunakan algoritma α. Kesesuaian dan ketepatan perilaku dari ketiga model ini adalah 1.000. Ini menunjukkan bahwa pengelompokan trace dapat mendukung identifikasi varian proses yang sesuai dengan subkumpulan kasus yang homogen. Apalagi model yang dibangun memiliki kualitas yang jauh lebih baik.



Gambar 4. Model proses turunan dari tiga kelompok.

Kita bisa mempertimbangkan sebuah profil dengan n item menjadi fungsi, yang menugaskan untuk melacak sebuah vektor hi1, i2..., ini. Proviser log dapat digambarkan sebagai mengukur sekumpulan *trace* dengan sejumlah profil, menghasilkan vektor agregat (berisi nilai untuk setiap item terukur dalam beberapa urutan yang ditentukan). Vektor yang dihasilkan ini kemudian dapat digunakan untuk menghitung jarak antara dua *trace*, dengan menggunakan metrik jarak.

Gambar 5 menunjukkan hasil dari contoh log contoh dari Gambar 4 dengan dua profil. *Activity Profile* menentukan satu item per jenis aktivitas (yaitu, nama *event*) yang ditemukan di log. Mengukur item aktivitas dilakukan dengan hanya menghitung semua jejak peristiwa yang memiliki nama aktivitas tersebut. *Originator Profile*, yang juga ditunjukkan dalam contoh ini, serupa dengan *Activity Profile*, Item *Originator log*, menghitung berapa banyak *event* yang disebabkan oleh masing-masing *Originator* per *trace*. Setiap baris tabel sesuai dengan vektor profil dari satu *trace* di log.

Case ID	Activity Profile									Originator Profile								
	A	B	C	D	E	F	G	H	I	John	Mike	Sue	Pete	Jane	Fred	Clare	Robert	Mona
1	1	1	0	1	1	1	1	0	1	1	2	2	1	1	0	0	0	0
2	1	1	1	1	1	0	1	0	1	2	0	0	0	0	1	2	1	1
3	1	1	0	1	1	1	1	0	1	1	1	2	2	1	0	0	0	0
4	1	1	1	1	0	0	0	1	1	2	0	0	0	0	1	3	0	0
5	1	1	1	1	1	0	1	1	0	2	0	0	0	0	1	2	2	0
6	1	1	0	1	0	0	0	1	1	1	1	3	0	0	0	0	0	0

Gambar 5. Profil aktivitas dan originator untuk contoh log dari Gambar 3

(Song, Minseok, Christian W. Günther, and Wil MP Van der Aalst. "Trace clustering in process mining." *International Conference on Business Process Management*. Springer, Berlin, Heidelberg, 2008)

2.3 Dasar Teori

2.3.1 Netbeans

Netbeans sebagai IDE ditujukan untuk memudahkan pemrograman Java. Pada bulan Februari 2006 para instruktur Java dari Sun Microsystem mengikuti training untuk beralih dari pemrograman Java manual (memakai editor teks dan *command prompt*) ke pemrograman GUI dengan Netbeans. Netbeans berbasis visual dan *event-driven*. Sama seperti IDE lainnya, misal *Borland Delphi* dan Microsoft Visual Studio. Netbeans mencakup *compiler*, *builder* dan *debugger internal*. Hal ini memudahkan proses pasca perancangan program. Proses *deployment* atau tes dapat dilakukan dengan *Netbeans*.

Netbeans seperti juga konsep Java sangat fleksibel. Sepanjang library Java tersedia, maka kita dapat melakukan pemrograman untuk jenis aplikasi apapun. Kita dapat membuat aplikasi dekstop (J2SE), Pemrograman web dan enterprise (J2EE) dapat dilakukan secara visual:

a. Koneksi server database melalui JDBC dapat dilakukan dari Netbeans, baik pada saat perancangan maupun deployment program

b. Pembuatan komponen beans

c. Pembuatan *Java Server Pages* (JSP), web module (*servicelocator* dan *servlet*), web *service*s dengan menggunakan wizard yang telah disediakan.

Netbeans juga menyertakan paket *web-server Apache Tomcat, Sun Java System Application Server, GlassFish* dll. Server ini dapat diakses dari dalam Netbeans, baik pada saat perancangan maupun saat *deployment* aplikasi web.

2.3.2 Java

Java adalah bahasa pemrograman yang berorientasi objek yang menggunakan

bahasa sederhana untuk dijalankan diberbagai *platform.* Java bersifat *opensource*, yang artinya java bisa berjalan pada berbagai sistem operasi dan dapat dikembangkan atau dimodifikasi. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model objek yang lebih sederhana serta dukungan rutin-rutin aras bawah yang minimal. Aplikasi-aplikasi berbasis java umumnya dikomplikasi ke dalam *p-code (bytecode)* dan dapat dijalankan pada berbagai *Mesin Virtual Java* (JVM).

2.3.3 MariaDB

MariaDB merupakan versi pengembangan terbuka dan mandiri dari MySQL. Sejak diakuisisinya MySQL oleh Oracle pada September 2010, Monty Program sebagai penulis awal kode sumber MySQL memisahkan diri dari pengembangan dan membuat versi yang lebih mandiri yakni MariaDB. MariaDB adalah sebuah implementasi dari sistem manajemen basisdata relasional (RDBMS) yang didistribusikan secara gratis dibawah lisensi GPL (General Public License). Setiap pengguna dapat secara bebas menggunakan MariaDB, namun dengan batasan perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersial.

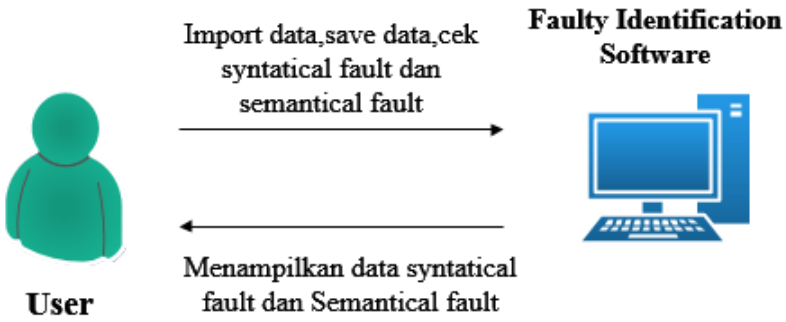
BAB III
ANALISIS DAN PERANCANGAN

3.1 Analisis Sistem

Analisis merupakan sebuah kegiatan penguraian atau penyelidikan terhadap suatu pokok masalah untuk memperoleh sebuah pemahaman, pengertian, dan arti sebenarnya dari suatu permasalahan. Analisis memiliki peranan terhadap pembangunan sebuah sistem agar sesuai dengan kebutuhan pengguna dan dapat dimanfaatkan secara opimal.

3.1.1 Deskripi Umum Sistem

Pada Gambar 6 menjelaskan cara kerja *Faulty Identification Software*.



Gambar 6. Deskripsi Umum *Faulty Identification Software.*

Deskripsi sistem pada gambar 6 menjelaskan proses kerja sistem aplikasi yang meliputi:

1. User dapat meng-*import* data dalam format CSV, menyimpan data ke dalam format CSV dan cek data syntactical fault dan semantical fault.
2. Aplikasi mengidentifikasi *fault* dan jika ada *fault* terdektekis akan menampilkan data yang *fault*

3.1.2 Format Input

Pada Gambar 7 adalah salah satu contoh data *event log* yang akan diidentifikasi menggunakan aplikasi yang akan dibuat.

Case ID	Event ID	Timestamp	Activity	Resource	Costs	Outcome
1	35654423	30-12-2010:11.02	register request	Pete	50	Request registered
1	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	Request examined
1	35654425	05-01-2011:15.12	check ticket	Mike	100	Ticket checked
1	35654426	06-01-2011:11.18	decide	Sara	200	Decided
1	35654427	07-01-2011:14.24	reject request	Pete	200	Requested rejected
2	35654483	30-12-2010:11.32	register request	Mike	50	Request registered
2	35654485	30-12-2010:12.12	check ticket	Mike	100	Ticket checked
2	35654487	30-12-2010:14.16	examine casually	Sean	400	Request examined
2	35654488	05-01-2011:11.22	decide	Sara	200	decided
2	35654489	08-01-2011:12.05	pay compensation	Ellen	200	Compensation paid
3	35654521	30-12-2010:14.32	register request	Pete	50	Request registered
3	35654522	30-12-2010:15.06	examine casually	Mike	400	Request examined
3	35654524	30-12-2010:16.34	check ticket	Ellen	100	Ticket checked
3	35654525	06-01-2011:09.18	decide	Sara	200	Decided
3	35654526	06-01-2011:12.18	reinitiate request	Sara	200	request initiated
3	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	Request examined
3	35654530	08-01-2011:11.43	check ticket	Pete	100	Ticket checked
3	35654531	09-01-2011:09.55	decide	Sara	200	decided
3	35654533	15-01-2011:10.45	pay compensation	Ellen	200	Compensation paid
4	35654641	06-01-2011:15.02	register request	Pete	50	Request registered
4	35654643	07-01-2011:12.06	check ticket	Mike	100	Ticket checked
4	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	Request registered

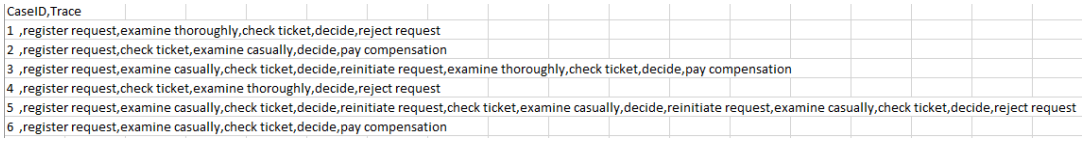
Gambar 7. Cuplikan sebuah data dalam *event log*

Pada format input dengan format CSV terdiri dari :

1. *Case id* merupakan sebuah angka yang digunakan untuk memberi atau mengidentifikasi sebuah *event*
2. *Event* id digunakan agar dapat mengidentifikasi sebuah *event*,
3. *Timestamp* (dd-MM-yyyy:HH.mm) menunjukkan waktu sebuah aktivitas,
4. *Activity* merupakan rangakaian kegiatan,
5. *Resource* merupakan nama-nama orang yang bertanggung jawab dalam pelaksanaan dalam suatu kegiatan,
6. *Costs* harga dalam sebuah *event*
7. *Outcome* merupakan hasil dari setiap *event*

3.1.3 Format Output

Pada Gambar 8 adalah salah satu contoh *output* dalam aplikasi



CaseID,Trace
1 ,register request,examine thoroughly,check ticket,decide,reject request
2 ,register request,check ticket,examine casually,decide,pay compensation
3 ,register request,examine casually,check ticket,decide,reinitiate request,examine thoroughly,check ticket,decide,pay compensation
4 ,register request,check ticket,examine thoroughly,decide,reject request
5 ,register request,examine casually,check ticket,decide,reinitiate request,check ticket,examine casually,decide,reinitiate request,examine casually,check ticket,decide,reject request
6 ,register request,examine casually,check ticket,decide,pay compensation

Gambar 8. Hasil Trace

Pada hasil output terdiri dari :

1. *Case id* ialah beberapa *event* dikelompokkan menjadi sebuah *case*.
2. *Trace* menunjukkan urutan dalam sebuah aktivitas (*event*) dalam *case* tersebut

3.1.4 Analisis Kebutuhan Sistem

Tabel 3 dan Tabel 4 menyajikan analisis kebutuhan sistem yang terdiri dari dua yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

1. **Spesifikasi Perangkat Keras**

Jenis Perangkat	Spesifikasi
Processor	AMD A8
Harddisk	1 TB
Memory	4 GB

Tabel 3. Spesifikasi Perangkat Keras

2. **Spesifikasi Perangkat Lunak**

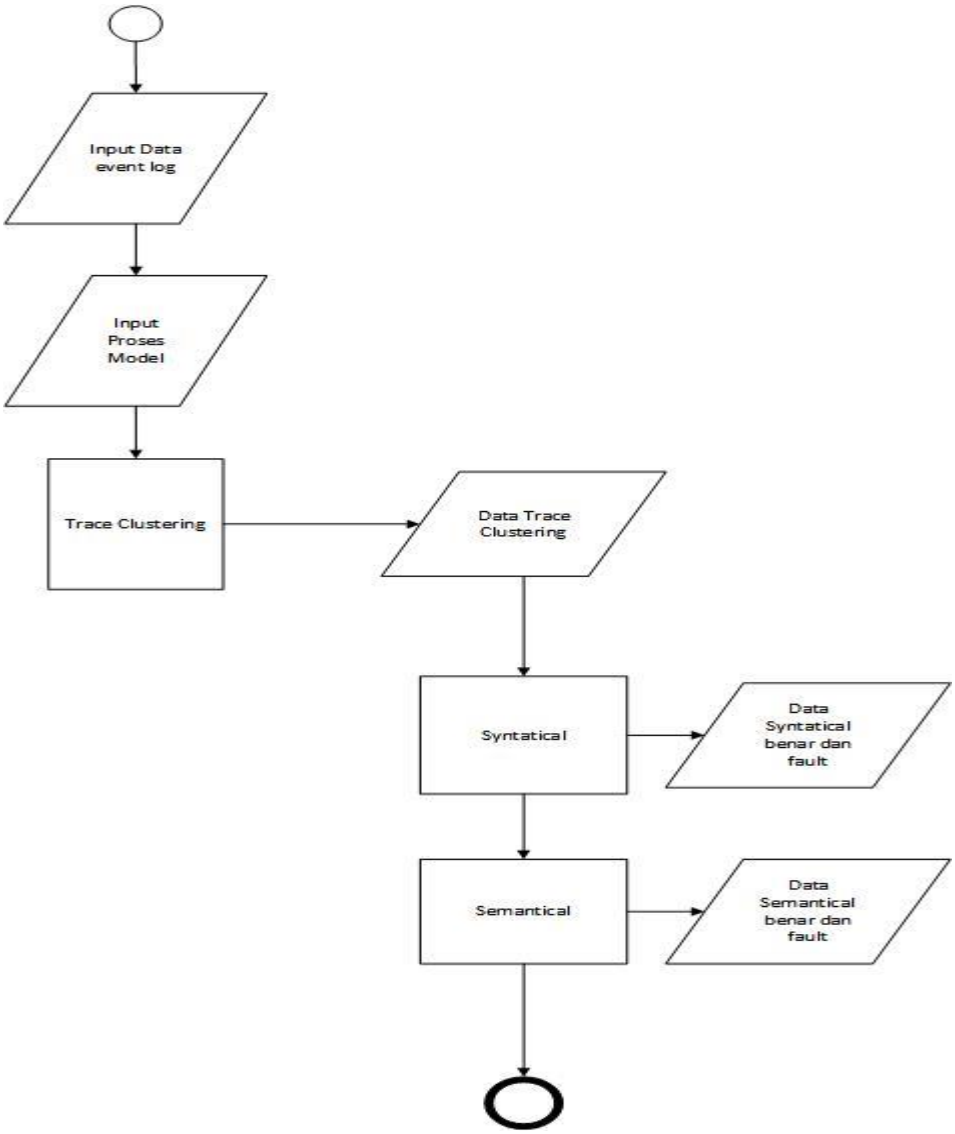
Jenis Perangkat Lunak	Nama Perangkat Lunak
Web Server dan Database	XAMPP
Dokumentasi	MS.Word,Excel,Visio
Teks Editor	Notepad++
Browser	Mozila Firefox
Aplikasi Pendukung Lainnya	Icons8

Tabel 4. Spesifikasi Perangkat Lunak

3.2 Perancangan Sistem

3.2.1 Flowchart

Pada Gambar 9 adalah alur kerja aplikasi dalam bentuk *flowchart*

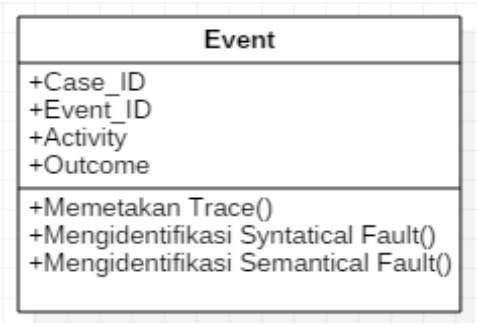


Gambar 9. Alur Kerja Aplikasi Dalam Bentuk *Flowchart*

Gambar 9 memperlihatkan alur kerja aplikasi berjalan, pertama memasukkan data dan proses model lalu akan melakukan proses *trace clustering* sehingga data yang telah di *cluster* akan diidentifikasi *syntactical fault* dan *semantical fault*.

3.2.2 Rancangan Kelas

4 Pada perancangan kelasnya hanya menggunakan satu kelas dalam pembuatan aplikasi dapat dilihat pada Gambar 10.



Gambar 10. Rancangan Kelas

4.2.1 Algoritma

1. Untuk melakukan identifikasi *trace* dalam *event log*

   Trace bertujuan agar data tidak terpisah dengan mengelompokkan berdasarkan *id case* yang sama. Dengan cara memeriksa *id case* data masukan. Jika mempunyai *id case* yang sama akan di bentuk menjadi satu trace. Contohnya *case id 1* maka setiap case id yang bernomor 1 akan merupakan satu trace.

   Algoritma Untuk Mengindentifikasi *Trace*

   Input : *Event Log*

   Output : *Trace*

```
For setiap event di dalam event log do
    If case id yang sama then
        Tambahkan log ke dalam satu
        kelompok trace
    End
End
```

2. Algoritma untuk mengidentifikasi apakah dalam sebuah *trace* terdapat *fault* Jika atribut *event* atau *trace* tidak memenuhi Proses Model, maka *trace* tersebut dinyatakan mengandung *fault.* Untuk menentukan penyebab *fault*, maka harus

diperiksa atribut dari setiap *event* atau aktivitas sehingga dapat ditentukan *event* mana yang menyebabkan terjadinya *fault* tersebut.

Algoritma untuk menentukan *syntatical fault*

Input : *Trace* dan Proses Model

Output : *Trace* dengan *Syntatical Fault*

```
For setiap trace T1...Tn do
    If trace tidak sesuai
        dengan proses model then
        trace mengandung fault
    End
End
```

Algoritma untuk menentukan *semantical fault*

Input : *Activity*

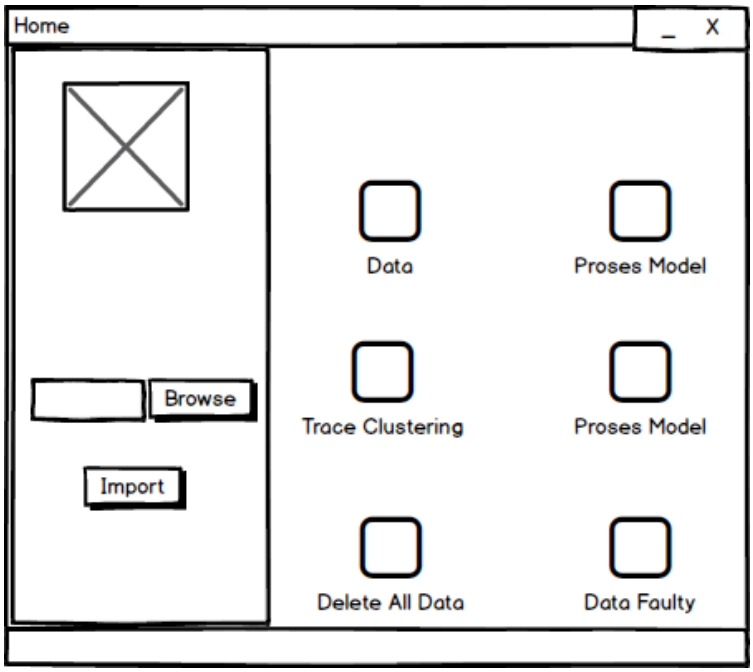
Output : *Trace* dengan *Semantical Fault*

```
For setiap activity A1...An do
   setiap outcome O1...On
    If trace tidak sesuai
        dengan proses output then
        data mengandung fault
    End
End
```

3.3 Rancangan Antar Muka

Aplikasi ini akan dibangun sedemikian rupa dengan perancangan antarmuka. Terdapat 7 perancangan antarmuka dari aplikasi yang akan dibangun.

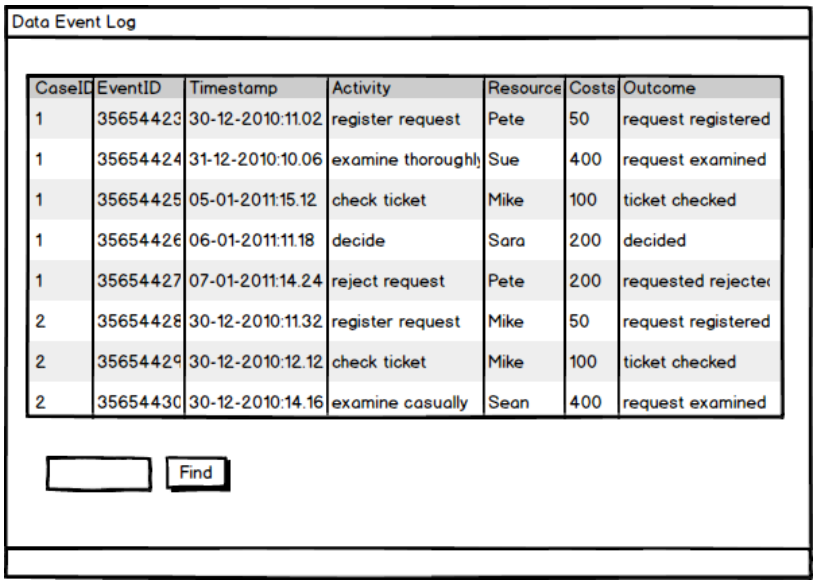
3.3.1 Antarmuka Beranda



Gambar 11. Rancangan Antarmuka Beranda

Pada halaman ini dirancang sederhana agar pengguna dapat dengan mudah menggunakan aplikasi tersebut. Tombol untuk meng-*import* data berada di samping agar lebih praktis.

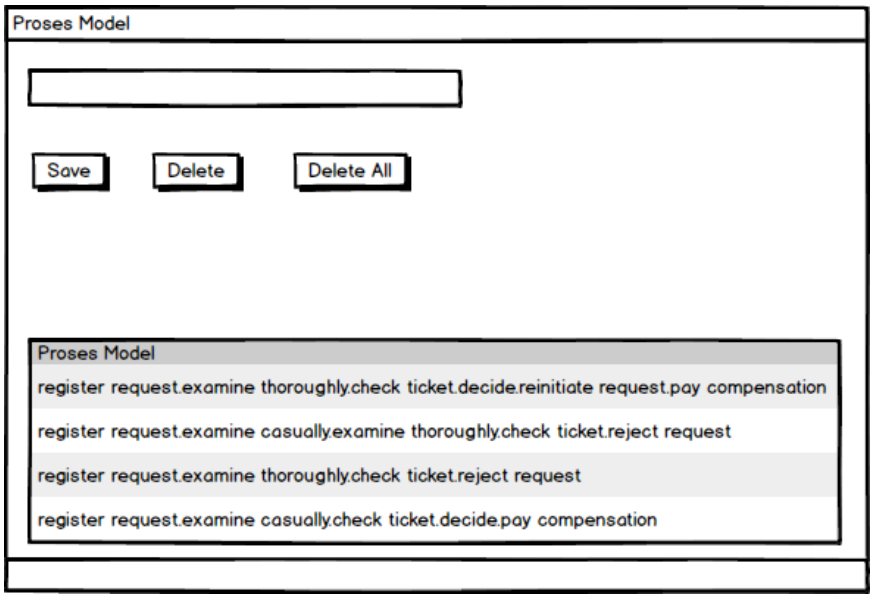
3.3.2 Antarmuka Data Event Log



Gambar 12. Rancangan Antarmuka Data Event Log

Pada halaman ini dirancang untuk menampilkan seluruh data yang telah di masukkan ke dalam aplikasi. Terdapat tombol cari untuk memudahkan pencarian bila diperlukan.

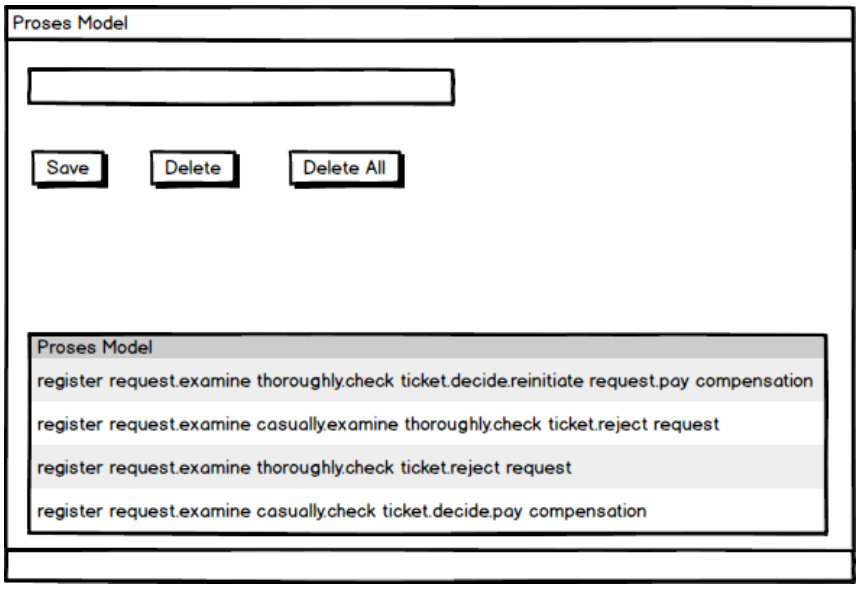
3.3.3 Antarmuka Proses Model Syntatical



Gambar 13. Rancangan Antarmuka Proses Model Syntatical

Pada halaman ini dirancang untuk memasukkan proses model untuk data *syntatical* dan menampilkannya.

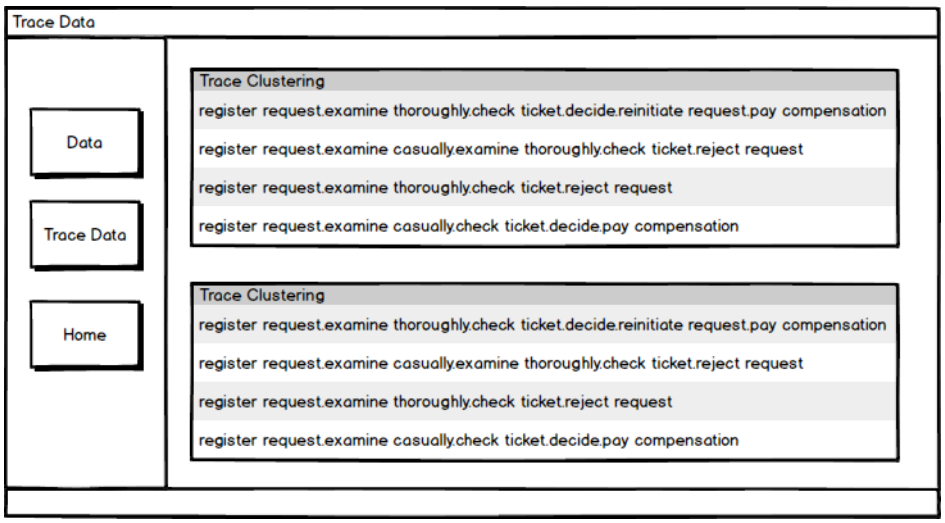
3.3.4 Antarmuka Proses Model Semantical



Gambar 14. Rancangan Antarmuka Proses Model Semantical

Pada halaman ini dirancang untuk memasukkan proses model untuk data *semantical* dan menampilkannya.

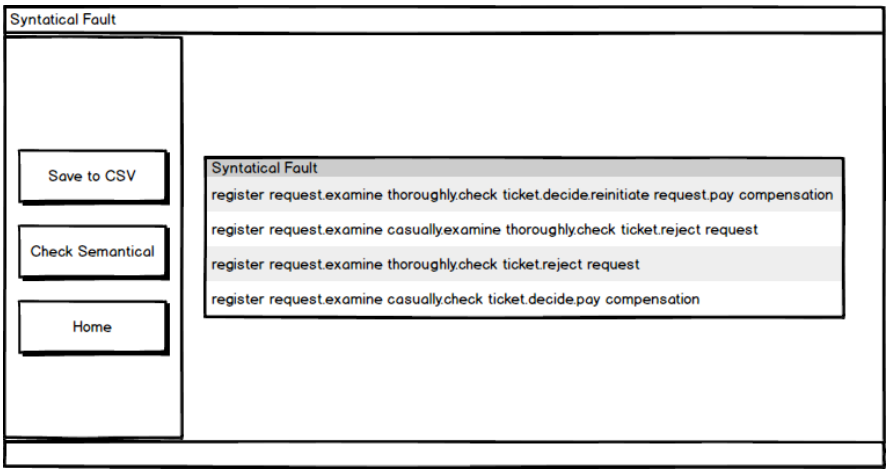
3.3.5 Antarmuka Trace Clustering



Gambar 15. Rancangan Antarmuka Trace Clustering

Pada halaman ini dirancang untuk malukakan dan menampilkan *Trace Clustering* sebelum melakukan identifikasi *syntactical fault* dan *semantical fault*.

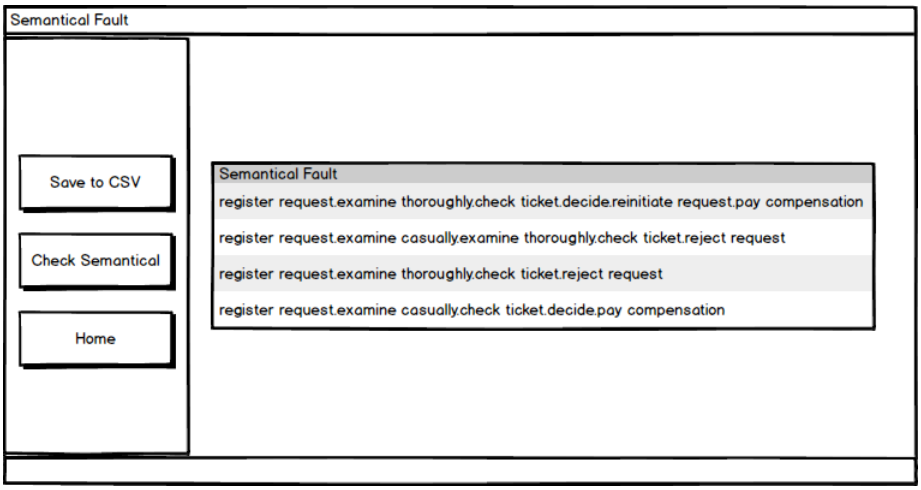
3.3.6 Antarmuka Syntatical Fault



Gambar 16. Rancangan Antarmuka Syntatical Fault

Pada halaman ini dirancang untuk menampilkan data *syntactical fault*.

3.3.7 Antarmuka Semantical Fault



Gambar 17. Rancangan Antarmuka Semantical Fault

Pada halaman ini dirancang untuk menampilkan data *semantical fault*.

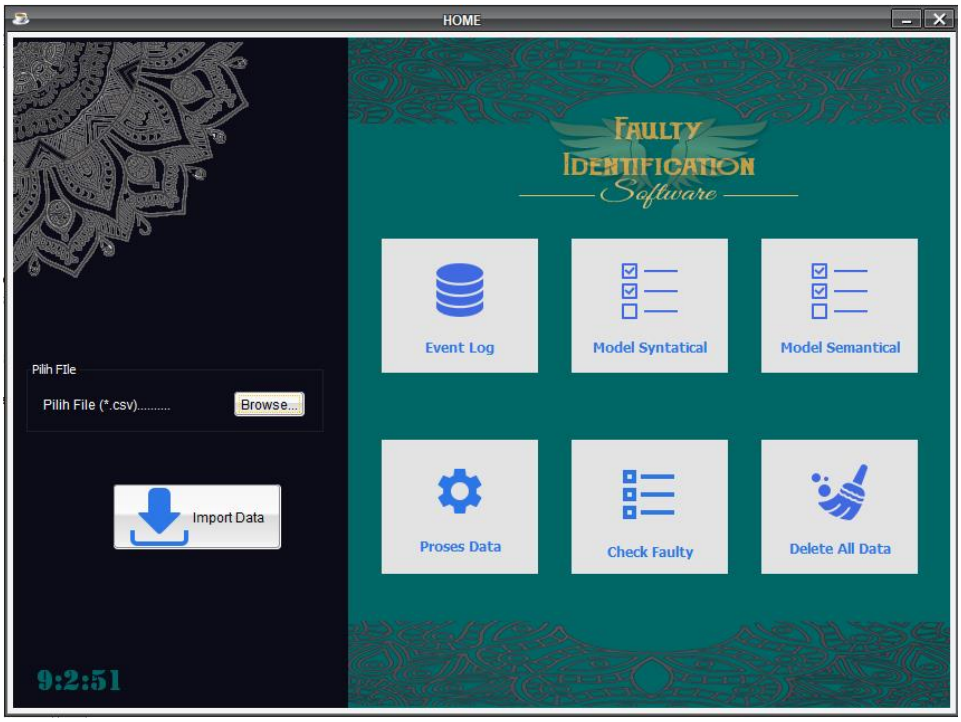
BAB IV
HASIL DAN PEMBAHASAN

Setelah melakukan tahap perancangan maka tahap selanjutnya adalah implementasi dan pengujian terhadap produk. Implementasi dari tahap perancangan tersebut adalah sebagai berikut:

4.1 Implementasi Antarmuka

4.1.1 Halaman Beranda

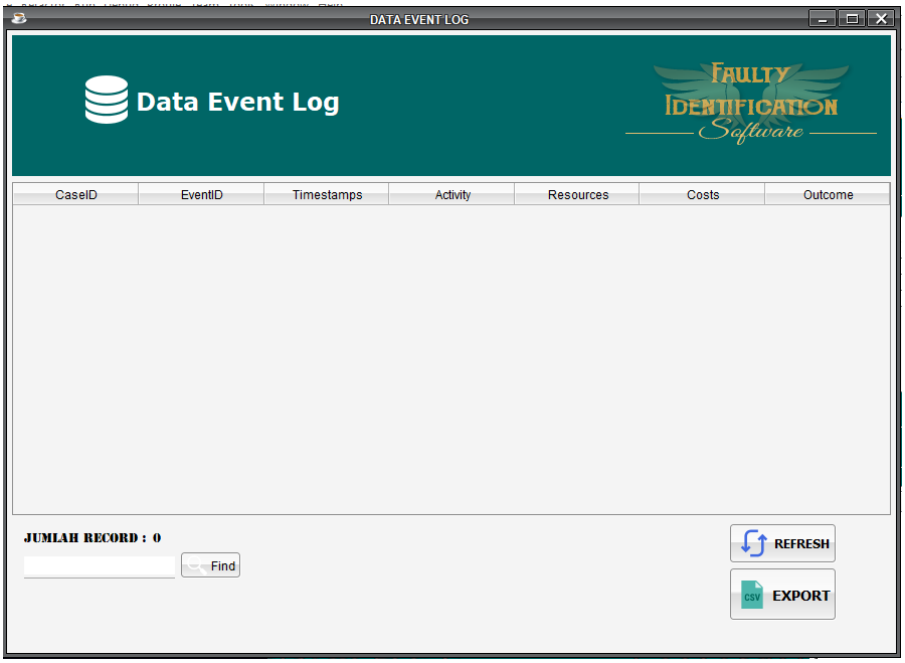
Pada halaman beranda terdapat tombol untuk meng-*import* data *event log* ke dalam aplikasi dalam format csv.



Gambar 18. Implementasi Antarmuka Beranda

Pada halaman ini dibuat sederhana namun tetap memiliki keindahan dan seni dari segi tampilannya. Terdapat enam tombol utama untuk memproses data, di sebelah kiri ditempatkan khusus untuk meng-*import* data. Enam tombol utama di buat berurut yang dimulai dari kiri tombol "Event Log" hingga tombol "Delete All Data" agar memudahkan pengguanan yang sesui dengan alur proses yang di tetapkan.

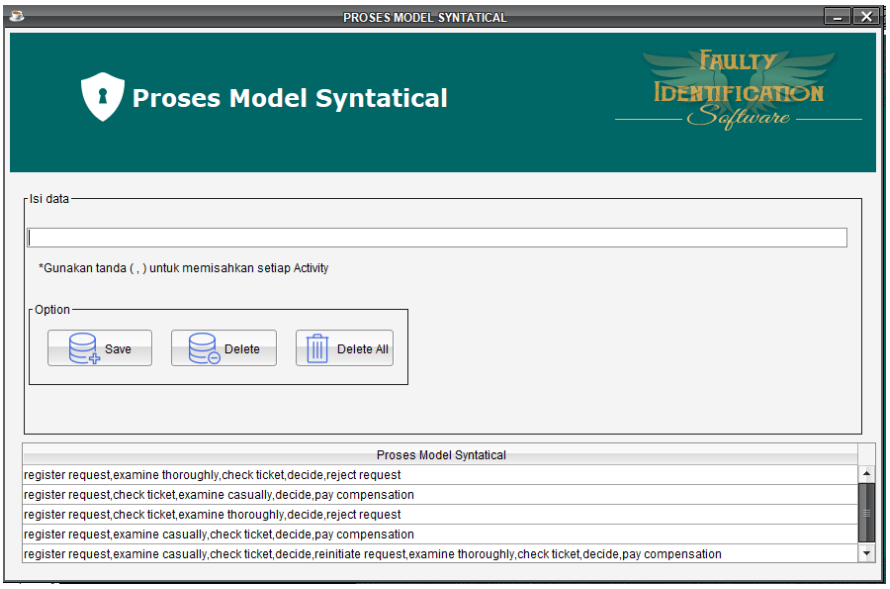
4.1.2 Halaman Antarmuka Data Event Log



Gambar 19. Implementasi Antarmuka Data Event Log

Halaman ini dibuat untuk menampilkan seluruh data yang telah di masukkan ke dalam aplikasi, terdapat 3 tombol yang disediakan, yaitu: *find* untuk mencari data yang diperlukan, tombol *export* untuk menyimpan data dalam format CSV dan tombol *refresh* untuk menampilkan kembali keseluruhan data.

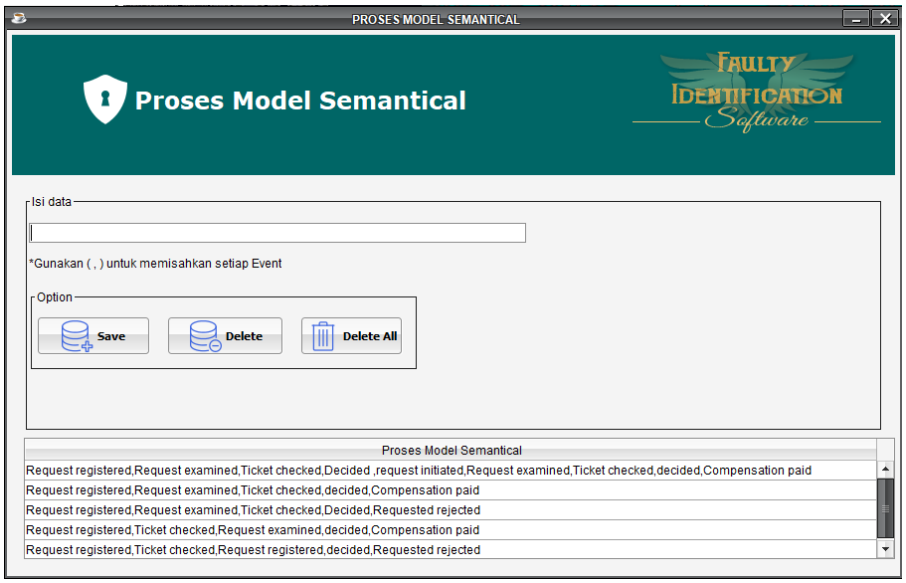
4.1.3 Halaman Antarmuka Proses Model Syntatical



Gambar 20. Implementasi Antarmuka Proses Model Syntatical

Halaman ini dibuat untuk memasukkan proses model *syntatical* sebelum melakukan identifikasi *fault*. Terdapat tiga tombol yang disediakan, yaitu: tombol *save* untuk menyimpan data proses model, tombol *delete* untuk menghapus data satu persatu dan tombol *delete all data* untuk menghapus seluruh data proses model *syntatical*.

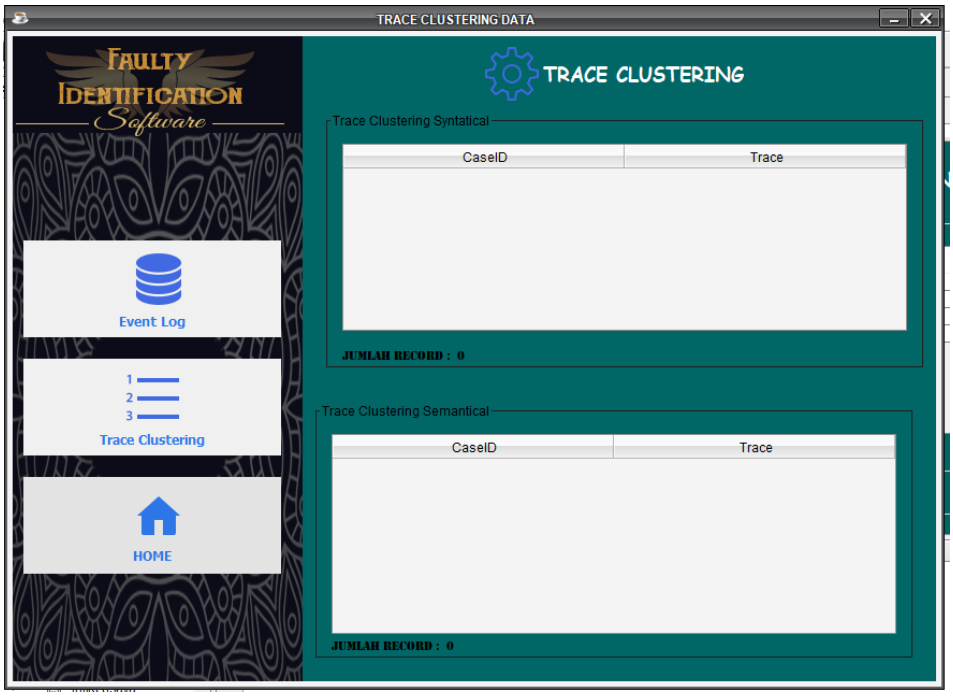
4.1.4 Halaman Antarmuka Proses Model Semantical



Gambar 21. Implementasi Antarmuka Proses Model Semantical

Halaman ini dibuat untuk memasukkan proses model *semantical* sebelum melakukan identifikasi *fault*. Terdapat tiga tombol yang disediakan, yaitu: tombol *save* untuk menyimpan data proses model, tombol *delete* untuk menghapus data satu persatu dan tombol *delete all data* untuk menghapus seluruh data proses model *semantical*.

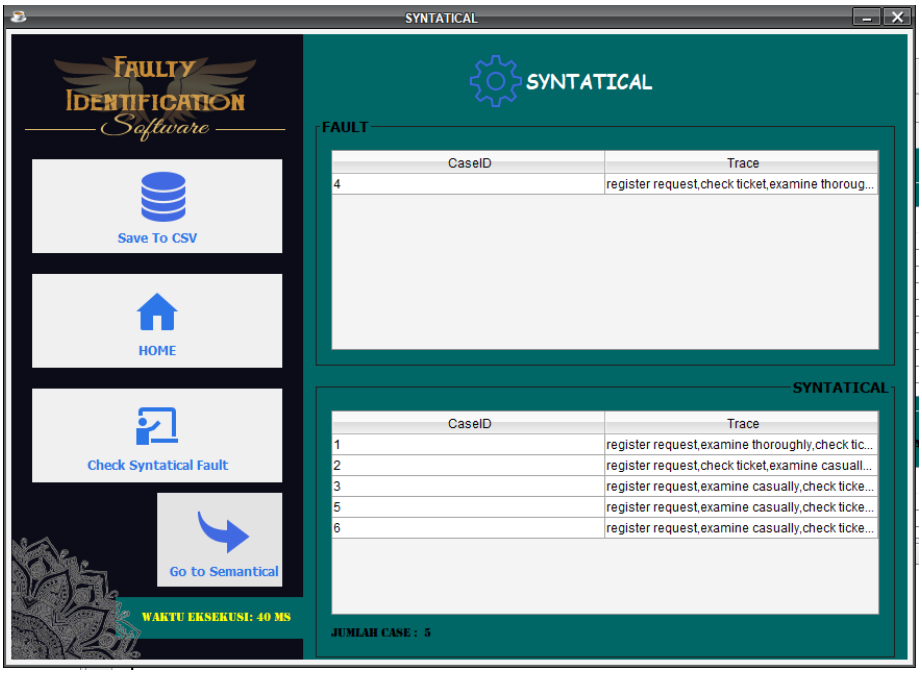
4.1.5 Halaman Antarmuka Trace Clustering



Gambar 22. Implementasi Antarmuka Trace Clustering

Halaman ini dibuat untuk melakukan proses *trace clustering* dan menampilkan hasilnya.

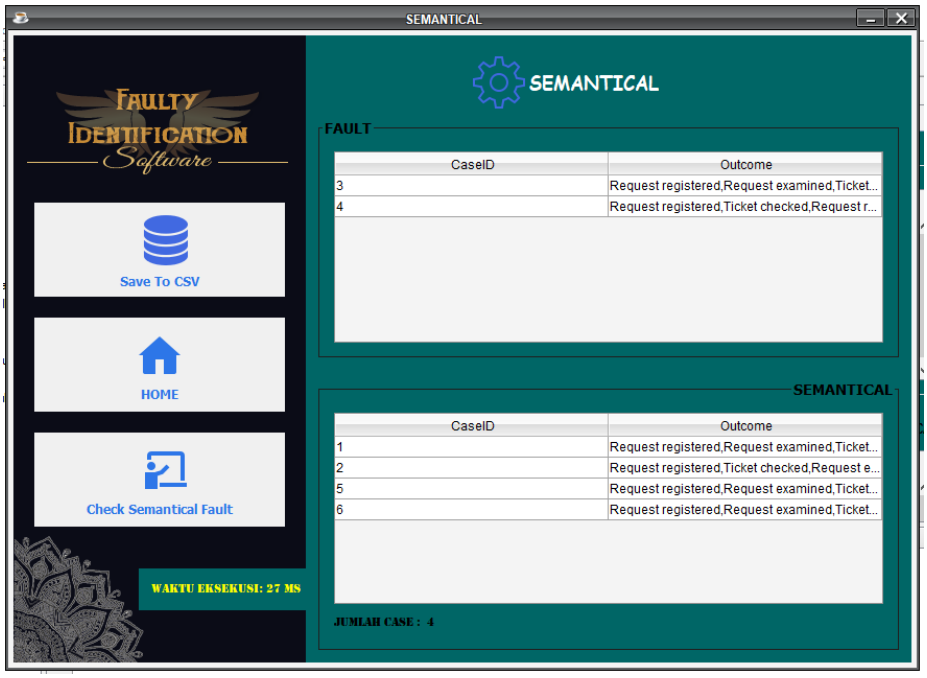
4.1.6 Halaman Antarmuka Identifikasi Syntatical Fault



Gambar 23. Implementasi Antarmuka Syntatical Fault

Halaman ini dibuat untuk melakukan identifikasi *syntatical fault* dan menampilkan data yang memiliki *fault* dan yang tidak memiliki *fault*.

4.1.7 Halaman Antarmuka Identifikasi Semantical Fault



Gambar 24. Implementasi Antarmuka Semantical Fault

Halaman ini dibuat untuk melakukan identifikasi *semantical fault* dan menampilkan data yang memiliki *fault* dan yang tidak memiliki *fault*.

4.2 Pengujian

4.2.1 Deskripsi Pengujian

Pengujian yang dilakukan dengan mengamati hasil eksekusi melalui data uji dan memerikasa fungsionalitas suatu system secara lengkap.

4.2.1.1 Data Uji

Data uji diambil dari buku karya Wil van der Aalst berjudul "*Process Mining: Data Science in Action*." terbitan Springer Berlin Heidelberg (2016) halaman 3-23. Contoh data uji dapat dilihat pada Gambar 25.

Case ID	Event ID	Timestamp	Activity	Resource	Costs	Outcome
1	35654423	30-12-2010:11.02	register request	Pete	50	Request registered
1	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	Request examined
1	35654425	05-01-2011:15.12	check ticket	Mike	100	Ticket checked
1	35654426	06-01-2011:11.18	decide	Sara	200	Decided
1	35654427	07-01-2011:14.24	reject request	Pete	200	Requested rejected
2	35654483	30-12-2010:11.32	register request	Mike	50	Request registered
2	35654485	30-12-2010:12.12	check ticket	Mike	100	Ticket checked
2	35654487	30-12-2010:14.16	examine casually	Sean	400	Request examined
2	35654488	05-01-2011:11.22	decide	Sara	200	decided
2	35654489	08-01-2011:12.05	pay compensation	Ellen	200	Compensation paid
3	35654521	30-12-2010:14.32	register request	Pete	50	Request registered
3	35654522	30-12-2010:15.06	examine casually	Mike	400	Request examined
3	35654524	30-12-2010:16.34	check ticket	Ellen	100	Ticket checked
3	35654525	06-01-2011:09.18	decide	Sara	200	Decided
3	35654526	06-01-2011:12.18	reinitiate request	Sara	200	request initiated
3	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	Request examined
3	35654530	08-01-2011:11.43	check ticket	Pete	100	Ticket checked
3	35654531	09-01-2011:09.55	decide	Sara	200	decided
3	35654533	15-01-2011:10.45	pay compensation	Ellen	200	Compensation paid
4	35654641	06-01-2011:15.02	register request	Pete	50	Request registered
4	35654643	07-01-2011:12.06	check ticket	Mike	100	Ticket checked
4	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	Request registered

Gambar 25. Contoh Data Uji

Pada data uji ini terdiri dari :

1. *Case id* merupakan sebuah angka yang digunakan untuk memberi atau mengidentifikasi sebuah *event*
2. *Event* id digunakan agar dapat mengidentifikasi sebuah *event*,
3. *Timestamp* (dd-MM-yyyy:HH.mm) menunjukkan waktu sebuah aktivitas,
4. *Activity* merupakan rangakaian kegiatan,
5. *Resource* merupakan nama-nama orang yang bertanggung jawab dalam pelaksanaan dalam suatu kegiatan,
6. *Costs* harga dalam sebuah *event*
7. *Outcome* merupakan hasil dari setiap *event*

Data uji ini merupakan data yang benar atau tidak mengandung fault.

4.2.2 Hasil Pengujian

Pengujian dilakukan 6 kali dengan masing-masing 100 *record*, 200 *record*, 300 record, 400 *record*, 500 *record*, 600 *record*, 700 *record*, 800 *record*, 900 *record*

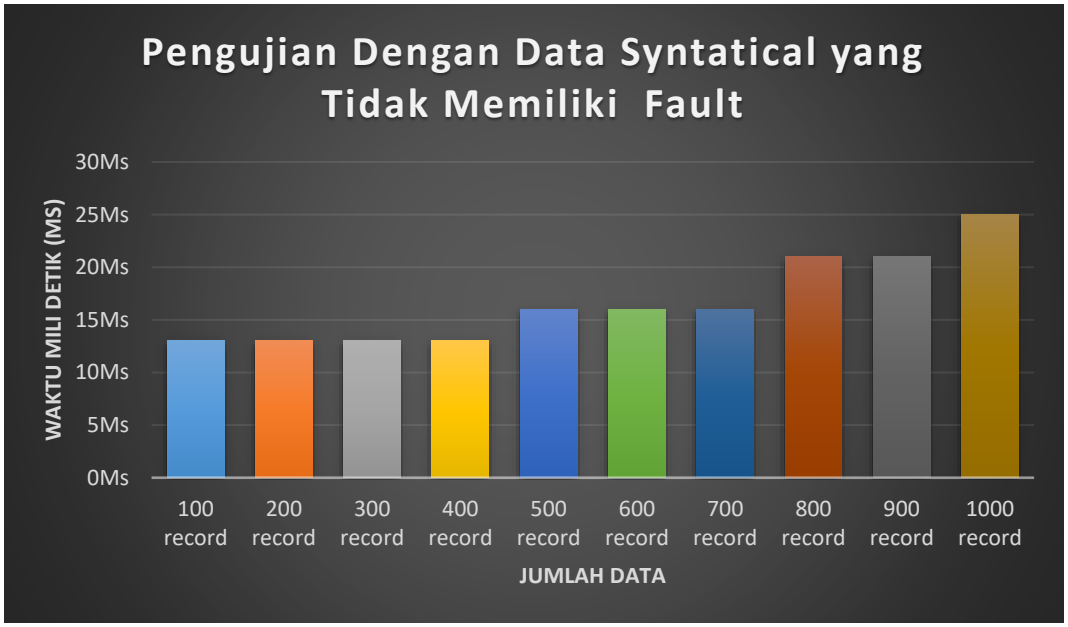
dan 1000 *record*. Dalam masing-masing data uji tersebut, dimasukkan data yang *fault* untuk menguji akurasi aplikasi dalam menemukan *fault* atau kesalahan dalam data *event log*.

4.2.2.1 Pegujian Aplikasi Dengan Data Tidak Memiliki *Fault*

Aplikasi diuji dengan memasukkan data yang tidak memiliki *fault* untuk mengetahui kecepatan dan akurasi aplikasi dalam mendeteksi data yang tidak memiliki *fault*.

Tabel 5. Hasil Pengujian Data *Syntatical* Tidak Memiliki *Fault*

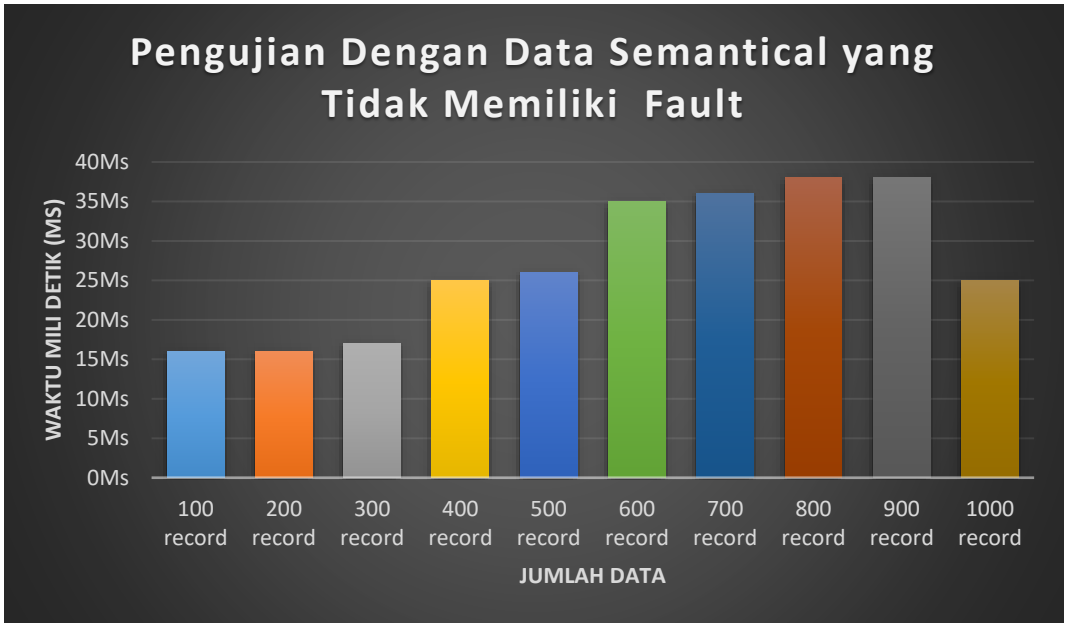
No.	Jumlah Record	Total Trace	Total Trace yang Benar	Total Trace yang Fault	Waktu yang Diperlukan
1.	100	16	16	0	13 ms
2.	200	32	32	0	13 ms
3.	300	48	48	0	13 ms
4.	400	64	64	0	13 ms
5.	500	80	80	0	16 ms
6.	600	92	92	0	16 ms
7.	700	112	112	0	16 ms
8.	800	128	128	0	21 ms
9.	900	144	144	0	21 ms
10.	1000	160	160	0	25 ms
Rata-Rata					16.7 ms



Gambar 26. Grafik Pengujian Dengan Data *Syntatical* Yang Tidak Memiliki *Fault*

Tabel 6. Hasil Pengujian Data *Semantical* Tidak Memiliki *Fault*

No.	Jumlah Record	Total Trace	Total Trace Yang Benar	Total Trace Yang Fault	Waktu Yang Diperlukan
1.	100	16	16	0	16
2.	200	32	32	0	16
3.	300	48	48	0	17
4.	400	64	64	0	25
5.	500	80	80	0	26
6.	600	92	92	0	35
7.	700	112	112	0	36
8.	800	128	128	0	38
9.	900	144	144	0	38
10.	1000	160	160	0	40
Rata-Rata					28.7 ms



Gambar 27. Grafik Pengujian Dengan Data *Semantical* Yang Tidak Memiliki *Fault*

Dari pengujian ini dapat disimpulkan bawah pada pengujian data *syntactical* dan *semantical* yang tidak memiliki *fault*, aplikasi tetap berjalan dengan baik dan dapat mengidentifikasi 100% fault dengan rata-rata waktu yang diperlukan adalah 22.7 ms.

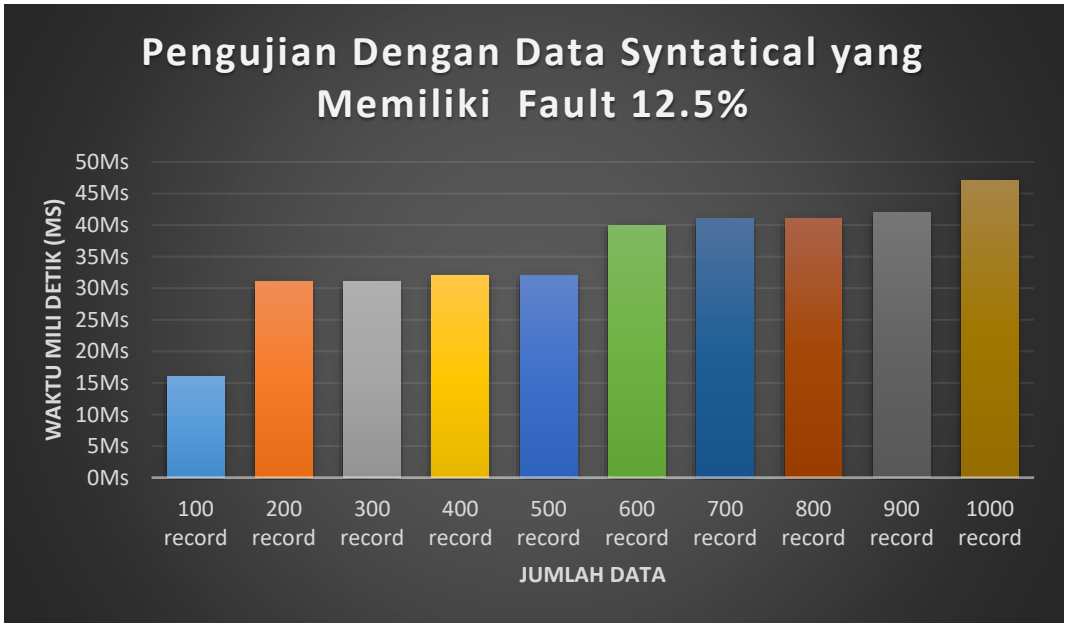
4.2.2.2 Pengujian Aplikasi Dengan Data Yang memiliki Fault 12.5%

Aplikasi diuji dengan memasukkan data yang memiliki *fault* sebanyak 12.5% pada setiap datanya. Pada data dengan 100 *record* di masukkan data yang *fault* sebanyak 12.5% hingga sampai data dengan 1000 *record*. Sehingga menampilakan kecepatan dan akurasi aplikasi dalam mendeteksi data yang memiliki *fault*.

Tabel 7. Hasil Pengujian Dengan Data *Syntatical* Yang Memiliki *Fault* 12.5%

No.	Jumlah Record	Total Case	Total Case Yang Benar	Total Case Yang Fault	Fault (%)	Fault Yang Ditemukan (%)	Waktu Yang Diperlukan
1.	100	16	14	2	12.5%	100%	16 ms
2.	200	32	28	4	12.5%	100%	31 ms
3.	300	48	42	6	12.5%	100%	31 ms
4.	400	64	56	8	12.5%	100%	31 ms

5.	500	80	70	10	12.5%	100%	32 ms
6.	600	92	80	12	12.5%	100%	40 ms
7.	700	112	98	14	12.5%	100%	41 ms
8.	800	128	112	16	12.5%	100%	41 ms
9.	900	144	126	18	12.5%	100%	42 ms
10.	1000	160	141	19	12.5%	100%	47 ms
Rata-Rata						100%	35.2 ms

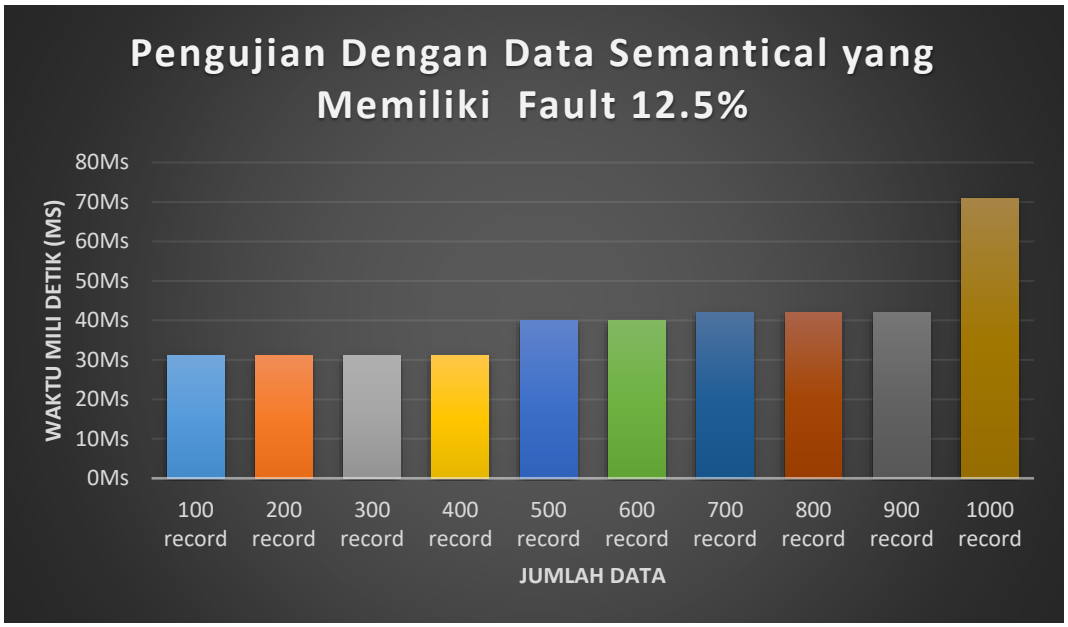


Gambar 28. Grafik Pengujian Dengan Data *Syntatical* Yang Memiliki Fault 12.5%

Tabel 8. Hasil Pengujian Data *Semantical* Yang Memiliki *Fault* 12.5%

No.	Jumlah Record	Total Case	Total Case Yang Benar	Total Case Yang Fault	Fault (%)	Fault Yang Ditemukan (%)	Waktu Yang Diperlukan (ms)
1.	100	16	14	2	12.5%	100%	31 ms
2.	200	32	28	4	12.5%	100%	31 ms
3.	300	48	42	6	12.5%	100%	31 ms
4.	400	64	56	8	12.5%	100%	31 ms
5.	500	80	70	10	12.5%	100%	40 ms
6.	600	92	80	12	12.5%	100%	40 ms
7.	700	112	98	14	12.5%	100%	44 ms
8.	800	128	112	16	12.5%	100%	42 ms

9.	900	144	126	18	12.5%	100%	42 ms
10.	1000	160	141	19	12.5%	100%	71 ms
Rata-Rata						100%	40 ms



Gambar 29. Grafik Pengujian Dengan Data *Semantical* Yang Memiliki *Fault* 12.5%

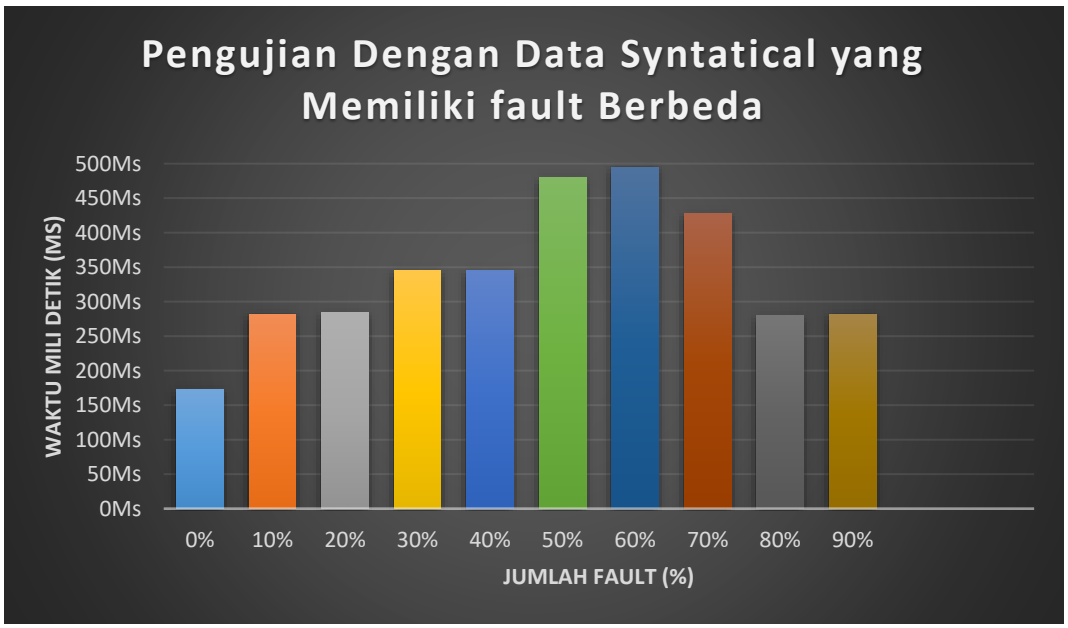
Dari pengujian ini dapat disimpulkan bawah pada pengujian dengan data *syntactical* dan *semantical* yang memiliki *fault* sebanyak 12.5% dari setiap data 100 *record*, 200 *record* sampai dengan 1000 *record*, aplikasi tetap berjalan dengan baik dan dapat mengidentifikasi 100% fault dengan rata-rata waktu yang diperlukan adalah 37.6 ms.

4.2.2.3 Pengujian Aplikasi Dengan Data Yang Sama

Aplikasi diuji dengan memasukkan data yang sama banyak namun memiliki *fault* yang berbeda-beda pada setiap datanya. Misalnya pada data dengan 100 *record* memiliki 0% *fault*,200 *record* memiliki 10% *fault*, 300 *record* memiliki 20% *fault*, 400 *record* memiliki 30% *fault*, 500 *record* memiliki 40% *fault*, 600 *record* memiliki 50% *fault*, 700 *record* memiliki 60% *fault*, 800 *record* memiliki 70% *fault*, 900 *record* memiliki 80% *fault* dan 1000 *record* memiliki 90% *fault*.

Tabel 9. Hasil Pengujian Data *Syntatical* Yang Memiliki *Fault* Berbeda

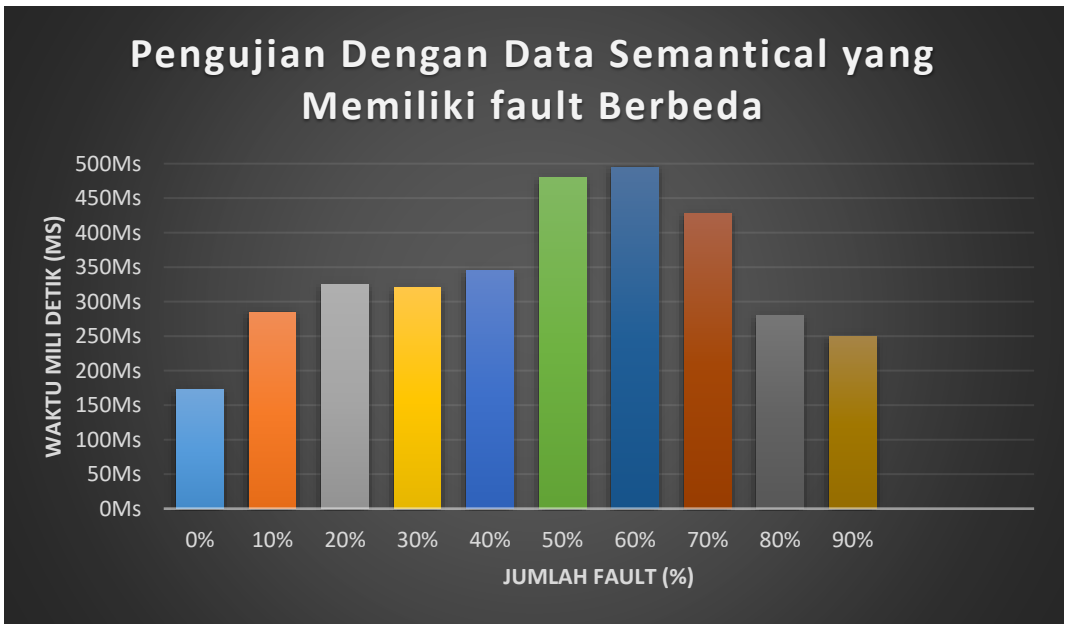
No.	Jumlah Record	Total Case	Total Case Yang Benar	Total Case Yang Fault	Fault (%)	Fault Yang Ditemukan (%)	Waktu Yang Diperlukan (ms)
1.	500	80	80	0	0%	100%	172 ms
2.	500	80	72	8	10%	100%	282 ms
3.	500	80	64	16	20%	100%	284 ms
4.	500	80	56	24	30%	100%	345 ms
5.	500	80	48	32	40%	100%	345 ms
6.	500	80	40	40	50%	100%	480 ms
7.	500	80	32	48	60%	100%	495 ms
8.	500	80	24	56	70%	100%	428 ms
9	500	80	16	64	80%	100%	280 ms
10.	500	80	8	72	90%	100%	282 ms
Rata-Rata						100%	339.3 ms



Gambar 30. Grafik Pengujian Dengan Data *Syntatical* Yang Memiliki *Fault* Berbeda

Tabel 10. Hasil Pengujian Data Semantical Yang Memiliki Fault Berbeda

No.	Jumlah Record	Total Case	Total Case Yang Benar	Total Case Yang Fault	Fault (%)	Fault Yang Ditemukan (%)	Waktu Yang Diperlukan (ms)
1.	500	80	80	0	0%	100%	172 ms
2.	500	80	72	8	10%	100%	285 ms
3.	500	80	64	16	20%	100%	325 ms
4.	500	80	56	24	30%	100%	320 ms
5.	500	80	48	32	40%	100%	345 ms
6.	500	80	40	40	50%	100%	480 ms
7.	500	80	32	48	60%	100%	495 ms
8.	500	80	24	56	70%	100%	428 ms
9	500	80	16	64	80%	100%	280 ms
10.	500	80	8	72	90%	100%	250 ms
Rata-Rata						100%	338 ms



Gambar 31. Grafik Pengujian Dengan Data Semantical Yang Memiliki Fault Berbeda

Dari pengujian ini dapat disimpulkan bawah pada pengujian dengan data *syntactical* dan *semantical* yang memiliki *fault* sebanyak berbeda namun memiliki jumlah data yang sama dari data 500 *record*, aplikasi tetap berjalan dengan baik dan dapat mengidentifikasi 100% fault dengan rata-rata waktu yang diperlukan adalah 338.65 ms.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan tahapan analisis, perancangan dan implementasi serta pengujian pada aplikasi yang dibangun menggunakan bahasa pemograman Java dan DBMS MariaDB serta berbasis desktop dan menggunakan Algoritma Trace Clustering, maka didapatkan kesimpulan bahwa :

1. Aplikasi telah diimplementasikan sesuai desain atau rancangan.
2. Aplikasi telah dapat mencapai tujuan yang direncanakan.
   Sesuai dengan hasil pengujian, maka aplikasi dapat mengidentifikasi *fault* baik *syntactical fault* maupun *semantical fault* dengan baik. Untuk data tanpa *fault*, rata-rata waktu diperlukan adalah 22,7ms. Untuk *syntactical* dan *Semantical* dengan jumlah *fault* 12.5% waktu yang diprelukan adalah 37.6ms dan *fault* yang dapat ditemukan adalah 100%.
3. Dari pengujian yang dilakukan terhadap aplikasi, disimpulkan bahwa:
   a. aplikasi dengan data uji yang tidak memiliki *fault* memakan waktu lebih sedikit dari pada data uji yang memiliki *fault*;
   b. semakin banyak data uji dalam sebuah *event log* akan memerlukan waktu lebih banyak;
   c. semakin banyak *fault* di dalam sebuah *event log* maka waktu yang diperlukan akan makin banyak, namun setelah *fault* mencapai 50% maka jumlah waktunya akan menurun kembali.

5.2 Saran

Demi kinerja dan pengembangan sistem informasi menuju produk aplikasi yang lebih baik, penulis ingin menyampaikan beberapa saran kepada pengembang selajutnya, antara lain:

1. Perlunya penambahan fitur menampilkan data *fault* lebih detail
2. Dapat mengimport data selain format csv.

DAFTAR PUSTAKA

Ferreira, D., Zacarias, M., Malheiros, M. and Ferreira, P., 2007. Approaching process mining with sequence clustering: Experiments and findings. Business Process Management, pp.360-374.

Song, M., Günther, C.W. and Van der Aalst, W.M., 2008, September. Trace clustering in process mining. In International Conference on Business Process Management (pp. 109-120). Springer, Berlin, Heidelberg.

Suriadi, S., Wynn, M.T., Ouyang, C., ter Hofstede, A.H. and van Dijk, N.J., 2013, June. Understanding process behaviours in a large insurance company in Australia: A case study. In International Conference on Advanced Information Systems Engineering (pp. 449-464). Springer, Berlin, Heidelberg.

Van der Aalst, W.M., 2016. Process mining: data science in action. Springer.

BIODATA PENULIS

Yoshua Fernandes Sirait lahir di Sidikalang pada tanggal 28 Juli 1996. Anak pertama dari empat bersaudara dari pasangan J. Sirait dan B. Hutagalung. Penulis tinggal di Kav. Sagulung Baru Kel. Sungai Binti Kec. Sagulung. Penulis menyelesaikan pendidikan di Sekolah Dasar Negeri 007 Batam pada tahun 2009. Pada tahun itu juga penulis melanjutkan Pendidikan di Sekolah Menegah Pertama Negeri 36 Batam dan lulus pada tahun 2012, kemudian melanjutkan Sekolah Menegah Atas Negeri 17 Batam dan tamat pada tahun 2015. Pada tahun 2015 penulis melanjutkan pendidikan di perguran tinggi negeri, tepatnya di Politeknik Negeri Batam Jurusan Teknik Informatika Prodi Informatika.

Penulis juga aktif di organisasi yang ada di kampus Politeknik Negeri Batam. Penulis terlibat secara aktif di dalam Badan Eksekutif Mahasiswa (BEM), serta juga aktif di Unit Kegiatan Mahasiswa PD.El-Shaddai Polibatam yang merupakan salah satu organisasi keagamaan yang ada di Politeknik Negeri Batam.