BAB IV

IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini akan dijelaskan tahapan proses implementasi sistem beserta pembahasannya berdasarkan hasil analisa kebutuhan dan perancangan perangkat lunak yang telah dibuat.

1. Lingkungan Implementasi

Implementasi dilakukan untuk proses perubahan dari bentuk representasi rancangan ke dalam bahasa pemrograman sehingga dapat dimengerti oleh komputer. Lingkungan implementasi yang akan dijelaskan meliputi lingkungan implementasi perangkat keras dan lingkungan implementasi perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan pada penelitian ini adalah sebuah PC (*Personal Computer*) dengan spesifikasi sebagai berikut :

1. *Processor* *Intel*® Core i3 2.40 GHz
2. Memori 2 GB
3. *Harddisk* kapasitas 500 GB
4. Monitor 14.0”
5. *Keyboard*
6. *Mouse*

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan pada penelitian ini adalah sebagai berikut:

1. Sistem operasi *Microsoft Windows 7 Ultimate*.
2. *Netbeans* IDE 8.0 dan JDK 1.8.0 dalam mengembangkan aplikasi.
3. MySQL, ysitu perangkat lunak untuk manajemen basis data SQL.
4. Implementasi Program

Berdasarkan rancangan pembuatan sistem pada bab 3, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut. Secara garis besar proses dikelompokkan menjadi 6 tahap, yaitu tahap *frequent 1-itemset*, tahap seleksi data, tahap pembangunan *FP-Tree*, tahap *FP-Growth*, tahap *generate rule,* dantahap *lift ratio*. Implementasi program terdiri dari *class-class* yang dijelaskan beserta fungsinya pada Tabel 4.1.

Tabel 4.1 *Class* dan fungsinya pada implementasi program

|  |  |
| --- | --- |
| Nama *Class* | Fungsi |
| *Class* Utama | Mengatur tampilan GUI dan semua proses sistem.  Terdiri dari *method* sebagai berikut:  1. jButton5ActionPerformed  2. Pilihan  3. jButton6ActionPerformed |
| *Class* Spesifikasi\_Smartphone | Main program atau menjalankan program.  Terdiri dari *method* main. |
| *Class* KoneksiDB | Mengatur koneksi ke *database*.  Terdiri dari *method* sebagai berikut:  1. getConnection  2. getData  3. dataTable1 |
| *Class* Item | Menyimpan ID, nama, frekuensi, dan *support* dari tiap *item*. Dan perhitungan frekuensi dan *support* tiap *item* beserta pengurutan *descending*-nya.  Terdiri dari *method* sebagai berikut:  1. FrequencyCount  2. SupportCount  3. SortItems |
| *Class* FPGrowthAlgorithm | Melakukan perhitungan *frequent 1-itemset*, melakukan seleksi data yang memenuhi *frequent 1-itemset*.  Terdiri dari *method* sebagai berikut:  1. FPGrowthAlgorithm  2. fpgrowth  3. fpgrowthMoreThanOnePath  4. addAllCombinationsForPathAndPrefix  5. saveItemset  6. get*Database*Size  7. Dataselect |
| *Class* Itemsets | Menyimpan itemset pada setiap level.  Terdiri dari *method* sebagai berikut:  1. Itemsets  2. addItemset  3. getLevels  4. getItemsetsCount  5. setName |
| *Class* Itemset | Menyimpan nilai *support* dari *frequent itemset* dan pemanggilan hasil *itemset*..  Terdiri dari *method* sebagai berikut:  1. Itemset  2. getItems  3. getSupportAB  4. size  5. get  6. setSupportAB  7. getSupportFreqItemset\_AsString  8. toString |
| *Class* FPTree | Melakukan pembentukan struktur data FPTree.  Terdiri dari *method* sebagai berikut:  1. addTransaction  2. fixNodeLinks  3. addPrefixPath  4. createHeaderList |
| *Class* FPNode | Menyimpan informasi node pada FPTree.  Terdiri dari *method* getChildWithID |
| *Class* GenerateRule | Melakukan perhitungan *support* dan *confidence* dari *frequent itemset* yang telah terbentuk. Dan pembangkitan *rule* beserta perhitungan *lift ratio*.  Terdiri dari *method* sebagai berikut:  1. Algorithm  2. SupportCount  3. Comparator  4. cloneItemSetMinusOneItem |
| *Class* AddRule | Menyimpan hasil *frequent itemset* yang dibangkitkan menjadi rule.  Terdiri dari *method* sebagai berikut:  1. addRule  2. getRulesCount  3. getRules |
| *Class* Rule | Menyimpan hasil itemset1 dan itemset2 dari *rule* beserta nilai *confidence* dan *lift ratio* nya.  Terdiri dari *method* sebagai berikut:  1. getLift  2. getSupportConsequent  3. getConfidence  4. getSupportAntecedent  5. toString  6. getItemset1  7. getItemset2 |

1. Implementasi Pembacaan Data Spesifikasi

Proses awal yang dilakukan adalah pembacaan data spesifikasi dari *database*. Implementasi dari proses ini ditunjukkan pada *source code* 4.1.

|  |
| --- |
| public boolean dataTable1(String qry){  try {  Statement st = (Statement) koneksi.createStatement();  ResultSet rs;  rs = st.executeQuery(qry);  ResultSetMetaData meta = (ResultSetMetaData) rs.getMetaData();  int col = meta.getColumnCount();  int baris = 0;  while (rs.next()) {  baris = rs.getRow();  }  String[] columns = {"no", "memoriinternal", "ram", "corecpu", "clockspeedcpu", "gpu" };  Data =new String[baris][col];  int x = 0;  rs.beforeFirst();  while (rs.next()) {  for (int i=0;i<columns.length;i++){\  Data[x][i] = rs.getString(columns[i]);  }  x++;  }  st.close();  rs.close();  return true;    } catch ( Exception ex) {  return false;  }  } |

*Source code* 4.1 Proses pembacaan data spesifikasi

Pada *method* dataTable1, dilakukan pengambilan data dari *database*. Inputan berupa query pada *method* ini digunakan sebagai perintah untuk mendapatkan data dalam *database*. Pengambilan data dilakukan dengan melakukan koneksi ke dalam *database* dan pengeksekusian perintah *query*. Pengambilan data dilakukan dengan mengambil data setiap kolom pada setiap baris dari *database* dengan perintah rs.getString(columns[i]). Data setiap kolom pada setiap baris dari *database* tersebut kemudian disimpan dalam variabel String Data[][]. Pengambilan data dari *database* ini dilakukan selama masih ada baris dalam *database* yang masih terisi data.

1. Implementasi Perhitungan Frekuensi Item

Proses selanjutnya setelah mendapatkan keseluruhan data spesifikasi dari *database* adalah proses perhitungan frekuensi setiap item. Implementasi dari proses ini ditunjukkan pada *source code* 4.2.

|  |
| --- |
| public static int FrequencyCount(String[][] Data, ArrayList<Item> item ) {  ArrayList<String> indexitem = new ArrayList<>();  int DataCount=0;    for (int i = 0; i < Data.length; i++) {  String var = Data[i][1];  if (!indexitem.contains(var)) {  indexitem.add(var);  item.add(new Item(item.size()+1,var, 1, 0));  }else {  int indexArr = indexitem.indexOf(var);  Item Item = item.get(indexArr);  Item.Frequency++;  item.set(indexArr, Item);  }  }  for (int i = 0; i < Data.length; i++) {  String var = Data[i][2];  if (!indexitem.contains(var)) {  indexitem.add(var);  item.add(new Item(item.size()+1,var, 1, 0));  }else {  int indexArr = indexitem.indexOf(var);  Item Item = item.get(indexArr);  Item.Frequency ++;  item.set(indexArr, Item);  }  }  for (int i = 0; i < Data.length; i++) {  String var = Data[i][3];  if (!indexitem.contains(var)) {  indexitem.add(var);  item.add(new Item(item.size()+1,var, 1, 0));  }else {  int indexArr = indexitem.indexOf(var);  Item Item = item.get(indexArr);  Item.Frequency ++;  item.set(indexArr, Item);  }  }  for (int i = 0; i < Data.length; i++) {  String var = Data[i][4];  if (!indexitem.contains(var)) {  indexitem.add(var);  item.add(new Item(item.size()+1,var, 1, 0));  }else {  int indexArr = indexitem.indexOf(var);  Item Item = item.get(indexArr);  Item.Frequency ++;  item.set(indexArr, Item);  }  }  for (int i = 0; i < Data.length; i++) {  String var = Data[i][5];  if (!indexitem.contains(var)) {  indexitem.add(var);  item.add(new Item(item.size()+1,var, 1, 0));  }else {  int indexArr = indexitem.indexOf(var);  Item Item = item.get(indexArr);  Item.Frequency ++;  item.set(indexArr, Item);  }  DataCount++;  }  return DataCount;  } |

*Source code* 4.2 Proses perhitungan frekuensi item

Pada proses ini, data dari dalam variabel String[][] Data digunakan untuk mendapatkan setiap item data spesifikasi. Dimana yang pertama kali dilakukan yaitu melakukan perulangan untuk setiap baris data. Dalam perulangan tersebut data yang pertama dipakai yaitu data pada kolom 1 ( Data[i][1] ) disimpan dalam variabel var. Lalu dilakukan pengecekan jika sebelumnya item pada kolom tersebut belum pernah ada dalam ArrayList<String> indexitem, maka item tersebut akan disimpan beserta ID nya dan frekuensi dari item tersebut diberi nilai 1. Apabila item tersebut sebelumnya telah ada dalam *array* indexitem, maka hanya akan melakukan penambahan nilai frekuensi dari item itu. Proses yang sama akan dilakukan terhadap seluruh kolom lainnya yang terdapat dalam variabel String[][] Data.

1. Implementasi Perhitungan *Support* Item

|  |
| --- |
| public static void SupportCount(ArrayList<Item> item, int DataCount) {  for (int i = 0; i < item.size(); i++) {  Item Item = item.get(i);  Item.Support = (double) Item.Frequency / DataCount;  item.set(i, Item);  }  **}** |

Proses selanjutnya setelah perhitungan frekuensi tiap item adalah proses perhitungan *support* setiap item. Implementasi dari proses ini ditunjukkan pada *source code* 4.3.

*Source code* 4.3 Proses perhitungan *support* item

Dalam proses perhitungan *support* item ini, dilakukan pembagian antara frekuensi setiap item melalui variabel Item.Frequency dengan jumlah data spesifikasi melalui variabel DataCount. Hasil pembagian tersebut kemudian disimpan dalam variabel Item.Support. Setelah itu hasil perhitungan *support* tadi disimpan ke dalam ArrayList<Item> item melalui perintah item.set(i, Item).

1. Implementasi Pencarian *Frequent 1-Itemset*

Proses selanjutnya adalah proses pecarian *frequent 1-itemset.* Pada proses ini item-item yang memiliki nilai *support* yang memenuhi *minimum support* akan dijadikan *frequent 1-itemset*. Implementasi dari proses ini ditunjukkan pada *source code* 4.4.

|  |
| --- |
| for(int i=1;i<=mapSupport.size();i++){  if(mapSupport.get(i)>= relativeMinsupp){  Freq1itemset.add(i);  }  } |

*Source code* 4.4 Proses pencarian *frequent 1-itemset*

Pada proses pecarian *frequent 1-itemset* ini apabila *support* item i yang terdapat dalam mapSupport.get(i) memenuhi *minimum support* yang terdapat dalam variabel relativeMinsupp maka item tersebut akan disimpan ke dalam ArrayList<Integer> Freq1itemset.

1. Implementasi Seleksi Data Spesifikasi

Pada proses ini dilakukan penyeleksian data spesifikasi. Dimana item dari data spesifikasi yang memenuhi *frequent 1-itemset* yang akan diambil sedangkan item yang tidak memenuhi *frequent 1-itemset* akan dihilangkan. Kemudian item-item setiap data spesifikasi akan diurutkan berdasarkan nilai *support* terbesar. Implementasi dari proses ini ditunjukkan pada *source code* 4.5.

|  |
| --- |
| for (int i = 0; i < DataIndex.size(); i++) {  List<Integer> transaction = new ArrayList<Integer>();  ArrayList<Integer> DataArray = DataIndex.get(i);  for (int j = 0; j < Freq1itemset.size(); j++) {  if (DataArray.get(0)==(Freq1itemset.get(j))) {  transaction.add(DataArray.get(0));  }  if (DataArray.get(1)==(Freq1itemset.get(j))) {  transaction.add(DataArray.get(1));  }  if (DataArray.get(2)==(Freq1itemset.get(j))) {  transaction.add(DataArray.get(2));  }  if (DataArray.get(3)==(Freq1itemset.get(j))) {  transaction.add(DataArray.get(3));  }  if (DataArray.get(4)==(Freq1itemset.get(j))) {  transaction.add(DataArray.get(4));  }  }  datselect.add(transaction);  Collections.sort(transaction, new Comparator<Integer>() {  public int compare(Integer item1, Integer item2) {  int compare = mapSupport.get(item2) - mapSupport.get(item1);  if (compare == 0) {  return (item1 - item2);  }  return compare;  }  }); |

*Source code* 4.5 Proses seleksi data spesifikasi

Proses seleksi data spesifikasi ini dilakukan dengan menyeleksi item-item pada setiap baris data spesifikasi yang disimpan dalam ArrayList<Integer> DataArray. Apabila item yang terdapat dalam DataArray tersebut sama dengan item yang terdapat dalam ArrayList<Integer> Freq1itemset, maka menunjukkan bahwa item tersebut memenuhi *frequent 1-itemset* sehingga akan disimpan dalam List<Integer> transaction.

Setelah didapatkan seluruh data spesifikasi yang telah melalui seleksi data maka item-item dalam data spesifikasi tersebut diurutkan berdasarkan nilai *support* tertinggi melalui fungsi *compare*. Pada fungsi *compare* ini dibandingkan apabila nilai *support* item kedua yang terdapat pada mapSupport.get(item2) lebih besar dari nilai *support* item pertama yang terdapat pada mapSupport.get(item1), maka akan dikembalikan urutan item dengan item 2 menempati urutan pertama sedangkan item 1 menempati urutan selanjutnya. Apabila sebaliknya maka yang dikembalikan adalah urutan item dengan item 1 menempati urutan pertama sedangkan item 2 menempati urutan selanjutnya.

1. Implementasi Pembangunan *FP-Tree*

Pada proses ini dilakukan pembangunan *FP-Tree*. Dimana data spesifikasi yang telah melalui proses penyeleksian dan pengurutan akan dibangun menjadi *tree*. Implementasi dari proses ini ditunjukkan pada *source code* 4.6.

|  |
| --- |
| public void addTransaction(List<Integer> transaction) {  FPNode currentNode = root;  for (Integer item : transaction) {  FPNode child = currentNode.getChildWithID(item);  if (child == null) {  FPNode newNode = new FPNode();  newNode.itemID = item;  newNode.parent = currentNode;  currentNode.childs.add(newNode);  if (!hasMoreThanOnePath && currentNode.childs.size() > 1) {  hasMoreThanOnePath = true;  }  currentNode = newNode;  fixNodeLinks(item, newNode);  }else {  child.counter++;  currentNode = child;  }  }  } |

*Source code* 4.6 Proses pembangunan *FP-Tree*

Dalam proses pembangunan *FP-Tree* ini digunakan *method* addTransaction untuk memasukkan data spesifikasi ke dalam *tree*. Langkah pertama pembangunan *tree* tersebut yaitu untuk item pertama dari data spesifikasi pertama dalam List<Integer> transaction akan menjadi newNode.parent. Sedangkan item selanjutnya akan menjadi currentNode.childs. Proses yang sama terus berjalan hingga item terakhir sehingga *FP-Tree* akan terbentuk secara lengkap.

1. **Implementasi *Conditional Pattern Base***

Pada proses ini dilakukan pencarian *conditional pattern base* dari *FP-Tree* yang telah dibangun. Dimana dari lintasan-lintasan yang telah terbentuk dalam *FP-Tree* dicari *prefix path* dan *suffix path* nya. Implementasi dari proses ini ditunjukkan pada *source code* 4.7.

|  |
| --- |
| List<List<FPNode>> prefixPaths= new ArrayList<List<FPNode>>();  FPNode path = tree.mapItemNodes.get(item);  while (path != null) {  if (path.parent.itemID != -1) {  List<FPNode> prefixPath = new ArrayList<FPNode>();  prefixPath.add(path);  FPNode parent = path.parent;  while (parent.itemID != -1) {  prefixPath.add(parent);  parent = parent.parent;  }  prefixPaths.add(prefixPath);  }  path = path.nodeLink;  }  Map<Integer, Integer> mapSupportBeta = new HashMap<Integer, Integer>();  for (List<FPNode> prefixPath : prefixPaths) {  int pathCount = prefixPath.get(0).counter;  for (int j = 1; j < prefixPath.size(); j++) {  FPNode node = prefixPath.get(j);  if (mapSupportBeta.get(node.itemID) == null) {  mapSupportBeta.put(node.itemID, pathCount);  } else {  mapSupportBeta.put(node.itemID, mapSupportBeta.get(node.itemID) + pathCount);  }  }  } |

*Source code* 4.7 Proses *conditional pattern base*

Proses *conditional pattern base* ini dilakukan dengan melakukan pencarian lintasan yang telah terbentuk melalui pembanguna *FP-Tree* dan *prefix path* yaitu node-node yang masih memiliki anak. Pada setiap lintasan, node-node yang memiliki anak disimpan dalam variabel List<FPNode> prefixPath sebagai *prefix* *path*. Lalu masing-masing *prefix path* tersebut dicari nilai *support*-nya dan disimpan dalam variabel Map<Integer, Integer> mapSupportBeta.

1. Implementasi *Conditional FP-Tree*

Pada proses ini dilakukan pencarian *conditional FP-Tree* dari *conditional pattern base* yang telah didapatkan. Dimana *prefix path* dan *suffix path* yang telah didapatkan dilakukan penyeleksian nilai *support*-nya. Apabila kombinasi *prefix path* dan *suffix path* tersebut memiliki nilai *support* yang memenuhi nilai *minimum support* maka akan disimpan menjadi *frequent itemset*. Keluaran dari proses ini berupa *frequent itemset*. Implementasi dari proses ini ditunjukkan pada *source code* 4.8.

|  |
| --- |
| void addPrefixPath(List<FPNode> prefixPath, Map<Integer, Integer> mapSupportBeta, int relativeMinsupp) {  int pathCount = prefixPath.get(0).counter;  FPNode currentNode = root;  for (int i = prefixPath.size() - 1; i >= 1; i--) {  FPNode pathItem = prefixPath.get(i);  if (mapSupportBeta.get(pathItem.itemID) < relativeMinsupp) {  continue;  }  FPNode child = currentNode.getChildWithID(pathItem.itemID);  if (child == null) {  FPNode newNode = new FPNode();  newNode.itemID = pathItem.itemID;  newNode.parent = currentNode;  newNode.counter = pathCount;  currentNode.childs.add(newNode);  if (!hasMoreThanOnePath && currentNode.childs.size() > 1) {  hasMoreThanOnePath = true;  }  currentNode = newNode;  fixNodeLinks(pathItem.itemID, newNode);  } else {  child.counter += pathCount;  currentNode = child;  }  }  } |

*Source code* 4. 8 Proses *conditional FP-Tree*

Yang pertama kali dilakukan pada proses *conditional FP-Tree* adalah melakukan seleksi item-item dalam *prefix path* yang memiliki nilai *support* yang memenuhi *minimum support*. Lalu item-item dalam *prefix path* yang telah memenuhi *minimum support* dibangun menjadi *tree* dan hasil pembangunan *conditional FP-Tree* tersebut merupakan *frequent itemset*.

1. **Implementasi *Generate Rule***

Pada proses ini, *frequent itemset* yang telah didapatkan dicari nilai *confidence-*nya. *Frequent itemset* yang memenuhi *minimum confidence* akan dibangkitkan menjadi *rule*. Implementasi dari proses ini ditunjukkan pada *source code* 4.9.

|  |
| --- |
| int[] itemset\_Antecedent = cloneItemSetMinusOneItem(lk.getItems(), item);  double support = SupportCount(itemset\_Antecedent);  double support\_Antecedent = (double) support;  double conf = lk.getSupportAB() / support\_Antecedent;  if (conf < minconf || Double.isInfinite(conf)) {  continue;  **}** |

*Source code* 4.9 Proses *generate rule*

Dalam proses *generate rule* ini terlebih dahulu dilakukan pencarian *itemset antecedent* melalui fungsi cloneItemSetMinusOneItem kemudian *itemset* tersebut disimpan dalam int[] itemset\_Antecedent. Setelah itu dilakukan perhitungan *support* dari *itemset antecedent* tersebut. Untuk perhitungan nilai *confidence*-nya dilakukan dengan membagi nilai *support* dari *rule* yang didapatkan melalui pemanggilan lk.getSupportAB() dengan nilai *support* dari *itemset antecedent* yang telah dihitung sebelumnya. Kemudian setelah didapatkan nilai *confidence* maka dilakukan penyeleksian, jika nilai *confidence* dari *frequent* *itemset* tersebut kurang dari *minimum confidence* maka *frequent itemset* tersebut tidak akan disimpan menjadi *rule*, sedangkan jika nilai *confidence*-nya memenuhi nilai *minimum confidence* maka *frequent itemset* tersebut akan disimpan menjadi *rule* beserta nilai *confidence*-nya.

1. Implementasi Perhitungan *Lift Ratio*

Pada proses ini, *rule* yang telah didapatkan dihitung nilai *lift ratio*-nya. *Rule* yang memiliki nilai *lift ratio* diatas 1 yang akan disimpan. Implementasi dari proses ini ditunjukkan pada *source code* 4.10.

|  |
| --- |
| double support\_Consequent = SupportCount(itemset\_Consequent);  double benchmark\_conf = ((double) support\_Consequent );  double lift = ((conf) / benchmark\_conf);  if (lift < minlift) {  continue;  }  rules.addRule(new Rule(itemset\_Antecedent, itemset\_Consequent, support, support\_Consequent, conf, lift)); |

*Source code* 4.10 Proses perhitungan *lift ratio*

Dalam proses perhitungan *lift ratio* ini terlebih dahulu dilakukan perhitungan *benchmark confidence*. *Benchmark confidence* didapatkan dari nilai *support itemset consequent*. Nilai *benchmark confidence* yang telah didapatkan disimpan dalam variabel benchmark\_conf. Kemudian untuk perhitungan *lift ratio*-nya didapatkan dengan membagi nilai *confidence* yang tersimpan dalam variabel conf dengan nilai *benchmark confidence* yang telah disimpan dalam variabel benchmark\_conf. Setelah didapatkan nilai *lift ratio* maka dilakukan penyeleksian *rule*. Jika *rule* tersebut memiliki nilai *lift ratio* yang memenuhi nilai *minimum lift ratio* yaitu 1, maka *rule* beserta nlai *lift ratio*-nya akan disimpan.

**4.3 Implementasi Antarmuka**

Implementasi antar muka sistem terdiri dari bagian *input*, tombol proses, tombol *clear*, dan bagian *output*. Bagian *input* terdiri dari 3 data *input*, yaitu *input* jenis sistem operasi, *input* *minimum support* dan *input* *minimum confidence*. Bagian *output* terdiri dari 8 submenu, antara lain: submenu data spesifikasi, submenu pengkodean, submenu transformasi data, submenu *1-itemset*, submenu *frequent 1-itemset*, submenu data seleksi, submenu *frequent itemset*, dan submenu hasil *rule*.

Untuk menggunakan sistem, bagian *input* terlebih dahulu diisi. Pada bagian *input*, jenis sistem operasi digunakan untuk memilih data yang akan dipakai sesuai dengan jenis sistem operasi yang dimasukkan yaitu Android atau Windows Phone. Pada bagian *input* *minimum support* diisi dengan angka antara 1 hingga 100. Begitu juga pada bagian *input* *minimum confidence* juga diisi angka dengan *range* antara 1 hingga 100. Setelah memasukkan ketiga *input*-an data kemudian ketika tombol proses ditekan maka data *output* pada submenu data spesifikasi, submenu pengkodean, submenu transformasi data, submenu *1-itemset*, submenu *frequent 1-itemset*, submenu data seleksi, submenu *frequent itemset*, dan submenu hasil *rule* akan ditampilkan pada masing-masing tabel. Apabila tombol *clear* ditekan maka seluruh data dalam tabel pada masing-masing submenu akan hilang.

Tampilan pada submenu data spesifikasi terdiri dari 1 tabel yang digunakan untuk menampilkan data spesifikasi secara keseluruhan. Tabel ini terdiri dari 6 kolom, yaitu kolom nomor, memori internal, RAM, core CPU, clock speed CPU dan GPU. Dimana masing-masing kolom tersebut menampilkan data sesuai dengan jenis spesifikasinya. Tampilan submenu data spesifikasi diilustrasikan pada Gambar 4.1.



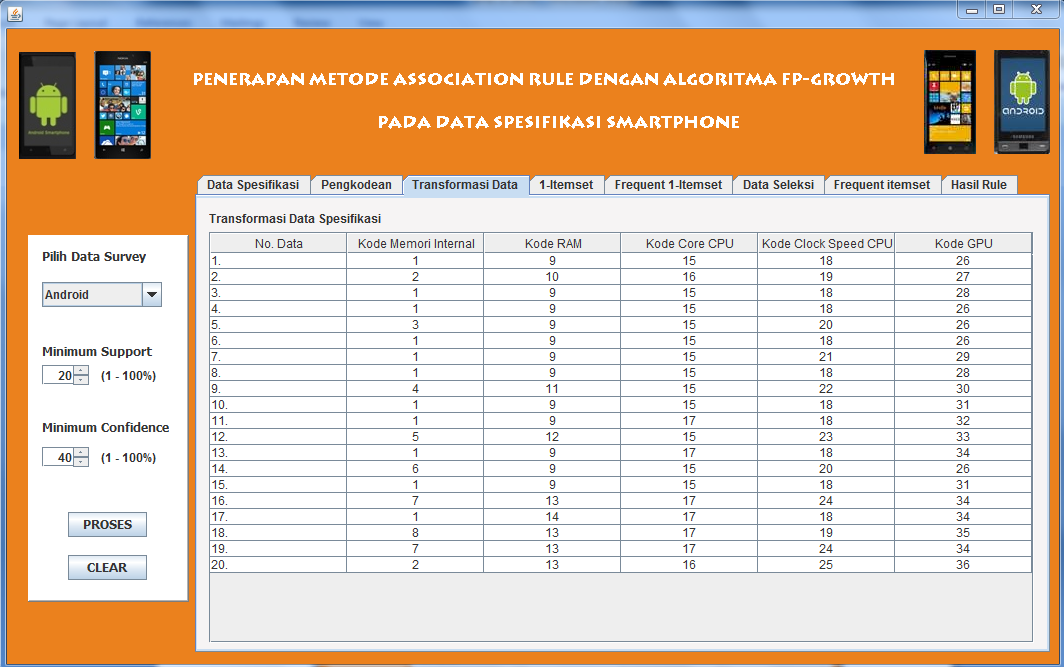
Gambar 4.1 Tampilan submenu data spesifikasi

Tampilan pada submenu pengkodean terdiri dari 1 tabel yang menampilkan pengkodean setiap item pada data spesifikasi. Tabel tersebut terdiri dari 2 kolom, yaitu kolom kode yang digunakan untuk menampilkan kode tiap item dan kolom items yang digunakan untuk menampilkan item spesifikasi. Tampilan submenu pengkodean diilustrasikan pada Gambar 4.2.



Gambar 4.2 Tampilan submenu pengkodean

Tampilan pada submenu transformasi data terdiri dari 1 tabel yang menampilkan hasil transformasi data spesifikasi ke dalam bentuk kode. Tabel tersebut terdiri dari 6 kolom, yaitu nomor, memori internal, RAM, core CPU, clock speed CPU dan GPU. Masing-masing kolom tersebut menampilkan spesifikasi yang telah ditransformasikan ke dalam kode. Tampilan submenu transformasi data diilustrasikan pada Gambar 4.3.



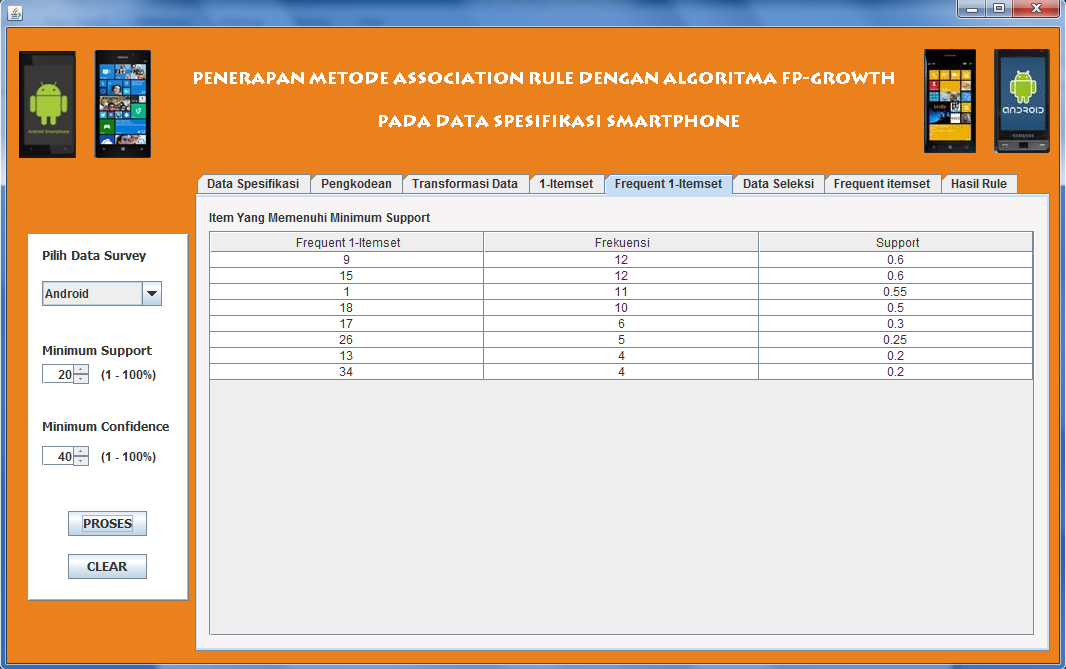
Gambar 4.3 Tampilan submenu transformasi data

Tampilan pada submenu 1-itemset terdiri dari 1 tabel yang menampilkan hasil perhitungan frekuensi dan *support* dari masing-masing items. Tabel tersebut terdiri dari 3 kolom, yaitu kolom kode items, kolom frekuensi dan kolom *support*. Pada kolom kode items digunakan untuk menampilkan kode setiap item, kolom frekuensi untuk menampilkan frekuensi setiap kode items, dan kolom *support* untuk menampilkan nilai *support* dari masing-masing kode items. Tampilan submenu 1-itemset diilustrasikan pada Gambar 4.4.



Gambar 4.4 Tampilan submenu *1-itemset*

Tampilan pada submenu *frequent 1-itemset* terdiri dari 1 tabel. Tabel ini menampilkan hasil penyeleksian item yang memiliki *support* yang memenuhi nilai *minimum* *support*. Tabel tersebut terdiri dari 3 kolom, yaitu kolom *frequent 1-itemset*, kolom frekuensi dan kolom *support*. Pada kolom *frequent 1-itemset* digunakan untuk menampilkan kode item yang menjadi *frequent 1-itemset*, kolom frekuensi digunakan untuk menampilkan frekuensi setiap *frequent 1-itemset*, dan kolom *support* digunakan untuk menampilkan nilai *support* dari masing-masing *frequent 1-itemset*. Tampilan submenu *frequent 1-itemset* diilustrasikan pada Gambar 4.5.



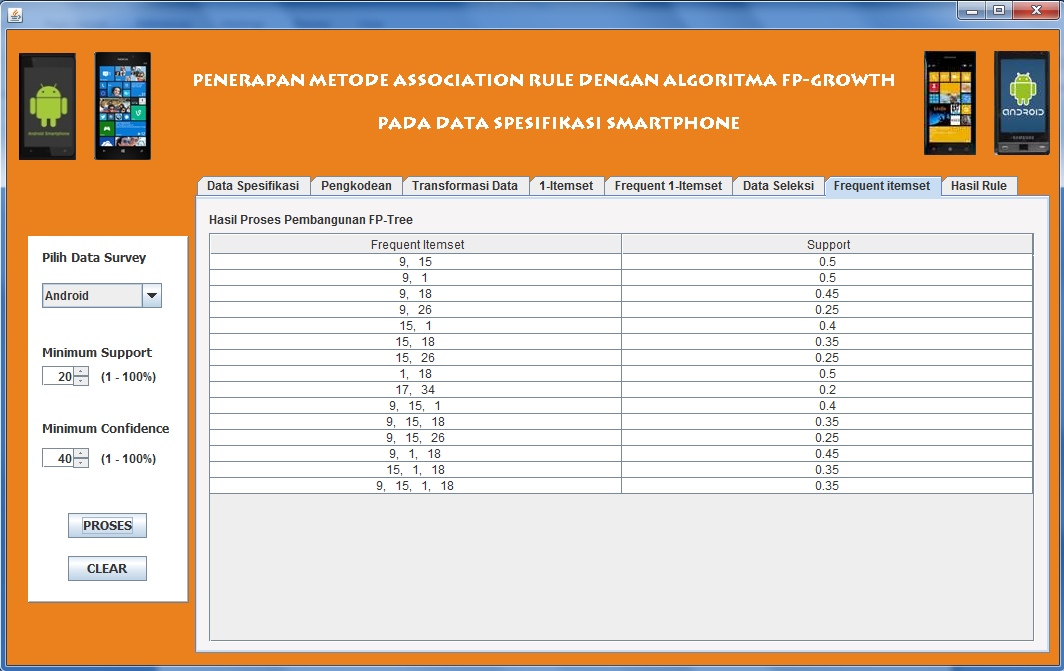
Gambar 4.5 Tampilan submenu *frequent 1-itemset*

Tampilan pada submenu data seleksi terdiri dari 1 tabel untuk menampilkan hasil seleksi data spesifikasi yang memenuhi *frequent 1-itemset*. Tabel tersebut terdiri dari 2 kolom, yaitu kolom nomor yang menampilkan nomor data spesifikasi, dan kolom hasil data seleksi untuk menampilkan hasil penyeleksian data spesifikasi yang telah diurutkan berdasarkan *support* terbesar. Tampilan submenu data seleksi diilustrasikan pada Gambar 4.6.



Gambar 4.6 Tampilan submenu data seleksi

Tampilan pada submenu *frequent itemset* terdiri dari 1 tabel untuk menampilkan *frequent itemset* hasil dari pembangunan *tree*. Tabel tersebut terdiri dari 2 kolom, yaitu kolom *frequent itemset* yang menampilkan data *frequent itemset*, dan kolom *support* yang menampilkan *support* dari *frequent itemset*. Tampilan submenu *frequent itemset* diilustrasikan pada Gambar 4.7.



Gambar 4.7 Tampilan submenu *frequent itemset*

Tampilan pada submenu hasil *rule* terdiri dari 2 tabel. Pada tabel pertama digunakan untuk menampilkan hasil *rule* beserta nilai *confidence* dan *lift ratio*nya. Tabel pertama tersebut terdiri dari 3 kolom, yaitu kolom *rule*, kolom *confidence*, dan kolom *lift ratio*. Kolom *rule* digunakan untuk menampilkan hasil *frequent itemset* yang telah dibangkitkan menjadi *rule*. Kolom *confidence* digunakan untuk menampilkan nilai *confidence* dari *rule*, dan kolom *lift ratio* digunakan untuk menampilkan hasil perhitungan *lift ratio* dari *rule*. Pada tabel kedua terdiri dari satu kolom yaitu kolom keterangan *rule* yang digunakan untuk menampilkan keterangan dari *rule* yang terbentuk. Tampilan submenu hasil *rule* diilustrasikan pada Gambar 4.8.



Gambar 4.8 Tampilan submenu hasil *rule*