W

;/--n

**LAPORAN KERJA PRAKTEK – KI091391**

PERIODE 1 MARET s/d 20 APRIL 2012

**RANCANG BANGUN APLIKASI CUSTOMER RELATIONSHIP MANAGEMENT PADA PERANGKAT BLACKBERRY DI PT KRIYAMUDA INDONESIA**

GAMAL ABDUL NASSER

5109100003

ALDY AHSANDIN

5109100704

Dosen Pembimbing

Arya Yudhi Wijaya, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya 2012

****

**LAPORAN KERJA PRAKTEK – KI091391**

PERIODE 1 MARET s/d 20 APRIL 2012

**RANCANG BANGUN APLIKASI CUSTOMER RELATIONSHIP MANAGEMENT PADA PERANGKAT BLACKBERRY DI PT KRIYAMUDA INDONESIA**

GAMAL ABDUL NASSER

5109100003

ALDY AHSANDIN

5109100704

Dosen Pembimbing

Arya Yudhi Wijaya, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Surabaya 2012

# LEMBAR PENGESAHAN I

LAPORAN KERJA PRAKTIK

PT. KRIYAMUDA INDONESIA

JUDUL :

**RANCANG BANGUN APLIKASI CUSTOMER RELATIONSHIP MANAGEMENT PADA PERANGKAT BLACKBERRY DI PT KRIYAMUDA INDONESIA**

LOKASI :

**PT. KRIYAMUDA INDONESIA**

Jalan Gubeng Kertajaya 8C/27D

Surabaya, 1 Maret 2012

Pembimbing Kerja Praktik,

**Rono Toga Iqbal S.Kom**

CEO PT Kriyamuda Indonesia

# LEMBAR PENGESAHAN II

LAPORAN KERJA PRAKTIK

PT. KRIYAMUDA INDONESIA

JUDUL :

**RANCANG BANGUN APLIKASI CUSTOMER RELATIONSHIP MANAGEMENT PADA PERANGKAT BLACKBERRY DI PT KRIYAMUDA INDONESIA**

LOKASI :

**PT. KRIYAMUDA INDONESIA**

Jalan Gubeng Kertajaya 8C/27D

Surabaya, 1 April 2012

Dosen Pembimbing,

**Arya Yudhi Wijaya S.Kom, M.Kom**

19840904 201012 1 002

# DAFTAR ISI

[LEMBAR PENGESAHAN I iii](#_Toc328554965)

[LEMBAR PENGESAHAN II iv](#_Toc328554966)

[DAFTAR ISI v](#_Toc328554967)

[DAFTAR GAMBAR ix](#_Toc328554968)

[DAFTAR TABEL xiii](#_Toc328554969)

[BAB 1 PENDAHULUAN 1](#_Toc328554970)

[1.1 Latar Belakang 1](#_Toc328554971)

[1.2 Rumusan Masalah 1](#_Toc328554972)

[1.3 Batasan Masalah 2](#_Toc328554973)

[1.4 Tujuan 2](#_Toc328554974)

[1.5 Waktu Pelaksanaan 3](#_Toc328554975)

[1.6 Sistematika Penulisan 3](#_Toc328554976)

[BAB 2 PROFIL PERUSAHAAN 5](#_Toc328554977)

[2.1 Sejarah Perusahaan 5](#_Toc328554978)

[2.2 Profil Perusahaan 5](#_Toc328554979)

[2.2.1 Nama dan Aspek Hukum 5](#_Toc328554980)

[2.2.2 Visi dan Misi Perusahaan 6](#_Toc328554981)

[2.2.3 Produk dan Layanan 7](#_Toc328554982)

[2.2.4 Struktur Organisasi 8](#_Toc328554983)

[BAB 3 TINJAUAN PUSTAKA 9](#_Toc328554984)

[3.1 Blackberry 9](#_Toc328554985)

[3.2 Blackberry SDK 10](#_Toc328554986)

[3.3 JSON 10](#_Toc328554987)

[3.4 SQLite Relational Database 11](#_Toc328554988)

[BAB 4 ANALISIS DAN PERANCANGAN 13](#_Toc328554989)

[4.1 Analisa Kebutuhan 13](#_Toc328554990)

[4.2 Definisi Umum Aplikasi 16](#_Toc328554991)

[4.3 Perancangan Fungsionalitas Aplikasi 17](#_Toc328554992)

[4.3.1 Fungsionalitas Aplikasi 17](#_Toc328554993)

[4.3.2 Cara Kerja Aplikasi 19](#_Toc328554994)

[4.4 Perancangan Data 21](#_Toc328554995)

[4.5 Perancangan User Interface 23](#_Toc328554996)

[4.5.1 User Interface Menu Utama 23](#_Toc328554997)

[4.5.2 User Interface Tambah Kontak 24](#_Toc328554998)

[4.5.3 User Interface ContactList Screen 25](#_Toc328554999)

[4.5.4 User Interface Sub MenuScreen 26](#_Toc328555000)

[BAB 5 IMPLEMENTASI 29](#_Toc328555001)

[5.1 Memanipulasi Data Kontak 29](#_Toc328555002)

[5.1.1 Kelas ContactData 30](#_Toc328555003)

[5.1.2 Kelas CBuilder 31](#_Toc328555004)

[5.1.3 Kelas ContactBuilder 32](#_Toc328555005)

[5.1.4 Kelas ContactList 41](#_Toc328555006)

[5.2 Memanipulasi Reminder 44](#_Toc328555007)

[5.2.1 Kelas ReminderData 45](#_Toc328555008)

[5.2.2 Kelas RBuilder 46](#_Toc328555009)

[5.2.3 Kelas ReminderBuilder 47](#_Toc328555010)

[5.2.4 Kelas ReminderList 51](#_Toc328555011)

[5.3 Koneksi ke Jaringan Internet 53](#_Toc328555012)

[5.4 JSON Parser 54](#_Toc328555013)

[BAB 6 UJI COBA DAN EVALUASI 59](#_Toc328555014)

[6.1 Lingkungan Uji Coba 59](#_Toc328555015)

[6.2 Manipulasi Kontak 59](#_Toc328555016)

[6.3 Manipulasi Reminder 61](#_Toc328555017)

[6.4 Koneksi Network 62](#_Toc328555018)

[6.5 JSON Parser 63](#_Toc328555019)

[BAB 7 KESIMPULAN DAN SARAN 65](#_Toc328555020)

[7.1 Kesimpulan 65](#_Toc328555021)

[7.2 Saran 65](#_Toc328555022)

[DAFTAR PUSTAKA 67](#_Toc328555023)

[LAMPIRAN SOURCE CODE 68](#_Toc328555024)

***[Halaman ini sengaja dikosongkan]***

# DAFTAR GAMBAR

[Gambar 2.1 Struktur organisasi dari PT Kriyamuda Indonesia. 8](#_Toc328555079)

[Gambar 4.1 Diagram arsitektur aplikasi secara umum. Aplikasi yang dibuat terletak pada Blackberry unit milik sales. Aplikasi berhubungan dengan web aplikasi secara *realtime*. 16](#_Toc328555080)

[Gambar 4.2 UseCase Diagram dari aplikasi yang dibuat. Terdiri dari dua aktor. Tiap *usecase* berasal gabungan dari kebutuhan umum aplikasi. 18](#_Toc328555081)

[Gambar 4.3 Aktifitas Diagram Sinkronisasi Kontak. Terdiri dari 2 *swimlane* yang merepresantasikan masing aktor. 19](#_Toc328555082)

[Gambar 4.4 Aktifitas Diagram Sinkronisasi *Reminder*. Terdiri dari 2 *swimlane* yang merepresantasikan masing aktor. 20](#_Toc328555083)

[Gambar 4.5 (a) CDM Diagram, terdiri dari dua tabel. Tiap ContactData setidaknya memiliki 1 atau lebih ReminderData.(b)PDM Diagram, terdiri dari dua tabel. *Primary Key* dari ContactData menjadi *Foreign Key* di ReminderData. 21](#_Toc328555084)

[Gambar 4.6 UI dari Menu Utama. Terdapat menu tiga pilihan utama yaitu “About”, “Contact”, dan “Reminder”. 23](#_Toc328555085)

[Gambar 4.7 UI dari tambah kontak. Hampir sama seperti pada UI tambah kontak di Blackberry. Terdiri dari koleksi *field-field* yang bisa diisi. 24](#_Toc328555086)

[Gambar 4.8 UI ContactList Screen, dimana perbedaannya terletak pada isi *command* di Context Menu. 25](#_Toc328555087)

[Gambar 4.9 UI Sub Menu Screen, dimana perbedaannya antara *submenu*terletak pada isi *command* di Context Menu. 26](#_Toc328555088)

[Gambar 5.1 Tabel kode kelas ContactData 30](#_Toc328555089)

[Gambar 5.2 Tampilan souce code implementasi kelas CBuilder 31](#_Toc328555090)

[Gambar 5.3 Kode fungsi contructor pada kelas ContactBuilder 33](#_Toc328555091)

[Gambar 5.4 Kode fungsi pengecekan kontak 34](#_Toc328555092)

[Gambar 5.5 Kode fungsi penyimpanan kontak 35](#_Toc328555093)

[Gambar 5.6 Implementasi fungsi penyetelan tanggal lahir 36](#_Toc328555094)

[Gambar 5.7 Kode fungsi penyetelan PIN Blackberry 37](#_Toc328555095)

[Gambar 5.8 Kode fungsi untuk melakukan penyetelan email pada kontak 38](#_Toc328555096)

[Gambar 5.9 Kode fungsi penyetelan nama 38](#_Toc328555097)

[Gambar 5.10 Fungsi-fungsi penyetelan nomor telepon 40](#_Toc328555098)

[Gambar 5.11 Kode fungsi – fungsi penyetelan alamat. 41](#_Toc328555099)

[Gambar 5.12 Implementasi kode penghapusan 41](#_Toc328555100)

[Gambar 5.13 Kode yang Berfungsi Menampilkan Daftar Kontak 42](#_Toc328555101)

[Gambar 5.14 Kode fungsi mendapatkan kontak berdasar ID 44](#_Toc328555102)

[Gambar 5.15 Kode Kelas ReminderData 45](#_Toc328555103)

[Gambar 5.16 Kode Kelas RBuilder 46](#_Toc328555104)

[Gambar 5.17 Kode fungsi inisialisasi ReminderBuilder 47](#_Toc328555105)

[Gambar 5.18 Kode fungsi pengecekan reminder 48](#_Toc328555106)

[Gambar 5.19 Kode fungsi penyimpanan reminder 49](#_Toc328555107)

[Gambar 5.20 Kode fungsi penyimpanan summary reminder 50](#_Toc328555108)

[Gambar 5.21 Kode fungsi penyimpanan waktu reminder 51](#_Toc328555109)

[Gambar 5.22 Kode fungsi penyimpanan related contact 51](#_Toc328555110)

[Gambar 5.23 Kode fungsi untuk mendapatkan list reminder 52](#_Toc328555111)

[Gambar 5.24 Kode fungsi untuk menghapus reminder 53](#_Toc328555112)

[Gambar 5.25 Kode koneksi Jaringan 54](#_Toc328555113)

[Gambar 5.26 File *Dummy* JSON untuk ujicoba. 55](#_Toc328555114)

[Gambar 5.27 Implementasi kode untuk fungsi pengambilan ID 56](#_Toc328555115)

[Gambar 5.28 Implementasi kode untuk fungsi membaca nama dan alamat 56](#_Toc328555116)

[Gambar 6.1 Tampilan uji coba fungsional manipulasi kontak 60](#_Toc328555117)

[Gambar 6.2 Mengakses data kontak pada aplikasi 60](#_Toc328555118)

[Gambar 6.3 Tampilan halaman uji coba *reminder* 61](#_Toc328555119)

[Gambar 6.4 Tampilan halaman uji coba pada *reminder*yang masuk dalam aplikasi 62](#_Toc328555120)

[Gambar 6.5 Hasil tampilan uji coba koneksi ke halaman google 62](#_Toc328555121)

[Gambar 6.6 Tampilan hasil dari JSON Parser berupa pengambilan data nama. 63](#_Toc328555122)

[Gambar A.0.1 Kelas BasicContactData sebagai model data dasar dari ContactData 69](#_Toc328555123)

[Gambar A.0.2 Kelas BasicReminderData sebagai model data dasar dari ReminderData 70](#_Toc328555124)

[Gambar A.0.3 Kelas AdvancedUIScreen sebagai *base class* dari semua screen. 71](#_Toc328555125)

[Gambar A.0.4 Kelas BBSampleAplication sebagai *starting class* dari aplikasi. 74](#_Toc328555126)

[Gambar A.0.5 Kelas ContactScreen, *screen* penambahan kontak melalui aplikasi. 78](#_Toc328555127)

[Gambar A.0.6 Kelas PhoneListenerCst, *interface* untuk mendapatkan status panggilan yang dilakukan melalui aplikasi. 80](#_Toc328555128)

[Gambar A.0.7 Kelas MainMenuScreen, *screen* pertama yang ditampilkan aplikasi. Berfungsi sebagai menu utama. 83](#_Toc328555129)

[Gambar A.0.8 Kelas PersistenceManager, kelas pengendali database yang digunakan. 88](#_Toc328555130)

***[Halaman ini sengaja dikosongkan]***

# DAFTAR TABEL

[Tabel 4.1 Field-field yang mampu ditampung data kontak Blackberry. 13](#_Toc328555131)

[Tabel 4.2 Detail dari tabel ContactData. Menyesuaikan dengan PDM Diagram pada Gambar 4.5(b). 22](#_Toc328555132)

[Tabel 4.3 Detail dari tabel ReminderData. Menyesuaikan dengan PDM diagram pada Gambar 4.5(b). 22](#_Toc328555133)

***[Halaman ini sengaja dikosongkan]***

# PENDAHULUAN

## Latar Belakang

PT Kriyamuda Indonesia merupakan sebuah software hause yang berkecimpung dibidang IT terutama bidang teknologi web dan aplikasi *mobile*. *Software house* ini sudah berdiri sejak tahun 2009, selama itu sudah banyak aplikasi dan program yang berhasil dikembangkan. Selain membuat produk PT Kriyamuda Indonesia juga menjalin barbagai rekanan dengan beberapa software hause lokal maupun internasional melalui beberapa jarangan yang dimilikinya. Dengan melihat sekilas profil *sotfware house* tersebut maka kami memetuskan untuk melakukan kerja praktek di software house tersebut. Sehingga kompetensi yang diharapkan dari kerja praktek ini dapat kami peroleh.

Salah satu proyek yang sedang dikerjakan oleh PT Kriyamuda Indonesia adalah proyek pengembangan aplikasi *mobileCRM (Customer Responsibility Management)* untuk sebuah perusahaan yang bergerak di bidang otomotif yaitu Auto2000. Aplikasi ini adalah bagian dari produk yang sudah ada, tapi difokuskan untuk mengawasi dan memudahkan kerja sales di perusahaan tersebut. Aplikasi ini dibuat menggunakan platform Blackberry. Tapi pada kesempatan kerja praktek ini lebih diarahkan ke riset pada pengembangan aplikasi di Blackberry menggunakan model aplikasi *CRM*.

## Rumusan Masalah

Permasalahan yang diangkat dalam penyusunan Kerja Praktek ini adalah:

1. Bagaimana memanfaatkan teknologi Blackberry SDK sebagai inti dari aplikasi yang akan dibangun?
2. Bagaimana memanfaatkan teknologi Blackberry SDK dan sinkronisasinya dengan web aplikasi yang berbasis PHP?

## Batasan Masalah

Batasan-batasan dalam pembuatan aplikasi pada kegiatan kerja praktik ini adalah sebagai berikut.

1. Aplikasi yang dibangun merupakan aplikasi mobile yang berjalan di platform Blackberry
2. Aplikasi ini dikembangkan menggunakan Blackberry SDK versi 5 untuk java
3. *Database Management System* yang digunakan aplikasi ini adalah SQLite.
4. Aplikasi ini mampu berjalan di Blackberry OS 5 ke atas.

## Tujuan

Kerja Praktik merupakan salah satu mata kuliah yang harus diambil guna memberikan pemahaman dan pengenalan tentang kondisi dunia kerja yang harus segera diketahui oleh mahasiswa. Selain itu, Kerja Praktik merupakan media komunikasi yang memungkinkan dilaksanakannya pertukaran informasi antara mahasiswa, lembaga perguruan tinggi, serta kalangan industri.

Adapun hal-hal khusus yang menjadi tujuan pelaksanaan Kerja Praktik ini adalah sebagai berikut.

1. Mempelajari dan memanfaatkan teknologi Blackberry SDK
2. Mensinkronisasikan aplikasi Blackberry dengan web aplikasi berbasis php
3. Membuat *prototype* aplikasi *CRM* yang membuat konsumen lebih dihargai.

## Waktu Pelaksanaan

Kerja praktik dilaksanakan selama satu setengah bulan, terhitung sejak tanggal 1 Maret 2012 sampai dengan tanggal 20 April 2012.

## Sistematika Penulisan

Laporan Kerja Praktik ini dibagi menjadi 5 bab dengan rincian sebagai berikut:

* **Bab I: Pendahuluan**

Pada bab ini, dijelaskan latar belakang, permasalahan yang dihadapi, tujuan, batasan masalah yang dihadapi, waktu pelaksanaan kerja praktik, serta sistematika penulisan laporan kerja praktik.

* **Bab II: Sejarah dan Profil Perusahaan**

Pada bab ini dijelaskan secara singkat mengenai PT Kriyamuda Indonesia.

* **Bab III: Tinjauan Pustaka**

Pada bab ini dijelaskan tentang tinjauan pustaka yang digunakan untuk menyelesaikan aplikasi yang dibuat.

* **Bab IV: Analisa dan Perancangan**

Pada bab ini dijelaskan mengenai desain antarmuka aplikasi serta desain basis data yang digunakan dalam aplikasi. Selanjutnya dijelaskan pula implementasinya.

* **Bab V: Implementasi**

Bab ini berisi kesimpulan dan saran dari proses pelaksanaan tugas praktik.

* **Bab VI: Uji Coba dan Evaluasi**

Bab ini berisi kesimpulan dan saran dari proses pelaksanaan tugas praktik.

* **Bab VII: Kesimpulan dan Saran**

Bab ini berisi kesimpulan dan saran dari proses pelaksanaan tugas praktik.

***[Halaman ini sengaja dikosongkan]***

# PROFIL PERUSAHAAN

## Sejarah Perusahaan

PT Kriyamuda Indonesia merupakan sebuah *software house* yang didirikan oleh beberpa orang yang memiliki visi yang sama pada 2009. Pendiri utama softwarea house ini adalah beberapa alumni ITS yang bertemu dalam media IKA ITS, salah satunya adalah Rono Toga Iqbal, S. Kom yang saat ini menjadi direktur utama perusahaan.

Pada awalnya *software house* ini berkecimpung di beberapa bidang, selain IT juga multimedia seperti pembuatan CD interaktif, animasi dan lain sebagainya. Tapi semenjak tahun 2010, terjadi perubahan yang signifikan. Beberapa pendiri awal *software house* ini sudah tidak begitu fokus pada perusahaan ini, sehingga hanya ada satu petinggi yang fokus.

Dan orang itu adalah, Rono Toga Iqbal, S. Kom. Selaku satu-satunya pendiri yang masih fokus pada perusahaan ini melakukan beberpa perubahaan pada sistem perusahaan. Dibawah pimpinannya perusahaan menjadi lebih terarah dan fokus pada satu bidang, yaitu web mobile.

Sampai saat ini perusahaan sudah memasuki tahun keduanya dan selama itu beberapa proyek baik itu secara mandiri maupun rekanan dengan *software house* lain telah berjalan dan selesai. Dan perlahan tapi pasti sekarang PT Kriyamuda Indonesia mulai menujukkan jati diri dan eksistensinya.

## Profil Perusahaan

### Nama dan Aspek Hukum

Kriyamuda Indonesia, PT

Notaris SRI URWANINGSIH, SH. , tanggal 9 Oktober 2009, nomor 13

SK Menkumham : AHU-54105.AH.01.01.Tahun 2009

SIUP : 503 / 6683.A / 436.6.11 / 2009

TDP : 13.01.1.74.21447

NPWP : 02.825.397.9-604.000

PKP : PEM-210 / WPJ.11 / KP.0103 / 2009

### Visi dan Misi Perusahaan

Visi

‘*Its Learness’*.

Setiap aplikasi yang dihasilkan olah Kriyamuda harus user friendly, bahkan kalau perlu tidak perlu manual book-nya. Karena itu ‘*its learness’*.

Misi

Misi Kriyamuda Indonesia tersirat dalam beberapa nilai dan filosopi sebagai berikut :

*Values*

*Focus is the key of speed, not skill. Sincere is the key of quality, not money. Happiness is the key of creativity, not time. By combining focus, sincere and happines, PT. Kriyamuda Indonesia is delivering a speed quality of creativity.*

*Philosopy*

*Clients and customers are not just king to us, they are inspirations that lays between heaven and earth, creating bridge between dream and reality. Their dream is our obsession to reach our very own dreams. We realize, that the only way to get to our dreams is by making every client’s and customer’s dream come true.*

*BEAUTIFUL TECHNOLOGY that WORKS!*

*Mankind is built on technology. From the first use of slate to the perfection of pc tablet, from the discovery of spoon to the exploration of moon, technology lives side by side with human kind. Begun as tools, technology put itself to be part of human life. Without technology, flinstone will be a very modern age.*

*Every human was created from love, not hate. That’s why human consist of beauty more then beast. It is also fair that only a beautiful technology can be involved in human evolution. To simplify human works, a beautiful technology always invented to create a better future*

### Produk dan Layanan

Berikut adalah beberapa produk dan layanan yang diberikan oleh PT Kriyamuda Indonesia.

#### Produk

Produk utama PT Kriyamuda Indonesia adalah sebagai berikut :

1. Iliski

Adalah sebuah *CRM* yang dikembangkan dengan lebih mendekati seperti social networking. Dimana sales dan customer dapat saling berinteraksi dengan mudah.

1. Humano

Adalah aplikasi untuk *HRD*. Dimana aplikasi akan dapat diintegrasikan dengan beberapa sistem pada *HRD*, diantaranya aplikasi absensi dengan sidik jari.Adalah implementasi dari *teknologi Augmented Reality* untuk keperluan komersil, seperi iklan atau pameran produk tertentu.

#### Dasar Pengolahan Citra

Layanan dari PT Kriyamuda sebenarnya tidak begitu berbeda dengan produk yang dibuat, hanya saja layanan lebih di arahkan ke produk yang dalam bentuk proyek baik mandiri maupun rekanan. Beberapa diantaranya adalah proyek aplikasi web dan desktop interaktif.

### Struktur Organisasi

Gambar ‎2.1 Struktur organisasi dari PT Kriyamuda Indonesia.

Dari Gambar 2.1 bisa dilihat bahwa dalam PT Kriyamuda Indonesia terdapat dua divisi. Dua divisi itu adalah divisi produksi dan divisi marketing. Di divisi produksi terdapat dua bagian yang memiliki tugas berbeda, yaitu bagian design bertugas dalam mendesign tampilan dari apa yang dikerjakan dan implementasi bertugas mengimplementasikan dan merancang sistem yang dibangun. Sedangkan divisi marketing bertugas untuk memasarkan dan penjagaan hubungan dengan klien yang pernah diajak kerjasama.

# TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai tinjauan pustaka yang mendukung penyusunan Laporan Kerja Praktek. Bab ini menjelaskan tentang Blackberry sebagai *platform* yang digunakan dan Blackberry SDK sebagai *development tool* yang digunakan.

## Blackberry

BlackBerry adalah perangkat *mobile email* dan *smartphone* yang dikembangkan dan dirancang oleh perusahaan Kanada Research In Motion (RIM) sejak tahun 1999.  
 PerangkatBlackBerry dirancang memiliki berbagai macam fungsi. Diantaranya berfungsi sebagai alat digitalpribadi, pemutar media,browser internet, perangkat game, dan banyak lagi.Mereka terkenal karenakemampuan mereka untukmengirim dan menerimaemail serta pesaninstandengan tetap menjaga tingkatkeamanan yang tinggimelalui *enkripsi*atau pengkodean pesan.Perangkat BlackBerrymendukungberbagai macamfitur pesan singkat, yaitu dengan adanya BlackBerry Messenger.  
 Pada tahun 2011, AccountBlackBerry menyumbang 3% dari penjualanperangkat *mobile*di seluruh dunia, sehingga membuat produsenRIMpembuatperangkat yang palingpopulerkeenam(25% daripenjualan perangkatmobilesmartphone). Selain itu layanan konsumen BlackBerry Internet tersedia di 91negara di seluruh dunia di lebih dari 500 *operator* layanan *mobile* yang menggunakan berbagai macam teknologi *mobile* pada Oktober2011, ada tujuh puluh juta pelanggan di seluruh dunia untuk BlackBerry. Saat inidi Karibia danAmerika Latin,memilikipenetrasitertinggi*smartphone* Blackberrydi seluruh duniadengan*penetrasi* sampai sekitar45 persen.

Perangkat Modern Blackberry berbasis GSM menggabungkan ARM7, 9 atau *prosesor* ARM 11, sementara yang lebih tua BlackBerry 950 dan 957 menggunakan Intel 80386 prosesor. Model BlackBerry terbaru yang disebut "Torch" memiliki Processor 1,2 GHz MSM8255Snapdragon, 768MB memori sistem), dan 8 GBon-board penyimpanan.

## Blackberry SDK

Pengembangan aplikasi di Blackberry bisa menggunakan berbagai *programming language*  antara lain, C/C++(NativeSDK), HTML(WebWorks, hanya untuk Blackberry OS 5/6 ke atas), Actionscript(AdobeAIR, hanya untuk Blackberry yang support Flash Player), Java(Android *runtime*, Blackberry SDK *for* Java).

Blackberry SDK *for* Java bisa didapatkan dalam bentuk lengkap beserta IDEnya atau berupa plugin untuk Eclipse. Keduanya baru ada untuk version 4 keatas dan baru bisa digunakan di OS Windows dan Mac. Blackberry SDK *for* Java merupakan turunan dari SDK J2ME yang telah dikembangkan sesuai dengan kemampuan dari platform Blackberry.

Di dalam SDK ini sudah terdapat emulator untuk testing dan MDS-CS(Mobile Data System-Connection Service) Simulator. MDS-CS Simulator ini bisa digunakan sebagai gateway antara simulator dan local resource.

## JSON

JSON (dilafalkan "Jason"), singkatan dari *JavaScript Object Notation*, adalah suatu format ringkas pertukaran data komputer. Formatnya berbasis teks dan terbaca-manusia serta digunakan untuk merepresentasikan struktur data sederhana dan larik asosiatif (disebut objek). Format JSON sering digunakan untuk mentransmisikan data terstruktur melalui suatu koneksi jaringan pada suatu proses yang disebut serialisasi. Aplikasi utamanya adalah pada pemrograman aplikasi web AJAX dengan berperan sebagai alternatif terhadap penggunaan tradisional format XML.

Walaupun JSON didasarkan pada subset bahasa pemrograman JavaScript (secara spesifik, edisi ketiga standar ECMA-262, Desember 1999) dan umumnya digunakan dengan bahasa tersebut, JSON dianggap sebagai format data yang tak tergantung pada suatu bahasa. Kode untuk pengolahan dan pembuatan data JSON telah tersedia untuk banyak jenis bahasa pemrograman. Situs json.org menyediakan daftar komprehensif pengikatan JSON yang tersedia, disusun menurut bahasa.

Format JSON dispesifikasikan di RFC 4627 oleh Douglas Crockford. Tipe media Internet resmi JSON adalah application/json sedangkan ekstensi berkasnya adalah .json.

## SQLite Relational Database

SQLite merupakan sebuah sistem manajemen basisdata relasional yang bersifat ACID-compliant dan memiliki ukuran pustaka kode yang relatif kecil, ditulis dalam bahasa C. SQLite merupakan proyek yang bersifat public domain yang dikerjakan oleh D. Richard Hipp.

Inti SQLite bukanlah sebuah sistem yang mandiri yang berkomunikasi dengan sebuah program, melainkan sebagai bagian integral dari sebuah program secara keseluruhan. Sehingga protokol komunikasi utama yang digunakan adalah melalui pemanggilan API secara langsung melalui bahasa pemrograman. Mekanisme seperti ini tentunya membawa keuntungan karena dapat mereduksi *overhead*, *latency times*, dan secara keseluruhan lebih sederhana. Seluruh elemen basisdata (definisi data, tabel, indeks, dan data) disimpan sebagai sebuah file. Kesederhanaan dari sisi disain tersebut bisa diraih dengan cara mengunci keseluruhan file basis data pada saat sebuah transaksi dimulai.

Pustaka SQLite mengimplementasikan hampir seluruh elemen-elemen standar yang berlaku pada ***SQL***-92, termasuk transaksi yang bersifat atomic, konsistensi basisdata, isolasi, dan durabilitas (dalam bahasa inggris lebih sering disebut ***ACID***), trigger, dan kueri-kueri yang kompleks. Tidak ada pengecekan tipe sehingga data bisa dientrikan dalam bentuk string untuk sebuah kolom bertipe integer. Beberapa kalangan melihat hal ini sebagai sebuah inovasi yang menambah nilai guna dari sebuah basisdata, utamanya ketika digunakan dalam bahasa pemrograman berbasis script (***PHP***, ***Perl***), sementara kalangan lain melihat hal tersebut sebagai sebuah kekurangan.

Beberapa proses ataupun thread dapat berjalan secara bersamaan dan mengakses basisdata yang sama tanpa mengalami masalah. Hal ini disebabkan karena akes baca data dilakukan secara paralel. Sementara itu akses tulis data hanya bisa dilakukan jika tidak ada proses tulis lain yang sedang dilakukan; jika tidak, proses tulis tersebut akan gagal dan mengembalikan kode kesalahan (atau bisa juga secara otomatis akan mencobanya kembali sampai sejumlah nilai waktu yang ditentukan habis). Hanya saja ketika sebuah tabel temporer dibuat, mekanisme penguncian pada proses multithread akan menyebabkan masalah. Update yang terkini (versi 3.3.4) dikatakan telah memperbaiki masalah ini.

# ANALISIS DAN PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan sistem perangkat lunak (*software*) agar dapat mencapai tujuan dari Kerja Praktek. Perancangan yang dibuat meliputi perancangan data dan perancangan aplikasi.

## Analisa Kebutuhan

Secara umum aplikasi yang akan dibangun memiliki kebutuhan sebagai berikut.

1. **Baca, simpan, dan delete data kontak milik device**

Dapat membaca, menambahkan, mengedit, dan menghapus data kontak milik Blackberry *device* dan menambahkannya ke aplikasi atau menghapusnya dari aplikasi. Data kontak didapatkan dari mengimport data kontak di *device*, dari sistem utama, dan dari aplikasi ini sendiri.

Tabel .1 Field-field yang mampu ditampung data kontak Blackberry.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Jumlah Per Field** | **Atribut yang Disupport** | **Keterangan** |
| Contact.  BIRTHDAY | 1 | PIMItem.  ATTR\_NONE |  |
| Contact.NAME | 1 | PIMItem.  ATTR\_NONE | Terdapat beberapa field, antara lain Contact.  NAME\_  GIVEN  (first name) dan Contact.  NAME\_  FAMILY  (last name) |
|  |  |  |  |
| **Field** | **Jumlah Per Field** | **Atribut yang Disupport** | **Keterangan** |
| Contact.TITLE | 1 | PIMItem.  ATTR\_NONE |  |
| Contact.ORG | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.  ORG\_YOMI | 1 | PIMItem.  ATTR\_NONE |  |
| Contact.ADDR | 1 | Contact.  ATTR\_WORK,Contact.ATTR\_HOME |  |
| Contact.NOTE | 1 | PIMItem.  ATTR\_NONE |  |
| Contact.TEL | 8 | Contact.  ATTR\_WORK,Contact.ATTR\_HOME, Contact.ATTR\_  MOBILE, Contact.  ATTR\_PAGER, Contact.ATTR\_FAX, Contact.  ATTR\_OTHER, Contact.  ATTR\_HOME2, Contact.  ATTR\_WORK2 |  |
| Contact.EMAIL | 3 | PIMItem.  ATTR\_NONE |  |
| Contact.PHOTO | 1 | PIMItem.  ATTR\_NONE |  |
| Contact.UID | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.  ANNIVERSARY | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.USER1 | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.USER2 | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.USER3 | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.USER4 | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.PIN | 1 | PIMItem.  ATTR\_NONE |  |
| BlackBerryContact.DCID | 1 | PIMItem.  ATTR\_NONE |  |

Data kontak Blackberry memilik batasan sebagaimana yang ada di Tabel 4.1. Data kontak Blackberry memiliki 18 *field* utama, yang masing memiliki batasan jumlah yang umumnya adalah satu dengan pengecualian dua *field* yaitu EMAIL dan TEL.

1. **Baca, simpan, dan delete data *reminder*/event milik device**

Dapat membaca, menambahkan, mengubah, dan menghapus data kontak milik Blackberry *device* dan menambahkannya ke aplikasi atau menghapusnya dari aplikasi. Nantinya apabila *reminder* telah dilakukan(status = 1) maka akan dikirim notifikasi ke sistem utama.

1. **Membuat phonecall dari aplikasi**

Dapat membuat *phonecall* dari *reminder* yang tercatat di aplikasi, sesuai dengan kontak yang terhubung dengan *reminder* yang ada. Fungsi ini akan memanggil aplikasi *phonecall* bawaan dari Blackberry dan mengeset nomer yang dihubungi.

1. **Membaca status phonecall dari aplikasi**

Membaca status dari *phonecall* yang dibuat, bila statusnya bukan *answered*  maka *reminder* belum diberi status selesai.

1. **Membuat email sesuai dengan template dari aplikasi**

Dapat membuat *email* dari *reminder* yang tercatat di aplikasi, sesuai dengan kontak yang terhubung dengan *reminder* yang ada. Fungsi ini akan memanggil aplikasi *email* bawaan dari Blackberry dan mengeset alamat *email* yang dituju, *subjectemail* dan isi *email*.

1. **Membaca status email yang dikirim**

Dapat membaca status *email* yang ada di *emailfolder* dan memeriksa status dari *email* yang di kirim.

1. **Membuat SQLite database**

Dapat membuat database berbasis SQLite, fungsi ini membutuhkan *device* yang mendukung memori eksternal, karena tidak bisa dibuat di memory internal. Pembuatan SQLite database dilakukan saat aplikasi pertama kali diinisialisasi.

1. **Membaca, menambahkan, dan menghapus data di SQLite database**

Dapat membaca,menambahkan dan menghapus data di SQLite database sesuai dengan bentuk tabel yang digunakan.

1. **Membuat koneksi ke sistem utama**

Mengkoneksikan aplikasi ke sistem utama, guna mendapatkan atau mengirimkan data. Hal ini dilakukan di *background* dan berjalan otomatis sesuai jadwal yang ditentukan.

## Definisi Umum Aplikasi



Gambar ‎4.1 Diagram arsitektur aplikasi secara umum. Aplikasi yang dibuat terletak pada Blackberry unit milik sales. Aplikasi berhubungan dengan web aplikasi secara *realtime*.

Aplikasi ini dikembangkan difokuskan untuk para sales sebagai *front* dari perusahaan. Aplikasi ini bertujuan untuk memberikan perhatian kepada konsumen yang lebih melalui sales yang bertanggung jawab pada masing-masing konsumen.

Secara umum aplikasi yang dibangun adalah bagian dari sistem yang sudah ada sebelumnya yaitu sistem yang berupa aplikasi web untuk *CRM*(*Customer Responsibility Management*).

Hal ini bisa dilihat di Gambar 4.1 bahwa Blackberry Unit milik Sales terhubung dengan Web Application CRM. Hubungan yang ada berjalan dua arah karena tujuannya adalah untuk mensinkronisasikan data yang ada di sistem utama dan data yang ada di aplikasi.

Data yang terkirim antara dua entitas ini adalah data kontak dan data *reminder*. Dari data yang ada sales bisa menghubungi Pelanggan/Klien melalui telepon atau *email* sesuai jadwal yang telah didapat di dalam data *reminder*.

Ketika sales berhasil menghubungi, nantinya status dari *reminder* yang ada akan diupdate dan dikirim ke Web Aplikasi CRM. Sehingga kerja seorang sales bisa lebih dipertanggungjawabkan dan Klien merasa lebih dihargai dan diperhatikan.

## Perancangan Fungsionalitas Aplikasi

Di dalam aplikasi ini hanya terdapat dua aktor yaitu Web Aplikasi *CRM* dan sales. Tapi didalam kerja praktek ini yang dikerjakan hanya yang disisi sales karena Web Aplikasi *CRM*sudah dikerjakan oleh pegawai di PT Kriyamuda Indonesia

### Fungsionalitas Aplikasi

Berikut ini adalah penjelasan masing-masing *usecase* yang ada di gambar 4.2.

****

Gambar ‎4.2 UseCase Diagram dari aplikasi yang dibuat. Terdiri dari dua aktor. Tiap *usecase* berasal gabungan dari kebutuhan umum aplikasi.

* + - 1. **Menambah data kontak**

Aplikasi ini bisa menambah data kontak konsumen baru yang belum ada di sistem utama, aplikasi, dan *phonebook*.

* + - 1. **Mengimport data kontak**

Aplikasi ini bisa menambah data kontak klien dari *phonebook* yang sudah ada atau dari Web Aplikasi CRM.

* + - 1. **Mengupdate *reminder***

Reminder diupdate statusnya ketika dilakukan dengan telpon atau *email* yang memiliki status berhasil.

* + - 1. **Menelpon**

Melakukan panggilan untuk mengupdate status *reminder*, hal ini dilakukan dengan fitur yang ada di Blackberry.

* + - 1. **Mengirim e-mail**

Melakukan pengiriman *email* untuk mengupdate status *reminder*, hal ini dilakukan dengan fitur yang ada di Blackberry.

* + - 1. **Sinkronisasi data kontak**

Data kontak disinkronisasikan antara unit dan sistem utama. Jika ada data baru maka akan ditambahkan ke masing-masing.

* + - 1. **Sinkronisasi data *reminder***

Data *reminder* disinkronisasikan antara unit dan sistem utama. Jika ada status baru maka akan ditambahkan ke database masing-masing.

### Cara Kerja Aplikasi

Ada dua aktifitas utama dalam aplikasi yaitu sinkronisasi kontak dan sinkronisasi *reminder*.

Aktifitas Sinkronisasi Kontak yang terlihat di Gambar 4.3 ini berjalan otomatis di *background* meskipun dalam diagram terlihat adanya aktor sales. Jadi disini sales hanya tinggal mengaktifkan Blackberry unitnya. Berikut ini adalah langkah-langkah yang terjadi di dalam aktifitas ini.

****

Gambar ‎4.3 Aktifitas Diagram Sinkronisasi Kontak. Terdiri dari 2 *swimlane* yang merepresantasikan masing aktor.

**Meminta Sinkronisasi Kontak**

Pada bagian ini aplikasi pada waktu-waktu tertentu akan mengirim permintaan untuk sinkronisasi data kontak.

**Kirim Kontak**

Dari permintaan yang ada, maka dibuatlah List Kontak yang dibutuhkan dan dikirim ke aplikasi.

**Cek List Kontak**

Dari semua data di list kontak yang diterima, maka dicocokkan datanya dengan yang ada dan bila ada perbedaan akan diurus sesuai dengan yang ditentukan**.**

**Kirim Kontak Baru**

Apabila di unit ada kontak baru dan belum ada di sistem utama, maka data kontak tersebuta akan dikirimkan.

**Simpan Kontak Baru**

Apabila di unit/sistem ada kiriman data kontak baru, maka data itu akan otomatis disimpan di database.

Aktifitas yang selanjutnya adalah sinkronisasi data *reminder*.Aktifitas Sinkronisasi *Reminder* yang terlihat di Gambar 4.4 ini berjalan semi otomatis, karena untuk sinkronisasi datanya aktifitas ini berjalan otomatis pada waktu tertentu tapi ketika melakukan *reminder* maka disini sales harus mengerjakannya secara manual. Berikut ini adalah langkah-langkah yang terjadi di aktifitas ini.

****

Gambar ‎4.4 Aktifitas Diagram Sinkronisasi *Reminder*. Terdiri dari 2 *swimlane* yang merepresantasikan masing aktor.

1. **Meminta Sinkronisasi *reminder***

Pada bagian ini aplikasi pada waktu-waktu tertentu akan mengirim permintaan untuk sinkronisasi data *reminder*.

1. **Kirim *reminder*.**

Dari permintaan yang ada, maka dibuatlah List *Reminder* yang dibutuhkan dan dikirim ke aplikasi.

1. **Simpan Reminder**

Simpan *reminder* di Blackberry unit dan di database aplikasi.

1. **Lakukan *reminder* lewat telepon**

Menggunakan fitur telepon pada Blackberry unit untuk melakukan *reminder*. Proses ini dianggap gagal sebelum telepon memiliki status *answered*.

1. **Lakukan *reminder* lewat email**

Menggunakan fitur *email* pada Blackberry unit untuk melakukan *reminder*. Proses ini dianggap gagal sebelum email memiliki status *delivered*.

1. **Kirim status *reminder***

Kirim status *reminder* ke sistem utama yang otomatis berjalan di waktu-waktu tertentu.

1. **Update status *reminder***

Update status *reminder*yang dilakukan menjadi sudah dilaksanakan.

## Perancangan Data

Di dalam database aplikasi ini hanya terdapat 2 tabel, yaitu tabel data kontak dan tabel data *reminder*. Masing-masing tabel tersebut menyimpan data-data yang diperlukan oleh aplikasi untuk berjalan dengan baik. Sedangkan detail dari tiap data dimasukkan dalam sistem yang telah ada di Blackberry.

Database dari aplikasi yang dikembangkan memiliki struktur yang bisa dilihat di Gambar 4.5.

****

**(a)**

****

**(b)**

Gambar ‎4.5 (a) CDM Diagram, terdiri dari dua tabel. Tiap ContactData setidaknya memiliki 1 atau lebih ReminderData.(b)PDM Diagram, terdiri dari dua tabel. *Primary Key* dari ContactData menjadi *Foreign Key* di ReminderData.

Berikut ini akan dijelaskan secara detail isi dan kegunaan masing-masing tabel pada Gambar 4.5.

* + - 1. **Tabel ContactData**

Digunakan untuk menyimpan id kontak yang terhubung dengan system utama. Terdiri dari 2 kolom yaitu UID\_Contactdan ID\_SYS\_Contact.

Tabel .2 Detail dari tabel ContactData. Menyesuaikan dengan PDM Diagram pada Gambar 4.5(b).

|  |  |  |
| --- | --- | --- |
| **Kolom** | **Tipe** | **Keterangan** |
| UID\_Contact | varchar (20) | Id kontak di *device*, berasal dari UID Contact yang ada di sistem Blackberry |
| ID\_SYS\_Contact | varchar(20) | Id kontak di sistem utama |

* + - 1. **Tabel ReminderData**

Digunakan untuk menyimpan id *reminder* yang terhubung dengan system utama. Terdiri dari 4 kolom yaitu UID\_Reminder, UID\_Contact, ID\_SYS\_Reminder, dan Status.

Tabel .3 Detail dari tabel ReminderData. Menyesuaikan dengan PDM diagram pada Gambar 4.5(b).

|  |  |  |
| --- | --- | --- |
| **Kolom** | **Tipe** | **Keterangan** |
| UID\_Reminder | varchar (20) | Id *reminder* di *device*, berasal dari UID Event yang ada di sistem Blackberry |
| UID\_Contact | varchar (20) | Id kontak di *device* yang berhubungan dengan aktifitas *reminder* |
| ID\_SYS\_  Reminder | varchar(20) | Id *reminder* di sistem utama |
| Status | int(11) | Menyimpan status dari aktifitas |

## Perancangan User Interface

Desain antarmuka yang menyusun aplikasi bisa dilihat mulai sub bagian 4.5.1 hingga 4.5.3.

### User Interface Menu Utama

(a)Menu

(b)Option

(c)Context Menu

Contact

Reminder

About

Gambar ‎4.6 UI dari Menu Utama. Terdapat menu tiga pilihan utama yaitu “About”, “Contact”, dan “Reminder”.

Terdapat 3 (tiga) bagian utama yang menyusun antarmuka menu utama seperti Gambar 4.6. Berikut ini adalah penjelasan untuk masing-masing bagian yang ada di Gambar 4.6.

* 1. ***Menu***

Merupakan bagian yang memuat menu utama aplikasi. Terdiri dari “About”, “Contact”, dan “Reminder”. Konten *Option* berubah sesuai dengan menu yang dipilih.

* 1. ***Option***

Bagian ini digunakan untuk memuat aksi-aksi yang bisa dilakukan. Ada aksi yang ketika dipilih langsung melakukan sebuah fungsi, ada juga yang membuka submenu lain yang tampak pada Gambar 4.9.

* 1. ***Context Menu***

Bagian ini digunakan untuk memuat aksi-aksi cepat yang berbeda tiap antarmuka dan merupakan bagian yang selalu ada di setiap antarmuka. Pada bagian ini yang ada hanya menu untuk menutup aplikasi ke bagian *background*.

### User Interface Tambah Kontak

(a)Header

(b)Form data

Nama depan :

Nama belakang :

.

.

Alamat :

(c)Context Menu

Gambar ‎4.7 UI dari tambah kontak. Hampir sama seperti pada UI tambah kontak di Blackberry. Terdiri dari koleksi *field-field* yang bisa diisi.

Terdapat 3 (tiga) bagian utama yang menyusun antarmuka Tambah Kontak pada gambar 4.7. Berikut ini adalah penjelasan untuk masing-masing bagian yang ada pada Gambar 4.7.

1. ***Header***

Merupakan bagian yang menunjukkan nama *UI Screen* yang sedang dibuka. Pada *UI*ini akan terlihat *header* yang bertuliskan ”Add Contact”.

1. ***Form data***

Bagian ini digunakan untuk memuat form yang bisa diisi.*Field-field*yang ada adalah nama depan, nama belakang, alamat rumah, telepon rumah, alamat kantor, telepon kantor, telepon pribadi, dll, sesuai dengan Tabel 4.1.

1. ***Context Menu***

Bagian ini digunakan untuk memuat aksi-aksi cepat yang berbeda tiap antarmuka dan merupakan bagian yang selalu ada di setiap antarmuka. Pada UI ini terdapat tambahan menu untuk menyimpan data kontak.

### User Interface ContactList Screen

(a)Header

(b)Name List

(c)Context Menu

Gambar ‎4.8 UI ContactList Screen, dimana perbedaannya terletak pada isi *command* di Context Menu.

Terdapat 3 (tiga) bagian utama yang menyusun antarmuka ContactList*Screen* pada Gambar 4.8. ContactList *Screen* adalah sebuah *UI Screen* yang menjadi dasar dari *submenu* “Import Contact From Book” dan “Application Contact”.Berikut ini adalah penjelasan untuk masing-masing bagian pada Gambar 4.8.

1. ***Header***

Merupakan bagian yang menunjukkan nama *UI Screen* yang sedang dibuka. Berganti-ganti sesuai dengan *submenu* yang dibuka yang pada konteks ini adalah ”Import Contact From Book” dan ”Application Contact List”.

1. ***Name List***

Bagian ini digunakan untuk memuatnama dari kontak yang ditampilkan.

1. ***Context Menu***

Bagian ini digunakan untuk memuat aksi-aksi cepat yang berbeda tiap antarmuka dan merupakan bagian yang selalu ada di setiap antarmuka. Umumnya *contextmenu* tambahan yang ada adalah menu*Back* untuk kembali ke *UIScreen* sebelumnya.

### User Interface Sub MenuScreen

(a)Header

(b)Option

(c)Context Menu

Gambar ‎4.9 UI Sub Menu Screen, dimana perbedaannya antara *submenu*terletak pada isi *command* di Context Menu.

Terdapat 3 (tiga) bagian utama yang menyusun antarmuka Sub Menu *Screen* pada Gambar 4.9. Berikut ini adalah penjelasan untuk masing-masing bagian pada Gambar 4.9.

1. ***Header***

Merupakan bagian yang menunjukkan nama *UI Screen* yang sedang dibuka. Berganti-ganti sesuai dengan *submenu* yang dibuka.

1. ***Option***

Bagian ini digunakan untuk memuat aksi-aksi yang bisa dilakukan di *submenu* yang sedang dibuka.

1. ***Context Menu***

Bagian ini digunakan untuk memuat aksi-aksi cepat yang berbeda tiap antarmuka dan merupakan bagian yang selalu ada di setiap antarmuka. Umumnya *contextmenu* tambahan yang ada adalah menu*Back* untuk kembali ke *UIScreen* sebelumnya.

***[Halaman ini sengaja dikosongkan]***

# IMPLEMENTASI

Bab ini akan menguraikan implementasi perangkat lunak yang meliputi algoritma dan kode program yang terdapat dalam perangkat lunak. Pada tahap implementasi dari tiap fungsi akan dijelaskan mengenai parameter masukan, keluaran,atau beberapa keterangan yang berhubungan dengan program dan teori.

## Memanipulasi Data Kontak

Fungsional memanipulasi kontak diwakili dengan kelas ContactData.javayang di dalamnya terdiri dari kelas-kelas yang berkerja untuk menambah, menghapus, mengubah, mengambil data kontak konsumen yang ada di sistem utama, aplikasi, dan *phonebook*.

Nama File : ContactData.java

Isi File : Kelas ContactData, kelas CBuilder, kelas ContactBuilder, kelas ContactList.

Di dalam kelas ContactData terdapat empat kelas yang memiliki peran yang berbeda-beda. Kelas ContactData adalah kelas yang berperan menyimpan variabel-variabelpenting pada kontak. Objek dari kelas ContactData akan diinisialisasikan dalam kelas abstrak CBuilder. Sedangkan Kelas ContactBuilder adalah kelas yang melakukan penyetelan data kontak seperti penambahan nama kontak, penambahan PIN Blackberry, dan sebagainya. Untuk kelas ContactList berperan menampilkan data-data yang sudah disetel seperti nama, email, PIN dan sebagainya. Implementasi kelas-kelas tersebut dijabarkan dalam operasi-operasi yang terdapat pada subbab selanjutnya.

### Kelas ContactData

Untuk menyimpan data-data semisal nama, email, alamat dari satu nomor kontak, diperlukan variabel-variabel yang menampungnya. Variabel-variabeltersebut terdapat dalam kelas ContactData. Variabel-variabel itu tergambar pada tabel 5.1.

|  |  |
| --- | --- |
| 1 | public final class ContactData extends BaseContactData{ |
| 2 | private String []names = null; |
| 3 | private String phoneNumber\_Mobile = ""; |
| 4 | private String phoneNumber\_Home = ""; |
| 5 | private String phoneNumber\_Work = ""; |
| 6 | private String email = ""; |
| 7 | private String blackberryPin = ""; |
| 8 | private long birthdayDate = 0; |
| 9 | private String []address\_Home = null; |
| 10 | private String []address\_Work = null; |
| 11 |  |
| 12 | public ContactData() { |
| 13 | // TODO Auto-generated constructor stub |
| 14 | } |
| 15 |  |
| 16 | public static ContactBuilder createContactBuilder(){ |
| 17 | return new ContactBuilder(); |
| 18 | } |
| 19 | public static ContactList createContactList(){ |
| 20 | return new ContactList(); |
| 21 | } |

Gambar .1 Tabel kode kelas ContactData

Pada gambar 5.1 baris kedua hingga baris kesepuluh adalah variabel yang menampung data kontak. Variabel names untuk menyimpan nama kontak, variabel phoneNumber\_Mobile untuk menyimpan nomor handphone, phoneNumber\_Home untuk menyimpan nomor telepon rumah, phoneNumber\_Work untuk menyimpan nomor telepon kantor, variabel email untuk menyimpan alamat email, blackberryPin untuk menyimpan PIN dari blackberry pelanggan, address\_Home untuk menyimpan alamat rumah, dan yang terakhir address\_Work untuk menyimpan alamat kantor. Sementara dua fungsi pada baris 16 dan 19 berfungsi membuat kelas ContactBuilder dan ContactList dimana masing-masing kelas tersebut akan dijelaskan pada subbab 5.1.3 dan 5.1.4.

### Kelas CBuilder

Kelas CBuilder adalah kelas abstrak yang berperan dalam penyimpanan objek dari kontak. Sehingga di dalam kelas CBuilder sendiri hanya terdapat variabel dan fungsi kosong.

|  |  |
| --- | --- |
| 1 | private static abstract class CBuilder{ |
| 2 | protected final ContactData contactData = new ContactData(); |
| 3 | protected boolean done; |
| 4 | protected boolean isAbleAccessContactList; |
| 5 | protected BlackBerryContactList contactList; |
| 6 | protected BlackBerryContact newContact; |
| 7 |  |
| 8 | public CBuilder(){} |
| 9 | protected abstract void check(); |
| 10 | protected abstract void checkContactList(); |
| 11 | } |
| 12 |  |

Gambar .2 Tampilan souce code implementasi kelas CBuilder

Pada gambar 5.2 variabel yang digunakan untuk menyimpan objek dari kelas ContactData terdapat pada baris kedua. Variabel ini nantinya akan menjadi variabel yang akan digunakan ketika aplikasi membutuhkan data kontak berdasar ID tertentu.

Dibaris ketiga terdapat variabel done yang akan mengembalikan nilai true atau false ketika suatu kontak telah dicek. Untuk mengeceknya dipanggil fungsi yang terdapat pada baris kesembilan.

Baris keempat adalah variabel Boolean yang berfungsi mengecek apakah suatu data kontak pada perangkat Blackberry dapat diakses atau tidak. Untuk melakukan pengecekan dipanggil fungsi checkContactList() pada baris kesepuluh.

Sementara pada baris keenam terdapat objek dari kelas bawaan Blackberry yang berperan mengakses data kontak yang tersimpan di perangkat Blackberry. Kumpulan dari kontak-kontak tersebut akan disimpan dalam suatu objek list. Dimana objek list itu ditulis pada baris kelima pada gambar 5.2

### Kelas ContactBuilder

Kelas ContactBuilder akan mengimplementasikan fungsi-fungsi atau operasi-operasi dari kelas abstrak CBuilder. Selain itu di dalam kelas ini juga akan ditambahkan operasi-operasi lain yang berperan memanipulasi data kontak.

#### Fungsi inisialisasi variabel

Inti dari kelasContactBuilder ini adalah untuk menginisialisasikan variabel sebelum digunakan. Ditambah dengan mencetak exception jika terjadi kesalahan.

Pada gambar 5.3 terdapat constructor dari kelas ContactBuilder dimana dalam baris kedua dibuat objek PIM Blackberry. Setelah objek PIM dibuat kemudian dilakukan try-catch pada baris empat hingga sepuluh. Dimana di dalam *syntax*try tepatnya dibaris empat dan enam terdapat kode untuk melakukan inisilisasi variabel-variabel kontak yang diambil dari kelas abstrak CBuilder. Sementara pada baris kelima, variabel newContact diinisialisasi dengan kontak baru.

|  |  |
| --- | --- |
| 1 | private ContactBuilder(){ |
| 2 | PIM pimInstance = PIM.getInstance(); |
| 3 | try{ |
| 4 | contactList = (BlackBerryContactList) pimInstance.openPIMList(PIM.CONTACT\_LIST, PIM.READ\_WRITE); |
| 5 | newContact = (BlackBerryContact) contactList.createContact(); |
| 6 | isAbleAccessContactList = true; |
| 7 | } |
| 8 | catch(PIMException e){ |
| 9 | // TODO Auto-generated catch block |
| 10 | e.printStackTrace(); |
| 11 | } |
| 12 | } |

Gambar .3 Kode fungsi contructor pada kelas ContactBuilder

#### Fungsi Pengecekan

Untuk melakukan pengecekan kontak sudah dibuat dan juga pengecekan daftar kontak yang tersimpan pada Blackberry, perlu dipanggillah fungsi check() dan checkContactList() yang sudah didefinisikan sebelumnya melalui kelas abstrak CBuilder. Kedua fungsi tersebut diimplementasikan dalam kelas ContactBuilder.

Pengecekan kontak sebenarnya hanya melakukan perbandingan kondisi dua variabel done dan isAbleAccessContactList yang sudah dibuat di kelas CBuilder. Kedua variabel tersebut dicek di gambar 5.4 baris tiga dan tujuh. Dimana untuk variabel pengecekan isAbleAccessContactList akan mengeluarkan IllegalStateExceptionapabila bernilai false karena aplikasi tidak dapat mengakses kontak dan akan mengeluarkan IllegalArgumentException apabila variabel done bernilai true.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | protected void check(){ |
| 3 | if(done) |
| 4 | throw new IllegalArgumentException  ("Do use other builder to create new instance"); |
| 5 | } |
| 6 | protected void checkContactList(){ |
| 7 | if(!isAbleAccessContactList) |
| 8 | throw new IllegalStateException  ("Unable to access contact list"); |
| 9 | } |
| 10 | ... |

Gambar .4 Kode fungsi pengecekan kontak

#### Fungsi Penyimpanan Kontak

Untuk memenuhi kebutuhan penyimpanan kontak, maka diperlukan satu fungsi khusus bernama save(). Inti kerja dari fungsi ini adalah menyimpan elemen-elemen data dari satu nomor kontak pada Blackberry yang terdiri dari nama, alamat, email, nomor telepon dan sebagainya.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public ContactData save(){ |
| 3 | try { |
| 4 | newContact.addStringArray  (BlackBerryContact.NAME, BlackBerryContact.ATTR\_NONE, contactData.names); |
| 5 | newContact.addString(BlackBerryContact.TEL, BlackBerryContact.ATTR\_MOBILE, contactData.phoneNumber\_Mobile); |
| 6 | newContact.addString(BlackBerryContact.TEL, BlackBerryContact.ATTR\_WORK, contactData.phoneNumber\_Work); |
| 7 | newContact.addString(BlackBerryContact.TEL, BlackBerryContact.ATTR\_HOME, contactData.phoneNumber\_Home); |
| 8 | newContact.addString(BlackBerryContact.EMAIL, BlackBerryContact.ATTR\_NONE, contactData.email); |
| 9 | if(contactData.birthdayDate !=0) newContact.addDate(BlackBerryContact.BIRTHDAY, BlackBerryContact.ATTR\_NONE, contactData.birthdayDate); |
| 10 | newContact.addString(BlackBerryContact.PIN, BlackBerryContact.ATTR\_NONE, contactData.blackberryPin); |
| 11 | if(contactData.address\_Work != null) newContact.addStringArray (BlackBerryContact.ADDR, BlackBerryContact.ATTR\_WORK, contactData.address\_Work); |
| 12 | if(contactData.address\_Home!=null) newContact.addStringArray (BlackBerryContact.ADDR, BlackBerryContact.ATTR\_HOME, contactData.address\_Home); |
| 13 | newContact.commit(); |
| 14 | done = true; |
| 15 | contactData.contactUID = newContact.getString  (BlackBerryContact.UID, 0); |
| 16 | PersistenceManager.getCurrentInstance() .insertContactData(contactData.contactUID); |
| 17 | } catch (PIMException e) { |
| 18 | // TODO Auto-generated catch block |
| 19 | e.printStackTrace(); |
| 20 | } |
| 21 | return contactData; |
| 22 | } |
| 23 | ... |

Gambar .5 Kode fungsi penyimpanan kontak

Pada gambar 5.5 baris empat hingga dua belas tertulis kode untuk menyimpan elemen-elemen data pada kontak ke dalam variabel newContact yang telah dibuat sebelumnya. Pada baris empat, kode menyimpan nama. Baris kelima menyimpan nomor *handphone*. Baris keenam menyimpan nomor telepon kantor. Baris ketujuh menyimpan nomor telepon rumah. Baris kedelapan menyimpan alamat email. Baris kesembilan menyimpan tanggal lahir. Baris kesebelas menyimpan alamat kantor. Baris kedua belas menyimpan alamat rumah.

Setelah elemen-elemen kontak tersebut diletakkan ke dalam newContact, selanjutnya newContact memanggil fungsi commit() untuk menyimpannya ke dalam database seperti yang tertulis pada baris ketiga belas.

Baris keempat belas gambar 5.5 menginisialisasikan variabel done bernilai true untuk menunjukkan bahwa kontak telah tersimpan. Sementara pada baris kelima belas dan keenam belas variabel newContact perlu meminta UID pada Blackberry.

Setelah kode tersebut dijalankan tidak lupa dicantumkan exception apabila terjadi kesalahan. Kode tersebut diletakkan pada baris ketujuh belas hingga dua puluh.

#### Fungsi Penyetelan Tanggal Lahir

Pada penyetelan tanggal lahir pada kontak, inputan yang dibutuhkan adalah tanggal lahir, bulan, serta tahun. Inputan itu kemudian disimpan dalam satu variabel *temporary* dimana selanjutnya satu variabel *temporary* itu akan menjadi nilai dalam elemen kontak tanggal lahir.

Seperti terimplementasikan pada gambar 5.6, data masukan tanggal, bulan dan tahun tertulis pada baris ketiga. Selanjutnya disimpan ke dalam variabel temporary pada baris enam hingga sembilan. Setelah variabel *temporary* tersebut terisi, variabel itu akan menjadi masukan pada variabel contactData.birthdayDate pada baris ke sepuluh.

|  |  |
| --- | --- |
| 1 | ... |
| 2 |  |
| 3 | public ContactBuilder setBirthDay(int date, int month, int year){ |
| 4 | checkContactList(); |
| 5 | check(); |
| 6 | Calendartemp = Calendar.getInstance(TimeZone.getDefault()); |
| 7 | temp.set(Calendar.DATE, date); |
| 8 | temp.set(Calendar.MONTH, month-1); |
| 9 | temp.set(Calendar.YEAR, year); |
| 10 | contactData.birthdayDate = temp.getTime().getTime(); |
| 11 | return this; |
| 12 | } |
| 13 |  |
| 14 | ... |

Gambar .6 Implementasi fungsi penyetelan tanggal lahir

#### Fungsi Penyetelan PIN Blackberry

Untuk melakukan penyetelan Blackberry PIN dibutuhkan masukan data PIN Blackberry. Seperti yang dilakukan pada gambar 5.7 baris ketiga terdapat masukan PIN Blackberry berupa stringbbPIN. Masukan itu kemudian akan disimpan ke dalam variabel contactData pada baris keenam.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ContactBuilder setBlackBerryPin(String bbPIN){ |
| 4 | checkContactList(); |
| 5 | check(); |
| 6 | contactData.blackberryPin = bbPIN; |
| 7 | return this; |
| 8 | } |
| 9 | ... |
| 10 | ... |

Gambar .7 Kode fungsi penyetelan PIN Blackberry

#### Fungsi Penyetelan Email

Pada penyetelan email kontak dibutuhkan masukan data email. Seperti yang dilakukan pada gambar 5.8 baris ketiga terdapat masukan PIN Blackberry berupa string email. Masukan itu kemudian akan disimpan ke dalam variabel contactData pada baris keenam.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ContactBuilder setEmail(String email){ |
| 4 | checkContactList(); |
| 5 | check(); |
| 6 | contactData.email = email; |
| 7 | return this; |
| 8 | } |
| 9 |  |
| 10 | ... |
| 11 | ... |

Gambar .8 Kode fungsi untuk melakukan penyetelan email pada kontak

#### Fungsi Penyetelan Nama

Untuk melakukan penyetelan data nama terdapat dua masukan yang dibutuhkan yaitu nama depan dan nama belakang. Mengantisipasi apabila data yang diisi hanya nama depan saja, maka kita membutuhkan dua fungsi. Satu fungsi dengan dua masukan nama depan dan belakang serta satu fungsi lainnya dengan masukan nama depan saja.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ContactBuilder setName(String first){ |
| 4 | checkContactList(); |
| 5 | check(); |
| 6 | if(contactData.names == null)contactData.names = new String[contactList.stringArraySize(BlackBerryContact.NAME)]; |
| 7 | contactData.names[BlackBerryContact.NAME\_GIVEN] = first; |
| 8 | return this; |
| 9 | } |
| 10 | public ContactBuilder setName(String first, String last){ |
| 11 | checkContactList(); |
| 12 | check(); |
| 13 | if(contactData.names == null)contactData.names = new String[contactList.stringArraySize(BlackBerryContact.NAME)]; |
| 14 | contactData.names[BlackBerryContact.NAME\_GIVEN]=first; |
| 15 | contactData.names[BlackBerryContact.NAME\_FAMILY]= last; |
| 16 | return this; |
| 17 | } |
| 18 | ... |
| 19 | ... |

Gambar .9 Kode fungsi penyetelan nama

Dalam masing-masing fungsi memiliki alur logika yang sama yaitu terlebih dahulu mengecek apakah data nama yang terdapat pada variabel contactData bernilai null. Jika memenuhi maka kemudian elemen data nama pada contactData akan dibuatkan objek nama dari BlackberryContact.NAME. Nama itu selanjutnya akan diisi dengan masukan nama depan atau kedua-duanya. Logika tersebut diimplementasikan pada baris keenam untuk masukan nama depan saja dan baris ketiga belas untuk masukan nama depan dan nama belakang pada gambar 5.9.

#### Fungsi Penyetelan Nomor Telepon

Alur logika penyetelan nomor telepon adalah dengan memasukkan data masukan ke dalam elemen data kontak secara langsung. Alur logika ini berlaku untuk semua jenis nomor telepon baik nomor handphone, telepon rumah, ataupun kantor.

Seperti masukan-masukan pada fungsi penyetelan nomor telepon dalam gambar 5.10 baris ketiga, kesembilan, dan kelima belas, ketiga masukan tersebut selanjutnya akan langsung dimasukkan ke dalam contactData sesuai jenis elemen datanya. Misalnya untuk nomor telepon akan langsung dimasukkan ke dalam elemen data phoneNumber\_Home pada baris keenam.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ContactBuilder setPhone\_Home(String tel){ |
| 4 | checkContactList(); |
| 5 | check(); |
| 6 | contactData.phoneNumber\_Home = tel; |
| 7 | return this; |
| 8 | } |
| 9 | public ContactBuilder setPhone\_Mobile(String tel){ |
| 10 | checkContactList(); |
| 11 | check(); |
| 12 | contactData.phoneNumber\_Mobile = tel; |
| 13 | return this; |
| 14 | } |
| 15 | public ContactBuilder setPhone\_Work(String tel){ |
| 16 | checkContactList(); |
| 17 | check(); |
| 18 | contactData.phoneNumber\_Work = tel; |
| 19 | return this; |
| 20 | } |
| 21 | ... |
| 22 | ... |

Gambar .10 Fungsi-fungsi penyetelan nomor telepon

#### Fungsi Penyetelan Alamat

Untuk melakukan penyetelan alamat terdapat dua masukan yang dibutuhkan yaitu nama jalan dan nama kota. Untuk memenuhi kebutuhan pengisian dua alamat, yaitu alamat rumah dan alamat kantor maka dibutuhkan dua fungsi.

Masukan nama jalan dan nama kota akan dimasukkan langsung ke dalam data kontak seperti pada gambar 5.11 baris lima hingga ke tujuh untuk penyetelan alamat rumah. Sedangkan baris empat belas hingga tujuh belas untuk penyetelan alamat kantor.

|  |  |
| --- | --- |
| 1 | public ContactBuilder setWorkAddress(String street, String town){ |
| 2 | checkContactList(); |
| 3 | check(); |
| 4 | if(contactData.address\_Work == null)contactData.address\_Work = new String[contactList.stringArraySize(BlackBerryContact.ADDR)]; |
| 5 | contactData.address\_Work[BlackBerryContact.ADDR\_STREET]=street; |
| 6 | if(contactList.isSupportedArrayElement(BlackBerryContact.ADDR\_LOCALITY, BlackBerryContact.ADDR)){ |
| 7 | contactData.address\_Work[BlackBerryContact.ADDR\_LOCALITY]= town; |
| 8 | } |
| 9 | return this; |
| 10 | } |
| 11 | public ContactBuilder setHomeAddress(String street, String town){ |
| 12 | checkContactList(); |
| 13 | check(); |
| 14 | if(contactData.address\_Home == null)contactData.address\_Home = new String[contactList.stringArraySize(BlackBerryContact.ADDR)]; |
| 15 | contactData.address\_Home[BlackBerryContact.ADDR\_STREET]= street; |
| 16 | if(contactList.isSupportedArrayElement(BlackBerryContact.ADDR\_LOCALITY, BlackBerryContact.ADDR)){ |
| 17 | contactData.address\_Home[BlackBerryContact.ADDR\_LOCALITY]= town; |
| 18 | } |
| 19 | return this; |
| 20 | } |

Gambar .11 Kode fungsi – fungsi penyetelan alamat.

### Kelas ContactList

Kelas ini berfungsi menampilkan dan menghapus kontak yang ada. Di dalamnya terdapat dua kelas utama yaitu kelas deleteContactByUID()dan kelas getContactDataByUID() dimana keduanya berperan memenuhi kebutuhan fungsi yang tersebut.

Kelas menghapus kontak diimplementasikan oleh fungsi deleteContactByUID()yang memiliki masukan ID kontak. Di dalamnya terdapat kode untuk menampilkan daftar kontak yang dimaksud lalu menghapusnya. Implementasi logika tersebut terjabarkan dalam gambar 5.12 baris keempat. Dimana fungsi getContactList()didapat dari fungsi yang me-*return*  data contactList atau daftar kontak yang ada seperti pada gambar 5.13.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public void deleteContactByUID(String uid){ |
| 3 | try { |
| 4 | getContactList().removeContact(getContactList().getByUID(uid)); |
| 5 | } catch (PIMException e) { |
| 6 | // TODO Auto-generated catch block |
| 7 | e.printStackTrace(); |
| 8 | } |
| 9 | } |
| 10 | ... |
| 11 | ... |

Gambar .12 Implementasi kode penghapusan

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public BlackBerryContactList getContactList(){ |
| 3 | return contactList; |
| 4 | } |
| 5 | ... |
| 6 | ... |

Gambar .13 Kode yang Berfungsi Menampilkan Daftar Kontak

Sementara fungsi untuk menampilkan data kontak diimplementasikan oleh fungsi getContactDataByUID() dengan memasukan id kontak. Di dalam fungsi ini diperlukan try-catch berkali-kali untuk memastikan data yang akan ditampilkan benar-benar ada.

Ketika fungsi ini dijalankan yang pertama kali dibuat adalah mendapatkan objek kontak dari fungsi contactList()pada gambar 5.13 yang telah dijelaskan sebelumnya. Dari objek kontak itulah kemudian satu persatu akan ditampilkan elemen-elemen data pada kontak seperti nama, alamat, email dan sebagainya.

|  |  |
| --- | --- |
| 1 | public ContactData getContactDataByUID(String uid){ |
| 2 | try { |
| 3 | newContact = getContactList().getByUID(uid); |
| 4 | try{ |
| 5 | contactData.names = newContact.getStringArray(BlackBerryContact.NAME, 0); |
| 6 | }catch (Exception e) { |
| 7 | // TODO: handle exception |
| 8 | } |
| 9 | try{ |
| 10 | contactData.address\_Work = newContact.getStringArray(BlackBerryContact.ADDR, 0); |
| 11 | }catch (Exception e) { |
| 12 | // TODO: handle exception |
| 13 | } |
| 14 | try{ |
| 15 | contactData.address\_Home = newContact.getStringArray(BlackBerryContact.ADDR, 1); |
| 16 | }catch (Exception e) { |
| 17 | // TODO: handle exception |
| 18 | } |
| 19 | try{ |
| 20 | contactData.birthdayDate = newContact.getDate(BlackBerryContact.BIRTHDAY, 0); |
| 21 | }catch (Exception e) { |
| 22 | // TODO: handle exception |
| 23 | } |
| 24 | try{ |
| 25 | contactData.contactUID = newContact.getString(BlackBerryContact.UID, 0); |
| 26 | }catch (Exception e) { |
| 27 | // TODO: handle exception |
| 28 | } |
| 30 | try{ |
| 31 | contactData.email = newContact.getString(BlackBerryContact.EMAIL, 0); |
| 32 | }catch (Exception e) { |
| 33 | // TODO: handle exception |
| 34 | } |
| 35 | try{ |
| 36 | contactData.blackberryPin = newContact.getString(BlackBerryContact.PIN, 0); |
| 37 | }catch (Exception e) { |
| 38 | // TODO: handle exception |
| 39 | } |
| 40 | try{ |
| 41 | contactData.phoneNumber\_Mobile = newContact.getString(BlackBerryContact.TEL, 0); |
| 42 | }catch (Exception e) { |
| 43 | // TODO: handle exception |
| 44 | } |
| 45 | try{ |
| 46 | contactData.phoneNumber\_Work = newContact.getString(BlackBerryContact.TEL, 1); |
| 47 | }catch (Exception e) { |
| 48 | // TODO: handle exception |
| 49 | } |
| 50 | try { |
| 51 | contactData.phoneNumber\_Home = newContact.getString(BlackBerryContact.TEL, 2); |
| 52 | } catch (Exception e) { |
| 53 | // TODO: handle exception |
| 54 | } |
| 55 | } catch (PIMException e) { |
| 56 | // TODO Auto-generated catch block |
| 57 | e.printStackTrace(); |
| 58 | } |
| 59 | return contactData; |
| 60 | } |

Gambar .14 Kode fungsi mendapatkan kontak berdasar ID

Pada gambar 5.14 terjabarkan implementasi dari logika membaca data kontak. Baris kelima berperan membaca kontak nama. Baris kesepuluh dan kelima belas membaca alamat kantor dan rumah. Baris kedua puluh membaca tanggal lahir. Baris kedua puluh lima membaca id kontak. Baris 31 membaca email. Baris 36 membaca PIN Blackberry. Baris 41, 46 dan 51 membaca nomor telephone baik handphone, kantor maupun rumah.

## Memanipulasi Reminder

Fungsional memanipulasi *reminder* ini diimplementasikanoleh file ReminderData.java, yang didalamnya terdapat beberapa *inner class*yang berhubungan dengan kelas ReminderData sendiri.

Nama File : ReminderData.java

Isi : Kelas ReminderData, Kelas RBuilder, KelasReminderBuilder, Kelas ReminderList

Di dalam kelas ReminderData terdapat empat kelas yang memiliki peran yang berbeda-beda. Kelas ReminderData adalah kelas yang berperan menyimpan variabel-variabelpenting pada *reminder*. Objek dari kelas ReminderData akan diinisialisasikan dalam kelas abstrak RBuilder. Sedangkan Kelas ReminderBuilder adalah kelas yang melakukan penyetelan data *reminder* seperti penambahan keterangan *reminder* dan waktu pelaksanaan *reminder*. Untuk kelas ReminderList berperan mendapatkan data *reminder* yang dibuat oleh aplikasi. Implementasi kelas-kelas tersebut dijabarkan dalam operasi-operasi yang terdapat pada subbagian selanjutnya.

### Kelas ReminderData

Untuk menyimpan data-data semisal keterangan dan waktu *reminder*, diperlukan variabel-variabel yang menampungnya. Variabel-variabeltersebut terdapat dalam kelas ReminderData. Variabel-variabel itu tergambar pada gambar 5.15.

|  |  |
| --- | --- |
| 1 | public final class ReminderData extends BaseReminderData{ |
| 2 | private String summary = ""; |
| 3 | privatelong start = System.*currentTimeMillis*(); |
| 4 |  |
| 5 | public ReminderData() { |
| 6 | // TODO Auto-generated constructor stub |
| 7 | } |
| 8 | publiclong getStart() { |
| 9 | return start; |
| 10 | } |
| 11 | public String getSummary() { |
| 12 | return summary; |
| 13 | } |
| 14 | publicstatic ReminderBuilder createReminderBuilder(){ |
| 15 | returnnew ReminderBuilder(); |
| 16 | } |
| 18 | publicstatic ReminderList createReminderList(){ |
| 19 | returnnew ReminderList(); |
| 20 | } |

Gambar .15 Kode Kelas ReminderData

Pada Gambar 5.15 baris kedua hingga baris ketiga adalah variabel yang menyimpan keterangan *reminder*, variabel start untuk menyimpan waktu *reminder* aktif. Sementara dua fungsi pada baris ke-14 dan ke-18 berfungsi membuat kelas ReminderBuilder dan ReminderList dimana masing-masing kelas tersebut akan dijelaskan pada subbab 5.2.2 dan 5.2.3.

### Kelas RBuilder

Kelas RBuilder adalah kelas abstrak yang berperan dalam penyimpanan objek dari kontak. Sehingga di dalam kelas CBuilder sendiri hanya terdapat variabel dan fungsi kosong.

|  |  |
| --- | --- |
| 1 | privatestaticabstractclass RBuilder{ |
| 2 | protectedBlackBerryEventList eventList; |
| 3 | protectedBlackBerryEvent event; |
| 4 | protectedboolean \_done; |
| 5 | protectedboolean isAccessibleEventList; |
| 6 | protectedfinal ReminderData reminderData = new ReminderData(); |
| 7 | private RBuilder() {} |
| 8 | publicabstractvoid check(); |
| 9 | publicabstractvoid checkList(); |

Gambar .16 Kode Kelas RBuilder

Pada gambar 5.16 variabel yang digunakan untuk menyimpan objek dari kelas ReminderData terdapat pada baris keenam. Variabel ini nantinya akan menjadi variabel yang akan digunakan ketika aplikasi membutuhkan data *reminder* berdasar ID tertentu.

Dibaris keempat terdapat variabel done yang akan mengembalikan nilai true atau false ketika suatu *reminder* telah disimpan. Untuk mengeceknya dipanggil fungsi yang terdapat pada baris kedelapan.

Baris kelima adalah variabel Boolean yang berfungsi mengecek apakah suatu data *reminder* pada perangkat Blackberry dapat diakses atau tidak. Untuk melakukan pengecekan dipanggil fungsi checkList() pada baris kesembilan.

Sementara pada baris ketiga terdapat objek dari kelas bawaan Blackberry yang berperan mengakses data *reminder* yang tersimpan di perangkat Blackberry. Kumpulan dari kontak-kontak tersebut akan disimpan dalam suatu objek list. Dimana objek list itu ditulis pada baris kedua pada gambar 5.16.

### Kelas ReminderBuilder

Kelas ReminderBuilder akan mengimplementasikan fungsi-fungsi atau operasi-operasi dari kelas abstrak RBuilder. Selain itu di dalam kelas ini juga akan ditambahkan operasi-operasi lain yang berperan memanipulasi data *reminder*.

#### Fungsi Inisialisasi Variable

Inti dari fungsi ReminderBuilder ini untuk menginisialisasikan variabel sebelum digunakan. Ditambah dengan mencetak *exception* jika terjadi kesalahan.

Pada gambar 5.17 terdapat *constructor* dari kelas ReminderBuilder dimana dalam baris kedua dibuat objek PIM Blackberry. Setelah objek PIM dibuat kemudian dilakukan try-catch pada baris empat hingga sepuluh. Dimana di dalam *syntax*try tepatnya dibaris empat dan enam terdapat kode untuk melakukan inisilisasi variabel-variabel kontak yang diambil dari kelas abstrak RBuilder. Sementara pada baris kelima, variabel event diinisialisasi dengan *reminder* baru.

|  |  |
| --- | --- |
| 1 | private ReminderBuilder(){ |
| 2 | // TODO Auto-generated constructor stub |
| 3 | try { |
| 4 | eventList=(BlackBerryEventList)PIM.*getInstance*().openPIMList(PIM.*EVENT\_LIST*, PIM.*READ\_WRITE*); |
| 5 | event = (BlackBerryEvent) eventList.createEvent(); |
| 6 | isAccessibleEventList = true; |
| 7 | } catch (PIMException e) { |
| 8 | // TODO Auto-generated catch block |
| 9 | e.printStackTrace(); |
| 10 | } |
| 11 | } |

Gambar .17 Kode fungsi inisialisasi ReminderBuilder

#### Fungsi Pengecekan

Untuk melakukan pengecekan *reminder* sudah dibuat dan juga pengecekan daftar *reminder* yang tersimpan pada Blackberry, perlu dipanggillah fungsi check() dan checkList() yang sudah didefinisikan sebelumnya melalui kelas abstrak RBuilder. Kedua fungsi tersebut diimplementasikan dalam kelas ReminderBuilder.

Pengecekan *reminder* sebenarnya hanya melakukan perbandingan kondisi dua variabel done dan isAbleAccessContactList yang sudah dibuat di kelas RBuilder. Kedua variabel tersebut dicek di gambar 5.18 baris tiga dan tujuh. Dimana untuk variabel pengecekan isAbleAccessContactList akan mengeluarkan IllegalStateExceptionapabila bernilai false karena aplikasi tidak dapat mengakses kontak dan akan mengeluarkan IllegalArgumentException apabila variabel done bernilai true.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | protected void check(){ |
| 3 | if(done) |
| 4 | throw new IllegalArgumentException  ("Do use other builder to create new instance"); |
| 5 | } |
| 6 | protected void checkContactList(){ |
| 7 | if(!isAbleAccessContactList) |
| 8 | throw new IllegalStateException  ("Unable to access contact list"); |
| 9 | } |
| 10 | ... |

Gambar .18 Kode fungsi pengecekan reminder

#### Fungsi Penyimpanan Reminder

Untuk memenuhi kebutuhan penyimpanan kontak, maka diperlukan satu fungsi khusus bernama save(). Inti kerja dari fungsi ini adalah menyimpan elemen-elemen data dari satu *reminder* pada Blackberry yang keterangan dan waktu.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public ReminderData save(){ |
| 3 | try { |
| 4 | event.addString(BlackBerryEvent.SUMMARY, BlackBerryEvent.ATTR\_NONE, reminderData.summary); |
| 5 | event.addDate(BlackBerryEvent.START, BlackBerryEvent.ATTR\_NONE, reminderData.start); |
| 6 | event.addInt(BlackBerryEvent.ALARM, BlackBerryEvent.ATTR\_NONE, 300); |
| 7 | RepeatRule rule = new RepeatRule(); |
| 8 | rule.setInt(RepeatRule.FREQUENCY, RepeatRule.YEARLY); |
| 9 | event.setRepeat(rule); |
| 10 | event.commit(); |
| 11 | \_done = true; |
| 12 | reminderData.eventUID = event.getString(BlackBerryEvent.UID, 0); |
| 13 | PersistenceManager.getCurrentInstance().insertReminderData(reminderData.eventUID, reminderData.relatedContactUID); |
| 14 | } catch (PIMException e) { |
| 15 | // TODO Auto-generated catch block |
| 16 | e.printStackTrace(); |
| 17 | } |
| 18 | return reminderData; |
| 19 | } |
| 20 | ... |

Gambar .19 Kode fungsi penyimpanan reminder

Pada gambar 5.19 baris empat hingga sembilan tertulis kode untuk menyimpan elemen-elemen data pada kontak ke dalam variabel event yang telah dibuat sebelumnya. Pada baris empat kode menyimpan keterangan. Baris kelima menyimpan waktu. Baris keenam menyimpan lama alarm. Baris ketujuh hingga kesembilan menyimpan *repeat rule reminder*. Setelah elemen-elemen *reminder* tersebut diletakkan ke dalam variabelevent, selanjutnya event memanggil fungsi commit() untuk menyimpannya ke dalam database seperti yang tertulis pada baris ketiga belas.

Baris sebelas Tabel 5.19 menginisialisasikan variabel done bernilai true untuk menunjukkan bahwa kontak telah tersimpan. Sementara pada baris kedua belas variabel event perlu meminta UID pada Blackberry.

Setelah kode tersebut dijalankan tidak lupa dicantumkan *exception* apabila terjadi kesalahan. Kode tersebut diletakkan pada baris keempat belas hingga sembilanbelas.

#### Fungsi Penyimpanan Summary

Untuk melakukan penyetelan *summary* atau keterangan dibutuhkan masukan data keterangan berupa string. Seperti yang dilakukan pada gambar 5.20 baris ketiga terdapat masukan keterangan berupa stringsumm. Masukan itu kemudian akan disimpan ke dalam variabel reminderData pada baris keenam.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ReminderBuilder setSummary(String summ){ |
| 4 | checkList(); |
| 5 | check(); |
| 6 | reminderData.summary = summ; |
| 7 | returnthis; |
| 8 | } |
| 9 | ... |
| 10 | ... |

Gambar .20 Kode fungsi penyimpanan summary reminder

#### Fungsi Penyimpanan Waktu Reminder

Untuk melakukan penyetelan waktu *reminder*dibutuhkan masukan data tanggal, bulan, dan tahun berupa integer. Seperti yang dilakukan pada gambar 5.21 baris kedua terdapat masukan keterangan berupa intdate, month dan year. Masukan itu kemudian akan disimpan ke dalam variabel reminderData pada baris kesembilan yang sebelumnya diubah menjadi *millisecond* mulai dari baris lima hingga delapan.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public ReminderBuilder setStart(int date, int month, int year){ |
| 3 | checkList(); |
| 4 | check(); |
| 5 | Calendar temp = Calendar.*getInstance*(TimeZone.*getDefault*()); |
| 6 | temp.set(Calendar.*DATE*, date); |
| 7 | temp.set(Calendar.*MONTH*, month-1); |
| 8 | temp.set(Calendar.*YEAR*, year); |
| 9 | reminderData.start = temp.getTime().getTime(); |
| 10 | returnthis; |
| 11 | } |
| 12 | ... |

Gambar .21 Kode fungsi penyimpanan waktu reminder

#### Fungsi Penyimpanan Related Contact

Untuk melakukan penyetelan *related contact*dibutuhkan masukan data keterangan berupa string. Seperti yang dilakukan pada gambar 5.22 baris ketiga terdapat masukan keterangan berupa stringuid*contact*. Masukan itu kemudian akan disimpan ke dalam variabel reminderData pada baris keenam.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | ... |
| 3 | public ReminderBuilder setRelatedContact(String uid){ |
| 4 | checkList(); |
| 5 | check(); |
| 6 | reminderData.relatedContactUID = uid; |
| 7 | returnthis; |
| 8 | } |
| 9 | ... |
| 10 | ... |

Gambar .22 Kode fungsi penyimpanan related contact

### Kelas ReminderList

Kelas ReminderList akan mengimplementasikan fungsi-fungsi atau operasi-operasi dari kelas abstrak RBuilder. Selain itu di dalam kelas ini juga akan ditambahkan operasi-operasi lain yang berperan memanipulasi data *reminder*terutama untuk menghapus dan menampilkan.

#### Fungsi Mendapatkan List Reminder

Untuk mendapatkan *list* dari *reminder* tidak dibutuhkan masukan apapun karena sudah diinisialisasi di *constructor. Constructor* dari kelas ReminderList sama dengan kelas Reminder Builder yang tampak pada gambar 5.17. Fungsi ini hanya mengeluarkan data yang ada pada eventList seperti yang ada di gambar 5.23 baris tiga.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | publicBlackBerryEventList getEventList(){ |
| 3 | return eventList; |
| 4 | } |
| 5 | ... |

Gambar .23 Kode fungsi untuk mendapatkan list reminder

#### Fungsi Menghapus Reminder

Untuk melakukan penghapusan*reminder* dibutuhkan masukan data uid berupa string. Seperti yang dilakukan pada gambar 5.24 baris kedua terdapat masukan keterangan berupa stringuid*reminder*. Masukan itu kemudian akan digunakan untuk menghapus reminder pada baris keempat.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | publicvoid deleteEventByUID(String uid){ |
| 3 | try { |
| 4 | getEventList().removeEvent(getEventList().getByUID(uid)); |
| 5 | } catch (PIMException e) { |
| 6 | // TODO Auto-generated catch block |
| 7 | e.printStackTrace(); |
| 8 | } |
| 9 | } |
| 10 | ... |

Gambar .24 Kode fungsi untuk menghapus reminder

## Koneksi ke Jaringan Internet

Untuk melakukan koneksi ke jaringan internet kita membutuhkan satu kelas tersendiri yang memiliki variabel dan fungsi yang dapat memenuhi kebutuhan tersebut. Masukan yang dibutuhkan dari fungsi yang ada di dalamnya adalah alamat url dari data yang akan diakses lewat internet. Karena fungsi ini belum terintegrasi dengan aplikasi utama maka yang dilakukan adalah melakukan uji coba koneksi. Dalam kasus ini akan diuji coba bagaimana kelas yang dibuat mampu melakukan akses ke dalam jaringan.

Nama File : MyConnection.java

Isi File : fungsi MyConnection() dan fungsi OpenUrl()

Saat uji coba dijalankan, kelas ini membutuhkan *url* dari data yang diakses. Alamat *url* itu akan disimpan ke dalam satu variabel bernama \_url. Setelah \_url terisi dengan alamat, selanjutnya dipanggil fungsi openUrl() yang akan berusaha menghubungkan file uji coba dengan jaringan internet pada kasus ini halaman www.google.com.

|  |  |
| --- | --- |
| 1 | ... |
| 2 | public void openURL() throws JSONException{ |
| 3 | StreamConnection connection = null; |
| 4 | InputStream in = null; |
| 5 | StringBuffer buffer = new StringBuffer(); |
| 6 | try { |
| 7 | connection = (StreamConnection) Connector.open("http://localhost/test/sampledata.json"; |
| 8 | in = connection.openInputStream(); |
| 9 | byte[] buf = new byte[4096]; |
| 10 | int nRead = in.read(buf); |
| 11 | while (nRead > 0) { |
| 12 | buffer.append(new String(buf,0,nRead)); |
| 13 | nRead = in.read(buf); |
| 14 | } |
| 15 | //TempObject temp=new TempObject(); |
| 16 | //\_container.updateDestination (temp.fromJSON(buffer.toString())); |
| 17 | } catch (IOException error) { |
| 18 | } |
| 19 | } |
| 20 | ... |
| 21 | ... |
| 22 | ... |

Gambar .25 Kode koneksi Jaringan

Implementasi fungsi openUrl() tersebut dijabarkan dalam gambar 5.25. Pada fungsi tersebut dibutuhkan tiga kelas utama yaitu StreamConnection, InputStream, dan StringBuffer yang ditulis pada baris tiga hingga lima.

StreamConnection adalah kelas yang akan menampung data hasil baca dari *url* internet dan akan menjadi objek Connection. Objek itu awalnya belum dapat dibaca sebagai data string sehingga perlu diolah kembali. Hasil data baca *url* yang sudah menjadi objek streamConnection tersebut selanjutnya akan dibaca sebagai file masukan untuk objek dari kelas InputStream. Setelah itu data yang sudah menjadi input akan dibaca karakternya satu persatu dan akan disatukan menjadi satu data utuh yang dapat dibaca sebagai data string.

## JSON Parser

Setelah *url* dari jaringan internet dibaca sebagai data string. Maka saatnya membaca data tersebut menjadi data-data yang dapat digunakan dalam aplikasi.

File yang berada dalam url jaringan internet adalah file JSON. File JSON tersebut memiliki elemen-elemen data yang dibaca dengan menggunakan *library* Blackbbery yaitu JSON.me yang sudah terdapat pada Blackberry SDK 6 ke atas.

Terdapat banyak sekali fungsi-fungsi yang terdapat pada *library* ini namun tidak semuanya dibutuhkan. Fungsi-fungsi yang dibutuhkan dibuat dan disatukan dalam satu kelas tersendiri dan dipanggil sesuai dengan kebutuhan.

Nama File :JSONParser.java

Isi file :fungsi getObject\_ByID(),

fungsi getName\_fromJSON(),

fungsi getAddress\_fromJSON()

Karena koneksi jaringan dan JSONParser masih dalam tahap ujicoba dan belum terintegrasi dengan aplikasi utama maka file JSONParser tersebut hanya membutuhkan elemen kontak yang secukupnya seperti nama, alamat dan email.

Untuk melakukan ujicoba dibutuhkan file *dummy* dimana file *dummy* tersebut memiliki elemen-elemen id, nama, alamat, dan *email* seperti pada gambar 5.26.

|  |  |
| --- | --- |
| 1 | [ |
| 2 | { |
| 3 | "id": "00001", |
| 4 | "group":{ |
| 5 | "DATA1":"tes1", |
| 6 | "DATA2":"tes2", |
| 7 | "DATA3":"tes3" |
| 8 | }, |
| 9 | "nama": "Sukma Arbianto", |
| 10 | "Alamat": "Surabaya", |
| 11 | "Email": "sukmasuk2006@yahoo.com" |
| 12 | }, |
| 13 | ... |

Gambar .26 File *Dummy* JSON untuk ujicoba.

Konsep JSON parser ini diawali dengan permintaan Aplikasi utama Blackberry untuk dikirimkan PIN dari database CRM di web. Kemudian CRM akan mengirimkan PIN yang diminta itu. Selanjutnya aplikasi Blackberry akan mengakses data JSON pada jaringan dan akan mencocokkan PIN yang diberikan dengan file JSON yang dibaca. Apabila sama maka aplikasi Blackberry akan membaca data elemen JSON yang berID sesuai dengan PIN

Dari konsep itulah dibutuhkan fungsi untuk mendapatkan ID. Selanjutnya setelah ID dicocokkan dan memiliki kesamaan, dibacalah data nama dan *email*.

Fungsi untuk mengambil ID diimplementasikan dengan fungsi getObject\_byID() seperti pada gambar 5.27. dan fungsi untuk membaca nama dan *email* diimplementasikan oleh fungsi getName\_fromJSON() dan getEmail\_fromJSON() pada gambar 5.28.

|  |  |
| --- | --- |
| 1 | public JSONObject getObject\_ByID (String jsonString, String id) throws JSONException{ |
| 2 | //masukkan data JSON |
| 3 | JSONObject outer = new JSONObject(jsonString); |
| 4 |  |
| 5 | //Ambil array yang ada pada data |
| 6 | JSONArray inner=outer.getJSONArray(DATA); |
| 7 |  |
| 8 | //lakukan comparasi id |
| 9 | for(int i=0; i<inner.length();i++){ |
| 10 | JSONObject selected=new JSONObject(inner.getString(i)); |
| 11 | if(selected.getString(ID).equals(id)){ |
| 12 | return selected; |
| 13 | } |
| 14 | } |
| 15 | return null; |
| 16 | } |
| 17 | ... |

Gambar .27 Implementasi kode untuk fungsi pengambilan ID

|  |  |
| --- | --- |
| 1 | public String getName\_fromJSON(JSONObject data, String myname) throws JSONException{ |
| 2 | return data.getString(myname); |
| 3 | } |
| 4 |  |
| 5 | public String getAddress\_fromJSON(JSONObject data, String myaddress) throws JSONException{ |
| 6 | return data.getString(myaddress); |
| 7 | } |
| 8 | ... |

Gambar .28 Implementasi kode untuk fungsi membaca nama dan alamat

File masukan dari fungsi pengambilan ID seperti pada gambar 5.27 adalah file data string JSON dan file id yang dibutuhkan. File data string JSON itu selanjutnya akan dibaca sebagai objek dari JSONObject. Suatu kelas bawaan dari *library* JSON.me. Pembacaan data tersebut tergambar pada baris ketiga. Setelah data string tersebut menjadi file objek JSONObjek, satu persatu elemen di dalamnya dibaca sesuai dengan ID yang diminta. Alur tersebut tergambar pada baris kesembilan hingga kedua belas.

Sementara untuk membaca data nama dan alamat, kedua fungsi pada gambar 5.28 cukup mengembalikan dua nilai nama dan alamat dengan masukan JSONObjek.

***[Halaman ini sengaja dikosongkan]***

# UJI COBA DAN EVALUASI

Berikut ini adalah hasil dan evaluasi yang dilakukan pada masing-masing fungsional dan kerja aplikasi. Hasil dan evaluasi tersebut meliputi hasil gambar uji coba yang diambil dari simulator Blackberry serta penjelasan dari gambar tersebut.

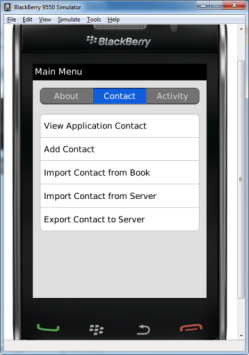
## Lingkungan Uji Coba

Lingkungan uji coba yang digunakan dalam pembuatan aplikasi ini meliputi perangkat lunak dan perangkat keras yang digunakan untuk proses rancang bangun aplikasi CRM pada perangkat Blackberry. Lingkungan uji coba merupakan komputer tempat uji coba sistem. Lingkungan uji coba yang digunakan pada Kerja Praktek ini adalah komputer dengan prosesorIntel(R) Core(TM)2 Duo CPU P8800 @2.66GHz 2.66GHz dan memori 4 GB. Komputer ini menggunakan sistem operasi Windows 7 Ultimate 64-bit. Perangkat pengembang yang digunakan adalah EclipseIndigo.

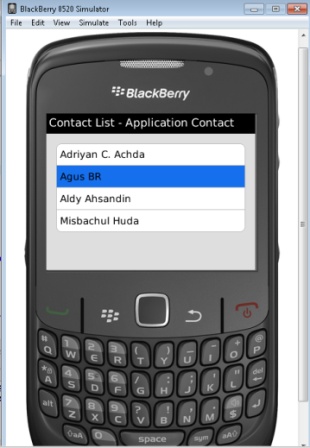
## Manipulasi Kontak

Hasil uji coba pada implementasi fungsional ini tergambar seperti pada gambar 6.1dan gambar 6.2 yang menampilkan beberapa operasi untuk melakukan manipulasi kontak. Operasi tersebut adalah menampilkan kontak, tambah kontak, import kontak dari phone book Blackberry, import kontak dari phone dari server pada aplikasi Web CMS.

Gambar 6.1 adalah hasil ujicoba aplikasi yang menampilkan fungsi-fungsi untuk memanipulasi kontak pada menu utama. Sementara pada gambar 6.2 adalah hasil dari ujicoba aplikasi mengakses data kontak yang diaplikasi



Gambar ‎6.1 Tampilan uji coba fungsional manipulasi kontak



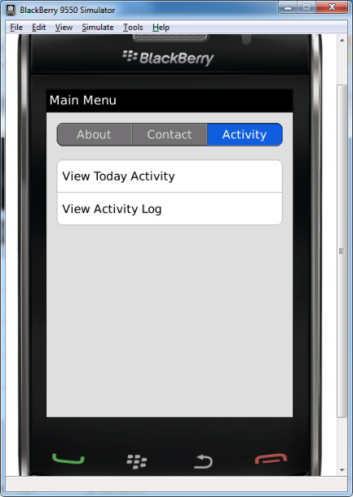
Gambar ‎6.2 Mengakses data kontak pada aplikasi

Setelah diimplementasikan kita mengevaluasi aplikasi ketika di deploy pada device Blackberry yang sebenarnya dan mendapati beberapa device tidak dapat menampilkan kontak dari phone book dengan baik. Bahkan aplikasi mengalami hang dan error. Kami belum mendapati bagaimana permasalahan ini diperbaiki karena alasan batasan waktu kerja praktek.

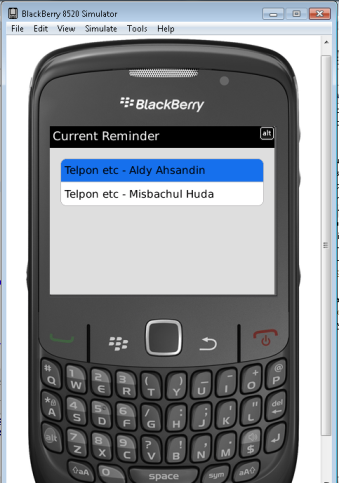
## Manipulasi Reminder

Hasil uji coba dari implementasi manipulasi *reminder* pada bab lima tergambar pada gambar 6.3 dan gambar 6.4. Pada gambar 6.3, aplikasi akan menampilkan fungsi-fungsi yang dapat dilakukan untuk memanipulasi reminder. Sementara untuk gambar 6.4 mampu menampilkan list dari *reminder* yang ada di aplikasi.

Dalam evaluasinya, aplikasi dapat melakukan *reminder* dengan telepon dan email. Hanya saja, untuk *reminder* dengan sms tidak dapat dilakukan karena fungsi sms yang harus mengakses ID Blackberry tersebut–menurut hasil observasi – diproteksi oleh sistem Blackberry karena alasan keamanan.



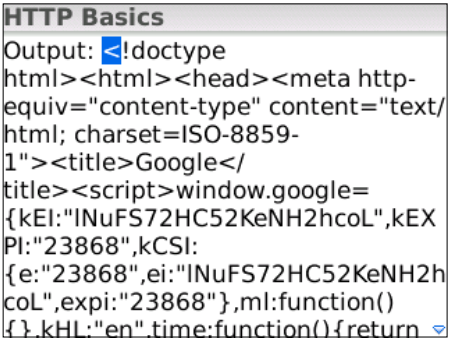
Gambar ‎6.3 Tampilan halaman uji coba *reminder*



Gambar ‎6.4 Tampilan halaman uji coba pada *reminder*yang masuk dalam aplikasi

## Koneksi Network

Untuk menguji apakah aplikasi dapat melakukan koneksi jaringan dan mengakses dengan baik, kami mencoba melakukan koneksi tersebut dengan jaringan internet seperti google. Jika dapat menampilkan halaman google dengan baik, berarti aplikasi dapat mengakses jaringan dengan baik. Dan hasilnya seperti yang ada di gambar 6.5.

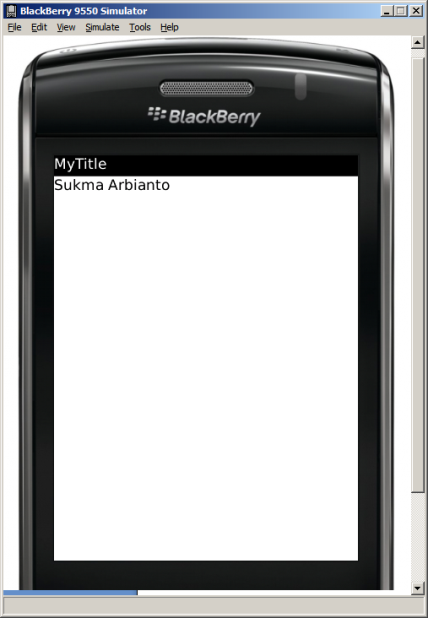


Gambar ‎6.5 Hasil tampilan uji coba koneksi ke halaman google

Gambar itu menunjukkan output html dari halaman google. Aplikasi dapat menampilkan html dari google menunjukkan bahwa aplikasi berhasil dapat melakukan koneksi ke jaringan internet.

## JSON Parser

Setelah mampu melakukan koneksi dengan jaringan, dilakukan uji coba project yang lain dan berdiri sendiri untuk membaca file JSON yang terdapat dalam jaringan. Setelah itu membacanya sesuai kebutuhan. Hasil dari uji cobanya nampak pada gambar 6.5. Projek uji coba tersebut akan membaca file JSON yang sudah tersedia pada jaringan dan berusaha mengambil ID pengguna bernomor 00001. Dan akan mengembalikan nilai berupa nama dari pemilik ID tersebut. Hasilnya jika melihat file JSON yang sudah ada, ID tersebut mengacu pada nama bernama Sukma Arbianto.



Gambar ‎6.6 Tampilan hasil dari JSON Parser berupa pengambilan data nama.

Melihat kemampuan dari file ujicoba,funsgi membaca data pada jaringan dipastikan mampu bekerja sesuai dengan kebutuhannya pada aplikasi. Hanya saja, karena keterbatasan waktu dan kemampuan, file testing pada uji coba ini tidak dapat digabungkan dengan aplikasi utama. Sehingga hasil uji coba ini berdiri sendiri dalam projek yang berbeda dan belum terintegrasi dengan aplikasi

# KESIMPULAN DAN SARAN

## Kesimpulan

1. Sebagian besar pelanggan Blackberry menggunakan sistem operasi 4 dan 5. Sementara Blackberry mulai mengembangkan kemampuannya dengan lebih baik pada sistem operasi 6 ke atas. Sehingga referensi yang banyak ditemukan adalah referensi untuk sistem operasiBlackberry6 ke atas dan sangat sulit menemukan sistem operasi 4 dan 5. Oleh karena itulah mengembangkan aplikasi di Blackberry lebih sulit daripada dengan Android.
2. Pada proses *running*  aplikasi di Blackberry *device*, tidak semua Blackberry dapat menjalankan aplikasi. Terkadang aplikasi berhenti dengan sendirinya sewaktu mengakses data kontak pada *phonebook* Blackberry *device*.

## Saran

1. Disarankan untuk menggunakan aplikasi PhoneGap untuk membuat aplikasi *native mobile* dengan menggunakan teknologi web. PhoneGap adalah alat pengembang berbasis HTML5 yang dapat digunakan di segala bentuk *mobile device*.
2. Karena terbatasnya *device* Blackberry yang dimiliki perlu dilakukan pengujian ke beragam versi Blackberry *device* untuk memastikan kemampuan dan kehandalan Blackberry *device* dalam menjalankan aplikasi.

***[Halaman ini sengaja dikosongkan]***

# DAFTAR PUSTAKA

[1] *Blackberry Development UI Guide* . <http://docs.blackberry.com/en/developers/deliverables/41577/>.(Diakses tanggal 20 Mei 2012)

[2] *JSON* . <http://www.json.org> .(Diakses tanggal 20 Mei 2012)

[3] *SQLite* . <http://www.sqlite.org>.(Diakses tanggal 20 Mei 2012)

***[Halaman ini sengaja dikosongkan]***

# LAMPIRAN SOURCE CODE

1. ***Source Code Aplikasi***

|  |
| --- |
| package com.rimdev.model;  publicclass BaseContactData {    protected String contactID;  protected String contactUID;    public BaseContactData() {  // TODO Auto-generated constructor stub  }    /\*\*  \* @return ContactData ID in Application  \*/  public String getContactID() {  return contactID;  }  /\*\*  \* @return BlackBerryContact UID  \*/  public String getContactUID() {  return contactUID;  }    publicvoid setContactID(String contactID) {  this.contactID = contactID;  }    publicvoid setContactUID(String contactUID) {  this.contactUID = contactUID;  }  } |

Gambar A.‎0.1 Kelas BasicContactData sebagai model data dasar dari ContactData

|  |
| --- |
| package com.rimdev.model;  public class BaseReminderData {  /\*\*  \*  \*/  protected String eventSysUID;  protected String eventUID;  protected String relatedContactUID;    public BaseReminderData() {  // TODO Auto-generated constructor stub  }    public void setEventSysUID(String eventSysUID) {  this.eventSysUID = eventSysUID;  }    public void setEventUID(String eventUID) {  this.eventUID = eventUID;  }    public void setRelatedContactUID(String relatedContactUID) {  this.relatedContactUID = relatedContactUID;  }    public String getEventSysUID() {  return eventSysUID;  }  public String getEventUID() {  return eventUID;  }  public String getRelatedContactUID() {  return relatedContactUID;  }  } |

Gambar A.‎0.2 Kelas BasicReminderData sebagai model data dasar dari ReminderData

|  |
| --- |
| package com.rimdev.samples;  import com.rimdev.model.ContactData;  import com.rimdev.model.ReminderData;  import com.rimdev.model.ReminderData.ReminderBuilder;  import com.rimdev.util.PersistenceManager;  import net.rim.device.api.ui.Font;  import net.rim.device.api.ui.MenuItem;  import net.rim.device.api.ui.Screen;  import net.rim.device.api.ui.UiApplication;  import net.rim.device.api.ui.component.LabelField;  import net.rim.device.api.ui.container.MainScreen;  public class AdvancedUIScreen extends MainScreen {    protected CloseMenuItem \_closeMenuItem;    protected class CloseMenuItem extends MenuItem  {  protected CloseMenuItem()  {  super("Back" , 100000, 5);  }  public void run()  {  // If successful, return to contact list.  if(onSave())  {  App().popScreen(App().getActiveScreen());  }  }  }    AdvancedUIScreen() {  // TODO Auto-generated constructor stub  this(0);  }  AdvancedUIScreen(long style) {  // TODO Auto-generated constructor stub  super(style);  }  public void setTitle( String title )  {  LabelField titleLabel = new LabelField( title );  titleLabel.setPadding( 4, 0, 3, 4 );  titleLabel.setFont( titleLabel.getFont().derive( Font.PLAIN, titleLabel.getFont().getHeight() + 2 ) );  super.setTitle( titleLabel );  }  protected UiApplication App()  {  return UiApplication.getUiApplication();  }  protected boolean onSavePrompt() {  // TODO Auto-generated method stub  return true;  }    public boolean onClose() {  // TODO Auto-generated method stub  App().requestBackground();  return true;  }  } |

Gambar A.‎0.3 Kelas AdvancedUIScreen sebagai *base class* dari semua screen.

|  |
| --- |
| package com.rimdev.samples;  import java.util.Calendar;  import java.util.Date;  import java.util.Enumeration;  import java.util.TimeZone;  import javax.microedition.io.Connector;  import javax.wireless.messaging.BinaryMessage;  import javax.wireless.messaging.MessageConnection;  import com.rimdev.model.ContactData;  import net.rim.blackberry.api.mail.Folder;  import net.rim.blackberry.api.mail.Message;  import net.rim.blackberry.api.mail.Session;  import net.rim.blackberry.api.mail.Store;  import net.rim.device.api.system.Alert;  import net.rim.device.api.system.ApplicationDescriptor;  import net.rim.device.api.system.ApplicationManager;  import net.rim.device.api.ui.UiApplication;  import net.rim.device.api.ui.component.Dialog;  /\*\*  \* This class extends the UiApplication class, providing a graphical user  \* interface.  \*/  public class BBSampleApplication extends UiApplication {  /\*\*  \* Entry point for application  \*  \* @param args  \* Command line arguments (not used)  \*/  private BackgroundThread backThread;  public static void main(String[] args) {  // Create a new instance of the application and make the currently  // running thread the application's event dispatch thread.  BBSampleApplication theApp = new BBSampleApplication();  theApp.enterEventDispatcher();  }  /\*\*  \* Creates a new BBSampleApplication object  \*/  public BBSampleApplication() {  // Push a screen onto the UI stack for rendering.  backThread = new BackgroundThread();  backThread.start();  pushScreen(new MainMenu());  }  private class BackgroundThread extends Thread {  private int H7, H17;  private boolean H7Done, H17Done;  private static final int periodCheck = 60000;  private int counter;  private boolean testSMS;    public BackgroundThread() {  /\*\*\* initialize parameters in constructor \*\*\*\*\*/  H7Done = false;  H17Done = false;  testSMS = false;  counter = 0;  }    public void run() {  while (true) {  /\*\*  \* @author Admin  \* <blockquote>  \* receive data from server if it's possible  \* </blockquote>  \*/  if((hour >= 7 && hour<17) && H7Done == false){  /\*\*  \* if ( process == done ) H7Done = true;  \*/  H17Done = false;  }  if((hour >= 17 ||(hour >= 1 && hour < 7)) && H17Done == false){  /\*\*  \* if ( process == done ) H17Done = true;  \*/  H7Done = false;  }  counter += 1000;  if(counter == periodCheck){  counter = 0;  /\*\*  \* check Email Status  \*/  Store store = Session.waitForDefaultSession().getStore();  Folder sent = store.list(Folder.SENT)[0];  Message[] msgList = sent.getMessages();  for(int i = 0; i < msgList.length ; i++){    }  }  sleep(2000);  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  }  }  } |

Gambar A.‎0.4 Kelas BBSampleAplication sebagai *starting class* dari aplikasi.

|  |
| --- |
| package com.rimdev.samples;  import java.util.Calendar;  import java.util.Date;  import java.util.TimeZone;  import net.rim.device.api.i18n.DateFormat;  import net.rim.device.api.system.Bitmap;  import net.rim.device.api.ui.\*;  import net.rim.device.api.ui.component.\*;  import net.rim.device.api.ui.container.\*;  import net.rim.device.api.ui.decor.Border;  import net.rim.device.api.ui.decor.BorderFactory;  // The pim packages.  import javax.microedition.pim.\*;  import com.rimdev.model.ContactData;  import com.samples.toolkit.ui.container.FieldSet;  import com.samples.toolkit.ui.container.ListStyleButtonSet;  import net.rim.blackberry.api.pdap.\*;  public final class ContactScreen extends AdvancedUIScreen  {  private EditField \_firstName,  \_lastName,  email\_Active,  mobile\_Phone,  home\_Phone,  work\_Phone,  addrStreet\_Home,  addrStreet\_Work,  addrTown\_Home,  addrTown\_Work;  private DateField birthday;  private SaveMenuItem \_saveMenuItem;  private String []genderList;  private ForegroundManager \_foreground;  private class SaveMenuItem extends MenuItem  {  private SaveMenuItem()  {  super("Save Contact" , 100001, 5);  }  public void run()  {  // If successful, return to contact list.  if(onSave())  {  App().popScreen(App().getActiveScreen());  }  }  }  /\*\*  \* Creates a new ContactScreen object  \*/  public ContactScreen()  {  super(NO\_VERTICAL\_SCROLL | USE\_ALL\_HEIGHT);  setTitle("Contact Detail - Create");    \_foreground = new ForegroundManager();    Border titleBorder = BorderFactory.createBitmapBorder( new XYEdges( 4, 12, 4, 12 ), Bitmap.getBitmapResource( "fieldset2\_title\_border.png" ) );  Border contentBorder = BorderFactory.createBitmapBorder( new XYEdges( 4, 12, 4, 12 ), Bitmap.getBitmapResource( "fieldset2\_body\_border.png" ) );    FieldSet itemSet = new FieldSet("Personal Information",titleBorder,contentBorder,USE\_ALL\_WIDTH);    genderList = new String[2];  genderList[0] = "Male";  genderList[1] = "Female";    \_saveMenuItem = new SaveMenuItem();  \_closeMenuItem = new CloseMenuItem();  \_firstName = new EditField("First Name: " , "");  itemSet.add(\_firstName);  \_lastName = new EditField("Last Name: " , "");  itemSet.add(\_lastName);  birthday = new DateField("Birthday: ", 0, DateFormat.getInstance(DateFormat.DATE\_SHORT));  itemSet.add(birthday);  \_foreground.add(itemSet);  FieldSet itemSet2 = new FieldSet("Contact Detail",titleBorder,contentBorder,USE\_ALL\_WIDTH);  mobile\_Phone = new EditField("Mobile: " , "", BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(mobile\_Phone);  home\_Phone = new EditField("Home: " , "", BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(home\_Phone);  work\_Phone = new EditField("Work: ", "", BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(work\_Phone);  email\_Active = new EditField("Email: " , "", BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_EMAIL);  itemSet2.add(email\_Active);  \_foreground.add(itemSet2);  FieldSet itemSet3 = new FieldSet("Address Detail",titleBorder,contentBorder,USE\_ALL\_WIDTH);  addrStreet\_Home = new EditField("Address(Home): ", "");  itemSet3.add(addrStreet\_Home);  addrTown\_Home = new EditField("Town(Home): ","");  itemSet3.add(addrTown\_Home);  addrStreet\_Work = new EditField("Address(Work): ", "");  itemSet3.add(addrStreet\_Work);  addrTown\_Work = new EditField("Town(Work): ","");  itemSet3.add(addrTown\_Work);  \_foreground.add(itemSet3);  add(\_foreground);  addMenuItem(\_saveMenuItem);  addMenuItem(\_closeMenuItem);  }  public ContactScreen(ContactData data)  {  super(NO\_VERTICAL\_SCROLL | USE\_ALL\_HEIGHT);  setTitle("Contact Detail - Edit");    \_foreground = new ForegroundManager();    Border titleBorder = BorderFactory.createBitmapBorder( new XYEdges( 4, 12, 4, 12 ), Bitmap.getBitmapResource( "fieldset2\_title\_border.png" ) );  Border contentBorder = BorderFactory.createBitmapBorder( new XYEdges( 4, 12, 4, 12 ), Bitmap.getBitmapResource( "fieldset2\_body\_border.png" ) );    FieldSet itemSet = new FieldSet("Personal Information",titleBorder,contentBorder,USE\_ALL\_WIDTH);    genderList = new String[2];  genderList[0] = "Male";  genderList[1] = "Female";    \_saveMenuItem = new SaveMenuItem();  \_closeMenuItem = new CloseMenuItem();  \_firstName = new EditField("First Name: " , data.getNames(BlackBerryContact.NAME\_GIVEN));  itemSet.add(\_firstName);  \_lastName = new EditField("Last Name: " , data.getNames(BlackBerryContact.NAME\_FAMILY));  itemSet.add(\_lastName);  birthday = new DateField("Birthday: ", data.getBirthdayDate(), DateFormat.getInstance(DateFormat.DATE\_SHORT));  itemSet.add(birthday);  \_foreground.add(itemSet);  FieldSet itemSet2 = new FieldSet("Contact Detail",titleBorder,contentBorder,USE\_ALL\_WIDTH);  mobile\_Phone = new EditField("Mobile: " , data.getPhoneNumber\_Mobile(), BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(mobile\_Phone);  home\_Phone = new EditField("Home: " , data.getPhoneNumber\_Home(), BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(home\_Phone);  work\_Phone = new EditField("Work: ", data.getPhoneNumber\_Work(), BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_PHONE);  itemSet2.add(work\_Phone);  email\_Active = new EditField("Email: " , data.getEmail(), BasicEditField.DEFAULT\_MAXCHARS, BasicEditField.FILTER\_EMAIL);  itemSet2.add(email\_Active);  \_foreground.add(itemSet2);  FieldSet itemSet3 = new FieldSet("Address Detail",titleBorder,contentBorder,USE\_ALL\_WIDTH);  addrStreet\_Home = new EditField("Address(Home): ", data.getAddress\_Home(BlackBerryContact.ADDR\_STREET));  itemSet3.add(addrStreet\_Home);  addrTown\_Home = new EditField("Town(Home): ",data.getAddress\_Home(BlackBerryContact.ADDR\_LOCALITY));  itemSet3.add(addrTown\_Home);  addrStreet\_Work = new EditField("Address(Work): ", data.getAddress\_Work(BlackBerryContact.ADDR\_STREET));  itemSet3.add(addrStreet\_Work);  addrTown\_Work = new EditField("Town(Work): ",data.getAddress\_Work(BlackBerryContact.ADDR\_LOCALITY));  itemSet3.add(addrTown\_Work);  \_foreground.add(itemSet3);  add(\_foreground);  addMenuItem(\_saveMenuItem);  addMenuItem(\_closeMenuItem);  }  private int getGenderIndex(String gdr){  if(gdr.equalsIgnoreCase("Male")){  return 0;  }  return 1;  }  /\*\*  \* @see net.rim.device.api.ui.Screen#onSave()  \*/  protected boolean onSave()  {  Calendar calendar = Calendar.getInstance(TimeZone.getDefault());  calendar.setTime(new Date(birthday.getDate()));  ContactData.createContactBuilder()  .setName(\_firstName.getText(), \_lastName.getText())  .setBirthDay(calendar.get(Calendar.DATE),calendar.get(Calendar.MONTH),calendar.get(Calendar.YEAR))  .setPhone\_Mobile(mobile\_Phone.getText())  .setPhone\_Home(home\_Phone.getText())  .setPhone\_Work(work\_Phone.getText())  .setEmail(email\_Active.getText())  .setHomeAddress(addrStreet\_Home.getText(), addrTown\_Home.getText())  .setWorkAddress(addrStreet\_Work.getText(), addrTown\_Work.getText())  .save();  return true;  }  } |

Gambar A.‎0.5 Kelas ContactScreen, *screen* penambahan kontak melalui aplikasi.

|  |
| --- |
| package com.rimdev.samples;  import net.rim.blackberry.api.phone.Phone;  import net.rim.blackberry.api.phone.PhoneListener;  import net.rim.device.api.ui.component.Dialog;  public class PhoneListenerCst implements PhoneListener {  public PhoneListenerCst() {  // TODO Auto-generated constructor stub  }  public void callAdded(int callId) {  // TODO Auto-generated method stub  }  public void callAnswered(int callId) {  // TODO Auto-generated method stub  Dialog.inform("End");  Phone.removePhoneListener(this);  }  public void callConferenceCallEstablished(int callId) {  // TODO Auto-generated method stub  }  public void callConnected(int callId) {  // TODO Auto-generated method stub  }  public void callDirectConnectConnected(int callId) {  // TODO Auto-generated method stub  }  public void callDirectConnectDisconnected(int callId) {  // TODO Auto-generated method stub  }  public void callDisconnected(int callId) {  // TODO Auto-generated method stub  Dialog.inform("End 2");  Phone.removePhoneListener(this);  }  public void callEndedByUser(int callId) {  // TODO Auto-generated method stub  Dialog.inform("End 3");  Phone.removePhoneListener(this);  }  public void callFailed(int callId, int reason) {  // TODO Auto-generated method stub  }  public void callHeld(int callId) {  // TODO Auto-generated method stub  }  public void callIncoming(int callId) {  // TODO Auto-generated method stub  }  public void callInitiated(int callid) {  // TODO Auto-generated method stub  }  public void callRemoved(int callId) {  // TODO Auto-generated method stub  }  public void callResumed(int callId) {  // TODO Auto-generated method stub  }  public void callWaiting(int callid) {  // TODO Auto-generated method stub  }  public void conferenceCallDisconnected(int callId) {  // TODO Auto-generated method stub  }  } |

Gambar A.‎0.6 Kelas PhoneListenerCst, *interface* untuk mendapatkan status panggilan yang dilakukan melalui aplikasi.

|  |
| --- |
| package com.rimdev.samples;  import java.io.IOException;  import javax.microedition.io.Connector;  import javax.wireless.messaging.BinaryMessage;  import javax.wireless.messaging.MessageConnection;  import com.rimdev.model.ContactData;  import com.rimdev.util.PersistenceManager;  import com.samples.toolkit.ui.component.ListStyleButtonField;  import com.samples.toolkit.ui.component.PillButtonField;  import com.samples.toolkit.ui.container.ListStyleButtonSet;  import com.samples.toolkit.ui.container.NegativeMarginVerticalFieldManager;  import com.samples.toolkit.ui.container.PillButtonSet;  import net.rim.device.api.system.EventLogger;  import net.rim.device.api.ui.Field;  import net.rim.device.api.ui.FieldChangeListener;  import net.rim.device.api.ui.Manager;  import net.rim.device.api.ui.MenuItem;  import net.rim.device.api.ui.UiApplication;  import net.rim.device.api.ui.component.LabelField;  public class MainMenu extends AdvancedUIScreen {  private ForegroundManager \_foreground;    private class SendMenuItem extends MenuItem {  public SendMenuItem() {  // TODO Auto-generated constructor stub  super("Send SMS", 100002, 5);  }  public void run() {  // TODO Auto-generated method stub  BgThread th = new BgThread();  th.start();  }  }    private class BgThread extends Thread{  public void run() {  try {  MessageConnection conn =  (MessageConnection)Connector.open("sms://085655311556");  BinaryMessage msgOut = (BinaryMessage) conn.newMessage(MessageConnection.BINARY\_MESSAGE);  msgOut.setPayloadData("my binary payload".getBytes("UTF-8"));  conn.send(msgOut);  System.out.println("Message sent");  } catch (Exception e) {  // TODO Auto-generated catch block\  System.out.println(e.getMessage());  e.printStackTrace();  }  }  }    private Manager contactMgr;  private Manager activityMgr;  private Manager aboutMgr;    private Manager \_bodyWrapper;  private Manager \_currentBody;    private PillButtonSet pillSet;  private PillButtonField pillContact,  pillActivity,  pillAbout;    private ListStyleButtonField addFromApplicationContact,  addFromBookContact, modifyContact, deleteContact, importContact, exportContact, viewActivity, viewLogActivity;  public MainMenu() {  // TODO Auto-generated constructor stub  super(NO\_VERTICAL\_SCROLL | USE\_ALL\_HEIGHT);  setTitle("Main Menu");  \_foreground = new ForegroundManager();  addMenuItem(new SendMenuItem());  pillSet = new PillButtonSet();  pillContact = new PillButtonField("Contact");  pillActivity = new PillButtonField("Activity");  pillAbout = new PillButtonField("About");  pillSet.add(pillAbout);  pillSet.add(pillContact);  pillSet.add(pillActivity);  pillSet.setMargin(15, 15, 5, 15);  \_foreground.add(pillSet);    \_bodyWrapper = new NegativeMarginVerticalFieldManager(USE\_ALL\_WIDTH);    contactMgr = new ListStyleButtonSet();    modifyContact = new ListStyleButtonField("View Application Contact", 0);  modifyContact.setChangeListener(new FieldChangeListener() {    public void fieldChanged(Field field, int context) {  // TODO Auto-generated method stub  App().pushScreen(new ContactListScreen(ContactListScreen.VIEW\_CONTACT));  }  });  contactMgr.add(modifyContact);    addFromApplicationContact = new ListStyleButtonField("Add Contact", 0);  addFromApplicationContact.setChangeListener(new FieldChangeListener() {  public void fieldChanged(Field field, int context) {  // TODO Auto-generated method stub  App().pushScreen(new ContactScreen());  }  });  contactMgr.add(addFromApplicationContact);    addFromBookContact = new ListStyleButtonField("Import Contact from Book", 0);  addFromBookContact.setChangeListener(new FieldChangeListener() {    public void fieldChanged(Field field, int context) {  // TODO Auto-generated method stub  App().pushScreen(new ContactListScreen(ContactListScreen.IMPORT\_CONTACT\_BOOK));  }  });  contactMgr.add(addFromBookContact);    contactMgr.add(importContact = new ListStyleButtonField("Import Contact from Server", 0));    contactMgr.add(exportContact = new ListStyleButtonField("Export Contact to Server", 0));    activityMgr = new ListStyleButtonSet();  activityMgr.add(viewActivity = new ListStyleButtonField("View Today Activity", 0));  viewActivity.setChangeListener(new FieldChangeListener() {  public void fieldChanged(Field field, int context) {  // TODO Auto-generated method stub  App().pushScreen(new ReminderListScreen());  }  });  activityMgr.add(viewLogActivity = new ListStyleButtonField("View Activity Log", 0));    aboutMgr = new ForegroundManager();  LabelField horizontalLabel = new LabelField( "Application Description" );  horizontalLabel.setPadding(5, 15, 5, 15);  aboutMgr.add(horizontalLabel);    pillSet.setSelectedField(pillAbout);  \_currentBody = aboutMgr;  \_bodyWrapper.add(\_currentBody);    pillContact.setChangeListener( new FieldChangeListener( ) {  public void fieldChanged( Field field, int context ) {  if( \_currentBody != contactMgr ) {  \_bodyWrapper.replace( \_currentBody, contactMgr );  \_currentBody = contactMgr;  }  }  } );  pillActivity.setChangeListener( new FieldChangeListener( ) {  public void fieldChanged( Field field, int context ) {  if( \_currentBody != activityMgr ) {  \_bodyWrapper.replace( \_currentBody, activityMgr );  \_currentBody = activityMgr;  }  }  } );  pillAbout.setChangeListener( new FieldChangeListener( ) {  public void fieldChanged( Field field, int context ) {  if( \_currentBody != aboutMgr ) {  \_bodyWrapper.replace( \_currentBody, aboutMgr );  \_currentBody = aboutMgr;  }  }  } );  \_foreground.add( \_bodyWrapper );  add( \_foreground );  addMenuItem(new InitContactDataItem());  }  } |

Gambar A.‎0.7 Kelas MainMenuScreen, *screen* pertama yang ditampilkan aplikasi. Berfungsi sebagai menu utama.

|  |
| --- |
| package com.rimdev.util;  import java.util.Vector;  import com.rimdev.model.BaseContactData;  import com.rimdev.model.BaseReminderData;  import net.rim.blackberry.api.pdap.BlackBerryContact;  import net.rim.device.api.system.ControlledAccessException;  import net.rim.device.api.ui.\*;  import net.rim.device.api.ui.component.\*;  import net.rim.device.api.ui.container.\*;  import net.rim.device.api.database.\*;  import net.rim.device.api.io.\*;  final public class PersistenceManager {  private Database \_sqliteDB;  private static PersistenceManager \_currentInstance;    private PersistenceManager() {  // TODO Auto-generated constructor stub  try {  URI uri = URI.create("file:///SDCard/Databases/testingDB.db");  \_sqliteDB = DatabaseFactory.openOrCreate(uri);    } catch (IllegalArgumentException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (MalformedURIException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (ControlledAccessException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DatabaseIOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DatabasePathException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }    public static PersistenceManager getCurrentInstance(){  \_currentInstance = (\_currentInstance == null) ? new PersistenceManager() : \_currentInstance;  return \_currentInstance;  }    public boolean openDB(){  try {  URI uri = URI.create("file:///SDCard/Databases/testingDB.db");  \_sqliteDB = DatabaseFactory.openOrCreate(uri);  return true;    } catch (IllegalArgumentException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (MalformedURIException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (ControlledAccessException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DatabaseIOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DatabasePathException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return false;  }    public boolean closeDB(){  try {  \_sqliteDB.close();  return true;  } catch (DatabaseIOException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return false;  }    public void initDatabase(){    try {  Statement sta = \_sqliteDB.createStatement("CREATE TABLE 'ContactData' (" +  "'UID\_Contact' varchar(20) NULL," +  "'ID\_SYS\_Contact' varchar(20) NULL)");  sta.prepare();  sta.execute();    Statement sta2 = \_sqliteDB.createStatement("CREATE TABLE 'ReminderData' ("+  "'UID\_Reminder' varchar(20) NULL,"+  "'UID\_Contact' varchar(20) NULL,"+  "'ID\_SYS\_Reminder' varchar(20) NULL," +  "'Status' INTEGER NULL)");    sta2.prepare();  sta2.execute();    } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    }  public int insertContactData(String uid\_contact, String id\_db){  try {  Statement sta = \_sqliteDB.createStatement("INSERT into ContactData (UID\_Contact,ID\_SYS\_Contact)"+  "values("+uid\_contact+","+id\_db+")");  sta.prepare();  sta.execute();  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return 0;  }  public int insertContactData(String uid\_contact){  try {  Statement sta = \_sqliteDB.createStatement("INSERT into ContactData (UID\_Contact,ID\_SYS\_Contact)"+  "values("+uid\_contact+",-1)");  sta.prepare();  sta.execute();  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return 0;  }  public int deleteContactData(String uid){  try {  Statement sta = \_sqliteDB.createStatement("Delete from ContactData where UID\_Contact = "+uid);  sta.prepare();  sta.execute();  } catch (Exception e) {  // TODO: handle exception  }  return 0;  }  public Vector getContactData(){  Vector result = new Vector();  try {  Statement sta = \_sqliteDB.createStatement("Select \* From ContactData");  sta.prepare();  Cursor c = sta.getCursor();  Row r;  while(c.next()){  BaseContactData bcd = new BaseContactData();  r = c.getRow();  bcd.setContactUID(r.getString(0));  bcd.setContactID(r.getString(1));  result.addElement(bcd);  }  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DataTypeException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    return result;  }    public int insertReminderData(String uid\_event,String uid\_contact,String id\_db\_event){  try {  Statement sta = \_sqliteDB.createStatement("INSERT into ReminderData(UID\_Reminder,UID\_Contact,ID\_SYS\_Reminder)"+  "values("+uid\_event+","+uid\_contact+","+id\_db\_event+")");  sta.prepare();  sta.execute();  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return 0;  }    public int insertReminderData(String uid\_event,String uid\_contact){  try {  Statement sta = \_sqliteDB.createStatement("INSERT into ReminderData(UID\_Reminder,UID\_Contact,ID\_SYS\_Reminder)"+  "values("+uid\_event+","+uid\_contact+",-1)");  sta.prepare();  sta.execute();  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return 0;  }    public int deleteReminderData(String uid){  try {  Statement sta = \_sqliteDB.createStatement("Delete from ReminderData where UID\_Reminder = "+uid);  sta.prepare();  sta.execute();  } catch (Exception e) {  // TODO: handle exception  }  return 0;  }    public Vector getReminderData(){  Vector result = new Vector();  try {  Statement sta = \_sqliteDB.createStatement("Select \* From ReminderData");  sta.prepare();  Cursor c = sta.getCursor();  Row r;  while(c.next()){  BaseReminderData brd = new BaseReminderData();  r = c.getRow();  brd.setEventUID(r.getString(0));  brd.setRelatedContactUID(r.getString(1));  brd.setEventSysUID(r.getString(2));  result.addElement(brd);  }  } catch (DatabaseException e) {  // TODO Auto-generated catch block  e.printStackTrace();  } catch (DataTypeException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }    return result;  }  } |

Gambar A.‎0.8 Kelas PersistenceManager, kelas pengendali database yang digunakan.