BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi dari sistem yang telah dibuat berdasarkan rancangan pada tahap sebelumnya akan dibahas dalam bagian ini.

4.1.1 Implementasi Sistem

Perangkat lunak yang dibutuhkan untuk pengembangan sistem adalah

1. Java 6 (OpenJDK Client VM 1.6.0 dan IcedTea 1.3.1 *Runtime Environtment*).
2. NetBeans 6.1 untuk membangun aplikasi yang merupakan editor yang mendukung *syntax highlighting* sehingga memudahkan pengembangan sistem.
3. Sistem Operasi BlankOn Meuligoe (BlankOn versi 4) yang merupakan distro Linux turunan Ubuntu Intrepid Ibex (8.10).
4. Emulator untuk pengimplementasian sistem di komputer sebelum dipasang di perangkat bergerak.

4.1.2 Implementasi MIDlet Syahfi

Class Syahfi merupakan turunan dari *class* MIDlet. Dalam kode sumber, ditandai dengan klausa `extends`. Hal ini diperlukan agar *class* Syahfi dapat diperlakukan sebagai sebuah objek MIDlet. *Class* MIDlet merupakan *class* abstrak dan di dalamnya terdapat metode-metode abstrak yang digunakan untuk mengatur siklus hidup aplikasi. Metode-metode tersebut adalah `startApp()`, `pauseApp()`, `destroyApp()`, serta `notifyDestroyed()`.

Class Syahfi juga bertugas sebagai pengendali antar muka aplikasi yang akan ditampilkan ke layar. Untuk tujuan tersebut pada kelas ini terdapat komponen Display. Pada sebuah MIDlet hanya terdapat satu objek Display. Objek display menyediakan metode untuk menggambar dan menampilkan elemen antarmuka grafis pada layar. Objek display juga menyediakan metode untuk mengetahui properti layar perangkat seperti apakah layar perangkat mendukung layar berwarna atau tidak. Dalam sebuah MIDlet hanya boleh ada satu buah Display. Pada *file Syahfi.java* terdapat proses membuat sebuah *form* baru, dimana *Learn* merupakan *class* turunan dari Form, kemudian *form* akan ditampilkan ke layar. Gambar 4.1 adalah kode sumber untuk melakukan hal ini.

```
public void startApp() {
    learn = new Learn(this, display);
    display.setCurrent(learn);
}
```

Gambar 4.1 Kode sumber untuk menampilkan *form Learn*

Untuk keluar dari midlet, digunakan metode *exitMIDlet*(). Kode sumbernya ditunjukkan oleh gambar 4.2.

```
public void exitMidlet(){
    destroyApp(false);
    notifyDestroyed();
}
```

Gambar 4.2 Kode sumber untuk keluar dari MIDlet

Pada saat pengguna keluar dari MIDlet, maka metode `destroyApp`() akan dijalankan sebelum MIDlet benar-benar tidak berjalan lagi. Metode `destroyApp`() akan memanggil `notifyDestroyed()`, dan

`notifyDestroyed()` akan memberitahu *platform* untuk menterminasi MIDlet dan membersihkan semua sumber daya yang mengacu pada MIDlet. Kode sumber metode `destroyApp()` terlihat pada gambar 4.3.

```
public void destroyApp(boolean unconditional) {
}
```

Gambar 4.3 Kode sumber metode `destroyApp`()

Kondisi inisialisasi yakni kondisi pause akan dijalankan oleh metode `pauseApp`(). Kode sumbernya terlihat pada gambar 4.4.

```
public void pauseApp() {
}
```

Gambar 4.4 Kode sumber metode `pauseApp()`

4.1.3 Implementasi Antarmuka Menu Utama (*Form Learn*)

Setelah masuk ke menu utama, pengguna akan langsung dihadapkan pada proses belajar. Bila ini adalah kali pertama pengguna menggunakan Syahfi, maka aplikasi hanya akan menampilkan pesan selamat datang dan anjuran untuk menambahkan *cards* ke dalam *record store*. Sejumlah menu dapat diakses dengan tombol penampil menu pada perangkat bergerak. Tampilan awal ketika menjalankan Syahfi terlihat pada gambar 4.5.

Gambar 4.5 Tampilan awal Syahfi

Class Learn akan memanggil *class Store* yang merupakan class khusus yang bukan merupakan turunan dari *class Form.* Fungsi utama *class Store* adalah menampung semua metode yang berkaitan dengan *record store*. Inisiasinya akan dimasukkan ke dalam *private variable* `store`. Untuk mengakses metode yang berkaitan dengan *record store*, koneksi ke *record store* harus dibuka terlebih dahulu dengan metode `open`(). Setiap koneksi ke *record store* yang terbuka ditutup dengan metode `close`(). Kode sumbernya terlihat pada gambar 4.6.

```
private Store store = new Store(Store.storageName);
.
.
      store.open();
/*
call method related with record store..
*/
      store.close();
```

Gambar 4.6 Gambaran singkat penggunaan *class Store*

Kode sumber untuk metode `open()` pada *class Store* ditunjukkan oleh gambar 4.7.

```
   boolean open(){
        try {
            store =
RecordStore.openRecordStore(this.storageName, true);
        } catch (RecordStoreException rse) {
        }
        if (store == null)
            return false;
        try {
            if (this.getStore().getNumRecords() <=0)//if there
is no store before, create and insert default data
                this.initiateData();
        } catch (RecordStoreNotOpenException rse){


        }
        return true;
    }
```

Gambar 4.7 Metode `open()` pada *class Store* yang berfungsi membuka koneksi

Kode sumber metode `close()` yang berfungsi menutup koneksi ditunjukkan oleh gambar 4.8.

```
boolean close() {
    try {
        if (store != null){
            store.closeRecordStore();
        }
    } catch (RecordStoreException rse){
        return false;
    }
    return true;
}
```

Gambar 4.8 Metode `close`() pada *class Store*

Yang dilakukan aplikasi pada kali pertama pemakaian adalah menyimpan data inisiasi ke dalam *record store*. Hal ini dikerjakan oleh metode `initiateData`() pada *class* `Store` pada saat pembuatan *record store*. Metode ini hanya sekali saja dipanggil. Bila sudah ada *record store*, maka metode ini tidak akan pernah dipanggil. Kode sumbernya terlihat pada gambar 4.9.

```
    void initiateData(){//do this each time a record store
created or reset
        String timeOfStart =
String.valueOf(System.currentTimeMillis()/1000);
        this.saveRecord(0, timeOfStart);//id 1
        this.saveRecord(0, "5");//id 2 : Number of 0 cards to
learn at once
        this.saveRecord(0, "1");//id 3: learn new cards in
random order, 0 means No, and 1 means Yes
        this.saveRecord(0, "0");//id 4: the length of Cards
(banyaknya cards) used by numOfCard()
        this.saveRecord(0, ":");//id 5: define the slots where
there is no card, or empty card (because of deletion)
        for (int i=6;i<cardStartId;i++)
            this.saveRecord(0,"");//reserved space for future
devs
    }
```

Gambar 4.9 Kode sumber metode `initiateData()` pada *class Store*

Ada 5 data penting yang disimpan pada 10 record pertama pada *record store* yang diberi nama `SyahfiData`. Data penting tersebut adalah sebagai berikut

1. *Time of Start*

   Data ini adalah informasi waktu penanda hari dimulai proses belajar. Bilangan yang tersimpan merepresentasikan jumlah detik dari tanggal 1 Januari 1970 atau yang dikenal dengan istilah *epoch time*.

2. *Number of grade 0 cards to learn at once*

   Data ini adalah informasi preferensi pengguna. Bilangan yang tersimpan menyatakan jumlah maksimal *grade 0 cards* yang akan dipelajari dalam satu kali waktu belajar.

3. *Learn new cards in random order*

   Nilai yang tersimpan pada *record* ini berisi preferensi pengguna. Bila bernilai

"0" maka *new cards* akan dipelajari secara urut (berdasarkan urutan pengguna dalam menginput *cards*). Sedangkan bila bernilai "1" maka *cards* akan diacak terlebih dahulu sebelum dimasukkan ke dalam *queue.*

4. *Number of Cards*

   Nilai yang tersimpan menyatakan banyaknya *card* dalam *record store* yang telah dimasukkan pengguna. Data *number of cards* akan dipakai oleh metode `numOfCards`(). Hal ini mencegah aplikasi berulang kali menghitung semua *card* setiap kali dibutuhkan. Nilai dari *record* ini akan berubah bila pengguna menambah atau menghapus *card.*

5. *Empty slots*

   Ketika sebuah *record* dihapus dari *record store*, maka nilai ID-nya tidak akan dapat dipakai untuk menyimpan *record* lagi. Hal ini menciptakan "hole" di dalam *record store.* Artinya, aplikasi akan butuh waktu lebih lama dalam memperoleh informasi tentang *card* karena pengaksesan *record* dilakukan secara sekuensial. Masalah ini diatasi dengan cara tidak benar-benar menghapus sebuah *record*, melainkan hanya "mengosongkan" isinya sehingga *record* dapat digunakan untuk menyimpan *card record* (*record* tentang card). Semua ID *card* yang dihapus oleh pengguna akan disimpan dalam *record* ini. Untuk *record* nomor 6 hingga 10 diisi dengan *blank-space* untuk keperluan pengembangan aplikasi selanjutnya (*reserved*).

Kode sumber untuk menampilkan pesan selamat datang bila belum ada card dalam *record store* ditunjukkan oleh gambar 4.10.

```
        store.open();
        int numOfCards = store.numOfCards();
        store.close();
        if (numOfCards == 0){
            append("Welcome to Syahfi! \n" +
                    "To start learning, please add some cards
from 'Edit Cards'"+
                    "-> 'Add Card' menu. \n\n" +
                    "Happy learning!");
        }else {
            showQuestion();
        }
```

Gambar 4.10 Kode sumber pesan selamat datang

Metode `append`() berfungsi untuk menambahkan item ke dalam *Form*. Item yang dimasukkan dapat berupa *string*, gambar atau objek lain. Bila sudah terdapat *card* dalam *record store*, maka aplikasi akan mulai menerapkan algoritma SM-2 hasil modifikasi Peter Bienstman pada *card* dan menampilkan hasilnya melalui metode `showQuestion`() pada *class Learn*. Kode sumber metode `showQuestion`() ditunjukkan oleh gambar 4.11.

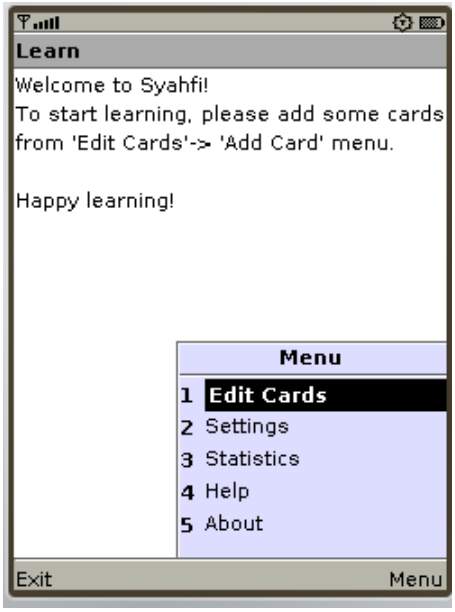
```
    private void showQuestion(){
        store.open();
        if ( revisionQueue.size() == 0 ){
            revisionQueue = store.rebuildRevisionQueue(false);
        }

        deleteAll();
        if ( revisionQueue.size() != 0){

            //get first question and remove it from
revisionQueue AFTER user grade it
            cardId = Integer.parseInt((String)
revisionQueue.firstElement());
            cardQuestion = store.getQuestionOfCard(cardId);
            cardAnswer = store.getAnswerOfCard(cardId);
            //cardGrade = store.getFieldOfCard(cardId, 1);

            store.close();
            //set teks on siQuestion
            append(siQuestion);
            siQuestion.setText(cardQuestion);
            append(btShowAnswer);

        }else {
            //remove siQuestion
            deleteAll();
            //user's asked for learning ahead option
            append("There is no more scheduled card");
            //add button 'learn ahead'
            append(btLearnAhead);
            store.close();
        }
    }
```

Gambar 4.11 Metode `showQuestion()` pada *class Learn*

Pertama-tama akan diperiksa isi dari `revision queue` (antrian *card* yang akan dipelajari). Bila *queue* belum ada, maka variabel `revisionQueue` akan diisi dengan nilai balik dari metode `rebuildRevisionQueue`() yang terdapat pada *class Store*. Kode sumber metode ini terdapat dalam bagian lampiran. Nilai balik ini bertipe `Vector` agar lebih memudahkan penambahan dan penghapusan *card* dalam *queue*. Parameter berupa `boolean` akan membedakan apakah *card* yang dijadwalkan untuk hari bersangkutan akan dimasukkan atau tidak. Bila bernilai `false`, berarti yang dimasukkan ke dalam *queue* hanyalah *card-card* yang dijadwalkan untuk hari itu.

Selanjutnya semua item dalam *Form* dihapus dengan menggunakan metode `deleteAll`(). Sampai di sini, isi *queue* akan diperika kembali. Bila isi *queue* masih ada, maka aplikasi akan menampilkan pertanyaan dari *card* dengan posisi antrian paling depan. Selanjutnya pertanyaan akan ditampilkan bersamaan dengan *button* “Show Answer”.

Apabila terdapat kondisi di mana *queue* tetap kosong walaupun sudah di-*rebuild*, ini berarti sudah tidak ada lagi *card* yang memenuhi syarat untuk ditampilkan hari itu. Pengguna akan diberi pesan bahwa tidak ada lagi *card* terjadwal dengan menambahkan *string* “There is no more cards to learn” pada layar. Aplikasi akan menampilkan *button* “Learn Ahead?” dan pengguna dapat memilih antara melanjutkan belajar dengan *card-card* yang seharusnya terjadwal di masa mendatang, mengakses menu aplikasi, atau keluar dari aplikasi. Tampilan aplikasi ketika menampilkan menu ditunjukkan oleh gambar 4.12.



Gambar 4.12 Tampilan awal aplikasi beserta menu

Tampilan aplikasi ketika pengguna sedang dalam proses belajar ditunjukkan oleh gambar 4.13.

Gambar 4.13 Tampilan awal proses belajar (menampilkan pertanyaan)

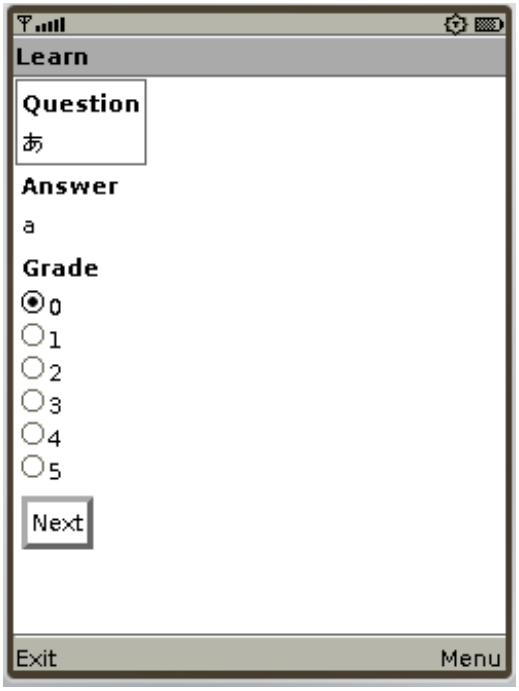
Pengguna akan membayangkan jawabannya di pikiran, lalu menekan button “Show Answer”. Selanjutnya metode `CommandAction` yang merupakan *listener* akan menjalankan fungsinya. Gambar 4.14 adalah kode sumber yang dijalankan ketika pengguna menekan *button* “Show Answer”.

```
        if (co == cmBtShowAnswer){
           delete(1);//delete btShowAnswer
           append(siAnswer);
           siAnswer.setText(cardAnswer);
           append(cgGrade);
           append(btGrade);
        }
```

Gambar 4.14 Kode sumber yang dijalankan ketika pengguna menekan *button* “Show Answer”

Pertama-tama aplikasi akan menghapus item ke-2, yakni item *button* “Show Answer” dan menggantinya dengan `siAnswer` (*String Item Answer*), *choice Group* untuk memilih *grade* baru bagi *card*, serta menambahkan *button* “Next” yang akan membawa pengguna ke proses selanjutnya.

Tampilan aplikasi setelah pengguna menekan *button* “Show Answer” ditunjukkan oleh gambar 4.15.



Gambar 4.15 Tampilan setelah pengguna menekan tombol "Show Answer"

Pada bagian ini pengguna akan diharuskan memberi *grade* pada *card* yang sedang dipelajari sesuai dengan tingkat kesulitan *card* menurut pengguna. Hal ini dapat dilakukan dengan cara memilih satu di antara pilihan pada `cgGrade` atau *Choice Group Grade*. Selanjutnya pengguna akan menekan *button* "Next" dan aplikasi akan menentukan interval. Interval digunakan untuk menentukan jadwal *card* akan dipelajari kembali. Kode sumber ketika pengguna menekan *button* "Next" ditunjukkan oleh gambar 4.16.

```
    private void processAnswer(){
        store.open();
        store.processAnswer(cardId,
cgGrade.getSelectedIndex());
        revisionQueue.removeElementAt(0);
        store.close();

        showQuestion();
    }
```

Gambar 4.16 Kode sumber metode `processAnswer`() pada *class Learn*

Aplikasi akan memanggil metode `processAnswer`() pada *class Store* dengan melewatkan parameter `cardId` dan *grade* dari *card* yang ditentukan sendiri oleh pengguna. Kode sumber metode `processAnswer()` pada *class Store* terdapat dalam bagian lampiran.

Ada lima kondisi terkait dengan pemberian grade pada card. Kondisi tersebut dijelaskan sebagai berikut

1. *Card* adalah *new card*

   Sebuah *card* dianggap sebagai *new card* bila nilai atribut *acquisition reps* (`acq_reps`) dan *retention repetition* (`ret_reps`) sama dengan 0.

2. *Card* tetap berada di fase akuisisi
3. *Card* berpindah dari fase akuisisi ke fase retensi
4. *Card* berpindah dari fase retensi ke fase akuisisi
5. *Card* tetap berada di fase retensi.

Masing-masing kondisi akan membuat nilai atribut *card* mendapat perlakukan berbeda. Nilai dari atribut *easiness* misalnya, hanya berubah bila *card*

mengalami kondisi tetap berada di fase rentensi.

Setelah aplikasi mengubah nilai atribut *card* sesuai dengan kondisi di atas, maka aplikasi akan menghapus elemen *card* yang sudah muncul dari *queue*. Hal ini membuat banyaknya card dalam *queue* menjadi berkurang. Demikian seterusnya hingga tidak ada lagi *card* yang bisa dipelajari. *Button* "Learn Ahead" akan ditampilkan bila sudah tidak ada lagi *card* yang dijadwalkan dipelajari hari itu. Gambar 4.17 adalah tampilan aplikasi ketika dalam *queue* sudah tidak ada *card* terjadwal lagi.

Gambar 4.17 Tampilan aplikasi saat *queue* sudah tidak berisi *card* terjadwal

Bila *button* "Learn Ahead" dipilih oleh pengguna, maka metode `commandAction`() akan dipanggil dan aplikasi akan menjalankan kembali metode `rebuildRevisionQueue`() dengan parameter `learnAhead` bernilai `true`.

Gambar 4.18 adalah potongan kode sumber ketika *button* "Learn Ahead" dipilih pengguna.

```
}else if (co == cmBtLearnAhead){
            store.open();
            revisionQueue = store.rebuildRevisionQueue(true);
            store.close();
            showQuestion();
        }
```

Gambar 4.18 Potongan kode sumber ketika *button* "Learn Ahead" dipilih

Aplikasi kembali menampilkan satu per satu *card* yang terdapat dalam *queue.* Demikian seterusnya hingga *queue* tidak berisi *card* lagi.

4.1.4 Implementasi Bagian *Cards*

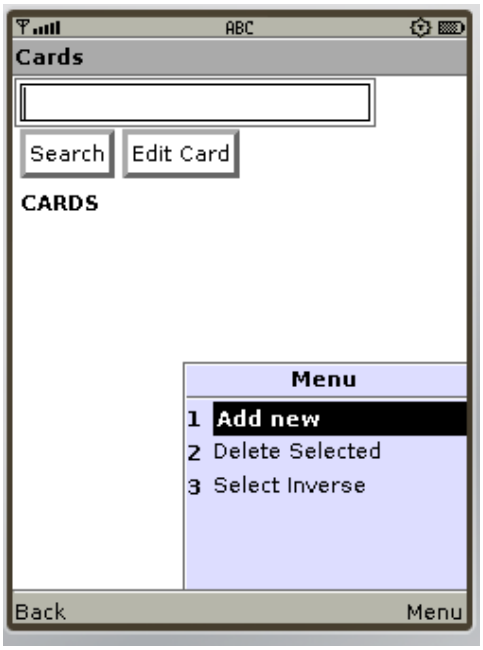
Menu *Edit Cards* dapat diakses dari menu utama. Pada bagian ini, *class* yang berperan adalah *class Card*. *Class Card* merupakan *class* turunan dari *class* `Form`. Class ini mengimplementasikan *class* `CommandListener` dan `ItemCommandListener`. Atribut pada *class* ini ditunjukkan oleh gambar 4.19.

```
    private Syahfi midlet;
    private Display display;


    private Command cmBack, cmAddNew, cmDeleteSelected,
cmSelectOtherwise, cmSearch, cmEdit;
    private TextField tfSearch;
    private StringItem siSearchButton, siEditButton;
    private String[][] cardData;
    private ChoiceGroup cgCards;
```

Gambar 4.19 Atribut pada *class Card*

Dalam kali pertama pemakaian (kondisi *record store* belum berisi *card*

sama sekali), tampilan aplikasi akan terlihat seperti gambar 4.20.



Gambar 4.20 Tampilan awal menu *Edit Cards*

Halaman *Edit Cards* berisi beberapa fungsi utama. Fungsi yang terdapat pada halaman ini adalah sebagai berikut

1. Menampilkan semua *card* yang telah dimasukkan pengguna.

   Kode sumber untuk menampilkan semua *card* yang telah dimasukkan pengguna ditunjukkan oleh gambar 4.21

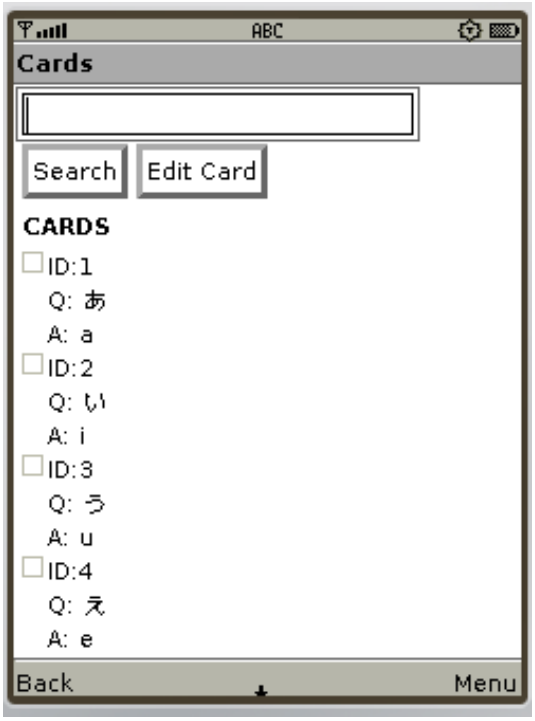
```
        //show all cards here..
        cgCards = new ChoiceGroup("CARDS", Choice.MULTIPLE);
        Store store = new Store(Store.storageName);
        store.open();
        int numOfCards = store.numOfCards();
        int numOfCardAttributes = store.numOfCardAttributes;
```

Gambar 4.21 Kode sumber untuk menampilkan *cards* (bagian 1 dari 2)

```
        cardData = new String[numOfCards][numOfCardAttributes
+1];

        cardData = store.readAllCards();

        store.close();

        for (int i=0;i<numOfCards;i++){

            if (cardData[i][0].equals(""))

                break;//just to make sure there is no space
after

            cgCards.append("ID:"+ cardData[i][0]+"\nQ: "+
cardData[i][10] + "\nA: "+ cardData[i][11], null);//0 means
ID, 10 means Question and 11 means Answer

        }

        append(cgCards);
```

Gambar 4.21 Kode sumber untuk menampilkan *cards* (bagian 2 dari 2)

Cards yang tersimpan dalam *record store* akan dibaca secara sekuensial dengan menggunakan metode `readAllCards`() pada *class Store*. Metode ini akan mengembalikan nilai berupa *array of String* yang hasilnya ditampilkan dalam bentuk *choice group* bertipe *multiple* pada halaman *Edit Cards*. Informasi yang ditampilkan pada halaman ini hanyalah informasi tentang ID *card*, atribut *question* serta atribut *answer*. Tampilan dari *card* yang sudah dimasukkan sebelumnya oleh pengguna terlihat pada gambar 4.22.



Gambar 4.22 Tampilan awal halaman *Cards*

2. Menampilkan *card* yang sesuai dengan kata kunci pencarian.

   Pada form aplikasi akan ditambahkan button "Search" dan mengeset `ItemCommandListener`. Penambahan *button* ini dilakukan dengan kode sumber pada gambar 4.23.

```
        siSearchButton = new StringItem(null, "Search",
Item.BUTTON);
        cmSearch = new Command("Search", Command.OK, 1);
        siSearchButton.setDefaultCommand(cmSearch);
        siSearchButton.setItemCommandListener(this);
        append(siSearchButton);
```

Gambar 4.23 Kode sumber penambahan *button* "Search"

Ketika pengguna menekan button "Search", aplikasi akan menjalankan potongan kode sumber yang tampak pada gambar 4.24.

```
        if (c == cmSearch){
            Store store = new Store(Store.storageName);
            store.open();
            String[][] filteredCardData;
            filteredCardData =
store.searchCard(tfSearch.getString());
            cgCards.deleteAll();
            if (filteredCardData != null){
                for (int i=0;i<filteredCardData.length;i++){
                    cgCards.append("ID:"+ filteredCardData[i]
[0]+
                                   "\nQ: "+ filteredCardData[i]
[10] +
                                   "\nA: "+ filteredCardData[i]
[11], null);//0 means ID, 10 means Question and 11 means
Answer
                }
            }
            store.close();
```

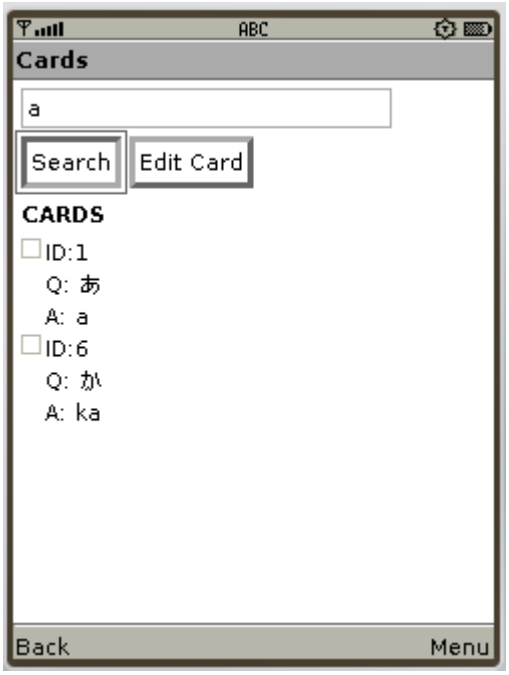
Gambar 4.24 Kode sumber yang dijalankan ketika pengguna memilih *button* "Search"

Yang dilakukan pertama kali adalah membuka koneksi ke *record store*. Selanjutnya terdapat variabel `filteredCardData` yang digunakan untuk menampung hasil pencarian (bertipe *array of string*). Lalu semua *item choice group* akan dihapus dari *form*. Tampilan *choice group* lalu akan berubah sesuai dengan banyaknya hasil pencarian yang didapat. Metode pada *record store* yang bertugas menangani pencarian adalah metode `searchCard()`. Gambar 4.25 menunjukkan kode sumbernya.

```
    public String[][] searchCard(String query){
        if (query.trim().equals("")){
            return readAllCards();
        }
        int numOfCards = this.numOfCards();
        int numOfCardAtt = this.numOfCardAttributes;
        String[][] cardData = new String[numOfCards]
[numOfCardAtt +1];//11+1, 1 for the ID
        int i=0;//index
        int cardID = 1;
        int j=0;//index of return cardData
        while (i < numOfCards){
            if (!
this.readRecord(convertToRecordId(cardID)).equals("")){
                if (this.readRecord(convertToRecordId(cardID)
+9).indexOf(query) != -1 |
                    this.readRecord(convertToRecordId(cardID)
+10).indexOf(query) != -1){//check only in Question and Answer
sections
                    cardData[j] = this.readCardWithID(cardID);
                    j++;
                }
                i++;
            }
            cardID++;
        }
        if (j ==0 )
            return null;//zero result
        String[][] filteredCardData = new String[j]
[numOfCardAttributes+1];
        for (int k=0;k<j;k++){
            filteredCardData[k] = cardData[k];
        }
        return filteredCardData;
    }
```

Gambar 4.25 Kode sumber method `searchCard` pada *class Store*

Tampak bahwa aplikasi Syahfi hanya mencari pada atribut *question* atau *answer*. Gambar 4.26 adalah tangkapan layar yang menampilkan hasil pencarian ketika pengguna menggunakan keyword huruf 'a'.



Gambar 4.26 Tampilan untuk hasil pencarian huruf 'a'

Bila pengguna tidak memasukkan karakter apapun pada *text field* kata kunci, semua *card* akan ditampilkan.

3. Mengedit *card* yang sudah dimasukkan sebelumnya.

   Pengguna dapat mengubah atribut *question* atau *answer* pada *card* tertentu dengan memasukkan ID *card* yang ingin diubah pada *text field* dan menekan button “Edit Card”. Penambahan button “Edit Card” dimungkinkan oleh potongan kode sumber pada gambar 4.27.

```
        siEditButton = new StringItem(null, "Edit Card",
Item.BUTTON);
        cmEdit = new Command("Edit", Command.OK, 1);
        siEditButton.setDefaultCommand(cmEdit);
        siEditButton.setItemCommandListener(this);
        append(siEditButton);
```

Gambar 4.27 Kode sumber untuk menambah *button* "Edit Card"

Button "Edit Card" akan ditambahkan setelah *button* "Search" sehingga posisinya berada setelah *button* "Edit Card" dan sebelum *choice group*. Setelah pengguna memasukkan ID *card* yang akan diedit pada text field, maka aplikasi akan memeriksa apakah input yang dimasukkan bernilai bilangan lebih dari 0. Ini merupakan filtering yang pertama. Kode sumber untuk melakukan hal ini ditunjukkan oleh gambar 4.28.

```
String cId = tfSearch.getString().trim();
            int cIdInt;
            try {
                cIdInt = Integer.parseInt(cId);
            }catch (NumberFormatException nfe){
                cIdInt = 0;
            }
```

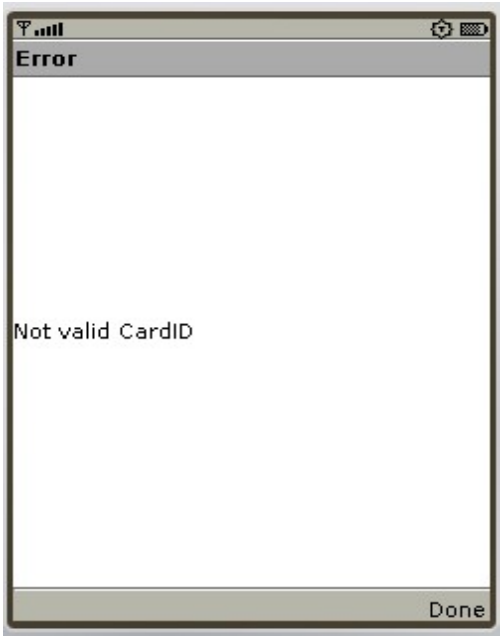
Gambar 4.28 Kode sumber penyaring masukan dari pengguna

Bila pengguna memasukkan format angka yang salah, maka nilai ID *card* yang ingin diubah akan diset sama dengan 0. Selanjutnya, aplikasi akan memeriksa kembali ID *card* yang dimasukkan pengguna dengan kode sumber yang ditunjukkan gambar 4.29.

```
            if ( cId.equals("") | cIdInt <=0){
                Alert al = new Alert("Error", "Not valid
CardID", null, AlertType.ERROR);
                al.setTimeout(Alert.FOREVER);
                display.setCurrent(al);
            }
```

Gambar 4.29 Kode sumber untuk memeriksa masukan ID *card* pengguna

Apabila ID *card* bernilai kosong atau tidak valid, maka aplikasi akan menampilkan pesan kesalahan dengan menggunakan *Alert*. *Alert* adalah sejenis pesan yang tampil di layar yang menampilkan teks maupun gambar ke layar yang berguna untuk menginformasikan sesuatu ke pengguna. Selanjutnya pengguna dapat memilih menu "Done" untuk kembali ke tampilan sebelumnya. Gambar 4.30 menunjukkan pesan kesalahan yang tampil bila pengguna salah memasukkan ID *card*.



Gambar 4.30 Pesan kesalahan ketika ID *card* tidak valid

Kode sumber pada gambar 4.31 adalah bagian yang dijalankan aplikasi setelah penyaringan masukan pengguna dijalankan. Ada satu kemungkinan lagi yaitu bila sebuah ID *card* tidak ada (mungkin karena sudah dihapus atau belum diisi).

```
                if ( store.isCardEmpty(cIdInt) ){
                    store.close();
                    Alert al = new Alert("Error", "ID doesn't
exist", null, AlertType.INFO);
                    al.setTimeout(Alert.FOREVER);
                    display.setCurrent(al);
                }else {
                    store.close();
                    EditCard ec = new EditCard(midlet,
display, cIdInt);
                    display.setCurrent(ec);
                }
```

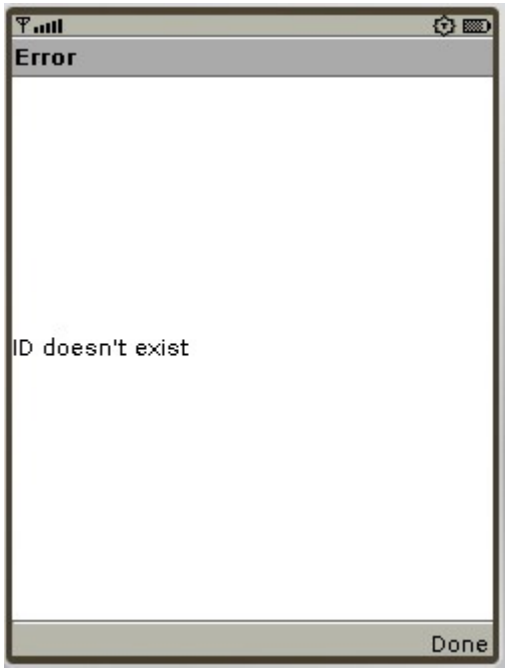
Gambar 4.31 Kode sumber yang dijalankan setelah masukan pengguna diperiksa

Metode yang memeriksa apakah sebuah Id *card* kosong atau tidak adalah metode `isCardEmpty`() pada *class Store*. Kode sumbernya ditunjukkan oleh gambar 4.32.

```
    public boolean isCardEmpty(int cardId){
        if (cardId <=0){
            return true;
        }else if ( readRecord(convertToRecordId(cardId)) ==
null ||
                readRecord(convertToRecordId(cardId)).equals("
")
                ){
            return true;
        }
        return false;
    }
```

Gambar 4.32 Kode sumber metode `isCardEmpty`() pada *class Store*

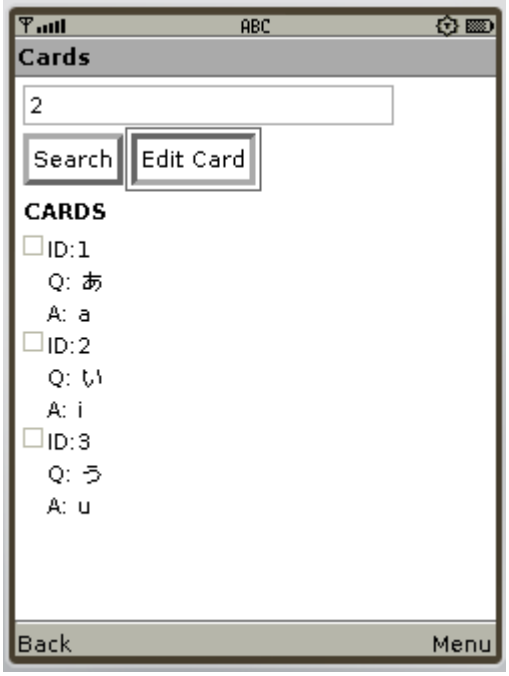
Apabila sebuah ID card kosong atau tidak ada, maka pengguna akan diberi informasi dengan menampilkan *Alert* seperti ditunjukkan oleh gambar 4.33.



Gambar 4.33 Tampilan pesan bila sebuah ID *card* kosong atau tidak ada

Bila ID *card* ditemukan maka aplikasi akan memanggil *class EditCard*

dengan parameter ID *card* yang dimasukkan pengguna. Gambar 4.34 menunjukkan tampilan aplikasi saat pengguna akan mengedit *card*.

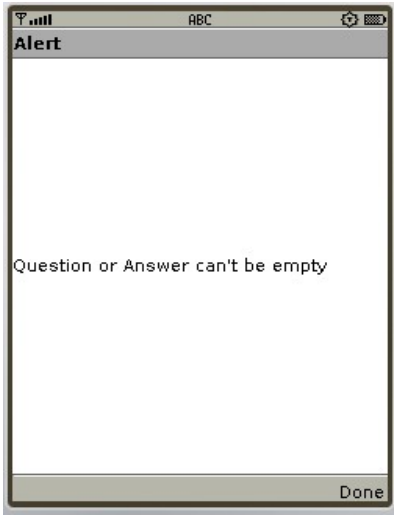


Gambar 4.34 Tampilan saat pengguna hendak mengedit *card*

Gambar 4.35 menunjukkan tampilan ketika *class EditCard* dipanggil untuk mengedit *card* dengan ID sesuai dengan yang diminta pengguna.

Gambar 4.35 Tampilan ketika *class EditCard* dipanggil untuk mengedit *card* dengan ID sama dengan 2

Bila pengguna tidak memasukkan *question* atau *answer* maka akan muncul pesan kesalahan bahwa *question* atau *answer* tidak boleh kosong. Gambar 4.36 menunjukkan pesan kesalahan yang dimaksud.



Gambar 4.36 Pesan kesalahan ketika *question* atau *answer* tidak diisi

Tampilan pesan kesalahan di atas ditangani oleh kode sumber pada gambar 4.37.

```
if (tfQuestion.getString().equals("") ||
tfAnswer.getString().equals("")){
                Alert al = new Alert("Alert", "Question or
Answer can't be empty", null, AlertType.INFO);
                al.setTimeout(Alert.FOREVER);
                display.setCurrent(al);
            }
```

Gambar 4.37 Kode sumber penampil pesan kesalahan

Bila sudah tidak ada masalah dengan masukan dari pengguna, maka aplikasi akan menyimpan *card*. Kode sumbernya ditunjukkan oleh gambar 4.38.

```
if (store.editCard(Integer.parseInt(cardData[0]),
tfQuestion.getString(), tfAnswer.getString())){
     Alert al = new Alert("Card added", "Card's succesfully
saved", null, AlertType.INFO);
     al.setTimeout(Alert.FOREVER);
     display.setCurrent(al);
}
```

Gambar 4.38 Kode sumber untuk menyimpan *card* pada *class EditCard*

Pada *class Store* yang berfungsi untuk mengedit *card* adalah metode `editCard`. Kode sumbernya ditunjukkan oleh gambar 4.39.

```
    public boolean editCard(int cardId, String question,
String answer){

        saveRecord(convertToRecordId(cardId)+9,
question);//index of field question is located in the 10th

        saveRecord(convertToRecordId(cardId)+10,
answer);//11th

        return true;

    }
```

Gambar 4.39 Kode sumber metode `editCard` pada *class Store*

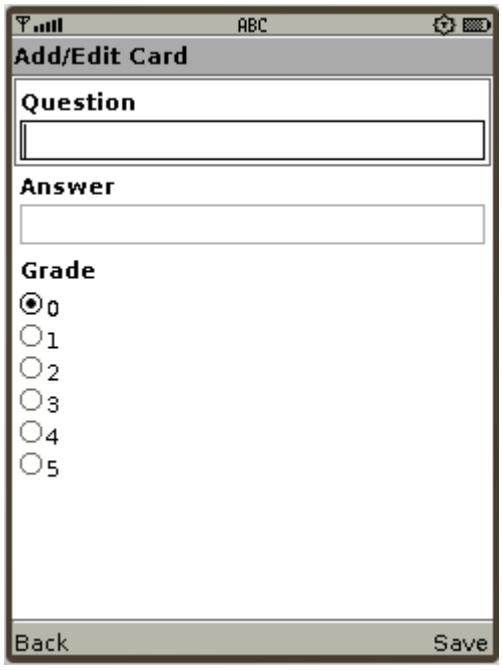
Pengguna hanya dapat mengedit nilai dari *question* dan *answer* dari *card*. Sedangkan nilai atribut lainnya tidak dapat ditentukan secara manual. Bila proses pembaruan *card* berhasil, maka akan ditampilkan pesan sebagaimana ditunjukkan pada gambar 4.40.

Gambar 4.40 Pesan kesuksesan penambahan *card*

Pengguna akan kembali ke tampilan sebelumnya yaitu tampilan *edit card* setelah *card* selesai disimpan.

4. Menyediakan menu untuk menambah *card* baru.

   Dari tampilan awal *class Card*, pengguna dapat memilih menu "Add New" yang berarti menambahkan *card* baru ke dalam *record store*. Setelah pengguna memilih menu "Add New" maka tampilan akan tampak seperti tampilan saat pengguna mengedit *card* tertentu, namun dengan isi text field question dan answer yang kosong. Gambar 4.41 menunjukkan tampilan fungsi penambahan *card* baru.



Gambar 4.41 Tampilan halaman "Add New" *card*

Kode sumber untuk menampilkan halaman "Add New" *card* terdapat

sepenuhnya sama dengan class *EditCard.* Perbedaan penggunaan *class* ini adalah parameter `cardId` ketika *class* dipanggil. Bila *cardId* bernilai sama dengan 0, maka aplikasi akan menampilkan halaman "Add New" *card*. Bila `cardId` bernilai lebih dari sama dengan 1 maka pengguna akan dihadapkan pada tampilan *Edit Card*. Kode sumber untuk menjadi pembeda terletak pada penambahan *item choice group*. Kode sumber untuk penambahan *item choice group* dapat dilihat pada gambar 4.42.

```
            cgGrade = new ChoiceGroup("Grade",
ChoiceGroup.EXCLUSIVE);
            cgGrade.append("0", null);
            cgGrade.append("1", null);
            cgGrade.append("2", null);
            cgGrade.append("3", null);
            cgGrade.append("4", null);
            cgGrade.append("5", null);
            append(cgGrade);
```

Gambar 4.42 Kode sumber penambahan *choice group* pada halaman "Add New"

Choice group bertipe `EXCLUSIVE` sehingga hanya satu *grade* yang dapat dipilih pengguna. Selanjutnya metode yang akan menangani penyimpanan *card* adalah metode `addNewCard` pada *class Store.* Metode `addNewCard` hanya memiliki 3 (tiga) parameter yakni *grade*, *question*, dan *answer.* Selebihnya, aplikasi akan menentukan nilai inisiasinya. Metode `addNewCard()` akan memanggil metode `saveCard`() yang memiliki parameter sebanyak 12 (dua belas) yakni sejumlah atribut dari *card.* Sebelum itu kode sumber yang dijalankan pada *class EditCard* ditunjukkan oleh gambar 4.43.

```
if (store.addNewCard(cgGrade.getSelectedIndex(),
tfQuestion.getString(), tfAnswer.getString())){
      Alert al = new Alert("Card added", "Card's succesfully
saved", null, AlertType.INFO);
      al.setTimeout(Alert.FOREVER);
      display.setCurrent(al);
}
```

Gambar 4.43 Kode sumber yang memanggil metode addNewCard()

Kode sumber metode `addNewCard()` ditunjukkan oleh gambar 4.44.

```
public boolean addNewCard(int grade, String question, String
answer){
    if (numOfCards() >= maximumNumberOfCardsAllowed){
    return false;
    }else {
      //defining some default values
       int cardId = getEmptySlots();//check emptySlots() first,
return 0 means new ID
      byte gr = (byte) grade;
      double easiness = averageEasiness();
      int l_rp = daysSinceStart();
      int newInterval = calculateInitialInterval(gr);
      newInterval += calculateIntervalNoise(newInterval);
      int n_rp = l_rp + newInterval;
      this.saveCard(
      cardId,//if it's 0, it means it'll be located in new
recordID
      gr,//it is byte!
      easiness,
      1,//ac_rp, see add_new_item about this default value
      0,
      0,
      1,//ac_rp_l, see add_new_item about this default value
      0,
      l_rp,
      n_rp,
      question,
      answer);
      int oldNumOfCards = this.numOfCards();
      this.saveRecord(4, String.valueOf(oldNumOfCards+1));
           return true;
      }
}
```

Gambar 4.44 Kode sumber metode `addNewCard` pada *class Store*

Pertama-tama akan diperiksa apakah jumlah *card* sudah mencapai batas maksimal *card* yang ditetapkan aplikasi. Hal ini dimaksudkan untuk menjaga kecepatan akses ke *record store*. Setelah itu akan diperiksa apakah terdapat *empty slot*. Empty slot adalah ID *card* yang tidak terpakai karena telah dihapus oleh pengguna. Penempatan *card* baru pada *empty slot* menjaga posisi antar *card* agar tetap rapat sehingga pada waktu dilakukan pencarian atau pengambilan data proses menjadi lebih cepat. Metode `getEmptySlots()` ditunjukkan oleh gambar 4.45.

```
    public int getEmptySlots(){
        //check record, if contains only ":", it means empty
        String emptySlots = readRecord(5);
        String blankCardId = "";
        if ( emptySlots.length() < 1 ){
            saveRecord(5, ":");
            return 0;
        }else if (emptySlots.length() == 1){
            return 0;
        }else {
            for (int i=1; i<emptySlots.length(); i++){
                if ( emptySlots.charAt(i+1) == ':' ){
                    blankCardId = emptySlots.substring(1,i+1);
                    emptySlots = ":" +
emptySlots.substring(i+2, emptySlots.length());
                    break;
                }
            }
            saveRecord(5, emptySlots);//save the rest back
            return Integer.parseInt(blankCardId);
        }
    }
```

Gambar 4.45 Kode sumber metode `getEmptySlots`()

Metode `getEmptySlots`() akan mengembalikan nilai 0 atau ID *card* yang tidak terpakai lagi untuk dapat digunakan menyimpan *card* yang baru. Apabila tidak ada *empty slot*, maka ID yang akan ditempati oleh *card* baru adalah ID setelah *record* terakhir yang dimasukkan pengguna. Untuk setiap penambahan *card*, maka secara otomatis nilai `numOfCards` pada *record store* juga akan berubah. Setelah aplikasi berhasil menambahkan *card*, akan ditampilkan pesan kesuksesan seperti tampak pada gambar 4.46.

Gambar 4.46 Pesan kesuksesan
setelah *card* berhasil ditambah

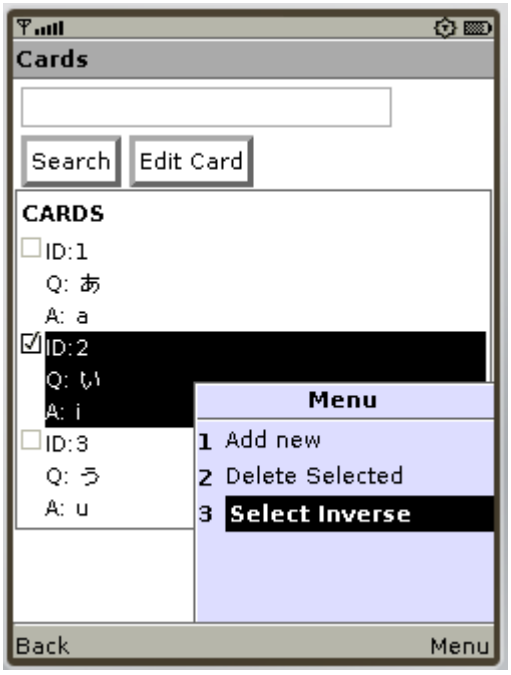
5. Menghapus *card* yang dipilih pengguna

   Pengguna dapat menghapus *card* dengan cara memilih terlebih dahulu *card* yang akan dihapus. Untuk memudahkan pengguna dalam memilih *card*, terdapat menu "Select Inverse" yang kode sumbernya terlihat pada gambar 4.47.

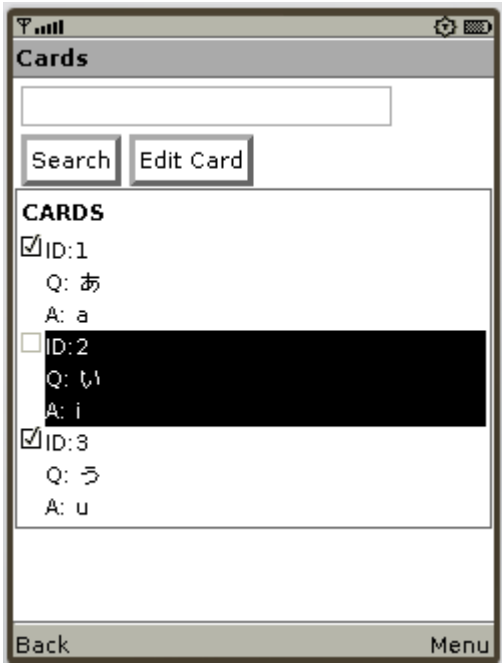
```
for (int i=0;i<cgCards.size();i++){
                cgCards.setSelectedIndex(i, !
cgCards.isSelected(i));
            }
```

Gambar 4.47 Kode sumber untuk menu "Select Inverse"

Gambaran penggunaannya ditunjukkan oleh gambar 4.48 dan 4.49.



Gambar 4.48 Tampilan penggunaan menu "Select Inverse" (bagian 1)



Gambar 4.49 Tampilan penggunaan menu "Select Inverse" (bagian 2)

Untuk menghapus *card* dipergunakan metode `deleteCard`() pada *class Store*. Kode sumbernya ditunjukkan oleh gambar 4.50.

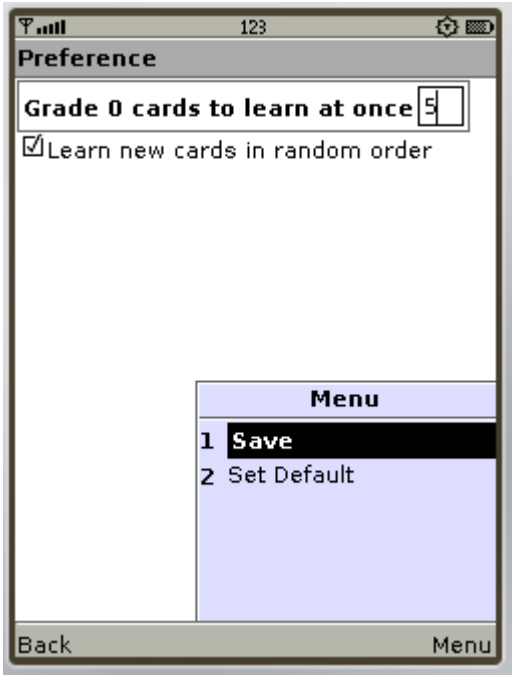
```
    public boolean deleteCard (int cardId){
        if (cardId <= 0){//there is no card with 0 as Id
            return false;
        }

        int startRecordId = convertToRecordId(cardId);
        int endRecordId = startRecordId + numOfCardAttributes;
        for (int i=startRecordId;i<endRecordId;i++){
            if (!blankRecord(i)){
                return false;
            }
        }

        //don't forget to -- numOfCards()
        int oldNumOfCards = numOfCards();
        if (oldNumOfCards > 0){
            this.saveRecord(4,
String.valueOf(oldNumOfCards-1));
        }

        //add blankId to emptySlots
        addToEmptySlots(cardId);

        return true;
    }
```

Gambar 4.50 Kode sumber metode `deleteCard`() pada *class Store*

ID dari *card* yang dihapus akan disimpan ke dalam *empty slot* serta nilai `numOfCard` akan diturunkan sebesar 1.

4.1.5 Implementasi Halaman *Settings*

Halaman *Setting* ditangani oleh *class Setting* yang merupakan *class* turunan dari *Form*. *Class* ini akan dipanggil melalui menu *Setting* pada tampilan utama (halaman *Learn*). Tampilan halaman *Setting* ditunjukkan oleh gambar 4.51.



Gambar 4.51 Tampilan halaman *Setting*

Terdapat 2(dua) buah menu yang dapat diakses pengguna yakni “Save” dan “Set Default”. Menu “Save” digunakan untuk menyimpan preferensi sedangkan menu “Set Default” digunakan untuk mengembalikan nilai asli dari preferensi. Aplikasi akan menjalankan potongan kode sumber pada gambar 4.52 untuk menyimpan nilai preferensi pengguna.

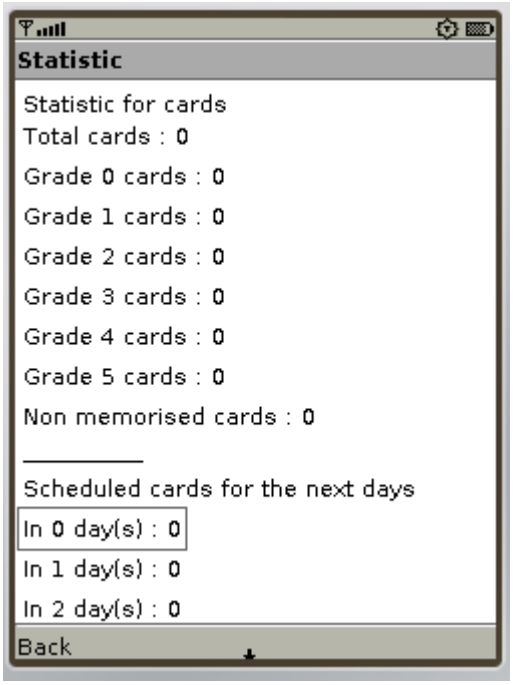
```
if (tfGrade0Cards2LearnAtOnce.getString().equals("")){
      Alert al = new Alert("Error","All fields are required",
null, AlertType.INFO);
      display.setCurrent(al);
}else {
      if (store.open()){
            store.saveRecord(2,
tfGrade0Cards2LearnAtOnce.getString());
            if (cgIsRandomOrderLearning.isSelected(0)){
                  store.saveRecord(3, "1");
             }else {
                  store.saveRecord(3, "0");
             }
      }
}
```

Gambar 4.52 Kode sumber yang dijalankan untuk menyimpan preferensi

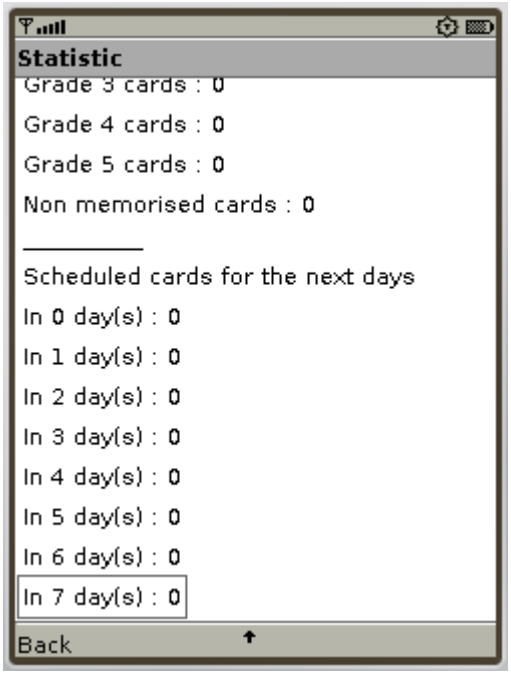
Preferensi “Grade 0 cards to learn at once” menunjukkan banyaknya *card* dengan *grade* 0 yang akan dipelajari pengguna dalam satu *queue* atau satu kali proses belajar. Sedangkan preferensi “Learn new cards in random order” menunjukkan apakah pengguna akan mempelajari *card* baru dengan urutan yang acak. Menu “Back” digunakan untuk kembali ke menu utama.

4.1.6 Implementasi Halaman *Statistics*

Halaman *Statistics* menampilkan data statistik tentang *card* yang dipelajari. Halaman ini dapat dipanggil dari menu utama dengan memilih menu “Statistics”. *Class Statistic* adalah *class turunan* Form yang berfungsi menampilkan halaman ini. Tampilan dari halaman ini ditunjukkan oleh gambar 4.53 dan 4.54.



Gambar 4.53 Tampilan halaman *Statistics* (bagian 1)



Gambar 4.54 Tampilan halaman *Statistics* (bagian 2)

Informasi yang ditampilkan pada halaman Statistics diatur oleh kode sumber gambar 4.55.

```
        this.store = new Store(Store.storageName);
        this.store.open();


        int[] cardData = store.getCardIds();
        int[] numOfCardWithGrade = new int[6];
        int numOfCards = cardData.length, nonMemorisedCards =
0;
        for (int i=0; i<numOfCards; i++){
            numOfCardWithGrade[this.store.getFieldOfCard(cardD
ata[i], 1)]++;//check card's grade
            if
(this.store.isDueForAcquisitionRep(cardData[i])){
                nonMemorisedCards++;
            }
        }


        append("Statistic for cards\nTotal cards : " +
numOfCards + "\n");
        append("Grade 0 cards : " + numOfCardWithGrade[0] + "\
n");
        append("Grade 1 cards : " + numOfCardWithGrade[1] + "\
n");
        append("Grade 2 cards : " + numOfCardWithGrade[2] + "\
n");
        append("Grade 3 cards : " + numOfCardWithGrade[3] + "\
n");
        append("Grade 4 cards : " + numOfCardWithGrade[4] + "\
n");
        append("Grade 5 cards : " + numOfCardWithGrade[5] + "\
n");
        append("Non memorised cards : " + nonMemorisedCards +
"\n__________\n");
```

```
        append("Scheduled cards for the next days\n");
        int oldCumulative = 0;
        int cumulative;
        for (int i=0;i<8;i++){            cumulative =
this.store.scheduledItems(i);
            append("In " + String.valueOf(i) + " day(s) : " +
                    String.valueOf(cumulative - oldCumulative)
+ "\n");
            oldCumulative = cumulative;
        }

        this.store.close();
```

Gambar 4.55 Kode sumber untuk menampilkan statistik card

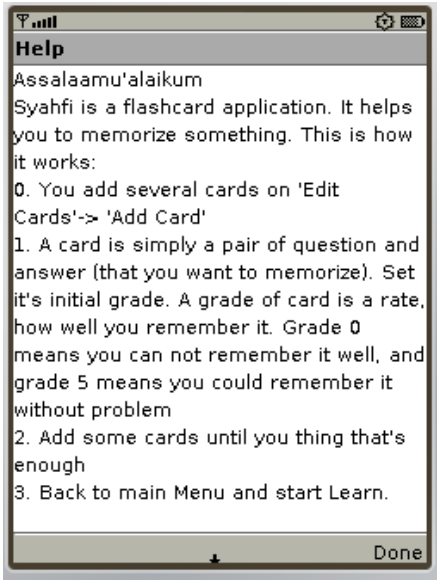
Metode `isDueForAcquisitionRep` digunakan untuk memeriksa apakah *card* belum diingat yang berarti *grade* dari *card* bernilai 0 atau 1. Kode sumber dari metode ini ditunjukkan oleh gambar 4.56.

```
    public boolean isDueForAcquisitionRep(int cardId){
        return (getFieldOfCard(cardId, 1) < 2);
    }
```

Gambar 4.56 Kode sumber metode `isDueForAcquisitionRep` pada *class Store*

4.1.7 Implementasi Antamuka *Help*

Halaman *Help* hanyalah sebuah *Alert* yang ditampilkan ketika pengguna memilih menu “Help” pada menu utama. Pada halaman ini terdapat informasi singkat tentang cara penggunaan aplikasi. Tampilan halaman ini ditunjukkan oleh gambar 4.57.



Gambar 4.57 Tampilan halaman *Help*

4.1.7 Implementasi Antamuka *About*

Halaman About berisi informasi singkat tentang aplikasi dan pengembangnya. Tampilan halaman ini ditunjukkan oleh gambar 4.58.

Gambar 4.58 Tampilan Halaman *About*

Halaman About juga merupakan Alert yang untuk menampilkannya pengguna dapat memilih menu “About” dari menu utama (halaman Learn).

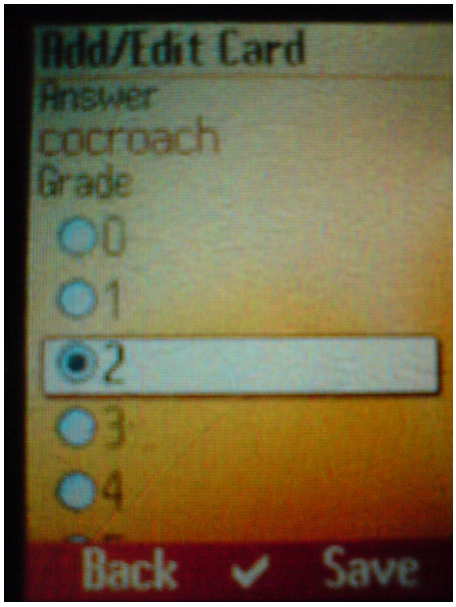
4.2 Pengujian Sistem

4.2.1 Pengujian pada emulator

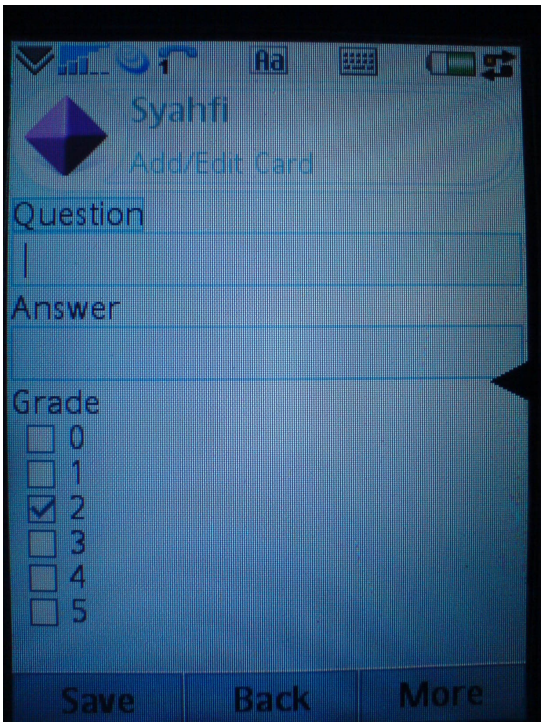
Emulator yang digunakan adalah j2mewtk versi 2.5.2 milik Sun Microsystem. Ketika diuji pada emulator, aplikasi dapat berjalan dengan baik.

4.2.1 Pengujian pada perangkat bergerak

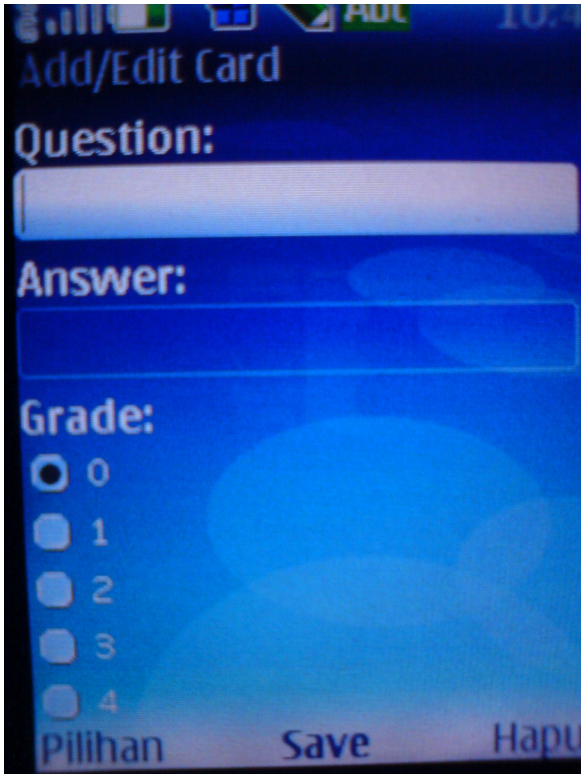
Pengujian pada perangkat bergerak dilakukan terhadap beberapa vendor dan tipe telepon seluler. Aplikasi dapat terpasang dan digunakan dengan baik pada Siemens ME75, Sony Ericsson M600, Nokia E51, Nokia E61i, dan Nokia 6282 Classic. Gambar 4.59 hingga 4.63 menunjukkan tampilan aplikasi saat beroperasi.



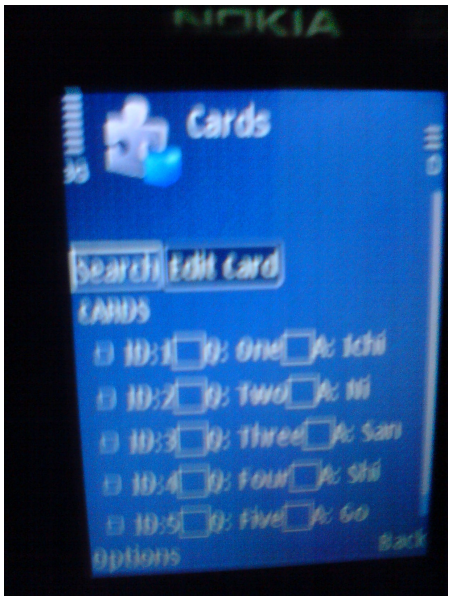
Gambar 4.59 Syahfi pada
Siemens ME75



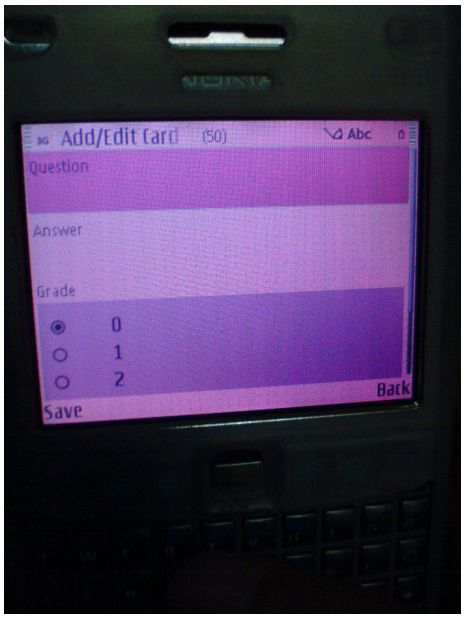
Gambar 4.60 Syahfi pada SE M600



Gambar 4.61 Syahfi pada 6282 Classic



Gambar 4.62 Syahfi pada Nokia E51



Gambar 4.63 Syahfi pada E61i