BAB III

METODE DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Metode penelitian yang digunakan dalam membangun dan mengimplementasikan algoritma *fuzzy hashing* untuk meningkatkan jumlah deteksi *malware* dengan metode *signature-based detection* adalah sebagai berikut.

1. Studi Literatur

   Pada tahap ini, hal-hal yang berkaitan dan digunakan dalam pembangunan aplikasi untuk mengimplementasi *fuzzy hash* dicari dan dipelajari dari berbagai artikel, buku teks, jurnal, presentasi, ataupun karya ilmiah terkait lainnya tentang *malware* dan algoritma yang dibahas.

2. Pengumpulan Data dan Sampel *Malware*

   Pada tahap ini, akan dilakukan pengumpulan data dan sampel *malware* dari berbagai web *library malware*, *internet forum computer security*, dan *antivirus online*. Orang lain dapat berpartisipasi mengirim *file* yang dicurigai sebagai *malware* melalui DropItToMe.

3. Perancangan Basis Data *Signature*

   Pada perancangan basis data, dirancang suatu tabel yang digunakan untuk menyimpan informasi mengenai *malware* dan digunakan sebagai acuan *signature* ketika dilakukan uji deteksi terhadap *file* lain.

4. Perancangan *Library Fuzzy Hash*

   Library fuzzy hash dibangun menggunakan bahasa pemprograman C# .NET 4.0 berdasarkan *fuzzy hashing API* (Hyldahl, 2010; Kornblum dan Grohne, 2013). *Library* ini digunakan untuk menghitung *fuzzy hash* dari suatu

file dan membandingkan satu *fuzzy hash* dengan satu *fuzzy hash* lain menggunakan *levenshtein distance.*

5. Perancangan Aplikasi *Dashboard*, *Database Signature Builder*, dan *Malware Detector*

   Aplikasi *Dashboard* dibangun sebagai aplikasi *launcher* dari aplikasi *Database Signature Builder* dan *Malware Detector* dan memiliki fungsi tambahan untuk mengatur *launcher* dari *context-menu file explorer*, mengatur *database signature default* yang telah dibuat menggunakan aplikasi *Database Signature Builder* serta melakukan pendeteksian *removable drive* yang baru terpasang pada komputer. Aplikasi *Database Signature Builder* dibangun untuk melaksanakan penghitungan *fuzzy hash* dan SHA256 (sebagai pembanding) dari sampel *malware* serta menyimpannya ke dalam *database* SQLite3, sedangkan aplikasi *Malware Detector* dibangun untuk melaksanakan pengujian apakah dengan implementasi *fuzzy hash* dapat meningkatkan jumlah deteksi *malware* dibandingkan dengan SHA256 berdasarkan *database signature* yang telah dibuat menggunakan *Database Signature Builder*. Aplikasi dibangun menggunakan bahasa pemprograman C# .NET 4.0 dengan model *windows-form application* dan *library fuzzy hash* yang telah dibuat pada langkah sebelumnya, digunakan dalam ketiga aplikasi tersebut.

6. Implementasi *Library* dan Aplikasi

   Pada tahap ini, kode program yang telah dirancang sebelumnya diimplementasikan dan dikompilasi menghasilkan tiga aplikasi berekstensi EXE dan satu *library fuzzy hash* berekstensi DLL (*Dynamic Link Library*).

7. Uji Coba dan Evaluasi

   Di dalam tahap ini, dilakukan uji coba terhadap aplikasi yang dibuat, dengan beberapa skenario pengujian, apakah hasilnya sesuai dengan tujuan penelitian.

8. Penulisan Skripsi

   Pada tahap ini, akan dilakukan penyusunan laporan yang memuat dokumentasi mengenai perancangan, pembuatan, serta hasil dari aplikasi yang telah dibuat dalam suatu karya ilmiah.

3.1.1 Variabel Penelitian

Dalam penelitian ini, hasil yang dicari dari implementasi *fuzzy hashing* adalah seberapa besar peningkatan jumlah atau tingkat deteksi *malware* yang menggunakan *fuzzy hash* jika dibandingkan SHA256 untuk sampel yang sama dan berapa tingkat akurasi deteksinya. Jadi, variabel yang dapat memengaruhi hasil penelitian adalah sampel *malware* dan *file* lain yang diuji, serta penentuan batas bawah toleransi deteksi kemiripan. Tingkat jumlah deteksi dan akurasi dipengaruhi oleh penentuan batas bawah toleransi deteksi kemiripan *hash file* dengan *hash* yang ada di *database signature*. Batas bawah toleransi ini disebut *threshold* dengan nilai dalam rentang 20% − 100%. Misalnya, nilai 20% menunjukkan untuk mendeteksi suatu *file* sebagai *malware*, maka cukup ditemukan minimal 20% kemiripan *hash file* dengan *hash* yang ada di *signature*.

3.2 Spesifikasi Perangkat

Dalam pengembangan aplikasi, dibutuhkan perangkat berupa perangkat keras (*hardware*) dan perangkat lunak (*software*) agar aplikasi ini dapat berjalan dengan optimal. Tabel 3.1 menunjukkan spesifikasi perangkat yang digunakan.

3.2.1 Perangkat Keras

Tabel 3.1. Spesifikasi PC yang digunakan dalam pengembangan aplikasi

Jenis Komponen	**Komponen yang digunakan**
CPU	AMD Athlon X2 64 4200+ 2.2Ghz
System memory	4 GB DDR2
Harddisk	160 GB SATA
GPU	nVidia GeForce 9500GT 1GB GDDR2

3.2.2 Perangkat Lunak

Perangkat lunak yang digunakan sebagai *operating system* pada PC adalah Windows 7 Ultimate 32 bit *build* 7600. Aplikasi yang digunakan dalam penelitian ini adalah Microsoft Visual Studio 2012 Premium (menggunakan *framework* .NET 4.0) sebagai IDE utama untuk membuat *library fuzzy hash*, program *Database Signature Builder* dan *Malware Detector*; HashMyFiles untuk menghitung *hash* suatu *file*, SublimeText untuk melihat isi heksadesimal suatu *file* atau dapat juga digunakan sebagai IDE sederhana, SQLiteMan untuk mengatur basis data SQLite3 dan *project* ssdeep untuk menghitung *hash* sebagai salah satu contoh implementasi *fuzzy hash*. Konfigurasi keamanan pada *operating system* dibuat serendah mungkin dan tidak ada *antivirus* yang diaktifkan supaya dalam pengujian, sampel tidak dihapus atau tidak dihalangi penyimpanannya baik oleh *operating system* maupun oleh *antivirus*.

3.3 Pengumpulan Data dan Pengambilan Sampel Malware

3.3.1 Pengumpulan Data

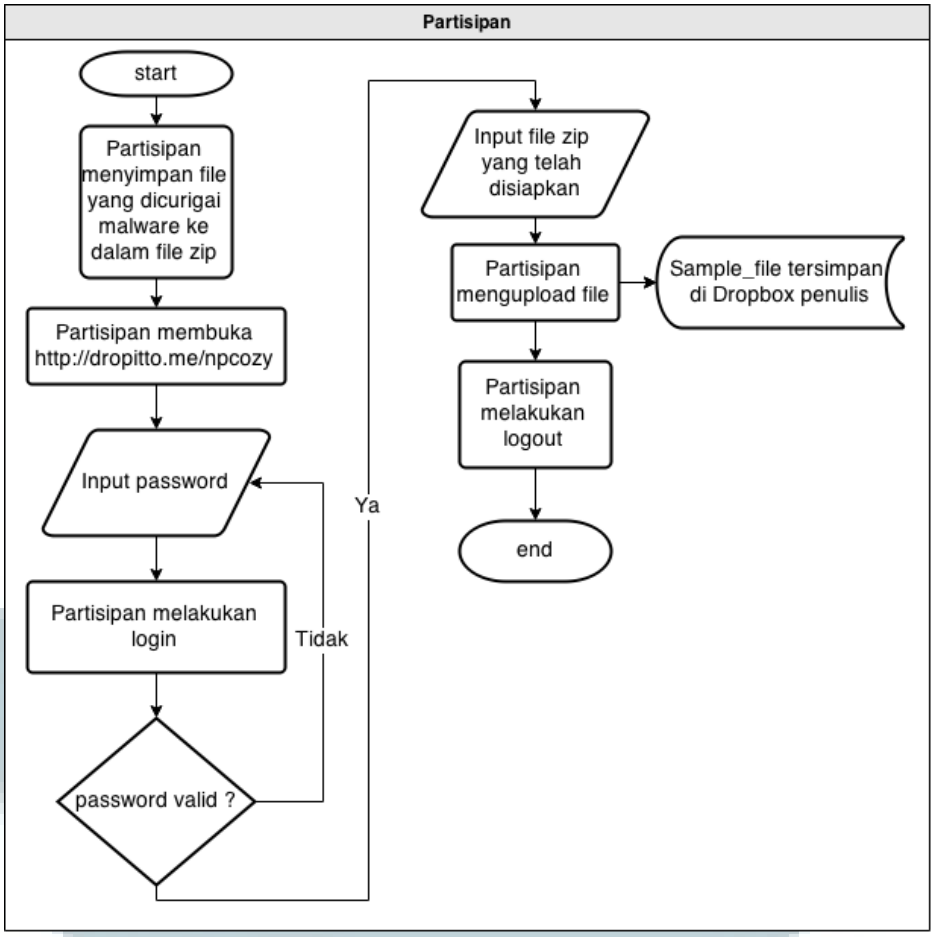
Data mengenai *fuzzy hashing* dan *malware* merujuk pada artikel, buku teks, jurnal, presentasi, publikasi atau laporan, serta informasi dari forum-forum

computer security yang disediakan beberapa produsen *antivirus*. *File malware* didapat dari berbagai web *library malware* dan forum *computer security* seperti VxHeaven, OpenMalware, VirusIndonesia, VirusTotal, dan VirusShare.

3.3.2 Pengambilan Sampel

Jumlah sampel *malware* yang digunakan dalam penelitian adalah 206 *malware* (104 *malware* dan 102 varian *malware* yang masih mirip dengan 104 *malware*). Berdasarkan *Roscoe's Simple Rules Of Thumb*, dalam kebanyakan *ex post facto* dan penelitian eksperimental, disarankan jumlah minimal sampel adalah 30 sampel dan jika sampel dipecah menjadi subsampel, jumlah minimal untuk setiap subsampel adalah 30 sampel (Hill, 1998).

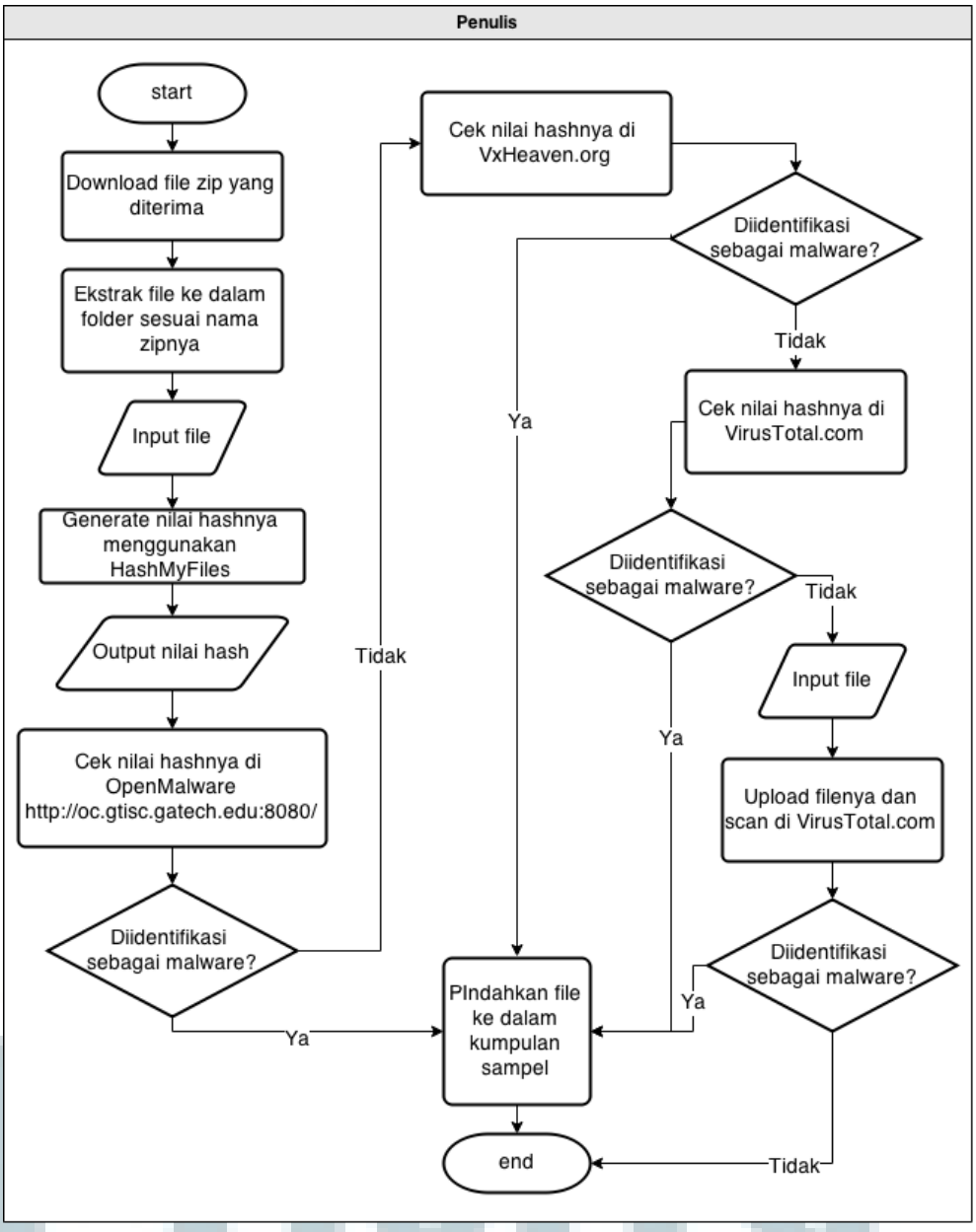
Sampel *malware* diambil secara acak dari *malware library* di internet, forum-forum *computer security* atau dari sumber lain. Untuk *malware executable*, pengambilan sampel berupa *file executable*. Untuk *malware scripting*, pengambilan sampel dapat berupa *file script* yang sudah jadi atau dapat berupa *source code* dan disimpan ulang ke dalam *file* di komputer. Beberapa virus merupakan hasil dari modifikasi sendiri terhadap virus yang telah ada sebagai simulasi varian dari virus. Sebagai tambahan, orang lain juga dapat berpartisipasi mengirim *file* yang dicurigai sebagai *malware* ataupun memang *file malware* melalui alamat http://dropitto.me/npcozy dan *password login* "polar" tanpa tanda kutip. Secara garis besar, alur partisipan melakukan *upload file* dalam langkah-langkah seperti yang ditunjukkan gambar 3.1.



Gambar 3.1. Alur proses pengiriman *file* yang dicurigai *malware* oleh partisipan melalui DropItToMe

Dikarenakan sistem DropItToMe dalam meng*upload file* menggunakan nama *file* apa adanya sesuai nama yang diberikan partisipan, maka jika partisipan kebetulan meng*upload file* dengan nama yang sama dengan nama *file* yang sudah ter*upload* sebelumnya, maka sistem penyimpanan Dropbox secara otomatis akan melakukan *replace* terhadap *file* yang lama. Oleh karena itu, ketika *link* DtopItToMe tersebut disebarkan kepada publik, diberikan petunjuk bahwa nama *file zip* yang akan di*upload* harus nama yang unik seperti alamat *email*, nomor telepon, NIM untuk mahasiswa, nama *id* akun *game* atau *social media*. *File* yang telah di*upload*, tidak langsung menjadi sampel *malware*. Selanjutnya dilakukan

download file tersebut satu per satu, mengekstraknya, dan dilakukan pemeriksaan seperti yang ditunjukkan gambar 3.2. Alur proses ini dilakukan apabila ada *file* yang telah terkirim ke Dropbox penulis.



Gambar 3.2. Alur proses pemeriksaan *file* yang telah dikirim partisipan

3.4 Perancangan Basis Data

Pada perancangan basis data, dirancang suatu tabel yang digunakan untuk menyimpan informasi mengenai sampel *malware* dan digunakan sebagai acuan *signature* ketika dilakukan uji deteksi terhadap *file* lain. Basis data yang dibangun adalah basis data sederhana yang hanya terdiri dari satu buah tabel yang bernama *signature*. Basis data ini diimplementasikan menggunakan *database* SQLite3 yang berupa *file based database*. Tabel 3.2 berikut menunjukkan rancangan tabel.

Tabel 3.2. Rancangan tabel *signature* yang berisi informasi *malware* sederhana

Nama Kolom	Tipe Data	Keterangan
hash	TEXT	berisi *hash* suatu *file* (*fuzzy hash* atau SHA256)
identified_name	TEXT	secara *default*, berisi nama *file* dari *file* yang *hash*-nya di-*generate*, dapat diubah manual sebelum kumpulan *hash* ditulis secara permanen ke *database*
identified_date	TEXT	tanggal yang otomatis diisi ketika *record* di*insert* ke dalam tabel
size	REAL	berisi ukuran *file* dalam *bytes* dari *file* yang *hash*-nya di-*generate*

File yang dihasilkan dari basis data *signature* dapat berekstensi .fuzzy dan .sha256. Penggunaan ekstensi tersebut hanya sebagai penanda jenis *hash* yang disimpannya dalam kolom *hash* pada tabel *signature*.

3.5 Perancangan Library Fuzzy Hash

Tujuan dari pembuatan *library* yang terpisah dari aplikasi utama adalah untuk meningkatkan modularitas dan portabilitas dari perangkat lunak yang dibangun dengan pendekatan *object oriented programming* (OOP). *Library fuzzy*

hash yang dibangun menggunakan bahasa pemprograman C# .NET 4.0 akan diperlukan dan digunakan oleh aplikasi lain (*Dashboard*, *Database Signature Builder*, dan *Malware Detector*) berfungsi untuk melakukan hal-hal berikut:

1. menghitung *fuzzy hash* dari suatu *file*, dan
2. membandingkan satu *fuzzy hash* dengan satu *fuzzy hash* lain menggunakan *levenshtein distance*,

Berikut penjelasan dari perancangan *library fuzzy hash* yang dipecah satu per satu berdasarkan fungsi *library* tersebut.

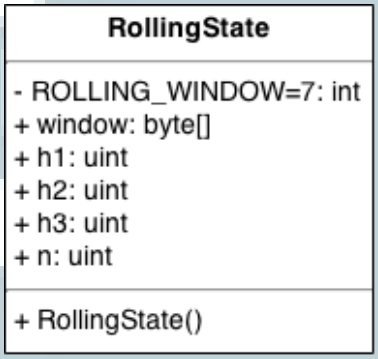
3.5.1 Perancangan Fungsi Penghitungan Fuzzy Hash suatu File

Berdasarkan *pseudocode* pada sub-bab 2.5 (lihat gambar 2.5 dan 2.6), terdapat berbagai *function* utama yang digunakan dalam proses menghitung *fuzzy hash*. Berikut tabel 3.3 menunjukkan daftar *function*-nya.

Tabel 3.3. Daftar *function* berdasarkan *pseudocode* pada sub-bab 2.5

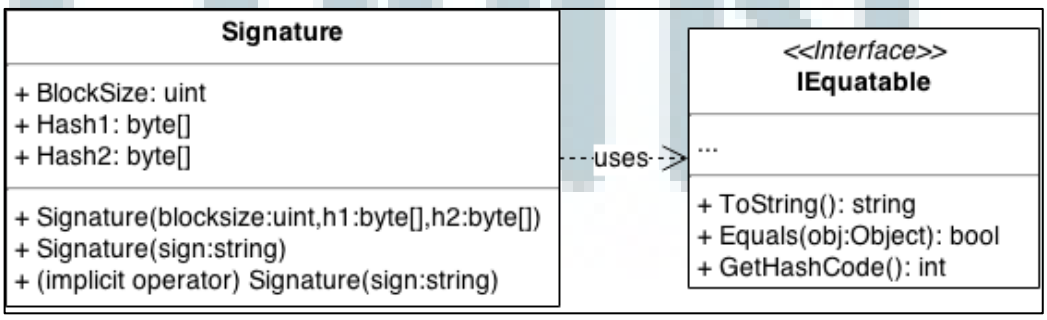
Nama function	**Input**	**Proses**	**Output**
`compute_initial_block_size()`	*Stream*	Menghitung *blocksize* awal berdasarkan panjang *file*	Besar *blocksize* awal
`initialize_rolling_hash()`	Struktur data *rolling* untuk menyimpan *state* dari *rolling hash*	Me*reset rolling state* menjadi nol	*Rolling state* dengan nilai baru
`initialize_traditional_hash()`	Nilai *hash* FNV	Me*reset hash* menjadi nilai FNV_INIT	Nilai *hash* FNV
`update_rolling_hash()`	Struktur data *rolling* untuk menyimpan *state* dan *byte* dari *file*	Meng*update rolling state*	Jumlah *rolling state* h1 + h2 + h3 untuk disimpan ke *rolling hash*
`update_traditional_hash()`	Nilai *hash* FNV dan *byte* dari *file*	Meng*update* nilai *hash* FNV	Nilai *hash* FNV

Sebelum *function-function* tersebut dibangun dalam suatu *class* utama yang dinamakan *FuzzyHashing*, maka *class-class* internal yang terlibat dalam proses penghitungan *hash* harus dirancang terlebih dahulu. *Class* internal tersebut adalah *RollingState* dan *Signature*. *RollingState* digunakan untuk menyimpan *state* penghitungan *rolling hash*. Gambar 3.3 berikut menunjukkan rancangan *class RollingState* yang berisi *field-field* yang diperlukan dalam *rolling hash* berdasarkan algoritma *spamsum* pada *pseudocode* sub-bab 2.5.



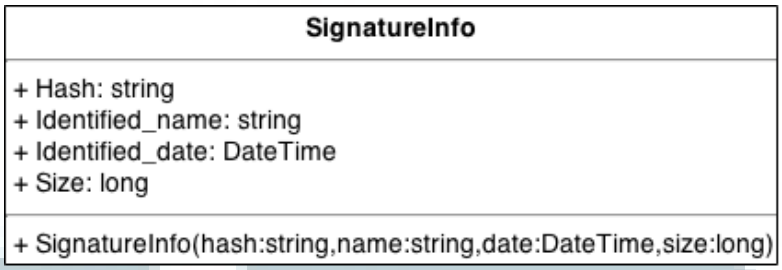
Gambar 3.3. Rancangan *class RollingState*

Class Signature digunakan untuk menyimpan hasil *fuzzy hash* yang terdiri dari *blocksize*, *hash_part1*, dan *hash_part2*. Dibuat beberapa *constructor* untuk mempermudah pembuatan *object Signature* dari *string* dan digunakan *interface* yang tersedia dalam bahasa C# untuk mempermudah konversi *hash* dari *Signature* ke dalam *string*. Gambar 3.4 berikut menunjukkan rancangan *class Signature*.



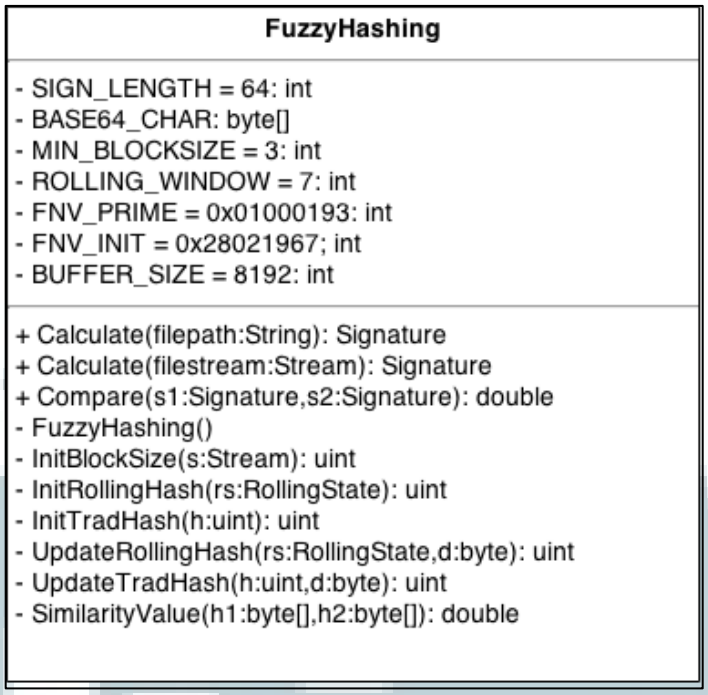
Gambar 3.4. Rancangan *class Signature*

Class SignatureInfo digunakan untuk mengenkapsulasi informasi *malware* yang di-*load* dari tabel *signature* (lihat tabel 3.2) untuk diteruskan ke dalam aplikasi. Gambar 3.5 berikut merupakan rancangan *class SignatureInfo*.



Gambar 3.5. Rancangan *class SignatureInfo*

Selanjutnya, *class fuzzy hash* dapat dirancang. *Class* ini berupa *static class* yang terdiri dari *method* internal berdasarkan tabel 3.3 dan juga terdiri dari *method public* untuk menerima paramater berupa alamat *file, filestream*, dan juga dua *signature* untuk dibandingkan kemiripannya yang dikirim oleh *user* melalui aplikasi lain yang menggunakan *library fuzzy hashing*. Nilai-nilai konstanta yang ditandai dengan nama *field* dalam huruf kapital berasal dari teori yang telah dijabarkan pada sub-bab 2.5 dan juga berdasarkan implementasi algoritma *spamsum*. SIGN_LENGTH merupakan panjang maksimal masing-masing *hash* dalam *fuzzy hash*; BASE64_CHAR berisi kumpulan karakter dalam *Base64 Data Encoding*; MIN_BLOCKSIZE adalah nilai *blocksize* minimal; ROLLING_WINDOW adalah ukuran *window* yang digunakan dalam *rolling hash*; FNV_INIT merupakan nilai yang digunakan dalam inisialisasi dan *reset* Fowler–Noll–Vo *hash* dan FNV_PRIME digunakan dalam *update* Fowler–Noll–Vo *hash* serta BUFFER_SIZE adalah ukuran *buffer* yang digunakan untuk membaca isi *file*. Gambar 3.6 berikut adalah rancangan *class FuzzyHashing*.



Gambar 3.6. Rancangan *class FuzzyHashing*

Method Calculate dengan parameter *filename* atau *filestream* digunakan untuk menghitung *fuzzy hash* yang menghasilkan *hash* dengan tipe data *Signature*. Gambar 3.7 berikut adalah contoh sederhana cara memanfaatkan *library fuzzy hash* untuk menghasilkan *fuzzy hash* menggunakan *console application.*

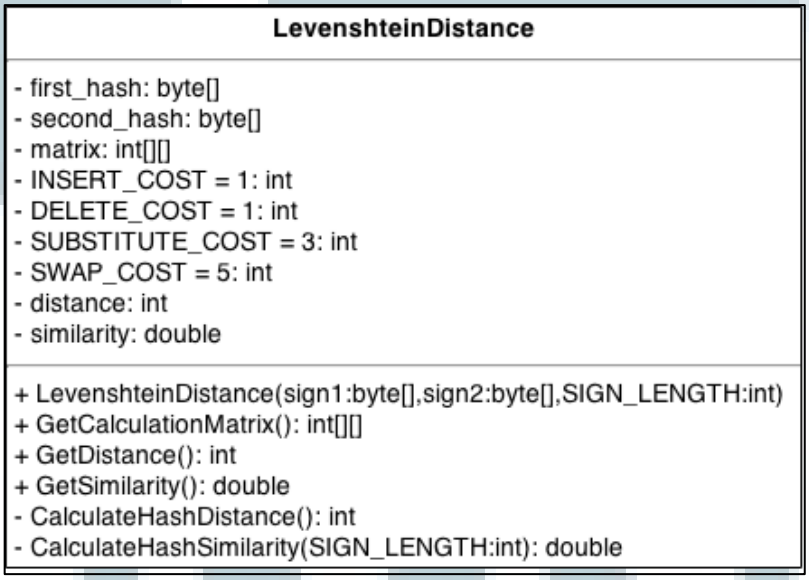
```
Signature sign1 = FuzzyHashing.Calculate("F:\test.txt");
Console.WriteLine(sign1.ToString());
```

Gambar 3.7. Contoh implementasi cara menghasilkan *fuzzy hash* dari suatu *file*

Method Compare yang terdapat dalam *class FuzzyHashing* digunakan untuk membandingkan kemiripan antara dua *signature*. *Method* tersebut secara internal memanggil *method SimilarityValue* yang mengembalikan nilai bertipe *double* dengan rentang nilai 0,00 – 100,00. Nilai tersebut berasal dari penghitungan hasil *distance* yang menggunakan *class LevenshteinDistance*. Rancangan *class LevenshteinDistance* dijelaskan dalam sub-bab 3.5.2 berikut.

3.5.2 Perancangan Fungsi Pembandingan Hash

Class LevenshteinDistance dibangun berdasarkan *pseudocode* algoritma 3 pada sub-bab 2.6. *Class* ini digunakan untuk menghitung *distance* dalam kumpulan *bytes* antara dua *signature* dan dapat mengembalikan *distance* tersebut dalam nilai bertipe *double* dari 0,00 – 100,00 yang merepresentasikan kemiripan antara dua *signature* yang dihitung. *Class* ini akan dipanggil secara internal oleh *method SimilarityValue* yang berada dalam *class FuzzyHashing*. Parameter yang dikirim adalah kumpulan *bytes* yang berasal dari dua *signature* dan panjang *hash* maksimal, sesuai dengan nilai SIGN_LENGTH, yaitu 64. Gambar 3.8 berikut menunjukkan rancangan *class LevenshteinDistance.*

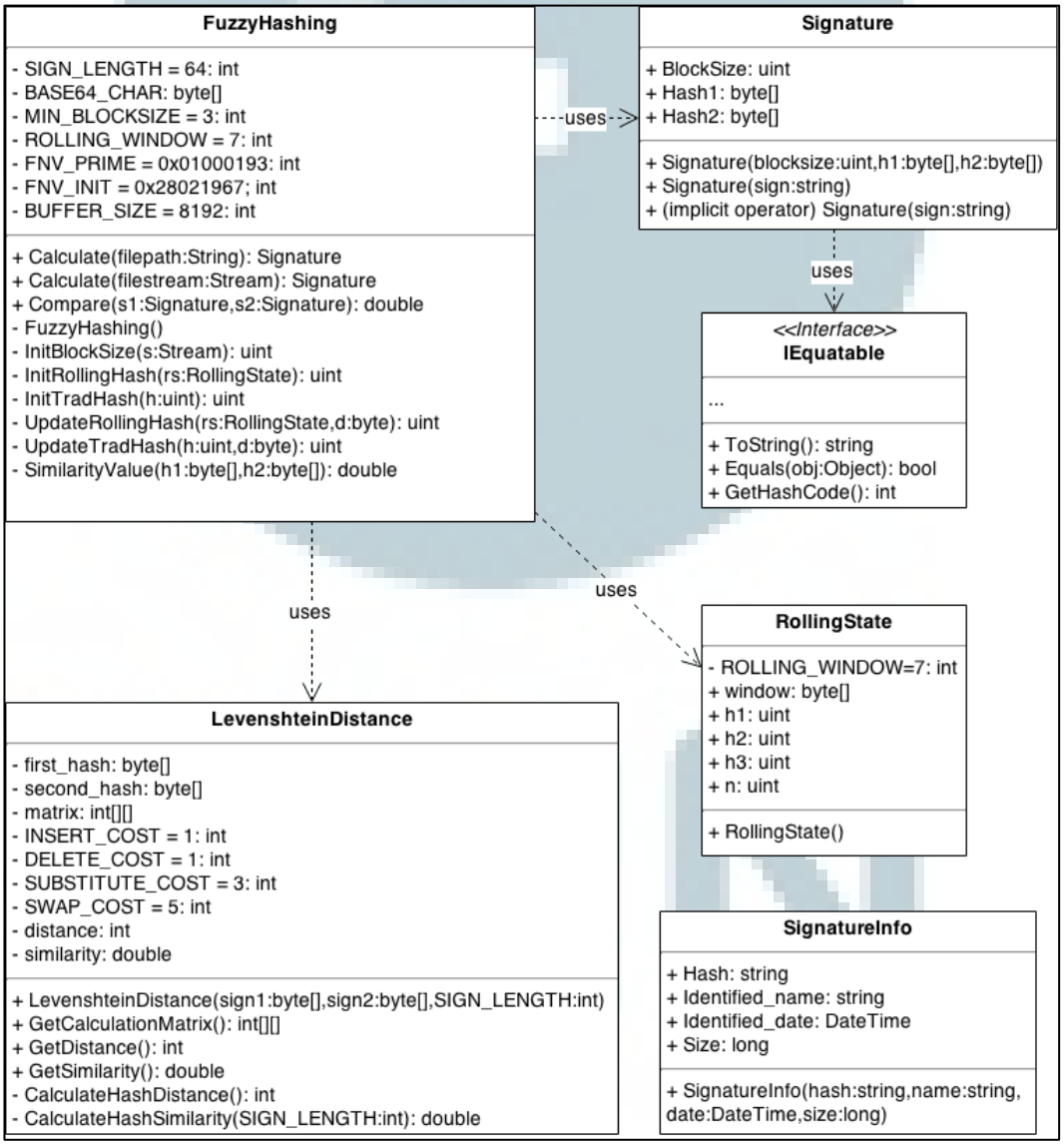


Gambar 3.8. Rancangan *class LevenshteinDistance*

Method GetDistance mengembalikan nilai *distance* dalam *integer* yang merepresentasikan *cost* yang dibutuhkan untuk mengubah kumpulan *bytes* antara dua buah *signature.* Nilai *distance* tersebut berasal dari penghitungan internal yang dilakukan *method CalculateHashDistance* ketika *constructor class* dijalankan atau ketika objek *LevenshteinDistance* dibuat. *Method GetSimilarity*

mengembalikan *distance* dalam nilai derajat kemiripan bertipe *double* dari 0,00 – 100,00. Nilai kemiripan ini berasal dari penghitungan internal sesuai dalam rumus (2.3) yang dilakukan *method CalculateHashSimilarity* ketika *constructor class* dijalankan. *Method GetCalculationMatrix* mengembalikan *array integer* 2D, digunakan untuk melihat angka-angka dalam proses penghitungan *distance* dengan pendekatan *dynamic programming*.

3.5.3 Hubungan antar Class dalam Library Fuzzy Hash



Gambar 3.9. *Class Diagram* dalam *library fuzzy hash*

Pada sub-bab 3.5.1 dan 3.5.2 sebelumnya telah dijelaskan satu per satu *class* yang berada di dalam *library fuzzy hash*. Gambar 3.9 menggambarkan *class-class* tersebut dalam suatu *class diagram*. Seperti yang ditunjukkan pada gambar tersebut, terlihat *class SignatureInfo* tidak memiliki relasi dengan *class* lainnya. *Class SignatureInfo* memang tidak digunakan secara langsung pada *class FuzzyHashing* dalam *library*. *Class* ini akan digunakan secara langsung dari aplikasi *Database Signature Builder* dan *Malware Detector* untuk mengenkapsulasi informasi *malware* yang di-*load* dari tabel *signature* di dalam *database* (lihat tabel 3.2) untuk diteruskan ke dalam aplikasi.

Gambar 3.10 berikut adalah contoh implementasi sederhana cara memanfaatkan *library* yang telah lengkap tersebut untuk menghasilkan *fuzzy hash* dan membandingkan *fuzzy hash* dua buah *file* menggunakan *console application*.

```
...
using FuzzyHashLib;
...

Signature sign1 = FuzzyHashing.Calculate("F:\prog.exe");
Signature sign2 = FuzzyHashing.Calculate("F:\progc.exe");
Console.WriteLine("FuzzyHash1 = " + sign1.ToString());
Console.WriteLine("FuzzyHash2 = " + sign2.ToString());
Console.WriteLine("Similarity = " +
FuzzyHashing.Compare(sign1,sign2) + " %");
...

Output:
FuzzyHash1 =
192:ODHKC75Zr3yqMPTRqz9SAPrOkjyFoqcCDq:M75ZrCqMdqhffeaqrW
FuzzyHash2 =
192:cDHKC75Zr3yqMPTRqz9SAPrOkjyFoqcCDq:i75ZrCqMdqhffeaqrW
Similarity = 97.06 %
```

Gambar 3.10. Contoh penggunaan *library fuzzy hash* pada *console application*

3.6 Perancangan Aplikasi

Aplikasi yang dibangun terdiri dari tiga aplikasi yang saling terkait, yaitu *Dashboard*, *Database Signature Builder*, dan *Malware Detector*. Berikut penjelasan dari masing-masing aplikasi dalam sub-bab 3.6.1 sampai dengan 3.6.3.

3.6.1 Aplikasi Dashboard

Dashboard dibangun sebagai aplikasi *launcher* dari aplikasi *Database Signature Builder* dan *Malware Detector*. *Dashboard* dibuat untuk memudahkan *user* menjalankan aplikasi *Database Signature Builder* dan aplikasi *Malware Detector* dari satu *form launcher*. Agar kedua aplikasi tersebut dapat dijalankan melalui *Dashboard, path* atau lokasi *executable file*-nya harus telah terkonfigurasi dengan benar. Jendela aplikasi *Dashboard* dapat di-*minimize* ke *system tray* dan selama *Dashboard* tidak secara eksplisit di-*exit* oleh *user*, aplikasi *Dashboard* berjalan di *background*. Jika *Dashboard* sedang berjalan di *background*, aplikasi ini juga berperan sebagai *detector* jika ada *removable drive* yang baru terpasang dan muncul *message* apakah *removable drive* tersebut ingin di-*scan* atau tidak, apabila lokasi *executable file Malware Detector* dan *database signature default* telah terkonfigurasi dengan benar. Pada *Dashboard*, isi *database signature default* dapat diperbaiki dengan disediakan operasi *update* bagian *identified_name*-nya dan operasi hapus *record hash*. Selain itu, terdapat fungsi tambahan untuk mengatur *launcher* dari *context-menu file explorer*. *Context-menu* pada *file explorer* berupa *menu shortcut* untuk menghitung nilai *fuzzy hash* atau *hash* SHA256 dari kumpulan *file* dalam satu *folder* beserta *subfolder* menggunakan *Database Signature Builder* dan *menu shortcut* untuk melakukan *scan* kumpulan *file* dalam satu *folder* beserta *subfolder* menggunakan *Malware Detector*.

3.6.2 Aplikasi Database Signature Builder

Database Signature Builder dibangun untuk melaksanakan penghitungan *fuzzy hash* dan SHA256 (sebagai pembanding) dari sampel *malware* serta menyimpannya ke dalam *database* SQLite3. Sampel *malware* yang telah dikumpulkan ke dalam suatu *folder*, akan di-*load* menggunakan aplikasi ini kemudian dihitung nilai *hash*-nya. Setelah dihitung nilai *hash*-nya, dapat dilakukan ekspor atau penyimpanan informasi dari *malware* yang telah dihitung ke dalam *file-based database* SQLite3. Informasi yang disimpan sesuai dengan kolom-kolom yang telah dijelaskan dalam tabel 3.2. Apabila dilakukan penghitungan *hash* SHA256, maka hasil ekspor berupa *file* dengan penanda ekstensi .sha256. Apabila dilakukan penghitungan *fuzzy hash*, maka hasil *ekspor* berupa *file* dengan penanda ekstensi .fuzzy. Jadi, satu *file database* hanya berisi *hash* sejenis. Ekspor ke dalam *database* dapat dilakukan dalam mode membuat *file database* baru, ataupun dalam mode menambah informasi *malware* ke *database* yang sudah diekspor sebelumnya (*insert to existing database*).

3.6.3 Aplikasi Malware Detector

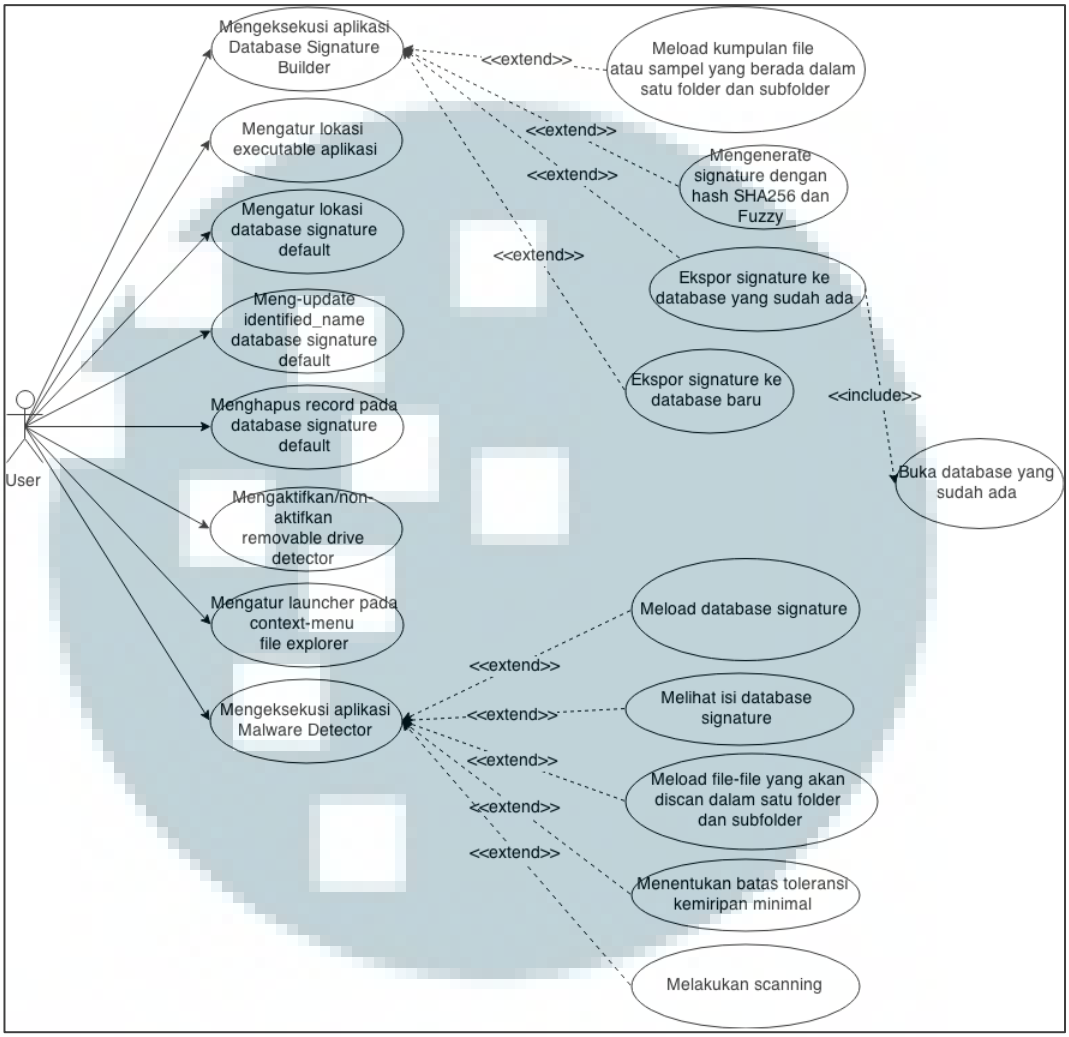
Malware Detector dibangun untuk melaksanakan pengujian apakah implementasi *fuzzy hash* dapat meningkatkan jumlah deteksi *malware* dibandingkan dengan SHA256 untuk jumlah sampel yang sama berdasarkan *database signature* yang telah dibuat menggunakan *Database Signature Builder*. *Signature* sampel *malware* yang telah disimpan ke dalam SQLite3 *database*, akan di-*load* menggunakan aplikasi ini dan disimpan ke dalam struktur data *Hashtable*. *Database signature* yang dapat di-*load* hanya satu di saat yang bersamaan, yang berisi *signature* SHA256 saja atau yang berisi *signature fuzzy hash* saja. Setelah

itu, *user* dapat me-*load file-file* yang berada di dalam satu *folder* dan *subfolder*-nya. Apabila *database* yang di-*load* berupa *database* .sha256, maka *file-file* akan dihitung nilai SHA256-nya saja, begitu juga apabila *database* yang di-*load* berupa *database* .fuzzy, maka *file-file* akan dihitung nilai *fuzzy hash*-nya saja. Dalam penelitian ini, digunakan metode deteksi *signature-based* sederhana berdasarkan nilai *fuzzy hash* dan SHA256 sebagai pembanding dari *malware* yang *hash*-nya telah disimpan. Deteksi dilakukan dengan menghitung nilai *hash* dari *file* yang sedang di-*scan* dan membandingkannya dengan kumpulan *hash* yang terdapat pada *database*. Pada SHA256, pencarian pada *database* dapat dilakukan melalui pasangan *key-value* pada sebuah *Hashtable* (*hash* sebagai *key*, informasi *malware* yang dienkapsulasi dalam *object SignatureInfo* sebagai *value*), apabila tidak ditemukan, maka dianggap *file* yang di-*scan* bersih. Sedangkan pada *fuzzy hash*, pencarian dimulai pada *database* melalui pasangan *key-value* untuk mendapatkan informasi *malware* dengan *fuzzy hash* yang sama persis. Apabila tidak ditemukan, dilakukan *looping* pada semua *key* untuk mendapatkan informasi *malware* dengan *similarity* tertinggi pada nilai toleransi kemiripan minimal tertentu sesuai yang telah ditentukan oleh *user*. Oleh karena itu, bila menggunakan *database signature* .fuzzy, *user* dapat menentukan toleransi kemiripan minimal dalam rentang 20,00 – 100,00.

3.6.4 Use Case Diagram

Use Case Diagram adalah diagram yang menjelaskan interaksi *user* dan sistem yang berupa perilaku yang dapat dilakukan oleh *user*. Sistem terdiri dari tiga aplikasi yang saling terkait, yaitu *Dashboard*, *Database Signature Builder*,

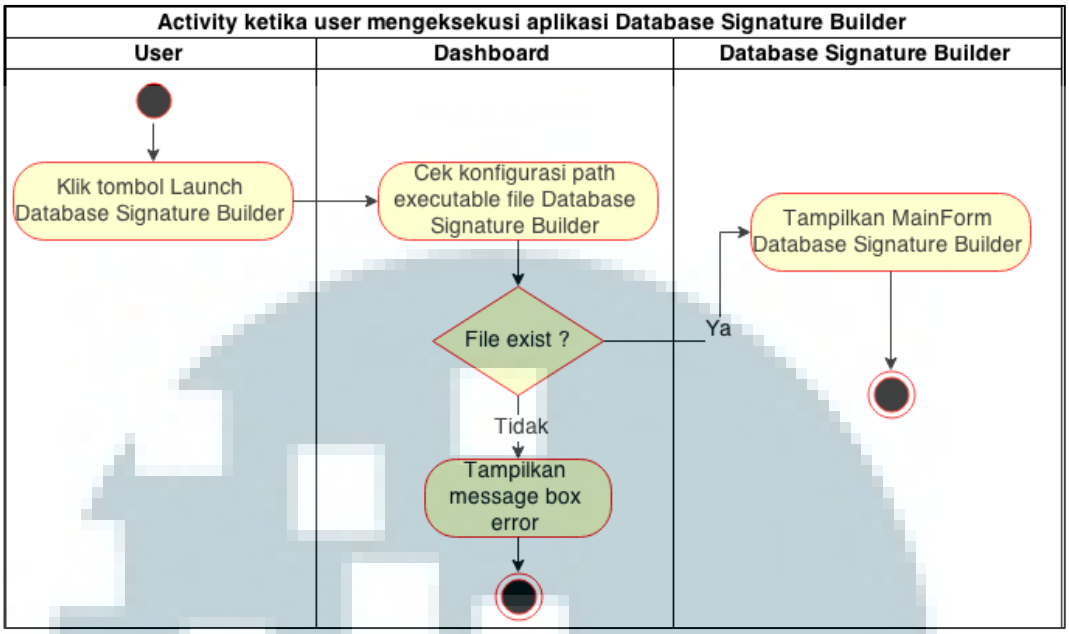
dan *Malware Detector*, serta dua buah *library*, yaitu *library fuzzy hash* dan *library* SQLite3. Gambar 3.11 menunjukkan *use case diagram* keseluruhan aplikasi.



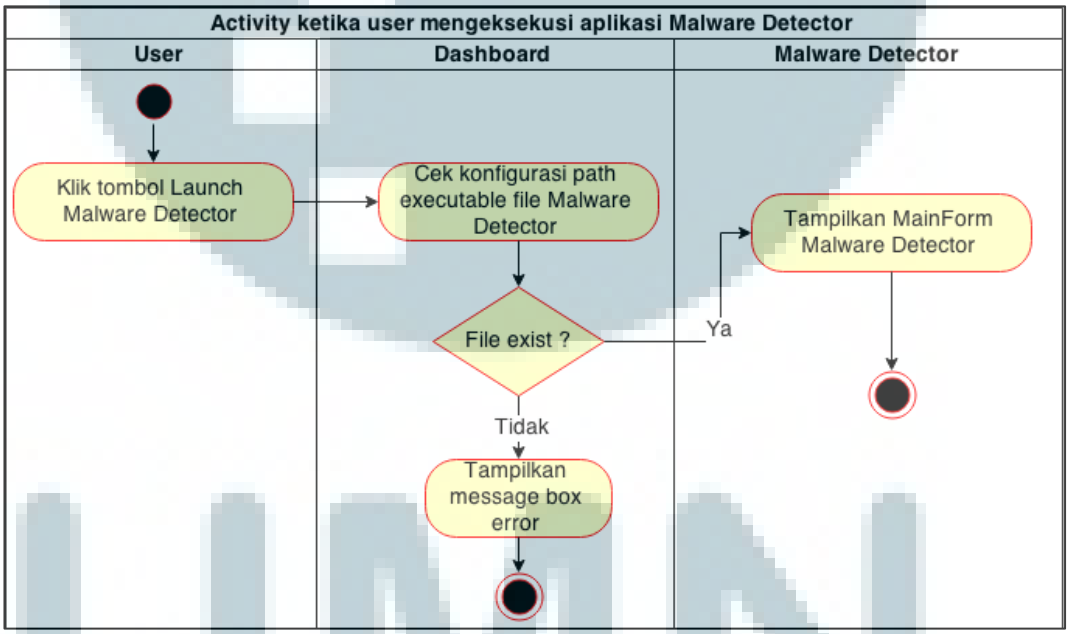
Gambar 3.11. *Use case diagram* keseluruhan aplikasi

3.6.5 Activity Diagram

Activity Diagram adalah diagram yang menjelaskan proses yang terjadi ketika *user* menjalankan atau mengerjakan suatu aktivitas. Gambar 3.12 sampai dengan gambar 3.19 berikut menunjukkan proses yang terjadi ketika *user* menjalankan aktivitas dalam *Dashboard* dan menunjukkan keterkaitan antar ketiga aplikasi serta *library fuzzy hash* yang dibuat.

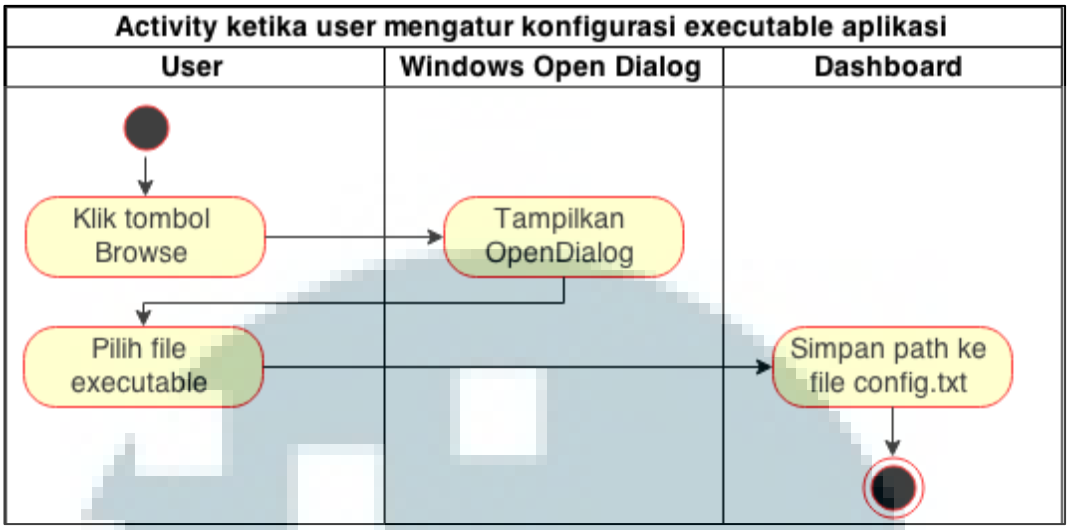


Gambar 3.12. *Activity Diagram* ketika mengeksekusi *Database Signature Builder*

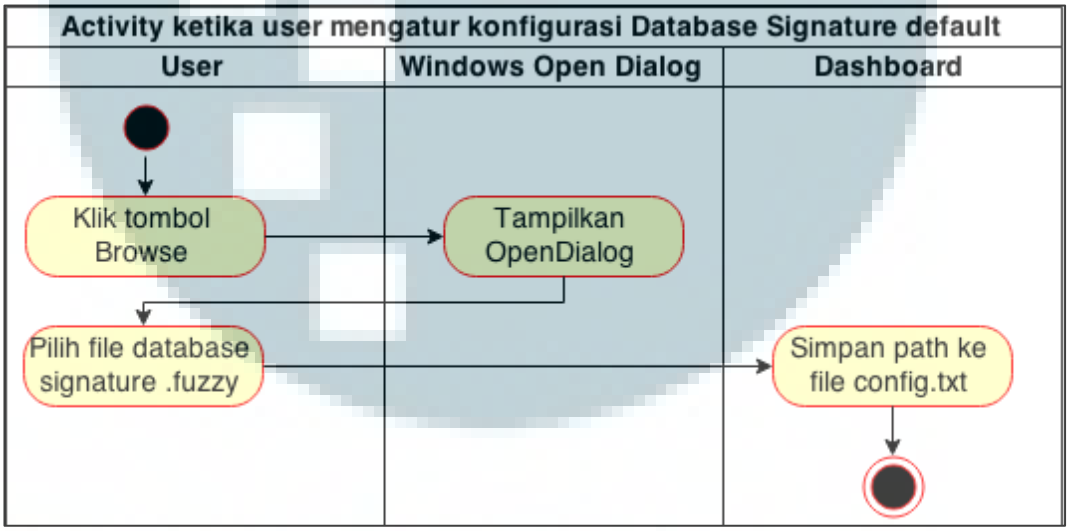


Gambar 3.13. *Activity Diagram* ketika mengeksekusi *Malware Detector*

Gambar 3.12 dan 3.13 menunjukkan *activity* ketika *user* mengeksekusi aplikasi *Database Signature Builder* dan *Malware Detector* dari *Dashboard*.

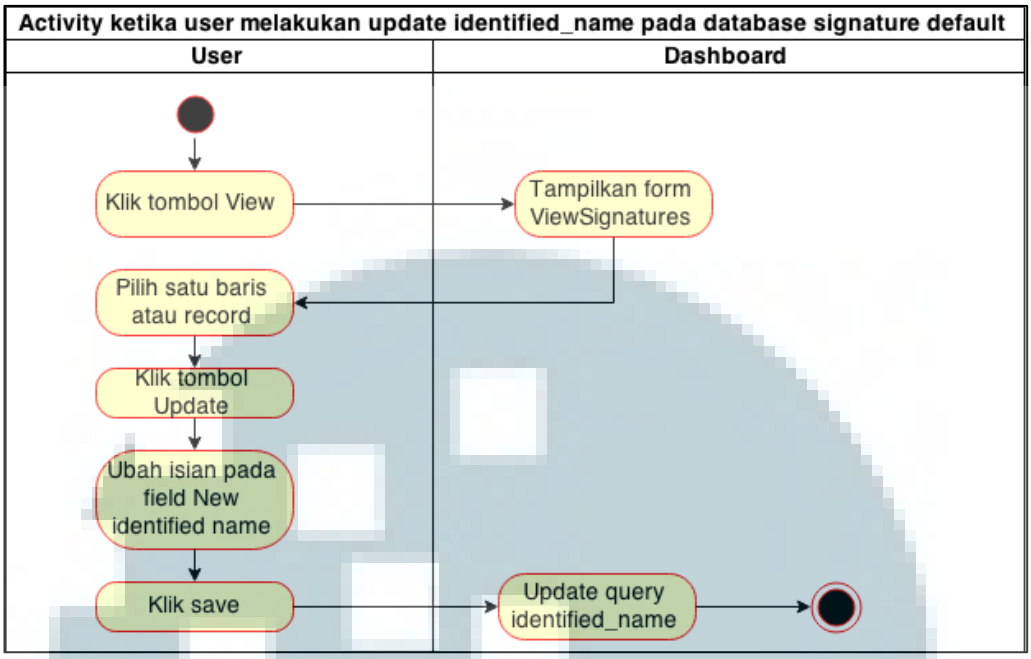


Gambar 3.14. *Activity Diagram* ketika mengatur konfigurasi *file executable* aplikasi

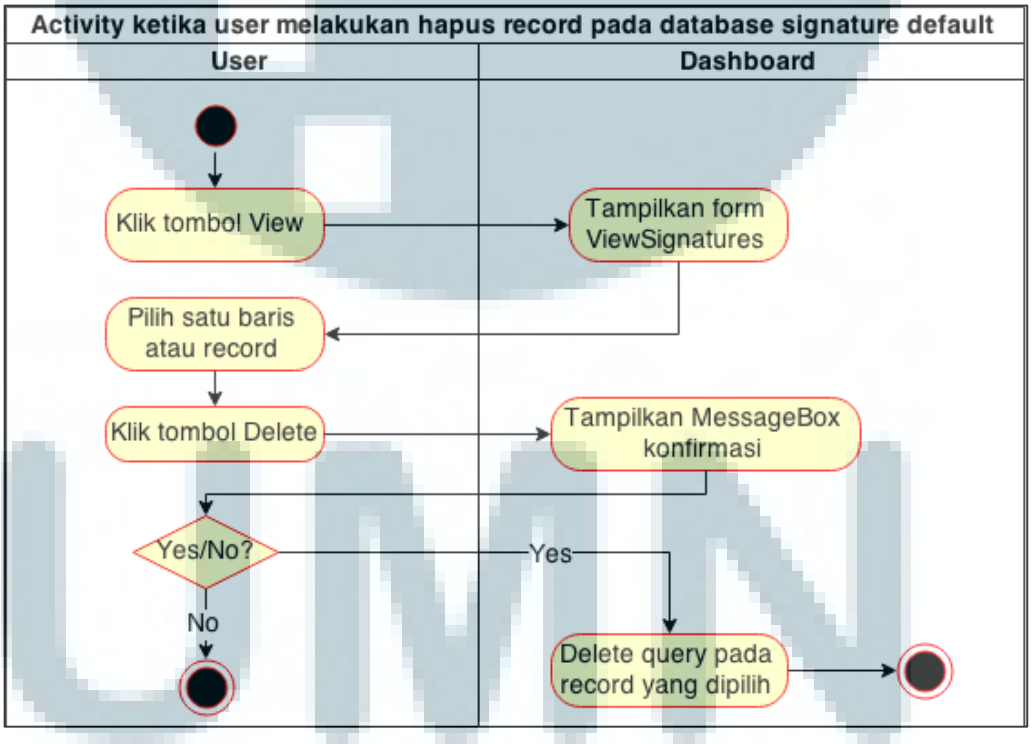


Gambar 3.15. *Activity Diagram* ketika mengatur konfigurasi *file database signature default*

Gambar 3.14 menunjukkan *activity* ketika *user* mengatur konfigurasi *path executable file* untuk *Database Signature Builder* dan *Malware Detector*. Gambar 3.15 menunjukkan *activity* ketika *user* mengatur konfigurasi *path file database signature default*, yaitu *file database* yang dihasilkan *Database Signature Builder* dan akan dijadikan acuan *default* ketika *Malware Detector* melakukan *scanning*.

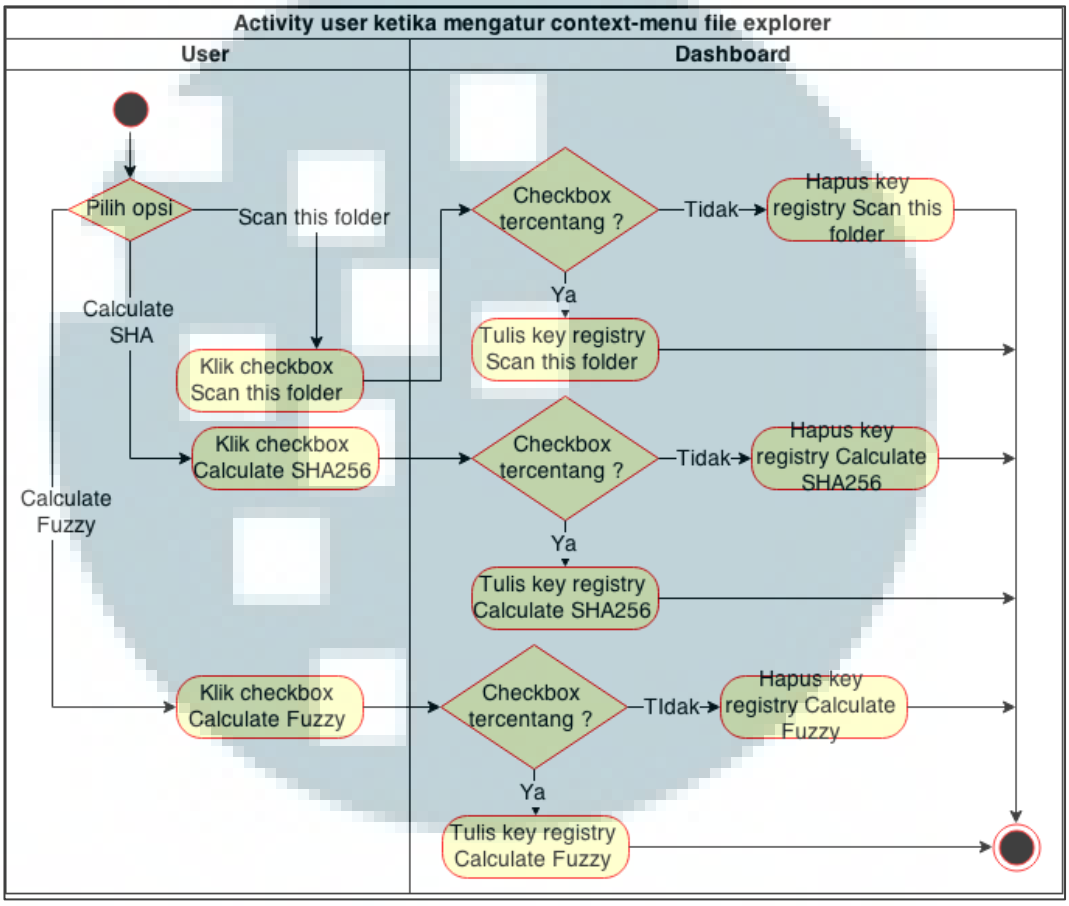


Gambar 3.16. *Activity Diagram* ketika melakukan *update identified_name malware* pada *database signature default*



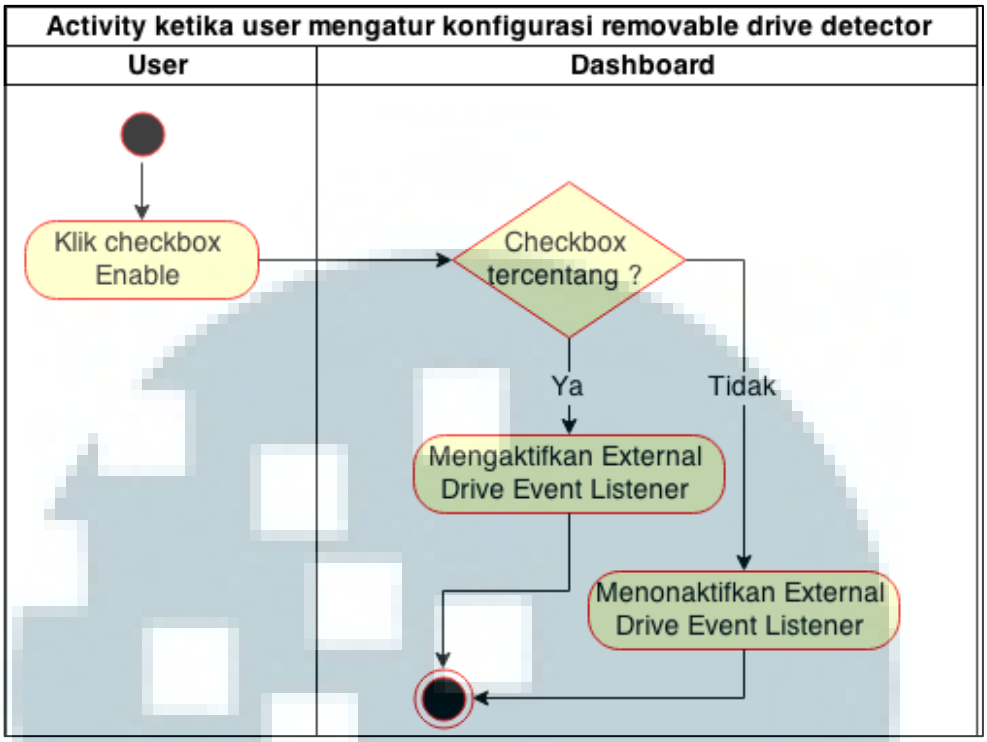
Gambar 3.17. *Activity Diagram* ketika melakukan hapus *record* informasi *malware* pada *database signature default*

Gambar 3.16 menunjukkan *activity* ketika *user* melakukan *update* informasi *malware* bagian *identified_name* pada *database signature default*. Gambar 3.17 menunjukkan *activity* ketika *user* melakukan hapus *record* informasi *malware* pada *database signature default*.



Gambar 3.18. *Activity Diagram* ketika mengaktifkan/menonaktifkan *context-menu file explorer*

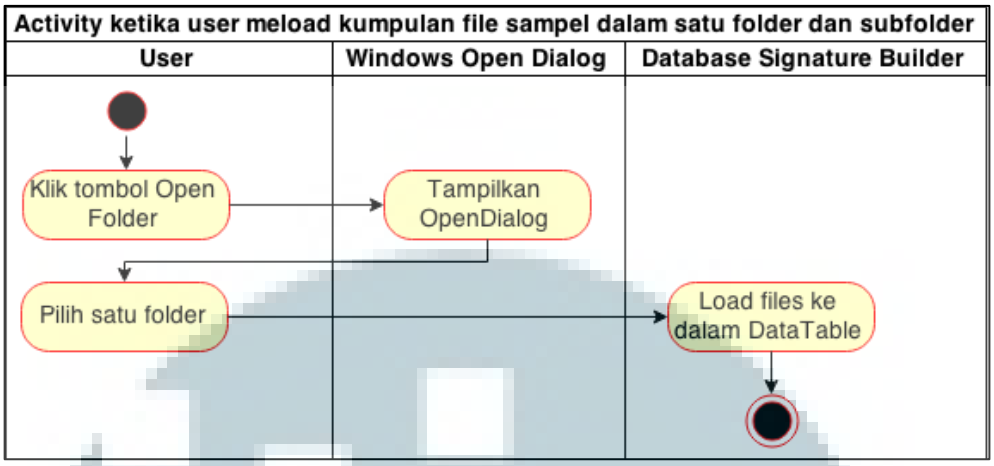
Gambar 3.18 menunjukkan *activity* ketika *user* mengaktifkan atau menonaktifkan *context-menu file explorer*. *Menu* ini akan muncul dalam kumpulan *menu* ketika sebuah *folder* di *file explorer* dilakukan klik kanan. Opsi ini akan berjalan sempurna apabila aplikasi dijalankan dalam mode *Administrator*, karena untuk mendukung opsi ini, program melakukan modifikasi *registry*.



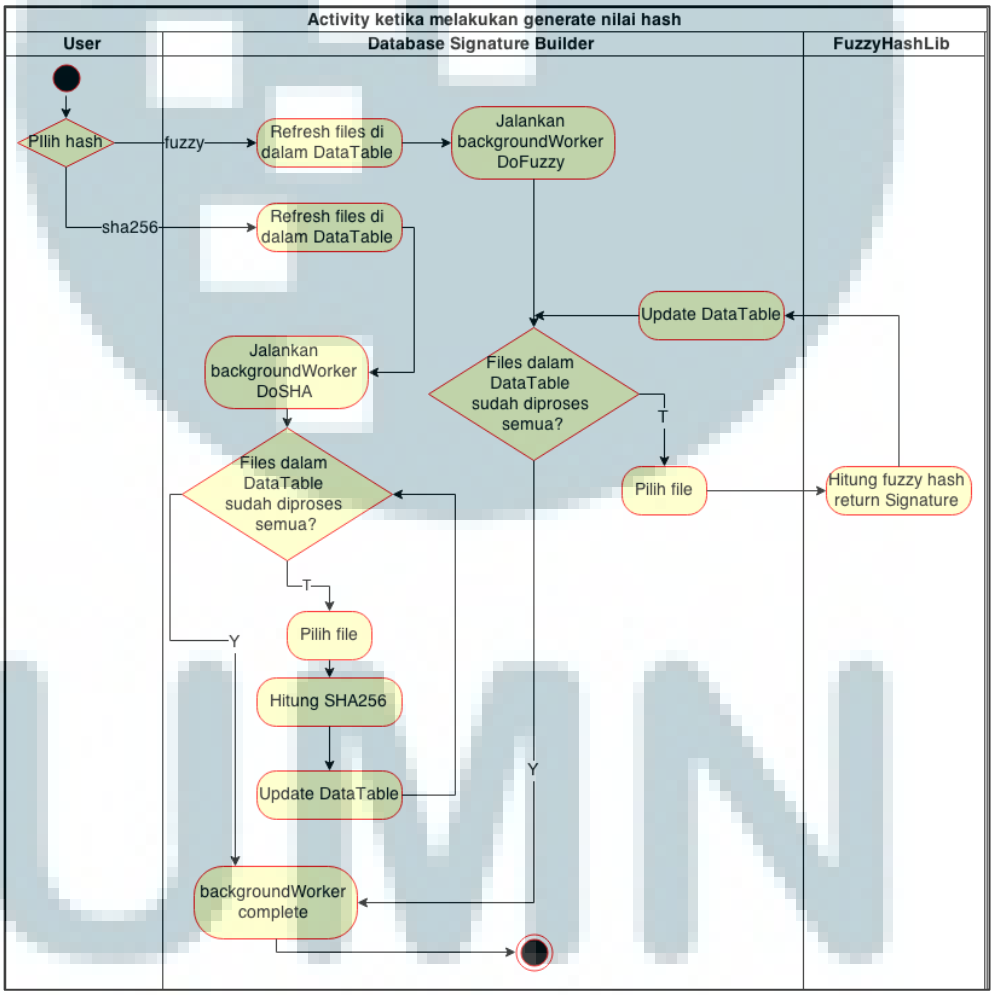
Gambar 3.19. *Activity Diagram* ketika mengaktifkan/menonaktifkan *removable drive detector*

Gambar 3.19 menunjukkan *activity* mengaktifkan/menonaktifkan *removable drive detector*. Untuk mendukung fungsi ini, digunakan *library* tambahan bernama *DriveDetector* yang dibangun oleh Dolinay (Dolinay, 2007). *Library* ini diintegrasikan di dalam aplikasi *Dashboard*. *DriveDetector* akan melakukan *monitoring* terhadap *system event* untuk mendeteksi *removable drive* yang baru terpasang dan terdeteksi oleh sistem. Opsi ini akan berjalan sempurna apabila aplikasi dijalankan dalam mode *Administrator*.

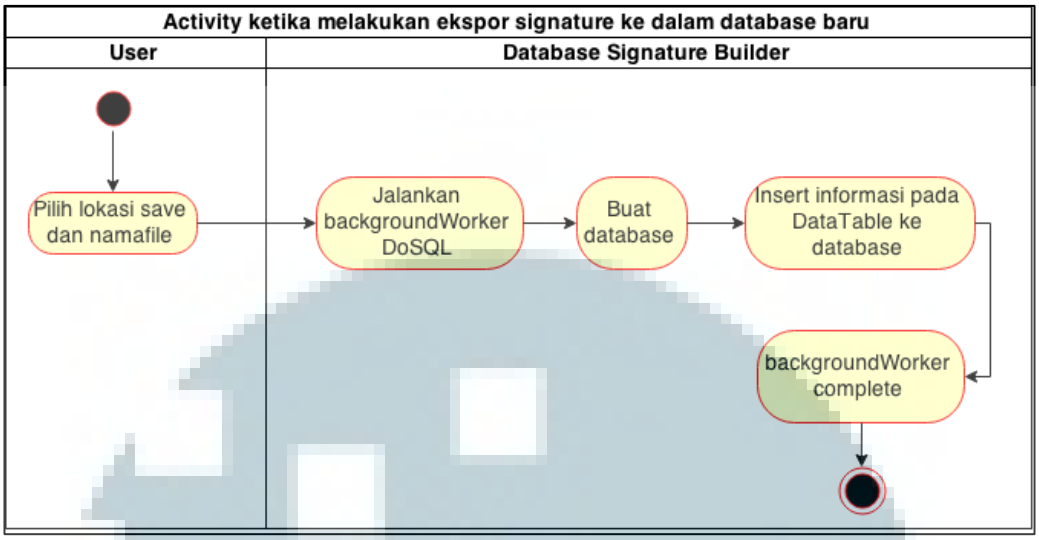
Gambar 3.20 sampai dengan gambar 3.33 berikut menunjukkan proses yang terjadi ketika *user* menjalankan aktivitas dalam *Database Signature Builder* dan *Malware Detector*, keterkaitan antar ketiga aplikasi serta *library fuzzy hash* yang dibuat.



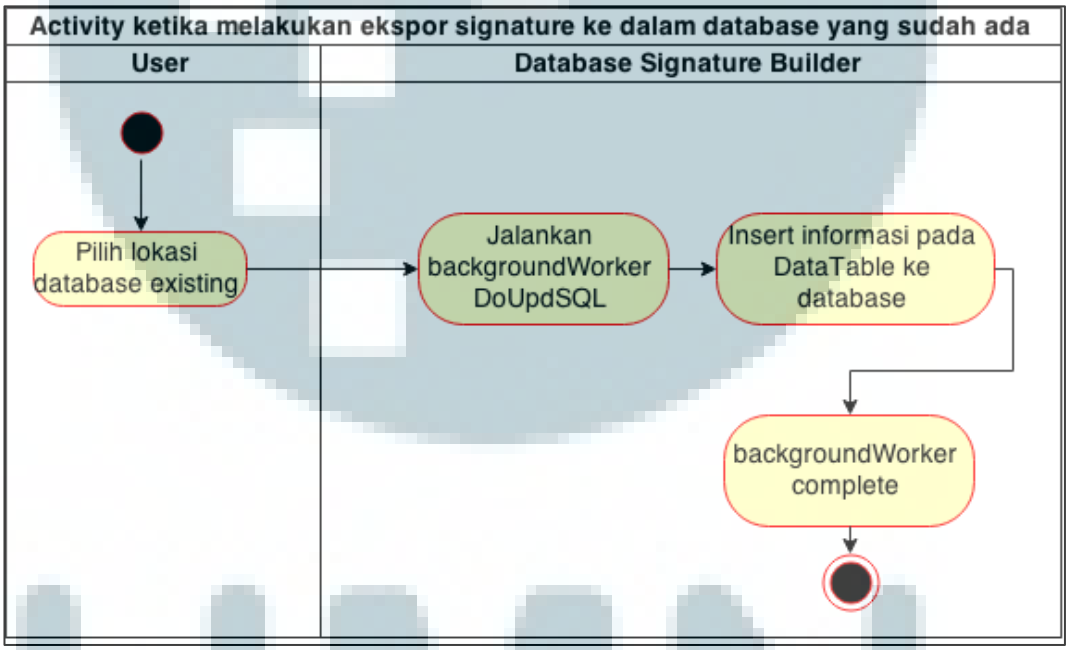
Gambar 3.20. *Activity Diagram* ketika me-*load file* sampel



Gambar 3.21. *Activity Diagram* ketika melakukan *generate* nilai *hash* dari *file* sampel yang telah di-*load*



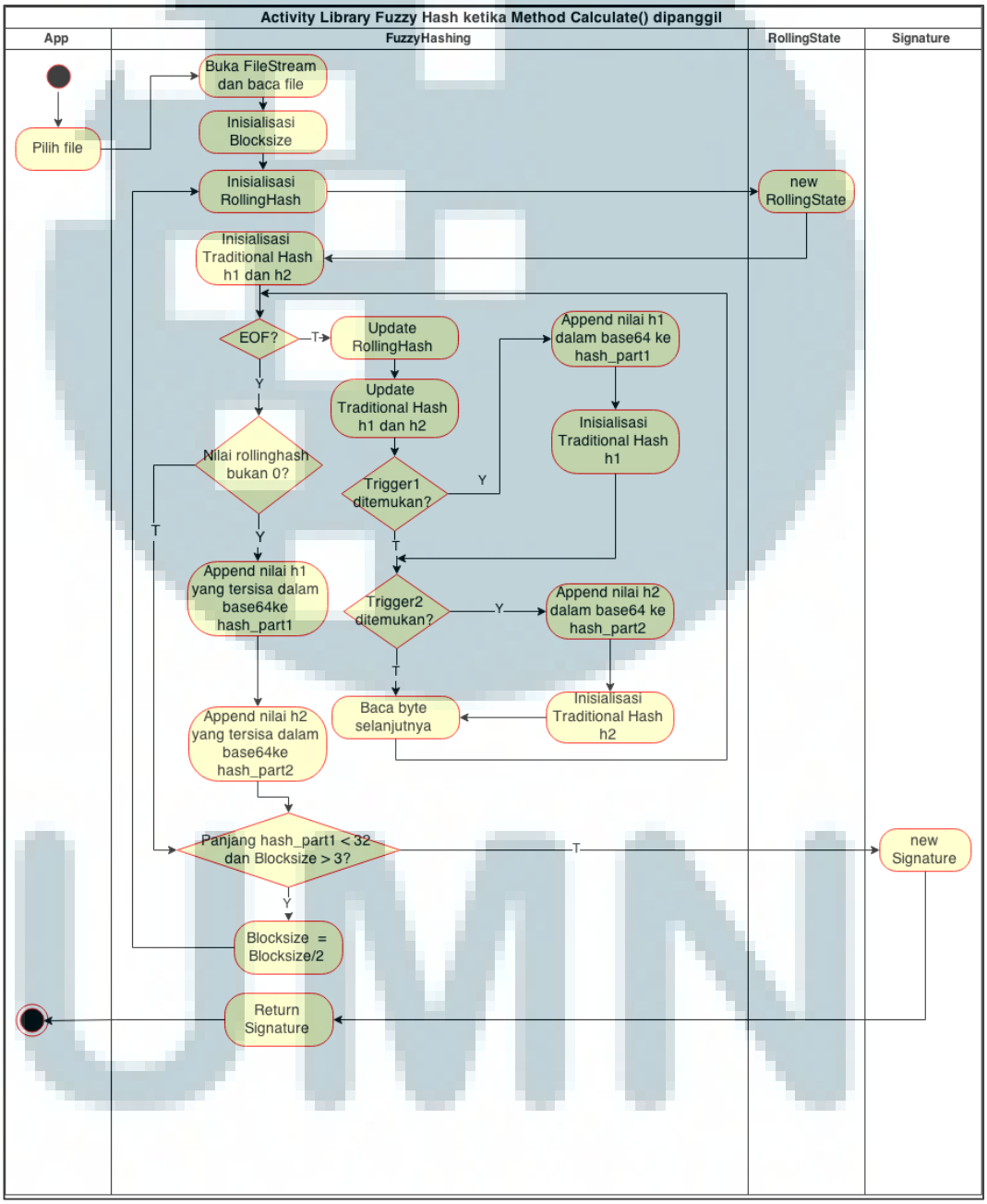
Gambar 3.22. *Activity Diagram* ketika melakukan ekspor ke dalam *database* baru



Gambar 3.23. *Activity Diagram* ketika melakukan ekspor ke dalam *database* yang sudah ada

Gambar 3.20 menunjukkan *activity* ketika *user* melakukan *load file* sampel ke dalam aplikasi. Kemudian, gambar 3.21 menunjukkan *activity* menghasilkan nilai *hash* SHA256 atau *fuzzy hash* dari *file* yang telah di-*load*. Selanjutnya, seperti yang ditunjukkan gambar 3.22 dan 3.23, nilai *hash* tersebut dapat diekspor

ke dalam *file database* baru ataupun ke *database* yang telah ada dengan struktur yang sama seperti tabel 3.2. *File database* ini akan digunakan oleh Aplikasi *Malware Detector*. Gambar 3.24 berikut menunjukkan *activity* yang terjadi dalam internal *library fuzzy hash* ketika dilakukan penghitungan *fuzzy hash* (pemanggilan *method Calculate*) suatu *file*.



Gambar 3.24. *Activity Diagram* ketika dilakukan penghitungan *fuzzy hash*

Gambar 3.25 dan 3.26 berikut menunjukkan contoh isi *database signature* yang dihasilkan. *File database* dibuka menggunakan aplikasi SQLiteMan.

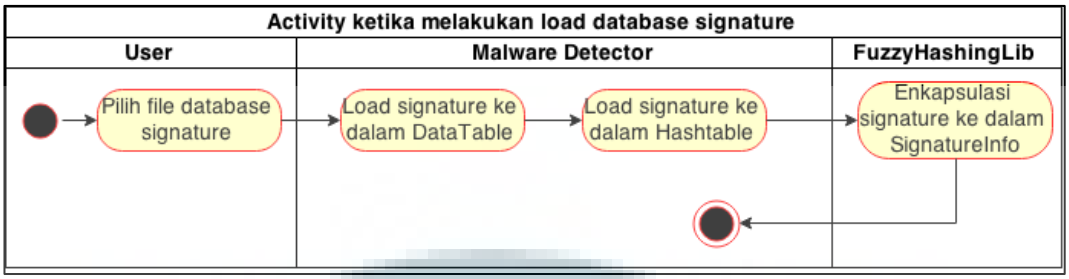
	hash	identified_name	identified_date	size
1	48:pPIKz2nf2sPBRez27TT27M7OQz2V2y8+wpDRERRWR...	Skripsi.sln	2015-02-25	2949
2	768:HaLOhOQ1SUhGKcwHEjdMUfXHSUhGKcwHEjdM...	Skripsi.v11.suo	2015-02-25	86528
3	24576:2NQOW38FYn8so0ZypaIf6zQVp+qE1QJdQYh:2N...	System.Data.SQLite.dll	2015-02-25	967168
4	96:Jo4+4Dcz02nXi7qmui13FDt09ibfx9m5lxCbPZxY0:9p...	Form1.cs	2015-02-25	3716
5	96:aph2atwZYL6YE+j+8CfvRqX/mh30yzbh:Cwa+e6YE+...	Form1.Designer.cs	2015-02-25	4893
6	96:fijrkiK5k5LPXbac9m5Lv6FzSvd4gIRjETUT2+0qSdvab...	Form1.resx	2015-02-25	5817
7	96:yWnpu5/ZgxyFEDxa4GE5vVBCHMOdsHS40b7M7W2...	FuzzyHashing01.csproj	2015-02-25	4089
8	12:V/DGr4KKr2nskiK3IS1ceAN5Hw8IJTuHmDbHUErUM...	Program.cs	2015-02-25	506
9	192:IutBqV9I58i1Xf7AU298SC5qqZrIzRnFNMgCcM/aPp...	FuzzyHashing01.exe	2015-02-25	12288
10	768:ioEjcDe5CzCeZCBjnDf3vKn3L+LOaQFT41HJWbz3W...	FuzzyHashing01.pdb	2015-02-25	34304

Gambar 3.25. Contoh isi *database signature* menggunakan *fuzzy hash*

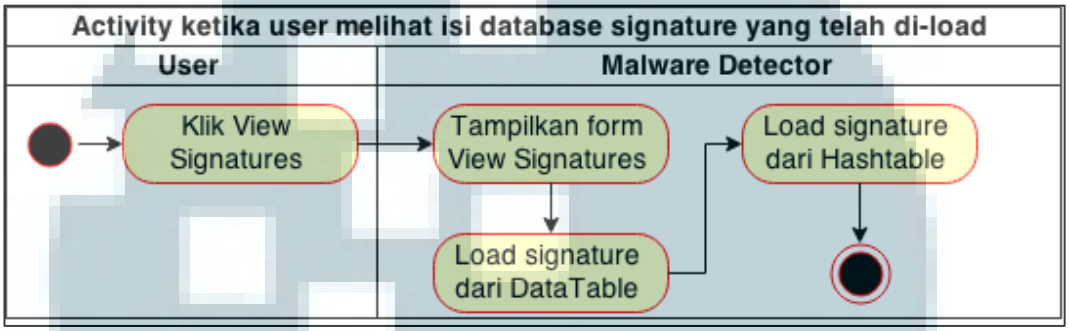
	hash	identified_name	identified_date	size
1	5e9297c141b1be07246efd80cdbc050eb695ae49bb8436952d...	Skripsi.sln	2015-02-25	2949
2	eb85f2d17e6c4c6d69a08adb09613dd6782cfd23263ec5ace1...	Skripsi.v11.suo	2015-02-25	86528
3	82aecf9b9064fba2fd08d859b66983ed6338bca0e34b8827f64...	System.Data.SQLite.dll	2015-02-25	967168
4	84abcf13c61dfab35b60ec2d46218c8a7fbef3f55d7a1c45f7fc...	Form1.cs	2015-02-25	3716
5	ff73fe060726e7e4262f7d4a4f306aa92a6fbad71f0a0d5863b1...	Form1.Designer.cs	2015-02-25	4893
6	4363cd7d5b8671c72442ce1a1bfc10d64ebd24b2d718b54bd...	Form1.resx	2015-02-25	5817
7	b4b6a8cded64af1632e201a2fc6f2b048fd9a301b9f5c070678...	FuzzyHashing01.csproj	2015-02-25	4089
8	17f2ab0ea31778344eaf014ba29047dae585b0725cee0129af7...	Program.cs	2015-02-25	506
9	310c7c385aa930686a284bb216148c93d8f9e7567cb03d0dae...	FuzzyHashing01.exe	2015-02-25	12288
10	cb4f7fc38abe4f0a35521961092c9cd72e19a298fdd0c247d05...	FuzzyHashing01.pdb	2015-02-25	34304

Gambar 3.26. Contoh isi *database signature* menggunakan SHA256

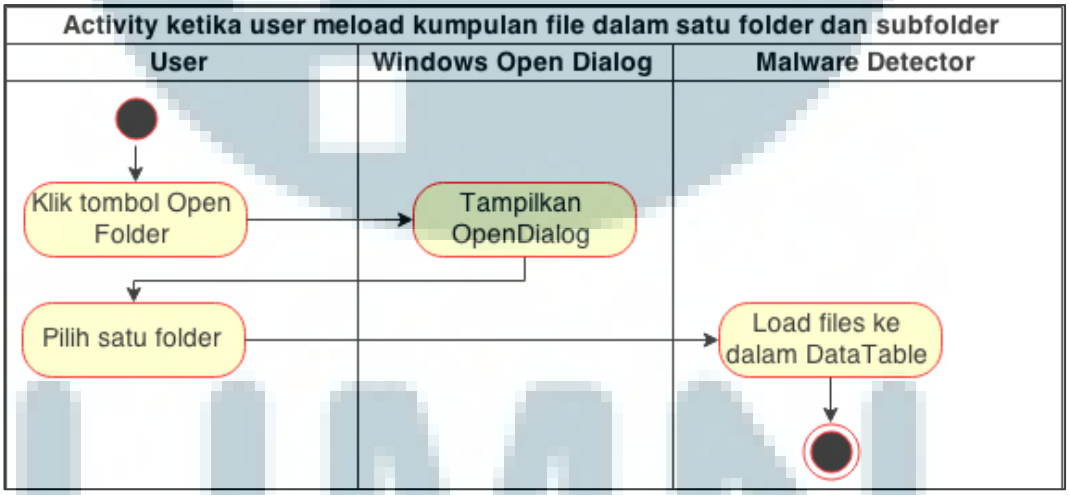
Jumlah *record* yang masuk ke dalam *database* belum tentu sama dengan jumlah *file* yang di-*hash*. Hal ini terjadi karena bila ditemukan nilai *hash* yang sama antara *record* yang sudah ada di *database* dan yang akan dimasukkan (terjadi *collision*), maka *record* tersebut tidak dimasukkan ke dalam *database*. Hal ini dilakukan mengingat apabila *hash* sama, maka isi *file*-nya juga sama. Jadi, informasi *malware* yang muncul pada saat deteksi adalah informasi pada *record* dengan *hash* yang telah ada di dalam *database*.



Gambar 3.27. *Activity Diagram* ketika melakukan *load file database signature*

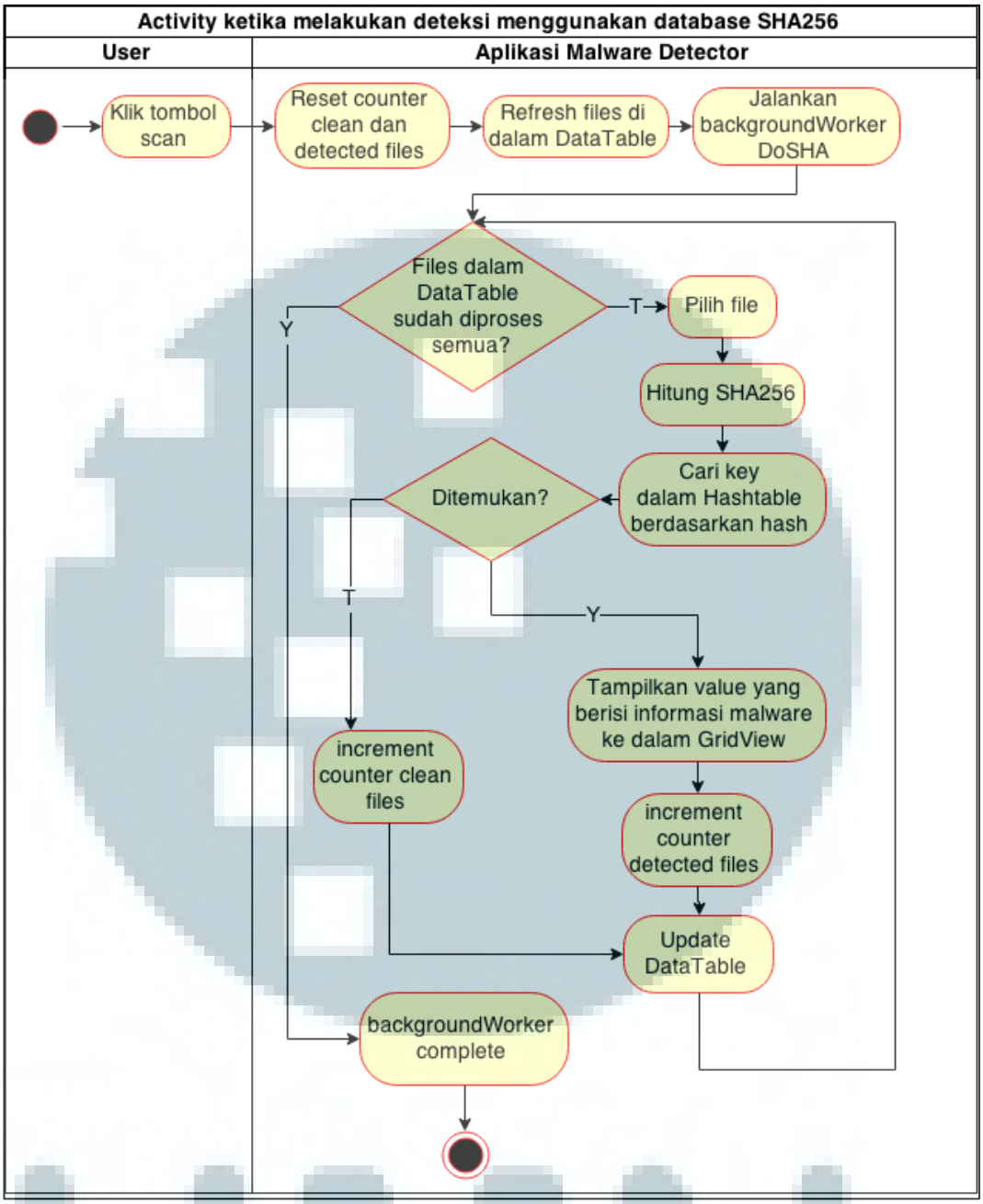


Gambar 3.28. *Activity Diagram* melihat isi *file database signature*



Gambar 3.29. *Activity Diagram* ketika melakukan *load file* yang akan di-*scan*

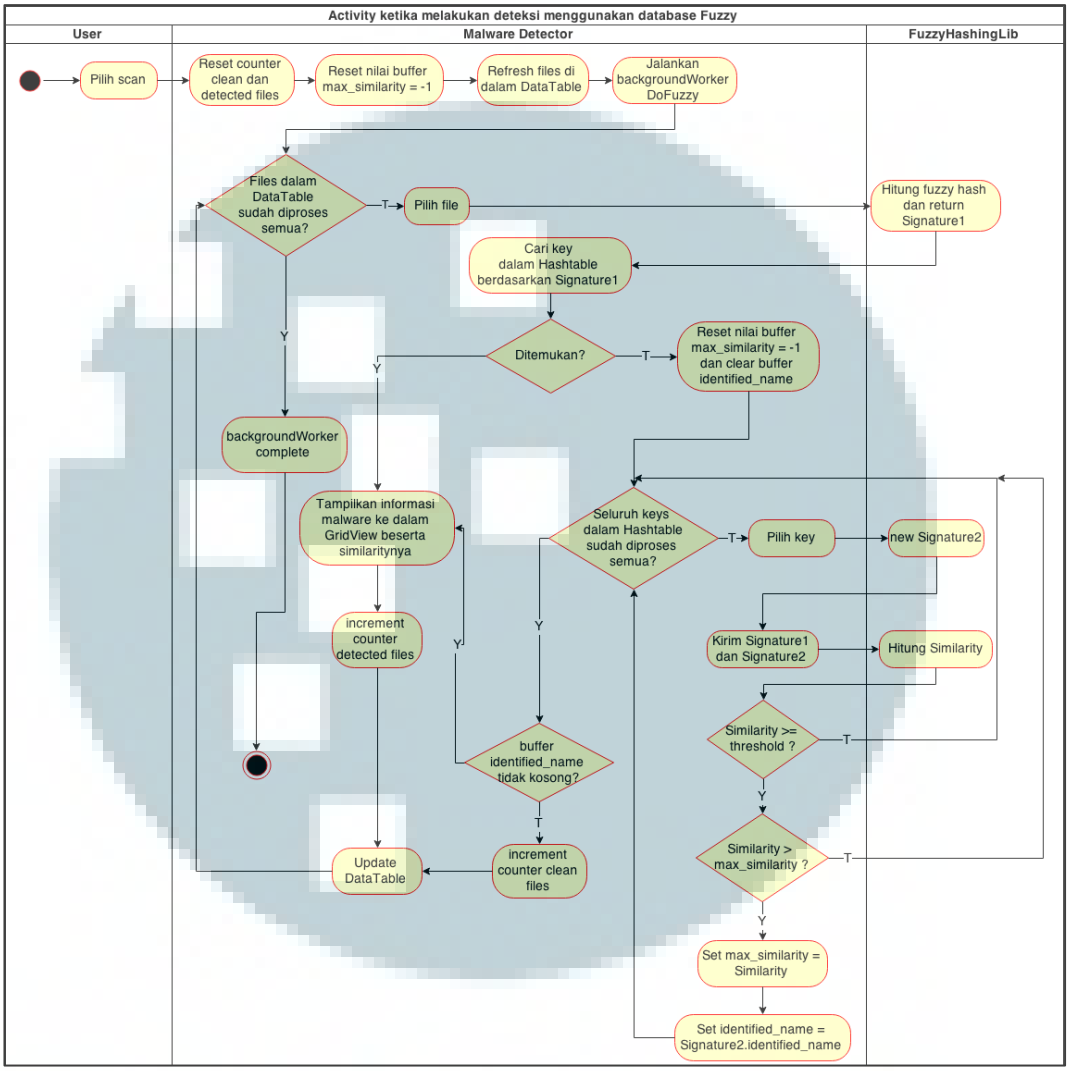
Gambar 3.27 menunjukkan *activity* ketika *user* melakukan *load file database signature*. Gambar 3.28 menunjukkan *activity* jika *user* ingin melihat informasi *hash* yang ada di dalam *database signature*. Setelah itu, *user* dapat melakukan *load file* yang akan di-*scan* seperti yang ditunjukkan gambar 3.29.



Gambar 3.30. *Activity Diagram* ketika melakukan deteksi menggunakan *database signature* SHA256

Gambar 3.30 menunjukkan *activity* deteksi menggunakan SHA256. Ketika *file* dihitung *hash*-nya, aplikasi melakukan pencarian/pengecekan apakah di dalam *Hashtable* ada *key* yang sesuai dengan nilai *hash*. Jika ditemukan, maka aplikasi akan menampilkan informasi *malware* yang berupa *identified_name* dalam *object*

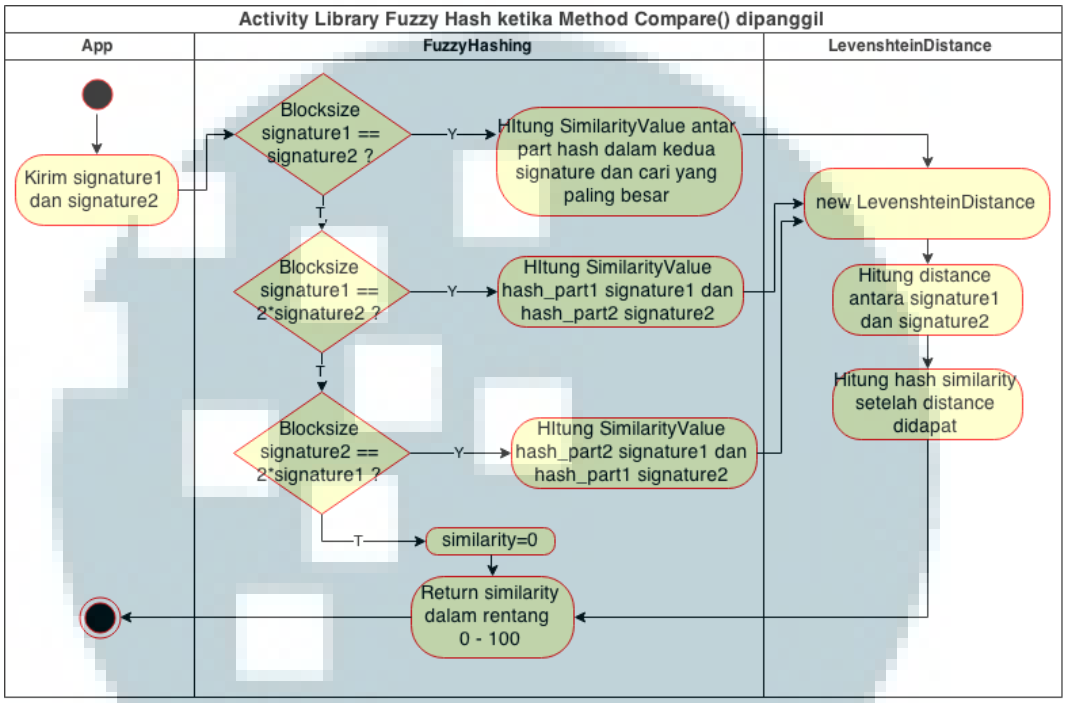
SignatureInfo yang tersimpan di dalam *Hashtable* tersebut, jika tidak ditemukan, akan diproses *file* selanjutnya hingga selesai.



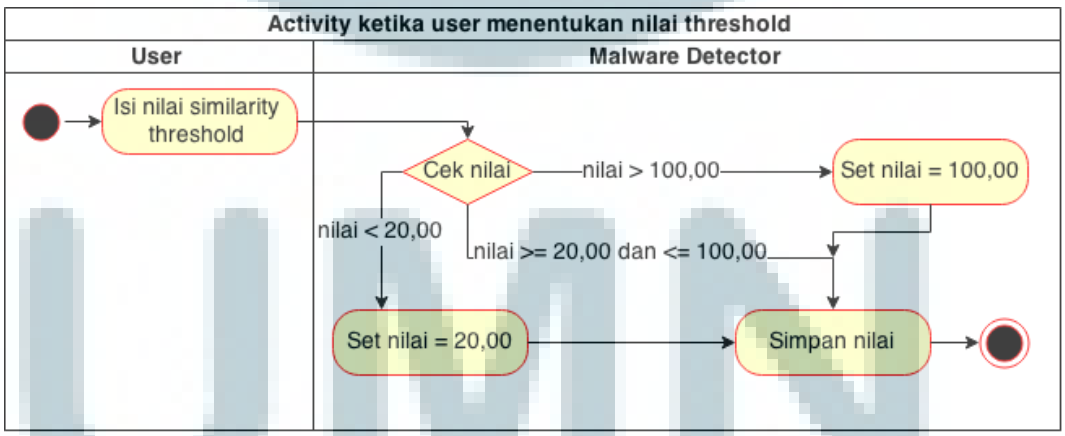
Gambar 3.31. *Activity Diagram* ketika melakukan deteksi menggunakan *database signature fuzzy*

Gambar 3.31 menunjukkan *activity* ketika *user* melakukan deteksi *malware* menggunakan *database signature fuzzy*. Ketika *file* dihitung *hash*-nya, aplikasi melakukan pencarian/pengecekan apakah di dalam *Hashtable* ada *key* yang sesuai dengan nilai *hash*. Jika ditemukan, maka aplikasi akan menampilkan informasi *malware* yang berupa *identified_name* dalam *SignatureInfo* yang

tersimpan di dalam *Hashtable* tersebut, jika tidak ditemukan, dilakukan *looping* pada semua *key* untuk mendapatkan informasi *malware* dengan *similarity* tertinggi pada nilai toleransi kemiripan minimal (*threshold*) sesuai pilihan *user*.



Gambar 3.32. *Activity Diagram* ketika dilakukan penghitungan *similarity*

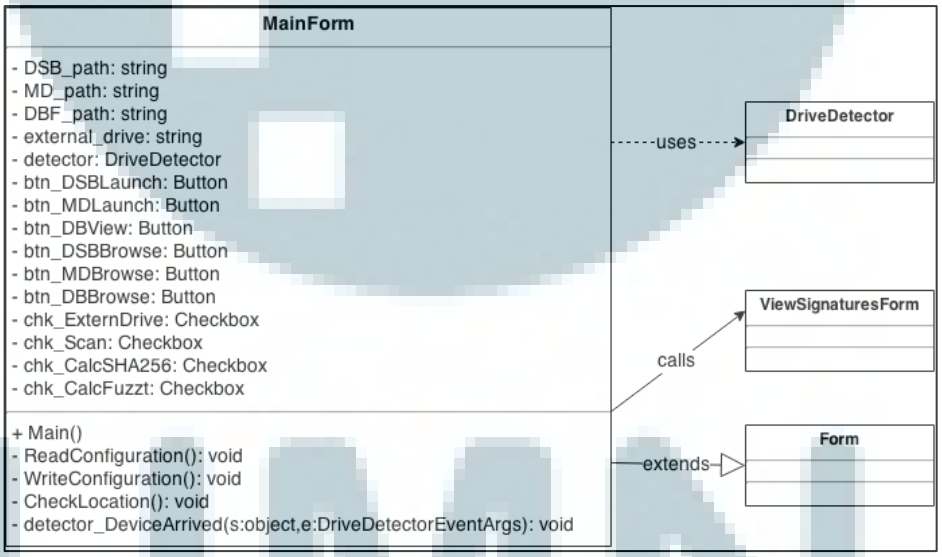


Gambar 3.33. *Activity Diagram* ketika penentuan nilai *similarity threshold*

Gambar 3.32 menunjukkan *activity* dalam internal *library fuzzy hash* ketika penghitungan *similarity* atau pemanggilan *method Compare* terhadap dua buah *signature*. Gambar 3.33 menunjukkan *activity* penentuan nilai *threshold*.

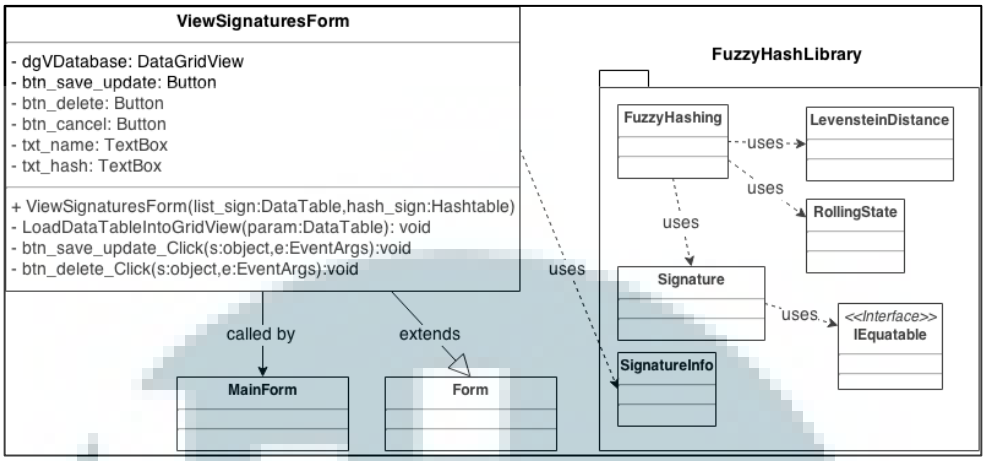
3.6.6 Class Diagram

Aplikasi *Dashboard* dibangun menggunakan bahasa pemprograman C# .NET 4.0 dengan model *windows-form application*. Aplikasi terdiri dari dua buah *form*. *Form* utama digunakan untuk pengaturan lokasi dan eksekusi aplikasi *Database Signature Builder* dan *Malware Detector*, pengaturan lokasi *database signature default*, pengaturan fungsi *removable drive detector*, dan pengaturan *launcher* pada *context-menu file explorer*. Gambar 3.34 menunjukkan rancangan *class form* utama pada *Dashboard*.



Gambar 3.34. *Class MainForm* pada *Dashboard*

Sedangkan *form view signatures* digunakan untuk melihat, meng-*update*, dan menghapus *record hash database signature default* yang telah dipilih. Gambar 3.35 menunjukkan rancangan *class form view signatures* dan hubungannya dengan *library fuzzy hash* pada *Dashboard*.



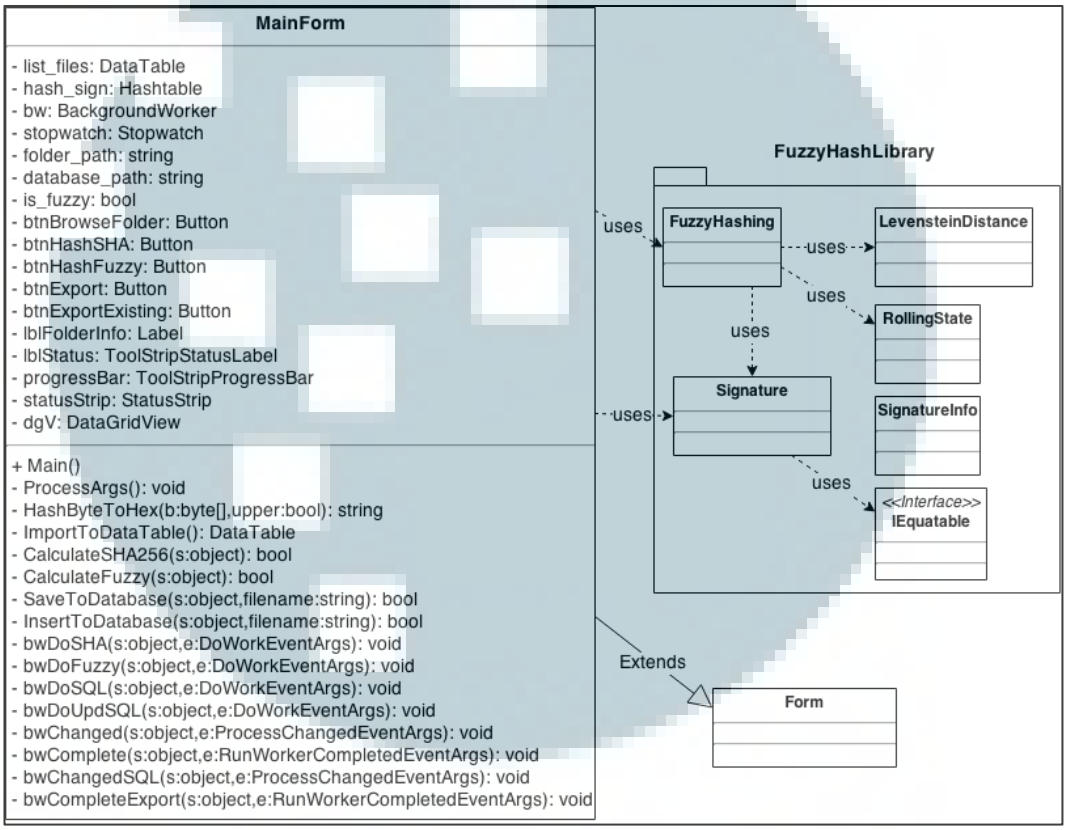
Gambar 3.35. *Class ViewSignaturesForm* pada *Dashboard*

Tabel 3.4 berisi penjelasan beberapa *method* dari *class MainForm* dan *ViewSignaturesForm* di dalam *Dashboard*.

Tabel 3.4. Keterangan beberapa *method* dari *class* dalam *Dashboard*

Nama method	**Keterangan**
`ReadConfiguration()`	Membaca isi *file* config.txt yang berisi infomasi *path Database Signature Builder*, *Malware Detector*, dan *file database signature*.
`WriteConfiguration()`	Menyimpan ke dalam *file* config.txt yang berisi infomasi *path Database Signature Builder*, *Malware Detector*, dan *file database signature*.
`CheckLocation()`	Memeriksa keberadaan *file* sesuai dengan informasi *path* yang tersimpan dalam config.txt
`Detector_DeviceArrived()`	Mendeteksi *removable drive* yang baru terpasang, kemudian memunculkan *message box* apakah *drive* ingin di-*scan* menggunakan *Malware Detector*
`LoadDataTableIntoGridView()`	Menampilkan isi dalam *DataTable* yang berupa daftar *signature* ke dalam *GridView*

Aplikasi *Database Signature Builder* dibangun menggunakan bahasa pemprograman C# .NET 4.0 dengan model *windows-form application* dan *library fuzzy hash* yang telah dibuat pada langkah sebelumnya, digunakan dalam aplikasi ini. Gambar 3.36 menunjukkan rancangan *class form* utama pada *Database Signature Builder*.



Gambar 3.36. *Class MainForm* pada *Database Signature Builder* beserta relasinya dengan *library fuzzy hash*

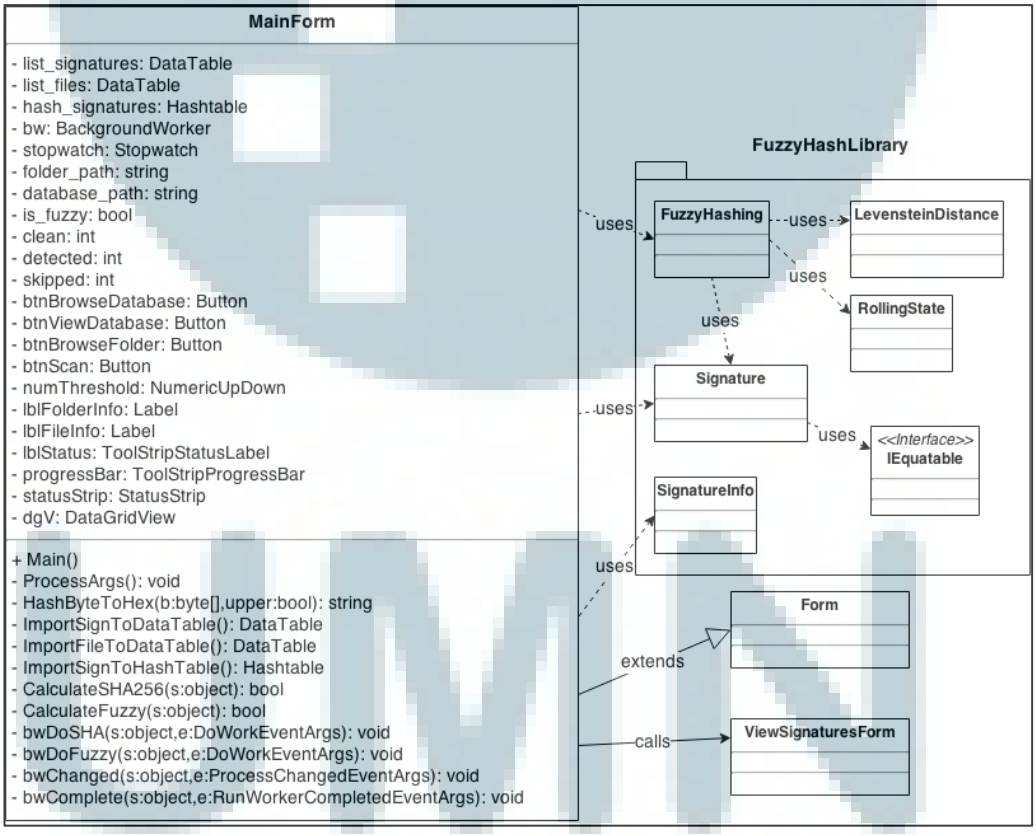
Class yang dibangun dalam aplikasi *Database Signature Builder*, berupa turunan *form* yang tersedia dalam C# ditambah beberapa komponen GUI, *field*, serta *method* untuk membantu proses penghitungan *hash* dan penyimpanannya ke dalam *database*. Tabel 3.5 berisi penjelasan beberapa *method* dari *class* tersebut.

Tabel 3.5. Keterangan beberapa *method* dari *class* dalam *Database Signature Builder*

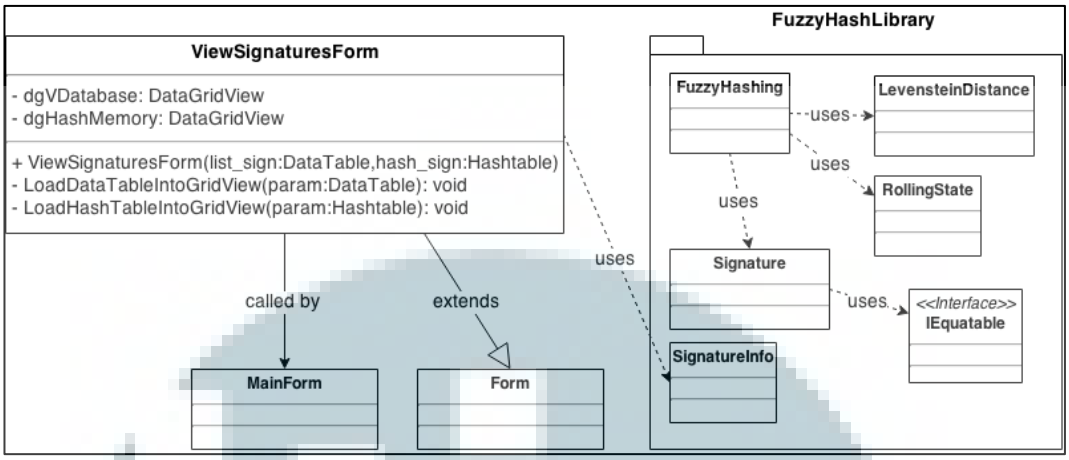
Nama method	**Keterangan**
`ProcessArgs()`	Memproses *argument* atau parameter ketika aplikasi dijalankan, apakah *hash* yang dipilih SHA256 atau *fuzzy*. Berguna ketika membangun *signature* langsung dengan mengeksekusi aplikasi ini dari *context-menu file explorer*.
`HashByteToHex(...)`	Melakukan konversi *byte* hasil *hash* SHA256 (kalkulasi *hash* SHA256 menggunakan *function* yang sudah tersedia dalam bahasa C#) ke dalam *string*.
`ImportToDataTable()`	Menyimpan *list files* ke dalam *DataTable* untuk ditampilkan ke *GridView* setelah *user* melakukan *open/browse folder*.
`CalculateSHA256(...)`	Melakukan penghitungan *hash* SHA256. *Method* ini panggil dari *method background worker* bwDoSHA ketika *user* mengklik *button* btnHashSHA.
`CalculateFuzzy(...)`	Melakukan penghitungan *hash fuzzy*. *Method* ini panggil dari *method background worker* bwDoFuzzy ketika *user* mengklik *button* btnHashFuzzy.
`SaveToDatabase(...)`	Menyimpan *DataTable* setelah *file* dihitung nilai *hash*-nya ke dalam *database* SQLite3 baru. *Method* ini panggil dari *method background worker* bwDoSQL ketika *user* mengklik *button* btnExport.
`InsertToDatabase(...)`	Menyimpan *DataTable* setelah *file* dihitung nilai *hash*-nya ke dalam *database* SQLite3 yang sudah ada. *Method* ini panggil dari *method background worker* bwDoUpdSQL ketika *user* mengklik *button* btnExportExisting.

Aplikasi *Malware Detector* dibangun menggunakan bahasa pemprograman C# .NET 4.0 dengan model *windows-form application* dan *library fuzzy hash* yang telah dibuat, digunakan dalam aplikasi ini. Aplikasi terdiri dari dua buah *form*. Satu *form* utama dan *form view signatures*. *Form* utama digunakan

untuk melakukan *load signature*, *load file-file*, dan *scan file*. *Form* utama terdiri dari beberapa *button*, satu buah *gridview*, dan *status bar*. Sedangkan *form view signatures* yang terdiri dari dua buah *gridview* dan *status bar* digunakan untuk melihat isi *database signature* yang di-*load*. Gambar 3.37 dan 3.38 menunjukkan rancangan *class Malware Detector* dan relasinya dengan *library fuzzy hash*. Pada *form view signatures*, disediakan dua *gridview* untuk melihat isi *signature* yang ada pada *DataTable* (sebelah kiri) dan isi *signature* yang ada pada *Hashtable* (sebelah kanan). Isi data pada *DataTable* sama dengan isi data pada *Hashtable*. *Hashtable* dengan struktur data pasangan *key-value* ini digunakan untuk deteksi *malware*.



Gambar 3.37. *Class MainForm* pada *Malware Detector* beserta relasinya dengan *library fuzzy hash* dan *form ViewSignatures*



Gambar 3.38. *Class ViewSignatures* pada *Malware Detector* beserta relasinya dengan *library fuzzy hash* dan *MainForm*

Tabel 3.6 berisi penjelasan beberapa *method* dari *class MainForm* dan *ViewSignaturesForm* di dalam *Malware Detector*.

Tabel 3.6. Keterangan beberapa *method* dari *class* dalam *Malware Detector*

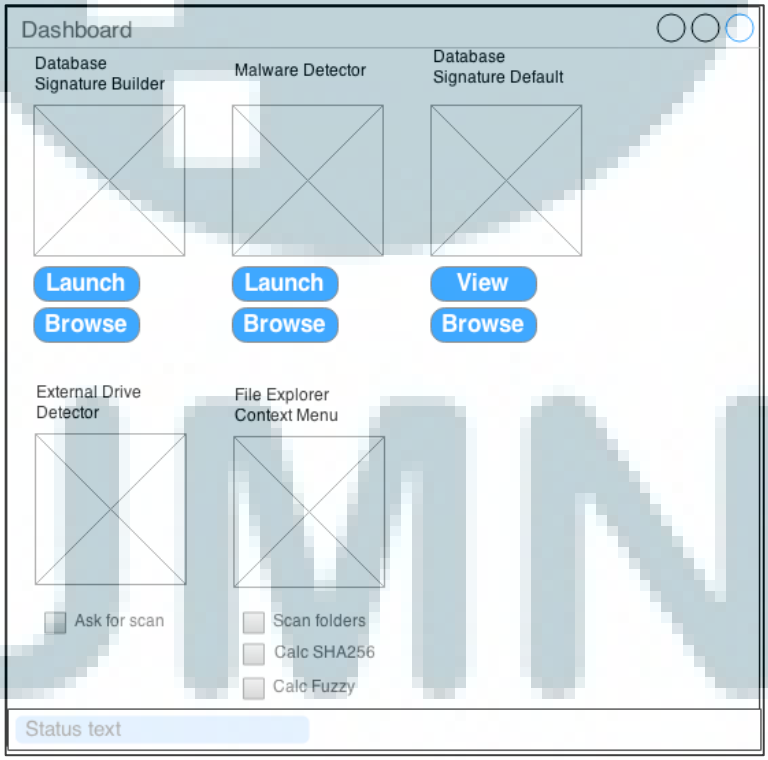
Nama method	**Keterangan**
`ProcessArgs()`	Memproses *argument* atau parameter ketika aplikasi dijalankan untuk menentukan *hash* yang dipilih dan alamat *folder* yang akan di-*scan*. Berguna ketika aplikasi dieksekusi dari *context-menu file explorer* dan *Dashboard*.
`ImportFileToDataTable()`	Menyimpan *list files* ke dalam *DataTable* untuk ditampilkan ke *GridView* setelah *user* melakukan *open/browse folder*.
`ImportSignToDataTable()`	Menyimpan hasil *query* "*select * from signature*" ke dalam *DataTable* setelah *user* melakukan *open database*.
`ImportSignToHashTable()`	Menyimpan isi *DataTable* ke dalam *Hashtable* dengan struktur data pasangan *key-value*. *Hashtable* ini digunakan untuk mempermudah deteksi *malware*. Pendeteksian dilakukan dengan *hash* sebagai *key* untuk mendapatkan informasi *malware* yang dienkapsulasi dalam *object SignatureInfo* sebagai *value*

Tabel 3.6. Keterangan beberapa *method* dari *class* dalam *Malware Detector* (Lanjutan)

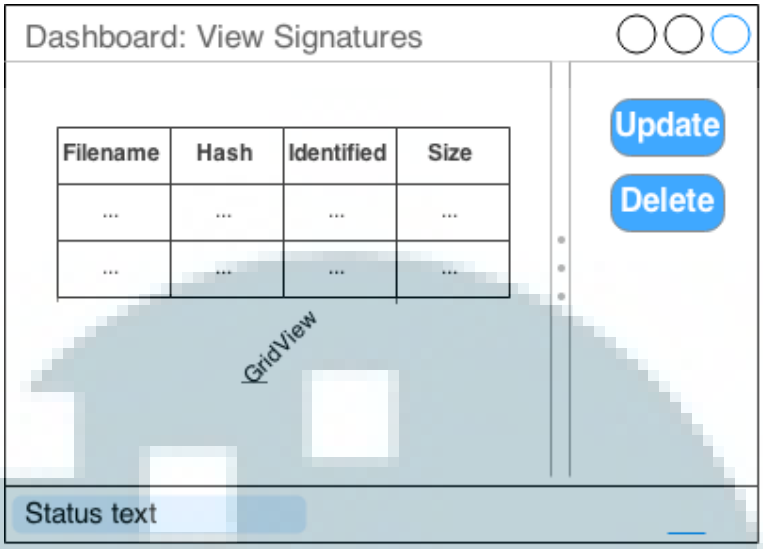
Nama method	Keterangan
`CalculateSHA256(...)`	Melakukan penghitungan *hash* SHA256. *Method* ini dipanggil dari *method background worker* bwDoSHA ketika *user* mengklik *button* btnScan.
`CalculateFuzzy(...)`	Melakukan penghitungan *hash fuzzy*. *Method* ini dipanggil dari *method background worker* bwDoFuzzy ketika *user* mengklik *button* btnScan.

3.6.7 Perancangan Antarmuka

Aplikasi *Dashboard* terdiri dari dua buah *form*. Gambar 3.39 menunjukkan rancangan antarmuka *form* utama dan gambar 3.40 menunjukkan rancangan antarmuka *form view signatures* pada *Dashboard*.

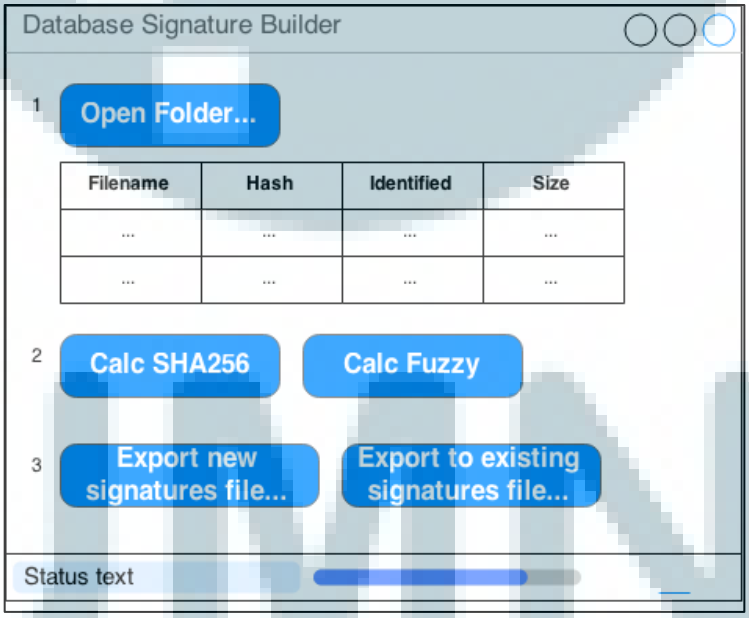


Gambar 3.39. Rancangan antarmuka *form* utama pada *Dashboard*



Gambar 3.40. Rancangan antarmuka *form view signatures* pada *Dashboard*

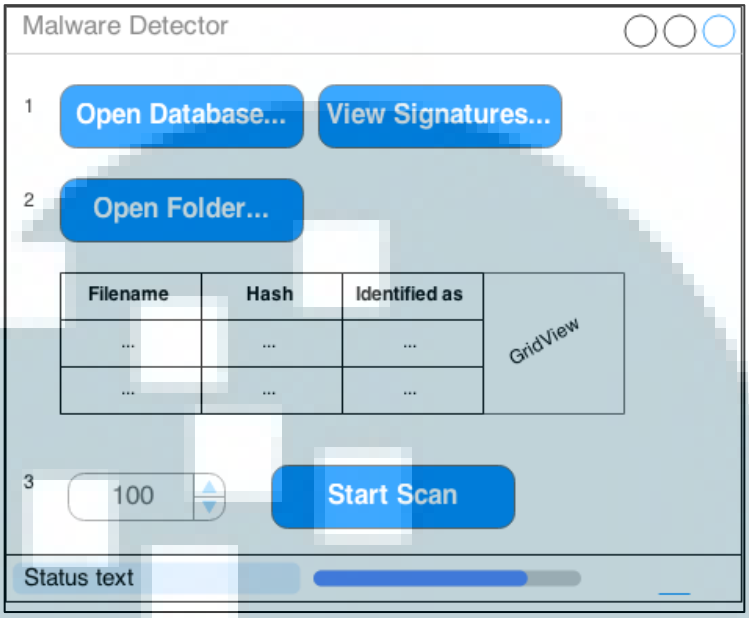
Aplikasi *Database Signature Builder* terdiri dari satu buah *form* yang terdiri dari beberapa *button*, satu buah *gridview*, dan *status bar*. Gambar 3.41 menunjukkan rancangan antarmuka pada *Database Signature Builder*.



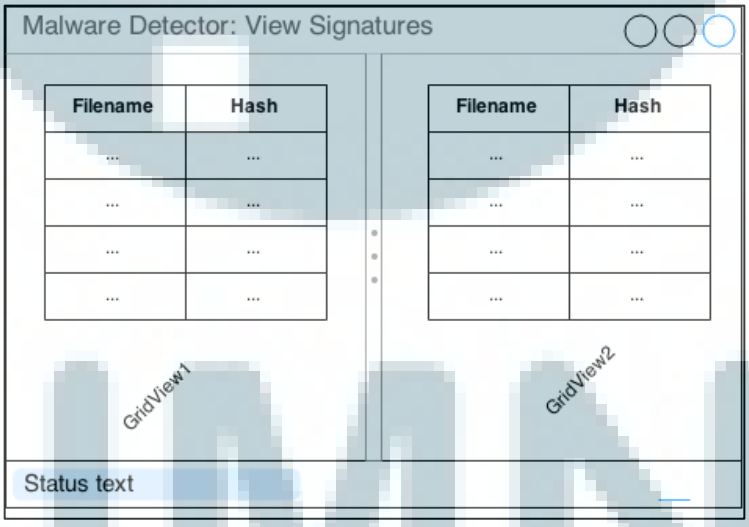
Gambar 3.41. Rancangan antarmuka aplikasi *Database Signature Builder*

Aplikasi *Malware Detector* terdiri dari dua buah *form*. Satu *form* utama dan *form view signatures*. Gambar 3.42 menunjukkan rancangan antarmuka *form*

utama dan gambar 3.43 menunjukkan rancangan antarmuka *form view signatures* pada *Malware Detector*.



Gambar 3.42. Rancangan antarmuka aplikasi *Malware Detector*



Gambar 3.43. Rancangan antarmuka aplikasi *Malware Detector* pada *form View Signatures*