BAB III
PELAKSANAAN KERJA MAGANG

3.1. Kedudukan dan Koordinasi

Dalam pelaksanaan kerja magang di perusahaan Cimrid Pty. Ltd., penulis berperan sebagai *developer* untuk sistem operasi Android. Kerja magang ini dikoordinasi oleh Franco Octavianus selaku *Product Development Manager* di Indonesia. Penulis diberi pelatihan dan pengetahuan mengenai sistem operasi Android oleh Bapak Michael Gunawan pada saat berada di Erina. Beliau memberikan beberapa contoh dan pelatihan bagaimana teknik pemrograman di dalam sistem Android berikut *source code* dan *database* yang perlu dikembangkan dengan penambahan beberapa fitur baru yang akan diimplementasikan dalam program tersebut.

3.2. Tugas yang Dilakukan

Pengembangan aplikasi *mobile* Invoice2go dilakukan oleh penulis dalam kerja magang selama 2 bulan. Beberapa tugas yang diberikan kepada penulis dapat diuraikan secara umum sebagai berikut:

a. Mempelajari konsep dasar sistem operasi android untuk mengetahui bagaimana *lifecycle* dari *activity*, *thread*, *fragmen*t *yang berjalan* di dalamnya.
b. Mempelajari struktur dari program aplikasi *mobile* Invoice2go yang terintegrasi dengan beberapa aplikasi lain seperti Sign2go, Receipts2go, dsb.
c. Menambahkan fitur universal search pada aplikasi tersebut untuk mencari data klien, dokomen, receipt hanya dengan memasukkan kata kunci tertentu pada 1 search bar
d. Optimalisasi tampilan dan pembuatan *custom listView* untuk *universal search* sehingga tampilan yang dihasilkan mirip dengan tampilan *table view* pada iOS
e. Optimalisasi peforma pencarian dengan menggunakan metode *Full-Text Search*
f. Menguji coba fitur tersebut apakah pencarian dengan jumlah data yang besar sudah cukup cepat, data yang dikeluarkan sesuai dengan kebutuhan, serta pengujian tampilan apakah sudah sesuai dengan yang ada pada iOS.

Setiap harinya selama di Erina, penulis memberikan laporan kepada pembina lapangan sejauh mana proses tersebut selesai. Kemudian, beliau akan membantu dengan memberikan solusi dan menentukan langkah selanjutnya. Kepada pemimpin perusahaan penulis memberikan laporan pada setiap hari Jumat kecuali pada tanggal 21 Juli 2013 karena pemimpin perusahaan sedang berada di keluar kota.

3.3. Uraian Pelaksanaan Kerja Magang

Pelaksanaan kerja magang yang dilakukan penulis ini dapat diuraikan menjadi tiga bagian, yaitu proses pelaksanaan, kendala yang ditemukan, serta solusi atas kendala yang ditemukan.

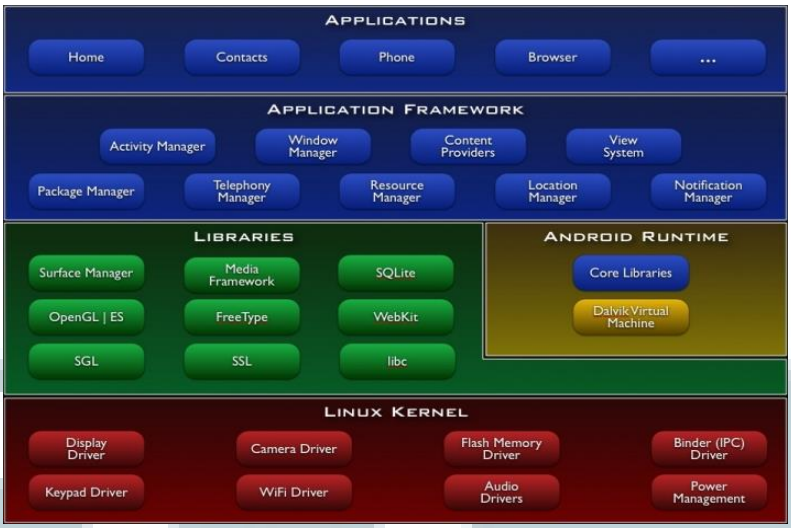
3.3.1 Proses Pelaksanaan

Ada beberapa hal yang dilakukan pada saat melaksanakan kerja magang, dimulai dari mempelajari konsep dasar sistem operasi Android seperti *lifecycle activity, fragment,* dan *custom view*, mempelajari struktur tabel yang ada di dalam aplikasi Invoice2go, serta struktur program dalam aplikasi tersebut. Setelah mengetahui bagaimana program yang ada berjalan kemudian mahasiswa memulai dengan membuat desain, evaluasi dan perbaikan desain, dan kemudian diimplementasikan dalam kode serta digabungkan dengan program yang sudah ada.

3.3.1.1 Konsep Dasar Sistem Operasi Android

Android adalah suatu sistem operasi berbasis Linux. Didesain secara khusus untuk perangkat mobile, sistem ini dihadirkan dengan berbagai fasilitas seperti *background process*, berbagai macam *user interface*, akses ke *system file*,dsb. Sistem ini memiliki keunggulan yaitu terdiri dari beberapa komponen dan dapat menggunakan komponen dari aplikasi lain. Android dikembangkan oleh Google yang bekerja sama dengan Open Handset Alliance.

Secara umum, arsitektur dari sistem operasi Android terbagi menjadi:



Gambar 3.1 Arsitektur Sistem Operasi Android (Kobayashi, 2011)

Dari arsitektur tersebut, seorang *developer* biasanya tidak perlu untuk mengakses hingga ke *libraries* dan Android *Runtime* ataupun hingga Kernel untuk membangun suatu aplikasi baru. Namun, *developer* hanya perlu mengakses pada 2 *layer* di atas yaitu *Applications* dan *Application Framework* saja. Hal ini dikarenakan *libraries* dan *runtime* sudah terenkapsulasi di dalam *Application Framework* secara umum.

Sistem operasi Android juga memberikan fasilitas kepada aplikasi di dalamnya untuk menggunakan komponen dari aplikasi lainnya. Hal ini mengacu pada konsep *task.* Sebagai contoh kita dapat menggunakan kamera untuk mengambil foto dan hasil foto tersebut langsung dapat kita terima dalam aplikasi kita dan dapat kita proses lebih lanjut melalui aplikasi kita.(Lars Vogel, 2013)

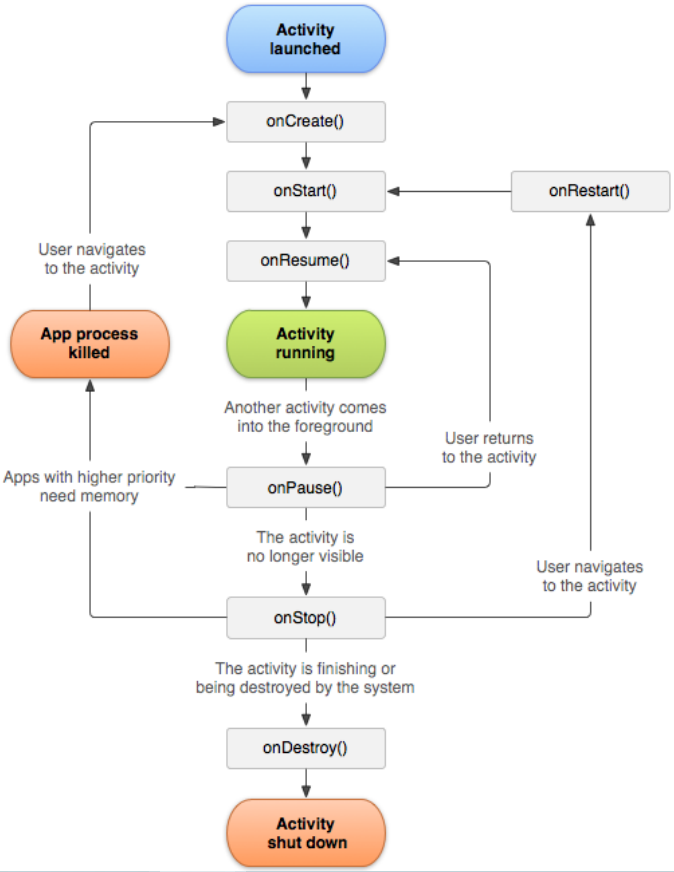
Hal lain yang juga ditawarkan dalam sistem operasi Android adalah Google Play. Di dalam Google Play ini *developer* diberikan kesempatan untuk menawarkan aplikasinya kepada masyarakat luas sehingga masyarakat luas dapat mendownload dan menggunakannya. Bila pembuat aplikasi mengunggah versi terbaru dan memilih untuk mempublikasikannya maka secara otomatis para pengguna aplikasi yang telah membeli atau mengunduh aplikasi tersebut dari Google Play akan diberikan notifikasi tentang *update* terbaru pada aplikasi tersebut (Android Developer Team, 2013).

Google Play juga dapat mengidentifikasi kompaitibilitas aplikasi dengan sistem Android pengguna. Bila versi sistem operasi Android pengguna tidak mendukung dengan aplikasi maka pengguna akan diberikan peringatan tentang hal tersebut sehingga bila tidak kompaitibel maka tidak dapat digunakan dan pengguna tidak dirugikan bila aplikasi tersebut berbayar.

Dalam pengembangan aplikasi untuk *platform* Android dibutuhkan sebuah *tool* yang disebut *Android Software Development Kit (*Android SDK) yang dapat diperoleh secara gratis. Aplikasi ini dapat digunakan untuk membuat, kompilasi, dan mengemas aplikasi Android. Bahasa pemrograman yang digunakan dalam aplikasi ini adalah Java. Di dalam aplikasi ini terdapat Android Debug Bridge(Adb) yang merupakan suatu alat untuk mengirim dan menjalankan aplikasi yang telah dibuat ke dalam peralatan Android ataupun *virtual device* yang telah disediakan. Tersedia berbagai macam pilihan *virtual device* untuk menjalankan aplikasi ini sehingga kita dapat melihat secara umum tampilan aplikasi kita di berbagai perangkat. Namun, untuk menjalankan *virtual device* tersebut, diperlukan spesifikasi komputer yang tinggi seperti membutuhkan *memory* minimal 4 GB.

A. Activity

Activity merupakan representasi visual dari aplikasi. Di dalam satu aplikasi dapat memiliki beberapa *activity*. Namun, hanya 1 *activity* yang dapat aktif dalam satu waktu saja. Bila *activity* berpindah dari satu ke yang lainnya maka *activity* yang aktif akan dihancurkan terlebih dahulu. Setelah itu *activity* lain baru di-*create*. Untuk mengembalikan *activity* yang sebelumnya telah dihancurkan, kita perlu menyimpannya di dalam suatu *container* sehingga pada saat dibuat ulang seolah-olah dapat me-*resume* yang telah dihancurkan sebelumnya.

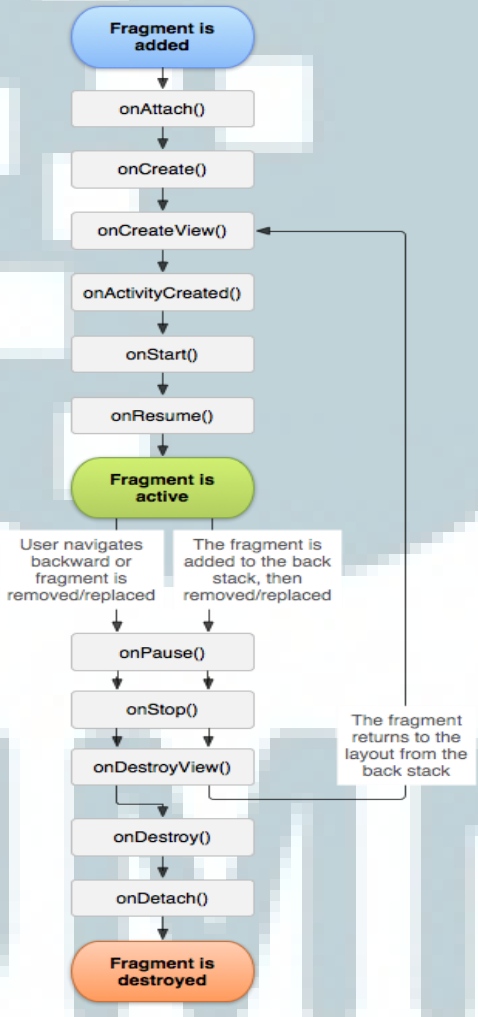


Gambar 3.2 Activity Lifecycle (Android Developer Team, 2013)

Salah satu hal yang perlu diperhatikan dalam pembuatan *activity* adalah bagaimana kita menyimpan kondisi terakhir dari *activity*. Bila kita merubah orientasi dari peralatan Android maka tidak secara otomatis *activity* menyimpan kondisi terakhirnya. Bila hal itu terjadi maka *activity* hanya akan membuat ulang secara keseluruhan sehingga perubahan yang kita lakukan tidak tersimpan. Untuk itu disediakan sebuah fitur untuk menyimpan kondisi tersebut yaitu *onSaveInstanceState(Bundle)*. Dengan fasilitas ini kita dapat menyimpan kondisi terakhir dan bila terjadi perubahan orientasi atau semacamnya Android akan mengembalikan kondisi terakhir yang tersimpan dan pada saat masuk ke *onCreate()*, sistem akan melihat data tersimpan dahulu dan mengembalikannya bila ada, bila tidak maka sistem akan membuat ulang secara keseluruhan (Lars Vogel, 2013).

B. Fragment

Fragment merupakan komponen dari *activity* yang mengenkapsulasi code aplikasi sehingga lebih mudah digunakan kembali dan mampu mengakomodir berbagai ukuran layar. Di dalam satu *activity* bisa memiliki beberapa *fragment* yang dapat aktif secara bersama-sama. *Fragment* dapat berperan seolah-olah seperti *activity* dan memiliki beberapa keuntungan karena *fragment* tidak langsung dihancurkan ketika selesai digunakan dan *fragment* dapat dipanggil dengan berbagai ukuran di depan *activity* ataupun *fragment* lain sesuai dengan kebutuhan(Lars Vogel,2013).



Gambar 3.3 Fragment Lifecycle (Android Developer Team, 2013)

Dalam *fragment*, secara umum kita perlu mendeklarasikan 3 metode yaitu: *onCreate*(), *onCreateView()*, dan *onPause()*. Perbedaan antara *onCreate()* dengan *onCreateView()* adalah dimana untuk inisialisasi diletakkan pada *onCreate()* sedangkan untuk membangun dan menampilkan *User Interface* diletakkan pada *onCreateView()*. Dalam *onCreateView*() akan mengembalikan nilai dari *view*. Sedangkan dalam *onPause*() kita menyimpan perubahan dan kondisi dari *fragment* yang ada sehingga ketika kita kembali ke *fragment* tersebut kita dapat mengembalikan kondisi terakhirnya.

Keunggulan lain dari penggunaan *fragment* adalah adanya fasilitas *Fragment Manager*. *Fragment Manager* memungkinkan user untuk melakukan transaksi *fragment*. Melalui *Fragment Manager* user dapat membuat perubahan menggunakan beberapa metode seperti *add(), remove(), dan replace().* Untuk mengeksekusi perubahan digunakan metode *commit*(), maka seluruh perubahan akan diterapkan secara bersama-sama. Sebelum dilakukan *commit*(), bila kita membuang salah satu *fragment* ada 2 pilihan yang dapat dipilih yaitu:

1. Bila kita ingin menghancurkan *fragment* tanpa perlu menyimpan kondisi sebelum dihancurkan, maka kita dapat langsung memanggil *commit*()
2. Bila kita ingin menyimpan kondisi terakhir *fragment* kita perlu menambahkan *addToBackStack()* sebelum di-commit sehingga kondisi terakhir *fragment* akan tersimpan dan dapat dipanggil kembali.

C. View

View adalah komponen di dalam Android yang digunakan sebagai *user interface*. Di dalam Android sudah tersedia *built-in view* pada android.view.*View class* seperti *edittext, button, textfield,*dsb. Dari *built-in view* tersebut kita dapat menggunakannya langsung dan dapat mengubah atribut di dalamnya sesuai kebutuhan. Namun, bila kita ingin merubah *behavior*-nya kita perlu membuat *custom v*iew yang merupakan suatu *class* dimana *class* tersebut menambahkan ataupun mengganti keseluruhan *behavior* dari *view* standar dengan membuat desainnya serta menggunakan *adapter* untuk mengolah bagaimana *view* tersebut akan menampilkan hasil di layar.

Penggunaan *custom view* dilakukan untuk tujuan khusus tertentu dari aplikasi yang dibuat. Hal ini dilakukan bila *built-in view* yang telah tersedia tidak dapat mengakomodir kebutuhan *developer* untuk menghasilkan tampilan yang diinginkan. Pembuatan *custom view* yang membuat suatu *class* baru dengan melakukan *extend* terhadap *view* standar seperti *textView* ataupun *listView* disertai dengan suatu xml dimana xml *file* ini akan memanggil *class* yang telah dibuat tadi sehingga dapat memunculkan tampilan sesuai yang dibutuhkan dan dibuat sebelumnya.

D. Adapter

Adapter merupakan sebuah komponen yang bekerja sebagai jembatan antara *view* dengan data yang akan ditampilkan di dalam *view*. *Adapter* menyediakan akses kepada data yang akan ditampilkan dan bagaimana cara menampilkannya. *Adapter* juga bertanggung jawab untuk membuat *view* untuk tiap *item* yang ada di dalam *dataset*. Ada beberapa *adapter* yang sudah *built-in* di dalamnya seperti: *simpleCursorAdapter* yang digunakan untuk membuat *view* pada *listView*. Namun, *adapter built-in* ini memiliki keterbatasan dalam menampilkan *view*. *Adapter* ini tidak dapat sembarangan digunakan untuk menampilkan *view* (Lars Vogel, 2013).

Seperti halnya *view*, *adapter* juga dapat dimodifikasi. Hal ini dilakukan agar dapat memenuhi keinginan pengembang aplikasi untuk menampilkan sesuatu secara berbeda dari yang standar. Pengembang tidak hanya dapat melakukan perubahan pada sisi *view* saja tetapi juga dapat merubah *adapter*. Untuk merubahnya kita perlu membuat suatu *class adapter* baru yang melakukan *extend* dari *adapter* tertentu seperti *simplecursor adapter* ataupun *baseAdapter* bila hanya memerlukan basis *adapter* yang akan dikembangkan lebih lanjut. Beberapa *method* yang dimiliki dapat dirubah sesuai dengan kebutuhan dengan melakukan *override* dan menambahkan ataupun mengganti *behavior*-nya.

3.3.1.2 SQLite

SQLite merupakan *database open source* yang telah terintegrasi dengan sistem operasi Android. SQLite memiliki kemampuan mirip dengan MySQL untuk menyelesaikan berbagai *query*. Beberapa fitur utama atau krusial dihadirkan di dalamnya.Namun, ada beberapa fitur yang dihilangkan di dalamnya seperti *foreign key*, *full outer join*, *right join*,dsb. sehingga mampu membuat SQLite berjalan hanya menggunakan *memory* sebesar 250KB. Walaupun beberapa fitur dihilangkan hal itu tidak membuat fungsionalitasnya berbeda dengan MySQL dikarenakan *user* dapat menggunakan berbagai alternatif untuk mendapatkan hasil serupa. Dilihat dari *syntax* secara umum tidak ada perbedaan antar keduanya (SQLite Developer Team,2013).

Dalam penggunaanya untuk menghemat *memory*, pencarian secara umum dilakukan menggunakan *query* berdasarkan *keyword* tertentu sehingga diperlukan *query* "*LIKE*" yang menyebabkan penurunan peforma secara signifikan. Namun, hal ini berdampak pada *database* yang berukuran besar terhadap peforma dari aplikasi. Penggunaan search dari *query* berdampak pada peforma. Namun, dapat menghemat penggunaan *memory* aplikasi yang terbatas hanya 16MB. Untuk itu digunakan salah satu fitur yang dimiliki oleh SQLite untuk tetap menghemat *memory* tetapi tetap memberikan hasil yang cepat untuk melakukan pencarian yaitu *Full Text Search* (FTS).

A. Full Text Search (FTS)

Full Text Search merupakan salah satu fitur yang tersedia di dalam SQLite yang menggunakan *virtual table* dalam proses pencariannya. Alasan utama digunakan fitur ini adalah mempercepat waktu pencarian *record* dari *database*. Penggunaan tabel biasa di dalam search yang menggunakan "*LIKE*" memakan waktu cukup lama bila berada dalam *database* yang berukuran besar. Hal ini tentu sangat mengganggu *user* yang ingin melakukan pencarian bila hal itu terjadi.

Sebagai contoh perbandingan pencarian dengan 517430 *record* dapat terlihat perbedaan signifikan. Dengan menggunakan tabel biasa dan *search term* "like"

diperlukan waktu 22,5 detik untuk melakukan proses ini. Sedangkan, bila menggunakan FTS dengan *search term* "*MATCH*" hanya diperlukan waktu 0,03 detik saja. Perbedaan yang sedemikian jauh tersebut tentu memberikan keuntungan sendiri bagi FTS untuk melakukan pencarian.

Dalam dokumen yang dipublikasikan dalam situs SQLite terdapat 2 versi FTS yang ada yaitu FTS3 dan FTS4. Perbedaan keduanya hanyalah terpaut sedikit yaitu

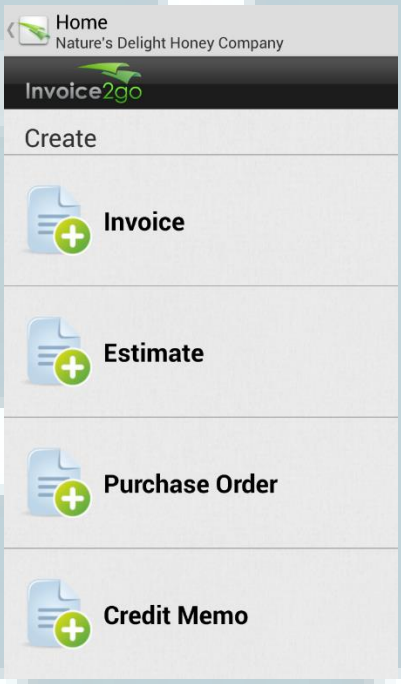
1. Optimalisasi peforma *query* pada FTS 4 sehingga dapat mempercepat proses pencarian dibandingkan menggunakan FTS 3
2. Memiliki beberapa fitur tambahan pada fungsi *matchinfo*() dalam FTS4
3. FTS4 memerlukan media penyimpanan lebih besar dibanding FTS3 karena ada beberapa penambahan fitur dan optimalisasi peforma
4. FTS4 memiliki kemampuan untuk kompresi data sehingga dapat mengurangi penggunaan media penyimapanan dan I/O

3.3.1.3 Aplikasi Invoice2Go

Aplikasi Invoice2go merupakan salah satu aplikasi *mobile invoicing* yang hadir di berbagai platform seperti PC, Android, dan iOS. Secara umum aplikasi ini hadir dalam 2 versi yaitu Plus dan Full. Perbedaan keduanya terletak pada dukungan terhadap penyimpanan *database* di dalam *cloud.* Pada versi plus pengguna diberikan aplikasi secara gratis. Namun, diberikan keterbatasan dengan jumlah dokumen yang dapat disimpan di *cloud*. Kemudian, bila pengguna ingin menambah kapasitas penyimpanan dihitung berdasarkan jumlah dokumen yang ingin disimpan. Tersedia beberapa pilihan paket di dalamnya sesuai kebutuhan pelanggan. Sementara itu, pada versi full aplikasi tidak diberikan secara gratis tetapi pengguna diminta membeli aplikasi tersebut pada Google Play ataupun App Store. Pada versi ini pengguna tidak diberikan fungsi sinkronisasi dengan *cloud* tetapi pengguna dapat menyimpan *database*-nya dengan jumlah tidak terbatas di dalam *smartphone* mereka (Invoice2go Team, 2013).

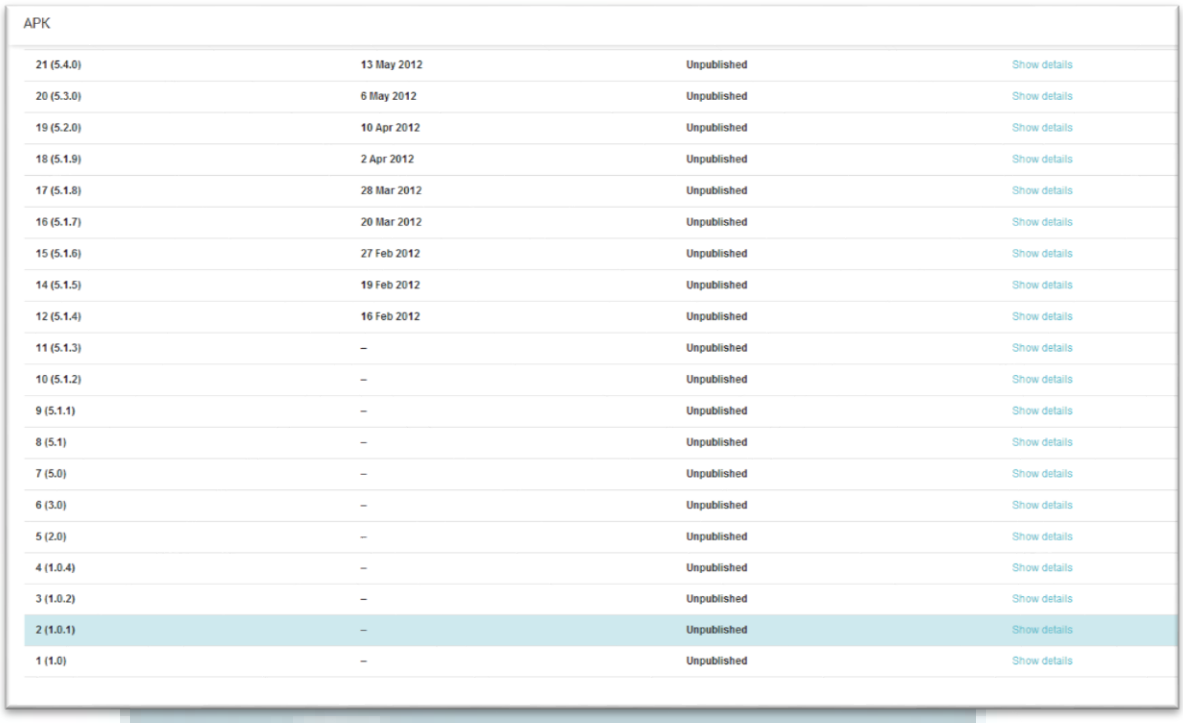
Dalam aplikasi Invoice2go ada beberapa aplikasi lain yang terintegrasi di dalamnya yaitu Apps2go. Apps2go terdiri dari beberapa aplikasi seperti: Sign2go, Receipts2go, Statements2go, Calendar2go, Maps2go, dan Scan2go. Aplikasi ini

merupakan aplikasi tambahan yang dapat dibeli satu per satu ataupun secara keseluruhan atau dipaketkan. Walaupun aplikasi ini dapat dibeli secara terpisah, aplikasi tersebut tetap harus berjalan bersama dengan aplikasi utama yaitu Invoice2go. Kegunaannya pun bervariatif dari untuk menandatangani dokumen, membuat *receipt*, dsb. Aplikasi – aplikasi tersebut menggunakan *database* yang satu dan sama dengan aplikasi utama sehingga bila melakukan *backup* ataupun *restore* pengguna tidak direpotkan.



Gambar 3.4 Tampilan awal aplikasi Invoice2go

Dalam perkembangannya aplikasi Invoice2go untuk Android telah mengalami perubahan dan pembaharuan. Penambahan fitur seiring berjalannya waktu dilakukan untuk membantu pengguna mendapatkan pelayanan tambahan dengan berbagai *update* yang dilakukan. Konsep awal pun tidak berubah yaitu *"Simple and easy to use"*. Pada versi 1.0 untuk Android aplikasi ini hanya berukuran 5.29 MB dengan 3 *screen layout* saja. Versi terakhir yang digunakan oleh penulis untuk dikembangkan adalah versi 7.16

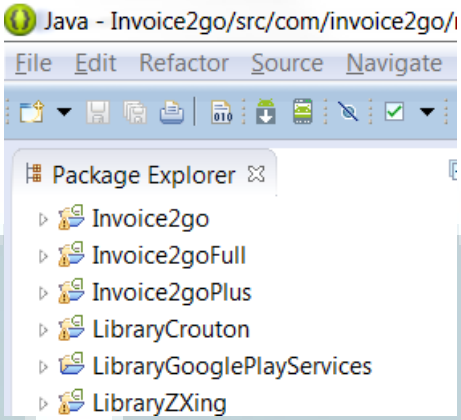


APK

21 (5.4.0)	13 May 2012	Unpublished	Show details
20 (5.3.0)	6 May 2012	Unpublished	Show details
19 (5.2.0)	10 Apr 2012	Unpublished	Show details
18 (5.1.9)	2 Apr 2012	Unpublished	Show details
17 (5.1.8)	28 Mar 2012	Unpublished	Show details
16 (5.1.7)	20 Mar 2012	Unpublished	Show details
15 (5.1.6)	27 Feb 2012	Unpublished	Show details
14 (5.1.5)	19 Feb 2012	Unpublished	Show details
12 (5.1.4)	16 Feb 2012	Unpublished	Show details
11 (5.1.3)	–	Unpublished	Show details
10 (5.1.2)	–	Unpublished	Show details
9 (5.1.1)	–	Unpublished	Show details
8 (5.1)	–	Unpublished	Show details
7 (5.0)	–	Unpublished	Show details
6 (3.0)	–	Unpublished	Show details
5 (2.0)	–	Unpublished	Show details
4 (1.0.4)	–	Unpublished	Show details
3 (1.0.2)	–	Unpublished	Show details
2 (1.0.1)	–	Unpublished	Show details
1 (1.0)	–	Unpublished	Show details

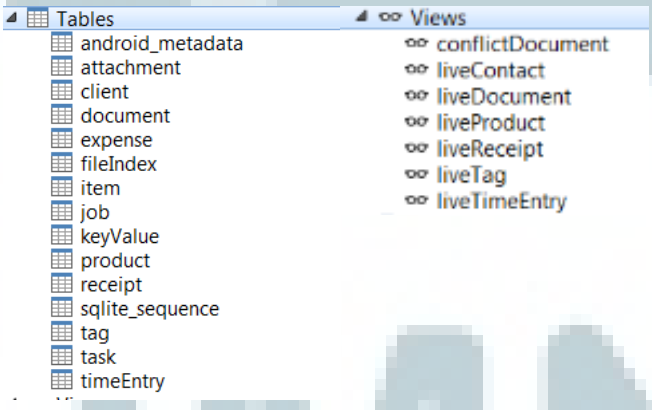
Gambar 3.5 APK yang telah di-release pada Play Store (Google Play Store, 2013)

Dengan perbedaan fitur utama yaitu fungsi *cloud* tersebut digunakan pendekatan yang sama terhadap kedua versi tersebut. Untuk itu digunakan basis yang sama terhadap keduanya. Kedua versi tersebut akan mengacu pada *code* utama yang satu dan sama. Sehingga, bila ada penambahan fitur kita tidak perlu merubah keduanya masing-masing, melainkan kita hanya perlu merubah *code* utama yang dijadikan basisnya. Dalam aplikasi ini digunakan *code* utama yang disebut Invoice2go dengan 2 *code* lain yang mengacu padanya yaitu Invoice2goFull dan Invoice2goPlus serta 3 *library* lainnya untuk menjalankan aplikasi ini.



Gambar 3.6 Tampilan Package Explorer Invoice2go pada Eclipse

Dalam pengembangan aplikasi Invoice2Go versi Android digunakan beberapa program lain di luar Eclipse yang telah terintegrasi dengan Android SDK yaitu Navicat dan TortoiseHG. Penggunaan Navicat dalam pengembangan aplikasi ini adalah untuk melihat struktur tabel, memodifikasi isi *database,* dan membandingkan *virtual tabel* dan tabel biasa.



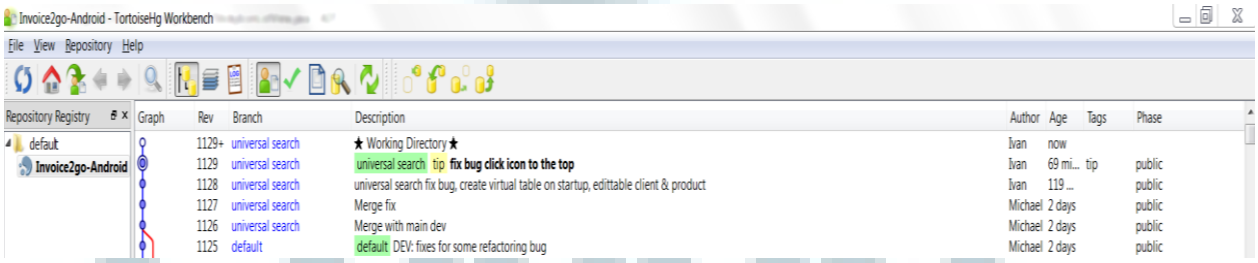
Gambar 3.7 Tampilan Navicat untuk database Invoice2go 25 July 2013

Di dalam aplikasi ini ada beberapa *tables* dan *views*. Perbedaan fungsi dalam implementasinya merupakan alasan utama digunakan 2 fitur ini. Tabel merupakan sebuah *database* yang aktual yang menyimpan berbagai data di dalamnya. Sedangkan, *views* merupakan sebuah tabel imajiner yang dikonstruksikan berdasarkan tabel. Sebagai contoh *database-Jul-25-2013.sqlite* untuk aplikasi Invoice2Go memiliki Tabel Document dan

Views liveDocument. Views liveDocument di dalam *view* akan langsung diperbaharui bila ada perubahan pada document seperti dihapus ataupun dimodifikasi tetapi hal itu tidak terjadi pada tabel document. Hal ini dimaksudkan supaya bila terjadi kesalahan penghapusan yang tidak disengaja user masih dapat me-*recovery* data dari tabel tersebut.

Penggunaan lain dari *views* adalah membuat suatu tabel imajiner baru untuk menampilkan hasil penggabungan beberapa tabel aktual. *Views* dapat digunakan seolah-olah kita membuat tabel baru dengan *field* gabungan antara beberapa tabel dan menampilkannya kepada user.

Penggunaan TortoiseHG dalam aplikasi ini dimaksudkan untuk mengunggah *repository* ke *cloud.* Keuntungan penggunaannya yaitu bila terjadi kesalahan yang fatal pada saat pengembangan perangkat lunak, kita dapat menarik kembali versi sebelumnya. Hal ini sangat membantu kita bila tidak dapat mengembalikan seperti sedia kala. Keuntungan lain dari penggunaan TortoiseHG adalah kita dapat melakukan penggabungan pekerjaan beberapa orang sekaligus. Dalam pengembangan aplikasi Invoice2go, aplikasi dikembangkan oleh beberapa pengembang. Pembagian tugas pun diberikan secara berbeda antara 1 pengembang dengan yang lainnya. Hal ini tentu sangat membantu bila kita ingin menggabungkan hasil pekerjaan pengembang yang 1 dengan yang lainnya.



Gambar 3.8 Tampilan TortoiseHG yang digunakan

3.3.1.4 Penambahan fitur universal search

Penambahan fitur *universal search* ini dilakukan pertama kali pada Invoice2go versi 7.15. Proses penambahan fitur ini dimulai dengan mempelajari dasar-dasar dari aplikasi ini. Dikarenakan ada banyak sekali *class* dan berbagai macam *layout* serta *adapter* ditemukan berbagai halangan untuk membedakan satu *class* dengan yang lain sehingga komunikasi menjadi kunci awal bagi penulis untuk mengetahui *fragment* atau aktivitas mana yang perlu dirubah atau dimodifikasi.

Pada awalnya penulis diberikan *source code* keseluruhan dari aplikasi Invoice2go dengan menarik *source code* tersebut dari *repository* yang telah ada menggunakan TortoiseHG. Pada saat kerja magang dilakukan di Erina, *repository* masih bersifat lokal. Selain itu penulis juga diberikan contoh *database* dari aplikasi ini. Dari *database* yang diberikan penulis diminta untuk membuat desain terlebih dahulu dan mengumpulakan desain tersebut. Kemudian, desain tersebut akan dievaluasi terlebih dahulu. Bila desain tersebut sudah memenuhi syarat, dimulailah proses *coding.*

Dalam *universal search*, penulis diminta untuk melakukan pencarian dari tabel yang ada yaitu: document, invoice, product, client, dan timeEntry. Untuk Tabel Document diperlukan pencarian lebih mendalam yaitu dengan melihat konteks dari dalam Tabel Document tersebut apakah ada product ataupun attachment yang ada di dalamnya. Sehingga diperlukan pencarian dengan penggabungan 3 tabel yaitu: Document, Attachment, dan Item dengan kesamaan DocumentID sebagai kunci pembandingnya.

Pertama kali, penulis memikirkan untuk mengimplementasikan metode sebelumnya yang telah digunakan untuk pencarian. Metode yang dimaksud adalah menggunakan *filtering.* Di dalam metode ini penulis akan menarik keseluruhan isi *database* dan disimpan di dalam *array list*. Kemudian, dari *array list* yang berisi keseluruhan isi *database* digunakan metode *filtering* untuk melakukan pencarian. Dalam hal ini penulis menggunakan suatu *database* baru untuk mengurangi kerumitan terlebih dahulu.

Metode *filtering* disini tidak menimbulkan masalah berarti dikarenakan pencarian di dalam *array list* jauh lebih cepat dibandingkan melakukan pencarian dengan *query*.

Namun, metode ini mengalami permasalahan pada saat pengambilan data bila *database* memiliki lebih dari 1000 *record*. Diperlukan waktu *loading* lebih dari 1,5 detik bila *database* memiliki *record* sekitar 1500 *record.* Waktu ini akan bertambah pada saat jumlah *record* semakin banyak.. Penulis mencoba mengakali waktu penarikan data tersebut menggunakan fasilitas *thread* di Android, tetapi *thread* tersebut hanya membuat aplikasi dapat bergerak dan penarikan dilakukan di *background process*. Namun, waktu *loading* tidak dapat dikurangi bila menggunakan metode ini. Hal lain yang dikhawatirkan adalah keterbatasan *memory* tiap aplikasi yang digunakan hanya sebatas 16MB. Tentu hal ini menjadi masalah bila *database* berukuran besar yang di buka dan akan masuk ke dalam *memory* semuanya. Hal ini dapat menyebabkan aplikasi menjadi *not responding* dan harus dihentikan. Atas alasan di atas metode ini tidak dapat digunakan.

Teknik kedua yang dicoba oleh penulis adalah menggunakan *query* biasa untuk menarik sebagian data saja. Hal ini dilakukan untuk menghindari penggunaan *memory* yang berlebihan pada *device.* Teknik *filtering* disini tidak diperlukan karena data hasil keluaran sudah berupa data yang sesuai dengan kata kunci yang dimasukkan oleh user. Untuk *database* berukuran kecil tidak terlihat bahwa pencarian menggunakan metode ini lambat tetapi bila *database* diperbesar ukurannya dan penggunaan *search term "LIKE"* hal ini menjadi permasalahan yang patut diperhatikan.

Penggunaan *search term "LIKE"* dengan banyak *field* dan *record* membuat penurunan peforma yang signifikan dibandingkan penggunaan "=". Hal ini dikarenakan "LIKE" akan membandingkan satu per satu huruf yang ada di dalam setiap kata dalam kalimat. Pada saat menggunakan *database* dengan 2000 *record* hal ini menjadi masalah dikarenakan diperlukan waktu lebih dari 2 detik untuk mengambil data dari Tabel Dokumen. Data tersebut digabungkan dengan 3 tabel lainnya menggunakan fungsi "*JOIN*". Hal ini dikarenakan setelah menggabungkan keempat tabel tersebut ada 23 *field* yang perlu dibandingkan isinya satu per satu.

Dua teknik tersebut yang telah dicoba digunakan tersebut ternyata tidak memenuhi ekspektasi awal. Pencarian masih tergolong lambat dan dapat mengganggu tingkat kepuasan konsumen. Menanggapi hal tersebut, pemilik perusahaan yaitu Chris Strode mengusulkan penggunaan *Full Text Search* (FTS). Hal ini dimaksudkan untuk

menghindari penggunaan “LIKE” yang membuat pencarian menjadi lebih lambat dibandingkan menggunakan “=” yang hanya dapat mencocokkan kata kunci dengan isi data bila keduanya benar-benar sama.

Untuk membuat pencarian mengeluarkan hasil yang sama, desain yang dibuat di kedua platform Android dan iOS dibandingkan terlebih dahulu sebelum memulai perubahan menggunakan metode FTS. Setelah kedua desain tersebut seragam dilakukan implementasi kembali dengan menggunakan metode tersebut.

Teknik FTS yang telah disepakati tersebut memiliki 2 versi yaitu versi 3 dan 4. Perbedaan keduanya ada pada optimalisasi peforma. Untuk memberikan hasil yang maksimal tetapi dapat mengakomodir semua pengguna digunakan dua kondisi berdasarkan versi Android yang digunakan pengguna. Versi 4 hadir pada Android versi 3.0 ke atas sehingga bila pengguna menggunakannya maka pencarian di dalam *device* mereka akan menggunakan FTS4 yang membuat pencarian menjadi lebih baik dibanding FTS3(Android Developer Team, 2013).

Teknik FTS ini digunakan karena dalam pencarian disini digunakan *virtual tabel* dan tidak memerlukan penggunaan “LIKE” yang menyebabkan penurunan peforma. Dengan menggunakan teknik ini didapatkan hasil pencarian dalam waktu kurang dari 1 detik. Dalam suatu pengetesan yang dilakukan pada SQLite didapatkan data perbandingan antara penggunaan “LIKE” pada *query* biasa dengan “MATCH” pada Full Text Search 3 dengan 517430 dokumen dapat dilihat pada tabel 3.1.

Teknik Pencarian	Waktu yang dibutuhkan
Query menggunakan “LIKE”	>22,5 detik
Full Text Search 3 menggunakan “MATCH”	<0,05 detik

Tabel 3.1 Perbandingan waktu keluaran penggunaan “LIKE” dan “MATCH”
(SQLite Developer Team, 2013)

Dalam implementasinya pencarian menggunakan metode ini menghasilkan hasil keluaran jauh lebih cepat dan tidak terjadi *freeze* dikarenakan menunggu waktu keluaran hasil pencarian. Diberikan *delay* sebelum mengeksekusi pencarian menggunakan FTS ini sebesar 500ms dan didapatkan hasil keluaran dibawah 1 detik. Sedangkan dengan

menggunakan *query* “LIKE” diperlukan waktu 1-2 detik dengan jumlah *record* yang sama yaitu 2000 dokumen. Namun, di sisi lain dengan penggunaan metode ini ukuran *database* menjadi lebih besar dari biasa yaitu sekitar 3 kali lebih besar daripada normal.

Metode	**Ukuran *Database***
Tanpa virtual tabel(non-FTS)	860 KB
Dengan virtual tabel (FTS4)	2,24 MB

Tabel 3.2 Perbandingan ukuran database dengan 2000 dokumen

Virtual table memiliki kebiasaan yaitu akan dihapus ketika koneksi diberhentikan. Dengan demikian kita perlu membuat ulang setiap kali koneksi terputus antara aplikasi dengan *database.* Untuk menyelesaikan masalah tersebut kita perlu membuat ulang *virtual table* ketika masuk ke dalam aplikasi atau ketika koneksi terhubung kembali. Untuk membuat tabel tersebut diperlukan waktu dan bila dilakukan pada saat memasuki *fragment* hal ini akan membuat degradasi peforma pencarian. Untuk itu, pembuatan *virtual table* dilakukan pada *thread* yang berjalan di *background* pada saat aplikasi dibuka pertama kali karena ketika koneksi terputus maka secara otomatis aplikasi akan *restart* kembali. Hal ini dilakukan sehingga dapat meningkatkan efisiensi waktu. Pencarian dapat berlangsung tanpa perlu menunggu pembuatan *virtual table* setiap masuk ke *fragment* universalSearch.

A. Struktur Tabel

Database yang digunakan dalam aplikasi ini adalah SQLite. Berikut adalah struktur tabel virtual yang dibuat oleh penulis untuk melakukan pencarian menggunakan teknik FTS.

Nama tabel : vclient

Fungsi : Tabel ini digunakan untuk menampung keseluruhan data yang diambil dari tabel client tetapi tidak semua *field* dimasukkan melainkan *field* yang diperlukan dan akan dibandingkan terhadap kata kunci saja.

Field Name	Type	Sumber Data
contactid	Integer primary key	Client.contactid
contactname	Varchar	Client.contactname
billingname	Varchar	Client.billingname
billingaddress1	Varchar	Client.billingaddress1
billingaddress2	Varchar	Client.billingaddress2
billingaddress3	Varchar	Client.billingaddress3
shippingname	Varchar	Client.shippingname
shippingaddress1	Varchar	Client.shippingaddress1
shippingaddress2	Varchar	Client.shippingaddress2
shippingaddress3	Varchar	Client.shippingaddress3
phone	Varchar	Client.phone
mobile	Varchar	Client.mobile
fax	Varchar	Client.fax
pager	Varchar	Client.pager
other	Varchar	Client.other
notes	Varchar	Client.notes

Tabel 3.3 Struktur tabel vClient

Nama Tabel : vdocument

Fungsi : Tabel ini digunakan sebagai referensi pencarian pada dokumen. Data diambil dari beberapa tabel lain antara lain tabel Document, Item dan Attachment sehingga perintah "JOIN" hanya digunakan pada saat pembuatan saja, tidak digunakan pada saat pencarian.

Field Name	Type	Sumber Data
documentID	Integer primary key	Document.documentID
contactname	Varchar	Document.contactname
emailRecipient	Varchar	Document.emailRecipient
addressLine1	Varchar	Document.addressLine1
addressLine2	Varchar	Document.addressLine2
addressLine3	Varchar	Document.addressLine3
shippingAddressLine1	Varchar	Document.shippingAddressLine1
shippingAddressLine2	Varchar	Document.shippingAddressLine2
shippingAddressLine3	Varchar	Document.shippingAddressLine3
documentNumber	Integer	Document.documentNumber
documentDate	Integer	Document.documentDate
documentType	Varchar	Document.type
description	Varchar	Item.description
attachmentdescription	Text	Attachment.description
title	Text	Attachment.title
filename	Text	Attachment.fileName
customfield1	Text	Document.customfield1
customfield2	Text	Document.customfield2
customfield3	Text	Document.customfield3
customfield4	Text	Document.customfield4
customfield5	Text	Document.customfield5
customfield6	Text	Document.customfield6
customfield7	Text	Document.customfield7

Tabel 3.4 Struktur tabel vdocument

Nama tabel : vproduct

Fungsi : Tabel ini digunakan untuk menampung keseluruhan isi data yang ada pada tabel product tetapi tidak semua *field* ditampung di sini melainkan hanya beberapa *field* yang akan dibandingkan dengan kata kunci pencarian

Field Name	Type	Sumber Data
productID	Integer primary key	Product.productID
category	Varchar	Product.category
description	Varchar	Product.description

Tabel 3.5 Struktur tabel vproduct

Nama tabel : vreceipt

Fungsi : Tabel ini digunakan untuk menampung keseluruhan isi data yang ada pada tabel receipt untuk melakukan pencarian data pada aplikasi Receipts2go tetapi tidak semua *field* ditampung di sini melainkan hanya beberapa *field* yang akan dibandingkan dengan kata kunci pencarian.

Field Name	Type	Sumber Data
receiptID	Integer primary key	Receipt.receiptID
description	Varchar	Receipt.description
category	Varchar	Receipt.category
vendor	Varchar	Receipt.vendor

Tabel 3.6 Struktur tabel vreceipt

Nama tabel : vtime

Fungsi : Tabel ini digunakan untuk menampung keseluruhan isi data yang ada pada tabel timeEntry untuk melakukan pencarian data pada aplikasi Receipts2go tetapi tidak semua *field* ditampung di sini melainkan hanya beberapa *field* yang akan dibandingkan dengan kata kunci pencarian.

Field Name	Type	Information
timeID	Integer primary key	timeEntry.timeID
day	Integer	timeEntry.day
start	Integer	timeEntry.start
finish	Integer	timeEntry.finish
description	Varchar	timeEntry.description
title	Varchar	timeEntry.title

Tabel 3.7 Struktur tabel vtime

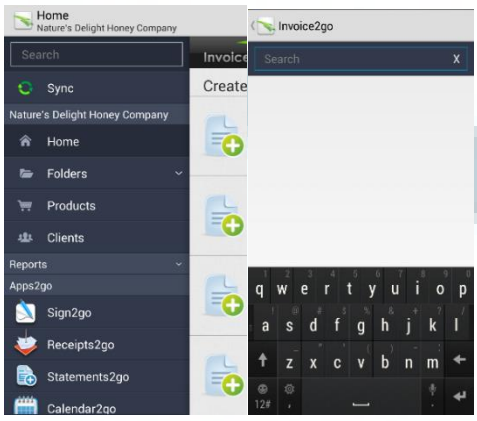
B. Alur Universal Search pada Invoice2go

Secara garis besar alur proses pencarian yang terjadi dalam *Universal Search* ini dimulai dari pembuatan tabel virtual. Proses pembuatan ini dilakukan di *background* pada saat aplikasi dijalankan atau *restart* menggunakan sebuah *thread* yang dibuat secara khusus untuk melakukan tugas ini. *Thread* ini bukan merupakan *User Interface* (UI) *Thread* sehingga tidak mengganggu waktu program *start-up* dan pengguna tidak terganggu akibat proses pembuatan proses tabel virtual tersebut karena pengguna tetap dapat melakukan hal lain selama proses ini berlangsung seperti menambah, merubah data dokumen ataupun klien maupun produk.

```
public void run() {
    if (UIUtils.isHoneycomb()){
        database.execSQL("CREATE VIRTUAL TABLE vclient USING fts4(contactname TEXT,billingname TEXT, " +
                "billingaddress1 TEXT,billingaddress2 TEXT,billingaddress3 TEXT,shippingname TEXT, " +
                "shippingaddress1 TEXT, shippingaddress2 TEXT, shippingaddress3 TEXT,phone TEXT);  ");
    }
    else
    {
        database.execSQL("CREATE VIRTUAL TABLE vclient USING fts3(contactname TEXT,billingname TEXT, " +
                "billingaddress1 TEXT,billingaddress2 TEXT,billingaddress3 TEXT,shippingname TEXT, " +
                "shippingaddress1 TEXT, shippingaddress2 TEXT, shippingaddress3 TEXT,phone TEXT);  ");
    }

    database.execSQL("INSERT INTO vclient select contactname, billingname, billingaddress1," +
            "billingaddress2,billingaddress3,shippingname,shippingaddress1,shippingaddress2,shippingaddress3,phone;");
}
.start();
```

Gambar 3.9 Potongan program pembuatan virtual tabel vclient

Pada bagian menu samping ditambahkan sebuah *editText* yang ketika diklik oleh pengguna akan mengarahkan pengguna ke suatu *fragment* baru. *Fragment* ketika aktif secara otomatis akan membuat *editText* mendapatkan fokus serta menampilkan *keyboard* sehingga user dapat langsung mengetik kata kunci pencarian.



Gambar 3.10 Tampilan Menu Samping dan Fragment UniversalSearch

Pencarian menggunakan FTS ini diimplementasikan dengan menggunakan *thread* yang berjalan di *background*. Hal ini dilakukan untuk membuat aplikasi tidak *freeze* untuk menunggu hasil pencarian sehingga user tetap dapat mengetik pada kolom untuk mencari dan pada saat bersamaan dilakukan aktivitas pencarian.

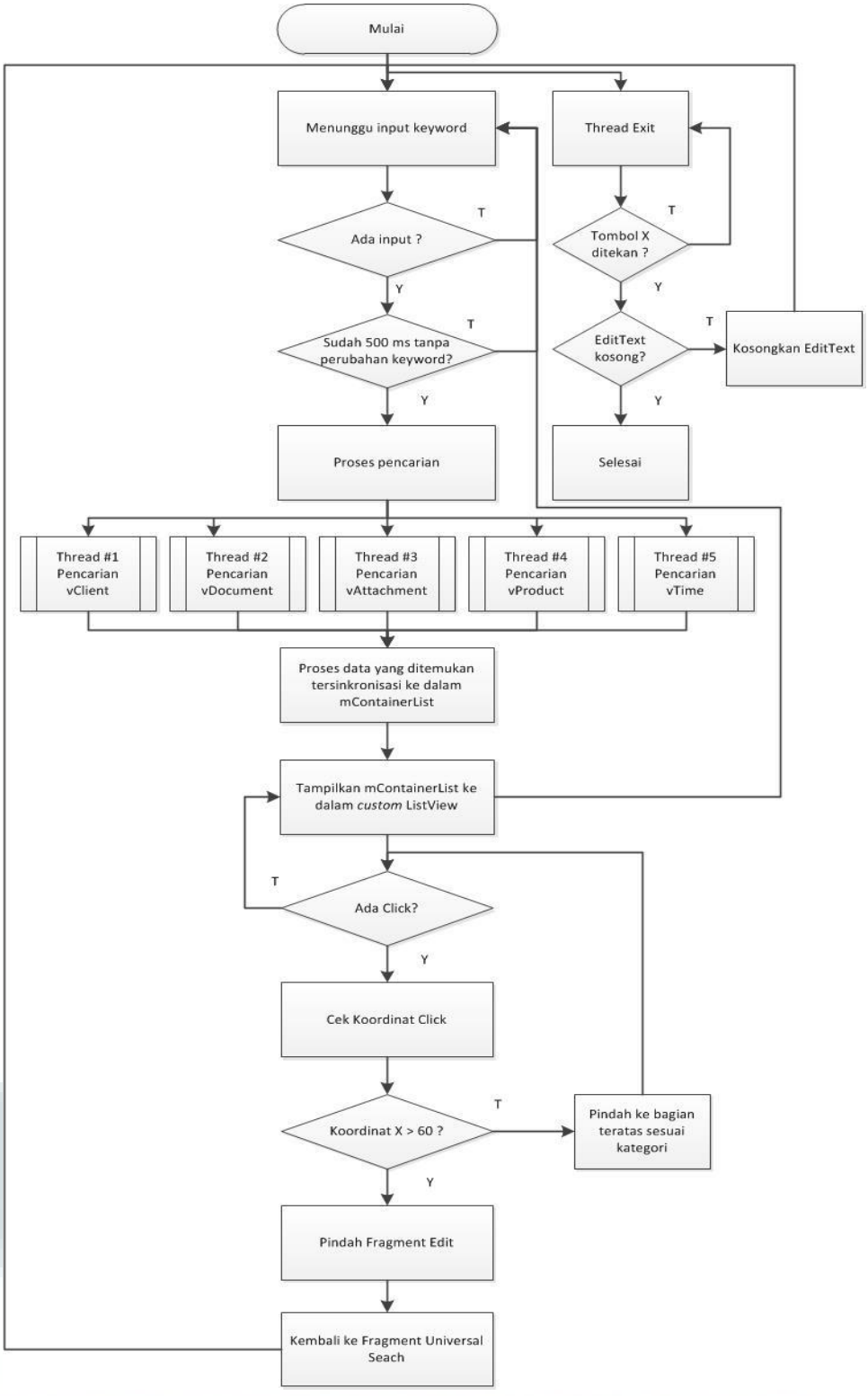
```
new Thread(new Runnable() {
    public void run() {
        mContainer=getdatabasehelper.findRowsbySql("SELECT billingname FROM vclient WHERE vclient MATCH '*"+keyword+"*'");
    }
}).start();
```

Gambar 3.11 Potongan program eksekusi pencarian client

Hasil pencarian dari *database* kemudian akan ditampung di dalam sebuah *container*. *Container* ini digunakan sebagai suatu penampung terhadap *view* yang akan dibuat dan ditampilkan. Hal ini diperlukan untuk tetap menyimpan *view* walaupun kita melakukan *scrolling* pada *custom listView* yang dibuat. Bila tidak ada *container* ini maka setiap kali *scrolling* dilakukan perlu dilakukan pemrosesan untuk menampilkan *view* tersebut.

Setiap hasil pencarian yang ditampilkan dapat dipilih. Ketika dipilih, maka *field* yang dijadikan *primary key* akan dikirimkan kembali untuk melakukan permintaan kepada *database* untuk mengambil data. Pengambilan data dilakukan menggunakan normal *query* dikarenakan hanya menggunakan "=" untuk pencarian sehingga tidak membuat lambat pencarian. Data hasil pencarian tersebut pasti unik karena merupakan *primary key* tersebut. Hasil tersebut kemudian digunakan untuk menampilkan menu untuk melihat dan dapat juga melakukan perubahan pada data. Setiap kategori memiliki tampilan yang berbeda. Penulis menggunakan *fragment* yang sudah ada untuk menampilkan data yang telah didapatkan dari *database.*

Secara umum, alur pencarian yang dilakukan dalam *fragment* universalSearch dapat dilihat pada gambar 3.12.



Gambar 3.12 Flow Chart Universal Search

Dalam *fragment* editClient dan editProduct yang dipanggil bila ada perubahan terhadap isinya, perubahan tersebut tidak langsung disimpan di dalam *database* melainkan hanya dikembalikan sebagai suatu objek yang bersifat sementara. Dengan demikian akan menyebabkan pencarian pada *universal search* menjadi tidak tepat hasil keluarannya. Hal ini diatasi dengan membuat suatu perubahan prosedur untuk menyimpan ke dalam *database.* Prosedur ini dibuat bukan pada *fragment* editClient ataupun editProduct melainkan pada *fragment* universalSearch. Pada *fragment* ini dipanggil *startActivityForResult(Intent, int).* Melalui *method* ini akan dipanggil tampilan untuk menampilkan *fragment* ataupun *activity* untuk melihat keseluruhan isi.

Setelah *fragment* atau *activity* dipanggil pengguna dapat merubah isi dari *field* yang ada dan melakukan *updating* sesuai kebutuhannya. *Method startActivityForResult(Intent, int)* akan memanggil *activity* tertentu dan memiliki keuntungan yaitu dapat membedakan hasil atau pilihan dari pengguna untuk menyimpan atau membatalkan transaksi perubahan tersebut pada saat sebelum *activity* tersebut dihancurkan.

Setelah *activity* selesai maka hasil transaksi akan dilihat dan dibedakan pada *method onActivtyResult((int requestCode, int resultCode, Intent data).* Dalam *onActivityResult* ada tiga parameter yang dapat membedakan hasil ketika kembali kepada *fragment* universalSearch. Parameter *requestCode* menentukan *activity* mana yang dipanggil misalkan editClient, editDocument, ataupun editProduct. Masing-masing *activity* memiliki *requestCode* yang berbeda. Sedangkan *resultCode* akan menentukan hasil dari transaksi yang dilakukan apakah pengguna melakukan perubahan dan menyimpannya atau tidak.

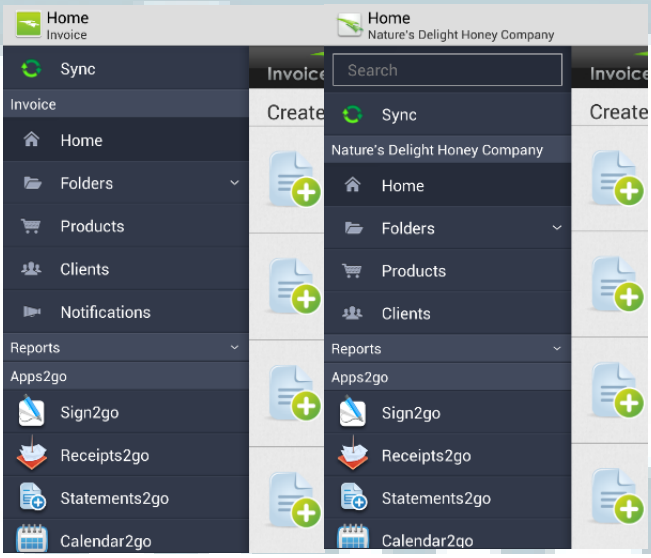
Melalui parameter *requestCode* dan *resultCode* dapat dibedakan prosedur untuk masing-masing setelah *activity* selesai dilakukan. Untuk editClient dan editProduct dimana pengguna menyimpan perubahan, dilakukan prosedur serupa yaitu menyimpan perubahan yang dilakukan ke dalam *database* dengan memanggil *method* untuk melakukan *update* pada masing-masing tabel. Kemudian, dilakukan *updating* pada *virtual table* sehingga data pencarian juga merupakan data terakhir. Sedangkan, pada pencarian tabel timeEntry, receipt, dan attachment pada bagian ini hanya ada penambahan

prosedur untuk melakukan *updating* pada *virtual table*. Hal ini dikarenakan pada *activity* sudah dilakukan *updating* pada tabelnya masing-masing sehingga tidak perlu lagi untuk dilakukan.

3.3.1.5 User Interface (UI)

Dalam aplikasi Invoice2go, UI merupakan salah satu hal yang menjadi perhatian utama. UI yang ada di dalamnya harus mudah diakses, menarik dan mudah untuk digunakan. Menanggapi hal itu, diperlukan suatu teknik dimana pengguna dapat dengan cepat beradaptasi dengan penambahan fitur *universal search* pada aplikasi ini.

Perubahan yang dilakukan adalah memodifikasi *fragment* yang berisi menu samping dengan penambahan *text field*. Bila *text field* ini berada dalam kondisi aktif, maka pengguna akan diarahkan ke suatu *fragment* baru yang dibuat oleh penulis. *Fragment* ini seolah-oleh merupakan suatu *text field* yang sama dengan yang ada pada menu samping tetapi sebenarnya berbeda.



Gambar 3.13 Perbedaan tampilan menu samping sebelum dan sesudah penambahan fitur Universal Search

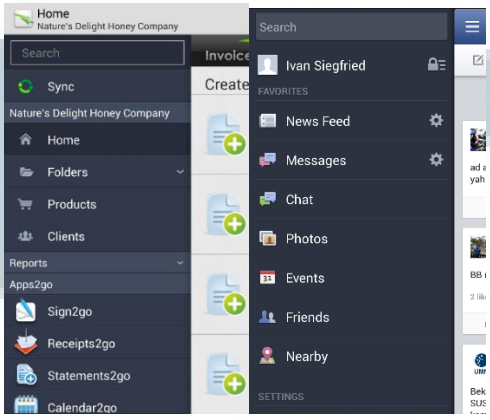
Untuk memudahkan pengguna, pada saat pengguna memilih Menu Search maka pengguna akan diberikan layar penuh untuk pencarian. Pencarian dilakukan pada

fragment baru. Pada saat *fragment* dipanggil maka secara otomatis fokus akan berada pada textView dan akan ditampilkan *keyboard* pada bagian bawah layar sehingga pengguna dapat langsung mengetik kata kunci pencariannya. Hal ini dilakukan dengan memanfaatkan *runnable().* Hal ini dilakukan untuk memberikan waktu tunda sementara sebelum *keyboard* ditampilkan. Bila tidak diberikan waktu tunggu maka *keyboard* tidak selalu muncul. Hal ini dikarenakan dalam pembacaan program waktu dieksekusi tidak terlaksana dengan benar karena UI *thread* juga mengerjakan penggambaran seluruh *fragment*.

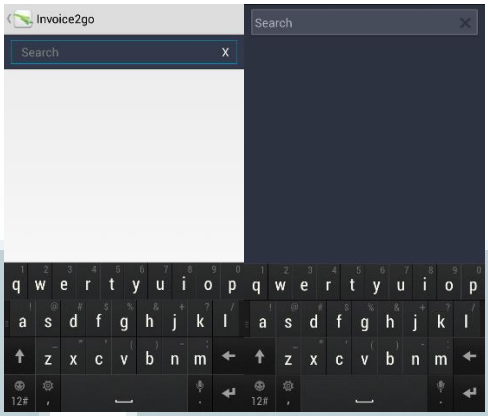
```
EditText editText = (EditText) findViewById(R.id.uvsearch);
editText.requestFocus();
editText.postDelayed(new Runnable() {
    @Override
    public void run() {
        InputMethodManager keyboard = (InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);
        keyboard.showSoftInput(editText, 0);
    }
},200);
```

Gambar 3.14 Potongan program pada saat fragment aktif

Secara umum, *behavior* yang ada fitur *univeral search* ini mirip dengan fungsi pencarian pada Facebook untuk Android. Hal ini secara tidak langsung dapat membuat pengguna menjadi lebih terbiasa dan aplikasi ini menjadi lebih '*user friendly'*. Untuk menampilkan data hasil pencarian digunakan komponen *listView* yang telah tersedia dalam SDK dengan beberapa perubahan untuk memberikan pengalaman lebih baik kepada pengguna. Perbandingan antara *behavior universal search* pada aplikasi Invoice2go dengan *search* pada Facebook dapat dilihat pada gambar 3.15 dan 3.16.



Gambar 3.15 Tampilan Perbandingan Menu Samping Invoice2go dengan Facebook



Gambar 3.16 Tampilan Perbandingan Fragment Invoice2go dengan Facebook

Konsep awal yang diinginkan adalah pembagian *list* menjadi dua bagian. Di bagian kiri terletak gambar yang menunjukkan jenis dari data yang didapatkan seperti: dokumen, klien, ataupun produk sedangkan pada bagian kanan terletak judul ataupun deskripsi dari data yang ditampilkan. Untuk memperoleh hasil demikian ada beberapa cara yang dilakukan oleh penulis. Namun, semuanya menggunakan konsep yang sama yaitu merubah *behavior* standar dari *listView* dengan mengganti beberapa metode yang dimiliki.

Cara pertama yang dilakukan penulis adalah mengimplementasikan 2 buah *listView* untuk menampilkan hasil pencarian. *ListView* pertama hanya menampilkan *icon* saja sedangkan *ListView* kedua menampilkan teks. Kedua buah *listView* ini disinkronisasikan untuk *scrolling* sehingga akan bergeser secara bersamaan. *listView* pertama merupakan sebuah *StickyListHeaders* yang merupakan suatu *library* yang merubah *behavior* suatu *listView* standar dengan menampilkan *header* selalu di paling atas (Emil Sjölander,2013). Walaupun digeser maka *header* akan tetap berada di paling atas *listView* tersebut. Dengan menggunakan *library* ini penulis dapat menyajikan tampilan layaknya *tableView* seperti pada platform iOS.

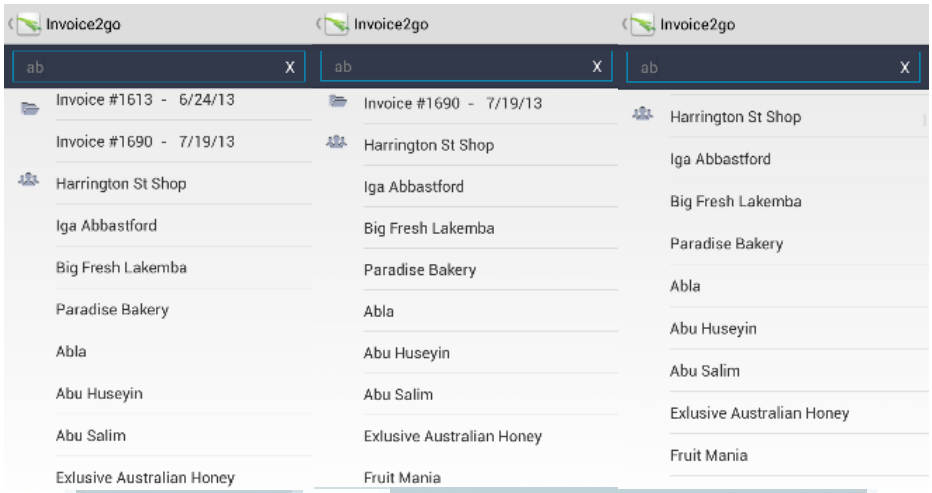
Teknik penentuan *header* dilakukan penulis dengan melakukan proses pada *adapter*. Data pertama dari setiap kategori dianggap sebagai *header* dan diberikan *icon* untuk ditampilkan sedangkan data lainnya tidak dianggap sebagai *header*. Dengan

melakukan beberapa modifikasi ringan pada *listView* tersebut maka tampilan yang diinginkan pun dapat tercapai.

Teknik pertama ini mengalami permasalahan pada saat *scrolling* dan penggunaan *memory* yang besar untuk menyimpan 2 buah *arrayList* yang berisi data dan gambar. Masalah utama yang menjadi sorotan dalam implementainya adalah *scrolling* yang tidak dapat berjalan lancar dan patah-patah. Walaupun sudah digunakan suatu *container* untuk menyimpan keseluruhan *view* tersebut hal ini tetap tidak dapat diatasi karena dalam waktu yang sama perlu dikerjakan pemrosesan dua buah *listView*.

Teknik kedua yang digunakan sebagai alternatif adalah membuat sebuah *custom listView*. Dalam pembuatannya penulis mempelajari contoh dari *library* StickyListHeader dan memodifikasinya menjadi dua bagian. Dengan penggunaan sebuah *custom listView* kelemahan dari implementasi 2 buah *listView* dapat dihilangkan. Tidak terjadi duplikasi penggunaan *memory* untuk penyimpanan *arrayList* dan optimalisasi peforma dalam *scrolling* dapat tercapai.

Dalam *custom listView* ini, *listView* normal dibagi menjadi 2 bagian yaitu: bagian kiri untuk menampilkan gambar yang menunjukkan kategori dan bagian kanan berisi beberapa *textView*. Keduanya dibatasi sebesar 60 piksel untuk *icon* sedangkan sisanya bagian kanan digunakan untuk *textView*.



Gambar 3.17 Proses Scrolling pada custom listView

Dalam menampilkan *icon*, *icon* yang ada diambil dari berbagai *icon* yang ada seperti *icon* aplikasi Receipts2go, *icon* Products, dsb. Karena berasal dari berbagai sumber ukuran *icon* pun berbeda satu sama lain. Untuk itu digunakan suatu metode untuk melakukan *resize* pada *icon* tersebut dengan tetap mempertahankan *aspect ratio*-nya untuk memperoleh tampilan *icon* yang baik.

```
Bitmap image = BitmapFactory.decodeResource(mContext.getResources(), id);
int heightresized = image.getHeight()/scaled;
int widthresized = image.getWidth()/scaled;
if(heightresized > widthresized)
    image = Bitmap.createScaledBitmap(image, image.getWidth()/widthresized, image.getHeight()/widthresized, false);
else
    image = Bitmap.createScaledBitmap(image, image.getWidth()/heightresized, image.getHeight()/heightresized, false);
StickyIconHeader header = new StickyIconHeader(image, i);
mHeaders.add(header);
```

Gambar 3.18 Potongan program cara menampilkan gambar pada custom listView

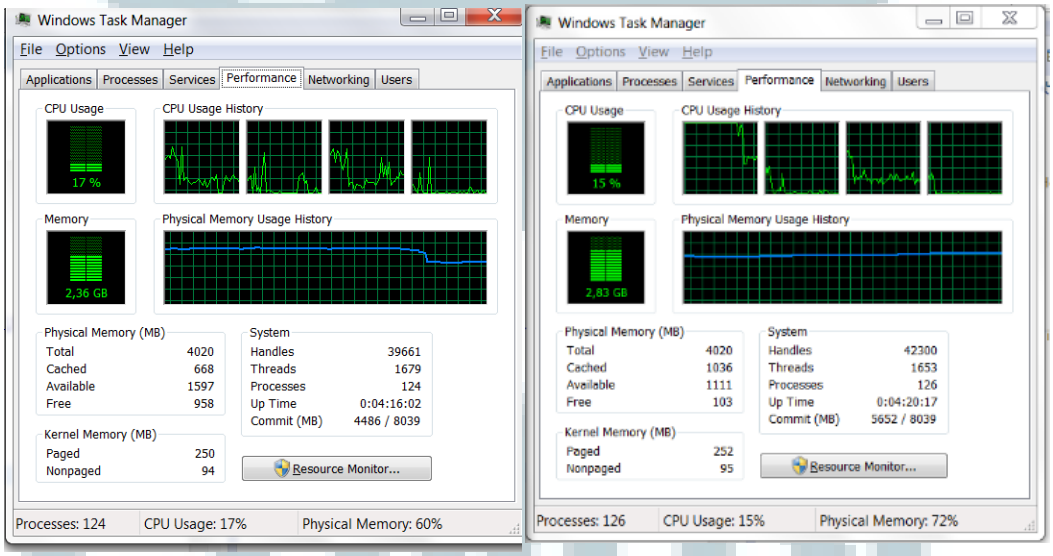
```
private void drawStickyHeader(Canvas canvas) {
    // Get the first position that is visible in the list
    int firstVisible = getFirstVisiblePosition();

    // Get the number of children displayed in the list
    int numOfChild = getChildCount();

    for(int viewPosition = 0; viewPosition < numOfChild; viewPosition++) {
        int currentPositionInList = firstVisible + viewPosition;

        // Get the bottom position of the cell
        int bottom = getChildAt(viewPosition).getBottom();
        // Get the top position of the cell
        int top = getChildAt(viewPosition).getTop();

        boolean hasHeader = mWrappedAdapter.isHeader(currentPositionInList);
        // Draw dynamic header
        if(hasHeader) {
            if(viewPosition == 0)
                mHeaderBitmap = mWrappedAdapter.getHeaderBitmap(firstVisible + viewPosition);
            canvas.drawBitmap(mWrappedAdapter.getHeaderBitmap(firstVisible + viewPosition), 5, Math.max(top, 0), mPaint);
        }
    }

    // Draw static header
    if(!(mHeaderBitmap == null)) {
        canvas.drawBitmap(mHeaderBitmap, 5, 0, mPaint);
    }
}
```

Gambar 3.19 Potongan program cara membuat custom listView untuk menampilkan Sticky Icon

3.3.1.6 Pengujian Aplikasi

Pengujian aplikasi Invoice2go dilakukan dalam beberapa tahap. Tahap awal yang dilakukan adalah pengujian yang dilakukan sendiri. Pengujian ini dilakukan dengan melakukan pengujian terhadap peforma aplikasi, keakuratan pencarian, dan efek samping terhadap aplikasi secara keseluruhan setelah penambahan fitur. Dalam tahap ini digunakan beberapa *smartphone* Android versi 4.1.1 untuk melakukan pengujian. Penggunaan *smartphone* dalam hal ini dilakukan karena penggunaan simulator membutuhkan waktu yang lama untuk pengiriman aplikasi dan instalasi aplikasi. Selain itu, kekurangan dari simulator adalah lambatnya *rendering* untuk gambar. Hal ini menyebabkan tidak semua *frame* dapat ditampilkan dan menjadikan tampilan navigasi menjadi tidak lancar.

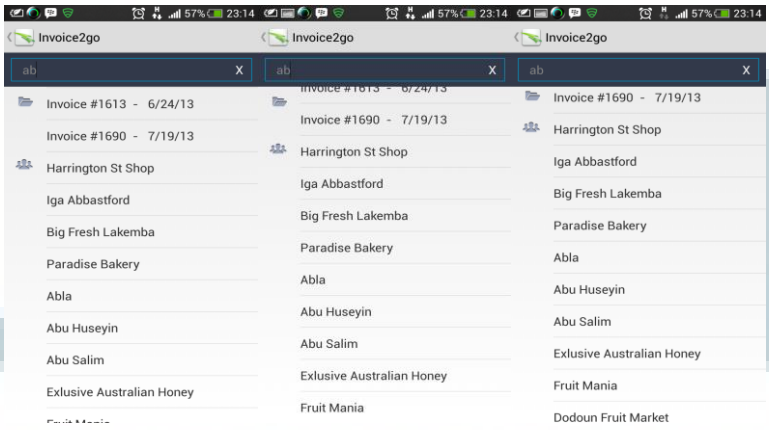
Gambar 3.20 Tampilan emulator Nexus 4



Gambar 3.21 Tampilan perbandingan task manager saat tidak menjalankan emulator (kiri) dengan saat menjalankan emulator (kanan)

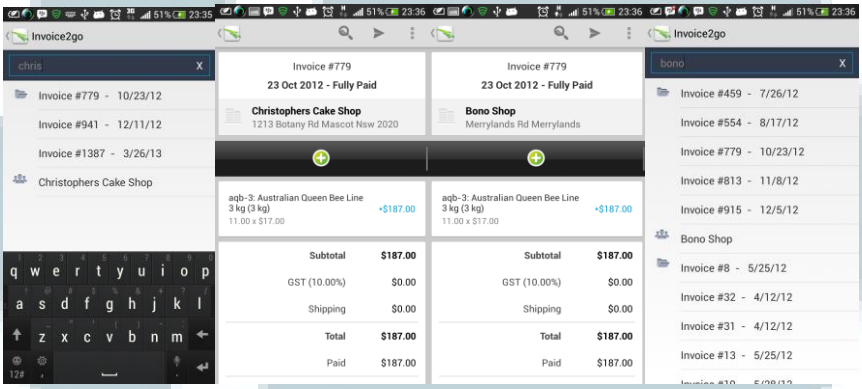
Tahap kedua pengujian dilakukan bersama dengan pembimbing lapangan. Pengujian ini dilakukan menggunakan beberapa versi Android dengan beberapa *device* . Hal ini dilakukan untuk mendeteksi kesalahan bila ada beberapa fitur yang tidak dapat digunakan pada versi sebelumnya. Untuk itu digunakan perangkat Android versi 2.3 (Gingerbread) untuk memenuhi standar minimal penggunaan aplikasi ini. Beberapa perangkat Android disiapkan antara lain Acer Iconia Tab, Asus Nexus 7, Samsung Galaxy SIII, HTC One X, dan HTC Legend. Sebuah *smartphone prototype* dari Intel pun disediakan di sana. Namun, untuk menggunakannya harus dilakukan pengiriman APK melalui Eclipse yang berjalan di Ubuntu dikarenakan Eclipse di Windows belum dapat mendeteksinya.

Pengujian dilakukan untuk melihat ukuran dan penempatan *layout* apakah sudah tepat atau belum. Pengujian bersama pembimbing lapangan ini dilakukan karena pembimbing lapangan lebih memahami struktur dari aplikasi dan lebih mengetahui apakah ada perubahan dari aplikasi secara keseluruhan sebelum dan sesudah dilakukan penambahan fitur oleh penulis. Setelah dilakukan pengujian, pembimbing lapangan akan memberi masukan dan saran untuk ke depannya agar penambahan fitur dapat berjalan lebih baik dan lancar. Penempatan dan ukuran dari tulisan juga disesuaikan dengan yang sudah ada agar tercapai konsistensi dalam penulisan. Permasalahan dan kekurangan dalam pengembangan aplikasi ini kemudian dianalisis dan dibenahi. Saran akan penempatan *layout* dan ukuran gambar maupun tulisan diimplementasikan kemudian.



Gambar 3.22 Pengecekan behavior custom listView

Hal lain yang juga divalidasi adalah apakah pencarian akan menghasilkan keluaran yang valid bila ada data yang dirubah lalu kembali ke *fragment* tersebut. Hasil divalidasi bila ada perubahan tersebut baik dirubah isinya maupun dihapus ataupun ada penambahan *entry*. Kemudian juga dilakukan pengujian pada *scrolling* apakah sudah *smooth* atau belum. *Behavior* dari *custom listview* pun diuji apakah sudah tepat atau belum menampilkan *icon*-nya.



Gambar 3.23 Validasi perubahan data pada Invoice#779

Pemilik perusahaan pada tahap akhir akan melihat dan mengevaluasi hasil pengembangan aplikasi yang dilakukan penulis dan memberi saran dan masukan untuk menjadikan aplikasi menjadi lebih baik. Saran dan masukan dari pemilik kemudian akan diimplementasikan ke dalam aplikasi tersebut. Pada tahap pengujian ini juga akan dibandingkan bagaimana perbandingan peforma dan sisi kenyamanan bagi pengguna setelah penambahan fitur tersebut antara *platform* Android dengan iOS. Untuk memudahkan dan membuat pengguna menjadi nyaman, *user interface* keduanya dibuat semirip mungkin sehingga bila pengguna menggunakannya pada *platform* yang berbeda pengguna tidak perlu membutuhkan waktu yang lama untuk beradaptasi.

Setelah dilakukan pengujian tersebut, hasil kerja penulis dipresentasikan di depan *developer* lain dan pemilik perusahaan. Dalam presentasi ini dilakukan pengujian ulang aplikasi tersebut dan mencari kekurangan serta *bug* yang ada di dalam aplikasi sebelum benar-benar di-*release* ke dalam Play Store. Kualitas pengembangan aplikasi lebih diutamakan dibandingkan *deadline* merupakan salah satu paham yang dianut di

perusahaan ini sehingga pengujian aplikasi dilakukan oleh beberapa *developer* sebelum di-*release* ke masyarakat luas.

3.3.2 Kendala yang Ditemukan

Dalam penambahan fitur *universal search* pada aplikasi Invoice2go ditemukan kendala dalam pembuatan *custom listView*. Penulis diminta untuk merubah *behavior* dari *listView* sehingga seolah-olah memiliki 2 bagian. Bagian kiri berisi gambar untuk mendeskripsikan kategori sedangkan bagian kanan berisi teks. Pengimplementasian dua buah *listView* membuat pemrosesan menjadi berat dan tidak efisien. Dampak negatif lain yang ditemukan adalah penurunan kecepatan untuk melakukan *scrolling*. Satu-satunya cara adalah dengan membangun sebuah *listView* dengan perubahan struktur di dalamnya.

3.3.3 Solusi untuk Kendala yang ditemukan

Dari kendala yang ditemukan penulis, penulis melakukan perubahan pada struktur dasar sebuah *listView*. Bagian kiri penulis buat sebesar 60 piksel untuk menampilkan gambar sedangkan sisanya diperlakukan secara normal. Perubahan *method* dispatchDraw dilakukan untuk mengalokasikan tempat untuk bagian gambar

```
@Override
protected void dispatchDraw(Canvas canvas) {
    canvas.save();
    canvas.translate(60, 0);
    super.dispatchDraw(canvas);
    canvas.restore();
    drawStickyHeader(canvas);
}
```

Gambar 3.24 Potongan program dispatchDraw

Dengan merubah bagian ini maka dapat *listView* dapat dipisahkan. Langkah berikutnya yang dilakukan adalah memodifikasi *behavior* dari bagian kiri agar berlaku seperti *Sticky Header*. Hal ini dilakukan sehingga tiap kategori hanya akan menampilkan 1 gambar/simbol saja pada bagian atas tiap kategori ataupun pada bagian teratas layar.