BAB III

PELAKSAAN KERJA MAGANG

3.1 Kedudukan Dan Koordinasi

Kerja magang dilakukan selama 2 bulan lebih 1 minggu di Setjen DPR RI pada divisi Bidang Data dan Sarana Informasi yang berlokasi di Jl. Jenderal Gatot Subroto, Jakarta. Kedudukan yang didapat pada kerja magang ini adalah sebagai *web programmer*. Dalam pelaksanaannya, penulis berada dalam suatu tim *web programmer*, dimana tim ini terdiri dari tiga orang *programmer*. Dua orang *programmer* lainnya juga berasal dari universitas yang sama dengan penulis, yaitu Universitas Multimedia Nusantara.

Anggota dari tim *web programmer* ini terdiri dari penulis Noer Muhammad Ghozali sebagai *web programmer* yang bertanggung jawab dalam membangun modul transaksi dan modul *search* pada *website online shop* koperasi DPR RI, Deandrha Agusta sebagai *web programmer* yang bertanggung jawab dalam membangun modul katalog produk dan *layout* pada *website online shop* koperasi DPR RI, dan Fachrin Hafizh Fauzan yang bertanggung jawab dalam membangun modul *login* dan modul kategori pada *website online shop* koperasi DPR RI. Setiap anggota tim mempunyai tanggung jawabnya masing – masing untuk setiap modul yang ada. Dengan bertanggung jawabnya setiap anggota tim terhadap modulnya masing – masing, setiap tim harus memastikan modul yang dibuat dapat berjalan dengan baik tanpa adanya *bug* maupun *error*.

Adapun *function – function* yang ada di dalam modul transaksi yang menjadi tanggung jawab penulis adalah sebagai berikut :

1) *Function Add to Cart*
2) *Function Delete From Cart*
3) *Function Add to Wishlist*
4) *Function Delete From Wishlist*
5) *Function Add to Cart From Wishlist*
6) *Function Check Out*
7) *Function Confirmation Order*
8) *Function Invoice*
9) *Function Order History*

Sedangkan, *function* yang ada di dalam modul *search* yang menjadi tanggung jawab penulis adalah *function search product*.

Koordinasi yang dilakukan oleh setiap anggota tim adalah berkomunikasi dalam setiap pengerjaan modulnya karena folder aplikasi yang digunakan adalah folder aplikasi yang sama dimana menggunakan *intranet* untuk menghubungkan satu komputer dengan komputer lainnya. Jadi, folder yang digunakan merupakan *shared folder,* dimana jika dilakukan modifikasi pada suatu *file* di satu komputer, maka pada komputer lain yang terhubung dengan jaringan *intranet* ini juga akan mengalami perubahan modifikasi pada *file* tersebut.

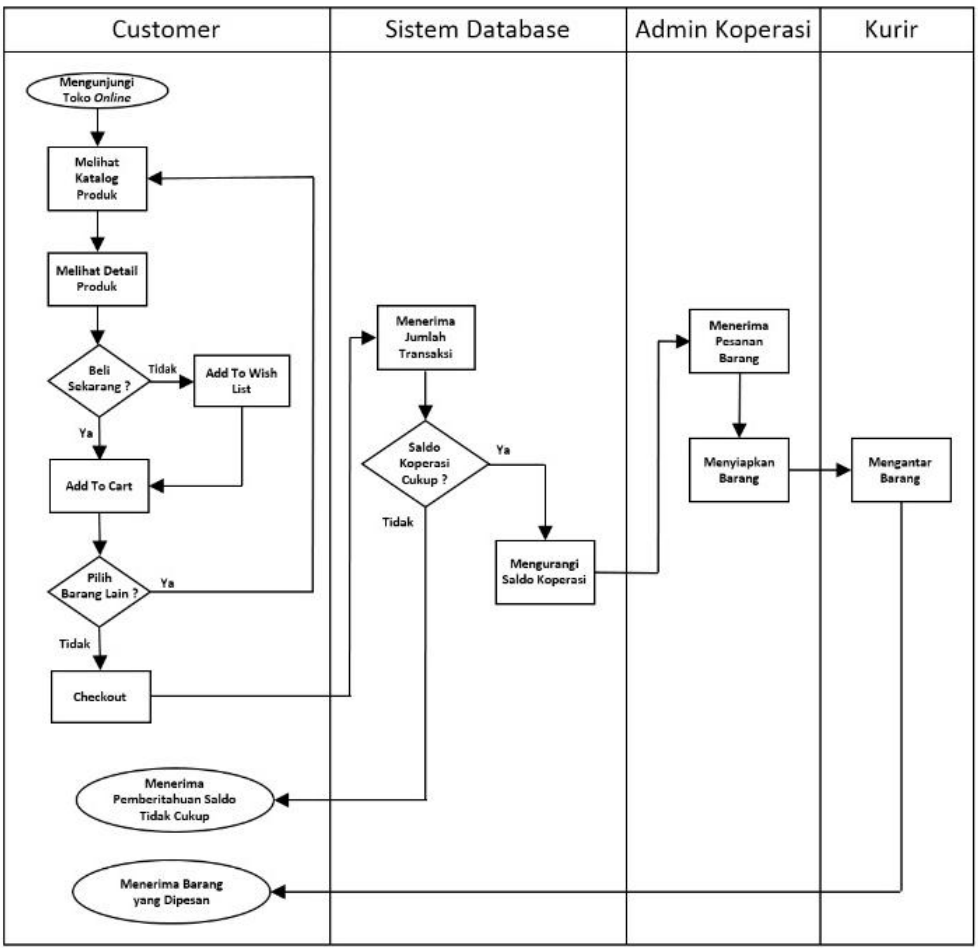
Dengan diterapkannya *shared folder,* antara anggota tim diharuskan saling berkomunikasi untuk berkoordinasi dengan baik agar pengembangan *website* tidak

saling bertabrakan. Contoh pada kasus ini adalah jika satu komputer melakukan modifikasi *file* master, dan komputer lainnya juga melakukan modifikasi terhadap *file* master yang sama dan pada waktu yang bersamaan pula, maka yang melakukan *'save'* terlebih dahulu lah yang akan tersimpan hasil modifikasinya. Tanpa koordinasi yang baik, pengembangan *website* tidak akan berjalan dengan optimal karena modifikasi yang dilakukan pada suatu *file* tertentu bisa saja tidak berarti jika pada komputer lain telah dilakukan *'save'* terlebih dahulu.

Penggunaan *shared folder* bertujuan untuk memudahkan anggota tim ketika ingin menjalankan *website* secara keseluruhan untuk pengetesan karena setiap modul telah diletakkan pada folder yang sama. Hal ini juga dapat mempercepat pengerjaan pembangunan *website* karena setiap anggota tim tidak perlu memindahkan atau mengumpulkan *file*-nya di satu tempat yang sama untuk dilakukan pengetesan. Hal ini sangat penting karena pengetesan *website* akan dilakukan setiap modifikasi *file* telah selesai.

3.2. Perancangan Proses Bisnis

Sebelum mengembangkan toko *online* ini, sangat penting untuk mengetahui proses bisnis yang berjalan pada koperasi. Perancangan toko *online* ini berdasarkan informasi dari pembimbing lapangan tentang proses bisnis yang akan digunakan. Untuk melihat lebih detail tentang proses bisnis yang digunakan maka dapat melihat Gambar 3.1 yang berisi proses bisnis menggunakan *Flowchart* Diagram.



Gambar 3.1 Proses Bisnis *Flowchart* Diagram

3.3. Timeline Pekerjaan

Waktu yang dibutuhkan dalam kegiatan kerja magang ini kurang lebih selama 10 minggu. Dalam rentang waktu tersebut, terdapat beberapa tugas yang dilakukan sesuai dengan waktu yang dibutuhkannya. *Timeline* pekerjaan pada kegiatan kerja magang ini terdapat pada Tabel 3.1.

Tabel 3.1 *Timeline* Pekerjaan

Tugas yang Dilakukan	Minggu Ke -									
	1	2	3	4	5	6	7	8	9	10
Pembentukan *Database*										
Pencarian *Template*										
Pencarian *Framework*										
Mempelajari *Zend Framework*										
Membuat *Services* untuk Modul Transaksi										
Membuat *Controller* untuk Modul Transaksi										
Menyesuaikan *View* Modul Transaksi										
Membuat *Services* untuk Modul *Search*										
Membuat *Controller* untuk Modul *Search*										
Menyesuaikan *View* Modul *Search*										

3.4. Tugas yang Dilakukan

Tugas yang dilakukan oleh penulis selama melakukan kerja magang di Setjen DPR RI pada divisi Bidang Data dan Sarana Informasi secara rincinya adalah sebagai berikut :

a. Pembentukan *Database*

Hal pertama yang dilakukan oleh penulis ketika mendapat tugas untuk membangun toko *online* bagi koperasi DPR RI adalah mempersiapkan *database* apa saja yang akan dibutuhkan. Pada tahap awal ini, penulis mengumpulkan berbagai tabel dan kolom yang sekiranya dibutuhkan untuk pengembangan *website* toko *online*. Setiap tabel yang dibutuhkan dibuat hubungannya dengan tabel yang lain. Tabel yang dibuat adalah berdasarkan kebutuhan *function* dari DPR RI.

b. Pencarian *Template*

Pada awalnya, ketika penulis mendapatkan tugas untuk membuat *online shop* bagi koperasi DPR RI, penulis menginginkan penggunaan *template* maupun *Content Management System* yang selanjutnya disebut sebagai CMS sebagai dasar dari pengembangan *online shop*-nya. Contoh CMS yang akan digunakan oleh penulis adalah *PrestaShop* dan *OpenCart*.

c. Pencarian *Framework*

Pada proses ini, dilakukan pencarian *Framework* sebagai kerangka kerja dari pengembangan *online shop* berbasis *website*. Pencarian *Framework* ditujukan untuk memudahkan pengembangan *website* karena pengembangan dapat dilakukan sesuai dengan kerangka kerja yang telah ditetapkan.

d. Mempelajari *Zend Framework*

Pada proses ini, penulis mempelajari *Zend Framework* dimana pada proses sebelumnya telah didapatkan hasil pencarian *Framework,* yaitu *Zend Framework*. *Zend Framework* termasuk ke dalam hal yang baru bagi penulis. Oleh sebab itu, penulis mempelajari terlebih dahulu tentang *Zend Framework* pada bagian struktur maupun komponen – komponennya. Dan pada akhirnya diketahui bahwa *Zend Framework* menggunakan *Model, View, Controller Pattern* (MVC *Pattern*).

e. Membuat *Services* untuk Modul Transaksi

Pada *Zend Framework,* terdapat komponen *Services* yang merupakan komponen untuk menghubungi aplikasi dengan *database*. Komponen *Services* berisi berbagai macam *query* yang dapat digunakan oleh aplikasi. *Query* yang ada pada *Services* menggunakan *query* dengan struktur yang dimiliki oleh *Zend Framework*, tidak seperti *query* MySQL pada umumnya. Pada proses ini, *Services* dibuat untuk menyediakan *query database* yang dibutuhkan oleh modul transaksi. *Services* yang dibuat untuk modul transaksi ini adalah :

1. *PosService*
2. *PosDataService*
3. *StokService*

f. Membuat *Controller* untuk Modul Transaksi

Setelah membuat *Services*, selanjutnya adalah membuat *Controller* yang digunakan untuk melakukan kontrol terhadap apa saja data yang akan ditampilkan oleh komponen *View*. Komponen ini dibuat untuk menghubungkan komponen *Services* dengan komponen *View*. Dengan menggunakan *Controller* maka data yang diinginkan dapat dipanggil oleh *Services* dan ditampilkan oleh komponen *View*. *Controller* yang dibuat untuk modul transaksi ini adalah :

1. *ViewCartController*
2. *WishListController*
3. *InvoiceController*
4. *OrderHistoryController*

g. Menyesuaikan *View* Modul Transaksi

Setelah *Controller* dan *Services* selesai dibuat, tahap terakhir adalah menyesuaikan *View* atau tampilan dengan fungsionalitas yang ada pada modul transaksi itu sendiri. Komponen *View* ini berisi halaman yang menjadi tampilan atau *User Interface* dari *website* toko *online*. Data yang ditampilkan oleh komponen *View* ini berasal dari komponen *Controller* yang memanggil data dalam *database* melalui komponen *Services*. Secara garis besar, seperti itulah cara kerja MVC *Pattern*. Komponen *View* yang harus disesuaikan untuk tampilan dalam modul transaksi adalah :

1. *View-cart*
2. *Wish-list*
3. *Checkout*
4. *Invoice*
5. *Order-History*

h. Membuat *Services* untuk Modul *Search*

Selain *Services* untuk modul transaksi, *Services* untuk modul *Search* juga harus dibuat. Hal ini karena setiap modul membutuhkan *file Services*-nya masing – masing yang dapat memanggil data dari *database* berdasarkan apa yang dibutuhkan oleh modul tersebut. *Services* yang dibuat untuk menjalankan modul *Search* berada dalam *StokService*, sehingga komponen *StokService* harus dimodifikasi agar dapat digunakan oleh modul *Search* ini.

i. Membuat *Controller* untuk Modul *Search*

Pembuatan *Controller* ini bertujuan untuk menghubungkan *Services* pada modul *Search* dengan komponen *View*. *Controller* ini akan mengatur segala hal yang berhubungan dengan *function* yang ada pada modul *Search*. *Controller* yang dibuat untuk modul *Search* ini adalah *SearchController*.

j. Menyesuaikan *View* Modul *Search*

Setelah komponen *Services* dan *Controller* untuk modul *Search* dibuat, selanjutnya adalah menyesuaikan tampilan pada *website* toko *online*

koperasi DPR RI pada modul *Search*. Komponen *View* ini akan menampilkan segala hal yang telah diatur dalam komponen *Controller* untuk modul *Search*.

3.5. Uraian Pelaksanaan Kerja Magang

Dalam pelaksanaan kerja magang ini, proses pelaksanaan dari tugas – tugas yang telah disebutkan pada bagian sebelumnya adalah sebagai berikut :

3.5.1 Pembentukan *Database*

Setelah penulis beserta tim pengembang *website* mendapatkan tugas untuk membuat toko *online*, hal pertama yang dilakukan oleh penulis adalah membuat dan menyusun *database*. Penulis membuat struktur *database* beserta tabel – tabelnya yang dibutuhkan dalam pengembangan toko *online* ini. Dan setelah selesai menyusun struktur *database*-nya, maka didapatkan beberapa tabel dalam *database* yang dibutuhkan oleh penulis untuk membuat modul Transaksi dan modul *Search*.

Tabel – tabel yang dibutuhkan oleh penulis untuk membuat modul Transaksi dan modul *Search* adalah :

1) Tabel Stok.

   Tabel ini digunakan untuk menyimpan keseluruhan data produk yang dijual oleh koperasi DPR RI. Tabel stok ini dibutuhkan oleh penulis untuk pembuatan modul *Search*, dimana *function* dari modul *Search* ini akan memanggil produk – produk yang dicari oleh *customer*. Tidak hanya itu, tabel stok ini juga dapat digunakan untuk

menampilkan katalog produk yang berada pada modul lainnya. Pada modul yang lain, katalog produk yang ditampilkan tidak hanya mengambil data dari tabel Stok ini saja, tapi juga mengambil data dari tabel Kategori dan Jenis. Namun, pada modul *Search* ini, tabel yang dibutuhkan hanya tabel Stok ini saja. Tabel ini sebelumnya sudah tersedia pada *database* di *server* DPR RI, struktur *database* yang belum disesuaikan adalah seperti pada Tabel 3.2.

Tabel 3.2 Struktur awal dari tabel stok yang belum disesuaikan

No.	Nama	Jenis
1.	id	int(11)
2.	kode	varchar(255)
3.	nama	varchar(255)
4.	sku	varchar(255)
5.	barcode	varchar(255)
6.	jumlah_minimum	int(11)
7.	jumlah	int(11)
8.	satuan	varchar(11)
9.	harga	double
10.	persentase_margin	double
11.	deskripsi	varchar(255)
12.	nomor_rak	varchar(11)
13.	user_input	varchar(100)
14.	tanggal_input	datetime
15.	user_update	varchar(100)
16.	tanggal_update	datetime

Setelah dilakukan penyesuaian agar *database* dapat digunakan, struktur baru dari tabel stok dapat dilihat pada Gambar 3.2.

#	Nama	Jenis	Penyortiran	Atribut	Kosong	Bawaan
1	id	int(11)			Tidak	*Tidak ada*
2	kode	varchar(11)	utf8_general_ci		Ya	*NULL*
3	nama	varchar(50)	utf8_general_ci		Ya	*NULL*
4	sku	varchar(30)	utf8_general_ci		Ya	*NULL*
5	barcode	int(15)			Ya	*NULL*
6	kategori	varchar(30)	utf8_general_ci		Ya	*NULL*
7	id_kategori	int(11)			Ya	*NULL*
8	id_merek	int(11)			Ya	*NULL*
9	id_satuan	int(11)			Ya	*NULL*
10	jumlah_minimum	int(11)			Ya	*NULL*
11	jumlah	int(11)			Ya	*NULL*
12	satuan	varchar(11)	utf8_general_ci		Ya	*NULL*
13	harga_beli	double			Ya	*NULL*
14	harga_jual	double			Ya	*NULL*
15	persentase_margin	double			Ya	*NULL*
16	deskripsi	varchar(255)	utf8_general_ci		Ya	*NULL*
17	nomor_rak	varchar(11)	utf8_general_ci		Ya	*NULL*
18	status	tinyint(1)			Ya	1
19	user_input	varchar(50)	utf8_general_ci		Ya	*NULL*
20	tanggal_input	datetime			Ya	*NULL*
21	user_update	varchar(50)	utf8_general_ci		Ya	*NULL*
22	tanggal_update	datetime			Ya	*NULL*

Gambar 3.2 Struktur tabel Stok

2) Tabel *Pos*Data.

Tabel ini digunakan untuk menyimpan data transaksi yang terjadi pada toko *online* ini. Pada tabel *Pos*Data, keseluruhan data yang disimpan merupakan data hasil olahan dari modul Transaksi, dimana untuk membedakan data berdasarkan kondisinya (apakah berada dalam *My cart* atau *My Wish-list*, dan lain sebagainya) menggunakan kolom status. Jadi, setiap data transaksi yang berubah kondisinya, kolom status pada data tersebut juga ikut berubah. Struktur awal database yang belum disesuaikan adalah seperti pada Tabel 3.3.

Tabel 3.3 Struktur awal dari tabel *Pos*Data yang belum disesuaikan

No.	Nama	Jenis
1.	id	int(11)
2.	id_pos	int(11)
3.	jumlah	int(11)
4.	harga	double
5.	total	double
6.	user_input	varchar(100)
7.	tanggal_input	datetime
8.	user_update	varchar(100)
9.	tanggal_update	datetime

Status yang terdapat pada tabel ini adalah sebagai berikut :

a. Status ‘1’, merupakan status yang digunakan untuk menunjukkan bahwa transaksi yang dilakukan *customer* adalah memasukkan produk ke dalam *My cart*.
b. Status ‘2’, merupakan status yang digunakan untuk menunjukkan bahwa transaksi yang dilakukan *customer* adalah memasukkan produk ke dalam *Wish-list.*
c. Status ‘3’, merupakan status yang digunakan untuk menunjukkan bahwa transaksi yang dilakukan oleh *customer* adalah melakukan pemesanan terhadap produk yang bersangkutan dan produk sedang dalam proses pengiriman.
d. Status ‘4’, merupakan status yang digunakan untuk menunjukkan bahwa transaksi yang dilakukan oleh *customer*

sudah berakhir, dimana produk yang dipesan oleh *customer* sudah diterima dan data transaksi masuk ke dalam *function* '*Order History'*.

Struktur dari tabel *Pos*Data dapat dilihat pada Gambar 3.3.

#	Nama	Jenis	Penyortiran	Atribut	Kosong	Bawaan
1	id	int(11)			Tidak	*Tidak ada*
2	id_pos	int(11)			Ya	*NULL*
3	id_stok	int(11)			Ya	*NULL*
4	jumlah	int(11)			Ya	*NULL*
5	harga	decimal(10,0)			Ya	*NULL*
6	total	decimal(10,0)			Ya	*NULL*
7	status	tinyint(1)			Ya	1
8	user_input	varchar(50)	utf8_general_ci		Ya	*NULL*
9	tanggal_input	datetime			Ya	*NULL*
10	user_update	varchar(50)	utf8_general_ci		Ya	*NULL*
11	tanggal_update	datetime			Ya	*NULL*

Gambar 3.3 Struktur Tabel *Pos*Data

3.5.2 Pencarian *Template*

Pada tugas ini, penulis beserta tim *website programmer* mencari *template* di *internet* untuk digunakan sebagai dasar dari pembuatan *website* toko *online* koperasi DPR RI. Setiap anggota di dalam tim mencari *template* terbaik yang bisa digunakan dan mengumpulkannya sehingga didapat beberapa *template*. Setelah *template* tersebut dikumpulkan, tim mencari *template* terbaik untuk selanjutnya diterapkan sebagai dasar pembuatan toko *online* koperasi DPR RI.

Pada saat itu, penulis mencoba untuk menggunakan *template* yang berupa CMS (*Content Management System*), dimana keseluruhan *function* yang dibutuhkan dalam toko *online* sudah tersedia dan bisa dijalankan secara langsung. Penulis mencoba *template PrestaShop* dan menyesuaikannya dengan kebutuhan toko *online* koperasi DPR RI.

Dalam *PrestaShop, function* yang dimiliki terbilang cukup banyak dan lengkap. Dimulai dari *function* untuk menambah produk hingga *function* untuk memberikan pemberitahuan jika terdapat pesanan dari *customer*, keseluruhan *function* tersebut sudah tersedia dan bisa dijalankan secara langsung. Yang perlu dilakukan untuk menyesuaikan *template PrestaShop* ini dengan toko *online* koperasi DPR RI adalah menonaktifkan beberapa *function* maupun fitur yang dianggap tidak perlu.

Penulis mencoba untuk menjalankan *template PrestaShop* ini menggunakan *tool* XAMPP dan dijalankan secara *offline* melalui *localhost.* Tampilan *Front-end* dari *template PrestaShop* ini sangat menarik dengan desain tampilan yang sangat baik dan modern.

Namun, dibalik keindahan tampilan *Front-end* yang dimiliki oleh *PrestaShop*, tampilan *Back-end* yang digunakan oleh admin untuk mengatur operasional toko *online* terasa begitu rumit dan terlalu kompleks. Meskipun demikian, tampilan *Back-end* yang dimilikinya juga dapat dikatakan baik. Dengan begitu banyaknya *function* dan fitur yang dimiliki oleh *PrestaShop* pada sisi *Back-end*, terdapat kemungkinan bagi admin merasa kesulitan dalam melakukan operasional toko *online*.

Template yang ditemukan oleh anggota tim lain juga merupakan *template* yang berupa CMS, dimana pengoperasian pada sisi *Back-end* juga dirasa terlalu rumit dan kompleks. Berdasarkan pertimbangan tersebut, pembimbing lapangan dari Setjen DRP-RI memutuskan untuk tidak menggunakan *template* berbasis CMS sebagai dasar dari pembuatan toko *online* koperasi DPR RI ini. DPR RI menginginkan toko *online* yang mudah digunakan baik dari sisi *Front-end* maupun dari sisi *Back-end*.

DPR RI tidak menginginkan toko *online* yang mempunyai terlalu banyak *function* dan fitur yang dapat membuat admin merasa kesulitan untuk menggunakannnya. Karena *template* berbasis CMS dirasa terlalu rumit dan kompleks, maka DPR RI menginginkan pembuatan toko *online* dari awal, dimana setiap *function* dan fiturnya dibuat sendiri dan disesuaikan dengan kebutuhan yang telah ditentukan.

3.5.3 Pencarian *Framework*

Setelah pencarian *template* dilakukan dan didapatkan hasil yang tidak begitu memuaskan, tugas selanjutnya adalah mencari *Framework* yang cocok untuk digunakan dalam pengembangan *website* toko *online* untuk koperasi DPR RI ini. Tujuan dari penggunaan *Framework* adalah sebagai kerangka kerja dimana dapat memudahkan pengembangan *website* karena pengembangan *website* dilakukan secara terstruktur menggunakan *Framework* yang bersangkutan.

Pada pencarian *Framework* ini, penulis bersama tim menemukan *Framework CodeIgniter* yang dirasa cukup baik untuk digunakan. Setelah mempelajari sedikit tentang kegunaan dan fungsi dari *Framework CodeIgniter*, pembimbing lapangan dari Setjen DPR RI memutuskan untuk menggunakan *Zend Framework* sebagai kerangka kerja pengembangan *website* toko *online* ini. Pemilihan *Zend Framework* ini didasarkan pada penggunaan *Zend Framework* sebagai *Framework* dasar dalam sistem IT yang ada di dalam DPR RI secara keseluruhan. Diharapkan dengan menggunakan *Zend Framework* untuk *website* toko *online* ini, toko *online* koperasi DPR RI dapat diintegerasikan ke sistem IT DPR RI dengan mudah.

3.5.4 Mempelajari *Zend Framework*

Setelah mengetahui bahwa *Framework* yang akan digunakan dalam pengembangan *website* toko *online* ini adalah *Zend Framework*, penulis beserta tim langsung mencoba dan mempelajari *Zend Framework* itu sendiri. Dimulai dari bagaimana struktur yang dimiliki oleh *Zend Framework* ini, hingga bagaimana format penulisan *query* yang digunakan dalam *Zend Framework*.

Zend Framework merupakan *Framework* yang menggunakan MVC *Pattern* (*Model*, *View*, *Controller Pattern*), dimana setiap komponen tersebut mempunyai fungsinya masing – masing yang tidak saling bersinggungan satu sama lain. Namun, *Zend Framework* tidak hanya menggunakan ketiga komponen itu saja, terdapat satu komponen lagi yang berfungsi untuk

menampung berbagai macam *query* yang dibutuhkan oleh aplikasi untuk memanggil data dari *database*, sehingga komponen secara keseluruhan yang digunakan oleh *Zend Framework* adalah sebagai berikut :

1) Komponen *Model*.

   Komponen ini berfungsi sebagai penghubung antara aplikasi ataupun *function* ke *database*. Tanpa komponen *model* ini, *query* yang dijalankan oleh aplikasi tidak akan mampu memanggil data dalam *database*. Setiap tabel dalam *database* mempunyai komponen *Model*-nya masing – masing, sehingga data dalam setiap tabel dapat dipanggil oleh aplikasi dan ditampilkan dalam halaman *website*. Dengan kata lain, fungsi dari komponen *Model* ini adalah untuk mendeklarasikan lokasi tabel yang berada pada *database* sehingga aplikasi dapat memanggil data melalui komponen *Services* menggunakan komponen *Model* ini. Contoh komponen *model* yang digunakan untuk memanggil data dari tabel '*pos_data*' adalah seperti pada Gambar 3.4.

```php
<?php
class PosData extends Zend_Db_Table {
    protected $_name;
    protected $_schema;
    protected $_adapter;
    protected $_db;

    public function init()
    {
        $this->_name = 'pos_data';
        $this->_schema  = 'db_toko';
        $this->_adapter = 'db_toko';
        $this->_db = Zend_Registry::get('db_toko');
    }
}
?>
```

Gambar 3.4 Komponen *Model* untuk Tabel *pos_data*

2) Komponen *View*.

Komponen ini berfungsi untuk mengatur desain yang akan ditampilkan oleh halaman *website*. Setiap halaman mempunyai komponen *View*-nya masing – masing. Komponen *View* pada setiap halaman ditentukan dalam suatu folder yang bernama halaman itu sendiri dan di dalamnya terdapat *file index.phtml* yang merupakan *file* utama pada halaman tersebut. Contoh kutipan *function* yang berada di komponen *View* untuk menampilkan daftar *invoice* pada halaman '*Invoice*' dapat dilihat pada Gambar 3.5.

```
    foreach($this->rowsItemInvoice as $row1)
{

?>

    <tr style="font-size:12;">

    <td><b><? echo ''. $row1->id .'' ?></b></td>
    <td><b><? echo ''. $row1->nama .'' ?></b></td>
    <td align="center"><? echo ''. $row1->jumlah .'' ?></td>
    <td align="center"> <? echo 'Rp. '. $this->FormatNumber($row1->harga, 2) .'' ?></td>
    <td align="center"><? echo 'Rp. '. $this->FormatNumber($row1->total, 2) .'' ?></td>
    </tr>

<? }
```

Gambar 3.5 Kutipan *Function* dalam Komponen *View*

3) Komponen *Controller*.

Komponen ini berfungsi untuk mengatur data yang akan ditampilkan oleh komponen *View*. Apa yang ditampilkan oleh komponen *View* adalah berasal dari komponen *Controller* ini. Sehingga tanpa komponen *Controller*, komponen *View* tidak akan dapat menampilkan data apapun dari *database*, karena yang mengatur alur data adalah komponen *Controller*. Dan setiap halaman yang terdapat

dalam *website* toko *online* ini mempunyai komponen *Controller*-nya masing – masing, seperti contohnya halaman '*View-Cart*' mempunyai *Controller* yang bernama '*ViewCartController*'. Contoh *function* dalam komponen *Controller* yang digunakan untuk menampilkan data pada *index* dari halaman *Wish-list* terdapat pada Gambar 3.6.

```
public function indexAction()
{
    //$user_log = Zend_Auth::getInstance()->getIdentity()->nama;
    //$username = $this->getIdentity()->nama;
    $this->view->rowsItemUser = $this->PosDataService->getAllDataUser();
    $this->view->rowsItemWish = $this->PosDataService->getAllDataJoinWish();
    $this->view->rowsItem = $this->PosDataService->getAllDataJoin();
    $this->view->rows = $this->PosService->getAllData();
    $this->view->KategoriRows = $this->KategoriService->getAllData();
    $this->view->MerekRows = $this->MerekService->getAllData();
    $this->view->RecentRows = $this->StokService->getAllData();
    $this->view->jenisRows = $this->JenisService->getAllData();

    //$this->_redirect('/login');
}
```

Gambar 3.6 *Function* dalam Komponen *Controller*

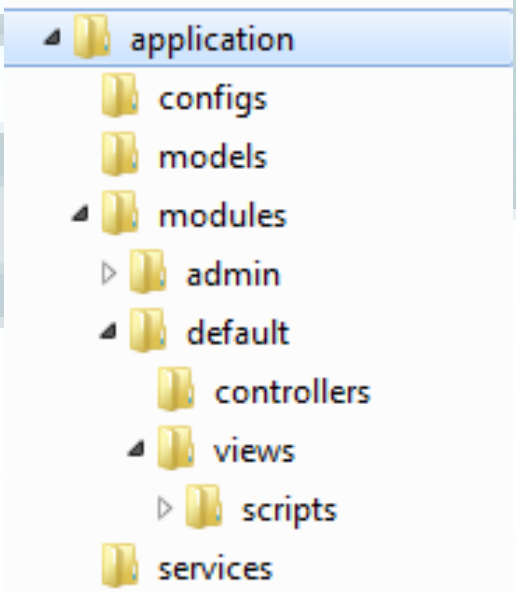
4) Komponen *Services*.

Komponen ini merupakan komponen tambahan yang dimiliki oleh *Zend Framework*. Komponen ini berisi berbagai macam *query* yang dibutuhkan oleh aplikasi untuk memanggil data dalam *database*. Setiap tabel yang berada dalam *database* mempunyai komponen *Services*-nya masing – masing, seperti contohnya pada tabel *pos_data*, maka komponen *Services*-nya adalah *Pos*Data*Service*. Contoh *query* yang terdapat dalam komponen *Services* ini adalah seperti pada Gambar 3.7.

```
function getAllData()
{
    $select = $this->stok->select()
        ->setIntegrityCheck(false)
        ->from('stok', array('*'))
        ->where('status = 1')
        ->order('tanggal_update desc')
        ->limit(4, 0);
    $result = $this->stok->fetchAll($select);
    return $result;
}
```

Gambar 3.7 *Query* dalam Komponen *Services*

Berdasarkan komponen – komponen yang dimiliki oleh *Zend Framework* tersebut, maka struktur folder dalam *Zend Framework* adalah seperti pada Gambar 3.8.



Gambar 3.8 Struktur Folder dalam *Zend Framework*

Pada Gambar 3.8, seluruh komponen *Model* terdapat dalam folder '*models*'. Komponen *View* dan *Controller* yang digunakan pada *Front-end* dari aplikasi toko *online* ini berada pada dalam folder default, dimana seluruh komponen *Controller*-nya berada pada folder '*Controllers*' dan seluruh

komponen *View*-nya berada pada folder '*View*s → *script*s'. Hal ini juga berlaku untuk komponen *View* dan *Controller* bagi tampilan *Back-end* dari aplikasi toko *online* ini, dimana komponen *View* dan *Controller* nya berada pada folder 'admin'. Sedangkan, keseluruhan *Services* yang berisi *query – query* yang dibutuhkan oleh aplikasi berada pada folder '*Services*'.

Setelah mempelajari *Zend Framework*, dapat disimpulkan beberapa perbedaan mendasar antara *Zend Framework* dengan *Framework* lain. *Framework* lain yang dijadikan sebagai bahan pembanding adalah *CodeIgniter*, karena penulis sempat mempelajari *CodeIgniter* terlebih dahulu sebelum akhirnya mempelajari *Zend Framework*. Beberapa perbedaan mendasar dari *Zend Framework* dan *Framework CodeIgniter* terdapat pada Tabel 3.4.

Tabel 3.4 Perbandingan *Zend Framework* dengan *CodeIgniter*

Zend Framework	***CodeIgniter***	***PrestaShop***
Hanya Mendukung PHP versi 5	Mendukung PHP versi 4 & 5	Hanya Mendukung PHP versi 5
Kustomisasi terbatas pada Arsitektur MVC *Pattern*	Kustomisasi lebih luas dari aturan Arsitektur MVC *Pattern*	Kustomisasi hanya terbatas pada *website* berbasis *e-commerce*
Pengembangan sangat sulit karena format pemrograman berbeda	Pengembangan *CodeIgniter* dengan PHP tidak terlalu sulit	Pengembangan sangat mudah dengan adanya *Plugin* yang tersedia
Cakupan pengembangan sangat luas tergantung pada *skill* pengembang	Cakupan pengembangan sangat luas dan lebih mudah	Pengembangan *Function* terbatas pada *Plugin* yang tersedia

Berdasarkan perbandingan yang ditampilkan pada Tabel 3.4, dapat disimpulkan bahwa pengembangan *website* dengan menggunakan *Zend Framework* mempunyai tingkat kesulitan yang lebih tinggi dibanding dengan menggunakan *Framework CodeIgniter* maupun CMS *PrestaShop*. Hal ini karena *Zend Framework* mempunyai format bahasa PHP yang berbeda, sehingga membutuhkan *skill* PHP yang lebih tinggi dibanding dengan *CodeIgniter*. Namun, pengembangan dengan *Zend Framework* maupun dengan *CodeIgniter* dapat mencakup pengembangan yang sangat luas dibanding dengan *PrestaShop* karena *PrestaShop* hanya terbatas pada pengembangan *website* berbasis *e-commerce*.

3.5.5 Membuat *Services* untuk Modul Transaksi

Pada bagian ini, penulis membuat *Services* untuk Modul Transaksi. *Services* yang dibuat pada modul transaksi ini dibagi menjadi beberapa bagian, yaitu :

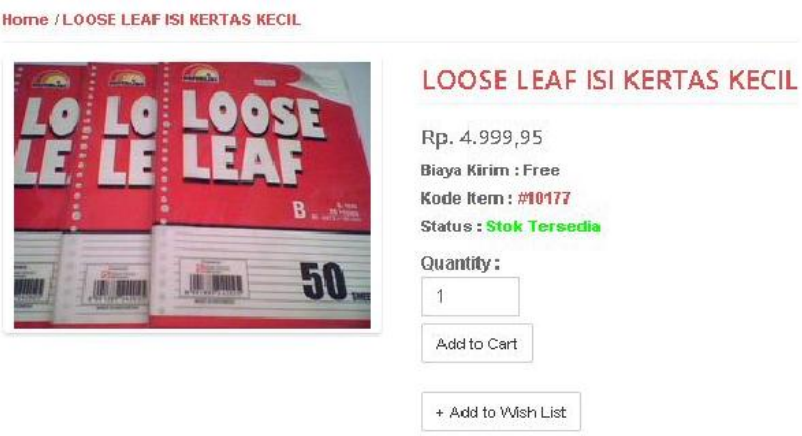
1. *PosService*
2. *PosDataService*
3. *StokService*

Ketiga *Services* di atas mempunyai fungsinya masing – masing sesuai dengan *function* yang akan dibuat pada Modul Transaksi. Dengan kata lain, *function – function* yang ada pada Modul Transaksi akan menggunakan *query – query* yang berada pada ketiga *Services* tersebut. Pada bagian ini akan dijabarkan proses pembuatan *Services* untuk setiap *function* yang ada.

Sedangkan *function – function* yang ada pada Modul Transaksi adalah sebagai berikut :

1. *Add to Cart*

   Query yang digunakan pada *function Add to Cart* ini akan memasukkan data stok ke dalam tabel *pos_data*, dimana tabel *pos_data* ini merupakan tabel yang digunakan untuk aktivitas keseluruhan transaksi yang terjadi pada *website* toko *online* ini. *Function Add to Cart* ini berada pada halaman '*Single-item*', dimana *function* ini hanya dapat diakses jika pengguna masuk ke dalam halaman dari '*single-item*' yang berisi deskripsi lengkap dari suatu produk tertentu yang dipilih. Gambar 3.9 memperlihatkan tampilan *Add to Cart* untuk suatu produk yang berada pada halaman '*single-item*'.



Gambar 3.9 Tampilan *Add to Cart* dari Suatu Produk

Ketika tombol *Add to Cart* yang berada di bawah *field Quantity* diklik, maka detail dari produk tersebut akan disimpan ke dalam tabel *pos_data* dengan *user_input* berisi ID pengguna. Produk yang

dimasukkan ke dalam tabel *pos_data* ini akan mempunyai nilai status '1'. Parameter yang digunakan pada *query* tersebut diambil dari komponen *Controller*. Untuk memasukkan detail produk tersebut ke tabel *pos_data*, maka digunakan *query* seperti pada Gambar 3.10.

```
function addDataPos($id_stok, $jumlah, $harga, $total, $status)
{
    $user_log = Zend_Auth::getInstance()->getIdentity()->pengguna;
    $tanggal_log = date('Y-m-d H:i:s');
    $id_session = Zend_Controller_Front::getInstance()->getRequest()->getParam('id');


    $params = array(
        'id_stok' => $id_stok,
        'jumlah' => $jumlah,
        'harga' => $harga,
        'total' => $total,
        'status' => $status,
        'user_input' => $user_log,
        'tanggal_input' => $tanggal_log,
        'user_update' => $user_log,
        'tanggal_update' => $tanggal_log
    );
    $this->pos_data->insert($params);
    $lastId = $this->pos_data->getAdapter()->lastInsertId();
    return $lastId;
}
```

Gambar 3.10 *Query* yang Digunakan untuk *Function Add to Cart*

2. *View-Cart*,

Query yang digunakan dalam halaman *My cart* pada *function View-Cart* ini berada pada *PosDataServices*. *Query* ini akan menampilkan keseluruhan data produk yang telah dimasukkan oleh pengguna ke dalam keranjang belanjanya (*My cart*), *query* ini mengambil data yang pada kolom *user_input* sama dengan ID dari pengguna dan pada kolom status mempunyai nilai '1'. *Query* ini juga mengambil data dari tabel *pos_data* dan mencocokkan *id_stok* pada tabel *pos_data* dengan id pada tabel stok, tujuannya adalah untuk menampilkan nama produk yang

tidak terdapat pada tabel *pos_data*. *Query* yang digunakan dapat dilihat pada Gambar 3.11.

```
function getAllDataJoin()
{
    $user_log = Zend_Auth::getInstance()->getIdentity()->pengguna;

    $sql = "SELECT a.id, a.jumlah, a.harga, a.total, a.user_input,
            b.id, b.nama, a.status
            FROM db_toko.pos_data a
            JOIN db_toko.stok b ON a.id_stok = b.id
            WHERE a.status = '1' AND a.user_input = '" . $user_log . "'";

    $db = Zend_Registry::get('db');
    $stmt = $db->query($sql);
    $stmt->setFetchMode(Zend_Db::FETCH_OBJ);
    $result = $stmt->fetchAll();
    return $result;
}
```

Gambar 3.11 *Query* yang Digunakan pada *Function View-Cart*

3. *Add to Wish-list*

   Query yang digunakan pada *function Add to Wish-list* ini akan memasukkan data stok ke dalam tabel *pos_data*. Sama seperti *function Add to Cart*, ketika tombol *Add to Wish-list* diklik maka akan menyimpan detail produk ke tabel *pos_data*, namun status yang akan dimasukkan bernilai '2'. *Query* yang digunakan sama dengan *query* pada *function Add to Cart* seperti pada Gambar 3.9. yang membedakan hanyalah pada komponen *Controller*-nya dimana *Controller* pada *function* ini mempunyai nilai status '2', sehingga parameter yang digunakan bernilai '2'.

4. *Wish-list*

Query yang digunakan dalam halaman *Wish-list* ini berada pada *Pos*Data*Services*. *Query* ini akan menampilkan keseluruhan data produk yang telah dimasukkan oleh pengguna ke dalam daftar *Wish-list* atau produk yang diinginkan untuk dibeli pada lain kesempatan. *Query* ini mengambil data yang pada kolom *user_input* sama dengan ID dari pengguna dan pada kolom status mempunyai nilai '2'. Sama seperti *query* untuk *function View-Cart*, *Query* ini juga mengambil data dari tabel *pos_data* dan mencocokkan *id_stok* pada tabel *pos_data* dengan id pada tabel stok, tujuannya adalah untuk menampilkan nama produk yang tidak terdapat pada tabel *pos_data*. *Query* yang digunakan dapat dilihat pada Gambar 3.12.

```
function getAllDataJoinWish()
{
    $user_log = Zend_Auth::getInstance()->getIdentity()->pengguna;

    $sql = "SELECT a.id, a.jumlah, a.harga, a.total, a.user_input,
            b.id, b.nama
            FROM db_toko.pos_data a
            JOIN db_toko.stok b ON a.id_stok = b.id
            WHERE a.status = '2' AND a.user_input = '" . $user_log . "'";

    $db = Zend_Registry::get('db');
    $stmt = $db->query($sql);
    $stmt->setFetchMode(Zend_Db::FETCH_OBJ);
    $result1 = $stmt->fetchAll();
    return $result1;
}
```

Gambar 3.12 *Query* yang Digunakan pada *Function Wish-list*

5. *Checkout*

Query yang digunakan pada *function Checkout* akan mengubah status dari keseluruhan produk yang ada di *My cart*. Pada tabel *pos_data*,

status yang bernilai ‘1’ (produk berada dalam *My cart*) akan diubah menjadi bernilai ‘3’. Namun tidak semua data pada tabel *pos_data* yang mempunyai status ‘1’ diubah, tapi hanya data yang pada *field* ‘*user_input*’-nya sama dengan ID pengguna saja. Ketika statusnya menjadi ‘3’, maka produk – produk tersebut akan ditampilkan pada halaman Admin, dimana produk tersebut menjadi pesanan yang harus diproses oleh Admin. Parameter status yang digunakan dalam *query* ini adalah berasal dari komponen *Controller*, dimana komponen *Controller* inilah yang akan mengatur keseluruhan aktivitas data. *Query* yang digunakan untuk *function* ini dapat dilihat pada Gambar 3.13.

```
function editStatus($id_stok, $status)
{
    $user_log = Zend_Auth::getInstance()->getIdentity()->pengguna;
    $tanggal_log = date('Y-m-d H:i:s');
    $id_stok = Zend_Controller_Front::getInstance()->getRequest()->getParam('id');

    $params = array(
        'status' => $status,
        'user_update' => $user_log,
        'tanggal_update' => $tanggal_log
    );
    $where = array();
    $where[] = $this->pos_data->getAdapter()->quoteInto('id_stok = ?', $id_stok);
    $where[] = $this->pos_data->getAdapter()->quoteInto('user_input = ?', $user_log);
    $this->pos_data->update($params, $where);
}
```

Gambar 3.13 *Query* yang Digunakan pada *Function Checkout*

6. *Order History*

Function ini ditujukan untuk menampilkan *History* dari keseluruhan aktivitas belanja pengguna pada *website* toko *online* koperasi DPR RI. *Query* dari *function* ini berada pada *PosService*, dimana halaman *Order-History* ini akan menampilkan data yang berada pada tabel ‘*pos*’. Tabel ‘*pos*’ digunakan untuk menampung data pembelian produk oleh

pengguna per satu transaksi, bukan per satu produk seperti pada tabel '*pos_data*'. *Query* akan mengambil data transaksi sesuai dengan yang dimiliki oleh masing – masing pengguna. Dari data yang ditampilkan pada halaman *Order-History* ini, selanjutnya pengguna dapat menampilkan detail transaksi dari masing – masing *invoice* dengan menggunakan *function Invoice* pada poin selanjutnya. Dengan demikian, *query* yang digunakan untuk menampilkan *History* transaksi yang dilakukan oleh pengguna adalah seperti pada Gambar 3.14.

```
function getAllDataByCustomer()
{
    $user_log = Zend_Auth::getInstance()->getIdentity()->nama;

    $select = $this->pos->select()
        ->setIntegrityCheck(false)
        ->from('pos', array('*'))
        ->where('status <> 9')
        ->where('user_input = ?', $user_log);

    $result = $this->pos->fetchAll($select);
    return $result;
}
```

Gambar 3.14 *Query* yang Digunakan dalam *Function Order-History*

7. *Invoice*

Function ini akan menampilkan produk – produk yang telah dipesan oleh *customer* berdasarkan dari kode *invoice* yang dipilih. Untuk mengakses halaman yang berisi kumpulan *invoice*, pengguna harus membuka halaman *Order-History*, dimana halaman tersebut ditujukan untuk menampilkan *History* dari aktivitas belanja kita. Pembuatan kode beserta detail dari *Invoice* berada pada *PosService*, dan *PosDataService*

akan menampilkan keseluruhan data produk dari masing – masing *Invoice* yang dipilih oleh pengguna. *Query* yang digunakan untuk menampilkan detail dari *invoice* yang dipilih adalah seperti pada Gambar 3.15.

```
function getAllDataJoinInvoice($id_pos)
{
    $status = '3';
    $user_log = Zend_Auth::getInstance()->getIdentity()->pengguna;
    $id_pos = Zend_Controller_Front::getInstance()->getRequest()
    ->getParam('id');

    $sql = "SELECT a.id, a.jumlah, a.harga, a.total, a.user_input,
            b.id, c.nama, a.id_pos
            FROM db_toko.pos_data a
            JOIN db_toko.pos b ON a.id_pos = b.id
            JOIN db_toko.stok c ON a.id_stok = c.id
            WHERE a.status = '3' OR a.status = '4' AND a.id_pos =
            '" . $id_pos . "' AND a.user_input = '" . $user_log . "'";

    $db = Zend_Registry::get('db');
    $stmt = $db->query($sql);
    $stmt->setFetchMode(Zend_Db::FETCH_OBJ);
    $result1 = $stmt->fetchAll();
    return $result1;
}
```

Gambar 3.15 *Query* yang Digunakan untuk Menampilkan Detail *Invoice*

3.5.6 Membuat *Controller* untuk Modul Transaksi

Pada bagian ini, penulis membuat *Controller* yang digunakan oleh Modul Transaksi. Seperti yang telah dijelaskan sebelumnya bahwa pada Modul Transaksi ini terdapat beberapa *Function* yang digunakan. Jadi, pembuatan komponen *Controller* ini juga akan berdasarkan pada masing – masing *function* tersebut, yaitu sebagai berikut :

1. *Add to Cart*

   Controller untuk *function* ini berada pada *SingleItemController*, karena *function Add to Cart* ini berada pada halaman *Single-item*. *Syntax* yang

digunakan dalam komponen *Controller* ini supaya *query* pada komponen *Services* dapat dijalankan adalah seperti pada Gambar 3.16.

```
if ( $this->getRequest()->isPost() )
{
    $id_stok = $this->getRequest()->getParam('id');
    $id_session = $this->getRequest()->getParam('id');
    $harga = $this->_helper->CCur($this->getRequest()
    ->getParam('harga'));
    $jumlah = $this->getRequest()->getParam('jumlah');
    $total = $this->_helper->CCur($harga * $jumlah);

    if(isset($_POST['submit_cart'])) {
    $status = '1';

    $this->PosDataService->addDataPos($id_stok, $jumlah,
    $harga, $total, $status);

    $this->_redirect('/single-item/index/id/'.$id_stok);
```

Gambar 3.16 *Syntax* pada *Controller* untuk *Function Add to Cart*

Pada Gambar 3.16, yang dilakukan pertama kali oleh *Controller* adalah mendeklarasikan variabel – variabel yang selanjutnya akan dijadikan parameter untuk dijalankan oleh *query* pada komponen *Services*. Setelah itu, *Controller* menjalankan *function* addData*Pos* yang berada pada *Pos*Data*Service* untuk memasukkan detail produk ke dalam tabel *pos_data*.

2. *View-Cart*

Controller yang digunakan oleh *function* ini bernama *ViewCartController*. Untuk menampilkan data *My cart* pada *function View-Cart* ini, *syntax* yang digunakan adalah :

$this->View->rowsItem = $this->PosDataService->getAllDataJoin();

3. *Add to* Wish List

Controller untuk *function* ini berada pada SingleItem*Controller*, karena *function Add to Wish-list* ini juga berada pada halaman *Single-item*. *Syntax* yang digunakan agar *query* pada komponen *Services* dapat dijalankan adalah seperti pada Gambar 3.17.

```
if(isset($_POST['submit_wish'])) {
$status = '2';

$this->PosDataService->addDataPos($id_stok, $jumlah,
$harga, $total, $status);

$this->_redirect('/single-item/index/id/'.$id_stok);
}
```

Gambar 3.17 *Syntax* pada *Controller* untuk *Function Add to Wish-list*

Pada Gambar 3.17, deklarasi variabel yang dilakukan sama dengan yang berada pada *Controller* untuk *function Add to Cart*, yang membedakannya hanyalah nilai pada variabel status diubah menjadi '2'. Jadi, *query* yang dijalankan juga sama dengan *query* dari *function Add to Cart*.

4. *Wish-list*

Controller yang digunakan oleh *function* ini bernama WishList*Controller*. Untuk menampilkan data pada halaman Wish List ini, *syntax* pada *Controller* yang digunakan adalah :

$this->View->rowsItemWish = $this->PosDataService->

getAllDataJoinWish();

5. *Order History*

   Controller yang digunakan oleh *function* ini bernama *OrderHistoryController*. Untuk menampilkan data pada halaman *Order History* ini, *syntax* pada *Controller* yang digunakan adalah :

   $this->View->getInvoices = $this->PosService->

   getAllDataByCustomer();

6. *Invoice*

   Controller yang digunakan oleh *function* ini bernama *InvoiceController*. Untuk menampilkan detail transaksi dari *Invoice* yang dipilih pada halaman *Order History*, *syntax* pada *Controller* yang digunakan adalah :

   *$this->View->rowsItemInvoice = $this->PosDataService->getAllDataJoinInvoice('*$id*_pos');*

3.5.7 Menyesuaikan *View* Modul Transaksi

Setelah membuat komponen *Services* dan *Controller* untuk Modul Transaksi, yang dilakukan selanjutnya oleh penulis adalah menyesuaikan *View* atau tampilan dari Modul Transaksi itu sendiri supaya pengguna dapat menggunakannya dengan baik melalui *interface* / tampilan tersebut. Tampilan pada komponen *View* yang disesuaikan oleh penulis adalah sebagai berikut :

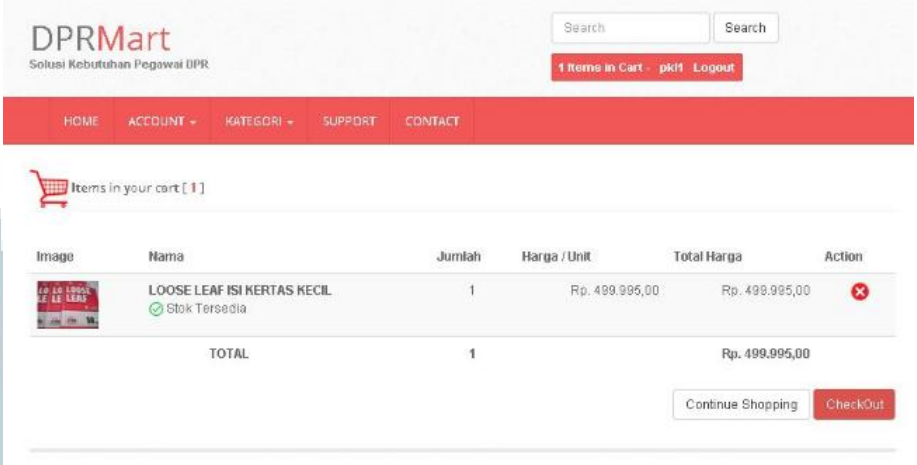
1. *View-Cart*

Halaman ini akan menampilkan produk – produk yang berada pada *My cart*, dimana data yang akan ditampilkan sudah diatur oleh komponen *Controller* dan *Services* seperti yang telah dijelaskan sebelumnya pada bagian pembuatan *Controller* dan *Services* untuk *function View-Cart* ini. Variabel yang digunakan untuk menampilkan data pada *My cart* adalah variabel '*rowsItem*', dimana variabel tersebut sudah dideklarasikan pada komponen *Controller*. Perintah yang diberikan pada *query* di komponen *Services* adalah '*FetchAll*', sehingga data - data yang berupa *multi-records* dapat ditampung secara keseluruhan pada variabel tersebut. Untuk menampilkan data tersebut menjadi sebuah barisan, maka digunakan *function* '*foreach*'. Sehingga *syntax* yang digunakan pada halaman *View-Cart* ini adalah seperti pada Gambar 3.18.

```
<?php foreach($this->rowsItem as $row) { ?>
<tr style="font-size:14;">
<?php $img_src = (file_exists(BERKAS_PATH . '\\stok\\' . $row->id . '.jpg'))
? $this->url_berkas . '/stok/' . $row->id . '.jpg' : $this->url_berkas .
'/stok/stok_kosong.gif'; ?>
<td><? echo '<a href ="/single-item/index/id/' . $row->id . '">
<img src="' . $img_src . '" height="50" width="65"" alt="" /></a>' ?></td>
<td><b><? echo ''. $row->nama .'' ?></b><br /><img src=
"http://www.procliparts.com/resize/900w/cliparts/pele/tqqczqcr9-check-circle.png"
height="15" width="15" />  Stok Tersedia</td></td>
<td align="center"><? echo ''. $row->status .'' ?></td>
<td align="right"> <? echo 'Rp. '. $this->FormatNumber($row->harga, 2) .'' ?></td>
<td align="right"><? echo 'Rp. '. $this->FormatNumber($row->total, 2) .'' ?></td>
<?
echo '<td align="center">  <a href="/view-cart/delete/id/'. $row->id .
'" onclick="return confirmDelete();"><img src=
"https://cdn1.iconfinder.com/data/icons/ui-icons-2/512/wrong-01-512.png"
height="20" width="20" /></a></td>';?>
</td>
</tr>
<tr>
<? $total_jumlah+= $row->jumlah;?>
<? $total_harga+=  $row->harga;?>
<? $subtotal+= $row->total; } ?>
    <td align="center" colspan="2"><b>TOTAL</b></td>
    <td align="center"><b><?  echo ''. $total_jumlah .'' ?> </b></td>
    <td align="right"></td>
    <td align="right"><b>Rp. <? echo ''. $this->FormatNumber($subtotal, 2) .'' ?></b></td>
    <td></td>
</tr>
```

Gambar 3.18 *Syntax* yang Digunakan pada Halaman *View-Cart*

Lalu, hasil tampilan yang didapat dari *syntax* tersebut adalah seperti pada Gambar 3.19.



Gambar 3.19 Tampilan *My cart* pada *Website*

2. *Wish-list*

Syntax yang digunakan untuk menampilkan data pada daftar *Wish-list* adalah seperti pada Gambar 3.20.

```
<h5 class="title">My Wish List</h5>

  <?if (count($this->rowsItemWish) > 0) {?>
    <table class="table table-striped tcart"> <thead>
        <tr style="font-size:15;align:center;">
          <th>Nama</th><th>Jumlah</th><th>Harga / Unit</th>
          <th>Total Harga</th><th>Add to Cart</th><th>Action</th></tr></thead>
      <tbody>

      <?php foreach($this->rowsItemWish as $row1) { ?>

      <tr style="font-size:12;">
      <td><b><? echo ''. $row1->nama .'' ?></b></td>
      <td align="center"><? echo ''. $row1->jumlah .'' ?></td>
      <td align="center"> <? echo 'Rp. '. $this->
      FormatNumber($row1->harga, 2) .'' ?></td>
      <td align="center"><? echo 'Rp. '. $this->
      FormatNumber($row1->total, 2) .'' ?></td>
      <td align="center"><a href="/wish-list/edit/id/<? echo''. $row1->id .'' ?>">
      <img src="http://www.axialis.com/objects/ip_icon_05_sample5d.png"
      height="20"width="20" /></a></td>
      <td align="center"><a href="/wish-list/delete/id/<? echo ''. $row1->id .'' ?>"
      onclick="return confirmDelete();"><img src=
      "https://cdn1.iconfinder.com/data/icons/ui-icons-2/512/wrong-01-512.png"
      height="23" width="23" /></a></td>
      </tr>
```

Gambar 3.20 *Syntax* yang Digunakan pada Halaman Wish List

Pada Gambar 3.20, tidak semua *syntax* ditulis secara langsung oleh penulis. Beberapa *syntax* seperti *syntax* yang mengandung *class* maupun *style* pada tabel sudah ada terlebih dahulu, namun dengan data yang statis. Yang dilakukan oleh penulis adalah menyesuaikan tampilan yang sudah ada tersebut dengan data – data yang telah ditentukan. Lalu, hasil tampilan yang didapat dari *syntax* tersebut adalah seperti pada Gambar 3.21.

Gambar 3.21 Tampilan Wish List pada *Website*

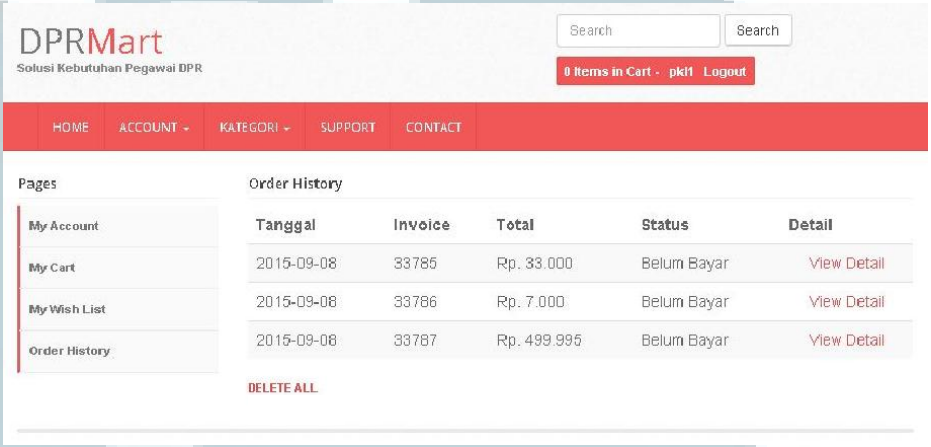
3. *Order History*

Syntax yang digunakan untuk menampilkan data pada daftar *Wish-list* adalah seperti pada Gambar 3.22.

```
<h5 class="title">Order History</h5>
<table class="table table-striped tcart">
    <thead><tr>
        <th>Tanggal</th> <th>Invoice</th><th>Total</th>
        <th>Status</th><th>Detail</th></tr></thead><tbody>
    <?php foreach($this->getInvoices as $row) { ?>
      <tr> <td><? echo '' . $row->tanggal . '' ?></td>
      <td><?php echo '' .  $row->id . '' ?></td>
      <td><? echo 'Rp. ' .  $this->FormatNumber($row->total) . '' ?></td>
      <td><? if ($row->status == 3) { echo 'Belum Bayar';
      if ($row->status == 4 ) { echo 'Sudah Bayar'; } } ?></td>
      <td align="center"><?php echo '<a href="/invoice/index/id/' .
      $row->id . '">View Detail</a>'; ?></td> </tr>
  <? }  ?>
    </tbody>
  </table>
```

Gambar 3.22 *Syntax* yang Digunakan pada Halaman *Order History*

Sama seperti *syntax* yang digunakan pada tampilan *Wish-list*, *syntax* pada tampilan halaman *Order History* ini juga sudah memiliki format tabelnya sendiri sehingga penulis hanya menyesuaikan tabel tersebut dengan data yang diambil dari *database*. Lalu, hasil tampilan yang didapat dari *syntax* tersebut adalah seperti pada Gambar 3.23.



Gambar 3.23 Tampilan *Order History* pada *Website*

3.5.8 Membuat *Services* untuk Modul *Search*

Pada bagian ini, penulis membuat *Services* untuk Modul *Search*. *Services* ini berisi *query – query* untuk pemanggilan data dalam *database*. *Query* pada komponen *Services* ini berada dalam '*StokServices*'.

Query pada *Services* ini digunakan untuk memanggil keseluruhan data dari tabel 'stok' yang nama produknya sama dengan apa yang diketikkan oleh pengguna pada *field* pencarian produk. *Query* yang digunakan terdapat pada Gambar 3.24.

```
function getAllDataSearch($search)
{
    $search = Zend_Controller_Front::getInstance()->getRequest()->getParam('cari');
    $select = $this->stok->select()
        ->setIntegrityCheck(false)
        ->from('stok', array('*'))
        ->where('nama like ?', '%'.$search.'%')
        ->where('status = 1');
    $result = $this->stok->fetchAll($select);
    return $result;
}
```

Gambar 3.24 *Query* pada *Services* untuk Modul *Search*

Pada *query* dalam Gambar 3.24, akan dijelaskan maksud dari *syntax* yang ada, dimana dalam *query* ini terdapat 9 pernyataan, yaitu :

1. '*function getAllDataSearch*($*Search*)', pernyataan ini adalah sebuah deklarasi fungsi yang bernama '*getAllDataSearch*' dan menggunakan parameter '$*Search*'. Untuk menjalankan *query*-nya, *function* ini harus dipanggil oleh komponen *Controller*.
2. '$*Search* = *Zend_Controller_Front::getInstance()->getRequest()->getParam('cari');*', pernyataan ini adalah sebuah deklarasi variabel dari '$*Search*' dimana $*Search* ini akan menampung parameter yang berupa kata – kata yang ditulis pengguna pada *field* pencarian produk.
3. *'$select = $this->stok->select()*', pernyataan ini mempunyai arti dimana *query* akan melakukan *select* data pada *database* dalam kolom 'stok'.
4. *'->setIntegrityCheck(false)',* pernyataan ini menunjukkan bahwa pengecekan integritas data tidak akan dilakukan dengan tujuan supaya keseluruhan data yang berhubungan dengan kata pencarian yang ditulis oleh pengguna dapat ditampilkan.

5. *'->from('stok', array('*'))'*, merupakan pernyataan yang menunjukkan bahwa pengambilan data dalam *database* berasal dari tabel 'stok'.
6. *'->where('nama like ?', '%'.$Search.'%')'*, pernyataan ini menunjukkan bahwa data yang akan diambil harus memiliki kesesuaian atau kesamaan antara apa yang diketikkan oleh pengguna dengan data yang berada pada tabel stok di kolom 'nama'.
7. *'->where('status = 1');'*, pernyataan ini bertujuan untuk mengambil data pada tabel stok dimana hanya data yang mempunyai status = 1 yang akan ditampilkan. Pada tabel stok ini, status = 1 mempunyai arti bahwa data stok tersebut masih aktif dan tersedia produknya di toko.
8. *'$result = $this->stok->fetchAll($select);'*, pada pernyataan ini, hasil dari keseluruhan data yang diambil akan ditampung dalam variabel *'$result'*. Metode yang digunakan untuk menampung data pada variabel ini adalah *'fetchAll'* dimana dengan metode ini dapat memungkinkan variabel menyimpan data dalam jumlah banyak. Dengan kata lain, keseluruhan data yang diambil dari *database* akan ditampung ke dalam satu variabel. Tidak seperti jika menggunakan metode '*fetchRow'*, dimana data yang ditampung dalam variabel hanya satu data saja.
9. *'return $result;'*, pernyataan terakhir ini akan mengembalikan keseluruhan data yang telah ditampung dalam variabel *'$result'* sehingga data tersebut dapat ditampilkan di halaman pencarian (komponen *View*) melalui komponen *Controller*.

3.5.9 Membuat *Controller* untuk Modul *Search*

Pada bagian ini, penulis membuat *Controller* untuk Modul *Search* dimana komponen ini digunakan untuk mengatur data apa saja yang akan ditampilkan pada komponen *View*. Dan data yang akan ditampilkan pada komponen *View* adalah data hasil pencarian produk yang menggunakan *function* '*getAllDataSearch*' untuk pencarian datanya, seperti yang telah dibuat pada bagian sebelumnya.

Untuk memanggil *function* yang ada pada komponen 'Stok*Services*', *Controller* harus melakukan deklarasi terlebih dahulu untuk mendefinisikan *StokServices*. Deklarasi ini bertujuan membuat *Class* baru yang akan digunakan untuk menampung *function – function* yang ada dalam *StokServices*. Deklarasinya menggunakan *syntax* '*$this->StokService = new StokService();'*

Setelah komponen *StokServices* didefinisikan, selanjutnya adalah memanggil *function* '*getAllDataSearch*' untuk melakukan *query* pemilihan data yang berhubungan atau sesuai dengan kata yang diketikkan oleh pengguna pada *field* pencarian produk. Pemanggilan *function* yang dilakukan menggunakan *syntax '$result = $this->View->StokXRows = $this->StokService->getAllDataSearch($Search);*'. *Syntax* tersebut memiliki arti bahwa *function* '*getAllDataSearch*' akan dipanggil dan hasil dari *query*-nya akan ditampung dalam variabel '*StokXRows*'. Variabel *StokXRows* inilah yang selanjutnya akan digunakan pada komponen *View*.

Namun dalam penggunaannya, setiap halaman yang menampilkan produk harus menggunakan '*Pagination*'. *Pagination* adalah pemisahan produk menjadi beberapa halaman karena ditemukan produk yang berjumlah sangat banyak. Tujuan dari penggunaan *Pagination* ini adalah untuk membatasi jumlah produk yang ditampilkan dalam satu halaman, sehingga untuk melihat produk yang lain maka pengguna dapat berpindah ke halaman selanjutnya.

Zend Framework ini mempunyai suatu *function* yang sudah tertanam di dalamnya untuk membuat *Pagination*. *Function* dari *Pagination* ini dibuat dalam komponen *Controller* yang dalam *Zend Framework* disebut sebagai *Paginator*. *Syntax* yang digunakan dalam *function Pagination* ini adalah seperti pada Gambar 3.25.

```
$page=$this->_getParam('page',1);
$paginator = Zend_Paginator::factory($result);
$paginator->setItemCountPerPage(9);
$paginator->setCurrentPageNumber($page);
$this->view->paginator=$paginator;
```

Gambar 3.25 *Function* untuk Menjalankan *Pagination*

Pada Gambar 3.25, data yang ditampung dalam variabel *$result* akan dipecah ke dalam per sembilan item, sehingga dalam satu halaman hanya akan menampilkan sembilan data yang selanjutnya dikonversi menjadi sembilan item produk, dan untuk melihat data atau produk yang lain dapat dilakukan pada halaman selanjutnya.

3.5.10 Menyesuaikan *View* Modul *Search*

Setelah komponen *Services* dan *Controller* telah dibuat untuk modul *Search*, selanjutnya adalah menyesuaikan komponen *View* pada modul *Search* ini supaya data yang dicari oleh pengguna dapat ditampilkan dalam halaman pencarian.

Untuk menampilkan data tersebut, maka komponen *View* harus memanggil variabel yang ada pada komponen *Controller*. Pada kasus ini, variabel yang akan dipanggil adalah '*StokXRows'* yang pada bagian sebelumnya telah dibuat terlebih dahulu.

Terdapat dua penyesuaian yang akan dilakukan dalam modul *View* ini, yaitu sebagai berikut :

1. Penyesuaian pernyataan hasil pencarian. Dalam halaman pencarian ini, harus disediakan informasi kepada pengguna berapa banyak produk yang ditemukan berdasarkan kata yang diketik oleh pengguna pada *field* pencarian produk. Format yang digunakan oleh informasi pada halaman pencarian ini adalah "Pencarian produk (produk yang dicari). Ditemukan (jumlah pencarian) item". Untuk membuat informasi tersebut, maka *syntax* yang digunakan adalah seperti pada Gambar 3.26.

```
<li>Pencarian produk "<?php $search = $_POST['cari'];
$jumlah_cari = count($this->StokXRows);
echo "$search";?>". Ditemukan <?php echo "$jumlah_cari";?> item</li>
```

Gambar 3.26 *Syntax* yang Digunakan untuk Informasi Pengguna

2. Penyesuain tampilan dari masing – masing produk yang ditampilkan dalam halaman pencarian. Format dari tampilan produk yang ditampilkan adalah seperti pada Gambar 3.27.

Gambar 3.27 Contoh Format Tampilan Produk dari Hasil Pencarian

Karena dalam modul *View* ini menggunakan *function Pagination*, maka variabel yang dipanggil dalam komponen *Controller* adalah variabel '$*paginator*', bukan variabel '*StokXRows*' lagi. Hal ini dapat terjadi karena variabel *StokXRows* sudah dimasukkan ke dalam *function Pagination* yang dimiliki oleh *Zend Framework* melalui variabel '$*paginator*'. Untuk membuat tampilan produk seperti pada gambar di atas, sudah tersedia beberapa *syntax* yang menggunakan *class* pada properti tabel, namun *syntax* tersebut masih menggunakan data statis, bukan menggunakan data yang diambil dari *database*. Oleh karena itu, penulis menyesuaikannya sehingga *syntax* tersebut dapat menampilkan

data dari *database* yang merupakan produk yang dicari oleh pengguna. Sehingga *syntax* yang dibutuhkan menjadi seperti pada Gambar 3.28.

```
foreach($this->paginator as $row) {

    $img_src = (file_exists(BERKAS_PATH . '\\stok\\' . $row->id .
    '.jpg')) ? $this->url_berkas . '/stok/' . $row->id . '.jpg' :
    $this->url_berkas . '/stok/stok_kosong.gif';

  echo '   <div class="col-md-4 col-sm-6">
           <!-- Each item should be enclosed in "item" class -->
           <div class="item">
             <!-- Item image -->
             <div class="item-image">
               <a href="/single-item/index/id/' . $row->id . '">
               <img src="' . $img_src . '" /></a>
             </div>
             <!-- Item details -->
             <div class="item-details">

             <h5><a href="/single-item/index/id/' . $row->id . '">
             ' . $row->nama . '</a></h5>
               <div class="clearfix"></div>
               <!-- Para. Note more than 2 lines. -->
               <p>' . $row->deskripsi . '</p>
               <hr />
               <!-- Price -->
              <center> <div class="item-price">Rp ' . $this->
              FormatNumber($row->harga_jual) . '</div> </center>
               <!-- Add to cart -->';

             echo '
               <div class="clearfix"></div>
             </div> </div> </div>'; }
```

Gambar 3.28 *Syntax* yang Digunakan untuk Membuat Tampilan Produk

3.6 Kendala yang Dihadapi

Ketika penulis melakukan kerja magang di Sekretariat Jenderal DPR RI, ada beberapa kendala yang dihadapi dalam pelaksanaannya, yaitu :

1. Ketika pembimbing lapangan memutuskan untuk menggunakan *Framework* bernama *Zend Framework*, penulis beserta tim belum mengenal sama sekali tentang *Framework* tersebut. Hal ini membuat penulis beserta tim merasa

kesulitan dalam melakukan pengembangan *website* toko *online* untuk koperasi Sekretariat Jenderal DPR RI ini. Dengan adanya kendala tersebut membuat pengembangan *website* memakan waktu lebih lama dan membutuhkan kerja sama tim yang sangat baik.

2. Dalam pengembangan *website* tersebut, penulis beserta tim tidak mendapat hak akses ke dalam *database* secara langsung. Hal ini membuat pengembangan sistem *database* menjadi lebih lambat karena penulis beserta tim harus menghubungi pembimbing lapangan terlebih dahulu jika memerlukan pengubahan struktur *database*. Dengan adanya kendala ini, penulis beserta tim tidak dapat melakukan pengembangan *website* dengan cepat karena dibutuhkan konfirmasi terlebih dahulu kepada pembimbing lapangan ketika akan mengubah struktur *database*.

3.7 Solusi

Solusi yang dapat disarankan oleh penulis untuk menghadapi kendala tersebut adalah :

1. Dengan mempelajari *Zend Framework* terlebih dahulu secara detail beserta dengan struktur dan komponennya. Kerja sama tim juga dibutuhkan dalam solusi ini, yaitu dengan cara membagi pengetahuan jika salah satu anggota tim telah menguasai beberapa hal tentang *Zend Framework*. Hal ini membuat penulis beserta tim harus melakukan berbagai macam percobaan secara langsung untuk mengetahui lebih lanjut tentang *Framework* ini. Dengan banyak mencoba menulis *syntax – syntax*, penulis beserta tim dapat mengetahui lebih

rinci tentang format – format penulisan *syntax* pada *Zend Framework*, begitu juga dengan penulisan format *query* yang berbeda dengan *query* SQL pada umumnya. Penguasaan terhadap *Zend Framework* ini sangat penting karena keseluruhan sistem IT dari Sekretariat Jenderal DPR RI menggunakan *Zend* sebagai *Framework* dasarnya. Hal ini membuat pemilihan *Zend Framework* menjadi lebih diutamakan agar hubungan antar sistem yang ada dapat saling diintegerasikan satu sama lain dengan baik dan mudah.

2. Solusi yang dibutuhkan untuk tidak diberikannya hak akses ke dalam *database* secara langsung adalah dengan membuat *user_id* khusus yang bisa digunakan oleh tim pengembang. *User_id* yang dimaksud adalah *username* yang diberikan akses '*GRANT*' pada *database* toko yang digunakan. Hal ini membuat tim pengembang dapat melakukan akses *database* toko secara langsung tanpa menyentuh *database* lain yang berada diluar lingkup pengembangan *website*. Penggunaan *user_id* ini berdasarkan pada kebijakan Sekretariat Jenderal DPR RI dimana untuk mengakses sistem IT harus menggunakan *user_id* yang sudah ditentukan. Karena *user_id* ini mempunyai '*privilege*'-nya masing – masing di dalam sistem IT.